# Efficient Boosted Ensemble-based Machine Learning in the Context of Cascaded Frameworks

A thesis presented in partial
fulfilment of the requirements

for the degree of

Doctor of Philosophy in Computer Science

at Massey University,
Auckland, New Zealand

Teo Sušnjak

2012

## Abstract

The ability to both efficiently train robust classifiers and to design them for fast detection is an important goal of machine learning. With an ever increasing amount of available data being generated, the task of expeditiously producing real-time capable classifiers is becoming more challenging. In the context of the increasing complexity of the task, ensemble-based learning methods have proven themselves to be effective approaches for satisfying these requirements.

Ensemble methods produce a number of weak models that are strategically combined into a single classifier. They have been particularly effective when combined with boosting algorithms and strategies that structure the ensembles into cascades. The strength of cascaded-ensembles lies in the separate-and-conquer approach they employ during the training of each layer. Class decision-boundaries for trivial cases are learned in the early rounds, while more difficult decision boundaries are refined with each succeeding layer. In a two-class problem domain, non-target instances learned in initial layers are removed and replaced by more complex samples, frequently referred to as bootstrapping. With this procedure, efficient coarse-to-fine learning is accomplished.

The contribution of this thesis lies in three main areas that centre around the concept of improving the efficiency in the training and execution process. The first explored ways in which the conventional ensemble-cascades could be combined with an even more aggressive separate-and-conquer strategy that further partitions the ensemble inside each layer. The focus was on the two-class learning problem and used face detection as the medium to observe the trade-offs involved concerning both the accuracy and the efficiency of the resulting classifiers. The algorithm was further developed in a way that enabled the bootstrapping of positive samples within a cascade, alongside the conventional approach that bootstraps only the negative samples. Secondly, the negative effect of dynamic environments on static classifiers on binary class problems was considered. A method was developed which enabled the cascaded classifiers to efficiently adapt to the changing environment on domains with high volume streaming data. This environment was simulated using face detection as well. Lastly, the open problem of creating integrated multiclass cascades was researched and an algorithm was devised.

Overall, the findings have shown that invariably a trade-off is incurred between reduced training runtimes resulting from aggressive separate-and-conquer strategies and the accuracy of the final classifiers. Using the CMU MIT test dataset, the experiments showed that though the proposed positive sample bootstrapping component succeeded in significantly reducing the training runtimes without compromising the accuracy, the general decomposition strategy did lower the accuracy when compared to the benchmark Viola-Jones classifiers. The proposed adaptive cascade learning algorithm for drifting concepts was also evaluated on a face detection problem set. The results demonstrated its ability to effectively adapt to dynamic environments in high speed data streams without requiring explicit re-training of the individual classifiers. The multiclass cascaded algorithm was compared to three existing algorithms on 18 UCI datasets. It was found to be, on average, several times faster to train and to execute, while generating comparable accuracy rates. The algorithm exhibited scalability to large datasets but was found to be susceptible to producing overly complex classifiers on datasets with a large number of class labels.

# Acknowledgements

My thanks to the Institute of Information and Mathematical Sciences at Massey University for enabling me to grow over the years in my academic pursuits and for fostering an environment which allowed this to take place. Thanks in particular to the faculty at the Department of Computer Science. You have inspired, encouraged and taught me not just knowledge, but the value of it. My gratitude goes to all the administration staff at the institute for their help and assistance over the years, as well as to all my colleagues from whom I have learned much.

I am grateful to the Tertiary Education Commission for providing me with a generous doctorate scholarship. I express my thanks also to the Ministry of Science and Innovation for their internship grant that enabled me to implement research and expertise arising from this research into the industrial setting. A special thanks to Compac Sorting Ltd. for their support in the final year and for providing me with an opportunity to apply my research to solving real-world problems.

I would like to extend my thanks to the members of the doctoral defence committee, Prof. Pitoyo Hartono, Prof. Alvis Fong and Assoc. Prof. Chris Scogings for their time as well as valuable input and engagement with me in a fruitful discussion. Also a special thanks to Prof. Anne de Bruin for her role as convenor of the examination.

Thanks to my friends and family for your support over the years and a special thanks to a few of you who have reviewed the initial manuscript. I look forward to now seeing and spending more time with all of you. Christoph Schumacher, without your encouragement I do not think I would have embarked on this path. Thank you for your friendship and continuous support.

Brian and Beverley for never doubting, but always believing. Expressing my gratitude to you both seems so inadequate. To my mum and dad who have unceasingly encouraged and urged me on from distant shores, thank you; ovo je i vaš uspjeh, hvala vam za sve.

Ken Hawick, my heartfelt gratitude to you for being my co-supervisor. You have been a source of wisdom, advice and encouragement during many phases of this research. To my supervisor Andre Barczak, I am so indebted to you. You have been my teacher, my mentor and have also become a close friend. I look forward to our conversations over coffee continuing for many years to come.

To my beautiful wife Sarah; I love you. You have been my rock every step of the way.

**Teo Sušnjak**
Auckland, New Zealand
October, 2012

# Contents

# List of Figures

# List of Tables

# List of Algorithms

# List of Acronyms

| | |
|---|---|
| BPSL | Bootstrapped PSL |
| BPSL.r | Bootstrapped PSL with re-sampling |
| CMU MIT | Carnegie Mellon University - Massachusetts Institute of Technology (Face detection dataset) |
| CTF | Coarse-To-Fine learning |
| DP | Domain Partitioning (Multiclass weak learner) |
| ECC | AdaBoost.ECC using Error Correcting Codes |
| ECOC | Error Correcting Output Codes |
| M2 | Adaboost.M2 |
| MCS | Multiple Classifier Systems |
| OAA | One-Against-All |
| OAO | One-Against-One |
| OC | AdaBoost.OC using Output Coding |
| PSL | Parallel Strong classifier within the same Layer algorithm |
| RIPPER | Repeated Incremental Pruning to Produce Error Reduction algorithm |
| ROC | Receiver Operating Curve |
| UCI | University of California Irvine (Machine learning dataset repository) |
| VJ | Viola-Jones (Ensemble-cascade algorithm) |

# Chapter 1

# Introduction

Today's society is becoming ever more computerized. As a result there has been an immense growth in both the methods for generating and collecting data. This is producing a deluge of data comes from areas with which people interact daily. Most common of these are telecommunications, financial transactions, social networks, general internet activity, medical, security surveillance and demographic data to name a few [61, 184]. Given that the cost of data storage and data sensors is decreasing, its availability and pervasiveness is becoming even greater. Gleaning information from these sources remains important and necessary as it can yield important discoveries which impact on society and the lives of individuals in all aspects.

While this stream of data is seemingly unending, the human resources needed for analyzing, classifying and discovering important information from it are not. What is required, are automatic tools that efficiently and accurately, with scalability, transform raw data into information with useful knowledge about it. This is where artificial intelligence and more specifically, the field of machine learning as the topic of this thesis, makes its contribution.

### Machine Learning

The primary research area of machine learning is investigating how computer programs find meaningful patterns in the input data that can then be used to make future predictions on unseen data samples. Depending on the nature of the predictions, these algorithms can be split into classification and regression problems. Classification induces a model whose output is discrete values which categorize a sample into a predicted class label. Regression on the other hand models a continuous-valued function and its output is numerical. In that sense, regression predicts missing or unknown values.

Machine learning can be divided into several categories. Under supervised learning, the algorithm is provided with training samples and the expected outputs associated with them. The goal of this most common group of algorithms is to learn to map the input samples to their correct and known output values. On the other hand, unsupervised learning methods are furnished with training samples without being explicitly told of their correct outputs. This form of learning is a synonym for clustering. In some problem domains, it is not possible to provide all the correct output labels and so a combination of labeled and unlabeled samples is employed. Such methods belong to the category of semi-

supervised learning and are particularly useful in domains where the true interpretation of the samples is either too costly or time consuming to provide. Active learning involves a human in the learning process by posing questions that ask the user to label a given number of samples. As each sample is labeled, the algorithm refines its model using the knowledge from the user.

This research however focuses solely on classification under the supervised learning paradigm. In more formal terms, the problem of supervised classification learning can then be stated as algorithms that learn a training set $D$ consisting of $n$ input and output pairs of samples $\{(x_1, y_1), ..., (x_n, y_n)\}$. Each sample consists of a feature vector $x_i = (x^1, ..., x^d) \in \mathbb{R}$ comprising of $d$ number of dimensions and an associated class label $y_i \in Y$ where $Y \in \{0, ..., k\}$ for a $k$-class classification problem. The outcome of the learning is a classifier or hypothesis $H$, which ideally induces an accurate decision boundary for separating the classes, and also one that closely approximates the true boundary for the entire domain $\mathbb{R}$.

Induction of classifiers usually becomes problematic when there is a very large number of training samples. This often results in the formation of complex class boundaries which are difficult to learn accurately for classifiers that attempt to generate a single rule to describe the data. The situation is also exacerbated when the feature vectors describing the data are high dimensional. Feature extraction can frequently be associated with high computational overheads, particularly when operating in computer vision settings. In such environments, the additional runtime costs for extracting high volumes of expensive features can make the training and execution of classifiers impractical. In addition, with the advent of streaming data, it is becoming apparent that there is a profound need for classifiers to possess the ability to rapidly adapt to dynamic environments and to the changes that occur to the learned class descriptors.

The popularity of a specific family of machine learning algorithms termed ensemble-based methods has grown significantly in recent years due to their ability to address many of the emerging challenges in the field. Paradoxically, the strength of ensemble-based methods lies in their unambitious approach to the task of learning. While most learning algorithms attempt to induce a single optimal classification model to describe complex data, ensemble methods instead often generate multiple classifiers whose predictive strength is unassumingly only slightly better than a random guess. However, given an intelligent combination strategy, the *weak classifiers* can be effectively combined in ways that their collective voting strength is frequently able to match the best existing classification algorithms.

In their seminal work Viola and Jones [175] expanded the principles of ensemble-based methods further with the concept of coarse-to-fine learning and classifier evaluation. This breakthrough made the training of classifiers on datasets comprising of hundreds of millions of samples considerably more feasible, while creating efficient classifiers that could effectively and efficiently perform classification in real-time on the problem of face detection. Their innovation came in the form of structuring the ensemble classifiers into cascades whereby each successive layer comprised of more powerful classifiers with the ability to refine the class decision boundary in both learning and detection in an iterative manner. This approach has greatly contributed to the advancement in developing modularized and accurate classifiers on many domains; however, the training of effective

cascaded classifiers using the Viola-Jones method is non-trivial and is associated with complex trade-offs concerning the final accuracy and the training/detection runtimes.

In an effort to further reduce training runtime complexities associated with large-scale learning problems using cascaded classifiers, Barczak et al. [5] proposed to introduce an additional nested cascade within each of the layers. This algorithm was termed PSL, to denote the idea of nested cascades as being **p**arallel **s**trong classifiers within the same **l**ayer. The goals of this research were to work within the framework of cascaded classifiers and to explore ways of expanding the contribution of the PSL algorithm in ways that further build upon the ideas of coarse-to-fine learning, while attempting to control the trade-offs between increased efficiency and compromises in classification accuracies. The following section outlines the specific objectives and thesis questions that guided this research, as well as the scope within which it was conducted.

## 1.1 Objectives

This research set out to accomplish the following objectives:

1. Investigate the ability of the binary-class PSL algorithm to produce effective classifiers on a challenging real-world problem set in the domain of face detection.

2. Make extensions to the algorithm that enable it to scale to large training datasets and to study the efficiency/accuracy trade-off.

3. Propose a method for achieving real-time adaptability of PSL-like classifiers on dynamic and evolving environments that possess concept drifts of varying magnitudes.

4. Apply the underlying separate-and-conquer[1] principle of the PSL algorithm to the problem of multiclass classification.

5. Explore the feasibility of using the PSL method as a meta-learning framework for converting existing multiclass boosting-classifiers into cascaded ensembles.

### Thesis Questions

Due to the nature of the research which branched into several topics within the domain of classification, numerous questions evolved and took shape over the course of the study. These questions were:

1. Are PSL-like algorithms a viable alternative for training classifiers on large and challenging datasets which have imbalanced class-distributions? How do the PSL-like methods effect the trade-off relationship between the training runtime-complexity and accuracy on the chosen problem domain?

[1] The separate-and-conquer approach focuses on learning one class label per iteration and optimizes the loss according to this class only. Subsequent, training samples belonging to this class are removed from further induction. In contrast, the divide-and-conquer approach considers the combined loss of all class labels at each iterative decision point. All training samples are retained until the leaf nodes of the resulting tree are reached.

2. Can the underlying separate-and-conquer approach of PSL be further leveraged enabling the bootstrapping of positive samples on a binary-class problem? What effect does this strategy have on the scalability of cascaded frameworks to large-scale datasets?

3. What is the feasibility of integrating into PSL, methods for real-time adaptive learning in dynamic environments with drifting concepts?

4. Can the current binary PSL algorithm be modified in order to make it applicable as a coarse-to-fine method for learning multiclass problems?

5. Is it possible to apply the coarse-to-fine learning principles to existing multiclass boosting algorithms which generate single-layer ensembles, in order to create cascaded classifiers?

### Scope

Given the wide range of topics within this research that involved both binary and multiclass learning within static and dynamic environments, the scope of this research has therefore been limited to: (1) developing ensemble-based learning and considering other ensemble methods for the purposes of comparison, (2) working with boosting algorithms in combination with proposed training frameworks as the underlying vehicle for generating diverse classifiers, (3) employing weak-learners as inputs to the boosting algorithms for both development and comparisons with benchmark algorithms, (4) supervised learning whereby the training algorithm is provided with complete knowledge as to the mappings of sample instances to their associated class labels, (5) classification methods only, thus excluding regression.

## 1.2 Contributions

The outcome of this research has been the development of new algorithms. The conventional procedure for training binary-class cascaded classifiers has been expanded, making possible the bootstrapping of positive samples through a coarse-to-fine learning strategy. As a result, this has enabled the usage of much larger training datasets with reduced training runtimes. An adaptive learning algorithm was devised for this type of cascaded classifiers. It was designed for classifiers operating in high-volume binary-class domains which are evolving, and thus causing the underlying target class-descriptors to change with time. The goal was to develop a method capable of rapidly adapting to the changing environment without requiring the explicit re-training of classifiers due to time constraints. Finally, the principles underpinning the previous algorithms were employed to develop an algorithm for constructing integrated multiclass ensemble-cascades. The generic qualities of this algorithm were empirically shown by demonstrating how it can be used as a meta-learning algorithm for converting existing single-layered multiclass learning methods into cascaded ensembles.

With the support of the Ministry of Science and Innovation in New Zealand, practical contributions of this research have involved establishing links with industry. The outcome

of this has been that elements from this research have been applied to finding solutions to challenging real-world machine learning and computer vision problems.

## Published works

The following papers have been published as a direct result of this research.

**Peer reviewed conference and journal papers:**

- Susnjak, T., Barczak, A. and Hawick, K. (2011), Adaptive cascade of boosted ensembles for face detection in concept drift, Neural Computing & Applications , Springer, pp. 1-12.

- Susnjak, T., Barczak, A., Reyes, N. & Hawick, K. (2011), A New Ensemble-Based Cascaded Framework for Multiclass Training with Simple Weak Learners, In Proceedings of the 14th International Conference on Computer Analysis of Images and Patterns (CAIP), LNCS. Springer

- Susnjak, T., Barczak A. L. Reyes N. and Hawick K. Coarse-to-fine multiclass learning and classication for time-critical domains. Manuscript submitted for publication,November 2011.

- Susnjak, T., Barczak, A. and Hawick, K. (2010a), Adaptive ensemble based learning in non-stationary environments with variable concept drift, in Neural Information Processing. Theory and Algorithms, Vol. 6443 of Lecture Notes in Computer Science, Springer, pp. 438-445.

- Susnjak, T., Barczak, A. and Hawick, K. (2010b), A modular approach to training cascades of boosted ensembles, in Structural, Syntactic, and Statistical Pattern Recognition, Vol. 6218 of Lecture Notes in Computer Science, Springer, pp. 640-649.

- Susnjak, T. and Barczak, A. (2009), Accelerated classifier training using the PSL cascading structure, in, Advances in Neuro-Information Processing, Vol. 5506 of Lecture Notes in Computer Science, Springer, pp. 945-952.

**Technical reports:**

- Susnjak, T., Barczak, A. L. C. and Hawick, K. A. (2010), A novel bootstrapping method for positive datasets in cascades of boosted ensembles, Research Letters in the Information and Mathematical Sciences Volume 14, pp.17-24, ISSN 1175-2777, Institute of Information and Mathematical Sciences, Massey University Albany.

- Susnjak, T., Barczak, A. L. C. and Hawick, K. A. (2009), Accelerated face detector training using the PSL framework, Research Letters in the Information and Mathematical Sciences Volume 13, pp.68 - 80, ISSN 1175-2777, Institute of Information and Mathematical Sciences, Massey University

## Thesis Structure

Following the introduction, this thesis is organized into 9 chapters. Chapter 2 reviews the broad field of ensemble-based learning. Since there currently exists no definitive taxonomy of ensemble-based learning [148], greater emphasis is placed on reviewing methods that are directly related to this research. Given the lack of an agreed and standardized taxonomy of ensemble-based systems in the published literature, the chapter presents the underlying fundamentals which underpin the field while emphasizing other subfields in proportion to their relation to the objectives of this research. This chapter highlights some ongoing challenges and open questions within ensemble-based learning, to which this research attempts to make a contribution.

Chapter 3 discusses the general and common methods employed across all the experiments in all the chapters. Each chapter thereafter, presents algorithms and results dealing with different sets of questions and expands slightly on the general methodology in order to discuss the specific details of its experimental implementations.

The preliminary findings of the applicability of PSL to the problem of face detection are presented in Chapter 4. This leads on to research using larger datasets for the purpose of developing modifications that enable a greater degree of scalability in Chapter 5. The problem of PSL classifier real-time adaptability in dynamic environments is discussed in Chapter 6 before the focus shifts towards multiclass learning. Chapter 7 proposes modifications to the PSL training which enable multiclass learning, while the subsequent $8^{th}$ chapter demonstrates how existing single layered ensemble algorithms can be converted into effective cascaded classifiers. The concluding $9^{th}$ chapter then follows.

# Chapter 2

# Ensemble-Based Machine Learning

The goal of this chapter is to provide a short overview of the primary components of ensemble learning. Since no definitive taxonomy of ensemble-based learning currently exists [148], greater emphasis is placed on reviewing methods directly related to this research.

The theory of ensemble learning is initially covered with the focus primarily being on methods for creating diverse classifiers as well as methods for combining the output of classifiers. The concept of boosting is reviewed from the perspective of binary and multi-class learning, while relevant boosting algorithms to this research are presented in greater detail. Finally, the chapter concludes with the introduction of coarse-to-fine learning methods within the context of the emerging challenges they attempt to address.

## 2.1 Theory

At the heart of the computational learning theory lies the principle of learnability. It can be said that a target concept is *strongly learnable* if the learner, when exposed to instances of an unknown concept, is with high probability able to induce a classifier that is strongly correlated with the true function in polynomial time. In this sense, the learner is said to be probably approximately correct (PAC) [168]. Conversely, a target concept for which a learner induces classification rules that are only just better than a random guess, is loosely correlated with the true function and is said to be *weakly learnable*. The question posed by ensemble based learning is therefore: "Is it possible to induce a strong classifier, by generating a committee of weak classifiers?".

In the seminal work, Schapire [139] provided an affirmative answer to this question by demonstrating that the notions of strong and weak learnability are equivalent. His research showed how the weak learnability of an algorithm can be *boosted* in order to arrive at a strong classifier. This laid down the foundation of boosting itself, which is presently the most widely used ensemble-based method for machine learning and the most significant learning idea introduced in the past two decades [148].

Since then, a raft of successful applications has been developed involving ensemble-based methods to difficult real-world problems which extend across a broad range of domains [112, 134]. For example, applications in remote sensing have been developed

for mountain terrain classification [56], agricultural [54] as well as urban [37] land classification. In the sphere of person classification, the ground braking work by Viola and Jones [175] demonstrated real-time face detection, person identification [163], and speech recognition [106]. Medical applications have been designed for pharmaceutical molecule classification [164] as well as MRI [121] and ECG [87] classification. In addition, ensemble learning has been applied to finance [84], manufacturing [96], spam email [125] amongst others. The most recent research is located in the proceedings of dedicated conferences for ensemble based methods such as the annual Conference on Multiple Classifier Systems (MCS) [138].

There are a number of theoretical reasons why ensemble-based solutions have proved to be effective and preferable over single classifier methods. Firstly, it is well understood that the accuracy of a classifier is to a large extent dependent on appropriately matching a learning algorithm to the properties of the data for a given application domain [112]. Ensemble methods mitigate against the dependence between a learning algorithm and the application domain. They accomplish this by generating numerous classifiers that differ in type or parameter instantiations and whose outputs can be intelligently combined in order to reach a more informed decision. Ensemble methods therefore provide robust statistical justifications for their use. Though an individual classifier within an ensemble may generalize better than the entire ensemble on some problems, both the risk of an ensemble performing poorly is reduced and its applicability across a range of problems is significantly increased.

Ensemble methods are intrinsically more capable of handling datasets with varying complexities. Large datasets can be efficiently handled by ensemble methods through partitioning. Classifiers can be trained on disjoint or overlapping dataset slices. Frequently, the task can then be reduced to an embarrassingly parallelizable one, through which more tractability can be brought into the learning process.

Periodically, the boundaries separating the classes are too complex to be defined with a given inducer due to overlapping class boundaries, noise and outliers. In these instances, ensemble methods are also able to overcome the learning complexity through divide-and-conquer approaches. With an intelligent decomposition strategy, an ensemble classifier can be designed whereby each constituent classifier learns the class boundary of a different subset of the input space. When combined, the ensemble is then able to represent a comprehensive understanding involving all the class boundaries.

Ensemble methods are also effective at handling problems with sparse data or imbalanced learning which is frequently associated with degradations in accuracy [62]. Ensemble learning lends itself well to such problems since it can be combined with re-sampling techniques. In these cases, diverse classifiers can be generated using unstable inducers to form robust ensembles from randomly generated overlapping datasets. Lastly, ensemble methods enable the fusion of multiple data sources in their training as opposed to single classifiers. This can yield much more accurate classifiers, as well as introduce more efficiency during detection, through coarse-to-fine strategies.

In order to appreciate how ensemble methods realize these benefits, it is important to explore two fundamental components of ensemble learning, namely, ensemble diversity and ensemble output-combination strategies.

### 2.1.1 Ensemble Diversity

In ensemble systems, it is necessary that individual classifiers make classification errors on different instances. Therefore, the goal of ensemble methods is to generate multiple classifiers with sufficiently different class boundaries or inductive biases [105]. Ideally, the individual classifiers will agree on the target class instances and disagree on non-target class instances. With a strategic combination of the classifiers' votes, the collective generalization ability will be better than that of any individual classifier within the ensemble.

The importance of diversity has been demonstrated by Ali and Pazzani [1], Tumer and Ghosh [167], showing how it can reduce the variance error without increasing the bias error. Indeed, diversity has also been shown [6] to decrease bias error as well when it is combined with the theory of large margins.

Ensemble diversity can be obtained using several approaches. The most common method is to manipulate the training datasets for each iteration. Boosting achieves this by altering the instance weights after each round, enabling the inducer to focus on the more difficult samples. Bagging [12] generates a different subset of the original dataset by means of re-sampling with replacement for each round. Both methods employ unstable classifiers as their base models in order to generate appreciatively different decision boundaries even on small perturbations in data and training parameters.

Maimon and Rokach [97] attain diversity through a dataset partitioning strategy which randomly partitions a dataset into $m$ disjoint sets with the difference that all the data is processed. Chawla et al. [21] demonstrate scalability as well as accuracy by applying this method on a distributed environment with thousands of small disjointed subsets. Other methods promote ensemble diversity by manipulating the inducer. In the case of C4.5 [127], this can be done by varying the confidence level parameter which greatly affects the learning. By executing C4.5 with different parameters for a number of iterations, sufficiently diverse decision trees can be generated in order to form an ensemble. Likewise, several neural networks can be trained, whereby diversity is obtained by selecting different topologies [110], varying the number of nodes [69] as well as initialization weights and error goals.

Diversity can also be accomplished by combining altogether different classifiers [103, 178], like neural networks, decision trees, nearest neighbor and SVMs, though this is used only on specific problem domains that warrant them [120]. Similarly, different features can be used on different classifiers. Of this category, the random subspace method [64] has been applied to numerous applications by generating different classifiers through the use of random feature subsets.

#### Measuring Diversity

A number of measures have been proposed in order to provide a quantitative assessment of ensemble diversity. The pairwise approach generates $\frac{L(L-1)}{2}$ diversity values for $L$ number of classifiers. For each comparison of classifiers $i$ and $j$, a $2 \times 2$ matrix is created that records the fraction of instances that both are in agreement in classifying correctly and incorrectly as well is the fraction of instances on which they disagree. From these results, the correlations and the Q-Statistics are calculated. In addition, the probabilities for disagreement and the double fault measures can be deduced.

The non-pairwise measures produce a single value for the entire ensemble and usually the entropy measure and the Kohavi-Wolpert variance is used in this instance. However, determining which diversity measure is the most effective has proven elusive to the research community. After extensive research, Kuncheva [81] concludes that no diversity measure consistently correlates with high accuracy measures.

Inevitably, the question "which ensemble diversity method is best"will be raised and numerous research endeavours have examined this. There is broad agreement that the "no free lunch theorem" [186] applies in this regard [120]. In essence, the most appropriate method depends on the structure of data and prior knowledge while, simplest and least complicated approaches should be given priority according to Occam's Razor.

### 2.1.2 Combining Ensemble Outputs

The second key component of ensemble systems is the aggregation strategy, which combines the individual classifier votes and formulates a final classification decision. These can generally be divided into two categories.

The first category called classifier fusion, considers the decisions of all classifiers and combines them into a single classification. Classifier selection on the other hand, presumes that there is an expert authority with competence to select the most appropriate classifier or classifiers for a given prediction task.

Combination approaches can also be categorized into trainable and non-trainable. For trainable approaches, the combination parameters are determined by an additional algorithm subsequent to the training of an ensemble. The non-training methods generate the necessary combination parameters during the ensemble-generation phase.

Classifier outputs can be either class labels or continuous-valued outputs that represent the degrees of confidence each classifier assigns to class labels. The continuous-valued outputs can be used to estimate the class conditional posterior probabilities on which numerous classifiers rely. However, for the purposes of this discussion, only combination methods for classifiers that output class labels are summarized, since these types of classifiers were exclusively used in this research.

Considerable effort has been invested into determining which ensemble combination methods are the best. There is general agreement that no universally best method exists and that the combination methods are highly dependent on the particular problem at hand [2, 49, 74].

#### Ensemble Fusion

In ensemble fusion, each classifier possesses knowledge about the entire feature space. Therefore, some form of majority or average voting schemes are often employed [81]. Given a data instance $(x_i, y_i)$ and the final ensemble classifier $H(x_i) \in Y$, where $Y \in \{-1, 1\}$ is the domain for a binary class problem, let the prediction of a class label $c$ of the $t^{th}$ classifier $h_t$ be defined as $h(x_i)$ where

$$h_t(x_i) = \begin{cases} 1 & y_i = c \\ 0 & y_i \neq c \end{cases}$$

then *plurality* or *majority voting* can be defined as:

$$H(x_i) = \arg\max_{c=1}^{C} \sum_{t=1}^{T} h_t(x_i). \tag{2.1}$$

Majority voting has been shown to be an optimal combining scheme given minor assumptions that: (1) the ensemble comprises an odd number of classifiers; (2) for any instance $x$, the probability for each classifier selecting the correct class label $c$ is $p$; and (3) that the classifier outputs are independent [120]. This combination scheme is often used as a benchmark against newly proposed combination methods [133].

Variations of majority voting can be defined where the ensemble selects the class $c$ on which every classifier agrees (*unanimous voting*) or as *simple majority*, whereby at least one more than the half of all the classifiers within an ensemble must reach consensus.

If it can be established that the classifiers possess varying degrees of competence, then it is reasonable to expect an improvement in accuracy when greater importance is assigned to more qualified classifiers. *Weighted majority voting* attempts to address these situations and can under these circumstances produce higher accuracies than plurality voting. Supposing there exists a method of quantifying each classifier's future performance, then it is possible to assign a weight $\alpha_t$ to each classifier $h_t$ that is proportional to its estimated accuracy. Therefore, classifiers under this combination scheme predict class $c$ for an instance $x_i$ if

$$H(x_i) = \arg\max_{c=1}^{C} \sum_{t=1}^{T} \alpha_t h_t(x_i) \tag{2.2}$$

where the total weighted vote for class $c$ is greater than that received by any other class.

Different methods exist for estimating classifier competencies for the assignment of weights. Algorithms like AdaBoost determine classifier competencies during the ensemble generation process and base the weights on the performances of the classifiers on the training data. The $\alpha_t$ value is calculated as $\frac{1}{2}log(\frac{1-\varepsilon_t}{\varepsilon_t})$, where $\varepsilon_t$ represents the weighted training error of a classifier $h_t$ on round $t$.

Methods like *performance weighting* have the ability to decouple the ensemble generation phase and the weight assignment procedure. With this approach, weights can be set proportional to the accuracy of each classifier on a validation set as a *post-hoc* procedure [110] after the ensemble has been generated. The weight $\alpha_t$ can be defined as

$$\alpha_t = \left( \frac{(1-\varepsilon_t)}{\sum_{j=1}^{T}(1-\varepsilon_j)} \right). \tag{2.3}$$

**Ensemble Selection**

The theory behind ensemble selection is to devise a scheme which is capable of selecting a classifier or classifiers that are most capable for evaluating a given instance $x$. The concept works well when classifiers are trained to specialize on different regions of the input space and are assigned the sole responsibility for predicting objects within them. This approach was first suggested by Dasarathy and Sheela [28], and was used by them to combine a

linear classifier and a $k$-nearest neighbor classifier. Though ensemble selection strategies have not been as widespread as ensemble fusion, Kuncheva [81] argues that it is likely the better of the two, provided that the classifiers are well trained.

Sometimes expert classifiers for a given instance $x$ are identified dynamically at runtime. Such methods are referred to as *a priori*, since they identify competent classifiers solely on the location of the instance $x$ in the input space, prior to determining what class labels the classifiers might suggest. Rastrigin and Erenstein [128] demonstrated this strategy using the $k$-NN method. Based on the location of $x$, this method locates $k$ nearest *classifiers* from which the most accurate is selected. The information on which classifiers represent the $k$ nearest neighbors for given inputs $x$, is gathered from the training or validation sets.

Dynamic methods like this suffer from increased computational loads during runtime. Alternatively, regions of competence can be estimated before the classification phase. Frequently, the training datasets are partitioned into such regions and a classifier or a group of classifiers are trained on them as experts. The partitioning process may divide the datasets horizontally as subsets of rows, or vertically as subsets of attributes. Kuncheva [78, 79] proposes a clustering and selection algorithm whereby the $k$-means procedure is used to create competence regions for specialized classifiers. The drawback however, is that the effectiveness of such strategies tends to depend more on the manner in which the clusters are created, rather than on the process of accurately estimating the competence levels between the regions [81].

This research relies extensively on the concept of ensemble selection that is inherent within the cascaded classifiers and is discussed in greater detail later in this chapter. In essence, cascade classifiers can be categorized as ensemble selection methods since they are ordered into hierarchical layers. In order to classify an instance $x$, the first layer is evaluated against this instance. If the confidence of its prediction exceeds a given threshold, then the classifier assigns a label to the instance $x$, otherwise the prediction is forwarded to the next layer in the hierarchy. Each layer is trained on datasets that represent different portions of the input space and in the case of cascades, this input space is of increasing complexity. Thereby, a more informed decision can be made for an instance $x$ as it is passed to succeeding layers.

The mixture-of-experts [70] algorithm is a sophisticated ensemble selection technique. It trains a *gating network* which serves as a dynamic rule-combination method. The gating network is learned using training data and it selects from different possible combination strategies to produce the most accurate instance-specific results. The mixture-of-experts has been found to be effective for classification problems where certain classifiers in unison, consistently classify given instances correctly and incorrectly [133].

### 2.1.3 Optimal Ensemble Training Procedures

Determining the number of classifiers to train is a crucial component of ensemble learning. The size of an ensemble has an effect on both the generalization, training and detection runtimes. There are three general categories into which strategies for deciding the ensemble size can be placed. Firstly, the size of an ensemble can be determined prior to training. Many ensemble based algorithms provide tunable parameters for specifying the number of iterations that are to be executed. This parameter can be set by a practitioner for

different problem domains. AdaBoost and Bagging are typical algorithms that provide these parameters. In the case of cascade training, both the total number of classifiers within each layer and the total number of layers can be bounded.

Strategies also exist for determining the ensemble size dynamically during the training. Usually these strategies continue training based on the criterion that each additional classifier contributes to an increased accuracy measure. Once this condition can no longer be satisfied, the training finishes [13]. Kuncheva [81] offers a strategy for combining diversity measures directly into ensemble design during the training runtime. The proposal is to employ the kappa-error diagram and the Pareto optimal set for calculating the degree of diversity being contributed by each classifier, with the condition of halting training once the diversity fails to increase.

Just as tree induction sometimes benefits from pruning, the same principle can be applied to ensembles. Zhou and Chen [198] proved that it is indeed possible to select out a subset of the original ensemble which is better than the original. Subsequent research by Liu et al. [91] showed that small subsets of the ensemble can preserve the accuracy and encapsulate the diversity of the larger ensemble. These successes spawned numerous ensemble *thinning* approaches; however, the two most popular approaches remain ranking-based and search-based methods [134].

Ranking based methods operate on the level of individual classifiers. Each classifier is ranked individually based on a criterion and the best classifiers above a certain threshold are selected. A separate validation set can be used for ranking classifiers based on their accuracy.

Search-based methods tend to consider the collective accuracies of subsets of classifiers over the search space of all possible ensemble subsets [198]. Rokach [132] proposes ranking the classifiers based on their receiver operating curve (ROC) performance. The top classifiers are selected to form the initial ensemble subset. Subsequently, a search over the remaining classifiers is performed. New classifiers are appended to the subset until several successive additions no longer improve the performance.

Clustering-based pruning methods on the other hand, perform an additional phase prior to the thinning procedure. The first phase clusters the classifiers based on the double fault measure of diversity from the matrices of pairwise comparisons [55], or a $k$-means algorithm [83].

Determining which of the three strategies for designing ensembles is universally the most effective, remains an open question. Choosing an appropriate strategy depends on the problem at hand. Using a predetermined number of classifiers prior to training may result in inefficient and redundant classifiers. However, dynamic halting criteria risks creating poor classifiers if the training is terminated after initially generating several overfitted classifiers, despite the ability of subsequent classifiers to produce the best accuracy [102]. Finding optimal subsets that match or improve the accuracies of the original ensemble is NP-hard [98]. This becomes intractable on complex real-world problems that require thousands of classifiers for a given classification task. At least in the case of AdaBoost, classifier thinning has not been found to be effective for accuracy improvements [98]. It has been suggested that this is likely to be due to the fact that boosting amplifies diversity by design and as a result is resistant to redundancy Kuncheva [81] .

## 2.2 Boosting

Boosting [139] is a general method for iteratively improving the accuracy of weak learners. Schapire proved that weak learners, that are only slightly better than a random guess, can be combined to constitute strong learners which in turn yield classifiers that are capable of arbitrarily high accuracy rates. The initial boosting algorithm has been described as one of the most significant developments in the recent history of machine learning.

Boosting proceeds to generate ensembles by re-sampling the data and combining the outputs through majority voting similar to Bagging. However, boosting differs in that it strategically re-samples data in a way that is geared to focus the learning on more difficult samples.

The original boosting algorithm creates three weak classifiers. The first weak classifier $c_1$, is trained on a randomly generated subset $S_1$. The next classifier $c_2$ is trained on a more informative subset $S_2$, which comprises of instances where the first half are correctly classified by $c_1$ and the other half are misclassified by $c_1$. The final classifier $c_3$ is learned from instances of $S_3$, which consists of instances that both $c_1$ and $c_2$ disagree on.

Given the error $\varepsilon_t$ of the best performing classifier $t$ and provided that all classifiers generate $f(\varepsilon) < 0.5$, Schapire demonstrated that the error of the boosting algorithm is bounded from above by

$$f(\varepsilon_t) = 3\varepsilon_t - 2\varepsilon_t^3 \tag{2.4}$$

meaning that the boosted ensemble will always perform better than the best individual classifier. Due to this remarkable property, the concept of boosting has attracted immense research. The most celebrated product of this has been AdaBoost which was initially designed to handle two-class problems. Its intuitiveness and sound performance guarantees have seen it described as being one of the most influential algorithms in computational intelligence [120]. Its success and popularity has spawned a vast number of variant algorithms as well as extensions of it to multiclass problems that are discussed later in the chapter.

### 2.2.1 Binary Class Problems

AdaBoost (Algorithm 1) improves on the original boosting algorithm by introducing an iterative process. The key departure from the original boosting algorithm lies in its usage of non-uniform distributions $D$ of the data to replace re-sampling and the use of a weighted majority combination instead of majority voting.

AdaBoost constructs an ensemble initially with a uniform distribution of weights over all instances. On each distribution $D_t$, a weak classifier $h_t$ is induced and added to the ensemble. A weighted error $\varepsilon_t$ over $D_t$ is then calculated and verified that it is $\varepsilon_t < \frac{1}{2}$. The voting weight of $h_t$ is calculated as being inversely proportional to $\varepsilon_t$ and is assigned to $h_t$ as its estimated predictive competence. Subsequently, the distribution $D_{t+1}$ is updated whereby the weight of each misclassified sample is increased in proportion with the accuracy of $h_t$. The weights are normalized with the effect that the weight of the correctly classified samples is decreased. With this mechanism, the learning for each distribution $t + 1$, greater learning emphasis is placed on more difficult samples by compensating for previous mistakes.

**Algorithm 1**: AdaBoost

**Input** : A set of training examples $(x_1, y_1), ...(x_n, y_n)$ where $x_i$ is a feature $\in$ X and $y_i$ is a class $\in \{-1, +1\}$, $n$ is the number of elements and with $T$ as the number of boosting rounds.

**Output**: A linear combination of weak classifiers $h_t(x_i)$, each a factor $\alpha_t$.

$D_1(x_i) = \frac{1}{n}, i = 1, ..., n$
**for** $t = 1$ **to** $T$ **do**
  $h_t = \text{WeakLearn}(D_t)$
  $\varepsilon_t = \sum_i^n (D_t(x_i) h_t(x_i) y_i)$
  $\alpha_t = \frac{1}{2}\ ln(\frac{(1-\varepsilon_t)}{\varepsilon_t})$
  **for** $i = 1$ **to** $n$ **do**
    $D_{t+1}(x_i) = D_t(x_i)\ exp(-\alpha_t y_i h_t(x_i))$
  **end**
  $D_{t+1}(x_i) = \frac{D_{t+1}(x_i)}{\sum_i^n D_{t+1}(x_i)}$
**end**
**return** $H(x) = \text{Sign}(\sum_t^T \alpha_t h_t(x))$

At classification time, an instance $x_i$ is evaluated by every weak classifier $h_t$. Each evaluation $h_t(x_i)$ returns a value for one of the class labels $\{-1, 1\}$, that is multiplied by the confidence coefficient $\alpha_t$. The sum of all terms results in either a positive or a negative value. The *sign* of this sum signifies the class label that the ensemble has voted on.

The repeated distribution updates result in the weak classifiers sequentially minimizing the exponential bound on the error rate. Provided that $\varepsilon_t < \frac{1}{2}$ for each $h_t$, the upper bound on the weighted training error is guaranteed to decrease monotonically with each additional $h_{t+1}$. Generalization bounds have also been developed from the perspective of *voting margins*. For a two-class problem where $y \in \{-1, 1\}$, the voting margin $m$ is defined as

$$m = \left( \frac{y_t \sum_{\text{t=1}}^{T} \alpha_t h_t(D)}{\sum_{\text{t=1}}^{T} |\alpha_t|} \right) \tag{2.5}$$

where $m$ is positive if and only if the classifier $H$ correctly classifies a sample. Moreover, the value $m$ represents the degree to which the average number of weighted votes over $D$ for a correct class label, exceeds the average weighted vote for the incorrect class labels. As the voting margin increases, the degree of confidence in the classification increases and the expected test error decreases [141].

Subsequent research in the last decade has witnessed a plethora of AdaBoost variants: RealBoost [45] extends the classification outputs into confidence-rated predictions; Gentle AdaBoost [46] enables more resilience to outliers; Asymmetric AdaBoost modifies the re-weighting scheme to favour positive samples; Float Boost [88] aims to reduce ensemble redundancy through pruning; Local boosting [196] reports higher accuracy and robustness to noise through boosting-by-resampling; instance-based on-line boosting [85, 111]; Bühlmann and Hothorn [18] propose Twin Boost which reportedly has an improved fea-

ture selection mechanism that yields both simpler and more accurate ensembles.

### 2.2.2 Multiclass Problems

Multiclass classification presents considerably greater accuracy and performance challenges than binary-class problems since the probability of making an error is higher. A common approach has been to decompose multiclass problems into a series of more manageable binary subproblems instead of applying explicit multiclass-algorithms directly [94]. There are a number of methods available for achieving this decomposition.

#### Multiple Binary Classifiers

Traditionally, the most popular decomposition approaches have been the one-against-all (OAA) and one-against-one (OAO) strategies [3, 86]. The OAA approach constructs $k$ number of classifiers for a $k$-class problem. Each classifier is trained to distinguish each class $i$ from the remaining classes in which case a binary-class boosting algorithm suffices.

At detection time, the ideal outcome of the combination of classifiers is a unique vote for a single predicted class. Since this is often not the case, ties are either broken arbitrarily [58], or an additional value that signifies confidences of the classifiers is derived and used to resolve conflicting predictions [77]. A known disadvantage of this method is the unbalanced nature of the training sets for the classifiers, which may complicate the induction of accurate class decision boundaries [62, 72, 134].

The OAO approach, known also as the round robin classification, proceeds to binarize the multiclass problem into a series of pairwise classification sub-problems. Each class $i$ is trained to differentiate between all other individual class instances. This results in $\frac{k(k-1)}{2}$ number of classifiers being generated. Although OAO generates a total number of binary classifiers which is in the order of $k^2$, the actual training runtime is usually not excessive since the training task of differentiating only two classes is much simpler. Due to the fact that there are fewer instances being considered, the inducer has a much greater freedom in fitting the decision boundary between two classes than between the entire set of classes [50].

The final prediction of OAO classifiers is most commonly achieved through a majority voting scheme [77]. A crucial shortcoming with the majority voting scheme in this context, is that many votes are generated by classifiers which have been trained on pairs of irrelevant class labels. Consequently, the votes from irrelevant classifiers are also integrated into the final decision which can adversely influence the final result. Platt et al. [117] devise a hierarchical strategy to integrate the decision making process of OAO classifiers through directed acyclic graphs (DAG), which resemble binary decision trees. The DAG-based strategy works by propagating decisions through the tree by executing a classifier to differentiate class $i$ from class $j$ at each node. The decision at each node is concluded as *not class i*, when $j$ is predicted and vice versa until the leaf nodes are reached. Platt et al. [117] show that the DAG approach is efficient and accurate at aggregating decisions and demonstrate how an uncompromised accuracy can be achieved without executing all classifiers within an ensemble.

### Error Correcting Output Codes

Dietterich and Bakiri [30] applied principles from coding theory in the form of error-correcting output codes (ECOC) usually associated with communication across noisy channels, to the problem of multiclass decomposition and classification resolution. ECOCs have since experienced widespread use and success on many applications.

The ECOC method constructs a coding matrix, an example of which can be seen in Table 2.1. The matrix provides a scheme for training classifiers. The scheme specifies the different grouping combinations of positive and negative sets that the $k > 2$ classes are to be reduced to. For a $k$ class problem, $k$ rows are created for each class. $n > k$ columns are generated, where each column represents one classifier $f_i$. The bit values in each column $i$ define a training scheme for relabeling the set of multiclass labels into binary classes. Each row then becomes a unique codeword for a given class label. Given an unseen input $x$, each binary classifier corresponding to a column with output $f_i(x) \in \{0, 1\}$, is executed. Their combined output produces a binary codeword. This output code is then matched against the rows in the coding matrix to provide the resolution of the predictions. The class with the closest matching codeword according to the Hamming distance becomes the classified label.

Table 2.1: Example of a coding matrix for a four class problem.

| | classifiers | | | | | |
|---|---|---|---|---|---|---|
| class | $f_1$ | $f_2$ | $f_3$ | $f_4$ | $f_5$ | $f_6$ |
| $c_1$ | 1 | 1 | 1 | 1 | 1 | 1 |
| $c_2$ | 0 | 0 | 0 | 0 | 1 | 1 |
| $c_3$ | 0 | 0 | 1 | 1 | 0 | 0 |
| $c_4$ | 0 | 1 | 0 | 1 | 0 | 1 |
| output | $f_1(x)$ | $f_2(x)$ | $f_3(x)$ | $f_4(x)$ | $f_5(x)$ | $f_6(x)$ |

The quality of the coding matrix is crucial to the success of the method. The effectiveness of the coding matrix can be assessed using two criteria. The first is the row separability. The length of the codewords $n$ must be larger than the minimum required to uniquely differentiate between all the classes. The extra redundancy in the coding vectors at each row enables the classes to be more separated from each other according to the Hamming distance. This redundancy and distance allows for the error correction or compensation to take place in the presence of misclassifications by several classifiers. The second is the diversity of the columns, whereby the columns are required to be as uncorrelated as possible.

Dietterich and Bakiri [30] initially showed how the error correcting properties of codewords could be effectively unified with multiclass training, whereby for each column of the coding matrix, the classifier learns to discriminate between two subsets of all classes. Allwein et al. [3] then generalized the approach further by demonstrating how the two subsets of each bit vector no longer had to represent all possible classes.

### Multiclass Boosting

Initial multiclass extensions of AdaBoost, resulted in AdaBoost.M1[44]. The primary obstacle to initial forms of multiclass AdaBoost, was its requirement that the weak learner

generate weighted errors at each round that are better than $\frac{1}{2}$. For most multiclass problems of $k > 2$, this requirement is much harder to satisfy than random guessing of $\frac{1}{k}$. Consequently, stronger and more computationally intensive weak learners like C4.5 had to be used. As a byproduct, the training runtimes increased. More recently, Eibl and Pfeiffer [35] and Zhu et al. [199] proposed AdaBoost.M1W and SAMME respectively. These modifications enabled AdaBoost to operate with weak learners that produce errors which are slightly better than random guessing of $\frac{1}{k}$.

Freund and Schapire [45] also proposed a powerful boosting algorithm called AdaBoost.M2 which is designed to overcome the rigid error demands of its early predecessors. They designed an algorithm whose strength lies in that it not only learns the difficult samples, but it also learns from incorrect class labels, where a distribution is introduced for mislabeled instances to couple the learning and the boosting process. This is achieved by introducing the idea of a *plausible* set $\mu$ where $\mu \subset Y$ and $Y$ is the entire set of class labels. The weak hypothesis can then make a binary prediction for a given sample $x$, in which it only has to predict whether a sample belongs to the plausible set or not. The output of the weak hypothesis is a *pseudoloss* measure. The hypothesis with the lowest pseudoloss over all possible combinations of the plausible set, becomes the hypothesis for round $t$. In this manner, AdaBoost.M2 guarantees that the best discriminating hypothesis between all classes is selected at each round.

Due to its strength as a multiclass learning algorithm, AdaBoost.M2 is used as one of the control methods in this research. Algorithm 2 lists the steps where the notation $[\![\pi]\!]$ used here and subsequently, is defined as *1* if proposition $\pi$ holds, otherwise *0*.

**Algorithm 2**: AdaBoost.M2

**Given**: $(x_1, y_1), ..., (x_m, y_m)$ where $x_i \in X, y_i \in Y$ to make uniform over all incorrect labels

**Output**: Hypothesis $H_{final}(x) = \arg\max_{\ell \in Y} \sum_{t=i}^{T} \alpha_t [\![\ell \in \tilde{h}_t(x)]\!]$

Initialize $\tilde{D}_1(i, \ell) = [\![\ell \neq y_i]\!]/(m(k-1))$

**for** $t = 1$ **to** $T$ **do**

- Train weak learner using pseudoloss defined by $\tilde{D}_t$.
- Get weak hypothesis $\tilde{h}_t : X \rightarrow 2^Y$
- Let $\tilde{\epsilon}_t = \frac{1}{2} \sum_{i=1}^{m} \sum_{\ell \in Y} \tilde{D}_t(i, \ell) \cdot ([\![y_i \notin \tilde{h}_t(x_i)]\!] + [\![\ell \in \tilde{h}_t(x_i)]\!])$
- Let $\alpha_t = \frac{1}{2} \ln(\frac{1-\tilde{\epsilon}_t}{\tilde{\epsilon}_t})$
- Update $\tilde{D}_{t+1}(i, \ell) = \frac{1}{Z_t} \cdot \tilde{D}_t(i, \ell) \exp(\alpha_t([\![y_i \notin \tilde{h}_t(x_i)]\!] + [\![\ell \in \tilde{h}_t(x_i)]\!]))$
- where $Z_t$ is the normalization factor so that $\tilde{D}_{t+1}$ will sum to 1.

The first three steps of Algorithm 2 are tightly bound and represent the enhanced communication between the weak learner and boosting. For each round of boosting, a weak hypothesis is generated for each permutation of a plausible set $\mu_t \in 2^Y$. The goal of the learner is to minimize the pseudoloss defined as:

$$\tilde{\epsilon}_t = \frac{1}{2} \sum_{i=1}^{m} \sum_{\ell \in Y} \tilde{D}_t(i, \ell) \cdot ([\![y_i \notin \tilde{h}_t(x_i)]\!] + [\![\ell \in \tilde{h}_t(x_i)]\!])$$

where the loss measure penalizes the hypothesis for not including the correct label $y_i$ belonging to sample $x_i$ in the plausible set as $y_i \notin \tilde{h}_t(x_i)$, and additionally penalizes each incorrect label $\ell \neq y_i$ that is part of the plausible set as $\ell \in \tilde{h}_t(x_i)$. The $\alpha_t$ value

is calculated as in binary AdaBoost, after which the algorithm proceeds to update the distribution using the same approach used for calculating the pseudoloss measure. The final hypothesis classifies an instance according to the label which occurs most frequently in the plausible sets that is selected by the weak hypotheses.

Though experiments with AdaBoost.M2 have indicated that it performs very well, there are some drawbacks to it. The first is its training runtime [44, 58]. From line 2 in Algorithm 2, it becomes clear that the number of possible plausible sets rises exponentially with the number of class labels in the training set. An exhaustive search over the entire possible range of permutations of the plausible set rapidly becomes unfeasible for datasets containing class labels $k > 10$. The second disadvantage is encountered in the process of implementing the algorithm. Since the boosting and the weak learner are tightly coupled, it requires modifications to the weak learner in order to enable it to use the pseudoloss measure which complicates the implementation.

A common criticism of the two-stage decomposition approach to training classifiers independently and then combining their prediction votes as seen in OAA, OAO and ECOC is that the binarizing step is done *a priori* without considering the properties and characteristics of a dataset [3, 72, 94]. Furthermore, Allwein et al. [3], Jin and Zhang [72] point out that in respect to ECOC, some artificially created binary sub-problems may be difficult to learn due to complicated multimodal decision boundaries. Finding an optimal coding matrix for ECOC methods has been shown to be NP-hard [27]; however, some heuristics have been proposed. The most popular of these are AdaBoost.OC (output coding) [44] and AdaBoost.ECC (error-correcting code) [58], which have effectively unified boosting with ECOC [134]. Due to their widespread usage and the still ongoing research surrounding them [72, 86, 134, 154], both boosting algorithms are implemented in this research as baseline models.

**Algorithm 3**: AdaBoost.OC

**Given**: $(x_1, y_1), ..., (x_m, y_m)$ where $x_i \in X, y_i \in Y$ to make uniform over all incorrect labels

**Output**: Hypothesis $H_{final}(x) = \arg\max_{\ell \in Y} \sum_{t=i}^{T} \alpha_t [\![h_t(x) = \mu_t(\ell)]\!]$

1 Initialize $\tilde{D}_1(i, \ell) = [\![\ell \neq y_i]\!]/(m(k-1))$
2 **for** $t = 1$ **to** $T$ **do**
3 Compute colouring $\mu : Y \to \{0, 1\}$
4 Let $U_t = \sum_{i=1}^{m} \sum_{\ell \in Y} \tilde{D}_t(i, \ell) [\![\mu_t(y_i) \neq \mu_t(\ell)]\!]$
5 Let $D_i = \frac{1}{U_t} \cdot \sum_{\ell \in Y} \tilde{D}_t(i, \ell) [\![\mu_t(y_i) \neq \mu_t(\ell)]\!]$
6 Train weak learner on examples $(x_1, \mu_t(y_1)), ..., (x_m, \mu_t(y_m))$ weighted according to $D_t$
7 Get weak hypothesis $h_t : X \to \{0, 1\}$
8 Let $\tilde{h}_t(x) = \{\ell \in Y : h_t(x) = \mu_t(\ell)\}$
9 Let $\tilde{\epsilon}_t = \frac{1}{2} \sum_{i=1}^{m} \sum_{\ell \in Y} \tilde{D}_t(i, \ell) \cdot ([\![y_i \notin \tilde{h}_t(x_i)]\!] + [\![\ell \in \tilde{h}_t(x_i)]\!])$
10 Let $\alpha_t = \frac{1}{2} \ln(\frac{1 - \tilde{\epsilon}_t}{\tilde{\epsilon}_t})$
11 Update $\tilde{D}_{t+1}(i, \ell) = \frac{1}{Z_t} \cdot \tilde{D}_t(i, \ell) \exp(\alpha_t([\![y_i \notin \tilde{h}_t(x_i)]\!] + [\![\ell \in \tilde{h}_t(x_i)]\!]))$
12 where $Z_t$ is the normalization factor so that $\tilde{D}_{t+1}$ will sum to 1.

AdaBoost.OC and AdaBoost.ECC merge ECOC principles with boosting and also

overcome significant computational drawbacks of AdaBoost.M2. The ECOC-based boosting methods iteratively generate columns of the coding matrix so that the confusion between classes is reduced at each boosting round. Algorithm 3 describes the detailed steps of AdaBoost.OC, while theoretical proofs for it can be found in [44].

AdaBoost.OC proceeds by first generating a coding function that the authors refer to as the *colouring* $\mu$. The colouring function generates the columns of the coding matrix and maps all class labels $Y \rightarrow \{0,1\}$. Similar to AdaBoost.M2, AdaBoost.OC maintains an additional distribution $\tilde{D}$ at each round $t$. By applying the colouring function $\mu_t$ to $\tilde{D}$, value $U_t$ is generated which represents the error correcting ability of the column matrix $\mu_t$. $\mu_t$, $U_t$ and the distribution $D_t$ are then combined to compute the weight of distribution $D_t$. A binary classifier $h_t$ is subsequently learned on the adjusted distribution $D_t$. This differs from AdaBoost.M2 which learns on $\tilde{D}_t$. The pseudo error $\tilde{\epsilon}_t$ is then used as in AdaBoost to produce the confidence value $\alpha_t$. Lastly, the distribution $\tilde{D}$ is updated as in AdaBoost.M2 with $\alpha_t$. The final hypothesis $H$ on sample $x$ is computed as being the class label $l$, which receives the highest weighted vote from all class labels returned by $h_t(x)$.

AdaBoost.ECC resembles AdaBoost.OC in many respects (Algorithm 4), but differs in that it does not calculate the $\alpha_t$ value based on the pseudo-error at each round $t$. Instead, it selects $\alpha_t$ and $\beta_t$ values at each round which represent the positive and negative votes of the hypothesis $X \rightarrow \{-1, 1\}$ on the binary problem. According to its authors [58], it thus represents a more direct reduction of multiclass learning to the binary form. At classification, the function $g_t(x) \cdot \mu_t(l)$ returns the vote of the value for a class label $l$. The final hypothesis $H(x)$ outputs the class label $y$ which scores the maximum sum of all the constituent hypotheses $g_t(x)$. The authors maintain that AdaBoost.ECC is a computationally simpler algorithm, which in their research has displayed superior results over AdaBoost.OC.

**Algorithm 4**: AdaBoost.ECC

**Given**: $(x_1, y_1), ..., (x_m, y_m)$ where $x_i \in X, y_i \in Y$ to make uniform over all incorrect labels

**Output**: Hypothesis $H_{final}(x) = \arg\max_{\ell \in Y} \sum_{t=i}^{T} g_t(x)\mu_t(\ell)$

Initialize $\tilde{D}_1(i, \ell) = [\![\ell \neq y_i]\!]/(m(k-1))$

**for** $t = 1$ **to** $T$ **do**

- Compute colouring $\mu : Y \rightarrow \{-1, 1\}$
- Let $U_t = \sum_{i=1}^{m} \sum_{\ell \in Y} \tilde{D}_t(i, \ell)[\![\mu_t(y_i) \neq \mu_t(\ell)]\!]$
- Let $D_i = \frac{1}{U_t} \cdot \sum_{\ell \in Y} \tilde{D}_t(i, \ell)[\![\mu_t(y_i) \neq \mu_t(\ell)]\!]$
- Train weak learner on examples $(x_1, \mu_t(y_1)), ..., (x_m, \mu_t(y_m))$ weighted according to $D_t$
- Get weak hypothesis $h_t : X \rightarrow \{-1, 1\}$
- Compute the weight of positive and negative votes $\alpha_t$ and $\beta_t$ respectively
- Define: $g_t(x) = \begin{cases} \alpha_t & \text{if } h_t(x) = 1 \\ \beta_t & \text{if } h_t(x) = -1 \end{cases}$
- Update $\tilde{D}_{t+1}(i, \ell) = \frac{1}{Z_t} \cdot \tilde{D}_t(i, \ell) \exp\{(g_t(x_i)\mu_t(\ell) - g_t(x_i)\mu_t(y_i)) \cdot \frac{1}{2}\}$
- where $\tilde{Z}_t$ is the normalization factor so that $\tilde{D}_{t+1}$ will sum to 1.

## 2.3 Emerging Challenges and Coarse-to-fine Learning

In recent years, automated acquisition and storage of new data has resulted in large and evolving training datasets becoming common place. Streaming data has also become more ubiquitous. As a result, machine learning is presently confronted with significant new challenges arising from changes in the nature of data and its volume. In order to tackle these challenges, ensemble learning has been greatly aided by being combined with coarse-to-fine strategies. These challenges are summarized as:

1. the computational costs of learning
2. the classification runtime-constraints
3. timely and accurate class-boundary induction from large datasets
4. classification accuracy on datasets with imbalanced class-distributions
5. classifier adaptation in evolving and non-stationary environments

With the growth of training data, the computational complexity of large-scale learning has become a limiting factor [9]. Strategies which attempt to overcome the intractability of some learning tasks are invariably confronted with the trade-offs in respect to the accuracy of the resulting classifier [129].

In addition, the bottleneck in many large-scale classification problems is the prohibitive cost of classification at runtime due to the combination of large classifier ensembles and computationally intensive feature extraction. These problems also present a fundamental trade-off between the learning complexity and classification error, due to runtime-computational constraints that need to be considered when designing more efficient classifiers [180].

Complexity of learning can also be viewed from the perspective of class boundary induction with weak learners. The ability of a given base learner on some datasets may not be strong enough to separate the classes [45]. The separability of classes often becomes a problem within a multiclass setting when striving to maintain low computational costs on the induction algorithm. The trade-off arises between using more powerful tree induction algorithms like C4.5, which may result in an improved accuracy, or sacrificing a degree of accuracy in favour of faster learning [44, 58].

Induction on datasets with highly skewed class-distributions comes with unique difficulties. Research in this area has been ongoing and while many methods have been proposed in order to mitigate the challenges, unresolved questions remain. The primary issue is the ability of imbalanced datasets to seriously compromise the accuracy of most standard learning algorithms [62, 71].

Simply put, a class imbalance problem can be defined as an unequal distribution of values between classes. This is referred to as the *between-class* imbalance. However, there are different forms of class imbalances and they are all not necessarily detrimental [7]. In order for a compromise in accuracy to occur in class-imbalanced datasets, the presence of another form of skew termed as *within-class* imbalance needs to take place. The within-class imbalance defines cases when a minority class is represented by an additional subclass. The within-class variability worsens predictive performance due to the generation

of too specific rules around severely underrepresented subclasses of a minority class. In addition, if the minority class also has noisy samples, this results in over-fitted rules that further lead to increased error rates [62].

Streaming data is by nature evolving and non-stationary. As such, it poses significant problems to statically trained classifiers [31, 99, 100, 177]. In numerous domains, additional knowledge about a target concept needs to be incorporated through new object descriptors as they become available. In other situations, aspects of a target concept are rendered irrelevant as the class descriptors for the target concepts morph with time due to unforeseen changes in the operating environment. These emerging problems require that the learning algorithms acquire some measure of adaptability. In many domains, the adaptability needs to be efficiently realized without resorting to retraining of an entire ensemble and thereby discarding all previously learned information and requiring lengthy re-training runtimes [177].

There exist a number of strategies that can loosely be termed coarse-to-fine (CTF) learning methodologies. CTF learning combines ensemble approaches with higher levels of abstraction and hierarchies in order to explicitly or implicitly address some of the above problems. The strategy of CTF methodologies is to iteratively generate intermediate solutions to a problem, which are increasingly specific and more refined at each step.

CTF learning is beginning to gain more traction within the ensemble systems community due to its successes in tackling some difficult problems within computer vision and natural language processing [115]. Though CTF methods have not yet received sufficient study and been formalized as a general machine learning problem, there is increased focus on this task as evidenced by specialized Coarse-to-Fine Learning and Inference international workshops [82].

### 2.3.1 Cascades of Ensembles

A conventional non-cascaded ensemble classifier $H$, can be said to consist of $T$ weak classifiers $h_{i,\ldots,T}$. During the training phase, if boosting is used as a method of diversity, then each weak classifier is exposed to every training sample. At classification, the weak classifiers are combined linearly, where the evaluation of each $h_i(x)$ is necessary in order to formulate a decision for $H(x)$. Ensemble classifiers of this type can be described as single-layer or monolithic classifiers.

#### Cascaded Classifier

A cascaded classifier on the other hand, is comprised of $H_{i,\ldots,L}$ layers. Each layer $H_i$ consists of $T$ weak classifiers $h_{i,j,\ldots,T}$. In the training phase, the weak classifiers $h_{i,j}$ are not exposed to the same training samples. Given a complete negative set $\mathbb{N}$, the classifiers at each layer $i$ are trained only on a subset of the whole, where $N_i \subset \mathbb{N}$, while the positive set $P$ is unaltered. Initially, $N_i$ is generated randomly, and subsequently the intermediate classifier $H_i$ is used to replace the samples from $N_i$ which it correctly classifiers, with new samples from $\mathbb{N}$ that it misclassifies. As a result, each layer $i$ learns to differentiate samples of increasing difficulty and the decision boundary is refined from coarse to higher levels of precision.

Each layer $i$ also represents an independent round of boosting, whereby the distribution

**Algorithm 5**: Viola-Jones Cascade

**Given**: $P, \mathbb{N}, N_i$ denote sets of positive and two negative datasets respectively, where $N_i$ is the negative set used on layer $i$ and $N_i \subset \mathbb{N}$. $\langle f_i, d_i \rangle$ denote the pair of false positive and hit rates respectively for a given layer $i$

**Output**: Cascade classifier $H()$ comprised of $H_{i,\dots,L}$ layers, each consisting of $h_{i,j,\dots,T}$ weak classifiers and a layer threshold $\theta_i$.

```
H = InitializeCascadeClassifier()
for each layer i to L do
    N_i = Bootstrap(ℕ, H)
    ⟨P, N_i⟩ = InitializeWeights()
    j = 0
    repeat
        h_{i,j} = WeakLearn(P, N_i)
        ⟨P, N_i⟩ = ReweightSamples(h_{i,j})
        θ_i = FindBestThreshold()
        ⟨f_i, d_i⟩ = Validate(P, N_i)
        j = j + 1
    until j > T or LayerTargetSatisfied(f_i, d_i) ;
    H_i = ⟨h_i⃗, θ_i⟩
```

weights of each training sample are reset. The training error targets for each layer are simplified so that the requirement is revised to only learning 50% of the negatives from $N_i$. The targets for the hit rates are ordinarily set very close to 100%.

The hit rate target presents a problem especially as the latter layers are trained and as the samples become harder to differentiate. In order to solve this, an artificial layer threshold (or bias) value $\theta_i$ is adjusted for each layer. In effect, this value lowers the confidence margin and allows a higher number of positive samples to be accepted, which enables targets to be met. The trade-off is that a lot more false positives are also introduced, meaning that after each $h_{i,j}$ is trained, the threshold value needs to be manipulated in an *ad hoc* manner until the false positive rates also fall within the acceptable targets. The final TPR and FPR of the whole classifier are the product of the corresponding rates in each layer $H_i$ [171]. Algorithm 5 details the steps for producing the Viola-Jones cascade classifier.

At detection time, the cascaded classifier has a form of a degenerative decision tree [126]. The output of one layer acts as input to the next layer. Each successive classifier is designed to make predictions which are closer to the class decision boundaries. A positive classification of the cascaded classifier $H$ is a conjunction of all classifier levels where $\forall i, H_i(x) = 1$. If the prediction of any layer is $H_i(x) = 0$, then succeeding layers are not invoked and the classification for the negative class is adopted.

The initial motivation behind the cascaded classifiers was for the realization of real-time detection. The cascaded classifiers as such succeeded in providing robust solutions to requirements of classification under runtime constraints. The underlying strength of the cascades is their ability to acquire features for classification on demand. The outcome is that all features do not have to be calculated for each sample. Computationally efficient classifiers are usually sufficient to provide a reliable judgment at a coarse level, while if it

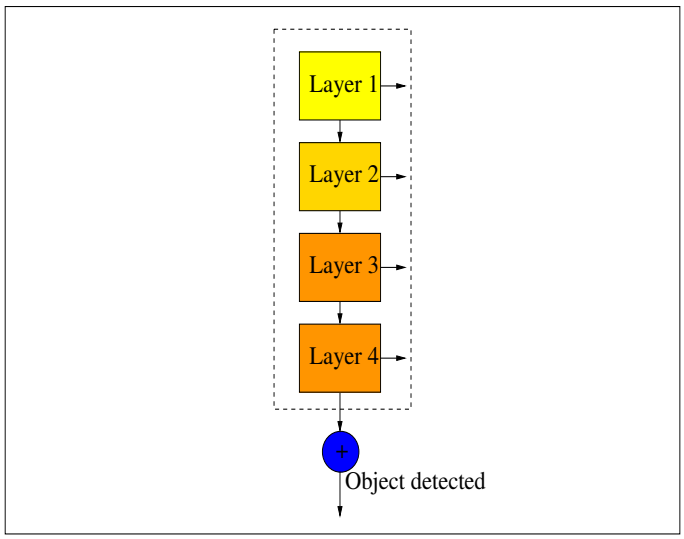



Figure 2.1: Cascaded classifier in the form proposed by Viola and Jones [173].

is deemed interesting, a sample is forwarded to increasingly more complex and computationally intensive features for reconsideration. Not only is this characteristic of cascades valuable in real-time detection environments, but it has also been shown to be immensely helpful in medical applications where a number of medical tests have been reduced for patients [129].

However, cascades come with two competing objectives; namely classification efficiency and accuracy [129, 180]. Since many samples are evaluated by the initial layers, while only a few samples are evaluated by the last layers, any increase in the accuracy of the coarse layers, results in detection-runtime degradation. The advantage of cascades is that they provide a mechanism for trading off the accuracy and the efficiency of layers. For instance, this can be achieved through the parameterizing of the ensemble layer size and feature type complexity. In addition, Viola-Jones demonstrate how a threshold value for each layer can be tuned which determines the operating point of the classifier at every layer.

Bourdev and Brandt [10] attempt to get around the trade-off between accuracy and efficiency by maximizing both through their proposed *soft cascades*. The idea is to first train a single layer classifier and subsequently, augment the classifier with rejection thresholds. The rejection thresholds assume a behaviour of a cascade by enabling a rapid rejection of negative samples without evaluating the entire classifier. Thus, not only can high accuracy be attained; but, also real-time detection is realized. However, Zhang and Viola [195] discuss the difficulty of setting rejection thresholds for a single layered classifier as part of post-processing. Finding the optimal trade-off between positive detections and the false positives is not trivial.

Cascaded approaches have demonstrated their ability to significantly reduce large-scale learning to more computationally tractable tasks [170]. This has been shown to be particularly effective in rare-event domains [175], but also more widely applicable to datasets with balanced class distributions [93].

The mechanism by which large scale learning is made tractable under cascaded frame-

works is through the *informed sampling* of $N_i$ at the begining of each layer $i$. The majority class $N$ is reduced in size after each layer $H_i$ is trained, whereby all samples $H_i(x_i) = 0 \wedge y_i = -1$ are removed from $N$. Thereby, the redundant information from the majority class $N$ is not used and as a consequence, *explicit* learning only takes place on a subset of samples from $N$.

Liu et al. [93] explore the bootstrapping mechanism of cascade training in order to apply it to the problem of imbalanced learning. They show how the bootstrapping procedure can be used as an *informed undersampling* strategy for providing balanced positive and negative datasets. Their results indicate that the bootstrapping procedure preserves relevant information and is particularly effective at addressing the deteriorated accuracy of minority classes on datasets with skewed class distributions.

