

Copyright is owned by the Author of the thesis. Permission is given for a copy to be downloaded by an individual for the purpose of research and private study only. The thesis may not be reproduced elsewhere without the permission of the Author.



COMPUTATIONAL METHODS FOR A GENERALISED ACOUSTICS ANALYSIS WORKFLOW

A THESIS PRESENTED IN PARTIAL FULFILMENT
OF THE REQUIREMENTS FOR THE DEGREE
OF MASTER OF SCIENCE IN COMPUTER SCIENCE

by
Yukio Fukuzawa

Supervisors:
Dr. Matthew Pawley
Prof. Stephen Marsland
Dr. Andrew Gilman

School of Natural and Computational Sciences
Massey University
Albany, New Zealand

February 2022

1	Overview	3
1.1	Introduction	3
1.2	Challenges	5
1.2.1	A precise way to segment syllables using both visualisation and sound playback	6
1.2.2	Syllables needs to be labelled in a flexible way, at different scales	7
1.2.3	A method for collaborative work between experts in regards to syllable classification	8
1.2.4	A user-friendly software that allows users to easily extend feature ex- traction capability	8
1.3	Contribution	9
1.4	Why <i>Koe</i> was written	9
1.5	Outline of the thesis	11
2	Design and functionality of <i>Koe</i>	12
2.1	Introduction	12
2.2	Languages, framework and libraries	12
2.2.1	<i>Koe</i> as a webapp	13
2.2.2	Data is stored in the backend and can be exported to the end user . . .	14
2.2.3	<i>Koe</i> 's backend is implemented in Python to maximise extendability . .	15
2.2.4	Libraries	15

2.3	Database design	16
2.3.1	<i>Koe's</i> database is designed to be user centred	17
2.4	Front-end design and framework	20
2.4.1	<i>Koe</i> workflow	20
2.4.2	Front-end design	20
2.4.3	Components of a page	22
2.4.4	Extendability	22
2.5	Functionalities of <i>Koe</i>	25
2.5.1	Upload and segment recordings	25
2.5.2	Extract acoustic features from units	25
2.5.3	Classify units	26
2.5.3.1	Interactive ordination plots	26
2.5.3.2	Unit tables	27
2.5.3.3	Class exemplars	28
2.5.3.4	Classification granularity	28
2.5.3.5	Validate classification through independent labelling	28
2.5.4	Analyse sequence structure	28
2.5.4.1	Filter songs by subsequence	29
2.5.4.2	Discover and visualise vocal patterns using sequence rule mining	29
2.5.5	Conclusions	30
3	Background	31
3.1	Basic concepts of Digital Signal Processing (DSP)	31
3.1.1	Sounds are signals recorded as a function of time, but best represented as a function of time-frequency	31
3.1.2	Analyse sounds in time-frequency	32
3.1.2.1	Naive Fourier Transform: the maths	32
3.1.2.2	Fast Fourier Transform: the practical implementation	33
3.1.2.3	Short-Time Fourier Transform: the usage	33
3.2	Sound production and perception in birds versus in humans	34

3.2.1	Birds can produce two sounds at once, within a wide range of frequency	34
3.2.2	Bird vocalisations are acoustic signals structured in time and frequency	36
3.2.3	Birds perceive sounds differently from human	37
3.3	Computational models based on sound production and perception	38
3.3.1	Model of sound production: the source-filter model	38
3.3.2	Model of frequency perception: non-linear filter-bank	39
3.3.3	Model of loudness perception: Equal loudness contour	41
3.4	Conclusions	42
4	Syllable segmentation	43
4.1	Introduction	43
4.2	Manual segmentation in <i>Koe</i>	44
4.3	Related work for automatic segmentation in birdsongs	45
4.4	Procedural algorithms	48
4.4.1	Endpoint detection in time using energy threshold: Harma method	48
4.4.2	Boundary detection in time and frequency using image processing: Lasseck method	50
4.5	Heuristic approaches	53
4.5.1	Feed-forward Neural Network with fixed size input	54
4.5.2	Recurrent Neural Network with variable size input	54
4.6	Data and evaluation	56
4.6.1	Data	56
4.6.2	Evaluation	56
4.6.3	Results	58
4.7	Conclusion	58
5	Feature representation	63
5.1	Introduction	63
5.2	Acoustic features	64
5.2.1	Descriptive features	65
5.2.1.1	Time domain features	65

5.2.1.2	Frequency domain perceptual features	66
5.2.1.3	Use of descriptive features in related work	68
5.2.2	Abstract features	68
5.2.2.1	Frequency domain physical features	68
5.2.2.2	Cepstral domain features	69
5.2.2.3	Use of abstract features in related work	70
5.3	Feature length standardisation	71
5.3.1	Aggregative methods	73
5.3.1.1	Summary methods	74
5.3.1.2	Resampling methods	74
5.3.2	Model-based methods	75
5.4	Implementation in <i>Koe</i>	77
5.4.1	User interface	77
5.4.2	<i>Koe</i> 's task queue	79
5.4.3	Implementation and expandability	79
5.4.4	Storage	81
5.4.4.1	Physical storage	82
5.4.4.2	Store and retrieve data	83
5.5	Conclusion	84
6	Visualisation and classification	86
6.1	Introduction	86
6.2	Cluster analysis	87
6.2.1	Hierarchical clustering	87
6.2.2	Semi-automatic clustering	89
6.2.3	Dimensionality reduction	90
6.3	User interface	90
6.3.1	Submit jobs to construct ordination and calculate similarity index . . .	90
6.3.2	Using similarity index to sort syllables in the unit table	91
6.3.3	Using ordination for bulk labelling in an interactive cluster visualisation	92
6.4	Implementation in <i>Koe</i>	94

6.4.1	Construct ordination	94
6.4.2	Calculate similarity index	95
6.4.3	Collaborative labelling	96
6.5	Case study: Validating classification with independent labelling	97
6.6	Conclusions	98
7	Sequence analysis	99
7.1	Introduction	99
7.2	Syntax discovery via sequence structure	100
7.2.1	Manual examination of subsequence via filtering	101
7.2.2	Automated subsequence discovery using N-gram	102
7.2.3	Automated subsequence discovery using SPADE	103
7.2.4	Analysis via visualisation with network models	105
7.3	Implementation	106
7.3.1	SPADE	106
7.3.2	Networks	107
7.4	Case studies: Evaluating song structure in NZ bellbirds	108
7.4.1	Using SPADE parameters	109
7.4.2	Using networks	110
7.5	Conclusion	110
8	Conclusions	112
8.1	Key contributions	112
8.2	Future work	113
8.3	Data availability	113
	Bibliography	114

LIST OF FIGURES

2.1	ER Diagram of the database structure generated by Django from the declared models	17
2.2	A simplified ER Diagram of <i>Koe</i> 's database structure.	18
2.3	<i>Koe</i> 's acoustics workflow and its accessibility from the main app	21
2.4	Organisation of a page in <i>Koe</i>	22
2.5	Other components of <i>Koe</i> 's main section	23
2.6	<i>Koe</i> 's interactive Ordination view	26
2.7	Unit table view in <i>Koe</i>	27
2.8	<i>Koe</i> 's songs list view	29
3.1	Waveform of a typical birdsong	32
3.2	The bird's syrinx and some complex sounds it creates	35
3.3	Multiple level structure of a birdsong revealed by a spectrogram	37
3.4	The source-filter model	39
3.5	Mel and Bark filter banks	40
3.6	Equal-loudness-contours	41
4.1	<i>Koe</i> 's syllable segmentation and adjustments	45
4.2	Spectrogram and the spectral peak value curve Harma method uses to find syllables	50
4.3	Processes of Lasseck's segmentation method	52
4.4	Syllable end-point detection with MLP using fixed size input	60

4.5	Syllable end-point detection with RNN using variable size input	61
4.6	Mean and two standard deviations of specific scores of different segmentation algorithms	62
5.1	Workflow	64
5.2	Descriptive acoustics features	65
5.3	Descriptive acoustics features	69
5.4	Summary methods	74
5.5	Resampling methods	75
5.6	Audio auto-encoder	76
5.10	Directory structure of feature values stored on the hard drive.	83
6.1	Illustration of hierarchical clustering using UPGMA	89
6.2	Control panels accessible under submenu <i>Extract features and compare</i> to create an ordination for clustering or similarity index for sorting.	91
6.3	Left side control panel of the unit table.	91
6.4	<i>Koe's</i> unit table after filtered and sorted by similarity index.	92
6.5	<i>Koe's</i> Interactive Ordination view	93
6.6	Accessing labelled data from other team members	97
7.1	<i>Koe's</i> songs list view	101
7.2	Table of subsequences and their SPADE parameters in <i>Mine sequence structure</i> page.	105
7.3	An example of a graph for a subset of syllables in <i>Koe</i>	106
7.4	Control panel in <i>Mine sequence structure</i> page to change graph parameters. . . .	108
7.5	Network visualisations of male and female song sequence structure.	111

ACKNOWLEDGEMENTS

This thesis is the result of a collective effort by so many people to salvage my failed PhD attempt after six years of struggles. However, I will always remember these six years fondly. I have met wonderful people, developed lasting friendships, and even found my missing half at this very institute! For all the people that are involved in this journey of mine, I would like you to know that I'm grateful for your supports during these years. I walked out of the process with a master thesis, a brand new acoustics tool under my belly, and a wife! If that's not a success, I don't know what is!

With that out of the way, I would like to thank all those who have joined me on this journey and made this outcome possible.

First I would like to thank Dr. Andrew Gilman, my main supervisor during the PhD period, who sparked my interest in academia and helped me get into PhD in the first place. You are a knowledgeable dude, a good friend, and a guy with an interesting sense of humour.

I want to thank Prof. Dianne Brunton for being unbelievably supportive during my entire time at Massey, despite not officially being my supervisor. Above all things, you are passionate about students and with you we always find someone we can trust and confide in. We are so lucky to have you to lead the institute.

I want to thank Dr. Mat Pawley, my main supervisor during the last two years. You are a great source of encouragement and a down-to-earth supervisor. A big part of *Koe*, sequence mining using SPADE, was your suggestion, and I wouldn't have made that without your encouragement.

I want to thank Prof. Stephen Marsland, my co-supervisor during this entire time. I truly

admire your deep knowledge of machine learning and hope that one day I can talk to people about it with complete confidence the way you explained things to me.

I want to thank Wesley Webb, my dear friend and co-author of *Koe*. *Koe* would not have been where it is now without your contribution. You're such a great dude, always humble and joyful, and a great listener. Plus serious presentation and Photoshop skills.

I want to thank Niloofar Aflaki, my wife, for your love and support which made it possible for me to complete this thesis. During my third and fourth year doing PhD, seeing you every now and then in the kitchen or at Zumba class made those gloomy days more bearable.

I want to thank my parents who helped bring me to New Zealand in the first place, and have been supportive of all the decisions I made. They were supportive when I quit a job to start a PhD, and were supportive when I quit a PhD to start a new job. They couldn't care less whether I have a PhD or not, as long as I'm happy - which is pretty rare for Asian parents. I'm grateful to be their son.

There are still many people I wish to thank, albeit not directly involved with my study, they in more than a handful of ways, have made my PhD life a little more enjoyable. So thank you Tim, Judy, Elijah and Claire; Jun and Shona, Azadeh, Hong and Peter; Tony; as well as all the admin staff at INMS.

1.1 Introduction

Acoustic communication plays important roles in the lives of many animal species, functioning in mate attraction, species recognition and resource defence (Marler and H. W. Slabbekoorn 2004, Ch.6). Unlike visual signals which need a direct line of sight, sound can propagate over long distances and where visibility is poor, e.g. dense vegetation (Howard 1999). Most vocalising taxa have been the subjects of bioacoustics research to some extent but none has been more extensively investigated than songbirds. The vocal activities of songbirds are amongst the most proliferate in the animal kingdom. The diversity of songbird morphology leads to a broad range of acoustic capability from subsonic to supersonic, from simple chirping to complex vocal patterns that bear many parallels to human speech (Sainburg et al. 2019). One crucial similarity between human speech and birdsong is that they are both combinatorial signals (Eimas et al. 1971; Zuidema and de Boer 2009) - meaning that the vocalisation patterns can be subdivided into smaller units. Evidence of combinatorial processing is found in species of other taxa such as Campbell's monkey (Ouattara et al. 2009), parrots (Dahlin and Wright 2012), rock hyraxes (Kershenbaum, Ilany, et al. 2012) however these are exceptions rather than the norm, whereas in songbird this process is ubiquitous (Kershenbaum, Déaux, et al. 2018). Combinatorial signal processing is the key to enable the reuse or adaptation of algorithms/tools developed for speech processing for birdsong, which makes this thesis possible.

In recent years, speech processing has become one of the most researched topics in machine

learning. We already enjoy the results of these research through the introduction and continuous improvement of digital assistance such as Siri and Cortana. Some aspects of speech processing can even be considered solved, for example, automatic speech recognition (ASR) and machine translation (Hirschberg and Manning 2015). This success in speech processing has inspired computer scientists to turn their attention to animal vocalisations. Studies of bird vocalisation have broad implications for other fields of biology and can even change or deepen the way we understand our own species, as they can potentially answer questions about the evolution of language, the process of language learning in children, etc. From a computer scientist's point of view, my main aim is to investigate how computer algorithms can be used to facilitate various kind of research of birdsong.

A large proportion of reported research in birdsong is concentrated on the identification of bird species from their songs or calls. Competitions for species recognition from songs have been held such as NIPS4B 2013, ICML4B 2013, MLSP (annual) and BirdCLEF (annual). Smartphone apps such as WARBLR (Warblr 2020) have been developed for identifying the vocalising species from a live recording taken by user. A less well researched topic is the analysis of songs from the same species, such as mimicry of brood parasites (Ranjard, M. G. Anderson, et al. 2010), song crystallisation of juvenile birds (Derégnaucourt et al. 2005; O. Tchernichovski et al. 2004; Wellock and Reeke 2012), impacts of environmental noise or vegetation density on song characteristics (Brumm and H. Slabbekoorn 2005; Francisco et al. 2002), dialects between different populations (Luis F. Baptista and King 1980; Rothstein and Fleischer 2007; Williams and P. J B Slater 1990), and differences between the vocalisation of male and female (Webb et al. 2021). These are different case studies of the same broad analysis which looks to find subtle differences between one set of vocalisation and the other. In this thesis, I refer to this analysis as dialect analysis, although the terms "dialect" in human speech is usually used in a much narrower scope.

The analysis of dialects in birdsong is also different from the analysis of human speech. In humans, dialects are usually pre-defined (Nerbonne and Heeringa 2001); studies are carried out to analyse the differences after the fact. For birds, it is hard to tell whether the differences we perceive with our human ears are also significant to the bird. The reverse is also true: birds generally have much higher temporal acoustic resolution; two sets of vocalisation might be dis-

tinguishable to the birds, but do not sound dissimilar to us until they are played back in slow speed. Without an established method to determine whether dialects exist or how to quantify them, it is no surprise that dialect analysis still requires heavily manual operations and decision-making from experts. My thesis therefore aims at investigating and developing computer algorithms that can be used by biologists to speed up their analysis.

These algorithms broadly include: syllable segmentation, syllable classification, clustering, visualisation, and sequence analysis, which are now implemented in *Koe*, a web-based software that I developed in collaboration with a biologist at my institution, Dr. Wesley Webb. This thesis is centred around the science and practicality of *Koe* for bioacoustics research.

1.2 Challenges

In contrast with human speech, we don't have a priori knowledge of bird's perception of units, including temporal boundary (where a unit starts and ends) and how much a unit can vary before it is perceived as a distinct class (Lachlana and Nowickia 2015). This leads to two major challenges: how to determine the vocalisation unit of a species (segmentation), and how to determine whether two units are different enough to be considered distinct (classification). These are the core issues computer scientists face when designing algorithms for bioacoustics research.

Birdsong is thought to be hierarchical in structure, divisible into different levels of complexity (Berwick et al. 2011). The common approach is to divide a song into the smallest vocal units based on the durations of the sounds and the silent gaps between them. These units can have various names depending on the species, but the most common terminology is "syllable".

The concept of a "syllable" is discussed in depth in Section §3.2.2. For now, a syllable can be understood as a continuous sound signal separated from the other units by a silent gap. Having a satisfactory set of syllables precisely segmented and consistently labelled is a crucial step prior to any analysis that can be then carried out. However, the accuracy requirements differ from analysis to analysis. For studies that do not look deeply at the syllable label, a crude input is good enough as long as they contain enough information to differentiate the totality of these syllables as a whole, and therefore a faster albeit less accurate algorithm for segmentation and classification might satisfy. For example, species identification doesn't even need the syllables

to be labelled, as long as the input can be transformed into a numeric representation that helps distinguish one species from another. For dialect analysis, accuracy trumps speed, especially if the raw data is limited; the accuracy cost to implement an automatic algorithm often outweighs potential time-saving benefits.

The field of acoustics software is very fragmented. Software that provides solutions for accurate manual processes exists, however they are often specialised in only one part. Raven (Bioacoustics Research Program 2011) is good at partitioning long recordings into individual songs. Luscinia (R F Lachlan 2007) is good at segmenting syllables. Excel is so far still a widely used tool for syllable classification. Software such as Sound Analysis Pro (Ofer Tchernichovski et al. 2000) and Avisoft (Specht 2002) (both proprietary) provide a convenient interface to extract acoustics measurements from syllables, but they are locked in a small set of features and the users have no way to introduce other features. None of these software provides sequence analysis, the researchers often have to create bespoke code themselves to get this done. In order for a biologist to conduct a typical dialect analysis, they have to transfer their data between three, four different software plus their own coding. An end-to-end software solution that can piece together all these fragments is still lacking. In Section §1.2.1, I detail what such software needs to fulfill.

1.2.1 A precise way to segment syllables using both visualisation and sound playback

How much gap is required to separate syllables is up to the intuition of the researcher based on their experience. There's currently no way to tell whether this gap length is correct from the bird's perspective. Some species might be able to discern a smaller gap, some might only recognise a longer gap. Even if the gap length could be correctly identified, automated gap detection in birdsong recordings from the field could still be difficult due to unwanted noise masking the silent gaps. Often for in-depth analysis of birdsongs, precision is far more important than speed, thus the common approach is still to carry out segmentation manually.

Manual segmentation is usually performed not only by listening to the sound, but also by looking at the spectrogram - a visualisation of the sound in time and frequency. A syllable is presented as a blob of pixels representing high energy signal on a plain background. The start

and end points of this pixel blob is the start and end of the syllable in time. Depending on how the spectrogram is generated, each pixel can represent longer or shorter period of time. Using both the spectrogram and playback, the biologist can speed up the segmentation process significantly, especially if the spectrogram can be fine-tuned and the playback speed can be adjusted.

1.2.2 Syllables needs to be labelled in a flexible way, at different scales

How much a syllable can change before it is considered a different type is an unsolved problem. In human speech, virtually every person has their own slightly different way to pronounce the same word. However our vocabulary is constructed as discrete categories. It is well understood that categorical perception is also ubiquitous in the animal kingdom (Baugh et al. 2008; Ehret and Haack 1981; May et al. 1989; Wyttenbach et al. 1996) and especially in birds (Lachlana and Nowickia 2015; Nelson and Peter Marler 1989). However, when there is a gradient of incremental differences between exemplars, it can be difficult to define category boundaries without some form of verification from the animal. Robert Lachlan and his student Lies Zandberg (in. comm.) at the University of London are doing just that by performing playback experiments on zebra finches and correlating their behaviours with differences in acoustics of incrementally different syllables. Their experiment is currently underway and when published will be the first ever to attempt finding correct categories of syllables from the birds perspective. This kind of study is difficult and expensive, so biologists are likely to require their own expert opinion to perform categorisation of sound units.

Analysing dialects by song structure is therefore heavily predicated on having "correct" syllable categorisation. If the criteria for "being different" is too relaxed, one risks having too many unique syllable types and too few instances per category. On the other hand, a more inclusive approach will risk overlooking legitimate dialectal variants. A better software solution should support labelling at different scales, but none of the existing software solutions had support for this.

1.2.3 A method for collaborative work between experts in regards to syllable classification

Even with multi-scale labelling scheme, it is impossible to avoid personal biases. Different experts rarely arrive at identical categorisation even for a stereotypic repertoire. When manual categorisation is involved, most studies settle on a classification that the researchers self-assess to be "reasonable" (A. E. Jones et al. 2001). Inter-observer reliability assessment to establish the degree of agreement between experts has been carried out only in a small number of studies, for example, (Kershenbaum, Blumstein, et al. 2016; Nelson, Hallberg, et al. 2004; Parker et al. 2012; Soha et al. 2004; Tomback and Myron Charles Baker 1984). As well as the time-consuming issue each researcher faces when they label sounds, existing software solutions don't provide enough support for collaboration at this step. A better solution should have built-in features that allow experts to share, assess, and work on the same syllable set simultaneously, where the products from individual labellers can be quickly aggregated and assessed.

1.2.4 A user-friendly software that allows users to easily extend feature extraction capability

Feature extraction is arguably the most important part of any acoustics analysis. If segmentation and labelling are done perfectly but the features are not suitable, the results are meaningless. On the other hand, there exist features that are tolerant to noise or imperfect timing of the segmentation. So it is more crucial to get the right set of features than it is to get the segmentation and labelling done perfectly. On this front, all software being used for acoustic analysis offers feature extraction, but the levels of quality vary. As most of them are closed source, the user is left at the software creator's mercy to have certain features they need implemented. One exception is *Luscinia* which is open source; however, extending on *Luscinia* is an enormous task as there is a severe lack of scientific computation for Java, the language *Luscinia* is written in. A better solution should be open source and easily extended. Because programming at a level required to extend a software via plug-ins is outside the expertise of many biologists, this is not an option. Extending the software by having it written in languages that have strong support from the scientific community is a much more achievable direction.

1.3 Contribution

In this thesis, I develop a computer process for analysing birdsong dialect and implement a brand new software named *Koe* that facilitates real-world analysis of that nature. *Koe* can also be used for other kinds of analysis of birdsongs, and can also be used for other animal species too. As of the time this thesis is written, *Koe* has a user base of nearly 800 users all around the world, including bird enthusiasts, biology students and researchers. Collectively *Koe* is now hosting more than 1300 datasets, many of which are non-bird sounds. During the process of writing *Koe*, I and my colleagues published or have submitted the following papers:

- Fukuzawa, Y., Marsland, S., Pawley, M.D.M., & Gilman, A. (2016, November). *Segmentation of harmonic syllables in noisy recordings of bird vocalisations*. In 2016 International Conference on Image and Vision Computing New Zealand (IVCNZ) (pp. 1-6). IEEE.
- Fukuzawa, Y., Webb, W. H., Pawley, M.D.M., Roper, M. M., Marsland, S., Brunton, D. H., & Gilman, A. (2020). *Koe: Web-based software to classify acoustic units and analyse sequence structure in animal vocalizations*. *Methods in Ecology and Evolution*, 11(3), 431-441.
- Webb, W. H., Roper, M. M., Pawley, M.D.M., Fukuzawa, Y., Harmer, A. M., & Brunton, D. H. (2021). *Sexually distinct song cultures in a songbird metapopulation*. bioRxiv.
- Roper, M. Michelle, Webb, W.H, Fukuzawa Y, Evans C. Hammer, M.T. A., Brunton, D. H. (2021). *Sexual and temporal variation in New Zealand bellbird song repertoires*. bioRxiv.

1.4 Why *Koe* was written

Initially, the software that developed into *Koe* was a tool to provide both manual segmentation and classification of bellbird song syllables on the same platform, in an attempt to get a cleaner dataset from what my colleagues at Massey University, Dr. Wesley Webb and Dr. Michelle Roper had been collecting and processing for years. Prior to *Koe* they were limited to a cumbersome workflow because of the number of different software involved in the process, which became too unwieldy as more data was added.

An ideal workflow would be iterative, where at the last step the biologists could go back to the first step and refine the result. However as the data had to be exported and imported through

several applications, this was next to impossible. The result was a chunky, inconsistently labelled dataset. Many syllables that sounded distinctly different from each other were given the same label, and vice versa, many syllables that sounded similar were given different labels. Analyses performed on this initial dataset produced results that were no better than chance.

Koe was written to integrate all tasks in the bioacoustics workflow into one unified platform. The data is imported to *Koe* as raw recordings, then they are segmented into syllables, and then labelled. If during the labelling process the biologist detects any inconsistency, it is very simple to go back and correct the mistake. For example, a syllable initially labelled X under re-examination might be split into two syllables Y and Z, then all the researcher needs to do is go to the segmentation page, split syllable X into two syllables, and give them label Y and Z. In *Koe*, this change will be propagated automatically throughout the workflow, whereas in other software, the entire process needs to be redone.

In classification, one scale of granularity is often not enough. We spent much time discussing whether the label set {A, B, B, C} is more applicable than the set {A₁, A₂, B₁, B₂, C₁, C₂, ...}. The solution was to allow more than one level of labelling simultaneously, and we settled at having two scales: a broad scale and a fine scale. Another obstacle to efficient labelling was the sheer number of syllables involved. We examined one dataset (of bellbird song) which contained 21845 individual syllables. On a typical screen, only about 20 syllables can be shown in one page due to the space necessary to display spectrograms. Even with the broad scale, it was hard to keep track of the syllable labelled a few pages ago. To alleviate this issue, another feature was added to *Koe*: bulk labelling by cluster analysis. Syllables are transformed into feature space and dimensionally reduced to two, such that they can be visualised as datapoints in a graph. Syllables that are acoustically similar appear near each other in the graph, where the user can then group them with a lasso tool and label them in bulk.

Bulk-labelling datapoints in two-dimensional acoustic feature space is intuitive, however it depends on how well the features can represent acoustics differences. To arrive at a good set of features is one of the key challenges of acoustic analysis. There is no rule about what features are better than other, so most studies chose them through trial and error. Therefore, when implementing *Koe*, instead of favouring my own choice of features and forcing the user to stay within that choice like other software does, I incorporated as many acoustic features I

could find in the literature. *Koe*'s feature set includes all of *Sound Analysis Pro*, *Luscinia*, *Raven*, and other well known features. It is also possible for the user to define their own features by extending on *Koe*'s source code, which is released under MIT licence. *Koe*'s feature extraction is written entirely in Python using well-supported scientific libraries. Although a little slower than code written in C/C++, it makes the extension of *Koe* easier for students and researchers in biology or other fields that are not involved in programming.

1.5 Outline of the thesis

This thesis is centred around the scientific aspects of *Koe*. It is organised as follows:

- **Chapter 2:** I discuss in broad terms the functionalities of *Koe* and the decisions for the programming framework and user interface design that deliver these functionalities.
- **Chapter 3:** I introduce key concepts of Digital Signal Processing (DSP) for bioacoustics, the differences of sound production and perception between birds and humans, and the models thereof.
- **Chapter 4:** I discuss syllable segmentation methods for birdsongs including manual and automation attempts, and how these techniques are implemented in *Koe*.
- **Chapter 5:** I review acoustics features that are commonly used in bio-acoustics.
- **Chapter 6:** I discuss feature-space clustering methods and how it is visualised in *Koe* for rapid syllable labelling.
- **Chapter 7:** I discuss how sequence analysis is implemented in *Koe* and its application in analysing birdsong dialects. The chapter is concluded with a case study using *Koe* to analyse song structures of male and female Bellbirds.
- **Chapter 8:** Conclusions and future work

2.1 Introduction

Existing acoustics software prior to *Koe* were designed for single users and distributed as stand alone, installable packages. Some are only available for Windows (*Sound Analysis Pro*), some require installing extra software to function (*Sound Analysis Pro* requires MySQL, *Luscinia* requires Java). *Koe* is a web app, to the best of my knowledge, at the time of writing this thesis, it's still the first and only acoustic software distributed as a web app. Being a web app removes all obstacles for the general public from accessing the platform. In addition, *Koe* is also packaged as a Docker image, allowing more tech-savvy users to download and install it locally on their own computer or for their institution. For researchers who are familiar with coding, the source code of *Koe* is available for download, change and run at localhost. Table 2.1 shows the targeted users, purpose of use, and the URL where the three distributions of *Koe* can be found. In this chapter, I discuss in broad terms the functionalities of *Koe* and the decisions for the programming framework and user interface design that deliver these functionalities.

2.2 Languages, framework and libraries

Before the inception of *Koe*, I was involved in two open source programs: AviaNZ¹ which was created by my supervisor Prof. Stephen Marsland, and *Luscinia*² by Robert Lachlan in an at-

¹<https://www.avianz.net/>

²<https://rflachlan.github.io/Luscinia/>

Table 2.1: Three kinds of distribution of *Koe*. *Koe* is distributed through three methods: as an free online tool, as a runnable Docker image, as an MIT-licence open source project.

Type	Online	Docker image	Source code
User	Researchers and the general public	Researchers or technicians at an institution	Programmers / Researcher with coding knowledge
Purpose	To use the most recent version of <i>Koe</i> without any setup	To set up their own <i>Koe</i> server for speed and keeping their data private	To customise <i>Koe</i> , e.g. extend functionality
Access point	www.koe.io.ac.nz	hub.docker.com/repository/docker/crazyffan/koe	github.com/fzyukio/koe

tempt to extend their functionality to serve what I and my colleagues needed - a software that allows labelling of syllables to be carried out rapidly and with multi-granularity support. Both programs were set up in a way that rendered this job near impossible in reasonable time. So *Koe* was invented out of necessity to fulfil an immediate need. However the design from the beginning has always been to make *Koe* more than just a short-term solution. By comparing existing acoustic software (paid and free) with what I envisaged for *Koe*, three design principles were made:

- *Koe* needs to be accessible without complicated installation process
- *Koe* needs to support multi-user out of the box
- *Koe* needs to be easily extendable

2.2.1 *Koe* as a webapp

Unlike all other acoustics software, *Koe* runs as a webapp. The official site is publicly available at <https://koe.io.ac.nz>, but it can also be setup to run locally and made accessible only to local users of an institution. As a webapp, there is no need for the users to install anything, all they need is a modern web browser. This proves useful for collaboration, for example, cooperative labelling of a dataset by multiple experts, or projects involving the general public. In Chapter 6, I presented a case study that was organised as a citizen science project with a large number of non-expert participants. This project was only possible thanks to *Koe* being accessible as a webapp.

2.2.2 Data is stored in the backend and can be exported to the end user

Data in *Koe* consists of 1) raw audio files, 2) compressed audio files, 3) syllable end-points, 4) spectrograms, 5) label data and 6) extracted feature data. All data in *Koe* is user-specific and is foreign-key constrained to the user key. They are stored on the server using various storages:

- Raw and compressed audio files are stored physically on the hard drive. Unlike *Luscinia* which stores everything in an H2 database (making the program extremely slow and memory consuming), only the ID and name of the audio files are stored in the database. Storing audio files on the hard drive also allows *Koe* to read only an individual segment of a file and not the entire file, significantly reducing load time. This is necessary because in *Koe* there are many processes that involve only certain kinds of syllables; for example, the user can filter syllables based on their length, labels, start and end points, etc. It would be much slower to read 100 audio files in full just to extract 200 syllables from them than to read only the parts of 100 files that contains those 200 syllables.
- Syllable metadata is stored in the database. The metadata includes only start and end point, min, max and average fundamental frequency, making it very light-weight and filterable.
- Syllable label is stored in the database and is foreign key constrained to several identifiers
 - Syllable key: which identifies the syllable it is labelled for.
 - User key: which identifies the person who gives a syllable its label.
 - Attribute key: which identifies the granularity of the syllable, e.g. broad-scale or fine-scale.
- Spectrograms are stored as static images to speed up front-end rendering. A spectrogram of a syllable never changes regardless of the label and the user, therefore it is unnecessary to generate each syllable on the fly at the front end. Storing spectrograms as static images increases hard drive use, however as the images are small and compressed (PNG format) the increase is not significant and the speed gained is a well worth the trade off.

- Extracted feature data is stored as binary files on the hard drive. This is the second most space-intensive data after raw audio files. The number of distinct features that can be extracted from each syllable is more than 100, therefore reading feature data is the most intense in hard-drive access. Well-known binary storage libraries such as H5 are not fast enough to accommodate this large number of features. I wrote a new binary storage library that allows rapid data access similar to the way syllables are read from sound files. The design and implementation of this library is detailed in section [§5.4.4.1](#).

Users of *Koe* own their data, thus I added functionalities to the front end to allow users to download any of the aforementioned data (audio files, extracted feature data, and label data), as well as exporting data from the table they are viewing in CSV format. The exported label data can also be used to restore their dataset if they have lost or damaged their data i.e. by changing labels in bulk by mistake. It also facilitates data migration should they wish to move their data from the official *Koe* website to a local copy of *Koe* running on their own server.

2.2.3 *Koe*'s backend is implemented in Python to maximise extendability

At the time *Koe* was being conceived, there were two languages with the most support from the scientific community: Python and R. The levels of support differ from domain to domain. Scientific computation and plotting libraries often support both languages, while Python has an edge on acoustics libraries, and R has stronger support for statistics. However, there is no mature webserver framework for R, while for Python there are numerous - most notably Django and Flask. The decision was made to select Django over Flask because it is more mature and flexible than Flask.

2.2.4 Libraries

Koe makes use of the following libraries for scientific computation:

- NumPy (Python Software Foundation [2017](#)) for fast data manipulation, as almost every computation performed in *Koe* is based on array or matrix.
- Scipy (E. Jones et al. [2014](#)) which provides basic signal processing and statistical tools
- librosa (Mcfee et al. [2015](#)) is used to extract many well-known acoustics features.

- Scikit-learn (Pedregosa et al. 2012) for dimensionality reduction
- Scikit-image (der Walt et al. 2014) for image-based features.
- Tensorflow (TensorFlow 2017) is used only at the backend for machine-learning tasks. This area is still under development and once completed will expand the capability of *Koe* significantly.

In addition, I have written a number of libraries to provide scientific computation not supported by any library, these are:

- **python-matlab-functions**³ is a python reimplementation of several useful functions in Matlab
- **mt-spec-features**⁴ implements all multi-tapering features which are the core of Sound Analysis Pro (Mcfee et al. 2015)
- **python-cspade**⁵ is reimplementation of the algorithm Spade (Zaki 2001) in C++, with python wrapper written in CPython

In order to provide an easy way to add more features to *Koe*, all features provided in the aforementioned libraries are wrapped in a common interface, such that the call to any feature has the same signature no matter which library it comes from.

2.3 Database design

The Django framework is database-agnostic and comes bundled with an Object Relational Mapping (ORM) framework that builds appropriate database structure for any backend based on the declared models and their relationship. For example, a simple database "driver" with two models - Person and Car - in Code block 1 will generate the database structure presented in figure 2.1.

I chose MySQL for *Koe*'s database backend. This decision is made based on the observation that MySQL has very high performance and is well-understood by researchers and IT/CS students (it is the database of choice for teaching at Massey University). However, advanced users

³<https://github.com/fzyukio/python-matlab-functions>

⁴<https://github.com/fzyukio/mt-spec-features>

⁵<https://github.com/fzyukio/python-cspade>

```

1 from django.db import models
2
3 class Person(models.Model):
4     name = models.CharField()
5     birth_date = models.DateField()
6
7 class Car(models.Model):
8     make = models.CharField()
9     year = models.IntegerField()
10    model = models.CharField()
11    owner = models.ForeignKey(Person)

```

Code block 1: Simple database model declared in Python following Django's ORM modelling system. The database has two models: a Person model with name and birth date and a Car model with make, year, model and is owned by a Person

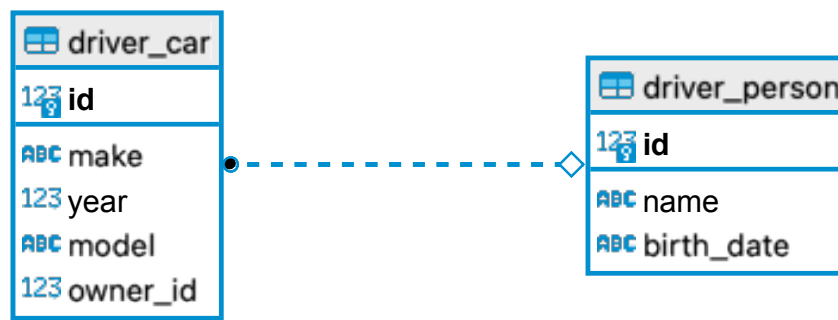


Figure 2.1: ER Diagram of the database structure generated by Django from the models declared in Code block 1. The actual database is SQLite, imported into DBeaver to generate this diagram.

of *Koe* using the Docker or source code distribution on their own server are free to choose any database they like. *Koe* has been tested to run smoothly on PostgresDB and SQLite. PostgresDB has the benefit of high performance and supports concurrency better than MySQL, however it is more difficult to set up. SQLite is stored as a single file on the hard drive so it is very easy to set up and migrate, however it's not designed to have high performance on large scale.

Koe's database models are quite complicated. Figure 2.2 shows a trimmed down version of the diagram that visualises the relations between models in *Koe*. Note that there are far more actual tables in MySQL needed to realise these models and relationship.

2.3.1 *Koe*'s database is designed to be user centred

Unlike other bioacoustics software, *Koe* requires the user to sign up and log in before they can start their work. Users need to first create a Database, of which they are the owner, or be invited

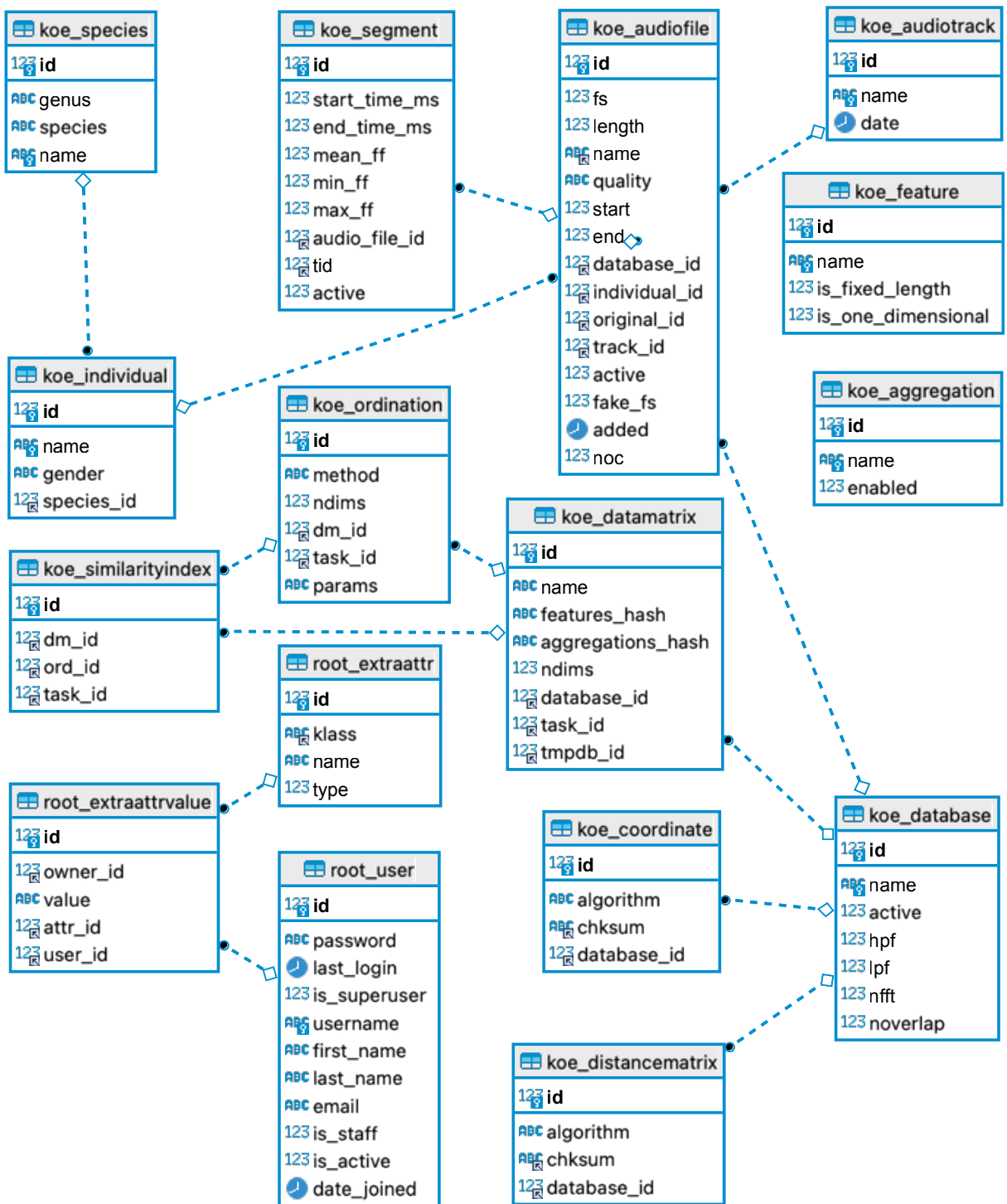


Figure 2.2: A simplified ER Diagram of Koe's database structure. The actual database is MySQL. DBeaver connects to the database to generate this diagram

to work on someone else's database, which they can be granted different access rights. The owner has full control of what other users can do, in particular, there are 9 levels of access that a user can have, from lowest to highest, as specified in Table 2.2. The lowest 6 permissions (View, Annotate, Import data, Copy files, Download files, Add files) are considered safe as what the user is permitted to do does not affect other users' data at all. *Modify segments* and *Delete files* change the collection of syllables as well as altering the start and end points of the syllables, thus are considered "unsafe". The safe permissions are designed to facilitate various needs of a citizen science project, such as allowing the general public to participate in data labelling, or make recordings and upload files to an open dataset. The unsafe permissions are reserved for close associates of the database owner, allowing collaborators to adjust the dataset without having to consult the entire group.

