QUEEN'S UNIVERSITY BELFAST

Gesture Recognition for Musician Computer Interaction

by

Nicholas Edward Gillian

BSc Music Technology, Queen's University Belfast 2006

MA Sonic Arts, Queen's University Belfast 2007

A thesis submitted in partial fulfillment for the degree of Doctor of Philosophy

in the
Faculty of Arts, Humanities and Social Sciences
School of Music & Sonic Arts

March 2011

"If we knew what we were doing, it wouldn't be called research, would it?"

Albert Einstein

Abstract

This thesis investigates how machine learning can be applied to the automatic recognition of a musical gesture by a computer. The recognition of gestures in a musical domain provides a number of interesting research challenges over and above the recognition of gestures in the more general field of human-computer interaction as, due to its real-time musical application, a low-latency, highly robust, user-configurable recognition system is required. These research challenges raise a number of fundamental questions related to the application of gesture recognition for musician-computer interaction; such as what differences, if any, are there between the application of machine learning for the classification of musical gestures from that of the classification of other gestures used throughout various fields within human-computer interaction? This thesis addresses such questions and in doing so tests the applicability of the leading machine learning algorithms for the recognition of musical gestures along with developing a number of new algorithms specifically for the recognition of musical gestures.

The work in this thesis focuses primarily on the discrete classification of a musical gesture, as opposed to the continuous mapping of a movement to a sound or control parameter. The scope of the research presented in this thesis is therefore constrained to the design and evaluation of machine learning algorithms that can be used to classify both static musical postures and musical gestures that consist of a cohesive sequence of movements that occur over a variable time period (i.e. temporal gestures).

The principal contributions of this thesis include a number of novel machine learning algorithms that have been specifically developed for the recognition of both static musical postures and temporal musical gestures. Another major contribution of this thesis is the development of a real-time gesture recognition software tool that has been designed to operate independently from any one specific piece of sensor hardware or audio software. Rather than pre-training the software tool to recognise a specific set of musical gestures, such as the communicative gestures of a conductor or the expressive gestures of a pianist for example; the software has instead been designed to facilitate a musician to train the recognition algorithms with the specific gestures that musician wants to use. The software has therefore been designed to enable a performer, regardless of their programming abilities, to quickly train a computer to recognise their musical gestures using a number of powerful machine learning algorithms, including the algorithms that have been specifically developed as the result of this thesis. The work in this thesis therefore not only contributes to the domains of musician-computer interaction and the more general field of human-computer interaction; it also facilitates performers to directly apply these contributions to their compositions, performances and/or research.

Acknowledgements

First and foremost, I would like to thank both Sile O'Modhrain and Ben Knapp for their amazing supervision skills, guidance, encouragement, inspiration and friendship. Thank you for believing in me and giving me this extraordinary opportunity to embark on this adventure. Thank you Ben for always asking the right questions and thank you Sile for always having the right answers.

I would like to thank Michael Alcorn, for his vision, leadership and hard-work in establishing the fantastic research and teaching facility that is SARC. Thanks to Pearl Young, Marian Hanna and Kirk Shilliday for taking the University and E.U. red tape and making it into beautiful little bows. A special thanks to Chris Corrigan for always making me laugh any time I needed a cable or microphone.

I owe many thanks to Cathy Craig and Michael Gurevich for their excellent comments and feedback during my differentiation.

Thanks to all those past and present Ph.D. students within SARC who have made the many many hours spent in the lab a joy. Special thanks to Peter Bennett for always inspiring me with his sketches and ideas, Matthew Rodger for the many discussions and SKILLS adventures, Nicholas 'Alpha' Ward for always being a continuous source of knowledge, wisdom and chaos, Sebastian Heinz for his endless enthusiasm, Javier 'J.J.' Jaimovich for always interrupting me with Matlab questions, Steve 'The Daddy' Davis for our thought provoking discussions on the world and its inhabitants, Migel Ortiz-Pérez, Adnan Marquez-Borbon, Brennon Bortz, Niall Coghlan, Cavan Fyans, Donal O'Brien, Tom Davis, Henry Vega, Grant Ford, Fernando Gualda, Dionysis Athinaios for his beats, Orestis Karamanlis for his amazing music, Sarah Orr for telling me all about imaginary numbers many many years ago, Sarah Bass, Chris McClelland and countless others who I have no doubt forgot, you know who you are.

I am incredibly grateful for the many friends both outside the lab and who I have met along this adventure. Special thanks to Will Young, Alyson Campbell, Anna Newell, Sarahjane 'Belto' Belton, Deborah 'Cheeky Monkey' Varoqui, Caroline Teulier, Sebastien Petracca Villard, Gregory Zellic, Steve Sinclair, Donald Glowinski, Neil Harrision, Jo Harrison, Cindy and Uel, Kimberly 'Coconut' O'Hara, Ashley Cassidy, Mark 'Lover' Todd, Louise Harvey, Leanne Cochrane, David Goodall, Robyn Farah, Lesley 'Bisous' Oman and Craig Oman.

I would like to thank Sarah Nicolls, Bob Pritchard and Alexander Jensenius for their thoughts, questions and comments.

A tremendous thanks to all the EyesWeb team, for creating such a great piece of software to work with and for always welcoming me so warmly to Casa Paganini. I would particularly like to thank Paolo Coletta and Gualtiero Volpe for their support, technical advice and enthusiasm. I would also like thank all those working at the BIOMOBIUS team, especially Brian O'Mullane for his technical skills and friendship.

This Ph.D. was funded by the E.U. project SKILLS, I am indebted to everyone I met and worked with whilst working on this project. I would particularly like to thank Julian Lagarde, Benoit Bardy, Denis Mottet, Stas Krupenia, Pablo Hoffmann, Sabine 'Sabineee' Webel, Timo Engelke, Katharina Hertk and Uli Bockholt.

A great big bear-sized thanks goes to Samira 'Poupette' Bouazzaoui for being the best coffee buddy one could ever ask for; I will treasure every piece of crunchy cheese cake and chicken caesar salad (without chicken) we enjoyed together. A very very special thanks to Johann 'Yogi Bear' Issartel, who has been along side me every step of this adventure, on the infrequent bad days and on the many many many fantastic days. Thank you for everything - without your friendship this would have been a different story.

Finally, a massive thanks to all my family for all their support, encouragement, nourishment and financial assistance over nearly the last eight years of university. A special thanks to both Nanny and Ivan for always being there - things would have been so much harder without you both!

For Mum - you are simply amazing

Contents

List of Figures

List of Tables

Chapter 1

Introduction

We can only see a short distance ahead, but we can see plenty there that needs to be done.

Alan Turin

Performers have been using sensors to trigger, control and manipulate sounds via computer software for over 50 years, such as the pioneering work by Mathews (1991a) for example, who used an augmented radio baton to trigger and continuously control a MIDI synthesizer. Eminent technological breakthroughs over the last half-century have resulted in, amongst other things, the production of inexpensive ubiquitous sensor devices, exceptionally powerful computer hardware and real-time music composition and performance software. The combination of inexpensive sensors, powerful computer hardware and real-time audio processing software is particularly advantageous for a musician; especially in a live performance scenario. The accessibility of this technology affords a performer exciting new interaction paradigms facilitating the performer to use their own body movements to control a machine even when their hands are busy playing an instrument.

However, despite the ever decreasing cost in sensor technology and the rapid advances in computer systems, it is still extremely difficult for a performer to interact with a computer in the same way as they would interact with other performers live on stage. Directly mapping the output of a sensor to the frequency of an oscillator, for example, is one thing; but actually getting a computer to 'understand' this signal and the underlying movement that created it is quite another. To enable effective musician-computer interaction that rivals that of performer-performer interaction a computer must be able to recognise and respond to the musical cues, movements and directions of the performer. Teaching a computer to recognise these musical gestures is not an easy task, however, as

first an appropriate sensor must be used to capture the gesture and second the computer must some-how learn to recognise the pattern that occurs in the sensor data (or features derived from the data) when a corresponding gesture takes place.

To facilitate a computer in recognising these patterns a performer may want to apply the tools and techniques developed by a branch of artificial intelligence called *machine learning*. The goal in machine learning is to attempt to 'teach' a computer to recognise a specific pattern, such as a musical conducting gesture, by using a computer algorithm to analyse a data set in the hope that the algorithm can automatically discover the underlying pattern in the data. Machine learning has been successfully applied to solve complex pattern recognition problems in areas such as automated speech recognition (e.g., Rabiner, 1989, Benzeghiba et al., 2007), fingerprint recognition (e.g., Kong et al., 2009, Liu, 2010), optical character recognition (e.g., Mori et al., 1992, Plamondon and Srihari, 2000), face recognition (e.g., Zhao et al., 2003, Zhang and Gao, 2009) and much more and would therefore provide an appropriate tool for the recognition of musical gestures.

1.1 Thesis Overview

This thesis investigates how machine learning can be applied to the recognition of musical gestures, with the goal of making gestural interaction as viable an interaction paradigm between a musician and a computer as it currently is between two performers. Whereas the majority of previous work in gestural interaction for musical control, such as in Waisvisz (1985), Mulder (2000), Tanaka and Knapp (2002) and much more, has focused on the continuous mapping of a movement to a sound or control parameter; the work in this thesis focuses on the discrete classification of a musical gesture. This thesis specifically investigates how a computer can be trained to recognise both **static postures** and gestures that consist of a cohesive sequence of movements that occur over a variable time period, which shall be referred to as **temporal gestures**.

The work in this thesis can be differentiated from the previous work in the discrete classification of musical gestures, such as that undertaken by Merrill and Paradiso (2005), Bevilacqua et al. (2007) and Bevilacqua et al. (2009), because it is not constrained to the recognition of a specific set of gestures, does not require a dedicated piece of sensing hardware and is not limited to the control of one piece of computer software. This research also stands apart from the recently published work by Fiebrink (2011), who examined machine learning through the lens of human-computer interaction in the context of computer music and performance, as this work addresses the recognition

of temporal gestures. This thesis also presents a number of novel machine learning algorithms that have been specifically designed for the recognition of musical gestures.

1.2 Research Questions, Aims and Objectives

The main topic of this dissertation is gesture recognition for musician-computer interaction and the main research question is:

How can machine learning be applied to the recognition of musical gestures?

This research question formed the foundation of a set of aims and objectives upon which this dissertation is based. These are to:

1. Evaluate what differences, if any, there are between the application of machine learning for the recognition of musical gestures from that of the recognition of other gestures used throughout various fields within human-computer interaction.
2. Test the applicability of the leading machine learning algorithms for the recognition of musical gestures.
3. Develop new algorithms specifically for musician-computer interaction if they are required.
4. Develop software tools that can facilitate real-time musician-computer interaction for any user; even those with no knowledge of machine learning or who have limited computer skills.

1.3 Thesis Contributions

The contributions made in this thesis are as follows:

- *Definition of Musician-Computer Interaction*

  This thesis defines the term **musician-computer interaction** (**MCI**) as a specific subfield of the larger research area of human-computer interaction (**HCI**). MCI focuses specifically on technology that can enable musicians to interact with computers and provides a number of interesting research and design challenges

over and above the more general field of HCI as, due to its real-time musical application, a low-latency highly robust user-configurable system is required. MCI was therefore defined to help differentiate the design and research goals from other areas of HCI. This definition can be found in chapter 3.1.

- *Adopting A Machine Learning Approach For MCI*

  This thesis presents the motivations for a performer to adopt a machine learning approach to enable the automatic recognition of musical gestures to be used for MCI. This thesis proposes that the application of machine learning for the recognition of musical gestures requires a paradigm shift from the common training, testing, deployment and evaluation strategies used throughout other areas of HCI that also use gesture recognition. Evidence is presented to show that the goal of a gesture recognition system for MCI should be to achieve a low intra-personal generalisation error, as opposed to the inter-personal generalisation error goal that is common in other areas of HCI. The MCI machine learning design, development, deployment and evaluation strategies are then applied to create a design criteria for a recognition system for musical gestures which in turn provides the motivation for a new software tool to enable the recognition of discrete gestures for MCI. These arguments can be found in chapter 3.1.3.

- *Recognition of Static Musical Gestures*

  This thesis presents a novel algorithm called the **Adaptive Naïve Bayes Classifier** (**ANBC**) which can be used for the recognition of static musical gestures. The ANBC algorithm has five significant advantages for the classification of static musical gestures:

  1. The input to the ANBC algorithm consists of an N-dimensional vector, resulting in the input to the algorithm not being constrained to only work with one type of sensor, such as a mouse or camera.
  2. An N-dimensional weighting coefficients vector enables the user to specify which of the N dimensions of input data are salient for a particular gesture. This enables one general classifier to be used in scenarios were several classifiers would have been required.
  3. The ANBC algorithm can be rapidly trained with a small number of training examples.
  4. The ANBC algorithm can be used to recognise static musical gestures in a continuous stream of data that may also contain non-gestural data without having to first train a null-model, such as a noise model that is used in speech recognition.

5. The ANBC algorithm can automatically adapt itself as a performer adapts their own gestures over, for example, the course of a rehearsal period.

The ANBC algorithm can be found in chapter 4.

- *Recognition of Temporal Musical Gestures*

This thesis makes some specific contributions to the discrete classification of temporal gestures for both MCI and for the wider HCI community. Three existing machine learning algorithms have been specifically adapted for the recognition of musical gestures. Each algorithm has been extended to:

1. Classify any N-dimensional signal.
2. Be rapidly trained with a small number of training examples.
3. Recognise temporal musical gestures in a continuous stream of data that may also contain non-gestural data without having to first train a null-model.

These machine learning algorithms include **Hidden Markov Models**, **Support Vector Machines** and **Dynamic Time Warping**. Each algorithm has been tested using a specifically captured data set of temporal gestures with all three modified algorithms achieving excellent recognition results, some even achieving 100% recognition. The advantages and disadvantages of each algorithm have been summarized for their potential application in MCI. The algorithms, results and summaries can be found in chapters 5 to 7.

- *A Flexible Gesture Recognition System For MCI*

One of the major contributions of this thesis is that all of the algorithms developed, design concepts proposed and training strategies suggested have all been encapsulated within one flexible software program that enables composers, performers and researchers to actually use this work to recognise their musical gestures. The software, which is called the SARC EyesWeb Catalog (**SEC**), has been designed to operate independently from any one specific piece of sensor device or audio software and operates online[1] and in real-time. The SEC's main advantage for MCI is that, due to its modular design, it facilitates a performer, regardless of their programming abilities, to quickly train a computer to recognise their musical gestures using a number of powerful machine learning algorithms. The SEC is presented in chapter 3.3.

[1] i.e., sensor data can be streamed into the software and be classified as a gesture is being performed, rather than offline recognition where a performer makes a gesture, saves the sensor data to a file, loads the file into an analysis program and then gets a recognition result

1.4 Thesis Outline

The remaining chapters of this thesis cover:

Background and Related Work :

Chapter 2 reviews the prior work conducted in gesture recognition in the fields of music, human-computer interaction and other related areas. It also provides an overview of machine learning, presenting some key terminology used throughout this thesis.

Gesture Recognition Systems for Musician Computer Interaction :

Chapter 3 sets out the theoretical foundations on which the remainder of the thesis is based. It first describes in detail the area of research defined as musician-computer interaction (**MCI**). This is followed by a discussion of gestural interaction with respect to its use as a control method for a performer and puts forward a case for the use of a machine learning approach to recognise a musician's gestures. The common design, development, training and evaluation strategies for gesture recognition systems are then reviewed and the strategies of such systems for HCI and MCI are differentiated. The chapter is concluded by presenting the SARC EyesWeb Catalog (**SEC**), a machine learning toolbox that has been specifically designed for musician computer interaction.

Recognition of Static Semiotic Musical Gestures :

Chapter 4 presents the ANBC algorithm and illustrates how it can be used for the recognition of static semiotic musical gestures. It will be shown how the algorithm can automatically adapt itself to accommodate a performer as they adapt their own gestures over, for example, the course of a rehearsal period. The chapter concludes with an experiment designed to evaluate the adaptive classification abilities of the ANBC algorithm. The results of this experiment show that a signicant overall improvement was achieved when the adaptive element of the algorithm was used.

Multivariate Temporal Gesture Recognition :

Chapter 5 provides an overview of the recognition of temporal gestures and discusses why this can prove to be a complex classification task. This is followed by a description of the numbers-shapes data set, a data set containing ten participants performing ten temporal gestures that was specifically collected to test the multivariate temporal recognition algorithms in this thesis. The Hidden Markov Model (**HMM**) algorithm is then presented and it will be shown how the algorithm has been adapted for MCI. The chapter concludes with a number of experiments designed to evaluate the classification abilities of the HMM algorithm. The experiments show that the modified HMM algorithm achieved a moderate recognition

rate of 87% on the numbers-shapes data set when the gestures were pre-segmented by the user. However, the HMM algorithm was deemed unsuitable for the classication of musical gestures because it failed to recognise the same gestures from a continuous stream of data and also took a long time to train.

Support Vector Machines :

Chapter 6 describes a powerful machine learning algorithm called Support Vector Machines (**SVM**) and illustrates how the algorithm has been modified for the recognition of temporal musical gestures. A number of feature extraction algorithms that have been specifically applied to represent multivariate temporal gestures are presented along with a description of how the SVM algorithm has been adapted to classify gestures from a continuous stream of data. The chapter concludes with a number of experiments designed to evaluate the multivariate temporal classification abilities of the SVM algorithm. The experiments show that the SVM algorithm achieved an excellent classification result of 99.28% when the gestures from the numbers-shapes data set were pre-segmented by the user and that it could be rapidly trained. However, although the SVM algorithm was successful at accurately rejecting null gestures (i.e. any movement that is not a gesture the algorithm was trained to recognise), the algorithm failed to achieve a useable recognition rate from a continuous stream of data.

Dynamic Time Warping :

Chapter 7 presents the Dynamic Time Warping algorithm and illustrates how it has been adapted for the recognition of temporal musical gestures. It will be shown how the DTW algorithm has been extended to classify any N-dimensional signal and automatically compute a classification threshold to reject any data that is not a valid gesture from a continuous stream of data that also contains null gestures. The modified DTW algorithm is evaluated using the numbers-shapes data set and the results show that it can be trained rapidly with a small number of training examples and it achieved an excellent recognition rate of 99.37% when the gestures from the test temporal data set were pre-segmented by the user. The DTW algorithm also achieved a moderate recognition result of 84.18% when classifying the same gestures from a continuous stream of data that also contained null gestures. The chapter concludes with a summary of the advantages and disadvantages of all three of the multivariate temporal recognition algorithms, suggesting which algorithm may be most appropriate to use in a number of hypothetical recognition scenarios.

Conclusion :

Chapter 8 summarises the work described in this thesis and highlights the contributions achieved. Future work is also discussed.

Chapter 2

Background and Related Work

Experience is the name everyone gives to their mistakes.

Oscar Wilde

This chapter reviews the prior work conducted in gesture recognition in the fields of music, human-computer interaction and other related areas. Prior to reviewing the gesture recognition literature a brief overview of machine learning is given[1], presenting some key terminology[2] that will be used throughout this thesis.

2.1 Machine Learning Fundamentals

Consider a rudimentary interaction scenario in which a musician wants to trigger a sound to play if they place their hand in a certain region of space, for example above a specific key on a piano. To capture their movements the performer might use a camera that can track the position of their hands or they might place a light sensor directly in line with the region of space being used as the 'trigger zone'. To enable a computer to recognise whether the musician's hand is in the trigger zone they might create a simple recognition system in which the raw data from the sensor is passed through a threshold algorithm which will trigger a sound to play if the sensor data exceeds a set threshold value. For this system there is one parameter value that needs to be set in the threshold algorithm, namely the value at which a response should be triggered. Setting this threshold value manually may be a trivial matter if only one sensor is being used; however this quickly becomes a complex task if more than one sensor or threshold value is required.

[1] Interested readers can find a comprehensive introduction to machine learning in Bishop (2006) and Duda et al. (2001)

[2] key terminology is highlighted in ***bold italics***

It is in this instance that a musician may want to adopt a machine learning approach in which a large set of M examples called a ***training set***, $\mathbf{x} = \{x_1, x_2, \ldots, x_M\}^\intercal$, are used by a machine learning ***algorithm*** to tune the parameters of an adaptive function or ***model***. Each example in the training set, also known as a ***training vector***, is not limited to just one dimension but can consist of an N dimensional vector of values, $\mathbf{x}_i = \{x_1, x_2, \ldots, x_N\}$. The adaptive model being used could be as simple as the thresholding algorithm in the previous example, in which it would have one input - the sensor value, one output - a value representing the system's state when the input is greater than the threshold, and one parameter - the threshold value. Alternatively, the adaptive model could consist of multiple inputs, outputs and numerous parameters.

Regardless of the complexity of the model being used, the overall concept in machine learning is the same in that a set of training data are used to tune the parameters of the model to achieve the minimum error with respect to some pre-determined error criteria. The specific criteria used will depend on the type of algorithm being applied to train the model and the type of classification problem that the user wants to solve; both of these points will be discussed in more detail shortly.

Once the model has been trained it can then determine the identity of new previously unseen input vectors, which could be comprised of a ***test set*** in offline testing or 'live' data in real-time prediction. The ability to categorise correctly new examples that differ from those used for training is known as ***generalisation***. In practical applications, the variability of the input vectors will be such that the training data can comprise only a tiny fraction of all possible input vectors, and so generalisation is a central goal in pattern recognition (Bishop, 2006).

2.1.1 Types of Learning

If the corresponding output of the model, given an input, is known in advance then the training process is referred to as ***supervised learning***. In other pattern recognition problems, the training data could consist of M unlabeled training examples, in which case the problem is referred to as ***unsupervised learning***[3]. In unsupervised learning the goal of the training process might be to automatically cluster the training data into K similar groups or project the data from a high-dimensional space into a lower dimensional space, e.g. reducing the data down to three dimensions so it can be visualised. For supervised learning, the goal of the training process will be to minimise the error value between the known output of the model, which is referred to as a ***target value***, and the actual output of the model given the current parameter values. The target values can

[3]Several other types of learning exist such as ***semi-supervised learning*** and ***reinforcement learning*** among others

be expressed using a ***target vector***, $\mathbf{t}$, with an input vector - target vector pair $\{\mathbf{x}_i, \mathbf{t}_i\}$ for each training example. As was the case for an input vector, each target vector can itself be a T dimensional vector, $\mathbf{t}_i = \{t_1, t_2, \ldots, t_T\}$. N, the number of dimensions in each training example and T can be different sizes, however, the dimensions must be consistent for any given recognition task.

The target values can either be a finite number of discrete integer values, with one integer value representing one category or class; alternatively the target values can be one or more continuous variables. For the simple thresholding example, the output of the model could therefore consist of either a discrete value, such as 0 for no trigger and 1 for trigger, or a continuous variable in the range [0.0 1.0] indicating the likelihood of the performer's hand being in the target zone. If the target values are a finite number of discrete categories then the learning problem is referred to as a ***classification*** problem, alternatively, if the learning problem is to output one or more continuous variables then the task is called ***regression***. As the work in this thesis is primarily focused on the discrete recognition of a musical gesture, as opposed to the continuous mapping of a gesture to a sound or control parameter, all of the algorithms presented throughout this thesis are used to solve classification problems rather than regression problems.

2.1.2 Training a Model

The result of running a machine learning algorithm can be expressed as a function $h_\phi(\mathbf{x})$, parameterised by the model parameters ϕ, which takes an input vector $\mathbf{x}$ as input and that generates an output vector $\mathbf{y}$, encoded in the same format as the target vector $\mathbf{t}$. This is illustrated in Figure 2.1. The precise form of the function $h_\phi(\mathbf{x})$ is determined during the training phase, also known as the learning phase, on the basis of the training data as illustrated by Figure 2.2.

A classification model for the simple thresholding example could therefore be represented by:

$$h_\phi(x) \equiv$$
$$h(x|\phi) = \begin{cases} 1, & \text{if } x >= \phi \\ 0, & \text{otherwise} \end{cases} \tag{2.1.1}$$

which would classify the input x as belonging to class 1 if its value was greater than or equal to the threshold parameter ϕ, with all other values of x being classified as 0. The objective of training this simple thresholding classifier could be to use the labeled training data to estimate the best value for ϕ that maximises the number of correctly

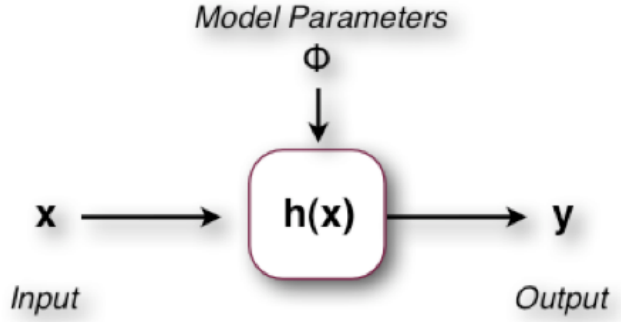


FIGURE 2.1: A machine learning algorithm, expressed as a function $h_\phi(\mathbf{x})$ with input $\mathbf{x}$, output $\mathbf{y}$ and parameterised by ϕ.

classified training examples. The training algorithm used to estimate the best value for ϕ could therefore be given by the following pseudocode:

Initialisation : Set ϕ to a random value

Training Loop :

1. Run all M training examples through the model using the current estimate of ϕ to classify each training example as either class 0 or class 1
2. Calculate ξ, the classification error, which could be given by:

$$\xi = \frac{\text{Number of incorrectly classified examples}}{\text{Number of examples}}$$

3. **IF** ξ has improved from the previous loop then update ϕ, return to step 1

   IF ξ has not improved from the previous loop then stop the training process

The exact method for updating ϕ during stage 3 of the training loop will vary depending on the actual optimisation algorithm being applied by the machine learning algorithm used to train the model.

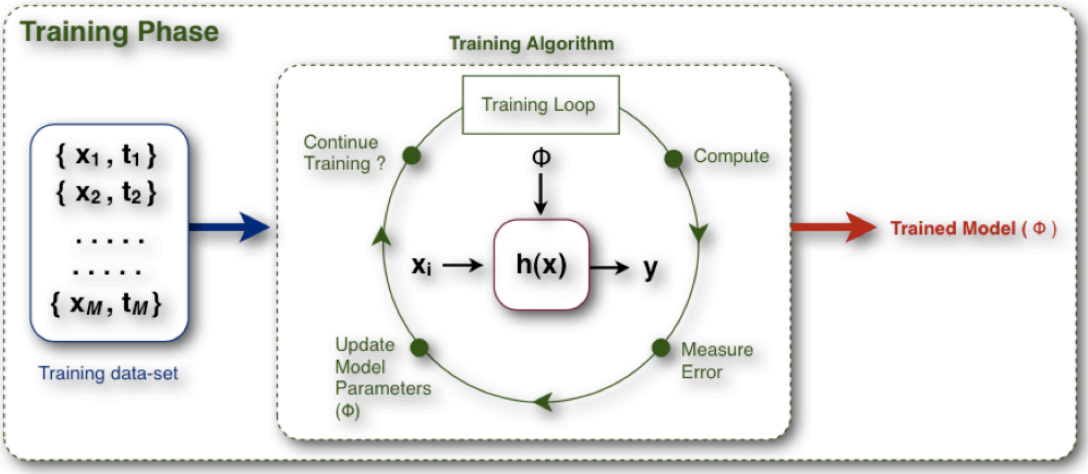


FIGURE 2.2: An illustration of the training phase of a machine learning algorithm. The precise form of the function $h_\phi(\mathbf{x})$ is determined during this phase on the basis of the training data. The resulting trained model can then be used to determine the identity of new, previously unseen, input vectors.

2.1.3 Pre-processing

It is common for the original input variables to be pre-processed in some way to reduce both the computational load and complexity of the recognition problem. The pre-processing stage could consist of simply scaling or normalising the data to a standard range, smoothing it to remove noise, or by transforming it into some new subspace of variable, where it is hoped, the recognition problem will be easier to solve. This pre-processing stage is sometimes also called ***feature extraction***. It should be noted that the new test data must be pre-processed using the same feature extraction method as used in the training data.

2.1.4 Post-processing

In addition to pre-processing the raw data prior to input to a machine learning algorithm, it is also common to process the output of a machine learning algorithm prior to using its output to make a decision, such as either triggering or not triggering a sound to play. This post-processing stage could consist of waiting for a number of consecutive 'trigger' classification results before a sample is triggered or by combining the classification results of a number of machine learning algorithms together to create one super-classifier. The post-processing stage also enables the output of the machine learning algorithm to be combined with some additional domain-specific information to provide additional context, such as the performer is not even on stage yet so no samples should be played

regardless of the output of the machine learning algorithm. Figure 2.3 illustrates the processing chain for a generic gesture recognition system.

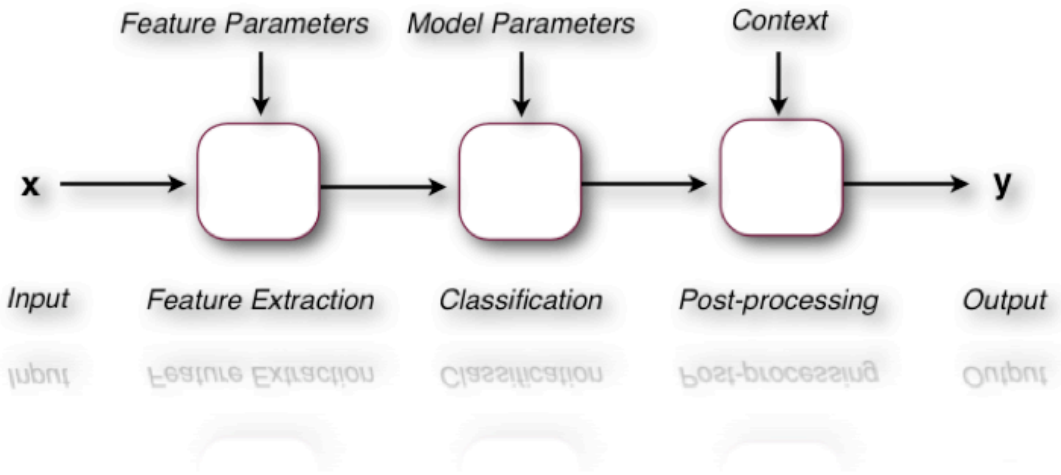


FIGURE 2.3: An illustration of the processing chain for a gesture recognition system. The input data is first pre-processed, which could consist of simply smoothing the data or extracting useful features, with the resulting processed data being used as input to a trained classification algorithm. The output of the classifier is then post-processed, which enables additional information such as context to be combined with the output of the classifier before a final decision is made and output by the recognition system.

2.1.5 Underfitting, Overfitting and Model Selection

Care must be taken when training any machine learning algorithm, particularly when the training sample size is small or when the number of parameters in the model is large. In these instances, it is common for the model to give unreliable classification on any data that did not feature in the training set. This is referred to as ***overfitting*** and occurs when the model's parameters are too closely fitted on the training data, resulting in high variance in the model's error rate (the percentage of incorrectly classified instances in the data set). The opposite is true when the model's parameters are poorly fitted on the training data and is referred to as ***underfitting*** which results in a high bias in the model's error rate. To mitigate underfitting and overfitting, the model must be presented with enough training data so that it can build a statistical model of the process which generates the data, rather than learning an exact representation of the training data itself.

To reliably test a trained model's classification performance, the model's error rate should be calculated. It is unwise to use the ***resubstitution error*** (error rate on the training set) as this can give poor predictions of the algorithm's classification ability on new data. It is useful therefore to have a method of estimating a model's ability to classify previously unseen input vectors, which can be achieved by calculating a model's

generalisation error. A number of generalisation error functions can be used to help tune the parameters of a model. This can be achieved by using a small subset of the training data to continuously train and test the model under different parameter values, with the parameter values resulting in the lowest generalisation error then being used to train the model using all the data.

2.1.6 Generalisation Error

The ability of a machine learning algorithm to correctly classify new input examples that differ from those used to train its model can be measured by its generalisation error. This is a quantitative error function that measures the trained models ability to correctly classify a previously unseen input taken from a test set. If a large amount of data is available (e.g. thousands of training examples) then the training set and test set can easily be created by taking two independent samples and using one for training and one for testing. In some instances, it is also useful to create a third data set called the ***validation set*** which is used during an algorithm's training phase to continually tune the model's parameters. In many real world applications, however, it can often be expensive and time consuming to collect a large data set and simply segmenting the data into a training and test set is inappropriate. This is particularly the case in MCI, where each composer/performer may want to create their own recognition system and may not want to spend time collecting hundreds of training examples for each of the gestures in their vocabulary. In scenarios where only a limited amount of training data is available, a ***hold-out*** procedure, where some of the data is held-out of the training set and used for the test set, can be applied to access a reliable estimate of a trained algorithm's generalisation error. One of the most common hold-out procedures is cross-validation.

2.1.6.1 Cross-Validation

Cross-validation involves partitioning the data set into complementary subsets, training the model on one subset (the training set) and validating the model on the second subset (the test set). To reduce the variability of the generalisation error, multiple rounds of cross-validation can be performed using different partitions and the validation results averaged.

Cross-validation is a particularly apt method of estimating an algorithm's classification ability for MCI as it enables a model to be robustly tested with a limited amount of training data. This therefore reduces the time required to collect the training data (as less is required) and allows a performer to quickly test if the gestures they have trained the algorithm with are senseable and aesthetically suitable. There are two

common types of cross-validation: ***repeated sub-sampling validation*** and ***K-fold cross-validation***.

2.1.6.2 Repeated Sub-Sampling Validation

Repeated sub-sampling validation, as illustrated in Figure 2.4, involves randomly splitting the data set into a training data set and a test data set. For each random split, the model is trained using the training-data and validated using the test-data. The results are then averaged over the splits. The advantage of this method over K-fold validation is that the ratio of the training/test split is not dependent on the number of iterations. The disadvantage of this method is that some observations may never be selected in the validation subsample, whereas others may be selected more than once.

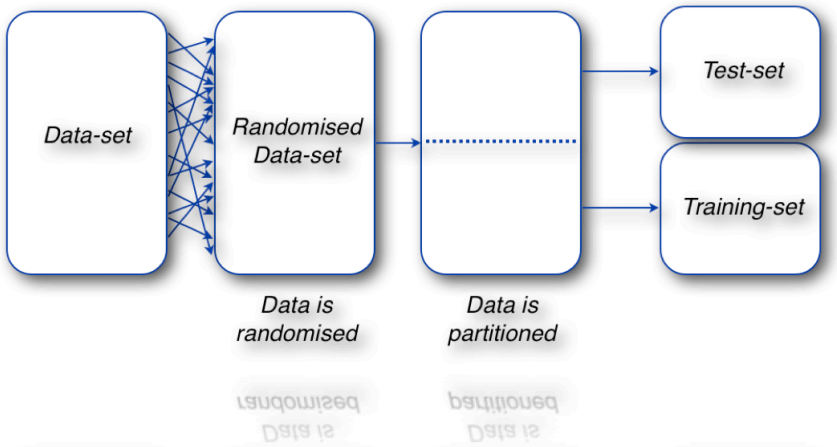


FIGURE 2.4: One iteration of repeat sub-sampling validation.

2.1.6.3 *K*-fold Cross-Validation

K-fold cross-validation, as illustrated in Figure 2.5, involves partitioning the original sample into randomly partitioned K subsamples (a K value of 10 is commonly used). A single subsample is retained as the test data and the remaining K-1 subsamples are used as the training data. The models predictive capabilities can be accessed using the test data and the cross-validation process is then repeated K times, with each of the K subsamples used exactly once as the validation set. The K results from each of the folds can be averaged to produce a single generalisation estimation. This method has the advantage over repeated random sub-sampling of using all observations for both training and testing, with each observation being used for validation exactly once. One extreme version of K-fold cross-validation is ***leave-one-out*** cross-validation, where the K value is essentially set to M, the number of training examples in the data set. Although

leave-one-out cross-validation has the advantage of making maximal use of the training data available, it is extremely slow to perform as the machine learning model must be trained and tested M times.

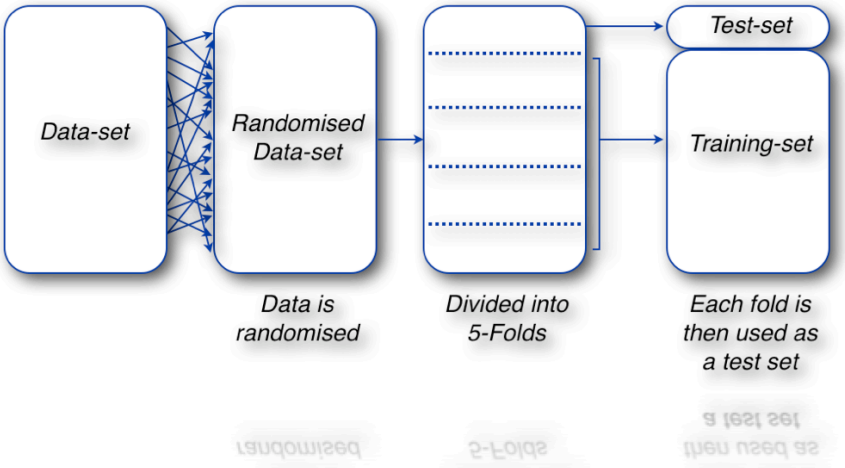


FIGURE 2.5: K-Fold Cross Validation, with a K value of 5.

2.1.6.4 Stratified Validation

Both of the cross-validation methods above have a stratified version in which the sub-sections or folds are selected so that the mean response value is approximately equal in all sub-sections. In the case of a dichotomous classification task, this means that each sub-section or fold would contain roughly the same proportions of the two types of class labels. This is particularly useful if the response values are unbalanced as an equal distribution of response values will be present in both the training and test sets.