### Parallel Cascades

Periodically, a single cascaded classifier can become over strained in its attempt to accommodate all the within-class variability of a target object [90]. The consequence is that the generalizability of the cascaded classifier in highly complex datasets becomes compromised. To counter this, the intuitive approach is to transform the problem into a series of multiclass learning tasks.

As discussed in 2.1.2, the target class training samples can be divided into subsets with each one representing a sub-pattern class [197] that is akin to clusters of ensembles. A cascade classifier is then usually trained for each sub-pattern. Once all the individual subclass-pattern classifiers are trained, they are deployed at runtime by executing them in parallel as in Figure 2.2 with their results being merged. The merged result indicates which sub-pattern class has been detected with the highest probability.

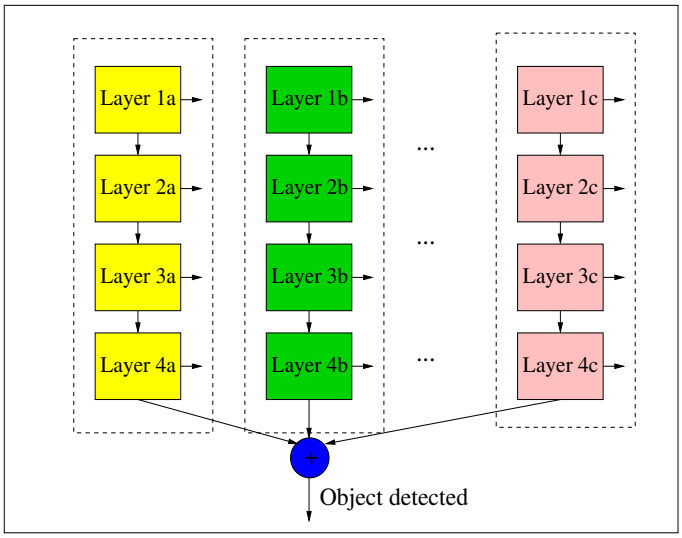



Figure 2.2: The structure for parallel cascaded classifiers [90].

This method was first proposed by Schneiderman and Kanade [146] who created an array of parallel detectors for multi-view faces and vehicles. The benefits of this approach enabled the successful detection of faces with an out-of-plane rotation for the first time.

Through the process of clustering the dataset, training efficiency was also realized. The drawback was that the detection runtime was in the order of minutes per frame and thus rendered it of little value to real-time systems.

Zhang et al. [197] employed the same training methodology to create a multi-view face detector, though they achieved real-time detection speeds through a novel arrangement of multiple detectors into an efficient pyramid scheme. Their system still underwent parallel executions of detectors whose results had to be merged in order to discern a classification.

The idea of generating clusters of ensembles, where each partition possesses a dedicated classification specialization, has also been explored in various forms in conjunction with the problem of non-stationary environments. Elwell and Polikar [36], Muhlbaier and Polikar [107] train an additional ensemble for each new snapshot of incoming data that is incorporated into the existing ensemble. Wang et al. [177] discuss the shortcomings of conventional ensemble systems as having a lack of *semantic awareness*, which refers to each individual classifier of an ensemble not having a corresponding hidden concept it can be associated with. The outcome being, that there are higher update costs since a larger proportion of the ensemble require updates as concepts change, together with higher computational costs of identifying the relevant classifiers which must undergo modification.

### Cascaded Trees

Alternatively, the problem of within-class variability for target objects as well as the shortcomings of building a parallel cascaded classifier is addressed by Lienhart et al. [90], who proposed a tree classifier training structure shown in Figure 2.3. The tree training structure produces a single classifier. At each node within the tree structure a $k$-nearest clustering algorithm is employed to construct branches of the classifier. The branching has the effect of dividing up the patterns of sub-classes of the target object into its constituent categories and thus enabling the learning algorithm to produce more discriminant classifiers. The branching itself is efficient since it is followed through only if the resulting classifier's accuracy is improved and the detection runtime complexity does not deteriorate.

This methodology has been shown to be twice as fast at detection runtime as two parallel classifiers and even faster than a single classifier with accuracies either preserved or improved upon [90]. The complexity involved with individually training multiple classifiers is removed as well as the empirical methodology of selecting sub-pattern classes during training. However, there are still disadvantages with this structure. The threshold adjustments associated with the cascaded structure remain. The root node still has to undergo training with the entire positive training sample set. This may be difficult for complex datasets with very large in-class variability [5]. In such a scenario, the number of weak classifiers may increase significantly (adversely affecting both the training and detection runtimes), or the threshold adjustments may result in a substantially large false detection rate. Lastly, the $k$-nearest clustering algorithm is also likely to increase the training runtime of the detector.

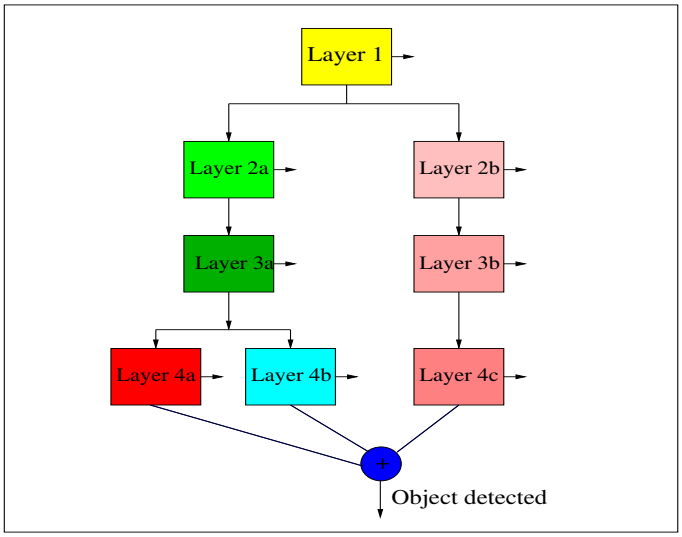



Figure 2.3: A detector tree of boosted classifiers proposed by Lienhart et al. [90].

### 2.3.2 Ensembles of Nested Dichotomies

Research motivated by the quest to simplify the decision boundaries of multiclass learning, has produced methods which decompose the multiclass training task into binary trees of smaller subproblems. A tree recursively splits sets of class labels at each level, until each leaf node learns to differentiate a binary class set. This principle approaches a complex learning task by first performing general discrimination which is then refined at each step until the final prediction can be formulated. Using this approach generates at most $k-1$ classifiers [94]. This hierarchical decomposition of a multiclass problem is referred to as nested dichotomies in statistics, and is a standard technique for tackling multiclass learning with the use of logistic regression [40].

Dong et al. [32], Frank and Kramer [42] demonstrated how to construct and effectively combine an ensemble of nested dichotomies. By using C4.5 and logistic regression as base learners, they show that this approach produces classifiers that are mora accurate than the standard approach of applying these learners directly to a multiclass problem. In addition, their results compare favorably with the accuracies of OAO, OAA and ECOC methods. However, the structure of a tree and the manner in which class labels are grouped influences results [32, 94]. Determining the tree configurations that yield the optimal accuracies is not a trivial exercise. This problem is magnified particularly with the absence of *a priori* domain knowledge regarding the datasets and appropriate class label groupings. The number of possible tree combinations grows rapidly with the number of class labels. For a $k$ class dataset, the total possible number can be calculated using a double factorial $(2k-3)!!$ This makes an exhaustive brute force search prohibitive over the entire possible range of configurations. Frank and Kramer [42] randomly sample from the space of all possible trees in order to find the best configurations. Dong et al. [32] shrink the size of the search space by limiting it to trees that are balanced in respect to class labels and the size of the datasets. Their approach addresses the shortcomings of protracted training runtimes for nested dichotomies when the tree structures become

extremely unbalanced. They achieve this while maintaining high accuracy rates.

Recently Rodríguez et al. [131], proposed a method for constructing forests of nested dichotomies. By combining the forests of nested dichotomies with a variant of AdaBoost called MultiBoost [179], they produced training framework, which yields an increase in the diversity of the base classifiers and subsequently resulted in improved accuracies over ensembles of nested dichotomies.

## 2.4 Summary

The two fundamental principles in ensemble systems of diversity and vote-combination provide ensemble-based approaches with considerable flexibility to tackle the most challenging of machine learning problems. Given that ensemble-based systems intersect with such a wide spectrum of disciplines from computer science to statistics and data mining, the benefits are that many varieties of strategies have become available to innovate with.

Ensemble solutions based on CTF strategies continue to provide effective mechanisms for trading-off the computational loads of inducing decision boundaries on massive and complex data streams with those of accuracy. Informed bootstrapping methods used on cascades, enable large reductions of data that an inductive algorithm explicitly needs to learn on, while generating real-time capable classifiers. Meanwhile, learning from imbalanced datasets has remained an ongoing issue for accurate classification which can be mitigated to a degree by cascaded classifiers.

Arguably, one of the most relevant and interesting challenges of late has been the need for classifier adaptability to changing environments. Much recent research has involved ensemble-based learning and its natural applicability to this as yet unresolved problem. The capacity of ensemble systems to be partitioned into clusters, whereby, each cluster becomes responsible for encoding specific concept drifts holds promise.

The succeeding chapters explore the various strategies developed in the course of this research that leverage coarse-to-fine learning principles in order to address some of the open questions and challenges outlined.

# Chapter 3

# Methodology

This chapter describes the general methods that were common to all experiments in this research. It covers the overview of the major datasets, feature types and their extraction methods as well as the types of weak learners employed. Training procedures and classifier evaluation methods are also discussed together with the description of the environment and the hardware used. Since the methodology here is not exhaustive, the additional specifics and details of each experiment are expounded on in the relevant chapters.

## 3.1 Face Detection Datasets and Feature Types

A total of 15000 facial images were collected for training face detection classifiers. The images were gathered from various publicly available datasets. The sources were: FERET[1], Yale *Face Database B* [53] and the face database from the Vision Group of Essex University. Where it was required, the images were cropped in order to isolate only the facial features to the exclusion of other areas of the head as much as possible. All facial images were extracted and vectorized to a reduced size of $24 \times 24$ pixels. The $24 \times 24$ pixel size has been a standard dimension[2] for training facial images.

The images were predominantly frontal without rotation and articulation; however, particularly on the Yale dataset, images with some articulation, rotation and strong illumination variations that resulted in feature occlusion were incorporated into the corpus. In addition, the images from the Essex University Vision Group contained facial gestures. The accuracy of a classifier is limited by the quality of the training sets used and it was understood from the outset that due to the large in-class variation, the resulting classifiers would not compare to the state-of-the-art face detectors; however, this was not the goal. The incorporation of such challenging images was seen as an interesting test scenario for the abilities of the algorithms to handle outliers. The images themselves were trained without any pre-processing steps to counter the effects of the differing lighting conditions, nor was any additional synthetic data generated for fine tuning classifiers for the test dataset.

[1] URL "http://www.frvt.org/FERET/default.htm"

[2] This was the dimension used by Viola and Jones [173] and claimed to be the standard by Huang et al. [66]. $24 \times 24$ is also the default setting for the OpenCV implementation for face detection training. However, some notable face detectors have used different kernel sizes: Rowley [135] $20 \times 20$, Poggio and Sung [119] $19 \times 19$, Garcia and Delakis [52] $36 \times 36$.

The negative dataset consisted of a total of 2500 images. The dataset was a mixture of images downloaded from the internet, consisting of a variety of patterns, as well as a combination of high resolution photo images with different landscapes and backgrounds. Using the sliding window approach at different positions and scales on the original images, hundreds of millions of negative samples were generated from these 2500 images. The bootstrapping approach of Sung and Poggio [155] and discussed in Section 2.3.1 was used at each layer, whereby all correctly classified negative samples were removed from the current negative training set. They were then replaced by new negative samples which the current classifier classified as false positive. Viola and Jones [173] replace negatives with new subwindows from an image whose positions and scales are randomly selected. The experiments here did not randomly select parts of an image, but instead sequentially scanned the image using a subwindow. Each subwindow representing a negative sample scanned a different negative image for samples. This approach carried lower computational overheads and was also easier to repeat in order to validate the training code.

The test set for non-adaptive classifiers was the standard benchmark CMU MIT dataset [136]. This dataset consists of 130 gray-scale images which contains 507 faces. Using the raster scanning method discussed below, 72,654,174 negative samples can be generated from all the images. This dataset was specifically created for detecting frontal views of human faces and is also made up of low resolution and low quality images. Due to the presence of some cartoon-like faces of different quality, some publications have only used subsets of this dataset for testing. This research has considered the entire set.

This research considered adaptive face detection classifiers in changing environments which required an additional domain-specific dataset. Due to a lack of publicly available datasets for this problem, a custom dataset was created. The specifics of it are presented in Chapter 6.

### Haar-like Features

In order to combine pattern recognition with face detection, a decision had to be made as to what feature types to use for extracting information from the images. Papageorgiou et al. [113] introduced Haar-like features to machine learning by modifying the original Haar wavelet decomposition to allow them to perform object recognition at a finer resolution. Since the successful application of Haar-like features to face detection by Viola and Jones [173], they have become common to vision detection systems where the target objects display consistent geometrical properties. For this reason, Haar features were applied in this research. The Haar features used in this research are shown in Figure 3.1.

Haar-like features are a mixture of square and rectangle filters, each divided into two main regions. The filter is applied to an image at different scales and the value for each filter is the difference of the two regions; where each region is the sum of all its pixel intensity values. The complete Haar-like feature set is applied to an entire image frame producing an over-complete feature space. The amount of information that accompanies this type of a feature set is much greater than the amount of actual raw image data.

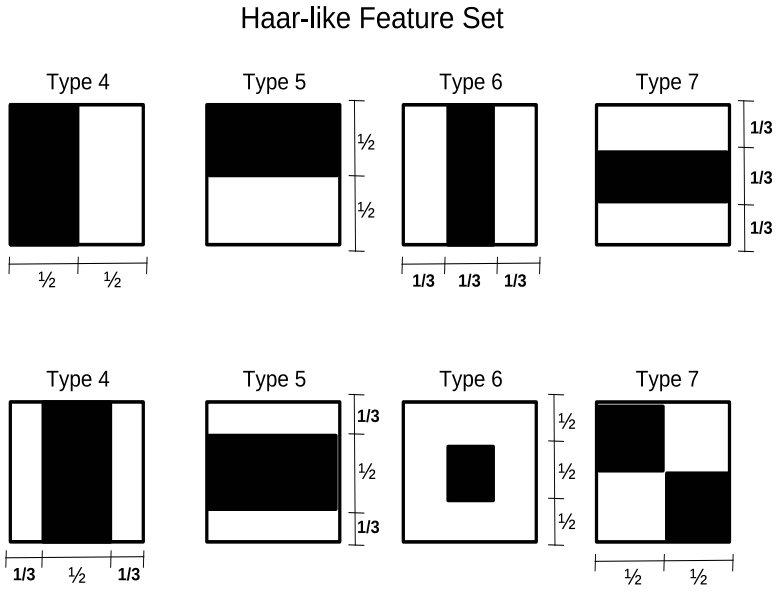



Figure 3.1: Haar-like feature set.

### Object Detection

When using feature-based approaches that are combined with Haar wavelets, it is necessary to perform an exhaustive search over the entire image. The object detection phase of the face detectors in this research consisted of several steps. The process involved is as follows:

1. the entire image is converted to gray-scale
2. an intermediate *integral image* is computed based on pixel intensities
3. a sliding window exhaustively scans the entire integral image at differing positions and scales
4. for each weak classifier, a single Haar feature is extracted from the integral image
5. a classification is made to accept the window as the target object or reject

Many state-of-the-art detectors perform pre-processing of an image before a classifier is applied. This is usually in the form of normalization in order to minimize the effects of large illumination variances. The experiments here however omitted this step and performed detection on the raw data.

During the testing of the classifiers on the CMU MIT dataset, each positive detection was compared to the ground truth file. The ground truth file provided four pixel positions that define a rectangle around a positive object. However, there is a large degree of subjectivity involved in determining how large or small the defined rectangle needs to

become before a subwindow that encloses a target object is no longer a face. As a result, a rule was formulated which confirmed a positive detection only if all four corner pixel positions were displaced from the ground truth positions up to the maximum of 25% of the width in the x-axis, as well as the height in the y-axis.

Since the training images possess a degree of translational variability, the face detector classifier is able to handle small shifts in an image. For this reason, the sliding window did not need to scan every pixel, but could be shifted a predefined $\Delta$ pixels across and down the two axes following each classification. The choice of $\Delta$ has a significant effect on the speed of the detector but can also reduce accuracy. In all the experiments $\Delta$ was set to 2, while the scaling of the sliding window was set to 1.2[3]. The details of the Haar features and their extraction are summarized in Table 3.1.

Fast feature calculation was performed through the intermediate representation of an image termed the integral image. Any point $(x, y)$ on an integral image defined as

$$ii(x, y) = \sum_{x' \leq x, y' \leq y} i(x', y'), \tag{3.1}$$

over the original image $i(x, y)$ contains the sum of all pixel intensities to the left, and above of $(x, y)$, as well as inclusive. The integral image can be computed rapidly with one pass. Since Haar wavelets consist of rectangles, all that is needed in order to calculate the sum of intensities within it is the four corner pixels. Beginning with the top left corner pixel, let the four points on an integral image be defined as:

$$\begin{aligned}
\psi_1 &= ii(x, y) \\
\psi_2 &= ii(x + w, y) \\
\psi_3 &= ii(x, y + l) \\
\psi_4 &= ii(x + w, y + l)
\end{aligned}$$

where $w$ is the width and $l$ is the length, then the sum of the intensities bounded by the four points can be calculated as $\psi_4 + \psi_1 - (\psi_2 + \psi_3)$. This calculation requires only four array accesses which enables extremely fast calculation of the Haar feature sets.

Table 3.1: Training dataset details together with the properties for constructing the *static* classifier.

| **Property** | **Attribute** |
|---|---|
| Number of Haar-like feature types | 8 |
| Maximum Haar-like features per image sub-window | 200,000 |
| Minimum pixel area size per Haar-like feature | 16 |
| Sliding window scale factor | 1.2 |
| Sliding window shift step $\Delta$ | 2 |
| Initial sliding window size | $24 \times 24$ |

[3]These settings closely resemble the chosen configuration by Viola and Jones [173].

## 3.2 Multiclass Classification Datasets

The multiclass datasets consisted of 18 benchmark problem sets from the University of California, Irvine (UCI)[4] machine learning repository [41]. They included datasets with even and skewed class distributions, as well as datasets with considerable variations in the numbers of class labels, sample sizes and attributes. Unlike the datasets used for face detection, the multiclass datasets required no separate feature extraction components. The datasets comprised of instances, represented by feature vectors that were extracted using various methods which enabled the emphasis to be solely invested into the learning algorithms themselves. The details of the datasets are shown in Table 3.2.

Some datasets were furnished with separate training and test datasets and when these were available, they were adhered to. For datasets without an explicit separation for training and testing, 10-fold cross validation, discussed below, was employed.

Table 3.2: Properties of the multiclass datasets.

| Dataset | Training Set | Test Set | Classes | Features | Class Distribution |
|---|---|---|---|---|---|
| letter | 16000 | 4000 | 26 | 16 | balanced |
| pendigit | 7494 | 3498 | 10 | 16 | balanced |
| optdigit | 3823 | 1797 | 10 | 64 | balanced |
| satimage | 4435 | 1762 | 6 | 36 | skewed |
| vowel | 528 | 462 | 11 | 10 | balanced |
| segmentation | 2310 | 10-fold-cv | 7 | 19 | balanced |
| vehicle | 846 | 10-fold-cv | 4 | 18 | balanced |
| glass | 214 | 10-fold-cv | 7 | 9 | skewed |
| iris | 150 | 10-fold-cv | 3 | 4 | balanced |
| factors | 2000 | 10-fold-cv | 10 | 216 | balanced |
| fourier | 2000 | 10-fold-cv | 10 | 76 | balanced |
| waveform | 5000 | 10-fold-cv | 3 | 40 | balanced |
| waveform noise | 5000 | 10-fold-cv | 3 | 40 | balanced |
| morphological | 2000 | 10-fold-cv | 10 | 6 | balanced |
| shuttle | 43500 | 14500 | 7 | 9 | skewed |
| robot navigation | 5456 | 10-fold-cv | 4 | 4 | skewed |
| yeast | 1484 | 10-fold-cv | 10 | 8 | skewed |
| page blocks | 5473 | 10-fold-cv | 5 | 10 | skewed |

## 3.3 Weak Learner

The decision stump is the primary weak learner used in this research. Decision stumps are degenerate trees consisting of only two leaves. They are the simplest type of weak learners in that their requirement is often reduced to producing error rates that are only just better than random guessing. The obvious advantage of using such simple inducers is their low runtime overheads. Decision stumps are commonly combined with boosting. Boosting has been shown to perform better when employed with weaker inducers like decision stumps, while empirical evidence has shown that it has the tendency to overfit if used with more

[4]URL "http://archive.ics.uci.edu/ml/"

complex learners [141]. The disadvantage is that for datasets with complex and multiple class boundaries, the discriminative strength of decision stumps may not be enough to achieve an adequate class separation.

Using an efficient technique described by Viola and Jones [173], the optimal split or threshold for the decision stump, together with the direction, can be calculated rapidly with one pass over a given feature vector $\vec{f}$, where each value $f_i$ corresponds to a sample pair of $(x_i, y_i)$. The feature vector $\vec{f}$ is first sorted. As the list is traversed from the first element to the last, only four sums are evaluated and maintained: total sum of the positive sample weights $tp$, total sum of the negative sample weights $tn$, the sum of all positive sample weights below the current sample $sp$ and the sum of all negative sample weights below the current sample $sn$. Given these sums, the minimum training error of either side of the threshold can then be calculated as

$$\epsilon_i = min(sp + (tn - sn), sn + (tp - sp)) \tag{3.2}$$

The lowest training error $\epsilon_i$, together with the threshold value $i$ and the direction, is kept track of across the traversal of the sorted list and becomes the optimal threshold split. A slightly more complex domain partitioning learner that calculates multiple thresholds for multiclass problems is presented as part of multiclass learning in Chapter 7.

## 3.4 Training Procedures

Due to the repeatability and the deterministic nature of the algorithms used for training the face detectors, the results of the classifiers were gathered from only one round of their training. There were instances where some algorithms utilized random selections of samples at specific stages of learning. Those algorithms were verified for repeatability by being run several times confirming their stability and low variance. Running all face training algorithms numerous times and providing averages of the results was prohibitive in this training domain. This was due to the fact that the training runtimes stretched out over weeks and the computing resources described below were inadequate for this.

The sensitivity of the measurements for the training runtimes was calculated by running a fraction of face classifier training 10 times. The execution runtimes of the face detectors was calculated as an average over 10 runs.

Due to the non-deterministic nature of many of the algorithms used for multiclass training and due to the considerably smaller computational overheads, all experiments were run multiple times. On datasets which provided both training and test sets, the experiments were run 10 times.

In the absence of explicit test sets, 10-fold cross validation was used to mitigate bias that might be associated with any particular set of samples [184]. The process of 10-fold cross validation randomly divides a dataset into 10 splits or folds. Training is then performed on nine of the ten folds, while the testing is carried out on the remaining fold. In this research the training was performed 10 times on each fold, while all algorithms trained on the same folds. The error estimates from all runs were then averaged to produce an overall error estimate alongside which the standard error was reported.

### Implementation and Hardware

All experiments were implemented in C. The OpenCV library was used in order to facilitate image processing for face detector training. The bulk of the classifier training was performed on the machines at the Computer Vision Laboratory at Massey University. Initially these comprised of 16 machines with Intel 1.93GHz CPUs and 2GB RAM running Linux Centos OS. These machines were used for the initial phase of this research which covers the experiments in Chapter 4. These machines were eventually replaced with 16 new Intel 4 Core 2.63GHz PCs with 4GB RAM, running on Centos OS. These machines provided the processing power for the majority of the remaining experiments. The experiments on adaptive learning were performed on a personal computer whose specifications are an Intel 4 Core 2.83GHz CPU with 8GB RAM and running Linux Kubuntu. Adjustments were made for runtime results of any other experiment that was also run on this machine to allow for the slightly different clock speeds of the machines in the Computer Vision Laboratory.

## 3.5 Classifier Evaluation Methods

A variety of evaluation measures were used to assess the algorithms and the classifiers in this research. While the methods for evaluating the generalization of binary and multiclass classifiers necessarily differed, common to all algorithms and classifiers was the analysis of:

1. the training runtimes
2. the training convergence in terms of total error achieved or the rate at which the error decreased with increasing ensemble complexity
3. the size or the complexity of the generated ensemble classifiers
4. the detection runtimes of the resulting classifiers
5. the relative trade-offs between the total generalization error and the training runtimes

Additional criteria considered important by Rokach [134] were also used here when discussing the merits of an algorithm that involved such aspects as their ability to scale to larger training and detection tasks. In addition, the robustness of algorithms was also considered in respect to their ability to handle noisy data. Stability of an algorithm describes its ability to generate repeatable results and is an important attribute. This property was considered as well as the usability of the learning algorithms, which is represented by the provision of comprehensible controlling parameters that can be easily tuned. The flexibility of a learning method is desirable, whereby the designed algorithm is inducer and vote-combination method independent. The updatability of the resulting classifiers from a given algorithm was also taken into consideration.

### Face Detection Accuracy Analysis

For face detection the receiver operating curves (ROC) were used as the primary assessment criteria for the generalization of classifiers. The graphs were generated with the true positive rate (TPR) and false positive rates (FPR) derived from the sums denting the detections of true positives (TP), true negatives (TN), false positives (FP) and false negatives (FN) as

$$TPR = \frac{TP}{TP + FN} \tag{3.3}$$

$$FPR = \frac{FP}{TN + FP} \tag{3.4}$$

In recent years there has been an increase in the use of ROC graphs amongst the machine learning community. This can be attributed to the realization that simple classification accuracy is an insufficient metric for measuring the overall generalization ability of a classifier [39]. The ROC curves are a powerful tool for visualizing the generalization ability of a classifier. Particularly in this research, ROC graphs are especially useful since they are applicable in domains with skewed class distributions and unequal classification error costs [38]. A ROC curve has the ability to depict the relative trade-offs between the benefits, represented by the TPR, and the costs, being the FPR, across a whole range of classifiers' predictive ability.

### Multiclass Accuracy Analysis

The evaluation of the accuracy of multiclass algorithms required broader and more sophisticated methods, due to the larger number of datasets and algorithms being compared, as well as the characteristics of the datasets, which included balanced and skewed class-distributions.

In the multiclass setting, the conventional practice has been to present only the accuracy as the primary measure of generalization [29, 153], defined as

$$Accuracy = Recall = \left( \frac{\sum_{i=1}^{k} TP_i}{\sum_{i=1}^{k} (TP_i + FN_i)} \right) \tag{3.5}$$

where $i$ denotes a class label and $k$ denotes the number of classes in a dataset. This is sometimes expressed as the total error rate as $1 - Accuracy$. However, presenting only the accuracy has been shown to be inadequate and often misleading on class-imbalanced datasets [38, 62, 124]. The reason for this is that the minority classes can have a negligible impact on overall accuracy compared to the majority classes.

A more meaningful measure of accuracy for biased class-distributions is to calculate the geometric mean (g-mean) instead. Sun et al. [152] demonstrated how the g-mean of recall values of each class can be applied to the multiclass scenario by being calculated as:

$$G\text{–}mean = \left( \prod_{i=1}^{k} Recall_i \right)^{\frac{1}{k}} \tag{3.6}$$

which yields a single value that presents a balanced performance of a classifier across all

classes. For a more comprehensive analysis of a classifier at an individual class level, the F-measure is employed. The F-measure uses the precision of each class, defined as

$$Precision_i = \left( \frac{TP_i}{TP_i + FP_i} \right) \tag{3.7}$$

which takes into account the false positives, and combines it with the recall of each class to yield

$$F_i\text{–}measure = \left( \frac{2 \times Recall_i \times Precision_i}{Recall_i + Precision_i} \right). \tag{3.8}$$

The F-measure generates values from 0 to 1 under which high values are generated if both the precision and recall values are high.

Accordingly, in this research, the total error rates were used primarily on balanced datasets, though they are presented here alongside the g-mean values. On unbalanced datasets, the g-mean values were of primary reference, while the F-measure were resorted to for in-depth analysis of selected datasets. For completeness, the total error rates are also listed for unbalanced datasets as well.

Average ranks were used to summarize classifier accuracies as demonstrated by Demšar [29]. The non-parametric Friedman [47, 48] and Nemenyi [108] statistical tests were used to determine significant differences between the mean ranks amongst all classifiers. The Wilcoxon signed-ranks test [182] was used for pairwise comparisons of classifiers.

The use of Cohens Kappa [24] statistic as the measure of the degree of agreement or disagreement between the weak classifiers was considered. However, the potential usefulness of this test and the reliability of its results in the context of cascaded classifiers was doubtful due to the fact that for any given sample only a subset of weak classifiers is evaluated at a time.

# Chapter 4

# The PSL Algorithm and Applications to Face Detection

Considerable advances have been achieved in the area of real-time detection as well as overall improved classifier accuracies through the strategic usage of cascades in combination with boosted ensembles. However, according to Brubaker et al. [15], the greatest obstacle to a wider proliferation of cascaded strategies as solutions on computationally intensive domains with large feature spaces, lies in their protracted training runtimes. This limitation is not only seen as an impediment to producing effective object detection applications, but also makes the testing of new theories and algorithms, as well as the verification of others research a significant challenge [101]. The ground breaking face detector from Viola-Jones required weeks of computing to produce a cascaded classifier using multiple machines. Notwithstanding the continued advances in hardware since then which has enabled faster processing speeds, as well as the advent of ubiquitous availability of parallel processing through Graphics Processing Units (GPUs), the training runtime overheads still remain a challenge.

Previous experiments [5, 156] with the PSL have only considered relatively small datasets from the UCI repository with pre-processed feature values and limited total feature spaces. The PSL produced comparable results to those of Viola-Jones and single-layer ensemble classifiers. However, the findings there could not be extrapolated to real-world problems. The general aim in this chapter was to assess the viability of the PSL method to address the problems of training runtimes and efficiency in more challenging settings.

## 4.1 Motivation

The specific aims of this chapter can be summarized as:

1. Explore the applicability of the PSL algorithm on a real-world problem with challenging datasets containing imbalanced class-distributions and high training and detection runtime overheads.

2. Implement the conventional cascade algorithm by Viola-Jones and use as a benchmark for comparisons.

3. Study the degree to which training runtime overheads can be reduced.

4. Observe the trade-offs that might occur with respect to accuracy.

5. Ascertain the effects that different tunable parameters have on training, detection runtime and accuracy of the classifiers.

## 4.2 Related Work

The reasons behind protracted training runtimes of cascaded classifiers on difficult problem domains are attributed to several factors. Pham and Cham [116] identify the repeated extraction of large feature spaces as a primary contributor to slow training runtimes as demonstrated in face detection using Haar-like feature types. In such domains, the total size of the feature pool can easily comprise a search space of several billion features[1] even on conservatively sized datasets that utilize a modest number of Haar-like feature *types*. For each boosting round, this feature pool has to be both extracted from an image and searched through in order to find the feature with the lowest error rate. This process may require thousands of iterations in order to produce enough weak classifiers to comprise the final detector.

Various strategies for addressing the computational demands of a large feature pool have been proposed. The most intuitive has been that of Baluja et al. [4], Wu et al. [187] which employs feature filtering in order to limit its size. These strategies have produced very notable speed-ups. Criticism has been labeled against feature filtering approaches [116] on the grounds that the reductions in runtimes have been realized in part due to their ignoring the weight distribution of the boosting process. Because to this, the features selected by filtering methods have been described as being incompatible with the boosting approach since the theoretical guarantees for it no longer hold.

Wu et al. [188] use feature filtering as well as a caching strategy for extracted feature values in order to limit the computational cost involved in re-calculating them for each round of boosting. Consequently, only the feature weights have to be adjusted. This results in substantial reductions in runtimes; however, this approach imposes significant memory requirements when large training sets are used even with reduced feature pools.

Pham and Cham [116] achieve the greatest reductions in the amount of time required to train each weak classifier by applying statistical methods and assumptions regarding the distribution of the feature space. They avoid the re-calculation of Haar feature values at each boosting step by treating them as high-dimensional random vectors whose values are modelled as Gaussian distributions. Due to this, they have found it possible to use even larger feature sets that generate up to 300,000 features per image window. According to the authors, the expanded feature sets have been a contributing factor to the training of very accurate detectors. Though their empirical work has produced effective face detectors, theoretically it has not been established that Haar-like feature values follow a Gaussian distribution.

Other research efforts have branched down the path of increasing the discriminatory ability of the feature types with the end goal of not only improving accuracy but of also

[1] On each image, a conservative Haar-like feature set generates in the vicinity of 200,000 features per sub-window. This is multiplied by the number of training samples in the dataset that may range from 5,000 to 20,000. Finally, this process is repeated as in the case of Viola-Jones in excess of 3,000 iterations for each weak classifier. The final product therefore results in a feature pool numbering in billions.

decreasing the training runtimes [89, 104, 183, 189]. However, although most of these approaches yielded fewer weak classifiers, the additional computational costs associated with more powerful features did not succeed in shortening the actual training runtimes. Alternatively, a plethora of variations to the AdaBoost learning algorithm have been proposed [46, 88, 175], but no one variant has contributed to significant training runtime reductions.

The final contributing factor in protracted training runtimes involves cascade optimization. The optimal configuration of adjustable parameters that define individual layer detection and false acceptance rates are not known *a priori*. As a result, a strategy of *ad hoc* parameter formulation is normally adopted whereby multiple classifiers with different parameter settings are trained. Thereafter, the best candidate is chosen from amongst them. This trial and error process not only fails to formulate an optimal solution, but means that the overall training runtime involving numerous repetitions can be protracted to unreasonable durations. The optimization process is further hampered by the requirement of many classifiers to execute in real-time environments. This draws in another adjustable parameter that defines the number of weak classifiers per layer thus further complicating the training process.

Luo [95] proposes a system which is capable of optimizing a cascaded classifier once it has been trained in an arbitrary manner. It adjusts layer thresholds as part of post-processing and produces notable accuracy and performance results. This goes some way to simplifying the training procedures by placing the emphasis on only having to formulate a parameter combination which meets the desired training error. However, the research does not address how layer targets should be set for the initial training phase. These values greatly influence the bootstrapping of the training data that in turn affects the final accuracy. So in effect, this approach can only be seen as a strategy that determines an optimal operating setting for a classifier that is intrinsically sub-optimal and whose performance can only be marginally improved.

Brubaker et al. [15, 16] propose a more robust solution to the problem of cascade design by devising an optimization function. In their research, the optimization function adaptively chooses the target rates for the individual layers in a cascade based on the user requirement for the final cascade detection rates as well as the false positive rates. These targets are determined with the help of an independent evaluation dataset against which each layer's performance is tested. The end result was a face detector which performed comparably with the best current detectors. However, it is difficult to know how much of it can be attributed to the cascade optimization component since their training framework was modified significantly in other key components as well. Nonetheless, a viable approach to removing the empirical cascade parameter setting has been proposed.

Similarly, Sun et al. [151] developed a framework for analyzing the impact of learning of a single layer in a cascade on the entire cascaded classifier through a novel cost function which they claim yields an optimal learning goal for each cascade layer. Although, in their system it is not clear how to set the parameters to optimize for execution runtime for a desired detection rate. Chen and Yuille [23] on the other hand formulate an optimization criterion with a greedy algorithm for designing time-efficient cascades. Their focus is primarily on creating a cascaded classifier that produces an optimal speed performance given a desired overall accuracy which achieves speed-ups by trading off false acceptance

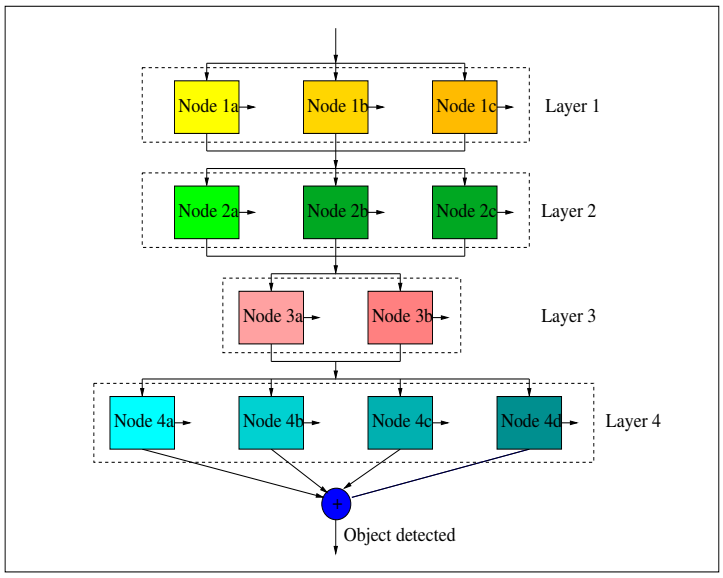



Figure 4.1: Structure illustrating parallel strong classifiers within the same layer (PSL).

rates against each layer's execution runtimes. An altogether different approach to finding a solution to cascade optimization is to remove the layers entirely during training and use "soft cascades" instead. The "soft cascades" achieve an ease of training by making the cascade training parameters as well as layer threshold adjustments redundant. The training involves creating a single monolithic ensemble classifier to which an algorithm is applied which builds into it a quasi cascade-like behaviour. Bourdev and Brandt [10] successfully augmented such a monolithic ensemble classifier with rejection thresholds that could be calibrated for a specific detection rate and execution runtime. Their architecture effectively decoupled the speed/accuracy trade-off associated with cascade training. Once the classifier was trained it was able to be rapidly adjusted for any point on the ROC surface without iterative *ad hoc* re-training.

Of the more recent and notable contributions, Raykar et al. [129] devised a strategy for optimizing cascade training. Their approach relaxes the hard cascaded structure that involves sequential training of each layer and proposed a method for jointly training all cascade layers. Once all the cascade layers are trained, their method is able to adjust the trade-off between the accuracy and the runtime cost by choosing the layer thresholds as a post-processing step. Along similar lines, Saberian and Vasconcelos [137] also adopted the approach of simultaneously training all layers of a cascaded in pursuit of overall cascade optimality. Their contribution was an introduction of a mathematical model that accounts for both the classification accuracy and cascade complexity. A new boosting algorithm FCBoost was proposed by them for cascade training by solving a Lagrangian optimization problem, where classification risk is minimized under complexity constraints.

## 4.3 The PSL Algorithm

The PSL was originally designed to address the difficulty involved with meeting 100% hit rates for the positive samples for each cascade layer. This is directly related to the design of AdaBoost which minimizes the total classification error rather than the false positive or false negative rates [175]. Therefore, under the conventional cascaded approach, satisfying the layer targets becomes more difficult with each layer as the separability between the classes decreases. This contributes to longer training runtimes and more complex ensembles. The hit rate requirements can be relaxed for succeeding layers, however, this inevitably results in weak classifier generalizability.

The PSL structure, shown in Figure 4.1, formulates a solution to these issues while also achieving a significant reduction in the training and execution runtimes. The method accomplishes this by transforming the internals of cascade-layer training to the same degree that Viola-Jones cascades transformed the monolithic ensemble structure. The PSL framework introduces horizontal cascading within each layer, which complements the vertical cascading that is existent in the Viola-Jones method. While Viola-Jones executes independent rounds of AdaBoost for each layer, the PSL structure provides a methodology for executing multiple, independent AdaBoost rounds within distinct *nodes*, situated in the same layer.

The workings of the Viola-Jones cascading framework was described in Section 2.3.1. It was shown there how Viola-Jones cascading operates at each layer of a cascaded classifier by executing continuous boosting rounds until the layer false alarm and hit rates have been met, before a new layer is commenced. The PSL framework decomposes the layer targets into smaller and more achievable intermediate steps. It does this by specifying a *criterion condition* for each AdaBoost node that places a limit on the number of boosting iterations each node can perform. Once the criterion condition is satisfied, the boosting process halts and a new AdaBoost node is started with the weight distributions for all samples reset.

The training of a PSL classifier and its strategy for handling datasets can be stated more formally as follows. Given a training dataset $D$, consisting of instances $(x_i, y_i)$ where $Y \in \{-1, 1\}$ is the domain for a binary class problem, $P$ is a set of all positive samples and $N$ denotes the set of all negatives, the PSL constructs a cascaded classifier $H$, that is comprised of $H_{i,\ldots,L}$ layers. Each layer $H_i$ consists of $n$ number of nodes that comprise of weak classifiers $h_{i,j,k}$, where $i$ signifies the layer of the weak classifiers, $j$ denotes the $j^{th}$ node it belongs to and $k$ represents its position in the node. Initial nodes for each layer are trained on the entire dataset $P$; however, subsequent nodes alter its composition. Therefore, let $P_{i,j}$ denote the positives used to train a given node $j$ within a certain layer $i$. Additionally let $N_i$ denote the negative dataset used on layer $i$ where $N_i \subset N$.

The training of $H_1$ begins by generating the first weak classifier $h_{i,j,1}$ using a decision stump learner. The generation of weak classifiers is combined with the re-weighting scheme of AdaBoost and it continues until $k = \Phi$, where $\Phi$ is a predetermined maximum number of weak classifier per node. Once this condition is met, AdaBoost stops and the composition of the training set for the positive samples is then modified. PSL proceeds to create a new dataset $P_{i,j+1}$ for the training of the next node $j + 1$, which comprises only of positive instances where each instance $(x_m, y_m) \in P_{i,j}$ is evaluated as $h_{i,j}(x_m) = -1$, or every

positive sample that is classified as a false negative. Therefore, each set $P_{i,j}$ is a subset of $P$ of a decreasing size for each $j+1$, that comprises of samples that are also of an increasing degree of difficulty to differentiate from the negatives. This step removes the redundant samples whose information has already been encoded within the classifiers, thus enabling the learning to focus on more difficult patterns. The training continues until all positives samples have been learned, where $P_{i,j} = \{\}$ and at least 50% of the negative samples have also been learned.

The positive set for $P_{i+1,j}$ for the subsequent layer is restored to full size, so that $P_{i+1,j} = P$, while the negative set $N_{i+1}$ is modified so that all samples $(x_m, y_m) \in N_{i,j}$ evaluated as $h_{i,j}(x_m) = -1$, are removed and randomly replaced with new samples from $N$, where $h_{i,j}(x_m) = 1$. The training continues until the maximum number of layers are created or until the entire negative dataset has been learned. The detailed procedure for constructing PSL classifiers can be seen in Algorithm 13.

**Algorithm 6**: PSL Algorithm

**Input** : Positive training set $\mathbb{P}$ and negative set $\mathbb{N}$ consisting of examples $(x_1, y_1), ...(x_M, y_M)$ where $x_m$ is a feature $\in$ X and $y_m$ is a class $\in \{-1, +1\}$, $M$ is the number of elements and with $T$ as the number of boosting rounds.

**Output**: PSL classifier $H_L$ comprising of $L$ layers and weak classifiers $h_{i,j,k}$, where $i$ signifies the layer of the weak classifiers, $j$ denotes the $j^{th}$ node of $n$ it belongs to and $k$ represents its position in the node.

**Given** : $P_{i,j}$ denotes the positives used to train node $j$ in layer $i$ where $P_{i,j} \subset P$. $N_i$ denote the negative dataset used on layer $i$ where $N_i \subset N$. $\Phi$ denotes the maximum number of weak classifiers per node.

$N_1 = \text{RandomSample}(\mathbb{N})$
**for** $i = 1$ **to** $L$ **do**
  $P_{i,1} = \mathbb{P}$
  **for** $j = 1$ **to** $n$ **do**
    **for** $k = 1$ **to** $\Phi$ **do**
      $h_{i,j,k} = \text{AdaBoost}(P_{i,j}, N_i)$
    $P_{i,j+1} = (x_m, y_m), \forall x \in \mathbb{P}$ where $(h_i(x_m) = -1) \wedge (y_m = 1)$
    **if** $P_{i,j+1} = \emptyset$ **then**
      *break*
  $N_{i+1} = \forall m, (x_m, y_m) \in N_i$, where $h_i(x_m) = -1$, replace with $(x_m, y_m) \in \mathbb{N}$, where $h_i(x_m) = 1$
  **if** $N = \{\emptyset\}$ **then**
    *break*
**return** $H_L(x) = \begin{cases} 1 & if\, \forall i\ H_i(x) = 1 \\ -1 & \text{otherwise} \end{cases}$, $H_i(x) = \begin{cases} 1 & \exists j\ h_{i,j}(x) = 1 \\ -1 & \text{otherwise} \end{cases}$, where $h_{i,j}(x) = \text{Sign}\left(\sum_{k}^{\Phi} \alpha_k h_{i,j,k}(x)\right)$

At detection time, the PSL classifiers combine a variety of strategies in order to formulate a prediction. Given an unknown instance $x$, the PSL classifier $H_L$ first uses the

weighted majority vote, (Section 2.1.2), at each node $h_{i,j}(x)$ to predict a sample. If $h_{i,j}(x) \geq 0$, then the instance is classified as a positive by node $j$, which is adopted as the decision of the layer $i$. The evaluation of the instance $x$ on layer $i$ ceases and the instance is forwarded to the layer $i+1$ for further consideration. However, if the given instance is classified as a negative, then it is forwarded to the next node $j+1$ in the same layer $i$ until either a unanimous decision to classify the sample as a negative is reached or a single positive vote by any other node is cast. The PSL classifier $H_L$ then predicts the instance $x$ as a positive only if the conjunction of all layers' decisions $H_i(x) = 1$.

The combination of the different ensemble voting schemes reflects the training procedures and the relative expertise of each cluster of ensembles within each node. While weighted majority voting is inherent to AdaBoost and reflects the relative degree of accuracy of a given weak classifier on a dataset which all the weak classifiers in that boosting round have been exposed to, the competencies of the nodes on the other hand reflect their limited expertise to only portions of data. Because a group of classifiers within each node $j$ is only exposed to a subset of the positive samples from $P$ during training, only its positive predictions can be accepted as being reliable. This is because each node $j$ is an expert on a given partition of a positive dataset that is determined by the misclassifications by the prior nodes. A negative prediction of node $j$ can with high probability be a positive sample from a different partition and therefore in order to classify a sample as a negative, all nodes in a layer require a unanimous vote. The same principle works in reverse at the cascade level between the votes of each layer where a unanimous decision is required to classify a sample as a positive, since not all layers have been exposed to all the negative samples in the training set.

The underlying principle behind the PSL approach is the separate-and-conquer strategy found in covering algorithms that are used frequently for inducing rules. Its contribution to ensemble learning, and more specifically to coarse-to-fine learning, is in its ability to simplify the learning process in respect to the positive samples. It applies to the positive samples the approach that is similar to the one used for handling negative datasets when training cascades. The consequence of this is that conceptually the training of classifiers at each layer is reduced in complexity, since the induction of class boundaries becomes easier with the removal of redundant samples even though the remaining samples are more difficult. The training time decreases since layer targets can therefore be satisfied faster and the computational complexity decreases also since the algorithm is exposed to fewer samples. The *ad hoc* layer threshold setting is also removed since the target positive hit rates no longer present a challenge. At the classification stage, the runtimes can also be expected to be faster since each layer is decomposed into a further cascade, which guarantees that all classifiers within it will only be invoked on most difficult samples.

## 4.4 Experimental Setup

The PSL framework was tested against the Viola-Jones (VJ) approach. The VJ approach has undergone numerous extensions and modifications in the last decade. Despite this, the VJ method still remains the accepted yardstick and recent methods continue to compare their results with it [129]. For this reason, VJ method was selected as the control method and also since the PSL is its direct and logical extension. In addition, the VJ

approach does not employ any feature pool manipulation methods, which means that the machine learning algorithms can be fairly compared without an interference from unknown variables.

The aim of the experiments was not to produce face detectors with industrial grade accuracies, hence smaller training sets were used. One of the key goals of the experiments was to observe the effects of varying PSL node sizes $\Phi$ on the generalization and runtimes. Therefore, four classifiers with different $\Phi$ parameters were trained on each dataset. Table 4.1 lists all training parameters. The training data used for all classifiers was a subset of the FERET face dataset.

Table 4.1: Training settings and dataset details.

| **Property** | **Attribute** |
|---|---|
| Positive and negative dataset sizes | 500,1000, 1500,2000 |
| Node sizes $\Phi$ (PSL only) | 5,10, 15,20 |
| Target training error | 0% |
| Target layer hit rate | 100% |
| Target layer false positive rate | 50% |
| Maximum nodes per layer (PSL only) | 10 |
| Maximum layers | 100 |

Classifiers were trained with balanced-sized negative and positive datasets. Negative bootstrapping in the form of [155], discussed in Section 2.3.1, was used to replace correctly predicted negative samples at the conclusion of each layer. The training continued until a zero error was achieved or until no more negative training samples could be generated. Consequently, though the explicit training involved balanced training sets, the actual learning was highly skewed since the classifiers were exposed to negatives numbering in millions. The largely disproportionate exposure of the classifier to negatives in respect to positives is necessary [175] and consistent with the operating domain for the classifiers, where rare-event detection is encountered. The classifiers were evaluated against the popular CMU MIT image dataset, whose details as well as those of the feature types and extraction methods can be found in Chapter 3.

## 4.5 Results

The results analysis is divided into three parts; namely the training phase, accuracy and the detection runtimes[2].

### Training Phase

A zero training error rate was attained by all classifiers. The training runtimes are shown in Figure 4.2. The figure shows that the PSL algorithm consistently produces classifiers at reduced total runtimes. The acceleration in the training of PSL classifiers increases over

[2]Parts of this research have been published in [157].

those of the VJ as the datasets increase in size. The PSL classifiers with smaller nodes also consistently produce shorter training runtimes across all datasets.

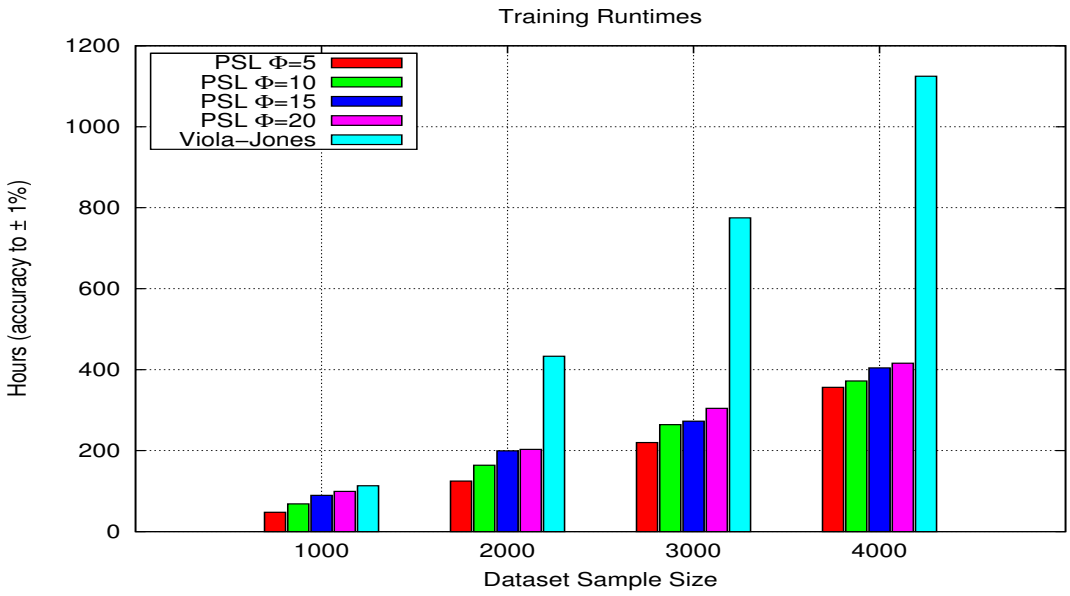



Figure 4.2: Training runtimes in seconds for all classifiers on the four CMU MIT datasets.

The primary reason for the reduction in training runtimes by PSL can be seen in Figure 4.3. The figure shows an example of the speed at which the number of positive samples are removed from the training after each subsequent node. The figures are selected from the final layer of each classifier which generally is the most complex for finding a separating boundary between classes [175].

While conventional methods like VJ explicitly train on all positives through an entire layer of a cascade, the PSL algorithm removes samples it has already learned at an exponential rate, in order to explicitly focus the learning process on more difficult samples. The cumulative reduction in the computational demand over all the layers in a cascade results in considerable training runtime reductions. Meanwhile, the need to employ artificial layer thresholds is removed, since the PSL structure rapidly converges to the required hit rate due to the stepwise simplification in the complexity of the class boundaries.

The following sequence of graphs in Figure 4.4 reveal the effects that the composition process of layers into nodes, has on the false positive rates. The graphs show the convergence of false positive rates of a typical PSL classifier on the last eight layers of a cascade.

Though most of the layers comprise only of two nodes, it suffices to demonstrate that the false positive rate which is achieved by the previous nodes, cannot be improved upon by successive nodes. The final false positive rate for a layer can, at best, equal the false positive rate of the initial node. However, in practice, this rate is considerably exceeded by the final nodes in each layer. The point is to illustrate that as the number of nodes per layer increases, in turn so does the false positive rate and the likelihood that layer targets may not be met. If layer targets are not met, then the rate at which negative samples are

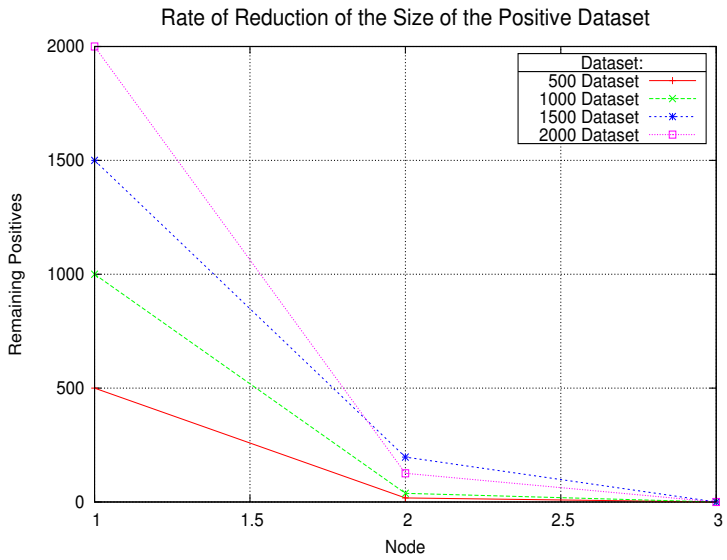



Figure 4.3: PSL classifiers ($\Phi = 10$) on four datasets and the rate at which the positive samples are removed during final layer training of a cascade.

learned decreases, since fewer unseen negatives are introduced at each succeeding layer and thereby the training runtime increases.

This is directly impacted by the $\Phi$ training variable that represents the maximum size of the nodes. Given a large enough $\Phi$, lower false positives per node become achievable. Additionally, the margin between the class boundaries increases which indicates a stronger generalization ability of the classifiers. However, if $\Phi$ is too large, then the runtime performance of PSL will resemble that of VJ cascades. Complementary figures to the convergence graphs here can be found in Appendix A.2.

The total number of weak classifiers making up the final classifiers are listed in Figure 4.5. The PSL classifiers consistently generate simpler solutions to the problems, while the VJ classifiers are larger by at least a factor of two. The variation in the size of $\Phi$ does not indicate that there is a correlation with the size of the ensemble in these experiments.

### Accuracy

The ROC curves in Figure 4.6 show classifiers for each of the training structures on the four different datasets. The more relevant intervals of the ROC curves are depicted. The strongest generalization across all datasets can be observed by the VJ classifiers. The accuracy of VJ on portions of the ROC curve depicting smaller false positive rates were several percentage points higher than those of the PSL classifiers.

A consistent feature amongst the PSL classifiers on this test dataset is that the classifiers trained with the smaller $\Phi$ values produced the weakest generalization rates. As node sizes increase, there is evidence that the generalization strengths of the classifiers also increases. This can be explained in terms of the margin theory [143]. The larger nodes are able to further increase the distances of each class from the separating boundaries during training. In turn, the accuracy of each node and consequently the accuracies of each layer,

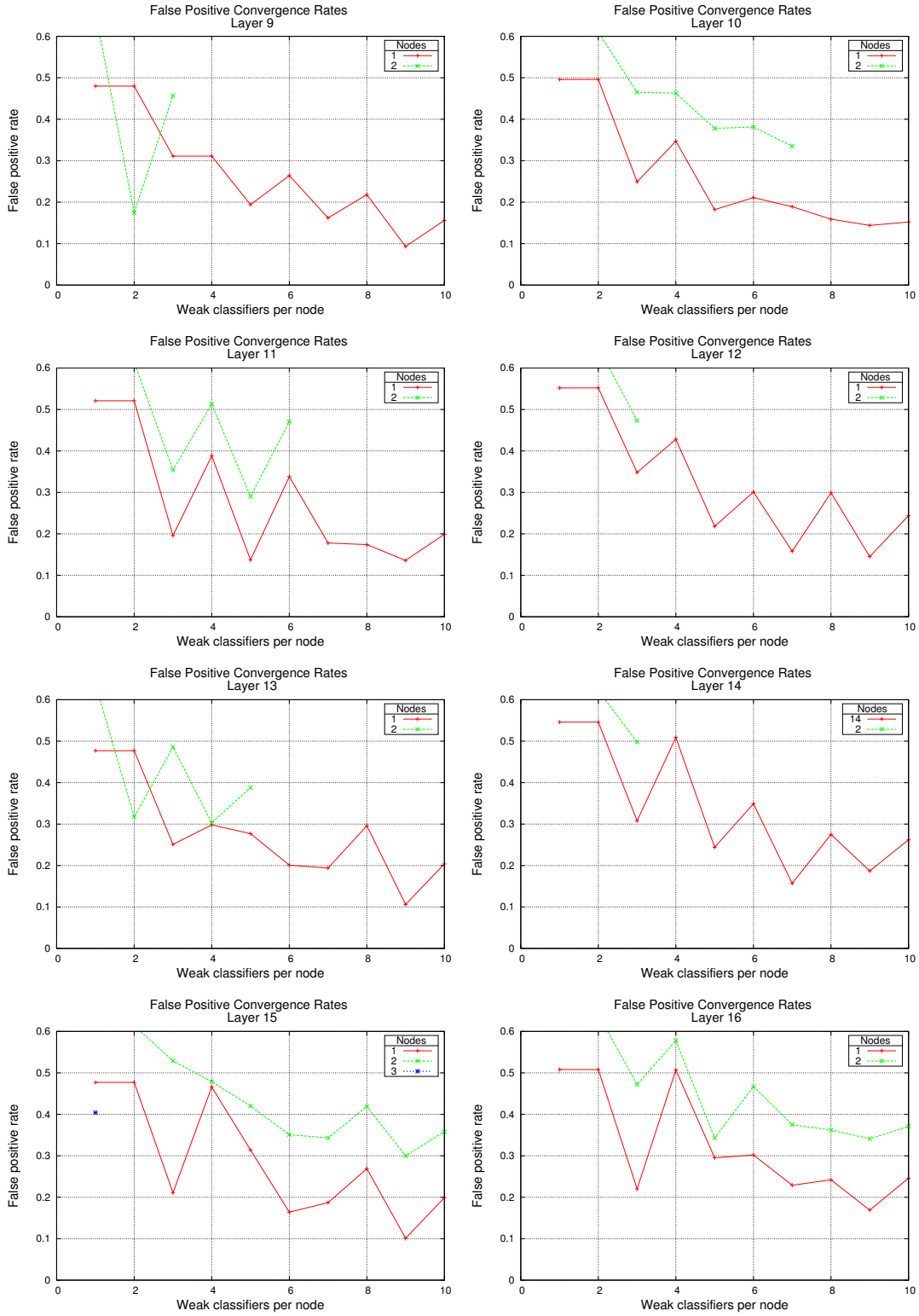



Figure 4.4: Graphs showing typical PSL convergence patterns for the false positive rates for a classifier $\Phi = 10$.

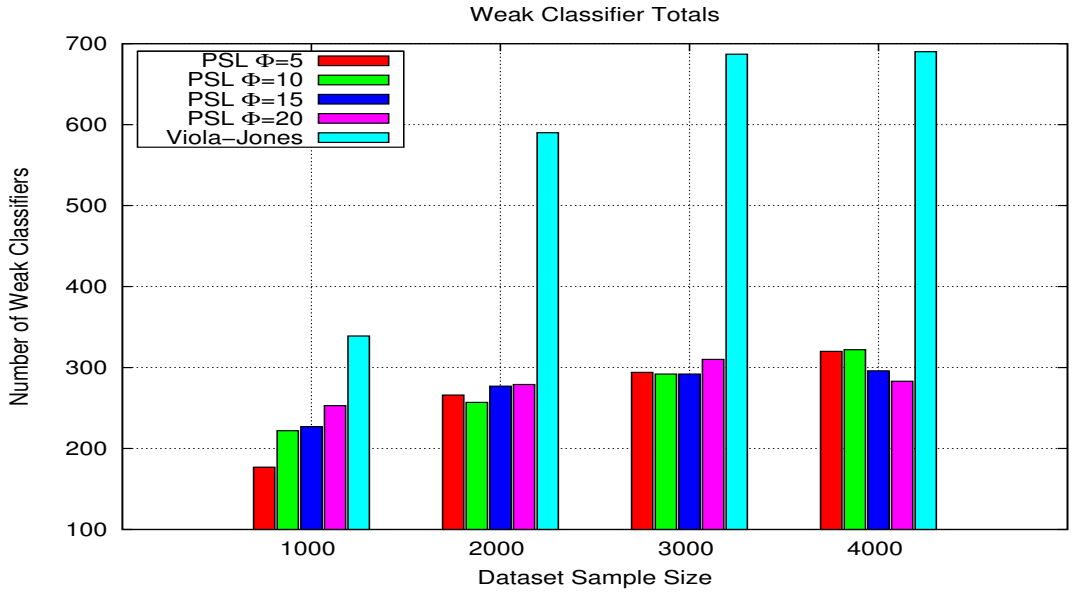



Figure 4.5: Weak classifiers totals per strong classifiers for each framework on the four CMU MIT datasets.

is expected to increase.

In saying that, the expectation that the classifiers with the largest nodes $\Phi = 20$ produce best generalization has not been met. Though there is a tendency for an improved generalization as the node sizes increase, there is no guarantee that the largest nodes invariably produce classifiers with the best overall accuracies. The subsequent chapter will examine the reasons for this in more depth.

Though the results indicate that the VJ method has produced stronger generalizations on this problem set than the PSL, this has been achieved at a considerable cost in the training runtimes. This trade-off can be seen Figure 4.7 which shows the classifier accuracies as the function of the cost of training the classifiers over the three datasets. The figures indicate that the PSL algorithm operates at a significantly lower training runtime cost from producing equivalent test error values. The divergence between the costs of the VJ and PSL structures increases as the size of the training sets becomes larger. Additionally, the lowest costs are recorded by PSL classifiers with the smallest values of $\Phi$. The cost consistently increased as the value $\Phi$ also increased.

### Detection Runtimes

The total number of weak classifiers making up a final strong classifier is to a large degree indicative of its detection runtime capabilities. However, the manner in which the classifiers are structured is even more crucial. Larger ensemble classifiers can be structured in such a way that enables them to execute faster than simpler ensemble classifiers if the initial layers of the cascade are comprised of fewer weak classifiers. This assertion holds provided that the initial layers have the capacity to reject large numbers of trivial negative samples. For this to happen, some explicit optimization for given detection runtimes

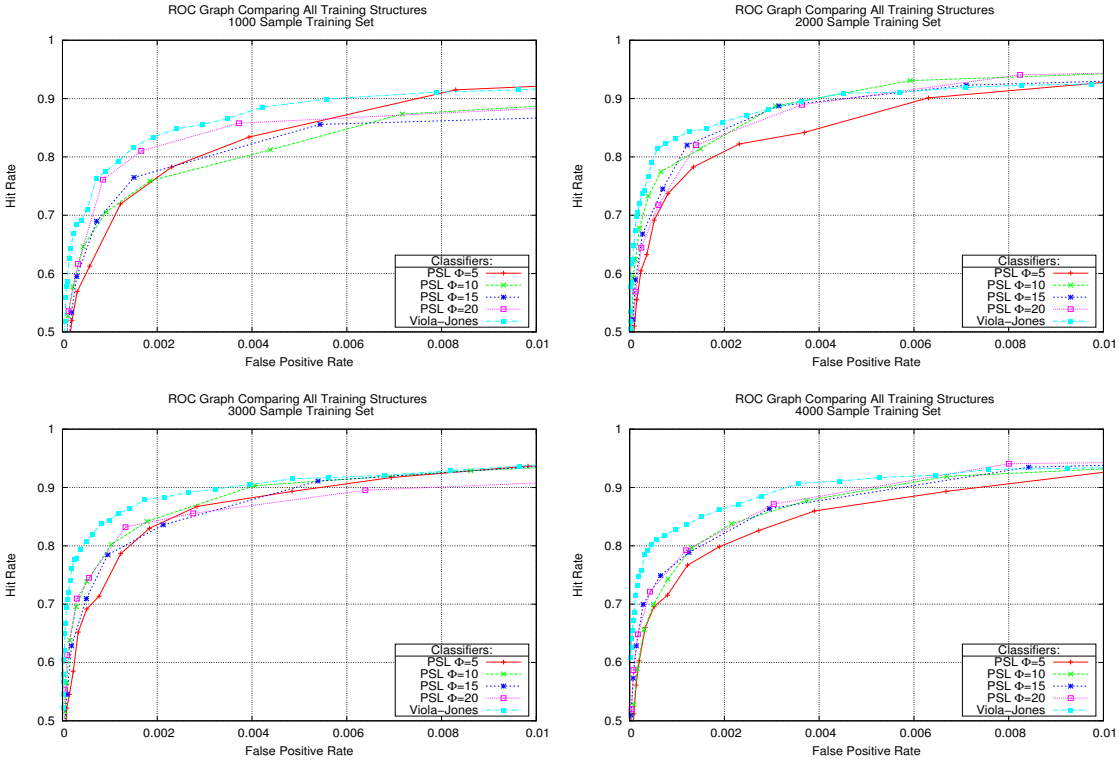



Figure 4.6: ROC graph curves on the CMU MIT datasets showing the generalization patterns of all classifiers.

need to be specified[3]. The classifiers here were not optimized, but were left unaltered in order to eliminate the optimization process as a potential variable that can affect their generalizability.

Consequently, the runtimes of the non-optimized and naive classifiers were compared. Figure 4.8 displays these runtimes as total seconds required to perform all detections on the entire test set. The VJ performances were frequently the slowest. The speed of the PSL classifiers decreased as the node complexity increased. PSL classifiers with the largest values of $\Phi$ resembled the performance runtimes of VJ due to the fact that the initial layers of the cascade were forced to become larger.

The performance advantages of the PSL classifiers with the smallest nodes illustrate the point that the smallest overall ensembles are not necessarily the fastest. In reference to the Figure 4.5, PSL classifiers with $\Phi \in \{5, 10\}$ generated ensembles that were of equal size to the remaining PSL classifiers, while occasionally even being larger. However, due to their ability to create more compact layers especially in the initial stages, their detection runtimes were considerably faster than those of the larger nodes. This demonstrates an advantageous characteristic of the PSL algorithm in regards to its ability to tune the classifiers for a specific runtime requirement. This can be trivially achieved by pre-defining the initial nodes of a cascade to be of a smaller size, while the nodes in the

[3]Viola and Jones [175] discuss the importance of the first few layers of a cascade to the detection runtimes and also of the importance of being able to control this parameter.

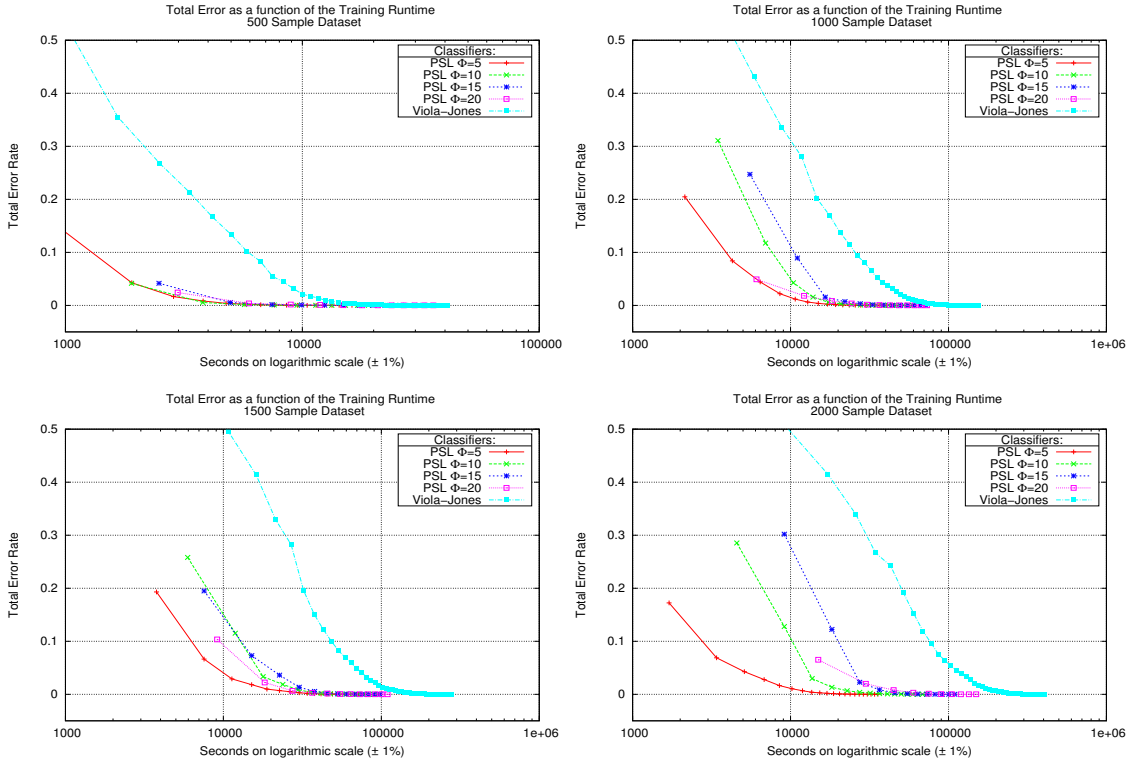



Figure 4.7: Total error rate on the CMU MIT test set, as a function of the training runtime.

latter layers can be increased to much larger sizes. Appendix B shows the results of experiments conducted on PSL-like classifiers with variable node sizes for different layers for the purpose of tuning for optimal performance runtimes. The results indicated that it is trivial to adjust a PSL-like classifier for a required runtime without significantly compromising their generalizability.