Table 2.2: User permissions in Koe. Asterisk denotes actions that effect other user's data

Permission	View	Annotate	Import data	Copy files	Download files	Add files	Modify segments	Delete Files	Admin
View label	X	X	X	X	X	X	X	X	X
Playback sound	X	X	X	X	X	X	X	X	X
View clustering	X	X	X	X	X	X	X	X	X
View sequence structure	X	X	X	X	X	X	X	X	X
Create new label set		X	X	X	X	X	X	X	X
Copy other user's label data			X	X	X	X	X	X	X
Copy files into their own database				X	X	X	X	X	X
Download compressed audio files					X	X	X	X	X
Download extracted feature data					X	X	X	X	X
Add new audio files						X	X	X	X
Adjust start and end points of syllable							X	X	X
Delete syllables *							X	X	X
Add new syllables *							X	X	X
Delete audio files (and all segments therefrom) *								X	X
Change other user's permission									X

2.4 Front-end design and framework

2.4.1 *Koe* workflow

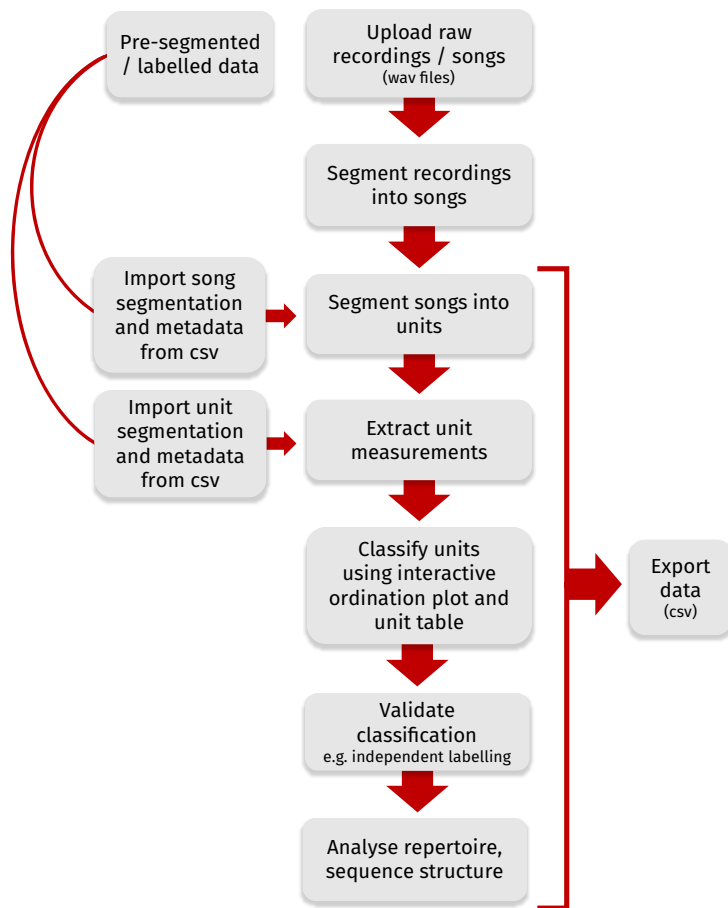
I designed *Koe* to provide end-to-end functionality in an intuitive and flexible workflow, as illustrated in Figure 2.3a. Each step in the workflow has a specific *page* in *Koe* where the user can perform the task (e.g. segmenting, measuring, classifying, etc.). Any step in the workflow can be revisited, and modifications will dynamically update throughout the program. A more detailed breakdown of these step is provided in *Koe*'s documentation ⁶. In general, the user starts with uploading their data to *Koe* in the form of raw recordings / songs (in wav format). Recordings are divided into songs, which are segmented into units. *Koe* extracts unit features to produce similarity indices and interactive ordination plots, which help a user to rapidly and accurately classify units. Once complete, classification can be validated by independent labelling. *Koe* provides tools for analysing repertoires and sequence structure. Data can also be exported for external use.

Similar to the back end, the front end of *Koe* is also designed to be easily extendable. From the start, this design has allowed *Koe* to evolve incrementally as I extended the back end to facilitate more modes of analysis. *Koe* provides one webpage for each stage of the acoustics process. In *Koe* these pages are accessible through a side menu, and are organised in an order similar to that of the corresponding stages in a typical acoustics process.

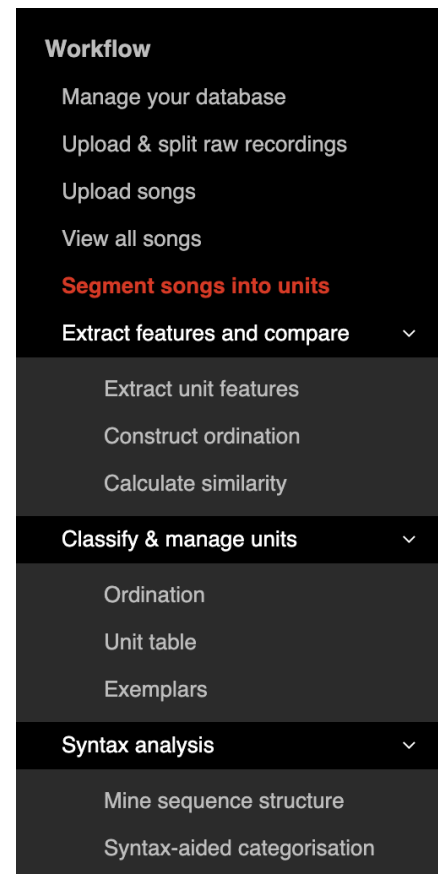
2.4.2 Front-end design

Koe uses a mixture of the Model-View-Controller (MVC) and Single-Page Application (SPA) pattern to take full advantage of both server-side and client-side rendering. When a page is visited the first time, the server generates skeleton HTML code from templates. When this code reaches the client and finishes loading all JavaScript codes, the final result is rendered via extra AJAX requests and shown to the user. From this point any user interaction with the page is handled by an AJAX request, i.e. the page does not reload after each action. This is necessary to keep the app responsive, because a full server-side rendering method will reload a page with

⁶<https://github.com/fzyukio/koe/wiki>



(a) A typical acoustics process carried out in *Koe*



(b) *Koe*'s side menu organisation of pages

Figure 2.3: *Koe*'s acoustics workflow and its accessibility from the main app. A *Koe* database is integrated; any step in the workflow can be revisited, and modifications will dynamically update throughout the program. Raw recordings / songs (in wav format) are uploaded. Recordings are segmented into songs, which are segmented into units. *Koe* extracts unit features to produce similarity indices and interactive ordination plots, which help a user to rapidly and accurately classify units. Once complete, classification can be validated by independent labelling. *Koe* provides tools for analysing repertoires and sequence structure. Data can also be exported for external use.

up to tens of thousands of rows every time there's a smallest change, such as to label a syllable.

AJAX requests from the client side are responded with JSON-formatted data, and not HTML code. This is yet another necessity as a typical user's dataset contains hundreds up to tens of thousand of syllables, each syllable has many properties, making it unfeasible to convert a dataset into ready-to-display HTML code in a timely manner, especially when there are multiple users active at the same time. Thus, upon an AJAX request, the server simply responds with the pure, raw data that can be queried directly from the database, and leaves rendering a responsibility of the client side.

2.4.3 Components of a page

Pages in *Koe* have a similar structure: a side menu on the left and a main section that spans the remaining space. The side menu contains a control section, which is defined by the page template, and a navigation menu to allow access to other pages. The main section can contain various component, such as spectrogram, song info, playback control, and a spreadsheet-typed grid. The grid is the most common component of a page, although some pages might have more than one grid (such as the Database management page), or no grid at all (such as the Ordination page). Figure 2.4 shows the layout of the syllable segmentation page with different components organised in the main section and Figure 2.5 shows other components used in different pages of *Koe*.

The screenshot shows the Koe interface for syllable segmentation. On the left is a side menu (1) with a control section (3) and a navigation menu (4). The main section (2) contains a sound sonogram (5) and a sound spectrogram (6) at the top. Below them is a playback controller (7) with 'Play' and 'Stop' buttons. A metadata table (8) is on the left, and a data grid (9) is on the right. The data grid contains the following information:

Start	ID	End	Duration	Family	Subfamily	Label	Note
<input type="checkbox"/> 119.00	9283	398.00	279.00	Chump		Ghh_Ghh_Chump	
<input type="checkbox"/> 600.00	9284	726.00	126.00	Chump		Chump	
<input type="checkbox"/> 1160.00	9285	1559.00	399.00	Pipe		Pipe(E[ε]bent)	
<input type="checkbox"/> 1747.00	9286	2241.00	494.00	Upsqueak		Upsqueak(harmonic)	
<input type="checkbox"/> 2490.00	9287	2715.00	225.00	Pipe_Down		Pipe_Down_Tink	
<input type="checkbox"/> 2884.00	9288	3166.00	282.00	RoughBuzz		Spiderfangs	
<input type="checkbox"/> 4539.00	9289	4825.00	286.00	Downsqueak		Downsqueak(alarmy)	
<input type="checkbox"/> 4953.00	9290	5051.00	98.00	Tink		Tink	
<input type="checkbox"/> 5660.00	9291	5795.00	135.00	Cough		Cough(short)	
<input type="checkbox"/> 6585.00	9292	6882.00	297.00	Chump		Ghh_Ghh_Chump	
<input type="checkbox"/> 7064.00	9293	7172.00	108.00	Chump		Chump	

Figure 2.4: Organisation of a page in *Koe*: **Side menu** 1, which contains the control section 3 and the navigation menu 4; **Main section** 2 which can contain any combination of: sound sonogram 5, sound spectrogram 6, playback controller 7, metadata 8, data grid 9

2.4.4 Extendability

Most of these components are used in more than one page, for example, the spectrogram section is used in both "Upload & split raw recordings" and "Segment songs into units"; the data grid is used in almost every page, sometimes more than once per page. In order to achieve that, these

Feature extraction queue

Following are pending feature extraction tasks submitted by all users for all databases and collections

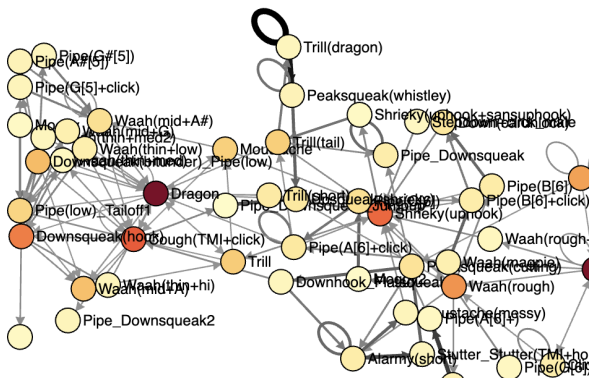
Task	User	Database	Name	Created	Started	Progress
New extraction						
Existing data matrices						
Bellbird_TMI: basic Completed						
Features						
<input type="checkbox"/> spectral_flatness	<input checked="" type="checkbox"/> spectral_bandwidth	<input type="checkbox"/> spectral_centroid				
<input type="checkbox"/> spectral_contrast	<input type="checkbox"/> spectral_rolloff	<input type="checkbox"/> mfcc				
<input type="checkbox"/> zero_crossing_rate	<input type="checkbox"/> total_energy	<input type="checkbox"/> aggregate_entropy				
<input type="checkbox"/> average_entropy	<input type="checkbox"/> average_power	<input type="checkbox"/> max_power				
<input checked="" type="checkbox"/> max_frequency	<input type="checkbox"/> frequency_modulation	<input type="checkbox"/> amplitude_modulation				
<input type="checkbox"/> goodness_of_pitch	<input checked="" type="checkbox"/> amplitude	<input type="checkbox"/> entropy				
<input type="checkbox"/> mean_frequency	<input type="checkbox"/> spectral_continuity	<input checked="" type="checkbox"/> duration				
<input type="checkbox"/> frame_environ	<input type="checkbox"/> average_frame_power	<input type="checkbox"/> max_frame_power				

(a) Job lodgement

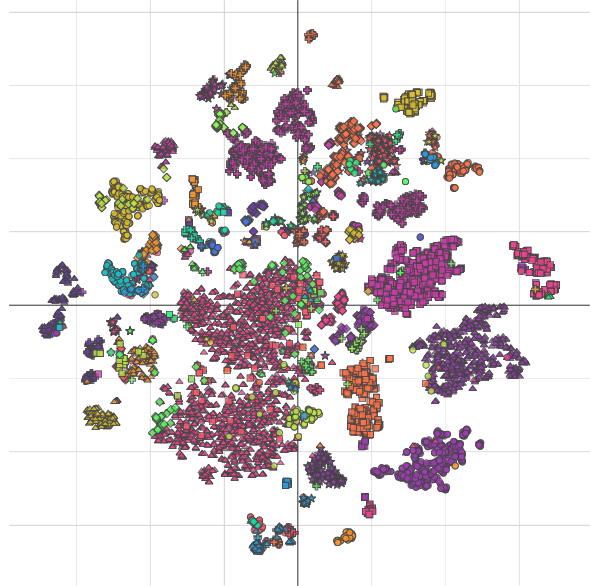
Databases		Collections		User & Permissions			Saved versions		
Name		Name		Username	Permission	Backup type	Form		
<input type="checkbox"/> BirdCLEF		<input type="checkbox"/> TMI_F		<input type="checkbox"/> wesley	Assign User	<input type="radio"/> segmentation	4		
<input type="checkbox"/> Bellbird_Jusodb				<input type="checkbox"/> supuser	Assign User	<input type="radio"/> labels	4		
<input checked="" type="checkbox"/> Bellbird_TMI				<input type="checkbox"/> DHB	Annotate	<input type="radio"/> segmentation	4		
<input type="checkbox"/> Test				<input type="checkbox"/> mroper	Delete Files	<input type="radio"/> labels	4		
<input type="checkbox"/> My_database				<input type="checkbox"/> Alisama96	Annotate	<input type="radio"/> labels	4		
<input type="checkbox"/> Warbler flight calls				<input type="checkbox"/> RedBeard	Annotate	<input type="radio"/> labels	4		

Start	End	Duration	Song	Spectrogram	Family	Subfamily	Label
520.00	617.00	97.00	TMI_2015_02_01_MMR016_01_M		Pipe	Z	Pipe(F6)
1066.00	1277.00	211.00	TMI_2015_02_01_MMR016_01_M		Stepup	Z	Stepup(C17-D17)
1534.00	1795.00	261.00	TMI_2015_02_01_MMR016_01_M		Pipe_Downsqueak	Z	Pipe_Downsqueak2
2264.00	2493.00	229.00	TMI_2015_02_01_MMR016_01_M		Stepup	Z	Stepup(C17-D17)
2834.00	3102.00	268.00	TMI_2015_02_01_MMR016_01_M		Pipe_Downsqueak		Pipe_Downsqueak2
430.00	554.00	124.00	TMI_2015_02_01_MMR017_01_F		Chump		Chilup
703.00	825.00	122.00	TMI_2015_02_01_MMR017_01_F		Stepdown		Stepdown_mid
877.00	967.00	90.00	TMI_2015_02_01_MMR017_01_F		Pipe		Pipe(D6female)
1045.00	1125.00	80.00	TMI_2015_02_01_MMR017_01_F		Chump		Chilup

(c) Multi-grid system



(b) Interactive force-directed graph



(d) Interactive MDS plot

Figure 2.5: Other components of *Koe*'s main section: **(a) Job lodgement**, here the figure shows the feature extraction section, where the user can lodge a job to extract features from syllables in their dataset. **(b) Interactive force-directed graph**, in *Koe* it is used for sequence analysis. **(c) Multi-grid system**, here the figure shows the main section of the database management page. **(d) Interactive multi-dimensional scaling (MDS) plot**, in *Koe* it is used for cluster analysis and bulk syllable labelling.

```

1 // Extend FlexibleGrid to change properties specific to this grid.
2 class SyllableGrid extends FlexibleGrid {
3   init() {
4     super.init({
5       'grid-name': 'labelling',
6       'grid-type': 'segment-info',
7       'default-field': 'label_family',
8       gridOptions
9     });
10  }
11 }
12 // Create a new grid to display syllables
13 export const syllableGrid = new SyllableGrid();

```

Code block 2: The instantiation of the syllable grid used in *Segment songs into units* page. The `SyllableGrid` extends `FlexibleGrid` to overwrite specific properties of the syllable grid.

```

1 const databaseGrid = new FlexibleGrid();
2 const collectionGrid = new FlexibleGrid();
3 const dbAssignmentGrid = new FlexibleGrid();
4 const versionGrid = new FlexibleGrid();
5 const syllableGrid = new FlexibleGrid();

```

Code block 3: The instantiation of all grids used in the *Database management* page. None of them extends `FlexibleGrid` as the default properties and behaviours of `FlexibleGrid` are sufficient.

components are written as generic classes. A specific page creates subclasses from these components to overwrite certain properties or behaviours. The grid system showcases the strength of this approach. Each grid is an instance of a `FlexibleGrid`, which is a heavily modified version of `SlickGrid`⁷. The modified version is partly available at <https://github.com/fzyukio/SlickGrid>, although most of *Koe*-specific plug-ins are only available in *Koe*'s source code.

A `FlexibleGrid`, as the name suggests, is written in a way that makes it very flexible. It can be used out of the box, or extended with very little overwriting. For example, Code block 2 shows how the data grid used in the syllable segmentation page (Figure 2.4) is instantiated by extending the `FlexibleGrid`, and Code block 3 shows how all five grids used in the database management page (Figure 2.5d) are instantiated directly from the `FlexibleGrid`. *Koe*'s grids are vastly different from each other, yet can be handled by essentially the same code.

⁷<https://github.com/mleibman/SlickGrid>

2.5 Functionalities of *Koe*

Each step of the workflow presented in section §2.4.1 has a program view tailored to that task. Figure 2.3b shows the navigation menu of *Koe* with links to access these pages. As can be seen, there is almost a one-to-one mapping between the workflow and the organisation of these links. In this section, I highlight significant features of each program view. The coming chapters will go into details of their algorithms and operations.

2.5.1 Upload and segment recordings

In *Koe*, the user can partition raw recordings into individual vocalisation bouts, termed "songs" (*Upload & split raw recordings* view), and segment songs into their constituent acoustic units (*Segment songs into units* view). The tasks are similar; they involve the user selecting start/end-points of songs or units, respectively, on a spectrogram, then committing the selections to the database (stored securely on the *Koe* server). It is up to the user to decide on segmentation criteria appropriate to their species/question. The user can adjust playback speed, spectrogram contrast and time-axis zoom. Saved selections become available in other program views. Acoustic units are not stored as audio segments, but as start/endpoint information referencing the source song; the user can freely readjust unit segmentation and program views will update dynamically. For any songs already segmented into units in other software, start/endpoints can be imported as a csv file. Pre-partitioned song files can also be uploaded directly to the database in *View all songs*. Users can grant database access to other users, with custom permission levels.

2.5.2 Extract acoustic features from units

In quantitative analysis, raw signals are often replaced by their compact representations using a specific set of features, extracted from the signals. Analyses performed on compact representations can be more effective and computationally efficient. In *Extract unit features* view, *Koe* can extract a wide array of spectral-, tempo- and chroma-related features for bioacoustic analyses⁸. Extracted features are utilised by *Koe* to construct ordinations and similarity indices (Sections §2.5.3.1 and §2.5.3.2), and can be exported as csv files for analysis in other software.

⁸See <https://github.com/fzyukio/koe/wiki>

2.5.3 Classify units

Manual classification requires the ability to visually/acoustically compare and label large numbers of units quickly. *Koe* offers interactive ordination plots, unit tables and class exemplars as complementary tools for this purpose.

2.5.3.1 Interactive ordination plots

A previously unexplored potential of ordination is to expedite the manual classification of units. *Koe*'s interactive ordination plots (Figure 2.6) incorporate audio playback, spectrograms and classification functionality, so that a user can simultaneously use their audio-visual perception of unit similarity and the structure of the data to rapidly and robustly classify units

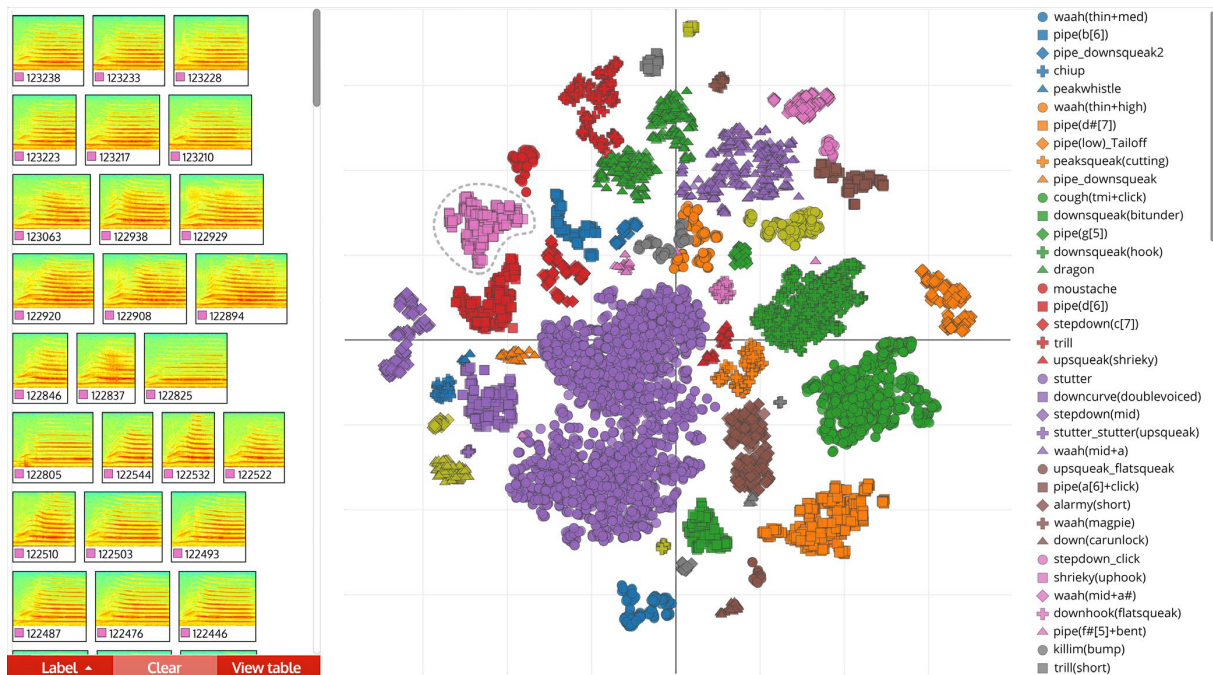


Figure 2.6: Interactive ordination view allows the user to encircle groups of points on the plot with the lasso tool, to view their spectrograms and hear their audio. Mousing over a point in a selection highlights the corresponding spectrogram in the left-hand panel. Selections can be labelled in bulk directly on the plot or opened as a unit table to view detailed unit information. The user can zoom, toggle the visibility of classes, and export the plot as a vector graphic. This example shows a t-SNE ordination of 7189 syllables of male and female bellbird song.

Koe implements three ordination techniques: Principal Component Analysis (PCA; (Pearson 1901)), Independent Component Analysis (ICA; (Comon 1994)), and t-distributed Stochastic Neighbour Embedding (t-SNE; (Van Der Maaten and Hinton 2008)). t-SNE aims to preserve local structure in the data and is particularly effective for defining and discriminating between

clusters. The user encircles groups of points on the plot to see spectrograms and hear playback of units; if a selection appears acoustically consistent, it is classified in bulk. The user can toggle visibility of each class independently and can zoom to examine structural detail.

2.5.3.2 Unit tables

As a complement to the ordination plot, units can be viewed as an interactive table (Figure 2.7). Each unit is represented by a row containing spectrogram, audio and associated information (class label, unit duration, song ID, individual ID, date, etc.). The table can be sorted/filtered by any column. A notable feature is the similarity index, which ranks units based on acoustic similarity. The index is produced as follows: from the raw feature measurements or from the ordination, *Koe* calculates pairwise Euclidean distance between each pair of units, then constructs a ladderised dendrogram using agglomerative hierarchical clustering (UPGMA) (Sokal 1958). The order of the dendrogram leaf nodes becomes the similarity index. Sorting by the similarity index column orders the table so that similar units arrange together, allowing them to be selected and labelled in large batches

3/38 Filter family: down|upsqueak

Song	<input type="checkbox"/>	Spectrogram	Label	Family	<input checked="" type="checkbox"/> Sex	Quality	Similarity Index	Duration	Date	Note
CUV_2016_11_05_	<input type="checkbox"/>		Down(notched)	<input checked="" type="checkbox"/> Down	<input type="checkbox"/> Bulk set value		52	111	2016-11-05	
CUV_2016_11_06_	<input type="checkbox"/>		Down(notched)	<input checked="" type="checkbox"/> Down	<input checked="" type="checkbox"/> F	EX	60	137	2016-11-06	
CUV_2016_11_05_	<input type="checkbox"/>		Down(notched)	<input checked="" type="checkbox"/> Down	<input checked="" type="checkbox"/> F	EX	62	117	2016-11-05	
CUV_2016_11_05_	<input type="checkbox"/>		Down(notched)	<input checked="" type="checkbox"/> Down	<input checked="" type="checkbox"/> F	G	70	113	2016-11-05	
LBI_2016_04_06_	<input type="checkbox"/>		Upsqueak(harmonic)	<input checked="" type="checkbox"/> Upsqueak	<input checked="" type="checkbox"/> M	EX	423	291	2016-04-06	
LBI_2016_04_06_	<input type="checkbox"/>		Upsqueak(harmonic)	<input checked="" type="checkbox"/> Upsqueak	<input checked="" type="checkbox"/> M	EX	430	312	2016-04-06	
LBI_2016_04_09_	<input type="checkbox"/>		Upsqueak(harmonic)	<input checked="" type="checkbox"/> Upsqueak	<input checked="" type="checkbox"/> M	VG	446	373	2016-04-09	
TMI_2014_03_19_	<input checked="" type="checkbox"/>		Upsqueak(shrieky)	<input checked="" type="checkbox"/> Upsqueak	<input checked="" type="checkbox"/> M	G	689	162	2014-03-19	
TMI_2014_11_10_M	<input checked="" type="checkbox"/>		Upsqueak(shrieky)	<input checked="" type="checkbox"/> Upsqueak	<input checked="" type="checkbox"/> M	G	721	172	2014-11-10	

Figure 2.7: *Koe*'s unit table is designed for classifying, annotating and filtering units. Each unit row contains a spectrogram which becomes enlarged during mouse-over. Unit audio plays when a spectrogram is clicked. The table can be sorted/filtered by any columns. Sorting by the similarity index column arranges units by spectral similarity for expedited labelling.

2.5.3.3 Class exemplars

A class catalogue is a useful reference during classification. *Koe* produces one automatically in Exemplars view, with exemplar spectrograms and playback for every class, making visual/acoustic comparison easy. The catalogue updates dynamically as classification progresses, displaying 10 randomly-chosen exemplars per class to reflect within-class variability.

2.5.3.4 Classification granularity

Without a priori knowledge of the animal's perception of units, a researcher must identify classes and assign units based on their own perception. Classifying units at multiple hierarchical levels of granularity increases robustness by enabling analyses at different scales. *Koe* offers up to three granularity levels (fine, medium, and broad-scale) for labelling in the Ordination and Unit table views. For example, the units in Figure 2.7 are labelled at two granularity levels: the broad-scale Upsqueak family is subdivided at the fine-scale into Upsqueak(harmonic) and Upsqueak(shrieky).

2.5.3.5 Validate classification through independent labelling

To ensure robustness of manual classification, a common validation method is to have several judges independently label the dataset and calculate their degree of agreement; high agreement lends credibility to the classification (Nelson, Hallberg, et al. 2004; Parker et al. 2012). Typically, 2–5 judges are used, but robustness can improve substantially with more judges (A. E. Jones et al. 2001). *Koe* facilitates independent labelling experiments. A subset of songs/units can be selected and copied to a new database. The database owner grants labelling access to participating judges, who label the dataset online. The labels of participants are automatically saved to the server and are compiled to evaluate concordance (see Section §6.5 for a real-world example using 74 judges).

2.5.4 Analyse sequence structure

For many animals, songs are comprised of acoustic units ordered into a sequence (Kershenbaum, Blumstein, et al. 2016). The aim of sequence analysis is to reveal patterns in the sequence structure of songs, which *Koe* can do in three ways, outlined below.

2.5.4.1 Filter songs by subsequence

In *View all songs* a user can filter for songs that contain a certain subsequence of unit labels (Figure 2.8). This could be used to identify all instances of a certain song type, for example.

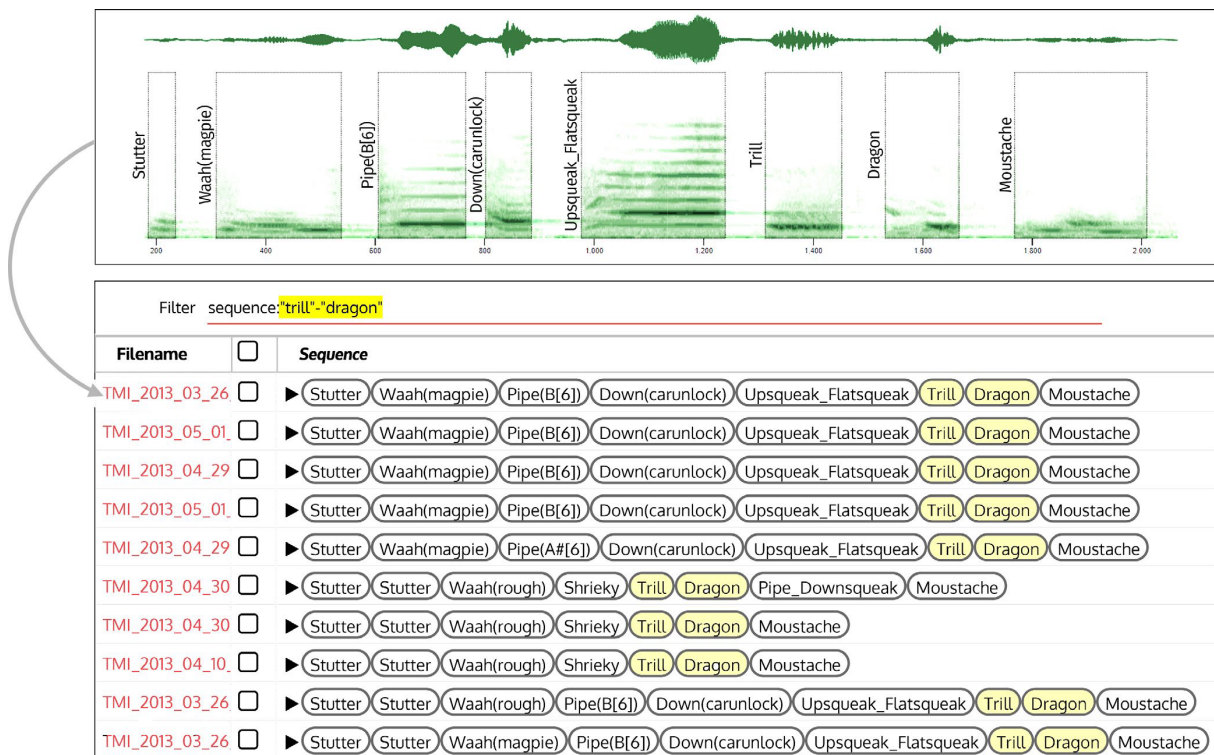


Figure 2.8: *View all songs* displays songs in an interactive table, with one song per row. Segmented songs (like the example in the top panel) are represented as a sequence of unit labels. Entire songs can be played, or individual labels clicked for unit playback and spectrograms. Here the filter has returned all songs containing *Trill* followed by *Dragon* (highlighted).

2.5.4.2 Discover and visualise vocal patterns using sequence rule mining

As one way of exploring rules that may govern song structure i.e. syntax; (Robert F. Lachlan et al. 2013), *Koe* uses the SPADE (constrained Sequential Pattern Discovery using Equivalence classes) algorithm (Zaki 2001) to discover commonly-occurring sequences in a set of songs.

Two-unit associations from SPADE can be visualised using a directed network. The network models the direction and strength of association between pairs of units, across a population of songs. Units are represented by nodes which are joined by lines (edges) if the units occur consecutively. The order of units is indicated by arrow direction, and strength of association between units (lift) is represented by edge thickness. Visually cluttered networks can be simplified using the filter, e.g. to show only associations with high lift.

Thus, *Koe* offers pattern recognition and sequence visualisation. If a user desires to make formal inferences, such as assessing the influence of experimental factors (fixed effects) and random effects of individuals on sequence structure, sophisticated Markovian frameworks exist (Sarkar et al. 2018). A user can export sequence data from *Koe* for external analysis.

2.5.5 Conclusions

Koe is an acoustics tool that provides a seamless workflow for biologist from the input of raw recording to the final results of feature extraction, clustering analysis and sequence analysis. It is designed to be flexible for end-user with any level of coding experience. *Koe* delivers high precision results by facilitate manual operations in most of the processes, making no assumption about the acoustics properties of the vocalisation of any bird species. This is due to the differences between birdsongs and human speech, rendering the application of speech processing techniques on birdsongs unreliable. In the next chapter, I go over these differences and how they affect the models of sound production/perception when applied to bird songs.

3.1 Basic concepts of Digital Signal Processing (DSP)

3.1.1 Sounds are signals recorded as a function of time, but best represented as a function of time-frequency

A sound is a mechanical wave of pressure which changes over time. As such it can be viewed as a time series sequence of air pressure. Sound can also be characterised by the frequency (or frequencies) at which it oscillates. When viewed as a time series, the sound is said to have a temporal structure: the pattern of energy it carries at a specific point in time. This is how audio signals are recorded and played back by a digital device such as a computer. More precisely, audio signals are represented as a function of the magnitude corresponding to the variant in voltage when a sound reaches the microphone. The plot of this function is called an *oscillogram*, or a *waveform* (the latter term is more frequently used). This plot is not always useful as it does not display any information about the frequency of the sound. Thus, it is far more common in acoustics to transform the raw audio input (pure time domain) into the time-frequency domain. The visualisation of a time-frequency domain function is called a spectrogram. Figure 3.1 shows both the waveform (top) and the spectrogram (bottom) of the same bird song for comparison: elements of the call display clear *spectral* contents that change from element to element, while the waveform only shows temporal and amplitude modulation.

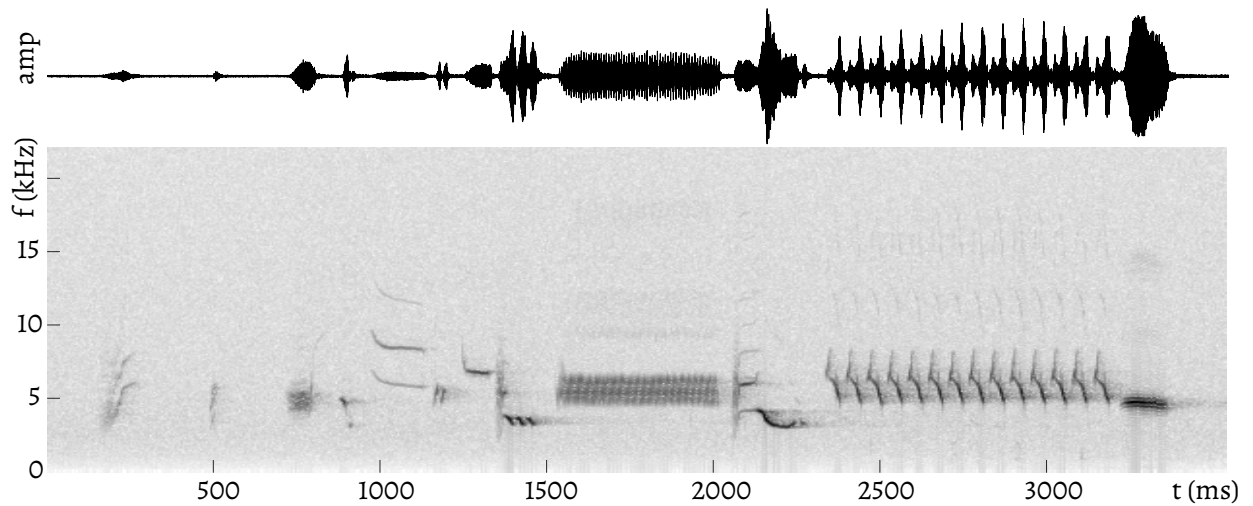


Figure 3.1: Oscillogram (top) and spectrogram (bottom) of a Bewick's Wren *thryomanes bewickii* song. FFT parameters: $f_s=44100\text{Hz}$, $n_{\text{FFT}}=512$, $\text{overlap}=256$, Hamming window. Copyright: Ted Floyd (Licensed under CC BY-NC-SA 4.0). Accessible at www.xeno-canto.org/318283. Only one of the three recorded songs is shown.

3.1.2 Analyse sounds in time-frequency

The earliest method to transform signal from a time domain to a time-frequency domain is the Fourier transform and it is still one of the most used tools in audio analysis. It is based on the observation made by the French mathematician Joseph Fourier in the 19th century that any periodic function can be expressed as the sum of an infinite number of sine and cosine waves.