2.1.6.5 Cross Validation Error Measures

Cross-validation can be measured using any quantitative measure of fit that is appropriate for the data and the model. If for example the classification problem was binary, each case in the test set is either correctly or incorrectly predicted. In this case the misclassification error rate or the positive predictive value could be used to summarise the fit. When the value being predicted is continuously distributed, the sum-of-squares error, mean squared error, or root mean squared error could be used to summarise the model's generalisation error. These error functions are normally used instead of the error's absolute value as they allow the error function to be treated as a continuous differentiable function (Bishop, 1995). The sum-of-squares error is simply the sum, over all the samples in the training set, of a squared error between the output of the function $h(\mathbf{x}_i)$ and that samples expected target value t_i. The mean squared error and root

mean squared error have the same properties as the sum-of-squares error, however they have the advantage that their values do not grow with the size of the input-target data set. The mean squared error and root mean squared error functions are therefore much better suited when the generalisation error of a number of different sized data sets need to be compared against each other.

The sum-of-squares error function is given by:

$$SSE = \sum_{i=1}^{M} \{h(x_i) - t_i\}^2 \tag{2.1.2}$$

where M is the number of training examples in the training set. The mean squared error is given by:

$$MSE = \frac{1}{M} \sum_{i=1}^{M} \{h(x_i) - t_i\}^2 \tag{2.1.3}$$

The root mean squared error is given by:

$$RMSE = \sqrt{\frac{1}{M} \sum_{i=1}^{M} \{h(x_i) - t_i\}^2} \tag{2.1.4}$$

2.1.7 Validation Methods used in this Thesis

The primary method for estimating the generalisation abilities of the machine learning algorithms presented throughout this thesis will be K-fold cross-validation. This is because, as previously mentioned in this chapter, cross-validation is a particularly apt method of estimating an algorithm's classification ability as it enables a model to be robustly tested with a limited amount of training data. This is extremely advantageous for MCI because, as will be expanded in more detail in chapter 3.2, a machine learning algorithm that can be quickly trained and tested using a limited number of training examples is particularly useful for a performer. Such an algorithm, for example, would facilitate the rapid prototyping and evaluation of any gesture-sound relationship the performer thinks may be useful for a real-time performance scenario.

2.1.8 Applying Machine Learning to Gesture Recognition

As this section has shown, applying machine learning to solve a recognition problem involves a number of fundamental changes in the tools, techniques and ways of thinking commonly used to solve other computational problems. Unlike many other computational problem solutions, which are explicitly programmed by the developer, machine

learning attempts to infer the solution to the problem directly from analysing example data of the problem itself.

If the problem is to recognise a gesture, adopting a machine learning approach provides a number of significant advantages over simply 'hard-coding' [4] a recognition system. Not only is a machine learning algorithm more flexible than the hard-coding approach, as once an algorithm is initially developed it can be applied to solve numerous recognition tasks just by retraining a new model, it will frequently be more robust to any variability or noise contained in the input to the algorithm. Machine learning algorithms can also be used to solve complex, high dimensional, non-linear recognition problems for which it could be almost impossible to simply hard-code a workable solution. In the following section, a general review is provided of the application of machine learning for the recognition of gestures throughout a wide range of areas within the field of human-computer interaction. This is followed by a specific review of the application of machine learning for the recognition of gestures throughout the domain of musician-computer interaction.

2.2 Gesture Recognition for Human Computer Interaction

Gestural interaction with computational devices, and therefore the recognition of gestures, has been the focus of research throughout many areas of HCI since the development of early applications in the 1960s; such as Ivan Sutherland's Sketchpad (Sutherland, 1964) which used an early form of stroke-based gestures using a light pen to grab and manipulate graphical objects on a tablet display. Teitelman (1964) was one of the first researchers to develop a trainable gesture recogniser that could classify hand drawn characters in real-time. Several other pen-based recognition systems followed in the 1960s and 1970s, such as the GRAIL system (Ellis et al., 1969) or in the AMBIT/G system (Christensen, 1968); with this form of interaction now being widely accepted throughout the HCI community (Karam, 2006).

Glove and magnetic sensor based systems, such as the work found in Bolt (1980), Zimmerman et al. (1986), Sturman et al. (1989), Quek (1994) and Wexelblat (1995), received a small amount of attention from researchers throughout the 1980s and early 1990s but this research was limited due to the large expense and technical requirements of the sensor technology. It was not until Freeman and Weissman (1995) first demonstrated a camera based system that enabled gestures to control the volume and channel functions of a television that the field of computer-vision rapidly started to grow. For the next decade, the HCI gesture recognition community could be generally segmented into two

[4] explicitly programming a machine to look for value a followed by value b and then value c

research categories, glove based recognition systems and computer-vision based recognition systems.

2.2.1 Glove Based Recognition Systems

Glove based recognition systems, of which excellent reviews can be found in Sturman and Zeltzer (2002) and Dipietro et al. (2008), provided the advantage of accurate tracking of the fingers and the hands position and orientation. They still suffered, however, from the limitations of the glove-based technologies in the 1980s in that they were often too expensive or physically bulky to prove a main-stream commercial success. A number of commercially available products were manufactured however, such as the Sayre glove, MIT LED glove, Digital Data Entry Glove, DataGlove, Dexterous HandMaster, Power Glove, CyberGlove and Space Glove (Dipietro et al., 2008).

2.2.2 Computer-Vision Based Recognition Systems

Computer-vision (**CV**) based systems, of which comprehensive reviews can be found in Moeslund et al. (2006), Erol et al. (2007) and Wachs et al. (2011), have a number of advantages over glove based systems in that the sensor technology is non-invasive and can be relatively cheap to purchase. CV does, however, create a number of difficult problems that need to be solved before any recognition system can be used, such as inferring the pose and motion of a highly articulated and self-occluding non-rigid 3D object from images (Moeslund et al., 2006). Human motion capture is one area of research that has become increasingly active within CV, despite this flurry of work though the number of main-stream commercial CV based recognition systems is still limited. The main-stream application of gesture recognition systems was finally achieved in 2006 by Nintendo with the launch of the Wiimote which was primarily based not on CV, but on an Inertial Measurement Unit.

2.2.3 Inertial Measurement Unit Based Recognition Systems

An Inertial Measurement Unit (**IMU**) is a device that measures an object's velocity, orientation and the gravitational forces created within it as it is moved using a combination of sensors such as accelerometers, gyroscopes and magnetometers. The primary advantage of an IMU based system is that it can be built very cheaply, therefore making the main-stream commercialisation of such a system possible. The small size and low cost of IMUs also makes them applicable for use in novel ways such as within mobile phones, such as the work in Patel et al. (2004), Cho et al. (2007), Gillian et al. (2009) and

Savage et al. (2010), or PDAs, such as the work in Harrison et al. (1998), Bartlett (2000), Hinckley et al. (2000), Eslambolchilar et al. (2004), Strachan et al. (2007), Murray-Smith and Strachan (2008), and many more.

IMUs have several advantages over CV based systems as they are not effected by poor lighting conditions and are small, light, they are not constrained to the space viewable by a camera, and they are robust enough to be easily attached to a user's person - thus making them particularly useful for a musician in a performance scenario. One disadvantage of an IMU based system is that is it can be difficult to estimate the exact position of the device because of the sensors employed within it. Nintendo overcame this limitation by combining the accelerometer sensors with a simple infrared LED tracking system which tracks two infrared beams emitted from a 'sensor bar' that the user places either above or below their television set (Turner, 2007). The integration of IMUs within mobile phones combined with the hacking of devices such as the Nintendo Wiimote has helped generate an active research interest into IMU based recognition systems. The Wiimote, for example, has been used as the primary sensor device in a large number of gesture recognition systems, such as Sreedharan et al. (2007), Kratz et al. (2007), Williamson et al. (2007), Rehm et al. (2008), Lee (2008), Fitz-Walter et al. (2008), Wu et al. (2009), Kratz and Rohs (2010), Hoffman et al. (2010) and Wang et al. (2010) to name but a few.

2.2.3.1 Custom Made IMU Recognition Systems

The low cost and small scale of inertial sensors has also encouraged a number of researchers to develop their own specific IMU based recognition systems as a means of providing research platforms that are independent from the hardware and software constraints of commercial products. Benbasat and Paradiso (2002), for example, developed a six-axis IMU system consisting of a two two-axis accelerometer and three single-axis gyroscope that could fully capture three-dimensional motion. This IMU device was combined with a gesture recognition algorithm that could analyse the data as simple motions, such as straight lines, twist, etc., with magnitude and direction. The recognition of these simple motions, which Benbasat refers to as *atomic gestures*, could then be combined together in a simple scripting application to enable the recognition of full composite gestures which could be connected to trigger a specific task or routine once recognised.

Keir et al. (2006) created 3motion, a 3D gesture interaction system consisting of a three-axis accelerometer and integrated bluetooth unit. This transmits a continuous data stream to a general-purpose gesture interaction software running on a host device

that matches the data against a library of 3D gestures to trigger actions. Junker et al. (2008a) designed an IMU based recognition system for the detection of routine user activities. The recognition system used a Hidden Markov Model (**HMM**) to recognise user activities such as pressing a light switch or drinking a glass of water. Prior to a window of data being classified using the HMM, a window of data was first divided into motion segments and compared to a number of templates. If the system considered a motion segment as possibly being part of gesture it was then sent to the HMM for classification. Pylvänäinen (2005) also used a HMM to recognise a user's hand gestures captured by a 3D accelerometer embedded in a handheld device.

Finally, Liu et al. (2009) developed *uWave*, an accelerometer-based personalised gesture recognition system that uses Dynamic Time Warping (**DTW**) to recognise a large selection of gestures for user authentication and interaction on 3D mobile interfaces.

2.2.3.2 IMU Recognition Algorithms

In terms of the classification algorithms used to recognise gestures captured by IMU based systems there is still no 'perfect' algorithm, although a small number of researchers have reported excellent classification results of 95% and above (Junker et al., 2008a, Wu et al., 2009, Liu et al., 2009, Li, 2010). Although no single recognition algorithm has been named as the most suitable for the classification of dynamic gestures captured by an IMU system, there are a number of algorithms that have consistently achieved excellent recognition results. These are Hidden Markov Models (e.g., Chen et al., 2003, Ward et al., 2005, Junker et al., 2008a, Park and Lee, 2011, Pylvänäinen, 2005, Al-Rajab et al., 2008, Just and Marcel, 2009,Whitehead and Fox, 2009), Dynamic Time Warping (e.g., Forbes and Fiume, 2005, Heloir et al., 2006, Liu et al., 2009, Leong et al., 2009) and Support Vector Machines (e.g., Wong and Cipolla, 2006, Wu et al., 2009). The recognition of IMU sensed gestures on mobile devices has also recently inspired the development of novel recognition algorithms that are fast, accurate and yet require little memory or processing over-heads to operate, such as the $1 algorithm (Wobbrock et al., 2007), $3 algorithm (Kratz and Rohs, 2010) and Protractor algorithm (Li, 2010).

2.2.4 Gesture Recognition for HCI Summary

Gesture recognition has been used throughout HCI for over half a century, however, it has only been in the last decade that gesture recognition based systems have been successfully integrated into commercial applications. This has been made possible by the ever decreasing cost of sensor devices combined with the extremely powerful machine learning algorithms that have been specifically developed for gesture recognition. Novel

interfaces like touch screens, IMU-based sensors, such as the Wiimote, and consumer priced depth sensors, such as the Microsoft Kinect[5], have made gestural control a viable interaction paradigm. The next section looks at how gesture recognition has been used to enable a musician to interact with a computer.

2.3 Gesture Recognition for Musician Computer Interaction

Machine learning algorithms have been successfully applied to a number of tasks throughout many areas of musician-computer interaction. Lee et al. (1992) and Fels (1995) were some of the first to apply the broad history of machine learning research on Artificial Neural Networks (ANN) to the field of MCI. Lee used an ANN to map the input from a radio baton, sensor glove or a MIDI keyboard to audio output and Fels mapped the input from a Cyberglove, 3-D tracker and a footpedal to a speech synthesiser. Fels work was later extended by Pritchard and Fels (2006) who used several ANN to allow the user to synthesise audio, speech and song in real-time. Modler et al. (2003) also applied an ANN to map the sensor data captured by a sensor glove to continuously control the parameters of a synthesis engine running in SuperCollider. Along with applying the ANN to continually map the glove data to synthesis parameters, Modler also used the ANN to recognise patterns in the glove data, such as the classification of certain hand postures like thumbs up or an extended index finger. The recognition of a specific symbolic hand gesture could then be used to trigger a sound, with the energy of the finger movement being mapped to control the damping factor of a plate model. Cont et al. (2004) created a number of ANN blocks for the Graphical User Interface (GUI) program Pure Data (Pd) that allowed a user to quickly train the system to recognise dynamic temporal gestures sensed by two perpendicular accelerometers. The network was trained using six constant speed circle gestures and was able to satisfactorily recognise a large variety of circles performed at different speeds and sizes.

A number of computer-vision based recognition systems have been created for the recognition of musical gestures, such as EyesWeb which contains a suite of software tools that have been applied to the classification of gestures on both basic (syntactic) and advanced (semantic) levels (Camurri et al., 2005). EyesWeb has also been used extensively to analyse dancers' movements, with the recognition of a specific gesture being used to control various musical parameters (Camurri et al., 2004). Nash and Blackwell (2008) used a motion capture system, consisting of a Vicon system using 8 cameras with markers placed on a pair of gloves and a belt that the user would wear, to recognise hand

[5]http://www.xbox.com/en-US/kinect

and body gestures that enabled a user to interact with a real-time polytempi notation system.

Merrill and Paradiso (2005) built the FlexiGesture, a two handed device that features a number of sensors including 3-degree-of-freedom (DOF) accelerometers, 3-DOF gyroscopes, 4-DOF squeezing, 2-DOF bending and 1-DOF twisting. The user could train the system to recognise up to 10 temporal gestures by pressing a 'trigger' button which starts the data recording process, releasing the button when they had completed the gesture. The system then asked the user to continually re-perform the gesture as it trained a template model for that gesture. Dynamic Time Warping (DTW) was used as the recognition algorithm and tests showed that the system was able to classify novel gestures into one of 10 classes with up to 98% accuracy. Bettens and Todoroff (2009) also used DTW to classify a performer's gestures captured by two wireless sensors, each containing 3-DOF accelerometers and 2-DOF gyroscopes, placed on the ankles of the performer. Bettens and Todoroff (2009) implemented their DTW algorithm within Max/MSP and Pure Data enabling the algorithm to be used to classify musical gestures in real-time.

Fiebrink et al. (2009) created a real-time, on-the-fly machine learning-based system called the Wekinator that could be trained by the user in a number of seconds. The Wekinator allows the user the ability to quickly experiment with input/output mappings and even form judgements on the quality of the mapping by training and running it in real-time and observing the sonic results. The system was used for a live performance in which 6 performers started the training/mapping process from scratch, live on stage, and each performer gradually converged on the mapping setup they wanted as the piece progressed. Fiebrink et al. (2009) extended this work by adding an additional 'play-along' paradigm to the Wekinator in which the user listened to a specific piece of music whilst mimicking the gesture they would have liked to have performed to make that sound. The system was then trained on this gesture-sound relationship and the user was able to create a sound or effect by performing the corresponding gesture.

Bevilacqua et al. (2007) developed a real-time continuous gesture recognition system for Max/MSP in which a Hidden Markov Model could continuously output, not only the likelihood of the user performing a given gesture at the current time, but also, where in that gesture the user might be. One of the main benefits of this system is that it has been specifically designed to be trained with the minimum possible training examples (in some cases even 1 example can be sufficient) (Bevilacqua et al., 2009). Bevilacqua et al. (2005) also developed the MnM toolbox for Max/MSP which is dedicated to mapping between gesture and sound, applying algorithms such as Principal Component Analysis (PCA) to reduce the dimensionality of the data, thus simplifying the mapping procedure.

A number of researchers have focused on capturing the natural gestures performed on acoustic instruments such as Overholt et al. (2009) who added a number of algorithms from the OpenCV library to their Multimodal Music Stand System (MMSS) to recognise the gestures of a flutist and used these to control a Max/MSP patch. Morales-Mazanares et al. (2005) also tried to recognise the gestures of a flautist, using a probabilistic model to estimate what the attacks or angular displacement of the instrument could infer about the player's gestures. The accurate classification of violin bowing gestures has also received attention from Rasamimanana et al. (2006), Young (2008) and Fiebrink (2011).

Finally, the recognition of a conductors gestures has given rise to a large body of research, evolving from the seminal work by Mathews (1991b) who created the Radio Baton, a low-frequency radio transmitter mounted at the end of a baton which was sensed by an array of receiving antennae. This enabled the position of the baton to be accurately tracked, with a simple thresholding algorithm being used to recognise the conducting 'beat' gestures. A number of systems followed Mathews' work that also used customised batons and thresholding recognition algorithms, such as in the work by Bertini and Carosi (1993) and Marrin (1996). Computer vision based systems have also been specifically developed for the classification of a conductor's gestures (e.g., Morita et al., 1991, Murphy et al., 2004, Tarabella, 2005, Friberg, 2005 and Ilmonen, 2003). Several other sensor technologies have been applied to capture a conductor's gestures, such as a custom-built conductor's jacket (e.g., Nakra, 1999), accelerometers (e.g., Sawada and Hashimoto, 1997, Dillon et al., 2006 and Bradshaw and Ng, 2008) and gyroscopes (Hofer et al., 2009). Several researchers have progressed beyond simple threshold based recognition algorithms and applied a number of machine learning algorithms to the recognition of a conductor's gesture, such as HMMs (e.g., Brecht and Garnett, 1995, Wilson and Bobick, 2000 and Kolesnik and Wanderley, 2004) and ANNs (e.g., Ilmonen, 2003 and Bruegge et al., 2007), which were used to detect a conductor's beat patterns, recognise amplitude indicative gestures and classify nuances such as volume up, volume down.

2.4 Musical Gestures

The work highlighted in section 2.3 has shown that machine learning algorithms have been successfully applied to the recognition of musical gestures in a wide variety of applications throughout musician-computer interaction. But what exactly are musical gestures? As Cadoz and Wanderley (2000) have illustrated, the term gesture is difficult to define; highlighting, from examples across a range of disciplines, the contrasting and

sometimes contradictory use of the term gesture. Zhao and Badler (2001) and McNeill (2000) have, however, tried to define a general framework that considers gestures from the viewpoints of communication, control and metaphor.

1. **Communication** is involved when gestures work as vehicles of meaning in social interaction. This use of the term is common in linguistics, behavioural psychology and social anthropology.

2. **Control** is involved when gestures work as elements of a system, such as in the control of computational and interactive systems. This is common in the fields of human-computer interaction, computer music and similar areas.

3. **Metaphor** is involved when gestures work as concepts that project physical movement, sound or other types of perception to cultural topics. This use of the term is common in cognitive science, psychology, musicology and other fields.

Musical gestures pose an interesting problem in that they can be seen to cover all three of the above areas. For example, musicians often use body gestures made by the head, arms, hands or entire body to communicate with other performers live on stage. Alternatively, a musician could use a specific hand gesture to control the amount of reverb on their instrument or to trigger a computer to play a certain sample. Likewise, musicians also use gestures to project metaphors to the audience, such as when a pianist performs an aesthetic 'follow-through' hand movement after playing the final chord of a piece. Based on work by Gibet (1987), Cadoz (1988), Delalande (1988), Wanderley and Depalle (2004b); Jensenius et al. (2009) divide musical gestures into four functional categories:

1. **Sound-producing gestures**, also called instrumental gestures (Cadoz, 1988) and effective gestures (Delalande, 1988), are those that physically produce sound. They can be further subdivided into **excitation**, the process of instigating a sound, such as hitting a piano key, and **modification**, the process of changing the qualities of sound, such as changing the pitch of a stringed instrument.

2. **Communicative gestures**, also called semiotic gestures (Wanderley and Battier, 2000), are intended mainly for communication and can be viewed in the same way Kendon (2004) and McNeill (1992) use the term. Communicative gestures can be further subdivided into **performer-performer** and **performer-perceiver** types of communication.

3. **Sound-facilitating gestures**, also referred to as accompanying gestures (Delalande, 1988), non-obvious performer gestures (Wanderley, 1999) and ancillary

gestures (Wanderley and Depalle, 2004a), support the sound-producing gestures in various ways. They can be further divided into **support**, **phrasing** and **entrained** gestures.

4. **Sound-accompanying gestures** are not involved in the sound production itself, but instead follow the music, for example, dancing. They are typically highly expressive gestures and commonly either mimic the sound-producing gestures, like air-guitar gestures, or involve movements that follow or 'trace' the sound.

These four functional categories and their sub-categories are illustrated in Figure 2.6.

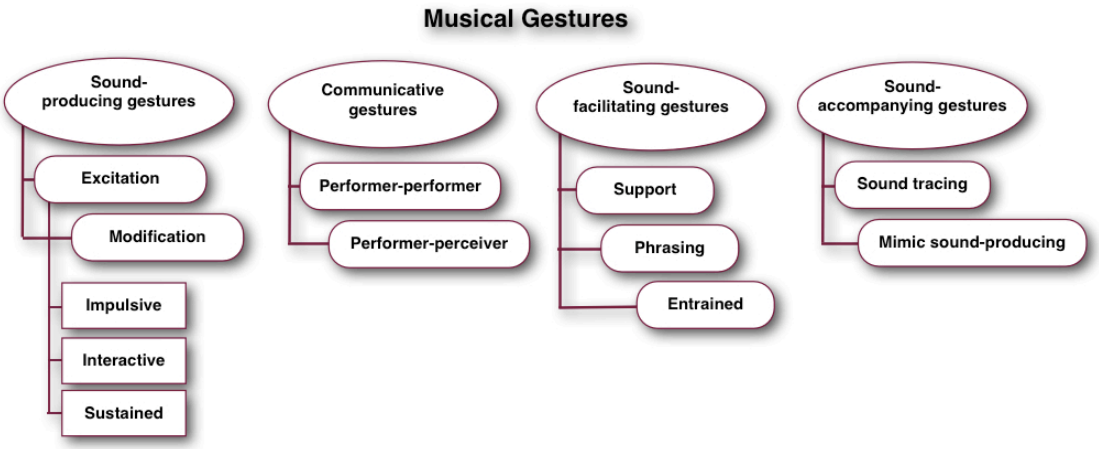


FIGURE 2.6: An illustration of the four functional categories of musical gestures as proposed by Jensenius.

2.4.1 The Musical Gesture Spectrum

It is clear from these attempts to define gesture, that the term can be interpreted in a large number of ways. Musical gestures in particular pose a huge challenge to define as they cover a broad spectrum of movements from the easily interpretable communicative semiotic gestures, such as an "OK" gesture, extending to the other end of the spectrum to highly expressive gestures which can be highly ambiguous. Expressive gestures, unlike highly communicative semiotic gestures, are ambiguous as they carry what Cowie et. al. call *implicit messages* (Cowie et al., 2001) or what Hashimoto calls KANSEI (Hashimoto, 1997). With an expressive gesture, unlike a semiotic gesture, *how* a movement is performed is equally as important as *what* movement was performed.

2.4.2 Higher Order Recognition

In many areas of HCI, the recognition of a gesture occurring is the primary concern. For music however, the goal maybe not to just recognise that a particular gesture has been performed, but to also recognise *how* that gesture was performed. For example, in 1990 Kurtenbach and Hulteen (Kurtenbach and Hulteen, 1990) defined a gesture as:

> *a motion of the body that contains information. Waving goodbye is a gesture. Pressing a key on a keyboard is not a gesture because the motion of a finger on its way to hitting a key is neither observed nor significant. All that matters is which key was pressed.*

For a musician however, the motion of the finger may indeed be significant. The musician may want to know, not only that the key was pressed, but importantly how it was pressed as this information could, for example, instruct the computer to trigger sample x with a specific attack and decay (a slow attack and decay if the key was pressed 'gently' or a fast attack and decay if the key was pressed 'aggressively'). It is common to see in laptop ensembles, for example, the performers sitting behind their computers gesticulating enthusiastically in certain sections of a piece as they trigger samples and effects using the keyboard. In this instance, their gestures have no control value (as the laptop is only aware that a key has been pressed) and such gestures therefore serve as an enhanced metaphor for the audience or as a communication method to other members of the ensemble. By combining extra sensors, such as accelerometers worn on the performer's hands, with the laptop's keyboard the performer could train the computer to recognise what type of hand gesture was made prior to a particular key being pressed. Extracting second-order parameters from a gesture, as opposed to simply recognising the gesture itself, could therefore be used to give the performer an additional dimension of real-time control.

This concept of parsing multiple degrees of information from one gesture via higher order recognition can be utilised throughout the musical gesture spectrum, from clearly defined semiotic gestures to ambiguous expressive gestures. For example, a deictic gesture such as "put that there" could provide two degrees of information, namely the object and the location the object should be placed. In a musical context, a conductor could direct a specific hand posture towards a group of performers in an ensemble, indicating to those musicians that they should perform whatever task was related to the current hand posture of the conductor. At the other end of the musical gesture spectrum, a pianist could perform a slow graceful follow-through gesture after playing a chord which could instruct the computer to diffuse the chord in a specific pre-defined manner, using the

velocity profile of the gesture to control for example an envelope placed on the diffused sounds. The exact relationship between what another performer (or computer) should infer based on the recognition of a particular gesture is open to interpretation by the performers and is what Wanderley and Battier (2000) refers to as a *Gestural Vocabulary*, also commonly referred to a *Gestural Lexicon* in certain fields of HCI and linguistics.

2.4.3 Musical Gestures Summary

A universal definition of the term *gesture* has still to be defined, despite the large body of work covering multi-disciplines. The difficulty in creating a universal definition of the term highlights how a gesture is contextually and culturally dependent. Musical gestures add an extra layer of complexity and ambiguity towards creating a universal definition of what a gesture might mean or infer as musical gestures cover both semiotic gestures, which are highly communicative and easily interpretable, to expressive gestures, which are highly ambiguous. The work in this thesis is based on the concept that a musical gesture taxonomy does not need to be universally defined. This facilitates a musician in creating their own unique definition of what *they* want their own musical gestures to infer. This enables a performer to define their own gestural vocabulary that might be used regularly with a group of other musicians or perhaps only once for a specific solo piece.

2.5 Summary

This chapter has provided an overview of machine learning, presenting some key terminology that will be used throughout the remainder of this thesis. It also provided a review of the previous work conducted on gesture recognition within the field of human-computer interaction, focusing specifically on the application of machine learning for the recognition of musical gestures. The next chapter establishes the theoretical foundations on which this thesis is based along with providing a workeable definition of the term musician-computer interaction. This is followed by a discussion of gestural recognition as a control paradigm for music along with providing some motivation as to why a performer might want to adopt a machine learning approach in order to teach a machine to recognise their gestures.

Chapter 3

Gesture Recognition Systems for Musician Computer Interaction

Without music, life would be a mistake.

Friedrich Nietzsche

This chapter sets out the theoretical foundations on which the remainder of the thesis is based. It first describes in detail the area of research defined as musician-computer interaction. This is followed by a discussion of why gestural interaction is a useful control method for a performer and a discussion of why a musician may want to adopt a machine learning approach in order to teach a machine to recognise their gestures. The common design, development, training and evaluation strategies for gesture recognition systems are then reviewed and the strategies of such systems for HCI and MCI are differentiated. The chapter is concluded by presenting the SARC EyesWeb Catalog, a machine learning toolbox that has been specifically designed for musician computer interaction.

3.1 Musician Computer Interaction

Musician-computer interaction (**MCI**), a subfield of the larger research area of human-computer interaction (**HCI**), focuses specifically on technology that can enable musicians to interact with computers. Music provides a number of interesting research challenges over and above the more general field of HCI as, due to its real-time musical application, a low-latency highly robust user-configurable system is required. Conventional HCI devices, such as the keyboard and mouse, offer only limited scope for a musician to control their audio software, particularly in a real-time performance scenario where the user

may want to have both discrete and continuous control over multiple parameters at the same time. A performer, for example, may want to trigger on/off a number of samples (discrete control) while at the same time continually modulating the cut-off frequencies of a number of filters (continuous control). It is this, somewhat contradictory, requirement for fine-grain simultaneous control of multiple parameters that makes designing interfaces for MCI such an interesting and challenging research area.

3.1.1 Existing Commercial Interfaces

In order to achieve this level of fine-grain multiple parameter real-time control; a large number of specifically designed commercial pieces of hardware and software have been developed. Hardware devices, like the Akai APC40 USB Performance Controller or the Korg MicroKontrol MC1 (shown in Figure 3.2), combined with software programs, such as Abelton Live[1] or Max/MSP[2], enable a musician to interact with a computer in a real-time performance scenario in ways that would not be possible with more conventional HCI devices. This is because hardware devices, like the APC40, provide a musician with both multi-functional discrete and continuous control in the form of toggle buttons, sliders and knobs. Dedicated MCI hardware devices also importantly provide the opportunity for the musician to specifically map the output from the device to the input of the music software program being used, as illustrated in Figure 3.1.

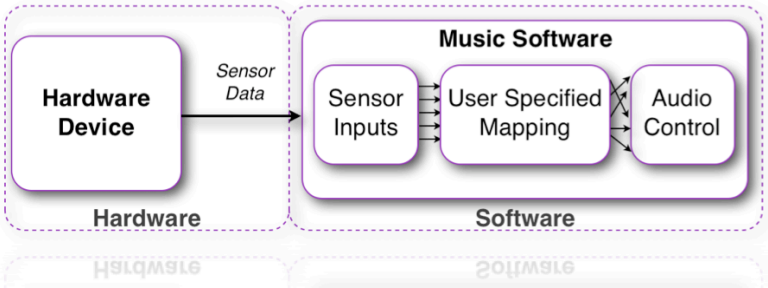


FIGURE 3.1: In MCI, the user will typically want to create a custom mapping between the hardware device and the music software it is connected to.

Using communication protocols such as Musical Instrument Digital Interface (**MIDI**) or Open Sound Control (**OSC**) (Wright and Freed, 1997,) the status value of a button or continuous value of a slider can be mapped by the user to control any audio setting or parameter the performer wishes. This is a key factor in the design methodology behind both hardware devices and software programs for MCI as a performer may desire the functionality to customise both their hardware and software for their own specific

[1] http://www.ableton.com/live
[2] http://cycling74.com/products/maxmspjitter/

musical requirements. This degree of customisation could range from the basic mapping of buttons and sliders in a mixer to match the corresponding controls in a software program, such as Pro Tools[3], to the user-specific mapping of a device to control an audio program the user themselves may have designed, for example using software like SuperCollider[4] or Chuck[5]. This level of user customisation that occurs in MCI is one of the main aspects that differentiates hardware/software design in MCI from the more general field of HCI.

(a) The Akai APC40 USB Performance Controller (b) The Korg MicroKontrol MC1

FIGURE 3.2: Two MCI controllers that give the user the ability to have both multiple parameter discrete and continuous real-time control.

3.1.2 New Interfaces for Musical Expression

The trend toward user-specific systems is evident well beyond the mainstream commercial controllers, such as the APC40 or MicroKontrol, as a large body of musicians regularly experiment with both designing and developing new MCI hardware and software systems. Annual conferences, such as the New Interfaces for Musical Expression (**NIME**) conference[6], the International Computer Music Conference[7] (**ICMC**) and the Sound and Music Computing (**SMC**) conference[8], all feature dozens of examples each year of new developments in hardware and software that have been specifically designed for MCI.

Free, customizable music-software, such as Pure Data[9], SuperCollider or Chuck, facilitates performers to create their own uniquely-tailored audio systems. A large number

[3] http://www.avid.com/US/products/Pro-Tools-Software
[4] http://supercollider.sourceforge.net/
[5] http://chuck.cs.princeton.edu/
[6] http://www.nime.org/
[7] http://www.computermusic.org/
[8] http://smcnetwork.org/
[9] http://puredata.info/

of performers are now also making use of the cheap open-source electronics platforms, like the Arduino[10], to create their own custom built sensor interfaces that can be used to gain real-time control over their specific audio systems. The combination of the ever-decreasing cost of sensor devices along with the hacking of existing motion controllers, such as the Nintendo Wiimote[11] or Microsoft Kinect[12], have now made it possible for a large number of composers, performers and researchers to use the data from these sensors as controllers for their music software. This has made accessible an exciting interaction paradigm, that was previously only feasible for a minority of researchers and engineers, in enabling a performer to use their own body gestures to interact with a computer. This trend is further supported by the fact that most undergraduate and graduate level courses in the broader area of 'music technology' now typically include modules on interaction design for music that provide student performers, composers and sound engineers with the skills to design and build their own digital musical instruments. Examples of these courses include SARC's Live Performance Systems module, CCRMA's HCI Performance Systems for Music course[13] and Princeton's Human-Computer Interface Technology course[14].

3.1.3 Gestural Interaction

Gestural interaction is of particular use to a musician as it enables them to control a specific parameter or effect, even if their hands are busy playing an instrument. Gestural interaction enables a musician to use aesthetic, expressive gestures to control a computer which is of great benefit in a live performance scenario. One key advantage of using gestural interaction for MCI is that it could enable the performer to control multiple parameters of a sound, such as pitch, timbre and on-set amplitude, simultaneously. This real-time control over multiple degrees of freedom is even difficult with current commercial MCI devices, therefore making gestural control a rewarding research area.

Gestural interaction would facilitate a musician to augment their own acoustic instrument with additional sensors, enabling them to control a synthesis program in real-time by performing a number of musical gestures, creating what Wanderley calls a Digital Musical Instrument (**DMI**) (Wanderley and Battier, 2000). It would also importantly enable a performer to control the synthesis program on a machine without using any physical instrument at all; allowing the musician to play what Mulder calls a Virtual

[10] http://www.arduino.cc/
[11] http://www.nintendo.com/wii
[12] http://www.xbox.com/en-GB/kinect
[13] https://ccrma.stanford.edu/course-overviews/music-250b
[14] http://www.cs.princeton.edu/courses/archive/fall07/cos436/

Musical Instrument (**VMI**) (Mulder, 1994). Alternatively, gestural interaction could facilitate a performer to use musical conducting gestures to simultaneously interact with a number of performers and a computer with one succinct movement.

3.1.4 Teaching A Machine To Recognise Musical Gestures

But how can a performer control a machine through the medium of musical gestures? To facilitate a performer to interact with a computer using musical gestures, the computer must be able to first ***sense*** the gestures, secondly ***recognise*** the gestures and finally know how to ***respond*** to the gestures. Thankfully, due to the ever-decreasing cost of sensor devices along with the hacking of existing motion controllers, performers now have access to the technology required to sense the vast majority of musical gestures. The task of instructing a computer how to respond to the recognition of a gesture has also been greatly simplified, thanks to the inherent flexibility of the existing pieces of real-time composition and performance software, as the computer just needs to know which process to trigger or manipulate when gesture x has been recognised. This just leaves the task of teaching a computer how to recognise a gesture.

In reality, the musician's goal is not to get the computer to 'understand' the musical gesture but instead the objective is to somehow teach the computer to recognise the underlying pattern that occurs in the sensor data when a given musical gesture is performed. The computer will then know that it should perform task x the next time it detects pattern y in the sensor data. So how can a computer be 'taught' to recognise the patterns contained in sensor data that might be associated with the performance of a musical gesture. One approach would be for the musician to perform the musical gesture, study the associated sensor data and then manually program a simple recognition algorithm that will trigger task x when the sensor data passes a given threshold value. Setting this threshold value manually may be a trivial matter if only one sensor is being used or if just one gesture needs to be recognised; however this quickly becomes a complex task if more than one sensor is being used, several threshold values are required or multiple gestures need to be recognised. The complexity of the task increases further if more intricate patterns need to be recognised, particularly patterns that evolve over time or in a multi-dimensional space. It is in these instances that a performer may want to adopt a machine learning approach in which the computer 'learns' how to recognise a gesture by directly analysing a number of examples of the sensor data that is associated with each gesture.

3.1.5 Adopting A Machine Learning Approach

By adopting a machine learning approach, a musician can teach a computer to recognise a set of G gestures by creating a training set consisting of a number of recordings of the musician performing each gesture (captured by whichever sensor device the performer thinks is most appropriate for the recognition task). The training set is then used to tune the parameters of an adaptive model using a machine learning algorithm with the objective that, once trained, the model can correctly output a specific value when given a specific input value. This could be a discrete value representing one of the G gestures in a classification task or a continuous variable in a regression task. If the appropriate sensor(s) or feature(s) are used to represent each gesture and a suitable machine learning algorithm is trained then the algorithm should be able to classify a new input vector as one of the G gestures it was trained with; even if the new input vector was not contained in the original training set. This would therefore enable a musician to use the classification abilities of a trained machine learning algorithm to recognise the performer's musical gestures in real-time live on stage.

3.1.6 Applying Machine Learning To MCI

By adopting a machine learning approach, a performer can employ a large number of very powerful tools to facilitate musical gestural interaction. The actual application of these machine learning tools is not, however, a simple task as the implementation of a machine learning algorithm can prove difficult for even proficient software engineers. This is not only due to the mathematical complexity of many machine learning algorithms, but is also related to the ways in which applying machine learning is different to traditional programming. Traditional programming, for example, allows developers to explicitly describe the behavior of a program, whereas systems that use machine learning must learn or infer these behaviors from the training data. Although machine learning libraries, such as Weka, and dedicated development environments, such as Gestalt (Patel et al., 2010), are enabling the integration of sophisticated machine learning algorithms within a developer's software program; this still precludes any musician with limited or no programming skills.