## 4.6 Discussion

The experiments have demonstrated that the PSL approach to training classifiers using a strategy that decomposes cascade layers into clusters of weak classifiers, ultimately leads to a substantial reduction in training runtimes. This was highlighted by the chosen problem domain of face detection with Haar-like features, where large feature spaces are generated and high overheads are involved in re-computing the features. By rapidly reducing the number of training samples that the algorithm explicitly sees and by requiring fewer weak classifiers to be generated, the PSL has shown its applicability in domains where large and expensive feature pools are employed.

However, at least on this problem domain, it has also been shown that the increased training efficiency comes at a notable trade-off cost in the final generalization strength of the classifiers. This can be understood through the theory of margins which states that the expected strength of a classifier's generalizability is contingent by the degree to which

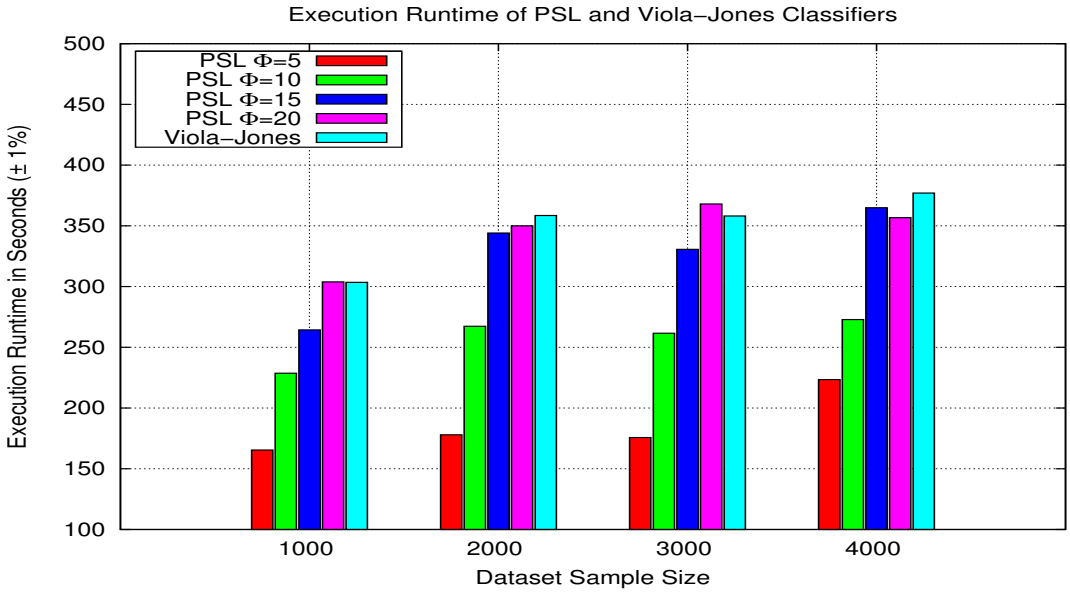



Figure 4.8: Execution runtimes for PSL and Viola-Jones classifiers on the CMU MIT test datasets.

the classifier's predictive confidence is developed at training. Since in the PSL scenario each node is restricted by a predefined and comparatively limited number of boosting rounds, the consequence is that the confidence margin cannot be extended to the same degree that it can be in the conventional cascaded approaches. Though any given layer within a PSL classifier might have the same number of weak classifiers as that of the VJ classifier, the VJ classifiers continually boost from the beginning of a layer to the end. They are therefore able to achieve a greater margin from the class separation boundary.

In order to counter this problem, the size of $\Phi$ can simply be increased. However, this is not likely to produce a satisfactory solution for two reasons. Firstly, the larger the $\Phi$ becomes, the more the PSL training will resemble that of the conventional cascade algorithm runtimes; albeit, without the threshold adjustment which will then make the convergence to layer training targets even more difficult. The likelihood of then generating even larger ensembles becomes a real possibility. Secondly, due to the fact that higher hit rates will become realizable with larger nodes, the outcome will be that only small numbers of positive samples will be available for training the final nodes. The strongly skewed training data will in that case not be conducive to attaining accurate classifiers.

Alternatively, the boosting strategy within each layer can be modified in such a way that it becomes continuous despite there being multiple nodes. The modification can allow the boosting to resume with each sample preserving its weights from where it left off each time a new node is created. The remaining positive sample weights can be normalized in order to account for the positive samples that are removed and the boosting process can continue extending the confidence margin of the remaining unlearned samples. The obvious outcome of this approach would be to undermine the independence of the nodes from one another which is an advantageous characteristic of the PSL. Alternative strategies

can be devised that preserve this. By modifying the current node termination strategy from being one that is training-error based with fixed node sizes, to one that is centred on training nodes until a given percentage of samples have attained a required confidence margin, then the accuracy of each node can theoretically be improved. The termination of each node would subsequently require that only samples that have attained a confidence margin above a certain threshold are removed from training. This would leave the positive samples that are correctly classified, but too close to the decision boundary, to undergo more training. Naturally, the training runtime of each layer can thus be expected to be slower than that of the current PSL since fewer positive samples would be removed after each node. However, given the sound principles of the margin theory, this strategy can be expected to bring significant generalization improvements.

The other contributing factor to the decreased generalizability of the PSL is due to its rapid reduction of the positive samples in each node which creates imbalanced learning problems. The perennial problem that divide-and-conquer and separate-and-conquer strategies face, is the danger of overfitting when recursively inducing rules on datasets of diminishing size [184]. Though imbalanced training, in and of itself, does not guarantee that a compromise in the generalizability will occur, it has however been shown that under certain conditions this can be be expected [62].

It is a well established fact that overfitting is likely to occur when training takes place in setting where the dimensionality of the data feature space overwhelmingly overshadows the number of training data [34, 169]. This is then exacerbated as the number of training samples are reduced to the point where small datasets are used for training, since irrelevant features can unexpectedly appear to be meaningful [92, 184]. This is precisely what occurs with PSL when it is deployed on the problem of face detection using Haar-like features. On each round of boosting, up to 200,000 Haar-like features are generated representing an over-complete feature set. As the PSL algorithm reduces the number of remaining positive samples per new node, it is highly likely that features are selected that overfit the decision boundaries on the reduced positive sets. As such, the PSL can be described as being susceptible to compromises in the generalizability of classifiers when used on problem domains with large feature pools.

## 4.7 Summary

In this chapter, the PSL framework was applied to the domain of face detection. The goals were to explore the ability of the PSL algorithm to reduce training runtimes over the conventional approaches like Viola-Jones; while, preserving the real-time capabilities of the resulting classifier and observing the trade-offs in the generalizability of the classifiers. The experiments also sought to gain a deeper understanding into the effects of the varying the $\Phi$ value for node complexity, on the training, detection and accuracy of the classifiers. The research has found that:

1. The PSL structure considerably reduced the training runtimes over the Viola-Jones style frameworks.

2. As the datasets increased in size, the degree of divergence in training runtimes also increased.

3. The faster training runtimes were achieved by a decreased generalizability on this problem domain.

4. The PSL rapidly learns and removes positives as it generates new nodes. However, as more nodes are generated the harder it becomes to meet the false positive layer targets.

5. Simpler classifiers are produced by the PSL algorithm.

6. Faster detection runtimes were registered by PSL classifiers with smaller values of $\Phi$.

# Chapter 5

# Advanced Bootstrapping Technique for Cascaded Classifiers

Initial experiments with the PSL method on real-world problems from the previous chapter revealed its potential for further development. Training runtimes have been reduced and the training process has become more tractable. Some compromise in the generalizability of the PSL classifiers, however, has been the trade-off.

With the advent of streaming data, the volume of training datasets for a number of problem domains has been steadily increasing. This often means that the size of the datasets has been increasing substantially for both the negative and positive samples. While the cascaded approaches have been shown to be effective at handling datasets that comprise of hundreds of millions to billions of negative samples, they have been able to accomplish this due to the presence of large skews in the positive datasets which necessarily comprised of a small fraction of the negative datasets [191].

The cascaded approaches are able to handle massive numbers of negative samples through the supervised under-sampling methods of Sung and Poggio [155] and Viola and Jones [175], which prevent the algorithm from having to *explicitly* learn the patterns of every sample. Massive positive datasets are therefore computationally unfeasible for conventional cascaded approaches since, to date, there are still limited capabilities within cascaded frameworks that enable bootstrapping on positive samples. The research in this chapter sought to explore ways of enabling positive sample bootstrapping to cascaded ensembles.

## 5.1 Motivation

This chapter considers the challenges found in domains where large positive datasets are necessary for training alongside massive negative sets. The proposal is to leverage the modular nature of the PSL structure in order to implement bootstrapping for positive as well negative samples and to explore the effect that much larger datasets have on the generalization ability of the PSL classifiers. Therefore, the questions that this chapter will attempt to answer are the following:

1. Can PSL be extended to include positive sample bootstrapping which is efficient without compromising accuracy?

2. Does the generalization of the PSL classifiers improve with larger datasets in comparison to the Viola-Jones approach?

3. Generally, what are the causes of lower accuracies within the PSL method when compared to the Viola-Jones classifiers?

## 5.2 Related Work

The research here continued using the face detection problem as the medium for exploring these questions. While the predominant body of research in the last decade has been devoted to improving various components of the CoBE architecture, only in recent years has research begun to surface with methods enabling cascaded frameworks and more specifically face detector training on large positive datasets [191]. Xiao et al. [190] were first to propose a bootstrapping approach. Their method consisted of training directly only a small subset of a massive positive dataset, while the distribution of the subset was iteratively compared to that of the entire positive dataset and updated correspondingly. Their face detector was implemented for distributed learning on a number of machines and displayed an impressive capability for handling large positive datasets.

Yan et al. [191] proposed a simpler approach for non-distributed processing in which training on a subset of an entire positive dataset was also applied. Their method employed iterative training of each cascade layer using only a positive subset. During the layer construction of their approach, the intermediate sub-classifier undergoes periodic validation against the entire positive dataset. When the detection rate falls below a designated target, a portion of misclassified positive samples from the massive positive set are added to the training subset. The learning resumes and continues until the targets are met.

## 5.3 PSL with Positive Sample Bootstrapping

The bootstrapped PSL (BPSL) is an intuitive extension of the original PSL algorithm. By continuing use of the same notation from Chapter 4.3, the BPSL departs from PSL with the introduction of a positive training set $P_{i,j}$ (termed the *base set*), which is a subset of the whole positive set $\mathbb{P}$ (termed the *reserve set*), and is of a fixed size $\beta$. For each initial node $h_{i,1}$ of each layer $i$, the positive subset $P_{i,1}$ is randomly sampled from $\mathbb{P}$ without replacement. Subsequently, following the completion of training a node $h_{i,j}$, the contents of $P_{i,j}$ is validated against the node so that all correctly learned samples as in $h_{i,j}(x_m) = 1, \forall m, (x_m, y_m) \in P_{i,j}$ are removed, which results in a new dataset $P_{i,j+1}$. The classifier $h_{i,j}$ is then applied to the remaining samples in the entire positive dataset $\mathbb{P}$ and the correctly classified samples are likewise removed from further training of the layer $i$. The next step augments $P_{i,j+1}$ with additional samples from $\mathbb{P}$ so that the size of the subset is $\beta$. The training for the layer continues until $P_{i,j+1} = \emptyset$. The training for a new layer $h_{i+1}$ begins by resetting all samples in $\mathbb{P}$ in order to make them available. In every other respect, BPSL proceeds identically as PSL. The complete procedure for the BPSL cascade training can be seen in Algorithm 7, while the Figure 5.1 demonstrates this graphically.

While the conventional bootstrapping procedure used for negative samples applies the bootstrapping after each layer $i$ is trained, the BPSL applies continuous positive samples

**Algorithm 7**: BPSL

**Input** : Positive training set $\mathbb{P}$ and negative set $\mathbb{N}$ consisting of examples $(x_1, y_1), ...(x_M, y_M)$ where $x_m$ is a feature $\in$ X and $y_m$ is a class $\in \{-1, +1\}$, $M$ is the number of elements and with $T$ as the number of boosting rounds.

**Output**: PSL classifier $H_L$ comprising of $L$ layers and weak classifiers $h_{i,j,k}$, where $i$ signifies the layer of the weak classifiers, $j$ denotes the $j^{th}$ node of $n$ it belongs to and $k$ represents its position in the node.

**Given** : $P_{i,j}$ denotes the positive dataset of size $\beta$, used to train node $j$ in layer $i$ where $P_{i,j} \subset P$. $N_i$ denote the negative dataset used on layer $i$ where $N_i \subset N$. $\Phi$ denotes the maximum number of weak classifiers per node.

$N_1 = \text{RandomSample}(\mathbb{N})$
**for** $i = 1$ **to** $L$ **do**
  $P_{i,1} = \text{RandomSample}(\mathbb{P}, \beta)$
  **for** $j = 1$ **to** $n$ **do**
    **for** $k = 1$ **to** $\Phi$ **do**
      $h_{i,j,k} = \text{AdaBoost}(P_{i,j}, N_i)$
    $P_{i,j+1} = (x_m, y_m), \forall x \in P_{i,j}$, where $h_{i,j}(x_m) = -1 \wedge y_m = 1$
    $s^P = \text{SizeOf}(P_{i,j+1})$
    **for** $k = 1$ **to** $\beta - s^P$ **do**
      $P_{i,j+1} = \text{RandomSample}(\mathbb{P}, 1)$, where $(h_i(x_m) = -1) \wedge (y_m = 1)$
    **if** $P_{i,j+1} = \emptyset$ **then**
      *break*
  $N_{i+1} = \forall m, (x_m, y_m) \in N_i$, where $h_i(x_m) = -1$, replace with $(x_m, y_m) \in \mathbb{N}$, where $h_i(x_m) = 1$
  **if** $N_{i+1} = \emptyset$ **then**
    *exit*

**return** $H_L(x) = \begin{cases} 1 & if\ \forall i\ H_i(x) = 1 \\ -1 & \text{otherwise} \end{cases}$, $H_i(x) = \begin{cases} 1 & \exists j\ h_{i,j}(x) = 1 \\ -1 & \text{otherwise} \end{cases}$, where $h_{i,j}(x) = \text{Sign}\left(\sum_{\text{k}}^{\Phi} \alpha_k h_{i,j,k}(x)\right)$

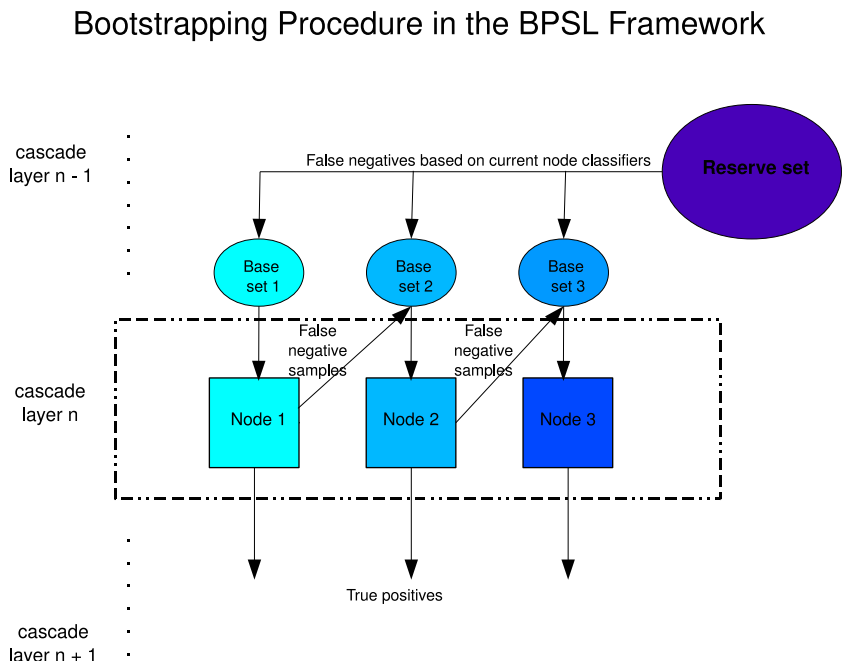



Figure 5.1: The propagation of the positive samples in the BPSL bootstrapping method.

boostrapping within each layer $i$ from node $j$ to $j+1$. The approach guarantees that though the learning algorithm *sees* all positive samples when each classifier $h_{i,j}$ is applied to $\mathbb{P}$, crucially it will not *explicitly learn* every sample in the reserve set $\mathbb{P}$. The rate at which the number of positive samples which have to be learned decreases within a layer, is contingent on the discriminative power and relevance of $h_{i,j}$ in respect to $\mathbb{P}$.

The BPSL differs from the bootstrapping approach of Xiao et al. [190] in that there is a sequential dependence between the types of samples that classifiers within each subsequent node are trained on. In the case of BPSL, clusters of more difficult patterns are learned at every additional node. Yan et al. [191] on the other hand continuously augment the same subset of the positive samples with the new ones that the previous classifiers have misclassified. Depending on the complexity of the dataset and the strength of the weak learner, the positive dataset may continue to expand to the point that the induction process slows down rapidly.

### 5.3.1 Experimental Setup

An additional 13000 facial images were collected and combined with the samples from the previous experiments to give a total of 15000 positive samples for comparing PSL, BPSL and Viola-Jones (VJ) methods. The details of the datasets are explained in Section 3.1. The additional images contained some degree of articulation and rotation as well as considerable illumination variations that sometimes resulted in feature occlusion. An example of these images can be seen in Figure 5.2. The negative dataset consisted of the identical 2500 background images from the experiments in the previous chapter.

Three positive datasets of different sizes were compiled from the pool of 15000 images. They consisted of 5000, 10000 and 15000 samples. On each dataset, one VJ classifier and four PSL and BPSL classifiers with different parameters for the node size $\Phi$ were trained. These parameters are consistent with the experiments in the previous chapter.

The BPSL classifiers with values for the base set size $\beta$ were 500 and 2000. These were deemed to be appropriate parameter values given the empirical results of the previous chapter in which datasets ranging from 500 to 2000 samples produced reasonable accuracy results that did not significantly vary from one another.

Figure 5.2: Example of the positive dataset instances.

The size of the negative datasets used on all BPSL classifiers for each node was fixed at 2000 samples. This meant that uneven training took place for BPSL positive base sets when $\beta = 500$. However, the use of unevenly-sized positive and negative datasets with a ratio of 1:4 is substantiated by Viola and Jones [174]. Their research similarly employed this approach to achieve higher rejection rates of negatives samples more rapidly at runtime. Bourdev and Brandt [10] also reported that it was possible to achieve accurate classifiers using uneven set sizes; however, they also found that the training is improved if the relative weight distribution between positive and negative samples is maintained. This recommendation was followed in these experiments. The remaining parameters are summarized in Table 5.1.

Table 5.1: Training settings and dataset details.

| **Property** | **Attribute** |
|---|---|
| Positive datasets size | 5000,10000,15000 |
| Negative dataset size | 2000 |
| Base-set size $\beta$ | 500,2000 |
| Node sizes $\phi$ | 5,10,15,20 |

Given that there is a degree of randomness in the BPSL algorithm as to which positive samples are selected for training of each node from a pool of all positive samples, naturally some variation in the accuracy of the classifiers can be expected. However, given the large number of classifiers that required training, the protracted training runtimes and the limited hardware resources; it was not feasible to train each classifier multiple times in order to extract the desirable statistical data covering means and standard deviations. Instead, only a selection of classifiers were trained multiple times and the statistical data extracted from them served as indicators for the expected variations that could be anticipated from the same classifiers using similar tunable parameters. These classifiers were

trained 10 times which enabled the analysis of the mean and the standard deviations to be conducted on the ROC curves in particular.

### 5.3.2 Results

The results analysis is divided into three parts; namely the training phase, accuracy and the detection runtimes[1].

#### Training Phase

The training runtimes for all classifiers can be seen graphically in Figure 5.3. The results confirm the findings from the previous chapter, which found that the PSL method trains classifiers significantly faster than the VJ approach. The figures also indicate that the bootstrapping procedure of BPSL for both $\beta = 500$ and $\beta = 2000$ has considerably reduced the training runtimes over the original PSL.

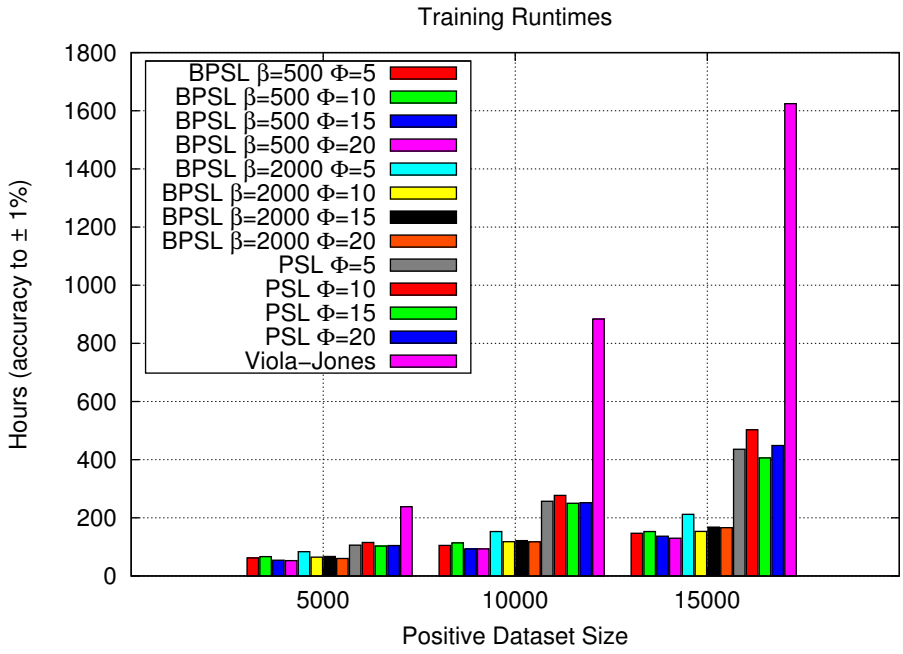



Figure 5.3: Training runtimes for PSL, BPSL and VJ classifiers

For clarity the training runtimes are summarized in Table 5.2 from the perspective of the speed-up factors attained between the various algorithms. The results indicate that as the training sets become larger, the speed-up factor of PSL and BPSL algorithms over VJ also became more acute. Likewise, the same pattern was observed when comparing the BPSL and PSL algorithms. The BPSL classifiers trained with base set $\beta = 500$ attained the fastest training runtimes. Overall, the results show that the BPSL algorithm is the most resistant to increases in the size of positive datasets.

[1]Parts of this research have been published in [158].

Table 5.2: Comparison between the training runtimes of VJ, PSL, BPSL in the form of speed-up factors. For PSL and BPSL, the results are averages across all values of $\phi$.

| Dataset | PSL vs. VJ | BPSL ($\beta = 500$) vs. VJ | BPSL ($\beta = 2000$) vs. VJ | BPSL ($\beta = 500$) vs. PSL | BPSL ($\beta = 2000$) vs. PSL |
|---|---|---|---|---|---|
| **5000** | 2.3 | 4.3 | 3.7 | 1.8 | 1.5 |
| **10000** | 3.3 | 8.4 | 6.7 | 2.6 | 1.9 |
| **15000** | 3.6 | 11.5 | 9.4 | 6.3 | 4.8 |

The rapid training runtimes of the BPSL classifiers can solely be attributed to the bootstrapping component which implicitly learns all relevant patterns of the positive class without explicitly needing to learn each individual instance. Figure 5.4 shows the example of the speed at which the size of the reserve set $\mathbb{P}$ is reduced following the training of each new node. The data is taken from the final layer of training which is ordinarily the most difficult for meeting the target positive rates. The data from three classifiers is shown with one classifier from each of the three datasets.

The figure indicates that following the training of the initial node, the number of remaining positive samples in the reserve set reduces exponentially. From the third node onwards, there is little to differentiate the classifiers in respect to the number of remaining unlearned positive samples in the reserve set. This indicates the ability of the BPSL approach to encapsulate relevant information about a dataset using only relatively small subsets, thereby removing the redundancy of having to explicitly train on every instance of an entire positive dataset.

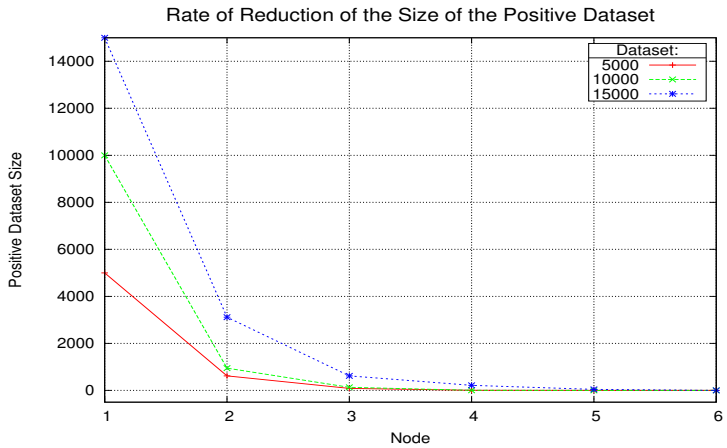



Figure 5.4: Convergence of positive samples at training towards 100% hit rates or inversely as 0% false negative rates. The convergence patterns of all three classifiers are plotted highlighting the trends of positive samples in the final layer of each classifier.

### Classifier Accuracy

Figure 5.5 plots the ROC curve for a single BPSL classifier that was trained 10 times in order to generate statistical information on how much variation can be expected from the BPSL classifiers discussed in the subsequent graphs. The classifier was trained on a 5000 sample dataset with the tunable parameters set as $\Phi = 15$ and $\beta = 500$ and is plotted at the same scale that the succeeding ROC curves are displayed in. The variation in the standard deviation of the hit rates increases for very low false positive rates, while the reverse is true of the false positive rates themselves. At the most critical and interesting points in the ROC curves that is found closest to the top left hand corner, the variation in the expected positive and false positive rates can be expected to be the smallest.

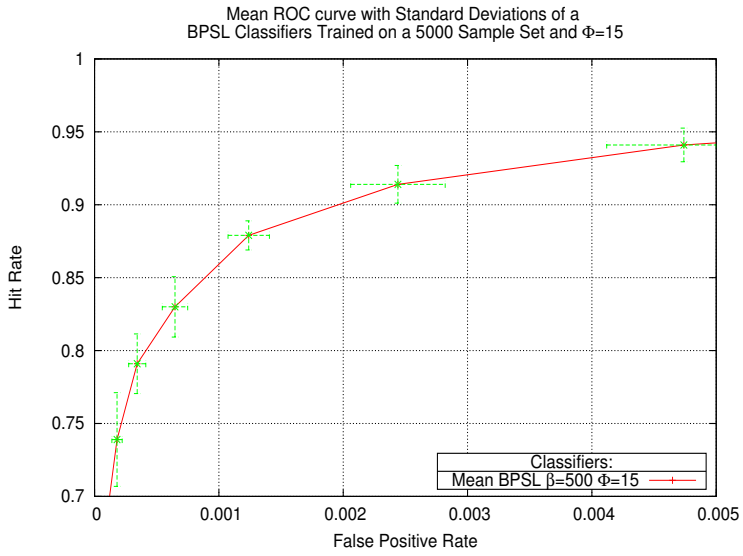



Figure 5.5: The mean ROC curve with the standard deviations of both the hit and false positive rates of a BPSL classifier ($\Phi = 15$ and $\beta = 500$) trained on a 5000 sample dataset generated from ten training repetitions.

For brevity, the ROC curves for classifiers trained on 10000 and 15000 size datasets are featured in Figure 5.6[2]. The ROC curves highlight the more critical regions of trade-offs between the hit and false detection rates and for clarity, each graph depicts curves from classifiers with a different value of $\Phi$. From these figures it is apparent that the VJ classifiers have generalized substantially better than the PSL and BPSL classifiers with different permutations of parameters on this test dataset.

Between the PSL and BPSL classifiers, the comparison of accuracies varied over the range of training parameters and datasets sizes. For the parameter $\Phi = 5$, there was a similar accuracy across all classifiers. Likewise, where $\Phi$ was set to 10, the accuracies were comparable, though BPSL $\beta = 500$ dropped off somewhat on the largest dataset. For $\Phi = 15$, the BPSL $\beta = 2000$, performed slightly better on average, while the PSL displayed poor accuracy on the 10000 dataset. The PSL arguably generalized slightly better than the BPSL classifiers for $\Phi = 20$.

[2]The remaining ROC curves from the 5000 sample dataset are located in Appendix B.1.

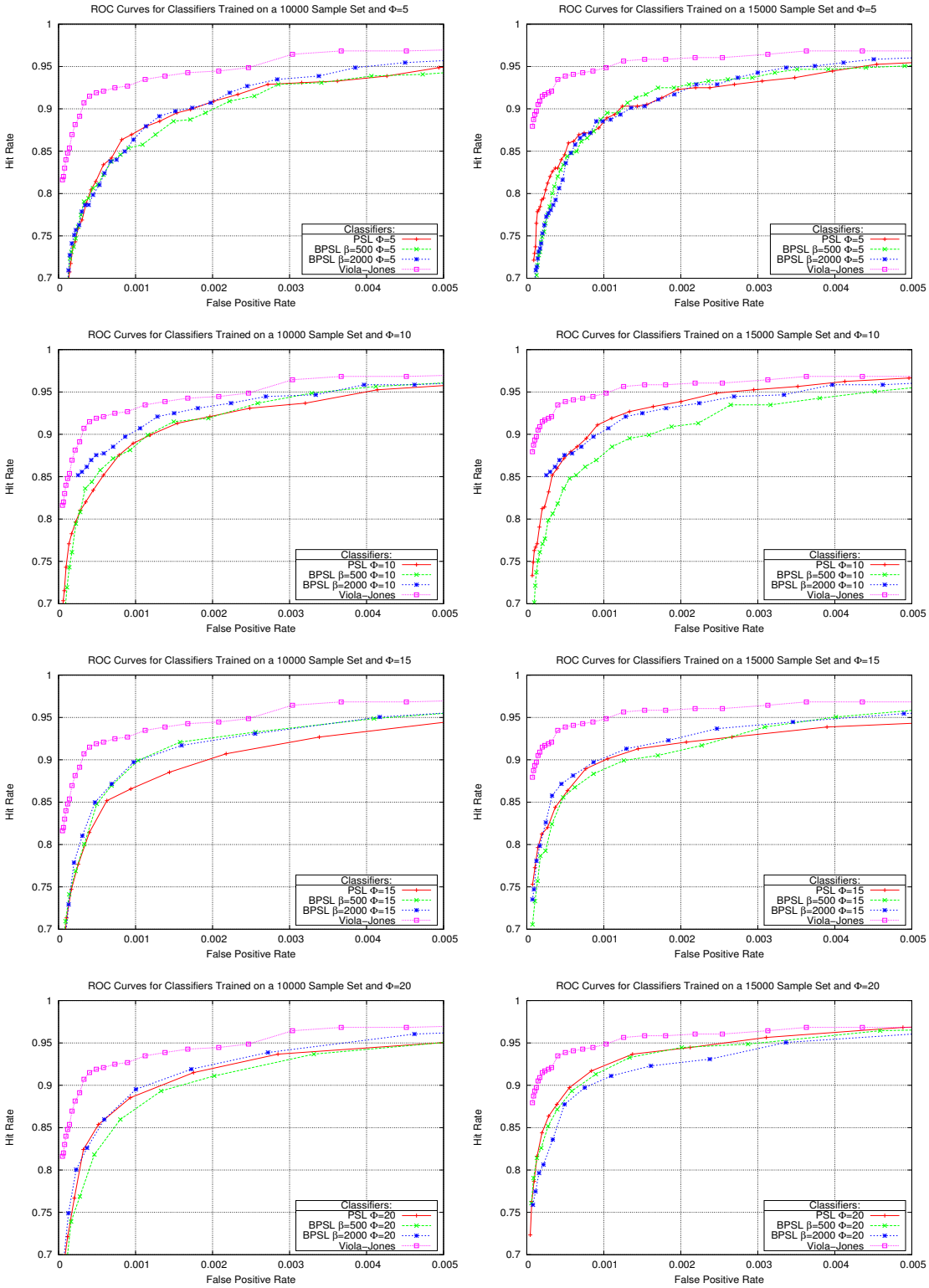



Figure 5.6: Classifier ROC graph curves on the CMU MIT dataset.

Though the PSL-like classifiers have not generalized as well as the VJ classifiers on this dataset, the degree of the trade-off that was involved in the lower training costs of the former algorithms can be seen graphically in Figure 5.7. The superior accuracy of the VJ classifiers, which ranged between 5-10 percentage points on the critical points of the ROC curves, can be contextualized by their training runtime costs that overshadow those of the PSL and BPSL classifiers. The training costs associated with the BPSL classifiers are noticeably lower than those of the PSL, and visibly decrease as the training sets increase in size. Given that there was a lack of evidence to suggest that the bootstrapping component resulted in a deterioration of accuracy for the BPSL classifiers, it can be said that the bootstrapping mechanism is effective at reducing training runtimes without compromising the generalizability of PSL-like classifiers.

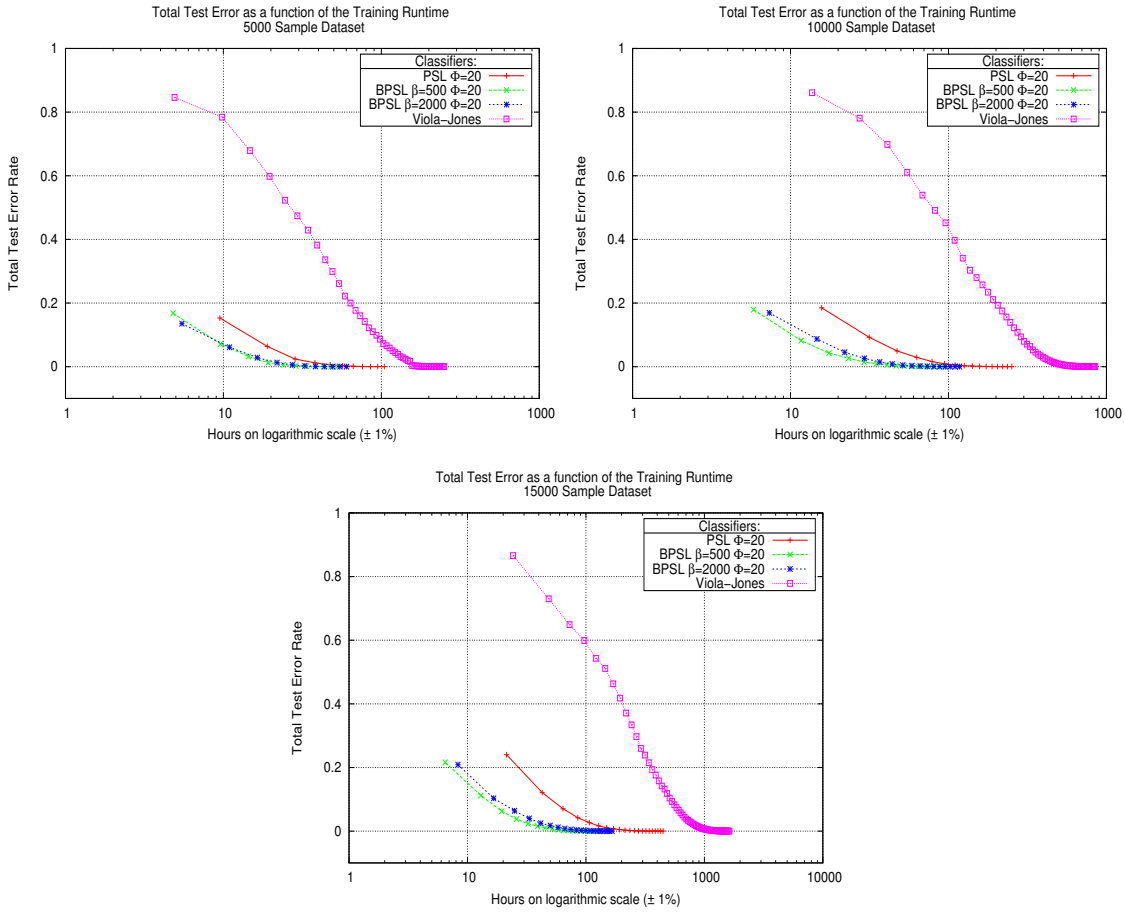



Figure 5.7: The cost/benefit trade-off between the overall test error rates and the training runtimes of Viola-Jones and PSL-like algorithms on the CMU MIT dataset.

The PSL-like algorithms provide a mechanism for trading-off a degree of accuracy for the benefit of faster training times and the ability to handle very large positive datasets. The degree to which the accuracy of these methods has deteriorated in these experiments is applicable to this domain only, and is likely to vary from one domain to the next. Though the introduction of this amount of error may make these algorithms impractical

for the required precision in face detection given this test data, for other domains, the increased error might be of an acceptable level given the trade-off in respect to the reduced training runtimes.

In order to better understand the root causes of the lower accuracy of the PSL-like methods, additional analysis was performed. The investigation centered around analyzing the PSL-like classifiers down to the node level of each layer. The goal was to scrutinize the quality of the weak classifiers that cluster together and define each node by examining what training instances they tended to be *exposed* to and what types of instances they successfully *learned*.

The diminishing size of the positive datasets used on nodes that are positioned towards the latter stages within each layer was first identified as a possible contributing factor of accuracy deterioration. The decreasing size of the positive subsets meant that the degree of skewness between the positives and the negatives increased as additional nodes were created. Previously shown Figures 5.4 and 4.3, demonstrated the speed at which the number of available positive samples were removed from the layer training. In these figures it was observable that the number of available positives in the trailing nodes tended to be very few in relation to the negatives.

The rapid reduction of the positive training samples was presented as the mechanism through which PSL efficiently achieved layer targets, simplified class decision boundaries and accelerated training. While research [7] showed that imbalances of this sort do not automatically result in a deterioration in accuracy, the class-imbalances have been shown to do so in the presence of in-class variations of the target concept or in the presence of noisy samples. As previously pointed out, it is also generally accepted that generating classifiers on datasets of decreasing size with large feature pools, does elevate the likelihood of irrelevant features being selected.

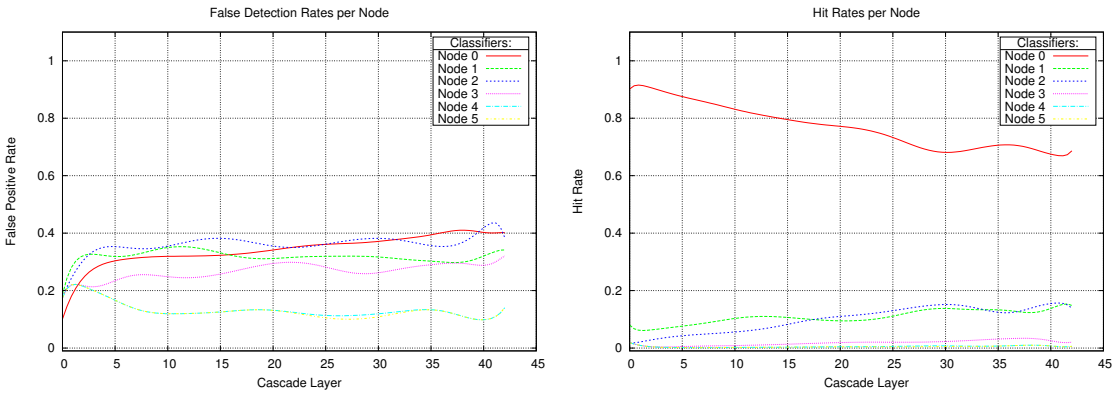



Figure 5.8: BPSL classifier $\Phi = 10$ from a 15000 sample dataset. Proportional false positive rates per node for each layer and a node's hit rate contribution for each layer.

The next step involved analyzing the accuracies between the nodes situated within the same layers. The hit and false positive rates of each node were isolated and plotted as shown in Figure 5.8. The figure shows a typical example of this analysis on a selected classifier. The false positive rates for each node represent the proportion of misclassifica-

tions a node makes in respect to the total samples it has been exposed to. The hit rate represents the contribution that each node makes to the overall layer hit rates.

While it was expected that the initial nodes display high false positive rates since they are exposed to more samples, this was also expected to be balanced out by high hit rates as confirmed by the figure. The significant piece of information came from the ratio of hit rates to false positive rates for nodes which are positioned towards the trailing section of the layers. It showed that the contribution that these nodes make toward positive detections is low compared to their relative false positive rates. The evidence pointed to a deterioration in accuracy being introduced by the trailing nodes within each layer.

In the final step, both the character of the base sets and the types of instances that each node tends to learn were analyzed. This yielded informative patterns which can be seen in Figure 5.9. The figure shows four image clusters a) to d) consisting of positive samples. The clusters represent the types of positive images that nodes at different positions within a layer tended to learn. Clusters a) and c) represent the types of images that the first nodes within a layer tended to learn. These were taken from a selected classifier from two different layers in a cascade. Clusters b) and d) represent the types of images that comprised the positive datasets for the last nodes within a layer and the images they eventually learned.

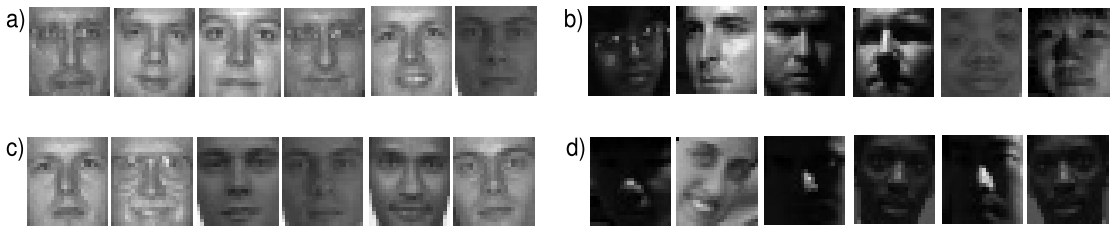



Figure 5.9: Examples of positive images learned at different points within the layers on a 15000 sample BPSL $\Phi = 10$ classifier. Cluster a) node 1 layer 30 b) last node layer 30 c) node 1 layer 42 d) last node layer 42.

The positive images learned by initial nodes in clusters a) and c) from the figure, clearly embody the primary patterns of the positive class. The remaining clusters on the other hand visibly represent non-standard representations of faces. Clusters b) and d) depict unrepresentative patterns through the presence of varying degrees of articulation and rotations. These images can be said to describe the sub-concepts of the positive class, while a number of them can also be classed as noisy samples due to strong illumination variances that obscure key features.

Therefore, the insights gained from more detailed analysis is that the PSL-like training approaches employ a type of informed clustering within each layer that is guided by information learned in previous nodes. Since the node sizes are kept to relatively small sizes, only the most representative patterns are learned initially. The effect of this is that in subsequent nodes, these types of patterns are filtered out as being redundant. This isolates class sub-concepts as well as outliers. The clustering mechanism therefore delays training the most unrepresentative positive samples until the final nodes in a layer. The issue then is that the sub-concepts and the outliers become dominant in the training

sets and consequently their patterns are overemphasized in the learning. If the test data does not contain large numbers of samples which contain sub-concept patterns, then the generalization is likely to deteriorate as witnessed by results here. Also, if the training dataset possesses a significant amount of noisy data, then the learning process risks placing undue learning emphasis on them.

Ultimately, the results demonstrate that the generalization ability of PSL-like classifiers is only as strong as the combined generalization strength of all its weakest nodes located in the latter sections within the layers. Consequently, modifications to the PSL-like approaches were required in order to mitigate the algorithm's susceptibility to the disproportionate influence that sub-concepts and outliers have on the learning process in these nodes.

In summary, the findings show that:

1. PSL-like learning results in class-imbalanced learning.

2. Sub-concept patterns and noisy samples are isolated and concentrated in the final nodes of each layer.

3. PSL-like algorithms tend to overemphasize unrepresentative patterns in the presence of significant in-class variability and noisy samples by selecting poor features.

## 5.4 Methods for Improving the Accuracy

The problem of learning on imbalanced datasets and its associated challenges have been topics of extensive research in recent years. A brief review is offered here due to its relevance. Typically, random sampling methods that create balanced distributions without generating synthetic samples have been used. Of these, random over-sampling proceeds by replicating selected samples of a target class, while under-sampling removes data belonging to larger classes in order to restore balance. Both strategies introduce their own sets of problems. Over-sampling may lead to over-fitting, while under-sampling may lead to a loss of information regarding the majority class [62].

Some more complex and successful strategies include the synthetic minority over-sampling technique (SMOTE) [11], which artificially generates additional samples of a minority class. Adaptive synthetic sampling methods like Borderline-SMOTE [60] and ADA-SYN [63] address the shortcomings of SMOTE, which tends to overgeneralize in its generation of new samples leading to an overlapping between classes. Meanwhile, cluster-based sampling methods have been found to be powerful due to the flexibility they provide in their ability to target specific problems.

Boosting has also been applied directly to the problem of imbalanced learning. AdaBoost.M2 has been combined with synthetic sampling methods like SMOTE to yield SMOTEBoost [20], while Guo and Viktor [57] showed how to combine AdaBoost.M1 with a data generation approach that was termed DataBoost-IM. In addition, cost-sensitive learning has also been combined with boosting which enables it to focus learning on minority classes through predefined cost-matrices.

While the various data generation approaches have been shown to be effective, they are also complex and computationally expensive. What is more, determining the costs

of different classes in cost-sensitive learning is especially challenging [62]. As such, these approaches were not considered as suitable solutions to the problems of PSL. They are also not applicable since the nature of the in-class variability of the datasets changes from one node to the next, whereby the more representative samples are systematically removed and the minority class sub-concepts are thus isolated eventually becoming the majority. The challenge facing the current version of PSL-like algorithms is that though their goal is to learn the minority sub-concepts, a strategy is required that promotes this in a way that does not give an inordinate amount of emphasis to patterns from these samples.

The task is also complicated by the nature of the problem domain and the types of features being used. Haar-like features tend to produce up to 200,000 features per sample which is much larger than the number of training samples. As mentioned earlier, this poses a problem since it has been well established that high dimensional data tends to produce over-fitted classifiers in the presence of disproportionately smaller training sets [33, 34, 169]. With the reduction of training samples taking place per node, and the concentration of unrepresentative instances taking place in the trailing nodes, the challenge of high dimensionality of the feature sets becomes even greater.

In domains such as these, it is common to apply some form of dimensionality reduction in order to lessen the risk of producing over-fitted classifiers. This is generally achieved by either using a feature selection or a feature extraction approach [33]. More commonly feature selection is employed which seeks to identify a subset of features from the total feature space with the goal of minimizing the redundancy and irrelevance of the final feature subset. This is often achieved by applying predefined criteria based upon class separability or classification performance [22]. Meanwhile, feature extraction utilizes all the available features and projects them into fewer dimensions which has the effect of reducing the comprehensibility of the features themselves.

Dimensionality reduction certainly is a viable option for addressing the problems here. A simple criteria which only makes features available to the trailing nodes that have been used by the initial nodes found at each layer, as well as a selection of previously top ranking features holds considerable potential. However, though such a strategy holds much promise in improving the generalizability, a degree of comparative fairness regarding the training runtimes with algorithms that do not employ feature selection would be compromised. Since this is an important component of the study, the pursuit of this strategy is left for future work.

Numerous research has also found AdaBoost itself to be susceptible to outliers. Variations of AdaBoost have been proposed with the goal of producing algorithms that are more robust to noise. Gentle AdaBoost [46] and Local boosting [196] are two proposed variants. However, in PSL-based training the boosting component in theory should not require modifications, since the boosting rounds are few and therefore the generated rules do not become too specific. This means that even in the presence of outliers, provided there is a sufficiently large number of quality samples, a significant negative effect on the generalizability of the nodes need not be expected.

Two intuitive and straightforward strategies for improving the accuracy of PSL-like classifiers were pursued. The first involved pruning the under-performing nodes (ensemble thinning) from the cascaded classifier. The second consisted of a strategic re-sampling approach to balance out the unrepresentative patterns.

### Ensemble Thinning

In Section 2.1.2, the concept of *thinning* an ensemble-classifier in order to produce a subset which performs better than the original ensemble was reviewed. Ranking-based methods for thinning an ensemble were described as operating on an individual classifier level in order to rank each one against a validation set. Thereafter, the worst performing classifiers are deleted. Search-based thinning strategies tend to consider the collective accuracies of different combinations of subsets of classifiers, while cluster-based thinning methods group classifiers together through an additional phase prior to the thinning procedure.

The initial strategy of thinning was intuitive in that it only involved recognizing that the PSL itself performs a ranking-based procedure of classifiers within each layer and orders them in clusters of PSL nodes. The next step involved introducing a thinning parameter $\gamma$, to define how many nodes required pruning from each layer. The node-thinning strategy was naive in that it did not explicitly evaluate the nodes for accuracy but instead removed $\gamma$ nodes from each layer regardless of their generalizability or the total number of existing nodes within a layer.

The thinning strategy was evaluated on the classifiers from the previous section with two different values of $\gamma$, where $\gamma$ was set to 1 and 2. Figure 5.10 shows the results of the thinning procedure on the 15000 positive sample dataset where $\gamma = 1$. The results indicate that the thinning-strategy improved the accuracy of the original classifier in nearly every instance. In the majority of cases, the ROC curves of the pruned classifiers shifted towards the ideal top-left corner position of the graphs. In some instances the improvement in the accuracy was up to five percentage points.

The results of thinning for $\gamma = 2$ did not improve the accuracy of the classifiers. The thinning strategy proved to be too naive when $\gamma$ was greater than 1, since the earlier layers of a cascade tended to consist of only a few nodes, for which this method was too aggressive. Overall, the results indicated that there is scope for further refinement of the thinning strategy. More sophisticated strategies can be devised which more intelligently prune layers by taking into consideration the total number of nodes within a layer, or by beginning the pruning process onwards from a selected layer in a cascade. Additionally, the thinning method can be combined with strategies which prune the nodes based on their performances on validation sets. The advantage of the current approach is that it requires virtually no post-processing overheads.

Moreover, the byproduct of the clustering process, in the form of nodes which takes place within PSL-like methods, is that it can be used as a procedure for identifying samples that are outliers or noisy. These samples can then be identified during the learning phase and removed from training subsequent layers. An additional parameter can easily be introduced to the cascade learning process which permanently removes and cleans the dataset of such samples. For instance, this parameter can represent the minimal proportion of unlearned samples within a base set. Once the percentage of unlearned samples falls below a predefined value, then these samples can be identified as noisy or representing irrelevant class sub-concepts and deleted from training.

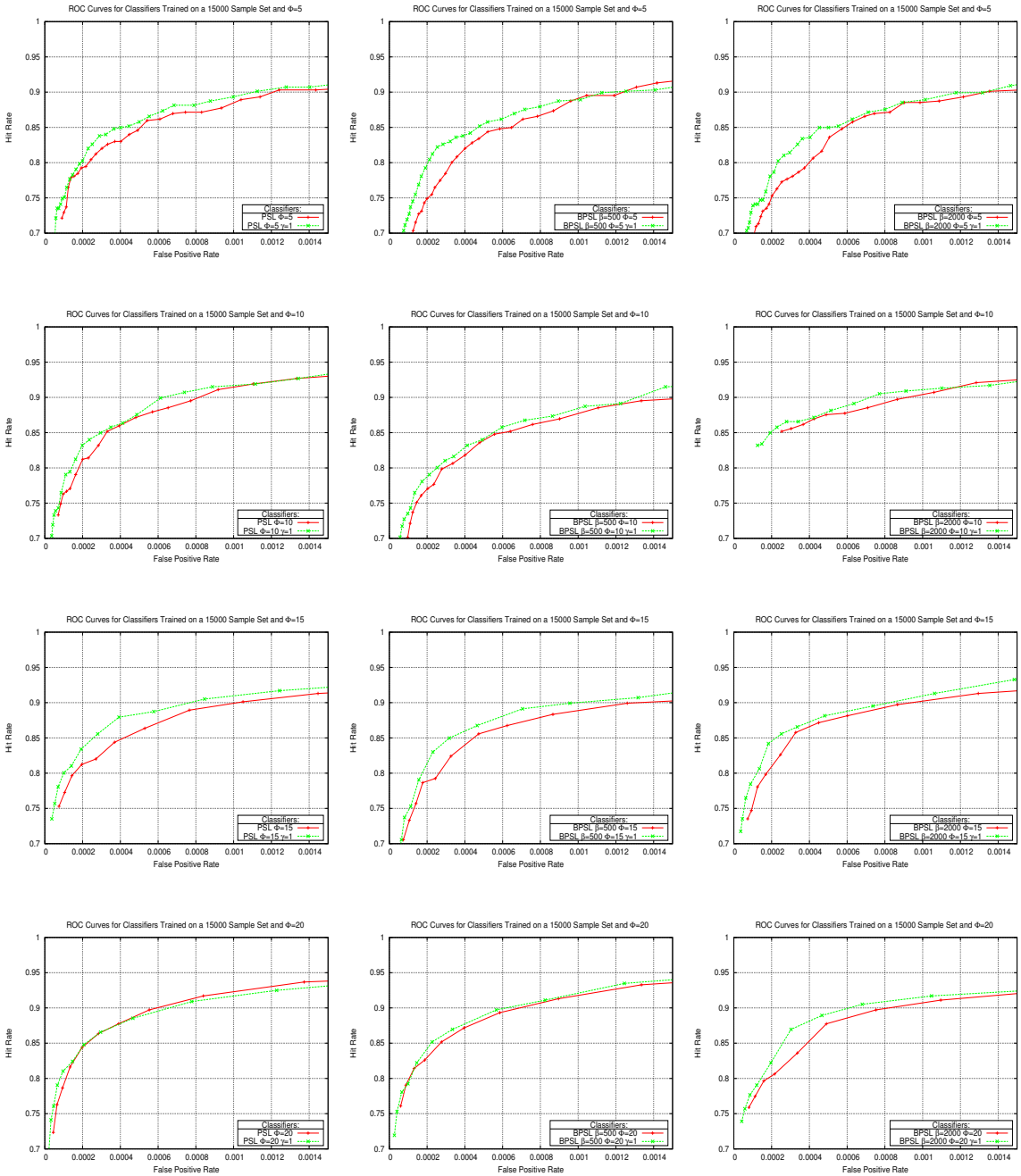



Figure 5.10: ROC curves for PSL and BPSL classifiers showing the results of pruning where the pruning parameter $\gamma = 1$.

### Informed Re-sampling

While the first strategy involved a simple post-processing step, the second strategy consisted of modifying the training sets on which the trailing nodes are exposed to, with the goal of balancing out the unrepresentative patterns of the target class with more general patterns.

The alternative strategy performed random re-sampling of positive samples without replacement from the set of positive samples which have already been correctly learned by preceding nodes. As such, the re-sampling is *informed* and is only carried out when the size of the base sets falls bellow the specified size $\beta$. The base sets are augmented by $n$ number of samples which represent the difference between the specified set size $\beta$ and the current number of unlearned positive samples. The re-sampled positives are referred to as *secondary positives*, while the remaining misclassified samples are termed *primary positives*.

Once a node is trained, the secondary samples that are classified as false negatives remain within the training set for the next node. The secondary samples that have been predicted as true positives are replaced with new true positive samples as classified by the prior nodes. This strategy helps to steer the learning process away from creating overly specific rules based on the minority samples in the trailing nodes.

A further strategy was required in order to ensure that the learning focused on the primary samples rather than on re-learning patterns from the secondary positives. This was needed because the skewness between the primary and secondary positives becomes more acute for each successive node with the consequence that the primary samples are not learned, resulting in over-inflated layers with large numbers of nodes. Due to this, two additional mechanisms were used. The first increased the weights of the primary positives and the second relaxed the $\Phi$ size requirement for a fixed number of weak classifiers per node.

Given a 50-50 split of the total available weights between the positive and negative samples, the strategy for increasing the weights of the primary samples involved a function that allocated more weights to the primary samples as their relative proportion in respect to the secondaries increased. The modification of sample weights was designed to begin once the proportion of primaries to secondaries reached a predefined threshold $t$. With $t = 0.5$, the weight re-allocation function was defined as:

$$w_j = t + \frac{t - p_j}{2}, \tag{5.1}$$

where $w_i$ is the proportion of total positive weights allocated to the primary samples for node $j$ and $p_j$ is the proportion of primary samples that comprise the positive base set for node $j$ when $p_j < t$. Once $p_j$ falls beyond the predefined threshold, the total proportion of the re-allocated weights to the primary samples ranges from 0.5 to 0.75.

The $\Phi$ node size parameter was also relaxed in respect to the increasing skewness of the primary and secondary positives. The number of weak classifiers per node $j$ was increased by $1 - p_j$ as soon as the number of primary positives in a given node fell below $\beta$. Therefore, as the primaries became scarcer, the size of the node size $\Phi$ increased up to the maximum of twice its original size. This procedure was also necessary in order to limit the total number of nodes being generated by each layer. Algorithm 8 summarizes

the entire procedure.

**Algorithm 8**: **BPSL with Re-sampling (BPSL.r) procedure for a given layer $i$.**

**Given** : $P_{i,j}$ denotes the positive dataset of size $\beta$, used to train node $j$ in layer $i$ where $P_{i,j} \subset \mathbb{P}$. $\Phi$ denotes the maximum number of weak classifiers per node.

$P_{i,1} = \text{RandomSample}(\mathbb{P}, \beta)$
$offset = 0$
**for** $j = 1$ **to** $n$ **do**
  **for** $k = 1$ **to** $\Phi + offset_j$ **do**
    $h_{i,j,k} = \text{AdaBoost}(P_{i,j}, N_i)$
  $P_{i,j+1} = (x_m, y_m), \forall x \in P_{i,j}$, where $h_{i,j}(x_m) = -1 \wedge y_m = 1$
  $s^P = \text{SizeOf}(P_{i,j+1})$
  **for** $k = 1$ **to** $\beta - s^P$ **do**
    $P_{i,j+1} = \text{RandomSample}(\mathbb{P}, 1)$, where $(h_i(x_m) = -1) \wedge (y_m = 1)$
    $s^P = s^P + 1$
  **if** $P_{i,j+1} = \emptyset$ **then**
    *break*
  **if** $s^P < \beta$ **then**
    **for** $k = 1$ **to** $\beta - s^P$ **do**
      $P_{i,j+1} = \text{RandomSample}(\mathbb{P}, 1)$, where $(h_i(x_m) = 1) \wedge (y_m = 1)$
    $offset_j = \Phi \left(1 - \frac{s^P}{\beta}\right)$
  $w_j = t + \frac{1}{2}\left(\frac{s^P}{\beta}\right)$
  **foreach** $(x_m, y_m)$ **in** $P_{i,j+1}$ **do**
    **if** isPrimaryPositive$(x_m, y_m)$ **then**
      $(x_m, y_m) = \frac{w_j}{s^P}$
    **else**
      $(x_m, y_m) = \frac{1 - w_j}{\beta - s^P}$

The experiments consisted of re-training all the BPSL classifiers described in Section 5.3.1 under the same conditions. BPSL.r refers to the new BPSL classifiers that have been trained with the re-sampling component. In order to establish the degree of meaningfulness of the trailing nodes, the BPSL.r component was also combined with the previous strategy of pruning where $\gamma = 1$.

The following sequences of graphs plot the ROC curves of PSL, BPSL, BPSL (with ensemble thinning), BPSL.r and BPSL.r (combined with thinning). The first sequence of ROC curves is shown in Figure 5.11 where the size of the base set parameter $\beta = 2000$ and $\Phi = 10$. The first graph plots a BPSL and a BPSL.r classifier which have been trained 10 times in order to provide statistical data in respect to variations that can be expected in the subsequent ROC curves for both types of classifiers. Once again, the standard

deviations for the hit and false positive rates are the smallest for the most critical regions of the ROC curves.

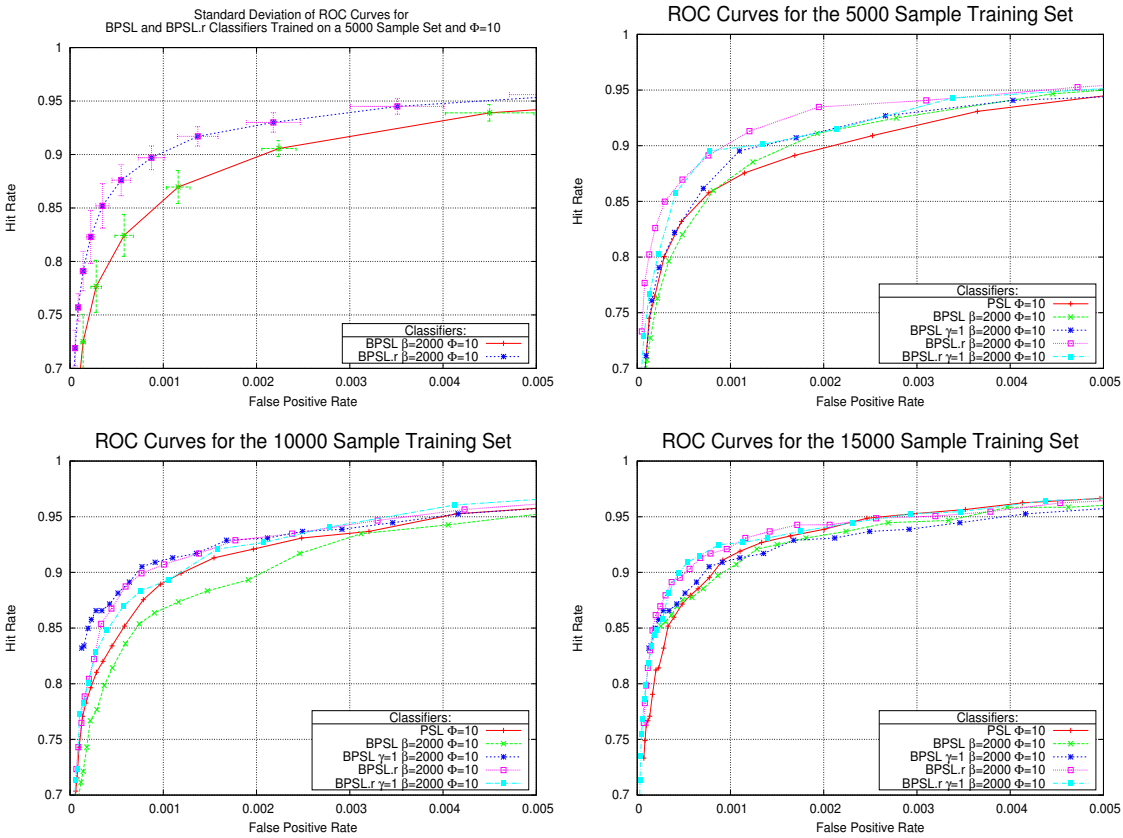



Figure 5.11: ROC curves for BPSL.r, BPSL and PSL classifiers. The top left graph indicates the mean and the standard deviations of two selected classifiers.

The remainder of the graphs indicate that on each of the datasets, a noticeable improvement in accuracy has been realized by the proposed strategies over the original PSL and BPSL classifiers. BPSL.r classifiers were consistently amongst the best performing across the three datasets. Thinning strategies combined with both BPSL and BPSL.r also yielded improved generalizability over the original implementations on all datasets.

The next sequence of graphs in Figure 5.12 shows results from classifiers where $\beta$ was also set to 2000, while the node size was increased to $\Phi = 15$. In addition, VJ classifiers are plotted alongside the other classifiers in order to demonstrate the degree to which the difference in accuracy has been improved by the proposed strategies. On the 5000 sample dataset, a modest improvement by the BPSL.r and BPSL with thinning, can be seen over BPSL. Both of the former algorithms increased their generalizability on slightly different sections of the ROC curves. The BPSL.r classifier attained strong accuracy on the 10000 sample dataset where its generalizability was comparable to that of VJ. BPSL.r combined with thinning produced marginally lower accuracy while BPSL combined with thinning once more improved upon the original BPSL and PSL. Similar patterns are repeated in the 15000 sample dataset where the BPSL.r was once again the best performing classifier.

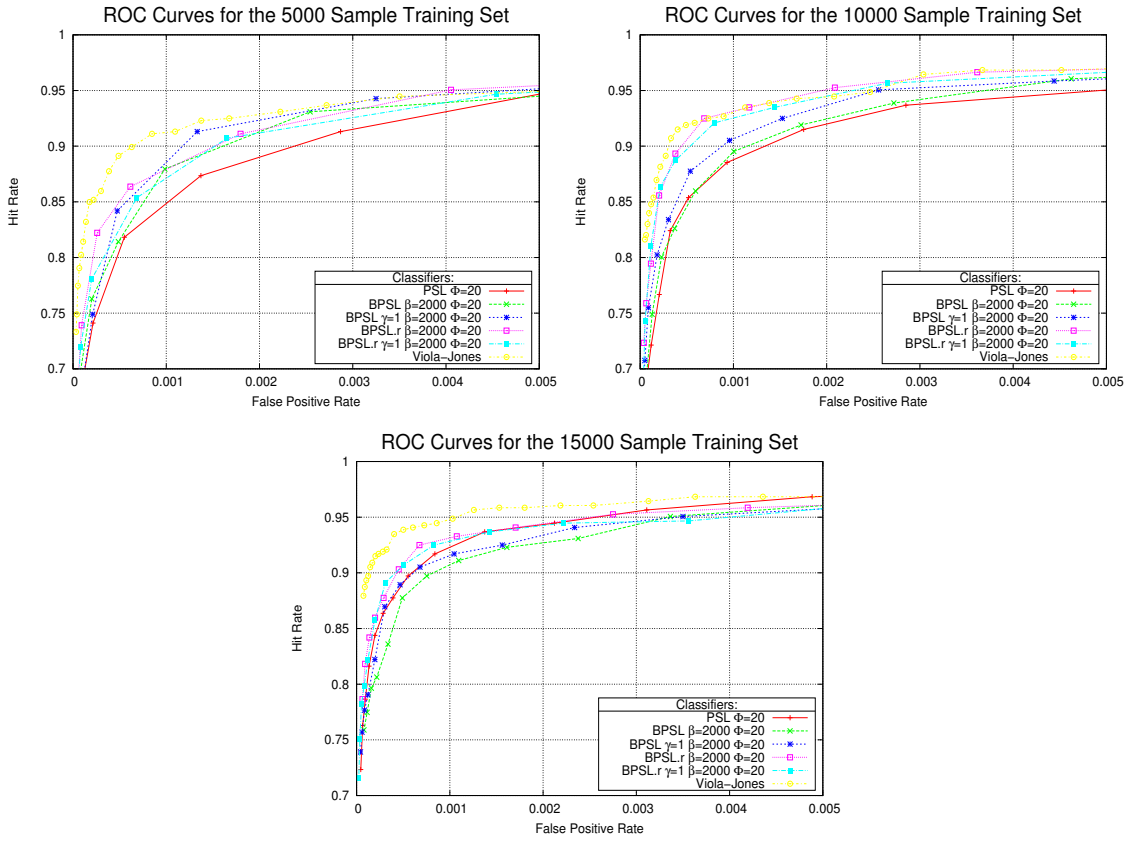



Figure 5.12: ROC curves for BPSL.r, BPSL and PSL classifiers where $\Phi = 20$.

This time PSL attained higher accuracy than BPSL with thinning; however, the thinning strategy increased the accuracy of its classifier once more over BPSL. The analysis of the accuracies per node throughout each layer for a selected BPSL.r classifier where $\Phi = 20$ and $\beta = 2000$, on a 15000 sample dataset can be seen in Appendix A.1.

The final sequence of ROC graphs in Figure 5.13 displays classifiers with smaller base sets where $\beta = 500$ and $\Phi = 15$. Overall, the degree of improvement on classifiers with smaller base sets where $\beta = 500$ was not as large as it was with $\beta = 2000$. On the 5000 sample dataset, the BPSL.r classifier clearly outperformed the remaining classifiers, while not lagging far behind VJ. The BPSL with thinning failed to improve the generalization of BPSL for the first time, while BPSL.r combined with pruning, also decreased the accuracy. BPSL with thinning, and BPSL.r both registered better accuracies on the 10000 sample datset over the BPSL and PSL classifiers while VJ edged slightly ahead. On the last dataset, the improvements of the proposed strategies were more muted. Though both the BPSL.r and BPSL improved between 1-3 percentage points over the naive implementations, the gap between the VJ and these classifiers still remained considerable.