3.1.2.1 Naive Fourier Transform: the maths

Mathematically, a discrete Fourier transform is a series of filters (a filter bank) where each filter converts a time series sequence into a single value, which is interpreted as the magnitude of the corresponding frequency component. Mathematically, the magnitude of the k -th frequency component, $\mathcal{X}[k]$, is calculated as:

$$\mathcal{X}[k] = \frac{1}{N} \sum_{n=0}^{N-1} x[n] e^{-2\pi i n \frac{k}{N}} \quad (3.1)$$

Where $x[n]$ is the original time series sequence and N is the total number of frequency components.

A Fourier transform with $N=512$ results in 512 frequency components, of which only the first half is useful because for sequences of real numbers such as audio signal, the frequency domain is always symmetric. Together these 256 components cover the range of frequencies from 0 to a

cut-off threshold called the Nyquist frequency, which is expressed in Hz and has value equal to half the sampling rate (f_s). E.g. an audio with $f_s = 10000$ samples per second has the Nyquist frequency of 5000 Hz. The k -th frequency component has centre frequency of $\frac{k}{N} \frac{f_s}{2}$ Hz.

3.1.2.2 Fast Fourier Transform: the practical implementation

Calculating all k components according to equation 3.1 is very time-consuming (it has computational complexity of $O(n^2)$). In 1965 Cooley and Tukey invented Fast Fourier Transform (Cooley and Tukey 1965) which reduces the complexity to $O(n \log(n))$ and quickly became an indispensable algorithm in digital signal processing.

3.1.2.3 Short-Time Fourier Transform: the usage

An important assumption the Fourier transform makes is that the time series sequence is *continuous* and *stationary*. Both are unrealistic for real life audio signals. Human speech and bird songs are considered stationary only in short time (typically 20-30 ms (Raju et al. 2012)), thus anything longer than that needs to be analysed frame by frame, with each frame enclosing a period of stationary signal. Each frame is typically overlapping with the previous as to not miss out the fine details between two successive frames. This gives rise to the technique called Short-Time Fourier Transform (STFT) that does exactly that. The result is not one sequence of frequency magnitudes but a collection of frame-level frequency components lined up in time. A visualisation of this two-dimensional data is called a *spectrogram*, where analysis in time and frequency can be done *simultaneously*.

The frame size decides the frequency and time resolution of the spectrogram. A frame needs to be chosen such that it is large enough to cover low-frequency components (that change slowly hence more samples are required) but small enough to capture fast-changing and short-lived components at high frequencies. The number of frequency bins is (because of symmetry) always half of the frame size, while the frequency range is entirely dependent on the sampling rate. Thus the frequency resolution is proportional to the frame size. However, larger frame size results in poorer time resolution, since the location of short-lived, high-frequency components dissolve in a larger time window. This is known as the *principle of uncertainty* in time-frequency, which is closely related to the infamous *Heisenberg uncertainty principle*. Choosing an

acceptable frame size is dependent on the nature of the sound (e.g. the rate of change in human speech is less than in birdsongs, thus an application of birdsong analysis generally should use a smaller frame size, although it also depends on the actual species) as well as the performance constraints (smaller frame size results in more frames, each needs one Fourier transform operation thus more computational power is needed).

3.2 Sound production and perception in birds versus in humans

3.2.1 Birds can produce two sounds at once, within a wide range of frequency

Studies into the parallels between bird vocalisations and human speech have revealed a number of commonalities in terms of communication, acquisition and development (Patricia K Kuhl 1989; P. Marler 1970). Even the effect of social interaction on the ontogeny of vocalisation repertoires in human babies and young birds is similar (Doupe and P K Kuhl 1999; P. K. Kuhl 2003). These parallels exist despite striking differences in the sound producing organs (Riede and Goller 2010). However, there are key differences between birdsongs and human speech, making it less straightforward to apply speech processing techniques to birdsongs.

In birds, sounds are produced by the syrinx which is part of the respiratory system and is located at the caudal end of the trachea. Both sides of the syrinx can function at the same time and independently from each other which allow some bird species to create two sounds simultaneously. In contrast, the human larynx is located at the top of the throat and can only produce one sound at a time. However, human can make even more complex sounds than birds by skilfully coordinating the *larynx*, tongue, lips and teeth, the ability that is lacking in birds.

Figure 3.2 shows the generalised anatomy of the avian syrinx, illustrating the double voice mechanism and examples of the double voicing as it appears in spectrograms. Their range of frequency can be much wider than human speech and the rate of change can be much steeper. This might challenge many well-known techniques used in human speech as to whether they can be effectively applied to birdsongs. The acoustics properties of human speech are more restrictive than that of bird songs, due to many biological constraints we share as a species. Human speech typically has fundamental frequency in the range 50-250Hz (male) and 120-400Hz (female)(Bagshaw et al. 1993), and we don't vary our pitch change much while speaking.

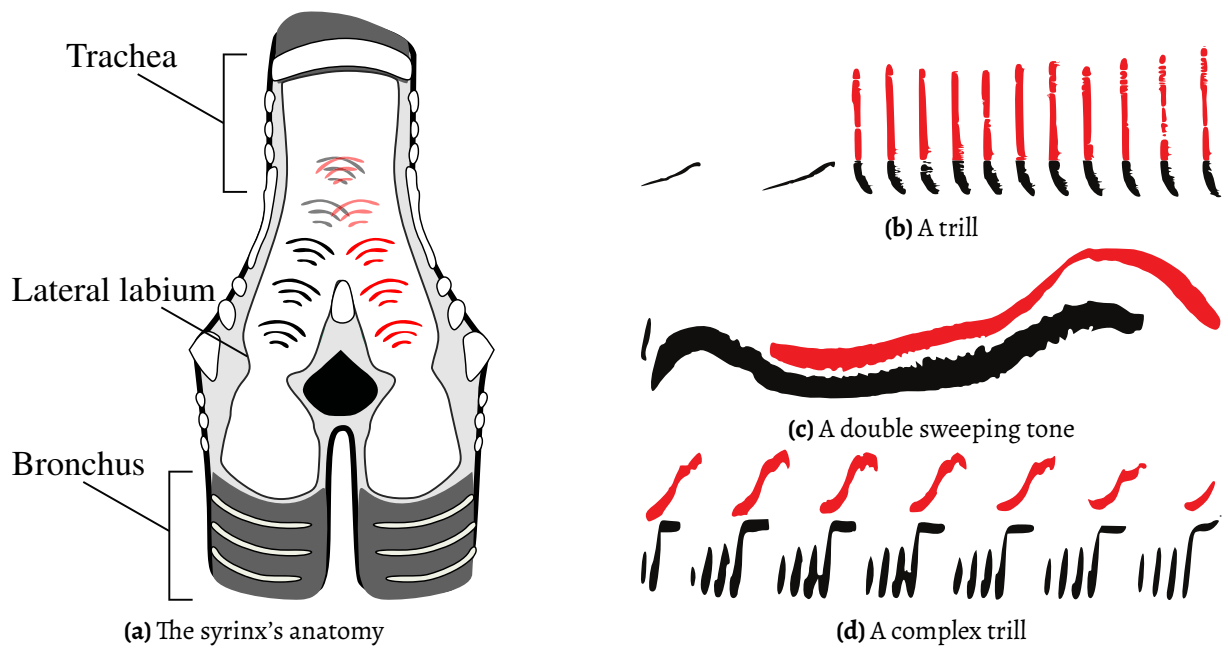


Figure 3.2: The bird's syrinx (a) and spectrograms of some complex sounds birds can make with the use of their double-voice syrinx, which contribute different spectral content to the sound (red and black parts). b) A Northern Cardinal's trill c) A Brown Thrasher's double sweeping tone d) A Wood Thrush's extremely complex trill with a series of higher-pitch sweeping tones layered above a series of lower-pitch mini-trills. Figure adapted from <https://academy.allaboutbirds.org/birdsong/>

This is far less variable than that of a typical bird species.

Birds on the other hand can have several of magnitude wider range of frequency (for example, Table 3.1 shows the frequency ranges of several New Zealand bird species as observed by Priyadarshani (Priyadarshani 2016)) and they have wide-spread ability to make sweeping sounds that span a large range of frequency in a short amount of time.

Table 3.1: List of species, their call types and frequency range

Species/call type	Observed frequency range (Hz)
North Island brown kiwi	
Male	500 - 8000
Female	500 - 6500
Ruru	
"Trill" sound	500 - 8000
"More" sound	500 - 2000
"Pork" sound	500 - 2000
Kakapo	
"Booming" sound	0 - 800
"Chinging" sound	1000 - 12000

3.2.2 Bird vocalisations are acoustic signals structured in time and frequency

Birdsong is thought to be hierarchical in structure, able to be divided into different levels of complexity (Berwick et al. 2011), however the naming of these levels has not been standardised (Thompson et al. 1994). Given the diversity of bird vocalisations, it is entirely understandable that finding a one-size-fits-all naming system for all species is difficult and probably useless. There have been attempts in proposing a standardised system of naming such as Shiovitz (1975) who found 20 terms that were used to describe 10 levels of vocal units in indigo buntings and proposed to consolidate them into 6 categories. Thompson et al. (1994) found in addition 9 more terms and recognised that proposing a rigid naming system is futile. Instead they proposed to use a formula using three basic units called T, Mr and Ms to quantify any naming system. However these proposals have not been adopted. The common theme however, is to divide a song into smaller vocal units based on the duration of the units and the silent gaps between them, based on the discernment of human audition. The smallest unit of bird vocalisation is the “*element*” (sometimes referred to as “*note*”), which is very short and typically go together with other elements to form a syllable. In many cases, birds produce two sounds at once using two sides of their syrinx simultaneously (See section §3.2.1) and each sound is an element with no gap between them. Syllable is most commonly used to describe the unit that are separated from other units by a silent gap, although as Kershenbaum, Blumstein, et al. (2016) points out, changes in acoustic features without the presence of a silent gap can also indicate that a new syllable has started. A group of syllables or elements often go together in a pattern is a trill, motif, or phrase. A song can be a long phrase or contains multiple phrases. The information in birdsong is encoded in the types of unit present and sometimes their temporal arrangement (Kershenbaum, Blumstein, et al. 2016).

In this thesis, I use the definition of a syllable as defined by Ranjard and Ross (2008): “*a syllable is part of a song characterised by a high value of autocorrelation of the signal and with a continuity in the fundamental frequency*”. Multiple syllables that follow a particular order form a phrase and a song can contain multiple phrases. Figure 3.3 shows the structure of a typical song of New Zealand’s North Island saddleback (*Philesturnus rufusater*) using a spectrogram.

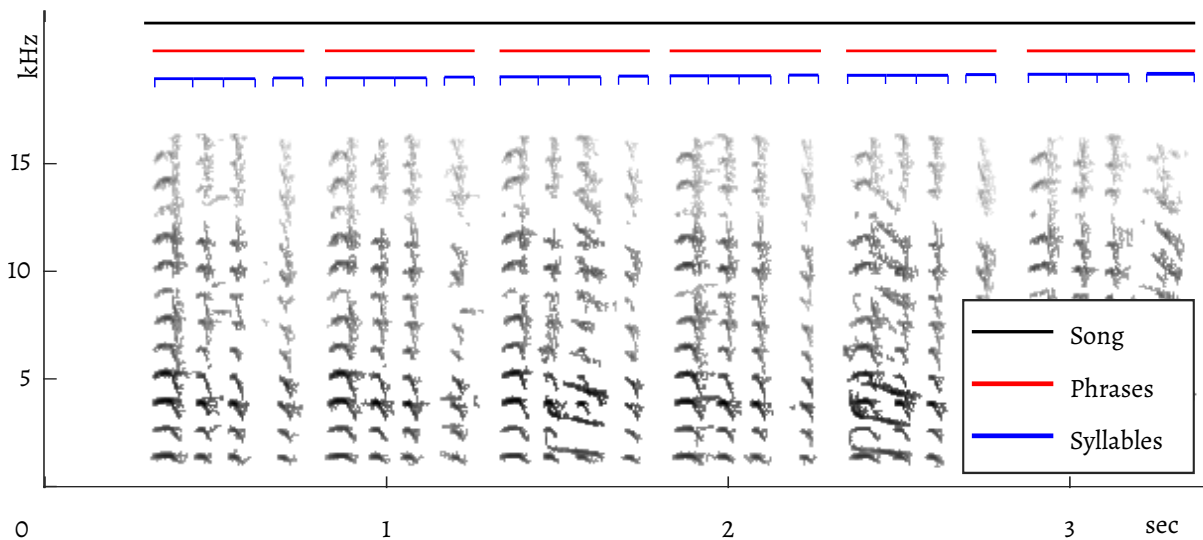


Figure 3.3: Spectrogram of a typical song of *Philesturnus rufusater* showing multiple level structure. (Excerpt of Xeno-canto Media ID #114331 with noise removed for better clarity). The black line indicates the duration of the song, which consists of 6 phrases whose durations are indicated by the red lines. The blue lines indicate the duration of a syllable.

3.2.3 Birds perceive sounds differently from human

An average person can hear sound in the range 20Hz to 20000Hz, however, the sensitivity is not the same over all frequencies (R. J. Dooling 2004). In particular, the human's ear is much more sensitive to changes of pitches at a lower frequency than at a higher frequency. For example a difference of 100 Hz between two sounds at 200 and 300 Hz is much easier to notice than the same difference between 10000 and 10100 Hz. This is also obvious from the point of Western musicology: the note one octave above is exactly twice the frequency, for example C1 is 130.8 Hz, C2 is 261.6 Hz, C3 is 523.2 Hz and so on. There are 12 semitones in an octave, and the frequency gap between two adjacent notes are geometrically equal, about 6% ($\sqrt[12]{100\%} \approx 6\%$), but exponential in absolute value. A jump of one octave sounds similar in magnitude no matter the starting node, even though in absolute frequency (Hz) the jump size differs dramatically. This gap gives rise to the notion of *critical bandwidth response* (Zwicker et al. 1957). A critical band is the range of frequencies between a lower and higher cut-off frequency f_1 and f_2 within which the cochlear is unable to discern any frequency difference. The centre frequency of the band f_c characterises how either f_1 and f_2 feels like to the cochlear. This phenomenon comes from the uneven distribution of hair cells in the cochlear (a spiral-shape organ inside the ears of vertebrates). On the basilar membrane, hair cells close to the innermost end (the *apex*) are responsible for detecting low frequencies. And located on the other end, the *base*, are cells sensitive

to high frequency. In humans, the first 40% of the membrane corresponds to frequencies below 1000 Hz, while the remaining 60% is responsible for a much larger range (from 1000Hz to 20000Hz). Similar frequency mappings are found in the cochlears of other animals (Greenwood 1990; Von Békésy and Wever 1960), including chicken (also (Manley et al. 1987)) and barn owl (Köppl et al. 1993).

3.3 Computational models based on sound production and perception

3.3.1 Model of sound production: the source-filter model

Speech and birdsongs are acoustic signals with a high level of complexity. However, the production process is quite simple and could be described using a *source-filter model* (Chiba and Kajiyama 1958). This model was first developed to show how speech is produced in humans. In this model, the sound begins as an excitation signal emitted by periodic opening and closing of the vocal folds (voiced speech) or air flow pushed by the lungs (unvoiced speech). At this stage, the sound has certain fundamental frequency and accompanying harmonics. The sound travels through the throat where certain harmonics of the sound are suppressed or amplified by resonating with the vocal, nasal and pharyngeal tract. The effect of resonance can be seen on a typical spectrogram of human speech. The fundamental frequency in the human voice is typically much lower than the Nyquist, thus there are many harmonics stacking on top of the fundamental. However, some harmonics appear with much higher intensity than the other. These are called *formants*, which characterise the voice of the speaker, and thus are widely used for individual identification. On top of that, occasional hisses and popping sounds might be added to the final product by actions of the tongue, lips and teeth, creates “voiced fricatives” such as [z] and [v]. Mathematically, the model can be written as the convolution of three terms: the excitation source $u[n]$, vocal tract $v[n]$ and radiation $r[n]$:

$$y(n) = u(n) \otimes v(n) \otimes r(n) \quad (3.2)$$

A visualisation of the model is given in figure 3.4. A similar process of sound production

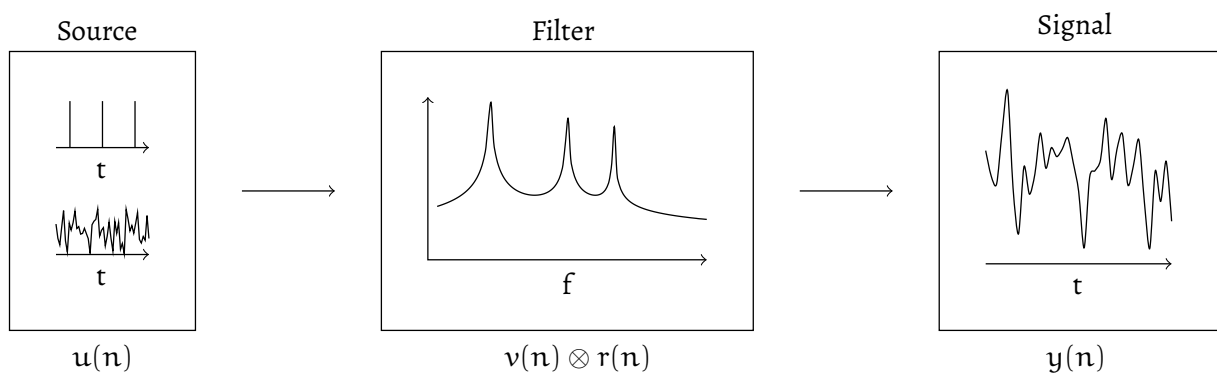


Figure 3.4: The source-filter model for sound production: signal first originates from the excitation source - illustrated as harmonic (the upper plot) and/or random noise (the lower plot) - then goes through an all-pole filtering process and finally exits as the sound we can hear and record. Figure is used with permission, with modifications. Copyright Hyung-Suk Kim. Accessible at ccrma.stanford.edu/~hskim08/lpc/lpc.pdf.

exists in birds. Existing experiments suggests that the oscillation of the *lateral labia* inside the *syrinx* acts as the sound source (Goller and Ole N. Larsen 1997; Ole N Larsen and Goller 1999). Nowicki (Nowicki 1987) showed conclusive evident of the vocal tract resonance in oscine birds by putting them in Heliox (air with nitrogen replaced by helium), which alters the fundamental frequency by 3-5%. Consequently, this breaks the resonance effect, making the harmonics that used to be suppressed displayed clearly on the spectrogram. Many birds can suppress or enhance particular harmonics thanks to the characteristics of their vocal tract (Riede, Suthers, et al. 2006). There is also evidence that the head and beak movements play a role in altering acoustic properties of the final sound (Westneat et al. 1993). The similarity of sound production in birds and humans suggest that using the same source-filter model could be appropriate.

3.3.2 Model of frequency perception: non-linear filter-bank

In section §3.2.3 I pointed out that the hearing mechanism of humans and birds have a similar structure, that is to have more sensitivity towards changes in low pitch sounds. This suggests that a more appropriate representation of sound should also pay more attention at the lower end of the frequency range. However, the commonly used spectrograms generated from Discrete Fourier Transform has constant frequency resolution, for example: if the input signal is sampled at 44100 Hz and the number of FFT points is 512, the width of all frequency bins will be $\frac{F_{\text{Nyquist}}}{n_{\text{bins}}} = \frac{44100}{2} / \frac{512}{2} = 86.13\text{Hz}$. In order to represent sound with different sensitivity level i.e. more sensitive to low frequency, there are two common methods used in acoustics: a) rescaling

the frequency map and b) produce the spectrogram using a non-linear frequency transform, such as warped Fourier transform or wavelet transform. The latter is far less commonly used due to computational cost involved, whereas the first technique is much more efficient. It is just a simple matrix multiplication after acquiring the spectrogram, which is only $O(n \log n)$ complex and readily implemented in most software and even hardware. Rescale is done by mapping the frequency to a psycho-acoustical scale, such as (commonly) Mel scale, Bark scale, Greenwood scale, Human factor scale. These scales use their own unit such as mel, bark, etc instead of Hz. A *mel* on a mel-scale is a unit of pitch that is designed to be perceived by listener as equal distance one from the other. There are more than one formula to convert linear frequency into mels, most of them map 1000Hz to 1000 mel as a reference point. The formula given by (O'Shaughnessy 2000) is arguably the most popular:

$$\text{Mel}(f) = 1125 \times \ln \left(1 + \frac{f}{700} \right) = 2595 \times \log_{10} \left(1 + \frac{f}{700} \right) \quad (3.3)$$

A *Bark*-scale is not logarithmic but also exhibits incremental space between units. Units corresponding to frequency lower than 500 are mostly linearly spaced. It is formulated by (Zwicker 1961)

$$\text{Bark}(f) = 13 \arctan(0.00076f) + 3.5 \arctan((f/7500)^2)$$

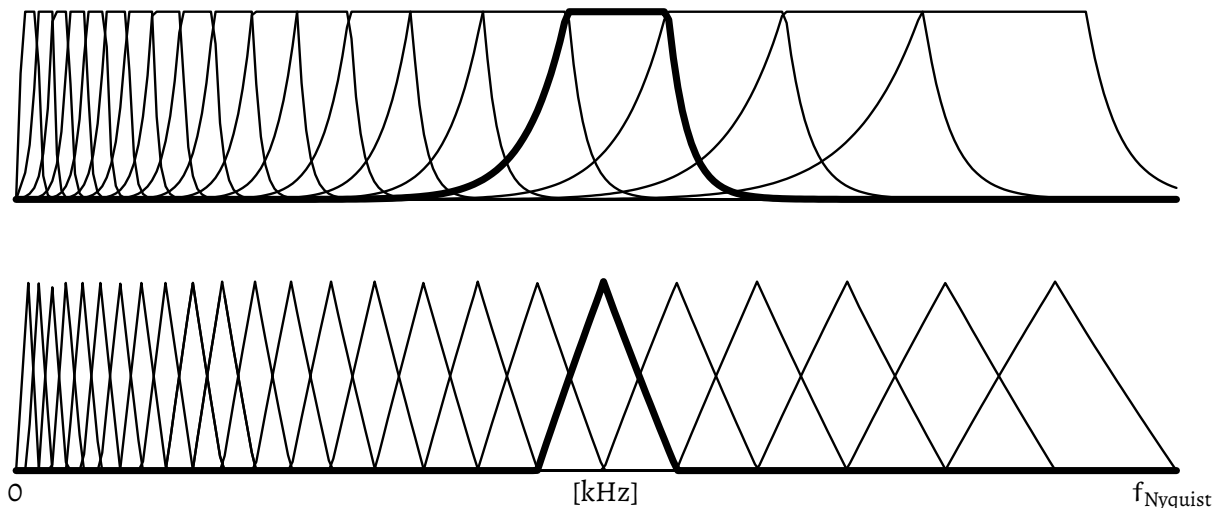


Figure 3.5: Bark (top) and Mel (bottom) filter bank (not normalised). In bold are two random filters illustrating the bandwidth response

A filter bank consists of several triangular overlapping filters with a constant width in mel/bark

scale (exponential width in frequency) is used to mimic the cochlear’s increasing critical bandwidth response. The exact position of the centre frequencies and width of each filter varies between applications. Typically a mel filter bank has 12-20 filters, while a bark filter bank has 24 filters. Figure 3.5 illustrate the difference in bandwidth magnitude of the frequency response of the two scales. Using filter bank also serves as a dimensionality reduction or a lossy compression.

3.3.3 Model of loudness perception: Equal loudness contour

Similar to the sensation of pitch, the loudness of sound is perceived differently at different frequencies. However, the relationship between loudness and frequency is a little more complicated. This relationship is formalised as the function of sound pressure level (SPL) required to maintain the same loudness level, measure by the unit *phon*, over the range of hearing. This function is called the *equal loudness contour*, with five contours at 20,40,60,80 and 100 *phon* illustrated in Figure 3.6. These curves are acquired empirically by averaging the equal-loudness perception of the young participants without significant hearing impairment. The point of reference (1 *phon*) is the SPL of 0dB at 1KHz and its contour is the absolute threshold of hearing. The sudden “dips” (the red segments) of all contours in the range 2-5KHz correspond to the frequency range where human ears are most sensitive to.

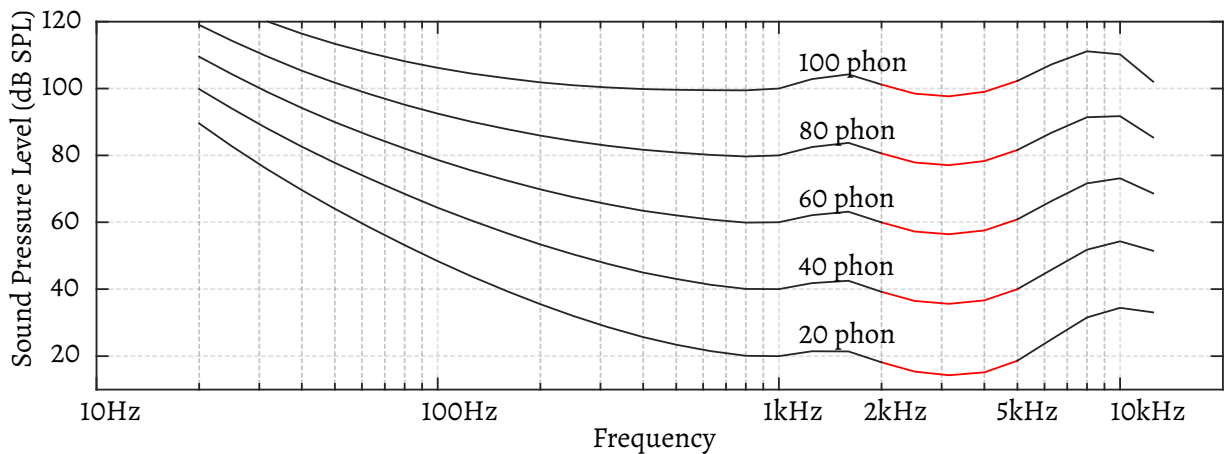


Figure 3.6: Equal loudness contours according to ISO 226:2003

The equal loudness contours are widely used as a preprocessing step, for example in calculating perceptual linear prediction (Hermansky 1990) or specific loudness (Pampalk et al. 2002; Timoney et al. 2004). The contour values are multiplied with the intensity of the sound at the

corresponding frequency to compensate for the final loudness perceived by the human ear. The use of this technique for animals, particularly birds, is not sufficiently investigated. (R. J. Dooling et al. 1978) measured the perception of loudness in four species of birds and found that they in general have similar curve-like response as human's, but later concluded that a generalisation of such equal-loudness contours for all species is not practical (R. Dooling et al. 2005).

3.4 Conclusions

Birds and humans share many acoustics capabilities - hence human can whistle birdsongs and parrots can mimic human voices. However, the mechanisms of sound production are quite different, so as the way sounds are perceived. It is not clear how much these differences can affect the way birdsong analysis is carried out using techniques and tools that are developed for speech. This is one of the reasons *Koe* does not emphasise on automated processes. In the next chapter, I discuss the science and implementation of syllable segmentation. This is one part of the workflow that is well-tuned for manual processing and also has built-in automation.

4.1 Introduction

In this thesis, syllable segmentation involves identifying the starts and ends of song syllables in time. This process is also called “endpoint detection”. Correctly identifying syllable endpoints is not trivial. Several factors that complicate this process are often encountered in bird-song recording but can be a challenge to mitigate. These factors include: static noise coming from the recorder, ambient noise from the environment such as running water, wind or the rustling sound of vegetation, reverberation¹, and overlapping vocalisation of other nearby animals. A substantial proportion of birdsong analysis still relies on manual segmentation even though the authors typically indicate that automatic segmentation is their final aim, for example (K. Adi et al. 2010; Z. Chen and Maher 2006; Fox 2008; Franzen and Gu 2003; Chang Hsing Lee, Han, et al. 2008; Vallejo et al. 2007). Automatic methods for syllable segmentation exist but new development in this field has stalled since 2009. This may be due to the emphasis on species recognition applications in computational birdsong research, which does not require individual syllables to be segmented precisely, or at all. The dominance of segmentation-free approach can be observed in many recent works on species recognition such as in solutions submitted to the annual BirdCLEF competition, see Kahl et al. (2019) for a summary. However, while segmentation may not be necessary for machine-learning based species recognition, it

¹signal energy extending in time past the point the syllable has stopped being produced, often due to the reflection of the original signal when it contacts various surfaces

remains essential for fine-scale analysis of dialects and individual variation. The continued importance of segmentation is evidenced by recent methods papers, which introduce segmentation algorithms (using simple thresholding methods), for example (Barmatz et al. 2019; Zhao et al. 2017, use signal energy), (Bhatia et al. 2019, use mutual information). Nevertheless, a number of algorithms have been developed throughout the years. In this chapter, I first explain how manual syllable segmentation is done in *Koe*. Then the discussion of automated segmentation algorithms follows with an overview of existing methods developed specific to birdsongs, including my experimental neural network models. Finally, these algorithms are evaluated using selected bellbird songs against the ground-truth segmentation that was done manually using the manual process in *Koe*.

4.2 Manual segmentation in *Koe*

In *Koe* there are two types of segmentations: from a long recording (e.g. a few hours) into individual songs (several seconds to a few minutes, depending on the species), and from a song into individual syllables. This section discusses the latter.

Manual segmentation involves the user selecting the start and end points of each syllable on a spectrogram, then committing the selections to the database. Saved selections become available in other program views. The user can adjust playback speed, spectrogram contrast and time-axis zoom. For any songs already segmented into units in other software, start/endpoints can be imported as a csv file. Segmentation data created outside *Koe* can also be uploaded directly to the database in *Unit table* page. The user can freely re-adjust existing segmentation and program views will update dynamically. *Koe* stores only individual songs on the hard drive. The start and end points of each syllable are stored on the database with reference to the song they come from. This way not only are storage requirements minimal, but when a syllable is deleted or has its endpoints changed, the operation will be instantaneous. Figure 4.1 shows this process in action.

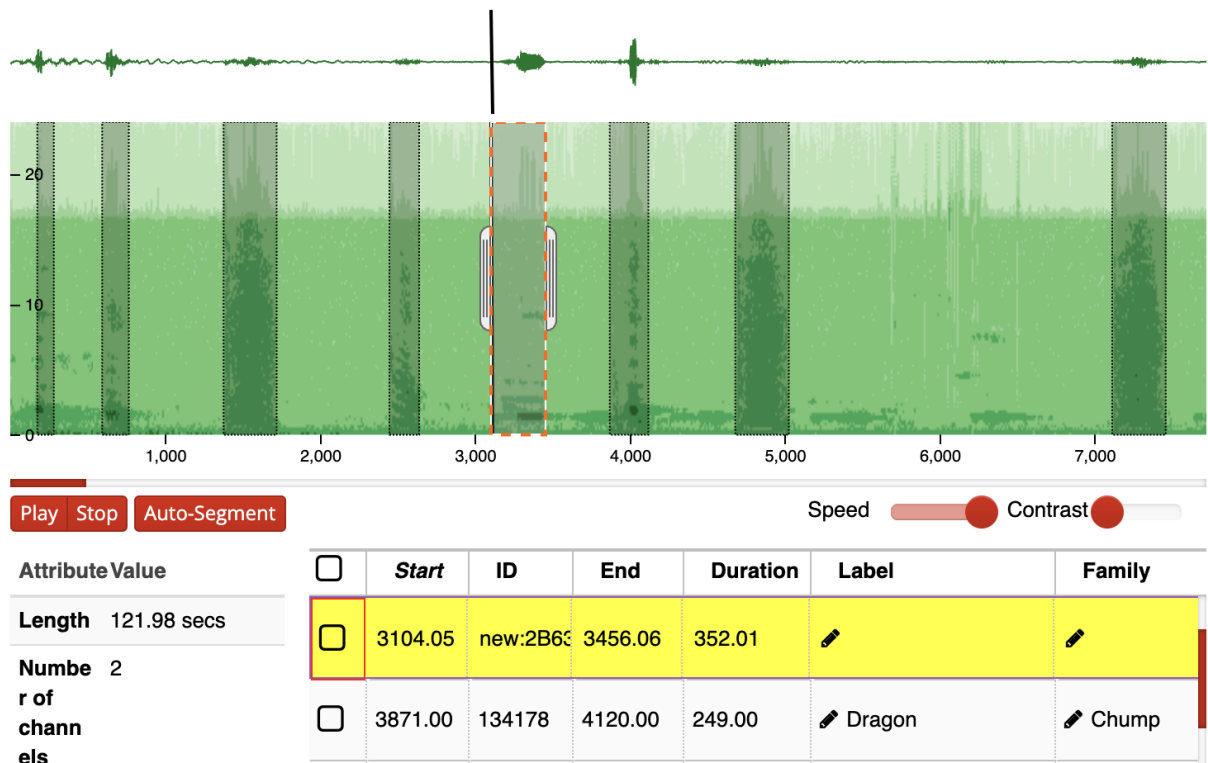


Figure 4.1: Koe's syllable segmentation view showing a song with manually segmented syllables. The highlighted syllable is newly segmented and is in the adjustment process. The user can drag the handles on both sides to change where it begins and ends.

4.3 Related work for automatic segmentation in birdsongs

Syllables in birdsong are typically defined as continuous sounds separated from each other by a brief silent interval, which is normally found empirically, mostly in the range 50 to 300 ms (Ondracek and Hahnloser 2013). This is different from human speech, where a syllable usually refers to a combination of a single vowel sound and the immediately associated consonant sound; successive speech syllables need not be separated by a silent interval (Wilbrecht and Nottebohm 2003). In speech, isolated utterances (words) in clearly articulated speech (each word is followed by an intentional pause) can be defined similarly to birdsong syllables as having sufficiently long silent intervals in between. The RS algorithm (Rabiner and Sambur 1974) is one of the first algorithms developed for the purpose of segmenting articulated words. It finds regions of continuous high value (above certain threshold) on the amplitude envelope to mark the start and end points of the voice parts and then use zero-crossing rate for fine tuning. Adaptation of this method to birdsong can still be found in recent works, using constant threshold such as in (Barmatz et al. 2019; Große Ruse et al. 2016; Jinnai et al. 2012; Lakshminarayanan et al. 2009; Papadopoulos et al. 2015; Qian et al. 2015; Ranjard and Ross 2008), or us-

ing an adaptive threshold such as in (Somervuo, Aki Harma, and Seppo Fagerlund 2006; Wang et al. 2013). This approach produces an acceptable result given the recording was acquired in relatively noise-free environment and all utterances have a similar amplitude.

An algorithm proposed by Harma (Aki Harma 2003) has become widely used to segment birdsong syllables. Similar to the RS algorithm and derivatives, it operates on the signal energy. The main difference is that the energy is analysed in time-frequency domain to find peak energy ridges in the spectrogram. Each syllable is identified as the duration of high energy centered around a local maximum of the energy ridges. The energy threshold is relative to the energy of the local maximum, hence it is adaptive by definition. There are several advantages of this method compared to RS derivatives: 1) RS's signal envelope does not distinguish voiced signal (with energy concentrated at a narrow range of frequency bins) with high energy noise (having energy distributed over a broad range of frequency) 2) an adaptive threshold calculated based on the energy of the local maximum is simple but effective in detecting faint syllables. In addition, Harma's method requires low computational power (see Section §4.6.3). These advantages explains its popularity in birdsong applications. It is used in its original form without modification such as in (Koops et al. 2014; Koops et al. 2015; Chang Hsing Lee, Y. K. Lee, et al. 2006) or with small adjustments of the stopping criteria (Chou and Ko 2011; Chou and Liu 2009; Chou, Liu, and B. Cai 2008). However, due to noise insensitivity, it often fails to detect broadband "noisy" syllables where the energy is not concentrated in a discrete set of frequencies, but is spread cross a frequency range.

An alternative approach to syllable segmentation in time-frequency domain was proposed where image processing methods were employed on the spectrogram to find regions of interest - contiguous blobs of high pixel values. The first algorithm proposed by Fodor (Fodor 2013) follows the following steps: 1) binary threshold spectrogram at 90th percentile; 2) apply blob detection on the acquired binary mask to find regions of interest. These are contiguous blobs of "on" pixels which correspond to concentration of energy. Lasseck (Lasseck 2013) significantly improved the method by employing an adaptive binary thresholding method which he coined "median clipping". Blob detection with median clipping - or the "Lasseck's method" is effective in picking out the signal that stands out in comparison with the noise profile at each particular time point and frequency band. Unlike Harma's method, Lasseck's method can detect noisy

syllables equally well as it does tonal syllables. However because the signals are found as two dimensional subsections of the spectrogram image (localised in both time and frequency), signals with clear harmonic stacks are segmented into several blobs because each harmonic forms a separate region of high intensity. I proposed an algorithm to solve this problem by utilising the nature of harmonics being integer multiple of the fundamental frequency (Fukuzawa, Marsland, et al. 2017). The proposed algorithm is able to merge harmonic parts of a syllable into one and can even separate overlapping signals (e.g. from a different bird), however it requires a high level of signal to noise ratio.

Several machine learning methods for segmenting syllables have been proposed. Similar ideas to Lasseck's method can be found in (Kaewtip, Tan, Alwan, et al. 2013; Neal et al. 2011) where a supervised machine learning method (Random Forest) was used to perform binary classification of spectrogram pixels as "belong to" or "not belong to" a syllable. However, there are several problems with this approach. First, it is far more difficult to create a training dataset for pixel-level classification, as opposed to simple endpoints of the syllables. Second, multi-harmonic syllables can end up being fragmented and the process of putting them back as one syllable isn't trivial, as pointed out in my paper (Fukuzawa, Marsland, et al. 2017). Taking a different approach, Kaewtip, Tan, Taylor, et al. (2015) isolated spectrogram images of syllables as templates and used Dynamic Time Warping to find matching segments on the input spectrogram. The results were then refined using a trained Support Vector Machine. The authors only used 4 templates so it is questionable that this method can work in case of hundreds of different syllable kinds. DTW is also prohibitively expensive to run in such cases.

Machine learning methods for endpoint detections have also been proposed. Ranjard (2009) uses a Hidden Markov Model trained on several acoustic features such as auto-correlation and spectral rolloff to classify each spectrogram frame as "noise" or "syllable". Inspired by this algorithm, my approach in this thesis is to use even more sophisticated machine learning algorithms and less emphasis on manually finding an optimal set of features.

In the coming sections, I detail the implementation of two existing syllable segmentation methods, as well as my own algorithms which utilise complex machine learning architecture. I chose the Harma method - due to its popularity - to represent energy-threshold-based one-dimensional endpoint detection. The Lasseck method is chosen to represent image-processing

based segmentation in two dimension. I implemented an algorithm similar to Ranjard’s method but using a multi-layer neural network and raw spectrogram input to represent a frame-based machine-learning approach. Finally, inspired by the success of sequence-to-sequence auto-encoders in applications of speech and time series (Deng et al. 2010; Serban et al. 2017; Yeh et al. 2019), I evaluated a machine-learning segmentation method using a sequence-to-sequence auto-encoder.

4.4 Procedural algorithms

4.4.1 Endpoint detection in time using energy threshold: Harma method

The original algorithm proposed by Harma (Aki Harma 2003) is as follow:

1. Set the stopping threshold $\Theta = 30\text{dB}$.
2. Convert birdsong audio to spectrogram using FFT. This spectrogram is a two dimensional matrix denoted $S(f, t)$ where f is frequency bin index and t is time frame index.
3. Repeat steps 4-9 until finish
4. Find f_n and t_n , such that $S(f_n, t_n)$ is the maximum value in the spectrogram. This position represents the maximum amplitude position of n^{th} sinusoidal syllable.
5. Store amplitude $A = 20 \times \log_{10}(S(f_n, t_n))$ (dB)
6. if $A < \Theta$ stop
7. Starting from $S(f_n, t_n)$, trace the maximum peak of $S(f, t)$ ($\alpha = 10 \times \log_{10}(\max(S(f, t)))$) for $t > t_0$ and for $t < t_0$ until $\alpha < A - \Theta$
8. Store the syllable endpoints as $[t_b, t_e]$ where t_b is the begin time corresponding to the t found in step 7 where $t < t_0$, and vice versa for the end time t_e .
9. Set $S(f, t_b \rightarrow t_e) = 0$ to delete the area where the current syllable as been found.