It would therefore be beneficial for a musician to have access to a system that enables the user, regardless of their technical skills, to apply any number of machine learning algorithms to classify their musical gestures. As a classifier rarely exists in a vacuum (Duda et al., 2001), any recognition system that facilitates a musician in using a machine learning algorithm to classify their musical gestures must also allow the user to specify

which feature extraction method should be applied to the raw data or which post-processing method should be applied to the output of the classifier. But how should such a recognition system work? Can the design and training strategies employed for the recognition systems in other areas of HCI be directly applied to MCI? Can a generic one-size-fits-all 'black-box' pre-trained recognition system be created for music? This thesis argues that such a generic recognition system is completely impracticable for musical gestures and that the design and training strategies adopted throughout many areas of machine learning and HCI need to be reevaluated for MCI. Section 3.2 provides the evidence for the reevaluation of the common design, development, training and evaluation strategies for application of gesture recognition systems for MCI.

3.2 Gesture Recognition Design Strategies For MCI

The design, development, training and evaluation of gesture recognition systems for MCI requires a paradigm shift from the common strategies employed in other areas within HCI that also use gesture recognition. This is due to the three main aspects that differentiate the systems that use gestures in MCI compared with that of other areas within HCI:

1. **Input Ambiguity:** For MCI, a musician may want to use many forms of sensing devices, from commercial devices such as a webcam or Wiimote to custom designed hardware. The input to a MCI gesture recognition system, and therefore the input to the recognition algorithms within such systems, may not be known in advance.

2. **Output Ambiguity:** For MCI, a musician may want to use any number of commercial pieces of audio software or could have built their own specific audio software for real-time composition and/or performance. Therefore the software that the output of a MCI gesture recognition system is connected to may not be known in advance.

3. **User-Specific Gestural Vocabularies:** For MCI, the gestural vocabulary (the relationship between a gesture and its corresponding action) being used by the musician could be unique to that performer or could even have been specifically designed for a single instrument or composition.

The common thread that links these points is that because the fundamental components of a musical gesture recognition system are ambiguous, i.e. its input, what it recognises and what it controls, the system cannot be trained prior to being distributed to the community of end-users.

3.2.1 Gesture Recognition Systems for HCI

In contrast, the designers and developers of a gesture recognition system for HCI, such as that used in computer games or in the classification of sign language, will be fully aware of what the input to the system is. They will also know what the output of the system will be controlling, such as the movement of a character in a game, and will know which gestural vocabulary is being used, such as the American Sign Language. This provides the developers of such systems the opportunity to develop, train, test and refine the underlying recognition of each gesture; ensuring that a robust recognition system is distributed to the end-user. Commercial HCI gesture recognition systems need to have excellent generalisation abilities; as the majority of end-users will not have contributed to the training set used to create the underlying recognition models. This means that a large amount of training data, recorded from perhaps hundreds of users, is required to ensure a robust generalisable recognition model.

There are, of course, exceptions to this generalisation of pre-defined gestural vocabularies within HCI. User-defined gestural vocabularies have been applied in areas such as mobile phone security (e.g., Farella et al., 2006) along with an interface design concept that exploits a user's ability to use their own body space as an interface to a mobile device (e.g., Angesleva et al., 2003). These examples, however, represent only a small percentage of areas within the field of HCI that have employed user-defined gestural vocabularies, with the majority of gesturally controlled systems using a pre-defined gestural vocabulary.

3.2.2 Gesture Recognition Systems for MCI

For MCI, however, it would be difficult to create the pre-trained recognition systems that are common throughout HCI. This is because it is almost impossible to establish a pre-trained universal musical gestural vocabulary that works for every composer and performer in every musical context. It would be possible to create pre-defined systems for several commercial music applications, such as using the built in web camera of a computer to recognise the hand gestures of the user to control the playback functionality of a music player. In this example, it would be easy to think of a number of basic hand gestures that could easily be learned by the user to control commands such as play, stop, next track etc. and such a system could therefore be pre-trained and ready for use when purchased.

However, this type of pre-trained system will not be applicable for the majority of other musical interaction scenarios, such as for a live performance. As section 2.4 has

illustrated, even defining the term gesture is difficult as musical gestures cover a broad spectrum of movements from the easily interpretable communicative semiotic gestures, such as an "OK" gesture, to highly expressive gestures which are ambiguous. For music, therefore, a fixed pre-trained universal gestural vocabulary may not be a viable solution, as each composer or performer may want to define their own vocabulary to use for a specific instrument or even just one piece. A fixed, pre-trained vocabulary would also be pointless in this instance as a common sensor platform would be required for the system to recognise any of the pre-trained gestures being performed, rendering any custom piece of MCI hardware useless.

3.2.3 The Intra-personal Generalisation Goal

What would be much more practical for a musician, particularly in a live performance scenario, is a system that features a flexible input/output configuration and that can be trained quickly using gestures from the performer's vocabulary. Such a system would not require the inter-personal generalisation abilities found in other areas of HCI; instead it would simply need to provide a good intra-personal generalisation for the one performer that initially trained the system. If another performer wanted to use their own input device or gestural vocabulary to control the same audio software, then they would simply have to retrain the machine learning system with their own gestures. This intra-personal generalisation goal, which represents a considerable paradigm shift from many areas of machine learning and HCI, would not only offer the performer the advantage of being able to use their own hardware to capture gestures from their own gestural vocabulary and use these to control their own specific audio software, it would also result in the requirement for a lower number of training examples per gesture - leading to a reduced amount of time spent in data collection and computational-training time.

3.2.4 Fast Training, Fast Testing, Fast Prototyping

For MCI, it is essential that the training process of each machine learning algorithm occurs as quickly as possible. An efficient training phase will enable a performer to quickly decide upon a possible gestural vocabulary to use, train the recognition system and then, importantly, test the real-time prediction abilities of the system by performing the gestures and checking whether they have been correctly classified. Testing the system in this manner not only validates if a robust intra-personal generalisation error has been achieved, it also tests the aesthetic and practical validity of the gestures themselves. If a performer is unhappy with either then they can choose to change the feature extraction method or parameters of the machine learning algorithm being used and retrain the

model. Alternatively, if the gestures are either aesthetically or practically unsuitable, the performer could remove one or more of the gestures from the training set and replace them with more suitable movements. In ensemble performances, for example, a group may decide to change subtle communication gestures, and it would be important that an MCI system could adapt to absorb such changes, even at very short notice. A recognition system that can be quickly trained and tested will enable a musician to rapidly prototype any action-sound relationship they think may be useful for a real-time performance scenario, test the validity of such gestures and then focus their time on the musical elements of the performance.

3.2.5 The Bias-Variance Tradeoff

Care must be taken, however, in the design and development of any gesture recognition system or machine learning algorithm that is to be used for MCI. Constraints would need to be put in place, for example, to stop the algorithms under-fitting or over-fitting when a limited number of training examples are used in the models training phase. A general assumption commonly made in the design of many machine learning algorithms is that the variability observed within the training set will roughly estimate the variability of any similar data in a test set. A model should therefore be able to successfully estimate the limits of this variability, and thus correctly estimate the decision boundaries that determine what a new datum is classified as, when a large training set is used. What if, however, a performer knew that the data set they were using to train an algorithm would not accurately represent the inherent variability of that data? For example, if the training set only contained two training examples per gesture for each of the G gestures the performer wanted the algorithm to recognise. In this instance it would be beneficial for the performer to make the algorithm over-estimate the variability of the small training set by weighting the algorithm to favor a high-bias when training the model as opposed to high-variance and thus prevent the model from over-fitting its parameters on the small training set. A parameter that allowed the performer to control the bias-variance tradeoff of an algorithm would facilitate the performer in initially training a model with a very small data set, enabling the performer to quickly validate whether the gestures are appropriate. If the performer is satisfied with their gestural vocabulary they can then choose to record additional training examples for each gesture, reduce the ‘ bias-variance’ parameter to its default value and retrain the algorithm to create a more robust model.

3.2.6 Adaptive Models

An alternative to training a model from random initialised values for each gesture is to use a machine learning algorithm that could, once trained, slowly adapt its model over an extended time period. This would enable the performer to initially train a model with a small training set and set the bias-variance parameter to a value to mitigate under-fitting or over-fitting. The model could then be used to classify the performer's gestures in real-time, while in parallel combining the real-time data with the original training data to continuously retrain and refine the model's parameters. As the size of the training data set increased the bias-variance parameter could slowly be reset to its default value. Care must be taken, however, in the design of an adaptive algorithm. This is because any 'new' training data collected during the online prediction phase will have been self-labeled by the algorithm and therefore a small number of incorrectly labeled training examples could create a 'run-away' model that becomes less effective at each update step. Precautions would therefore need to be put in place to ensure the model did not update itself too quickly to help mitigate against a run-away model.

An adaptive algorithm that increased its generalisation abilities at each update would not only enable a performer to quickly train the system with their own gestural vocabulary, it would also allow the algorithm to adapt its trained model as a musician adapts their own movements. For example, an adaptive algorithm could slowly optimize its trained models over the course of a rehearsal period, refining an algorithm's parameters as the performer slowly refines their own gestures. This would also facilitate new paradigms for musical pedagogy, allowing a musician to play an instrument that also learns in parallel with its user. Over time an algorithm's parameters could be slowly optimized as a performer increases their skill level and knowledge of a particular instrument or piece, providing the performer with more degrees of freedom and more fine-grain control as the player's skill and abilities increase over time.

3.2.7 Error Tolerances

One of the key aspects for the recognition of discrete musical gestures is that a performer will have little tolerance for any classification-errors made by the system. This is because, unlike the continuous mapping of a movement to a sound or parameter, the recognition of a discrete gesture will be used to trigger or control a discrete event. A musician therefore, particularly in a live performance scenario, will not want to have to perform the same gesture repetitively until it is recognised by the system and the corresponding event is triggered. False-positive classification errors will also not be tolerated, as this type of error could drastically affect the performance of a piece, triggering an inappropriate

sample or effect to be played at the wrong time or incorrectly progressing the piece to the wrong section. The machine learning algorithms employed for the classification of discrete musical gestures must therefore, above all else, give very high classification results.

3.2.8 Risk

Along with controlling the bias-variance tradeoff of an algorithm it would also be beneficial for a performer to have control over how an algorithm uses the classification result to make an actual decision, such as whether a sound should be triggered or not. A performer may want to simply train an algorithm to achieve the minimum-error-rate classification, with the 'cost' of the algorithm making a false-postive classification error being equal to the algorithm failing to recognise an actual gesture. However, some performers may want to weight the cost of one type of error over another, creating a recognition system that will minimize the total expected cost, or *risk*, of making a specific type of classification error. A musician, for example, might want a recognition system to be as robust as possible against making false-positive classifications at the expense of the system failing to recognise a small number of their musical gestures. The performer might even want to control a recognition systems risk dynamically, re-weighting the cost of making one type of error at section a of a piece and changing this weighting value at section b of the piece.

3.2.9 Validating An Intra-Personal Classification Algorithm

For MCI, it is important for a performer to evaluate an algorithm both qualitatively and quantitatively. Qualitative testing, where the performer trains a model and then tests the algorithm in real-time by performing each gesture hoping the system will correctly classify that gesture, not only validates the classification abilities of the algorithm but it also tests the aesthetic and practical validity of the gestures themselves. Qualitative testing also evaluates other practical issues such as recognition latency or real-time processing overheads. Quantitative testing, where the performer trains a model with a training set and then tests the model with a test set containing previously unseen data, is important as it helps to estimate the generalisation abilities of the algorithm for a specific recognition problem. The quantitative classification results can then be used to inform other performers or researchers who might have a similar recognition problem as to which is the most suitable algorithm for that given task.

An intra-personal generalisation error would call for a new quantitative method of evaluating the classification abilities of a machine learning algorithm. For example as previously described in chapter 2.1.6, in many machine learning applications, a large amount of data is collected from perhaps hundreds of users and the data is split into a training set and a test set. The machine learning algorithm is then trained with the training set and evaluated with the test set. If the training data is difficult or expensive to acquire, then a hold-out validation method such as K-fold cross-validation is used instead. Both of these validation methods are suitable for estimating the generalisation abilities of an algorithm that will be used in an inter-personal recognition system. These methods are not suitable for validating the intra-personal classification abilities of an algorithm however; as they assume that either a large amount of training data is available and/or the training data from several different users can be combined into one combined data set. A more suitable generalisation metric for MCI would be to use the average cross-validation error (**ACVE**) calculated by independently computing the cross-validation error for N participants and then averaging this result. The ACVE is suitable for MCI because it can accurately estimate the intra-personal generalisation abilities of a machine learning algorithm, while at the same time being validated by a large number of different users to ensure that the estimated generalisation abilities of an algorithm are not misrepresented by one 'good' or 'bad' participant.

3.2.10 Design Strategies Summary

The design, development, training and evaluation strategies for any gesture recognition system that will be used for MCI therefore require a paradigm shift from the common strategies employed in other areas of HCI and the machine learning community. This is to enable a user to create, train, test and refine a recognition system that the user thinks is most suitable to solve their recognition problem. The design strategies for a gesture recognition system for MCI can therefore be summarised as:

- **Design:**
  The algorithms used for the recognition of musical gestures should be designed to work with any N-dimensional signal so that they are not constrained to work with the data or features from a specific sensor. They should also be designed to enable the user to adjust specific settings such as the bias-variance tradeoff or minimize the risk of the system making a specific type of error.

- **Development:**
  A gesture recognition system and the machine learning algorithms within such a system should be developed in a manner that enables the user to configure the

system in whatever way they think may result in the best classification of their gestures for their particular needs. This includes being able to use any sensor as input to the system and route the output of the system to control whatever audio software the performer maybe using. The performer should also have the option to specify their own recognition chain; i.e. choosing a pre-processing method to use on the raw sensor data, the feature extraction algorithm to apply to this data and what features should be input to the machine learning algorithm, and finally to select the post-processing functions that should be applied to the output of the classifier to make the final classification decision. This should all be developed in such a manner that enables a performer with even limited technical or programming skills to use the system.

- **Training:**
  Any machine learning algorithm for MCI should be optimized to be trained quickly using a small number of training examples. Once trained the model should achieve a low intra-personal generalisation error.

- **Evaluation:**
  The machine learning algorithms used for the recognition of musical gestures should be evaluated to test the intra-personal generalisation abilities of the algorithms using a suitable error measure such as the average cross-validation error.

3.3 Creating a Gesture Recognition System for MCI

Using the design strategies outlined in section 3.2, a review and evaluation of the current gesture recognition systems and machine learning toolboxes that were freely available in the MCI, HCI and machine learning communities was conducted to establish whether any of these toolboxes met the criteria. A large number of machine learning software tools exist, either within mathematical analysis programs such as Matlab[15], as standalone libraries that can be integrated into a developer's own software environment such as the WEKA machine learning library (Hall et al., 2009) or the LIBSVM (Chang and Lin, 2001) Support Vector Machines library, or as previously mentioned in 2.3, within higher-level GUI based applications such as Max/MSP and Pure Data. However, despite the large number of machine learning toolboxes available, it was very difficult to find an existing software tool for gesture recognition that had been developed for on-line recognition, was not built for a specific sensor device, such as a mouse or camera, that was not designed for a audio environment and did not have to work with a pre-trained gestural vocabulary.

[15] http://www.mathworks.co.uk/

Out of all the toolboxes and gesture recognition systems that were reviewed, the Wekinator (Fiebrink et al., 2009) was the only software that came close to fulfilling the design criteria outlined in section 3.2. Its main drawback, however, was that it did not contain any recognition algorithms that could be used for the real-time classification of temporal musical gestures.

The lack of a system that fulfilled the design criteria outlined in section 3.2 therefore motivated the design and develop of software that could be used to enable gesture recognition for MCI. Rather than design and develop a recognition system from scratch, it was decided to build upon an existing software environment that would facilitate the addition of the feature extraction and machine learning algorithms that could be used specifically for MCI. After reviewing a number of suitable programs that allow the addition of third party libraries, such as Max/MSP, Pure Data and Simulink, the free program EyesWeb[16] was chosen as the most appropriate base platform to develop a dedicated machine learning toolbox for MCI.

3.3.1 The SEC

The dedicated MCI machine learning toolbox, called the SARC EyesWeb Catalog (**SEC**), has been fully integrated as a third party library within EyesWeb. EyesWeb is an open software platform that was established to support the development of real-time multimodal distributed interactive applications and already features a large number of algorithms for processing both video, audio, and generic data signals (details of which can be found in Camurri et al., 2004 and Camurri et al., 2007). The SEC therefore adds a complementary set of machine learning algorithms to the existing algorithms within EyesWeb.

EyesWeb is a GUI orientated program (that runs on the Microsoft Windows[17] operating system) which features a *patch window* onto which the user can drag a number of *blocks* that represent a specific algorithm or function. A block, as shown by Figure 3.4, will commonly feature a number of input, output and parameter pins, with one block's output pin being connected to another block's input pin to create a signal flow between the two respective blocks. EyesWeb functions in two modes, the *design* mode and the *run* mode. In the design mode, the user can add new blocks to the patch window, delete existing blocks from the patch window, connect one block to another block and so on. In run mode, the data will actually be passed between blocks, sent and received over network connections, processed and analysed. Using a small number of blocks in EyesWeb, for example, a performer could build a patch to capture real-time data from

[16] `http://www.infomus.org/EywMain.html`
[17] EyesWeb is currently being ported to the Linux operating system

FIGURE 3.3: A number of example EyesWeb patches. The three patches shown are all example patches for the SEC blocks that enable a novice user to understand how a block can be used and provide a working demo that the user can interact with.

a sensor unit, filter the data and plot the results without having to write a single line of code, as shown by Figure 3.5. EyesWeb also enables any performer with more technical skills to develop their own blocks, which may be required to perform a specific type of feature extraction or to interface with a certain piece of hardware. All the blocks in EyesWeb are written in c++, giving the developer the ability to write fast, efficient code which is a necessity for real-time machine learning due to the large number of calculations required. EyesWeb therefore provides an excellent environment for both technical and non-technical users as complex signal processing operations can be easily constructed by connecting a number of blocks together or alternatively a custom block can be developed to perform one specific task.

3.3.2 The SEC Blocks

The SEC (Gillian et al., 2011b) features over 80 blocks that have been specifically designed as a result of the work reported in this thesis for the recognition of musical gestures. The SEC blocks are organised into the following 5 categories:

- **General:**
  This contains some rudimentary blocks that can load/save data, cyclical buffers and blocks that convert from one data type to another etc.

- **Math:**
  This contains blocks for a number of mathematically functions such as normalisation and scaling, vector and matrix operations, calculating first and second order derivatives, integration and mapping tools etc.

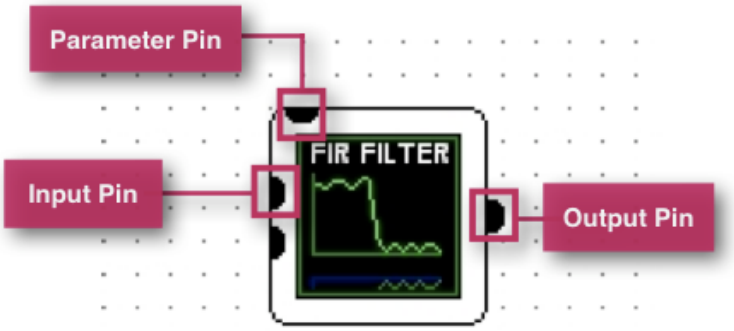


FIGURE 3.4: An SEC FIR filter block, showing the input pin, parameter pin and output pin.

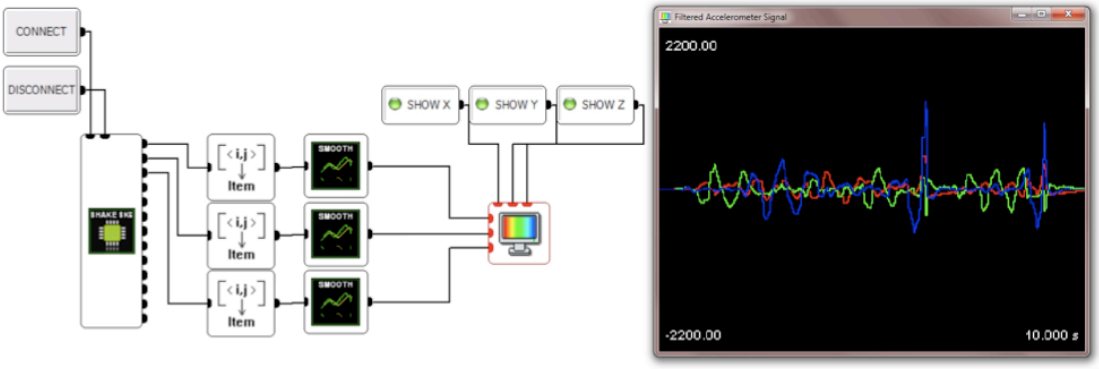


FIGURE 3.5: A basic EyesWeb patch showing how the output from a sensor device (in this case the accelerometer signal from the SHAKE SK6) can be filtered and the resulting signal visualized without the user having to write one signal line of code.

- **Gesture Recognition:**
  This contains a number of blocks that can be used for gesture recognition. This includes some common machine learning algorithms such as: Artificial Neural Networks, Hidden Markov Models, Support Vector Machines, K-Means clustering, Fuzzy C-Means clustering, K-Nearest Neighbor classification and State Machines along with a number of common feature extraction methods such as Vector Quantization, Principal Component Analysis and Symbolic Aggregate Approximation. The gesture recognition category also features a number of algorithms that have been specifically developed for the recognition of musical gestures such as the Adaptive Naive Bayes Classifier (Gillian et al., 2011a) and N-Dimensional Dynamic Time Warping (Gillian et al., 2011c). These algorithms will be the focus of the upcoming chapters.

- **Sensor Units:**
  This contains a number of blocks for interfacing directly with hardware sensor units such as the Wiimote, the SHAKE and Infusion System's Wi-microDig[18].

- **Signal Processing:**
  This contains a number of blocks for performing signal processing operations, such as filtering, dead-zone and envelope extraction.

3.3.3 Using the SEC for MCI

The machine learning algorithms within the SEC enable a performer to use one or more musical gestures to control and manipulate the performer's composition or improvisation software in real-time. For example, a musician could use a classification algorithm such as N-Dimensional Dynamic Time Warping (ND-DTW) to classify a specific conducting gesture and use the recognition of this movement to trigger the computer to start manipulating the live audio recording of the musician the gesture was directed towards. At the same time, the performer could use a regression algorithm like an Artificial Neural Network (ANN) to continuously map the velocity at which the performer made the conducting gesture to control the degree of the warping effect on the live audio recording.

3.3.4 Middleware Design Architecture

Rather than targeting the SEC for just one specific piece of audio software, it has been designed to function as middleware enabling the user to pipe their sensor data

[18] http://infusionsystems.com

into the SEC via a number of standard communication protocols, such as Open Sound Control (**OSC**) (Wright and Freed, 1997). After recognition the classification results can be piped out of the recognition system to control any piece of audio or visualization software that use the same communication protocols, as illustrated by Figure 3.6. A middleware design architecture also enables the SEC to run on an independent machine from that which is running the audio software which is beneficially for CPU intensive recognition algorithms. A performer can therefore write their own software to capture and parse the real-time data from whatever sensor(s) they might be using and pipe this data into EyesWeb via OSC. Alternatively, a performer could directly implement the sensor interface as an additional EyesWeb block.

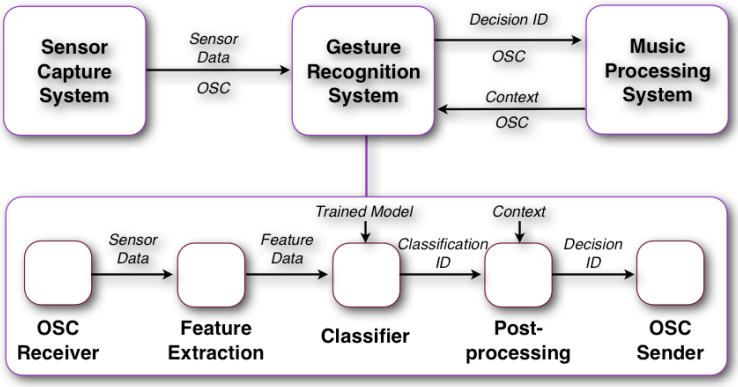


FIGURE 3.6: An illustration of the middleware architecture of a gesture recognition system for MCI. Sensor data can be piped into the system via communication protocols such as OSC, the system can then output the corresponding classification predications via OSC for use in controlling the user's custom audio software.

3.3.5 Creating a Robust Recognition System

A robust recognition system commonly requires an appropriate pre-processing or feature extraction stage prior to any classification by a trained machine learning algorithm, with the predicted classification label being post-processed prior to being acted upon. A user may therefore want to experiment with various feature extraction algorithms or post-processing functions as well as testing which machine learning algorithm works best for the recognition of their gestures. It is for this reason that each feature extraction algorithm or machine learning algorithm has been encapsulated as an individual EyesWeb block as this enables the user to connect the blocks together to create the recognition system the user thinks maybe most appropriate for solving their recognition problem. One of the major advantages of using a patch-based GUI program such as EyesWeb is that multiple recognition algorithms can be used in parallel, with the output of one classifier providing contextual information for another classification chain. For example

the predicted event of one classifier could be used to permit/deny the output of a second classifier being acted upon.

3.3.6 Training a Machine Learning Algorithm

Prior to using any machine learning algorithm it must first be trained. The SEC features a number of useful tools to facilitate a user to efficiently create a training set and then quickly train a machine learning algorithm. Each algorithm, for example, will commonly have a dedicated block for recording training data, a second block for training the algorithm and a third block for the real-time classification of any new data using the trained model. This three block design enables the user to create a specific 'training patch' for recording and training the algorithm and a separate 'prediction patch' for real-time classification that may also contain other trained machine learning algorithms, post-processing algorithms and network connections to communicate with other audio/visual software.

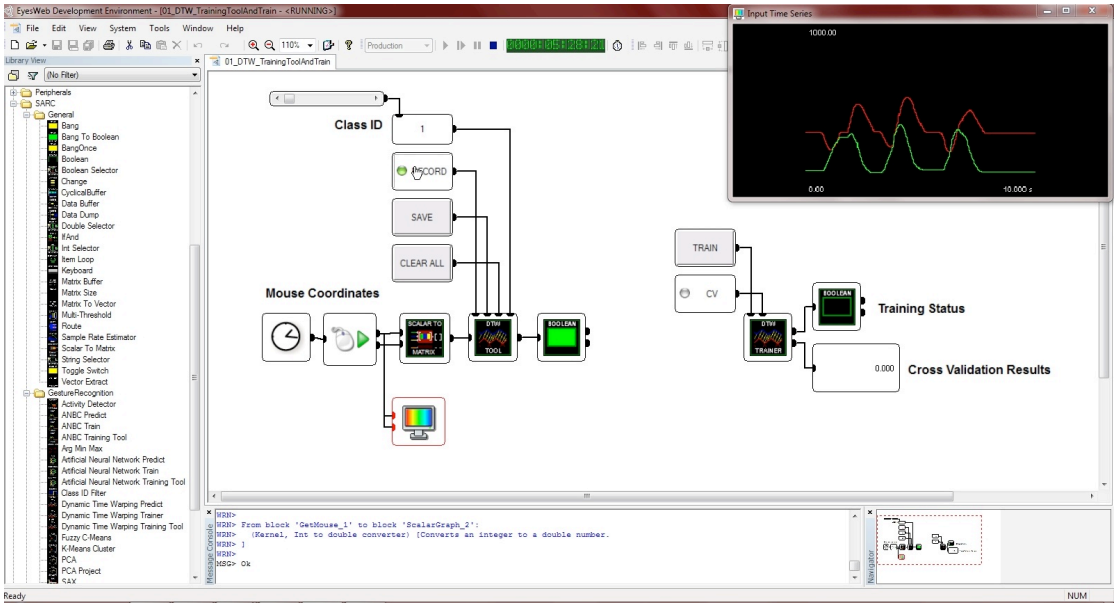


FIGURE 3.7: A training patch for the ND-DTW algorithm.

The training patch, illustrated in Figure 3.7, will have the identical sensor input and feature extraction methods as the prediction patch, see Figure 3.8, but can also contain a number of helpful features that assist the user in collecting and labeling the training data. This could consist of timer functions, for example, that enable the user to press a key to prepare the system to record a two-handed gesture. After a predefined delay the user can then start to perform the gesture while the system records the training data, automatically labeling each training sample with the ID value of that gesture. After a further predefined delay the system stops recording the gesture and the user can either record another example of the same gesture or move onto the next gesture in their

vocabulary. When the user has created a number of training examples for each gesture they can save the training data to a file and then use this to train the machine learning algorithm. Each algorithm will then save its trained model to a file to enable it to be loaded by the real-time classification block.

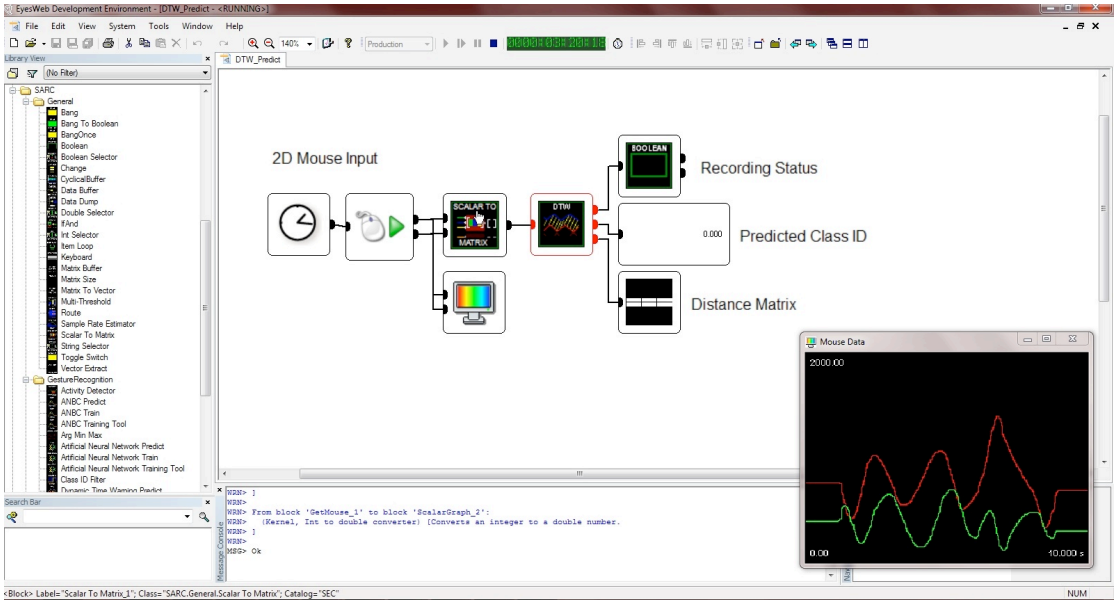


FIGURE 3.8: A prediction patch for the ND-DTW algorithm.

The user can select whether they wish to train the algorithm using an automatic validation method, such as K-fold cross-validation, to estimate the generalisation ability of the trained model or to decide whether they want to devote all of the available data to training the model and instead test the algorithm 'online' using the prediction patch. Either way, if a poor model has been created the user can quickly reload the original training data in the training patch and modify some of the parameters of the machine learning algorithm or even change the feature extraction method and quickly retrain a new model with the updated settings. Alternatively, the performer could use the one training set to train and validate several algorithms each with different settings all at the same time to determine the best features/algorithm/parameters to use. The performer then simply needs to load the best model into the predication patch. These examples illustrate the advantages of using three separate blocks to create a training set, actually train a model and finally perform real-time prediction on new data using the trained model.

3.4 Summary

This chapter has established the theoretical foundations on which the remainder of the thesis is based. It first described the area of research defined as musician-computer

interaction. This was followed by a discussion of why gestural interaction is a useful control method for a performer along with why a musician may want to adopt a machine learning approach in order to teach a machine to recognise their gestures. The common design, development, training and evaluation strategies for gesture recognition systems were then reviewed and the strategies of such systems for HCI and MCI were differentiated. The chapter was concluded by presenting the SEC, a machine learning toolbox that has been specifically designed for MCI. The following chapter presents an algorithm that has been specifically developed, using the design criteria outlined in this chapter, for the recognition of semiotic musical gestures.

Chapter 4

Recognition of Static Semiotic Musical Gestures

Do not worry about your difficulties in Mathematics.
I can an assure you mine are still greater.

Albert Einstein

This chapter presents a novel algorithm called the **Adaptive Naïve Bayes Classifier** (**ANBC**) that has been specifically designed for the recognition of static semiotic musical gestures. The chapter describes how the ANBC algorithm can automatically adapt itself to accommodate a performer as they adapt their own gestures over, for example, the course of a rehearsal period. The chapter concludes with an experiment designed to evaluate the adaptive classification abilities of the new algorithm.

4.1 Semiotic Gestures

Semiotic gestures are body movements intended to communicate meaningful information. Rime and Schiaratura Rimé and Schiaratura (1991) segmented semiotic gestures into four categories:

1. **Symbolic**: These are gestures that, within each culture, have come to have a single meaning. For example, the "OK" hand gesture or the gestures that make up the American Sign Language.

2. **Deictic**: These are the types of gestures most generally seen throughout HCI and consist of pointing gestures or gestures otherwise directing the listeners attention

to specific events or objects in the environment. For example a pointing gesture such as, 'put that there'.

3. **Iconic**: These gestures are used to convey information about the size, shape or orientation of the object of discourse. For example the hand gesture that accompanies the phrase, 'the fish was this big'.

4. **Pantomimic**: These gestures typically mimic an action using some invisible tool or object in the speaker's hand. For example, when a speaker says "I turned the steering wheel hard to the left", while at the same time performs the action of turning an imaginary wheel with both hands.

This chapter focuses on the recognition of those semiotic gestures that consist of a key static posture, as opposed to temporal gestures that consist of a cohesive sequence of movements that occur over a variable time period. Some static semiotic gestures may also contain a preparation phase in which the performer needs to move, for example, their arms into the 'the fish was this big' posture. However, the key phase of the gesture is the final static posture (indicating that indeed the fish was huge) and it is this phase that any classification algorithm needs to recognise. An easy way to differentiate whether a semiotic gesture is a static posture or temporal gesture is if it can be explained to someone with just one image, in the case of a static posture, or if several images need to be used to fully understand the gesture.

4.1.1 Semiotic Musical Gestures

Semiotic gestures, both static postures and temporal gestures, are frequently used throughout many genres in music to enable performers to communicate and cue other performers live on stage. This could consist of subtle looks between players in a trio or more obvious commands of a conductor in front of a choir. The performance artist Lawernce D. 'Butch' Morris (see Figure 4.1), for example, uses a number of semiotic gestures in *conduction*, an improvisation conducting style that he has refined over 20 years, during 150 performances in 23 countries. Using conduction, Morris directs and conducts an improvising ensemble using over 20 hand and baton gestures. Morris can shape in real time nearly every aspect of a performance. This includes controlling changes in tempo, key and pitch; instructing performers to improvise freely or on a certain melody; repeating a passage as an ostinato or riff and instructing the musicians to remember a melody that can be used later as a motif. The automated recognition of semiotic gestures by a machine would enable a performer to interact with a computer in the same manner that Morris interacts with his ensemble.

FIGURE 4.1: The performance artist 'Butch' Morris who regularly uses a number of semiotic gestures in *conduction*, an improvisation conducting style that he has refined over 20 years, to communicate with other musicians in a live performance.

4.2 Designing A Classifier For Semiotic Musical Gestures

Given the frequent occurrence of semiotic gestures within a musical performance and their power in facilitating performer-performer communication, a classifier was designed that could robustly recognise semiotic musical gestures, with the focus of the algorithm to recognise static postures as opposed to semiotic gestures that evolve over time (the recognition of temporal gestures will be addressed in the next chapter). Using the design criteria proposed in section 3.2, a number of design constraints for a semiotic gesture classification algorithm were established:

1. It should be a general purpose, N-dimensional classifier - in other words it should not be designed for one specific sensor or data type but work with any N-dimensional feature vector.

2. It should be able to be trained quickly with a low number of training examples for each gesture in the model.

3. It should automatically calculate a classification threshold for each gesture in the model so that a null-model is not required for real-time continuous recognition.

One additional design constraint was added for the recognition of static semiotic musical gestures. This was that the algorithm, once trained, should be able to automatically adapt itself to provide the user with the best classification results if the user adapts their

own movements. This is particularly useful for a musician as they might define a set of gestures to use at the start of a rehearsal session and then over the course of the rehearsal period slightly modify and refine these gestures. If this was to occur, the user should not have to then retrain the system every 10 minutes to accommodate these small changes in their movements. Alternatively, the algorithm should instead automatically refine its own model to provide the best recognition results for the user. The user can then turn-off this adaptive element of the algorithm when they feel that they are happy with their own gestures along with the classification performance of the algorithm. There are only a small number of examples of machine learning algorithms that are suitable for gesture recognition and can automatically adapt their own models online, such as the work by Licsar and Sziranyi (2005) who developed a vision-based hand gesture recognition system with interactive training aimed to achieve a user-independant application by on-line supervised training. Babu et al. (2010) also created an online adaptive radial basis function neural network for robust object tracking. However, both these algorithms did not fulfill the design constraints for a semiotic musical gesture classification algorithm, as the algorithm was either restricted to use just a video camera as input to the recognition system or a large number of training examples were required because of the complexity of the model being used. A novel algorithm was therefore developed specifically for the recognition of semiotic musical gestures. The algorithm is called the **Adaptive Naïve Bayes Classifier (ANBC)**, and will now be presented in detail.