Overall, however, the results can be summarized by concluding that the BPSL.r has consistently demonstrated and improved accuracy over the BPSL and PSL classifiers. The improvements were slightly higher for larger sizes of $\beta$. When combined with thinning, there was no indication of an improved generalizability of the classifiers. This testified to

the improved quality of the trailing nodes and their integral role in maintaining a high accuracy, which was previously not the case. In almost all cases, BPSL with thinning did improve the accuracy of the BPSL classifiers and occasionally even displayed slightly stronger generalization rates over the BPSL.r classifiers. However, the degree of their accuracy improvement was not as large or as consistent as that of the BPSL.r classifiers.

As expected, the trade-off associated with the increased generalizability brought about by the informed re-sampling strategy of the BPSL.r classifiers is an increase in the training runtimes. In Figure 5.14, the training runtimes for all algorithms are displayed. In order to highlight the trends, the training runtimes are listed as the averages over all four classifiers with different values of $\Phi$ for each algorithm. Overall, the fastest training runtimes are recorded by the BPSL ($\beta = 500$) classifiers and the runtimes increase slightly for these classifiers as the $\beta$ parameter is also increased. The training runtimes of BPSL.r ($\beta = 500$) showed that there was a modest increase in the cost of training classifiers using the re-sampling strategy, while this cost increased substantially as the $\beta$ parameter was also made larger. Despite the higher cost associated with the re-sampling facility, the BPSL.r continued to display faster training runtimes than PSL.

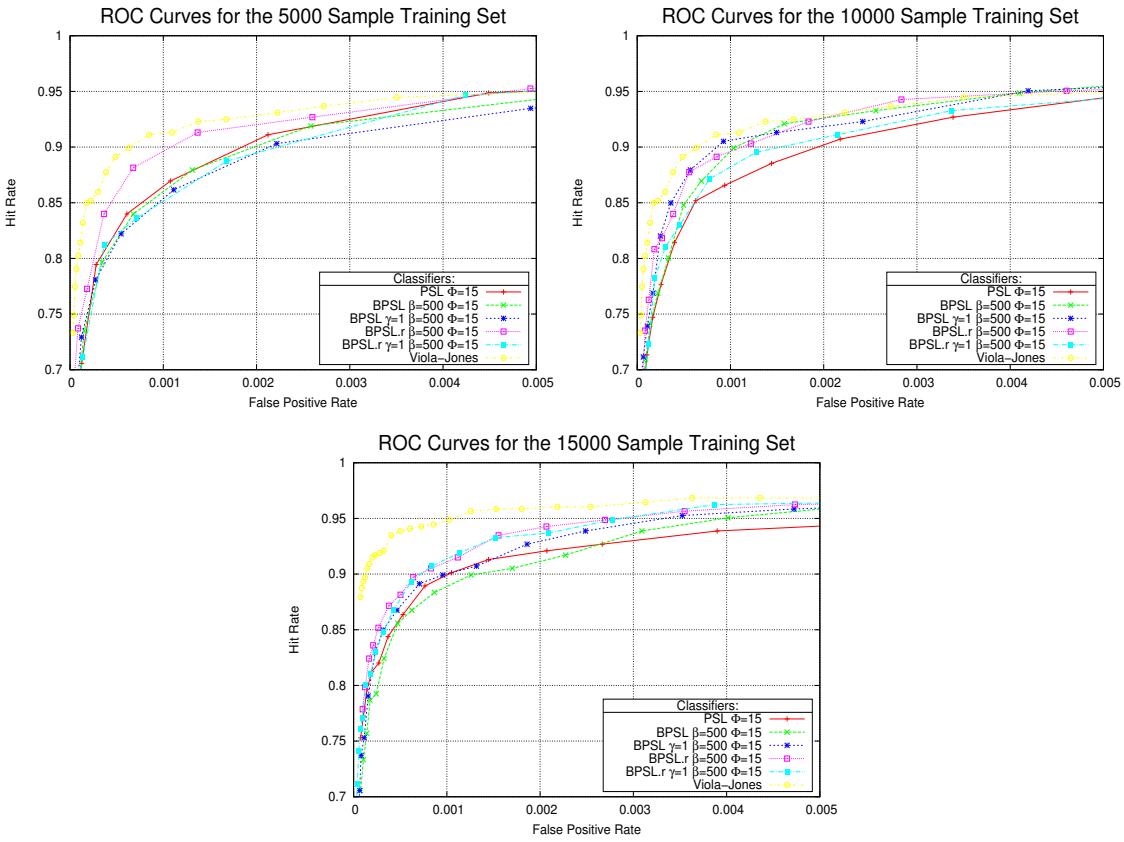



Figure 5.13: ROC curves for BPSL.r, BPSL and PSL classifiers where $\Phi = 15$.

Table 5.3 provides a more detailed comparison between the pairs of BPSL.r, BPSL and PSL algorithms, together with different parameter values for $\beta$. The figures show the extra training runtime required in percentages between the pairs of algorithms as the

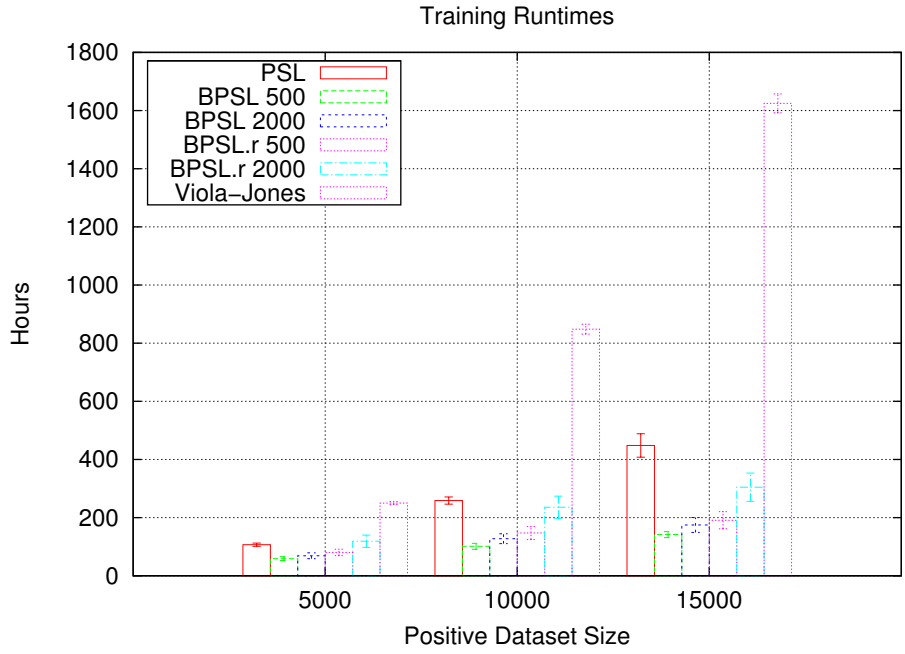



Figure 5.14: Averaged training runtimes for each of the BPSL.r, BPSL and PSL algorithms across all parameter values for $\Phi$ as well as those of the Viola-Jones classifiers.

training datasets increase. The figures once more represent averages over all node sizes of $\Phi$ and are listed with their standard deviations.

Beginning with the comparison of BPSL.r and BPSL where $\beta = 500$, the training runtimes increase substantially for the BPSL.r as the dataset size becomes larger. Comparing the same algorithms with $\beta = 2000$, the results show that though there is a significant increase in the training runtimes of the BPSL.r algorithm over the naive implementation, the cost of training does appear to increase at a substantially with the dataset sizes. In comparing the PSL with the BPSL.r algorithm, the extra computational requirement clearly increases for the PSL with the larger datasets for both values of $\beta$. However, on the smallest datasets tested here, the PSL did register slightly faster training runtimes on average than the BPSL.r with $\beta = 2000$. Lastly, the BPSL.r algorithm is compared with itself on two different values of $\beta$. On $\beta = 500$, the training runtimes are clearly faster on smaller datasets, but as the datasets become larger, the additional training cost associated with using a larger value of $\beta$ on this algorithm, decreases substantially. On the largest datasets, there is very little to differentiate the training runtimes on both parameter settings.

The way in which this translated to the total number of weak classifiers generated by each algorithm can be seen in Table 5.4. For all PSL-like classifiers, the totals have been averaged over each of the four classifiers for every value of $\Phi$, with the standard deviation reported. The PSL and BPSL algorithms generated the most concise classifiers across all datasets, with BPSL ($\beta$ =2000) producing the simplest ensembles. The informed resampling strategy of BPSL.r indicated that it generates ensembles that are approximately 30% larger than the PSL and BPSL classifiers which is a tradde-off for a notable im-

Table 5.3: The extra training runtime required expressed in percentage and the standard deviation between the BPSL.r, BPSL and PSL algorithms.

| Dataset | **BPSL.r ($\beta$ =500)** vs. **BPSL ($\beta$ =500)** | **BPSL.r ($\beta$ =2000)** vs. **BPSL ($\beta$ =2000)** | **PSL** vs. **BPSL.r ($\beta$ =500)** | **PSL** vs. **BPSL.r ($\beta$ =2000)** | **BPSL.r ($\beta$ =2000)** vs. **BPSL.r ($\beta$ =500)** |
|---|---|---|---|---|---|
| **5000** | 45% ±11.5% | 73% ±9.8% | 29% ±15.8% | -8% ±14.1% | 31% ±15.3% |
| **10000** | 82% ±30.0% | 85% ±19.4% | 46% ±30.2% | 12% ±15.1% | 23% ±19.2% |
| **15000** | 102% ±33.4% | 75% ±18.6% | 64% ±33.7% | 49% ±22.5% | 2% ±19.0% |

provement in the generalizability. The VJ classifiers, as expected, produced considerably larger ensembles than all the PSL-like algorithms; however, it generated a considerably less complex ensemble for the 15000 dataset than it did for the 10000 dataset which made the difference between it and the PSL-like algorithms not as extensive as expected.

Table 5.4: Total weak classifiers generated by each of the training structures. Sums expressed as averages over all four classifiers for each value of $\Phi$ with the standard deviation.

| **Dataset** | **PSL** | **BPSL ($\beta$ =500)** | **BPSL ($\beta$ =2000)** | **BPSL.r ($\beta$ =500)** | **BPSL.r ($\beta$ =2000)** | **Viola-Jones** |
|---|---|---|---|---|---|---|
| 5000 | 610 ±71 | 641 ±59 | 604 ±53 | 786 ±99 | 757 ±39 | 1319 - |
| 10000 | 1130 ±173 | 1182 ±140 | 1118 ±153 | 1527 ±154 | 1401 ±180 | 2955 - |
| 15000 | 1583 ±253 | 1703 ±184 | 1459 ±179 | 2075 ±274 | 1959 ±155 | 2217 - |

## 5.5 Discussion

The BPSL approach to training weak classifiers bears some resemblance to bagging in that different partitions of data are randomly selected for training classifiers. However, the resemblance ends there since BPSL fundamentally differs to the bagging approach in that the sampling of new samples is informed by previous classifiers' performances and correctly learned samples are eventually removed from learning[3].

There are clear advantages in using BPSL over PSL. The training runtimes are substantially reduced while there is no evidence that the generalizability of the classifiers is reduced. However, the advantage brought by the efficient runtimes of PSL-based methods is likely to be outweighed by overfitting in cases where the training sets are highly complex in the sense that they contain considerable in-class variability and noisy data.

Given highly complex datasets, PSL training requires that additional strategies be employed in order to prevent the rules becoming too specific. The strategies developed here present the practitioner with a trade-off. When training with strict runtime constraints, then the ensemble thinning method is to be preferred since it has definitely been shown to improve the performance of the classifiers while requiring no additional post-processing overheads. Pruning a node from each layer is sufficient to realize significant improvements.

[3]It can also be said that the iterative removal of learned samples shares similarities with IREP [51] family of rule induction algorithms, while the informed re-sampling component used for preventing overfitting has functional parallels to the way the former algorithms employ pruning datasets.

More aggressive pruning should only take place on final layers of a cascade since they are comprised of larger numbers of nodes.

Under more flexible runtime constraints, the informed re-sampling strategy is to be preferred since it has consistently demonstrated the highest accuracies. This approach should be applied by using base sets that are ideally of the same size as those of the negative datasets. While the re-sampling strategy does generate the best accuracy, the resulting classifiers should be validated by also performing ensemble thinning, since on occasion this can yield slightly improved results and slimmer ensembles without incurring any computational overheads.

An alternative strategy to preventing overfitting taking place on PSL-based methods is to abandon the requirement of achieving 100% hit rates for each layer. By removing the positive samples which fail to be correctly trained from subsequent layers, this strategy can imitate the thinning strategy but it can in theory accomplish even more since such undesirable patterns will not have the opportunity to compromise other nodes in subsequent layers.

Lastly, the results indicate once more that there is no consistently discernible difference between the accuracies and runtime performances of PSL-based classifiers trained on different node size parameters $\Phi$. The recommendation is that when training classifiers using PSL approaches, several classifiers with varying values for $\Phi$ should be created and thereafter the best performing classifiers should be selected.

## 5.6 Summary

The research here conducted experiments on the PSL algorithm with larger datasets. A PSL-based algorithm was also developed for bootstrapping positive samples in a similar way to that which exists for negative datasets. This strategy holds potential for efficient cascade training on massive positive datasets. The algorithm uses a method of informed under-sampling which enables it to learn all relevant patterns of a target concept without explicitly training on every sample in a positive dataset.

The problem of overfitting was confirmed to be occurring on PSL-based algorithms when training on datasets with high degrees of in-class variation and large numbers of noisy samples. Two successful strategies were devised to counter the effects of overfitting. One involved ensemble thinning, while the second employed informed over-sampling.

The summary of the primary findings of this chapter are:

1. The accuracy of PSL in relation to conventional cascading algorithms does not increase sufficiently by just increasing the size of the positive training datasets.

2. The positive sample boostrapping capability of BPSL considerably reduces the training runtimes of PSL, while maintaining similar generalization patterns.

3. PSL-based algorithms encourage imbalanced-learning and perform implicit informed-clustering of the positive dataset whereby the learning of the most complex samples is delayed for the final nodes. The datasets used for the final nodes consist of high concentrations of samples that either represent class sub-concepts or are noisy. When a training set possesses a high number of such samples, then overfitting is likely to occur.

4. Applying ensemble thinning in the form of removing a specified number of trailing nodes improves the overall accuracy.

5. The strategy of re-sampling from redundant positive samples (BPSL.r) and augmenting the base positive sets mitigates the overfitting.

6. BPSL.r generalizes better than BPSL and PSL. There are times when combining both BPSL.r and ensemble thinning produces superior accuracy. The improved generalization comes at the expense of increased training runtimes. As the parameter $\beta$ for determining the positive base set size increases, the accuracy increases as does the training runtime cost.

# Chapter 6

# Enabling Adaptive Learning on Rare-event Detection Domains

The underlying assumption of classical approaches for training classifiers has been that their operating domains are stationary [107, 177]. However, with the advent of streaming data and long life classification systems, it has become clear that these assumptions no longer hold. Consequently, it has become apparent that traditional techniques for application are inadequate on emerging domains [99, 100].

The data streams in dynamic domains such as e-commerce, economic and financial data analysis, sensor systems, email spam filtering and a host of other burgeoning fields, possess distributions and class descriptors that constantly change with time, due to some *hidden context* [181]. Therefore, the *target concepts* from the original training data may no longer correspond with the current representations of the target concept. As a result of an increasing loss of relevance between the concepts' representations and current snapshots of data, a deterioration in the runtime accuracies ensues. This phenomenon is referred to as *concept drift* [145].

Variations in data can take a sudden or a gradual form and may also be recurring, whereby previously active concepts reappear in a randomly cyclical fashion. Although an increased understanding about the anticipated types of drifts improves the chances of devising a successful adaptive strategy [80], often the magnitude and frequency of drifting contexts is not known *a priori*. To a large degree, training effective classifiers is contingent on formulating strategies based on foreseen types of changes inherent to an operating domain. Due to this, training accurate classifiers for unpredictable non-stationary environments is non-trivial [109, 166] and is still an open problem.

The situation is further compounded by the fact that many target concepts in streaming data are also rare-occurring [177]. Biased-class distribution problems of this kind can hinder and impose significant limitations on accuracies attainable by standard learning methods [71] and are a considerable ongoing challenge [177]. This situation is encountered in a large and growing list of domains such as adversarial monitoring environments like network intrusion, surveillance and transaction fraud detection.

In addition to the trends that have given rise to expanding numbers of available data with seemingly unlimited and continuously flowing streams, there has also been a tendency for these streams to be dense and high-speed [200]. This poses difficulties for not only realizing real-time event detection, but even more, so for modelling and handling concept-

drifts in a timely manner. Failure to expeditiously update models to time-evolving environments will eventually result in a decrease in prediction accuracy [177]. For instance, this can be critical in robotics where there is a reliance on computer vision components. The incoming data is high-speed and large scale, with drifting concepts caused by different lighting conditions. A problem domain of this kind requires real-time processing and adaptation in order for time-sensitive tasks such as obstacle avoidance and planning to be effective [123].

### 6.0.1 Motivation

The motivation for this chapter is to propose a drift handling algorithm that is applicable to PSL-like classifiers. The challenge was to devise a method with minimal computational overheads which can adapt in real-time to changing concepts in the presence of large volume and high-speed streaming data. This is not attainable using most existing approaches that continuously revise and re-train new classifiers. The aim was to devise a model that makes no assumptions about the types of drift in its operating domain and can therefore be equally robust to both gradual and abrupt concept drifts, while also addressing the issue of recurring contexts. The research was also motivated by the challenges presented by rare-class distributions in classification. The goal was to address this difficulty by exploiting the classifier structuring process of the attentional cascade [172] together with the challenge of meeting real-time execution requirements. The questions that this chapter sought to address were:

1. Can effective adaptability of PSL-like classifiers be realized using methods which do not explicitly re-train existing classifiers within an ensemble?

2. How responsive will the proposed algorithm be to a variety of drifts with different magnitudes and unknown cycles?

3. Is the adaptability realizable within real-time constraints?

Artificial datasets like STAGGER [144] and SEA [149] concepts have become common benchmarks for measuring the effectiveness of proposed algorithms for handling concept drift. However, the relevance of classifier performances on these datasets in respect to real-world problems has not been established [122]. There currently exists a lack of robust publicly available real-world datasets for benchmarking concept drift-handling algorithms due to limited degrees of actual drift in them [166]. As a result various concept drift types have had to be artificially introduced to these datasets. Bifet et al. [8] also note that popular benchmark datasets are insufficient for testing scalability of algorithms for massive data streams.

These issues served as a motivation to create a real-world dataset containing natural concept drifts. A decision was made to continue with the problem of face detection and to explore methods of realizing classifier adaptability in this domain. By implementing exhaustive raster scanning of a given image frame using sub-windows of increasing sizes, it was possible to create a suitable problem domain which provided a large, high-speed data stream, where accuracy and real-time drift response is paramount. In addition, the problem domain satisfied the requirement of consisting of a biased-class distribution as

well as possessing naturally occurring concept drifts of varying magnitude which could be easily captured.

### 6.0.2 Past Research

Ensemble-based techniques have been shown to be an effective and scalable approach to addressing concept drifting challenges in streaming data [147, 149, 176, 200]. The modularity they afford gives them the ability to retain relevant historical information as well as to effectively incorporate new knowledge [36].

In order to maintain classifier relevance to current snapshots of streaming data, ensemble systems generally employ either the *constant-update* or *detect-and-retrain* methods for revising their constituent experts [130]. These can be seen as evolving and event driven respectively. Most ensemble systems belong to the former category. These methods are not concerned with detecting or quantifying the magnitude of drift that might be occurring. Instead, these approaches assume that drift is always taking place in the environment and therefore, they continuously adapt classifiers to the current environment. This continuous-update strategy comes with the loss of responsiveness to sudden drifts [130] and the increased risk of modelling noise. Wang et al. [177] point out the high model update-cost associated with this approach since all valid classifiers need to be consulted and new classifiers learned on data that may not even contain concept drifts. Hence, this strategy may cause severe limitations on the systems ability to operate in high-speed data streams.

Alternatively, the *detect-and-retrain* class of strategies actively monitor changes in the underlying distribution of data and trigger a classifier update once a predetermined criteria or performance threshold are surpassed. Under these approaches, the environment is assumed to be mostly stationary [36]. As a result, they are more responsive to sudden drifts [130] and they may experience varying degrees of accuracy deterioration under gradual drifts before the classifier is updated.

Ensemble-based systems maintain an updated state in respect to the non-stationary environments using a combination of strategies. These typically involve dynamic modifications of classifier weights and voting techniques, mechanisms that add new competent classifiers and remove the irrelevant, as well as approaches based on batch (offline) and instance (online) learning [36, 80].

Batch learning approaches train on entire partitions of incoming streaming data instead of single data instances. Recent approaches [36, 73, 107, 176], train new classifiers for every newly generated batch of data, altering both the structure of an ensemble as well as the weights of existing classifiers in order to maintain a corresponding state to the current environment. Wang et al. [177] recognize the high update costs associated with continuous global updates of a model in massive data streams and devise a framework for rapidly revising only relevant components of an ensemble and training new classifiers when required. Scholz and Klinkenberg [147] similarly address the problem of high density data streams and propose an event driven approach that trains and appends a new classifier to an ensemble only once drift is deemed to have occurred. All the above approaches employ various strategies for concept forgetting, involving either some form of ensemble pruning or a reduction in classifier competence weights.

Nishida et al. [109] employ a hybrid approach that combines both batch and online

learning. Their batch-trained classifiers preserve previously learned knowledge and thus provide robustness to recurring contexts, while online updated classifiers maintain a higher relevance to current snapshots of data by continuously adapting to each incoming instance. Kolter and Maloof [75, 76] propose pure online learning approaches that train and incorporate new experts into the ensemble when the ensemble makes a collective mistake. Each time an individual expert makes a mistake, their competence value decreases. An expert is deleted from the ensemble once its competence value falls beneath a predetermined threshold.

Pockock's [118] claim that Oza's Online Boosting [111] algorithm is suboptimal for drifting concepts, since it assumes that streaming data are independent and identically distributed, motivated them to make modifications to it, that they empirically show, yields some accuracy advantages. Their extensions still enabled them to preserve advantages of the original algorithm that included fixed memory usage associated with maintaining a preset number of experts in an ensemble and the incremental nature of the algorithm.

## 6.1 Implementation Details

There are three components that constitute the proposed drift handling algorithm. The first involves training a static PSL-like classifier. The second component takes the static classifier and a dataset (training or validation dataset) as input in order to assign to its various nodes competence values. This is described in the next section. The third component adapts the classifier to streaming data by parameterizing the cascade layers based on the competence values learned in the previous phase and is detailed in the following section.

### 6.1.1 Assigning Competence Values to the Static PSL-classifier

Experiments on enabling bootstrapping to PSL in Chapter 5 demonstrated that the nodes consisted of very diverse classifiers as attested to by the varied accuracies between the nodes in each layer. Though some nodes were shown to overfit data under certain conditions, it became apparent that since each node was trained on mostly non-overlapping positive datasets, each one came to specialize at predicting distinct portions of the distribution of the target concept.

Intuitively, the next step involved recognizing that each node, made up of a cluster of weak classifiers, could be seen as an individual ensemble classifier itself. Theoretically, the competence level of each node could then be determined based on its individual performance on any combination of the training or validation datasets as well as streaming data containing drifting concepts. Using this strategy, the intention was to leverage the confidence votes of each node in their contribution to making a final detection as concept drifts takes place and affects different class descriptors.

This strategy was pursued and in the process of assigning confidence values to the nodes, the nested cascade within each layer was transformed into an ensemble of node classifiers. The outcome was that the previous vote-combination strategy of one-node-one-vote had to be revised in order to accommodate weighted voting based on the predictive competence of each node. The cascade architecture of a single layer is shown in Figure 6.1. The evaluation of a node becomes no longer conditional on the outcome of the predictions

**Algorithm 9**: ***Concept Drift Learning***

**Given**: $T_i$ = threshold for layer $i$, $\alpha_{ij}$ = confidence value for node $j$ on layer $i$, $\left|TPR\right|$ and $\left|FPR\right|$ true and false positive rate matrices for all nodes, $\left|vote\right|$ = matrix of sums of all $\alpha_{ij}$ values for each layer representing possible thresholds, $L_{ij}$ = loss value for a node $j$ on layer $i$, $thresh_i$ = threshold value for layer $i$, $\omega$ = false positives weight adjustment, $\upsilon$ = true positives weight adjustment, $E_k$ = error metric for a given threshold value $k$.

**Input** : PSL-like classifier $h$, $SD_n$ = static dataset vector $\{(x_1, y_1), ...(x_n, y_n)\}$ where each sample $x$ corresponds to a class label $y \in \{-1, 1\}$, $D_n^t = t^{th}$ snapshot of non-stationary dataset containing $n$ samples,

**Line**

3 $H$ = TrainPSLClassifier(SD$_n$)

4 $\left|TPR\right|$, $\left|FPR\right|$ = EvaluateClassifier(SD$_n$, h)

6 **foreach** $i^{th}$ cascade layer **do**

7 **foreach** $j^{th}$ *node* $H_i$ **do**

9 $$L_{ij} = \left(\frac{FPR_{ij}}{2}\right) + \left(\frac{1 - TPR_{ij}}{2}\right)$$

10 $$H_{ij}\alpha_{ij} = \frac{1}{2} ln \left(\frac{1 - L_{ij}}{L_{ij}}\right)$$

12 **foreach** $t^{th}$ snapshot of dataset stream $D_n^{1,..,T}$ **do**

14 **if** $H_i(D_n^t) >$ *acceptable error* **then**

16 lower begining cascade layer $i$

18 **if** $H_i(D_n^t) \leq$ *acceptable error* **then goto** step 12

20 **foreach** $x$ sample in dataset stream $D_{1,..,n}^t$ **do**

21 **foreach** $i^{th}$ cascade layer **do**

22 $vote_{i,k} = H_i(x)$

23 update statistics $FPR_i, TPR_i$ based on each value $vote_i$

24 **foreach** $i^{th}$ cascade layer **do**

25 **foreach** $k^{th}$ possible threshold value in $vote_i$ **do**

27 $$E_k = \left(\frac{FPR_{i,k} \times \omega}{2}\right) + \left(\frac{(1 - TPR_{i,k}) \times \upsilon}{2}\right)$$

29 **if** $E_k < E_{min}$ **then**

30 $T_i = vote_{i,k}$

31 $E_{min} = E_k$

of the prior nodes and thus every node within each layer is guaranteed to be executed. The combination of the weighted confidence voting replaces the veto and the unanimous voting of the previous scheme.

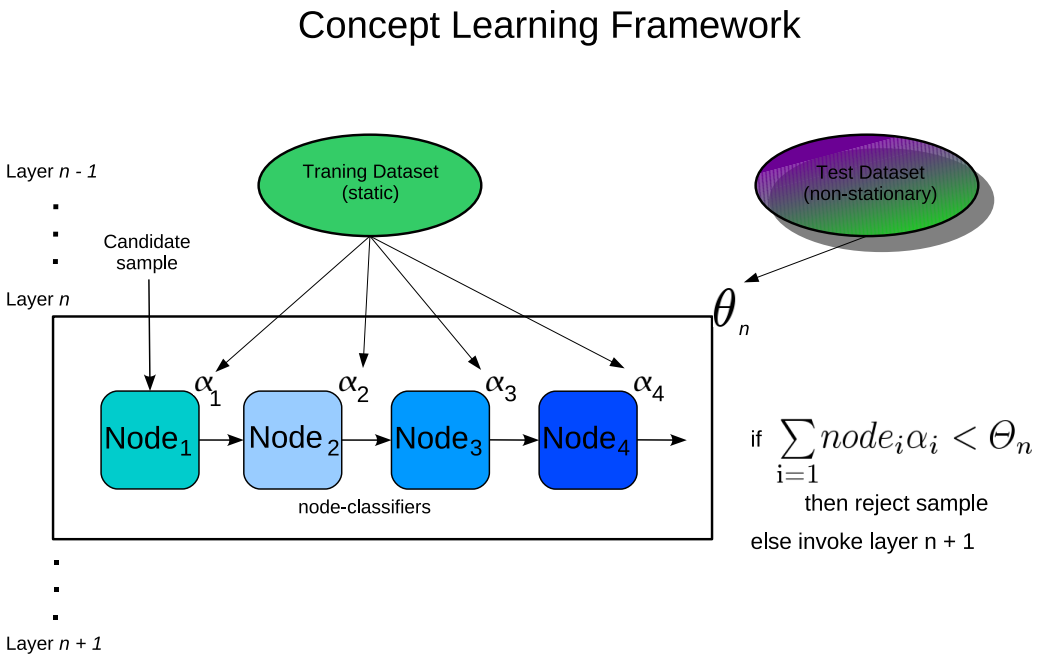



Figure 6.1: Diagram of the concept-drift learning algorithm.

Algorithm 9 outlines in detail the entire concept drift learning method. Once an initial PSL-like classifier has been trained on a static dataset (Step 3), it is first validated against it to gather competence values for all the nodes (Step 6). Due to the large class distribution skews that discriminate against the performances on the positive detections, instead of using the overall error rate, a performance metric was devised that was in effect the average between the false positive rate and the false negative rate in respect to the target class (Step 9)[1]. This performance metric of each individual node-classifier was used to calculate the competence values in the form used by Yoav and Schapire [192] for determining the voting strength of individual weak classifiers

$$h_{ij}\alpha = 0.5\ ln\left(\frac{1 - E_{ij}}{E_{ij}}\right) \tag{6.1}$$

where each confidence value $\alpha$ is assigned to a $j^{th}$ node $h$ on an $i^{th}$ cascade layer, based on the performance metric $E$. Usually ensemble combination rules consist of a simple *weighted majority vote* as in

[1]The alternative could have also been the g-mean measure by using the products of the accuracies of the classifier in respect to both the negative and the positive class, combined with a weight adjustment.

$$H_i(x) = sign\left(\sum_{\mathrm{j}=1}^{n} \alpha_{ij} h_{ij}(x)\right) \tag{6.2}$$

where $H$ represents a prediction for a layer $i$ using the sum of $n$ number of ensembles applied to on an instance $x$ whose prediction is determined by the *sign*. This was modified by parameterizing each layer with a minimum confidence threshold value $\Theta_i$

$$H_i(x) = \begin{cases} 1 & \text{if } \sum_{\mathrm{j}=1}^{n} \alpha_{ij} h_{ij}(x) > \Theta_i \\ 0 & \text{otherwise} \end{cases} \tag{6.3}$$

in which a sample instance $x$ is positively predicted and passed to succeeding layers for more rigorous testing only if the sum of confidence values for the current layer surpass it, thus producing more robust collective decisions.

### 6.1.2 Concept-Drift Learning Algorithm

In contrast to the node-classifier confidence weights, the layer confidence thresholds are learned and optimized on the incoming data streams from the application domain in which drift is present. In Step 20, an optimal layer threshold $\Theta_i$ for layer $i$ is computed by first creating a distribution of all sums of node-classifier confidences. By treating each distinct sum as a threshold, an error metric can then be calculated for each one. Additionally, the confidence threshold value can be set to either favour higher hit rates or lower false positive rates by varying the weights of $\omega$ and $\upsilon$ values in (Step 27). Once the algorithm has completed classifying all instances from a current data stream and all layer confidence sums with their respective errors have been computed, the sum with the lowest error rate for each layer is selected as the optimal threshold (Step 29).

Once threshold learning is finished, the classifier is ready to be redeployed and can begin to handle drift (Step 12). Unlike most ensemble based methods, the algorithm makes use of a trigger mechanism to inform it that drift is occurring in the environment. In its current form, the algorithm uses the classification error rate as a trigger for drift handling to begin if the generalization ability falls beyond a predefined level (Step 14).

Initial experiments have shown that employing layer confidence thresholds achieves an aggressive strategy for eliminating false positive detections that can sometimes also reduce positive hit rates if not carefully applied. This observation can be combined with the flexibility of cascaded classifiers which allows it be combined with a strategy that varies the number of layers that utilize confidence thresholds depending on its current generalization. During runtime, the drift learning algorithm can progressively increase the number of layers to which layer thresholds are applied (Step 16) until a sufficient number of false positives has been eliminated, while the calculation of the preceding layers is preserved in the original form. The ability of the framework to progressively increase the number of layers that use confidence thresholds becomes the algorithm's facility for handling gradual drifts.

As data streams begin to drift more acutely, the algorithm activates proportionally larger numbers of layer confidences in respect to increasing false positive rates. This

proceeds until the drift stabilizes or until all available layers with confidence thresholds are exhausted. If all layer thresholds have been deployed and the error rate is still above an acceptable level, then the layer thresholds have been rendered irrelevant to current conditions and subsequently optimal layer threshold learning is re-initiated (Step 18).

The concept of using layer thresholds shares some affinity with the manner in which Viola-Jones employ the layer thresholds. Their approach adjusts the layer thresholds at training for the purpose of controlling the trade-off between the TPR and FPR. Similarly, the proposed layer thresholds also modify for each layer the operating point on the ROC curve. The approach outlined here differs in that the layer thresholds are learned based on clusters of weak classifiers that are trained on different partitions of data with distinct boosting rounds and thus do not explicitly represent the boosted margin theory of Yoav and Schapire [192].

## 6.2 Experiment Design

The algorithm was tested on a collection of images containing faces in different settings. The comparisons were conducted between the performances of a statically trained classifier with and without the application of the proposed adaptive algorithm. The underlying classifier was first trained offline using the BPSL method and the datasets described in Chapter 5.3. The resulting classifier comprised of 24 layers with 3-6 nodes per layer. Table 6.1 summarizes the details of the static classifier and its training procedures.

Table 6.1: Training and dataset details together with the properties for constructing the *static* classifier.

| **Property** | **Attribute** |
|---|---|
| Positive images | 5000 |
| Negative images | 2500 |
| Cascade layers | 24 |
| Ensemble-clusters per layer | 3-6 |
| Ensemble-cluster size | 1-20 |

The test images used for concept learning were completely independent from those used for the training of the static classifier. The test dataset consisted of 100 images comprising of $640 \times 480$ pixel dimensions, collected from a web-cam from different and varying environments. Each frame provided a stream of 336,980 instances in the form of image sub-windows to be evaluated. The total number of evaluated samples for the dataset exceeded 33 million, thus simulating a large volume and high-speed data stream. Initial sub-window size was $24 \times 24$ pixels and increased by the factor of 1.2 after exhaustive raster scanning of an image at an increment of 2 pixels per scan. Table 6.2 lists properties of the test set and the properties of the adaptive learning.

In compiling this dataset, the aim was to include instances of gradual and abrupt shifts as well as recurring contexts and data distribution changes over time. Gradual shifts were modelled through illumination changes and by panning the camera in a given

Table 6.2: Characteristics of the concept-drift learning dataset and the method.

| **Property** | **Setting** |
|---|---|
| Total images | 100 |
| Image size | 640 × 480 |
| Sub-window kernel size | 24 × 24 |
| Sub-window scale increase factor | 1.2 |
| Sub-window raster scan pixel increment | 2 |
| Total sub-windows per image | 336,980 |
| Images used for learning per drift phase | 1-5 |
| Sub-window scale increase factor | 1.2 |
| Sub-window raster scan pixel increment | 2 |
| Weighted adjustment for false positive detections $\omega$ | 0.5 |
| Weighted adjustment for true positive detections $\upsilon$ | 1.33 |


Figure 6.2: Sample test image sequence with various forms of drift. Starting from the top, row a) Example of gradual drifts becoming abrupt due to illumination changes in an indoor environment, row b) succeeded by sudden drifts into an outdoor setting, containing both data distribution changes through absences of faces and gradual changes modelled by camera panning into row c) concluding with a sudden change to indoors with a combination of the above drifts in row d.

environment. Sudden environment changes in the form of outdoor and indoor settings were used to model abrupt shifts as were rapid illumination variations. Image pre-processing steps such as contrast stretching and uniformity correction were not used in order to keep the processing overheads low and to make the detection task more challenging. Recurring contexts were modelled by repeating similar images from previously seen settings, while distribution shifts were created by exposing the algorithm to images with and without faces. Example images from the test dataset can be seen in Figure 6.2.

A maximum of one face per image was present in order to simplify the extraction of the ground truth data concerning face coordinates. The extraction of facial coordinates was assisted by constructing a head mask with attached fluorescent markers which were used for distinguishing positives from negatives during learning and verification phases. All learning was conducted automatically by the algorithm without manual intervention. The initial frame was programmed to trigger threshold learning irrespective of the offline trained classifier's performance.

In order to discover the optimal length of learning time required for the classifier on this domain, five experiments were conducted in which a different number of frames were utilized for each training phase. Initial tests relied on learning from a single frame and extended to the maximum of five consecutive frames. Since accuracy analysis was performed using all images, including the frames that were utilized for learning, for reasons of fairness the experiments that learned multiple frames applied their intermediate learning to their current classifier after each subsequent frame. Concept drift learning occurred on frames succeeding the initial frame whose error rate triggered the learning facility.

Due to the rare-event operating environment of face detectors, higher weights were assigned to positive samples than to the negatives when layer thresholds were learned. The purpose of this was to assist in maintaining high hit rates; however, the algorithm was equally capable of learning in images where faces were absent. The algorithm initiated gradual drift adaptability each time false positives were encountered. Concept learning for abrupt shifts was initiated if a false negative detection occurred or if false positives continued to occur on the lowest cascade layer set for calculating thresholds. In the experiments using a classifier with 24 layers, the eighth layer was set as the lowest for which layer threshold learning would be performed.

## 6.3 Results

The experimental results first cover the accuracy comparisons between the static classifier and the drift learning classifiers. Since face detection is a rare-event detection domain, focus was on analyzing the reduction of the false positive rate. The adaptive classifiers are then compared to each other in order to yield some insight as to what might be the optimal learning time-window for this algorithm in similar computer vision domains. The best performing adaptive classifier was then selected for a more detailed accuracy comparisons with the static classifier in addition to the analysis of runtimes and responsiveness to gradual and abrupt drifts[2].

[2]Parts of this research have been published in [159, 160].

### 6.3.1 Analysis of Multi-Frame Concept-Drift Learning

The results given in Figure 6.3 depict the logarithmic rate of decrease of the total number of false positive detections as a function of each successive cascade layer. At the end of each individual accuracy curve, the total number of false positive detections from each classifier are displayed. From this figure, the static classifier demonstrates the slowest rate of decrease in the false positive detections per cascade layer and ultimately produced the highest number of total false positive detections. All adaptive classifiers attained a reduction in the total number of false positive detections that ranged from 94%-99% over those of the static classifier. On this particular test dataset, of the five adaptive classifiers, the best accuracy was achieved by drift-learning performed on a window containing a single image frame.

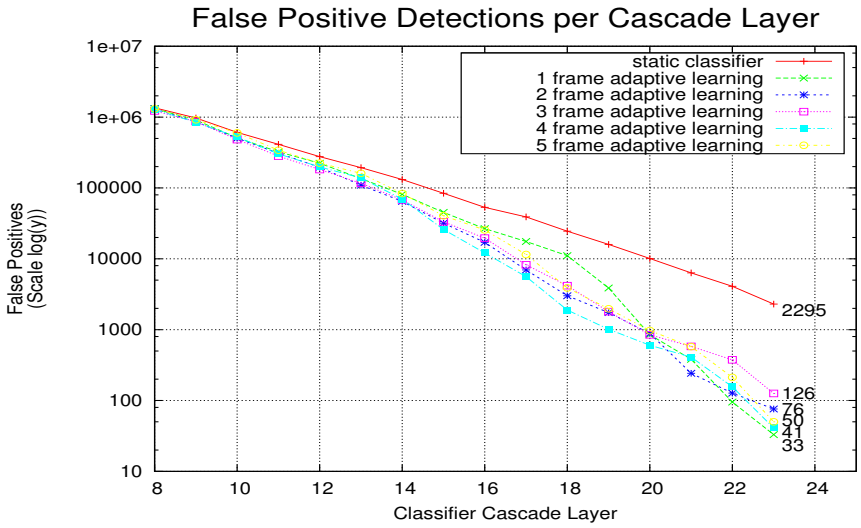



Figure 6.3: Total number of false positive detections per layer for each classifier and learning approach. The total number of false positive detections is shown at the end of each curve.

The following observations regarding the speed of responsiveness and adaptation of all classifiers are made from Figure 6.4. This figure approximates the actual runtimes through curve smoothing for the purpose of readability and in order to highlight overall trends. As expected, the figure points to generally longer runtimes for adaptive learning that takes place on larger windows. Learning on multiple snapshots of data proved to be particularly more expensive between frames 50-75. These frames correspond to images that contained both patterns of increased difficulty to classify, as well as a sequence of more frequent abrupt drifts that resulted in a high concentration of explicit threshold learning.

The total number of explicit learning phases did not vary significantly between the five adaptive classifiers, with all classifiers triggering 7-8 learning phases. From this it can be concluded that learning on larger snapshots of incoming data, did not result in an overall decrease in learning runtimes. This might have been expected had learning on larger data snapshots actually increased the accuracy, since fewer explicit learning phases would

have been triggered. On the contrary, both the accuracy of classifiers deteriorated when multiple snapshots of data were employed and lengthier runtimes were observed when using multiple data snapshots in the presence of frequent abrupt-drifts. On this particular domain, with this pattern and magnitude of environmental change, the algorithm has demonstrated an ability to rapidly adapt to drifting conditions using minimal data. The subsequent analysis focused in more detail on the characteristics of the single-frame adaptive classifier and compares it to the static classifier.

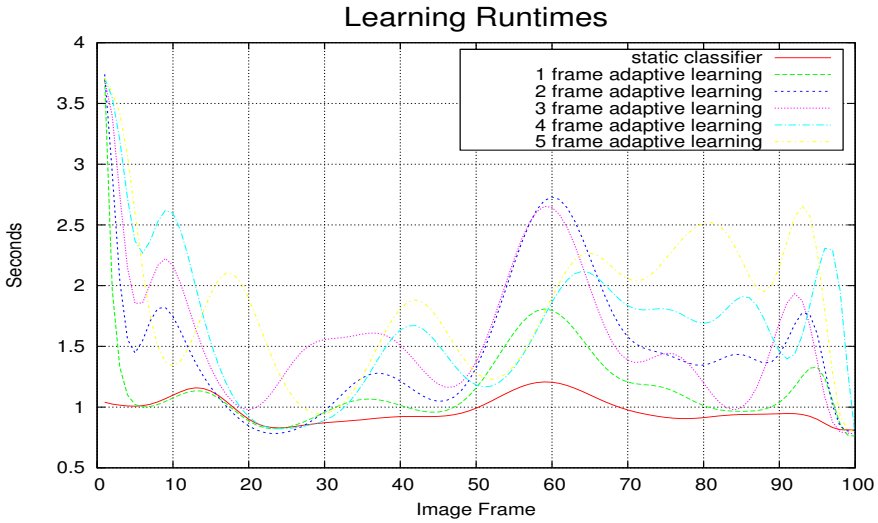
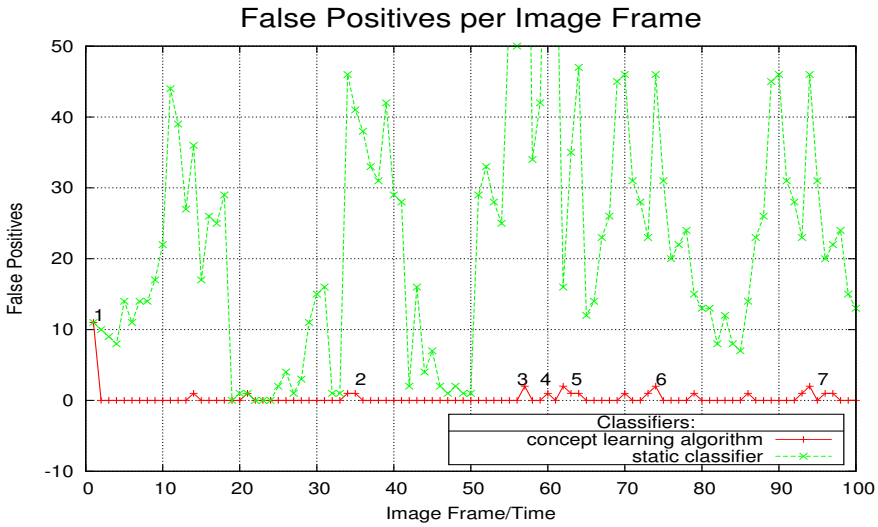



Figure 6.4: Execution runtimes for all classifiers displayed in approximated line curves that highlight general trends.

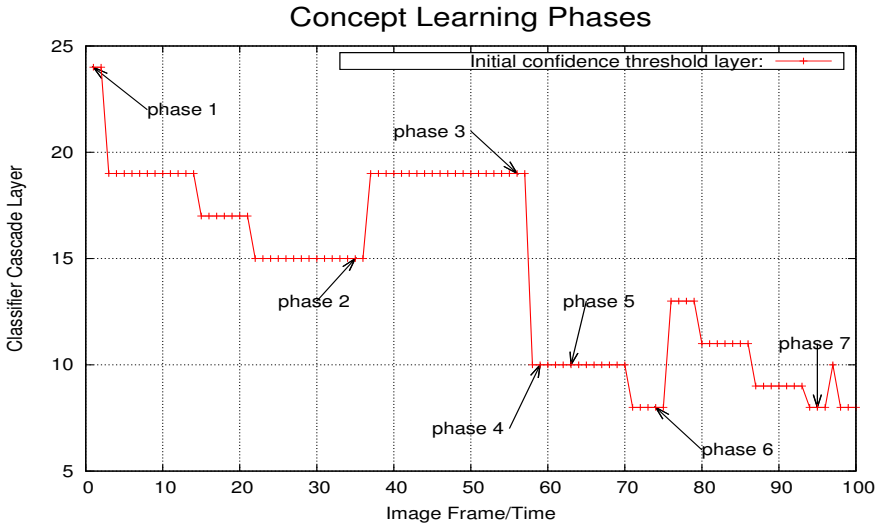
### 6.3.2 Results of Single-Frame Concept-Drift Learning

Given in Figure 6.5a, are comparison results of the total number of false positive detections per frame between the static and the adaptive classifiers. The figure shows that the false positive detections for the static classifier seldom stayed at zero, while the upper range exceeded 50 false detections per frame. By applying the drift-learning algorithm, the false positive detections were removed from all but 18% of the total images, while their total per image never exceeded two false positive detections. In total, 33 false positive detections belonging to the single-frame adaptive classifier were confined to only 18 images, while the distribution of 2295 false positive detections of the static classifier for all practical purposes, rendered most frames unusable. With regard to the overall hit rates, however, the reduction in false positive detections came at a small price. The static classifier correctly classified all positive samples while the drift-handling algorithm produced three false negative detections.

Recurring contexts were primarily simulated in the first 50 frames. They were made more challenging by inserting in the middle an abrupt drift in order to ascertain if concept forgetting would ensue. From the same figure, very low false positive detections can be observed occurring for the adaptive classifier throughout this phase and affirm that there is robustness in this algorithm to cyclical environmental shifts.




a)




b)

Figure 6.5: a) Total number of false positive detections per frame for a single frame learning classifier. Positions of the learning phases from 1-7. b) The pattern of the learning algorithm's adaptation to gradual drifts for each successive frame. As detection difficulty increases, the algorithm increases the number of layers for which threshold confidences are used by selecting lower layers as initial starting points.

### 6.3.3 Response Patterns to Gradual and Abrupt Drifts

Figure 6.5b demonstrates the adaptation pattern of the algorithm to gradual concept drifts per frame. More specifically, it shows how out of the total 24 layers of the cascaded classifier, an increasing number of layer thresholds were employed for classification as the accuracy deteriorated. This is seen in the figure in the manner in which the initial cascade layer for calculating layer thresholds decreases in response to increasing false positive detections. This figure is complemented by Figure 6.5a, by showing the magnitude of false positive detections which initiate the gradual concept-drift-handling facility. By noting the removal of subsequent false positive detections in Figure 6.5a, the effectiveness of the algorithm to handle gradual drifts can be seen.

Figure 6.6: An example of an image sequence that triggered the second concept-drift learning phase.

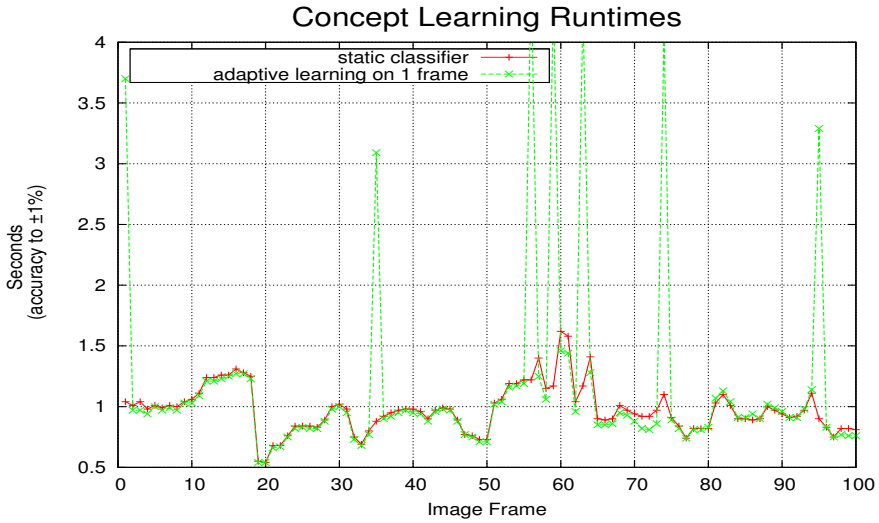
The high concentration of abrupt concept drifts that took place between frames 50-75, as mentioned earlier, is also pointed out in Figure 6.5b by phases 3-6. These transitions caused higher error rates to occur, resulting in the initiation of four layer threshold learning phases in short succession. Despite this, the algorithm still demonstrated robustness in its responsiveness to frequent sudden drifts in the data. This can be observed in the decrease and rapid stabilization of the false positive detections after abrupt drifts have taken place. Figure 6.6 shows an example of a sequence of images containing an abrupt drift that was simulated through an extreme illumination change and which in turn activated a layer threshold learning phase.

### 6.3.4 Learning and Detection Runtimes

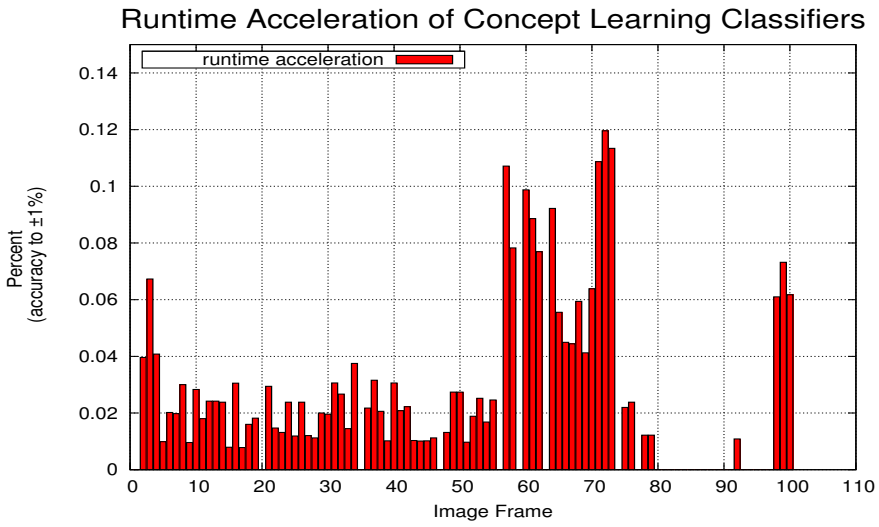
The cost of drift learning in extra runtime amounts to 3-4 times the classification time of the static classifier (Figure 6.7a). There is however sufficient scope for optimizing the learning phase through parallelization that would result in an overall marginal performance penalty for time critical applications. Arguably for most applications, the modest detection time increase is likely to be acceptable in exchange for a considerable accuracy improvement.

Contrary to expectations, the detection runtimes of the adaptive classifiers using layer thresholds are consistently faster when they are not in a learning phase, than that of the static classifier (Figure 6.7b). The detection runtime acceleration of up to 10% is attributed to the fact that the learned layer thresholds correctly classify more negative samples in earlier cascade layers, thus preventing their further propagation to subsequent layers and in the process decreasing computation costs. The faster detection runtime of the adaptive classifiers mitigates to some degree the increase in runtimes brought on by

the learning phases of layer thresholds.




a)




b)

Figure 6.7: a) An accurate comparison of runtimes between the single-frame learning classifier and the static classifier. b) The percentage of increased execution runtime of the single-frame learning classifier over the static classifier.

## 6.4 Discussion

Wang et al. [177] justifiably point out that most ensemble-based solutions are made up of individual weak classifiers which are inherently not semantically aware. The point of this is that, in these approaches it is hard to correspond individual weak classifiers with a hidden concept and as a result, a costly, global update of an ensemble becomes necessary. Consequently, a vulnerability to high-speed data streams materializes. According to their definition, an ensemble-based framework is semantically aware if it meets three criteria:

- each member of an ensemble must be responsible for independently predicting a subset of samples
- members of the ensemble must be revisable without affecting other components
- each component must explicitly correspond to a certain hidden concept

The design of the proposed solution meets these criteria through the decomposition of the classifier into node-classifiers at each layer. Each node-classifier is trained on a given subset of samples and thus a correct prediction of an instance is contingent only on a single node-classifier per layer. The independence between the node-classifier ensures that extensions to this framework can easily be made in the form of adding and replacing new node-classifiers during runtime, which can then be explicitly associated with hidden concepts.

There are several other noteworthy properties of this approach to handling drifting concepts in non-stationary environments. The proposed algorithm rapidly realizes adaptation even in large data streams. Depending on the operating domain and the complexity of feature extraction, this can be achieved with a modest computational penalty, thus preserving real-time classification.

The algorithm is also efficient since only layer thresholds are learned and updated instead of individual weak classifiers. For this reason, the algorithm achieves a unique balance between stability and plasticity in learning that is unlike existing approaches. Its greater emphasis on classifier stability allows it to preserve previously learned information and makes it strongly robust to recurring contexts, while still being sufficiently responsive to changes in its operating environment.

This research has shown that the algorithm is particularly effective in rare-event operating domains where extremely low false positive detections are paramount. The algorithm has demonstrated an ability to improve accuracies of classifiers whose high false positive rates have otherwise rendered them unpractical. As a consequence, this approach to drift-handling may contribute to simplifying and accelerating the offline training process in general, by reducing the otherwise required number of negative training samples.

In addition to this, the experiments have shown the capability of the method to correctly classify positive samples even when learning takes place on snapshots of data which only contain negative samples. As long as the static classifier has been trained on sufficiently large positive datasets, this property may be employed to provide a form of background modeling that can be performed before a classifier is deployed.

A positive characteristic of this algorithm has been its ability to adapt to changing conditions using minimal snapshots of incoming data. For the particular problem domain and test dataset used in this research, it became apparent that there was no inherent advantage for the algorithm in exposing it to larger data snapshots. This may prove to be an equally occurring characteristic in other domains and will therefore be a subject of future research.

The calculation of layer thresholds has also been shown to speed up the detection runtime since more negative samples are rejected by earlier layers resulting in a decrease in computation loads.

### Model Limitations

The shortcomings of the algorithm lie in its current limitation of being unable to explicitly learn novel positive samples, or positive samples whose class description has been radically modified over time due to environmental changes. In that sense, the algorithm is not incremental, but rather adaptive in nature. As mentioned earlier, the modular structure of the proposed framework enables extensions for incremental learning that would consist of augmenting each cascade layer with new node-classifiers[3]. Therefore, at present the algorithm is restricted to correctly classifying positives containing patterns learned during its offline training phase. Hence, this approach is limited in its effectiveness to environments whose target concept does not undergo substantial modifications of its class descriptions, but rather where changes in the environment are moderate and recurring. However, the fact that the approach does not explicitly learn new classifiers, nor alter weights of the existing ones, does make it resistant to incorrectly learning and modeling noise.

In its present form, the algorithm is limited to binary class problems only. However this can be overcome, since all multi-class problems can be reduced to a series of two-class classification tasks [3, 94]. This results in $k$ number of distinct classifiers being trained, for which the algorithm can be applied. Also, the approach requires that the classifier ensemble be decomposed into a cascade-like structure with distinct clusters of classifiers. Existing classifiers would therefore need to be re-trained to take advantage of this method.

In the proposed system, it is also difficult to determine the optimal balance of responsiveness to concept-drift between gradual adaptation, implemented in the algorithm through progressive increases in the number of layers to which layer-thresholds are applied, and the explicit re-learning of all layer-thresholds when the magnitude of drift is perceived to render previous learning irrelevant. If the balance between the two modes of handling drift is designed poorly, then either the adaptation of the system will be too slow, causing long durations of classification errors before re-learning of layer thresholds is initiated, or the system will become a constant-update approach possibly leading to performance degradation.

Finally, the detection of drift as well as the adaptation of the system is reliant on supervised learning and cannot be done without explicit input of ground truth describing class labels of new instances. However, as alluded to earlier, this can be alleviated to some degree in binary-class rare-event domains in which degrees of adaptation can be achieved by only exposing the algorithm periodically to background samples.

## 6.5 Summary

The foremost aim of the chapter was to explore ways of enabling real-time learning adaptability for PSL-like classifiers. In particular, the motivation behind the research was for problem domains where explicit training of new ensembles is computationally infeasible due to large volumes of streaming data as well as for cases where the data is represented by immense dimensionalities. The goal was to replicate such an operating environment

[3] In the earlier work [156], it was shown how the PSL algorithm is capable of learning incrementally by explicitly training additional nodes at each layer for misclassified samples, which are then appended to the associated layer.

and to demonstrate the ability of this algorithm to adapt to diverse forms of drift without inducing concept forgetting.

The main research findings of this chapter are as follows:

1. The PSL-like classifiers can be effectively combined with adaptive learning algorithms for handling drifting-concepts on large-scale binary class domains.

2. The adaptive algorithm is capable of significantly improving the accuracy of an offline-trained classifier without resorting to explicit re-training of weak classifiers.

3. The proposed algorithm is able to adapt classifiers operating on high-volume data streams in real-time with the capacity to respond to all forms of concept drift.

4. The algorithm does not suffer from concept forgetting as commonly experienced in recurring contexts, thus creating a balance between both plasticity and stability of learning that is sustainable over time.

5. Using this method, effective adaptive learning can take place on samples belonging to majority class labels for problems with large distribution skews. This is particularly useful on rare-even domains where the variation in the background samples is large, while the integrity of the class descriptors of the target concepts is more stable.

6. The trade-offs however come in the form of limitations of the algorithm to explicitly learn and integrate novel information for the target concepts. This restricts the method's effectiveness to respond to target concepts which undergo radical modifications to their class descriptors as well as to new samples of the target class whose descriptors have not been learned offline.

# Chapter 7

# Expanding the Coarse-to-fine Principles to Multiclass Learning

By using face detection as a medium, previous chapters have established the efficacy of the PSL-like framework in its application to a range of binary-class classification problems. Subsequent focus of this research turned to the multiclass problem. The specific goal was to explore the ability of applying the PSL ensemble-decomposition strategy to the challenges of learning multiple class labels.

Historically, the majority of machine learning research focused has been on the two-class or binary problems. A number of popular and successful machine learning techniques such as boosting, SVM [26] and the RIPPER[1] [25] algorithms were originally designed with the binary problem in mind [94]. However, the reality is that many classification problems do involve predictions that require an assignment to one of multiple possible classes. Since the probability of making a wrong prediction in the multiclass domain is higher than for binary classification, multiclass problems are therefore considered to be inherently more difficult to solve, especially as the number of classes increases [94].

There are many varieties of algorithms that are well suited to the classification requirements of multiclass learning. Amongst these, two major groups of classification approaches can be identified. The first is referred to as a divide-and-conquer strategy and proceeds by creating a decision tree from a given training dataset. The decision points at each stage of the tree are calculated based on tests from selected feature types. The datasets are partitioned based on the outcome of this test and the process continues recursively until the required accuracy is achieved. The algorithms belonging to this category are CART [14], ID3 [126], C4.5 [25] and its extension C5.0, which have collectively been the most sophisticated tree induction methods of the past two decades [150].

The second family of algorithms describe methods that employ a separate-and-conquer[2] strategy. These types of algorithms approach the problem by focusing the learning on instances of one class label at a time. Rules are generated that attempt to correctly predict as many instances of a target class label while excluding a maximal number of instances not belonging to it. After each round of rule generation, the correctly classified instances

[1] Though RIPPER itself was designed as a multiclass algorithm, it is a direct descendant of REP [17] and IREP [51] which were initially conceived as binary-class learning algorithms.

[2] This group of algorithms is often referred to as 'covering'algorithms within the data mining community. The covering refers to the instances that are predicted sequentially by conjunctions of rules.

are removed and the process continues to iteratively learn the remaining instances. Of these, PRISM [19], IREP [51] and RIPPER [25] algorithms stand out as some of the most effective strategies.

Suh [150] describes the divide-and-conquer strategy as favouring accuracy, while the separate-and-conquer strategy as favouring simplicity. Frank and Witten [43] combine the two approaches into one through a strategy of partial decision-tree induction. This method proceeds as a separate-and-conquer algorithm but differs in how the rules are generated at each round. The approach taken is to construct a pruned decision tree at each round with the current samples and to select the leaf with the largest coverage. The combined approach replaces the need for global optimizations and mitigates against overpruning [184].

Alternative strategies to multiclass problems consider successful binary classification techniques that have already been developed and adapt their internal operations for a multiclass setting. However, it is not trivial to extend the internals of binary learning algorithms to multiclass and in some cases it is also quite impractical [114]. Hsu and Lin [65] have shown that in the case of SVMs, this leads to higher training costs. For these reasons, one of the most common approaches involves decomposing a multiclass problem into a series of binary learning subtasks [86]. By using this strategy, the computational complexity for inducing a classifier is reduced. In instances where only pairs of class labels are considered, they can often be linearly separated, while this may not be at all possible when considering the entire multiclass problem at once [134].

The most current assesement by experts in the field concludes that the application of the cascaded classifiers to the challenges of multiclass learning is still an open problem [194]. The most common methods have involved either constructing separate parallel cascades for each class or building cascaded detector trees (reviewed in Chapter 2.3.1). Recently, the most notable contributions to the design of integrated multiclass cascaded classifiers have been by Verschae and del Solar [170]. In their research, a multiclass boosting algorithm called VectorBoost [67] was combined with a domain-partitioning weak classifier to build a cascaded classifier which in addition reuses information from previous layers in order to produce compact and robust multi-view face detection classifiers.

## 7.1 Motivation

In this chapter, a unique multiclass learning method that decomposes the training and detection task into PSL-like cascades is discussed. A new multiclass weak learner for combining with the cascaded training framework is also presented. The research here sought to answer the following questions:

1. Can the PSL-like decomposition strategy be applied to multiclass learning in such a way that complex problems can be learned efficiently and accurately?

2. What are the effects of this training approach on the training and evaluation runtimes of the classifiers on datasets of varying sizes?

3. How does this approach to learning affect the generalization of the classifiers trained on datasets with balanced and unbalanced class distributions?

4. Given the inherent difficulties in learning minority class labels on datasets with skewed distributions, does the decomposition strategy mitigate the challenges of learning on these kinds of datasets?

## 7.2 PSL Multiclass Learning Framework

One of the primary objectives of this ensemble-based multiclass algorithm was to devise an approach to learning complex problems efficiently. In order to do this, a fast multiclass weak inducer was needed. The inducer needed to be weak so as to neither overfit the data, nor to overshadow the learning ability of the cascaded framework by inducing class boundaries that were too accurate. This section begins with a description of the cascaded multiclass framework which will make the design of the multiclass learner more clear.

### 7.2.1 Multiclass Cascade

The underlying principle of the proposed multiclass cascade is the separate-and-conquer strategy. As such, it bears conceptual similarities with covering algorithms like RIPPER. The algorithm refines the cascaded classifier in a stepwise fashion by continuously removing most separable class labels, in order to focus on more difficult classes.

Given a training set $D = \{(x_1, y_1), ..., (x_n, y_n)\}$ where $x_i$ is a sample vector $x_i \in X$ and $y_i$ is a class label $y_i \in Y$ and $Y \in \{0, ..., k\}$, the multiclass algorithm constructs a cascade classifier $H_{k-1}$ that consists of $k-1$ number of layers. Each layer $i$ is specifically trained to predict a single class label $c$, as $H_i^c(x) \in Y$, otherwise the prediction is passed on to $H_{i+1}^c(x)$.

The cascaded classifier $H_{k-1}$ is also two dimensional, whereby each layer contains within it a further *nested* cascade $H_{i,j}^c$ (Figure 7.1). The nested cascade facilitates the coarse-to-fine learning of each layer, which enables the convergence to low training error rates. For clarity each layer $j$ of a *nested* cascade, denoted as a pair of $(M_j, B_j)$, is refered to as *nodes*. $M_j^\gamma(x)$ denotes a multiclass node whose prediction $\gamma \in Y$ and $\gamma \neq H_{i,j}^c(x)$, while $B_j$ denotes a binary predictor whose output is $B_j(x) \in \{-1, 1\}$.

The cascade training proceeds as follows: in the first layer $H_i$, the initial multiclass node $M_{i,j}$ is trained on all samples using a generic multiclass algorithm until a predefined $\Phi$ number of boosting iterations are completed. Once this criterion is met, node $M_{i,j}$ is assessed for accuracy based on individual class error rates. The most separable class label $\gamma$ is then identified for separation and assigned to node $M_{i,j}^\gamma$ as its target class for prediction.

All *correctly* predicted samples $(M_{i,j}^\gamma(x_i) = \gamma) \wedge (y_i = \gamma)$ are then trained against all the *incorrectly* predicted samples as class $(M_{i,j}^\gamma(x_i) = \gamma) \wedge (y_i \neq \gamma)$, using a binary learning approach. The resulting binary node $B_{i,j}$ functions as an auxiliary node to the multiclass node. The auxiliary node $B_{i,j}$ is trained until all the false positive samples with respect to class $\gamma$ have been correctly learned. All correctly learned samples $(M_{i,j}^\gamma(x_i) = \gamma) \wedge (y_i = \gamma) \wedge (B_{i,j}(x_i) = 1)$, belonging to class $\gamma$ are then removed from subsequent training of layer $H_i$. The training of node $M_{i,j+1}$ proceeds as with the initial node; howbeit, with $k - j$ class labels to learn.

The coarse-to-fine learning continues until all the most separable class labels and their instances have been removed from layer $H_i$. If at the end of the layer training, there

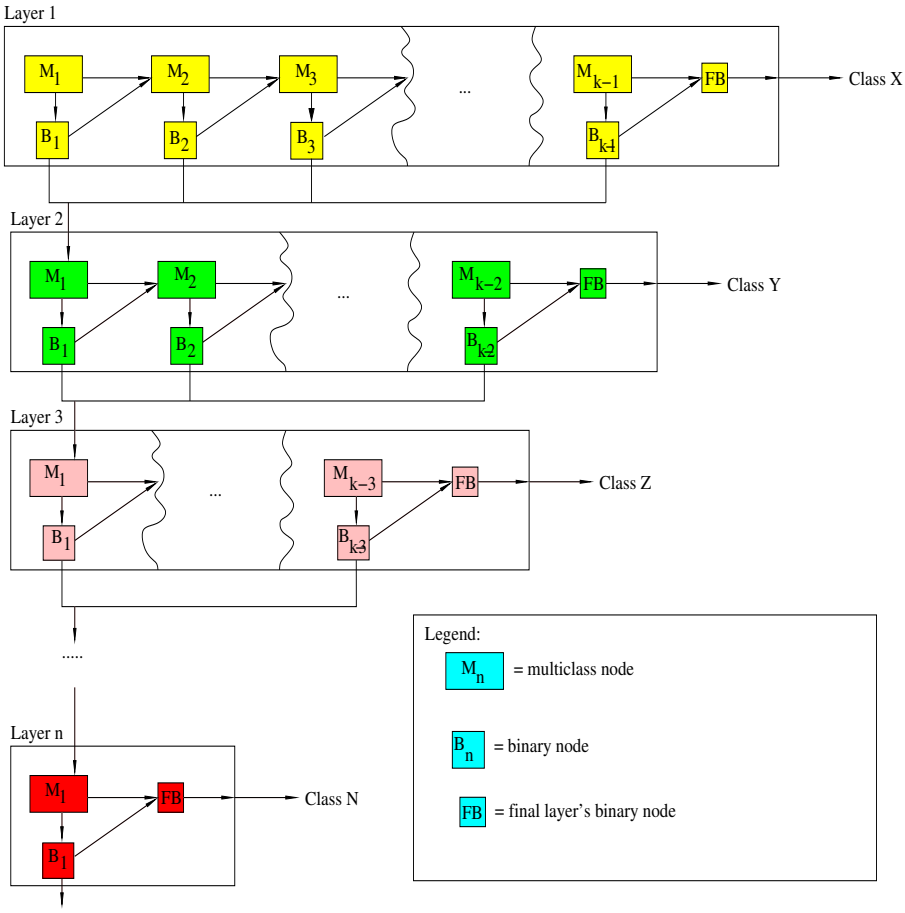



Figure 7.1: Cascaded multiclass framework.

remain instances which have not been correctly learned and associated with correct class labels, then a final binary node $FB_{i,j}$ is trained, whose output is $\{-1, 1\}$. In the training of the $FB_{i,j}$ node, the incorrectly learned samples are designated as negatives, while the instances belonging to the last remaining class label $c$ as the positives. The binary training proceeds until a predetermined error rate is achieved. Subsequently, the class label $c$ is assigned as the predictor for layer $H_i^c$. All instances belonging to class label $c$ are eliminated from further training of layers $H_{i+1}$. In turn, each succeeding $H_{i+1}$ layer with $k - i$ class labels reduces the computational demands and the complexity of class separability. Algorithm 10 outlines the details of the multiclass learning framework.