Algorithm 1: Harma's method

Data: Spectrogram $S(f, t)$

Result: L : A list of syllable endpoints

$L \leftarrow \emptyset$;

$\Theta \leftarrow 30$ (db);

$A \leftarrow 20 \times \log_{10}(\max(S(f, t)))$;

while $a \geq \theta$ **do**

$t_0 \leftarrow \operatorname{argmax}(S(f, t))$;

$t_b \leftarrow t_0 - 1$;

$t_e \leftarrow t_0 + 1$;

while $\max(S(f, t_b)) \geq A - \Theta$ **do**

$t_b \leftarrow t_b - 1$;

end

while $\max(S(f, t_e)) \geq A - \Theta$ **do**

$t_e \leftarrow t_e + 1$;

end

$L \leftarrow L \cup [t_b, t_e]$;

$S(f, t_b \rightarrow t_e) \leftarrow 0$; $f = 0..Nyquist$;

$A \leftarrow 20 \times \log_{10}(\max(S(f, t)))$;

end

In practice, since only the amplitude of the peak frequency of each frame is used, this algorithm is operating in one dimension by extracting from the spectrogram the dominant frequency ridge and use it as input, see Figure 4.2. This algorithm is often used as is or with slight modifications of the global and local threshold values. One problem with this approach is that decibel is not an absolute scale and can vary depending on the FFT algorithm, so these values are somewhat arbitrary. In my implementation, first the spectrogram is normalised to have power spectral density values in range $[0 - 1]$. The global stopping threshold Θ is set at 0.5 and the local stopping threshold is $\theta = a_{t_0} - 0.2$

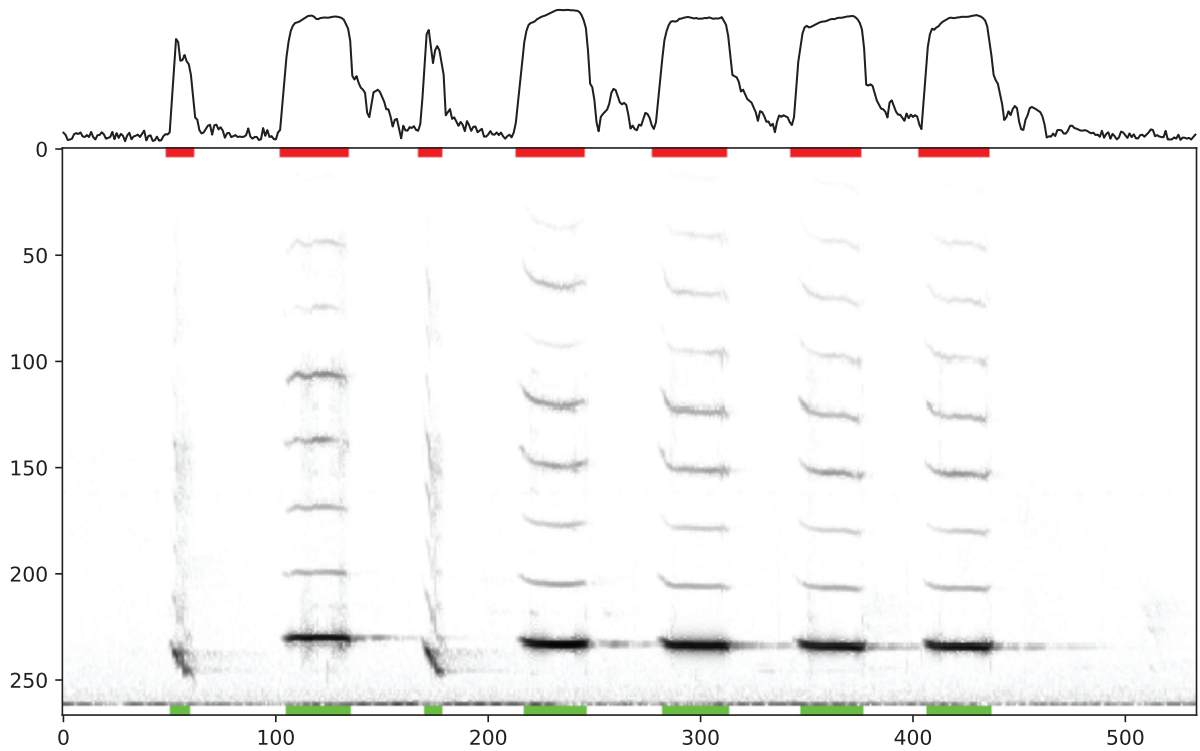


Figure 4.2: Top: the spectral peak value curve that Harma method uses to find syllables. Bottom: spectrogram of a bellbird songs where red segments denote ground-truth syllables and green segments denote results from the algorithm. Although Harma method is written to take two dimensional time-frequency spectrogram as input, in effect it is operating in one dimensional input

4.4.2 Boundary detection in time and frequency using image processing: Lasseck method

The original algorithm proposed by Lasseck, called “*Median Clipping*” is as follows:

1. Convert birdsong audio to spectrogram.
2. Remove spectrogram rows representing the relevant frequency below 170 or above 10000 Hz.
3. Gaussian smoothing
4. Binary thresholding according to Equation 4.1

$$S_{r,c} > 3 \times \max(\text{median}(S_r), \text{median}(S_c)) \quad (4.1)$$

Where $S_{r,c}$ denotes a pixel value in the spectrogram at row r and column c that is being thresholded, S_r is a row of spectrogram pixel values at row r , S_c is a column of spectrogram pixel values at column c

5. Morphological removal of spurious pixels and small objects
6. Blob detection (after filling holes)

The novelty of Median Clipping is that it uses the median value of a row or a column as a pseudo-adaptive threshold for each pixel, so that the algorithm can be very fast yet effective in picking out the signal that stands out in comparison with the noise profile at each particular time point and frequency band. The blobs segmented out using this method have been used directly as templates to perform template matching on audio recordings with unknown species to detect what species are present in the recordings, which proved to be quite a successful approach, winning a number of competitions of recognising bird species in audio recordings (MLSP, NISP4B, BirdCLEF 2013, 2014, 2015).

This method works well at segmenting out pockets of high energy in the time-frequency space (generally syllables), however, it is also prone to segmenting out blobs that correspond to nothing but noise and breaking whole syllables into multiple blobs. It also does not deal with harmonic syllables and segments each harmonic separately, frequently into smaller and smaller blobs for higher harmonics as these do not carry as much power as the fundamental. Figure 4.3 (bottom figure) shows one typical case of this scenario. We found this method good at dealing with narrow-band noise that frequently presents in the recordings, but does not segment syllables correctly, either breaking up one syllable into multiple blobs or linking multiple syllables into a single blob. This latter result is mostly due to reverberations present in the recordings that blur the syllables out along the time axis at the end of vocalisation and effectively creating an overlap with the next syllable.

In order to produce comparable results with Harma and my algorithms which segment syllables in time only, it is necessary to detect the actual syllable endpoints from separate harmonic blobs. In my implementation of Lasseck's method, a simple voting scheme applies to merge harmonic blobs together. Details as following:

1. From detected signal blobs, construct a binary mask of the spectrogram (having F rows and T columns) with blob content given the value of 1 and background gets the value of 0. Figure 4.3 (middle figure) shows an example of this binary mask.
2. Merge blobs that are overlapping in time (e.g. harmonics of the same syllable) by summing the columns. The result is a one dimensional vector with length T . The top figure

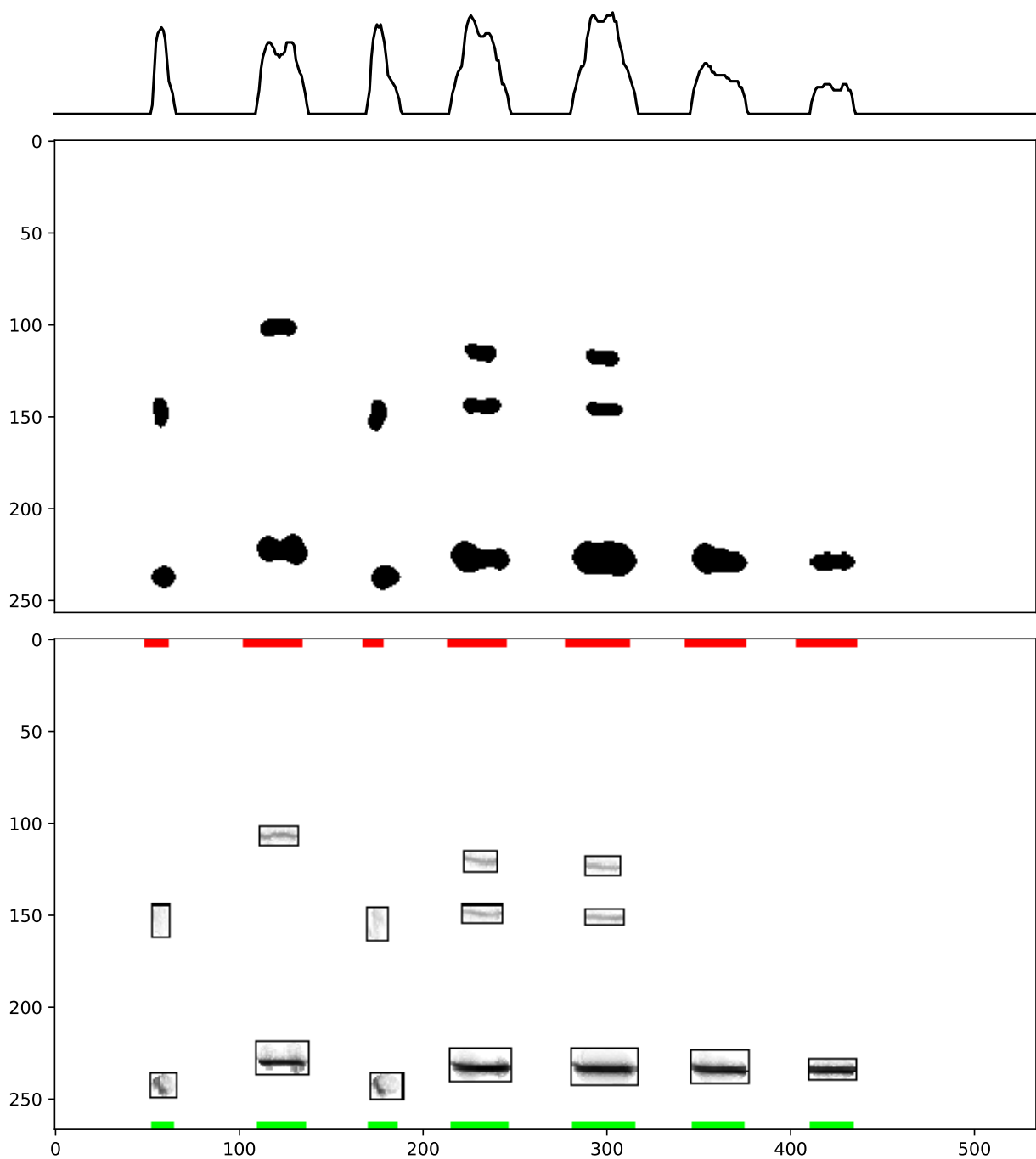


Figure 4.3: Top: accumulated binary mask curve. Middle: binary mask after median clipping. Bottom: signal blobs found using blob detection where red segments denote ground-truth syllables and green segments denote results from the algorithm.

of Fig. 4.3 is the result of summing the binary mask given in the middle figure.

3. The syllables can now be isolated by finding the points where column sums transition from zero to non zero and vice verse.

4.5 Heuristic approaches

Recent development in species recognition is heavily machine-learning based, however machine learning methods specifically for segmenting birdsong syllables (animal vocalisation syllables in general) are rare. Pixel-level supervised segmentation can be found in Neal et al. (2011), Kaewtip, Tan, Alwan, et al. (2013) and frame-level segmentation using machine learning is documented in Ranjard (2009). Overall, the classifiers of these machine learning methods have simple structure. I implemented a generic feed-forward neural network and achieved high accuracy. Finally to tackle the problem of variable length, a more complicated recurrent neural network was also implemented and produced statistically better accuracy. Both neural networks methods achieve higher accuracy than Harma and Lasseck methods, however they are slower. The detailed performance comparisons between these methods are provided in Section §4.6.

The neural network approach consists of two phases: the training phase and the application phase. In the training phase, the model is given a set of input and correct output. The input is typically a one-dimensional array, while the output can be a singular value or an array of values.

The model consists of several connected layers of randomly initialised weights that simply multiplies with the input and the results are used as input for the next layer. The initial output of the model is completely random. The correct output is then used to give feedback to the layers in a process called gradient descent, and the process repeats, each time the weights are slightly adjusted and the output slightly approaches the correct output.

The basic structure of both neural network based methods are as following: the input is k -consecutive spectrogram frames stacked into one dimension. The output is a k -dimensional binary indicator where 1 corresponds to syllable-positive frames and 0 to syllable-negative frames. The value of k is closely related to the average gap between syllables: if it is too small, the neural network is prone to random short bursts of noise being recognised as positive. Including more frames in the input provides context to the classifier and should increase accuracy rates, but risks overlooking small gaps that nonetheless genuinely separate syllables. Section §4.5.1 explains in details how such neural network can be implemented given a reasonable value of k , and section §4.5.2 shows how a Recurrent Neural Network (RNN) structure can overcome this issue by not fixing the value of k altogether.

4.5.1 Feed-forward Neural Network with fixed size input

As the each layer of a neural network has fixed number of neurons, a traditional feed-forward neural network (also called Multi-Layer Perceptron) requires that the input has fixed size, whereas in reality syllables vary in length. Thus, the spectrogram needs to be cascaded into equal-sized chunks by concatenating k consecutive frames into one input vector in the following manner:

1. At the first frame f_1 , place a window of k frames to encompass all frames in the range f_1 to f_k .
2. Stack all frames in this window into a one-dimensional vector.
3. The expected outcome is 1 if the window contains a syllable frame, or 0 if it contains only non-syllable frames.
4. To train, fetch this vector and the expected outcome to the neural network.
5. To predict, fetch this vector to the trained neural network and collect the outcome, which represents the probability that the input contains a syllable frame.
6. Move to the next frame f_2 and repeat until frame f_{l-k+1} where l is the total number of frames in the original audio segment.

Figure 4.4 illustrate this process where $k = 2$. There is an inherent issue with this neural network model, which can be observed quite clearly from the illustration. The cascaded vectors are disconnected from each other as they are independently trained. The order in which they are used to train the model does not matter. Whereas, to determine whether a frame is a syllable frame or not depends significantly on the adjacent frames. A high value of k lessens this disconnectedness by providing more context, but significantly increases the size of the network. In the next section, Section §4.5.2 I discuss how a recurrent neural network can overcome this issue.

4.5.2 Recurrent Neural Network with variable size input

An RNN is structurally similar to an MLP, except that the neural values of the hidden layer are circled back to itself in the next training iteration, hence "recurrent". The hidden layer at each training iteration has information of all the previous samples that it has seen, thus the order

in which the training takes place reflects the temporal arrangement of the input frames, provided that they are fetched into the network in the same order they appear. This way, the input of a recurrent neural network is effectively all the previous frames combined with no size limitation, however it should be obvious that historical frames have diminishing contribution to the outcome of the current iteration. This is intuitively similar to how the perception of sound works in humans and animals.

One caveat of this approach is that there is nothing that comes prior to the first frame, for which the hidden layer values can be calculated. Thus a special frame, called **START** token, needs to be predefined, and will be the first frame to go into the neural network. The **START** token is a vector of size equal to that of the hidden layer, with values chosen such that it cannot be mistaken for a normal frame. For example, if the calculated neural values of the hidden layer must be positive, or must be within range $[0 - 1]$, then a **START** token with all negative values will be sufficiently distinctive. With the correct settings, the RNN can then be trained and used in the following manner, which is also illustrated in Figure 4.5

1. Initialise the neural network, typically by randomising the neurons' values.
2. Define the **START** token.
3. The expected output of the network at each frame f_i is the prediction that the next frame is a syllable frame (1) or not (0)
4. At the first frame f_1 , fetch the frame values to the input layer. Fetch the outcome of both the input layer and the **START** token to the hidden layer. Keep a copy of the outcome of the hidden layer.
5. Do the same for subsequent frames f_i , except that instead of the **START** token, use the values of the hidden layer at the previous iteration.
6. To train, use gradient descent to back propagate the error ($|\text{Expected output} - \text{actual output}|$) to the previous layer and adjust their neuron weights.
7. To predict, collect the outcome, which represents the probability that the input frame is a syllable frame.

4.6 Data and evaluation

4.6.1 Data

Ground truth of the segmentation was syllable endpoints extracted from manual annotation using the program *Luscinia*, and then refined carefully by hand in *Koe* by several biologists. Only audio data from fully segmented audio was included. These are songs or calls of other birds in the background that the biologists wished not to include to avoid contaminating data for sequence analysis later.

Harma and Lasseck methods require no training. To train a machine learning algorithm, 23 audio files with high quality segmentation were selected. These files are relatively short, and contains no additional vocalisations than the syllables that had been segmented by hand. These files are listed in Table 4.1.

4.6.2 Evaluation

The segmentation of syllables from bird songs is similar to that of phonemes/isolated words in running human speech. Thus, we can borrow some evaluation methods for speech segmentation to evaluate syllable segmentation results. Manual segmentation by trained experts is used as the baseline for the evaluation. A syllable is defined as the segment within two boundaries in time. A boundary is deemed “correct” if it falls within the vicinity of one baseline boundary. Several statistical measures can be derived directly from the comparison between baseline boundaries and segmented boundaries, including N_{gt} : the number of boundaries in the ground-truth; N_f : the number of detected boundaries and N_{hit} : the number of boundaries correctly detected. A 50% tolerance applies to judge if the boundaries are correct, i.e. if the correct segment and predicted segment overlaps more than 50% that counts as a hit. If one correct segment overlaps with two or more predicted segments more than 50% both, that counts as one hit. Many scoring schemes have been proposed to evaluate the segmentation result. Some

Table 4.1: Files used to train MLP and RNN. These files come from *Koe's* database *Bellbird_TMI*. Syllable endpoints are manually marked by Dr. Wesley Webb. Columns $l(f)$ shows the total length of the file in millisecond. Columns n_s shows the number of syllables. Column $\sum_1^{n_s} l(s)$ shows to cumulative lengths of all syllables.

File name	$l(f)$	n_s	$\sum_1^{n_s} l(s)$
TMI_2014_03_20_MMR126_01_FVG.	5329	13	2316
TMI_2015_03_24_MMR291_02_F.G.HlB-BM	6651	24	3022
TMI_2015_10_01_MMR049_05_F.EX.HY-WM	6278	31	3262
TMI_2014_02_25_MMR067_01_M.OK.BPu-WM	5047	17	2943
TMI_2013_03_11_MMR067_01_M.OK.HPu-OM	3495	10	1495
TMI_2013_03_12_MMR080_02_M.OK.HPu-OM	4639	15	1496
TMI_2015_09_16_MMR021_03_M.OK.HPu-OM	5808	6	1526
TMI_2015_10_19_MMR073_01_M.G.HPu-OM	5465	6	1337
TMI_2014_03_19_MMR085_01_M.OK.OW-GyM	5213	6	1619
TMI_2015_10_19_MMR074_04_M.OK.RBr-lBM	6610	7	1817
TMI_2013_04_09_MMR137_01_M.OK.YBk-M	3205	13	1477
TMI_2013_05_01_MMR106_01_M.G.G-OM	5258	5	1286
TMI_2013_04_01_MMR046_01_F.G.YB-GM.(A)	5988	18	3226
TMI_2014_12_19_MMR087_01_M.G.HBk-dGM.(A)	7011	8	2142
TMI_2014_12_31_MMR157_05_M.G.HB-WM.(A)	5350	6	1592
TMI_2015_02_15_MMR058_01_M.G.RBr-lBM.(A)	7011	6	1744
TMI_2015_10_13_CHJ025_03_M.G.RBr-lBM.(A)	7000	16	2934
TMI_2015_01_04_MMR171_01_F.OK.HR-RM.(A)	5006	25	2127
TMI_2015_12_14_MMR159_03_F.G.lBO-BM.(A)	6072	15	840
TMI_2014_12_30_MMR137_01_F.OK.RBr-BM.(A)	6994	20	923
TMI_2014_12_21_MMR107_01_F.OK.O-lG(Obr-lGM)	6358	8	369
TMI_2014_11_02_MMR026_02_F.OK.PpB-lBM.(HY)	5121	18	1370
TMI_2014_11_02_MMR026_03_F.OK.PpB-lBM.(HY)	5041	20	1510
Total	129950	313	42373

of which are listed below:

$$\text{Hit rate (HR)} = N_{\text{hit}}/N_{\text{gt}}$$

$$\text{Precision (P)} = N_{\text{hit}}/N_f$$

$$\text{Recall (R)} = N_{\text{hit}}/N_{\text{gt}}$$

$$\text{F1 score (F)} = 2PR/(P + R)$$

Where the closer the F1 score is to 1 and R-value is to 0, the more similar the segmentation result is to the ground-truth.

4.6.3 Results

Whisker plots in Figures 4.6a, 4.6b, 4.6c show the average and two standard deviations of F1, Precision and Recall scores respectively of 5 different segmentation methods. MLP and RNN are clearly significantly better than Harma, Lasseck and median-clipped Harma. T-test also shows that RNN has higher F1 score and precision, however both have statistically similar recall rates. The reason for this slightly better performance of RNN can be explained by RNN being inherently aware of the temporal relationship between frames, whereas MLP treats all frames with equal importance.

As stated in Chapter 1, *Koe* is a tool for high-precision analysis, the speed and computational complexity are of lesser importance. However, future development of *Koe* will include making automatic segmentation accessible to the users, where the algorithms will be running in real time and thus speed will become more important. In order to assess the speed performance of these algorithm, I recorded their elapsed time needed to run over each audio file and calculated the average time it took to process one second of the input file. We can see that Harma method was extremely fast as expected (on average 0.00016 second to process one second of audio), and recurrent neural network was the slowest. Lasseck method and Lasseck's enhanced Harma method were on par at second slowest. RNN is unsurprisingly the slowest because each frame must be processed in sequence. MLP on the other hand, is quite fast despite having similar structure and load time requirements as RNN. This is because frames are processed independently in an MLP and thus with batch processing, they can be processed simultaneously.

4.7 Conclusion

The neural network approach appeared to be more effective for the segmentation of bird songs than other procedural methods. In particular, recurrent neural networks appeared to outperform traditional MLP and this can be attributed to their ability to model the dynamics of the temporal structures of the signal. However, the RNN approach is heavily penalised in terms of speed due to its complexity, while MLP is significantly faster and only underperforms RNN slightly. Harma segmentation is unmatched in speed, being several orders of magnitude faster

than all other methods, in exchange for significantly worse results. However, all of them were reasonably fast with the slowest (RNN) still clocking at 70 ms per second of audio file. Where computational power is severely restricted, for example, in remote/handheld devices, Harma might still be useful. In most computers nowadays however, the neural network is obviously a wiser choice.

Comparison aside, it is important to note that none of these algorithms outperforms manual segmentation, or even come close to that. For high precision analyses, which is the aim of *Koe*, manual segmentation is still irreplaceable. The precision of segmentation is important for the next step of the workflow: feature extraction, as many features are end-point sensitive. In Chapter 5 that follows, I present features that are extractable using *Koe*, including those most commonly used in acoustics as well as useful features that were previously only implemented by proprietary software until the introduction of *Koe*.

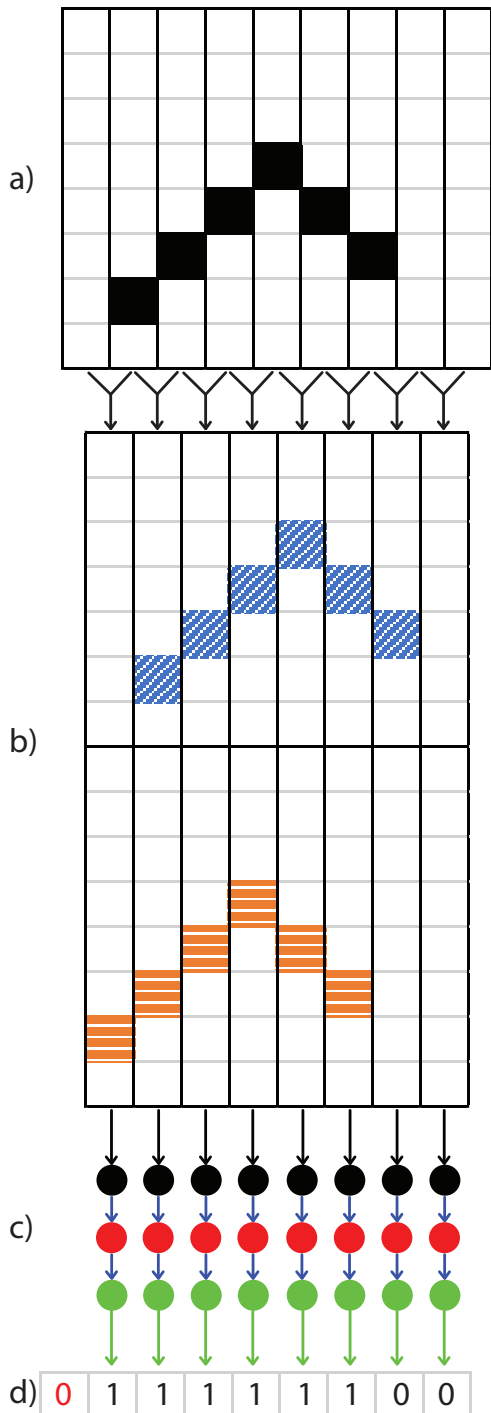


Figure 4.4: A simplified process of syllable end-point detection using multi-layer perceptron with fixed size input. a) Spectrographic representation of a syllable, with silent gaps at both ends.

b) Concatenate k frames into one vector. Here, $k = 2$.
 c) Each vector is fed into the neural network. The neural network is visualised repeatedly for each input, but keep in mind that there's only one neural network. For simplicity, the neural network has three layers: ● → ● → ● circles represent the input, hidden and output layer, respectively.
 d) The output of each vector indicates whether the original input is syllable-positive (1) or syllable-negative (0). Notice that the number of output is $k - 1$ less than the number of original frames, due to concatenation. The output of the first $k - 1$ frames is defined typically to be 0, as a bird song typically starts with a brief silence before the first syllable starts. Here, the number 0 in red indicates that the first frame in this example is defined to be syllable-negative. Syllables endpoints coincide with the begin and end of a string of positive output.

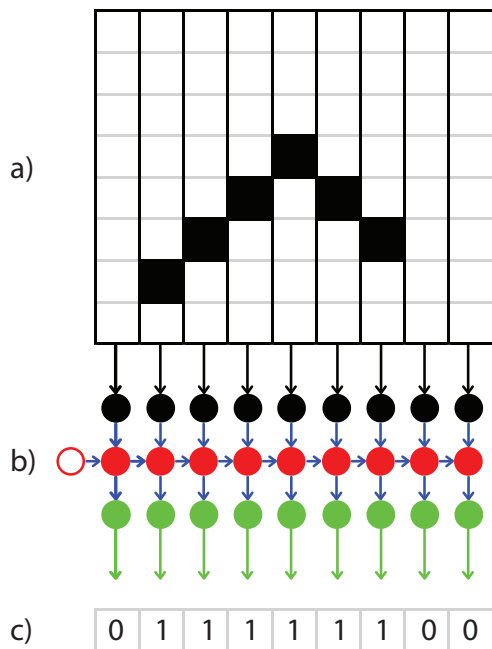
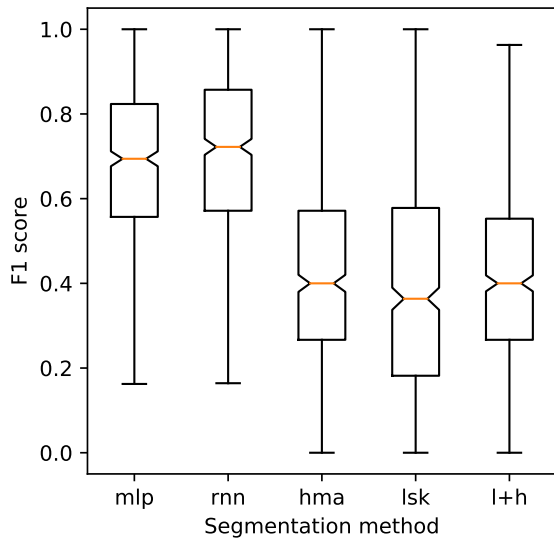
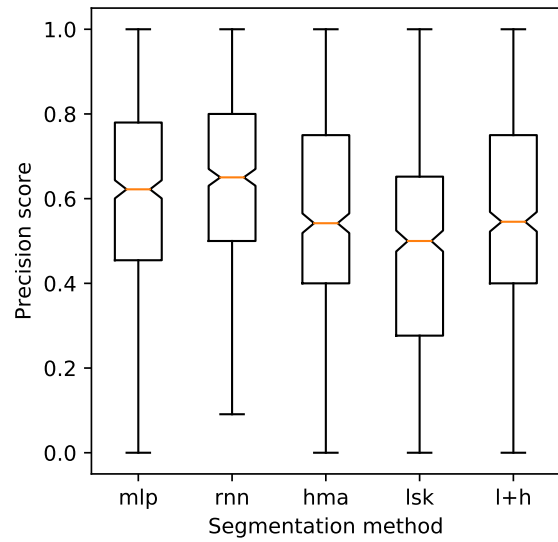


Figure 4.5: A simplified process of syllable end-point detection using recurrent neural network with variable size input.

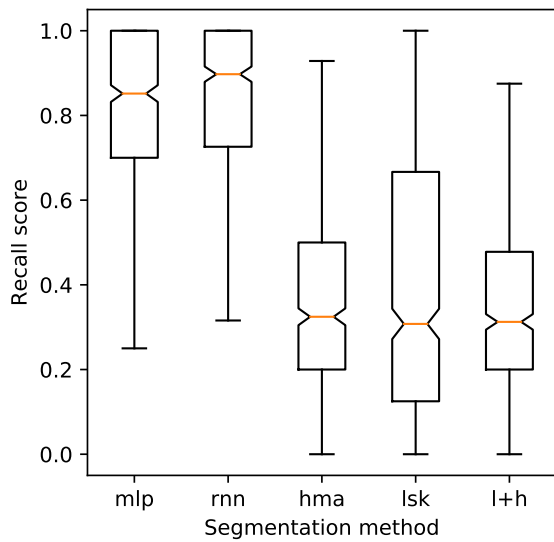
- a) Spectrographic representation of a syllable, with silent gaps at both ends.
- b) Each frame is fetched into the neural network. The neural network is visualised repeatedly for each input, but keep in mind that there's only one neural network. For simplicity, the neural network has three layers: ●→●→● circles represent the input, hidden and output layer, respectively. However, the values of the hidden layer are also used as the input to itself in the next frame. For the first frame, a predefined **START** token ○ is used in place of the previous hidden layer values.
- c) The output at index i indicates whether the next frame f_{i+1} is a syllable frame (1) or not (0).
Syllables endpoints coincide with the begin and end of a string of positive output.



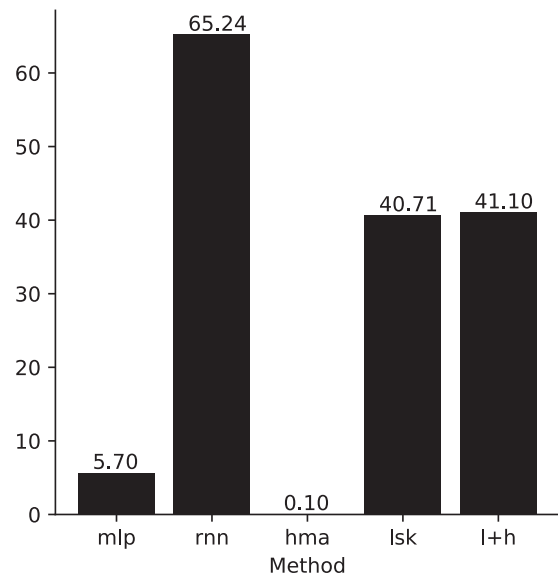
(a) F1 scores



(b) Precision



(c) Recall



(d) Average time elapsed

Figure 4.6: Mean and two standard deviations of specific scores of different segmentation algorithms

5.1 Introduction

Birdsong audio signals are stored on a digital device as a series of air pressure values sampled at equal intervals which can then be used to reproduce the original sound. This series of air pressure values in its raw form is often ill-suited to study complex vocalisation due to its inability to present properties of sounds in a meaningful way. In order to be used in any computer algorithm, birdsong syllables must first be represented in a feature space that facilitates mathematics operations. An ideal feature representation does not only present sounds in a lower dimensional space but also captures only relevant acoustic information while being invariant to background noise.

Figure 5.1 illustrates the general process of feature extraction. The majority of acoustic features are frame-based: a fixed-length window is progressively slid across the original signal, capturing a segment of sound that is short enough to assume that it is stationary; then one value (scalar or vector) is calculated from that segment. A long signal has more frames than a short signal, making their feature vectors have different lengths. Birdsong syllables almost always have different durations, so in general their feature vectors have variable length. This complicates the comparison between syllables as many algorithms (for example: Euclidean and other L_p metrics, most neural network architectures, most dimensionality reduction methods) require the input to be of the same size. It is therefore necessary to standardise feature length before they can be used in further analysis. Common standardisation methods include:

extracting statistics such as minimum, maximum, median, standard deviation; resampling sequences; and using edit distance algorithms such as DTW to calculate pair-wise similarity values.

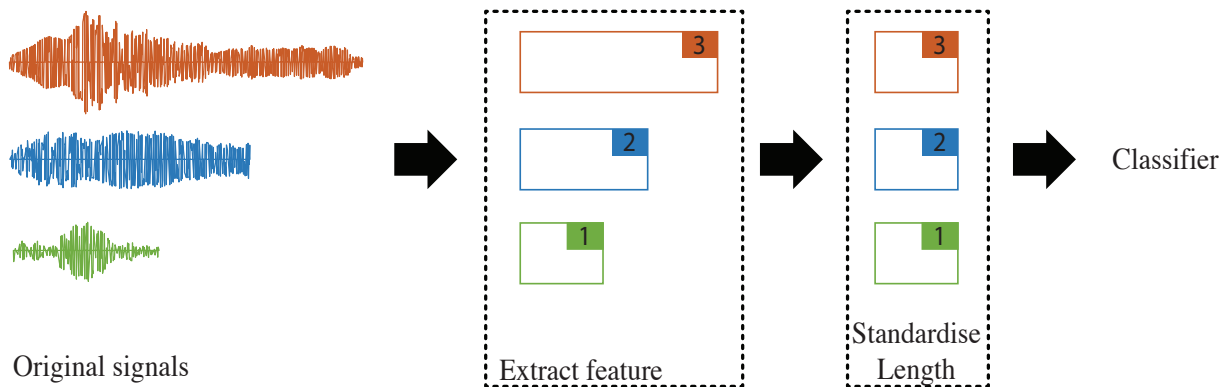


Figure 5.1: Workflow of various acoustic feature extraction from bird song

Acoustic features are very diverse, and in many studies new features catering to specific species and environment are proposed spontaneously. In this chapter, I first present an overview of features that are frequently used in birdsong analysis. This is followed by standardisation methods. Finally, I demonstrate how feature extraction and standardisation are performed and implemented in *Koe*.

5.2 Acoustic features

Most acoustics features used in birdsong analysis are borrowed from speech processing. A large number of acoustic features have been proposed over time: early features were proposed to reflect human-perception of sound such as “brightness”, “highness”, “noisiness”, “flatness”; later more complex and abstract features such as MFCC and Wavelet coefficients became more mainstream.

Mitrovic et. al. (Mitrović et al. 2010) studying content-based audio retrieval have come up with a novel taxonomy of acoustic features based on the origin and methods to extract them. By organising acoustic features into meaningful groups, their taxonomy facilitates the selection of features for particular tasks. Mitrovic et. al.’s taxonomy includes several domains, in which time (temporal), frequency (spectral) and quefreny (cepstral) are most used in speech processing and animal acoustics. Other domains such as modulation frequency, phase and eigen domains contain features that are mostly useful for music analysis and are thus excluded

from my research. I arranged their taxonomy into two groups: descriptive and abstract.

5.2.1 Descriptive features

Descriptive features include time domain and the perceptual subdomain of the frequency domain. The majority of these features are one dimensional and named to reflect human perception of sound. These features are listed in Figure 5.2.

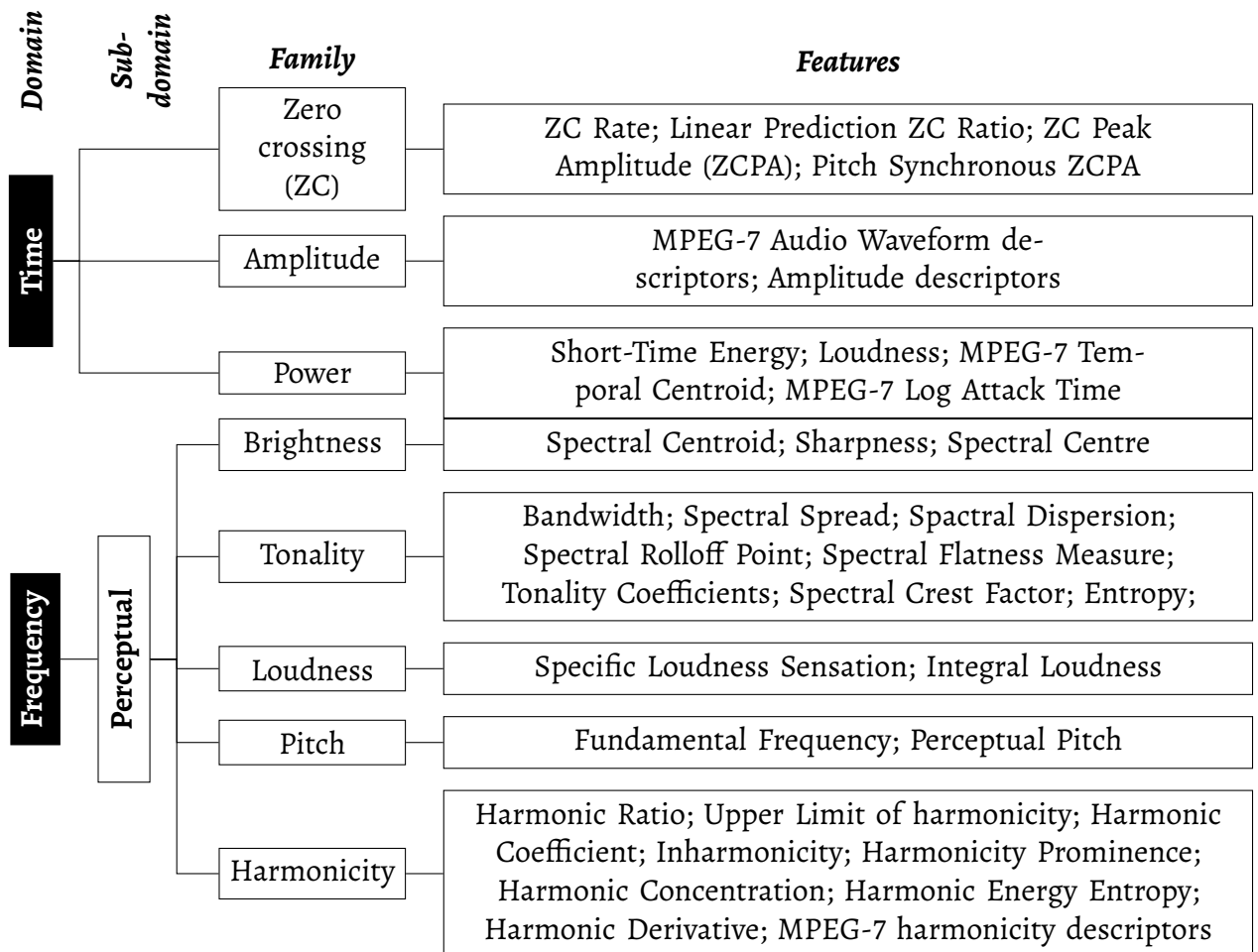


Figure 5.2: Taxonomy of descriptive acoustic features.