4.3 Adaptive Naïve Bayes Classifier

The Adaptive Naïve Bayes Classifier **(ANBC)** is a supervised machine learning algorithm based on a simple probabilistic classifier called Naïve Bayes that itself is based on Bayes' theory and is particularly apt for the classification of static symbolic musical gestures. Like a Naïve Bayes Classifier, ANBC makes a number of basic assumptions about the data it is attempting to classify, most significantly that all the variables in the data are independent. However, despite these naïve assumptions, Naïve Bayes Classifiers have proved successful in many real-world classification problems (e.g., Domingos and Pazzani, 1997, Sebe et al., 2002, Li and Anderson-Sprecher, 2006, Lu et al., 2010, Duda et al., 2001). Rish, 2001 has also shown in an empirical study that the Naïve Bayes Classifier not only performs well with completely independent features, but also with functionally dependent features, which is surprising given the algorithm's naïve assumptions. One major advantage of the ANBC algorithm for the recognition of musical gestures is that it requires a small amount of training data to estimate the parameters of each model. This is mainly due to the naïve assumption that each variable in the data is independent, as the parameters for each dimension can be computed independently and

it therefore does not suffer from the 'curse of dimensionality' (Bishop, 2006). The Naïve Bayes Classifier has been specifically updated with an adaptive online training function along with the automatic computation of a classification threshold for each gesture in the model. Prior to explaining the adaptive element of the algorithm, its foundations will first be described.

4.3.1 Bayes' Theory

The Adaptive Naïve Bayes Classifier is based on Bayes' theory:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \tag{4.3.1}$$

which gives the likelihood of event A occurring given the observation of event B. Here $P(A)$ represents the prior probability of event A occurring and $P(B)$ is a normalising factor to ensure that all the posterior probabilities sum to 1.

Using Bayes' theorem, the Naïve Bayes Classifier predicts the likelihood of gesture g_k occurring given the observation of sensor value x:

$$P(g_k|x) = \frac{P(x|g_k)P(g_k)}{\sum_{i=1}^{G} P(x|g_i)P(g_i)} \tag{4.3.2}$$

Note that $P(B)$, the normalising factor, has now become the summation of the likelihood of all the G gestures in the model occurring given the observation of sensor value x. In most real-world applications, $P(g_k)$, the prior probability of observing gesture k, will be equally likely for all the gestures and given by $1/G$ (in which case it could simply be ignored).

Because a Naïve Bayes Classifier makes the naïve assumption that each dimension of data is independent, equation (4.3.2) can easily be extended to calculate the posterior probability of gesture g_k occuring given the observation of the N-dimensional vector $\mathbf{x}$:

$$P(g_k|\mathbf{x}) = \frac{P(\mathbf{x}|g_k)P(g_k)}{\sum_{i=1}^{G} P(\mathbf{x}|g_i)P(g_i)} \tag{4.3.3}$$

where $\mathbf{x} = \{x_1, x_2, \ldots, x_N\}$. As each dimension is assumed to be independent, $P(\mathbf{x}|g_k)P(g_k)$, becomes:

$$P(\mathbf{x}|g_k)P(g_k) = \prod_{n=1}^{N} P(\mathbf{x}_n|g_k)P(g_k) \tag{4.3.4}$$

4.3.2 The Gaussian Density Function

The structure of a Naïve Bayes classifier is determined by the conditional densities $P(\mathbf{x}|g_k)$ along with the prior probabilities $P(g_k)$. For the classification of static musical gestures, the multivariate Gaussian density is a suitable density function to use, particularly in the instance where the feature vector $\mathbf{x}$ for a given gesture g_k is a continuous-valued, randomly corrupted version of a single prototype vector $\boldsymbol{\mu}_k$ (Duda et al., 2001). This is commonly the case for a static symbolic musical gesture, which will feature a specific body pose that will be slightly corrupted by both human and sensor variability, hence why the Gaussian is a good model for the actual probability distribution. Other density functions such as the Radial Basis Function, Cauchy distribution (e.g., Sebe et al., 2002) or Dirichlet distribution (e.g., Wong and Chang, 2011) would also be suitable.

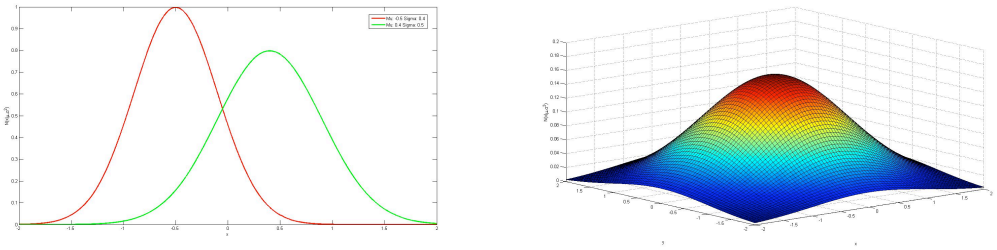
The univariate Gaussian density function, also commonly referred to as the normal distribution, is specified by two parameters, its mean μ and its variance σ^2:

$$\mathcal{N}(x|\mu, \sigma^2) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) \tag{4.3.5}$$

The multivariate Gaussian density function in N dimensions is given as:

$$\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{(2\pi)^{N/2}|\boldsymbol{\Sigma}|^{1/2}} \exp\left(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^{\intercal}\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})\right) \tag{4.3.6}$$

where $\mathbf{x}$ is an N-dimensional column vector, $\boldsymbol{\mu}$ is an N-dimensional mean vector, $\boldsymbol{\Sigma}$ is a N-by-N covariance matrix, and $|\boldsymbol{\Sigma}|$ and $\boldsymbol{\Sigma}^{-1}$ are its determinant and inverse respectively. Figure 4.2 shows the effect of modifing the μ and σ parameters for two, one-dimensional Gaussian distributions along with illustrating the distribution of a Gaussian in two-dimensional space.



(a) Two one-dimensional Gaussian distributions (b) A two-dimensional Gaussian distribution

FIGURE 4.2: (a) Two one-dimensional Gaussian distribution with $\mu_1 = -0.5$, $\mu_2 = 0.4$ and $\sigma_1 = 0.4$, $\sigma_2 = 0.5$. (b) A two-dimensional Gaussian distribution with $\mu = (0.0, 0.0)$ and $\Sigma = (1.0, 0.0; 0.0, 1.0)$.

Using the multivariate Gaussian, $P(\mathbf{x}|g_k)$ can be replaced by:

$$P(\mathbf{x}|g_k) \sim \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \tag{4.3.7}$$

Instead of having to compute the determinant and inverse for each $\boldsymbol{\Sigma}_k$, the multivariate Gaussian density function can be calculated by taking the product of N independent univariate Gaussian distributions, each with their own mean and variance values:

$$P(\mathbf{x}|g_k) \sim \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \boldsymbol{\sigma}_k^2) = \prod_{n=1}^{N} \frac{1}{\boldsymbol{\sigma}_n\sqrt{2\pi}} \exp\left(-\frac{(\mathbf{x}_n - \boldsymbol{\mu}_n)^2}{2\boldsymbol{\sigma}_n^2}\right) \tag{4.3.8}$$

4.3.3 Adding a Weighting Coefficient For An N-Dimensional Model

For the recognition of musical gestures, it is beneficial to add an additional weighting coefficient (ϕ_{kn}) for the nth dimension of the kth gesture. This weighting coefficient adds an important feature for the Adaptive Naïve Bayes Classifier as it enables one general classifier to be trained with a high number of multi-dimensional signals, even if a number of signals are only relevant for one particular gesture. This would enable one general classifier to recognise, for example, both left and right hand gestures independently, without the position of the left hand affecting the classification of a right handed gesture for instance. By setting the left handed sensor dimensions weighting coefficients to 0 for any right handed gesture and the right handed sensor dimension's weighting coefficients to 1, any left handed movements will be ignored for a right handed gesture. The opposite weighting coefficient values could also be set for any left handed gesture, or for a gesture that required both hands, all the weighting coefficients could be set to 1. This simple addition of a weighting coefficient enables one general classifier to be trained for left handed gestures, right handed gestures and two handed gestures, rather than creating and training three individual classifiers. This weighting coefficient can either be set manually by the user or could even be set by computing the overall significance of each dimension for each particular gesture. A Gaussian model (Φ) for the kth gesture can therefore be represented by:

$$\boldsymbol{\Phi}_k = \{\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k, \phi_k\} \tag{4.3.9}$$

Equation (4.3.8) can therefore be updated with a weighting coefficient to give:

$$P(\mathbf{x}|g_k) \sim \mathcal{N}(\mathbf{x}|\boldsymbol{\Phi}_k) = \prod_{n=1}^{N} \begin{cases} \text{if } \boldsymbol{\phi}_n > 0, & \frac{1}{\boldsymbol{\sigma}_n\sqrt{2\pi}} \exp\left(-\frac{(\mathbf{x}_n - \boldsymbol{\mu}_n)^2}{2\boldsymbol{\sigma}_n^2}\right) \boldsymbol{\phi}_n \\ \text{otherwise}, & 1 \end{cases} \tag{4.3.10}$$

To stop a weighting coefficient value of 0 setting the product over all dimensions to 0, regardless of the other values or weights, the current product will only be multiplied by the nth dimensional Gaussian value if the nth dimensional weight coefficient is greater than 0. If the nth dimensional weighting coefficient is equal to 0 then that dimension should be ignored and therefore 1.0 is used instead. Figure 4.3 illustrates two, two-dimensional weighted Gaussians.

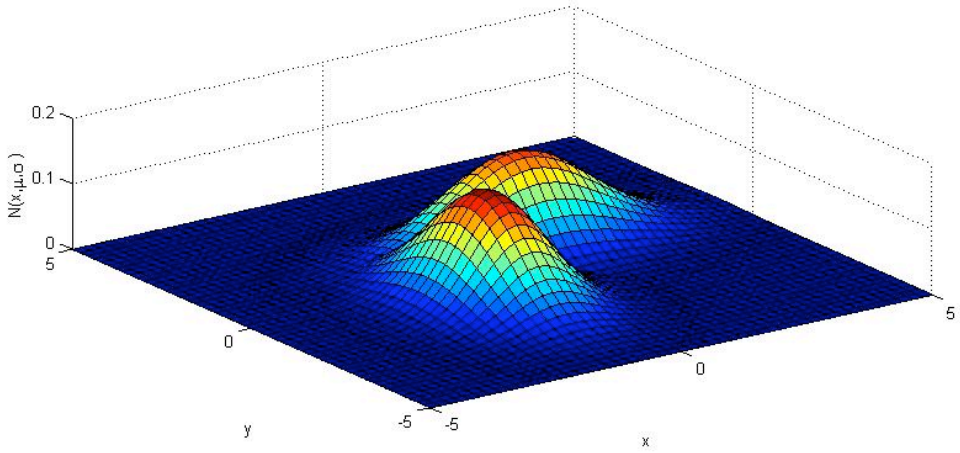


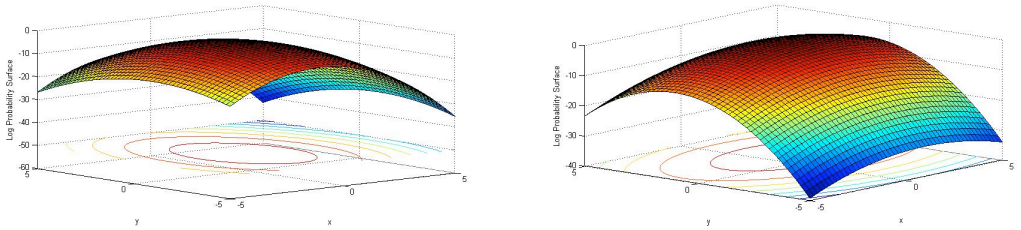
FIGURE 4.3: Two two-dimensional weighted Gaussian distributions with: $\mu_1[-1-1]$, $\mu_2[1\ 1]$, $\sigma_1[1.2\ 0.8]$ $\sigma_2[0.8\ 1.4]$ and $\phi_1[1\ 1]$, $\phi_2[1\ 1]$.

4.3.4 Real-World Computational Concerns

As the product of a large number of small probabilities can easily underflow the numerically precision of a computer, it is more practical to take the sum of the log of each weighted Gaussian rather than the product:

$$\ln \mathcal{N}(\mathbf{x}|\mathbf{\Phi}_k) = \sum_{n=1}^{N} \ln \begin{cases} \text{if } \boldsymbol{\phi}_n > 0, & \frac{1}{\boldsymbol{\sigma}_n\sqrt{2\pi}} \exp\left(-\frac{(\mathbf{x}_n-\boldsymbol{\mu}_n)^2}{2\boldsymbol{\sigma}_n^2}\right)\boldsymbol{\phi}_n \\ \text{otherwise}, & 1 \end{cases} \tag{4.3.11}$$

Computing the log of the function not only stops numerically underflow, it also simplifies the subsequent mathematical analysis. Because the logarithm is a monotonically increasing function of its argument, maximization of the log function is equivalent to maximization of the function itself (Bishop, 2006). Like the case in equation (4.3.10), the log of the weighted Gaussian is only taken if the nth dimensional weighting coefficient is greater than 0, otherwise the log of 1 is used instead which gives 0 and therefore achieves the desired result. Figure 4.4 illustrates the log probability surface for the two weighted Gaussians that were shown in Figure 4.3.



(a) The log probability surface for the first weighted Gaussian

(b) The log probability surface for the second weighted Gaussian

FIGURE 4.4: The log probability surfaces for each of the 2-dimensional weighted Gaussians.

4.3.5 Training The Gaussian Model

Using a Gaussian function, the Adaptive Naïve Bayes Classifier requires $G(3N)$ parameters, assuming that the each of the G-gestures require specific values for the N-dimensional $\boldsymbol{\mu}_k$, $\boldsymbol{\sigma}_k$ and ϕ_k vectors. Assuming that ϕ_k is set manually by the user, the $\boldsymbol{\mu}_k$, $\boldsymbol{\sigma}_k$ values can easily be calculated in a supervised learning scenario by grouping the input training data $\mathbf{X}$, a matrix containing M training examples each with N dimensions, into their corresponding classes. The values for μ and σ of each dimension (n) for each class (k) can then be estimated by computing the mean and variance of the grouped training data for each of the respective classes.

$$\mu_{kn} = \frac{1}{M_k} \sum_{i=1}^{M} \mathbf{1}\left\{\mathbf{X}_{in}\right\} \qquad 1 \leq k \leq G, 1 \leq n \leq N \tag{4.3.12}$$

$$\sigma_{kn} = \sqrt{\frac{1}{M_k - 1} \sum_{i=1}^{M} \mathbf{1}\left\{(\mathbf{X}_{in} - \mu_{kn})^2\right\}} \qquad 1 \leq k \leq G, 1 \leq n \leq N \tag{4.3.13}$$

where M_k is the number of training examples in the kth class and $\mathbf{1}\{\cdot\}$ is the indicator bracket that gives 1 when the training label of example i equals g and 0 otherwise.

4.3.6 Preventing Over-Fitting

Although the Gaussian distribution is a suitable function to use when the number of training examples is small compared with more complex distributions with a high number of parameters, it is still prone to the problem of bias. In particular, it can be shown that the maximum likelihood solution given by taking the sample mean and sample variance will commonly underestimate the true variance of a distribution (Bishop, 2006). This is a key example of over-fitting when a limited number of training examples are presented to the learning algorithm. The bias of the maximum likelihood solution will, however,

become significantly less as the number of M_k training points increases, and in the limit $M_k \to \infty$ the maximum likelihood solution for the variance equals the true variance of the distribution that generated it. A performer should therefore ensure that they do not attempt to train the ANBC algorithm with a very limited number of training examples as this would cause the algorithm to severely over fit its model. This should not be a major issue for a performer however as, because the algorithm has been designed for the recognition of static postures, the musician only needs to record a few seconds of them performing each posture to collect the training examples for each gesture. If the sensor they are using was sampled at 100Hz for example, five seconds of training would still create five hundred training examples per gesture and thus this should help lower the possibility of the model from severe over-fitting.

4.3.7 Classification Using The Gaussian Model

After the Gaussian models have been trained for each of the G classes, an unknown N-dimensional vector $\mathbf{x}$ can be classified as one of the G classes using the *maximum a posterior probability* estimate (**MAP**). The MAP estimate classifies $\mathbf{x}$ as the kth class that results in the maximum a posterior probability given by:

$$\arg\max_k \quad P(g_k|\mathbf{x}) = \frac{P(\mathbf{x}|g_k)P(g_k)}{\sum_{i=1}^{G} P(\mathbf{x}|g_i)P(g_i)} \qquad 1 \leq k \leq G \tag{4.3.14}$$

As the denominator in equation (4.3.14) is common across all gestures it can therefore be ignored without effecting the results. If $P(g_k)$ is a constant scalar that is equal across all of the G gestures then it can also be ignored, leaving the maximum likelihood which, when using the logarithm of the weighted Gaussian model, is equivalent to:

$$\arg\max_k \quad \ln \mathcal{N}(\mathbf{x}|\Phi_k) \qquad 1 \leq k \leq G \tag{4.3.15}$$

Using equation (4.3.15), an unknown N-dimensional vector $\mathbf{x}$ can be classified as one of the G classes from a trained ANBC model. If $\mathbf{x}$ actually comes from an unknown distribution that has not been modeled by one of the trained classes (i.e. if it is not any of the gestures in the model) then, unfortunately, it will be incorrectly classified against the kth gesture that gives the maximum log-likelihood value. A rejection threshold, τ_k, must therefore be calculated for each of the G gestures to enable the algorithm to classify any of the G gestures from a continuous stream of data that also contains non-gestural data.

4.3.8 Computing a Suitable Confidence Measure For Real-Time Recognition

For the rejection threshold, we desire a value that indicates how confident the classifier is in predicting that $\mathbf{x}$ actually came from the kth distribution. In some applications it would be possible to use the normalised value resulting from Bayes' theorem and classify $\mathbf{x}$ as class k if its prediction value was above some pre-defined value, such as 0.5. Unfortunately though, this approach will not work for the classification of a semiotic gesture in a continuous stream of data which may also contain segments of non-gestural data. Bayes' theorem cannot be used in this instance because, as $P(B|A)P(A)$ is normalised by $P(B)$, a poor prediction value when normalised may unfortunately yield a very confident prediction value, resulting in a false-positive classification error if $\mathbf{x}$ is not a gesture.

This error can easily be mitigated however by using the log-likelihood value of the kth predicted gesture as a measure of how confident the algorithm is that $\mathbf{x}$ is in fact gesture k. Using the log of the weighted Gaussian function as a confidence measure, a suitable rejection threshold can therefore be computed during the algorithms training phase to enable the rejection of non-gestural data in the real-time classification phase. The rejection threshold, τ_k, can be computed for each of the G gestures by taking the average confidence level of all the training data for class k minus γ standard deviations:

$$\tau_k = \mu_k^* - \left(\sigma_k^* \gamma\right) \tag{4.3.16}$$

where μ_k^* and σ_k^* are the average confidence values and standard deviation of the confidence levels respectively for the kth gesture given by:

$$\mu_k^* = \frac{1}{M_k} \sum_{i=1}^{M} \mathbf{1} \left\{ \ln \mathcal{N}(\mathbf{X}_i | \Phi_k) \right\} \tag{4.3.17}$$

$$\sigma_k^* = \sqrt{\frac{1}{M_k - 1} \sum_{i=1}^{M} \mathbf{1} \left\{ \left(\ln \mathcal{N}(\mathbf{X}_i | \Phi_k) - \mu_k^* \right)^2 \right\}} \tag{4.3.18}$$

Here γ is a constant scalar value that can be adjusted by the user until a suitable level of classification has been achieved. The γ parameter enables the performer to further mitigate the effects of over-fitting, as by setting γ to a value greater than 1.0 will lower the threshold value and enable 'noisier' data than that in the training data set to be classified as gesture k. Using the rejection threshold, a gesture will only be classified as k if its log-likelihood estimation is greater than or equal to that classes' threshold value. Otherwise, $\mathbf{x}$ will be classified as a null gesture, usually with an I.D. value of 0:

$$\hat{k} = \begin{cases} k & \text{if}(\ \ln \mathcal{N}(\mathbf{x}|\Phi_k) \geq \tau_k) \\ 0 & \text{otherwise} \end{cases} \tag{4.3.19}$$

4.3.9 Computing a Rejection Threshold

Using the log of the weighted Gaussian function as a confidence measure, a suitable threshold value, τ_k, can be computed by taking the average confidence level of all the training data for class k minus γ standard deviations:

$$\tau_k = \mu_k^* - \left(\sigma_k^* \gamma\right) \tag{4.3.20}$$

where μ_k^* and σ_k^* are the average confidence values and standard deviation of the confidence levels respectively for the kth gesture given by:

$$\mu_k^* = \frac{1}{M_k} \sum_{i=1}^{M} \mathbf{1} \left\{ \ln \mathcal{N}(\mathbf{X}_i|\mathbf{\Phi}_k) \right\} \tag{4.3.21}$$

$$\sigma_k^* = \sqrt{\frac{1}{M_k - 1} \sum_{i=1}^{M} \mathbf{1} \left\{ \left(\ln \mathcal{N}(\mathbf{X}_i|\mathbf{\Phi}_k) - \mu_k^* \right)^2 \right\}} \tag{4.3.22}$$

Here γ is a constant scalar value that can be adjusted by the user until a suitable level of classification has been achieved. The γ parameter enables the performer to further mitigate the effects of over-fitting, as by setting γ to a value greater than 1.0 will lower the threshold value and enable 'noisier' data than that in the training data set to be classified as gesture k.

Using the rejection threshold, a gesture will only be classified as k if its likelihood estimation is greater than or equal to that classes' threshold value. Otherwise, $\mathbf{x}$ will be classified as a null gesture, usually with an I.D. value of 0.

$$\hat{k} = \begin{cases} k & \text{if}(\ \ln \mathcal{N}(\mathbf{x}|\mathbf{\Phi}_k) \geq \tau_k) \\ 0 & \text{otherwise} \end{cases} \tag{4.3.23}$$

4.3.10 Adaptive Online Training

One key element of the Naïve Bayes Classifier, is that it can easily be made adaptive. Adding an adaptive online training phase to the common two-phase (training and prediction) ethos provides some significant advantages for the recognition of static musical

gestures. During the adaptive online training phase the algorithm will not only perform real-time predictions on the continuous stream of input data; it will also continue to train and refine the models for each gesture. This enables the performer to initially train the algorithm with a low number of training examples after which, during the adaptive online training phase, the algorithm can continue to train and refine the initial models, creating a more robust model as the number of training examples is increased. The adaptive online training phase also importantly facilitates the algorithm to adapt its initial model as the performer themselves adapts and refines their own gestures as may happen over the course of a rehearsal period for example.

For the Adaptive Naïve Bayes Classifer, the adaptive online training works as follows:

After the musician has initially trained the algorithm, they can use it in real-time to classify their musical gestures. During this real-time prediction, the musician can choose to turn on the adaptive online training mode. In this mode the algorithm will slowly refine $\boldsymbol{\mu}_k, \boldsymbol{\sigma}_k$ and τ_k for each of the G gestures, overwriting the previous models that have been computed earlier. For the adaptive online training phase, the user must first decide on three parameters, the maximum training buffer size, the update rate and γ the scalar on the number of standard deviations (see equation (4.3.20)). These parameters control the maximum number of training examples to save for each class in the model, how fast the algorithm retrains the model and the number of standard deviations to use when calculating the classification threshold in the model respectively.

If $\mathbf{x}$ is correctly classified as g_k and is greater than or equal to τ_k then:

- Add $\mathbf{x}$ to the training buffer, removing the oldest training example if the buffer is full and increment the update counter by 1.
- If the update counter is equal to the update rate then recompute $\boldsymbol{\mu}_k, \boldsymbol{\sigma}_k$ and τ_k using the data in the training buffer. These are calculated using equations (4.3.12), (4.3.13), (4.3.21) and (4.3.22). Reset the update counter to 0.

Using a limited size first-in, first-out (FIFO) buffer, set by the maximum training buffer size parameter, ensures that only the most recent training examples are used to refine the model, allowing the $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ vectors to slowly change as the user refines their own movements. Setting a fixed buffer size also ensures that an unfeasible amount of memory is not consumed by thousands of training examples over the course of a long rehearsal session. An individual FIFO buffer must be used for each of the G gestures to ensure that a large amount of new training data for one class does not 'pop-out' the original training data in any of the other classes. The speed at which the algorithm adapts can be controlled by the update rate parameter, allowing the performer to control how

sensitive the adaptive component of the algorithm will be to their latest gestures. The overall sensitivity of the system, both for the adaptive online training phase and for the standard real-time prediction can be controlled by the performer using the γ parameter.

4.3.11 Strengths and weaknesses of the ANBC algorithm

The greatest strength of the Adaptive Naïve Bayes Classifier is also, perhaps, its greatest weakness. This is the algorithm's ability to automatically adapt its model by adding the latest classified input vector to the data that will then be used to recompute model. In the best case this self-labelled data will help to create a more robust model, however, in the worst case a small number of incorrectly labelled training examples could create a 'run-away' model that becomes less effective at each update step. To mitigate this problem a parameter was added to the EyesWeb implementation of the algorithm that enables the user to reload the original ANBC model if the real-time classification abilities of an updated model starts to perform poorly. The user can also ensure that they have set the buffer size, update rate and γ parameters to the most appropriate values.

The exact value of these parameters will vary depending on the sensors and features being input to the algorithm and on the types of gestures the user wishes to classify, however, the following general rules can be applied. The buffer size should be set to a value that maximizes the amount of correctly labelled training data for each class, while minimizing the number of 'old' training examples that are no longer relevant because the user has modifier their original gesture. If the training buffer size is set to a value that is too large then the model may start to under-fit, resulting in an increasing number of false-positive classifications. If the training buffer size is set to a value that is too small then the model may over-fit, resulting in a poor estimation of the model's variance parameters which may cause a gesture to be incorrectly classified as a null-gesture. The most appropriate value for the update rate parameter will depend on the sampling frequency of the data that is input to the algorithm along with the speed at how fast the performer thinks their gestures could change. The user should choose an update rate value that enables the algorithm to slowly recompute the ANBC model so that the parameter values are not rapidly changing every second. The update rate parameter should however be fast enough so that no training data is being wasted, i.e. old training data is being removed from the FIFO buffer before the data was every used to recompute a new ANBC model. Finally, the user should set the γ parameter to an appropriate value that achieves the user's desired level of classification robustness. Setting the γ parameter to a low value will reduce the possibility of the algorithm making a false-postive classification error, however, a low γ value may also result in the algorithm not being able to classify an actual gesture if the data is slightly 'noisier' than that of the

training data. A high γ parameter will enable 'noisy' gestures to be correctly classified, however, setting this parameter too high may result in false-positive classification errors.

Through the real-time application of using the ANBC algorithm to classify static semiotic gestures, it became quickly evident that one of the algorithms key strengths is the its ability to automatically compute τ_k, the classification threshold for the gth gesture. This classification threshold enables the ANBC algorithm to classify the musical gestures from a continuous stream of data that also contains null gestures without having to explicitly train a null-class or tell the algorithm that one of the gestures has just been performed.

One possible weakness of using the Gaussian distribution as the foundations of the ANBC algorithm is that the Gaussian model can only be used for the classification of linearly separable data. The ANBC algorithm could possibly be improved in the future by implementing functions that can be used for the classification of non-linear data by using, for example, the kernel trick (which will be described further in chapter 6.1.2) to map the non-linear problem in the original data space to a linearly separable problem in a higher dimensional feature space (Bishop, 2006).

4.4 Implementation of the ANBC algorithm in EyesWeb

To enable the Adaptive Naïve Bayes Classifier algorithm to be used by any performer, regardless of their programming abilities or prior understanding of machine learning, the algorithm has been fully implemented within EyesWeb. The ANBC algorithm can be found in the SARC gesture recognition catalog, and consists of three separate blocks: the ANBC Training Tool block, the ANBC Training block and the ANBC Prediction block.

4.4.1 The ANBC Training Tool block

The ANBC Training Tool block enables a performer to quickly collect the labelled training data required to train the ANBC algorithm. The block allows the performer to use any N-dimensional feature vector as input to the ANBC algorithm, giving the user the option to manually assign a weighting coefficient value for each of the N dimensions. The user can then perform a number of repetitions of each of the G gestures they wish the algorithm to recognise, setting the block to record the input feature vector as the user performs each gesture. The user can then save all of the labelled training data to a file when they are satisfied that enough training data has been recorded.

The ANBC Training Tool block features two inputs, both of which are [1 by N] double matrices, the first for the N-dimensional feature vector and the second for the N-dimensional weights vector. To provide the user with feedback on the current status of the block, it features two outputs. The first output is a boolean datatype that provides the current recording status of the block, i.e. it will output true if the block is recording the current input data. The second output of the block is an integer value representing the current number of training examples that have been recorded in the blocks internal buffer. The block also features five parameter pins for setting the class label value for the corresponding current input, e.g. labeling a few seconds of recorded data as belonging to class 1, along with buttons to set the blocks recording status to true or false, save the current training data to a file or to clear all of the currently collected training data. The final parameter pin enables the user to set the name and location of the file that the training data will be saved to.

Figure 4.5 shows an example patch that has been created to demonstrate to a novice user how to operate the ANBC Training Tool block. In this example the input to the block consists of the x and y coordinates of a mouse, with both values having equal weighting coefficients of 1. If the user wanted to record some training data using their own specific sensor device then they would simply have to replace the current two-dimensional input vector, containing the x and y values of the mouse, with an N-dimensional feature vector containing the raw values or features from their own sensor device(s). After the user has updated the weights vector to also have N input values they can then run the patch and start to record a number of training examples for each of the G classes that they wish the ANBC algorithm to recognise.

4.4.2 The ANBC Train block

The ANBC Train block enables a performer to quickly train an ANBC model, using the training data collected using the ANBC Training Tool block. The ANBC Train block loads the labelled training data from the file created by the ANBC Training Tool block and attempts to train an ANBC model. If successful, the ANBC block will output 'true' on its first output pin along with saving the trained ANBC model to a file. If unsuccessful, the block will output 'false' and print an error message informing the user of what the error may have been. Figure 4.6 shows an example patch that has been created to demonstrate to a novice user how to operate the ANBC Train block.

The ANBC Train block features no input pins and two output pins showing the training status of the block and the results of the cross validation testing, if the user enables the cross validation training. The block also features six parameter pins consisting of a

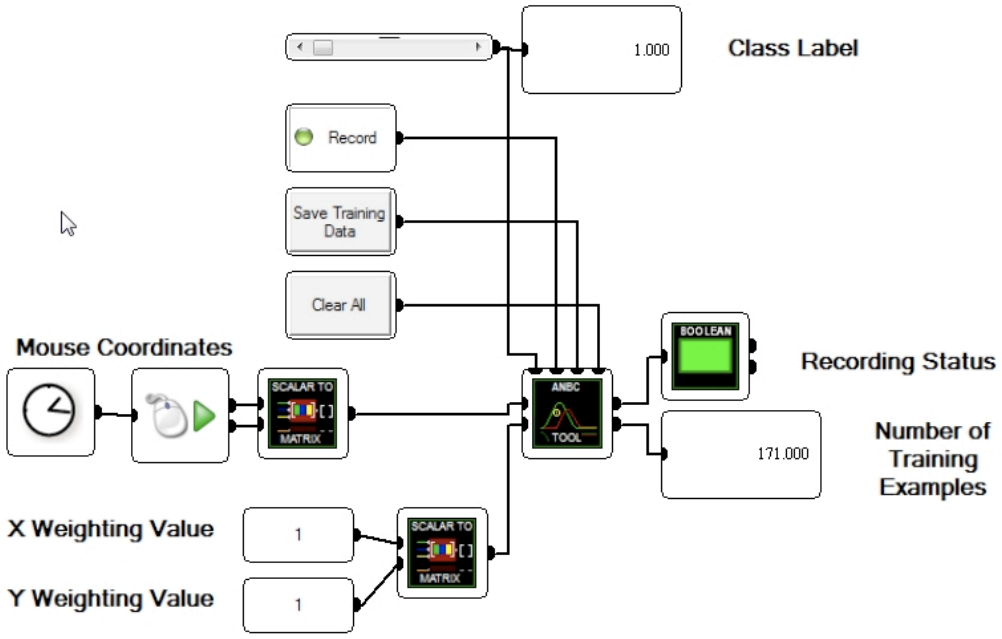


FIGURE 4.5: An example patch demonstrating the use of the ANBC Training Tool block. The input to the block consists of the x and y coordinates of the mouse, with both inputs having the equal weighting values of 1.

button that loads the training data from a file and starts the ANBC training algorithm and a pin enabling the user to set the γ parameter. One boolean parameter pin controls if cross validation is used to test the model and an integer parameter pin enables the user to set the number of folds to be used. The remaining two pins allow the user to specify the name and location of the file containing the training data and the file to which the ANBC model will be saved after the model has been trained. Figure 4.6 shows an example patch that has been created to demonstrate to a novice user how to operate the ANBC Train block.

4.4.3 The ANBC Predict block

The ANBC Predict block enables a performer to use the ANBC algorithm to classify the performer's gestures from a continuous stream of data that may also contain null gestures. The input to the block should consist of the same N-dimensional feature vector setup that was used for input to the ANBC Training Tool block. The ANBC block will classify each new N-dimensional input, outputting the predicted classes' ID if the input is classified as one of the G classes in the ANBC's trained model, otherwise outputting 0. The block also features five other output pins, containing the maximum log-likelihood

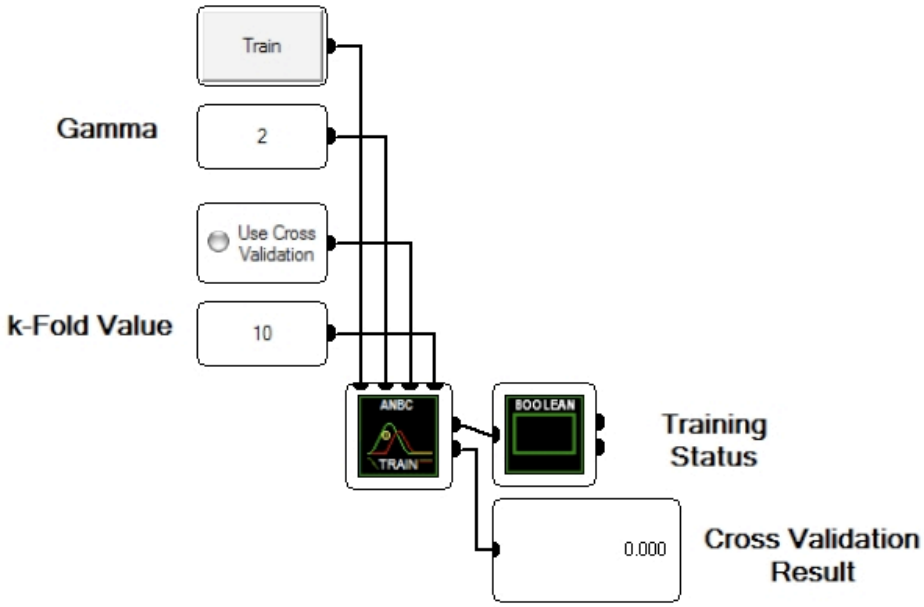


FIGURE 4.6: An example patch demonstrating the use of the ANBC Train block.

value of the current prediction, a G-dimensional vector containing the log-likelihood value for all of the G classes in the trained model and three double matrices containing the current model's parameter values for μ, σ and τ.

In addition to the input and output pins, the ANBC Predict block also features eight parameter pins. These pins consist of two file name parameters enabling the user to specify the name and location of the model file created by the ANBC Train block and the original training data used to train the model created by the ANBC Training Tool block. The remaining parameter pins are all used in the adaptive online training phase of the algorithm. These pins consist of three buttons to turn on/off the adaptive training, a button to reload the original training data and a button to overwrite the original ANBC model file with the latest updated model. The final three parameter pins control the algorithm's maximum buffer size, the update rate and the γ parameter. Figure 4.6 shows an example patch that has been created to demonstrate to a novice user how to operate the ANBC Predict block.

4.4.4 Summary of the ANBC block design

Encapsulating the ANBC algorithm's training and prediction functions across the three ANBC blocks enables a performer to efficiently record and label some training data,

train an ANBC model and finally use the algorithm in real-time to predict and refine the underlying model using the adaptive training function. Saving the original training data to its own file, separated from the ANBC model parameter file, enables the user to train multiple ANBC models, each with a different γ parameter for example, and quickly evaluate the classification abilities of each model using cross validation or real-time classification without having to collect the training data multiple times. Saving the training data to its own file also enables the ANBC Predict block to load this data and use it to populate the FIFO buffer, thus enabling any new training data to be combined with the original data that was used to first train the model. If the performer feels that the adaptive online training function has created a very robust model they can turn off the adaptive training to fix the model's parameters at their current values and also set the ANBC Predict block to overwrite the initial model and training data contained in the files with the current model and the latest training data contained in the FIFO buffer. Alternatively, if the performer thinks that the algorithm's classification abilities have started to degrade because of a run-away self-labelling problem, the user can simply reload the original model, reverting the ANBC model and the training data in the FIFO buffer back to their original conditions.

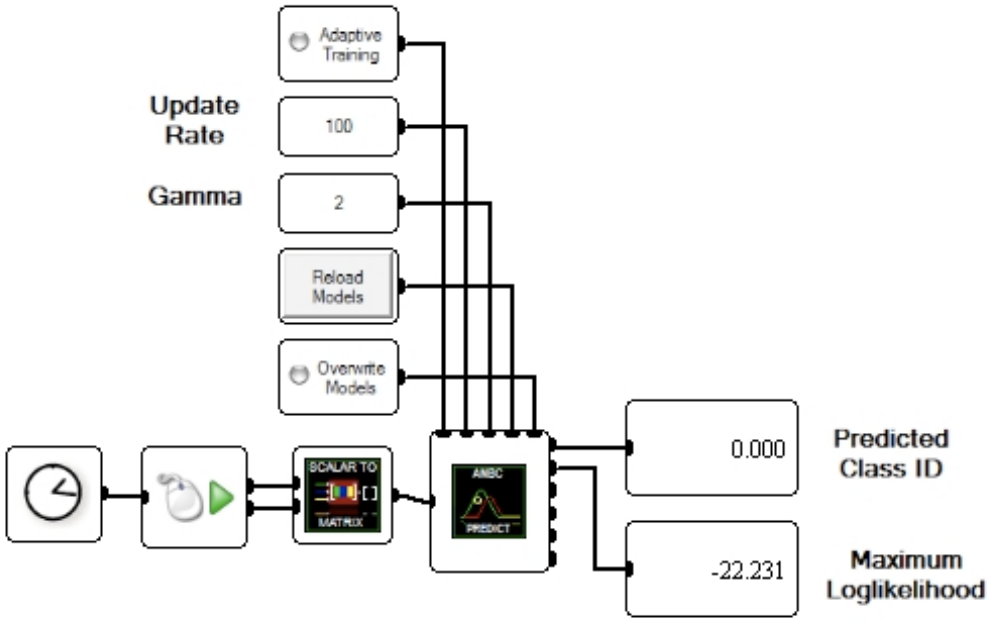


FIGURE 4.7: An example patch demonstrating the use of the ANBC Predict block.