Key components of separate-and-conquer strategies include strategies for (1) determining the sequence in which class labels are selected for learning and subsequently, (2) the methods of refining the classification accuracy of a chosen class label from the remaining classes.

The approach taken by RIPPER for selecting class labels, involves sorting the list of class labels in an ascending order of size distribution and learning labels by beginning with the smallest class labels first. The multiclass differs in that it makes no decision on which class label to focus on learning *a priori*. Instead it equally focuses learning all classes and the most separable class label $\gamma$ is assigned to the current $M_{i,j}$ node, while the correctly

**Algorithm 10**: PSL Multiclass Cascade

**Given**: $n$ = training set size, $c$ = number of class labels, $\Phi$ = max weak classifiers, $P_n$ positive and $N_n$ negative dataset vectors $\{(x_1, y_1), ...(x_n, y_n)\}$ for binary training where each sample $x$ corresponds to a class label $y \in \{-1, 1\}$

**Input**: $D_n^c$ = multiclass dataset vector $\{(x_1, y_1), ...(x_n, y_n)\}$ where each sample $x_i$ is a feature vector $\in X$ corresponding to a total of $k$ classes with each class label denoted as $y \in Y$ and $Y$ is the set of all class labels.

**Output**: $H^{final}$ = cascaded classifier comprising of multiclass $M$ and binary $B$ and $FB$ nodes.

**for** $i$ = *1* **to** $k$ **do**
  FlagSamplesForTraining($D_n$) where $H(x_i) \notin y_i$
  **for** $j$ = *1* **to** $k$ - $i$ **do**
    `// train new multiclass node`
    **for** $k$ = *1* **to** $\Phi$ **do**
      $M_{i,j}$ = MultiClassWeakLearn($D_n$)
      Boost($D_n$, $H_{i,j}^M$)
    $M_{i,j}^\gamma = \gamma$, where $\gamma$ = BestClassAccuracyLabel($D_n$, $M_{i,j}$)
    $error^\gamma = M_{i,j}^\gamma(D_n)$
    **if** $error^\gamma > target^{multiclass}$ **then**
      $P_n = (x_i, y_i)$, where $(M_{i,j}^\gamma(x_i) = \gamma) \wedge (y_i = \gamma)$ `// train supporting binary node`
      $N_n = (x_i, y_i)$, where $(M_{i,j}^\gamma(x_i) = \gamma) \wedge (y_i \neq \gamma)$
      **repeat**
        $B_{i,j}$ = BinaryClassWeakLearn($P_n$, $N_n$)
        Boost($P_n$, $N_n$, $B_{i,j}$)
        $error^{binary} = B_{i,j}(P_n, N_n)$
      **until** $error^{binary} > target^{binary}$ ;
      RemoveSamplesFromLayerTraining($P_n$, $D_n$) where $B_{i,j}(x_i) = 1 \wedge y_i = \gamma$
  $H_{i,j}^c = c, \forall M_i^\gamma$ where $\gamma \neq c$
  **if** $H_{i,j}^c(D_n) > target^{final}$ **then**
    $P_n = (x_i, y_i), \forall x_i$, where $y_i = c$ `// train final binary node`
    $N_n = (x_i, y_i)$, where $(H_{i,j}^c(x_i) = c) \wedge (y_i \neq c)$
    **repeat**
      $FB_{i,j}$ = BinaryClassWeakLearn($P_n$, $N_n$)
      Boost($P_n$, $N_n$, $FB_{i,j}$)
      $error^{binary} = FB_{i,j}(P_n, N_n)$
    **until** $error^{binary} > target^{final}$ ;
return $H^{final}$

predicted instances are then removed from further layer training. The justification for choosing the best performing class label is supported by research conducted into training classifiers using binary directed trees. Training classifiers using binary directed trees falls under the umbrella of hierarchical decomposition strategy[3] with which the multiclass cascades share much in common. In their work with binary directed trees, Takahashi and Abe [165] propose that initial levels of a tree should divide the most separable classes. Their intuitive argument is that by dividing the most separated classes in the initial levels of a tree first, a higher overall accuracy can be achieved.

The second key aspect of the framework concerns how the accuracy of a node $M_{i,j}$ is to be refined in its prediction for class label $\gamma$. The accuracy in respect to hit and false positive rates of a class label chosen for removal, have implications for both the training phase and the generalizability of a classifier. The hit rate of the selected class $\gamma$ determines the proportion of its samples that will be removed from further training of the layer. If this is low then the class separability of the renaming sub-tasks is not likely to be greatly reduced. Also the training will not become less computationally expensive. However, a negligible false positive rate is even more crucial, since class label $c$ for layer $H_i$ is not known *a priori*. This means that a cumulative error for instances of class $c$, over all auxiliary nodes in a layer, renders the final accuracy of $H_i^c(x)$ impractical. For this reason the binary class learning focuses primarily on lowering the false positive rate instead of the true positive rate in respect to class label $\gamma$. RIPPER for example differs in that once a class is selected for learning, it uses the information gain measure of correctness of the induced rules, to increase its accuracy in respect to the true positive rate.

The complexity of the cascade training can be approximated by showing the total numbers of multiclass and binary weak classifiers that are generated. For $k$ classes, the number of multiclass nodes $M_k$ is

$$total_M = \frac{k^2 - k}{2} - 1 \tag{7.1}$$

and binary nodes $B_k$

$$total_B = \frac{k^2}{2} + 1 \tag{7.2}$$

Suppose that the size of all $M_i$ nodes is fixed at $\Phi$ weak classifiers, while the size of $B_i$ nodes is flexible but cannot exceed $\Psi$ weak classifiers. We know that the initial $B_{1,1}$ node is most complex, and if $\Psi$ is set sufficiently high, then the size of $B_{i,j}$ decreases logarithmically after each node and layer. The total number of classifiers is therefore

$$total = \Phi\left(\frac{k^2 - k}{2} - 1\right) + log(\Psi)\left(\frac{k^2}{2} + 1\right) \tag{7.3}$$

where the number of class labels is the dominating term in the complexity.

The runtime procedure for classifying a sample ensures that only a subset of the cascaded classifier is evaluated as opposed to monolithic ECOC-based, OAO and OAA approaches. As an unseen sample $(x_i, y_i)$ enters the cascade at $H_{1,1}$, it is forwarded to a subsequent layer for classification if any combination of nodes produces a prediction

[3]Hierarchical decomposition strategies recursively divide a learning task into simpler subtasks.

$$((M_{i,j}^{\gamma}(x_i) = \gamma) \wedge (y_i = \gamma)) \wedge (B_{i,j}(x_i) = 1) \tag{7.4}$$

which predicts it as matching its target class label and if the associated binary node confirms it as a positive. A sample is forwarded to subsequent layers until it reaches a layer $H_i^c$, where the condition (7.4) fails in all cases and the final node $FB_{i,j}(x_i) = 1$ confirms it as a positive for class label $\gamma$. The output of the nodes on previous portions of the cascade is not considered by each node and therefore each component is independent. As a sample instance propagates further into the cascade, the classification accelerates since there are a decreasing number of nodes at each layer to evaluate.

To further differentiate the multiclass cascade from existing separate-and-conquer approaches, the multiclass cascade neither employs pruning nor global optimization steps as do some rule induction algorithms[4]. In addition, since separate-and-conquer approaches tend to be rule induction algorithms, they produce classification rules that are human readable while the ensemble based methods like the multiclass cascade generate classifiers that are incomprehensible.

### 7.2.2 Multiclass Weak Learner

Schapire and Singer [142] present a method for generating a weak hypothesis based on partitioning the domain $\vec{x}^f \in \mathbb{R}$ that comprises of all sample values $x_i$ from a dataset in respect to a given feature $f$ into distinct and disjoint intervals, referred to also as bins. The idea is, if sample values representing the input space $\vec{x}$ is seen as a distribution, then it can be partitioned into confidence intervals. As a result, a meaningful confidence value can be associated with each value $x_i$ depending on what bin it falls into which is otherwise non-existent when only hard thresholds are used. The authors state that this additional information about a sample can be extracted and used to improve both learning and detection.

This general principle is extended here to the problem of generating multiclass weak hypotheses and is also inspired by the work of [173], who demonstrated how optimal thresholds for simple decision stump learners can be calculated with speed and efficiency. A variety of existing methods for generating multiclass hypotheses could have been used instead. Of these, C4.5 was a candidate. However, such sophisticated methods, though fast on small and moderately sized datasets [149], can be computationally expensive on larger datasets [150] and are hard to callibrate so that they do not overfit the data [59].

For these reasons the motivation here was to examine how well a cascaded framework can learn and generalize given a weak classifier, that is both inexpensive and unlikely to overfit. With that in mind, the aim was to design a multiclass weak learner for the cascaded framework, that complements the 'separate-and-conquer' strategy, by producing high accuracies for at least one class label. Though overall error rates that are only marginally better than random guessing as in $\frac{K-1}{K}$ for $K$ classes was desirable, it was not absolutely required for every weak classifier.

While Schapire and Singer [142] divide the binary-class domain $\vec{x}$ into a predetermined number of disjoint *bins*, the learner presented here divides the vector $\vec{x}$ into $K$ possibly

[4]Both IREP and RIPPER employ pruning while RIPPER uses sophisticated global optimization after the construction of rules for each class label.

**Algorithm 11**: Domain-partitioning Weak Learner

**Given**: $T_k^+$ = total weights for all samples $(x_i, y_i)$ where $k = y_i$, $T_k^-$ = total weights for all samples $(x_i, y_i)$ where $k \neq y_i$, $\omega_k^+$, $\omega_k^-$ = current sum of weights, $Z_k$ = normalization coefficient for class $k$, $\varepsilon_k^{\rightarrow}$ = current error for threshold pointing right, $\varepsilon_k^{\leftarrow}$ = current error for threshold pointing left, $E_k$ = minimum error for class $k$

**Input**: training set $D = \{(\vec{x}_1, y_1), ..., (\vec{x}_n, y_n)\}$ where $\vec{x}^f$ is a feature vector $f$ of a total $F$ features and $y_i$ is a class label where $y_i \in Y$ and $Y \in \{0, ..., K\}$. Each sample $x_i$ has an associated weight $w_i$.

**Output**: multiclass weak classifier $h$ consisting of $(\vec{t1}^f, \vec{t1}^f, \vec{E})$ where $\vec{t1}_k^f, \vec{t2}_k^f$ are primary and secondary thresholds respectively for a feature $f$ and class $k$

$\forall_k$, initialize $T_k^+$, $T_k^-$
$\forall_k$, compute $Z_k$
**foreach** *feature* $f$ **to** $F$ **do**
  $\vec{x} = \forall_i, (x_i^f, y_i)$
  sort $\vec{x}$
  **foreach** *value* $x_i$ **to** $x_n$ **do**
    **foreach** *class* $k$ **to** $K$ **do**
      $\omega_k^+ = \sum_{k=0}^{K} Z_k \cdot w_i$, **where** $y_i = k$
      $\omega_k^- = \sum_{k=0}^{K} Z_k \cdot w_i$, **where** $y_i \neq k$
      $\varepsilon_k^{\leftarrow} = \omega_k^+ + T_k^- - \omega_k^-$
      $\varepsilon_k^{\rightarrow} = \omega_k^- + T_k^+ - \omega_k^+$
      **if** $E_k^f < Min(\varepsilon_k^{\leftarrow}, \varepsilon_k^{\rightarrow})$ **then**
        $E_k^f = Min(\varepsilon_k^{\leftarrow}, \varepsilon_k^{\rightarrow})$
        $t1_k^f = x_i, GetBestErrorDirection(\varepsilon_k^{\leftarrow}, \varepsilon_k^{\rightarrow})$
  $\forall_k$, re-initialize $T_k^+$, $T_k^-$ with respect to $t1_k^f$
  $\forall_k$, compute $Z_k$
  **foreach** *value* $x_i$ **to** $x_n$ **do**
    **foreach** *class* $k$ **to** $K$ **do**
      **if** $(t1_k^f \ is \ \leftarrow \wedge \ x_i < t1_k^f) \vee (t1_k^f \ is \ \rightarrow \wedge \ x_i > t1_k^f)$ **then**
        $\omega_k^+ = \sum_{k=0}^{K} Z_k \cdot w_i$, **where** $y_i = k$
        $\omega_k^- = \sum_{k=0}^{K} Z_k \cdot w_i$, **where** $y_i \neq k$
        **if** $t1_k^f \ is \ \leftarrow$ **then**
          $\varepsilon_k^{\rightarrow} = \omega_k^- + T_k^+ - \omega_k^+$
        **else**
          $\varepsilon_k^{\leftarrow} = \omega_k^+ + T_k^- - \omega_k^-$
        **if** $E_k^f < Valid(\varepsilon_k^{\leftarrow}, \varepsilon_k^{\rightarrow})$ **then**
          $E_k^f = Valid(\varepsilon_k^{\leftarrow}, \varepsilon_k^{\rightarrow})$
          $t2_k^f = x_i$
return $f = Min(E^f)$

overlapping bins, where each bin $j_k$ represents a bin that is associated with a single class label. The weak learner proceeds by first sorting the feature vector $\vec{x}$. The learner then traverses $\vec{x}$, searching for $K$ optimal thresholds $t_k$ for each class with an associated direction. At each potential threshold point $x_i$, the error is calculated $K$ times for each class. However, the error is calculated with respect to a binary distribution, where an even sum of weights is assigned to both the positive and the negative sets. In effect, the error calculation reduces to a form of one-against-all training. The first traversal of $\vec{x}$ is sufficient to generate $K$ optimal threshold values with respect to a binary distribution for each class, together with their directions. Algorithm 11 details these steps.

An additional traversal of $\vec{x}$ is necessary to generate a bound on the first threshold so that bins and partitions can be created. This bound is referred to as a secondary threshold and it represents an optimal threshold with respect to the first threshold, being based on the error of the binary distribution. Each secondary threshold must lie within the value range that corresponds to the direction of the first threshold. The direction of the secondary threshold needs to be opposite to the direction of the first threshold so that a coherent partition of a class can be created (Figure 7.2).

Each partition $j_k$ is assigned a confidence value based on its accuracy. The average error rate of all partitions defines the overall error rate for the given hypothesis and a given feature. Predictions for samples on overlapping partitions is awarded to the partition with the highest confidence.

Given a total pool of $F$ features, for each feature vector $\vec{x}^f$ the weak learner executes a quicksort function which is of order $O(Nlog(N))$ for $N$ samples and traverses the vector $\vec{x}^f$ twice. This results in a total number of iterations amounting to

$$F(Nlog(N) + 2N) \tag{7.5}$$

which can be expressed in terms of complexity without the linear component as

$$O(FNlog(N)) \tag{7.6}$$

Under usual conditions, the size of the training samples therefore dominates the complexity; however, exceptions can occur when training images using Haar-like features, whose total feature space can be in excess of 200,000 features.

## 7.3 Experiment Design

Experiments were performed using the above cascading algorithm, combined with the domain-partitioning (DP) weak-learner and AdaBoost.M1[5] [44] for multiclass nodes, while Discrete AdaBoost and decision stumps were used for auxiliary nodes. The experiments were conducted on 18 multiclass datasets from the UCI machine learning repository. The dataset attributes are described in Chapter 3.2. The combination of the two is henceforth referred to as the cascaded.DP algorithm. AdaBoost.OC, ECC and M2 were implemented in C to compare with the proposed algorithm.

[5] AdaBoost.M1 is identical to Discrete AdaBoost and differs only in that it is able to handle multiple classes. Its only point of difference is in the evaluation of the ensemble, whereby the class label with the maximum sum of weighted votes becomes the predicted label.

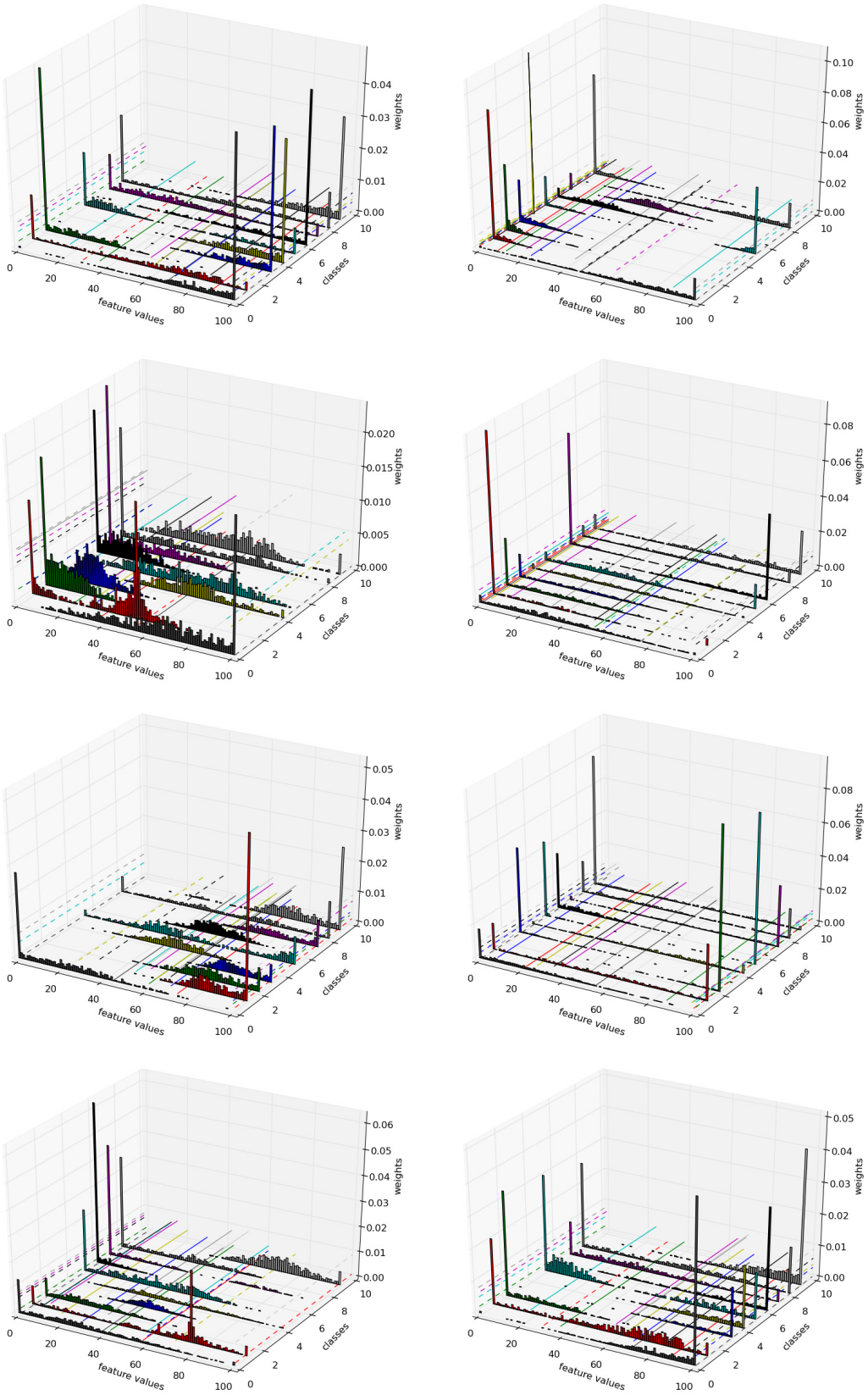



Figure 7.2: An example of the proposed multiclass weak learner selecting the best features on the first 8 boosting iterations of the Pendigit dataset from left to right.

The justification for selecting AdaBoost.OC and AdaBoost.M2 for comparisons is because they have been shown to clearly perform better than other contending methods like naive ECOC and Bagging in experiments carried out by Schapire [140]. Experiments on similar datasets by Jin and Zhang [72] also indicated a stronger performance by AdaBoost.OC over ECOC methods. Both AdaBoost.OC and AdaBoost.ECC are still widely used [133] and AdaBoost.ECC has been until recently a subject of research and extension [154]. In addition, the experimental results from Li [86] have shown that the performance strength of AdaBoost.ECC is greater than OAO and OAA approaches which would also have been suitable candidates for comparisons. In addition, Guruswami and Sahai [58] claim that AdaBoost.ECC is both theoretically and experimentally superior than AdaBoost.OC.

For datasets with both training and test sets, experiments were executed ten times; otherwise, 10-fold cross-validation was employed in conjunction with 10 training repetitions for a total of 100 runs. All results were averaged and in the presence of randomness, standard error was reported. Four different cascaded.DP classifiers were trained for each dataset. These classifiers were trained with different parameter settings for the value $\Phi$ that determines the maximum number of weak classifiers per multiclass node. The value for $\Phi$ was set to 5, 10, 25 and 50. The training terminated once a layer for each class label was constructed.

For a fair comparative analysis, ECC, OC and M2 classifiers were also trained using weak inducers in the form of decision stumps. The terminating condition for these algorithms was a predetermined maximum number of boosting iterations. This upper limit was in line with experiments in [35, 86] and is shown alongside the test error rates in the results section[6].

The AdaBoost.M2 algorithm requires that the entire search space of all possible combinations of class labels be examined for all feature types at every boosting iteration in order to find the lowest pseudoloss combination. Thus, the complexity of an exhaustive search becomes $F \times 2^k$, where F is the number of features and $k$ is the number of class labels[7]. This is the primary performance bottleneck for AdaBoost M2; however, since the number of class labels is usually not greater than 10, it is feasible to perform an exhaustive search over all possible class combinations. In the case of the Letter dataset though, there are 26 class labels which makes the comprehensive evaluation over the complete search space unfeasible. In order to achieve practical training runtimes, the total search space of functionally unique set combinations for the Letter dataset was minimized from the total possible size of 33,554,431 to 1,000[8] randomly sampled class combinations.

[6]Experiments by Schapire [140], Sun et al. [154] set the ensemble size for similar datasets to 500; however, they used stronger inducers than the decision stumps here and so were able to achieve faster convergences in fewer boosting rounds.

[7]While the exhaustive number of possible permutations is $2^k$, not all permutations form sets which are functionally unique. Due to this, the total number of unique set combinations can be reduced. Calculating this figure becomes a combinatorial problem and can be calculated as $\frac{k!}{1!(k-1)!} + \frac{k!}{2!(k-2)!} + , ..., + \frac{k!}{2(\frac{k}{2})!(\frac{k}{2})!}$ up to $\frac{k}{2}$ class labels when $k$ is an even number. When $k$ is odd, then the last term is not used and the terms up to to $\frac{k-1}{2}$ class labels are summed.

[8]The considerable reduction of the search space was justified on the basis that a comparative training runtime to other algorithms was required as a major component of these experiments. In addition, the assumption was made which expected that a set of sufficiently diverse colourings would be generated, with

For M2, a fast method for determining all plausible class label combinations was devised using bitwise operators. Firstly, each class label was assigned a value $\forall i \in \{1..k\}$ : $(y_i, i)$. Each value $j \rightarrow 2^k$ was then cast into an unsigned integer value. The component bits $\{1..k\}$ of the unsigned value $j_{1..k}$ were then associated with $(y_i, i)$ labels. These could then be efficiently extracted at each iteration through a shift operator. The plausible set $\mu_t$ was constructed consisting of class labels $y$ where the bit values $j_{i..k} \rightarrow (y_i, i)$ and the bit $j_i = 1$.

The same basic approach was used to determine the colouring $\mu_t$ for AdaBoost.OC and ECC with slight modifications. Schapire [140] showed that the size of the colouring set $\mu_t$ needs to be $\frac{k}{2}$. This time an array $\vec{a}$ was generated consisting of $n \in \{1..2^k\}$ possible number of integer values, whose number of positive bits equaled $\frac{k}{2}$. In the case that there was an odd number of class labels, then the value with positive bits $\frac{k}{2} + 1$ was selected. At training, a new random number $j^n$ was generated at each boosting round which determined the colouring and from which $\frac{k}{2}$ number of class labels were extracted as before.

It was not straightforward devising a fair and a completely accurate method for comparing the training and execution runtimes of cascaded and their associated single-layer classifiers. This was so since both runtimes are directly affected by the size of a classifier's ensemble and the point on their receiver operating curves (ROC) that they are being compared to. In the case of cascaded.DP classifiers, the ensemble size is not predetermined prior to training, while for the single-layer classifiers it is, since very low training errors on complex datasets with weak learners are often hard to attain. Since determining the optimal criterion for halting the training of single-layer classifiers is subjective, it challenges the validity of comparing classifiers only on their absolute training and execution runtimes and gives an occasion to unfairly prejudice the runtime performance of the control algorithms if their predetermined ensemble size is unnecessarily large.

Consequently, two values were used to provide a balanced comparison between the pairs of classifiers. One value reports a factor by which one algorithm trains or executes a *single weak classifier* faster than the other. This figure provides fairness to single-layer algorithms which could have been trained with fewer boosting iterations in order to attain a comparable generalization. The second value provides the same kind of comparison, however this time from the perspective of the entire classifier in order to provide a balanced view. Therefore, conclusive statements about which algorithm performs more efficiently will only be warranted if a superior performance is measured on both metrics.

### 7.3.1 Multiclass Cascade Implementation

The final layer-nodes and auxiliary nodes within each layer were implemented as binary-class PSL cascades. The auxiliary nodes however, consisted of a truncated PSL cascade made up of only one layer. This single layer was designed to train and detect samples in an inverse approach in respect to positive and negative samples compared to the original. This meant that a positive detection required an unanimous vote from all nodes.

The justification for implementing the two kinds of nodes in this approach is as follows. Firstly, the target hit and false positive rates for each layer had to be close to zero. This

the capability of producing classifiers that are adequately accurate and representative of the M2 algorithm.

was determined since one of the drawbacks of ensemble cascades is that errors in preceding layers propagate to succeeding ones [10]. This means that a false positive detection at any point in a cascade cannot be rectified by latter layers and consequently the accuracy can be expected to suffer.

The auxiliary nodes were not implemented as complete cascades since a zero training training error rate was unnecessary. As mentioned earlier, the primary goal of auxiliary nodes was to attain negligibly low false positive rates in respect to the class label that is assigned to a multiclass node, while high hit rates were secondary. Since one limitation of AdaBoost is its ability to only minimize a quantity related to the total classification error [175] instead of explicitly being able to minimize either the false positive or false negative rates, a truncated PSL-cascade was required with an explicit ability to minimize the false positive detections.

Secondly, this PSL like principles were used to implement these nodes instead of monolithic ensembles because it has been shown [156] that the naive approach requires much greater ensembles to reach the specified targets. There is also no guarantee that a convergence to these targets would occur, while the penalty is protracted runtimes and an inflated ensemble.

## 7.4 Results

The analysis of the experiments is divided into three parts. The training phase examines the convergence patterns, runtime durations and test accuracy in respect to training runtimes. The following section focuses primarily on the generalization characteristics of the classifiers while the subsequent section analyzes the execution runtime properties[9].

### 7.4.1 Training Phase

Figure 7.3 shows training convergence patterns from a selection of datasets. These figures represent the typical trends observed across all datasets, while the remaining graphs are located in Appendix C.1. Past research has already theoretically and empirically shown that boosting algorithms exponentially reduce the upper bound on the training error [45] and this decrease is seen in the figures for the M2, OC and ECC algorithms. In comparison to the cascaded.DP classifiers, the first observation is that the exponential rate of decrease in the training error by the control algorithms is not matched by the cascaded.DP classifiers in the initial boosting rounds. The fastest convergence rates are in most cases recorded by the ECC algorithm and followed by the OC and M2 algorithms.

The training convergence graphs in Figure 7.4, exemplify an additional trend which recurs on a number of datasets. On these datasets, though the cascaded.DP classifiers still require a higher initial costs for lowering the training error, they are able to catch up to the control algorithms. In some cases the cascaded classifiers attain zero training error rates at significantly lower costs in boosting iterations than the OC and M2 algorithms, while convergence rates for the latter algorithms notably slow down in the convergence and are unable to attain the same training error rates.

[9]Parts of this research have been published in [161].

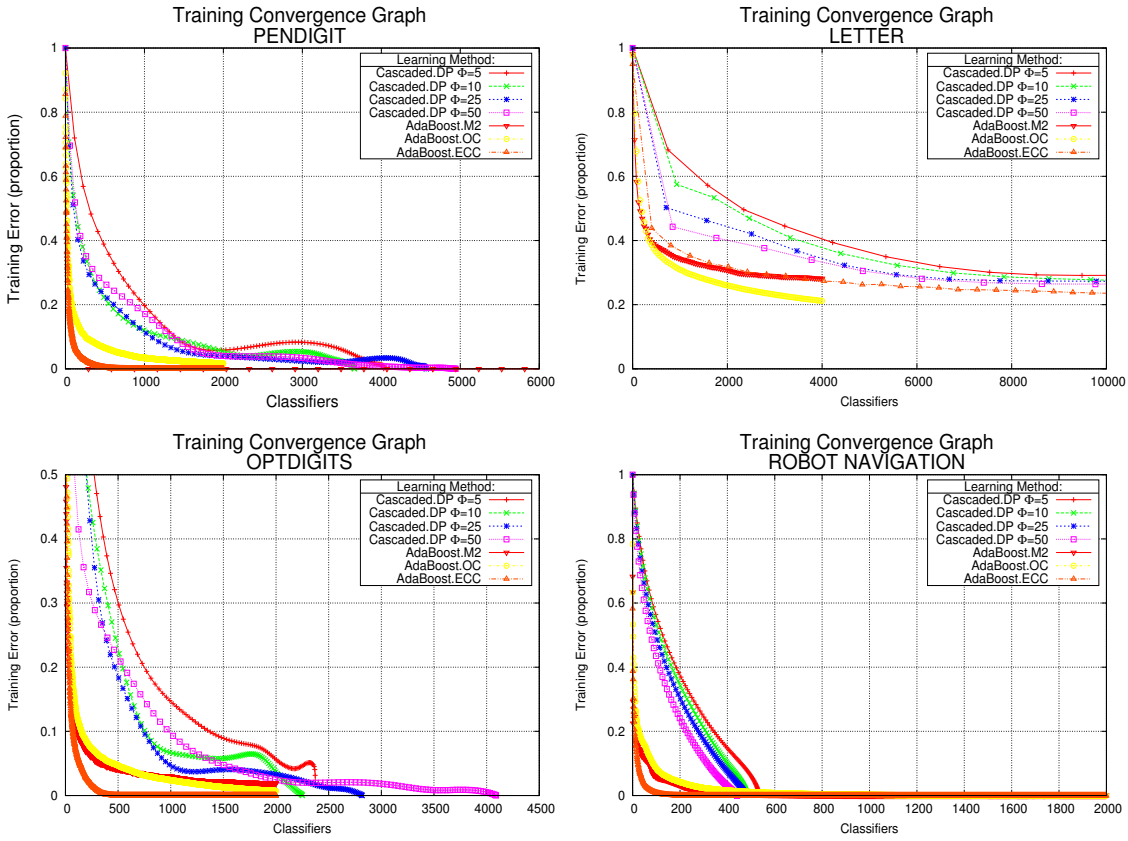



Figure 7.3: Training error convergence graphs for the Pendigit, Letter, Optdigits and Robot Navigation datasets.

Although, the training error is typically an optimistically biased measure of a classifier's generalization ability, in the majority of cases the cascaded classifiers attained a zero training error that was only matched by ECC classifiers. Very low training error rates can sometimes serve as evidence of over-training leading to over-fitting and memorizing dataset patterns [134]. However, this is unlikely to be the case with the cascaded.DP classifiers for two reasons. Firstly, because of its 'separate-and-conqueror' approach, low training error can always be expected due to continuous reductions of the problem. Secondly, because the dataset is altered after each node, the learning task changes. Combined with a weak inducer, this leaves a narrow window of opportunity for the memorization of patterns to occur.

The low training error rates of the cascaded classifiers are often achieved while generating significantly smaller ensembles than those of OC and M2 classifiers. This was observed on Vowel, Segmentation, Vehicle, Glass, Iris and Factors datasets. However, there are domains on which the cascaded classifiers are susceptible to generating more complex ensembles than all the control algorithms. This was seen in Pendigit, Letter, Satimage, Optdigits, Yeast and Morphological datasets. The general characteristics of the latter group of datasets reveals that they comprise a comparatively larger number of class labels. Since the number of class labels within a dataset determines the total number

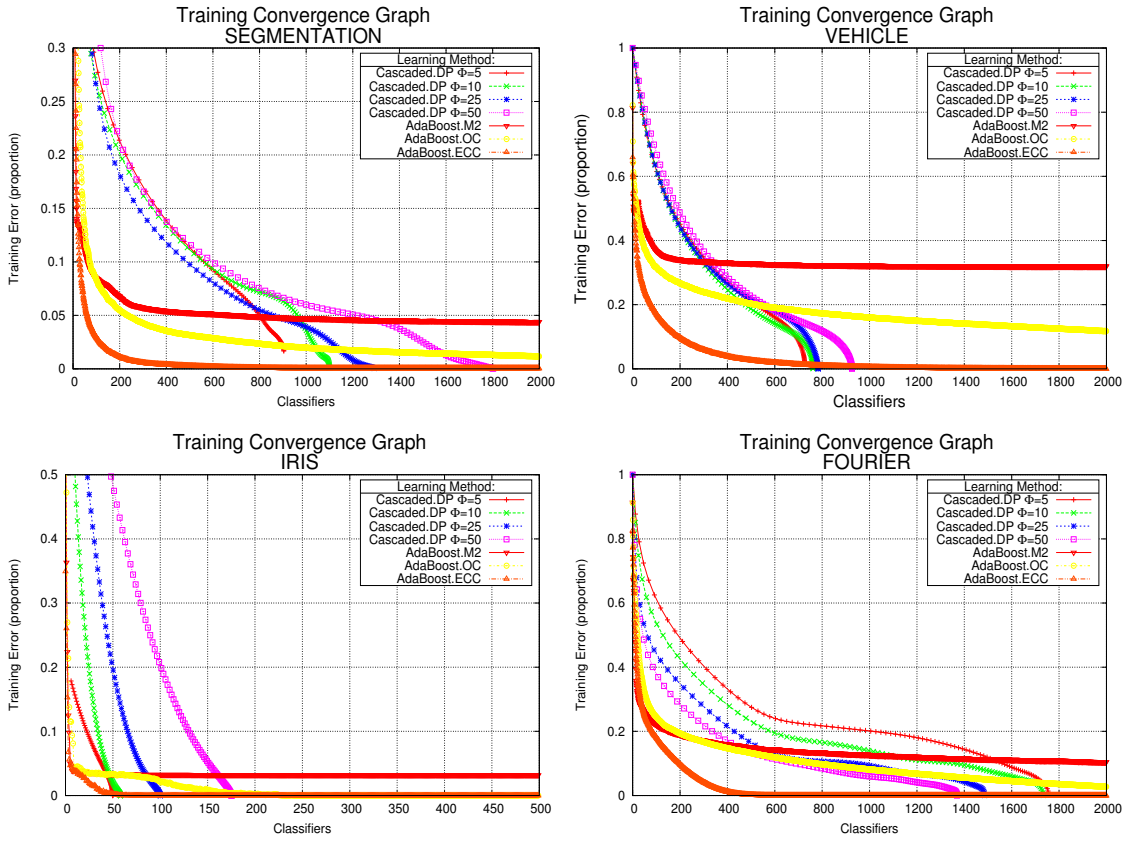



Figure 7.4: Training error convergence graphs for the Segmentation, Vehicle, Iris and Fourier datasets.

of layers for cascaded classifiers, this can be taken as evidence that cascaded classifiers are prone to producing larger ensembles with increasing numbers of class labels.

There is no consistent pattern amongst the cascaded classifiers allowing for definitive statements to be made as to what value of $\Phi$ yields best convergences. There is evidence that on datasets with larger class label numbers, the cascaded classifiers with larger node sizes generate more complex ensembles. This can be seen in Letter, Pendigit, Vowel and Optdigit datasets which range from 10 to 26 class labels. However, Factor, Fourier and Morphological datasets also comprise 10 class labels but fail to indicate that large node sizes necessarily generate overly complex ensembles.

All control algorithms produced monotonic convergence graphs testifying to their effective and efficient learning abilities. Slight non-monotonic patterns were observed by cascaded classifiers on Pendigit, Satimage, Vowel and Shuttle datasets. On these datasets, some cascaded classifiers underwent several phases of increasing training error rates before eventually resuming a reduction of the overall training error.

The periodic non-monotonic phases can be attributed to the separate-and-conquer process which focuses learning and refining a classifier for a single class label at a time. Occasionally, while the error improves for a selected class label for a given layer, the intermediate overall training error for the remaining class labels may increase due to some

class boundary configurations and may consequently be higher than that of the previous layers. Therefore the learning of a specific class label can become overshadowed by the total training error of the remaining classes and can thus not be accurately reflected by the total error figures.

**Training Runtimes**

The comparative classifier training runtime-performances are recorded in Table 7.1[10]. The table presents the factor by which the cascaded.DP classifiers have generated classifiers faster than the control algorithms on each dataset. The overall trends are summarized using the arithmetic mean and the standard deviation as well as the geometric mean. Arguably, in this instance the geometric mean provides a more meaningful value since the data is ratio-scale and due to the ability of the geometric mean to act as a more accurate measure of location when the data are highly skewed to the right [193].

The pairwise comparisons in the table indicate that:

1. The contrast between the runtimes of the cascaded.DP and M2 classifiers is immediately noticeable. The factor by which the cascaded algorithm generates classifiers faster than M2 ranges up to thousand-fold increases on datasets that are large in respect to class label numbers such as Letter, Pendigits, Optdigits and Vowel. The slow execution runtimes of M2 are not surprising because it learns a weak classifier for every possible permutation of class combinations at each boosting round and retains only the one with the lowest pseudo loss error.

2. The cascaded.DP algorithm has also consistently generated faster training runtimes than OC and ECC on both the absolute runtimes per ensemble and on the per weak-classifier metric. Of the two algorithms, faster training runtimes were recorded by ECC over OC in part due to the fact that the ECC omits calculating the pseudo error at each boosting step of learning and thus has a lower overhead.

3. The summary of each table in the form of the arithmetic and the geometric means indicates that the average rate by which the cascaded.DP classifiers outperform the control algorithms lessens with the increase in the training parameter $\Phi$. This is expected as will be shown by later figures, because higher values of $\Phi$ result in the generation of significantly larger ensembles for the cascaded.DP classifiers. For classifiers with largers values of $\Phi$, the consequence is that a much greater portion of the ensemble becomes composed of multiclass rather than binary stump weak classifiers, which are more computationally intensive to compute.

4. For $\Phi = 25$, the performances of some cascaded.DP classifiers are reduced close to parity in respect to ECC and OC classifier runtimes. On the Glass dataset, the

[10]For Iris, Factors, Robot Navigation, Waveform, Waveform Noise Pageblocks and Shuttle datasets, possible objections can be raised regarding the predetermined total size of the ensemble that was set for the control algorithms. It can be claimed that in some instances only small improvements in their convergences occur after a certain number of boosting rounds and therefore the total size of the ensembles could have been smaller since additional training prejudices the training and the execution runtimes of these algorithms. However, some of these datasets consists of biased-class distributions, for which the total training error metric does not adequately convey improvements for the minority classes. In addition, even once the zero training error has been achieved, continued boosting training has been shown to not lead to overtraining but to instead often result in an improved generalization.

Table 7.1: Training runtimes expressed as a factor by which the Cascaded.DP classifiers train a single weak classifier and an entire ensemble faster than the control algorithms on each dataset. Value 1.0 represents parity. Values below 1.0 signify a slower runtime.

| Cascaded.DP $\Phi = 5$ vs. | OC | | ECC | | M2 | |
|---|---|---|---|---|---|---|
| | per classifier | per ensemble | per classifier | per ensemble | per classifier | per ensemble |
| letter | 34.4 | 3.4 | 22.7 | 2.2 | 31440.4 | 1541.6 |
| pendigits | 10.5 | 5.1 | 7.7 | 3.7 | 10493.2 | 5048.5 |
| satimage | 6.2 | 4.8 | 5.4 | 4.1 | 580.1 | 442.7 |
| vowel | 4.1 | 7.5 | 2.7 | 4.9 | 7022.6 | 12664.8 |
| optdigits | 5.0 | 4.2 | 4.3 | 3.6 | 8357.0 | 7040.5 |
| segmentation | 5.9 | 14.5 | 4.5 | 11.2 | 803.3 | 1973.8 |
| vehicle | 3.9 | 10.7 | 3.3 | 9.0 | 56.7 | 157.2 |
| glass | 2.2 | 3.5 | 1.6 | 2.5 | 142.6 | 230.8 |
| iris | 2.5 | 25.1 | 2.0 | 20.1 | 21.1 | 210.9 |
| factors | 2.2 | 5.6 | 2.1 | 5.3 | 3950.5 | 10234.5 |
| fourier | 3.3 | 3.3 | 2.9 | 2.9 | 5062.8 | 5114.0 |
| robot navigation | 3.5 | 12.0 | 3.2 | 10.8 | 54.3 | 185.7 |
| waveform | 3.5 | 5.3 | 3.2 | 4.8 | 22.7 | 33.9 |
| waveform noise | 3.6 | 6.9 | 3.3 | 6.4 | 24.1 | 46.9 |
| yeast | 11.0 | 3.0 | 7.0 | 1.9 | 7539.5 | 2041.6 |
| morphological | 12.5 | 3.8 | 7.8 | 2.3 | 7350.7 | 2206.4 |
| page blocks | 7.2 | 9.1 | 5.6 | 7.1 | 178.2 | 225.8 |
| shuttle | 3.2 | 19.9 | 2.4 | 14.5 | 395.9 | 2421.6 |
| Mean | 6.93 | 8.2 | 5.09 | 6.53 | 4638.66 | 2878.94 |
| Std.Dev. | 7.53 | 6.2 | 4.81 | 4.9 | 7607.14 | 3747.46 |
| Geometric mean | 5.1 | 6.6 | 4.0 | 5.2 | 731.3 | 908.1 |

(a) Cascaded.DP $\Phi = 5$ training runtime comparison.

| Cascaded.DP $\Phi = 10$ vs. | OC | | ECC | | M2 | |
|---|---|---|---|---|---|---|
| | per classifier | per ensemble | per classifier | per ensemble | per classifier | per ensemble |
| letter | 23.0 | 2.3 | 15.2 | 1.5 | 21073.6 | 1067.6 |
| pendigits | 7.6 | 4.1 | 5.6 | 3.0 | 7600.9 | 4084.3 |
| satimage | 5.2 | 4.0 | 4.5 | 3.4 | 481.3 | 369.1 |
| vowel | 2.8 | 4.1 | 1.9 | 2.7 | 4800.6 | 7018.4 |
| optdigits | 3.5 | 3.1 | 3.0 | 2.7 | 5890.2 | 5238.0 |
| segmentation | 4.3 | 8.7 | 3.3 | 6.7 | 580.1 | 1186.2 |
| vehicle | 3.6 | 9.0 | 3.0 | 7.6 | 52.0 | 132.1 |
| glass | 1.9 | 2.6 | 1.3 | 1.9 | 123.1 | 171.0 |
| iris | 1.9 | 14.6 | 1.5 | 11.7 | 15.9 | 122.7 |
| factors | 1.6 | 3.2 | 1.5 | 3.1 | 2882.0 | 5899.7 |
| fourier | 2.4 | 2.2 | 2.1 | 1.9 | 3676.6 | 3368.4 |
| robot navigation | 3.1 | 11.0 | 2.8 | 9.9 | 47.6 | 170.0 |
| waveform | 4.3 | 5.9 | 4.0 | 5.4 | 27.7 | 37.8 |
| waveform noise | 3.6 | 6.8 | 3.3 | 6.3 | 24.2 | 46.1 |
| yeast | 9.6 | 2.6 | 6.2 | 1.7 | 6581.6 | 1788.7 |
| morphological | 10.4 | 3.0 | 6.5 | 1.9 | 6136.7 | 1789.9 |
| page blocks | 5.2 | 5.6 | 4.0 | 4.4 | 129.3 | 139.8 |
| shuttle | 2.3 | 10.5 | 1.7 | 7.7 | 280.9 | 1285.4 |
| Mean | 5.35 | 5.76 | 3.96 | 4.64 | 3355.8 | 1884.18 |
| Std.Dev. | 5.1 | 3.62 | 3.23 | 3.05 | 5204.02 | 2255.45 |
| Geometric mean | 4.1 | 4.8 | 3.2 | 3.8 | 584.9 | 665.6 |

(b) Cascaded.DP $\Phi = 10$ training runtime comparison.

| Cascaded.DP $\Phi = 25$ vs. | OC | | ECC | | M2 | |
|---|---|---|---|---|---|---|
| | per classifier | per ensemble | per classifier | per ensemble | per classifier | per ensemble |
| letter | 11.9 | 1.1 | 7.8 | 0.7 | 10846.3 | 499.2 |
| pendigits | 4.6 | 1.9 | 3.3 | 1.4 | 4540.7 | 1926.1 |
| satimage | 4.0 | 2.9 | 3.4 | 2.5 | 368.3 | 266.9 |
| vowel | 1.7 | 1.7 | 1.1 | 1.1 | 2951.6 | 2831.3 |
| optdigits | 2.1 | 1.5 | 1.8 | 1.3 | 3523.5 | 2501.6 |
| segmentation | 2.5 | 3.9 | 1.9 | 3.0 | 340.7 | 525.3 |
| vehicle | 2.8 | 7.0 | 2.4 | 5.8 | 41.1 | 101.7 |
| glass | 1.2 | 1.0 | 0.9 | 0.7 | 78.0 | 65.9 |
| iris | 1.3 | 5.3 | 1.0 | 4.2 | 10.9 | 44.1 |
| factors | 1.1 | 1.4 | 1.1 | 1.3 | 2037.1 | 2540.0 |
| fourier | 1.4 | 1.0 | 1.2 | 0.9 | 2120.0 | 1623.9 |
| robot navigation | 2.4 | 7.5 | 2.2 | 6.7 | 37.5 | 115.1 |
| waveform | 3.9 | 5.2 | 3.5 | 4.8 | 24.8 | 33.6 |
| waveform noise | 3.2 | 6.1 | 3.0 | 5.7 | 21.8 | 41.2 |
| yeast | 6.7 | 1.6 | 4.3 | 1.0 | 4590.0 | 1100.2 |
| morphological | 6.5 | 1.8 | 4.1 | 1.1 | 3843.8 | 1058.0 |
| page blocks | 4.3 | 4.3 | 3.4 | 3.3 | 107.4 | 106.6 |
| shuttle | 1.4 | 3.9 | 1.0 | 2.8 | 172.2 | 474.3 |
| Mean | 3.5 | 3.28 | 2.64 | 2.69 | 1980.87 | 880.83 |
| Std.Dev. | 2.7 | 2.17 | 1.73 | 1.98 | 2802.81 | 979.26 |
| Geometric mean | 2.8 | 2.6 | 2.2 | 2.1 | 398.4 | 361.2 |

(c) Cascaded.DP $\Phi = 25$ training runtime comparison.

| Cascaded.DP $\Phi = 50$ vs. | OC | | ECC | | M2 | |
|---|---|---|---|---|---|---|
| | per classifier | per ensemble | per classifier | per ensemble | per classifier | per ensemble |
| letter | 6.1 | 0.5 | 4.0 | 0.3 | 5570.0 | 235.2 |
| pendigits | 1.6 | 1.0 | 1.2 | 0.7 | 1627.8 | 952.2 |
| satimage | 2.4 | 1.8 | 2.1 | 1.5 | 226.6 | 165.1 |
| vowel | 1.4 | 0.7 | 1.0 | 0.5 | 2457.9 | 1202.2 |
| optdigits | 0.6 | 0.8 | 0.5 | 0.6 | 953.2 | 1260.0 |
| segmentation | 0.9 | 1.8 | 0.7 | 1.4 | 119.8 | 245.5 |
| vehicle | 0.5 | 4.2 | 0.4 | 3.6 | 7.0 | 61.8 |
| glass | 1.0 | 0.5 | 0.7 | 0.4 | 63.8 | 32.4 |
| iris | 11.0 | 2.1 | 8.8 | 1.6 | 92.4 | 17.3 |
| factors | 1.3 | 0.7 | 1.2 | 0.7 | 2331.9 | 1288.7 |
| fourier | 0.4 | 0.5 | 0.4 | 0.5 | 656.4 | 838.3 |
| robot navigation | 1.8 | 4.5 | 1.6 | 4.1 | 27.9 | 69.8 |
| waveform | 2.1 | 3.9 | 1.9 | 3.6 | 13.4 | 25.2 |
| waveform noise | 21.3 | 5.1 | 19.8 | 4.7 | 144.1 | 34.4 |
| yeast | 4.4 | 0.9 | 2.8 | 0.6 | 3013.2 | 616.8 |
| morphological | 3.2 | 0.9 | 2.0 | 0.6 | 1895.1 | 521.6 |
| page blocks | 3.0 | 2.7 | 2.3 | 2.1 | 74.9 | 66.0 |
| shuttle | 1.5 | 2.5 | 1.1 | 1.9 | 187.3 | 310.6 |
| Mean | 2.29 | 1.95 | 1.75 | 1.62 | 1216.43 | 441.29 |
| Std. Dev. | 1.39 | 1.54 | 0.92 | 1.42 | 1540.5 | 466.9 |
| Geometric mean | 2.0 | 1.4 | 1.5 | 1.1 | 279.8 | 197.4 |

(d) Cascaded.DP $\Phi = 50$ training runtime comparison.

ECC algorithm records faster performances on both the comparisons of ensemble-to-ensemble training runtimes and weak-classifier-to-weak-classifier runtimes. On the remaining datasets such as Letter, Fourier and Waveform Noise, though the training runtimes of the ECC were slightly faster than the cascaded.DP in comparing the entire ensemble runtimes, this was in each case balanced out by the faster generation of each weak classifier by the cascaded.DP algorithm.

5. At $\Phi = 50$, the clear performance advantage of the cascaded classifiers over the OC and ECC disappears though the overall averages are still comparable and slightly favourable on some metrics towards the cascaded classifiers. The datasets on which the control algorithms clearly outperform the cascaded algorithm by registering faster rates on both metrics, are the Vowel, Optdigits, Glass and Fourier.

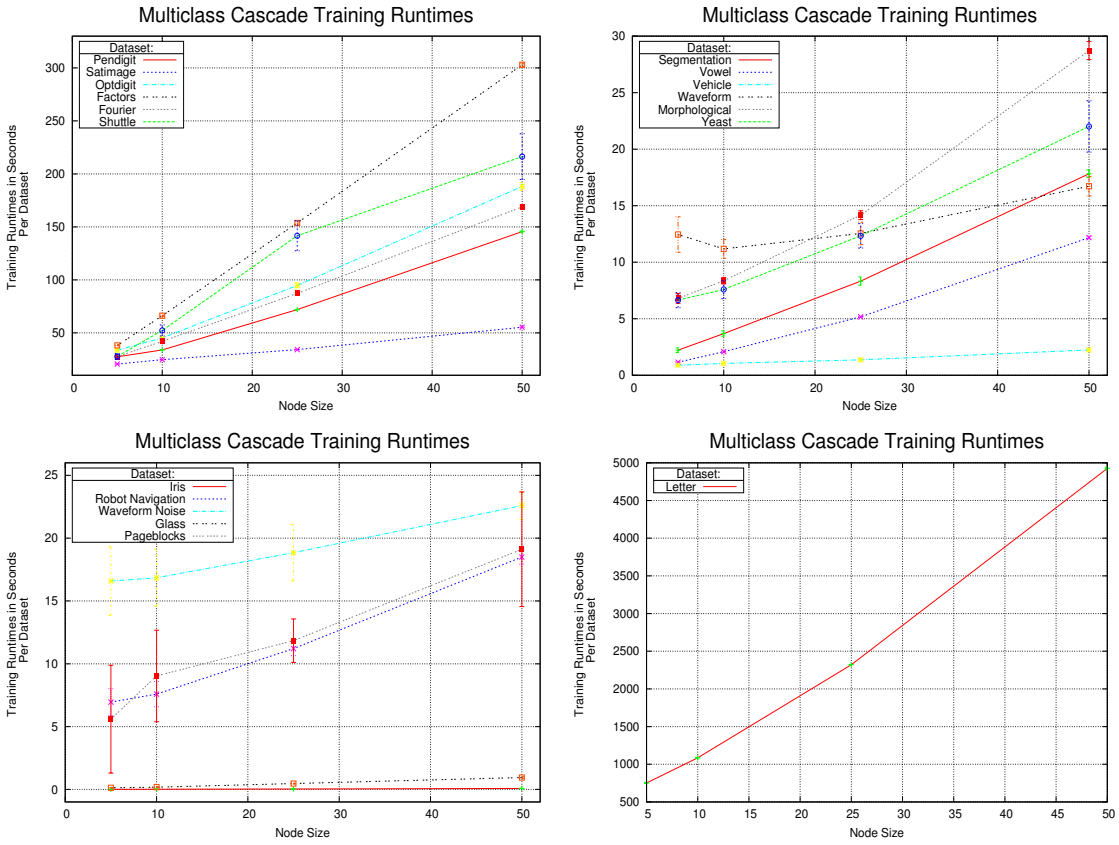



Figure 7.5: The effect on training runtimes with the increase in $\Phi$ on the cascade multiclass algorithm.

Figure 7.5 examines in more detail the training runtimes of the cascaded algorithm. In these figures the relationship between the node sizes and the training runtimes is highlighted. The relationship between these two variables can generally be summarized as being linear, with the classifiers trained on the Waveform dataset being an exception. The rates of increase in the runtimes vary significantly between datasets. While the training runtimes are only moderately affected by increasing node sizes for the Iris, Glass,

Waveform and Waveform Noise datasets; a much greater effect on the training runtimes is observable on the Satimage, Shuttle, Letter and Optdigit datasets. It can be seen from Table 3.2 that the classifiers which record the strongest rates of increase tend to have a larger number of both class labels and features in a dataset[11].

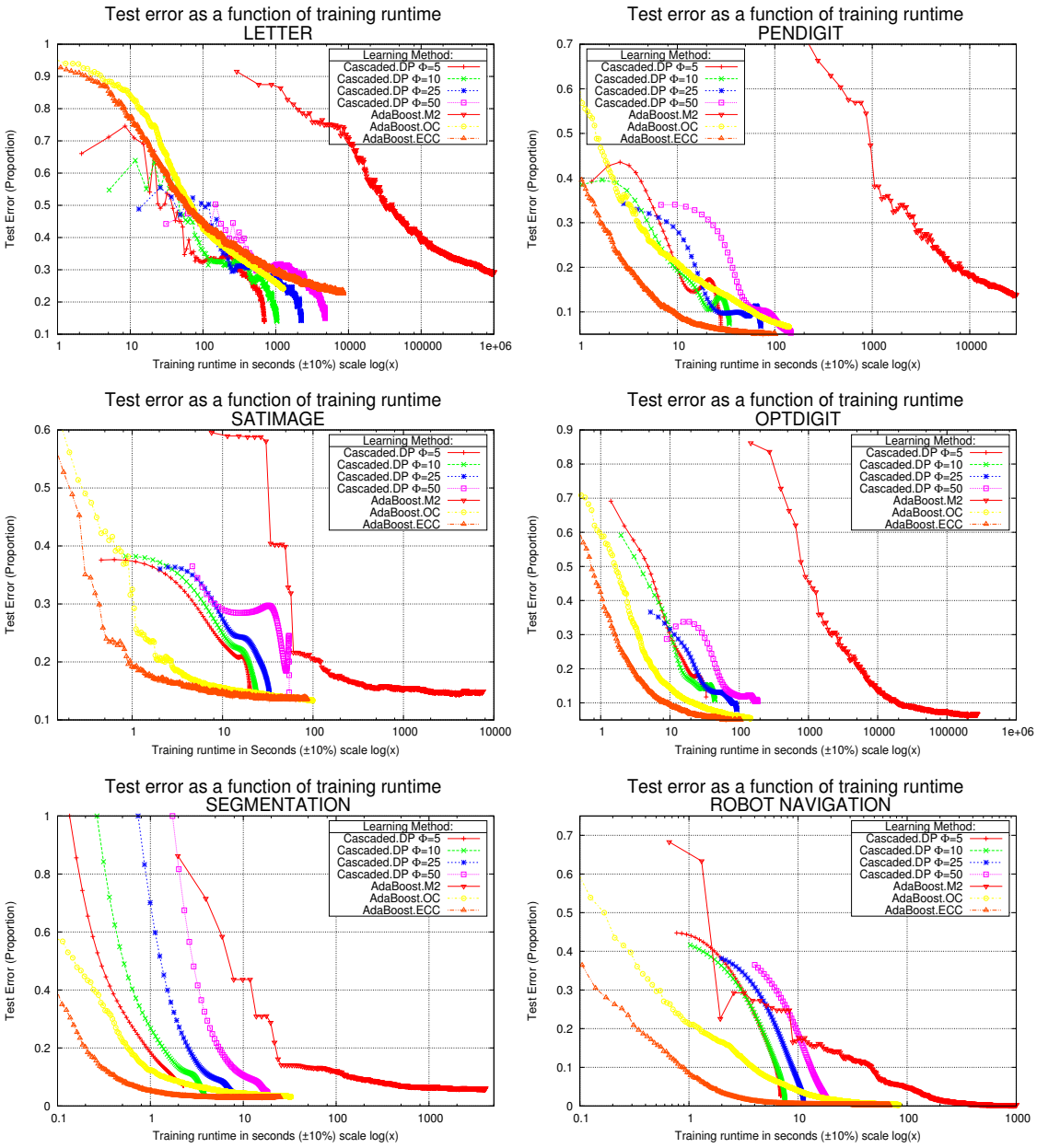



Figure 7.6: Classifier accuracy on test data as a function of training runtime.

[11]Attempts were made to fit a function to the data from all sets and to show that there is a strong correlation between these variables and the training runtimes. Though these variables bear the strongest influence on the runtimes, the efforts did not yield fruit as there are other hidden variables that also influence the rate of increase in the runtimes as the node complexity increases. These variables are the size of the datasets as well as the complexity and the separateness of the classes. While the first is already quantified, the second is not straightforward to quantify.

It can also be observed from the error bars in the figures that some datasets show considerably larger variations in training runtimes. This is because for some datasets, a test set was provided while for others, the 10-fold cross validation approach was used. All classifiers trained on datasets using 10-fold cross validation exhibited a more pronounced variation as the error bars show, which represents the sensitivity of runtimes to the 10 datasets composed of different sample distributions and class boundaries. The smaller variation of other classifiers reflects more the actual sensitivity of the measurements for the runtimes, since the dataset was the same on each run.

Finally, the efficiency of the training phases is now considered from the perspective of attained accuracy as a function of the training runtime. This can be seen in Figure 7.6, which plots on the x-axis the cost as training runtime on a logarithmic scale and the achieved accuracy on unseen data on the y-axis. A selection of results from various datasets is shown while the remainder is located in the Appendix C.2.

Graphs for the Satimage, Optdigit and Segmentation datasets are overwhelmingly representative of the overall trends seen across all 18 datasets. The ECC algorithm consistently produced the best test error results for the cost represented as the training runtime. This was followed by the OC algorithm which did not match the ECC generalization ability in the initial stages, but was however able to catch up in the latter stages of training.

Both algorithms in the same manner dominated the cascaded classifiers in the initial stages, but the cascaded classifiers likewise demonstrated their ability to match the test error rates of ECC and OC in the latter stages. This pattern was observed on the Satimage, Vowel, Segmentation, Vehicle, Glass and Robot Navigation datasets where at least one of the cascaded classifiers eventually achieved some of the lowest test error rates. The results from the Letter dataset were of significant interest. This particular dataset comprises of the largest number of class labels and training samples. The cascaded algorithm generated an inordinate number of weak classifiers on this problem set; however, the cascaded classifiers with the smallest value of $\Phi$ produced the best generalization results for a given training runtime cost.

Amongst the cascaded classifiers themselves, a clear pattern emerged which indicated that the classifiers with smaller values of $\Phi$ dominated the performances of the classifiers with larger nodes. It was also observable that that there was no obvious correlation between larger node sizes and higher accuracy rates for cascaded classifiers.

There are two reasons why the ECC and OC classifiers have displayed the tendency to produce a stronger classifier generalization in the initial rounds of training. The first is that both algorithms focus equally on learning all class labels at each boosting round. By dichotomizing the problem into plausible sets, they are able to construct the columns of the error-correcting coding-matrix at each round, which enables the differentiation between all classes in a stepwise manner. The cascaded method refines the learning problem into smaller sub-tasks. In doing so, the cascaded method instead focuses on correctly learning one class label at a time and uses considerable resources to refine the classification for one class label. This results initially in lower generalization rates. However, as seen from most curves describing the generalization patterns of the cascaded classifiers, they rapidly decrease the error rates particularly in the latter phases, since the learning task becomes increasing simpler. The patterns for the OC and ECC classifiers on the other hand slow

down and plateau in the latter stages, since further refinement of the decision boundaries between class labels becomes increasingly difficult for these algorithms.

On the other hand, the M2 algorithm consistently demonstrated higher training runtime costs than all the other classifiers for a given accuracy. The high computational complexity became even more evident on datasets containing 10 or more class labels as seen on Letter, Pendigit, Vowel, Optdigit, Factors, Fourier, Yeast and Morphological datasets. On these sets, the equivalent accuracy measure required training runtimes that were often at least an order of magnitude longer than those of OC and ECC.

### 7.4.2 Generalization

The generalization patterns for classifiers on unseen data from a selection of datasets are shown in Figure 7.7. These figures encapsulate the main trends, while the remaining graphs can be found in Appendix C.3. The figure plots the overall test error rates as a function of the number of weak classifiers executed in an ensemble for each dataset.

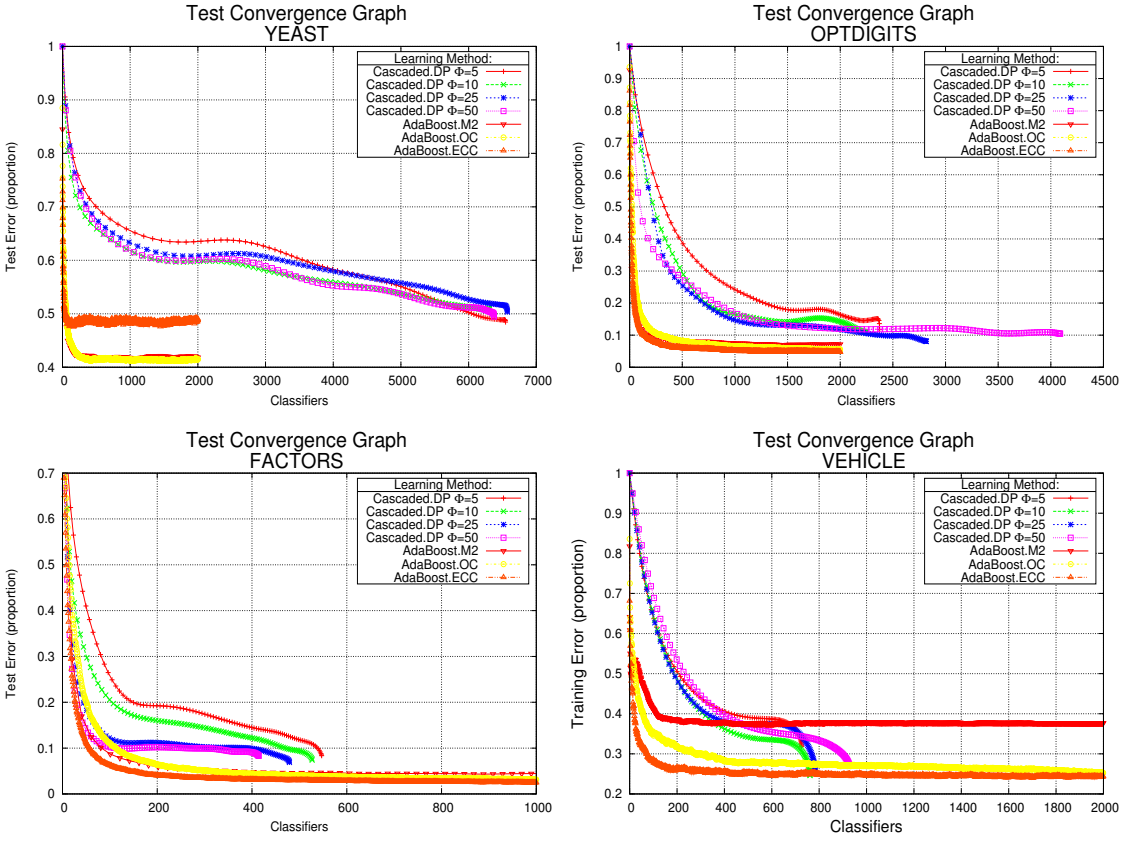



Figure 7.7: Test error convergence graphs.

These figures closely resemble the patterns of the training phases from Figures 7.5 in a number of respects. Firstly, the dominance of the ECC, OC and M2 classifiers over the cascaded classifiers in the early phases is repeated. This occurs almost without exception across all datasets. In the overwhelming majority of datasets, the ECC classifiers produce faster convergences to their optimal generalization rates than the OC classifiers. The

Factors, Vehicle and Optdigit datasets represent these patterns. This is in line with the research findings of its authors [58]. However, ECC has not always attained the best generalization results over OC as show on the Yeast dataset. This is also finds support in the findings of [154].

In respect to the generalization of the cascaded classifiers, the results across the 18 datasets have shown three main patterns:

1. The cascaded classifiers frequently generated smaller ensembles than the control algorithms and have succeeded in most cases to match their accuracies. This was observed in $\frac{7}{18}$ of the cases. The results from the Vehicle dataset are representative of such patterns.

2. The cascaded classifiers produced considerably larger ensembles than the control algorithms. In the majority of these cases, even with larger ensembles, the cascaded classifiers did not record accuracy rates that matched those of the control algorithms. This occurred in $\frac{6}{18}$ of the datasets. The graphs for the Yeast and Optdigits datasets are representative of these patterns.

3. In $\frac{4}{18}$ of the cases which are represented by the Factors dataset, the cascaded classifiers produced smaller ensembles than the control algorithms but were, however, unable to match the generalization of the control algorithms in general.

The point on which the test error graphs Figures 7.7 and the training error graphs to a large extent do not agree is found in the discrepancy between the final error rates of the respective graphs. As mentioned previously in the training phase analysis section, most cascade classifiers achieved a zero training error rate. However, in many cases the final test error rates were well above the zero training error. Although, the training error is always a more optimistic measure of a classifier's final generalization rate; the expected test error is usually anticipated to be in the vicinity of the final training error rate. This pattern can be observed between the training and error rates of the OC and M2 classifiers which did not achieve a zero training error as often as ECC. Usually the magnitude of this discrepancy indicates the degree to which the learning algorithm has over-fitted or memorized the data.

Nonetheless, when observing the test error curves of the cascaded classifiers, there is a lack of evidence to suggest that overfitting has taken place. Although there are non monotonic patterns found in some cascaded classifiers, particularly on the Pendigit, Satimage, and Morphological datasets, the final test error rates always finish with figures that were lower than the point at which non-monotonic and seemingly over-fitting behaviour begun.

**Balanced Class-distribution Datasets**

While the graphs in Figures 7.4 and 7.6 are useful for shedding light on the runtime and complexity characteristics of classifiers, ultimately none of these measures are of any importance if the final accuracy of a classifier is poor. Table 7.2 records the accuracy of all classifiers on datasets with unbiased class distributions.

The table lists two accuracy measures. The first is the overall error as a proportion of all classifications, together with the average standard error. The second measure is the

g-mean which takes into consideration the accuracy of a classifier on each class label and produces a value from 0 to 1 that signifies a balanced accuracy measure across all class labels. On class-balanced datasets, this measure is expected to closely reflect the test error figures. The table also lists the sizes or complexities of the final classifiers. Simpler classifiers are always preferred. For classifiers with similar accuracies, this attribute can provide information that indicates which algorithm might be preferable in that domain. The table highlights in boldface the classifiers which attain best accuracies for both measures and also highlights classifiers that simultaneously generate smallest ensembles.