5.2.1.1 Time domain features

The time domain represents changes in air pressure when the sound reaches a microphone, which is measured by voltage value. Features in this category are necessarily extracted directly from the oscillogram without any transformation. Thus, they tend to have excellent time resolution and low computational cost. Common time domain features include:

- **Zero-Crossing features:** the main feature is Zero Crossing Rate (ZCR) which measures how many times the voltage values change sign, formally defined as the number of zero crossings within one second, and is an efficient way to approximate the dominant frequency (Kedem 1986). Related features include *Linear Prediction Zero Crossing Ratio* (LPZCR), *Zero Crossing Peak Amplitude* (ZCPA) was proposed to extract frequency information, including intensities from the zero crossings without Fourier transform.
- **Amplitude:** is the absolute value of the signal's waveform
- **Power:** includes *Short-time energy* (STE) defined as the mean square of the amplitude over a period of time and *Volume* or *Loudness* defined as the root mean square of the amplitude over a period of time. *Log Attack Time* (LAT) is the logarithm of time duration from the beginning of the sound to the point where the envelope reaches its first maximum

5.2.1.2 Frequency domain perceptual features

Frequency domain has the largest group of acoustic features, which can be further divided into two subdomains: physical (discussed later in Section §5.2.2) and the perceptual. Perceptual features are derived from human's perceptual of sound, thus they are often expressed in language terms that are intuitive to understand. All features in this group are extracted from the magnitude part of the spectrum.

- **Brightness** refers to the tilt of power towards higher frequencies, i.e. a sound becomes brighter when more energy is concentrated in high frequencies. Brightness features characterise the distribution of power over the frequency spectrum. These features include: *Spectral Centroid* is the centre of gravity of the spectrum, calculated as the weighted mean of the frequencies using their normalised magnitude as weights; *Spectral Centre* is the frequency where half of the energy in the magnitude spectrum is above and half is below. It is one instance of *Spectral Rolloff Point*, which is defined as the $N\%$ percentile of the power spectrum. In practice, the value of N is usually at the higher end (85, 95, etc).
- **Tonality:** is a measure of relative pitch strength of the perceived pitch of a sound. Tonality features include: *Spectral Flatness Measure* (SFM), also known as *Wiener entropy* is a measure of the noisiness of the spectrum, where higher values correspond to a more uniform

distribution of power over all frequencies; *Spectral Crest Factor* (SCF) bears the opposite meaning to SFM, in other words it measures how peaky the spectrum is. A noisy spectrum has a low value of SCF compare to a clear tonal spectrum. It is calculated as the ratio of the peak over the mean frequency value. Similar to Wiener entropy are *Shannon entropy*, *Renyi entropy*. *Bandwidth* (BW), a.k.a. *spectral spread* is usually defined as the magnitude-weighted average of the differences between the spectral components and the spectral centroid.

- **Loudness:** measures the human's sensation of the magnitude of the sound when it reaches the ears. Loudness features are usually computed with respect to perceptual scale. *Specific Loudness*, proposed by Zwicker (Zwicker et al. 1957), is defined as the loudness caused by excitation of the auditory system over one single perceptual scale unit z (which can be Mel, Bark, rectangular, etc.) *Total Loudness* of a spectrum is the sum of all *Specific Loudness* over the perceptual scale.
- **Pitch:** is widely used in many areas of acoustic analysis and in different contexts can refer to either the *Fundamental Frequency* or the perceived sensation of the auditory system. Pitch determination is not a trivial task. A number of methods have been proposed for pitch extraction. A comprehensive review of these methods is done by Hess (W. Hess 1983; W. J. Hess 1992). In this study, I use the term *Fundamental Frequency* instead of pitch to avoid ambiguity. *Fundamental Frequency* (FF) is the lowest frequency of a harmonic series. It concurs with the rate of oscillation of the sound source. The FF is the same as the dominant frequency in case of pure-tone signals, which makes it relatively straightforward to determine. However, this is not the case with speech and many bird's vocalisations, which produce complex sound (having harmonic structure). The fundamental frequency of a harmonic series can be determined as either the lowest harmonic or more reliably (in the case of missing fundamental) by finding the largest common denominator of the harmonics.
- **Harmonicity:** relates to the proportion of harmonic components in a signal. *Harmonic Ratio* (HR) is the ratio of harmonic power to total power. The formula to calculate harmonicity is given in the MPEG-7 specification (Kim et al. 2006, Pg. 33)

5.2.1.3 Use of descriptive features in related work

Use of frequency features in the "descriptive" class is very limited in the analysis of birdsong. Fundamental frequency is one exception that can be found being used on its own in early studies, for example: (Z. Chen and Maher 2006; Franzen and Gu 2003; Ito et al. 1996; Jancovic et al. 2013). Often several descriptive features are used together; for example in analysis of song development (Derégnaucourt et al. 2005; O. Tchernichovski et al. 2004) and comparing calls of brood parasite (Ranjard, M. G. Anderson, et al. 2010). However, manually selecting a set of features is *ad hoc* in nature and should be discouraged. A slightly better approach is to include a large number of descriptive features and hope that a machine learning model will learn to assign more weights to more relevant features. Studies that used only descriptive features in this way are rare, examples are (Acevedo et al. 2009; Myron C. Baker and Logue 2003; Z. Chen and Maher 2006). Often when feature selection is involved, descriptive features are used in addition to abstract features such as MFCCs.

5.2.2 Abstract features

The abstract feature group, which includes cepstral domain and the physical subdomain of the frequency domain, are features that are only meaningful mathematically i.e. they don't map onto human perception of sound. These features are listed in Figure 5.3.

5.2.2.1 Frequency domain physical features

Physical features are mechanical properties pertaining to audio signal in general and not to human perception, therefore it is often not possible to assign semantic meanings to these features. Features in this group include:

1. **Multi-resolution features** are coefficients of Wavelet transform and related transformations. These coefficients can also be used to construct a scalogram - a multiresolution spectrogram. Existing features extracted from a spectrogram can also be modified to be extracted from a scalogram. In addition, there are unique features of wavelet transform coefficients such as *Doubechies Wavelet Coefficients Histogram* (DWCH) (Li et al. 2003).
2. **Spectral features** are calculated from the spectrogram, they include: *Subband Energy Ra-*

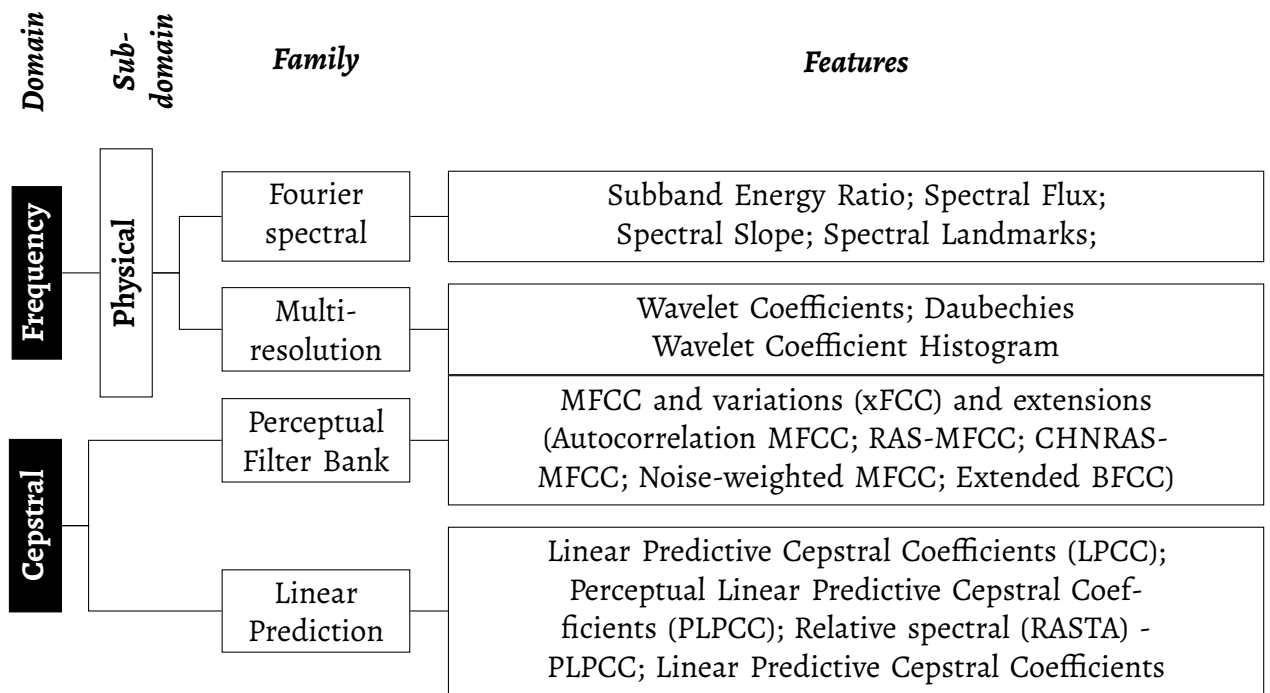


Figure 5.3: Taxonomy of abstract acoustic features.

tio (SER) or *Spectral Subband Energy Ratio* (SSER), defined as the ratio of the sum of energy in one subband over the total energy of the whole spectrum; *Spectral Flux* (SF) is a measure of the changes in the shape of the spectrum over time. SF is computed as the Euclidean (L^2) distance of two consecutive spectral frames with normalised PSD; *Spectral slope*: is the slope of a linear regression $x(f) = \text{slope} \times f + \text{intercept}$ where $x(f)$ is the magnitude function and f is the frequency value.

5.2.2.2 Cepstral domain features

Cepstral analysis operating on the cepstral domain is a technique developed by Bogert et al. (1963) to distinguish between earthquakes and underground nuclear explosions. This technique originated from the observation that the echoes of an explosion have clear harmonic structure while seismic activities do not. This makes it ideally suited to the study of natural sounds that also have harmonic structures. One of the most frequent applications of the cepstral domain is in calculating cepstral coefficients (CCs). CCs are simply the first n values of a cepstral series, which is the result of Fourier transforming the power spectrum. Most used cepstral coefficients include Mel Frequency Cepstral Coefficients (MFCC) and Linear Predicted Cepstral Coefficients (LPCC). The number of coefficients, which is also the number of dimensions of the extracted

feature, is often a configurable parameter. The mathematical details of calculating MFCC from acoustic signal can be found in Sahidullah and Saha (2012). For LPCC, first the parameters of Linear Predictive Coding (LPC - a model based on the source-filter model as presented in Section §3.3.1) are extracted from each sound frame, then either the Auto-regression or Levinson-Durbin recursive algorithms are used to derive their cepstral coefficients. Algorithmic details of both methods can be found in Zbancioc and Costin (2003).

5.2.2.3 Use of abstract features in related work

The most widely used are MFCC and LPCC, which are based on the human's perception and production of sound, respectively. As mentioned above, when feature selection is involved, often descriptive features and abstract features are used together. Examples of this approach can be found in Somervuo, Aki Harma, and Seppo Fagerlund (2006), Seppo Fagerlund (2007), Gunasekaran and Revathy (2010), Lasseck (2015), Lasseck (2016). Abstract features are also used on their own without mixing with descriptive features, for example using MFCC only: Kwan et al. (2004), J. Cai et al. (2007), Graciarena et al. (2010), Weninger and Schuller (2011), Koops et al. (2014); LPC only: Selouani et al. (2005), Juang and T. M. Chen (2007); Wavelet coefficients only: Sun et al. (2013); Wavelet and MFCC: Chou and Liu (2009); MFCC and LPC: Ranjard and Ross (2008), Lakshminarayanan et al. (2009), Fodor (2013). A number of studies also use modified versions of MFCC to make it less tied to human perception. Such modifications include:

- Using a different scale than Mel-scale, for example Human Factor Cepstral Coefficients (GFCC): Wielgat et al. (2007); Bark-frequency Cepstral Coefficients (BFCC): Boulmaiz et al.; Mitrović et al. (2016; 2006); Greenwood function cepstral coefficients (GFCC): K. Adi et al. (2010)
- Adjusting MFCC's parameters, for example: adjusting the bandwidths of the mel filterbanks: Shannon and Paliwal (2003), Ranjard and Ross (2008); bypassing the decorrelation step: Stowell and Plumbley (2014).

MFCC appears to be the most widely used. The reason for using MFCC in birdsong appears to be simply due to its successful in speech analysis. Attempts to evaluate MFCC using bird sounds as data have been made but drew mixed conclusions. Chang Hsing Lee, Y. K. Lee, et

al. (2006) and Chou and Ko (2011) found that MFCC outperforms LPCC, Fox (2008) found little difference between MFCC, BFCC and LPCC. Between MFCC and statistical features, Seppo Fagerlund (2007) found that MFCC performs better although a mixture of both works best, while Briggs, Raich, et al. (2009) found that spectral density outperforms MFCC. Comparison between MFCC and BFCC showed no significant differences in speech (Shannon and Paliwal 2003) and animals (Boulmaiz et al. 2016), but Skowronski and Harris (2004) found that GFCC is more robust to noise in speech recognition tasks.

A thorough comparison of all acoustics features for birdsong is still lacking, but it is highly unlikely to yield any universally "best" feature set. Due to the diversity of vocalisation between different species, each feature may be more suitable for one species and less effective for others. Rather than attempting to establish a broad claim about what the "best" features are, it is wiser to perform feature selection on a case-by-case basis using empirical evidence.

5.3 Feature length standardisation

Syllables vary in length; even syllables of the same type will be sung at slightly different length for each utterance. The lengths of the feature vectors are therefore proportionally unequal. This makes it difficult to express syllables efficiently as points in a finite dimensional vector space. However, a vector space where data points can be differentiated by their relative positions is a fundamental process of many computational analyses. Often there is no natural way to calculate the distance between two vectors of different lengths. This difficulty is increased when dealing with sequential data, such as birdsong syllables in this case, because the structure of the underlying process is often hard to infer (Bicego et al. 2003). For most computer algorithms, it is necessary to perform standardisation of input length as a preprocessing step. This step is intrinsic even in algorithms that were proposed to process variable length input directly.

In a study on sequence data clustering, Bicego (Bicego et al. 2003) identified three approaches to feature length standardisation: *feature-based*, *proximity-based* and *model-based*. The *feature-based* approach extracts a set of features that captures the dynamics of the temporal pattern. *Proximity-based* approaches aim at devising a similarity or distance measures between sequences regardless of their difference in length. Finally, in *model-based* approaches, the main effort is to find an analytical model (or a set of models) that best fits the data, through which each sequence

can be mapped to a data point in a latent space. A similar categorisation is found in (Ye 2003) where two approaches are identified based on whether an assumption can be made regarding the source or method of generation of the sequences. The *discriminative approach* makes no such assumption, thus it is equivalent to the combination of *feature-based* and *proximity-based* whereas the *generative approach* does thus is equivalent to the model-based approach. Based on these studies, I identified four categories of feature length standardisation methods: *aggregative methods*, *image descriptive methods*, *dictionary methods*, and *model-based methods*.

For *image descriptive methods*, image processing techniques are applied to detect blobs of high energy (regions of interest - ROI) in two dimensional representations (e.g. a spectrogram, for example in (Briggs, Lakshminarayanan, et al. 2012; Ruiz-Muñoz et al. 2015) or a Wavelet scalogram (Seppo Fagerlund 2007; Selin et al. 2007)), from which a number of descriptors can be extracted. However, these methods can only work under the assumption of high energy to noise ratio, such that within the end points of a syllable, only one ROI is found. Thus, it is not generally applicable to birdsongs, especially those with high harmonic structures.

For *dictionary methods*, syllables are represented by their proximities to a set of representative templates (a *dictionary* where each template is a *codeword*). Proximity can be calculated using methods such as Dynamic Time Warping (Chu and Blumstein 2011; Kogan and Margoliash 1998; Ranjard and Ross 2008; Somervuo and Aki Harma 2004), spectrogram cross-correlation (Myron C. Baker and Logue 2003; Lasseck 2013; Lasseck 2015; Potamitis 2015), k-means (Briggs, Raich, et al. 2009; Somervuo and Aki Harma 2004; Wellock and Reeke 2012). I identified several issues with these methods which make them ill-suited for the multi-user and highly interactive nature of *Koe*. First, the dictionary is specific to one species thus the results are not comparable between species. Second, the quality of these methods depend very much on the proximity method. Finally, proximity measures are slow and exponentially so as the size of the dictionary increases.

For *aggregative methods*, feature vectors are stretched or squeezed to a fixed length. If there are little changes in the acoustic properties of the syllable frames, the syllables can be assumed stationary. In this case, they can be aggregated using *summary methods* and temporal order of frames is completely discarded. The assumption of stationary is true in limited cases, such as insect sounds. However, it is not uncommon to see studies of birdsong to make this assump-

tion, either implicitly or explicitly. On the other hand, *resampling methods* perform stretching or squeezing the feature sequences to a set length. This approach is more acceptable than *summary methods*, as it is justifiable to assume that during a small number of frames, the acoustic properties are close to stationary. In *Koe*, both *summary methods* and *resampling methods* are available to the users. More details are given in Section §5.3.1.

For *model-based methods*, feature vectors are converted to a representation in latent space by a heuristic model (which involves training), such as Hidden Markov Model (HMM) or Neural Networks. Through training, these models are able to learn the temporal structure of the syllables one time step at a time and build a low fidelity representation of the entire duration in their core (the hidden layers of a neural network, or the hidden states of a HMM). Because the core has fixed length, the low fidelity representation is also fixed length. Through this process, a syllable of any length can be compressed into a data point in latent space, where its temporal structure is abstract but not lost and its acoustic properties are abstract but not distorted. This is intuitively very similar how our brains form a rough idea about a sound we hear, no matter how long or short it is. In *Koe*, this type of feature length standardisation is not yet available to the front-end user, however initial experiment shows promising results. More details are given in Section §5.3.2.

However, only *aggregative methods* are available to *Koe*'s users. *Model-based methods* are experimental but showing promising results.

5.3.1 Aggregative methods

This category expands the definition of *feature-based* by (Bicego et al. 2003) to include methods that discard the temporal order of syllable frames. If there is little change in the acoustic properties of the syllable frames, the syllables can be assumed stationary. In this case, they can be aggregated using *summary methods* and completely lose the order of the frames. On the other hand, *resampling methods* perform stretching or squeezing the feature sequence to a set length - through which process the frame order isn't completely lost, but distorted. The last set of methods, named the *region of interest (ROI)-descriptor* methods, involves image processing techniques to detect the area with high concentration of acoustic signal and then use description of this area to represent the signal. These methods can only be used if the primary feature value has

spectrogram-like visualisation, such as spectrum, wavelet coefficients and MFCCs.

5.3.1.1 Summary methods

Simple statistical properties that summarise the whole acoustic feature sequence are surprisingly common in birdsong analysis. In the majority of studies that employed these methods, abstract acoustic features such as MFCCs tend to be the only source or one of the sources from which statistical properties are calculated. Means and variances of descriptive features and mean MFCCs were used in Seppo Fagerlund (2004), Fagerlund and Harma (2005), Seppo Fagerlund (2007), Evangelista et al. (2014), Qian et al. (2015). Average, min, standard deviation of MFCCs and Mel spectra were used in Stowell and Plumbley (2014). Koops et al. (2014) used only MFCCs but aggregated in three different ways for comparison: only mean; mean + variance; mean + variance, speed (first derivative), acceleration (second derivative) and means + variances of three equal segments (to be explained in Section §5.3.1.2). Myron C. Baker and Logue (2003) used the center spectrogram frame of the syllable. Very rarely, statistical properties of a single descriptive feature were used, for example: Z. Chen and Maher (2006) extracted 12 properties of the dominant frequency sequence. Figure 5.4 illustrate the process using an example of two unequal length feature sequences being summarised by average.

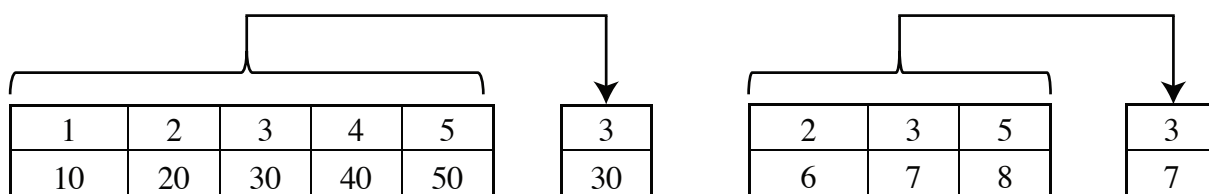


Figure 5.4: Two two-dimensional sequences of different lengths (left and right) summarised using average into two-dimensional feature vectors of the same length (length=1)

5.3.1.2 Resampling methods

Resampling methods for length standardisation often take the form of subdividing the feature sequence into a fixed number of equal-length subparts before applying summary methods in the section above. This approach is a middle ground between discarding temporal information and keeping it intact. The level of distortion of the temporal information following this approach depends on the number of subparts the syllables are divided into. The number of subparts is three in Pozzi (Pozzi et al. 2010), Chen (W. P. Chen et al. 2012), and Koops (Koops

et al. 2014), eight in Acevedo (Acevedo et al. 2009), while in Nanayakkara (Nanayakkara et al. 2007) this number is 20. Figure 5.5 illustrate the resampling process using an example of two unequal length feature sequences being resampled by using average of three subparts.

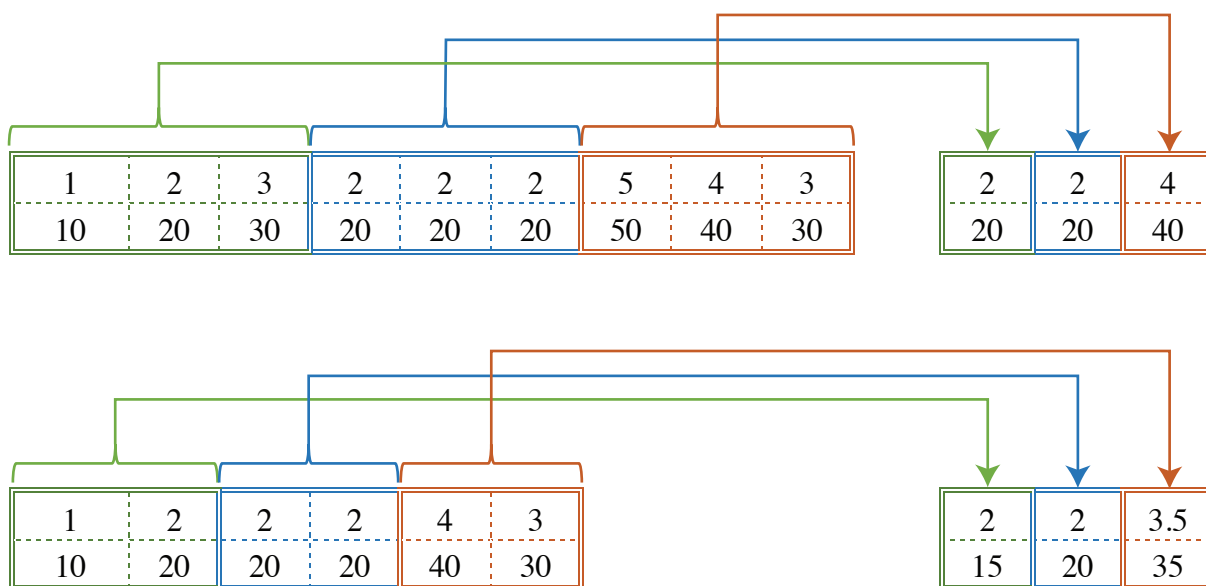


Figure 5.5: Two two-dimensional sequences of different lengths (top and bottom) resampled using average of three subparts into two-dimensional feature vectors of the same length (length=3). The colours represent the correspondence of subparts' original value and final value after resampled

5.3.2 Model-based methods

In this section I review neural network-based methods and follow up with the implementation of an auto-encoder in *Koe*. Hidden Markov Model is a viable choice, however it is not used in *Koe*. Example of works that used HMM for syllable standardisation can be found in Lakshminarayanan et al. (2009), I. F. Chen and Chin Hui Lee (2013) and Levin et al. (2013).

Recently, deep learning has been used for encoding acoustic data into fixed-size vectors. For supervised learning: Maas et al. (2012) trained a regression CNN using acoustic sequences as input and semantic decomposition (such as bags of phonemes) as output. Kamper et al. (2016) used a Siamese network to encode a pair of acoustic sequences and minimise or maximise a distance depending on whether a pair comes from the same or different classes. In both of these works, variable length inputs were normalised (by dividing into 4 subparts in Maas et al. (2012) and by zero-padding in Kamper et al. (2016)). True handling of variable length input starts with G. Chen et al. (2015) uses Long short-term memory (LSTM) to obtain whole word embeddings for a query-by-example search task. For unsupervised machine learning, Chung et al. (2016)

and Shen et al. (2018) used sequence to sequence auto-encoder (SA) to train a recurrent neural network (RNN) to reconstruct audio segments. The hidden layer is used to encode arbitrary length sequences into fixed-size representation. Values of an encoded input in this hidden layer are said to be coordinates of a latent space, and can be used to reconstruct the original sequences undistorted at their full length. Recall Figure 4.5 and the RNN model presented in Section §4.5.2, there an RNN is trained to predict whether the next frame f_i is a syllable frame (expected output 1) or not (expected output 0). An auto-encoder is very similarly structured, except that the network is trained to predict the value of the next frame (expected output is vector f_{i+1}). Once trained, the RNN can be considered a pair of encode-decoder network. From the input layer to the central hidden layer is the encoder, and from the central hidden layer to the output later is the encoder, as depicted in Figure 5.6.

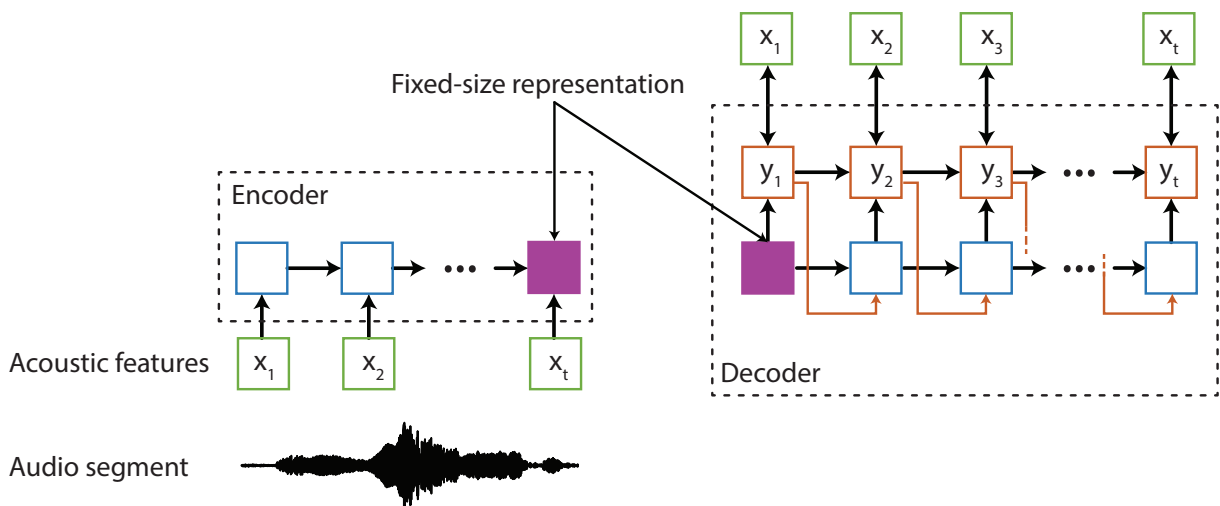


Figure 5.6: An audio auto-encoder is a recurrent neural network (RNN) with two parts: an encoder and a decoder. An audio segment of arbitrary length n has acoustic feature sequence $x = (x_1, x_2, \dots, x_n)$. The encoder encodes this sequence in a fixed-size vector (magenta block) and the decoder decodes this vector to a new sequence $y = (y_1, y_2, \dots, y_n)$. The training process is to minimise the reconstruction error $MSE = \sum_{i=0}^n (x_i - y_i)^2$

Normally, an auto-encoder will have more than one hidden layer, and the structure of the neural network is symmetric, i.e. two parts have the same number of layers and the number of neurons at each layer, except the directions of connection are opposite. Given a syllable as input, the information encoded at the central hidden layer can then be used to reconstruct the essence of the original syllable. A recent work using the same neural network structure for similar type of data is by Yeh et al. (2019) where they trained a time-series auto-encoder to reconstruct factory plant's sensory data for fault prediction. When compared to other machine

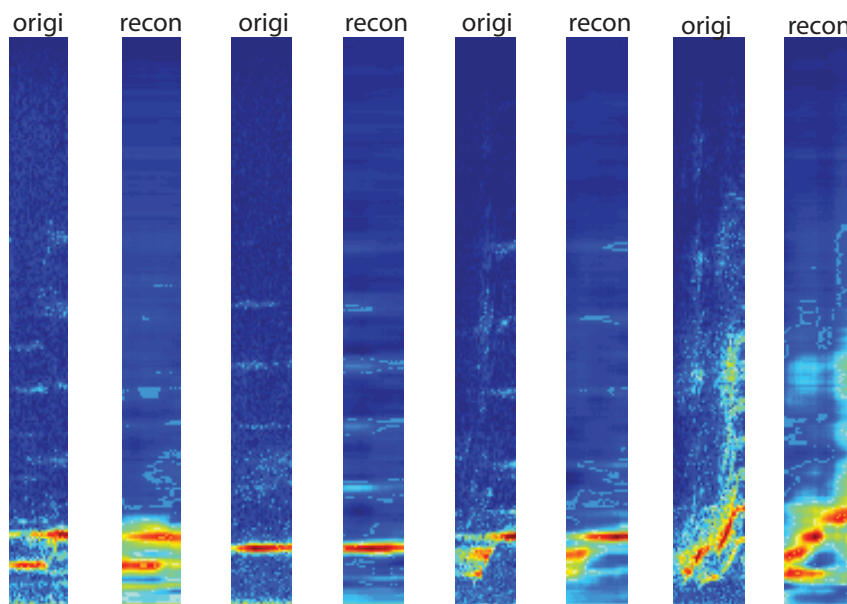


Figure 5.7: Side-by-side spectrograms of sample original syllables (origi) and their corresponding auto-encoder reconstruction. Auto-encoder structure is: 257 (input) \times $257 \times 128 \times 257 \times 257$ (output). Training samples are syllable end points and audio data from the same files listed in Table 4.1.

learning methods, the auto-encoder appeared to be more effective especially in case of data imbalance. Inspired by successful operation on time-series data, I implemented an auto-encoder in *Koe* as an experiment to test how well it performs on audio data. Although this is still experimental, the results are very encouraging. Figure 5.7 shows spectrograms of several original syllables and their auto-encoder reconstruction side by side. We can observe that the auto-encoder is able to capture the essence of the syllables while being relatively noise-proof. This is the proof of concept necessary to justify bringing neural networks to *Koe*'s client side.

5.4 Implementation in *Koe*

5.4.1 User interface

In *Koe*, feature extraction can be found under submenu "Extract features and compare". The page, shown in Figure 5.8, is a control panel where the user can select any combination of features, then any combination of aggregative standardisation methods and then submit the job to *Koe*'s task queue where it will be picked up once a worker becomes available. This is necessary as feature extraction is a time and resource consuming task. Once a job is done, the user will get a notification via email. The finished job can be used as input for other analysis or downloaded (as a binary file) to be used in other software.

Bellbird_TMI: Bellbird_TMI_female_song_added_syllable_changes_9-11-18 | Completed

Features

- | | | | |
|---|--|--|--|
| <input checked="" type="checkbox"/> spectral_flatness | <input checked="" type="checkbox"/> spectral_bandwidth | <input checked="" type="checkbox"/> spectral_centroid | <input checked="" type="checkbox"/> spectral_contrast |
| <input checked="" type="checkbox"/> spectral_rolloff | <input checked="" type="checkbox"/> mfcc | <input checked="" type="checkbox"/> zero_crossing_rate | <input checked="" type="checkbox"/> total_energy |
| <input checked="" type="checkbox"/> aggregate_entropy | <input checked="" type="checkbox"/> average_entropy | <input checked="" type="checkbox"/> average_power | <input checked="" type="checkbox"/> max_power |
| <input checked="" type="checkbox"/> max_frequency | <input checked="" type="checkbox"/> frequency_modulation | <input checked="" type="checkbox"/> amplitude_modulation | <input checked="" type="checkbox"/> goodness_of_pitch |
| | | | <input checked="" type="checkbox"/> amplitude |
| <input checked="" type="checkbox"/> entropy | <input checked="" type="checkbox"/> mean_frequency | <input checked="" type="checkbox"/> spectral_continuity | <input checked="" type="checkbox"/> duration |
| <input checked="" type="checkbox"/> frame_entropy | <input checked="" type="checkbox"/> average_frame_power | <input checked="" type="checkbox"/> max_frame_power | <input checked="" type="checkbox"/> dominant_frequency |
| | | <input type="checkbox"/> spectral_flux | <input type="checkbox"/> spectral_crest |
| <input type="checkbox"/> spectral_skewness | <input type="checkbox"/> spectral_kurtosis | <input type="checkbox"/> spectral_decrease | <input type="checkbox"/> harmonic_ratio |
| <input type="checkbox"/> fundamental_frequency | <input type="checkbox"/> mfc | <input type="checkbox"/> mfcc_delta | <input type="checkbox"/> mfcc_delta2 |
| | <input type="checkbox"/> log_attack_time | | |

Aggregation methods

- | | | | |
|---|---|--|---|
| <input checked="" type="checkbox"/> mean | <input checked="" type="checkbox"/> median | <input checked="" type="checkbox"/> std | <input checked="" type="checkbox"/> divcon_3_mean |
| <input checked="" type="checkbox"/> divcon_5_mean | <input checked="" type="checkbox"/> divcon_7_mean | <input type="checkbox"/> divcon_3_median | <input type="checkbox"/> divcon_3_std |
| <input type="checkbox"/> divcon_5_median | <input type="checkbox"/> divcon_5_std | <input type="checkbox"/> divcon_7_median | <input type="checkbox"/> divcon_7_std |
| <input type="checkbox"/> min | <input type="checkbox"/> max | <input type="checkbox"/> variance | <input type="checkbox"/> begin |
| <input type="checkbox"/> end | | | |

Name

Bellbird_TMI_female_song_added_syllable_changes_9-11-18

Download

Submit

Figure 5.8: Main panel of Koe's *Feature Extraction* page. User can choose any combination of features, and any number of aggregative standardisation methods. The user can name the job submit it to Koe's task queue. Once finished, the results can be downloaded as a binary file.

5.4.2 Koe's task queue

Koe employs Celery (Solem 2016) to manage and execute distributed tasks in the background independently from the main web server. Up to 10 tasks can be executed simultaneously in Koe's official webapp. For the docker distribution, this number is configurable and for development environment, Celery can be disabled for Koe to handle tasks directly and to allow the developer to debug. When a task is executed, Celery workers can access Koe's database to query and store information such as syllable endpoints and task progress, such that Koe can notify the user when the task finishes or how much percentage-wise the task has progressed, as shown in Figure 5.9

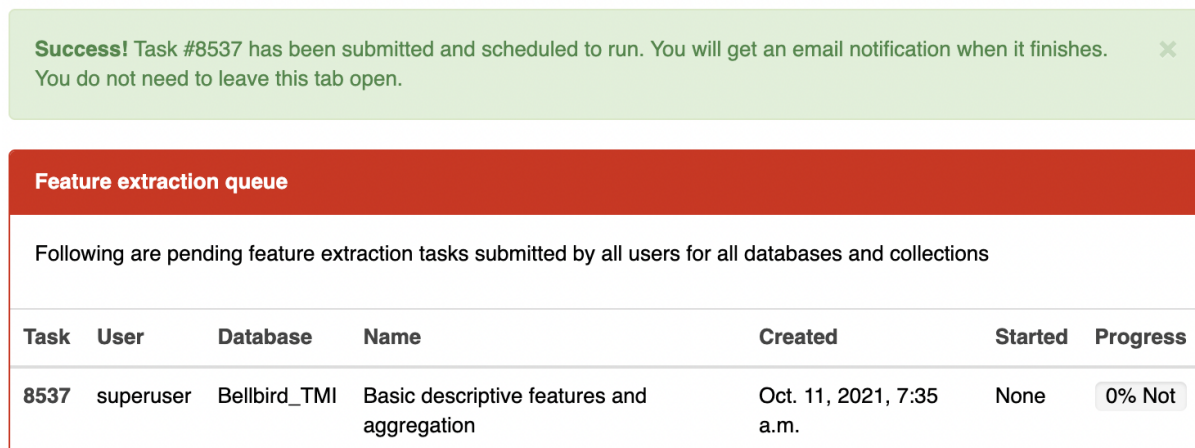


Figure 5.9: Main panel of Koe's *Feature Extraction* page. User can choose any combination of features, and any number of aggregative standardisation methods. The user can name the job submit it to Koe's task queue. Once finished, the results can be downloaded as a binary file.

5.4.3 Implementation and expandability

The list of features and aggregation methods shown in Figure 5.9 come from a database query. Specifically, table `koe_feature` and table `koe_aggregation` (shown in Figure 2.2 in Chapter 2) store metadata for each feature and aggregation methods implemented in Koe's backend. This arrangement allows more flexibility in choosing what can be made available to the users and what can be experimental. New features and aggregation methods can be developed and tested at the backend and will not be available to the user until the developer is certain that they perform correctly.

Koe's features come from three main sources: librosa (Mcfee et al. 2015), SciPy (E. Jones et al. 2014), and self-implemented. In order to keep the process consistent and to allow future expandability, all function calls to these features are wrapped in a common interface shown in Code block 4

```

1 def feature_name(args):
2     # Unrolled args
3     fs, nfft, noverlap = unroll_args(args, ['fs', 'nfft', 'noverlap'])
4
5     # Perform calculation, or call external libraries
6     feature_value = ...
7     return feature_value

```

Code block 4: Common interface of all feature extraction methods

All feature extraction functions have the same function signature where `args` is a dictionary with default key-value pairs given in Table 5.1.

Table 5.1: Key-value pairs that can be provided through the `args` parameters for all feature extraction functions.

Key	Value
<code>wav_file_path</code>	Absolute path to the original <code>wav</code> file.
<code>fs</code>	Sampling frequency of the <code>wav</code> file.
<code>start</code>	Begin of the syllable (millisecond) with respect to the beginning of the audio file.
<code>end</code>	End of the syllable (millisecond) with respect to the beginning of the audio file.
<code>center</code>	A boolean indicating whether padding should be used in calculating power spectral density.
<code>nfft</code>	Number of FFT bins.
<code>noverlap</code>	Number of samples to advance for each frame.
<code>lpf</code>	Low-pass filter frequency.
<code>hpf</code>	High-pass filter frequency.
<code>win_length</code>	Length of the frame (in samples) overwhich the feature is extracted.