4.5 Evaluating the ANBC Algorithm

The adaptive classification abilities of the Adaptive Naïve Bayes Classifier algorithm were tested using a simple 'free-space' pointing based experimental task. Participants were asked to define a number of target areas within a fixed region of space that they then had to return to when prompted. The ANBC algorithm was then used to classify if the participant's hands were in the correct area of space when prompted; and if the classification results would improve when the adaptive function of the algorithm was used. To constrain this task as much as possible a rudimentary game orientated task was used instead of a musical orientated task. To achieve this we created a virtual boxing game called 'Air Makoto' in which participants were asked to strike a number of virtual targets when prompted. The ANBC algorithm, combined with a punch detection algorithm, were used to recognise if the participant was able to successfully hit the correct virtual target within a limited time scale.

4.5.1 Air Makoto

Air Makoto is a virtual boxing game loosely based on the martial arts training game Makoto[1]. In Makoto, a player stands in the center of an equilateral triangle, with a 6-foot tall metal column situated on each of the three corners of the triangle. Each column features ten clear panels containing lights and pressure sensors, all of which are spaced at intervals and face the player at the center of the triangle. Each column also features a speaker and represents an 'opponent' for the player to battle with. The player uses one piece of equipment, consisting of a four foot fiber-glass pole with lightly padded ends. The object of Makoto is for the player to continually strike the randomly appearing lights on each of the columns as fast as possible using the pole. When the game starts, one of the lights on a random column's target panels light up and the column emits a tone through its speaker. The player must hit the target panel with the pole; a successful strike will cause the light to go out and the column to emit a confirming tone. Shortly after the player has done so, another randomly selected target panel will light up on a randomly selected column. The objective is to hit each light in sequence, as quickly as possible, without missing any, as the computer controlling the lights monitors the player's reaction time. As the game progresses, the interval between each new light and the amount of time it is lit decreases, with the overall objective of the game to make it to the end of the final level without missing a single panel.

Air Makoto uses a similar game design, with the exception that only two columns are used, both of which are imaginary. The player must therefore define where in space

[1] http://www.makoto-usa.com/new/index.html

they want the columns' target panels to be located. For simplicity, three target panels for each column were used and the player was asked to 'punch' the air targets when prompted, rather than hitting them with a pole. Using Air Makoto, the classification abilities of the ANBC algorithm were tested by using the algorithm to recognise whether a player had successfully hit the corresponding target panel when prompted. A Polhemus Liberty 6-degrees of freedom (DOF) magnetic tracker was used to track the participants' movements, using custom built motion capturing software. The Polhemus was sampled at 120Hz using two tracking sensors, one of each mounted on the top of a small glove that each participant was asked to wear on their left and right hands. The Polhemus data was streamed directly into EyesWeb via OSC, after which the position data from both sensors was sent to the ANBC block for training/prediction, along with being sent to a hit detection algorithm to recognise the punch gestures. EyesWeb then sent the ID's of any punch gestures that were recognised via OSC to Processing[2] which contained a game engine, to keep track of the participant's progress during a game, and a visual engine, that provided the participant with a 3D virtual game environment for visual feedback, as illustrated in Figure 4.8. The input vector to the ANBC algorithm consisted of a 6-dimensional vector containing the smoothed x, y and z position values of each sensor mounted on the participant's hands. The position data was smoothed using a simple moving average filter with a buffer size of 5.

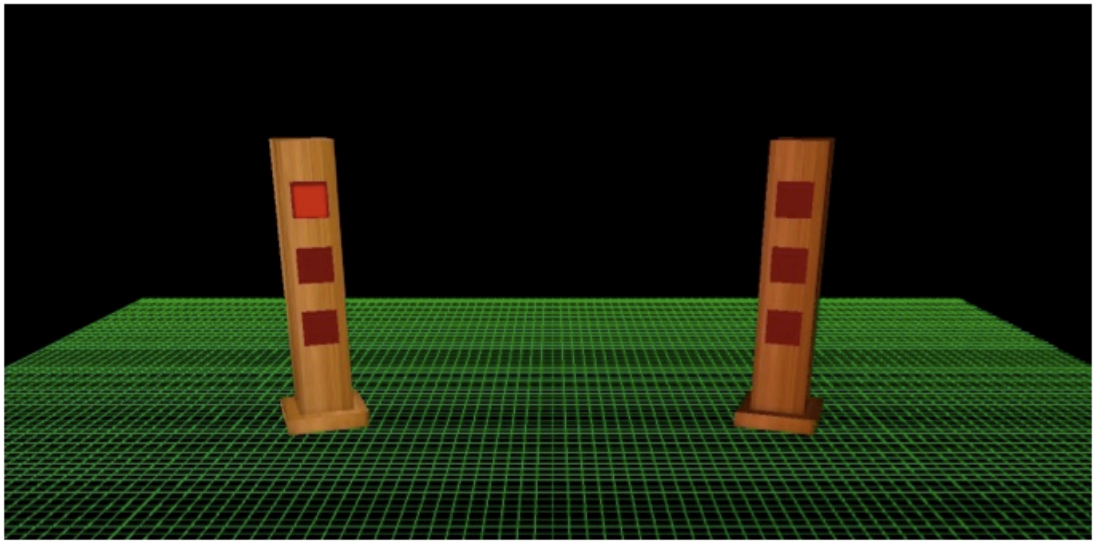


FIGURE 4.8: The 3D virtual game environment used in Air Makoto.

[2]http://processing.org/

4.5.2 Hit Detection

In Air Makoto, a participant was evaluated as being able to correctly hit a target panel if they made a punching gesture at the imaginary location of the correct target panel before the target panels light went out. The ANBC algorithm was used to detect if the player's hand was in the correct target area, however, the game also required a way of detecting if a punch gesture was made. A punch gesture was detected by taking the first derivative of the position data from the X, Y and Z axis of both sensors on the left and right hands. The position data was first low pass filtered using a moving average filter with a buffer size of 5 prior to differentiation. The differentiated signal was then passed through a dead zone block which zeroed any value between the range of -1.0 to 1.0, offsetting any value either above or below this range by -1. The output of the dead zone block was passed through a threshold crossing block that was triggered with an upwards threshold crossing above the value of 0.1. The signals resulting from each of these signal processing steps are illustrated in Figure 4.9. Using these signal processing techniques, a robust punch detection algorithm was created as the thresholding block would only trigger an output if a significant negative-positive change of direction occurred in any of the three axes of either hand. If a threshold crossing was detected then EyesWeb would check to ensure that the ANBC algorithm was predicting that one of the corresponding target areas was active, areas 1,2 or 3 for a left handed punch or areas 4, 5 or 6 for a right handed punch, if the correct punch occurred in the correct area then EyesWeb would send a message to the Air Makoto game engine running in Processing to inform it of the punch.

4.5.3 Location And Setup

This experiment took place in the *Interaction Room* at the Sonic Arts Research Centre, Queen's University Belfast. Each participant was asked to stand on a marked location in the room and face a large projection screen situated three meters in front of them and two meters to their right. A pair of speakers was placed on either side of the screen to provide audio feedback. The projection screen displayed the Air Makoto virtual game scene, which consisted of two wooden columns placed on the left and right of the main view (as illustrated in Figure 4.8). Each column featured three dark red panels, which would change to bright red when the participant needed to hit them.

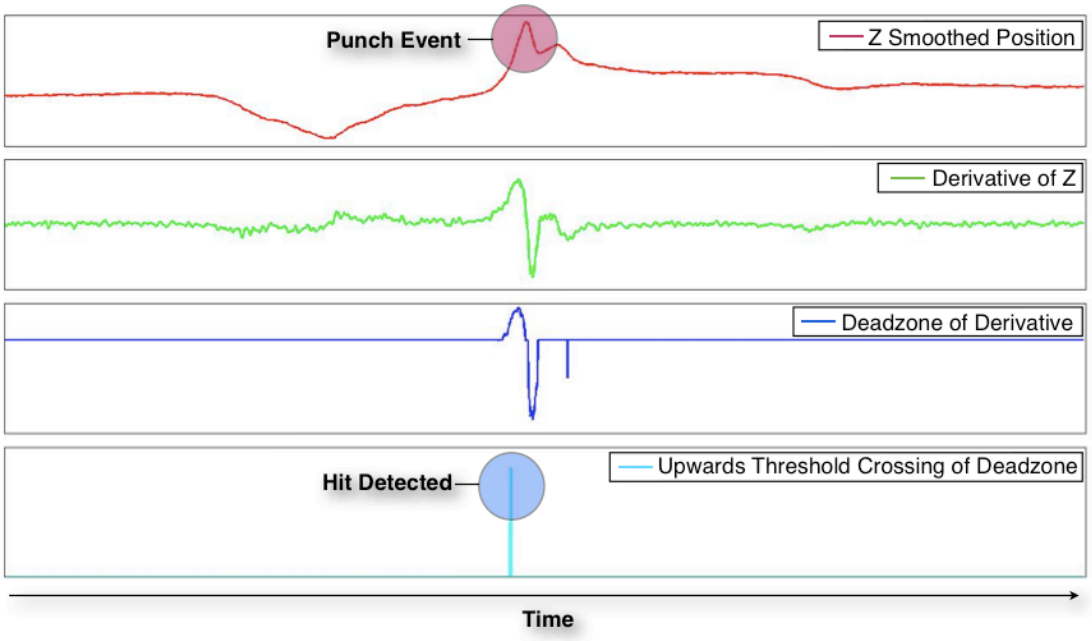


FIGURE 4.9: An example of the four main signal processing steps of the hit detection algorithm used to detect punches in the Air Makoto game. Moving from top down the four images shows: z position smoothed data, first derivative of z, dead zone of the derivative signal and finally the upwards threshold detection on the dead zone signal.

4.5.4 Participants

Twelve participants were recruited from the SARC research community via email. The sample group consisted of nine males and three females with an average age of 29 (standard deviation of 2.96). Six of the participants were right handed and none of the participants had any conditions that would have affected them in performing any of the movements required in this experiment.

4.5.5 Method

A within-subject experimental design was used, in which each participant was asked to play the Air Makoto game in two conditions. Condition A used the ANBC algorithm without the adaptive online training mode and condition B used the ANBC algorithm with the adaptive online training mode. Prior to playing the game in either condition, each participant was given specific instructions about how to play the game and what they needed to do to train the system to recognise the location of their target panels. None of the participants were told that the Adaptive Naïve Bayes Classifier was being used to recognise their gestures. The experiment was divided into three phases, with an initial data collection phase followed by a practice phase and a game phase. The

initial data collection phase was only run once per participant with the data captured in this phase being used to train the ANBC model. The practice and game phases were repeated for each of the two conditions. The order in which each participant completed the two conditions was randomised to account for any learning effects that might occur over the previous practice and game phases.

4.5.5.1 Data Colletection Phase

To gather the initial training data required to train the ANBC algorithm for both conditions, the participant was asked to move their hand around the location of where they wanted to place each of the three target panels for each column. The participants were asked to only use their left hand to train and hit the three target panels on the left-most column and to only use their right hand to train and hit the three target panels on the right-most column. For the actual training stage, each target panel on the screen would light up yellow indicating for the participant to move their respective hand to the location that they wanted that target panel to be placed. The target panel would then light up red, indicating that the training data was being recorded, at which point the participant was instructed to move their hand around the location of the target panel covering a sphere with a diameter of approximately 12-inches. After 5 seconds the training data for that target panel would stop being recorded and the next panel would light up yellow indicating that the training data for that target panel was about to be recorded. This was repeated until the training data for all of the target panels was recorded.

4.5.5.2 Practice Phase

The participant then entered a practice phase which lasted for one minute. In the practice phase all audio and visual feedback was turned on. For condition A, the original training data was simply reloaded and the adaptive training mode was turned off. For condition B however, the adaptive training mode was turned on during the practice phase. At no stage in the experiment could the participants see a representation of the position of their hands in the virtual world as this would have made the game too easy. However, during the practice phase an additional piece of visual feedback was provided in the form of a white square that would light up around any target panel if the participant had their hand in the location of that target panel. This visual feedback gave the participants valuable information in terms of whether they had their hands in the correct location or not. The participants also received audio feedback in the form of a punching noise if they were able to successfully 'hit' an illuminated target panel

in the time allotted. This audio feedback informed the participants as to whether they were punching in the correct location and also making the correct punching gesture to trigger a punch. All of the 10 participants were able to perform the correct punching gestures, if they could remember where they had placed the target locations.

4.5.5.3 Game Phase

After the participants had completed their one minute practice phase, they were then asked to play the main game during which their successful hit scores would be recorded. The main game lasted a total of two minutes, during which time the participants had to hit 50 randomly selected virtual target panels. To ensure the game was not too easy for the participants, each panel was only illuminated for 1.5 seconds, with the result that a participant had to react very quickly to hit the correct target panel. During the main game the participants only received the visual feedback informing them of which target panel they needed to hit. The 'correct target area' visual feedback and 'correct punch noise' audio feedback were both turned off, resulting that each participant was unsure whether they were hitting the correct target panel in time or whether they were even punching the correct area of space at all. At the end of the two minute game the correct hit accuracy score was displayed on the screen, informing the participant how well they had performed overall during that main game. Each participant was then given a small amount of time to rest before starting the practice phase again, only this time with a different condition being used. After the second practice phase the participant then played the main game one final time after which their scores were recorded.

4.5.5.4 ANBC Settings

For this experiment, the maximum training buffer size parameter was set to 600 to ensure that the number of training examples in the initial training data set would be equal to the number of training examples used to retrain the ANBC algorithm during the practice phase in condition B. The update rate was also set to 240, resulting in the ANBC model being recomputed every two seconds during the practice phase in condition B. The γ parameter was set to 5 for all conditions.

4.5.6 Results

Table 4.1 contains the results for all 12 participants across both conditions. All of the participants, with the exception of participant 8, achieved a higher score in condition B which used the adaptive function compared with condition A which just used the

training data collected in the initial data collection phase. A paired *t-test* analysis on these results showed that there was a significant overall improvement between the participants' scores in condition A when compared to that of the participants' scores in condition B ($p = 0.0028$).

4.5.7 Discussion

The participants' scores in table 4.1 show that all the participants, with the as mentioned exception of participant 8, achieved a higher score in condition B when the adaptive online training function was used as compared with condition A in which only the initial training data was used; but why? One observation noted during the course of the study may explain these results, in that the majority of participants found it difficult to remember exactly where they had placed some or all of their target zones, even 30 seconds after they had just specified their locations. Because of this inability to locate the target zones, many of the participants had to spend the first 30 seconds of the practice phase just locating one or several of the target zones. In condition B, a difficult target zone slowly adapted itself until the participant found it easy to locate, with many of the participants remarking "*ah, now I remember where it is*", unaware that the algorithm was adapting the location and size of the target zone as the participant was exploring its possible location in space. The outcome of this adaptive training resulted in the fact that, for the majority of participants, they were consistently successful at hitting the flashing target panels by the time the practice mode ended. In condition A however, many of the participants were still unsure of exactly where they needed to punch for one or more of the target panels by the time the practice mode ended. This observation highlights two important points for the application of such 'free space' gestures for both MCI and the wider HCI community. The first is a performer's ability to remember the precise location of a point in space and the second is the importance of some form of visual or audio feedback to inform the user how far they are from any target location. For this experiment a world-centered frame of reference (**FoR**) (O'Modhrain (2004)) was used, in which the participant's movements were tracked relative to the 3D space in which they were moving. The participants may have found it easier to locate the target areas if a body-centered FoR, in which the target areas were always relative to the user's body, was used instead. A body-centered FoR may have helped the user, as a target area placed at eye-level and arms reach at the user's right, for example, would always be at this body-centered-location irrespective of where in the room the participant moved. A body-centered FoR could have been achieved using a third tracking sensor placed on the participant's chest, for example, from which the position coordinates of all the other sensors could be translated.

Participant #	1	2	3	4	5	6	7	8	9	10	11	12
Condition A	25	24	28	13	13	16	34	38	18	17	40	21
Condition B	29	42	31	24	17	31	42	36	19	19	45	36

TABLE 4.1: The scores from the game phase of Air Makoto for all 12 participants for conditions A and B. The adaptive training was only used in condition A. Each participant gained one 'point' for each target correctly hit within the alloted time, with the maximum score possible in either condition of 50 points.

Obviously, the participants would have achieved a higher score if they were allowed to move their hands around a much larger area of space in the initial data collection phase as this would have created a much larger 'target zone', enabling the participant to be less accurate. To mitigate this, the participants were deliberately instructed to only move their hands around a spherically volume with an approximate diameter of 12 inches. This constraint, combined with the speed at which the random panels appeared in the main game phase ensured that the game was difficult enough to prove a challenge to the participants. This is confirmed by the results over all participants and across both conditions as none of the participants were able to successfully hit all 50 of the targets.

Participant 8, the only participant to achieve a better score in condition A over condition B, achieved an above average score ($\mu A = 23.92$, $\mu B = 30.92$) of 38 and 36 for conditions A and B respectively. A possible reason of this participant achieving a better score in condiiton A over condition B is that his 'target zones' were already optimally trained from the initial training data and the difference in score simply resulted from a better performance in condition A over condition B.

4.5.8 Conclusion

The results from this study suggest that the classification abilities of the ANBC algorithm are significantly improved when the adaptive online training function is used. It is believed that the adaptive online training function improved the participants' scores in the Air Makoto game because the majority of participants failed to remember the exact location that they had defined their 'target panels'. The adaptive online training function therefore enabled, in combination with the visual and audio feedback in the practice phase, the location and size of each target zone to adapt to the participants practice gestures. This resulted in the majority of participants being able to successfully 'hit' each target panel by the end of the adaptive training phase, were as in the none-adaptive training phase the same participants struggled to achieve consistent 'hits'. Although this study shows the benefits of an adaptive model in a simple pointing style task, it is believed that the ANBC algorithm will prove as useful in the classification

of similar musical semiotic gestures, particularly those static gestures that consist of a continuous-valued, randomly corrupted version of a single prototype vector which may itself slowly change over time.

4.6 Summary

This chapter has presented the Adaptive Naïve Bayes Classifier, an algorithm that has been specifically designed for the recognition of static semiotic musical gestures. The standard Naïve Bayes Classifier was extended with the ability to automatically compute a classification threshold for each of the G classes in a trained model, enabling the algorithm to classify null gestures without having to first compute a null-model. A novel adaptive element to the algorithm was also added, enabling a trained ANBC model to automatically adapt itself to accommodate a performer as they adapt their own gestures over, for example, the course of a rehearsal period. The chapter was concluded with a study to demonstrate the classification abilities of the ANBC algorithm, with the results showing that a significant overall improvement was achieved between the participants' scores in the condition that used the adaptive element of the algorithm. The next chapter now looks at how non-static musical gestures, i.e. gestures that consist of a cohesive sequence of movements that occur over a variable time period, can be recognised and therefore be used for musician-computer interaction.

Chapter 5

Recognition of Multivariate Temporal Musical Gestures

Failures are finger posts on the road to achievement.

C. S. Lewis

The previous chapter presented the Adaptive Naïve Bayes Classifier, a novel algorithm that has been specifically designed for the recognition of static symbolic musical gestures. This chapter addresses the recognition of multivariate temporal musical gestures. The challenges of recognising temporal gestures, as opposed to static postures, are first outlined. This is followed by a description of a data set containing multivariate temporal gestures that has been specifically collected to test the algorithms in the remainder of this thesis. A powerful machine learning algorithm called an Hidden Markov Model (HMM) is presented along with a description of how the standard HMM has been modified for the recognition of multivariate temporal musical gestures. The chapter is concluded with a number of experiments designed to evaluate the classification abilities of the modified HMM algorithm.

5.1 Multivariate Temporal Gestures

Temporal gestures consist of a cohesive sequence of movements that occur over a variable time period and generally pose a more challenging recognition problem than classifying static postures. Due to the nature of human movement, any repeated movement will feature temporal variability, even in an optimal task condition. This variability can be both intrapersonal (the variability of one individual attempting to repeat the

same movement) and interpersonal variability (the variability of multiple individuals attempting to repeat the same movement). Such variability makes the classification of any gesture that involves a temporal element (as opposed to a static posture) difficult as the gesture may never be performed at exactly the same speed, amplitude or location. If the classification problem features high dimensional data, the machine learning algorithm must not only model the relationship between the multidimensional data at time t, but also model how this relationship changes over time. This type of pattern recognition problem can be viewed as a *multivariate temporal classification problem* (Bishop, 2006).

5.1.1 An Overview Of The Classification Problem

One issue that commonly occurs in multivariate temporal classification, is how two vectors of different lengths can be compared. This is because the more common distance measures between two vectors such as the Euclidean distance, root mean squared distance or the cosine of the angle between the two vectors will not work for vectors of different lengths. This renders common distance measures useless for the recognition of temporal musical gestures. If there is any temporal variability in the movement at all then the distance measure will give a poor classification result.

A machine learning model must therefore take the temporal variation of a signal into account, as is the case with algorithms such as a Hidden Markov Model or the Condensation Algorithm. Alternatively, the temporal variation can be removed by the feature extraction stage, after which a feature vector containing the normalised temporal window, or features that describe it, can be used as input to a machine learning model for classification.

The difference between these two approaches can be expressed via the analogy of analysing pictures and videos captured of the same event. In the former method, the feature extraction process will look at each picture, with no regard for any previous picture it has seen, and will try and classify the current picture into one of n discrete possible types of pictures. It will then send to the machine learning algorithm (such as a Hidden Markov Model), the index that represents the nth discrete picture. The model will now try to classify the event by looking at the order in which the last w indexed values arrived. Alternatively, in the latter method, the feature extraction process will view the entire video and create a feature vector that best describes how the video changes over time. This feature vector is then used as the input to an algorithm such as a Support Vector Machine or an Artificial Neural Network for classification.

5.1.2 The Performance Factor

Perhaps one of the most challenging aspects of using real-time gesture classification in a live performance scenario is the effect the performance itself will have on the gestures being made. This is due to the stress that is to be expected when performing any type of live piece in front of an audience (whether using real-time recognition or not). Further, the context of performance is inherrently a communication between performer and audience, so that the performer will likely adapt their movements as they 'perform' in front of their spectators. The performance context can cause the musician to make a number of temporal and spatial errors that they may not have made during rehearsals. If the machine learning model being used does not have a robust generalisation error, then any deviation from the original gestures that were performed during the training of the model in rehearsal will fail to be correctly classified during the live performance. This is of particular importance when the sensors being used are biometric, such as EMG (Electromyography) or GSR (Galvanic Skin Response), as the baseline values that are captured by these devices during the rehearsal may be drastically different from that captured during the live performance.

There are a number of steps that can be taken to mitigate this problem. The first is to use a robust calibration method in both rehearsals (when the data to train the model may be captured) and at the start of the live performance. This could be as simple as taking the minimum and maximum values of the sensors at that time and then using min-max normalisation to scale any value between a fixed output range (i.e. 0.0 - 1.0). Another option is for the performer to train their machine learning model just prior to going on stage (perhaps in a mirrored system backstage). Here it is hoped that the difference between the stress levels of the performer minutes prior to the performance and that of their levels when they are on stage will be insignificant; thus the models ability to recognise the gestures on stage should not be affected.

Alternatively, the performer could use a completely different approach and train their models live on stage. One interesting method for approaching live-training is to use a *play-along* metaphor, where the performers will mimic their gesture along to a pre-processed piece of audio (e.g. as in Merrill and Paradiso (2005) and Fiebrink et al. (2009)). At the same time the machine learning model will learn the mapping between the performer's movements and the event or effect being triggered. The performer can then take 'live-control' of the system after the model has evaluated itself using cross-validation and the resulting generalisation error has reached a pre-defined value.

5.1.3 Gesture Segmentation

One of the main problems with the recognition of multivariate temporal signals is in determining when the start and the end of a gesture occurs in a continuous stream of input data. This is one area of research across many areas within HCI that has seen little progress despite numerous attempts at solving the problem (Junker et al., 2008b). This task is difficult due to the *segmentation ambiguity* between two gestures and the *spatio-temporal variability* that occurs in all human movement (Mitra and Acharya, 2007). The segmentation ambiguity occurs when the performer moves from their current state to the *a priori* state of the next gesture. In moving to this *a prior* state the system may incorrectly classify these intermediate movements with a reference gesture, yielding what is known as a *false-positive* classification.

5.1.3.1 Trigger Keys

There are a number of methods for approaching the gesture segmentation issue. The first is to use an 'alt' or 'trigger' key (as used by Merrill and Paradiso (2005)). This method is perhaps the most robust as the system will never try to classify a gesture unless the user has triggered the 'alt' key. A similar method is to use a particular range of values of a sensor as a 'gesture zone', with the system only classifying the data when the values of that sensor are within the gesture zone. If, for example, the performer was using an accelerometer placed on their hand, they could define the gesture zone to be when they turn their palm up towards the ceiling, with no gestures being classified if their hand is not in this orientation. If spatial information can be derived from the sensors being used (such as a video camera or a magnetic tracker), then areas of space can be used as gesture zones. Using this method, a system will not try to classify any movements as a gesture until the user moves into one of the designated gesture zones.

5.1.3.2 Sliding Windows

Alternatively, the performer could use another common gesture segmentation method and use a fixed size sliding window that is run continually over the data. This method has the benefit of continually analysing the data without the user having to trigger something first. It does however have the drawback that the system may classify a body movement that is not intended to be a gesture (a null gesture) as one of the gestures in the system's trained model. The sliding window may also cut off some of the gesture if it is performed particularly slowly, or alternatively include a lot of non-gestural information in a window if the gesture is performed very quickly. To mitigate the false classification

of a null gesture, this type of segmentation method requires either a null model (such as a noise or silence model in speech recognition) or a classification threshold value (as was used in the Adaptive Naïve Bayes Classifier in the previous chapter) to stop the system classifying null gestures.

5.1.3.3 Activity Detection

Another gesture segmentation method that is frequently employed is to use a state machine to detect periods of low movement activity in one or more channels of the sensor data (as used by Yoon et al. (2001)). The performer of the gesture would intentionally insert a short pause prior to the start of a gesture and also just after they have completed it. The state machine would then detect the two periods of low activity and take the recorded data between these sections and give this to the recognition process for classification. This method has the benefit of excluding any intermediary movements from being incorrectly classified but does have the drawback of introducing an unwanted delay in the recognition process along with constraining the movements of the performer. This method is more suited as a training tool for segmenting gestures in the training stage of a system rather than a segmentation method for real-time recognition.

5.1.3.4 Musical Segmentation Cues

Music affords one important advantage for the segmentation of musical gestures as the sound, or lack-of sound, can itself be used to segment and verify that a specific gesture has started or ended. For example, a performer could use the occurrence of a specific note, such as an F#, to trigger the computer to start analysing their hand gesture directly after the playing of this note. A pitch following algorithm could be used to analyse the real-time signal from a microphone capturing the sound of the performer playing. If the trigger note has been played the computer could then analyse the movement of the performer and use this movement to control how the recently captured audio is warped/effected/mapped by the machine.

5.1.4 Multivariate Temporal Recognition Summary

Multivariate temporal gestures are more difficult for a machine learning algorithm to recognise than static postures because of the inherent variability in human movement combined with the complex task of segmenting a multivariate temporal gesture from a continuous stream of data. There are a number of methods for segmenting a multivariate temporal gesture, such as using a trigger key, continually analysing the data with a

sliding window, first applying an activity detection algorithm to remove periods of low activity from being analysed and using musical segmentation cues to trigger the start or end of an analysis window. Throughout this thesis, each multivariate temporal machine learning algorithm has been tested using both the 'trigger key' and 'sliding window' segmentation methods. The focus was placed on these segmentation methods as they both help validate the real-time classification abilities in two opposing ways; namely they validate the robustness of a classifier in the best-case scenario (i.e. when a gesture has been manually segmented using a trigger key) and the user-friendliness of a classifier (i.e. when the system automatically detects a gesture in a continuous stream of data using a sliding window). The trigger key and sliding window segmentation methods were chosen because:

1. **Trigger Key**: This segmentation method is the most robust out of the four segmentation methods as the performer is manually segmenting a gesture for the machine learning algorithm by pressing a key to indicate when a gesture starts and ends. This segmentation method therefore validates the 'best-case' classification abilities of any machine learning algorithm. Although this segmentation method may provide the most robust classification results, it is somewhat cumbersome for MCI as it forces a musician to press some form of trigger key to indicate that they are performing a gesture. This may not be a problem for some musicians as it might be possible to easily incorporate a simple trigger key on some part of their instrument. However, a large number of musicians may find a trigger key both impracticable and aesthetically unpleasing.

2. **Sliding Window**: This segmentation method is not as robust as the trigger key method, however, it has a significant advantage for MCI as it can be used to automatically segment a gesture. This may be more practicable and aesthetically pleasing for a musician as they do not need to press any key to indicate to the recognition algorithm that they have just performed a gesture.

The activity detection and musical segmentation cues are both useful segmentation methods, however, they only work in certain scenarios. For example, the activity detection algorithm only works in a scenario where a performer is able to make a small pause in their movement either before or after a gesture. Alternatively, the musical segmentation cue only works if a performer is actually playing an instrument and would therefore not be suitable for the control of a VMI or to recognise the gestures of a conductor for example. It was for these reasons therefore that neither the activity detection or musical segmentation cues would be used to validate the machine learning algorithms in this thesis. Although the aforementioned segmentation methods were not used to validate

the machine learning algorithms in this thesis, each method has been regularly used in various live recognition scenarios. EyesWeb blocks, such as the Activity Detector and State Machine blocks, have therefore been developed to enable other performers to use these segmentation tools if they believe that segmentation method is appropriate for their recognition scenario.

5.2 The Numbers-Shapes Data Set

In order to test a machine learning algorithm's ability to recognise a multivariate temporal signal, a data collection session was ran during which 10 participants where asked to perform 10 different temporal gestures, illustrated in Figure 5.1. Rather than asking each participant to perform 10 specific musical temporal gestures, 10 generic gestures were chosen instead as this assured that each participant would fully understand the gesture they needed to perform, regardless of their musical experience. Each participant was asked to 'air-draw' the following numbers and shapes:

Gesture 1-5: The numbers 1- 5

Gesture 6: A square

Gesture 7: A circle

Gesture 8: A down-beat conducting gesture

Gesture 9: A side-beat conducting gesture

Gesture 10: A triangle

5.2.0.1 Location and Setup

The recording sessions took place in December 2009 in the *Surround Sound Studio* at the Sonic Arts Research Centre, Queen's University Belfast. A Polhemus Liberty^{TM}6-degrees of freedom (DOF) magnetic tracker was used to track the participants' movements, using custom built capturing software in EyesWeb. The Polhemus was sampled at 120Hz with one sensor placed on each of the participants' wrists on top of a wristband to provide maximum comfort. Each participant was asked to stand on a marked location in the room and face a screen situated three meters directly in front of them. The screen was used to display the animations of each gesture the participant needed to perform. Small speakers where placed either side of the screen to provide audio feedback to the

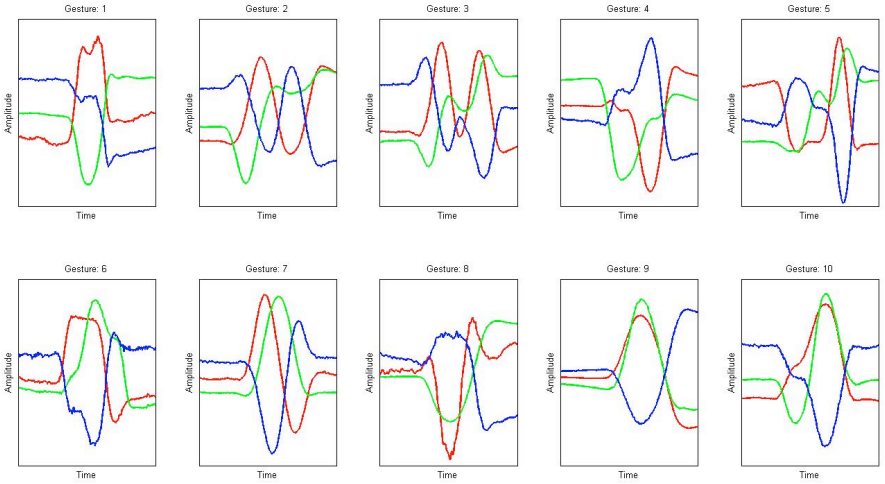


FIGURE 5.1: An example of each of the gestures in the Numbers-Shapes data set. This data is taken from the first trial of each gesture for participant one. The data has been znormalised and the red, green and blue colors represent the x, y and z axis respectively.

participant. The base-station of the magnetic sensor was placed one meter to the rear and to the left of the participants' location. All the channels of data along with the current gesture index and gesture state (see section 5.2.0.3) were recorded to a file on a laptop computer using EyesWeb.

5.2.0.2 Participants

Ten participants were recruited via a distributed email from the QUB School of Music & Sonic Arts. The sample group consisted of seven males and three females with an average age of 29 (standard deviation of 2.8). All of the participants were right-handed. None of the participants had any conditions that would have affected them in performing any of the gestures required in this experiment.

5.2.0.3 Automatic Gesture Tagging

To enable the easy extraction of the gestures from the recorded data, each participant was asked to define a 'gesture zone' with their left hand. This zone was simply an area of space that the participant would move their left hand to, prior to performing a gesture. The ANBC algorithm described in chapter 4.3 was used to recognise if the participant's left hand was either in or out of the 'gesture zone'. The participant would hold their left hand in this location, while at the same time performing the gesture with their right hand. The participant would then move their left hand from the 'gesture zone' after

they had completed the gesture. A 120Hz sine wave was played through the speakers in front of the participant to inform them that their hand was in the gesture zone and that the system was recording their gesture. EyesWeb then tagged the current motion data with either a 0 when the participant was not in the gesture zone and a 1 when they were performing a gesture (see Figure 5.2). This data was then saved to a file. Each participant was given time to practice the order in which they should move their hands for each gesture performance, i.e right hand to the *a priori* gesture position, left hand to the gesture zone, right hand performs the gesture, left hand out of the gesture zone, both arms relaxed, repeat.

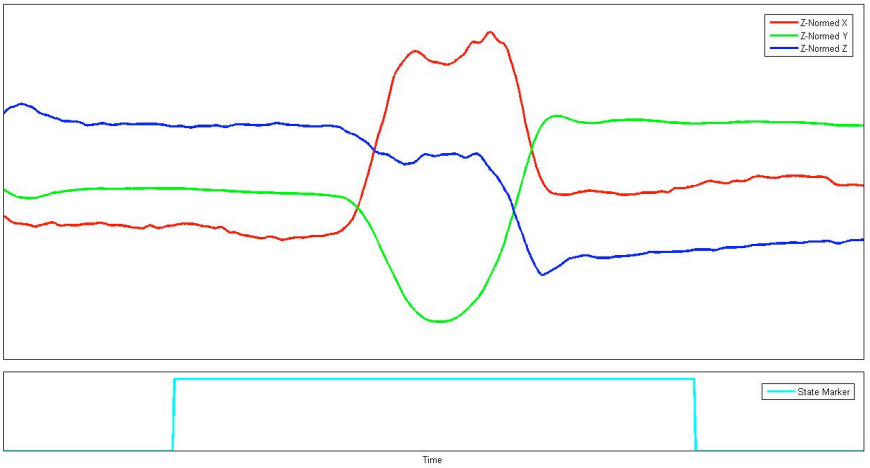


FIGURE 5.2: Tagging the gesture data with the gesture state marker.

5.2.0.4 Instructions

After filling out the consent form and questionnaire, each participant was informed of the purpose of the study and the procedure of the recording session. Each participant was then instructed where to stand and the Polhemus sensors were attached to the participant's wrists. Each participant could then define the area of space they wanted to use to indicate they were performing a gesture. Each participant was shown an animation of the gesture they next had to perform. When the participant was happy that they understood the gesture, the data collection system would be turned on and the participant would perform 25 repetitions of the gesture, with a short pause in between each repetition. The researcher present would count the number of repetitions and instruct the participant when they could stop. If the participant felt that they had made an error during a gesture, they would notify the researcher and that incorrect gesture would be removed from the data during the post-processing stage. The participant was

then shown the next gesture and repeated 25 repetitions of that until all 10 gestures were complete. The participant could rest at anytime either during or in between a gesture data collection session. The order in which each participant was presented the 10 gestures was randomised to account for fatigue.

5.2.0.5 Post-processing

Prior to classification, each participants' data was loaded into Matlab[1] and grouped into a common data structure. The 25 repetitions of each participants' gestures were segmented using the automatic gesture tagging system described in section 5.2.0.3. The segmented data was then saved to a number of file formats appropriate for subsequent processing.