Table 7.2: Accuracy results of all classifiers on datasets with uniform class distributions. Total error results on test sets represented as proportions with the average standard errors where applicable. G-mean values are also listed as well as the total size of the ensembles in parentheses.

| | | **Multiclass Cascaded** | | | | **OC** | **ECC** | **M2** |
|---|---|---|---|---|---|---|---|---|
| **Dataset** | **Measure** | **Node Size $\Phi$** | | | | | | |
| | | **5** | **10** | **25** | **50** | | | |
| letter | test error | **0.137** | **0.139** | **0.142** | **0.150** | 0.242 ±0.002 | 0.302 0.±008 | 0.306 |
| | G-mean | **0.860** | **0.860** | **0.857** | **0.849** | 0.753 | 0.429 | 0.685 |
| | ensemble size | 81578 | 78956 | 86918 | 94711 | 4000 | 4000 | 4000 |
| pendigits | test error | 0.067 | 0.071 | 0.067 | 0.053 | 0.067 ±0.001 | **0.050 ±0.002** | 0.160 |
| | G-mean | 0.928 | 0.927 | 0.932 | 0.946 | 0.933 | **0.950** | 0.914 |
| | ensemble size | 4157 | 3722 | 4715 | 5085 | 2000 | 2000 | 2000 |
| vowel | test error | 0.616 | **0.541** | 0.735 | 0.688 | 0.552 ±0.006 | 0.568 ±0.01 | 0.57 |
| | G-mean | 0.359 | **0.431** | 0.230 | 0.280 | 0.421 | 0.386 | 0.390 |
| | ensemble size | 1109 | 1368 | 2085 | 3419 | 2000 | 2000 | 2000 |
| optdigits | test error | 0.104 | 0.108 | 0.081 | 0.103 | 0.055 ±0.001 | **0.048** ±0.002 | 0.072 |
| | G-mean | 0.880 | 0.891 | 0.92 | 0.894 | 0.944 | **0.951** | 0.93 |
| | ensemble size | 2374 | 2249 | 2817 | 4089 | 2000 | 2000 | 2000 |
| segmentation | test error | 0.055 ±0.008 | 0.039 ±0.005 | 0.051 ±0.007 | 0.046 ±0.004 | 0.032 ±0.012 | **0.030 ±0.008** | 0.058 ±0.013 |
| | G-mean | 0.935 | 0.956 | 0.954 | 0.953 | **0.969** | **0.969** | 0.940 |
| | ensemble size | 814 | 978 | 1297 | 1513 | 2000 | 2000 | 2000 |
| vehicle | test error | 0.30 ±0.02 | 0.25 ±0.02 | 0.260 ±0.009 | 0.270 ±0.01 | 0.252 ±0.04 | **0.24 ±0.03** | 0.38 ±0.03 |
| | G-mean | 0.664 | 0.729 | 0.711 | 0.700 | 0.720 | **0.732** | 0.574 |
| | ensemble size | 721 | 787 | 809 | 976 | 2000 | 2000 | 2000 |
| iris | test error | 0.07 ±0.02 | 0.08 ±0.02 | 0.09 ±0.02 | **0.05 ±0.02** | 0.06 ±0.04 | 0.07 ±0.04 | 0.05 ±0.05 |
| | G-mean | 0.925 | 0.918 | 0.912 | **0.946** | 0.935 | 0.932 | 0.945 |
| | ensemble size | 50 | 65 | 123 | 228 | 500 | 500 | 500 |
| factors | test error | 0.08 ±0.01 | 0.07 ±0.01 | 0.07 ±0.01 | 0.08 ±0.01 | 0.026 ±0.008 | **0.023 ±0.005** | 0.04 ±0.01 |
| | G-mean | 0.916 | 0.926 | 0.931 | 0.917 | 0.974 | **0.977** | 0.960 |
| | ensemble size | 772 | 977 | 1604 | 2664 | 2000 | 2000 | 2000 |
| fourier | test error | 0.24 ±0.03 | 0.24 ±0.02 | 0.24 ±0.02 | 0.25 ±0.02 | **0.18 ±0.02** | 0.19 ±0.02 | 0.19 ±0.02 |
| | G-mean | 0.745 | 0.740 | 0.740 | 0.733 | **0.806** | 0.797 | 0.799 |
| | ensemble size | 1980 | 2183 | 2611 | 3619 | 2000 | 2000 | 2000 |
| waveform | test error | 0.19 ±0.01 | 0.192 ±0.009 | 0.19 ±0.01 | 0.18 ±0.01 | **0.145 ±0.008** | 0.18 ±0.01 | 0.18 ±0.01 |
| | G-mean | 0.786 | 0.778 | 0.798 | 0.805 | **0.855** | 0.819 | 0.820 |
| | ensemble size | 1339 | 1469 | 1477 | 1566 | 2000 | 2000 | 2000 |
| waveform noise | test error | 0.18 ±0.01 | 0.19 ±0.01 | 0.18 ±0.02 | 0.19 ±0.01 | **0.14 ±0.01** | 0.174 ±0.007 | 0.18 ±0.01 |
| | G-mean | 0.800 | 0.796 | 0.810 | 0.799 | **0.856** | 0.825 | 0.816 |
| | ensemble size | 1030 | 1050 | 1056 | 1061 | 2000 | 2000 | 2000 |
| morphological | test error | 0.35 ±0.02 | 0.35 ±0.03 | 0.34 ±0.02 | 0.36 ±0.02 | **0.29 ±0.03** | 0.33 ±0.03 | 0.29 ±0.03 |
| | G-mean | 0.584 | 0.593 | 0.611 | 0.583 | **0.671** | 0.617 | 0.670 |
| | ensemble size | 6663 | 6857 | 7266 | 8376 | 2000 | 2000 | 2000 |
| Mean Ranks | Mean Test Error Rank | 4.83 | 4.63 | 4.75 | 4.46 | 2.17 | 2.33 | 4.33 |
| | Mean G-mean Rank | 5.38 | 4.67 | 4.58 | 2.83 | 2.29 | 2.58 | 3.83 |

The table shows that on 4 out of the 12 datasets, at least one of the cascaded classifiers

attained the lowest test error rates while generating simpler ensembles than the control algorithms on 3 of these. As expected, the g-mean measures were also in agreement with the overall error rates. The ECC algorithm produced the best generalizations on five of the 12 datasets. The OC scored the best test errors on four datasets as did the cascaded.DP classifiers, while the M2 classifiers did not achieve a first rank on any dataset.

The results from Table 7.2 regarding the cascaded classifiers also suggest that there is no strong correlation between the node parameter $\Phi$ and the observed generalization rates. As the value of $\Phi$ increases, there are no noticeable trends on the classifier accuracy, implying that there is a degree of randomness taking place. This is explained because cascaded classifiers construct different class label orderings within the cascades with different values of $\Phi$. As a result, each ordering generates different distributions and class boundaries that vary in their separability and consequently affect the overall test error rates.

The final rows of the table summarize the generalization performances of all algorithms by comparing the mean ranks of the test errors and the g-means. Though this measure is informative and was used in all subsequent accuracy comparisons; arguably it is also less fair towards the cascaded classifiers because it individualizes the performances of the cascaded classifiers with different node sizes and does not present their collective accuracy. However, the measure is useful in determining whether there is a relationship between node sizes and improved accuracy.

The best mean ranks for the test error are achieved by the OC and ECC algorithms respectively with very little to differentiate between them. They are then followed by the M2 and closely by the cascaded classifiers. The Friedman test was once again used to determine if the attained averages were significantly different from the expected mean rank of 4. The calculated values for the statistical analysis are presented in Table 7.3.

Table 7.3: Results of the Friedman and Iman-Davenport tests with the critical differences based on the Nemenyi test for analyzing the mean ranks of the accuracy results from Table 7.2.

| | Friedman Value $\chi_F^2$ | Value in $\chi^2(6)$ | | Iman-Davenport Value $F_F$ | Value in $F_F(6,66)$ | | | Nemenyi Test Critical Difference | |
|---|---|---|---|---|---|---|---|---|---|
| | | $\alpha \leq 0.1$ | $\alpha \leq 0.025$ | | $\alpha \leq 0.1$ | $\alpha \leq 0.05$ | $\alpha \leq 0.025$ | $\alpha \leq 0.1$ | $\alpha \leq 0.05$ |
| test error | **10.56** | 10.645 | - | **1.89** | 1.86 | - | - | 2.375 | 2.601 |
| G-mean | **14.598** | 10.645 | 14.449 | **2.797** | 1.86 | 2.24 | 2.63 | 2.375 | 2.601 |

The value for the Friedman statistic was calculated as $\chi_F^2 = 10.56$ . The critical value for $\chi^2(6)$ at the significance level of $\alpha \leq 0.1$ is 10.645. The Friedman statistic fell just under the critical value which indicates that there is insufficient statistical evidence to conclude that the mean ranks differ from one another using this test. However, the less conservative Iman-Davenport [68] test produced the statistic $F_F = 1.89$, which being distributed according to the F-distribution with $k-1$ and $(k-1)(N-1)$ degrees of freedom, yielded the value of $F_F(6, 66) = 1.86$ at $\alpha \leq 0.1$. Based on this, the null-hypothesis could be rejected with there being enough evidence to suggest that differences between the algorithms does exist.

By proceeding to the Nemenyi test in order to calculate the CD, the value arrived at

for the $\alpha \leq 0.1$ significance level is 2.375. From this the following conclusions are drawn:

1. The cascaded algorithm with parameters $\Phi = 5, 10, 25$ has performed significantly worse than the OC algorithm.

2. The ECC algorithm also performed significantly better than the cascaded algorithm with parameters $\Phi = 5, 25$.

With respect to the mean ranks based on the g-mean accuracies, the performance of the M2 and the cascaded classifiers with larger values of $\Phi$ improved. The mean rank of the cascaded classifiers with $\Phi = 50$ was comparable to the ECC and OC classifiers on this more balanced metric.

The Friedman statistic $\chi_F^2$ for the g-mean ranks was calculated as 14.598 and the corresponding critical value for $\chi^2(6)$ and $\alpha \leq 0.025$ significance level is 14.449. This suggested that once again there was sufficient evidence to conclude that the algorithms are different based in the g-mean ranks. The CD value arrived at for the Nemenyi test for $\alpha \leq 0.05$ was 2.601. From this the only conclusion could be drawn was that the OC algorithm performed significantly better than the cascaded algorithm with $\Phi = 5$. At a lower significance level of $\alpha \leq 0.1$, it could be stated that the same conclusion holds for the OC algorithm over the cascaded algorithm with $\Phi = 10$.

**Unbalanced Class-distribution Datasets**

Up to this point, 12 datasets were grouped together and analyzed for accuracy. The remaining six datasets involved skewed class distributions which required more in-depth analysis and thus were grouped together. In addition to the test error and g-mean, the F-value test was used. The F-value combines both the precision and recall properties of a classifier in relation to its performance on each class label. The F-value breaks down the accuracy analysis of a classifier to the level of its generalization ability on each class label. The F-value yields a measurement that ranges from 0 to 1 for each class label to indicate the strength of its accuracy.

Research has shown that training on classed-biased distributions of themselves does not necessarily imply that the generalization ability of a classifier will be compromised [62, 71]. Instead, the findings indicate that skewed distributions exacerbate certain characteristics within the training sets that eventuates in a deterioration of the final accuracies. Based on these findings, results from the six datasets showed that on five of these the generalization of classifiers on some minority classes was clearly compromised, while on one (Robot Navigation) dataset this was not as clearly evident.

Table 7.4 displays the generalization results from the datasets with skewed class distributions. The primary point of reference were the g-mean results, though they are also listed by the test error rates for completeness. The total sizes of the ensembles was in addition displayed alongside the results. The summary of the table in the form of mean ranks indicated that the overall accuracy of the cascaded classifiers in respect to the control algorithms has improved. The ECC classifiers achieved the best mean ranks while being succeeded by the cascaded classifiers for $\Phi = \{25, 50\}$. The lowest overall accuracies were attained by the M2 classifiers and the cascaded classifiers with the smallest node sizes.

Results of particular interest are found on the Yeast datasets where the OC ans M2 classifiers failed to generate a valid g-mean value due to their registering zero accuracy for at least one class label with in the dataset. Meanwhile, all the cascaded and the ECC classifies successfully all class labels on this dataset. Despite their better generalization, the cascaded classifiers once again produced overly complex ensembles on this dataset. This was due to the datasets consisting of a larger number of class labels.

Table 7.4: Accuracy on datasets with biased class distributions. The g-mean is the primary assessment metric though the overall error in proportions, together with the mean deviation is also listed. The ensemble size and the summary of the table in the form of mean ranks, based on the g-mean, is also provided.

| | | **Multiclass Cascaded** | | | | **OC** | **ECC** | **M2** |
|---|---|---|---|---|---|---|---|---|
| **Dataset** | **Measure** | **Node Size $\Phi$** | | | | | | |
| | | **5** | **10** | **25** | **50** | | | |
| satimage | G-mean | 0.827 | 0.83 | 0.839 | **0.841** | 0.837 | 0.839 | 0.8 |
| | test error | 0.152 - | 0.145 - | 0.139 - | 0.147 - | **0.134 ±0.001** | 0.138 ±0.003 | 0.182 - |
| | ensemble size | 2621 | 2608 | 2760 | 2745 | 2000 | 2000 | 2000 |
| shuttle | G-mean | 0.937 | 0.949 | 0.798 | 0.808 | **0.989** | **0.989** | 0.973 |
| | test error | 0.0004 - | 0.0003 - | 0.0005 - | 0.0004 - | **0.00007** - | **0.00007** - | 0.0006 - |
| | ensemble size | 327 | 437 | 726 | 1206 | 2000 | 2000 | 2000 |
| glass | G-mean | 0.591 | **0.667** | 0.602 | 0.562 | 0.451 | 0.649 | 0.429 |
| | test error | **0.28 ±0.03** | 0.32 ±0.03 | 0.30 ±0.03 | 0.33 ±0.01 | 0.32 ±0.09 | 0.28 ±0.08 | 0.34 ±0.09 |
| | ensemble size | 309 | 360 | 592 | 985 | 500 | 500 | 500 |
| pageblocks | G-mean | 0.714 | 0.727 | 0.749 | 0.766 | 0.778 | **0.809** | 0.652 |
| | test error | 0.039 ±0.003 | 0.040 ±0.005 | 0.039 ±0.009 | 0.036 ±0.007 | **0.030 ±0.004** | 0.031 ±0.004 | 0.040 ±0.004 |
| | ensemble size | 1579 | 1851 | 2015 | 2269 | 2000 | 2000 | 2000 |
| robot navigation | G-mean | 0.969 | 0.978 | 0.980 | 0.981 | 0.994 | **0.996** | 0.994 |
| | test error | 0.022 ±0.008 | 0.012 ±0.004 | 0.014 ±0.004 | 0.014 ±0.005 | 0.003 ±0.002 | **0.002 ±0.001** | 0.003 ±0.002 |
| | ensemble size | 585 | 560 | 651 | 799 | 2000 | 2000 | 2000 |
| yeast | G-mean | **0.345** | **0.34** | **0.377** | **0.344** | - | 0.331 | - |
| | test error | 0.494 ±0.005 | 0.491 ±0.004 | 0.491 ±0.004 | 0.506 ±0.005 | **0.41 ±0.04** | 0.49 ±0.03 | 0.42 ±0.04 |
| | ensemble size | 7386 | 7359 | 8344 | 9770 | 2000 | 2000 | 2000 |
| | Mean Ranks(G-mean) | 5 | 4.17 | 3.67 | 3.67 | 3.75 | **2.25** | 5.33 |

The results from the F-values on the Yeast dataset in Table 7.5 reveal that the OC and M2 neglected to learn as classes seven and nine. In contrast, the cascaded and the ECC classifiers produced some correct detections on these classes. Although the scores were low for both algorithms on class label seven, they were comparatively higher for the cascaded classifiers, particularly on class nine. On three of the four remaining minority class labels which experienced a deteriorating accuracy, OC edged ahead of the cascaded classifiers, while M2 scored highest F-values on class eight.

The ability of the cascaded classifiers to focus learning on the minority class labels was also repeated on the Glass dataset in Table 7.6. The F-measure indicates that there was a notable drop in accuracy for the minority classes (types 3, 4, 5) compared to the majority classes. In each case, the cascaded classifiers generated F-values that were higher than

Table 7.5: F-value results for each class on the Yeast dataset.

| Measure | Class | | Cascaded.DP | | | | OC | ECC | M2 |
|---|---|---|---|---|---|---|---|---|---|
| | Type | Distr. | 5 | 10 | 25 | 50 | | | |
| | 0 | 16.4% | 0.491 | 0.454 | 0.549 | 0.480 | 0.586 | 0.514 | 0.591 |
| | 1 | 28.9% | 0.483 | 0.473 | 0.483 | 0.481 | 0.565 | 0.494 | 0.561 |
| | 2 | 31.2% | 0.514 | 0.508 | 0.510 | 0.486 | 0.573 | 0.508 | 0.563 |
| | 3 • | 3.0% | 0.621 | 0.575 | 0.591 | 0.621 | **0.679** | 0.546 | 0.667 |
| **F-Value** | 4 • | 2.4% | 0.441 | 0.529 | 0.522 | 0.400 | **0.590** | 0.387 | 0.526 |
| | 5 • | 3.4% | 0.333 | 0.356 | 0.314 | 0.216 | **0.374** | 0.336 | 0.351 |
| | 6 | 11.0% | 0.693 | 0.699 | 0.669 | 0.742 | 0.799 | 0.723 | 0.801 |
| | 7 • | 2.0% | **0.033** | **0.033** | **0.067** | **0.033** | 0.000 | 0.031 | 0.000 |
| | 8 • | 1.3% | 0.250 | 0.200 | 0.250 | 0.400 | 0.173 | 0.380 | **0.429** |
| | 9 • | 0.3% | **0.400** | **0.400** | **0.400** | **0.400** | 0.000 | 0.297 | 0.000 |

those of the control classifiers. A particularly strong decrease is seen in the minority class 3, on which the cascaded classifiers preserved comparatively high accuracy rates.

Table 7.6: F-value results for each class on the Glass dataset.

| Measure | Class | | Cascaded.DP | | | | OC | ECC | M2 |
|---|---|---|---|---|---|---|---|---|---|
| | Type | Distr. | 5 | 10 | 25 | 50 | | | |
| | 0 | 13.6% | 0.807 | 0.759 | 0.828 | 0.828 | 0.839 | 0.914 | 0.887 |
| | 1 | 32.7% | 0.786 | 0.729 | 0.714 | 0.686 | 0.720 | 0.724 | 0.696 |
| **F-Value** | 2 | 35.5% | 0.776 | 0.645 | 0.763 | 0.737 | 0.703 | 0.726 | 0.656 |
| | 3• | 7.9% | 0.294 | **0.412** | **0.412** | 0.294 | 0.105 | 0.314 | 0.087 |
| | 4• | 4.2% | **0.778** | **0.778** | 0.556 | 0.556 | 0.472 | 0.766 | 0.659 |
| | 5• | 6.1% | 0.385 | **0.769** | 0.462 | 0.462 | 0.635 | 0.688 | 0.434 |

On the Page blocks dataset in Table 7.7, the results were somewhat reversed. Here, the dataset is dominated by one large majority class (type 1) with four considerably smaller classes making up the rest. None of the cascaded classifiers registered the highest F-value scores, instead both the OC and the ECC classifiers produced the best accuracies.

Table 7.7: F-value results for each class on the Page Blocks dataset.

| Measure | Class | | Cascaded.DP | | | | OC | ECC | M2 |
|---|---|---|---|---|---|---|---|---|---|
| | Type | Distr. | 5 | 10 | 25 | 50 | | | |
| | 0 • | 2.1% | 0.535 | 0.533 | 0.574 | 0.670 | 0.673 | **0.695** | 0.528 |
| | 1 | 89.77% | 0.985 | 0.984 | 0.984 | 0.985 | 0.985 | 0.984 | 0.980 |
| **F-Value** | 2 • | 6.01% | 0.845 | 0.826 | 0.846 | 0.851 | **0.884** | 0.879 | 0.825 |
| | 3 • | 0.51% | 0.607 | 0.643 | 0.714 | 0.714 | 0.724 | **0.785** | 0.565 |
| | 4 • | 1.61% | 0.697 | 0.751 | 0.697 | 0.663 | **0.883** | 0.835 | 0.869 |

In order to complete the analysis of the generalization patterns, statistical tests were performed on the g-mean ranks from Table 7.4. The relevant statistical values and significance levels of this analysis are summarized in Table 7.8. By applying the Friedman test to the mean ranks, the resulting value was $\chi_F^2 = 6.196$. The corresponding critical value for $\chi^2(6)$ and $\alpha \leq 0.1$ significance level was calculated as 10.645. This implied

that the null-hypothesis could not be rejected. Iman-Davenport's more powerful variant of the Friedman's test produces the value $F_F = 1.04$. The associated critical value for $F_F(6, 30)$ and the significance level of $\alpha \leq 0.1$ was 1.98. The outcome of this was that the null-hypothesis had to be accepted once again. Though the average ranks in the table provided a fair comparison by themselves, the results from both statistical tests however, this not warrant making conclusions suggesting that the differences between the mean ranks are of a significant level.

Table 7.8: Statistical test results for classifier accuracies on skewed-distribution datasets.

| Friedman Value $\chi^2_F$ | Value in $\chi^2(6)$ | | Iman-Davenport Value $F_F$ | Value in $F_F(6,30)$ | | Nemenyi Test Critical Difference | |
|---|---|---|---|---|---|---|---|
| | $\alpha \leq 0.1$ | $\alpha \leq 0.05$ | | $\alpha \leq 0.1$ | $\alpha \leq 0.05$ | $\alpha \leq 0.1$ | $\alpha \leq 0.05$ |
| **6.196** | 10.645 | 12.592 | **1.04** | 1.98 | 2.42 | - | - |

### 7.4.3 Runtime Phase

The evaluation of the execution runtimes mirrored the approach taken to analyze the training runtimes by using the pairwise comparison. The first value reports a factor by which the cascaded.DP classifiers execute a *single weak classifier* faster than the control algorithm, while the second value takes runtime on the entire ensemble into account. Table 7.9 presents this execution runtime comparisons for each value of $\Phi$. The summary of each section of the table is provided in the form of means and standard deviations as well as the geometric mean.

The pairwise comparisons in the table indicate that:

1. The cascaded.DP classifiers displayed a faster mean execution runtime than the control algorithms across all training parameters $\Phi$.

2. The cascaded classifiers failed to produce faster runtimes only on the Letter dataset because the cascaded classifiers generated overly complex ensembles.

3. The largest performance gains by the cascaded.DP classifiers were achieved over the runtimes of the ECC, OC and M2 classifiers respectively.

4. The performance gains of the cascaded.DP classifiers over the control algorithms decreased as the training parameter value $\Phi$ increased. This patterns was also observed for training runtimes. As $\Phi$ approached the size of 25, performance parity with control algorithms on some datasets with larger numbers of class labels was observed as seen on Pendigits, Vowel and Optdigits.

5. Although at $\Phi = 50$, the cascaded.DP classifiers were still faster on average than the control algorithms, the large performance gains seen on lower values of $\Phi$ for the most part disappeared.

Table 7.9: Execution runtimes expressed as a factor by which the Cascaded.DP classifiers evaluate a single weak classifier and an entire ensemble faster than the control algorithms on each dataset.

| Cascaded.DP $\Phi = 5$ vs. | OC | | ECC | | M2 | |
|---|---|---|---|---|---|---|
| | per classifier | per ensemble | per classifier | per ensemble | per classifier | per ensemble |
| letter | 13.7 | 0.7 | 29.3 | 1.4 | 9.9 | 0.5 |
| pendigits | 4.9 | 2.4 | 10.6 | 5.1 | 6.0 | 2.9 |
| satimage | 5.1 | 3.9 | 10.4 | 7.9 | 5.2 | 3.9 |
| vowel | 1.5 | 2.8 | 3.0 | 5.4 | 1.4 | 2.6 |
| optdigits | 3.2 | 2.7 | 6.8 | 5.7 | 4.0 | 3.4 |
| segmentation | 2.7 | 6.7 | 5.4 | 13.4 | 2.9 | 7.1 |
| vehicle | 2.7 | 7.6 | 7.1 | 19.7 | 1.9 | 5.3 |
| glass | 1.4 | 2.2 | 3.5 | 5.7 | 1.6 | 2.5 |
| iris | 0.9 | 8.6 | 1.7 | 17.2 | 0.8 | 7.6 |
| factors | 6.4 | 16.6 | 14.7 | 38.1 | 8.8 | 22.9 |
| fourier | 11.1 | 11.2 | 23.2 | 23.4 | 11.5 | 11.6 |
| robot navigation | 4.2 | 14.2 | 8.3 | 28.2 | 4.6 | 15.8 |
| waveform | 5.4 | 8.1 | 9.8 | 14.7 | 3.6 | 5.3 |
| waveform noise | 4.9 | 9.6 | 8.6 | 16.7 | 3.1 | 6.1 |
| yeast | 16.7 | 4.5 | 34.5 | 9.3 | 17.8 | 4.8 |
| morphological | 36.0 | 10.8 | 72.7 | 21.8 | 36.2 | 10.9 |
| page blocks | 19.2 | 24.3 | 30.0 | 38.0 | 9.5 | 12.1 |
| shuttle | 1.2 | 7.3 | 2.0 | 12.4 | 1.2 | 7.4 |
| Mean | 7.9 | 8.0 | 15.6 | 15.8 | 7.2 | 7.4 |
| Std.Dev. | 8.9 | 6.0 | 17.5 | 10.9 | 8.5 | 5.5 |
| Geometric mean | 4.8 | 5.9 | 9.7 | 12.0 | 4.4 | 5.5 |

(a) Cascaded.DP $\Phi = 5$ execution runtime comparison.

| Cascaded.DP $\Phi = 10$ vs. | OC | | ECC | | M2 | |
|---|---|---|---|---|---|---|
| | per classifier | per ensemble | per classifier | per ensemble | per classifier | per ensemble |
| letter | 8.5 | 0.4 | 18.2 | 0.9 | 6.2 | 0.3 |
| pendigits | 3.2 | 1.7 | 6.8 | 3.6 | 3.8 | 2.1 |
| satimage | 3.8 | 2.9 | 7.6 | 5.8 | 3.8 | 2.9 |
| vowel | 1.2 | 1.7 | 2.3 | 3.3 | 1.1 | 1.6 |
| optdigits | 2.0 | 1.8 | 4.2 | 3.7 | 2.5 | 2.2 |
| segmentation | 1.7 | 3.4 | 3.3 | 6.8 | 1.8 | 3.6 |
| vehicle | 2.1 | 5.4 | 5.5 | 14.0 | 1.5 | 3.8 |
| glass | 0.9 | 1.3 | 2.5 | 3.4 | 1.1 | 1.5 |
| iris | 0.7 | 5.0 | 1.3 | 10.1 | 0.6 | 4.4 |
| factors | 4.0 | 8.2 | 9.2 | 18.8 | 5.5 | 11.2 |
| fourier | 7.3 | 6.7 | 15.2 | 13.9 | 7.6 | 6.9 |
| robot navigation | 3.8 | 13.5 | 7.5 | 26.7 | 4.2 | 15.0 |
| waveform | 7.5 | 10.2 | 13.7 | 18.6 | 5.0 | 6.7 |
| waveform noise | 4.9 | 9.4 | 8.5 | 16.3 | 3.1 | 5.9 |
| yeast | 12.2 | 3.3 | 25.1 | 6.8 | 13.0 | 3.5 |
| morphological | 27.0 | 7.9 | 54.6 | 15.9 | 27.1 | 7.9 |
| page blocks | 7.7 | 8.3 | 12.1 | 13.1 | 3.8 | 4.2 |
| shuttle | 0.7 | 3.2 | 1.2 | 5.4 | 0.7 | 3.2 |
| Mean | 5.5 | 5.2 | 11.0 | 10.4 | 5.1 | 4.8 |
| Std.Dev. | 6.3 | 3.7 | 12.6 | 7.1 | 6.3 | 3.7 |
| Geometric mean | 3.4 | 3.9 | 6.9 | 7.8 | 3.2 | 3.6 |

(b) Cascaded.DP $\Phi = 10$ execution runtime comparison.

| Cascaded.DP $\Phi = 25$ vs. | OC | | ECC | | M2 | |
|---|---|---|---|---|---|---|
| | per classifier | per ensemble | per classifier | per ensemble | per classifier | per ensemble |
| letter | 3.5 | 0.2 | 5.1 | 3.7 | 2.5 | 0.1 |
| pendigits | 2.0 | 0.8 | 4.2 | 1.8 | 2.4 | 1.0 |
| satimage | 2.5 | 1.8 | 5.1 | 3.7 | 2.5 | 1.8 |
| vowel | 0.9 | 0.8 | 1.7 | 1.7 | 0.8 | 0.8 |
| optdigits | 1.2 | 0.9 | 2.5 | 1.8 | 1.5 | 1.1 |
| segmentation | 1.1 | 1.7 | 2.2 | 3.4 | 1.2 | 1.8 |
| vehicle | 1.5 | 3.8 | 3.9 | 9.7 | 1.1 | 2.6 |
| glass | 0.7 | 0.6 | 1.9 | 1.6 | 0.8 | 0.7 |
| iris | 0.7 | 2.7 | 1.3 | 5.4 | 0.6 | 2.4 |
| factors | 3.3 | 4.1 | 7.6 | 9.4 | 4.5 | 5.6 |
| fourier | 5.8 | 4.4 | 12.1 | 9.3 | 6.0 | 4.6 |
| robot navigation | 2.7 | 8.4 | 5.4 | 16.7 | 3.0 | 9.4 |
| waveform | 5.6 | 7.6 | 10.1 | 13.7 | 3.7 | 5.0 |
| waveform noise | 4.1 | 7.7 | 7.1 | 13.4 | 2.6 | 4.9 |
| yeast | 7.7 | 1.8 | 15.9 | 3.8 | 8.2 | 2.0 |
| morphological | 14.8 | 4.1 | 29.9 | 8.2 | 14.9 | 4.1 |
| page blocks | 10.4 | 10.4 | 16.4 | 16.2 | 5.2 | 5.2 |
| shuttle | 0.5 | 1.4 | 0.9 | 2.4 | 0.5 | 1.4 |
| Mean | 3.8 | 3.5 | 7.4 | 7.0 | 3.5 | 3.0 |
| Std.Dev. | 3.9 | 3.1 | 7.4 | 5.2 | 3.5 | 2.4 |
| Geometric mean | 2.5 | 2.2 | 4.9 | 5.2 | 2.3 | 2.1 |

(c) Cascaded.DP $\Phi = 25$ execution runtime comparison.

| Cascaded.DP $\Phi = 50$ vs. | OC | | ECC | | M2 | |
|---|---|---|---|---|---|---|
| | per classifier | per ensemble | per classifier | per ensemble | per classifier | per ensemble |
| letter | 1.9 | 0.1 | 3.0 | 2.2 | 1.3 | 0.1 |
| pendigits | 1.2 | 0.5 | 2.6 | 1.0 | 1.5 | 0.6 |
| satimage | 1.5 | 1.1 | 3.0 | 2.2 | 1.5 | 1.1 |
| vowel | 0.7 | 0.4 | 1.4 | 0.8 | 0.7 | 0.4 |
| optdigits | 0.9 | 0.5 | 2.0 | 1.0 | 1.2 | 0.6 |
| segmentation | 0.9 | 1.2 | 1.9 | 2.5 | 1.0 | 1.3 |
| vehicle | 1.2 | 2.5 | 3.1 | 6.3 | 0.8 | 1.7 |
| glass | 0.7 | 0.4 | 1.8 | 0.9 | 0.8 | 0.4 |
| iris | 0.6 | 1.4 | 1.2 | 2.7 | 0.5 | 1.2 |
| factors | 3.1 | 2.3 | 7.2 | 5.4 | 4.3 | 3.2 |
| fourier | 4.7 | 2.6 | 9.8 | 5.4 | 4.9 | 2.7 |
| robot navigation | 2.5 | 6.3 | 5.0 | 12.4 | 2.8 | 7.0 |
| waveform | 4.3 | 5.5 | 7.8 | 9.9 | 2.8 | 3.6 |
| waveform noise | 3.3 | 6.2 | 5.7 | 10.8 | 2.1 | 3.9 |
| yeast | 5.6 | 1.1 | 11.5 | 2.4 | 6.0 | 1.2 |
| morphological | 9.2 | 2.2 | 18.6 | 4.5 | 9.3 | 2.2 |
| page blocks | 11.1 | 9.8 | 17.5 | 15.4 | 5.5 | 4.9 |
| shuttle | 0.7 | 1.2 | 1.2 | 2.0 | 0.7 | 1.2 |
| Mean | 3.0 | 2.5 | 5.8 | 4.9 | 2.6 | 2.1 |
| Std. Dev. | 3.0 | 2.7 | 5.4 | 4.4 | 2.4 | 1.8 |
| Geometric mean | 2.0 | 1.4 | 4.0 | 3.3 | 1.8 | 1.3 |

(d) Cascaded.DP $\Phi = 50$ execution runtime comparison.

Table 7.10: The total numbers of weak classifiers generated per classifier with varying multiclass node sizes on selected datasets. The totals are broken up into the numbers of multiclass and binary weak classifiers generated as well as the average number executed at detection runtime.

| | | Total classifiers | | | Average classifiers executed | | |
|---|---|---|---|---|---|---|---|
| **Dataset** | $\Phi$ | Multiclass | Binary | Total | Multiclass | Binary | Total |
| letter | 5 | 1630 | 79948 | 81578 | 0.467 | 0.105 | 0.113 |
| | 10 | 3260 | 75696 | 78956 | 0.464 | 0.094 | 0.110 |
| | 25 | 8150 | 78768 | 86918 | 0.453 | 0.084 | 0.118 |
| | 50 | 16300 | 78411 | 94711 | 0.456 | 0.080 | 0.145 |
| pendigits | 5 | 230 | 3927 | 4157 | 0.486 | 0.168 | 0.186 |
| | 10 | 460 | 3262 | 3722 | 0.484 | 0.144 | 0.186 |
| | 25 | 1150 | 3565 | 4715 | 0.476 | 0.141 | 0.222 |
| | 50 | 2300 | 2785 | 5085 | 0.464 | 0.128 | 0.280 |
| satimage | 5 | 80 | 2541 | 2621 | 0.586 | 0.203 | 0.215 |
| | 10 | 160 | 2448 | 2608 | 0.599 | 0.231 | 0.253 |
| | 25 | 400 | 2360 | 2760 | 0.594 | 0.205 | 0.262 |
| | 50 | 800 | 1945 | 2745 | 0.616 | 0.239 | 0.349 |
| vowel | 5 | 280 | 829 | 1109 | 0.585 | 0.234 | 0.322 |
| | 10 | 560 | 808 | 1368 | 0.557 | 0.211 | 0.353 |
| | 25 | 1400 | 685 | 2085 | 0.492 | 0.198 | 0.395 |
| | 50 | 2800 | 619 | 3419 | 0.526 | 0.197 | 0.466 |
| optdigits | 5 | 225 | 2149 | 2374 | 0.497 | 0.182 | 0.212 |
| | 10 | 450 | 1799 | 2249 | 0.482 | 0.167 | 0.231 |
| | 25 | 1125 | 1692 | 2817 | 0.473 | 0.146 | 0.278 |
| | 50 | 2250 | 1839 | 4089 | 0.47 | 0.132 | 0.32 |
| segmentation | 5 | 121 | 693 | 814 | 0.504 | 0.135 | 0.192 |
| | 10 | 286 | 692 | 978 | 0.508 | 0.132 | 0.245 |
| | 25 | 660 | 637 | 1297 | 0.504 | 0.124 | 0.320 |
| | 50 | 902 | 611 | 1513 | 0.499 | 0.132 | 0.333 |
| vehicle | 5 | 35 | 686 | 721 | 0.636 | 0.346 | 0.360 |
| | 10 | 70 | 717 | 787 | 0.632 | 0.328 | 0.356 |
| | 25 | 175 | 634 | 809 | 0.625 | 0.300 | 0.372 |
| | 50 | 350 | 626 | 976 | 0.631 | 0.289 | 0.412 |
| glass | 5 | 80 | 229 | 309 | 0.550 | 0.344 | 0.399 |
| | 10 | 160 | 200 | 360 | 0.560 | 0.350 | 0.445 |
| | 25 | 400 | 192 | 592 | 0.571 | 0.374 | 0.508 |
| | 50 | 800 | 185 | 985 | 0.551 | 0.416 | 0.526 |
| iris | 5 | 20 | 30 | 50 | 0.628 | 0.407 | 0.496 |
| | 10 | 40 | 25 | 65 | 0.617 | 0.408 | 0.537 |
| | 25 | 100 | 23 | 123 | 0.620 | 0.427 | 0.581 |
| | 50 | 200 | 28 | 228 | 0.623 | 0.542 | 0.610 |

(a)

| | | Total classifiers | | | Average classifiers executed | | |
|---|---|---|---|---|---|---|---|
| **Dataset** | $\Phi$ | Multiclass | Binary | Total | Multiclass | Binary | Total |
| factors | 5 | 230 | 548 | 778 | 0.49 | 0.17 | 0.26 |
| | 10 | 460 | 528 | 988 | 0.49 | 0.15 | 0.31 |
| | 25 | 1150 | 480 | 1630 | 0.47 | 0.13 | 0.37 |
| | 50 | 2300 | 415 | 2715 | 0.47 | 0.12 | 0.41 |
| fourier | 5 | 230 | 1756 | 1986 | 0.52 | 0.25 | 0.28 |
| | 10 | 460 | 1735 | 2195 | 0.5 | 0.23 | 0.28 |
| | 25 | 1150 | 1487 | 2637 | 0.49 | 0.21 | 0.33 |
| | 50 | 2300 | 1370 | 3670 | 0.48 | 0.19 | 0.37 |
| robot navigation | 5 | 35 | 550 | 585 | 0.64 | 0.38 | 0.39 |
| | 10 | 70 | 490 | 560 | 0.62 | 0.38 | 0.41 |
| | 25 | 175 | 476 | 651 | 0.61 | 0.34 | 0.41 |
| | 50 | 350 | 449 | 799 | 0.6 | 0.33 | 0.45 |
| waveform | 5 | 20 | 1652 | 1672 | 0.68 | 0.32 | 0.32 |
| | 10 | 40 | 1482 | 1522 | 0.68 | 0.28 | 0.29 |
| | 25 | 100 | 1470 | 1570 | 0.69 | 0.27 | 0.3 |
| | 50 | 200 | 1482 | 1682 | 0.68 | 0.28 | 0.33 |
| waveform noise | 5 | 20 | 1155 | 1175 | 0.67 | 0.37 | 0.38 |
| | 10 | 40 | 1144 | 1184 | 0.67 | 0.34 | 0.35 |
| | 25 | 100 | 1093 | 1193 | 0.67 | 0.32 | 0.35 |
| | 50 | 200 | 974 | 1174 | 0.67 | 0.3 | 0.36 |
| yeast | 5 | 230 | 7156 | 7386 | 0.53 | 0.22 | 0.23 |
| | 10 | 460 | 6899 | 7359 | 0.51 | 0.2 | 0.22 |
| | 25 | 1150 | 7194 | 8344 | 0.51 | 0.19 | 0.24 |
| | 50 | 2300 | 7470 | 9770 | 0.49 | 0.19 | 0.26 |
| morphological | 5 | 230 | 6440 | 6670 | 0.5 | 0.16 | 0.18 |
| | 10 | 460 | 6408 | 6868 | 0.5 | 0.15 | 0.18 |
| | 25 | 1150 | 6141 | 7291 | 0.5 | 0.16 | 0.21 |
| | 50 | 2300 | 6127 | 8427 | 0.5 | 0.17 | 0.26 |
| page blocks | 5 | 55 | 1524 | 1579 | 0.54 | 0.15 | 0.16 |
| | 10 | 110 | 1741 | 1851 | 0.64 | 0.19 | 0.22 |
| | 25 | 275 | 1740 | 2015 | 0.6 | 0.19 | 0.25 |
| | 50 | 550 | 1719 | 2269 | 0.62 | 0.16 | 0.27 |
| shuttle | 5 | 110 | 217 | 327 | 0.51 | 0.48 | 0.49 |
| | 10 | 220 | 217 | 437 | 0.78 | 0.33 | 0.56 |
| | 25 | 550 | 176 | 726 | 0.87 | 0.37 | 0.75 |
| | 50 | 1100 | 106 | 1206 | 0.51 | 0.28 | 0.49 |

(b)

The key to the efficient execution of the cascaded classifiers lies in the structuring and the multiple exit points within an ensemble. As a result, most layers are only exposed to partial execution even if the target class label for a particular candidate instance is assigned to the last layer. Moreover, as a candidate sample propagates through a cascade, the size of the layers also decrease in size thus further limiting the execution time. Table 7.11a and Table 7.11b are provided in order to demonstrate the composition of the cascaded classifiers in respect to the total number of binary and multiclass weak classifiers, and the mean proportion of classifiers executed per dataset. The last column list the execution runtimes.

The average proportion of all weak classifiers executed per dataset varies, but in the majority of cases, the table shows that less than half of the ensembles is evaluated per sample. Some of the least efficient proportions are found on the Shuttle dataset, in which up to 75% of the ensemble undergoes evaluation. The most efficient figures are found with the Letter dataset, where the best performances required 11% of the total ensemble to be evaluated.

The mean total number of multiclass weak classifiers executed per samples is less variable and tends to be within the 40%-60% percent range. An exception to this occurs on the Shuttle dataset, where up to 87% of the multiclass weak classifiers are executed. This scenario can take place only on datasets with high class distribution skews, combined with a cascade ordering in which the majority class labels are assigned to final layers of the total cascade.

The total number of multiclass weak classifiers increases linearly with the size of the nodes, while the total number of binary weak classifiers remains largely the same and appears to be uncorrelated to the size of the nodes. The increase in runtimes for the classifiers as the nodes sizes increase can therefore be attributed to the higher expense of evaluating the multiclass weak classifiers. A linear relationship between the increase in the size of the multiclass nodes and the execution runtimes can be observed on a number of datasets in Figure 7.8.

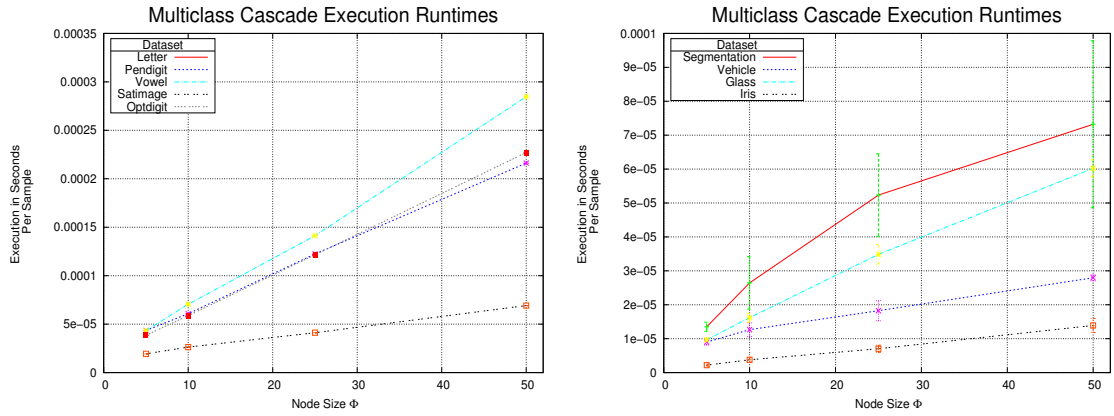



Figure 7.8: Multiclass cascade classifier execution runtimes from a selection of datasets.

Binary weak classifiers significantly outnumber the multiclass weak classifiers on each dataset. Despite this, their overall effect on the runtimes is minimal since a considerably

smaller proportion of the binary weak classifiers is executed per sample than that of the multiclass weak classifiers and due to their low computational complexity.

The values for the composition of the cascaded classifiers on the Letter dataset stand out in particular, due to the massive dimensions of the resulting ensembles. Perhaps these figures demonstrate the best and some of the limitations of the cascaded structure and specifically its inability to produce compact ensembles on datasets with large class labels. Although the complexity of the final classifiers has been shown to be extremely high of up to a factor of 10, the training and the execution runtimes however have been comparable. At least for the cascade.DP ($\Phi = 5, 10$) classifiers, both their training and execution runtimes as a measure of complexity were lower than those of the control classifiers.

## 7.5 Discussion and General Evaluation

**Multiclass Weak Learner:**

The proposed multiclass weak-learner has experimentally demonstrated that it has met the requirements of low computational overheads and sufficient discriminative strength to complement the stepwise decomposition strategy of the multicultural cascade. However, the simple domain-partitioning weak learner alone in combination with non-cascaded AdaBoost.M1 is unlikely to guarantee to be able to always satisfy the training error constraints of boosting, which require them to be below 50%. Given the weakness of the inducer, its scalability to datasets with very large numbers of class labels possessing limited feature spaces is questionable, even in a cascaded setting. In these instances, there is a high likelihood of such datasets containing complex and overlapping class distributions. The ability of this inducer to sufficiently separate class labels at initial layers to the degree in which it is of practical use may be difficult to realize.

**Complexity:**

From the perspective of implementation, the multiclass cascade is not straightforward to program and requires a greater effort than implementing the OC, ECC or M2 algorithms. There are many more steps involved in implementing the cascaded framework and it lacks the simplicity and elegance of implementation that the control algorithms possess. However, when the training runtimes are considered as a measure of computational complexity, the extra implementation effort is worthwhile. This is accentuated on larger datasets and especially in comparison to the runtimes of M2 in the presence of many class labels. The faster training runtimes can be considered an advantage, especially in machine learning domains found in industry which require frequent on-the-spot training of classifiers for customized problems.

Occam's razor provides a guideline that deems less complex solutions to a problem more preferable if their respective accuracies are comparable to the alternatives. Based on this, the research has shown that there are numerous occasions where the control algorithms produced comparable accuracies to the cascade classifiers, for considerably smaller ensembles. The exponential decrease of the upper bound on the training error for the control algorithms translated to similar generalization patterns that were not often matched by the cascaded classifiers in terms of cost.

One reason why successful separate-and-conquer algorithms like RIPPER achieve efficiency is due to their learning approach which selects a class label to learn *a priori*, which is followed by rules that successively refine the classification accuracy for the chosen class label. A source of inefficiency of the multiclass cascade algorithm which can result in complex classifiers lies in the reverse approach that it takes to learn in respect to RIPPER. In effect, the multiclass cascade algorithm proceeds with a goal that states "at each point, select and *refine* the classification accuracy for a class label which will not be selected as the final target class label for the current layer". It is the refining of the classification for a given non-target class label in the form of binary nodes, that often results in complex final classifiers. This is especially true on large datasets where the initial accuracy of the multiclass weak learner is not high.

The fact that the cascaded framework is susceptible to producing more complex solutions than the control algorithms is also offset by the reduced complexity of its execution phase. While the final classifier is frequently more complex on datasets with many class albels, the runtime execution of it has been shown to be consistently more efficient. The experiments have demonstrated that the more complex solutions have also resulted in more discerning classifiers, that have built-in exit points which ensure that only small proportions of its ensemble are evaluated. As such, the cascaded classifiers are more suitable towards high performance systems which process dense data streams in real-time.

**Generalization:**

The generalization results of the cascaded classifiers have demonstrated that on average and for given $\Phi$ parameter settings, they are capable of producing results that are as good as those of the control algorithms used in this research. The data revealed that there was a considerable degree of variance between the results of cascaded classifiers with different values of $\Phi$. The results showed that there was no definitive correlation between the complexity of the node sizes and the overall accuracy of a classifier on any one dataset. The results did point to some increase in accuracy with more complex nodes on average across all datasets; however, these results were not statistically significant.

The fact that the complexity of the nodes did not strongly correlate with the generalization ability of a classifier was also not surprising given the observations made by Dong et al. [32], Lorena et al. [94] regarding nested hierarchies with which the cascaded framework bears some resemblance. The similarities between nested hierarchies and the cascaded algorithm lie in that both approaches decompose the learning process and the final classifier into a tree-like structure where the branching is based on classes rather than on individual instances. The findings indicate that different orderings can strongly alter the accuracy of a classifier, while the optimal ordering of such a structure is in most cases not feasible to search exhaustively [32].

**Ensemble Diversity:**

The diversity of an ensemble is the first key principle in ensemble-based learning. The cascaded structure ensures that a high degree of diversity occurs through the process of constructing a degenerate tree structure, both within a layer as well is from one layer to the next. The effect of this on the diversity is to ensure that at every stage of training, the

combination of class labels being learned is different. The ever changing class distributions at every point of training explicitly maximizes the ensemble diversity, since unique information regarding class boundaries is learned.

**Ensemble Vote-Combination:**

The second key principle of ensemble based learning is the manner in which the individual votes of the weak classifiers are combined in order to formulate a prediction. The current strategy of combining votes is disjointed and naive. Currently, as a sample propagates within a layer from one node to the next, no past information is retained and used for subsequent evaluation. Likewise, as a sample propagates from one layer to the next, the successive layers are not aware of how a sample performed previously and what votes it received from different nodes.

At training, each node is assessed for its error rate and based on this it is assigned an overall confidence value; however, this confidence value is not currently made use of. Since each node encodes unique information about the class relationships, the final cascade classifier with all the nodes can be visualized as a confidence map. Consequently, as a sample traverses through a cascade, it collects information regarding nodes, its associated class labels and confidences that it has been positively classified on. This information, currently not utilized, can arguably be combined in a meaningful way that improves classification accuracy. Therefore, there is plenty of scope in future work for experimenting with more complex vote combination schemes.

**Scalability:**

The cascaded framework is scalable to large datasets; however, the scalability does not extend to large numbers of classes. There are limits to the total number of class labels within a dataset for which the cascaded structure is well suited. The Letter dataset with 26 class labels has demonstrated the upper limits of this range. As class numbers increase, the size of the cascaded classifier grows both horizontally within each layer, as well as vertically in respect to the number of cascade layers. As a result, both the training and the execution runtimes become more expensive. The experiments have shown that this can be offset to a degree by limiting the size of the multiclass nodes; but this is not likely to have sufficient influence on datasets with very large numbers of class labels.

A solution to the problem of scaling to datasets with many class labels is to convert the current cascade structure into a balanced tree in which each child node represents a cascade layer made up of multiclass nodes. Research into structuring ensembles into tree-like structures and nested hierarchies has already been explored with promising results [32, 42, 131].

Given that classifiers with large nodes do not scale well to problem sets with many class labels, the cascaded classifiers can be altered to handle variable node sizes at different layers. Alternatively, the cascaded algorithm can also be redesigned in a way so it removes instances belonging to more than just one class label from each step of node training. The effect of this would translate to significantly smaller overall ensembles and reductions in training runtimes. Consequently, the current cascading structure that is a form of a degenerate tree would become less restrained and arguably more efficient [42].

**Interpretability:**

Boosted ensemble-classifiers by their very nature are difficult to interperet. Their lack of interpretability becomes more acute with an increase in the size of the ensemble. In a conventional boosted ensemble it is also not possible to identify subsets of weak classifiers which are responsible for the classificfation of any one class label or pattern. However, though the components of a multiclass cascaded classifier are not easily interpretable; subsets of classifiers which are responsible for classifiying any given class label or pattern can be directly identified amd examined.

As a result, the ability exists to query the learning algorithm in order to determine exactly what samples were used at any one point in the training process for building decision boundaries. Likewise, the evaluation of each sample can be traced back to the path it has traversed in the cascaded structure; thus, identifying all nodes and layers that have contributed to a decision. This enables the reasoning of a classifier's decision to be clear and due to this, weaknesses in its ensemble can also be identified.

**Stability:**

The stability of the cascaded structure is also very high. In contrast to OC and ECC algorithms which randomly select set colourings and consequently produce different classifiers on identical datasets, the cascaded classifier is deterministic and will always produce the same classifiers given the same data. In instances when different batches of data from the same dataset are learned, the research has also showed that the level of stability is very high. The evidence for this can be seen in Table 7.2, where the training using the 10-fold cross-validation, generated mean standard error rates which were often lower than those produced by the control algorithms.

**Usability:**

Given that machine learning is an iterative process; it is therefore necessary for any learning algorithm to possess adjustable parameters that are comprehensible and easy to tune. Based on this, the cascaded structure can be described as being very usable. The primary adjustable parameter is the size of the multiclass nodes. It is both straightforward to understand and has been shown in the experiments to possess the ability to heavily influence the training and execution runtimes, as well as the generalization ability of a classifier.

**Flexibility:**

The cascaded framework is also flexible since each node is inducer-independent. The proposed weak learner can be substituted with any other multiclass learner, including the more powerful tree induction algorithms, when the dataset complexity becomes too high for the proposed inducer. This will however result in a considerable increase in training runtimes.

One alternative strategy is to replace the proposed learner with a decision stump threshold, and AdaBoost.M1 likewise with a multiclass version such as OC, ECC or M2. The advantage of this approach would be a simplified and faster learner and a boosting

algorithm that has proven bounds and is guaranteed to converge. The combination of these algorithms with the cascaded framework will be the subject of future research.

The flexibility also extends to the vote combination strategy and as such, the cascaded algorithm is combiner-independent. Previously, it was discussed how the current ensemble vote-combining method is simplistic and functions solely on accept/reject decisions to traverse the cascade. The structure is in place to enable the cascaded classifier to function also with a variety of more complex vote-combining strategies.

**Adaptability:**

The multiclass cascaded approach was originally conceived with the goal of it being extended into operating domains where continuous adaptability to changing environments is required. As discussed in Chapter 6, the need for machine learning approaches to go beyond static training and runtime capabilities and into dynamic and adaptable learning structures is becoming increasingly relevant. This has been made evident through the increasing trends towards long life classifier systems and applications in evolving environments. The dynamic environments of this kind, render segments of previously learned information useless, while introducing new patterns and sometimes even new class labels.

Potential exists for extending the multiclass cascade in such a way that the outputs from all the nodes from each layer are combined in such a way that they become inputs to a meta learning algorithm in the form of a continuous and thus adaptable stacked generalization [185]. The matrix of nodes can be perceived as a confidence map. As a sample propagates through the cascade, a vector of class labels and associated confidences based on its prediction can be generated. This feature vector could potentially encode useful information regarding relationships between class labels and how they change over time. Given an appropriate algorithm, this information can theoretically be used for efficiently learning concept drifts without explicitly training or altering the cascade structure.

Given a scenario in which the existing multiclass classifier is exposed to instances of a previously unlearned class label in an evolving detection domain; the adaptability of the cascade is realizable without altering the existing layers. In this the integration of the new class label is theoretically achievable by training a new layer for a given class label using a batch learning mechanism. Upon completion, the new layer can be appended to the beginning of a cascade, so that it is set as the initial layer. Provided that this layer is trained with a sufficient number of samples representing all existing class labels, there is no reason to believe that an additional layer would adversely affect the original portions of the cascade.

## 7.6 Summary

In this chapter, a unique cascaded training algorithm for multiclass ensemble-based problems was presented, together with an accompanying new weak learning method. Complex inducers for large multiclass learning problems tend to be expensive computationally with protracted training runtimes. Using 18 UCI datasets, the research demonstrated how a simple weak learner can be combined with a multiclass separate-and-conquer strategy involving a combination of multiclass and binary learning in order to realize arbitrarily low training errors.

The cascaded framework was compared against well established control algorithms: AdaBoost.OC, AdaBoost.ECC and AdaBoost.M2. Based on the experiments, the findings are as follows:

1. The combination of the proposed cascaded multiclass-cascade algorithm and the domain-partitioning weak learner is an effective strategy for producing classifiers.

2. The strategy of a stepwise decomposition of a multiclass learning problem into more manageable learning subtasks reduces the training complexity and shortens training runtimes.

3. The proposed algorithm does not exponentially decrease the upper bound on the training error like the control algorithms. Therefore, the control algorithms reach lower training errors for a smaller cost in the initial boosting rounds.

4. The tunable parameter $\Phi$ for determining the complexity of the multiclass nodes has a strong influence on the classifier accuracy, training and execution runtimes.

5. Smaller values of $\Phi$ result in less complex ensembles and thus shorter training and execution runtimes

6. Higher ensemble complexity does not guarantee that generalization will improve on a given dataset; however, there is evidence that on average, the accuracy does improve as the node sizes increase.

7. On balanced datasets, there were no statistically significant differences between the cascaded and control algorithms using the test error as the primary metric for comparing mean ranks. However, on the g-mean metric, there was evidence to indicate that the accuracy of cascaded classifiers trained with smaller values of $\Phi$ exhibited compromised generalization rates.

8. For unbalanced-class datasets there was no statistical difference between the mean ranks of the cascaded and the control algorithms using the g-mean as the generalization metric.

9. The cascade framework in its current form is not scalable to datasets with large numbers of class labels.

10. The cascaded classifiers consistently produce faster execution runtimes since only a fraction of the total ensemble is evaluated per sample.

# Chapter 8

# Decomposing Existing Monolithic Multiclass Boosters into Cascades

The previous chapter demonstrated the ability of the multiclass cascading algorithm to efficiently produce accurate classifiers by using the proposed domain-partitioning weak-learner as its underlying inducer. The algorithm showed that it was capable of achieving this with reduced runtimes and it also demonstrated that its classifiers were enabled for rapid execution. The results indicated that the control single-layered algorithms consistently displayed a better cost/benefit ratio in respect to the generalization for each round of boosting in the initial rounds as well as a stronger generalizations on certain datasets. The shortcomings of the single-layer algorithms was seen in their runtimes for training and detection phases. It is a weakness which can be expected to become magnified in high volume datasets and in setting where expensive feature types are employed. For this reason, classifiers produced by these algorithms may not be suitable for many types of time-critical domains.

## 8.1 Motivation

The purpose of this chapter is to make a contribution towards enabling existing single-layer multiclass algorithms becoming capable of producing classifiers for time-critical domains. The research proposes combining the multiclass cascading method from Chapter 7 with the single-layered single-layer multiclass algorithms AdaBoost.OC, AdaBoost.ECC and AdaBoost.M2. The fusion of these methods is proposed on the basis of substitution the current combination of the domain-partitioning weak learner and AdaBoost.M1 with the combination of the decision stumps and one of the single-layered multiclass boosting algorithms. The questions being asked are:

- Does the multiclass meta-learning cascading framework posses the ability to be generic by being able to incorporate other learning algorithms instead of the domain-partitioning weak learner?
- How does replacing the domain-partitioning weak learning with OC, ECC and M2 affect the training and execution runtimes as well as the overall generalization ability of the resulting classifiers?

- What are the effects on efficiency and effectiveness of single-layered OC, ECC and M2 when they are converted into cascaded classifiers?

## 8.2 Converting Single-Layered Multiclass Algorithms to Cascades

The procedure for combining OC, ECC and M2 with the cascaded framework is relatively straightforward due to the fact that all multiclass nodes are independent. The implication is that each round of boosting within a multiclass node is unaware of the learning in the preceding nodes. Because of this, each multiclass boosting algorithm is implemented as a stand-alone single-layered procedure with each node, without being fundamentally altered. As such, the main point of departure for OC, ECC and M2 in their cascaded implementation is that all sample weights are reset to a uniform distribution for each node. This enables the algorithms to easily handle the removal of class labels and instances from one node to the next as well as from each succeeding cascade layer. Secondly, in the process of building a cascade, once the number of remaining unlearned class-labels has been reduced to two, the multiclass boosting algorithms are no longer applicable. Instead, Discrete binary-class AdaBoost is applied.

Lastly, an additional modification was necessary for ECOC-based boosting algorithms like OC and ECC that concerns the manner in which the colouring sets are chosen. Recall that ECOC methods construct coding matrices for class label resolution at each boosting iteration $t$ (Algorithms 3 - 4). This is accomplished by reducing the problem into a binary learning task in which all classes are relabeled based on a colouring scheme $\mu_t$ at each iteration $t$, that is $\mu_t(y_i) \in \{0, 1\}$. The randomly selected colouring scheme is crucial since it determines the value $U_t$ from the distribution $\tilde{D}_t$ as in

$$U_t = \sum_{i=1}^{m} \sum_{\ell \in Y} \tilde{D}_t(i, \ell) [\![ \mu_t(y_i) \neq \mu_t(\ell) ]\!]. \tag{8.1}$$

The quantity $U_t$ is important since it determines the strength of the class-resolution capability of $\mu_t$ and therefore the goal is to maximize this value [140]. The problem arises within OC and ECC in choosing an effective colouring scheme after each new node is trained since some classes become partially and sometimes completely removed from the layer training. This can result in a ineffective values of $U_t$ being generated if changes in class distributions are not factored in. Without a strategy of mitigating this, the risk of generating increasingly suboptimal values of $U_t$ can be expected to become more likely with each additional new node.

In order to ensure that the colouring scheme for selecting $\mu$ favours constructing the coding matrix that emphasizes the resolution between class labels which have not yet been learned by a layer, straightforward modification to the generation of the colouring function of Schapire [140] was devised. The adopted approach involves the following steps: (1) Two sets $S_1$ and $S_2$ containing class labels are created. $S_1$ is populated with all current layer class labels which have not been learned and assigned to a node. $S_2$ is populated with the remaining class labels which have already been assigned to node. (2) Beginning with $S_1$, class labels are selected from it randomly without replacement and assigned to $\mu$ with

an alternating colouring until there are no more samples left in $S_1$. This ensures that an even split of unlearned labels is realized and the most effective value $U_t$ is achieved. (3) The previous step is repeated for all class labels in $S_2$.

## 8.3 Experiment design

Single-layered AdaBoost.OC, AdaBoost.ECC and AdaBoost.M2 classifiers from Chapter 7 were used as control classifiers. The cascading procedure was applied to each of the control algorithms, thus generating three additional classifiers. These are referred to as cascaded.OC, cascaded.ECC and cascaded.M2. For consistency with the previous chapter, each control algorithm was trained with node parameters where $\Phi = \{ 5, 10, 25, 50\}$. However, for brevity only cascaded classifiers with $\Phi = 10$ are presented here for analysis. Classifiers with $\Phi = 10$ were selected over others due to the empirical results which indicated that cascaded classifiers with $\Phi = 10$ displayed arguably the best trade-off between training/execution runtimes and the overall accuracy. For completeness, the accuracy and performance results of an equivalent classifier using the domain-partitioning weak learner with $\Phi = 10$ from the previous chapter were also used for comparative analysis with the new cascaded classifiers[1]. The remaining training procedures, parameters and datasets were equivalent to those outlined in Chapter 7.3.

## 8.4 Results

### Training Phase

Table 8.1 lists the comprehensive account of the training runtime-performances of each algorithm across all datasets. For clarity, instead of absolute values, the table reports the speed-up factors in training runtimes of cascaded classifiers over their associated single-layer equivalents that are in line with the approach taken in the previous chapter. The overall trends are summarized using the mean, standard deviationa and the geometric mean. The geometric mean was considered as being more meaningful since the data is ratio-scale and due to the ability of the geometric mean to act as a measure of location when the data are highly skewed to the right [193].

The table shows that on every dataset, the cascaded algorithm trained classifiers faster than their single-layered equivalents both on the absolute runtimes per ensemble and on the per weak-classifier metric. An exception occurred once however, on the Letter dataset comparing the cascaded.DP classifier and the ECC classifier, where the ECC algorithm recorded 20% faster absolute runtimes for the entire ensemble. The more relevant figures concerning the cascaded OC, ECC and M2, demonstrate that the training runtimes have been accelerated between 5-23 times over their single-layered implementations. The factor of speed-up was the largest for the cascaded.M2 classifiers over the conventional M2.

The selected training convergence graphs in Figure 8.1 summarize the general patterns seen across all classifiers[2]. The graphs are consistent with those in the previous chapter. Almost without exception across all the datasets in these experiments, the cascaded classifiers were not able to keep up with the control algorithms round-for-round.

[1]Parts of this research have been submitted for publication [162].

[2]Convergence patterns across all datasets are located in Appendix D.

Table 8.1: The factor of speed-up in the training runtimes between the cascaded and the single-layer algorithms.

| | cascaded.ECC vs ECC | | cascaded.OC OC | | cascaded.M2 vs M2 | | cascaded.DP vs ECC | | cascaded.DP vs OC | | cascaded.DP vs M2 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Dataset | weak clsf. | per ens. | weak clsf. | per ens. | weak clsf. | per ens. | weak clsf. | per ens. | weak clsf. | per ens. | weak clsf. | per ens. |
| letter | 23.4 | 1.6 | 15.2 | 2.1 | 27.9 | 2.2 | 15.4 | 0.8 | 23.1 | 1.2 | 21073.6 | 1067.6 |
| pendigits | 8.1 | 5.2 | 9.8 | 6.2 | 30.3 | 18.5 | 5.6 | 3.0 | 7.7 | 4.1 | 7600.9 | 4084.3 |
| satimage | 5.3 | 5.6 | 5.5 | 5.6 | 46.4 | 42.0 | 4.5 | 3.5 | 5.2 | 4.0 | 481.3 | 369.1 |
| vowel | 4.0 | 7.2 | 5.8 | 9.8 | 8.5 | 15.9 | 2.0 | 3.0 | 3.0 | 4.4 | 4800.6 | 7018.4 |
| optdigits | 6.2 | 6.1 | 6.2 | 5.9 | 19.0 | 17.1 | 3.1 | 2.8 | 3.6 | 3.2 | 5890.2 | 5238.0 |
| segmentation | 5.0 | 14.7 | 6.1 | 17.3 | 12.7 | 32.8 | 3.3 | 6.7 | 4.3 | 8.9 | 580.1 | 1186.2 |
| vehicle | 3.8 | 12.3 | 4.4 | 13.3 | 17.0 | 48.5 | 2.9 | 7.4 | 3.5 | 9.0 | 52.0 | 132.1 |
| glass | 2.7 | 3.8 | 3.7 | 4.3 | 10.9 | 11.5 | 1.4 | 2.0 | 2.1 | 2.9 | 123.1 | 171.0 |
| iris | 1.6 | 17.1 | 2.5 | 26.2 | 5.5 | 61.5 | 1.3 | 10.2 | 2.0 | 15.3 | 15.9 | 122.7 |
| factors | 3.5 | 8.2 | 3.6 | 8.1 | 14.4 | 29.5 | 1.4 | 2.9 | 1.6 | 3.2 | 2882.0 | 5899.7 |
| fourier | 4.7 | 6.7 | 5.1 | 6.8 | 11.1 | 15.5 | 2.0 | 1.8 | 2.3 | 2.1 | 3676.6 | 3368.4 |
| robot navigation | 3.3 | 16.3 | 3.3 | 15.2 | 16.1 | 48.7 | 4.5 | 9.1 | 5.5 | 11.1 | 84.0 | 170.0 |
| waveform | 2.6 | 4.5 | 2.9 | 4.8 | 16.3 | 30.0 | 3.6 | 4.9 | 4.2 | 5.7 | 27.7 | 37.8 |
| waveform noise | 2.5 | 6.3 | 2.7 | 6.0 | 14.7 | 37.8 | 3.2 | 6.1 | 3.5 | 6.6 | 24.2 | 46.1 |
| yeast | 6.3 | 1.5 | 8.4 | 2.0 | 53.4 | 15.6 | 6.3 | 1.7 | 9.5 | 2.6 | 6581.6 | 1788.7 |
| morphological | 9.4 | 3.4 | 14.9 | 5.2 | 57.6 | 18.4 | 5.8 | 1.7 | 10.2 | 3.0 | 6136.7 | 1789.9 |
| pageblocks | 2.9 | 3.5 | 2.7 | 3.3 | 24.4 | 43.3 | 3.5 | 3.8 | 5.0 | 5.4 | 129.3 | 139.8 |
| shuttle | 2.2 | 6.7 | 3.1 | 10.6 | 14.5 | 38.7 | 1.6 | 7.3 | 2.2 | 10.1 | 280.9 | 1285.4 |
| Mean | 5.4 | 7.3 | 5.9 | 8.5 | 22.3 | 29.3 | 4.0 | 4.4 | 5.5 | 5.7 | 3357.8 | 1884.2 |
| Std. Dev. | 4.9 | 4.7 | 3.9 | 6.2 | 15.4 | 15.9 | 3.2 | 2.8 | 5.1 | 3.8 | 5202.7 | 2255.5 |
| Geometric Mean | 4.31 | 5.89 | 4.99 | 6.81 | 18.36 | 23.84 | 3.22 | 3.55 | 4.25 | 4.68 | 603.61 | 665.63 |

This observation was true, especially in the initial training rounds. The graphs for Fourier and Waveform Noise datasets are representative of 66% of the instances in which the cascaded algorithms were able to catch up to the convergence rates of the control algorithms and tended to exceed their final training errors very rapidly. In the remaining 33% of the instances that are exemplified by the Pendigit and Morphological datasets, the cascaded algorithms were not capable of producing classifiers which were able to catch up to the control classifiers at any point during their convergences, though they often attained zero training error at some later point in contrast to the control algorithms.

Once again, the slower convergence patterns of the cascaded classifiers are attributed to the extra weak-classifiers being generated by the framework at the points when the learning is focusing on separating one class at a time from the rest. Therefore, the cost of improvement in correctly classifying one class label is not reflected in overall test error rates. The second contributing factor was the re-starting of the boosting process afresh after each node and layer. This resulted in an under-utilization of previously learned information that affects the speed of learning.

In comparison to the control algorithms, the cascaded classifiers displayed convergence rates that were occasionally non-monotonic in nature. This was observed most frequently in the cascaded.M2 classifiers which also displayed the greatest degree of non-monotonic behaviour as well. Of the four cascaded algorithms, cascaded.DP classifiers generally produced the fastest convergence patterns. They tended to be followed by cascaded.OC, cascaded.ECC and cascaded.M2 classifiers respectively.