Extra arguments can be added dynamically to the `args` object, in situation where the feature extraction function expects specific arguments, for example, `lpcc` and `lpc` requires parameter `order`. This common function signature allows all feature extraction methods to be called in the same way in loop-like fashion, specifically, by function `extract_segment_feature_for_audio_file` in package `feature_utils`, regardless of what combination of features the user requested. Code block 5 shows examples of two features, one is implemented by *Koe* and one by `librosa`. In this example, `spectral_bandwidth` is not implemented by *Koe* but is simply a wrapper to call `spectral_bandwidth` from library `librosa`. To do that the wrapper simply unrolls the arguments to get all the information needed to call the actual function, and the caller (`extract_segment_feature_for_audio_file`) does not need to know specifically what arguments the actual function needs. Acoustics feature extraction is a dynamic field where new features are invented frequently. Therefore, this set up will allow advanced users and future contributors of *Koe* to expand the feature set as they need.

```

1 def frame_entropy(args):
2     psd = get_psd(args)
3
4     # Entropy of each frame (time slice) averaged
5     newsg = (psd.T / np.sum(psd)).T
6     return np.sum(-newsg * np.log2(newsg), axis=0)
7
8 def spectral_bandwidth(args):
9     psd = get_psd(args)
10    fs, nfft, noverlap = unroll_args(args, ['fs', 'nfft', 'noverlap'])
11    hopsize = nfft - noverlap
12    return rosaft.spectral_bandwidth(y=None, sr=fs, S=psd, n_fft=nfft,
13                                   hop_length=hopsize)

```

Code block 5: Examples of feature extraction function using the common interface. `frame_entropy` is *Koe*-implemented, while `spectral_bandwidth` is a wrapper for a function in `librosa`. They all have the same function signature.

5.4.4 Storage

Extracted feature values are binary data that best stored on hard-drive instead of database. This is due to several reasons:

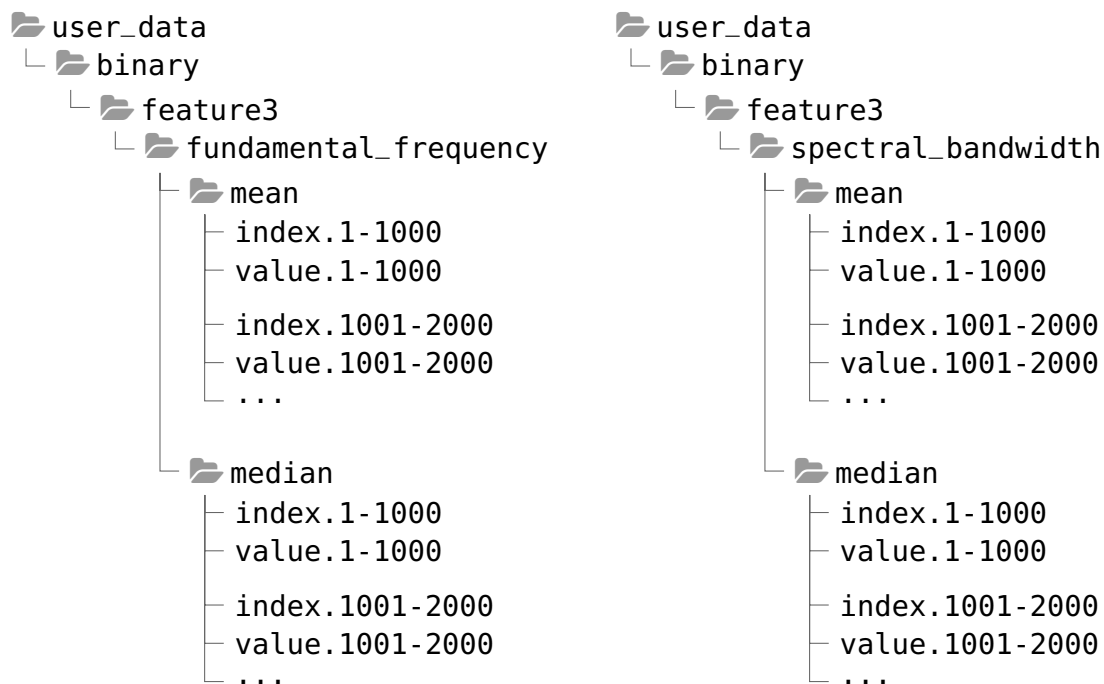
- I attempted to extend Luscinia (R F Lachlan 2007), which stores binary data on an H2 database (a SQL database engine written specifically for Java). From extensive experience with this software, I found that storing binary data on database is an extreme performance bottleneck.
- Once extracted, feature values are stable, so database update queries for this purpose is rare, thus using SQL database is missing the point.
- The size of the database never shrinks, thus on the rare occasion when an update query to change stored values is performed, it incurs permanent loss of storage. To recoup this loss, the database needs to be trimmed frequently, leading to higher cost of maintenance and more down times.

Thus a principle of software design in *Koe* is that no binary data can be stored in the database. Instead, feature metadata is stored against syllable ID, which reference the binary file where the actual data is stored on the hard drive. *Koe* initially used Hierarchical Data Format version 5 (HDF5) to store binary data, however I quickly realised that this data format is only suitable for hierarchical data, while feature values are not. In addition, *Koe* allows the user to choose any set of syllables from any combination of dataset to form a "collection", from which any combination of features and aggregation methods can be chosen. This requires random access to the binary data which is extremely slow in HDF5. To solve this issue, I wrote a new library tailored specifically for *Koe*'s needs, called [koe/binstorage3.py](#) (3 is the version of the current implementation).

5.4.4.1 Physical storage

On hard drive, binary feature data is stored in the a similar directory structure as shown in Figure 5.10, where Sub-figure 5.10a and Sub-figure 5.10b list the actual storage of the *fundamental frequency* and *spectral bandwidth* values of the first 2000 syllables on *Koe*'s server, respectively.

The current binary storage is version 3, hence the name [binstorage3](#) and this is not a random choice. Version 1 of the binary storage libraries stores everything in one big file, where version 2 does the opposite: one file per feature value per syllable ID. Both approaches lead to



(a) Fundamental frequency

(b) Spectral bandwidth

Figure 5.10: Directory structure of feature values stored on the hard drive.

downgraded performance. With one big file, the cost to append or update data grows exponentially as the data file grows. With individual files, the sheer number of files on the hard drive quickly overwhelms the operating system when other components of *Koe* perform a simple directory listing. Version 3 takes the middle ground by employing pagination. Here, each pair of **feature—aggregation method** forms a folder. Feature values of each block of 1000 syllables (according to their IDs) are concatenated into one file e.g. `fundamental_frequency/mean/value.1-1000` contains the mean fundamental frequency values of syllables #1 to syllable #1000. To know where the feature values of syllable #*x* starts and ends within the concatenated file, as well as the dimension details of the original feature value array, an index file e.g. `fundamental_frequency/mean/index.1-1000` mapping the ID to the necessary metadata is also provided.

5.4.4.2 Store and retrieve data

This library is implemented in `koe/binstorage3.py` and provides two overall operations: **store** and **retrieve**. The **store** operation internally calls one of the following operation: `store_new`, `update_simple`, `append`, `update_expand`. The specifications of these operations are provided in Table 5.2.

The key to the performance of `binstorage3` is that the value file is never rearranged. In the

Table 5.2: Operations to store and retrieve data by `binstorage3`

Operation	Description
<code>store_new</code>	Create a new value-index pair and write binary data when the syllable ID is not in the range covered by any of the existing value-index pairs.
<code>append</code>	Add binary data and index to the end of the value-index pair when the syllable ID is in the range covered by an existing value-index pair.
<code>update_simple</code>	Modify the value file and update the index file (if necessary) if the previous feature values of the target syllable is already stored and the new values (array) has the same or shorter length of the previous value array.
<code>update_expand</code>	If the previous feature values of the target syllable is already stored and the array of new value is longer than the previous array, then leave the previous value array untouched, append the new value array at the end of the value file, and update the index file.

rare case where it is necessary to update an existing feature value array with a longer array, the longer array is appended and the index of that syllable is simply updated to point to the new location, the previous value array is still there but never used. This might increase hard drive use but only by an insignificant amount.

To retrieve feature value for one syllable, `binstorage3` first identify the location of the value-index files, which is

`user_data/binary/feature3/<feature name>/<aggregation method>/`,

calculate the page that the syllable ID belongs, e.g. syllable ID 123456 belongs to the page `<123000-124000>`, thus the value-index files are respectively `value.123000-124000` and

`index.123000-124000`. Next, `binstorage3` opens the index file to search for the metadata associated with such ID. Finally, knowing the begin and end, and dimensions of the value array, `binstorage3` perform a `seek` and `read` to get the value array, without having to read the entire file.

5.5 Conclusion

The aim of feature extraction is to derive a representation of the signal that can be used to map each datapoint to a dimensional space where the distance between two datapoint reflect how similar they are. A good dimensional space enables the datapoints (in this case, birdsong syllable-

bles) to be sorted or clustered, which in turn allows operations such as classification or cluster structure analysis to take place. In Chapter 6 that follows, I present how these operations are carried out in *Koe*.

6.1 Introduction

Koe was initially written as a tool to facilitate manual labelling of New Zealand Bellbird (*Anthornis melanura*) syllables using both visual and audio cues. Bellbirds have a very diverse vocal repertoire, with hundreds of fine-scale syllable types. Some types appear very rarely while others are hundred times more frequent. This makes classification very difficult to do in existing software such as Luscinia, Raven, or Sound Analysis Pro as they operate on a single-file basis, thus are incapable of showing all segmented syllables at once. *Koe* partly solved this issue by providing a *Unit table* page, where all segmented syllables from all songs in the database are present in one table. However it is still impossible to display and keep track of tens of thousands of syllables in one single view. To solve this issue, I implemented a *similarity index* to sort the unit table by similarity, and later I implemented *interactive clustering visualisation* where all syllables are visualised in two or three dimensional space where they can be grouped by proximity and labelled in bulk. Both similarity index and clustering visualisation are the products of feature extraction presented in the Chapter 5. In this chapter, I start with introducing the concepts cluster analysis, followed by visualisation techniques. The user interface where cluster analysis and similarity index are utilised is presented next. This is followed by the implementation of these functions in *Koe*. Finally I present a case study to demonstrate how *Koe* can be a great tool for validating multi-expert labelling.

6.2 Cluster analysis

Cluster Analysis or *Clustering* is a technique widely used in data mining to partition a set of unlabelled data objects into groups (clusters) where members of the same group share similar characteristics. It is a form of machine learning where the training process is unsupervised, although typically some prior knowledge (such as the number of clusters or statistical properties of the clusters) are still required.

The classification of birdsong syllables into groups is useful for detecting overall changes, such as comparing songs between species, populations or individuals over space or time (Webb et al. 2021). Traditionally this has been done by (manual) expert's judgement, for example: Baptista and King studied geographic variations and dialects in the songs of the Puget Sound White-Crowned Sparrow (Luis F Baptista 1977) and the Montane White-Crowned Sparrows (Luis F. Baptista and King 1980) by dividing the segmented syllables into groups according to their types (such as note, whistle, buzz) and calculating the statistics for each group. However, manual classification inevitably introduces human's subjectivity into the process. While this is not bad practice *per-se*, it reduces the reproducibility of the process. Thus, Nelson et. al. (Nelson, Hallberg, et al. 2004) repeated the study on the Puget Sound White-Crowned Sparrow using two independent judges with different knowledge of the origin of the syllables to sort complex and simple syllables into predefined groups. The judges were allowed to create new categories if necessary. They found that the judges highly agreed with each other, thus justifying their use of the clustering result from a single judge later in their study. While the problem of human subjectivity can be partly mitigated by demonstrating agreement between independent judges, the only way to provide full reproducibility is by removing the human factor and using computer-based methods.

In this section, I introduce two clustering techniques that are available to *Koe's* users.

6.2.1 Hierarchical clustering

Hierarchical clustering is a method that aims to produce clusters in a multi-level tree-like structure. From the bottom up, each object belongs to a cluster which in turn belongs to a larger cluster and so on. At the top of the tree is the super cluster where all objects belong. The connectivity between two objects is determined by their proximity (distance) in a feature space. A

cluster closer to the bottom of the tree contains objects that are closer to each other, and vice versa. This tree-like structure is often visualised by a dendrogram, shown in Figure 6.1.

Hierarchical clustering is commonly used in biology to formulate the phylogenetic relationship between species or subspecies. The use of this method in birdsong analysis can be found in many studies. For example, M. E. Anderson and Conner (1985) classify dialects of the Northern Cardinal (*Cardinalis cardinalis*) based on frequency of appearance of each type of syllables; Podos et al. (1992) analyse song repertoire of the Song Sparrows; and Große Ruse et al. (2016) use syllables found in one song of the Red Warbler as input to compare the result of an automatic clustering framework with that of experts.

Hierarchical clustering can be implemented using either of two strategies:

- **Agglomerative method:** is a bottom-up strategy that starts with each data point being one cluster of its own and iteratively merges small clusters into one.
- **Divisive method:** is a top-down strategy that starts with all data points belonging to one cluster, and then iteratively dividing big clusters into small ones

Both strategies lead to identical results, given that the same *linkage criterion* is used. Linkage criteria are different ways to calculate the distance between clusters, or the radius of the enclosed cluster, thus they affect the final shape of the hierarchy. To name a few: *single-linkage* determines the cluster distance by shortest (minimum) distance from any member of one cluster to any member of the other cluster: $\min (d(a, b) : a \in A, b \in B)$, in contrast, *complete linkage* uses maximum cross-cluster distance $\max (d(a, b) : a \in A, b \in B)$ while UPGMA finds middle ground between the two by using average distance $\frac{1}{|A||B|} \sum_{a \in A} \sum_{b \in B} d(a, b)$. Figure 6.1 illustrates the clustering result using UPGMA for a toy dataset containing five data points with given pair-wise distances.

“Cut the tree” is the method of acquiring specific clustering configuration by setting a *cutoff* threshold to be the maximum radius of the cluster. The example shown on the right side of Figure 6.1 is the clustering configuration when the cutoff is set at 15. Two clusters $C_1 = [d, c]$ (in blue) and $C_2 = [e, b, a]$ (in red) are formed because their radii are 14 and 11 respectively. It appears straightforward but the cutoff threshold is crucial for the final clustering result and there is no objective way to determine it. Often trial and error are involved in finding a reasonable value for the cutoff threshold, relying on prior knowledge of the dataset (i.e. knowing

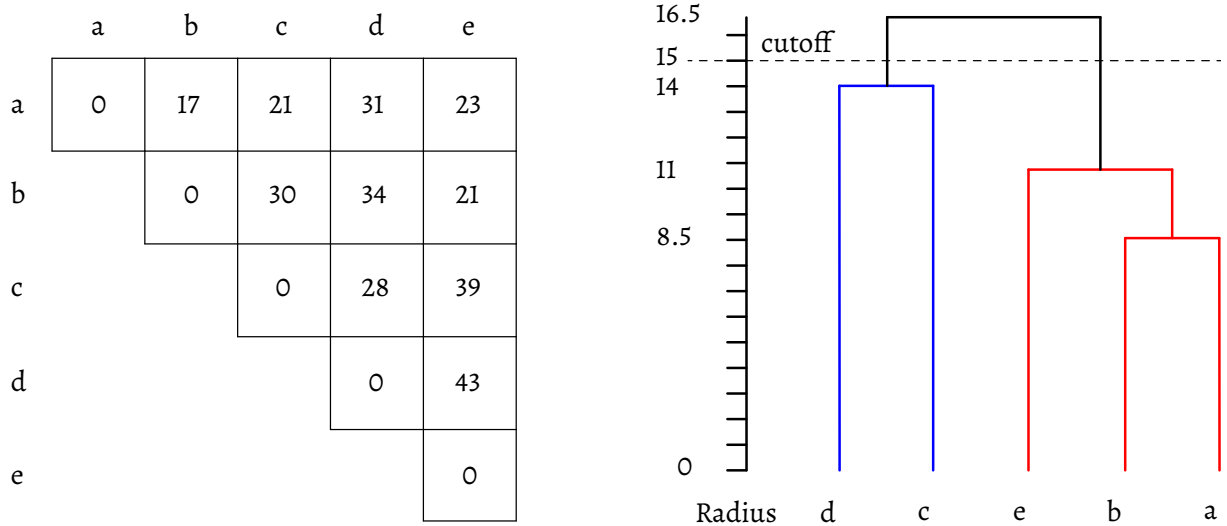


Figure 6.1: Illustration of hierarchical clustering using UPGMA. Left: upper triangle of the distance matrix. Right: clustering result.

roughly how the cluster should look like *a priori*). This is documented by Gil and Peter J B Slater (2000) in their study of Willow Warbler (*Phylloscopus trochilus*) song repertoire. They subjectively determined the cutoff threshold that would yield a “sizeable” number of clusters. Podos et al. (1992) tackles this problem by using a moat index, which describes the degree to which cluster groups are isolated or externally discontinuous to each other. The cutoff value that maximises the moat index is determined to be the optimum. Gil and Slater did try this approach but eventually decided against it because of the effect of outliers.

6.2.2 Semi-automatic clustering

While fully-automated clustering has been used in previous studies such as by Ranjard and Ross (2008) to study dialectal variations of birdsongs or by Chou and Ko (2011) to recognise species, it is ill-suited for syllable classification where there are many types of syllable and their distribution in a species’ repertoire is uneven, i.e. there are a lot more instances of commonly found syllables than that of rare syllables. This is the challenge that my colleagues and I faced with bellbird songs. I judged that semi-automatic clustering is most suitable in this situation. This is the technique where labelling is still done manually, but expedited by bulk-labelling of similar syllables in batches, rather than one-by-one (as in other software). Bulk-action is possible with a visual aid, such as a two- or three-dimensional plot, where syllables are clustered to a high degree. This technique has been employed by several previous studies such as by O. Tchernichovski et al. (2004) and Derégnaucourt et al. (2005) to study the song crystallisation process

in zebra finches. *Koe* provides two tools for bulk-labelling: by labelling all multi-selected rows of the unit table after sort with similarity index, and by labelling all units enclosed in a boundary drawn directly on an interactive two-dimensional plot. The next section explains in detail how these actions can be performed by the user.

6.2.3 Dimensionality reduction

Many frequently used features, which are extractable by *Koe*, are multi-dimensional; for example MFCC usually has at least 13 dimensions. Features are also usually used in combination with the intention that the more features are used, the more differentiable structure of the datapoints can be covered, and the correlation between features can be reduced afterwards using a technique called dimensionality reduction. This technique seeks to reduce the number of dimensions while preserving the differentiability as much as possible. This is similar to lossy compression techniques widely employed for audio and video insofar as the results should resemble the original data while size is significantly smaller. In *Koe*, dimensionality reduction is used mainly for visualisation. In order to present syllables as data points on a cluster plot, their acoustic features must first be reduced to two or three dimensions.

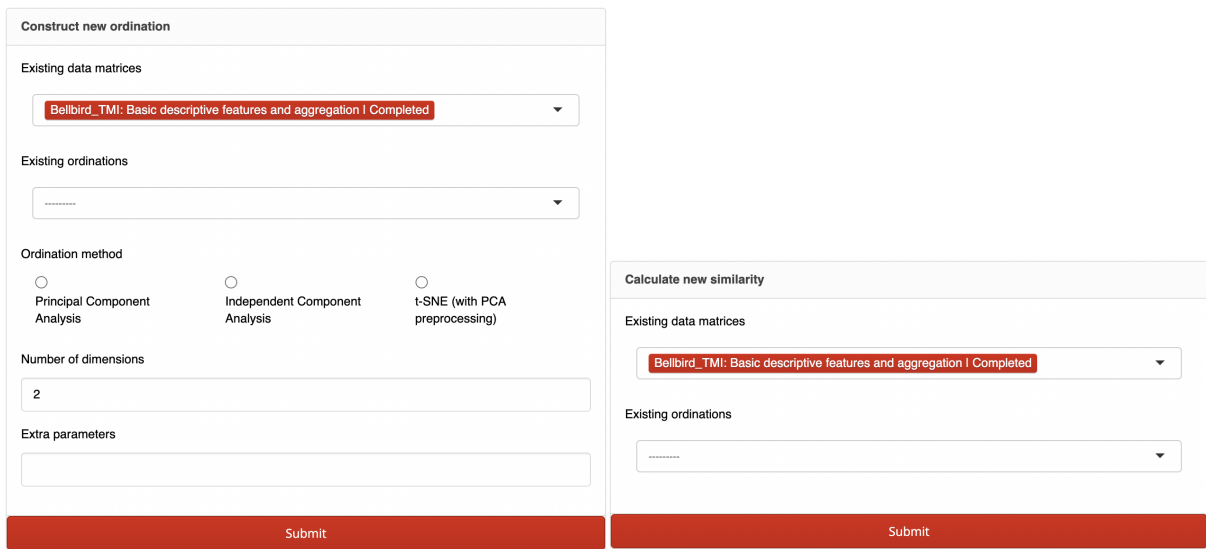
Koe implements popular ordination techniques, such as Principal Component Analysis (PCA), Independent Component Analysis (ICA) and Multi-Dimensional Scaling (MDS), as well as the more recently proposed t-distributed Stochastic Neighbour Embedding (t-SNE) (Van Der Maaten and Hinton 2008). t-SNE aims to preserve local structure in the data, so is particularly effective for defining and discriminating between different clusters.

6.3 User interface

6.3.1 Submit jobs to construct ordination and calculate similarity index

Construct ordination and *Calculate similarity* can be found under submenu *Extract features and compare*, in the same place with *Extract unit features*. Similarly, the user needs to submit a job to *Koe*'s task queue and wait for a notification email.

Figure 6.2 shows the control panels where the user can submit these jobs. Sub-figure 6.2a



(a) Control panel to construct ordination

(b) Control panel to calculate similarity.

Figure 6.2: Control panels accessible under submenu *Extract features and compare* to create an ordination for clustering or similarity index for sorting.

shows the form to create an ordination job. The user can choose a data matrix (which is an extracted feature set), the ordination method (either PCA, ICA or t-SNE), number of dimensions (can only be 2 or 3), and optionally any parameters that they wish to send directly to the underlying function. Sub-figure 6.2b shows the form to create a similarity index job. The similarity index can be extracted from raw feature values, or from an ordination.

6.3.2 Using similarity index to sort syllables in the unit table

Users of *Koe* can create as many similarity indices for the same database as they see fit. Once the job has finished, the similarity index can be selected and applied dynamically to an existing unit table. This can be done by selecting an option in the *Current similarity* combo box on the left side control panel, as shown in Figure 6.3.



Figure 6.3: Left side control panel of the unit table. Database *Bellbirds* has two similarity indices calculated from two different sets of feature values. Upon selection the index is applied dynamically to the unit table.

To demonstrate the usefulness of a similarity index, Figure 6.4 show the unit table of database *Bellbird_TMI* where a filter (`family:down|upsqueak`) is applied to show only syllables that are

classified to belong to either *Down* or *Upsqueak* family. This is done on purpose to show more than one variety of syllables within the limited space of a figure. The filtered table is then sorted by similarity index (ascending order). We can observe that the natural orders of syllables do put similar syllables near each other.

3/38 Filter family: down upsqueak										
Song	<input type="checkbox"/>	Spectrogram	Label	Family	Sex	Quality	Similarity Index	Duration	Date	Note
CUV_2016_11_05_	<input type="checkbox"/>		Down(notched)	<input checked="" type="checkbox"/> Down	<input type="checkbox"/>		52	111	2016-11-05	
CUV_2016_11_06_	<input type="checkbox"/>		Down(notched)	<input checked="" type="checkbox"/> Down	<input checked="" type="checkbox"/> F	EX	60	137	2016-11-06	
CUV_2016_11_05_	<input type="checkbox"/>		Down(notched)	<input checked="" type="checkbox"/> Down	<input checked="" type="checkbox"/> F	EX	62	117	2016-11-05	
CUV_2016_11_05_	<input type="checkbox"/>		Down(notched)	<input checked="" type="checkbox"/> Down	<input checked="" type="checkbox"/> F	G	70	113	2016-11-05	
LBI_2016_04_06_	<input type="checkbox"/>		Upsqueak(harmonic)	<input checked="" type="checkbox"/> Upsqueak	<input checked="" type="checkbox"/> M	EX	423	291	2016-04-06	
LBI_2016_04_06_	<input type="checkbox"/>		Upsqueak(harmonic)	<input checked="" type="checkbox"/> Upsqueak	<input checked="" type="checkbox"/> M	EX	430	312	2016-04-06	
LBI_2016_04_09_	<input type="checkbox"/>		Upsqueak(harmonic)	<input checked="" type="checkbox"/> Upsqueak	<input checked="" type="checkbox"/> M	VG	446	373	2016-04-09	
TMI_2014_03_19_	<input checked="" type="checkbox"/>		Upsqueak(shrieky)	<input checked="" type="checkbox"/> Upsqueak	<input checked="" type="checkbox"/> M	G	689	162	2014-03-19	
TMI_2014_11_10_N	<input checked="" type="checkbox"/>		Upsqueak(shrieky)	<input checked="" type="checkbox"/> Upsqueak	<input checked="" type="checkbox"/> M	G	721	172	2014-11-10	

Figure 6.4: Koe's unit table sorted by similarity index (ascending order) filtered to show only syllables in families *Down* or *Upsqueak*. Example data are New Zealand bellbird *Anthornis melanura* song units from database [Bellbird_TMI](#).

The unit table also allows bulk-labelling by multi-selecting syllables that look and sound similar, and then select "Bulk set value" from a dropdown menu from the *Label*, *Family* or *Subfamily* column. The sound of each syllable can be played back by clicking on its spectrogram. The playback speed can also be adjusted to better appreciate temporal details in short or complex syllables.

6.3.3 Using ordination for bulk labelling in an interactive cluster visualisation

Koe's unit table is a big advance on other software in that all units (syllables) are accessible in one table. However, the space limits of a screen still limit the number of units that can be viewed simultaneously. When classifying units, only seeing part of the table at a time can result in erroneous labelling. The difference between syllables can gradually grow slow enough to be un-

noticeable as the page scrolls, however the accumulative difference could add up significantly resulting in having syllables that look and sound differently being given the same label. *Koe* provides a solution for this issue with the *Interactive Ordination* plot, as shown in Figure 6.5. This plot is produced at the client side using the library Plotly (Inc. 2015) and thus is very fast and interactive. Interactions between the user and the plot is intercepted and handled in JavaScript.

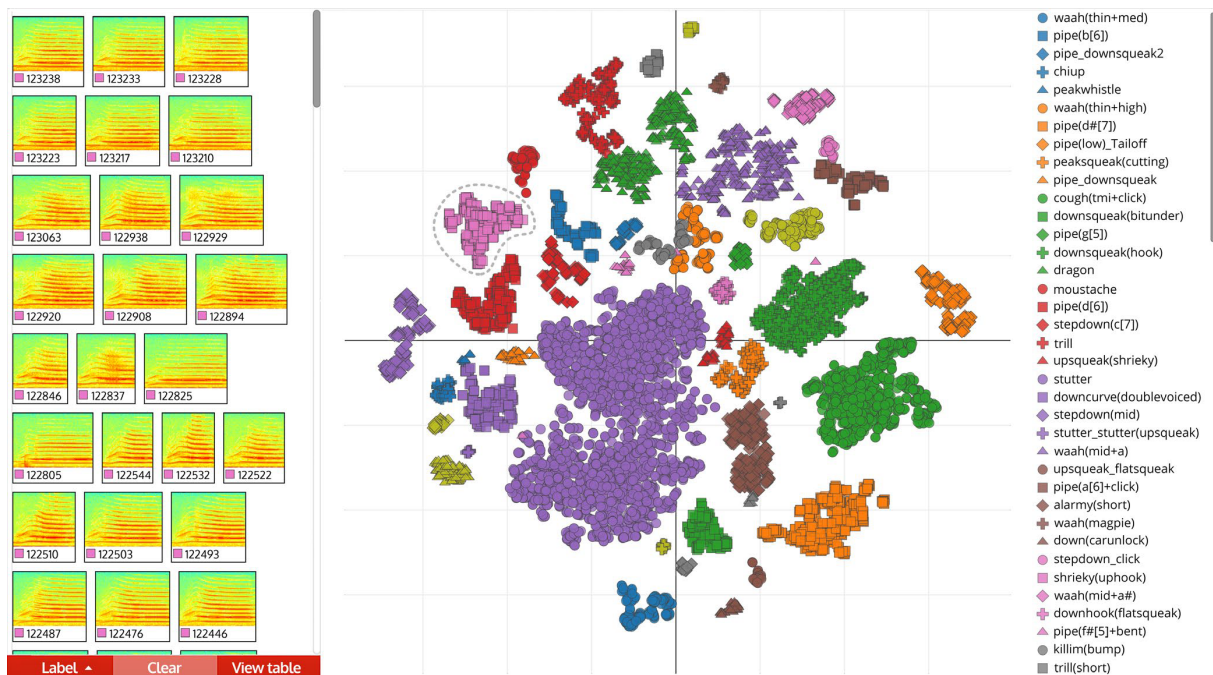


Figure 6.5: Interactive Ordination view allows the user to encircle groups of points on the plot with the lasso tool, to view their spectrograms and hear their audio. Mousing over a point in a selection highlights the corresponding spectrogram in the left-hand panel. Selections can be labelled in bulk directly on the plot or opened as a unit table to view detailed unit information. The user can zoom, toggle the visibility of classes, and export the plot as a vector graphic. This example shows a t-SNE ordination of 7189 syllables of male and female bellbird song.

In this plot, all syllables appear at once on a two dimensional space. The example shown here shows good separation of clusters after all of them have been labelled. The colour and symbol of each syllable are according to the label that it is given. The user typically starts out with an unlabelled data, where all syllables will have the same appearance (e.g. all are gray-rounded shaped). A lasso tool allows the user to make a selection of a group of syllable that appear as a cluster. Upon selection, the spectrogram of all syllables appear on the left side panel where the user can assess whether they look and sound the same. Syllables that don't look and sound like the rest can be excluded, and the final group can be bulk-labelled.

6.4 Implementation in *Koe*

6.4.1 Construct ordination

Upon receiving a job, a Celery worker at the backend calls `binstorage3`'s `retrieve` function to acquire the extracted feature values for the selected datamatrix and calls the selected dimensionality reduction methods to reduce the original data to the selected number of dimensions. All dimensionality reduction methods are wrapped in a common interface for better coding structure and increased extendability. The worker does not know how to call the underlying function specifically, but simply invoke the method by finding its function pointer from a dictionary of methods. In *Koe*, this dictionary is simply a declaration:

```
1 methods = {'pca': pca, 'ica': pca, 'tsne': tsne}
```

Where `pca`, `ica` and `tsne` are the wrappers to the following underlying functions:

Wrapper	Underlying function
<code>pca</code>	<code>sklearn.decomposition.PCA</code>
<code>ica</code>	<code>sklearn.decomposition.FastICA</code>
<code>tsne</code>	<code>sklearn.manifold.TSNE</code>

The wrappers have the same function signature shown in Codeblock 6. `data` and `ndims` (number of dimensions) are by definition required by any underlying function. They are included in an `params` dictionary together with any specific arguments (such as `verbose`, `perplexity`, `n_iter` for TSNE) are constructed and provided as keyword arguments (`**kwarg`). The `params` dictionary is updated with the custom parameters provided by the user, through which these specific arguments can be added, modified, or removed before the call is made. For example, without user-provided parameters, TSNE is called with `perplexity=10` by default. Declaring `perplexity=5` will change this value and `perplexity=None` will remove this parameter, declaring `learning_rate=100` will add this parameter to the final call.


```

1 def method_name(data, ndims, **kwargs):
2     # Construct an argument dictionary for this particular method
3     params = dict(...)
4
5     # Update the argument dictionary with the optional parameters
6     # that the user can input
7     params.update(kwargs)
8
9     # Call the underlying function to perform dimensionality reduction
10    result = ...
11    return result

```

Code block 6: Common interface to all dimensionality reduction methods

6.4.2 Calculate similarity index

A similarity index in *Koe* is an integer defined as the *natural order* of data points in a dendrogram. The natural order of any data point is an enumeration that runs from 1 to N where N is the number of data points. This number represents how early a data point can be found to have membership in a cluster by tracing the dendrogram branches from bottom up. For example, in Figure 6.1, (a) and (b) are the earliest data points that have cluster membership, followed by (e), and finally (d) and (c). Their natural orders therefore are [(a) (b)] = [1, 2]; (e) = 3; [(c), (d)] = [4, 5]. When two data points from a cluster, such as in case of (a) and (b), either of them can be assigned natural order 1 or 2. However in the case of (e) which is linked to an existing cluster, it can only possibly be assigned the one available similarity index 3.

Table 6.1: Calculating natural orders of syllables shown in Figure 6.1. Sub-table 6.1a shows the linkage calculated with average distance, sub-table 6.1b shows the natural orders of syllables calculated by *Koe*. Cluster indices and natural orders are based 0

Left cluster		Right cluster		Distance	Order	Data point
Index	Members	Index	Member			
0	(a)	1	(b)	17	0	(a)
4	(e)	5	(a+b)	22	1	(b)
2	(c)	3	(d)	28	2	(e)
6	(a+b+e)	7	(c+d)	33	3	(c)
					4	(d)

(a) Linkage tree

(b) Natural orders

The calculation of this example in *Koe* is detailed in Table 6.1. Upon receiving a job, a Celery worker at the backend calls *binstorage3*'s *retrieve* function to acquire either the ex-

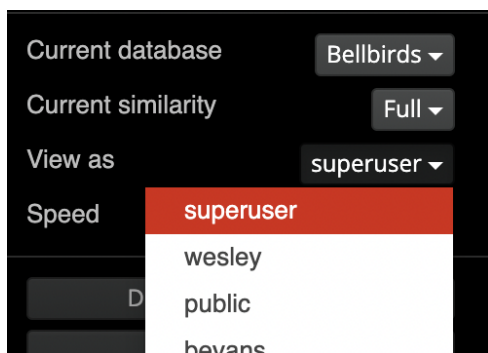
tracted feature values if the user has selected a datamatrix, or the ordination data if the user has selected an ordination. Following data retrieval, the upper left triangle of the distance matrix (`triu`) is calculated for all pairs of points in the dataset. In this example, $\text{triu}(a, b, c, d, e) = [17, 21, 31, 23, 30, 34, 21, 28, 39, 43]$. These numbers are pair-wise distances between any two data point, e.g. the distance between \textcircled{c} and \textcircled{d} is $\text{dist}_{c,d} = 28$. This distance array is transformed into a linkage list using `cluster.hierarchy.linkage` method available in SciPy. The linkage list can be understood as a cluster membership description for all the points and sub-clusters that can be formed. The outcome for this example is shown in Sub-table 6.1a. Tracing the cluster indices from smallest (0) to largest (7) tells us that data points \textcircled{a} and \textcircled{b} initially form clusters 0 and 1 respectively, then \textcircled{c} and \textcircled{d} form clusters 2 and 3 and \textcircled{e} forms cluster 4. After all individual data points have been assigned a cluster index, the linkage iteratively merge them into composite clusters where the distance between a singular cluster and a composite cluster, as well as between two composite clusters is calculated using average distance. For example, $\text{dist}_{e,[a,b]} = \frac{\text{dist}_{e,a} + \text{dist}_{e,b}}{2} = \frac{23+21}{2} = 22$. Thus, $[\textcircled{a}, \textcircled{b}]$ form cluster 5, $[\textcircled{a}, \textcircled{b}, \textcircled{e}]$ form cluster 6, $[\textcircled{c}, \textcircled{d}]$ form cluster 7, and finally $[\textcircled{a}, \textcircled{b}, \textcircled{e}, \textcircled{c}, \textcircled{d}]$ form the root cluster 8 (which is omitted from the linkage tree due to redundancy).

The natural order of a data point follows the order of cluster formation when such data point forms a singular cluster or is merged with others to form a composite cluster. In this example, following the formation of clusters from 1 to 8, we acquire the natural orders shown in sub-table 6.1b as: $\textcircled{a}=0, \textcircled{b}=1, \textcircled{c}=3, \textcircled{d}=4, \textcircled{e}=2$. This is the same order in base 0 as acquired previously using visual inspection of the dendrogram.

6.4.3 Collaborative labelling

In *Koe*, each user has their own user space to perform labelling according to their expertise while sharing the same set of syllables. Normally this could prevent collaboration as their labelling results are isolated from other experts. *Koe* solves this issue by allowing labelling data to be viewed and shared between members of a database.

Figure 6.6 shows two control panels where these action can take place. In *Unit table* page, the user can choose to view the same unit table but with other team member's label data instead



(a) View label data by other team members

Saved versions			
	Backup type	Forma...	Database
<input type="radio"/>	segmentation	4	Bellbird_TMI
<input type="radio"/>	labels	4	Bellbird_TMI
<input type="radio"/>	segmentation	4	Bellbird_TMI
<input type="radio"/>	labels	4	Bellbird_TMI
<input type="radio"/>	labels	4	Bellbird_TMI
<input type="radio"/>	labels	4	Bellbird_TMI

Load ZIP Load selected Delete selected

(b) Import label data from other team members

Figure 6.6: Control panels where label data from other team members can be viewed in *Unit table* (Sub-figure 6.6a) or imported (Sub-figure 6.6b) via a saved file in *Database management*

of their own. This requires no special permission, however in this mode, editing is disabled regardless of the permission the user is granted. Label data automatically updates when the other team member makes a modification. In order to make modification, for example to correct mistakes, label data from other team member must first be copied into the current user's workspace. This functionality is provided in *Database management* page and requires at least *Import data* permission and a ZIP file containing the label data, which can only be acquired by asking the author of the label set to make a save. This viewing/sharing process might be more convoluted than team members sharing their login, however it is the way that *Koe* honours the intellectual properties of its users. It is worth mentioning that in *Koe*, the user owns their data in all forms, despite the fact that *Koe* is free.

6.5 Case study: Validating classification with independent labelling

In this section I present a case study to demonstrate the usefulness of *Koe* in multi-expert labelling and label validation. In collaboration with Dr. Wesley Webb, we harnessed the citizen science potential of *Koe* to evaluate inter-observer reliability, using 74 judges (to our knowledge the largest number of judges yet used). From our labelled dataset of 22,000 bellbird syllables,

we constructed a subset database of 200 syllables in *Koe*. The 200 syllables consisted of 18 label classes from two populations (Tawharanui and Tiritiri Matangi), with 3-20 of each class, including 4 “other” syllables that did not match the exemplars. (The classes and numbers of each class were chosen randomly.) The 74 judges (all naïve to *Koe*, to spectrograms in general, and to bellbird song) created *Koe* accounts and were granted online access to the database. Judges were asked to label the syllables using the unit table by comparing against labelled class exemplars in a separate tab. They were told there were 3–20 of each class, and 3–20 “other” syllables that did not match the exemplars. Each judge worked independently to label the set of syllables over a 1–2 hour period. Afterwards, labels from all judges were retrieved and compiled. For each of the 200 syllables, the percentage of judges whose label matched our own label was calculated: average 89.6%; median 95.6%. This is a high degree of agreement for inter-observer reliability studies (Nelson, Hallberg, et al. 2004; Parker et al. 2012), lending validity to our manual classification.