5.2.0.6 Error Measures Used For Testing

The remainder of this chapter and the next two chapters present and evaluate three algorithms that can be used for the real-time recognition of multivariate temporal gestures. The following classification errors were used to evaluate the recognition abilities for each algorithm:

- Average Cross Validation Ratio:
  $(\mathbf{ACVR}) = \frac{1}{P}\sum_{p=1}^{P}\frac{1}{K}\sum_{k=1}^{K}\frac{\text{Number of correctly classified gestures in fold } k \text{ for participant } p}{\text{Total number of gestures in fold } k \text{ for participant } p}$
- Average Correct Classification Ratio:
  $(\mathbf{ACCR}) = \frac{1}{P}\sum_{p=1}^{P}\frac{\text{Number of correctly classified windows per participant } p}{\text{Total number of windows per participant } p}$
- Average Precision Ratio:
  $(\mathbf{APR}_i) = \frac{1}{P}\sum_{p=1}^{P}\frac{\text{Number of instances correctly classified for gesture } i \text{ for participant } p}{\text{Number of instances classified as gesture } i \text{ for participant } p}$
- Average Recall Ratio:
  $(\mathbf{ARR}_i) = \frac{1}{P}\sum_{p=1}^{P}\frac{\text{Number of instances correctly classified for gesture } i \text{ for participant } p}{\text{Total number of instances from gesture } i \text{ for participant } p}$
- Average Null Recall Ratio:
  $(\mathbf{ANRR}) = \frac{1}{P}\sum_{p=1}^{P}\frac{\text{Number of instances correctly classified as null for participant } p}{\text{Total number of null instances for participant } p}$

where P = the number of participants (i.e. 10) and K = the number of cross validation folds (i.e. 10).

These error values provide the following information:

[1] http://www.mathworks.co.uk/

- The ACVR provides an indication of the performance of the classifier across all the participants using cross validation
- The ACCR provides a global indication of the performance of the classifier across all the participants
- The APR provides an indication of the exactness of the classifier for each gesture across all the participants ignoring the null gestures
- The ARR provides an indication of the performance of the classifier over a specific gesture across all the participants ignoring the null gestures
- The ANRR provides an indication of the performance of the classifier at correctly classifying the null gestures

5.3 Hidden Markov Models

Hidden Markov Models (HMMs) are frequently used across a wide range of fields, such as speech recognition (e.g., Rabiner, 1989 and Inoue et al., 2011), on-line handwriting recognition (e.g., Al-Muhtaseb et al., 2008 and Awaidah and Mahmoud, 2009) and hand gesture recognition (e.g., Chen et al., 2003 and Al-Rajab et al., 2008), for the classification of sequential data. The standard Hidden Markov Model requires a discrete integer value, otherwise known as a *symbol*, as input for classification as opposed to a N-dimensional feature vector. An HMM can be extended to a number of continuous probability distributions, such as in the work by Pylvänäinen (2005), but the following description of the algorithm will focus on discrete probability distributions. In this instance an N-dimensional feature vector must be mapped to a discrete integer symbol using what is commonly known as a *codebook*. Prior to describing the HMM algorithm, a description of how an N-dimensional signal can be quantised into a discrete integer symbol using a technique known as *vector quantisation* will be presented.

5.3.1 Vector Quantisation

The standard HMM algorithm works under the assumption that the input to the algorithm for both training and testing will be a 1-dimensional vector of length T containing discrete observation symbols in the range of 1 to M. This means that, for many real-world problems, the output of an N-dimensional sensor must be quantised to a discrete observation symbols using a codebook.

5.3.2 Vector Quantisation Using k-means Clustering

The k-means clustering algorithm (MacQueen, 1967) is commonly used as a method for vector quantisation (e.g. as used by Yoon et al. (2001) and Chen et al. (2003)). Using the training set, the k-means algorithm attempts to group the training data into K clusters (with K set manually by the user). The K cluster centers can then be used to quantise any new datum by finding which cluster center the datum is closest to.

5.3.2.1 Training the k-means Algorithm

K-means is an unsupervised machine learning algorithm and can therefore be trained by grouping the unlabeled training data set $\mathbf{x} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_M\}$, consisting of M observations of a random N-dimensional Euclidean variable x, into K clusters. Each cluster is represented by $\boldsymbol{\mu}_k$, a N-dimensional vector containing the centers of the cluster.

The k-means algorithm is trained using a classic two-stage expectation maximization (**EM**) approach, where in the E-step, each of the data points is assigned to the kth cluster center that results in the minimum sum of the squares distance between the data point and the cluster's center. In the M-step, each of the cluster centers, $\boldsymbol{\mu}_k$, is then updated by recomputing the mean of all the data points that belong to that cluster. This two-step training process is repeated until either a set number of iterations has been reached (i.e. 1000) or until a convergence criteria has been reached (i.e. the cluster centers have not moved beyond a given threshold). The initial values for $\boldsymbol{\mu}_k$ are commonly set to the values of K random data points.

The k-means clustering algorithm is therefore (MacQueen, 1967):

1. Initialize $\boldsymbol{\mu}_k$ to a random data point from $\mathbf{x}$
2. **E-Step**: Set the ith class label, $\mathbf{c}_i$, to the kth closest cluster

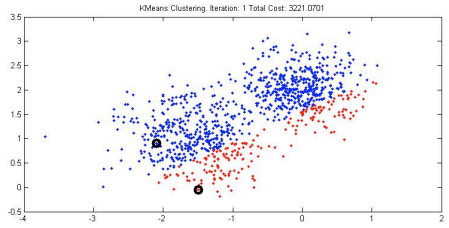
$$\mathbf{c}_i = \arg\min_k \sum_{j=1}^{N} \left(\mathbf{x}_{ij} - \boldsymbol{\mu}_{kj} \right)^2 \qquad 1 \leq i \leq M \tag{5.3.1}$$

3. **M-Step**: Set $\boldsymbol{\mu}_k$ to the mean of all of the data points assigned to cluster k
4. Repeat steps 2 - 3 until convergence

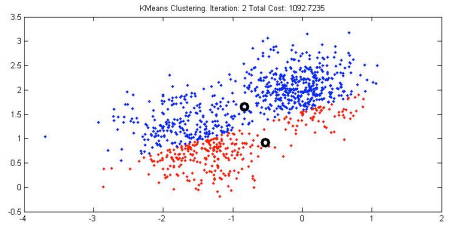
At each iteration, the objective function, θ, can be computed indicating the overall effectiveness of the algorithm. This function, also called a *distortion measure* (Bishop, 2006), represents the sum of the squares distances of each data point to its assigned cluster center. The overall goal of the k-means algorithm is to find the values of c_i and $\boldsymbol{\mu}_k$ so as to minimize θ.

$$\theta = \sum_{i=1}^{M} ||\mathbf{x}_i - \boldsymbol{\mu}_{c_i}||^2 \tag{5.3.2}$$

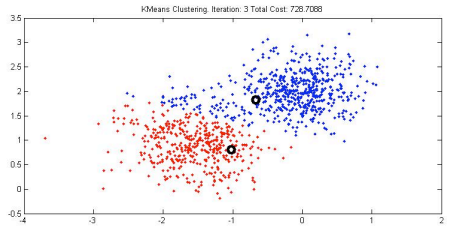
Using θ, the most suitable initial starting points for the clusters centers can be found by running several instances of the k-means algorithm and checking which instance has the minimum value of θ after the first few iterations. The instance with the minimum value of θ can then be continued until it has converged with the remaining instances stopped. Figure 5.3 illustrates the k-means clustering algorithm.



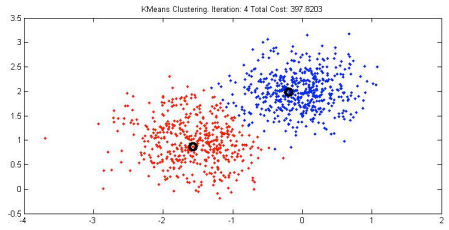
(a) Iteration 1



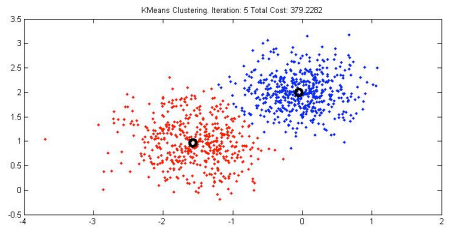
(b) Iteration 2



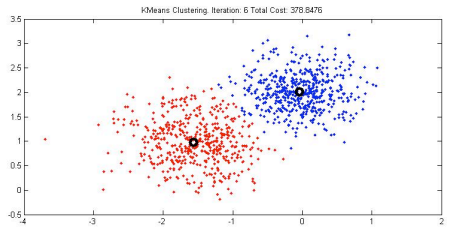
(c) Iteration 3



(d) Iteration 4

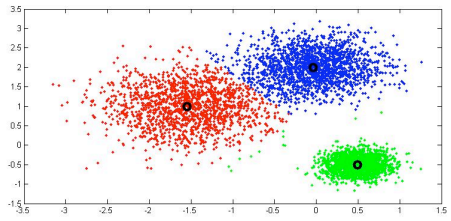


(e) Iteration 5

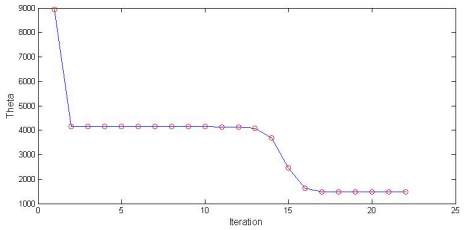


(f) Iteration 6

FIGURE 5.3: An illustration of the k-means clustering algorithm on a data set generated from two Gaussian distributions. The clustered data points are shown by the red and blue points with the cluster centers circled in black. For this problem, the algorithm converged in just 6 iterations.



(a) The converged clusters with each class shown by the red, green and blue points. The black circles indicate the cluster centers

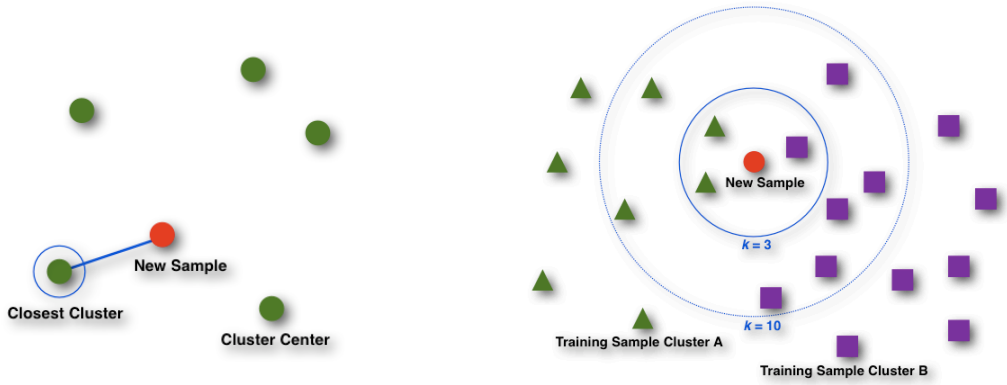


(b) The Objective Function for each Iteration

FIGURE 5.4: An illustration of the k-means algorithm being run on 3 classes of data generated by 3 Gaussian distributions. The left image shows the converged clusters after 25 iterations with the right image showing the objective function at each iteration.

5.3.2.2 Quantisation using the k-means Algorithm

After the k-means algorithm has converged the clustered training data can be used to create the codebook. Two methods are commonly used to quantise a new training/test sample using the pre-clustered data. The first is to compute the distance (typically Euclidean) between the new sample and each of the k-cluster centers, with the new sample's quantisation value set to the ID of the cluster center resulting in the minimum distance. The second method is to use all the pre-clustered data, rather than just the cluster centers, and use the k-nearest neighbor algorithm (k-NN) to compute the new sample's quantisation value. k-NN will assign a new sample's quantisation value to the ID which is most frequent among the k training samples nearest to that sample in the k-means cluster feature space.



(a) Quantisation using the minimum cluster center

(b) Quantisation using the k-nearest neighbor algorithm. Note that the new sample would be assigned to a different cluster if a different k value is used. Cluster A with $k = 3$ and cluster B with $k = 10$.

FIGURE 5.5: Quantisation using the minimum cluster center or the k-nearest neighbor algorithm.

The k-NN algorithm would be more appropriate to use if the between-cluster center distance was small, particularly if the within-cluster distance was large. It does, however, require more memory and is computationally more expensive than the minimum cluster distance algorithm, as every training example needs to be stored and sorted in a search tree. The minimum cluster distance algorithm alternatively only needs to store the cluster centers for each cluster. For this reason, the minimum cluster distance was used to quantise the numbers-shapes data in the experiments that are presented later in this chapter.

5.3.3 Vector Quantisation Using SAX

For many real-world classification problems, it was found that simply running the k-means clustering algorithm on the raw data (a) took too long as there was a large amount of data points to cluster and (b) did not work well when the data was noisy. To mitigate this, the data was first quantised using Symbolic Aggregate Approximation (**SAX**) after which the k-means clustering algorithm could be used to further quantise the SAX data.

SAX is a technique for numerosity reduction that was first proposed by Lin et al. (2003) and has since been used in a number of applications (e.g. Kasten et al., 2007, Lin et al. (2007) and Ergovic et al., 2009). SAX reduces a 1-dimensional time-series, $\mathbf{x} = \{x_1, x_2, \ldots, x_L\}$, of arbitrary length L to a string, $\hat{\mathbf{x}} = \{x_1, x_2, \ldots, x_S\}$, of arbitrary length S, where $S < L$, containing discrete values within a finite range. Each value of $\hat{\mathbf{x}}$ will be in the range of 1 to a, where a is the size of the alphabet being used. This is illustrated by Figure 5.6.

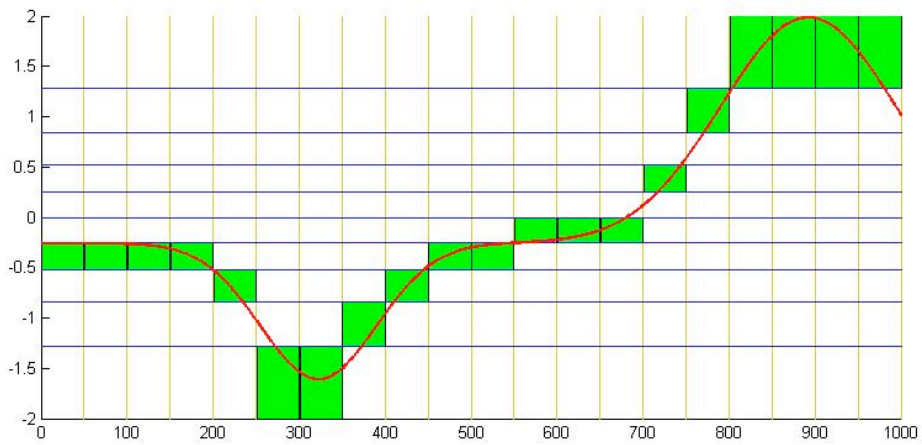


FIGURE 5.6: An illustration of the SAX algorithm being used to quantise a 1-dimensional waveform. The waveform is shown in red with the frame edges shown in yellow and the break-points shown by the horizontal blue lines. The quantised alphabet value is shown in green. Here the frame size (f) is set to 20 with an alpha size (a) set to 10.

Prior to discretising $\mathbf{x}$, the time-series is first normalised to have zero mean and unit variance. This is performed as it ensures that each time-series has a standard offset and amplitude variance. It is also required as the discretisation process for SAX uses a technique that produces symbols with equal probability that requires the data to have zero mean and unit variance. After the time-series has been normalised it is segmented into f frames, with the size of each frame (i.e. the number of data points in each frame) given by f/L. The mean value of each frame is calculated and it is this value that is

discretised using the alphabet. The discrete alphabet value assigned to the mean of the ith frame is given by:

$$\hat{\mathbf{x}}_i = a_j \qquad \text{if} \quad \Psi_{j-1} \leq \bar{\mathbf{x}}_i < \Psi_j \tag{5.3.3}$$

where $\bar{\mathbf{x}}_i$ is the mean value of the ith frame and Ψ is a lookup vector containing the break-point values of each letter in the alphabet. The break-point values for each letter in the alphabet are set so that the area under a Gaussian distribution of $\mathcal{N}(0,1)$ from Ψ_{j-1} to Ψ will be equal to $1/a$.

If the input time-series, $\mathbf{x}$, is in-fact a N-dimensional time-series as opposed to a 1-dimensional time-series, then each dimension is quantised independently using the SAX algorithm. The N quantised time-series are then recombined into a new N-dimensional time-series, $\hat{\mathbf{x}}$ which is used as input to the k-means quantisation algorithm. This is illustrated in Figure 5.7.

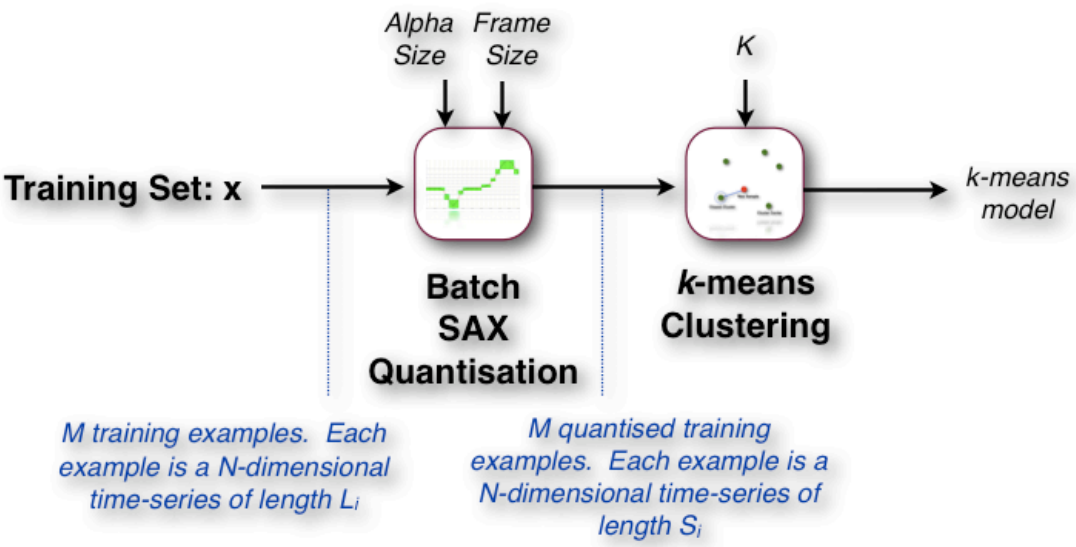


FIGURE 5.7: An illustration of the quantisation training phase in which the training set is first batch quantised by the SAX algorithm, with the resulting quantised training examples being used as input to the k-means clustering algorithm. The k-means algorithm will cluster the quantised training examples into K clusters, after which the K cluster centers can be used to create the codebook that is required for the HMM algorithm.

5.3.4 HMM Description

A Hidden Markov Model can be viewed as a state space model consisting of N discrete states. Given an observation sequence $\mathbf{O} = \{O_1, O_2, \ldots, O_T\}$ of length T, a trained HMM can estimate not only the probability of a single observation sequence being omitted by a given state, but also the probability of an entire observation sequence occurring given the model. This latter ability is key to the adaptation of HMMs for the recognition of musical gestures as G separate HMMs can be trained, one for each of the G gestures in a database, and an observation sequence (i.e. some real-time stream of data) can be classified as being the gth model that gives the maximum likelihood. In this work the standard HMM algorithm was specifically adapted to enable it to be used to classify multivariate temporal musical gestures from a continuous stream of data that also contains non-gestural data. Prior to describing how the algorithm has been adapted its foundations will first be described. The following description is based on the excellent HMM tutorial by Rabiner (1989).

5.3.5 HMM Components

Each HMM consists of the following elements:

1. N: The number of discrete states in the model. The state at time t is: $t = q_t$.
   Individual states are: $\mathbf{S} = \{S_1, S_2, \ldots, S_N\}$.

2. M: The number of discrete observation symbols per state.
   Individual symbols are: $\mathbf{V} = \{V_1, V_2, \ldots, V_M\}$.

3. A N-by-N matrix $\mathbf{A}$, called the transition matrix, giving the transition probability of the ith state moving to the jth state, where:

$$\mathbf{A}_{ij} = P(q_{t+1} = \mathbf{S}_j | q_t = \mathbf{S}_i) \qquad 1 \leq i, j \leq N \tag{5.3.4}$$

   As $\mathbf{A}$ is composed of probabilities, it must have the properties that satisfy:

$$0 \leq \mathbf{A}_{ij} \leq 1 \qquad 1 \leq i, j \leq N \tag{5.3.5}$$

   with

$$\sum_{j=1}^{N} \mathbf{A}_{ij} = 1 \qquad 1 \leq i \leq N \tag{5.3.6}$$

4. A N-by-M matrix $\mathbf{B}$, called the emissions matrix, that denotes the probability of symbol m being emitted from the state n. Each element of $\mathbf{B}$ is given by:

$$\mathbf{B}_{ij} = P(\mathbf{V}_j \text{ at } t | q_t = \mathbf{S}_i) \qquad 1 \leq i \leq N, 1 \leq j \leq M \tag{5.3.7}$$

with the normalisation condition of:

$$\sum_{j=1}^{M} \mathbf{B}_{ij} = 1 \qquad 1 \leq i \leq N \tag{5.3.8}$$

5. The initial state distribution for $t = 1$, $\boldsymbol{\pi} = \{\pi_1, \pi_2, \ldots, \pi_i\}$, where

$$\boldsymbol{\pi}_i = P(q_1 = \mathbf{S}_i) \qquad 1 \leq i \leq N \tag{5.3.9}$$

with the normalisation condition of:

$$\sum_{i=1}^{N} \boldsymbol{\pi}_i = 1 \tag{5.3.10}$$

6. A threshold value, τ, that enables the rejection of null gestures

An entire HMM is therefore given by:

$$\lambda = (\mathbf{A}, \mathbf{B}, \boldsymbol{\pi}, N, M, \tau) \tag{5.3.11}$$

An HMM based recognition system with G gestures is given by:

$$\boldsymbol{\lambda} = \{\lambda_1, \lambda_2, \ldots, \lambda_G\} \tag{5.3.12}$$

5.3.6 The Three Basic Problems for HMMs

With an HMM, there are generally three main problems to resolve:

1. Given the observation sequence $\mathbf{O} = \{O_1, O_2, ..., O_T\}$, and a model λ, how can $P(\mathbf{O}|\lambda)$, the probability of the observation sequence given the model, be efficiently computed?

2. Given the observation sequence $\mathbf{O} = \{O_1, O_2, ..., O_T\}$, and a model λ, how can a corresponding state sequence $\mathbf{S} = \{S_1, S_2, ..., S_T\}$ be chosen which is optimal in some meaningful sense (i.e. that best explains the observations)?

3. How can the model parameters $\lambda = (\mathbf{A}, \mathbf{B}, \boldsymbol{\pi})$ be adjusted to maximise $P(\mathbf{O}|\lambda)$?

Problem 1 can be viewed as the real-time prediction phase. In this phase the probability of viewing the captured observation sequence, given this particular model, needs to be computed. This problem can be solved using the **Forward-Backward** algorithm. Problem 2 can be viewed as attempting to uncover the most likely state sequence given a particular observation sequence. This can be solved using the **Viterbi** algorithm. Problem 3 can be viewed as the model training phase and can be solved by the **Baum-Welch** algorithm. For the recognition of musical gestures, problems 1 and 3 are the most critical to solve as these are used for real-time prediction and training respectively. The Viterbi algorithm, which is used to solve problem 2, will therefore be omitted from the following explanation as it is more useful for finding the optimal state sequences for continuous speech recognition for example. Interested readers can find a detailed description of the Viterbi algorithm in Rabiner (1989).

5.3.7 The Forward-Backward Algorithm

To enable the real-time recognition of a musical gesture, the probability of viewing the observation sequence $\mathbf{O}$, given the model λ, needs to be computed:

$$P(\mathbf{O}|\lambda) \tag{5.3.13}$$

One method of achieving this would be to calculate every possible state sequence of length T (the number of observations) and to add the joint probability over all the possible state sequences. This fairly straightforward approach has some severe drawbacks however, as it would require $O(2T(N^T))$ calculations, as at every time t, there are N possible states which can be reached and therefore there are N^T possible state sequences. This means that this method becomes unfeasible, even for small values of T and N. Thankfully an algorithm exists to resolve this issue, in the case of the Hidden Markov Model this is known as the *Forward-Backward* algorithm (Baum and Eagon, 1967, Baum and Sell, 1968). There are several variants of this basic algorithm, of which the most common, known as the *alpha-beta* algorithm, will be applied.

The forward-backward algorithm works under the assumption that, for the model to be in a given state at time t then it must have moved there from a given state (which could have been the same state) at time $t-1$ and it will advance to a given state (again this could be the same state) at time $t+1$. This basic assumption can be used to calculate, for each state in the model, the probability of advancing to that state at time t given all the partially observed data up until t along with the next observation at time $t+1$, i.e. the forward estimate, and the probability of being in that state at time t given all the future observed data, i.e. the backward estimate. The key to the

forward-backward algorithm is that since there are only N states in the model, all the possible state sequences must re-emerge into these N nodes, no matter how long the observation sequence. This reduces the number of computations from $O(2T(N^T))$ for the naïve algorithm above to $O(N^2T)$ for the forward-backward algorithm, thus changing the computational complexity of the algorithm from exponential to linear as a function of the observations.

5.3.7.1 The Alpha-Beta Algorithm

To implement the alpha-beta version of the forward-backward algorithm, two new T-by-N matrices are required to keep track of the forward and backward estimates at each time step (the forward matrix, $\boldsymbol{\alpha}$, and the backward matrix, $\boldsymbol{\beta}$). At each time step, t, the forward estimate for the jth state is given by the joint probability of observing the next observation $(t+1)$ in state j with the previous accumulated forward estimate at time t for state i along with the probability of moving from state i to state j:

$$\boldsymbol{\alpha}_{t+1j} = \sum_{i=1}^{N} \boldsymbol{\alpha}_{ti}\mathbf{A}_{ij}\mathbf{B}_{j(\mathbf{O}_{t+1})} \tag{5.3.14}$$

or, as j is constant over each summation and the observation emission probability of $\mathbf{B}$ will therefore be constant, equation (5.3.14) can be rewritten as:

$$\boldsymbol{\alpha}_{t+1j} = \left[\sum_{i=1}^{N} \boldsymbol{\alpha}_{ti}\mathbf{A}_{ij}\right]\mathbf{B}_{j(\mathbf{O}_{t+1})} \tag{5.3.15}$$

The backward estimate at time t for the jth state is given by the joint probability of observing the next observation $(t+1)$ in state j with the last computed accumulated backward estimate for state j at time $t+1$ along with the probability of moving from state i to state j:

$$\boldsymbol{\beta}_{t(i)} = \sum_{j=1}^{N} \mathbf{A}_{ij}\mathbf{B}_{j(\mathbf{O}_{t+1})}\boldsymbol{\beta}_{t+1j} \tag{5.3.16}$$

5.3.7.2 The Forward Algorithm

The forward estimate is given by:

1. Initialization:

$$\boldsymbol{\alpha}_{1i} = \boldsymbol{\pi}_i \mathbf{B}_{i(\mathbf{O}_1)} \qquad 1 \leq i \leq N \tag{5.3.17}$$

2. Induction:

$$\boldsymbol{\alpha}_{t+1j} = \left[\sum_{i=1}^{N} \boldsymbol{\alpha}_{ti} \mathbf{A}_{ij} \right] \mathbf{B}_{j(\mathbf{O}_{t+1})} \qquad 1 \leq j \leq N$$

$$1 \leq t \leq T-1 \tag{5.3.18}$$

3. Termination:

$$P(\mathbf{O}|\lambda) = \sum_{i=1}^{N} \boldsymbol{\alpha}_{Ti} \tag{5.3.19}$$

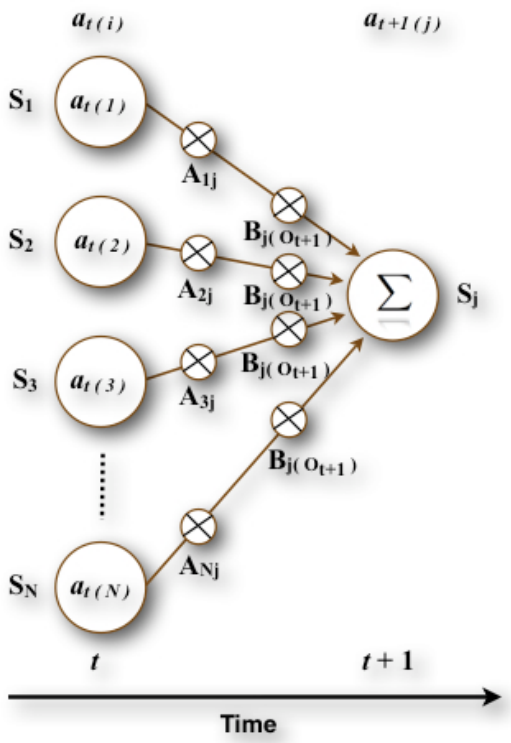


FIGURE 5.8: An illustration of one forward estimate at time t for state j.

The forward estimate is illustrated by Figure 5.8. It should be noted that to calculate $P(\mathbf{O}|\lambda)$, the likelihood of the observation sequence $\mathbf{O}$ occurring given the model, only the

forward part of the forward-backward algorithm is required. The backward algorithm does provide significant advantages for helping to solve problem 3, the actual training of a HMM, and it is therefore beneficial to describe it along with the forward algorithm.

5.3.7.3 The Backward Algorithm

The backward estimate is given by:

1. Initialization:

$$\boldsymbol{\beta}_{Ti} = 1 \qquad 1 \leq i \leq N \tag{5.3.20}$$

2. Induction:

$$\boldsymbol{\beta}_{ti} = \sum_{j=1}^{N} \mathbf{A}_{ij}\mathbf{B}_{j(\mathbf{O}_{t+1})}\boldsymbol{\beta}_{t+1j} \qquad 1 \leq i \leq N$$
$$T-1 \geq t \geq 1 \tag{5.3.21}$$

The backward estiamte is illustrated by Figure 5.9.

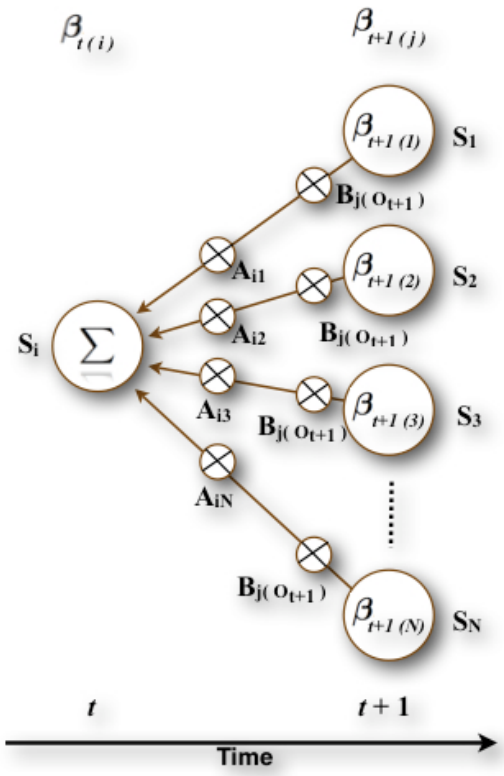


FIGURE 5.9: An illustration of one backward estimate at time t for state i.

5.3.8 The Baum-Welch Algorithm

Solution of the third problem for a Hidden Markov Model (how can the model parameters $\lambda = (\mathbf{A},\mathbf{B},\boldsymbol{\pi})$ be maximized with respect to $\mathbf{O}$) can be solved using the *Baum-Welch* algorithm. When used in conjunction with the forward-backward algorithm, the Baum-Welch algorithm can be viewed as a classic example of an expectation-maximization (EM) algorithm. In the expectation (E) step, the forward-backward algorithm is used to compute the expectation of the log-likelihood using the current estimated values of the matrices $\mathbf{A}$ and $\mathbf{B}$. This is then followed with the maximization (M) step, in which the Baum-Welch algorithm re-estimates the parameters of $\mathbf{A}$ and $\mathbf{B}$ by maximizing the expected log-likelihood function found in the E step. The two steps are then repeated with the updated values until the algorithm converges.

Baum et al. (1970) proved that, when used in conjunction with the forward-backward algorithm, their algorithm is guaranteed to either define a critical point of the likelihood function, $P(\mathbf{O}|\lambda)$ or improve the original estimate of λ so that the new model gives a greater likelihood of viewing the observation sequence $\mathbf{O}$. Unfortunately, any maxima found will only be guaranteed to be a local maxima only and in many practical problems, the optimization surface will be very complex with many local maxima.

The Baum-Welch algorithm works by counting, so that the re-estimation of the state transitions matrix, $\hat{\mathbf{A}}_{ij}$, is given by:

$$\begin{aligned}\hat{\mathbf{A}}_{ij} &= \frac{\text{expected number of transitions from state } S_i \text{ to } S_j}{\text{expected number of transitions from state } S_i} \\ &= \frac{\sum_{t=1}^{T-1} \boldsymbol{\alpha}_{ti}\mathbf{A}_{ij}\mathbf{B}_{j(\mathbf{O}_{t+1})}\boldsymbol{\beta}_{t+1j}}{\sum_{t=1}^{T-1} \boldsymbol{\alpha}_{ti}\boldsymbol{\beta}_{ti}}\end{aligned} \tag{5.3.22}$$

and the re-estimation of the state emissions matrix, $\hat{\mathbf{B}}_{ij}$, is given by:

$$\begin{aligned}\hat{\mathbf{B}}_{ij} &= \frac{\text{expected number of times in state } i \text{ and observing symbol } \mathbf{V}_j}{\text{expected number of times in state } i} \\ &= \frac{\sum_{t=1}^{T-1} \mathbf{1}\{\boldsymbol{\alpha}_{ti}\boldsymbol{\beta}_{ti}\}}{\sum_{t=1}^{T-1} \boldsymbol{\alpha}_{ti}\boldsymbol{\beta}_{ti}}\end{aligned} \tag{5.3.23}$$

where $\mathbf{1}\{\cdot\}$ is the indicator bracket that gives 1 if $\mathbf{O}_t = j$ and zero otherwise.

The re-estimation of $\hat{\boldsymbol{\pi}}_i$ is set as the number of times the model starts in state i at time $t = 1$:

$$\begin{aligned}\hat{\boldsymbol{\pi}}_i &= \text{expected frequency in state } \mathbf{S}_i \text{ at time } t = 1 \\ &= \frac{\boldsymbol{\alpha}_{1i}\boldsymbol{\beta}_{1i}}{\sum_{j=1}^{N} \boldsymbol{\alpha}_{1j}\boldsymbol{\beta}_{1j}}\end{aligned} \tag{5.3.24}$$

An important feature of the re-estimation procedure is that the stochastic constraints of the HMM parameters are automatically satisfied at each iteration. Therefore the following holds true:

$$\sum_{j=1}^{N} \hat{\mathbf{A}}_{ij} = 1 \qquad 1 \leq i \leq N \tag{5.3.25}$$

$$\sum_{j=1}^{M} \hat{\mathbf{B}}_{ij} = 1 \qquad 1 \leq i \leq N \tag{5.3.26}$$

$$\sum_{i=1}^{N} \hat{\boldsymbol{\pi}}_i = 1 \tag{5.3.27}$$

By computing the forward-backward algorithm followed by the Baum-Welch algorithm on an observation sequence, not only can an estimation of the states of the model be calculated, but also importantly, just from the observation sequence, an estimation of the model itself. An HMM can therefore be trained by initially setting the parameters **A,B** and $\boldsymbol{\pi}$ to normalised random values and then continually running the forward-backward algorithm followed by the Baum-Welch algorithm until the algorithm converges to a local maximum. The training algorithm can be considered to have converged at a local maximum if the change in **A** and **B** is less than a pre-determined threshold value. The training algorithm could also be stopped if it has not converged after a set number of iterations (e.g. 1000).

Finding a 'Semi-Optimal' Starting Position

Advantage can be taken of the observation that many problems quickly come close to their converged log-likelihood after only a few iterations, as illustrated by Figure 5.10. This facilitates a 'semi-optimal' local maxima to be found by repeatedly instigating a number of training iterations (e.g. 10) of the forward-backward algorithm followed by the Baum-Welch re-estimation from different random starting points of **A,B** and $\boldsymbol{\pi}$. After a number of EM iterations (e.g. 2) each instance of the training algorithm can be stopped and the full EM training can be performed using the current values of $\mathbf{A},\mathbf{B},\boldsymbol{\pi}$ that gave the best log-likelihood value at the end of a number of EM test iterations.

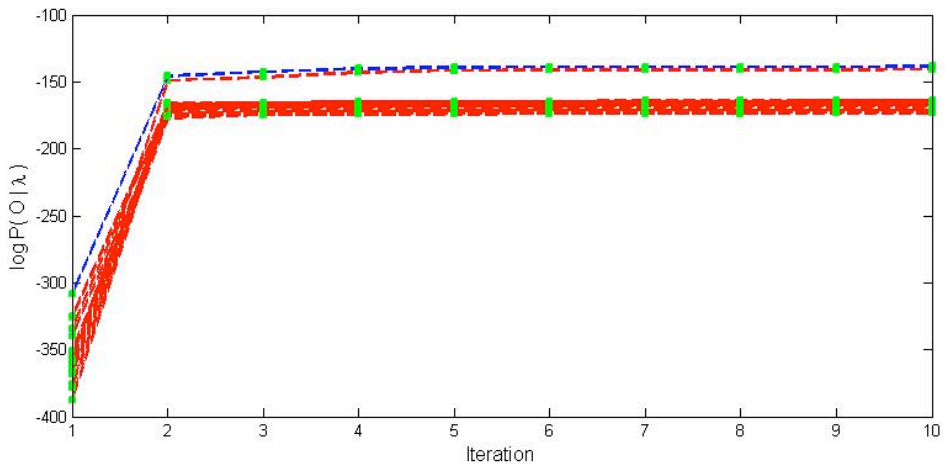


FIGURE 5.10: An illustration of the fast approximation of a model's final converged log-likelihood value. This Figure shows the log-likelihood error value during the first 10 training iterations of a simple 10 state model. The model was trained using 20 different random starting points of **A** and **B**. After just one iteration it is already clear which of the 20 training instances will provide the maximal converged log-likelihood (i.e. the blue line).