Lastly, Figure 8.2 once again depicts a subset of graphs from all the datasets in which the efficiency of the training phase in terms of the total error rate as a function of the training runtime was analyzed. On the x-axis, the cost as the training runtime is plotted on a logarithmic scale. The y-axis plots the accuracy on unseen data for all classifiers and across all datasets. The overall results have not greatly diverged from those in the

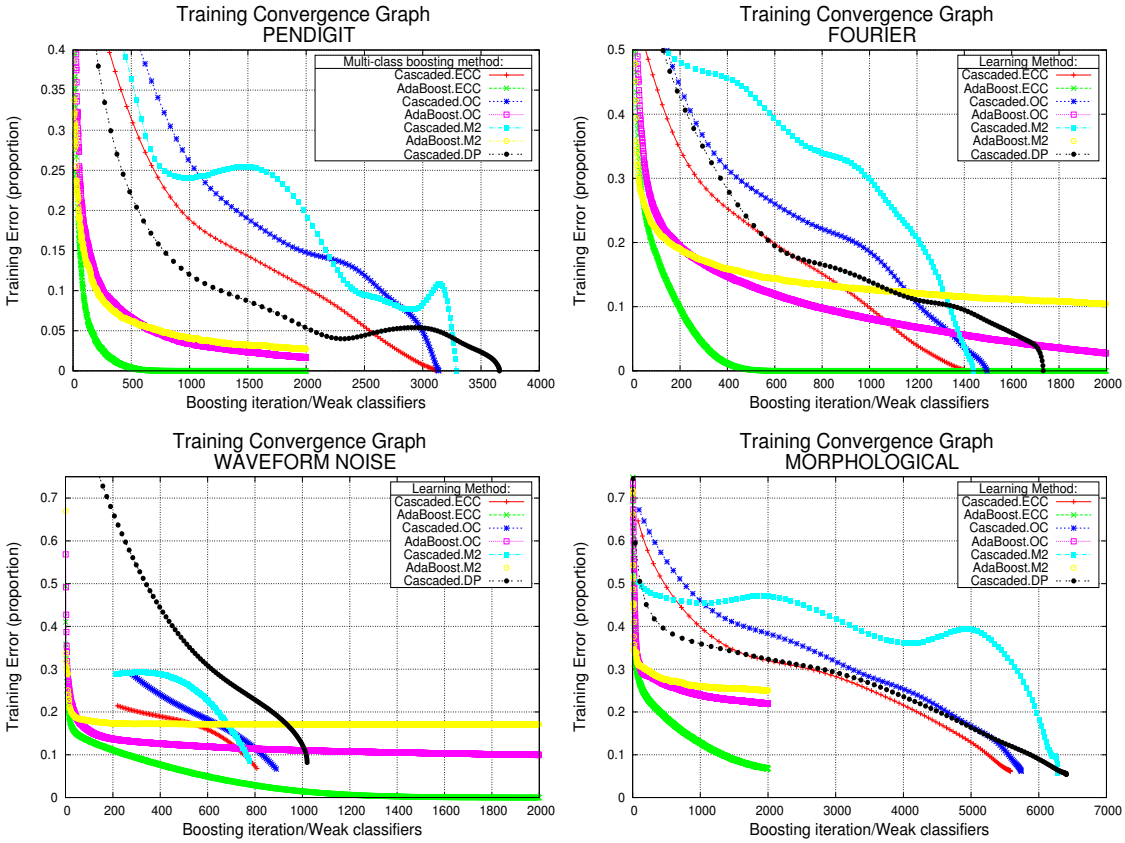



Figure 8.1: Training error convergence graphs.

previous chapter on the cost and efficiency of the cascaded algorithm. The control algorithms dominated the cascaded algorithms in the initial phases. However, the overall results can be summarized as falling into one of two categories; the first displaying patterns where the cascaded classifiers successfully caught up to their respective single-layer algorithms and coming close to parity, while the second category describes an outright stronger generalization and lower cost throughout.

The first category represents some 66% of the datasets. The Vehicle and Pendigit datasets exemplify these trends. The Letter dataset is of particular interest since this is arguably the most difficult dataset as it comprises of the largest number of class labels and samples. On this dataset, the cascaded classifiers outperformed the single layer classifiers even though they generated overwhelmingly larger ensembles.

The Optdigit, Factor and Fourier datasets typify the trends of around 33% of the datasets on which the cascaded classifiers were unable to produce the same training runtime efficiency as well as the final generalization rates as their single-layer equivalents. As expected, the training runtime costs of cascaded.M2 classifiers were greater than those of the other cascaded classifiers. However, comparing it with the original M2 implementation reveals that substantially lower runtime costs were required by the cascaded version in order to attain similar accuracy rates. The convergence patterns of the cascaded.ECC tended to be slightly more efficient than those of cascaded.OC, while the cascaded.DP

algorithm generally produce slightly less efficient classifiers than the previous two.

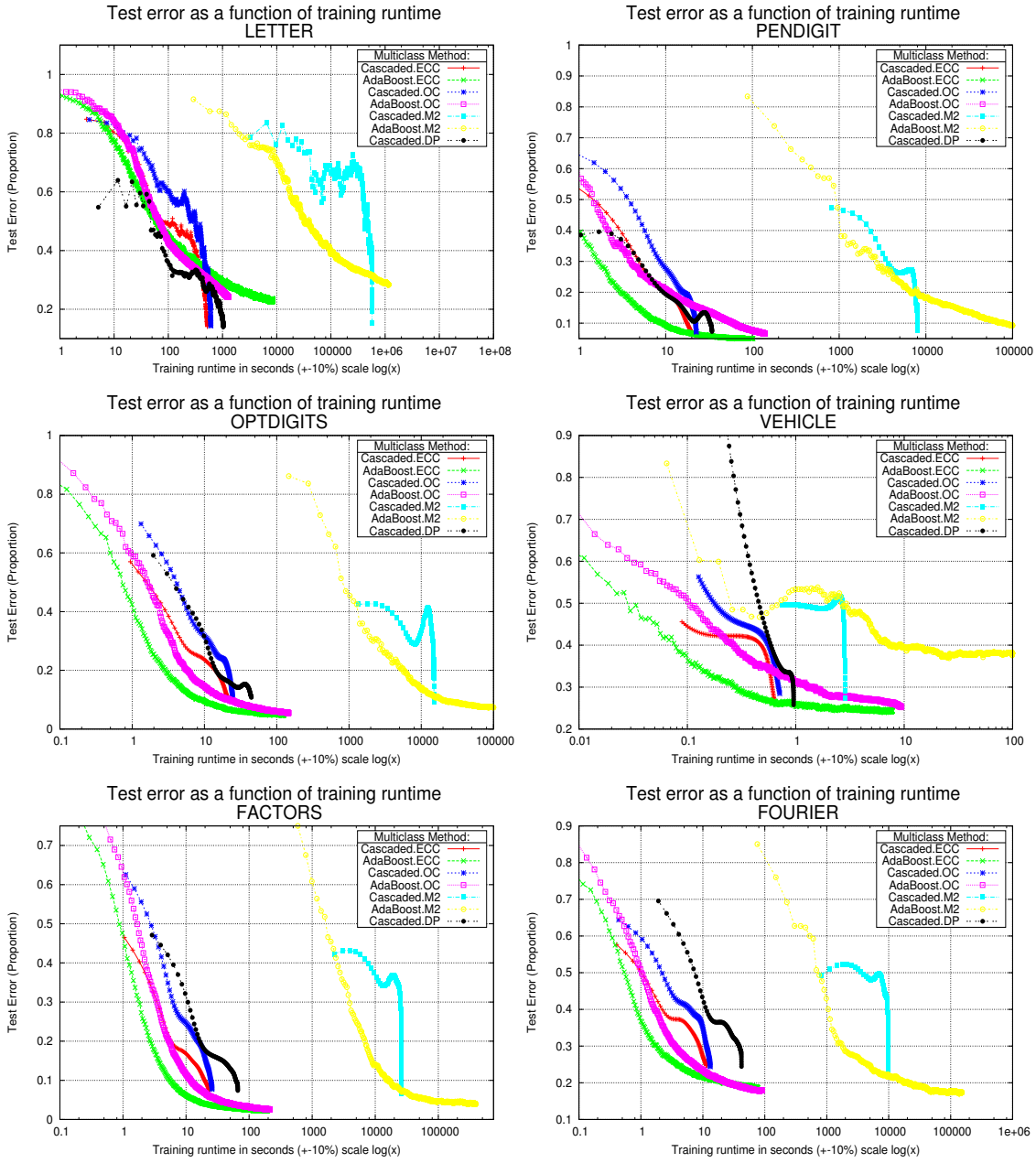



Figure 8.2: Classifier accuracy on test data as a function of training runtime.

## Generalization

Table 8.2 lists classifier accuracies on datasets with balanced class distributions together with the ensemble sizes required to attain them. The best recorded accuracies between the pairs of cascaded and single-layer classifiers are shown in boldface. The accuracies of the cascaded.DP classifiers are in boldface if they exceed each of the OC, ECC and M2 single-layer classifiers. All accuracies from the test error and the g-mean metrics are

Table 8.2: Accuracy on datasets with uniform class distributions. Results are presented as an overall error in proportions, together with the mean deviation. Additionally, the g-mean and the ensemble size values are provided with a summary in the form of average ranks.

| Dataset | Measure | Cascaded ECC | ECC | Cascaded OC | OC | Cascaded M2 | M2 | Cascaded DP |
|---|---|---|---|---|---|---|---|---|
| letter | test error | **0.152 ±0.006** | 0.302 ±0.008 | **0.154 ±0.003** | 0.242 ±0.002 | **0.166** - | 0.306 - | **0.139** - |
| | G-mean | **0.847** | 0.429 | **0.845** | 0.753 | **0.832** | 0.685 | **0.860** |
| | ensemble size | 57762 | 4000 | 58110 | 4000 | 51481 | 4000 | 78956 |
| pendigit | test error | 0.071 ±0.004 | **0.050 ±0.002** | 0.071 ±0.006 | **0.067 ±0.001** | **0.084** - | 0.160 - | 0.071 - |
| | G-mean | 0.929 | **0.950** | 0.928 | **0.933** | **0.915** | 0.914 | 0.927 |
| | ensemble size | 3114 | 2000 | 3138 | 2000 | 3286 | 2000 | 3722 |
| vowel | test error | 0.61 ±0.02 | **0.57 ±0.02** | 0.59 ±0.02 | **0.552 ±0.006** | 0.643 - | **0.574** - | 0.541 - |
| | G-mean | **0.36** | **0.386** | 0.374 | **0.421** | 0.25 | **0.390** | 0.431 |
| | ensemble size | 1107 | 2000 | 1176 | 2000 | 1073 | 2000 | 1368 |
| optdigit | test error | 0.108 ±0.005 | **0.048 ±0.002** | 0.107 ±0.008 | **0.055 ±0.001** | 0.100 - | **0.072** - | 0.108 - |
| | G-mean | 0.890 | **0.951** | 0.892 | **0.944** | 0.899 | **0.93** | 0.891 |
| | ensemble size | 2043 | 2000 | 2107 | 2000 | 2228 | 2000 | 2249 |
| segmentation | test error | 0.047 ±0.008 | **0.030 ±0.008** | 0.06 ±0.01 | **0.03 ±0.01** | **0.05 ±0.03** | 0.06 ±0.01 | 0.039 ±0.005 |
| | G-mean | 0.951 | **0.969** | 0.944 | **0.969** | **0.950** | 0.940 | 0.956 |
| | ensemble size | 678 | 2000 | 709 | 2000 | 773 | 2000 | 978 |
| vehicle | test error | 0.27 ±0.03 | **0.24 ±0.03** | 0.28 ±0.03 | **0.25 ±0.04** | **0.27 ±0.02** | 0.38 ±0.03 | 0.25 ±0.02 |
| | G-mean | 0.707 | **0.732** | 0.689 | **0.720** | **0.700** | 0.574 | **0.729** |
| | ensemble size | 623 | 2000 | 669 | 2000 | 702 | 2000 | 787 |
| iris | test error | **0.06 ±0.04** | 0.07 ±0.04 | 0.07 ±0.05 | **0.06 ±0.04** | **0.05 ±0.04** | 0.05 ±0.05 | 0.08 ±0.02 |
| | G-mean | **0.9390** | 0.932 | 0.9252 | **0.935** | **0.9458** | 0.9451 | 0.9183 |
| | ensemble size | 48 | 500 | 47 | 500 | 45 | 500 | 65 |
| factors | test error | 0.074 ±0.006 | **0.023 ±0.005** | 0.077 ±0.005 | **0.026 ±0.008** | 0.07 ±0.01 | **0.04 ±0.01** | 0.07 ±0.01 |
| | G-mean | 0.9259 | **0.977** | 0.9232 | **0.974** | 0.9256 | **0.960** | 0.9256 |
| | ensemble size | 860 | 2000 | 888 | 2000 | 977 | 2000 | 977 |
| fourier | test error | 0.25 ±0.02 | **0.19 ±0.02** | 0.24 ±0.02 | **0.18 ±0.02** | 0.24 ±0.01 | **0.19 ±0.02** | 0.24 ±0.02 |
| | G-mean | 0.7343 | **0.797** | 0.7394 | **0.806** | 0.7389 | **0.799** | 0.7397 |
| | ensemble size | 1404 | 2000 | 1496 | 2000 | 1437 | 2000 | 2183 |
| waveform | test error | 0.178 ±0.009 | **0.18 ±0.01** | 0.178 ±0.008 | **0.145 ±0.008** | **0.17 ±0.01** | 0.18 ±0.01 | 0.192 ±0.009 |
| | G-mean | **0.8222** | 0.819 | 0.8220 | **0.855** | **0.8275** | 0.8195 | 0.7781 |
| | ensemble size | 1133 | 2000 | 1210 | 2000 | 1091 | 2000 | 1469 |
| waveform noise | test error | 0.184 ±0.009 | **0.174 ±0.007** | 0.179 ±0.009 | **0.14 ±0.01** | 0.20 ±0.02 | **0.18 ±0.01** | 0.19 ±0.01 |
| | G-mean | 0.816 | **0.825** | 0.820 | **0.856** | 0.803 | **0.816** | 0.796 |
| | ensemble size | 808 | 2000 | 890 | 2000 | 778 | 2000 | 1050 |
| morphological | test error | 0.35 ±0.02 | **0.33 ±0.03** | 0.35 ±0.02 | **0.29 ±0.03** | 0.35 ±0.03 | **0.29 ±0.03** | 0.35 ±0.03 |
| | G-mean | 0.593 | **0.617** | 0.597 | **0.671** | 0.590 | **0.670** | 0.593 |
| | ensemble size | 5582 | 2000 | 5741 | 2000 | 6280 | 2000 | 6857 |
| Test Error Mean Rank | | 4.58 | 2.5 | 5.08 | **2.17** | 4.67 | 4.58 | 4.33 |
| G-mean Mean Rank | | 4.5 | 2.92 | 4.83 | **2.08** | 4.83 | 4.25 | 4.42 |

summarized as average ranks.

By considering the comparative accuracy between all algorithms, the OC algorithm generated classifiers which displayed both the best mean rank on the both the test error and the g-mean metrics. This was clsely followed by the ECC classifiers, while the remainder of the algorithms were clearly separated from the two leading algorithms by a sizeable margin.

It was observed from the ensemble sizes that the cascaded algorithm produced overly complex classifiers on datasets with large numbers of class labels. This was partucularly eccentuated on the Letter dataset which contains the largest number of class labels. To a lesser extent, this was also observable on the Morphological and Pendigit datasets. On datasets with small numbers of class labels, the cascaded framework generated simpler solutions.

Statistical tests were applied to the mean ranks in order to determine if their differences were statistically significant. Table 8.3 provides the summary of the values. Based on the Friedman $\chi_F^2$ and the Iman-Davenport $F_F$ tests, the differences between the mean ranks for the test error were significant at a $\alpha < 0.005$ and $\alpha < 0.0025$ levels respectively. Based on this, the null hypothesis was rejected which enabled the analysis to proceed with further *post-hoc* tests.

Table 8.3: Results of the Friedman and Iman-Davenport tests with the critical differences based on the Nemenyi test for analyzing the mean ranks of the accuracy results from Table 8.2.

| | Friedman Value $\chi_F^2$ | Value in $\chi^2(6)$ | | Iman-Davenport Value $F_F$ | Value in $F_F(6,66)$ | | Nemenyi Test Critical Difference | |
|---|---|---|---|---|---|---|---|---|
| | | $\alpha \leq 0.005$ | $\alpha \leq 001$ | | $\alpha \leq 0.0025$ | $\alpha \leq 0.001$ | $\alpha \leq 0.1$ | $\alpha \leq 0.05$ |
| test error | 18.911 | 18.55 | 22.46 | 3.918 | 3.82 | 4.32 | 2.375 | 2.6 |
| | | $\alpha \leq 0.05$ | $\alpha \leq 0.01$ | | $\alpha \leq 0.025$ | $\alpha \leq 0.01$ | $\alpha \leq 0.1$ | $\alpha \leq 0.05$ |
| G-mean | **13.857** | 12.59 | 16.812 | **2.622** | 2.61 | 3.09 | 2.375 | 2.60 |

The Nemenyi test produced a critical difference value of 2.6 for a significance level of $\alpha < 0.05$. At this level it was possible to conclude that the OC performed better than cascaded.OC on the test error metric. In addition, at a less significant level of $\alpha < 0.1$, it was possible to show that OC was more accurate than cascaded.ECC and M2 as well, while ECC was more accurate than cascaded.OC.

For the g-mean metric, the critical difference was sufficient to conclude that the OC generalized better than both the cascaded.OC and cascaded.M2 at $\alpha < 0.05$. At a less significant level of $\alpha < 0.1$, the accuracy of cascaded.ECC could also be included in this conclusion. Beyond this it was not possible to come to any more conclusions with the given test.

In order to focus solely on pairwise comparisons between the cascaded and the corresponding single-layered algorithms, the Wilcoxon signed-ranks test was applied to the results from Table 8.2. For this test only the test error accuracies was used. The cascaded.DP classifiers were compared to each one of the single-layer classifiers.

In comparing the accuracies between the cascaded.OC and the OC classifiers, the Wilcoxton test (Table 8.4) confirmed that the generalization of the latter was stronger

across the 12 datasets at a significance level of $\alpha < 0.05$. Meanwhile, the accuracy of ECC was stronger than that of the cascaded.ECC at a $\alpha < 0.1$ level. Regarding the remaining pairwise comparisons, no further conclusion could be reached using this test except that the accuracies between them were indistinguishable.

Table 8.4: Wilcoxon signed-ranks test for results from datasets with balanced class-distributions. Statistically significant results are accompanied by a ✓.

| **Comparison** | **R+** | **R-** | **significance** |
|---|---|---|---|
| Cascaded.ECC versus ECC | 14 | 64 | **0.1** ✓ |
| Cascaded.OC versus OC | 12 | 66 | **0.05** ✓ |
| Cascaded.M2 versus M2 | 40 | 38 | $> 0.5$ |
| Cascaded.DP versus ECC | 41 | 37 | $> 0.5$ |
| Cascaded.DP versus OC | 19 | 59 | 0.2 |
| Cascaded.DP versus M2 | 20 | 58 | 0.2 |

The accuracy of the algorithms on datasets with biased-class distributions were next considered and their results are shown in Table 8.5. The g-mean was in this case the only reliable measure of generalization; however, the test error is also shown for completeness. The summary of the table in the form of mean ranks showed that a change in overall trends had occurred in comparison to the results from the unbiased-class distribution datasets. The mean ranks indicated that the comparative performance of the cascaded algorithms had significantly improved on the more difficult datasets. While the single-layer ECC registered the best mean rank, it was followed by the cascaded.OC and subsequently by the joint performances of the cascaded.M2 and OC algorithms. The change in the trends was partially influenced by the complexity of some datasets like Yeast. On this dataset OC and M2 were not able to produce any detections for at least one of the class labels which meant that no meaningful g-mean values could be generated.

A detailed analysis of the generalization of the classifiers on the Yeast dataset for each class label is shown in the form of the F-measure analysis in Table 8.6. The highest accuracies from a pairwise comparison between the cascaded and the corresponding single-layer algorithms are in boldface and the minority class-labels of interest are signified by a •. The cascaded.ECC classifiers in particular demonstrated and increased ability to focus learning minority class labels than ECC. A similar trend was observed between the cascaded.OC and OC classifier but not to the same extent. Exceptions occurred on class labels 3 and 5 on which OC performed better. M2 displayed a reversal to these trends where it recorded higher accuracies on majority of the class labels over cascaded.M2. However, both OC and M2 neglected to learn minority class labels 7 and 9 while their cascaded version successfully generated detections for these class labels.

The F-measure results seen on the Yeast dataset generally favoured the cascaded classifier and this was consistent with half of the biased datasets, while the remaining half of the datasets were more favourable to the single layer classifiers of which the Satimage in Table 8.7 is representative. On this dataset, the trends were reversed and it can be clearly seen that both the ECC and OC classifiers have generalized better than their cascaded equivalents. Meanwhile, the cascaded.M2 classifier displayed a slightly better

Table 8.5: Accuracy on datasets with biased class distributions. The g-mean is the primary assessment metric though the overall error in proportions, together with the mean deviation is also listed. The ensemble size and the summary of the table in the form of mean ranks, based on the g-mean, is also provided.

| Dataset | Measure | Cascaded ECC | ECC | Cascaded OC | OC | Cascaded M2 | M2 | Cascaded DP |
|---|---|---|---|---|---|---|---|---|
| satimage | G-mean | 0.8316 | **0.8386** | 0.8319 | 0.837 | **0.83** | 0.8 | **0.83** |
| | test error | 0.149 ±0.003 | **0.138 0.003** | 0.150 0.004 | **0.134 0.001** | **0.140** - | 0.182 - | **0.145** - |
| | ensemble size | 1894 | 2000 | 1965 | 2000 | 2205 | 2000 | 2608 |
| shuttle | G-mean | 0.976 | **0.989** | 0.974 | **0.989** | **1.000** | 0.973 | 0.949 |
| | test error | 0.00018 ±0.00009 | **0.00007** - | 0.00021 ±0.00007 | **0.00007** - | **0.0** - | 0.00055 - | **0.00028** - |
| | ensemble size | 640 | 2000 | 586 | 2000 | 749 | 2000 | 437 |
| glass | G-mean | 0.536 | **0.649** | **0.603** | 0.451 | **0.533** | 0.429 | **0.667** |
| | test error | 0.34 ±0.07 | **0.28 ±0.08** | 0.33 ±0.10 | 0.32 ±0.09 | 0.35 ±0.08 | **0.34 ±0.09** | **0.32 ±0.03** |
| | ensemble size | 351 | 500 | 424 | 500 | 474 | 500 | 360 |
| page blocks | G-mean | 0.751 | **0.809** | 0.792 | **0.778** | **0.735** | 0.652 | 0.727 |
| | test error | 0.038 ±0.005 | 0.031 ±0.004 | 0.037 ±0.005 | **0.030 ±0.004** | 0.040 ±0.006 | **0.040 ±0.004** | 0.040 ±0.005 |
| | ensemble size | 1635 | 2000 | 1637 | 2000 | 1128 | 2000 | 1851 |
| robot navigation | G-mean | 0.978 | **0.996** | 0.983 | **0.994** | 0.982 | **0.994** | 0.978 |
| | test error | 0.015 ±0.002 | **0.002 ±0.001** | 0.011 ±0.003 | 0.003 ±0.002 | 0.011 ±0.003 | **0.003 ±0.002** | 0.012 ±0.004 |
| | ensemble size | 403 | 2000 | 430 | 2000 | 663 | 2000 | 988 |
| yeast | G-mean | **0.376** | 0.331 | **0.373** | - | **0.35** | - | **0.34** |
| | test error | **0.49 ±0.02** | 0.49 ±0.03 | **0.50 ±0.02** | **0.41 ±0.04** | 0.49 ±0.05 | **0.42 ±0.04** | 0.491 ±0.004 |
| | ensemble size | 8335 | 2000 | 8614 | 2000 | 6865 | 2000 | 7359 |
| G-mean Mean Rank | | 4.17 | **2** | 3.33 | 3.67 | 3.67 | 5.83 | 5 |

generalization pattern than M2. The remaining F-measure tables are located in Appendix D.

By returning to Table 8.5 and to the summary of the g-mean results, the Friedman and the Iman-Davenport statistics were calculated in order to determine if the ranks significantly diverged from the expected mean ranks. The relevant statistical results are shown in Table 8.8. The results of both tests revealed that there did not exist enough evidence to reject the null hypothesis. Therefore, the null-hypothesis is accepted which states that the mean ranks from the algorithms are sufficiently similar.

The final remaining statistical test was the Wilcoxon signed-ranks test on the g-means results for a more detailed pairwise comparisons (Table 8.9). The outcomes of this test enabled one conclusion to be reached. At a $\alpha < 0.1$ significance level, the test found that the cascaded implementation generalized better on the given datasets than the M2 classifiers.

### Detection Runtime Phase

Table 8.10 lists the detection runtimes for all classifiers across all datasets. The results are once again shown as a factor of speed-up that the cascaded classifiers achieved over the single-layered classifiers. The overall results in the form of the geometric mean indicates

Table 8.6: F-Values for the Yeast dataset. Minority classes of interest are denoted as •and an observed accuracy improvement over the single-layer classifiers as ✓.

| Class | Distr. | Cascaded ECC | ECC | Cascaded OC | OC | Cascaded M2 | M2 | Cascaded DP |
|---|---|---|---|---|---|---|---|---|
| 0 | 16.4 | 0.486 | 0.514 | 0.480 | 0.586 | 0.474 | 0.591 | 0.454 |
| 1 | 28.9 | 0.500 | 0.494 | 0.488 | 0.565 | 0.507 | 0.561 | 0.473 |
| 2 | 31.2 | 0.521 | 0.508 | 0.505 | 0.57 | 0.529 | 0.563 | 0.508 |
| 3 • | 3.0 | **0.595** ✓ | 0.546 | 0.599 | **0.679** | 0.556 | **0.667** | 0.575 |
| 4 • | 2.4 | **0.445** ✓ | 0.387 | **0.431** ✓ | 0.590 | 0.380 | **0.526** | 0.529 |
| 5 • | 3.4 | **0.344** ✓ | 0.336 | 0.355 | **0.374** | 0.225 | **0.351** | 0.356 |
| 6 | 11.0 | 0.706 | 0.723 | 0.696 | 0.799 | 0.736 | **0.801** | 0.699 |
| 7 • | 2.0 | **0.035** ✓ | 0.031 | **0.032** ✓ | 0.000 | **0.031** ✓ | 0.000 | **0.033** |
| 8 • | 1.3 | **0.389** ✓ | 0.380 | **0.389** ✓ | 0.173 | 0.348 | **0.429** | 0.200 |
| 9 • | 0.3 | **0.600** ✓ | 0.297 | **0.552** ✓ | 0.000 | **0.571** ✓ | 0.000 | **0.400** |

Table 8.7: F-Values for the Satimage dataset. Minority classes of interest are denoted as •and an observed accuracy improvement over the single-layer classifiers as ✓.

| Class | Distr. | Cascaded ECC | ECC | Cascaded OC | OC | Cascaded M2 | M2 | Cascaded DP |
|---|---|---|---|---|---|---|---|---|
| 0 | 24.29 | 0.833 | 0.844 | 0.834 | 0.844 | 0.851 | 0.813 | 0.867 |
| 1 | 22.37 | 0.965 | 0.978 | 0.971 | 0.974 | 0.965 | 0.957 | 0.975 |
| 2 • | 11.23 | 0.952 | **0.970** | 0.952 | **0.971** | 0.963 | **0.964** | 0.931 |
| 3 | 21.33 | 0.877 | 0.875 | 0.873 | 0.902 | 0.871 | 0.896 | 0.859 |
| 4 • | 10.02 | 0.584 | **0.592** | 0.588 | **0.590** | **0.590** ✓ | 0.515 | 0.573 |
| 5 • | 10.76 | 0.842 | **0.860** | 0.832 | **0.840** | **0.853** ✓ | 0.798 | 0.838 |

Table 8.8: Results of the Friedman and Iman-Davenport tests with the critical differences based on the Nemenyi test for analyzing the mean ranks of the accuracy results from Table 8.5.

| | Friedman Value $\chi_F^2$ | Value in $\chi^2(6)$ | | Iman-Davenport Value $F_F$ | Value in $F_F(6,30)$ | Nemenyi Test Critical Difference | |
|---|---|---|---|---|---|---|---|
| | | $\alpha \leq 0.1$ | $\alpha \leq 0.05$ | | $\alpha \leq 0.1$ | $\alpha \leq 0.1$ | $\alpha \leq 0.05$ |
| G-mean | 8.214 | 10.645 | 12.591 | 1.478 | 2.42 | - | - |

Table 8.9: Wilcoxon signed-ranks test for results from datasets with biased class-distributions. Statistically significant results are accompanied by a ✓.

| **Comparison** | **R+** | **R-** | **significance** |
|---|---|---|---|
| Cascaded.ECC versus ECC | 4 | 17 | $> 0.2$ |
| Cascaded.OC versus OC | 14 | 7 | $> 0.2$ |
| Cascaded.M2 versus M2 | 20 | 1 | **0.1** ✓ |
| Cascaded.DP versus ECC | 5 | 16 | $> 0.2$ |
| Cascaded.DP versus OC | 11 | 10 | $> 0.2$ |
| Cascaded.DP versus M2 | 18 | 3 | **0.2** |

that the cascaded classifiers achieved a considerable acceleration in detection runtimes over the single-layer classifiers. The process of converting single-layered ensemble algorithms to cascades can be expected to be accompanied by an acceleration of the detection runtimes that is in the vicinity of 4-5 times the normal runtimes.

Most of the exceptions to the general trends occurred on the Letter dataset. The detection runtimes of all single-layered classifiers were either at parity or faster than the per-ensemble execution runtimes of the cascaded classifiers. This has been a recurring trend for this datasets and once again the slower runtimes are attributed to the susceptibility of the multiclass cascaded classifier to produce overly complex ensembles on datasets with large numbers of class labels.

Exceptions occurred on four other occasions where the single-layered classifiers this time round produced faster execution runtimes on the per-weak classifier metric; however, in each case, these results were balanced by the faster overall runtimes of cascaded classifiers for the entire ensembles.

## 8.5 Discussion

The modularity of the proposed multiclass learning framework introduces higher interpretability of the final classifier. As a meta-learning algorithm, it is inducer flexible. This means that it can also be applied to other popular methods such as neural networks and support vector machines. Theoretically, it has the capacity to combine multiple types of inducers at different stages of cascade construction in order to better balance the computational and accuracy requirements.

The cascaded framework is scalable to large datasets; however, the scalability does not extend to class labels. In the experiments, the Letter dataset with 26 class labels demonstrated the upper limits of this range. The tests have shown that as class numbers increase, the size of the cascaded classifier grows both horizontally within each layer, as well as vertically in respect to the number of cascade layers. Consequently, both the training and the execution runtimes become more expensive, while the final classifier may also become overly complex.

Conversely, due to the modularity of the framework, it is well suited to training on massive datasets. With minor modifications, the proposed framework can process large

Table 8.10: The factor by which the execution runtimes are increased by the cascaded frameworks over the single-layer algorithms. The •denotes the factor of increase for instances where the reverse occurred and ✓signifies where it was balanced on the associated metric. Value 1.0 represents parity. Values below 1.0 signify a slower runtime

| | cascaded.ECC vs ECC | | cascaded.OC OC | | cascaded.M2 vs M2 | | cascaded.DP vs ECC | | cascaded.DP vs OC | | cascaded.DP vs M2 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Dataset | weak clsf. | per ens. | weak clsf. | per ens. | weak clsf. | per ens. | weak clsf. | per ens. | weak clsf. | per ens. | weak clsf. | per ens. |
| letter | 4.2 ✓ | 0.3 • | 14.5 | 1.0 | 8.7 ✓ | 0.7 • | 18.2 ✓ | 0.9 • | 8.5 ✓ | 0.4 • | 6.8 ✓ | 0.3 • |
| pendigits | 3.6 | 2.3 | 6.4 | 4.1 | 6.9 | 4.2 | 6.8 | 3.6 | 3.1 | 1.7 | 3.8 | 2.1 |
| satimage | 5.6 | 5.9 | 5.3 | 5.4 | 4.3 | 3.9 | 7.6 | 5.8 | 3.9 | 3.0 | 3.8 | 2.9 |
| vowel | 0.9 • | 1.7 ✓ | 3.0 | 5.1 | 1.8 | 3.4 | 2.3 | 3.3 | 1.1 | 1.6 | 1.1 | 1.6 |
| optdigits | 2.4 | 2.3 | 5.0 | 4.8 | 4.9 | 4.4 | 4.2 | 3.7 | 2.0 | 1.7 | 2.5 | 2.2 |
| segmentation | 2.2 | 6.6 | 4.5 | 12.6 | 4.2 | 10.7 | 3.3 | 6.8 | 1.7 | 3.5 | 1.8 | 3.6 |
| vehicle | 6.0 | 19.3 | 3.9 | 11.6 | 2.0 | 5.8 | 5.5 | 14.0 | 2.2 | 5.6 | 1.5 | 3.8 |
| glass | 1.8 | 2.6 | 2.5 | 2.9 | 2.3 | 2.4 | 2.5 | 3.4 | 0.9 | 1.3 | 1.1 | 1.5 |
| iris | 1.9 | 20.3 | 1.9 | 20.7 | 1.5 | 17.1 | 1.3 | 10.1 | 0.6 • | 4.6 ✓ | 0.6 | 4.4 |
| factors | 1.1 | 2.6 | 2.9 | 6.6 | 4.2 | 8.6 | 9.2 | 18.8 | 4.1 | 8.4 | 5.5 | 11.2 |
| fourier | 1.7 | 2.4 | 4.1 | 5.5 | 3.1 | 4.3 | 15.2 | 13.9 | 7.2 | 6.6 | 7.5 | 6.9 |
| robotnavigation | 3.5 | 17.3 | 3.0 | 14.1 | 2.7 | 8.3 | 13.2 | 26.7 | 7.2 | 14.6 | 7.4 | 15.0 |
| waveform | 5.6 | 9.8 | 3.4 | 5.7 | 2.3 | 4.2 | 13.7 | 18.6 | 8.2 | 11.1 | 5.0 | 6.7 |
| waveformnoise | 4.7 | 11.5 | 3.0 | 6.8 | 2.1 | 5.3 | 8.5 | 16.3 | 5.3 | 10.2 | 3.1 | 5.9 |
| yeast | 5.6 | 1.4 | 5.2 | 1.2 | 4.3 | 1.2 | 25.1 | 6.8 | 11.4 | 3.1 | 13.0 | 3.5 |
| morphological | 5.9 | 2.1 | 8.5 | 3.0 | 7.4 | 2.3 | 54.6 | 15.9 | 25.4 | 7.4 | 27.2 | 7.9 |
| pageblocks | 5.1 | 6.2 | 3.2 | 3.9 | 2.3 | 4.1 | 12.1 | 13.1 | 6.3 | 6.8 | 3.8 | 4.2 |
| shuttle | 1.6 | 5.1 | 2.6 | 8.8 | 2.2 | 6.0 | 1.2 | 5.4 | 0.6 • | 2.8 ✓ | 0.7 • | 3.2 ✓ |
| Mean | 3.5 | 6.7 | 4.6 | 6.9 | 3.7 | 5.4 | 11.4 | 10.4 | 5.5 | 5.2 | 5.3 | 4.8 |
| Std. Dev | 1.8 | 6.4 | 2.9 | 5.0 | 2.1 | 3.9 | 12.6 | 7.1 | 5.9 | 3.9 | 6.3 | 3.7 |
| Geometric Mean | 3.0 | 4.1 | 4.0 | 5.3 | 3.3 | 4.3 | 7.1 | 7.8 | 3.4 | 3.8 | 3.3 | 3.6 |

datasets through continuous sample bootstrapping. This can be implemented by introducing new samples at the level of nodes and layers, since the learning is independent at each step.

The independence between nodes and layers can also be leveraged in order to achieve adaptive learning in dynamic environments. As class descriptors undergo changes in nonstationary environments, individual nodes with a degradation in accuracy can be easily identified. Subsequently, they can be efficiently re-trained without negatively affecting the rest of the ensemble.

The decoupled relationship between the nodes and layers has its advantages but it also comes at a price. The trade-off for the high modularity and independence of the classifier components is a decrease in efficiency of both the training and the execution of a classifiers, as well as possible compromises in accuracy.

The slower convergence patterns of the cascaded classifiers in the initial phases, as witnessed in the experiments, can be partially attributed to the effects of discarding previously learned information in the form of sample weights, between nodes and layers. Potentially, the loss of previously learned information can also compromise the accuracy of classifiers since the growth of confidence margins between the decision boundaries of the class labels are restricted.

Additionally, a more sophisticated vote-combination method for the cascaded classifier, which integrates predictions of previous nodes and layers can arguably decrease the execution runtime of the classifiers even further. With this approach, a prediction of a class label could be arrived at in earlier layers of a cascade, given that the confidence of a prediction at a given point in the cascade exceeds a certain confidence threshold. This decision threshold can be learned at training and adjusted during runtime.

Interesting possibilities for future research into extending the cascading framework

have become apparent. Firstly, there exists the ability to extend the framework in such a way that it becomes explicitly computationally-aware. The intention is to explore ways of adding the capability to the system to train classifiers with runtime constraints for either the learning phase or detection phase, while taking into account the trade-offs in accuracy. The idea is to allow the nodes to be adaptive in their complexity at different stages of granularity.

Furthermore, the modularity and the independence of the nodes makes the cascading framework a natural fit for emerging areas like data fusion where complementary information from different data sources and features is integrated. It follows that this approach holds much potential for improving coarse-to-fine strategies in which both accuracy and runtime complexities can be balanced through the use of different feature types at various stages of granularity.

Lastly, up to this point, the coarse-to-fine learning has been achieved through decisions based on training error rates. Though this approach has often yielded satisfactory generalization for performance trade-offs, there is scope for improving the accuracy of the cascading methods by replacing the error-based decomposition with margin-based decisions which instead reflect confidences.

## 8.6 Summary

The experiments in this chapter have demonstrated how the multiclass cascading framework can be used as a meta-learning algorithm for converting existing single-layer ensemble-based algorithms into cascaded classifiers. The research here has shown how AdaBoost.OC, AdaBoost.ECC and AdaBoost.M2 can be converted into cascading algorithms with the results that:

1. The training runtime is accelerated.
2. Classifier detection runtime is decreased.
3. Some generalization compromises were are trade-off when converting OC and ECC into the proposed cascaded structure on datasets with balanced-class distributions.
4. Accuracy was preserved when converting the single-layer classifiers into proposed cascaded classifiers for minority classes on datasets with biased-class distributions. The results demonstrated that the cascading even slightly improved the accuracy on these datasets over the M2 algorithm

# Chapter 9

# Conclusion

The focus of this thesis was boosted ensemble-based learning methods, in the context of cascaded frameworks. By taking the Viola-Jones and the PSL cascading approaches to structuring an ensemble into different layers as a starting point, the emphasis of this research was to develop techniques that introduced more aggressive separate-and-conquer strategies to the learning process. The end goal was the reduction in the complexity of the classifier training task that would translate to shorter learning phases for problem domains with large and difficult datasets, while producing real-time detection capable classifiers.

The driving force behind the motivations of each chapter was the quest to investigate how far the training procedures of cascaded approaches could be further modularized in order to introduce higher degrees of coarse-to-fine learning while maintaining robust classifiers. Contributions were made in three distinct areas:

1. The conventional bootstrapping procedure for cascaded binary-class classifiers, that only involves negative training samples, was expanded to include the ability to bootstrap positive samples as well. This enables the coarse-to-fine learning to be applied to positive samples, with the consequence that much large positive datasets can be used with considerably lower training runtime overheads.

2. An adaptive learning algorithm was devised for the proposed cascaded classifiers in a high-volume binary-class domain with drifting concepts. The goal was to develop a method that is capable of rapidly adapting to the changing environment without requiring the explicit re-training of classifiers due to time constraints.

3. An algorithm for creating integrated multiclass ensemble-cascades was developed with an underlying separate-and-conquer strategy. It was also empirically shown how it can be used as a meta-learning algorithm for converting existing single-layered multiclass learning methods into cascaded ensembles.

Chapter 4 set out to compare the PSL algorithm against the conventional Viola-Jones cascaded approach to training classifiers on a real-world problem of face detection. The experiments sought to answer the key questions regarding the extent to which the training runtimes are affected by the PSL approach and whether trade-offs are involved with respect to the generalizability of its classifiers. Additional investigation was also conducted aimed to determine the effect that the primary training parameter $\Phi$ for regulating the size of the nodes have on the final classifiers.

The results showed that the PSL algorithm does significantly reduce the training runtimes of the classifiers compared to the conventional approach. The effect on the training runtimes became more acute as the training datasets' volume increased. In addition, considerably smaller ensembles were also generated by the PSL method resulting in faster detection runtimes. It was discovered that the number of advantages displayed by the PSL over the conventional approach were indeed accompanied by trade-offs in respect to the final test accuracy. The conventional approaches were observed to achieve a higher accuracy by several percentage points on all datasets. The PSL classifiers trained with smallest nodes displayed a tendency towards producing the lowest generalization results with also slightly reduced training runtimes. However, classifiers with large nodes did not indicate that an increase in the tunable parameter $\Phi$ values necessarily translated to better generalization rates.

The subsequent research in Chapter 5 considered the use of much larger training datasets than in the preliminary experiments. There were several questions the research sought to answer. Firstly, the aim was to investigate whether the accuracy of PSL would improve substantially in relation to the conventional cascade approaches when using considerably larger datasets for the problem of face detection. Secondly, the goal was to ascertain if the PSL-like training structure could be expanded in to enable positive sample bootstrapping with the ideal outcome of reduced training runtimes without causing an accuracy deterioration. Lastly, the objective of identifying root causes of a deteriorating accuracy was set with the aim of finding strategies to mitigate it.

The experiments indicated that while the PSL significantly lowered the training runtime overhead over the conventional approaches on large datasets, this was once again realized with a notable accuracy reduction. The bootstrapping component resulted in the new algorithm termed BPSL. The experimental findings showed that it was capable of implementing the bootstrapping of positive samples with no evidence of compromising the generalization, while considerably reducing the training runtimes of PSL. Further analysis of the PSL-based algorithms identified the susceptibility of the algorithm to overfit the data because of its aggressive separate-and-conquer strategy of iteratively reducing the size of the learning problem. The findings showed that PSL-based approaches apply informed-sampling of the training samples which clusters instances of the positive dataset within each layer into subsets of increasingly difficult patterns, coupled with a class-imbalanced learning task. It was discovered that as each node within a given cascade layer was trained, the proportion of positive samples with substantial in-class variability and noise also increased, resulting in undue learning emphasis being devoted to unrepresentative patterns.

The chapter proposed two effective strategies for overcoming the susceptibility of the PSL methods to overfit in the presence of high in-class variability and large proportion of noisy samples. The first strategy consisted of ensemble thinning, whereby a specified number of trailing nodes within each layer were removed. This strategy involved no post-processing overheads while improving the accuracy of all classifiers. The second strategy employed informed-resampling in order to address the imbalanced learning that takes place in the trailing nodes with the combination of unrepresentative samples. The informed-resampling method significantly improved the accuracy of the PSL-based algorithms. PSL-based training in combination with informed-resampling provided an effective alternative

to the conventional cascade training approach with more limited compromises in accuracy in return for notable training runtime reductions.

The problems associated with statically trained classifiers operating in dynamic and non-stationary environments was addressed in Chapter 6. The accuracy of static classifiers invariably deteriorates in environments with unforeseen drifts to the descriptors of target concepts. The research explored the possibility of devising a strategy for PSL-like classifiers which can utilize the clustering arrangement of the weak classifiers within each layer in order to realize the ability of the classifiers to adapt to changes in the environment in real-time. The experiments sought to determine the capacity of the proposed algorithm to achieve adaptability of PSL-like classifiers to diverse degrees of concept drift without requiring explicit re-training of weak classifiers or forgetting of previously learned information.

The experiments simulated a real-time critical and high volume domain in the problem of face detection. The proposed algorithm assigned competence values to all nodes and parameterized the cascade layers with learnable confidence thresholds. The experiments showed how rapid and accurate adaptability to changes in the environment of any degree could be realized by modifying the confidence thresholds for the decisions at each layer. The adaptability was real-time achievable since individual weak classifier weights were neither modified nor considered in the calculations, but instead the collective competences of the nodes was employed. As a consequence, the statically learned information was also preserved.

In Chapter 7, the focus of the research turned away from binary-class to multiclass learning. The aim here was to explore methods of employing the PSL-like principles to implementing effective and efficient multiclass algorithms using weak learners. The goal was to assess the plausibility of using the separate-and-conquer strategies in order to aid the learning by both simplifying the class boundaries and reducing training runtimes. Questions were posed surrounding the ability of this method to preserve accuracy and its ability to produce more rapid detection runtimes.

The investigation involved popular benchmark boosting algorithms comprising of AdaBoost.M2, AdaBoost.OC and AdaBoost.ECC. The experimental findings showed that PSL-based cascading was indeed effective at formulating a solution for multiclass learning problems using weak learners. The proposed algorithm showed how an iterative separate-and-conquer approach to learning class labels generated accurate classifiers which were comparable to the control methods. The training runtimes were reduced with the decreasing size of nodes being a primary determinant of the degree of the reduction. Smallest nodes also produced the fastest detection runtimes and exceeded the performances of the control classifiers; however, there was some evidence to suggest that on average, the accuracy of cascaded classifiers decreased with the reduction in the complexity of the node sizes. The proposed algorithm was found not to be scalable to problem sets with large numbers of class labels due to the its susceptibility in such instances to generate over-inflated ensembles. Smaller multiclass nodes limit the size of the final ensemble and since there was no strong correlation discovered between node size parameters and the generalization of the classifiers, smaller node sizes were found to be preferable.

Finally, given the favourable results of the multiclass-PSL algorithm, Chapter 8 considered the usability of the proposed algorithm as a meta-learning method for transforming

existing single-layer multiclass algorithms into cascaded algorithms. The experiments involved replacing the domain-partitioning weak learner used at each multiclass node with that of AdaBoost.M2, AdaBoost.OC and AdaBoost.ECC. The results demonstrated that the coarse-to-fine learning of the PSL-like approaches could effectively be applied to existing single-layer boosting algorithms. The outcomes of this were reduced training and detection runtimes of the resulting classifiers. While there was no evidence of a change in the generalization of converting AdaBoost.M2 to a cascaded structure, there was a statistically significant deterioration in the accuracy of AdaBoost.OC and AdaBoost.ECC when converting into cascades on datasets with balanced class distributions. Meanwhile, on class-balanced datasets, the effect of implementing the single-layer algorithms into cascades did not indicate any change in their accuracy.

## 9.1 Recommendations

Given binary-class problems with large training sets, the usage of BPSL method is recommended over the original PSL algorithm due to the ability of the former to significantly reduce the training runtimes through the bootstrapping of the positive samples, while also preserving accuracy. Since optimal tunable parameter values governing the size of the nodes and the positive sampled sets for bootstrapping for any given problem domain are unable to be determined *a priori*, several classifiers with different parameter settings should be trained. From this pool of classifiers, the best performing classifier should then be selected. For performance reasons, smaller values for both tunable parameters were found to be preferable.

It is also highly advisable to train BPSL classifiers with informed-resampling for the trailing nodes in order to mitigate the effects of noisy samples or the presence of substantial in-class variability of the target object. In order to ensure that overfitting has not occurred in the trailing nodes of each layer, classifier accuracy should be verified by combining with the ensemble thinning procedure outlined in Chapter 5. In the event that detection runtimes of classifiers are of primary importance, the tuning of the classifiers for optimal execution speed should be performed by employing variable nodes dimensions for different cascade layers. The initial layers should be set to minimal node sizes, which can gradually be increased with each new layer. Experiments have shown that though this has a significant effect on the runtimes of classifiers, their generalizability is largely unaffected.

In the area of classifier adaptability to changing environmental conditions, it is recommended that small snapshots of streaming data be used, since they have been shown to be adequate for achieving the required accuracy improvements and carry the smallest computational overheads.

It is advisable to employ the multiclass-PSL method on problem sets which are large but which however do not exceed 10 class labels, due to the tendency of the algorithm to generate overly-complex ensembles. Since the principle behind the algorithm is separate-and-conquer, it is suggested to only use this algorithm when the datasets are large with complex decision boundaries; otherwise, single-layer boosting algorithms with simple weak learners are likely to be more suitable due to their ability to boost the confidence margins which improve the generalizability.

Given the choice of converting one of the AdaBoost.M2, AdaBoost.OC or AdaBoost.ECC single-layer classifiers into cascaded classifiers in the form of the multiclass-PSL method, there was statistical evidence to suggest that no one algorithm generalized better than the others. Though in terms of training and evaluation performance improvements, the results indicated that the AdaBoost.M2 algorithm benefited considerably more than the others. As with the binary PSL, several multiclass-PSL classifiers with different node sizes should be trained for each problem domain in order to determine which parameter yields the best accuracy, while partiality towards smaller nodes for performance reasons is recommended.

## 9.2 Future Work

There is ample scope now to further expand the algorithms developed in this research and to also explore their applicability to real-world problems. Given the increasing interest in coarse-to-fine learning approaches and the growing relevance of topics like adaptive and incremental learning, together with data fusion, the cascade algorithms presented here do hold potential for addressing these areas.

The intention will be to pursue further modifications to the PSL-like approaches to ensemble training which minimize the accuracy trade-off that currently accompanies improvements in the reduction of the training complexity. The goal will be to modify the separate-and-conquer strategy that is currently driven by training error criteria with that of margin-based confidences. The proposal will therefore be to replace the current tunable parameter $\Phi$ that governs the node sizes and the training error criteria and instead introduce a confidence decision threshold. The confidence threshold based on the margin theory would be determining factor for controlling the rate at which samples are removed from each layer or forwarded for re-training by succeeding nodes. In theory, this modification can be expected to increase the accuracy of all nodes provided that it is also combined with strategies outlined in this research for handling imbalanced learning on the final nodes of each layer.

The idea will be to also extend this further to adaptive learning in non-stationary environments, whereby the proposed method in this research can be significantly improved upon by devising methods that are capable of adapting the decision threshold at level of each node and not only at each layer. The expectation is that the ability of the algorithm to incorporate novel information about a target object will increase.

The research in the immediate future will also seek to apply the principle of margins to create alternative multiclass cascading algorithms for boosting methods. The envisaged concept is simpler than the one presented in this research in that it proposes to construct cascaded classifiers without nodes. Instead, each layer would be trained a predetermined number of rounds after which, instances belonging to each class label would be removed from training if the classifier attains confidence levels beyond specified thresholds. Each layer then would become defined by thresholds for every class label. Instances which are trivial to discriminate from other classes could be expected to be learned early on and thereby be removed from training so that the learning would focus more on difficult samples and achieve efficient runtimes. Given the lessons learned in this research, such strategies would need to be carefully monitored for signs of over-fitting and methods put

in place to counter that. In addition, the vision is to also make use of the class decision thresholds at each layer to experiment with implementing adaptive learning for the multiclass problem. The advantage would be that rapid adaptability might be attainable due only to the modification of decision confidences, rather than in retraining new instances requiring the update of an entire ensemble.

Efforts are already underway to apply the algorithms from this research to real-world industrial settings. Suitable problem domains are those which require frequent training of real-time executable classifiers that are promptly available on potentially large datasets. An example of one such domain is that of industrial grading and sorting of fresh produce, in conjunction with defect detection using computer vision software. In these problem sets, the operators encounter fresh produce varieties that undergo constant feature changes both seasonally, as well as regionally. Thus, the operating context demands continuous re-training of classifiers on large amounts of data with the requirement that the classifiers be made available immediately.

Finally, data fusion is an intriguing emerging area in recent research for which the algorithms discussed here provide a natural fit. The PSL-like algorithms provide the required decoupling of their constituent weak classifiers which enables the fusing of different but complementary data sources. In future, opportunities will be pursued to procure datasets from domains which are suited for data fusion in order to explore the applicability of the methods researched here to provide alternative solutions.

# Appendices

# Appendix A

# Preliminary Experimental Results

## A.1 Supplementary Results

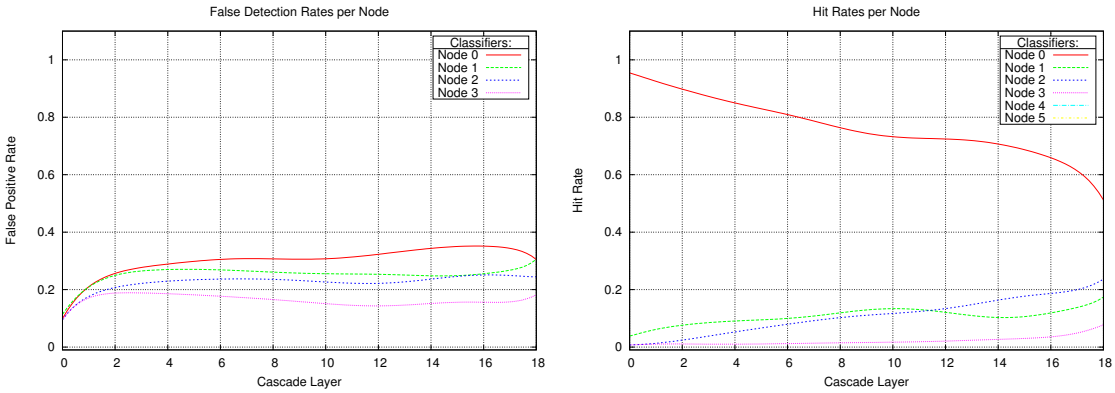



Figure A.1: The improved true positive/false positive ratio of a BPSL.r classifier $\Phi = 20$ and $\beta = 2000$ from a 15000 sample dataset. Proportional false positive rates per node for each layer and a node's hit rate contribution for each layer.

The diagrams in Figure A.1 show an improved ratio between the true positive and false positive rates of each node across all the layers of the cascade of a BPSL.r classifier. In contrast to the figures for the naive BPSL classifier, the trailing nodes of a BPSL.r classifier exhibit lower false positive rates, while the contribution of the trailing nodes towards an improvement in the true positive detections increases with each suceeding layer. In addition, the false positive rates of the BPSL.r nodes is much more stable across all layers compared to that of the BPSL nodes. This indicates that there is a lot less variability in the trailing nodes of BPSL.r with there being less susceptibility to outliers and thus an overall improved error rate.

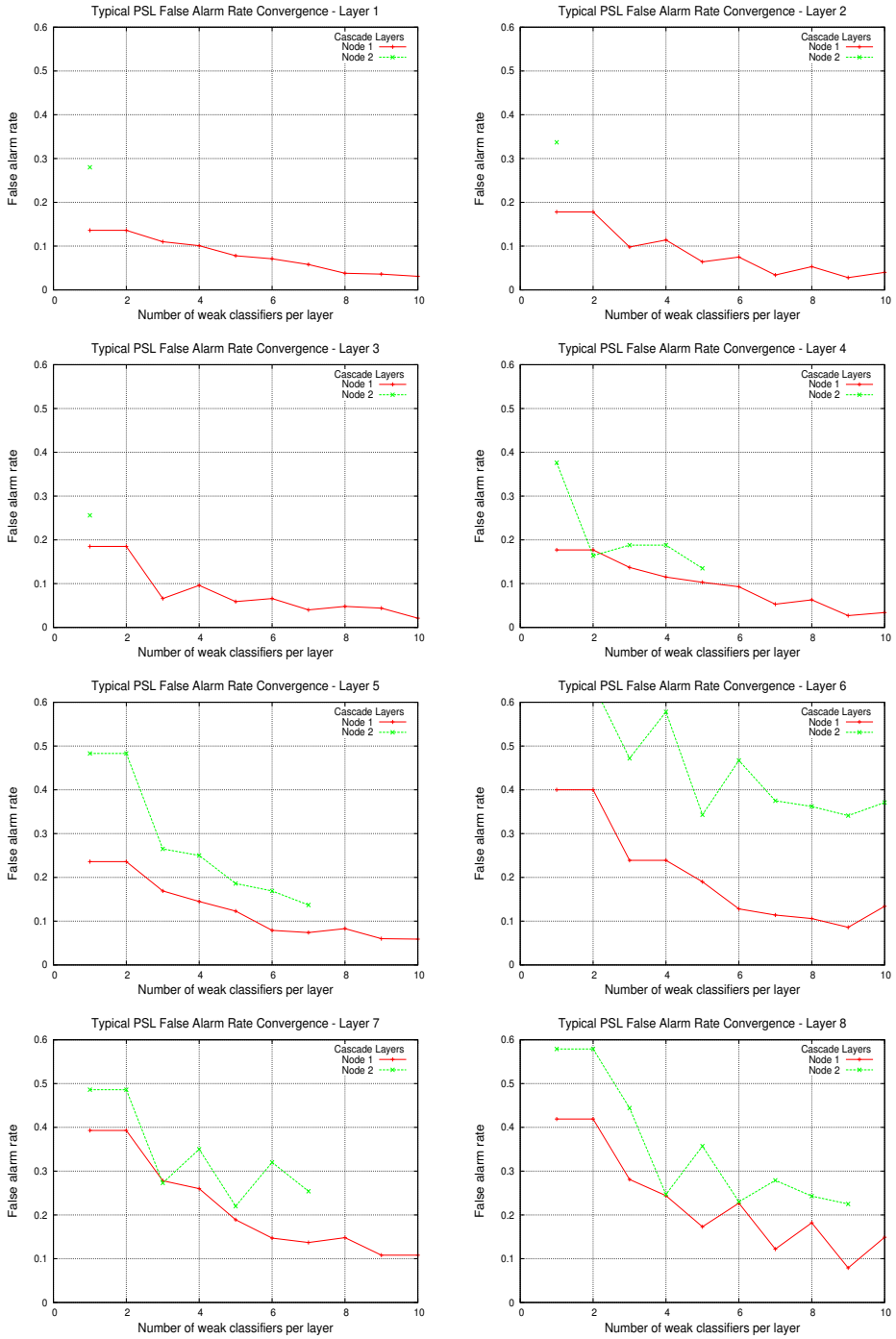



Figure A.2: Graphs showing typical convergence patterns of the false alarm rates for PSL classifiers in which the PSL classifier trained on a 1000 samples dataset with a node size of 10 is featured with layer targets of a 100% hit rate and 50% false alarm rate. Each graph represents the convergence of a single layer in which there are multiple plots, one for each node. The graphs demonstrate the an elevated false alarm rate at the end of layer and the increasing size of the false alarm rate as the number of nodes increase. Classifier layers from 1 - 8 are depicted.

# Appendix B

# Additional BPSL Graphs and Detection-Runtime Experiments

## B.1 ROC Graphs for BPSL Classifiers on 5000 Sample Dataset

Figure B.1 shows the ROC graphs comparing the BPSL classifiers with $\beta = 500$ and $\beta = 2000$ parameters over the range of different values of $\Phi$, against the PSL and VJ classifiers. The VJ classifiers have generalized on the CMU MIT datasets significantly better than the PSL-like classifiers across all four values of $\Phi$. However, the results indicate that there were no consistent trends that indicate that the bootstrapping component of the BPSL classifies has resulted in the deterioration in the accuracy.

## B.2 Execution Runtimes with Variable Values of $\Phi$

Experiments in this research have predominately involved PSL-like classifiers with fixed values of $\Phi$ throughout all the cascade layers. However, PSL-like classifiers can be easily tuned for more demanding detection runtime requirements by making the $\Phi$ parameter variable for different layers. Experiments here briefly demonstrate how the execution runtime of a PSL-like classifier can be significantly accelerated by training initial layers with smaller and more efficient nodes that increase in size for latter layers in order to produce more accurate weak classifers as the classification task becomes more difficult.

Three BPSL.r classifiers with a simple strategy for growing the $\Phi$ parameter were trained on datasets comprising 5000, 10000 and 15000 positive samples and using the methodology outlined in Chapter 5.3.1. The variable $\Phi$ was naively increased in three phases for different layers in a cascade, as shown in Table B.1.

Table B.1: The configuration of the $\Phi$ value per layer of the cascade for the BPSL.r classifier.

| Layers | 1-3 | 4-5 | $\geq 6$ |
|---|---|---|---|
| $\Phi$ | 1 | 5 | 10 |

Table B.2 lists the percentage by which the BPSL.r with flexible value of $\Phi$ executesd faster than a conventional BPSL.r classifier, where $\Phi$ was fixed with value 10. The results

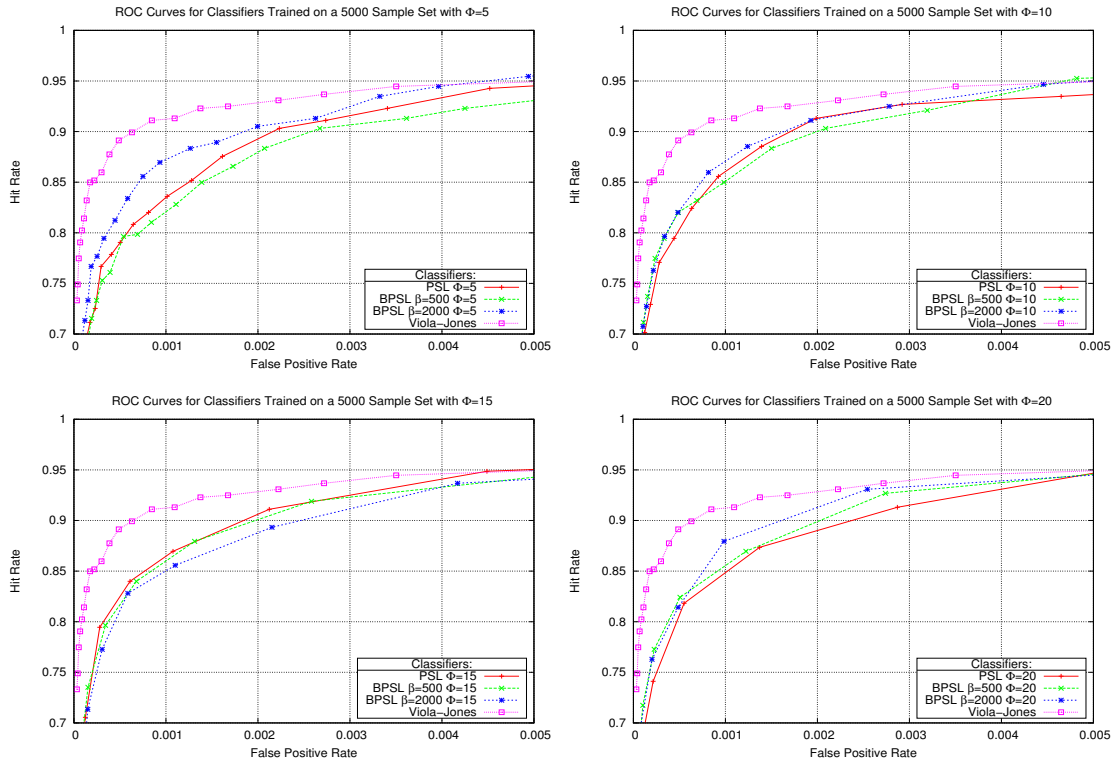



Figure B.1: Classifier ROC graph curves comparing PSL, BPSL and VJ classifiers on the CMU MIT dataset.

indicate that even with a simplistic strategy as this, the performance gains in varying the value of $\Phi$ for the initial layers of a cascade were significant. Arguably, given a more sophisticated strategy for varying the size of $\Phi$, even greater performance gains could be achieved. Likewise, the performance gains of varying the value $\Phi$ for initial layers can be expected to bring considerably larger performance improvements against other PSL-like classifiers which have $\Phi > 10$ fixed parameters.

Table B.2: The percentage by which the flexible BPSL.r outperforms the BPSL.r with fixed values of $\Phi$ for all layers of a cascade.

| | Flexible $\Phi$ BPSL.r vs. BPSL.r $\Phi = 10$ | | |
|---|---|---|---|
| Dataset size | 5000 | 10000 | 15000 |
| Faster execution | 36.0% ±0.1% | 33.6% ±0.1% | 26.6% ±0.1% |

The accuracies of BPSL.r classifiers with flexible node sizes are compared to their counterpart PSL classifiers from Chapter 5. The results indicate that in general, the flexible BPSL.r classifiers displayed some of the best accuracy results across the three datasets. There was no difference in accuracy between the flexible BPSL.r and the fixed BPSL.r on the 5000 sample dataset. On the 10000 and the 15000 datasets, the fixed

BPSL.r classifiers generalized slightly stronger than the flexible BPSL.r classifiers.

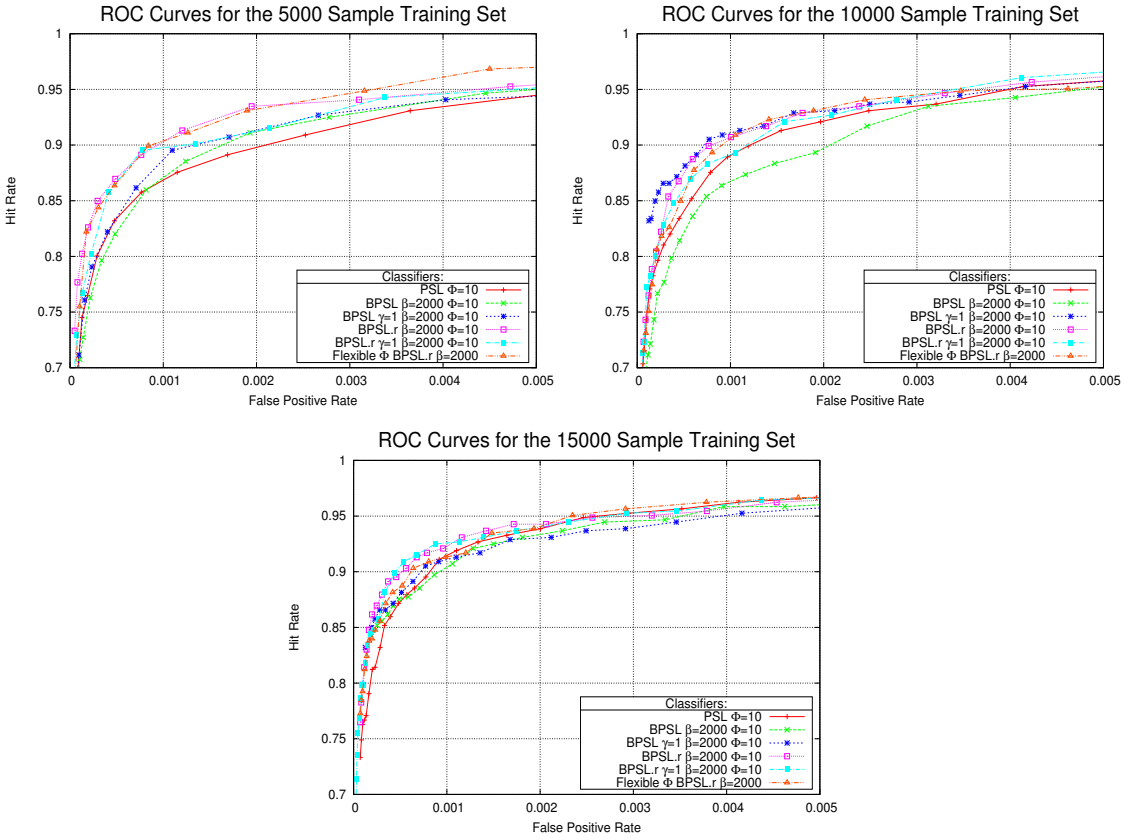



Figure B.2: ROC curves for BPSL.r classifiers with adaptive values of $\Phi$ for different layers and their comparison with the conventional BPSL.r, BPSL and PSL classifiers.

Overall, the results indicate that the detection runtime of the PSL-like classifers can be trivially tuned using simple strategies which train initial layers of a cascade with very small $\Phi$ values that are progressively increased. The strategy is effective since a smaller number of weak classifiers are evaluated in the early layer for classifying negative samples that clearly do not contain target patterns, while larger numbers of weak classifiers are executed only for samples which are closer to the decision boundary.

# Appendix C

# Supplementary Multiclass Algorithm Results

The complete collections of figures and tables pertaining to the analysis of the multiclass cascaded algorithm are displayed here.

**Training Phase** The training convergence of all classifiers across the 18 datasets is shown in Figure C.1. The primary observations were:

1. The exponential rate of decrease in the training error by the control algorithms is not matched by the cascaded.DP classifiers in the initial boosting rounds.

2. The fastest convergence rates are in most cases recorded by the ECC algorithm and followed by the OC and M2 algorithms.

3. On some datasets, the cascaded classifiers were able to catch-up to the control algorithms and attained zero training error rates at lower costs in boosting iterations than the OC and M2 algorithms.

4. The low training error was matched only by the ECC classifiers.

5. It was observed that was observed that on Vowel, Segmentation, Vehicle, Glass, Iris and Factors datasets the low training error rates of the cascaded classifiers are often achieved while generating significantly smaller ensembles than those of OC and M2 classifiers.

6. Cascaded classifiers were susceptible to generating more complex ensembles than all the control algorithms on Pendigit, Letter, Satimage, Optdigits, Yeast and Morphological datasets.

7. There was no consistent pattern amongst the cascaded classifiers which allowed for definitive statements to be made as to what value of $\Phi$ yields best convergences.