6.6 Conclusions

In this chapter I present different ways *Koe* facilitates rapid manual classification. This is arguably one of the most powerful tools that *Koe* has to offer. It is unique to *Koe* and does not exist in any existing software solutions for bioacoustics processing. In chapter 7 that follows, I present another tool that is unique to *Koe*: sequence analysis.

7.1 Introduction

Syntax refers to the set of rules governing song structure (how individual vocalisation units are strung together to form a sequence) (Robert F. Lachlan et al. 2013). It can be used as the primary basis for species recognition when acoustic properties at syllable-level are insufficient, i.e. when there are multiple birds in the same habitat that share similar vocalisation characteristics. An example of this is the New Zealand bellbird (*Anthornis melanura*) and the Tui (*Prosthemadera novaeseelandiae*), these two species make similar syllables that can be hard to tell apart without hearing the whole song.






Most birdsong has limited set of syntax rules which are primitive compared to human languages (Berwick et al. 2011). Whereas syntax of a human language is a very broad term that includes word order, grammatical relations, hierarchical sentence structure, subject-verb agreement, etc, syntax of birdsong is simply the temporal arrangements of vocalisation units (e.g. syllables) that appear with certain frequency in the repertoire. There is much debate about the complexity of syntax birds recognise. Kershenbaum, Bowles, et al. (2014) studies several species of animal and found that they can only recognise the repetition of syllables. In contrast, C. K. Adi (2008) and Katahira et al. (2011) found that Bangalese finch syntax exhibits high-order context-dependency (the meaning of one syllable depends on what comes before it). This finding was later supported by an analysis of the auditorial response in Bengalese finches brain (Abe and Watanabe 2011), but Beckers et al. (2012) argues that the claim is premature and the results

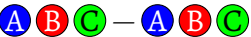



can be explained by simple acoustic similarity matching. *Koe* caters to this wide variation of syntax models by focusing on the discovery of repeated subsequences, where a subsequence can be a string of the same syllable or a group of syllables that frequently follow each other in the same order.

7.2 Syntax discovery via sequence structure

In an extensive review of acoustics sequences in animals, Kershenbaum, Blumstein, et al. (2016) generalised from previous studies six paradigms for information encoding using sequence structure. Table 7.1 shows five paradigms that can be found in songs of one individual. Paradigm *overlapping* is specific to duets between two individuals whereas *Koe* only analyses on individual basis, therefore it is omitted here.

Table 7.1: Five paradigms for individual-level information encoding using sequence structure
Kershenbaum, Blumstein, et al. (2016)

Type	Description	Visualisation
Repetition	Single unit repeated more than once	
Diversity	A number of distinct units are present. Order is unimportant	
Combination	Set of units has different information from each unit individually. Order is unimportant.	
Ordering	Set of units has different information from each unit individually. Order is important	
Timing	Timing between units (often between different individuals) conveys information	

Koe further refined these five paradigms by considering that the order of units are important, thus combining paradigms *Diversity*, *Combination* and *Ordering* into one which I simply call *Subsequence*. What Kershenbaum, Blumstein, et al. (2016) didn't address, however, is that diverse sequences can also be repeated, often with certain time gap between each repeat, i.e. the *Subsequence*  contains three repeats of  with different time gaps in between, which could be perceived as distinct from a single *Subsequence*  .

Koe provides several ways to analyse sequence structure. For a manual process, the user can have all song sequences that exist in the database displayed in a list and use *Koe*'s filtering func-

tion to narrow the list down to only those which contain a sequence. *Koe* also provides an automated process to automatically discover subsequences that appear at high frequency. Finally, the results of this automated process are visualised using directed graph to aid visual inspection.

7.2.1 Manual examination of subsequence via filtering

Filter `sequence:"trill"- "dragon"`

Filename	<input type="checkbox"/>	Sequence
TMI_2013_03_26	<input type="checkbox"/>	▶ Stutter Waah(magpie) Pipe(B[6]) Down(carunlock) Upsqueak_Flatsqueak Trill Dragon Moustache
TMI_2013_05_01	<input type="checkbox"/>	▶ Stutter Waah(magpie) Pipe(B[6]) Down(carunlock) Upsqueak_Flatsqueak Trill Dragon Moustache
TMI_2013_04_29	<input type="checkbox"/>	▶ Stutter Waah(magpie) Pipe(B[6]) Down(carunlock) Upsqueak_Flatsqueak Trill Dragon Moustache
TMI_2013_05_01	<input type="checkbox"/>	▶ Stutter Waah(magpie) Pipe(B[6]) Down(carunlock) Upsqueak_Flatsqueak Trill Dragon Moustache
TMI_2013_04_29	<input type="checkbox"/>	▶ Stutter Waah(magpie) Pipe(A#[6]) Down(carunlock) Upsqueak_Flatsqueak Trill Dragon Moustache
TMI_2013_04_30	<input type="checkbox"/>	▶ Stutter Stutter Waah(rough) Shrieky Trill Dragon Pipe_Downsqueak Moustache
TMI_2013_04_30	<input type="checkbox"/>	▶ Stutter Stutter Waah(rough) Shrieky Trill Dragon Moustache
TMI_2013_04_10	<input type="checkbox"/>	▶ Stutter Stutter Waah(rough) Shrieky Trill Dragon Moustache
TMI_2013_03_26	<input type="checkbox"/>	▶ Stutter Stutter Waah(rough) Pipe(B[6]) Down(carunlock) Upsqueak_Flatsqueak Trill Dragon Moustache
TMI_2013_03_26	<input type="checkbox"/>	▶ Stutter Stutter Waah(rough) Pipe(B[6]) Down(carunlock) Upsqueak_Flatsqueak Trill Dragon Moustache

Figure 7.1: In *View all songs* page, each segmented song is displayed in one row as a sequence of syllables. Manual sequence analysis can be carried out by applying a filter. Here, the filter `sequence:"trill"- "dragon"` is applied to narrow down only songs that contain *Trill* followed by *Dragon* (highlighted).

Koe displays all song sequences in *View all songs* page. Each song and the sequence of syllables (ideally already labelled) are displayed on one row. The user can narrow the list down to show only songs that contain a certain subsequence of unit labels. Figure 7.1 shows an example of this operation, where the filter is `sequence:"trill"- "dragon"`. The filter is case-insensitive and based on regular expression (Regex, JavaScript flavour), thus it is possible to filter the table based on their naming pattern; the user does not need to provide the exact match for their subsequence. For example, if all "pipe"-like syllables (those that have monotonic frequency content, thus appear on the spectrogram like a pipe) have name prefixed with "Pipe", a filter `sequence:"trill"- "Pipe.*"` will show songs that have sequence that have a *Trill* followed by any syllable that starts with *Pipe*, such as *Trill*→*PipeLow*;→*PipeHigh* etc.

7.2.2 Automated subsequence discovery using N-gram

N-gram is the general term for a technique used in modelling sequence that obeys Markovian properties, including human speech and birdsongs, where in practice N is a predefined number and is the length of the sequence that can be modelled by the Markov process. When N is 2, the model is called bigram, where the probability of each unit depends solely on the unit that comes before it, thus demonstrating first-order Markov properties. Similarly, 3-gram also called tri-gram, is the model where the probability of each unit depends on the previous two units, and so forth. Table 7.2 shows an example of subsequence extracted using N-gram with N ranges from 1 to 3.

Table 7.2: Subsequences that can be extracted by N-gram models. The original sequence is shown in the first row when $N=\infty$. Separated syllables are subsequences of the original sequence when $N=1$. Subsequent rows shows subsequences that can be extracted using bigram ($N=2$) and trigram ($N=3$)

N	Subsequence
∞ (original)	A B C D
1 (separate syllables)	A B C D
2 (bigram)	A B B C C D
3 (trigram)	A B C B C D

Bigram is widely used to study the repertoire complexity of a species, such is the Australian pied butcherbird (*Cracticus nigrogularis*) (Janney et al. 2016); also see Smith (2014) for several species including four birds. Specific applications of bigram and trigram to model sequence structure include Van Heijningen et al. (2009), Tsai and Xue (2014), Janney et al. (2016), Mathur and Kumar (2017). N-gram is intuitive and simple to implement. In five paradigms shown in Table 7.1, using N-gram can discover *Repetition*, *Combination* and *Ordering*. However, N-gram does not take into account the time gap between units. Moreover, N-gram cannot discover any pattern of subsequence that is longer than N units. Using a combination of N-grams with N goes from 1 to infinity can solve this issue, however this increases computational cost substantially.

N-gram was initially built into *Koe*, however it is now available only to advance users who can access the backend (via Docker or source code). This is because *Koe* now uses SPADE (Sequential Pattern Discovery using Equivalence classes) (Zaki 2001) - a more sophisticated algorithm that alleviates the aforementioned limitations.

7.2.3 Automated subsequence discovery using SPADE

The problem statement of sequential pattern mining was first proposed by Agrawal and Srikant (1995) and since then it has been an actively researched topic for more than two decades, arguably due to its practical application for large retail businesses. Barcode technology and computerised Point Of Sales (POS) have made it possible to collect and store massive amount of sales data, a.k.a. basket data. A record of basket data contains the number of items a customer bought, as well as timestamp and sometimes customer ID, particularly when a customer uses a smart loyalty card or credit cards. Soon, retailers started to explore ways to mine this data to study customer behaviour such as "x% of people who buy item A also buy item B" or "x% of people who buy item A will buy item B within the next n transactions", in order to create item bundles or run personalised promotions. The similarity between sequential patterns in shopping behaviour and those in bioacoustics is obvious. The same data mining technique can apply to explore patterns such as "there is an x probability that when syllable A is vocalised, syllable B will follow". When applied to birdsongs, a syllable can be considered an item and a group of syllables with small time gap in between can be considered one transaction.

SPADE is one of the newest algorithms proposed for sequential pattern mining. It is reported to be several orders of magnitude faster than previously proposed algorithms by utilising temporal joins along with efficient lattice search techniques. In addition, SPADE also provides ways to account for the time gap between transactions, effectively allowing two transactions that occur within a threshold to be merged into one. This enhancement proves useful to study vocalisation patterns when time gaps between syllables need to be taken into account.

To explain the use of SPADE for sequence analysis, consider a segmented and labelled song sequence an ordered list of x acoustic units denoted as $A \rightarrow B \rightarrow C \rightarrow \dots \rightarrow X$. A sequence rule has two parts: a left and right side. The rule states that when the left side occurs, the right side follows: [Leftside] \Rightarrow Rightside. For example, the rule $[A \rightarrow B] \Rightarrow C$ states that when the sequence $[A \rightarrow B]$ occurs, C comes next. The left side can be a sequence of any length, but in our implementation the right side is a single unit (see <https://github.com/fzyukio/koe/wiki>, Mine Sequence Structure).

The SPADE algorithm calculates three parameters which combine to indicate the credibility of sequence rules. These parameters are *Support*, *Confidence* and *Lift*. Their meanings are de-

scribed in Table 7.3. Credible rules have a large confidence factor, a large level of support and a value of lift greater than one, as defined below using the example rule, $[A \rightarrow B] \Rightarrow C$.

Table 7.3: Three parameters calculated by SPADE for each sequence

Parameter	Meaning
Support	The proportion of songs in the database that contain the entire sequence $A \rightarrow B \rightarrow C$ at least once.
Confidence	The proportion of those songs containing $A \rightarrow B$ that also contain $A \rightarrow B \rightarrow C$.
Lift	A measure of the strength of the association relative to chance. Lift is equal to the confidence of the rule, divided by the proportion of songs containing the right side. Thus it gives the ratio of (i) the proportion of songs with $A \rightarrow B$ that transition to C , versus (ii) the proportion of songs expected to contain $A \rightarrow B \rightarrow C$ by chance association of $A \rightarrow B$ and C .

To demonstrate how these parameters are calculated, consider the ten songs shown in Figure 7.1 to be a population of songs. The rule $[\text{Stutter} \rightarrow \text{Waah}(\text{magpie})] \Rightarrow \text{Pipe}(\text{B6})$ has a support of 0.4 since the entire sequence occurs in four of the 10 songs. The rule has a confidence of 0.8, because in four of the five songs that contain $\text{Stutter} \rightarrow \text{Waah}(\text{magpie})$, the transition to $\text{Pipe}(\text{B6})$ occurs. The proportion of songs with $\text{Pipe}(\text{B6})$ is 0.6, so the lift of this rule is $0.8/0.6 = 1.33$. That is, the association $[\text{Stutter} \rightarrow \text{Waah}(\text{magpie})] \Rightarrow \text{Pipe}(\text{B6})$ occurs in 1.33 times as many songs as expected by chance association of $[\text{Stutter} \rightarrow \text{Waah}(\text{magpie})]$ and $\text{Pipe}(\text{B6})$.

The *Mine sequence structure* under menu **Syntax analysis** provides a table of subsequences and their SPADE parameters for the user's selected database. Table 7.2 shows several subsequences extracted from the database *Bellbird_TMI*. Similar to other tables, the user can sort any column or combination of columns in ascending or descending order, as well as filtering the table using certain criteria to find subsequences of interest. In the given example, using filter `support:>0.2; chainlength:>1`, the results shows 8 subsequences that occur in 20% of the entire repertoire. The table is further sorted by *lift*, such that first sequence *Cough(TMI+click)→Downsqueak(hook)* is the one that has highest *lift*.

Chain length	Trans...	Support	Confidence	Lift	Association Rule
2	127	0.24	0.81	3.13	Cough(TMI+click) => Downsqueak(hook)
4	110	0.21	0.81	1.71	Stutter => Stutter => Stutter => Stutter
3	185	0.35	0.77	1.64	__PSEUDO_START__ => Stutter => Stutter
2	191	0.36	0.77	1.62	Stutter => Stutter
3	136	0.26	0.71	1.51	Stutter => Stutter => Stutter
4	128	0.24	0.69	1.46	__PSEUDO_START__ => Stutter => Stutter => Stutter
2	239	0.45	0.45	0.96	__PSEUDO_START__ => Stutter
2	135	0.26	0.54	0.54	Dragon => __PSEUDO_END__

Figure 7.2: Table of subsequences and their SPADE parameters in *Mine sequence structure* page. The subsequences are mined from the database *Bellbird_TMI*. Table is sorted descendingly by lift and filtered to show only subsequences that have two more syllables and have at least 20% support. `__PSEUDO_START__` and `__PSEUDO_END__` are not syllables, but represent the start and end of a song.

7.2.4 Analysis via visualisation with network models

Networks have long been used in cognitive science (see a review by Baronchelli et al. (2013)) to study the complex structure and dynamics of cognitive and behavioral processes. This line of study first found its way into the study of birdsong through the pioneer work of Sasahara et al. (2012), where each phrase type (similar to the notion of syllable in this thesis) can be viewed as one node in a graph. The transitions from one syllable type to another can be represented by a connection between adjacent nodes. Figure 7.3 shows an example of a graph produced for *Koe* users. Each node is depicted as a circle, the colour of a node indicates the prevalence of the syllable it represents (the darker the more prevalent). The arrows show the direction of the connection between nodes, the thickness and colour of an arrow indicates the strength of a connection (the thicker and darker, the higher the lift is). The ovals on the top left of a node indicates self-connection, or repetitiveness of the syllable. Red and blue lines connect a node with the "start of song" and "end of song" pseudo node. In this example, we can observe that songs often start with a Stutter repeated several times, followed by a *Waah(rough)*, *Stutter(cheet)*, or *Downcurve(doublevoiced)_Chiup*. There is a strong link between *Downcurve(doublevoiced)_Chiup*, *Stepdown_mid* and *Pipe(D#[7])*, meaning that the three syllables sequence is frequently seen in the repertoire.

Networks are useful for this line of study because they provide a high level overview of the

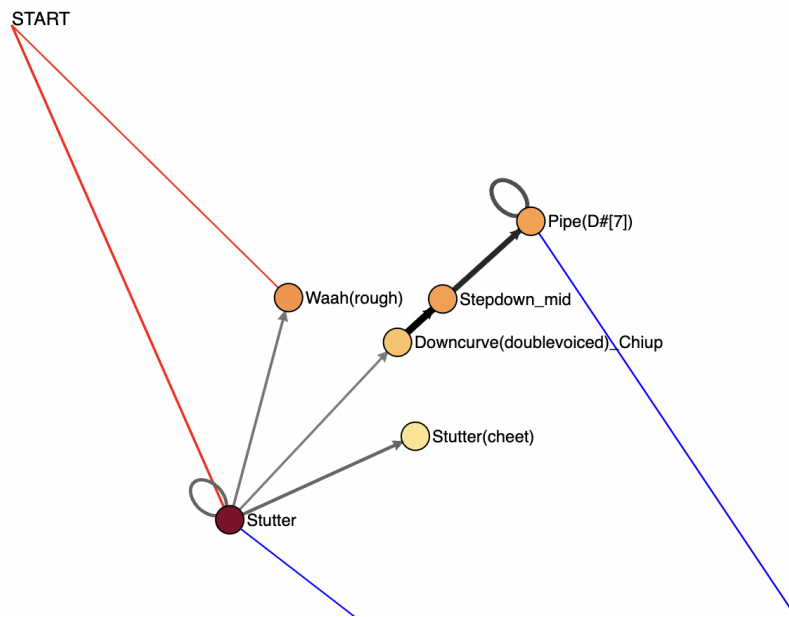


Figure 7.3: An example of a graph for a subset of syllables in *Koe*.

entire repertoire that can be visualised in one graph. Features extracted from the networks can also be used for qualitative comparison between repertoires. Examples of recent studies using networks are: Cody et al. (2015) who studied phrase sharing between individuals of the California thrashers (*Toxostoma redivivum*), Zsebők et al. (2020) studied how song sequences can encode individual's specific characteristics in male collared flycatchers (*Ficedula albicollis*). Beyond individuals, networks are also used to study song structures of one species, such as by Opaev (2016) (studying the Grey-crowned Warbler *Seicercus tephrocephalus*), or compare song structures of different populations of a species, such as Roach et al. (2012) (studying Hermit Thrush *Catharus guttatus*).

7.3 Implementation

7.3.1 SPADE

The original implementation of SPADE is in C++, which is available at <https://github.com/zakimjz/SPADE>. At the time that SPADE was first contemplated to be used in *Koe*, there was no Python implementation. There exists a package (Buchta et al. 2020) for the R language that provides similar functionality of SPADE, which in theory can be called from Python, however there are many disadvantages to this approach. First, R needs to be installed together with Python on the server and in the Docker image, making the process much more complicated than hav-

ing Python alone. Second, Python code needs to provide input to R and retrieve the results from R in the form of a text file, which severely degrades the performance when the input is large. To overcome this issue, I wrote a Python wrapper using Cython that can make calls and retrieve direct result from C++ code. The original implementation by Zaki was also rewritten entirely to enable memory sharing, as well as fixing various bugs related to memory management. This way the SPADE component can be preloaded and stay on the memory indefinitely, enabling the *Mine sequence structure* page to operate extremely fast. On a large database such as [Bellbird_TMI](#), it takes less than a second to mine and render all subsequences.

The rewritten C++ code is publicly available at <https://github.com/fzyukio/cspade-full>, and the Python wrapper is at <https://github.com/fzyukio/python-cspade>. Both versions were written to be compatible with three main operating systems and can be compiled without incident as long as there is a suitable C++ compiler (GCC on Unix/Linux, Clang on Mac and Microsoft Visual C++ on Windows). `python-cspade` is also provided as an installable package (available at <https://pypi.org/project/pycspade/>) for any python project using PIP, which is also the method of installation in *Koe*. Advanced users with access to the Docker image or source code do not need to compile SPADE separately, as it is already built-in to the installation process.

7.3.2 Networks

Koe's networks visualisation is implemented using the force-directed graph model in the D3.js library (Bostock et al. 2011). Force-directed graphs are widely used to simulate molecular dynamics such as the gravity or repellent forces between particles. Although birdsong syllables don't have these dynamics, the force of the network is still useful to position the syllables aesthetically. In *Koe*, the force is proportionate to the lift of a link, such that a strong lift produces a strong force, thus the syllables at both ends of a link are pulled closer together. In order to prevent collision, nodes also have an inherent repellent force similar to how two co-sign particles repel each other.

Data for the graph comes from two sources: all single-syllable subsequences are used to construct individual nodes, and all two-syllable subsequences found in the SPADE table are used to create links between connected nodes. Lift and supports are used to initialise the pull and repellent forces for each link. Next, nodes and links are fetched into `d3.forceSimulation` to

draw circles (visualisation of nodes) and arrows (visualisation of links). Finally, the pull and repellent forces are applied for the nodes to rearrange themselves. The graph typically reaches equilibrium in a few seconds, when the forces balance out and the nodes stop moving. The graph automatically updates when the table is filtered, giving the user freedom to analyse any subset of subsequences. This feature is particularly useful when the repertoire contains a large number of distinct syllables, which can make the graph crowded. Users can interact with the graph by using the mouse to move nodes around to rearrange the graph in a different equilibrium. This is useful if the nodes are too crowded at the initial stable state.

Koe provides a control panel on the left side of *Mine sequence structure* page through which the user can change the initial parameters of the graph as they see fit. In addition to forces between nodes, the graph in *Koe* also contains three external forces, exerted through three pseudo nodes: the pseudo start node at the beginning of a song, the pseudo end node at the end of a song, and a centre node, situated at the top left, bottom right, and centre of the graph, respectively. The pseudo start and end nodes can be used as part of the analysis, e.g. which syllables have strong tendency to come at the beginning/end of a song, while the centre node is simply to position the graph better. The other controls such as repulsion, distance can be used to achieve better node separation in cases where there are a small number of syllables tightly linked together or vice versa, the default parameters in this case might lead to a graph that is too centric, or too separated.

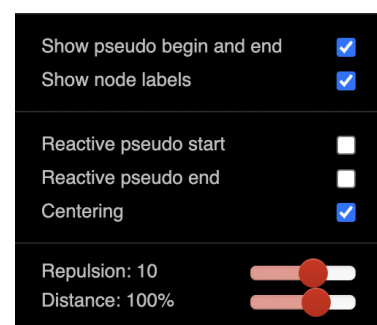


Figure 7.4: Control panel in *Mine sequence structure* page to change graph parameters.

7.4 Case studies: Evaluating song structure in NZ bellbirds

To analyse sequence structure in detail, *Mine sequence structure* shows the strength of association between unit classes using the SPADE algorithm (Zaki 2001), and produces network visualisations. We used this to compare the sequence structure of male and female bellbird song on Tiritiri Matangi Island, in the Hauraki Gulf, Auckland, New Zealand.

Table 7.4: Key features of male and female song structure as extracted by the SPADE algorithm. Results for NZ bellbird males (389 songs comprised of 5305 syllables) and females (138 songs comprised of 1884 syllables) recorded on Tiri between February 2013 and December 2015. Rule length is how many units constitute the association. Song count is how many songs the association occurs in. See text for definitions of support, confidence and lift.

Rule length	Association rule	Song count	Support	Confidence	Lift
Male population					
1	<i>Dragon</i>	201	0.52	—	—
1	<i>Cough(TMI+click)</i>	156	0.40	—	—
1	<i>Downsqueak(hook)</i>	135	0.35	—	—
...					
2	<i>Cough(TMI+click) → Downsqueak(hook)</i>	126	0.32	0.81	2.33
2	<i>START → Stutter</i>	113	0.29	0.29	0.95
2	<i>START → Cough(TMI+click)</i>	90	0.23	0.23	0.57
2	<i>Dragon → END</i>	115	0.30	0.57	0.57
2	<i>Moustache → END</i>	46	0.12	0.81	0.81
2	<i>Cough(TMI+click) → END</i>	45	0.12	0.29	0.29
...					
3	<i>[Trill → Dragon] → Moustache</i>	32	0.08	0.86	5.9
...					
Female population					
1	<i>Stutter</i>	129	0.93	—	—
1	<i>Chiup</i>	82	0.59	—	—
1	<i>Downcurve(doublevoiced)</i>	50	0.36	—	—
...					
2	<i>START ⇒ Stutter</i>	125	0.91	0.91	0.97
2	<i>Stutter ⇒ Stutter</i>	117	0.85	0.91	0.97
2	<i>Stutter ⇒ END</i>	52	0.38	0.40	0.40
2	<i>Downcurve(doublevoiced) ⇒ Chiup</i>	49	0.36	0.98	1.65
...					
9	<i>[Stutter → Stutter → Stutter → Stutter → Stutter → v → Stutter → Stutter] ⇒ Stutter</i>	21	0.15	0.75	0.80
...					
11	<i>[Stutter → Stutter → Stutter → Stutter → Stutter → Stutter → Stutter → Stutter → Stutter → Stutter] ⇒ Stutter</i>	10	0.07	0.77	0.82

7.4.1 Using SPADE parameters

One-unit ‘associations’ show prevalence of syllable classes in the population. The three most prevalent classes for males are *Dragon*, *Cough(TMI+click)* and *Downsqueak(hook)*, occurring in 52%, 40% and 35% of male songs, respectively. The three most prevalent female classes are *Stutter*, *Chiup* and *Downcurve(doublevoiced)*, occurring in 93%, 59% and 36% of female songs, respec-

tively. The most prevalent male two-unit association, *Cough(TMI+click)* \Rightarrow *Downsqueak(hook)*, occurs in 32% of songs. It has a confidence of 0.81; i.e. when *Cough(TMI+click)* occurs, *Downsqueak(hook)* comes next in 81% of songs. It has a lift of 2.33; i.e. *Downsqueak(hook)* follows *Cough(TMI+click)* in 2.33 times the number of songs as expected by chance. Male song most often starts with *Stutter* (29% of songs) or *Cough(TMI+click)* (23%). It most often ends with *Dragon* (30%), *Moustache* (12%) or *Cough(TMI+click)* (12%). By contrast, for females, the importance of *Stutter* is apparent. Not only does *Stutter* start in 91% of female songs, it also repeats (i.e. *Stutter* \rightarrow *Stutter*) in 85% and ends in 38%. Furthermore, *Stutter* is commonly repeated many (11) times.

7.4.2 Using networks

Sexual dimorphism in two-unit sequences was examined at the population level with network diagrams with a threshold of support > 0.05 (Figure 7.5). The songs of both sexes contain some nodes with many connections; these unit classes (e.g. *Dragon* in males, *Stutter* in females) can be preceded/followed by many different classes. Both sexes also contain nodes which are always preceded/followed by a certain class; for example, in males *Waah(mid+A)* is always followed by *Dragon*, and in females *Peakwhistle(TMI)* is always preceded by *Stutter*. Both sexes sing repeated units, represented by looped lines. For males: *Dragon*, *Stutter*, and *Cough(TMI+click)*. For females: *Stutter*, *Stutter(wavey)*, *Pipe(D#[7]+)*, and *Pipe(E[7])*. A notable difference in the female network is the chain of strongly associated nodes: *Killem* \rightarrow *Pipe(Csh[7]+)* \rightarrow *Upsqueak_Flat-squeak(1)* \rightarrow *Downsqueak(twopart)* \rightarrow *Down_Click_Stepdown* \rightarrow *Stepdown(C[7]-C[6])*.

7.5 Conclusion

This chapter presents several methods of sequence analysis made available to the front-end user. This is typically the last step in an acoustics process and thus concludes the workflow that *Koe* was create to facilitate.

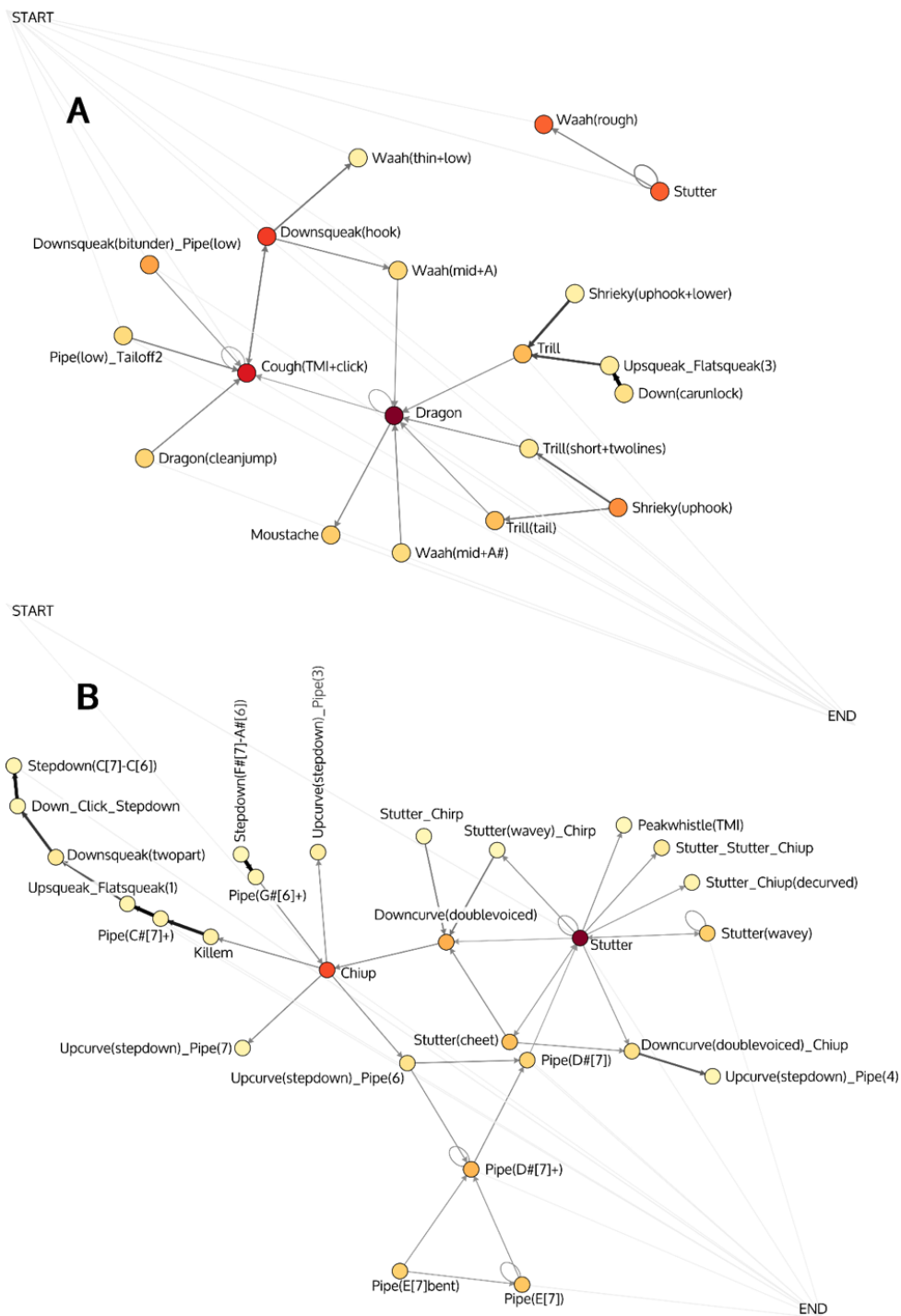


Figure 7.5: Network visualisations of male and female song sequence structure. The networks show two-unit associations for (A) male and (B) female NZ bellbird song on Tiri. The networks are produced from the SPADE data in Table 7.4. Key features of male and female song structure as extracted by the SPADE algorithm. Each node represents a unit class. Colours of the nodes represent frequency of occurrence, with darker, redder nodes being more frequently recorded classes. The arrows show directions of the association. Thickness and darkness of the arrows represent the strength of the association. The networks have been filtered to show only those associations that occur in more than 5% of songs. Classes that start or end in at least one instance are tethered to the START/END nodes, respectively, with thin grey lines.

Koe is an end-to-end solution for large-scale, high-resolution classification and analysis of animal vocalisations. It features tools for segmenting, comparing and classifying acoustic units and analysing sequence structure. Interactive ordination and unit tables provide a major advance in classification speed and robustness over existing methods. The use of SPADE and interactive force-directed graphs provides unmatched capability to perform sequence analysis and pattern discovery even for large repertoires of syllable types. I have presented several case studies to demonstrate the power of *Koe* at each step of the acoustics workflow. *Koe* was designed primarily for taxa/questions requiring acoustic classification; however, *Koe* is not limited to such cases. Where classification is not the aim, e.g. in animals where vocalisations show graded (non-discrete) variation, *Koe* is still a powerful tool for efficient extraction of measurements, which can be exported for analysis. Being web-based and accessible from any device, *Koe* is ideal for collaboration, teaching and citizen science.

8.1 Key contributions

Koe has been well-received after its release (see Fukuzawa, Webb, et al. (2020) for the official publication) and is now used by numerous researchers around the world (at the time this thesis is written, *Koe* is hosting more than 2000 databases). This thesis goes beyond the scope of documenting the design and functionality of a new software; each chapter in this thesis has aimed to answer key scientific questions: a) is it sensible to automated this process? b) If so, what

algorithm can be used to automated the process? c) If not, what is the reasoning for retaining manual process? By answering these questions, I have tried to establish from the computer science point of view a standard bioacoustics workflow that can be fine-tuned to the user's specific targets and needs.

The intended audience of this thesis is computer scientists and advanced users of *Koe* who wish to customise and expand existing functionality of *Koe* to suit their needs. Expandability has been the aim of *Koe* from its inception, as I have demonstrated throughout.

8.2 Future work

Advanced users of *Koe* will find an abundance of extra tools and features that *Koe* provides at the backend, such as automated segmentation, automatic classification, recursive labelling based on syntax analysis, various tools to import/export data to and from different sources. These tools have not been made available to the front-end users due to the limitations of in-browser computation, browser compatibility issues, but mostly due to the complexity of bringing machine learning to the browser. However, it is definitely possible to do so through libraries such as Tensorflow JS (Smilkov et al. 2019), Essentia.js (Joglar-ongay and Serra 2020). High performance at the client-side can be achieved by rewriting the computational core of *Koe* in WebAssembly (Webassembly Community Group 2020). It is worth mentioning that out of these future endeavours, client-side automatic segmentation has already been implemented in the most recent version of *Koe* and is undergoing quality assurance before releasing to the public.

Auto-encoder based feature extraction as presented in Section §5.3.2 has potential to greatly simplify one of the most challenging steps of the workflow by eliminating the needs for trial-and-error in feature selection. The proof of concept as presented in this thesis proves very promising, though considerable work is needed to incorporate auto-encoder into the feature extraction step.

8.3 Data availability

All *Koe* users have automatic access to the [Bellbirds_Case_Study](#) database analysed in this thesis. A user manual and step-by-step tutorial to reproduce our analyses is available at koe.io.ac.nz.

- Abe, K., & Watanabe, D. (2011). Songbirds possess the spontaneous ability to discriminate syntactic rules. *Nature neuroscience*, 14(8), 1067–1074 (cited on page 99).
- Acevedo, M. A., Corrada-Bravo, C. J., Corrada-Bravo, H., Villanueva-Rivera, L. J., & Aide, T. M. (2009). Automated classification of bird and amphibian calls using machine learning: A comparison of methods. *Ecological Informatics*, 4(4), 206–214 (cited on pages 68, 75).
- Adi, C. K. (2008). *Hidden Markov Model Based Animal Acoustic Censusing : Learning From Speech Processing* (Doctoral dissertation). Marquette University. (Cited on page 99).
- Adi, K., Johnson, M. T., & Osiejuk, T. S. (2010). Acoustic censusing using automatic vocalization classification and identity recognition. *The Journal of the Acoustical Society of America*, 127(May 2013), 874–883 (cited on pages 43, 70).
- Agrawal, R., & Srikant, R. (1995). Mining sequential patterns. *Proceedings - International Conference on Data Engineering*, 3–14 (cited on page 103).
- Anderson, M. E., & Conner, R. N. (1985). Northern Cardinal Song in Three Forest Habitats in Eastern Texas. *The Wilson Bulletin*, 97(4), 436–449 (cited on page 88).
- Bagshaw, P. C., Hiller, S. M., & Jack, M. A. (1993). Enhanced pitch tracking and the processing of f0 contours for computer aided intonation teaching. *Proc. Eur. Conf. Speech Commun.*, (September), 1003–1006 (cited on page 34).
- Baker, M. C. [Myron C.], & Logue, D. M. (2003). Population differentiation in a complex bird sound: A comparison of three bioacoustical analysis procedures. *Ethology*, 109(3), 223–242 (cited on pages 68, 72, 74).
- Baptista, L. F. [Luis F.]. (1977). Geographic Variation in Song and Dialects of the Puget Sound White-Crowned Sparrow. *The Condor*, 79(3), 356–370 (cited on page 87).
- Baptista, L. F. [Luis F.], & King, J. R. (1980). Geographical Variation in Song and Song Dialects of Montane White-Crowned Sparrows. *Condor*, 82(3), 267–284 (cited on pages 4, 87).
- Barmatz, H., Klein, D., Vortman, Y., Toledo, S., & Lavner, Y. (2019). Segmentation and Analysis of Bird Trill Vocalizations. *2018 IEEE International Conference on the Science of Electrical Engineering in Israel, ICSEE 2018*, 1–5 (cited on pages 44, 45).
- Baronchelli, A., Ferrer-i-Cancho, R., Pastor-Satorras, R., Chater, N., & Christiansen, M. H. (2013). Networks in Cognitive Science. *Trends in Cognitive Sciences*, 17(7), 348–360 (cited on page 105).
- Baugh, A. T., Akre, K. L., & Ryan, M. J. (2008). Categorical perception of a natural, multivariate signal: Mating call recognition in túngara frogs. *Proceedings of the National Academy of Sciences of the United States of America* (cited on page 7).
- Beckers, G. J., Bolhuis, J. J., Okanoya, K., & Berwick, R. C. (2012). Birdsong neurolinguistics: Songbird context-free grammar claim is premature. *NeuroReport*, 23(3), 139–145 (cited on page 99).
- Berwick, R. C., Okanoya, K., Beckers, G. J. L., & Bolhuis, J. J. (2011). Songs to syntax: The linguistics of birdsong. *Trends in Cognitive Sciences*, 15(3), 113–121 (cited on pages 5, 36, 99).
- Bhatia, R., Seth, H., & Rajan, P. (2019). Feature learning for bird-call segmentation using phase based features. *2018 13th International Conference on Industrial and Information Systems, ICIIS 2018 - Proceedings*, (978), 67–71 (cited on page 44).