Multi-Threaded Training

One major advantage of using the forward-backward/Baum-Welch algorithm as a training method for an HMM is that each model (i.e. each gesture) can be trained independently from the other models. This is of particular use on new machines that feature multiple processors as a multi-threaded training approach can be adopted in which each model's training routine is launched in a separate thread. This training approach greatly speeds up the overall training time for an HMM classification system as one model does not need to wait for the previous model to be trained before it can start its own training routine.

The forward-backward/Baum-Welch algorithm also has one other advantage in that, if a new gesture is added to an existing trained database or an existing gesture is removed, the entire database does not need to be retrained. Instead, a new model only needs to be trained for the new gesture, thus greatly reducing the training time. If an existing gesture is removed from the database then no training is required as the HMM classification system simply removes this model from its database. This is not the case for other machine learning algorithms, such as an Artificial Neural Network, as the entire system would need to be retrained from scratch any time a new gesture is added or removed.

5.3.9 Model Types

The previous section on the real-time prediction and training of a Hidden Markov Model has been described under the assumption that every state can be reached from every other state within the model in a finite number of steps. This type of model, referred to as an ergodic model or a fully connected model, may not provide the optimum representation of the problem being modeled. Instead, other HMM designs have been proposed that may provide a more appropriate account for the observed properties of the signal being modeled. One popular model design for the classification of a signal that evolves over time, such as in a speech or gesture recognition problem, is the *left-right* model. This model, also known as a Bakis model, has state transition probabilities that satisfy:

$$\mathbf{A}_{ij} = 0 \qquad \text{if } j < i \tag{5.3.28}$$

This does not allow any state transition to a state whose indices are lower than the current state, forcing the model to either remain in its current state or progress to a state with a higher (i.e. to the right of) state value than the current state. The left-right model also has the properties of:

$$\boldsymbol{\pi}_i = \begin{cases} 0, & i \neq 1 \\ 1, & i = 1 \end{cases} \tag{5.3.29}$$

which forces the model to start in state 1 (i.e. the left-most state) and therefore end in state N (i.e. the right-most state). The left-right model is illustrated in Figure 5.11, whereas Figure 5.12 illustrates an ergodic model.

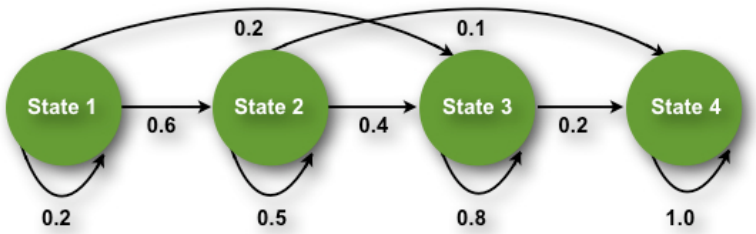


FIGURE 5.11: An illustration of a 4 state second-order left-right Hidden Markov Model. The values show the transition probabilities of state i moving to state j. For this example $\Delta = 2$.

The number of steps that a model may progress to can also be limited, allowing the model to only move to its own state or the state directly to its right; for example in a first order left-right model. This can be useful for gesture recognition for example as it

stops the model 'skipping' out the middle of a gesture by accident. This can be set by:

$$\mathbf{A}_{ij} = 0, \qquad \text{if } j > i + \Delta \tag{5.3.30}$$

where Δ is the number of steps that a model may progress to during one estimation update.

One advantage of using the look-up matrices $\mathbf{A}$ and $\mathbf{B}$ is that, regardless if the model type is left-right, ergodic or some variation of the two, no modification of the training or predication process needs to be changed. This is because any state transitions value initially set to 0 will remain at 0 throughout the re-estimation process.

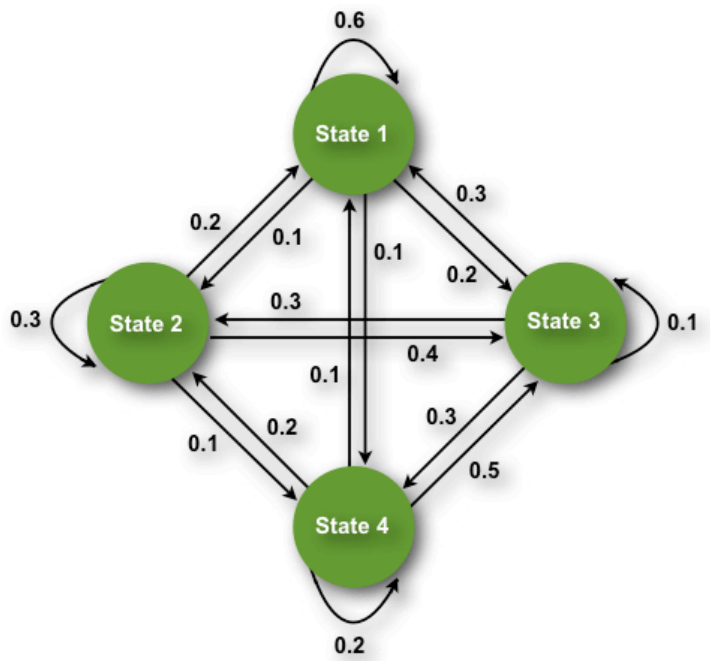


FIGURE 5.12: An illustration of a 4 state ergodic Hidden Markov Model. The values show the transition probabilities of state i moving to state j. Note that a state can transition back to itself and that the total probabilities for each state sums to 1.

5.3.10 Scaling

For a Hidden Markov Model, one real-world computational factor needs to be taken into consideration when actually implementing the algorithm. This is related to the recursive nature of the forward-backward algorithm as, for example, each new value of $\boldsymbol{\alpha}_{t+1j}$ is obtained by:

$$\boldsymbol{\alpha}_{t+1j} = \left[\sum_{i=1}^{N} \boldsymbol{\alpha}_{ti}\mathbf{A}_{ij}\right]\mathbf{B}_{j(\mathbf{O}_{t+1})} \qquad 1 \leq j \leq N$$
$$1 \leq t \leq T-1 \tag{5.3.31}$$

As each of the probability coefficients in $\mathbf{A}$ and $\mathbf{B}$ are either equal to or less than one, sometimes significantly less than one, this means that as the algorithm progresses over t, the values of $\boldsymbol{\alpha}_{t+1j}$ will exponentially decrease towards zero. This will force the calculation of $\boldsymbol{\alpha}$ to quickly underflow the dynamic range of the computer, even if double precision floating point is used. The same also holds true for $\boldsymbol{\beta}$.

To alleviate this problem, $\boldsymbol{\alpha}_{ti}$ and $\boldsymbol{\beta}_{ti}$ can be scaled using a scaling coefficient that is independent of i but instead depends on t. The key to this is to use the same scaling factor for $\boldsymbol{\beta}$ as was used for $\boldsymbol{\alpha}$ at each time t. The scaling of $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ does not need to be performed at every time-step, but can instead be performed when the values of either $\boldsymbol{\alpha}$ or $\boldsymbol{\beta}$ start to approach some pre-set threshold. If scaling is not performed then the scaling coefficients are set to 1 at that time.

The scaling coefficient, $\boldsymbol{\phi}_t$ at time t can be simply set as the sum over all the states at $\boldsymbol{\alpha}_{ti}$:

$$\boldsymbol{\phi}_t = \frac{1}{\sum_{i=1}^{N} \boldsymbol{\alpha}_{ti}} \tag{5.3.32}$$

therefore a new forward estimate is given by:

$$\boldsymbol{\alpha}_{t+1j} = \left[\sum_{i=1}^{N} \hat{\boldsymbol{\alpha}}_{ti}\mathbf{A}_{ij}\right]\mathbf{B}_{j(\mathbf{O}_{t+1})} \tag{5.3.33}$$

with the scaled $\hat{\boldsymbol{\alpha}}_{ti}$ set as:

$$\hat{\boldsymbol{\alpha}}_{ti} = \boldsymbol{\alpha}_{ti}\boldsymbol{\phi}_t \tag{5.3.34}$$

as $\hat{\boldsymbol{\beta}}_{ti}$, the scaled version of $\boldsymbol{\beta}_{ti}$, uses the same scaling coefficient it is given by:

$$\hat{\boldsymbol{\beta}}_{ti} = \boldsymbol{\beta}_{ti}\boldsymbol{\phi}_t \tag{5.3.35}$$

One nice feature of using scaling is that it does not effect the re-estimation equation, as $\boldsymbol{\phi}$ is canceled out of both the numerator and denominator and the original estimation equation therefore remains. The only real change to the HMM procedure because of the scaling process is in computing $P(\mathbf{O}|\lambda)$ as the values of $\hat{\boldsymbol{\alpha}}_{Ti}$ cannot simply be added together as they are already scaled. We can, however, make use of the property that

the unscaled estimates of $\boldsymbol{\alpha}$ at time T combined with all the scaling coefficients give:

$$\prod_{t=1}^{T} \boldsymbol{\phi}_t \sum_{i=1}^{N} \boldsymbol{\alpha}_{Ti} = 1.0 \tag{5.3.36}$$

therefore

$$\prod_{t=1}^{T} \boldsymbol{\phi}_t P(\mathbf{O}|\lambda) = 1.0 \tag{5.3.37}$$

which can be rearrange to give

$$P(\mathbf{O}|\lambda) = \frac{1}{\prod_{t=1}^{T} \boldsymbol{\phi}_t} \tag{5.3.38}$$

finally, the log of $P(\mathbf{O}|\lambda)$ can be computed as

$$\log\ P(\mathbf{O}|\lambda) = -\sum_{t=1}^{T} \log\ \boldsymbol{\phi}_t \tag{5.3.39}$$

Therefore, the log of the likelihood of observing $\mathbf{O}$ given the model can be computed, however the likelihood itself can not as it would be out of the dynamic range of the computer (Press et al., 2007).

5.3.11 Batch Training

Until now, it has been assumed that the Hidden Markov Model was only being trained with one observation sequence, $\mathbf{O}$ of length T. In many practical problems, however, a more robust model can be trained by collecting multiple observation sequences, i.e. record multiple trials of a musician performing a gesture. Training an ergodic model with multiple observation sequences not only helps to create a more robust estimation of the parameters, it is also a necessity for the case where the left-right design is being used. This is because, as the model can only stay in the same state or progress to a higher state, only a small number of observations will be recorded by each state before it progresses to the next state. This could cause certain emission probabilities to quickly head towards zero if only one observation sequence is used, resulting in a poor estimate of the model's emissions parameters. Therefore, in order to have sufficient data to make a reliable estimate of the model parameters, multiple observation sequences are required.

A model can be trained with K observation sequences, $\mathbf{O} = \{\mathbf{O}^{(1)}, \mathbf{O}^{(2)}, \ldots, \mathbf{O}^{(K)}\}$ where $\mathbf{O}^{(k)} = \{\mathbf{O}_1^{(k)}, \mathbf{O}_2^{(k)}, \ldots, \mathbf{O}_T^{(k)}\}$, by making the assumption that each observation sequence is independent of every other observation sequence. Therefore, as with one observation sequence, our goal during the training stage is to maximize λ with respect

to $P(\mathbf{O}|\lambda)$ so that with K observation sequences this becomes:

$$P(\mathbf{O}|\lambda) = \prod_{k=1}^{K} P(\mathbf{O}^{(k)}|\lambda) \tag{5.3.40}$$

As the Baum-Welch algorithm is essentially based on counting the frequencies of the occurrence of each event, the re-estimation algorithms can be modified for multiple observations by adding together the individual frequencies of each observation sequence. The modified re-estimation of $\hat{\mathbf{A}}_{ij}$ and $\hat{\mathbf{B}}_{ij}$ are therefore:

$$\hat{\mathbf{A}}_{ij} = \frac{\sum_{k=1}^{K} \frac{1}{P_k} \sum_{t=1}^{T^{(k)}-1} \hat{\boldsymbol{\alpha}}_{ti}^{(k)} \mathbf{A}_{ij} \mathbf{B}_j(\mathbf{O}_{t+1}^{(k)}) \hat{\boldsymbol{\beta}}_{t+1j}^{(k)}}{\sum_{k=1}^{K} \frac{1}{P_k} \sum_{t=1}^{T^{(k)}-1} \hat{\boldsymbol{\alpha}}_{ti}^{(k)} \hat{\boldsymbol{\beta}}_{ti}^{(k)}} \tag{5.3.41}$$

$$\hat{\mathbf{B}}_{ij} = \frac{\sum_{k=1}^{K} \frac{1}{P_k} \sum_{t=1}^{T^{(k)}-1} \delta(\mathbf{O}_t^{(k)}, j) \hat{\boldsymbol{\alpha}}_{ti}^{(k)} \hat{\boldsymbol{\beta}}_{ti}^{(k)}}{\sum_{k=1}^{K} \frac{1}{P_k} \sum_{t=1}^{T^{(k)}-1} \hat{\boldsymbol{\alpha}}_{ti}^{(k)} \hat{\boldsymbol{\beta}}_{ti}^{(k)}} \tag{5.3.42}$$

where P_k is given by $P(\mathbf{O}^{(k)}|\lambda)$. Note that, for each of the k observation sequences, the same scale factors will appear in each term over the summation of t as will appear in P_k and therefore cancel each other out. Because of this, using the scaled values of $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ will result in the unscaled re-estimate of $\hat{\mathbf{A}}_{ij}$. The same holds true for the re-estimation of $\hat{\mathbf{B}}_{ij}$.

5.3.12 Classification using the HMM Algorithm

Hidden Markov Models can be used for the real-time recognition of temporal musical gestures by creating and training one model for each of the G gestures that the user wants the system to recognise. After each of the models have been trained, a new unknown observation sequence $\mathbf{O}$ of length T can be classified using the maximum *a-posterior* probability estimate given by:

$$c = \arg\max_{g} \quad P(\mathbf{O}|\lambda_g) \qquad 1 \leq g \leq G \tag{5.3.43}$$

5.3.13 Calculating the Classification Threshold

To enable a trained HMM to be used for the real-time classification of musical gestures in a continuous stream of data that may contain null gestures, a classification threshold value must be calculated for each of the G models in the database. This way, an

observation sequence will only be classified as the gth gesture if the log-likelihood of the observation sequence $\mathbf{O}$ is greater than or equal to that model's classification threshold:

$$\hat{c} = \begin{cases} c & if(P(\mathbf{O}|\lambda_g) \geq \tau_g) \\ 0 & \text{otherwise} \end{cases} \tag{5.3.44}$$

τ_g, the classification threshold for the gth model, can be calculated after each model has been trained by taking the average log-likelihood prediction value over all the training data for the gth model plus the standard deviation of the log-likelihood prediction value over the training data times a user-configurable sensitivity margin, γ. γ can initially be set to a fixed number of standard deviations (e.g. 2) during the training phase and later adjusted by the user in the real-time prediction phase until a suitable classification/rejection level has been achieved. The classification threshold for the gth model is therefore given by:

$$\tau_g = \mu_g - (\sigma_g \gamma) \tag{5.3.45}$$

where

$$\mu_g = \frac{1}{K_g} \sum_{i=1}^{K_g} P(\mathbf{O}^{(i)}|\lambda_g) \tag{5.3.46}$$

$$\sigma_g = \sqrt{\frac{1}{K_g - 1} \sum_{i=1}^{K_g} \left(P(\mathbf{O}^{(i)}|\lambda_g) - \mu_g \right)^2} \tag{5.3.47}$$

where K_g is the number of training examples for the gth gesture.

Using a classification threshold for each model in a HMM classification system provides the advantage that a null model (like a noise or silence model as used in speech recognition) is not required. This is an advantage for MCI as it greatly decreases the amount of time the performer needs to spend in the training phase of the algorithm, as they do not need to collect the null-gesture training data and the null model does not need to be trained.

5.3.14 Laplace Smoothing

For the classification of musical gestures, one modification of the HMM training process is required, especially when the number of training examples for each gesture is small.

This modification ensures that none of the emission probability coefficients in $\mathbf{B}$ are set to 0. Setting an emission probability to 0 at any stage of the training phase has two unwanted properties. The first unwanted property is that, once a coefficient is set to 0, it cannot change regardless of how many examples of that symbol in that state occur at a later training iteration. The second, and more drastic property of setting an emissions coefficient to 0, is that the model will produce a zero probability result for any observation sequence that actually includes a symbol in a state with an emissions probability of 0.

To alleviate this issue, a form of Laplace Smoothing has been added to the re-estimation process of $\mathbf{B}$ to ensure that none of the emission probability coefficients are set to 0. Laplace Smoothing, also known as add-one smoothing, is a process commonly used in many areas of probability where 1.0 is added to the numerator in each probability estimate, with N added to the denominator to ensure that the estimate is normalised. This results in any value that would previously have had a zero probability estimate now having a very small probability instead. At the end of each iteration of the re-estimation stage, ϑ is added to the re-estimated value of $\mathbf{B}_{ij}$, after which each column of $\mathbf{B}$ is normalised to ensure the stochastic constraints outlined in equation (5.3.25) are maintained. As each re-estimated probability prior to Laplace Smoothing is less than 1.0, sometimes much less than 1.0, the Laplace Smoothing coefficient, ϑ, was set to to $1.0/M$, rather than 1.0 to ensure that any emissions probabilities that are close to 0 do not gain an unrealistic probability estimate.

The applied form of Laplace Smoothing for the re-estimation of $\hat{\mathbf{B}}$ is therefore given by:

$$\hat{\mathbf{B}}_{ij} = \frac{\hat{\mathbf{B}}_{ij} + \vartheta}{\sum_{k=1}^{M} \hat{\mathbf{B}}_{ik} + \vartheta} \qquad 1 \leq i \leq N, 1 \leq j \leq M \tag{5.3.48}$$

5.4 HMM Experiments on Synthetic Data

Several aspects of the classification abilities of the modified HMM algorithm were tested, all of which have an interest for the recognition of musical gestures. This included evaluating the classification abilities of the HMM algorithm with pre-segmented gestures along with testing the algorithm's ability to recognise the gestures in a continuous stream of data that also contained a number of null gestures. Prior to testing the overall classification abilities of an HMM on the multivariate temporal numbers-shapes data set described in section 5.2, a number of features of the algorithm with respect to a set of known synthetic models were first tested. This way, the underlying transition

and emissions probabilities that generate an observation sequence are known and this enables the estimation abilities of the HMM to be evaluated.

5.4.1 Evaluation of an HMMs Estimation Abilities

This experiment tested an HMMs ability to correctly estimate $\mathbf{A}$ and $\mathbf{B}$ with respect to the number of training examples available. 10 random first-order left-right models, each with 10 states and 6 symbols, were created and from these η training examples each with a length of 100 were generated using the `hmmgenerate` function in Matlab. η, the number of training examples, ranged from 10 to 1000 in increments of 10. After the training data for each increment of η had been created, it was loaded by a program that attempted to train 10 Hidden Markov Models using the forward-backward/Baum-Welch training routine with the current number of training examples in a maximum of 100 iterations. When each model had reached the maximum number of iterations its estimated $\mathbf{A}$ and $\mathbf{B}$ matrices were saved to disk. After all the models had been trained with each increment of η, the estimated $\mathbf{A}$ and $\mathbf{B}$ matrices were loaded into Matlab and the average estimated error was calculated. This error measure was calculated by taking the average error over all 10 models between the actual value of $\mathbf{A}$ that generated the observation sequences and the estimated value of $\mathbf{A}$, denoted $\bar{\mathbf{A}}$, along with the actual value of $\mathbf{B}$ that generated the observation sequences and the estimated value of $\mathbf{B}$, denoted $\bar{\mathbf{B}}$, for each increment of η.

The average estimated error for η training examples is therefore given by:

$$\xi_\eta = \frac{1}{K}\sum_{k=1}^{K} aErr_\eta^{(k)} + bErr_\eta^{(k)} \tag{5.4.1}$$

where $K = 10$ and $aErr_\eta^{(k)}$ and $bErr_\eta^{(k)}$ are given by:

$$aErr_\eta^{(k)} = \sum_{i=1}^{N}\sum_{j=1}^{N}(\mathbf{A}_{ij} - \bar{\mathbf{A}}_{ij}^{(\eta)})^2 \tag{5.4.2}$$

$$bErr_\eta^{(k)} = \sum_{i=1}^{N}\sum_{j=1}^{M}(\mathbf{B}_{ij} - \bar{\mathbf{B}}_{ij}^{(\eta)})^2 \tag{5.4.3}$$

5.4.1.1 Results & Discussion

Figure 5.13 shows the average estimated error for each increment of η. After calculating the average estimated error and standard deviation for each value of η, an exponential curve, with the form of $a * x^b$, was fitted to the average estimated error results. The

curve was fitted (with 95% confidence bounds) using the coefficients of $a = 5.29$ and $b = -0.11$ with an SSE of 8.44 and RMSE of 0.3. Figure 5.14 shows the derivative of the fitted line, indicating that the rate of change slows to a minimal change (i.e. has a gradient of $< 1.0e - 4$) at 220 training examples. This would therefore suggest that at least 220 training examples are required to minimise the average estimated error with the least amount of training examples.

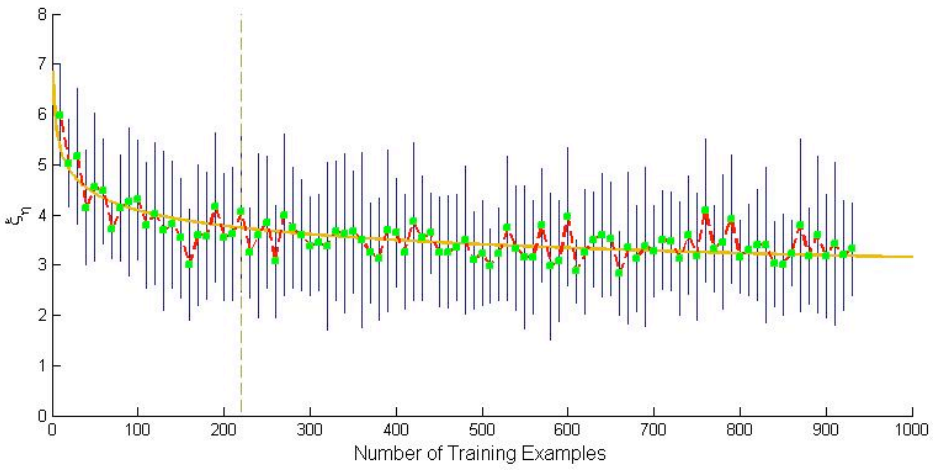


FIGURE 5.13: The average estimated error for each increment of η. The blue line shows the standard deviation over each value of k for that increment of η. The yellow line shows an exponential curve, with the form of $a * x^b$, fitted to the average estimated error for each increment of η. The horizontal brown dashed line shows the minimum change threshold crossing of the fitted line.

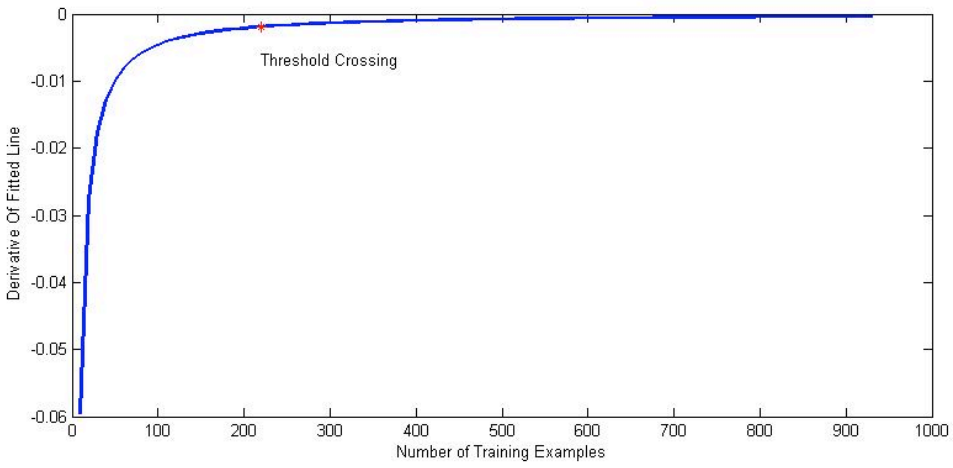


FIGURE 5.14: The derivative of the fitted line and the location of the first minimum change threshold crossing.

5.4.2 Evaluation of an HMMs Classification Abilities

This experiment tested an HMMs classification ability with respect to the number of training iterations the model has been trained with. 10 random first-order left-right models, each with 10 states and 6 symbols, were created and from these 100 training examples each with a length of 100 were generated using the `hmmgenerate` function in Matlab. 100 test examples were also created, with a length of 100, for each of the 10 models. After the training and test data sets had been generated, they were loaded by a program that attempted to train 10 HMMs using the training data. This program used a slightly modified HMM training routine that allowed each model to find the 'semi-optimum' starting estimate of the **A** and **B** matrices by running 1 iteration of the training routine from 20 different random initial values. After the 'semi-optimum' had been located for each model, one training iteration was run on the current estimates of **A** and **B**. The training was then stopped and each of the models were tested using the 1000 test examples (100 test examples for each of the 10 models). The Average Correct Classification Ratio (**ACCR**) was calculated using equation (5.4.4). The model was then trained with one more iteration, starting from the previous estimate of **A** and **B** and re-tested. This training and test cycle was repeated from 1 - 1000 training iterations.

The ACCR was calculated using:

$$\frac{1}{K}\sum_{k=1}^{K}\frac{\text{Number of correctly classified test gestures}}{\text{Total number of tested gestures}} \tag{5.4.4}$$

where K is equal to the number of models tested (i.e. 10).

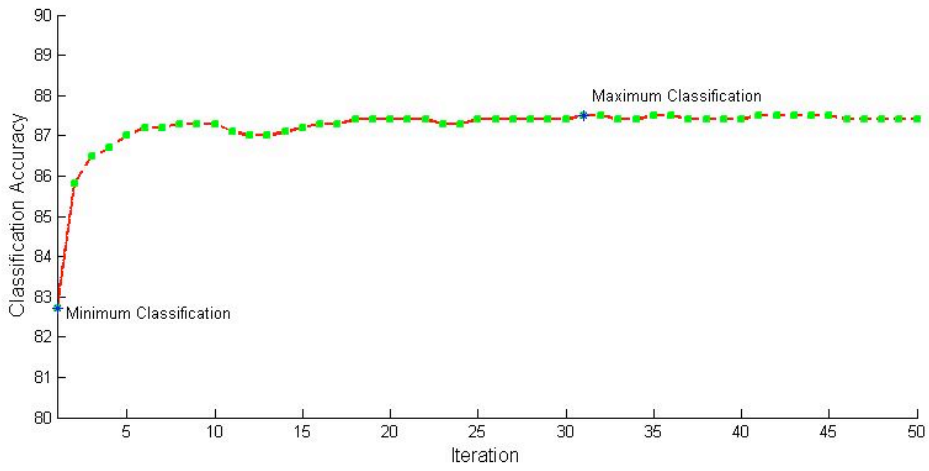


FIGURE 5.15: The classification results for the first 50 training increments.

5.4.2.1 Results

Figure 5.15 shows the ACCR results for the first 50 training increments. The results of this test suggest that the classification abilities of the algorithm are quickly improved in the first few iterations with the maximum classification result of 87.5% being achieved at iteration 31. Figure 5.16, which shows the ACCR results for all 1000 training increments, illustrates that the classification results decrease slightly as the iteration value increases above 100. A possibly reason for this maybe that the estimates of the **A** and **B** matrices start to over-fit with a large number of training iterations.

5.4.2.2 Discussion

The results of this test suggest that the classification abilities of the HMM algorithm improve after a small number of iterations, however the classification abilities of the algorithm do not significantly improve after this point. For the recognition of musical gestures, it is therefore of beneficial to limit the number of training iterations to, for example, 10 training iterations as this will still achieve a high recognition result with the advantage of having the algorithm train quickly.

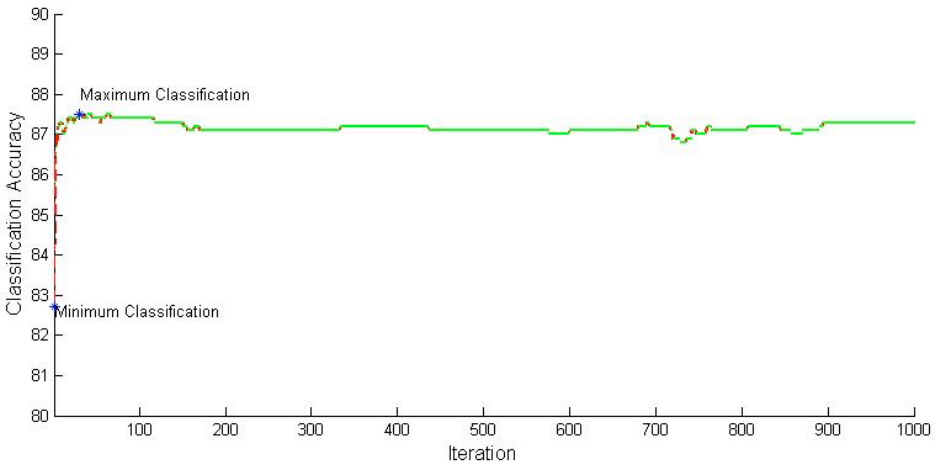


FIGURE 5.16: The classification results for all the training increments from 1 - 1000.

5.5 HMM Experiments on Real Data

Prior to testing the overall classification abilities of a modified Hidden Markov Model with respect to both pre-segmented data and a continuous stream of data, the parameter settings of an HMM were first evaluated. These tests, which used the data from just one participant, evaluated the choice of model that should be used (i.e. ergodic or left-right) along with the optimal number of states and symbols to use for the recognition of the gestures contained in the numbers-shapes data set described in section 5.2. The SAX parameters and γ parameter are also evaluated.

5.5.1 HMM Model Type Evaluation

This experiment tested an HMMs ability to correctly classify the pre-segmented data from the numbers-shapes data set with respect to the type of model design used. The pre-segmented data from the first participant was used to run this test with the following model designs being tested:

- Ergodic Model
- First-Order Left-Right Model
- Second-Order Left-Right Model

For each model design, a 10-state HMM was trained using 10-fold cross-validation. In each fold, the remaining data not used for training was tested against each model. The training and test data was quantised using the SAX algorithm, ($a = 10$, $f = 50$), followed by quantisation using the k-means algorithm with the cluster size set to 16. As the classification abilities of an HMM are based on the model's ability to estimate the transition and emissions probabilities for a given gesture and this estimation process starts from a random position, the test above was repeated 10 times and the overall cross-validation results were averaged. Repeating this test 10 times provides a good indication of how stable the overall classification results of each model design are; as a low standard deviation in the results of each model shows if one model type consistently outperforms the other model designs.

The Average Cross Validation Ratio (**ACVR**) was calculated for this test which gave an indication of the performance of the model, averaged across all of the 10 folds. The ACVR was calculated using:

$$\frac{1}{R}\sum_{r=1}^{R}\frac{1}{K}\sum_{k=1}^{K}\frac{\text{Number of correctly classified test gestures in fold } k}{\text{Total number of tested gestures in fold } k} \tag{5.5.1}$$

where K is equal to the number of cross-validation folds (i.e. 10) and R is equal to the number of test repeats (i.e. 10).

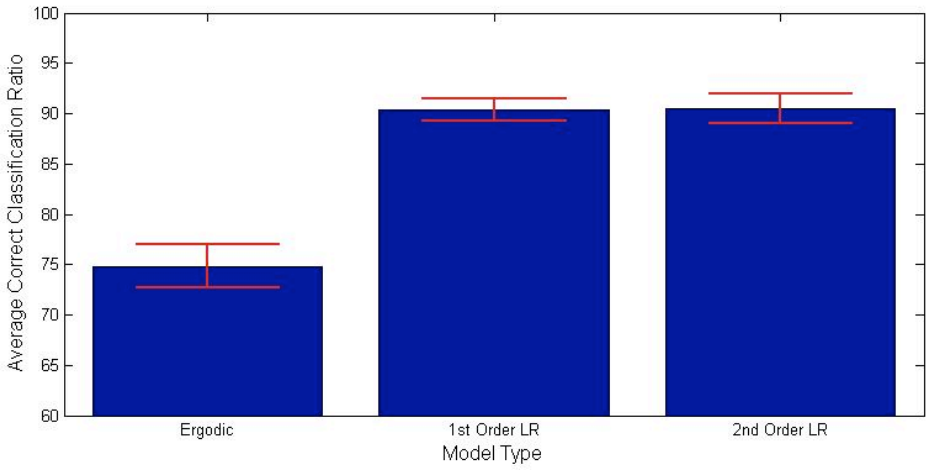


FIGURE 5.17: The ACVR results for each type of model design. The red bars show the standard deviation over each repeat.

5.5.1.1 Results & Discussion

Figure 5.17 shows the ACVR results for each type of model design. The results from this test show that both the first and second order left-right models consistently outperform the ergodic model. The first-order left-right model achieved an ACVR of 91.5% (σ = 1.08%) with the second-order left-right model achieving an ACVR of 90.5% (σ = 1.51%) and the ergodic model achieving an ACVR of 74.8% (σ = 2.15%). These results are consistent with those across the literature (Abou-Moustafa et al., 2004), even though this test has been run on a much smaller data set from one participant. The performance of the left-right models over that of the ergodic model could possibly be explained by the phenomenon of Ocham's razor, where a simple model is favored over a complex one. This is because the ergodic model has many more parameters, as it has all the possible state transitions connected. This type of model therefore requires more data to ensure that the model does not over-fit the training data which results in poor generalization. The results of this test therefore suggest that a first-order left-right model would be the most appropriate to use for the classification of the numbers-shapes data set.

5.5.2 HMM Number of States Evaluation

This experiment tested an HMMs ability to correctly classify the pre-segmented data from the numbers-shapes data set with respect to the number of states used in the

model design. 10 first-order left-right models, each with 16 symbols were trained using 10-fold cross-validation with the first participant's numbers-shapes data. The data was quantised using the SAX algorithm, with an a size of 10 and a f size of 50, followed by quantisation using the k-means algorithm with the cluster size set to 16. Each K-fold cross-validation iteration was trained with N, the number of states, set from 2 through 30, at the end of which the cross-validation result was calculated and saved. As the standard deviation in the ACVR results of the model design test for the first-order left-right was very low ($\sigma = 1.08\%$), the cross-validation value for this test will only be run once, rather than repeating the test 10 times.

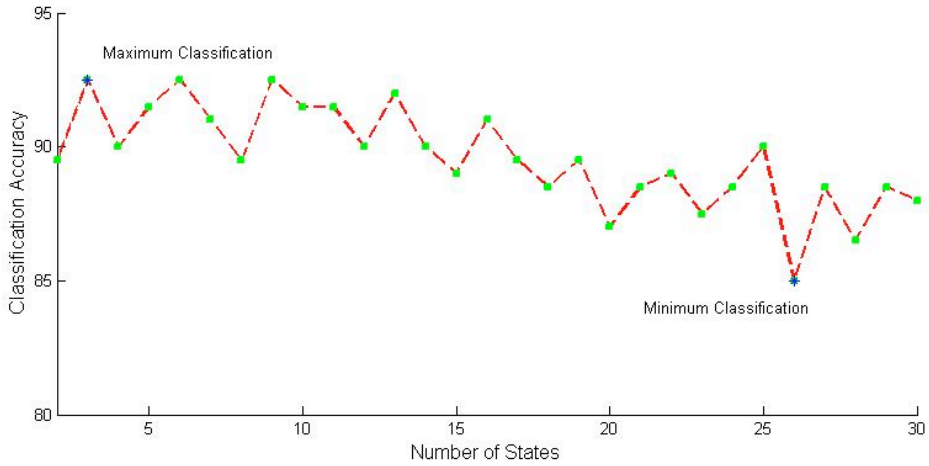


FIGURE 5.18: The ACVR results for each increment of N.

5.5.2.1 Results & Discussion

Figure 5.18 shows the cross-validation results for each increment of N. The maximum cross-validation value of 92.5% occurred with the number of states set to 2, 5 and 9 with the minimum cross-validation value of 85% occurring with the number of states set to 25. The mean cross-validation value over all the iterations of N was 89.62% with a standard deviation of 1.87%. These results suggest that the number of states parameter should be limited to, for example, 10 states as the classification ability of the algorithm decreases beyond this value. The reason for the drop in the cross-validation result maybe due to the increase in model complexity as the number of states increases. The results of this test therefore suggest that the optimal number of states to use for the recognition of the numbers-shapes data set is 2. This value should achieve the maximum classification result, while at the same time using the minimum number of states which reduces the overall complexity of the model. The following tests will therefore use 2 states.

5.5.3 HMM Number of Symbols Evaluation

This experiment tested an HMMs ability to correctly classify the pre-segmented data from the numbers-shapes data set with respect to the number of symbols used in the model design. As the number of symbols must match the number of quantisation values used in the k-means algorithm, this test also evaluates the most appropriate cluster size to use for the recognition of the numbers-shapes gestures. 10 first-order left-right models, each with 2 states were trained using 10-fold cross-validation with the first participant's numbers-shapes data. The data was quantised using the SAX algorithm, ($a = 10$, $f = 50$), followed by quantisation using the k-means algorithm. Each K-fold cross-validation iteration was trained with M, the number of states and also the number of clusters in the k-means algorithm, set in powers of 2 ranging from 2 to 1024. At the end of each fold the cross-validation results were calculated and saved.