8. Non-monotonic patterns were generated by cascaded classifiers on Pendigit, Satimage, Vowel and Shuttle datasets.

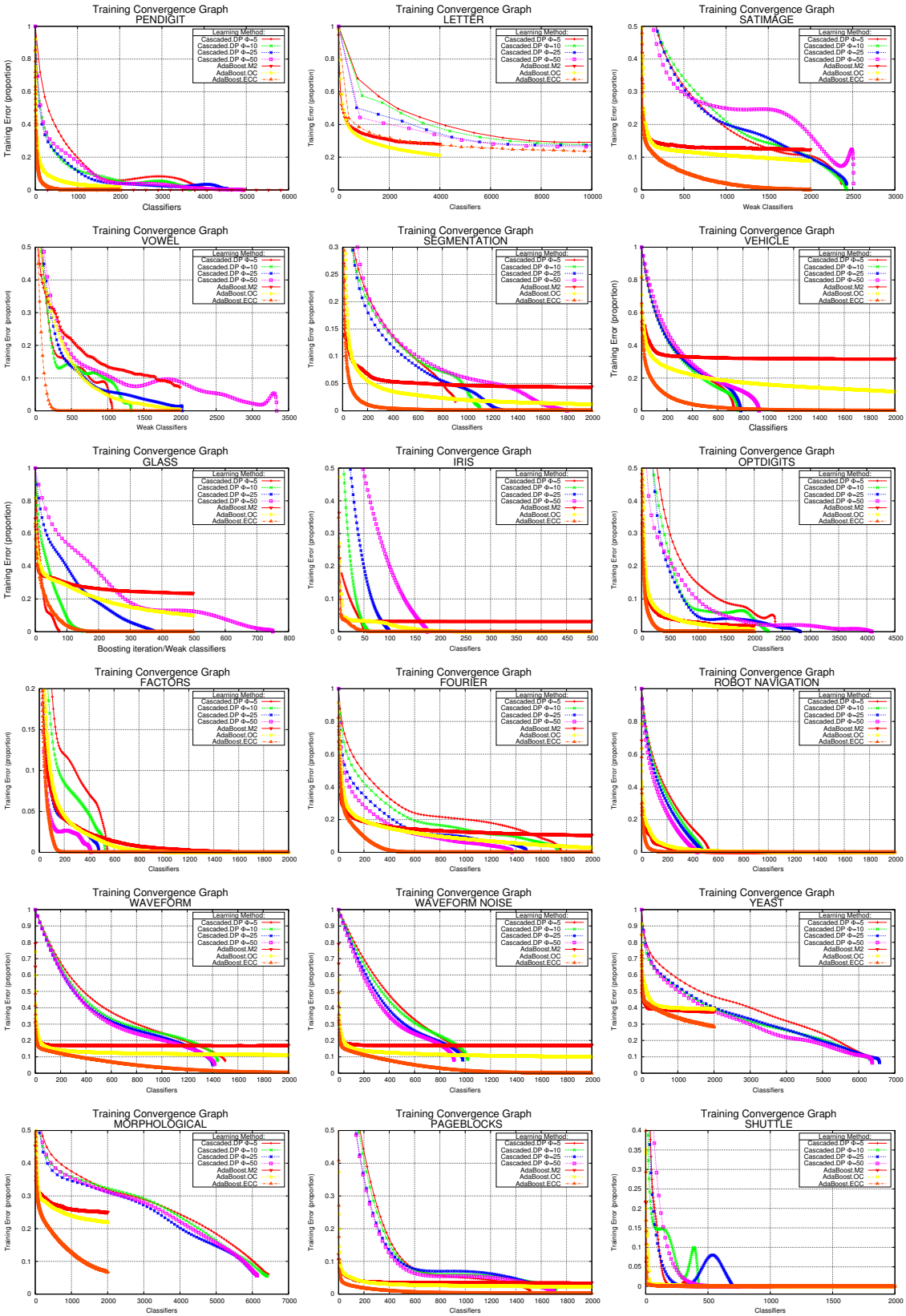



Figure C.1: Training error convergence graphs.

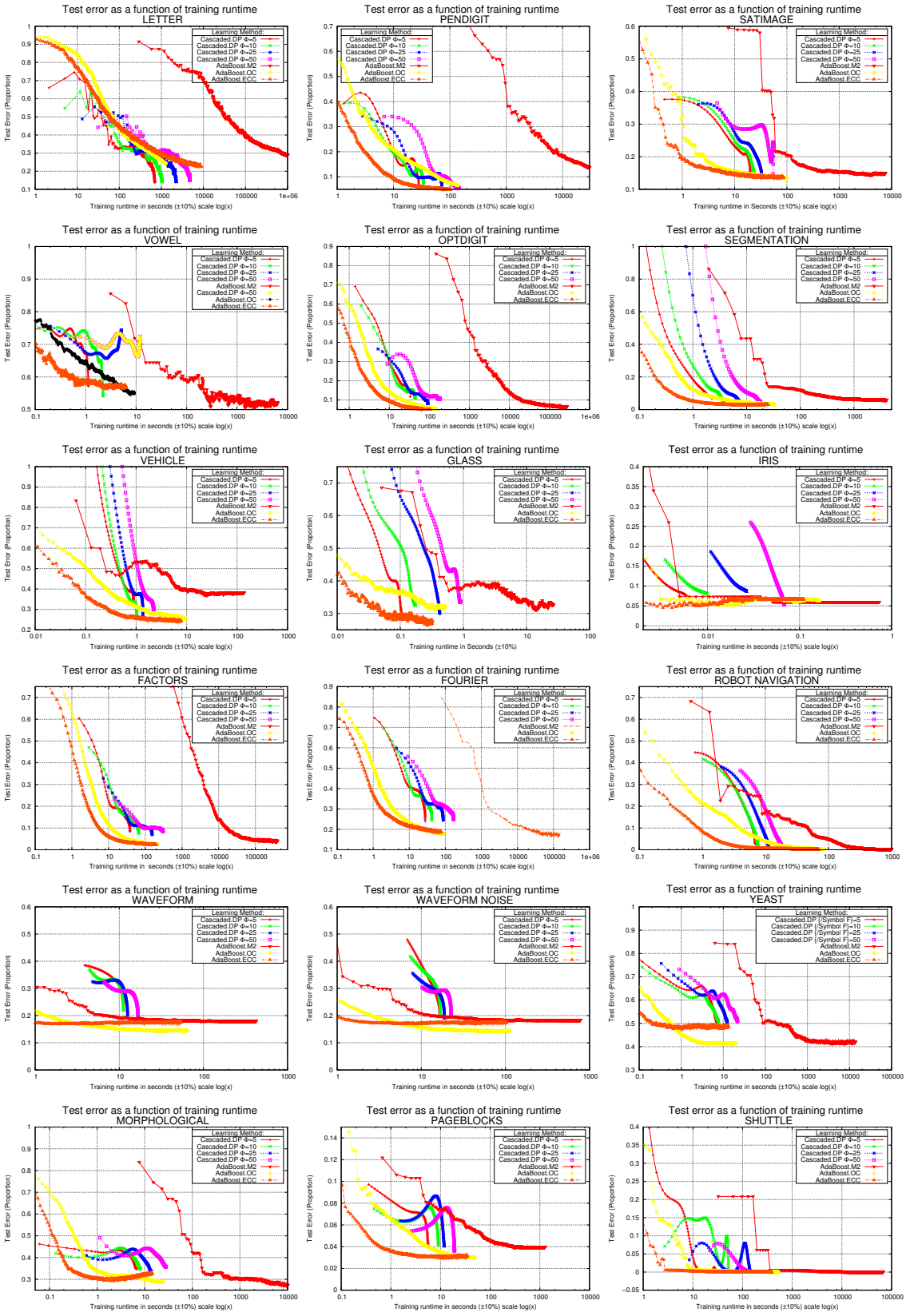



Figure C.2: Classifier accuracy on test data as a function of training runtime.

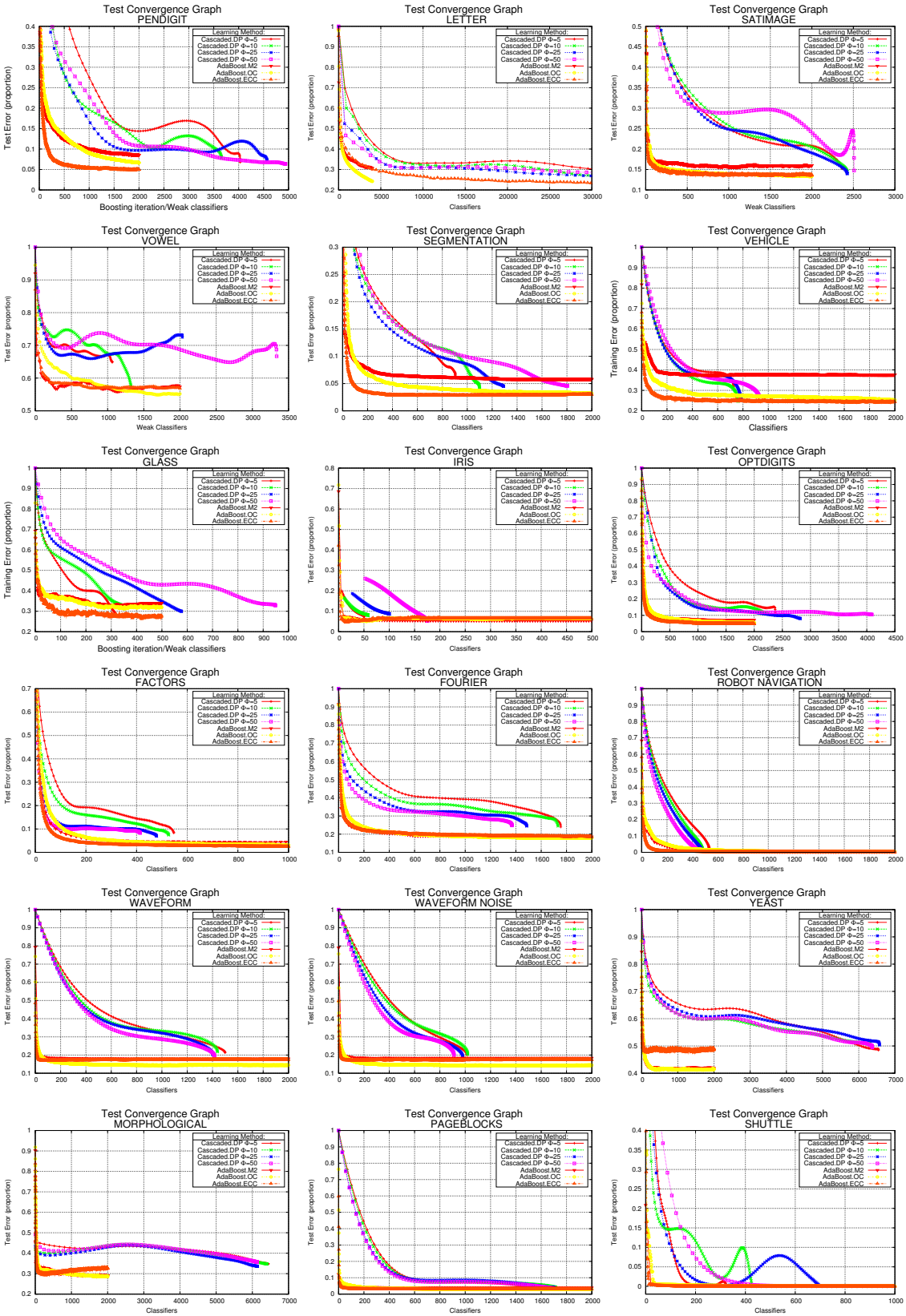



Figure C.3: Test error convergence graphs.

**Test Phase** Figure C.2 shows the cost/benefit relationship between the training runtimes and the test accuracy of all classifiers. The summary of the findings are:

1. The ECC algorithm consistently produced the best test error results.
2. Both ECC and OC algorithms dominated the cascaded classifiers in the initial stages, but the cascaded classifiers frequently demonstrated their ability to match the test error rates of ECC and OC in the latter stages as seen on Satimage, Vowel, Segmentation, Vehicle, Glass and Robot Navigation datasets.
3. Cascaded classifiers with smaller values of $\Phi$ dominated the performances of the classifiers with larger nodes.
4. M2 algorithm demonstrated higher training runtime costs than all the other classifiers for a given accuracy.

The test error results in Figure C.3 can be summarized as follows:

1. There was a sizable discrepancy between the final training error and the final error rates of the cascaded classifiers unlike that of the control algorithms.
2. The non-monotonic patterns of cascaded classifiers, particularly on the Pendigit, Satimage, and Morphological datasets did not indicate that over-training had occurred (exception being the Vowel dataset).
3. The cascaded classifiers frequently generated smaller ensembles than the control algorithms and have succeeded in most cases to match their accuracies. This was observed in $\frac{7}{18}$ of the cases.
4. The cascaded classifiers produced considerably larger ensembles than the control algorithms. In the majority of these cases, even with larger ensembles, the cascaded classifiers did not record accuracy rates that matched those of the control algorithms. This occurred in $\frac{6}{18}$ of the datasets.
5. In $\frac{4}{18}$ of the cases, the cascaded classifiers produced smaller ensembles than the control algorithms but were, however, unable to match the generalization of the control algorithms in general.

Table C.1 - Table C.3 Show the F-values attained by each classifier on every class label within their respective datasets.

Table C.1: F-value accuracy results for the shuttle dataset.

| Measure | Class | | Cascaded.DP | | | | OC | ECC | M2 |
|---|---|---|---|---|---|---|---|---|---|
| | Type | Distr. | 5 | 10 | 25 | 50 | | | |
| | 0 | 0.02% | **1.000** | **1.000** | 1.000 | 0.500 | **1.000** | **1.000** | **1.00** |
| | 1 | 78.60% | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.00 |
| | 2 | 0.09% | 0.846 | 0.923 | 0.846 | 0.923 | **0.960** | **0.960** | **0.96** |
| **F-Value** | 3 | 0.29% | **1.000** | **1.000** | 0.974 | 0.974 | **1.000** | **1.000** | 0.91 |
| | 4 | 15.35% | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.00 |
| | 5 | 5.63% | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.00 |
| | 6 | 0.02% | 0.75 | 0.75 | 0.25 | 0.5 | **1.000** | **1.000** | **1.00** |

Table C.2: F-value accuracy results for the robot navigation dataset.

| Measure | Class | | Cascaded.DP | | | | OC | ECC | M2 |
|---|---|---|---|---|---|---|---|---|---|
| | Type | Distr. | 5 | 10 | 25 | 50 | | | |
| | 0 | 15.14 | 0.970 | 0.983 | 0.978 | 0.978 | 0.997 | **0.999** | 0.998 |
| **F-Value** | 1 | 38.43 | 0.983 | 0.993 | 0.987 | 0.990 | 0.998 | 0.999 | 0.998 |
| | 2 | 40.41 | 0.977 | 0.989 | 0.992 | 0.988 | 0.997 | 0.998 | 0.998 |
| | 3 | 6.01 | 0.960 | 0.954 | 0.963 | 0.970 | 0.989 | **0.992** | **0.989** |

Table C.3: F-value accuracy results for the satimage dataset.

| Measure | Class | | Cascaded.DP | | | | OC | ECC | M2 |
|---|---|---|---|---|---|---|---|---|---|
| | Type | Distr. | 5 | 10 | 25 | 50 | | | |
| | 0 | 24.28% | 0.829 | 0.867 | 0.852 | 0.805 | 0.844 | 0.844 | 0.813 |
| | 1 | 22.3 | 0.971 | 0.975 | 0.968 | 0.971 | 0.974 | 0.978 | 0.957 |
| **F-Value** | 2 | 11.2% | 0.963 | 0.931 | 0.963 | 0.926 | 0.971 | 0.970 | 0.964 |
| | 3 | 21.3% | 0.867 | 0.859 | 0.881 | 0.878 | 0.902 | 0.875 | 0.896 |
| | 4 | 10.0% | **0.597** | **0.573** | **0.602** | **0.650** | 0.590 | 0.592 | 0.515 |
| | 5 | 10.7% | 0.797 | **0.838** | **0.827** | **0.858** | 0.840 | **0.860** | 0.798 |

# Appendix D

# Complete Graphs and Tables from the Results of Cascadizing OC, ECC and M2

**Training Phase:** Summary of main points from the training error graphs in Figure D.1:

1. Almost without exception across all the datasets in these experiments, the cascaded classifiers were not able to keep up with the control algorithms round-for-round.
2. In 66% of cases, the cascaded algorithms were able to catch up to the convergence rates of the control algorithms and tended to exceed their final training errors very rapidly.
3. On 33% datasets, the cascaded algorithms were not capable of producing classifiers which were able to catch up to the control classifiers at any point during their convergences. However, they often attained zero training error at some later point in contrast to the control algorithms.

**Test Phase:** Summary of main points from the graphs of classifier test accuracy as a function of training runtime for cascaded classifiers seen in Figure D.2:

1. The control algorithms dominated the cascaded algorithms in the initial phases.
2. On 66% of the datasets, the cascaded classifiers successfully caught up to their respective single-layer algorithms frequently coming close to parity.
3. In the remaining 33% of the instances, the single-layer classifiers displayed stronger generalization and lower costs throughout the entire convergence.

The convergences of the test error graphs in Figure D.3 strongly paralleled the primary trends seen in the training error convergences of Figure D.1.

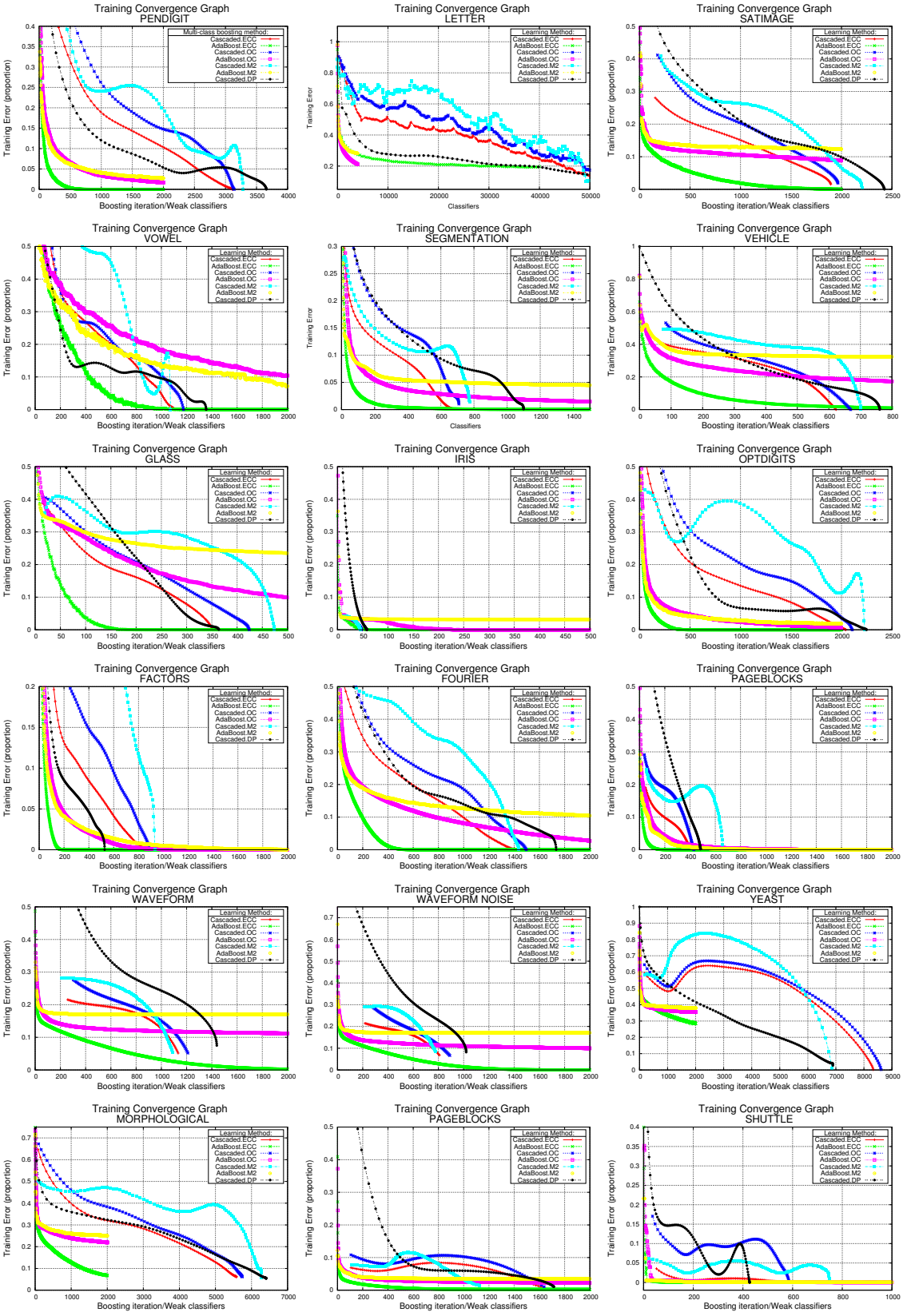



Figure D.1: Training error convergence graphs for cascaded classifiers.

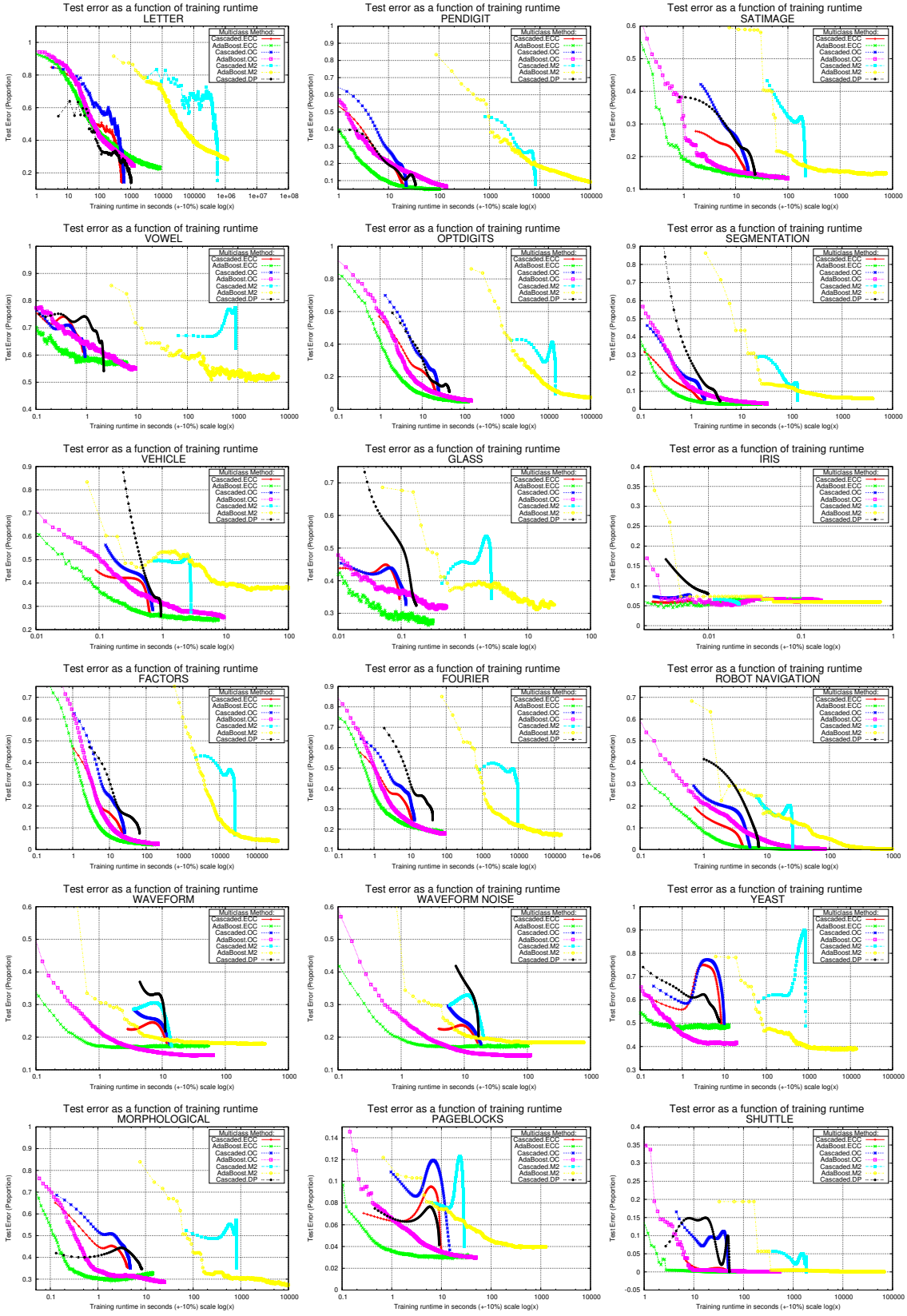



Figure D.2: Classifier accuracy on test data as a function of training runtime for cascaded classifiers.

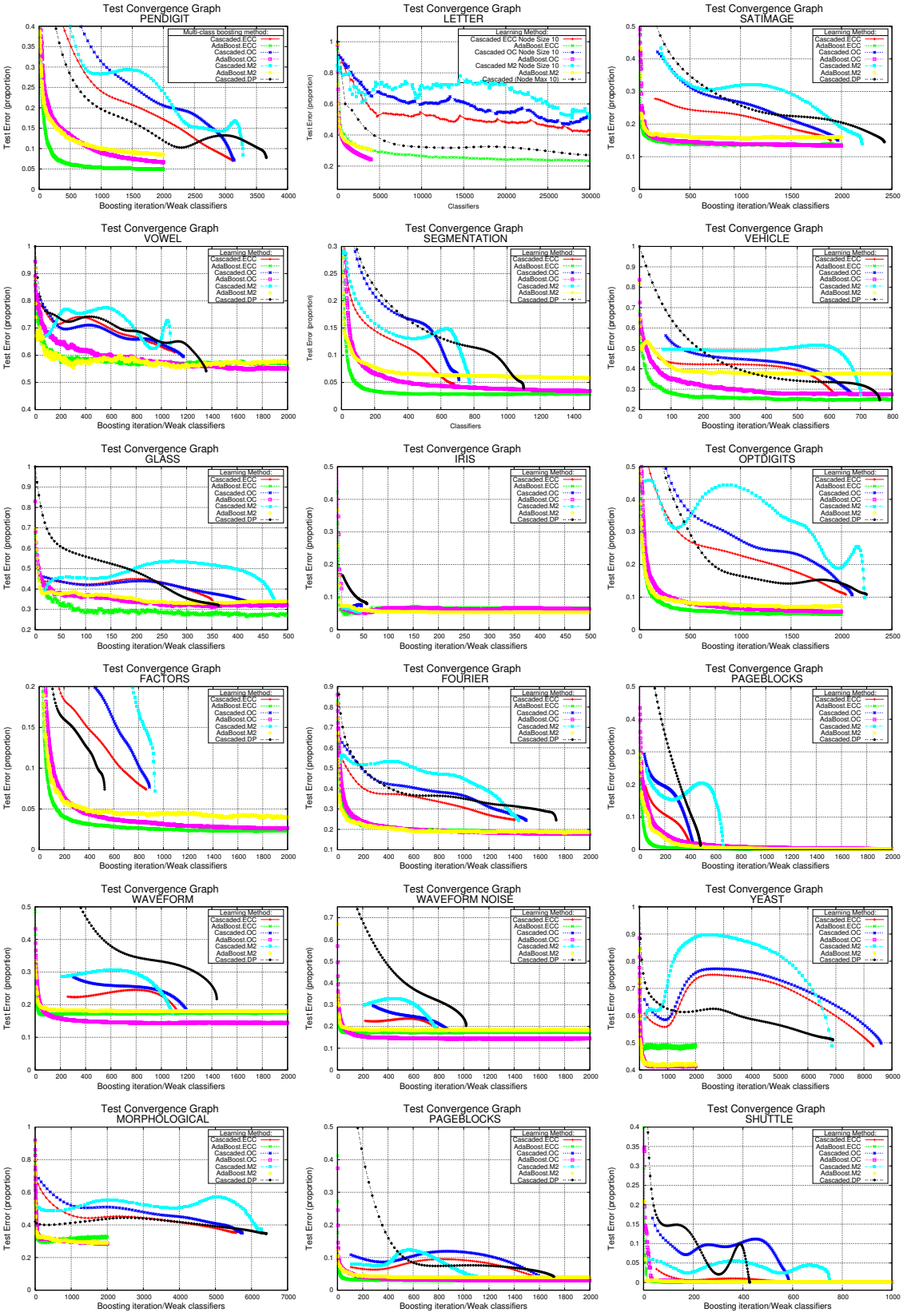



Figure D.3: Test error convergence graphs for cascaded classifiers.

Table D.1: F-Values for the Yeast dataset. Minority classes of interest are denoted as •and an observed accuracy improvement over the single-layer classifiers as ✓.

| Class | Distr. | Cascaded ECC | ECC | Cascaded OC | OC | Cascaded M2 | M2 | Cascaded DP |
|---|---|---|---|---|---|---|---|---|
| 0 • | 2.1 | **0.622** | 0.695 | **0.633** | 0.673 | **0.620** | 0.528 | 0.533 |
| 1 | 89.77 | **0.981** | 0.984 | **0.982** | 0.985 | 0.980 | 0.980 | 0.984 |
| 2 • | 6.01 | **0.866** | 0.879 | **0.866** | 0.884 | **0.851** | 0.825 | 0.826 |
| 3 • | 0.51 | **0.741** | 0.785 | **0.761** | 0.724 | **0.764** | 0.565 | 0.643 |
| 4 • | 1.61 | **0.735** | 0.835 | **0.792** | 0.883 | 0.671 | **0.869** | 0.751 |

Table D.2: F-Values for the Shuttle dataset. Minority classes of interest are denoted as •and an observed accuracy improvement over the single-layer classifiers as ✓.

| Class | Distr. | Cascaded ECC | ECC | Cascaded OC | OC | Cascaded M2 | M2 | Cascaded DP |
|---|---|---|---|---|---|---|---|---|
| 0 • | 0.02 | 1.000 | 1.000 | 0.974 | 1.00 | 1.000 | 1.000 | 1.000 |
| 1 | 78.60 | 1.000 | 1.000 | 1.000 | 1.00 | 1.000 | 1.000 | 1.000 |
| 2 • | 0.09 | 0.952 | 0.960 | 0.960 | **0.96** | **1.000** | 0.960 | 0.923 |
| 3 • | 0.29 | 0.994 | 1.000 | **0.986** | 1.00 | **1.000** | 0.909 | **1.000** |
| 4 | 15.35 | 1.000 | 1.000 | 1.000 | 1.00 | 1.000 | 1.000 | 1.000 |
| 5 • | 5.63 | 1.000 | 1.000 | 1.000 | 1.00 | 1.000 | 1.000 | 1.000 |
| 6 • | 0.02 | 0.961 | 1.000 | **0.97** | 1.00 | 1.000 | 1.000 | 0.750 |

Table D.3: F-Values for the Glass dataset. Minority classes of interest are denoted as •and an observed accuracy improvement over the single-layer classifiers as ✓.

| Class | Distr. | Cascaded ECC | ECC | Cascaded OC | OC | Cascaded M2 | M2 | Cascaded DP |
|---|---|---|---|---|---|---|---|---|
| 0 | 13.6 | 0.828 | 0.914 | 0.737 | 0.839 | 0.823 | 0.887 | 0.759 |
| 1 | 32.7 | 0.682 | 0.724 | 0.701 | 0.720 | 0.692 | 0.696 | 0.729 |
| 2 | 35.5 | 0.692 | 0.726 | 0.710 | 0.703 | 0.677 | 0.656 | 0.645 |
| 3 • | 7.9 | **0.289** | 0.314 | **0.313** | 0.105 | **0.252** | 0.087 | **0.412** |
| 4 • | 4.2 | **0.468** | 0.766 | **0.663** | 0.472 | 0.357 | 0.659 | **0.778** |
| 5 • | 6.1 | **0.546** | 0.688 | **0.650** | 0.635 | **0.617** | 0.434 | **0.769** |

Table D.4: F-Values for the Robot Navigation dataset. Minority classes of interest are denoted as •and an observed accuracy improvement over the single-layer classifiers as ✓.

| Class | Distr. | Cascaded ECC | ECC | Cascaded OC | OC | Cascaded M2 | M2 | Cascaded DP |
|---|---|---|---|---|---|---|---|---|
| 0 • | 15.14 | **0.975** | 0.999 | **0.983** | 0.997 | 0.977 | **0.998** | 0.983 |
| 1 | 38.43 | 0.992 | 0.999 | 0.993 | 0.998 | 0.995 | 0.998 | 0.993 |
| 2 | 40.41 | 0.986 | 0.998 | 0.990 | 0.997 | 0.988 | 0.998 | 0.989 |
| 3 • | 6.01 | **0.960** | 0.992 | **0.967** | 0.989 | 0.977 | **0.989** | 0.954 |

# Bibliography

[1] K.M. Ali and M.J. Pazzani. Error reduction through learning multiple descriptions. *Machine Learning*, 24(3):173–202, 1996.

[2] F.M. Alkoot and J. Kittler. Experimental evaluation of expert fusion strategies. *Pattern Recognition Letters*, 20(11-13):1361–1369, 1999.

[3] E.L. Allwein, R.E. Schapire, and Y. Singer. Reducing multiclass to binary: a unifying approach for margin classifiers. *J. Mach. Learn. Res.*, 1:113–141, September 2001. ISSN 1532-4435.

[4] S. Baluja, M. Sahami, and H.A. Rowley. Efficient face orientation discrimination. In *Proc. International Conference on Image Processing ICIP'04*, volume 1, pages 589–592 Vol. 1, 2004. doi: 10.1109/ICIP.2004.1418823.

[5] A. L. C. Barczak, M. J. Johnson, and C. H. Messom. Empirical evaluation of a new structure for adaboost. In *SAC '08: Proceedings of the 2008 ACM symposium on Applied computing*, pages 1764 – 1765, Fortaleza, Ceara, Brazil, 2008. ACM. ISBN 978-1-59593-753-7. doi: http://doi.acm.org/10.1145/1363686.1364109.

[6] P. Bartlett and J. Shawe-Taylor. Generalization performance of support vector machines and other pattern classifiers. *Advances in Kernel Methods, Support Vector Learning*, pages 43–54, 1999.

[7] G.E. Batista, R.C. Prati, and M.C. Monard. A study of the behavior of several methods for balancing machine learning training data. *ACM SIGKDD Explorations Newsletter*, 6(1):20–29, 2004.

[8] A. Bifet, G. Holmes, B. Pfahringer, R. Kirkby, and R. Gavaldà. New ensemble methods for evolving data streams. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '09, pages 139–148, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-495-9.

[9] L. Bottou and O. Bousquet. The tradeoffs of large scale learning. *Advances in neural information processing systems*, 20:161–168, 2008.

[10] L. Bourdev and J. Brandt. Robust object detection via soft cascade. *Computer Vision and Pattern Recognition, IEEE Computer Society Conference on*, 2:236–243, 2005. ISSN 1063-6919.

[11] K.W. Bowyer, N.V. Chawla, L.O. Hall, and W.P. Kegelmeyer. Smote: synthetic minority over-sampling technique. *Arxiv preprint arXiv:1106.1813*, 2011.

[12] L. Breiman. Bagging predictors. *Machine learning*, 24(2):123–140, 1996.

[13] L. Breiman. Pasting small votes for classification in large databases and on-line. *Machine Learning*, 36(1):85–103, 1999.

[14] L. Breiman, J. Friedman, C.J. Stone, and R.A. Olshen. Classification and regression trees. *Machine Learning*, 19:293–325, 1984.

[15] S. C. Brubaker, M. D. Mullin, and J. M. Rehg. Towards optimal training of cascaded detectors. *Proc. of Computer Vision. ECCV 2006*, 3951:325–337, 2006.

[16] S.C. Brubaker, J. Wu, J. Sun, M.D. Mullin, and J.M. Rehg. On the design of cascades of boosted ensembles for face detection. *Int. J. Comput. Vision*, 77(1-3): 65–86, 2008. ISSN 0920-5691.

[17] C.A. Brunk and M.J. Pazzani. An investigation of noise-tolerant relational concept learning algorithms. In *Proceedings of the 8th International Workshop on Machine Learning*, volume 961, pages 389–393. Citeseer, 1991.

[18] P. Bühlmann and T. Hothorn. Twin boosting: improved feature selection and prediction. *Statistics and Computing*, 20(2):119–138, 2010.

[19] J. Cendrowska. Prism: An algorithm for inducing modular rules. *International Journal of Man-Machine Studies*, 27(4):349–370, 1987.

[20] N.V. Chawla, A. Lazarevic, L.O. Hall, and K.W. Bowyer. Smoteboost: Improving prediction of the minority class in boosting. *Knowledge Discovery in Databases: PKDD 2003*, pages 107–119, 2003.

[21] N.V. Chawla, L.O. Hall, K.W. Bowyer, and W.P. Kegelmeyer. Learning ensembles from bites: A scalable and accurate approach. *Journal of Machine Learning Research*, 5:421–451, 2004.

[22] X. Chen and J.C. Jeong. Minimum reference set based feature selection for small sample classifications. In *Proceedings of the 24th international conference on Machine learning*, pages 153–160. ACM, 2007.

[23] X. Chen and A.L. Yuille. A time-efficient cascade for real-time object detection: With applications for the visually impaired. In *Computer Vision and Pattern Recognition-Workshops, 2005. CVPR Workshops. IEEE Computer Society Conference on*, pages 28–28. IEEE, 2005.

[24] J. Cohen et al. A coefficient of agreement for nominal scales. *Educational and psychological measurement*, 20(1):37–46, 1960.

[25] W.W. Cohen. Fast effective rule induction. In *In Proceedings of the Twelfth International Conference on Machine Learning*, pages 115–123, 1995.

[26] C. Cortes and V. Vapnik. Support-vector networks. *Machine Learning*, 20:273–297, 1995. ISSN 0885-6125.

[27] K. Crammer and Y. Singer. On the learnability and design of output codes for multiclass problems. *Machine Learning*, 47:201–233, 2002. ISSN 0885-6125.

[28] B.V. Dasarathy and B.V. Sheela. A composite classifier system design: Concepts and methodology. *Proceedings of the IEEE*, 67(5):708 – 713, may 1979. ISSN 0018-9219.

[29] J. Demšar. Statistical comparisons of classifiers over multiple data sets. *J. Mach. Learn. Res.*, 7:1–30, December 2006. ISSN 1532-4435.

[30] T.G. Dietterich and G. Bakiri. Solving multiclass learning problems via error-correcting output codes. *CoRR*, cs.AI/9501101, 1995.

[31] G. Ditzler, R. Polikar, and N. Chawla. An incremental learning algorithm for non-stationary environments and class imbalance. In *Pattern Recognition (ICPR), 2010 20th International Conference on*, pages 2997–3000. IEEE, 2010.

[32] L. Dong, E. Frank, and S. Kramer. Ensembles of balanced nested dichotomies for multi-class problems. In Alpio Jorge, Lus Torgo, Pavel Brazdil, Rui Camacho, and Joo Gama, editors, *Knowledge Discovery in Databases: PKDD 2005*, volume 3721 of *Lecture Notes in Computer Science*, pages 84–95. Springer Berlin / Heidelberg, 2005.

[33] R. Duangsoithong and T. Windeatt. Relevance and redundancy analysis for ensemble classifiers. *Machine Learning and Data Mining in Pattern Recognition*, pages 206–220, 2009.

[34] R.O. Duda, P.E. Hart, and D.G. Stork. *Pattern Classification*. John Wiley & Sons, 2000.

[35] G. Eibl and K.P. Pfeiffer. Multiclass boosting for weak classifiers. *J. Mach. Learn. Res.*, 6:189–210, December 2005. ISSN 1532-4435.

[36] R. Elwell and R. Polikar. Incremental learning of variable rate concept drift. *Multiple Classifier Systems*, pages 142–151, 2009.

[37] M. Fauvel, J. Chanussot, and J.A. Benediktsson. Decision fusion for the classification of urban remote sensing images. *Geoscience and Remote Sensing, IEEE Transactions on*, 44(10):2828 –2838, oct. 2006. ISSN 0196-2892.

[38] T.E. Fawcett. An introduction to roc analysis. *Pattern Recogn. Lett.*, 27(8):861–874, 2006. ISSN 0167-8655.

[39] T.E. Fawcett and F. Provost. Fraud detection. In *Handbook of data mining and knowledge discovery*, pages 726–731. Oxford University Press, Inc., 2002.

[40] J. Fox. *Applied regression analysis, linear models, and related methods*. Sage Publications, Inc, 1997.

[41] A. Frank and A. Asuncion. UCI machine learning repository, 2010. URL `http://archive.ics.uci.edu/ml`.

[42] E. Frank and S. Kramer. Ensembles of nested dichotomies for multi-class problems. In *Proceedings of the twenty-first international conference on Machine learning*, page 39. ACM, 2004.

[43] E. Frank and I.H. Witten. Generating accurate rule sets without global optimization. In *Proceedings of the Fifteenth International Conference on Machine Learning*, pages 144–151. Morgan Kaufmann Publishers Inc., 1998.

[44] Y. Freund and R.E. Schapire. Experiments with a new boosting algorithm. In *Proceedings of the 13th International Conference in Machine Learning*, pages 148–156, 1996.

[45] Y. Freund and R.E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1): 119–139, 1997. ISSN 0022-0000.

[46] J. Friedman, T. Hastie, and R. Tibshirani. Additive logistic regression: a statistical view of boosting. *Annals of Statistics*, 28:2000, 2000.

[47] M. Friedman. The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *Journal of the American Statistical Association*, 32(200): 675–701, 1937.

[48] M. Friedman. A comparison of alternative tests of significance for the problem of m rankings. *The Annals of Mathematical Statistics*, 11(1):86–92, 1940.

[49] G. Fumera and F. Roli. A theoretical and experimental analysis of linear combiners for multiple classifier systems. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 27(6):942–956, 2005.

[50] J. Fürnkranz. More efficient windowing. In *Proceedings of the fourteenth national conference on artificial intelligence and ninth conference on Innovative applications of artificial intelligence*, pages 509–514. AAAI Press, 1997.

[51] J. Fürnkranz and G. Widmer. Incremental reduced error pruning. In *International Conference on Machine Learning*, pages 70–77, 1994.

[52] C. Garcia and M. Delakis. Convolutional face finder: A neural architecture for fast and robust face detection. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 26(11):1408–1423, November 2004.

[53] A.S. Georghiades, P.N. Belhumeur, and D.J. Kriegman. From few to many: Illumination cone models for face recognition under variable lighting and pose. *IEEE Trans. Pattern Anal. Mach. Intelligence*, 23(6):643–660, 2001.

[54] G. Giacinto and F. Roli. Ensembles of neural networks for soft classification of remote sensing images. In *European symposium on intelligent techniques*, pages 166–170, Bari, Italy, 1997.

[55] G. Giacinto and F. Roli. Design of effective neural network ensembles for image classification purposes. *Image and Vision Computing*, 19(9-10):699–707, 2001.

[56] P.O. Gislason, J.A. Benediktsson, and J.R. Sveinsson. Random forests for land cover classification. *Pattern Recognition Letters*, 27(4):294 – 300, 2006. ISSN 0167-8655. Pattern Recognition in Remote Sensing (PRRS 2004).

[57] H. Guo and H.L. Viktor. Learning from imbalanced data sets with boosting and data generation: the databoost-im approach. *ACM SIGKDD Explorations Newsletter*, 6 (1):30–39, 2004.

[58] V. Guruswami and A. Sahai. Multiclass learning, boosting, and error-correcting codes. In *Proceedings of the twelfth annual conference on Computational learning theory*, COLT '99, pages 145–155, New York, NY, USA, 1999. ACM. ISBN 1-58113-167-4.

[59] M.A. Hall and L.A. Smith. Practical feature subset selection for machine learning. *Computer Science*, 98:181–191, 1998.

[60] H. Han, W.Y. Wang, and B.H. Mao. Borderline-smote: A new over-sampling method in imbalanced data sets learning. *Advances in Intelligent Computing*, pages 878–887, 2005.

[61] J. Han, M. Kamber, and J. Pei. *Data mining: concepts and techniques*. Morgan Kaufmann, 2011.

[62] H. He and E.A. Garcia. Learning from imbalanced data. *IEEE Transactions on Knowledge and Data Engineering*, 21:1263–1284, 2009. ISSN 1041-4347.

[63] H. He, Y. Bai, E.A. Garcia, and S. Li. Adasyn: Adaptive synthetic sampling approach for imbalanced learning. In *Neural Networks, 2008. IJCNN 2008.(IEEE World Congress on Computational Intelligence). IEEE International Joint Conference on*, pages 1322–1328. IEEE, 2008.

[64] T.K. Ho. The random subspace method for constructing decision forests. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 20(8):832–844, 1998.

[65] C.-W. Hsu and C.-J. Lin. A comparison of methods for multiclass support vector machines. *Neural Networks, IEEE Transactions on*, 13(2):415 –425, mar 2002. ISSN 1045-9227. doi: 10.1109/72.991427.

[66] C. Huang, B. Wu, H. AI, and S. Lao. Omni-directional face detection based on real adaboost. In *Image Processing, 2004. ICIP'04. 2004 International Conference on*, volume 1, pages 593–596. IEEE, 2004.

[67] C. Huang, H. Ai, Y. Li, and S. Lao. High-performance rotation invariant multiview face detection. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 29(4):671–686, 2007.

[68] R.L. Iman and J.M. Davenport. Approximations of the critical region of the fbietkan statistic. *Communications in Statistics-Theory and Methods*, 9(6):571–595, 1980.

[69] M.M. Islam, X. Yao, and K. Murase. A constructive algorithm for training cooperative neural network ensembles. *Neural Networks, IEEE Transactions on*, 14(4): 820–834, 2003.

[70] R.A. Jacobs, M.I. Jordan, S.J. Nowlan, and G.E. Hinton. Adaptive mixtures of local experts. *Neural computation*, 3(1):79–87, 1991.

[71] N. Japkowicz and S. Stephen. The class imbalance problem: A systematic study. *Intell. Data Anal.*, 6:429–449, October 2002. ISSN 1088-467X.

[72] R. Jin and J. Zhang. Multi-class learning by smoothed boosting. *Machine Learning*, 67:207–227, 2007. ISSN 0885-6125.

[73] M. Karnick, M.D. Muhlbaier, and R. Polikar. Incremental learning in non-stationary environments with concept drift using a multiple classifier based approach. In *Pattern Recognition, 2008. ICPR 2008. 19th International Conference on*, pages 1 –4, 2008.

[74] J. Kittler, M. Hatef, R.P.W. Duin, and J. Matas. On combining classifiers. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 20(3):226–239, 1998.

[75] J.Z. Kolter and M.A. Maloof. Dynamic weighted majority: a new ensemble method for tracking concept drift. In *Data Mining, 2003. ICDM 2003. Third IEEE International Conference on*, pages 123 – 130, 2003.

[76] J.Z. Kolter and M.A. Maloof. Dynamic weighted majority: An ensemble method for drifting concepts. *The Journal of Machine Learning Research*, 8:2755–2790, 2007.

[77] U.H.G. Kreßel. *Pairwise classification and support vector machines*, pages 255–268. MIT Press, Cambridge, MA, USA, 1999.

[78] L.I. Kuncheva. Clustering-and-selection model for classifier combination. In *Knowledge-Based Intelligent Engineering Systems and Allied Technologies, 2000. Proceedings. Fourth International Conference on*, volume 1, pages 185–188. IEEE, 2000.

[79] L.I. Kuncheva. Switching between selection and fusion in combining classifiers: An experiment. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 32(2):146–156, 2002.

[80] L.I. Kuncheva. Classifier ensembles for changing environments. In Fabio Roli, Josef Kittler, and Terry Windeatt, editors, *Multiple Classifier Systems*, volume 3077 of *Lecture Notes in Computer Science*, pages 1–15. Springer Berlin / Heidelberg, 2004.

[81] L.I. Kuncheva. *Combining pattern classifiers: methods and algorithms*. Wiley-Interscience, 2004.

[82] J.D. Lafferty, C.K.I. Williams, J. Shawe-Taylor, R.S. Zemel, and A. Culotta, editors. *Advances in Neural Information Processing Systems 23: 24th Annual Conference on Neural Information Processing Systems 2010. Proceedings of a meeting held 6-9 December 2010, Vancouver, British Columbia, Canada*, 2010. Curran Associates, Inc.

[83] A. Lazarevic and Z. Obradovic. Effective pruning of neural network classifier ensembles. In *Neural Networks, 2001. Proceedings. IJCNN'01. International Joint Conference on*, volume 2, pages 796–801. IEEE, 2001.

[84] W. Leigh, R. Purvis, and J.M. Ragusa. Forecasting the nyse composite index with technical analysis, pattern recognizer, neural network, and genetic algorithm: a case study in romantic decision support. *Decision Support Systems*, 32(4):361–377, 2002.

[85] C. Leistner, A. Saffari, P.M. Roth, and H. Bischof. On robustness of on-line boosting - a competitive study. In *Computer Vision Workshops (ICCV Workshops), 2009 IEEE 12th International Conference on*, pages 1362 –1369, 27 2009-oct. 4 2009.

[86] L. Li. Multiclass boosting with repartitioning. In *Proceedings of the 23rd international conference on Machine learning*, ICML '06, pages 569–576, New York, NY, USA, 2006. ACM. ISBN 1-59593-383-2.

[87] P. Li, K. Chan, S. Fu, and S. Krishnan. An abnormal ecg beat detection approach for long-term monitoring of heart patients based on hybrid kernel machine ensemble. *Multiple Classifier Systems*, pages 839–841, 2005.

[88] S.Z. Li, Z.Q. Zhang, H. Shum, and H.J. Zhang. Floatboost learning for classification. In *Proceedings of The 16-th Annual Conference on Neural Information Processing Systems (NIPS)*, pages 9–14, Vancouver, Canada, December 2002.

[89] R. Lienhart and J. Maydt. An extended set of haar-like features for rapid object detection. In *ICIP02*, pages I: 900–903, Rochester, NY, September 2002.

[90] R. Lienhart, L. Liang, and A. Kuranov. A detector tree of boosted classifiers for real-time object detection and tracking. In *ICME2003*, pages 277–280. IEEE, 2003.

[91] H. Liu, A. Mandvikar, and J. Mody. An empirical study of building compact ensembles. *Advances in Web-Age Information Management*, pages 622–627, 2004.

[92] H. Liu, E.R. Dougherty, J.G. Dy, K. Torkkola, E. Tuv, H. Peng, C. Ding, F. Long, M. Berens, and L. Parsons. Evolving feature selection. *Intelligent systems, IEEE*, 20(6):64–76, 2005.

[93] X.Y. Liu, J. Wu, and Z.H. Zhou. Exploratory undersampling for class-imbalance learning. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 39(2):539–550, 2009.

[94] A.C. Lorena, A.C. Carvalho, and J.M. Gama. A review on the combination of binary classifiers in multiclass problems. *Artif. Intell. Rev.*, 30:19–37, December 2008. ISSN 0269-2821.

[95] H. Luo. Optimization design of cascaded classifiers. *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, 1:480–485 vol. 1, June 2005. ISSN 1063-6919. doi: 10.1109/CVPR.2005.266.

[96] O. Maimon and L. Rokach. Ensemble of decision trees for mining manufacturing data sets. *Machine Engineering*, 4(1-2), 2004.

[97] O. Maimon and L. Rokach. Decomposition methodology for knowledge discovery and data mining. *Data Mining and Knowledge Discovery Handbook*, pages 981–1003, 2005.

[98] D.D. Margineantu and T.G. Dietterich. Pruning adaptive boosting. In *Machine learning: proceedings of the fourteenth International Conference (ICML'97), Nashville, Tennessee, July 8-12, 1997*, page 211. Morgan Kaufmann Pub, 1997.

[99] M. May and L. Saitta. Introduction: The challenge of ubiquitous knowledge discovery. In Michael May and Lorenza Saitta, editors, *Ubiquitous Knowledge Discovery*, volume 6202 of *Lecture Notes in Computer Science*, pages 3–18. Springer Berlin / Heidelberg, 2010.

[100] M. May, B. Berendt, A. Cornuejols, J. Gama, F. Giannotti, A. Hotho, D. Malerba, E. Menasalvas, K. Morik, R. Pedersen, et al. *Research challenges in ubiquitous knowledge discovery*. Chapman & Hall/CRC Press, 2008.

[101] B. McCane and K. Novins. On training cascade face detectors. In *Image and Vision Computing New Zealand*, pages 239–244, Palmerston North, 2003.

[102] D. Mease and A. Wyner. Evidence contrary to the statistical view of boosting. *Journal of Machine Learning Research*, 9:131–156, 2008.

[103] R. Michalski and G. Tecuci. Machine learning: A multistrategy approach. *Kaufmann: San Francisco, CA*, 4, 1994.

[104] T. Mita, T. Kaneko, and O. Hori. Joint haar-like features for face detection. In *10th IEEE International Conference in Computer Vision (ICCV'05)*, pages 1619–1626. IEEE, 2005.

[105] T.M. Mitchell. *Machine learning*. MacGraw Hill, 1997.

[106] D. Morrison, R. Wang, and L.C. De Silva. Ensemble methods for spoken emotion recognition in call-centres. *Speech communication*, 49(2):98–112, 2007.

[107] M.D. Muhlbaier and R. Polikar. Multiple classifiers based incremental learning algorithm for learning in nonstationary environments. In *Machine Learning and Cybernetics, 2007 International Conference on*, volume 6, pages 3618 –3623, 2007.

[108] P. Nemenyi. *Distribution-free multiple comparisons*. PhD thesis, Princeton University, 1963.

[109] K. Nishida, K. Yamauchi, and T. Omori. Ace: Adaptive classifiers-ensemble system for concept-drifting environments. *Multiple Classifier Systems*, 3541:509–509, 2005.

[110] D.W. Opitz and J.W. Shavlik. Generating accurate and diverse members of a neural-network ensemble. *Advances in neural information processing systems*, pages 535–541, 1996.

[111] N.C. Oza. *Online ensemble learning*. PhD thesis, University of California, Berkeley, 2001.

[112] N.C. Oza and K. Tumer. Classifier ensembles: Select real-world applications. *Information Fusion*, 9(1):4 – 20, 2008. ISSN 1566-2535. Special Issue on Applications of Ensemble Methods.

[113] C. Papageorgiou, M. Oren, and T. Poggio. A general framework for object detection. In *International Conference on Computer Vision*, 1998.

[114] A. Passerini, M. Pontil, and P. Frasconi. New results on error correcting output codes of kernel machines. *Neural Networks, IEEE Transactions on*, 15(1):45 –54, jan. 2004. ISSN 1045-9227. doi: 10.1109/TNN.2003.820841.

[115] S.O. Petrov. *Coarse-to-Fine Natural Language Processing*. PhD thesis, UNIVERSITY OF CALIFORNIA, 2009.

[116] M.-T. Pham and T.-J. Cham. Fast training and selection of haar features using statistics in boosting-based face detection. *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*, pages 1–7, Oct. 2007. ISSN 1550-5499. doi: 10.1109/ICCV.2007.4409038.

[117] J.C. Platt, N. Cristianini, and J. Shawe-Taylor. Large margin dags for multiclass classification. *Advances in neural information processing systems*, 12(3):547–553, 2000.

[118] A. Pocock, P. Yiapanis, J. Singer, M. Luján, and G. Brown. Online non-stationary boosting. *Multiple Classifier Systems*, pages 205–214, 2010.

[119] T. Poggio and K.K. Sung. Finding human faces with a gaussian mixture distribution-based face model. *Recent Developments in Computer Vision*, pages 435–446, 1996.

[120] R. Polikar. Essemble based systems in decision making. *IEEE Circuits and Systems Magazine*, 6(3):21–45, 2006.

[121] E. Pranckeviciene, R. Baumgartner, and R. Somorjai. Using domain knowledge in the random subspace method: application to the classification of biomedical spectra. *Multiple Classifier Systems*, pages 962–971, 2005.

[122] M.J. Procopio. *An experimental analysis of classifier ensembles for learning drifting concepts over time in autonomous outdoor robot navigation*. PhD thesis, University of Colorado, Boulder, CO, USA, 2007. AAI3284409.

[123] M.J. Procopio, J. Mulligan, and G. Grudic. Learning terrain segmentation with classifier ensembles for autonomous robot navigation in unstructured environments. *Journal of Field Robotics*, 26(2):145–175, 2009.

[124] F. Provost and T. Fawcett. Analysis and visualization of classifier performance: Comparison under imprecise class and cost distributions. In *Proceedings of the third international conference on knowledge discovery and data mining*, pages 43–48. AAAI Press, 1997.

[125] J. Pujara, H. Daumé III, and L. Getoor. Using classifier cascades for scalable e-mail classification. In *The 8th Annual Collaboration, Electronic messaging, Anti-Abuse and Spam Conference, CEAS'11*, Perth, Australia, 2011. ACM.

[126] J.R. Quinlan. Induction of decision trees. *Machine learning*, 1(1):81–106, 1986.

[127] J.R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.

[128] L.A. Rastrigin and R.H. Erenstein. Method of collective recognition. *Energoizdat, Moscow*, 1981.

[129] V.C. Raykar, B. Krishnapuram, and S. Yu. Designing efficient cascaded classifiers: tradeoff between accuracy and cost. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 853–860. ACM, 2010.

[130] J. Rodriguez and L.I. Kuncheva. Combining online classification approaches for changing environments. *Structural, Syntactic, and Statistical Pattern Recognition*, pages 520–529, 2010.

[131] J.J. Rodríguez, C. García-Osorio, and J. Maudes. Forests of nested dichotomies. *Pattern Recogn. Lett.*, 31:125–132, January 2010. ISSN 0167-8655.

[132] L. Rokach. Decomposition methodology for classification tasks: a meta decomposer framework. *Pattern Analysis and Applications*, 9(2):257–271, 2006.

[133] L. Rokach. Ensemble-based classifiers. *Artificial Intelligence Review*, 33(1):1–39, 2010.

[134] L. Rokach. *Pattern classification using ensemble methods*, volume 75. World Scientific Pub Co Inc, 2010.

[135] H.A. Rowley. *Neural network-based face detection*. PhD thesis, Carnegie Mellon University, May 1999.

[136] H.A. Rowley, S. Baluja, and T. Kanade. Rotation invariant neural network-based face detection. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 38–44, June 1998.

[137] M.J. Saberian and N. Vasconcelos. Boosting classifier cascades. In *Neural Information Processing Systems (NIPS)*. NIPS, 2010.

[138] C. Sansone, J. Kittler, and F. Roli, editors. *Multiple Classifier Systems - 10th International Workshop, MCS 2011, Naples, Italy, June 15-17, 2011. Proceedings*, volume 6713 of *Lecture Notes in Computer Science*, 2011. Springer. ISBN 978-3-642-21556-8.

[139] R.E. Schapire. The strength of weak learnability. *Machine learning*, 5(2):197–227, 1990.

[140] R.E. Schapire. Using output codes to boost multiclass learning problems. In *Proceedings of the Fourteenth International Conference on Machine Learning*, ICML '97, pages 313–321, San Francisco, CA, USA, 1997. Morgan Kaufmann Publishers Inc.

[141] R.E. Schapire. The boosting approach to machine learning: An overview. In *Workshop on Nonlinear Estimation and Classification*. MSRI, 2002.

[142] R.E. Schapire and Y. Singer. Improved boosting algorithms using confidence-rated predictions. *Machine Learning*, 37:297–336, 1999. ISSN 0885-6125.

[143] R.E. Schapire, Y. Freund, P. Bartlett, and W.S. Lee. Boosting the margin: A new explanation for the effectiveness of voting methods. *The annals of statistics*, 26(5): 1651–1686, 1998.

[144] J.C. Schlimmer and R. Granger. Beyond incremental processing: Tracking concept drift. In *Proceedings of the Fifth National Conference on Artificial Intelligence*, volume 1, pages 502–507, 1986.

[145] J.C. Schlimmer and R.H. Granger. Incremental learning from noisy data. *Machine learning*, 1(3):317–354, 1986. ISSN 0885-6125.

[146] H. Schneiderman and T. Kanade. A statistical method for 3d object detection applied to faces and cars. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2000)*, pages 1746–1759, 2000.

[147] M. Scholz and R. Klinkenberg. Boosting classifiers for drifting concepts. *Intelligent Data Analysis*, 11(1):3–28, 2007.

[148] M. Sewell. Ensemble learning. Technical Report 02, University College of London, 2011.

[149] W.N. Street and Y.S. Kim. A streaming ensemble algorithm (sea) for large-scale classification. In *Proceedings of the seventh ACM SIGKDD international conference on knowledge discovery and data mining*, KDD '01, pages 377–382, New York, NY, USA, 2001. ACM.

[150] S.C. Suh. *Practical Applications of Data Mining*. Jones & Bartlett Publishers, 2011.

[151] J. Sun, J.M. Rehg, and A. Bobick. Automatic cascade training with perturbation bias. *Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on*, 2:II–276–II–283 Vol.2, June-2 July 2004. ISSN 1063-6919. doi: 10.1109/CVPR.2004.1315174.

[152] Y. Sun, M.S. Kamel, and Y. Wang. Boosting for learning multiple classes with imbalanced class distribution. In *Data Mining, 2006. ICDM '06. Sixth International Conference on*, pages 592 –602, December 2006.

[153] Y. Sun, M.S. Kamel, A.K.C. Wong, and Y. Wang. Cost-sensitive boosting for classification of imbalanced data. *Pattern Recognition*, 40(12):3358–3378, 2007.

[154] Y. Sun, S. Todorovic, and J. Li. Unifying multi-class adaboost algorithms with binary base learners under the margin framework. *Pattern Recognition Letters*, 28 (5):631–643, 2007.

[155] K. Sung and T. Poggio. Example-based learning for view-based face detection. *IEEE Patt. Anal. Mach. Intell.*, 20(39-51), 1998.

[156] T. Susnjak. Accelerating classifier training using adaboost within cascades of boosted ensembles. Master's thesis, Massey University, Auckland, New Zealand, 2009.

[157] T. Susnjak and A. Barczak. Accelerated classifier training using the psl cascading structure. In Mario Koeppen, Nikola Kasabov, and George Coghill, editors, *Advances in Neuro-Information Processing*, volume 5506 of *Lecture Notes in Computer Science*, pages 945–952. Springer Berlin / Heidelberg, 2009.

[158] T. Susnjak, A. Barczak, and K. Hawick. A modular approach to training cascades of boosted ensembles. In *Structural, Syntactic, and Statistical Pattern Recognition*, volume 6218 of *Lecture Notes in Computer Science*, pages 640–649. Springer, 2010.

[159] T. Susnjak, A. Barczak, and K. Hawick. Adaptive ensemble based learning in non-stationary environments with variable concept drift. In Kok Wong, B. Mendis, and Abdesselam Bouzerdoum, editors, *Neural Information Processing. Theory and Algorithms*, volume 6443 of *Lecture Notes in Computer Science*, pages 438–445. Springer Berlin / Heidelberg, 2010.

[160] T. Susnjak, A. Barczak, and K. Hawick. Adaptive cascade of boosted ensembles for face detection in concept drift. *Neural Computing and Applications*, pages 1–12, 2011. ISSN 0941-0643. 10.1007/s00521-011-0663-x.

[161] T. Susnjak, A. Barczak, N. Reyes, and K. Hawick. A new ensemble-based cascaded framework for multiclass training with simple weak learners. In Pedro Real, Daniel Diaz-Pernil, Helena Molina-Abril, Ainhoa Berciano, and Walter Kropatsch, editors, *Computer Analysis of Images and Patterns*, volume 6854 of *Lecture Notes in Computer Science*, pages 563–570. Springer Berlin / Heidelberg, 2011. ISBN 978-3-642-23671-6.

[162] T. Susnjak, A. Barczak, N. Reyes, and K. Hawick. Coarse-to-fine multiclass learning and classification for time-critical domains. Manuscript submitted for publication, November 2011.

[163] J. Suutala and J. Röning. Methods for person identification on a pressure-sensitive floor: Experiments with multiple classifiers and reject option. *Information Fusion*, 9(1):21–40, 2008.

[164] V. Svetnik, A. Liaw, C. Tong, and T. Wang. Application of breiman's random forest to modeling structure-activity relationships of pharmaceutical molecules. *Multiple Classifier Systems*, pages 334–343, 2004.

[165] F. Takahashi and S. Abe. Decision-tree-based multiclass support vector machines. In *Neural Information Processing, 2002. ICONIP '02. Proceedings of the 9th International Conference on*, volume 3, pages 1418 – 1422 vol.3, November 2002.

[166] A. Tsymbal. The problem of concept drift: definitions and related work. *TCD-CS-2004-15, Dep. of Comp. Sci., Trinity College, Dublin, Ireland*, 4, 2004.

[167] K. Tumer and J. Ghosh. Error correlation and error reduction in ensemble classifiers. *Connection Science*, 8(3-4):385–404, 1996.

[168] L.G. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11): 1134–1142, 1984.

[169] V.N. Vapnik. *The nature of statistical learning theory*. Springer Verlag, 2000.

[170] R. Verschae and J. Ruiz del Solar. Coarse-to-fine multiclass nested cascades for object detection. *Pattern Recognition, International Conference on*, 0:344–347, 2010. ISSN 1051-4651.

[171] R. Verschae, J. Ruiz del Solar, and M. Correa. A unified learning framework for object detection and classification using nested cascades of boosted classifiers. *Mach. Vision Appl.*, 19(2):85–103, 2008. ISSN 0932-8092.

[172] P. Viola and M.J. Jones. Rapid object detection using a boosted cascade of simple features. In *CVPR01*, pages I:511–518, Kauai, HI, December 2001. IEEE.

[173] P. Viola and M.J. Jones. Robust real-time face detection. In *Proceedings of the Eighth International Conference On Computer Vision (ICCV-01)*, pages 747–747, Los Alamitos, CA, July 2001. IEEE Computer Society.

[174] P. Viola and M.J. Jones. Robust real-time face detection. *International journal of computer vision*, 57(2):137–154, 2004.

[175] P.A. Viola and M.J. Jones. Fast and robust classification using asymmetric adaboost and a detector cascade. In Thomas G. Dietterich, Suzanna Becker, and Zoubin Ghahramani, editors, *NIPS*, pages 1311–1318, 2001.

[176] H. Wang, W. Fan, P.S. Yu, and J. Han. Mining concept-drifting data streams using ensemble classifiers. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '03, pages 226–235, New York, NY, USA, 2003. ACM. ISBN 1-58113-737-0.

[177] P. Wang, H. Wang, X. Wu, W. Wang, and B. Shi. A low-granularity classifier for data streams with concept drifts and biased class distribution. *IEEE Transactions on Knowledge and Data Engineering*, 19:1202–1213, 2007. ISSN 1041-4347.

[178] W. Wang, P. Jones, and D. Partridge. Diversity between neural networks and decision trees for building multiple classifier systems. *Multiple Classifier Systems*, pages 240–249, 2000.

[179] G.I. Webb. Multiboosting: A technique for combining boosting and wagging. *Machine Learning*, 40:159–196, 2000. ISSN 0885-6125.

[180] D. Weiss and B. Taskar. Structured prediction cascades. In *Proc. AISTATS*, volume 1284, 2010.

[181] G. Widmer and M. Kubat. Learning in the presence of concept drift and hidden contexts. *Machine Learning*, 23:69–101, 1996. ISSN 0885-6125.

[182] F. Wilcoxon. Individual comparisons by ranking methods. *Biometrics Bulletin*, 1 (6):80–83, 1945.

[183] D. Withopf and B. Jahne. Improved training algorithm for tree-like classifiers and its application to vehicle detection. In B. Jahne, editor, *Proc. IEEE Intelligent Transportation Systems Conference ITSC 2007*, pages 642 – 647, 2007. doi: 10.1109/ITSC.2007.4357644.

[184] I.H. Witten, E. Frank, and M.A. Hall. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2011.

[185] D.H. Wolpert. Stacked generalization. *Neural Networks*, 5(2):241 – 259, 1992. ISSN 0893-6080.

[186] D.H. Wolpert and W.G. Macready. Coevolutionary free lunches. *IEEE Trans. Evolutionary Computation*, 9(6):721–735, 2005.

[187] J. Wu, J.M. Rehg, and M.D. Mullin. Learning a rare event detection cascade by direct feature selection. In *NIPS (Advances in Neural Information Processing Systems) 2003*, Vancouver, Canada, 2003.

[188] J. Wu, S.C. Brubaker, M.D. Mullin, and J.M. Rehg. Fast asymmetric learning for cascade face detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30:369 – 382, 2008.

[189] R. Xiao, L. Zhu, and H.J. Zhang. Boosting chain learning for object detection. In *ICCV '03: Proceedings of the Ninth IEEE International Conference on Computer Vision*, page 709, Washington, DC, USA, 2003. IEEE Computer Society. ISBN 0-7695-1950-4.

[190] R. Xiao, H. Zhu, H. Sun, and X. Tang. Dynamic cascades for face detection. *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*, pages 1–8, Oct. 2007. ISSN 1550-5499. doi: 10.1109/ICCV.2007.4409043.

[191] S. Yan, S. Shan, X. Chen, W. Gao, and J. Chen. Matrix-structural learning (msl) of cascaded classifier from enormous training set. In *Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on*, pages 1–7. IEEE, 2007.

[192] F. Yoav and R.E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. In *EuroCOLT '95: Proceedings of the Second European Conference on Computational Learning Theory*, pages 23–37, London, UK, 1995. Springer-Verlag. ISBN 3-540-59119-2.

[193] J.H. Zar. *Biostatistical analysis*, volume 564. Prentice hall Upper Saddle River, NJ, 1999.

[194] C. Zhang and Y. Ma. *Ensemble Machine Learning: Methods and Applications*. Springer-Verlag New York Inc, 2012.

[195] C. Zhang and P. Viola. Multiple-instance pruning for learning efficient cascade detectors. In *NIPS 2007*, 2007.

[196] C.X. Zhang and J.S. Zhang. A local boosting algorithm for solving classification problems. *Computational Statistics and Data Analysis*, 52(4):1928 – 1941, 2008. ISSN 0167-9473.

[197] Z.Q. Zhang, L. Zhu, S.Z. Li, and H.J. Zhang. Real-time multi-view face detection. In *FGR '02: Proceedings of the Fifth IEEE International Conference on Automatic Face and Gesture Recognition*, page 149, Washington, DC, USA, 2002. IEEE Computer Society. ISBN 0-7695-1602-5.

[198] Z.H. Zhou and Z.Q. Chen. Hybrid decision tree. *Knowledge-based systems*, 15(8): 515–528, 2002.

[199] J. Zhu, S. Rosset, H. Zou, and T. Hastie. Multi-class adaboost. Technical Report 430, Department of Statistics, Univesity of Michigan, 2006.

[200] X. Zhu, P. Zhang, X. Lin, and Y. Shi. Active learning from stream data using optimal weight classifier ensemble. *Systems, Man, and Cybernetics, Part B Cybernetics, IEEE Transactions on*, 40(6):1607–1621, 2010.