- Bicego, M., Murino, V., & Figueiredo, M. A. (2003). Similarity-based clustering of sequences using hidden markov models. *Lecture Notes in Artificial Intelligence (Subseries of Lecture Notes in Computer Science)*, 2734, 86–95 (cited on pages 71, 73).
- Bioacoustics Research Program. (2011). Raven Pro: Interactive Sound Analysis Software (Version 1.1). (Cited on page 6).
- Bogert, B. P., Healy, M. J. R., & Tukey, J. W. (1963). The quefrency analysis of time series for echoes: Cepstrum, pseudo-autocovariance, cross-cepstrum and saphé cracking. *Proceedings of the symposium on time series analysis*, 15, 209–243 (cited on page 69).
- Bostock, M., Ogievetsky, V., & Heer, J. (2011). D3 data-driven documents. *IEEE Transactions on Visualization and Computer Graphics*, 17(12), 2301–2309 (cited on page 107).
- Boulmaiz, A., Messadeg, D., Doghmane, N., & Taleb-Ahmed, A. (2016). Robust acoustic bird recognition for habitat monitoring with wireless sensor networks. *International Journal of Speech Technology*, 19(3), 631–645 (cited on pages 70, 71).
- Briggs, F., Lakshminarayanan, B., Neal, L., Fern, X. Z., Raich, R., Hadley, S. J. K., Hadley, A. S., & Betts, M. G. (2012). Acoustic classification of multiple simultaneous bird species: A multi-instance multi-label approach. *The Journal of the Acoustical Society of America*, 131(6), 4640 (cited on page 72).
- Briggs, F., Raich, R., & Fern, X. Z. (2009). Audio classification of bird species: A statistical manifold approach. *Proceedings - IEEE International Conference on Data Mining, ICDM*, 51–60 (cited on pages 71, 72).
- Brumm, H., & Slabbekoorn, H. (2005). Acoustic Communication in Noise. *Advances in the Study of Behavior*, 35(05), 151–209 (cited on page 4).
- Buchta, C., Hahsler, M., Diaz, D., Buchta, M. C., & Zaki, M. J. (2020). Package arulesSequences. (Cited on page 106).
- Cai, J., Ee, D., Pham, B., Roe, P., & Zhang, J. (2007). Sensor Network for the Monitoring of Ecosystem: Bird Species Recognition. *2007 3rd International Conference on Intelligent Sensors, Sensor Networks and Information*, 293–298 (cited on page 70).
- Chen, G., Parada, C., & Sainath, T. N. (2015). Query-by-example keyword spotting using long short-term memory networks. *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings, 2015-Augus*, 5236–5240 (cited on page 75).
- Chen, I. F., & Lee, C. H. [Chin Hui]. (2013). A Hybrid HMM/DNN approach to keyword spotting of short words. *Proceedings of the Annual Conference of the International Speech Communication Association, INTERSPEECH*, (August), 1574–1578 (cited on page 75).
- Chen, W. P., Chen, S. S., Lin, C. C., Chen, Y. Z., & Lin, W. C. (2012). Automatic recognition of frog calls using a multi-stage average spectrum. *Computers and Mathematics with Applications*, 64(5), 1270–1281 (cited on page 74).
- Chen, Z., & Maher, R. C. (2006). Semi-automatic classification of bird vocalizations using spectral peak tracks. *The Journal of the Acoustical Society of America*, 120(5), 2974 (cited on pages 43, 68, 74).
- Chiba, T., & Kajiyama, M. (1958). *The vowel: Its nature and structure* (Vol. 652). Phonetic society of Japan Tokyo. (Cited on page 38).
- Chou, C. H., & Ko, H. Y. (2011). Automatic birdsong recognition with MFCC based syllable feature extraction. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 6905 LNCS, 185–196 (cited on pages 46, 71, 89).
- Chou, C. H., & Liu, P. H. (2009). Bird species recognition by wavelet transformation of a section of birdsong. *UIC-ATC 2009 - Symposia and Workshops on Ubiquitous, Autonomic and Trusted Computing in Conjunction with the UIC'09 and ATC'09 Conferences*, 189–193 (cited on pages 46, 70).
- Chou, C. H., Liu, P. H., & Cai, B. (2008). On the studies of syllable segmentation and improving MFCCs for automatic birdsong recognition. *Proceedings of the 3rd IEEE Asia-Pacific Services Computing Conference, APSCC 2008*, 745–750 (cited on page 46).
- Chu, W., & Blumstein, D. T. (2011). Noise robust bird song detection using syllable pattern-based hidden Markov models, 345–348 (cited on page 72).
- Chung, Y. A., Wu, C. C., Shen, C. H., Lee, H. Y., & Lee, L. S. (2016). Audio Word2Vec: Unsupervised learning of audio segment representations using sequence-to-sequence autoencoder. *Proceed-*

- ings of the Annual Conference of the International Speech Communication Association, INTERSPEECH, 08-12-Sept, 765–769 (cited on page 75).
- Cody, M. L., Stabler, E., Sánchez Castellanos, H. M., & Taylor, C. E. (2015). Structure, syntax and “small-world” organization in the complex songs of California Thrashers (*Toxostoma redivivum*). *Bioacoustics*, 25(1), 41–54 (cited on page 106).
- Comon, P. (1994). Independent component analysis, a new concept? *Signal processing*, 36(3), 287–314 (cited on page 26).
- Cooley, J. W., & Tukey, J. W. (1965). An Algorithm for the Machine Calculation of Complex Fourier Series. *Mathematics of Computation* (cited on page 33).
- Dahlin, C. R., & Wright, T. F. (2012). Duet function in the yellow-naped amazon, *Amazona auropalliata*: Evidence from playbacks of duets and solos. *Ethology*, 118(1), 95–105 (cited on page 3).
- Deng, L., Seltzer, M., Yu, D., Acero, A., Mohamed, A., Hinton, G., & Way, O. M. (2010). Binary Coding of Speech Spectrograms Using a Deep Auto - encoder. *Interspeech*, (September), 1692–1695 (cited on page 48).
- Derégnaucourt, S., Mitra, P. P., Fehér, O., Pytte, C., & Tchernichovski, O. (2005). How sleep affects the developmental learning of bird song. *Nature*, 433(7027), 710–6 (cited on pages 4, 68, 89).
- der Walt, S., Schönberger, J. L., Nunez-Iglesias, J., Boulogne, F., Warner, J. D., Yager, N., Goullart, E., & Yu, T. (2014). scikit-image: image processing in Python. *PeerJ*, 2, e453 (cited on page 16).
- Dooling, R., Brittan-Powell, E., Lauer, A., Dent, M., & Noirot, I. (2005). The problem of frequency weighting functions and standards for birds. *The Journal of the Acoustical Society of America*, 118(3), 2018 (cited on page 42).
- Dooling, R. J. (2004). Audition: can birds hear everything they sing? *Nature's music* (pp. 206–225). Elsevier. (Cited on page 37).
- Dooling, R. J., Zoloth, S. R., & Baylis, J. R. (1978). Auditory sensitivity, equal loudness, temporal resolving power, and vocalizations in the house finch (*Carpodacus mexicanus*). *Journal of Comparative and Physiological Psychology*, 92(5), 867–876 (cited on page 42).
- Doupe, A. J., & Kuhl, P. K. [P K]. (1999). Birdsong and human speech: common themes and mechanisms. *Annual review of neuroscience*, 22, 567–631 (cited on page 34).
- Ehret, G., & Haack, B. (1981). Categorical perception of mouse pup ultrasound by lactating females. *Naturwissenschaften* (cited on page 7).
- Eimas, P. D., Siqueland, E. R., Jusczyk, P., & Vigorito, J. (1971). Speech perception in infants. *Science*, 171(3968), 303–306 (cited on page 3).
- Evangelista, T. L., Priolli, T. M., Jr, C. N. S., Angelico, B. a., & a.a. Kaestner, C. (2014). Automatic Segmentation of Audio Signals for Bird Species Identification. 2014 IEEE International Symposium on Multimedia, 223–228 (cited on page 74).
- Fagerlund, S., & Harma, A. (2005). Parametrization of inharmonic bird sounds for automatic recognition. 13th European Signal Processing Conference (EUSIPCO '05) (cited on page 74).
- Fagerlund, S. [Seppo]. (2004). Automatic Recognition of Bird Species by Their Sounds. *Department of Electrical and Communications Engineering \nLaboratory of Acoustics and Audio Signal Processing*, 56 (cited on page 74).
- Fagerlund, S. [Seppo]. (2007). Bird species recognition using support vector machines. *Eurasip Journal on Advances in Signal Processing*, 2007 (cited on pages 70–72, 74).
- Fodor, G. (2013). The Ninth Annual MLSP Competition: First place. *IEEE International Workshop on Machine Learning for Signal Processing, MLSP*, 1–2 (cited on pages 46, 70).
- Fox, E. J. S. (2008). *Call-independent identification in birds* (Doctoral dissertation). (Cited on pages 43, 71).
- Francisco, S., Slabbekoorn, H., & Smith, T. B. (2002). Habitat-Dependent Song Divergence in the Little Greenbul : an Analysis of Environmental Selection Pressures on Acoustic Signals. *Evolution; international journal of organic evolution*, 56(9), 1849–1858 (cited on page 4).
- Franzen, A., & Gu, I. Y. (2003). Classification of bird species by using key song searching: a comparative study. *IEEE International Conference on Systems, Man and Cybernetics.*, 1, 880–887 vol.1 (cited on pages 43, 68).

- Fukuzawa, Y., Marsland, S., Pawley, M., & Gilman, A. (2017). Segmentation of harmonic syllables in noisy recordings of bird vocalisations. *International Conference Image and Vision Computing New Zealand* (cited on page 47).
- Fukuzawa, Y., Webb, W. H., Pawley, M. D., Roper, M. M., Marsland, S., Brunton, D. H., & Gilman, A. (2020). Koe: Web-based software to classify acoustic units and analyse sequence structure in animal vocalizations. *Methods in Ecology and Evolution*, 11(3), 431–441 (cited on page 112).
- Gil, D., & Slater, P. J. B. [Peter J B]. (2000). Song organisation and singing patterns of the willow warbler, *Phylloscopus trochilus*. *Behaviour*, 137, 759–782 (cited on page 89).
- Goller, F., & Larsen, O. N. [Ole N.]. (1997). A new mechanism of sound generation in songbirds. *Proceedings of the National Academy of Sciences of the United States of America*, 94(December), 14787–14791 (cited on page 39).
- Graciarena, M., Delplanche, M., Shriberg, E., Stolcke, A., & Ferrer, L. (2010). Acoustic front-end optimization for bird species recognition. *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, 293–296 (cited on page 70).
- Greenwood, D. D. (1990). A cochlear frequency-position function for several species—29 years later. *Jasa*, 87(6), 2592–2605 (cited on page 38).
- Große Ruse, M., Hasselquist, D., Hansson, B., Tarka, M., & Sandsten, M. (2016). Automated analysis of song structure in complex birdsongs. *Animal Behaviour*, 112, 39–51 (cited on pages 45, 88).
- Gunasekaran, S., & Revathy, K. (2010). Content-Based Classification and Retrieval of Wild Animal Sounds Using Feature Selection Algorithm. *Machine Learning and Computing (ICMLC), 2010 Second International Conference on* (cited on page 70).
- Harma, A. [Aki]. (2003). Automatic identification of bird species based on sinusoidal modeling of syllables. *IEEE International Conference on Acoustics, Speech, and Signal Processing*, 5 (cited on pages 46, 48).
- Hermansky, H. (1990). Perceptual linear predictive (PLP) analysis of speech. *The Journal of the Acoustical Society of America*, 87(4), 1738–52 (cited on page 41).
- Hess, W. (1983). *Pitch determination of speech signals: algorithms and devices* (1st ed.). Springer-Verlag Berlin Heidelberg. (Cited on page 67).
- Hess, W. J. (1992). Pitch and voicing determination. *Advances in speech signal processing*, 3–48 (cited on page 67).
- Hirschberg, J., & Manning, C. D. (2015). Advances in natural language processing. *Science*, 349(6245), 261–266 (cited on page 4).
- Howard, R. (1999). *Principles of Animal Communication* (Vol. 45). (Cited on page 3).
- Inc., P. T. (2015). *Collaborative data science*. (Cited on page 93).
- Ito, K., Mori, K., & Iwasaki, S.-i. (1996). Application of dynamic programming matching to classification of budgerigar contact calls. *The Journal of the Acoustical Society of America*, 100(6), 3947–3956 (cited on page 68).
- Jancovic, P., Kokuer, M., Zakeri, M., & Russell, M. (2013). Unsupervised discovery of acoustic patterns in bird vocalisations employing DTW and clustering. *21st European Signal Processing Conference (EUSIPCO 2013)*, 1–5 (cited on page 68).
- Janney, E., Taylor, H., Scharff, C., Rothenberg, D., Parra, L. C., & Tchernichovski, O. (2016). Temporal regularity increases with repertoire complexity in the Australian pied butcherbird's song. *Royal Society Open Science*, 3(9) (cited on page 102).
- Jinnai, M., Boucher, N., Fukumi, M., & Taylor, H. (2012). A new optimization method of the geometric distance in an automatic recognition system for bird vocalisations. In *Société Française d'Acoustique* (Ed.), *Acoustics* (pp. 2439–2445). (Cited on page 45).
- Joglar-ongay, L., & Serra, X. (2020). *Essentia . Js : a Javascript Library for Music and Audio*. (May), 605–612 (cited on page 113).
- Jones, A. E., Ten Cate, C., & Bijleveld, C. C. (2001). The interobserver reliability of scoring sonagrams by eye: A study on methods, illustrated on zebra finch songs. *Animal Behaviour*, 62(4), 791–801 (cited on pages 8, 28).
- Jones, E., Oliphant, T., & Peterson, P. (2014). *SciPy: open source scientific tools for Python*. (Cited on pages 15, 80).

- Juang, C. F., & Chen, T. M. (2007). Birdsong recognition using prediction-based recurrent neural fuzzy networks. *Neurocomputing*, 71(1-3), 121–130 (cited on page 70).
- Kaewtip, K., Tan, L. N., Alwan, A., & Taylor, C. E. (2013). A robust automatic bird phrase classifier using dynamic time-warping with prominent region identification. *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, 768–772 (cited on pages 47, 53).
- Kaewtip, K., Tan, L. N., Taylor, C. E., & Alwan, A. (2015). Bird-phrase segmentation and verification: A noise-robust template-based approach. *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 758–762 (cited on page 47).
- Kahl, S., Stöter, F. R., Goëau, H., Glotin, H., Planqué, R., Vellinga, W. P., & Joly, A. (2019). Overview of BIRDCLEF 2019: Large-scale bird recognition in soundscapes. *CEUR Workshop Proceedings*, 2380, 9–12 (cited on page 43).
- Kamper, H., Wang, W., & Livescu, K. (2016). Deep convolutional acoustic word embeddings using word-pair side information. *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings, 2016-May*, 4950–4954 (cited on page 75).
- Katahira, K., Suzuki, K., Okanoya, K., & Okada, M. (2011). Complex sequencing rules of birdsong can be explained by simple hidden Markov processes. *PLoS ONE*, 6(9) (cited on page 99).
- Kedem, B. (1986). Spectral Analysis and Discrimination by Zero-Crossings. *Proceedings of the IEEE*, 74(11), 1477–1493 (cited on page 66).
- Kershenbaum, A., Blumstein, D. T., Roch, M. A., Backus, G., Bee, M. A., Bohn, K., Cao, Y., Carter, G., Căsar, C., Coen, M., Deruiter, S. L., Doyle, L., Edelman, S., Ferrer-i-cancho, R., Freeberg, T. M., Garland, E. C., Gustison, M., Harley, H. E., Huetz, C., ... Zamora-gutierrez, V. (2016). Acoustic sequences in non-human animals : a tutorial review and prospectus. *Biological Reviews*, 91(1), 13–52 (cited on pages 8, 28, 36, 100).
- Kershenbaum, A., Bowles, A. E., Freeberg, T. M., Jin, D. Z., Lameira, A. R., & Bohn, K. (2014). Animal vocal sequences: not the Markov chains we thought they were. *Proceedings of the Royal Society B: Biological Sciences*, 281(1792), 20141370 (cited on page 99).
- Kershenbaum, A., Déaux, É. C., Habib, B., Mitchell, B., Palacios, V., Root-Gutteridge, H., & Waller, S. (2018). Measuring acoustic complexity in continuously varying signals: how complex is a wolf howl? *Bioacoustics*, 27(3), 215–229 (cited on page 3).
- Kershenbaum, A., Ilany, A., Blaustein, L., & Geffen, E. (2012). Syntactic structure and geographical dialects in the songs of male rock hyraxes. *Proceedings of the Royal Society of London B: Biological Sciences*, 279(1470), 2974–2981 (cited on page 3).
- Kim, H. G., Moreau, N., & Sikora, T. (2006). *MPEG-7 Audio and Beyond: Audio Content Indexing and Retrieval*. (Cited on page 67).
- Kogan, J. A., & Margoliash, D. (1998). Automated recognition of bird song elements from continuous recordings using dynamic time warping and hidden Markov models: A comparative study. *The Journal of the Acoustical Society of America*, 103(4), 2185–2196 (cited on page 72).
- Koops, H. V., Balen, J. V., & Wiering, F. (2014). A Deep Neural Network Approach to the LifeCLEF 2014 Bird task. *CLEF2014 Working Notes*, 1180, 634–642 (cited on pages 46, 70, 74).
- Koops, H. V., Balen, J. V., & Wiering, F. (2015). Automatic Segmentation and Deep Learning of Bird Songs. *Experimental IR Meets Multilinguality, Multimodality, and Interaction*, 9283, 261–267 (cited on page 46).
- Köppl, C., Gleich, O., & Manley, G. A. (1993). An auditory fovea in the barn owl cochlea. *Journal of Comparative Physiology A*, 171(6), 695–704 (cited on page 38).
- Kuhl, P. K. [P. K.]. (2003). Human speech and birdsong: Communication and the social brain. *Proceedings of the National Academy of Sciences*, 100(17), 9645–9646 (cited on page 34).
- Kuhl, P. K. [Patricia K.]. (1989). On babies, birds, modules, and mechanisms: A comparative approach to the acquisition of vocal communication. *The comparative psychology of audition: Perceiving complex sounds*. (Cited on page 34).
- Kwan, C., Mei, G., Zhao, X., Ren, Z., Xu, R., Stanford, V., Rochet, C., Aube, J., & Ho, K. C. (2004). Bird classification algorithms: theory and experimental results. *Acoustics, Speech, and Signal Processing, 2004. Proceedings. (ICASSP '04). IEEE International Conference on*, 5, V–289–92 vol.5 (cited on page 70).

- Lachlan, R. F. [R F]. (2007). Luscinia: a bioacoustics analysis computer program. Version 1.0. *Computer program*. Retrieved from <https://github.com/rflachlan/Luscinia> on 21st September 2016 (cited on pages 6, 82).
- Lachlan, R. F. [Robert F.], Verzijden, M. N., Bernard, C. S., Jonker, P. P., Koese, B., Jaarsma, S., Spoor, W., Slater, P. J., & Ten Cate, C. (2013). The progressive loss of syntactical structure in bird song along an Island colonization chain. *Current Biology*, 23(19), 1896–1901 (cited on pages 29, 99).
- Lachlana, R. F., & Nowickia, S. (2015). Context-dependent categorical perception in a songbird. *Proceedings of the National Academy of Sciences of the United States of America* (cited on pages 5, 7).
- Lakshminarayanan, B., Raich, R., & Fern, X. Z. (2009). A Syllable-Level Probabilistic Framework for Bird Species Identification. *Eighth International Conference on Machine Learning and Applications, Proceedings*, 53–59 (cited on pages 45, 70, 75).
- Larsen, O. N. [Ole N], & Goller, F. (1999). Role of syringeal vibrations in bird vocalizations. (May) (cited on page 39).
- Lasseck, M. (2013). Bird song classification in field recordings: Winning solution for NIPS4B 2013 competition. *Proc. of int. symp. Neural Information Scaled . . .*, 1–6 (cited on pages 46, 72).
- Lasseck, M. (2015). Improved Automatic Bird Identification through Decision Tree based Feature Selection and Bagging. *Working notes of CLEF 2015 conference* (cited on pages 70, 72).
- Lasseck, M. (2016). Improving Bird Identification using Multiresolution Template Matching and Feature Selection during Training. *Working Notes of CLEF Conference* (cited on page 70).
- Lee, C. H. [Chang Hsing], Han, C. C., & Chuang, C. C. (2008). Automatic Classification of Bird Species From Their Sounds Using Two-Dimensional Cepstral Coefficients. *IEEE Transactions on Audio, Speech, and Language Processing*, 16(8), 1541–1550 (cited on page 43).
- Lee, C. H. [Chang Hsing], Lee, Y. K., & Huang, R. Z. (2006). Automatic recognition of birdsongs using mel-frequency cepstral coefficients. *Proceedings of the International MultiConference of Engineers and Computer Scientists*, 1(1), 17–23 (cited on pages 46, 70, 71).
- Levin, K., Henry, K., Jansen, A., & Livescu, K. (2013). Fixed-dimensional acoustic embeddings of variable-length segments in low-resource settings. *Asru*, 410–415 (cited on page 75).
- Li, T., Ogihara, M., & Li, Q. (2003). A comparative study on content-based music genre classification. *Proceedings of the 26th annual international ACM SIGIR conference on Research and development in information retrieval SIGIR 03*, 15(5), 282 (cited on page 68).
- Maas, A. L., Miller, S. D., O’Neil, T. M., Ng, A. Y., & Nguyen, P. (2012). Word-level Acoustic Modeling with Convolutional Vector Regression. *ICML WS on Representation Learning* (cited on page 75).
- Manley, G. a., Brix, J., & Kaiser, a. (1987). Developmental stability of the tonotopic organization of the chick’s basilar papilla. *Science (New York, N.Y.)*, 237(4815), 655–656 (cited on page 38).
- Marler, P. [P.]. (1970). Birdsong and speech development: could there be parallels? *American Scientist* (cited on page 34).
- Marler, P., & Slabbekoorn, H. W. (2004). Nature’s Music. *The Science of Birdsong*, 1, 513 (cited on page 3).
- Mathur, P., & Kumar, A. (2017). Ethological Data Mining of Bengalese Finch’s and White- Rumped Munia’s Song. *2017 13th International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery (ICNC-FSKD)*, 2306–2311 (cited on page 102).
- May, B., Moody, D. B., & Stebbins, W. C. (1989). Categorical Perception Of Conspecific Communication Sounds By Japanese Macaques. *Macaca Fuscata. Journal of the Acoustical Society of America* (cited on page 7).
- Mcfee, B., Raffel, C., Liang, D., Ellis, D. P. W., Mcvcar, M., Battenberg, E., & Nieto, O. (2015). librosa: Audio and Music Signal Analysis in Python. *PROC. OF THE 14th PYTHON IN SCIENCE CONF* (cited on pages 15, 16, 80).
- Mitrović, D., Zeppelzauer, M., & Breiteneder, C. (2006). Discrimination and Retrieval of Animal Sounds. *The 12th International Multi-Media Modelling Conference*, 00, 339–343 (cited on page 70).
- Mitrović, D., Zeppelzauer, M., & Breiteneder, C. (2010). Features for Content-Based Audio Retrieval. *Advances in Computers*, 78(10), 71–150 (cited on page 64).
- Nanayakkara, S. C., Chitre, M., Ong, S., & Taylor, E. (2007). Automatic Classification of Whistles Produced by Indo-Pacific Humpback Dolphins (*Sousa chinensis*). *OCEANS 2007 - Europe*, 1–5 (cited on page 75).

- Neal, L., Briggs, F., Raich, R., & Fern, X. Z. (2011). Time-Frequency Segmentation of Bird Song in Noisy Acoustic Environments. *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, 2012–2015 (cited on pages 47, 53).
- Nelson, D. A., Hallberg, K. I., & Soha, J. A. (2004). Cultural evolution of Puget Sound white-crowned sparrow song dialects. *Ethology*, 110(11), 879–908 (cited on pages 8, 28, 87, 98).
- Nelson, D. A., & Marler, P. [Peter]. (1989). Categorical perception of a natural stimulus continuum: Bird-song. *Science* (cited on page 7).
- Nerbonne, J., & Heeringa, W. (2001). Computational Comparison and Classification of Dialects. *Dialectologia et Geolinguistica*, 2001, 69–83 (cited on page 4).
- Nowicki, S. (1987). Vocal tract resonances in oscine bird sound production: evidence from birdsongs in a helium atmosphere. *Nature*, 325, 53–55 (cited on page 39).
- Ondracek, J. M., & Hahnloser, R. H. R. (2013). Advances in Understanding the Auditory Brain of Songbirds. *Insights from comparative hearing research* (pp. 347–388). Springer New York. (Cited on page 45).
- Opaev, A. (2016). Relationships between repertoire size and organization of song bouts in the Grey-crowned Warbler (*Seicercus tephrocephalus*). *Journal of Ornithology*, 157(4), 949–960 (cited on page 106).
- O’Shaughnessy, D. (2000). *Speech Communications: Human and Machine*. IEEE press, Newyork, 367–433 (cited on page 40).
- Ouattara, K., Lemasson, A., & Zuberbühler, K. (2009). Campbell’s monkeys concatenate vocalizations into context-specific call sequences. *Proceedings of the National Academy of Sciences of the United States of America*, 106(51), 22026–22031 (cited on page 3).
- Pampalk, E., Rauber, A., & Merkl, D. (2002). Content-based organization and visualization of music archives. *Proceedings of the tenth ACM international conference on Multimedia - MULTIMEDIA ’02*, 570 (cited on page 41).
- Papadopoulos, T., Roberts, S., & Willis, K. (2015). Detecting bird sound in unknown acoustic background using crowdsourced training data, 1–8 (cited on page 45).
- Parker, K. A., Anderson, M. J., Jenkins, P. F., & Brunton, D. H. (2012). The effects of translocation-induced isolation and fragmentation on the cultural evolution of bird song. *Ecology Letters*, 15(8), 778–785 (cited on pages 8, 28, 98).
- Pearson, K. (1901). LIII. On lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin philosophical magazine and journal of science*, 2(11), 559–572 (cited on page 26).
- Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, É. (2012). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* (cited on page 16).
- Podos, J., Peters, S., Rudnicki, T., Marler, P., & Nowicki, S. (1992). The Organization of Song Repertoires in Song Sparrows: Themes and Variations. *Ethology*, 90(2), 89–106 (cited on pages 88, 89).
- Potamitis, I. (2015). Unsupervised dictionary extraction of bird vocalisations and new tools on assessing and visualising bird activity. *Ecological Informatics*, 26(P3), 6–17 (cited on page 72).
- Pozzi, L., Gamba, M., & Giacoma, C. (2010). The use of artificial neural networks to classify primate vocalizations: A pilot study on black lemurs. *American Journal of Primatology*, 72(4), 337–348 (cited on page 74).
- Priyadarshani, N. (2016). *Wavelet-Based Birdsong Recognition for Conservation* (Doctoral dissertation). Massey University. (Cited on page 35).
- Python Software Foundation. (2017). NumPy — NumPy. (Cited on page 15).
- Qian, K., Zhang, Z., Ringeval, F., & Schuller, B. (2015). Bird sounds classification by large scale acoustic features and extreme learning machine. *2015 IEEE Global Conference on Signal and Information Processing (GlobalSIP)*, 1317–1321 (cited on pages 45, 74).
- Rabiner, L., & Sambur, M. (1974). An algorithm for determining the endpoint of isolated utterances. *American Telephone and Telegraph Company, THE DELL SYSTEM TECHNICAL JOCSBAL*, 54(February) (cited on page 45).
- Raju, N., Mathini, S., Lakshmi Priya, T., Preethi, P., & Chandrasekar, M. (2012). Identifying the population of animals through pitch, formant, short time energy - A sound analysis. *2012 Interna-*

- tional Conference on Computing, Electronics and Electrical Technologies, ICCEET 2012, 704–709 (cited on page 33).
- Ranjard, L. (2009). *Computational Biology of Bird Song Evolution* (Doctoral dissertation). Auckland University. (Cited on pages 47, 53).
- Ranjard, L., Anderson, M. G., Rayner, M. J., Payne, R. B., McLean, I., Briskie, J. V., Ross, H. A., Brunton, D. H., Woolley, S. M. N., & Hauber, M. E. (2010). Bioacoustic distances between the begging calls of brood parasites and their host species: A comparison of metrics and techniques. *Behavioral Ecology and Sociobiology*, 64(11), 1915–1926 (cited on pages 4, 68).
- Ranjard, L., & Ross, H. A. (2008). Unsupervised bird song syllable classification using evolving neural networks. *The Journal of the Acoustical Society of America*, 123(6), 4358–4368 (cited on pages 36, 45, 70, 72, 89).
- Riede, T., & Goller, F. (2010). Peripheral mechanisms for vocal production in birds - differences and similarities to human speech and singing. *Brain and Language*, 115(1), 69–80 (cited on page 34).
- Riede, T., Suthers, R. A., Fletcher, N. H., & Blevins, W. E. (2006). Songbirds tune their vocal tract to the fundamental frequency of their song. *Proceedings of the National Academy of Sciences of the United States of America*, 103(14), 5543–8 (cited on page 39).
- Roach, S. P., Johnson, L., & Phillmore, L. S. (2012). Repertoire composition and singing behaviour in two eastern populations of the Hermit Thrush (*Catharus guttatus*). *Bioacoustics*, 21(3), 239–252 (cited on page 106).
- Rothstein, S. I., & Fleischer, R. C. (2007). Vocal Dialects and Their Possible Relation to Honest Status Signalling in the Brown-Headed Cowbird. *The Condor* (cited on page 4).
- Ruiz-Muñoz, J. F., Orozco-Alzate, M., & Castellanos-Dominguez, G. (2015). Multiple instance learning-based birdsong classification using unsupervised recording segmentation. *IJCAI International Joint Conference on Artificial Intelligence, 2015-Janua(Ijcai)*, 2632–2638 (cited on page 72).
- Sahidullah, M., & Saha, G. (2012). Design, analysis and experimental evaluation of block based transformation in MFCC computation for speaker recognition. *Speech Communication*, 54(4), 543–565 (cited on page 70).
- Sainburg, T., Theilman, B., Thielk, M., & Gentner, T. Q. (2019). Parallels in the sequential organization of birdsong and human speech. *Nature Communications*, 10(1), 3636 (cited on page 3).
- Sarkar, A., Chabout, J., Macopson, J. J., Jarvis, E. D., & Dunson, D. B. (2018). Bayesian semiparametric mixed effects markov models with application to vocalization syntax. *Journal of the American Statistical Association*, 113(524), 1515–1527 (cited on page 30).
- Sasahara, K., Cody, M. L., Cohen, D., & Taylor, C. E. (2012). Structural Design Principles of Complex Bird Songs: A Network-Based Approach. *PLoS ONE*, 7(9) (cited on page 105).
- Selin, A., Turunen, J., & Tanttu, J. T. (2007). Wavelets in recognition of bird sounds. *Eurasip Journal on Advances in Signal Processing*, 2007 (cited on page 72).
- Selouani, S.-A., Kardouchi, M., Hervet, É., & Roy, D. (2005). Automatic Birdsong Recognition Based on Autoregressive Time-Delay Neural Networks. *2005 ICSC Congress on Computational Intelligence Methods and Applications*, 1–6 (cited on page 70).
- Serban, I. V., Sordoni, A., Charlin, L., Pineau, J., Courville, A., & Bengio, Y. (2017). A Hierarchical Latent Variable Encoder-Decoder Model for Generating Dialogues Iulian, 3295–3301 (cited on page 48).
- Shannon, B. J., & Paliwal, K. K. (2003). A Comparative Study of Filter Bank Spacing for Speech Recognition. *Microelectronic Engineering Research Conference*, 41, 310–312 (cited on pages 70, 71).
- Shen, C. H., Sung, J. Y., & Lee, H. Y. (2018). Language Transfer of Audio Word2Vec: Learning Audio Segment Representations Without Target Language Data. *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings, 2018-April*, 2231–2235 (cited on page 76).
- Shiovitz, K. A. (1975). The process of species-specific song recognition by the indigo bunting, *Passerina cyanea*, and its relationship to the organization of avian acoustical behavior. *Behaviour* (cited on page 36).
- Skowronski, M. D., & Harris, J. G. (2004). Exploiting independent filter bandwidth of human factor cepstral coefficients in automatic speech recognition. *The Journal of the Acoustical Society of America*, 116(3), 1774–1780 (cited on page 71).

- Smilkov, D., Thorat, N., Assogba, Y., Yuan, A., Kreeger, N., Yu, P., Zhang, K., Cai, S., Nielsen, E., Soergel, D., Bileschi, S., Terry, M., Nicholson, C., Gupta, S. N., Sirajuddin, S., Sculley, D., Monga, R., Corrado, G., Viégas, F. B., & Wattenberg, M. (2019). TensorFlow.js: Machine Learning for the Web and Beyond (cited on page 113).
- Smith, R. (2014). Complexity in animal communication: Estimating the size of N-gram structures. *Entropy*, 16(1), 526–542 (cited on page 102).
- Soha, J. A., Nelson, D. A., & Parker, P. G. (2004). Genetic analysis of song dialect populations in Puget Sound white-crowned sparrows. *Behavioral Ecology*, 15(4), 636–646 (cited on page 8).
- Sokal, R. R. (1958). A statistical method for evaluating systematic relationships. *Univ. Kansas, Sci. Bull.*, 38, 1409–1438 (cited on page 27).
- Solem, A. (2016). Celery - Distributed Task Queue — Celery 5.1.2 documentation. (Cited on page 79).
- Somervuo, P., & Harma, A. [Aki]. (2004). Bird song recognition based on syllable pair histograms. *Acoustics, Speech, and Signal Processing, 2004. Proceedings. (ICASSP '04). IEEE International Conference on*, 5, V–825–8 vol.5 (cited on page 72).
- Somervuo, P., Harma, A. [Aki], & Fagerlund, S. [Seppo]. (2006). Parametric representations of bird sounds for automatic species recognition. *IEEE Transactions on Audio, Speech and Language Processing*, 14(6), 2252–2263 (cited on pages 46, 70).
- Specht, R. (2002). Avisoft-saslab pro: sound analysis and synthesis laboratory. *Avisoft Bioacoustics, Berlin*, 2002, 1–723 (cited on page 6).
- Stowell, D., & Plumbley, M. D. (2014). Automatic large-scale classification of bird sounds is strongly improved by unsupervised feature learning. *PeerJ*, 2, e488 (cited on pages 70, 74).
- Sun, R., Marye, Y. W., & Zhao, H. A. (2013). Wavelet transform digital sound processing to identify wild bird species. *International Conference on Wavelet Analysis and Pattern Recognition*, 306–309 (cited on page 70).
- Tchernichovski, O. [O.], Lints, T. J., Derégnaucourt, S., Cimenser, A., & Mitra, P. P. (2004). Studying the song development process: Rationale and methods. *Annals of the New York Academy of Sciences*, 1016, 348–363 (cited on pages 4, 68, 89).
- Tchernichovski, O. [Ofer], Nottebohm, F., Ho, C. E., Pesaran, B., & Mitra, P. P. (2000). A procedure for an automated measurement of song similarity. *Animal behaviour*, 59(6), 1167–1176 (cited on page 6).
- TensorFlow. (2017). TensorBoard: Visualizing Learning. (Cited on page 16).
- Thompson, N. S., LeDoux, K., & Moody, K. (1994). A System for Describing Bird Song Units. (Cited on page 36).
- Timoney, J., Lysaght, T., & Schoenwiesner, M. (2004). Implementing loudness models in matlab. *Proc. of the 7th Int. Conference on Digital Audio Effects (DAFX-04)*, (1), 5–9 (cited on page 41).
- Tomback, D. F., & Baker, M. C. [Myron Charles]. (1984). Assortative mating by white-crowned sparrows at song dialect boundaries. *Animal Behaviour*, 32(2), 465–469 (cited on page 8).
- Tsai, W.-h., & Xue, Y.-z. (2014). On the Use of Speech Recognition Techniques to Identify Bird Species. *Computational Linguistics and Chinese Language Processing*, 19(1), 55–68 (cited on page 102).
- Vallejo, E. E., Cody, M. L., & Taylor, C. E. (2007). Unsupervised acoustic classification of bird species using hierarchical self-organizing maps. *Progress in Artificial Life, Proceedings*, 4828, 212–221 (cited on page 43).
- Van Der Maaten, L., & Hinton, G. (2008). Visualizing Data using t-SNE. *Journal of Machine Learning Research* (cited on pages 26, 90).
- Van Heijningen, C. A., De Visser, J., Zuidema, W., & Ten Cate, C. (2009). Simple rules can explain discrimination of putative recursive syntactic structures by a songbird species. *Proceedings of the National Academy of Sciences of the United States of America*, 106(48), 20538–20543 (cited on page 102).
- Von Békésy, G., & Wever, E. G. (1960). *Experiments in hearing* (Vol. 8). McGraw-Hill New York. (Cited on page 38).
- Wang, N.-C., Hudson, R. E., Tan, L. N., Taylor, C. E., Alwan, A., & Yao, K. (2013). Bird-phrase segmentation by entropy-driven change point detection. *Electrical Engineering*, 1, 773–777 (cited on page 46).
- Warblr. (2020). Warblr. (Cited on page 4).
- Webassembly Community Group. (2020). WebAssembly Specification. 1 (cited on page 113).

- Webb, W. H., Roper, M. M., Pawley, M. D., Fukuzawa, Y., Harmer, A. M., & Brunton, D. H. (2021). Sexually distinct song cultures in a songbird metapopulation. *bioRxiv* (cited on pages 4, 87).
- Wellock, C. D., & Reeke, G. N. (2012). Quantitative tools for examining the vocalizations of juvenile songbirds. *Computational Intelligence and Neuroscience, 2012* (cited on pages 4, 72).
- Weninger, F., & Schuller, B. (2011). Audio recognition in the wild: Static and dynamic classification on a real-world database of animal vocalizations. *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, 337–340 (cited on page 70).
- Westneat, M. W., Long, J. H., Hoese, W., & Nowicki, S. (1993). Kinematics of birdsong: functional correlation of cranial movements and acoustic features in sparrows. *Journal of Experimental Biology*, 182, 147–171 (cited on page 39).
- Wielgat, R., Zielinski, T. P., Potempa, T., Lisowska-Lis, A., & Król, D. (2007). HFCC based recognition of bird species. *Signal Processing: Algorithms, Architectures, Arrangements, and Applications, SPA 2007 - Workshop Proceedings*, 129–134 (cited on page 70).
- Wilbrecht, L., & Nottebohm, F. (2003). Vocal learning in birds and humans. *Mental Retardation and Developmental Disabilities Research Reviews*, 9(3), 135–148 (cited on page 45).
- Williams, J. M., & Slater, P. J. B. [P. J B]. (1990). Modelling bird song dialects: the influence of repertoire size and numbers of neighbours. *Journal of Theoretical Biology*, 145(4), 487–496 (cited on page 4).
- Wytttenbach, R. A., May, M. L., & Hoy, R. R. (1996). Categorical perception of sound frequency by crickets. *Science* (cited on page 7).
- Ye, N. (2003). *The Handbook of Data Mining*. (Cited on page 72).
- Yeh, Y.-c., Hsu, C.-y., & Yeh, Y.-c. (2019). Application of Auto-Encoder for Time Series Classification with Class Imbalance (cited on pages 48, 76).
- Zaki, M. J. (2001). SPADE: An efficient algorithm for mining frequent sequences. *Machine Learning*, 42(1-2), 31–60 (cited on pages 16, 29, 102, 108).
- Zbancioc, M., & Costin, M. (2003). Using neural networks and LPCC to improve speech recognition. *SCS 2003 - International Symposium on Signals, Circuits and Systems, Proceedings*, 2, 445–448 (cited on page 70).
- Zhao, Z., hua Zhang, S., yong Xu, Z., Bellisario, K., hua Dai, N., Omrani, H., & Pijanowski, B. C. (2017). Automated bird acoustic event detection and robust species classification. *Ecological Informatics*, 39(November 2016), 99–108 (cited on page 44).
- Zsebők, S., Herczeg, G., Laczi, M., Nagy, G., Vaskuti, É., Hargitai, R., Hegyi, G., Herényi, M., Markó, G., Rosivall, B., Szász, E., Szöllősi, E., Török, J., & Garamszegi, L. Z. (2020). Sequential organization of birdsong: relationships with individual quality and fitness. *Behavioral Ecology*, 1–12 (cited on page 106).
- Zuidema, W., & de Boer, B. (2009). The evolution of combinatorial phonology. *Journal of Phonetics*, 37(2), 125–144 (cited on page 3).
- Zwicker, E. (1961). Subdivision of the audible frequency range into critical bands (Frequenzgruppen). *The Journal of the Acoustical Society of America*, 33(2), 248 (cited on page 40).
- Zwicker, E., Flottorp, G., & Stevens, S. S. (1957). Critical Band Width in Loudness Summation. *The Journal of the Acoustical Society of America*, 29(5), 548–557 (cited on pages 37, 67).