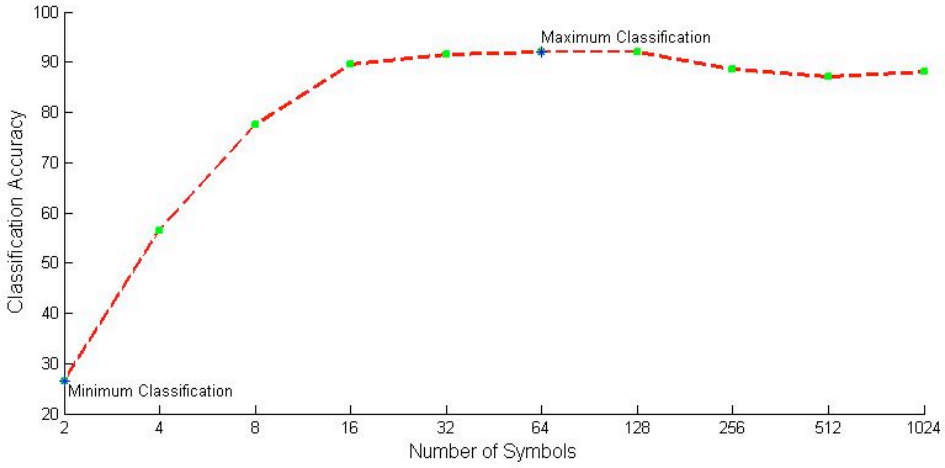


FIGURE 5.19: The cross-validation results for each increment of M.

5.5.3.1 Results & Discussion

Figure 5.19 shows the cross-validation results for each increment of M. This test shows that the number of symbols (and therefore the number of clusters used in the k-means algorithm) does have a significant effect on the classification abilities of the HMM algorithm. Using only 2 symbols, the cross-validation value was only 26.5%, this value significantly improved with each increase in the number of symbols up to a maximum cross-validation value of 92% with 64 symbols. After a symbol size of 128 the cross-validation value started to decrease which may suggest that not enough training data was available and the models started to over-fit with a large symbol size. A satisfactory classification value of 89.5% was achieved with just 16 symbols, just 1.5% less than the

maximum classification value with 64 symbols, which suggests that a symbol size of 16 would achieve a good compromise between a high classification result while at the same time reducing the overall complexity of the model. A symbol size of 16 will therefore be used to test the main numbers-shapes data set.

5.5.4 Evaluation of the SAX Alphabet Size

This experiment tested an HMMs ability to correctly classify the pre-segmented data from the numbers-shapes data set with respect to the SAX alphabet size parameter. 10 first-order left-right models, each with 2 states and 16 symbols were trained using 10-fold cross-validation with the first participant's numbers-shapes data. The data was quantised using the SAX algorithm, with an a size ranging from 2 through 20 in increments of 2 and a f size of 50, followed by quantisation using the k-means algorithm with the cluster size set to 16. At the end of each fold the cross-validation accuracy was calculated and saved.

5.5.4.1 Results & Discussion

Figure 5.20 shows the cross-validation results for each increment of a. The results of this test suggest that a SAX alphabet size of 14 should provide the highest classification results. In this test, an alpha size of 14 achieved a cross-validation accuracy of 93.5%, with the minimum cross-validation result of 70% being achieved with just 2 states. A SAX alphabet size of 14 will therefore be used to test the main numbers-shapes data set.

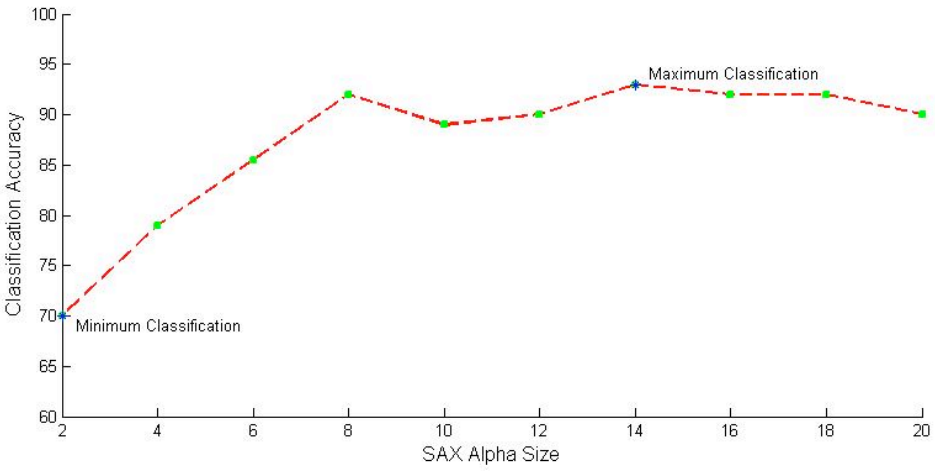


FIGURE 5.20: The cross-validation results for each increment of a.

5.5.5 Evaluation of the SAX Frame Size

This experiment tested an HMMs ability to correctly classify the pre-segmented data from the numbers-shapes data set with respect to the SAX frame size parameter. 10 first-order left-right models, each with 2 states and 16 symbols were trained using 10-fold cross-validation with the first participant's numbers-shapes data. The data was quantised using the SAX algorithm, with an a size of 14 and a f size ranging from 10 to 100 in increments of 10, followed by quantisation using the k-means algorithm with the cluster size set to 16. At the end of each fold the cross-validation accuracy was calculated and saved.

5.5.5.1 Results & Discussion

Figure 5.21 shows the cross-validation results for each increment of f. The results of this test suggest that the size of the SAX frame size does not have a major impact on the overall classification abilities of the algorithm. A classification result of $> 89\%$ was achieved for all of the frame sizes tested, with the minimum cross-validation result of 89.1% occurring with a f size of 90 and the maximum cross-validation result of 91.39% occurring with a f size of 100. Figure 5.21 shows the cross-validation results for each increment of f from 10 - 100. A SAX frame size of 50 will therefore be used to test the main numbers-shapes data set, as this provided a compromise between a good result of 92.09% (just 1.19% below the maximum classification result) and a large enough frame size which helps reduce the overall computation time of the algorithm.

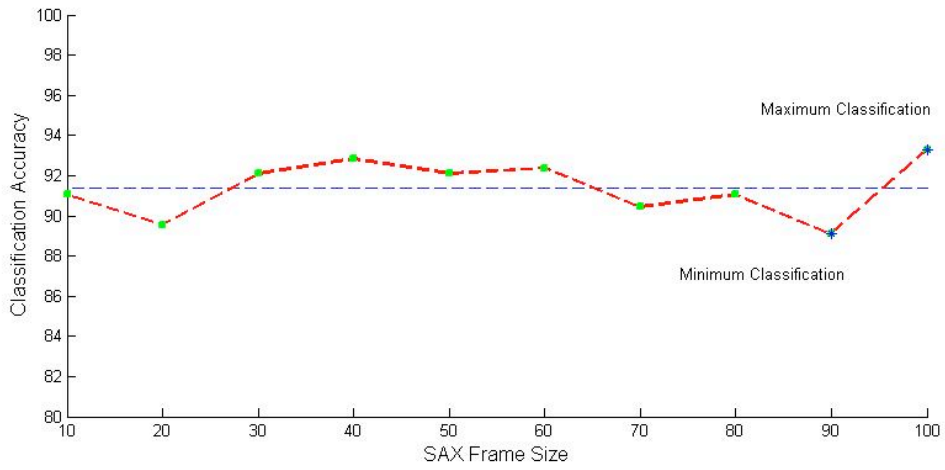


FIGURE 5.21: The cross-validation results for each increment of f.

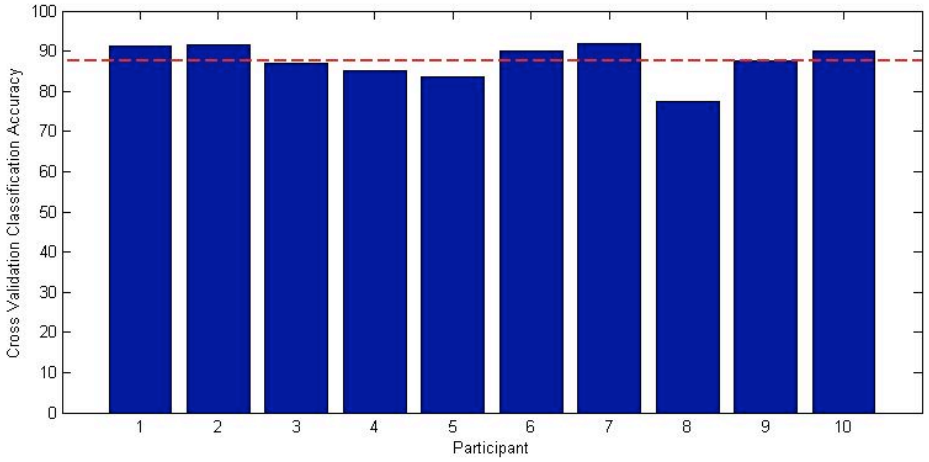


FIGURE 5.22: The cross-validation classification results for each of the 10 participants. The ACVR is shown by the red horizontal dashed line.

5.5.6 Evaluation of an HMMs Classification Abilities With Pre-Segmented Data

This experiment tested an HMMs ability to correctly classify the pre-segmented data from the numbers-shapes data set. For each of the 10 participants, 10 left-right first-order models, with 2 states and 16 symbols, were trained using 10-fold cross-validation. In each fold, the remaining data not used for training was tested against each model. A correct classification result was considered if the correct class label was predicted by the HMM (i.e. if the kth model gave the maximum log-likelihood value and this value was greater than the kth model's classification threshold). γ, the coefficient that controls the number of standard deviations to use for the classification threshold for each gesture, was set to 2.0. SAX and k-means clustering were used for quantising the data with each segmented gesture being split into 50 frames with the SAX alpha size set at 14 and 16 clusters were used for the k-means algorithm.

5.5.6.1 Results

The average cross-validation classification result across all 10 participants was 87.53% with a standard deviation of 4.51%. Participant 7 achieved the maximum cross-validation classification result with 91.74% and participant 8 achieved the minimum cross-validation classification result with 77.43%. Figure 5.22 shows the cross-validation results for each of the 10 participants. Table 5.1 shows the confusion matrix for each of the 10 gestures (summed over each round of cross-validation).

Gesture	G 0	G 1	G 2	G 3	G 4	G 5	G 6	G 7	G 8	G 9	G 10
G 0	0	0	0	0	0	0	0	0	0	0	0
G 1	42	599	8	6	0	3	9	0	19	0	14
G 2	47	2	605	20	12	2	9	0	1	0	2
G 3	63	0	25	540	0	4	16	2	0	0	0
G 4	33	0	2	0	661	2	0	0	1	1	0
G 5	51	0	0	10	0	627	2	0	0	0	0
G 6	82	6	17	6	0	0	572	17	0	0	0
G 7	63	7	1	4	0	2	26	574	2	21	0
G 8	72	6	1	4	0	0	0	3	593	11	0
G 9	30	1	1	0	0	0	1	8	11	628	0
G 10	40	9	2	1	0	1	1	0	0	0	646

TABLE 5.1: The confusion matrix across all 10 participants for each of the 10 gestures. G 0 represents the null gesture.

5.5.6.2 Discussion

This test showed that the HMM provided a moderate classification result across the 10 participants, with the algorithm achieving an average classification result of over 80% for 9 of the 10 participants, 6 of which achieved over 90%. Table 5.1 shows the confusion matrix, summed over all 10 participants for the 10 gestures. The confusion matrix shows that the majority of classification errors made (523 out of 865 = 60%) occurred when a gesture was classified as a null gesture (with ID 0). The miss-classification of a gesture as another gesture, rather than a null gesture, only made up a small percentage of the total classification errors. It should be noted that, for the majority of cases, simply lowering the classification threshold for each gesture would not have corrected those examples that were misclassified as a null gesture. This is because a large majority of the incorrectly classified gestures had an extremely low log-likelihood prediction value and therefore lowering the classification threshold would not have corrected these classification errors. A possible explanation for these extremely low prediction values is that the HMM had over-fitted and simply required a larger training data set to create a more robust model.

5.5.7 Evaluation of a HMMs Classification Abilities With Continuous Data

This experiment tested an HMMs ability to correctly classify data from the numbers-shapes data set in a continuous stream of data that also contains a number of null gestures. This evaluates two important aspects of a Hidden Markov Model for the recognition of multivariate temporal gestures. Namely the algorithm's ability to correctly classify a set of temporal gestures from a continuous stream of data and also

the algorithm's ability to reject any null gesture that is not contained in the model's database.

For each of the 10 participants, 10 left-right second-order models were trained using 10 randomly selected training examples from each of the 10 gestures. SAX and k-means clustering were used for quantising the data with each segmented gesture being split into 50 frames with the SAX alpha size set at 10 and 16 clusters were used for the k-means algorithm. After the models had been trained they were tested using a continuous stream of data. The continuous stream of data originated from the data collection phase of the numbers-gestures database and contained all of the data from each participant's recording. The continuous stream, therefore, contained not only each trial the 10 gestures the participant performed but also, importantly, the participant's movements in between each trial along with periods of rest.

The continuous stream was tested by running a sliding window of size w over the data stream in increments of 10. For each participant, w was set as the average window size of the 10 randomly selected training examples from each of the 10 gestures. The mean average window size across all 10 participants was 310 with the minimum average window size of 251 and the maximum average window size of 389. At each increment, the data within the sliding window was quantised using SAX and k-means quantisation with the same settings as used in the training phase. The gesture zone tag within the data was used to evaluate if the HMM had made the correct classification for each window of data. As some windows may cover a section of data that contains half a gesture and noise, the maximum class ID value contained in the gesture zone tag data was used as the gesture ID for that window of data. A correct classification result was considered if the correct class label was predicted by the HMM (i.e. if the kth model gave the maximum log-likelihood value and this value was greater than the kth model's classification threshold). The classification threshold for the kth model was set to average log-likelihood predication value over all the training examples for the kth class plus 2 standard deviations. The following classification errors were evaluated for this experiment: ACCR, APR, ARR and ANRR.

5.5.7.1 Results

The ACCR for all 10 participants was 44.80% with a standard deviation of 6.95%. Figure 5.23 shows the ACCR results for all 10 participants. Participant 10 achieved the maximum ACCR value of 53.58% with participant 9 achieving the minimum ACCR value of 30.89%. Table 5.2 shows the APR and ARR values for each of the 10 gestures, averaged over the 10 participants. The APR and ARR results show that the majority of

	G1	G2	G3	G4	G5	G6	G7	G8	G9	G10
APR	0.67	0.42	0.52	0.71	0.67	0.58	0.68	0.74	0.62	0.79
ARR	0.32	0.24	0.32	0.44	0.26	0.18	0.60	0.37	0.40	0.46

TABLE 5.2: The average precision and average recall for each gesture.

classification errors occurred, not from the misclassification of one gesture with another, but with the misclassification of a gesture as a null gesture. The ANRR value of 0.65 shows that the HMM model was able to correctly classify a null gesture 65% of the total number of analysis windows.

5.5.7.2 Discussion

These results show that, although the HMM can correctly classify the same gestures with the same settings when the gestures are pre-segmented, it can not classify the gestures with the same accuracy from a continuous stream of data. This presents a problem for the real-time recognition of musical gestures from a continuous stream of data, for example, during a live performance, as the performer could not rely on the system to automatically classify a gesture with any great accuracy. Instead, the performer would need to manually instruct the computer that they have just performed a gesture by, for example, pressing a foot-switch at the start and end of the gesture. The system could then take this segment of data and classify it against one of the gestures in its trained database.

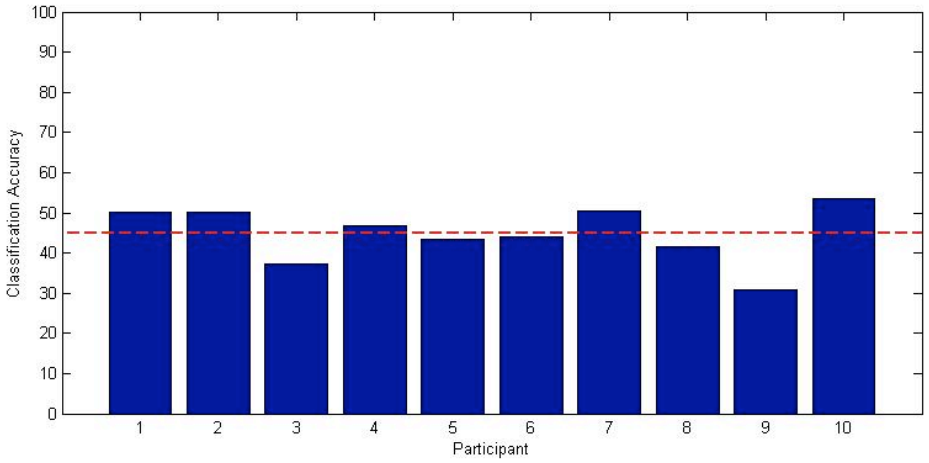


FIGURE 5.23: The classification accuracy for each participant. The red line indicates the overall ACCR.

5.5.8 HMM Summary

Hidden Markov Models have been used frequently across a wide range of fields to solve the problem of classifying temporally variable signals. In this chapter the following modifications have been made to the standard HMM algorithm to optimize it for the recognition of musical gestures with a limited number of training examples:

1. Applied Symbolic Aggregate Approximation along with k-means clustering as a quantisation method for high dimensional data
2. Added a multi-instance training routine to start the full training routine with a 'semi-optimal' local maxima
3. Used the reciprocal of M as a form of Laplace Smoothing to stop any emissions probabilities values being set to 0
4. Added a user-configurable threshold classification value for each gesture, thus alleviating the need for training a null gesture model
5. Adopted a multi-threaded training approach for each gesture to minimise the total training time of the algorithm

The experiments conducted on both synthetic and real data in sections 5.4 - 5.5.7 have shown that HMMs are not particularly suitable for the real-time recognition of musical gestures as:

1. They need a large amount of training data for each gesture in order to build a robust model
2. Even when a multi-threaded training routine is adopted, they take a long time to train. This is because a vector quantisation algorithm also needs to be trained along with the HMM model itself
3. They perform poorly in classifying a number of gestures from a real-time continuous stream of data

5.6 Summary

This chapter has addressed the recognition of multivariate temporal musical gestures. The challenges of recognising a temporal gesture, as opposed to a static posture, were first outlined. This was followed by a description of the numbers-shapes data set. The Hidden Markov Model algorithm was then presented along with a description of how the standard HMM algorithm has been updated for the recognition of multivariate temporal musical gestures. The chapter was concluded with a number of experiments designed to evaluate the classification abilities of the HMM algorithm, showing that HMMs are not particularly suitable for the real-time recognition of musical gestures (as they take a long time to train and perform poorly in classifying a number of multivariate temporal gestures from a real-time continuous stream of data). This presents a major problem for a musical gesture recognition system because a performer would not only need to wait a long time while their system trains itself, but the performer would also be constrained to use a trigger key or alternative approach to inform the algorithm that a gesture had just been made. In the next chapter an alternative powerful machine learning algorithm, called Support Vector Machines (SVM), is investigated and the algorithm is evaluated to determine if it can be applied for the recognition of multivariate temporal gestures. The results of the forthcoming analysis show how the SVM algorithm outperforms the HMM algorithm in terms of both training speed and classification accuracy.

Chapter 6

Support Vector Machines

Imagination will often carry us to worlds that never were.
But without it we go nowhere.

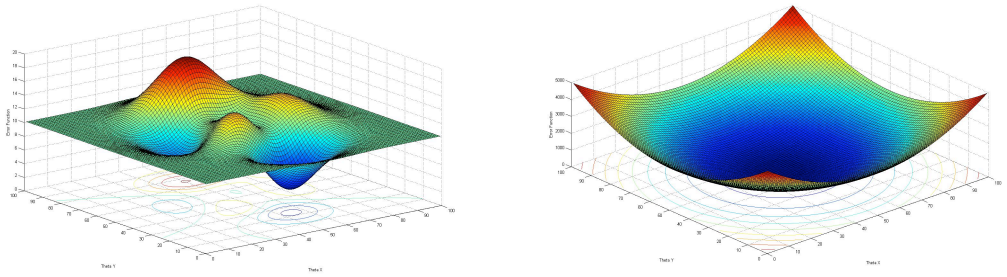
Carl Sagan

The previous chapter described how a Hidden Markov Model could be applied to classify multivariate temporal gestures. Although the HMM algorithm achieved a moderate recognition rate of 87% on the pre-segmented data from the numbers-shapes data set, it was deemed unsuitable for the classification of musical gestures because it failed to recognise the same gestures from a continuous stream of data and also took a long time to train. This chapter investigates how a powerful machine learning algorithm called Support Vector Machines (**SVM**) can be applied for the recognition of multivariate temporal gestures. This chapter shows how a number of feature extraction algorithms have been specifically applied to represent multivariate temporal gestures and how the SVM has been adapted to classify gestures from a continuous stream of data. The chapter is concluded with a number of experiments designed to evaluate the multivariate temporal classification abilities of the SVM using the numbers-shapes data set.

6.1 Support Vector Machines

Support Vector Machines (**SVM**) are a set of related supervised learning methods used for classification and regression (the regression version of the SVM is known as *Support Vector Regression*). SVM were originally formulated for two-class classification problems, and have quickly been accepted as a powerful tool for developing pattern classification and function approximation systems. One of the key properties of the

SVM is that the determination of the model parameters corresponds to a convex optimisation problem, and so, unlike an Artificial Neural Network, any local solution is also a global optimum (Bishop, 2006). This chapter describes how the SVM algorithm has been modified to optimize it for the recognition of musical gestures, which involves adding a thresholding function to reject null gestures. Prior to describing this, a general overview of the algorithm will first be presented, the interested reader can find a detailed description of the algorithm in Abe (2005), Bishop (2006) and Burges (1998).



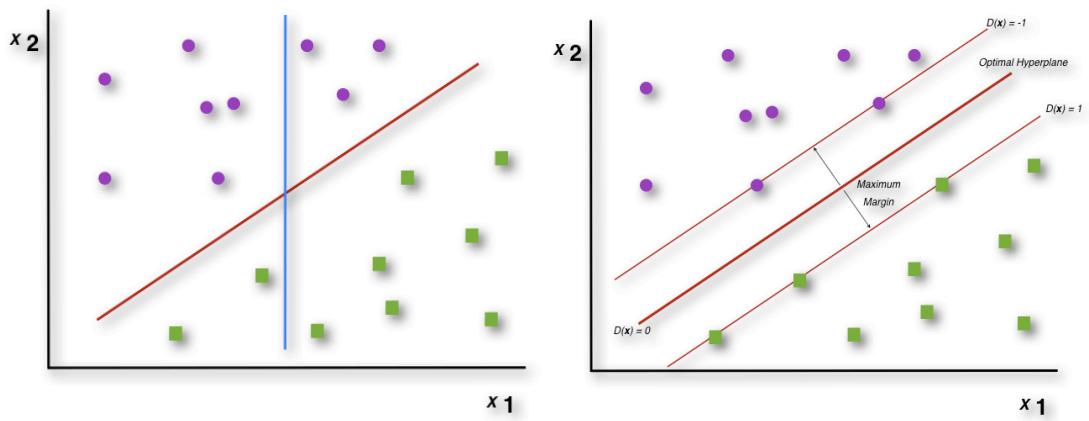
(a) Non-convex Optimisation Problem (b) Convex Optimisation Problem

FIGURE 6.1: SVMs are a Convex Optimisation Problem and therefore any local minima is also a global minima.

6.1.1 SVM

The core concept of the Support Vector Machine is based on finding a line through the feature space x that best separates the two classes of data (for now a simple two-class problem will be considered, however a number of approaches exist to extend the SVM to G classes). This is illustrated by Figure 6.2, which shows a two-class, linearly separable, classification problem. Clearly there could be multiple solutions for the location of this separating line, or *hyperplane*, with the most intuitive solution being to place the separating hyperplane so that the distance between it and any of the data points is maximized. Assuming that the supporting hyperplanes $D(\mathbf{x}) = -1$ and $D(\mathbf{x}) = 1$ include at least one training datum, the hyperplane $D(\mathbf{x}) = 0$ has the maximum margin (as indicated in Figure 6.2(b)). The region $\{-1 \leq D(\mathbf{x}) \leq 1\}$ is then the generalisation region for the decision function. The generalisation ability of a hyperplane therefore depends on the location of the hyperplane, and the hyperplane with the maximum margin is called the *optimal separating hyperplane*.

By defining the optimal separating hyperplane between these two data clusters, an unknown datum can be classified by projecting it into the feature space x and calculating which side of the hyperplane it falls on. The decision function, if the data is linearly separable, is therefore:



(a) Blue hyperplane = poor solution, red hyperplane = optimal solution

(b) Optimal separating hyperplane and the maximum margin

FIGURE 6.2: Two-class linearly separable classification example.

$$D(\mathbf{x}) = \mathbf{w}^\intercal \mathbf{x} + b \tag{6.1.1}$$

where $\mathbf{w}$ is a N-dimensional vector and b is a bias term. The unknown datum $\mathbf{x}$ can then be classified as Class 1 or 2 using:

$$\mathbf{w}^\intercal \mathbf{x} + b \begin{cases} \geq 1, & \text{for} \quad y_i = 1, \\ \leq -1, & \text{for} \quad y_i = -1. \end{cases} \tag{6.1.2}$$

Here, 1 and -1 on the right-hand sides of the inequalities can be a constant $a(> 0)$ and $-a$ respectively. The original inequalities in (6.1.2) can be maintained simply by dividing by a. One key concept of locating the optimal hyperplane is that, once it has been found, only the training examples that lie on the hyperplanes $D(\mathbf{x}) = -1$ and $D(\mathbf{x}) = 1$ are required. These training examples are known as the *support vectors*.

Using the support vectors the decision function is given by:

$$D(\mathbf{x}) = \sum_{i \in S} \alpha_i \, y_i \, \mathbf{w}_i^\intercal \mathbf{x} + b, \tag{6.1.3}$$

where S is the set of support vector indices, $\mathbf{w}_i^\intercal$ is the ith support vector, α_i are the nonnegative Lagrange multipliers that help transform the constrained problem into an unconstrained problem and b is given by:

$$b = \frac{1}{|S|} \sum_{i \in S} (y_i - \mathbf{w}^\intercal \mathbf{x}_i). \tag{6.1.4}$$

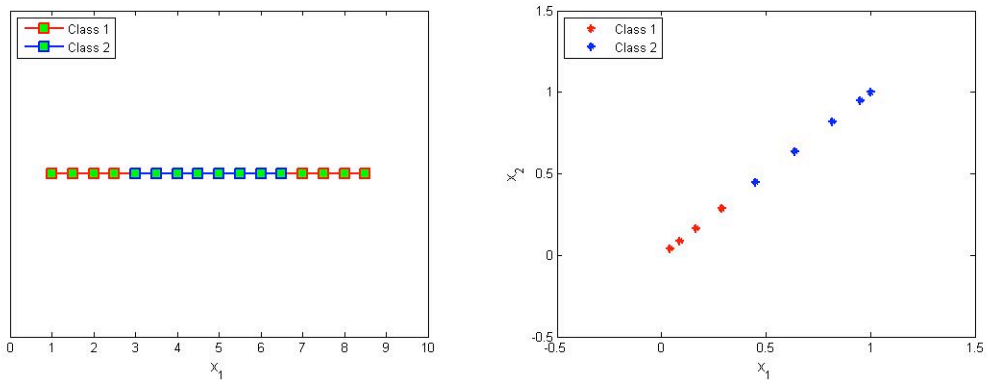
Using equation (6.1.3) an unknown datum $\mathbf{x}$ is classified into:

$$\begin{cases} \text{Class 1}, & \text{if} \quad D(\mathbf{x}) > 0, \\ \text{Class 2}, & \text{if} \quad D(\mathbf{x}) < 0. \end{cases} \tag{6.1.5}$$

If $D(\mathbf{x}) = 0$, $\mathbf{x}$ is on the boundary and is therefore unclassifiable. It is using this simplified example that a support vector machine functions. The interested reader can find a detailed description of how the SVM algorithm determines the optimal separating hyperplane in Abe (2005).

6.1.2 Mapping to a High-Dimensional Space

In a support vector machine the optimal hyperplane is determined to maximize the generalization ability. However, if the training data are not linearly separable, the obtained classifier may not have high generalization ability, even if the hyperplanes are determined optimally. Thus to enhance linear separability, the original input space is mapped into a high-dimensional dot-product space called the *feature space*. This mapping is performed using a *kernel function*. It is this mapping of a non-linear problem in the original data space to a linearly separable problem in a higher dimensional feature space that gives the SVM such power as a classification algorithm for complex real-world data sets. One of the key advantages of using a kernel function for mapping the data to a high-dimensional (and sometimes even infinite-dimensional) feature space is that the feature space does not need to be treated explicitly. This technique is known as the *kernel trick* and it enables the algorithm to use any kernel that satisfies *Mercer's condition* as a legitimate mapping function.



(a) A 1-dimensional linearly inseparable classification problem

(b) Mapping the data into a new, 2-dimensional feature space to make the data linearly separable. This was achieved using a RBF kernel

FIGURE 6.3: Mapping the original input space into a higher-dimensional feature space.

Examples of commonly used SVM kernel functions are:

- **Linear Kernel:** If a classification problem is linearly separable in the input space then it does not need to be mapped into a high-dimensional space. In such a case the linear kernel is used:
$$H(\mathbf{x}, \mathbf{x}') = \mathbf{x}^\intercal \mathbf{x}'. \tag{6.1.6}$$

- **Polynomial Kernel:** The polynomial kernel with degree d, where d is a natural number is given by:
$$H(\mathbf{x},\ \mathbf{x}') = (\mathbf{x}^\intercal \mathbf{x}' + 1)^d. \tag{6.1.7}$$

- **Radial Basis Function (RBF) Kernel:** The RBF kernel is given by:
$$H(\mathbf{x}, \mathbf{x}') = \exp(-\gamma ||\mathbf{x} - \mathbf{x}'||^2), \tag{6.1.8}$$
where γ is a positive parameter for controlling the radius.

6.1.3 Using Probabilistic Outputs For A Classification Threshold

One disadvantage with the common SVM is that the output from the algorithm is a discrete prediction value representing the label of the most likely class. The problem with this version of the algorithm is that there is no way of knowing *how likely* this label is. Thankfully, Platt (1999) presented an extension of the SVM algorithm that enables the output of a calibrated posterior probability value along with the label of the most likely class. Platt's algorithm approximates the posterior probability of class k occurring given $\mathbf{x}$ using the sigmoid function:

$$P(k = 1|\mathbf{x}) \approx P_{A,B}(f) \equiv \frac{1}{1 + \exp(Af + B)} \tag{6.1.9}$$

where $f = f(x)$ and A and B are two parameters that control the sigmoid function with A controlling the curvature of the sigmoid function and B controlling where the center of the function sits in relation to the x axis. Platt proposed an initial algorithm to determine A and B which was later improved by Lin et al. (2007). One of the key factors of this algorithm is that the SVM is trained as normal, with the parameter values required for the posterior probabilities being calculated on the sparse SVM model after it has been trained. This therefore ensures that a sparse model is maintained.

The posterior probability estimate, given by the sigmoid function, can therefore be used as a method to enable the SVM algorithm to reject a null gesture. As a sigmoid function is used, the threshold value required to reject a null gesture can simply be set to 0.5. If

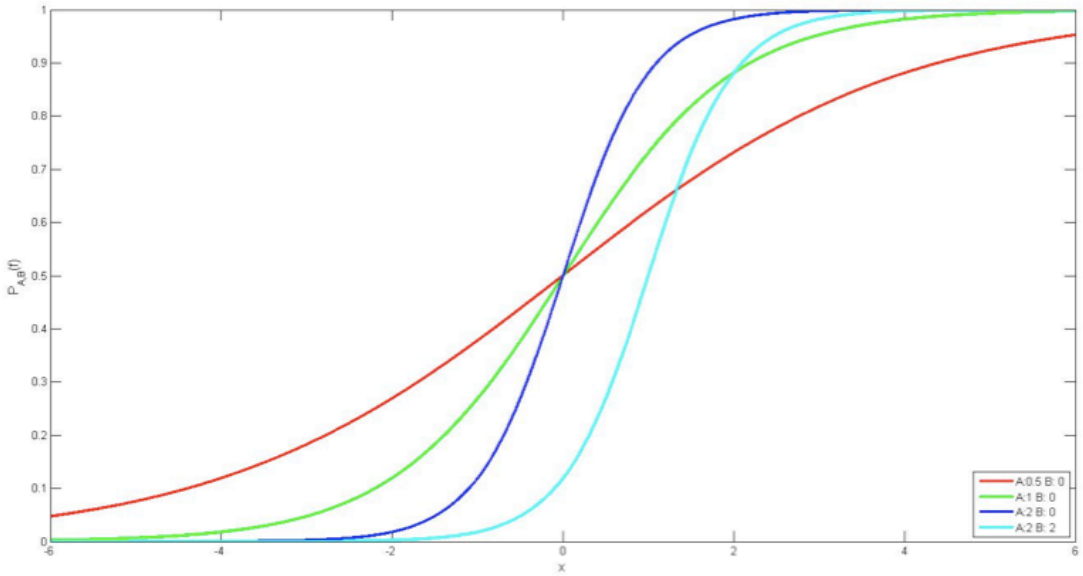


FIGURE 6.4: An illustration of the sigmoid function with three values for A set at 0.5, 1, 2 and 2 and the values for B set as 0, 0, 0 and 2.

the performer finds that some gestures are still being rejected then they can manually adjust this threshold value until a satisfactory compromise between the rejection of gestures and classification of false positives has been reached. Figure 6.5 illustrates the processing chain for the SVM algorithm.

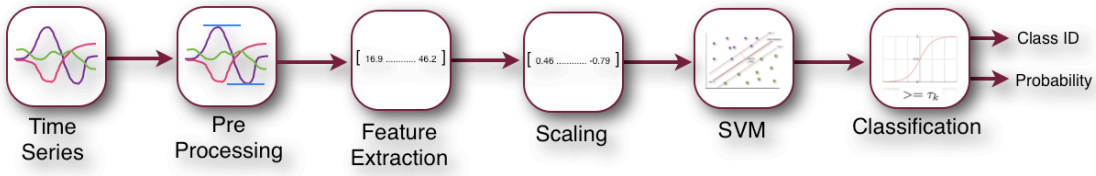


FIGURE 6.5: The processing chain for the SVM algorithm.

6.1.4 Using SVM to Classify Multivariate Temporal Data

Support Vector Machines, with their non-linear kernels and convex optimisation problems, present a formidable classification tool. SVM have one disadvantage however, in that N, the length of their input vector $\mathbf{x}$, needs to be a pre-determined fixed length. This means that the raw-data cannot be used as input to the SVM, as any temporal variation in the data will corrupt the correct classification of the input signal. It is therefore necessary to compute a number of features that best represent the multivariate temporal signal and use these features to create the SVM's input vector.

Signal A			Signal B		
μ	σ	$\|\|\mathbf{x}\|\|$	μ	σ	$\|\|\mathbf{x}\|\|$
-0.569	0.999	31.603	-0.569	0.999	31.603

TABLE 6.1: Mean, Standard Deviation and Euclidean Norm of Signal A and B.

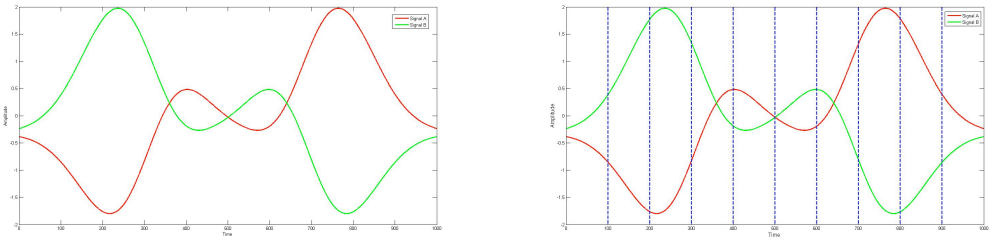
It is in choosing which features of the data to extract that 'makes-or-breaks' a machine learning algorithm; as features that do not describe the underlying problem well will result in poorly trained classification models. For temporal signals, features that express how the data is changing over the length of the captured time window should be used. To avoid the curse of dimensionality, feature extraction for a N-dimensional temporal signal was resolved by individually extracting the features for each dimension of data and then concatenating the features from each dimension into one single main feature vector. This feature vector was then used as input to the SVM algorithm for training and classification.

The classification abilities of the modified SVM algorithm were evaluated using the numbers-shapes data set with both segmented data and data in a continuous stream. Prior to presenting the results of these evaluations, the feature extraction methods used as input to the SVM algorithm will first be described.

6.1.5 Time Domain Features

A number of standard statistical features were calculated for each dimension of the multivariate time-series $\mathbf{x}$. These consisted of the mean (μ), standard deviation (σ), Euclidean norm ($||\mathbf{x}||$) and root mean squared (RMS) value of $\mathbf{x}$. These are useful statistical features, however, they all have one common drawback in that they do not describe well how a temporal signal is changing over time. Figure 6.6 illustrates this, showing two very different signals, signal b is the reverse of signal a. Despite the fact that these signals are clearly different, they still have the same mean, standard deviation and Euclidean norm, as indicated by table 6.1.

This problem was solved by segmenting the temporal signal into an equal number of frames and computing the mean, standard deviation, Euclidean norm and RMS value of each frame. The number of frames is given by N_f and the number of samples in each frame is given by $|\mathbf{x}|/N_f$. Increasing N_f will result in a more precise analysis, however, it will also increase the length of the input feature vector to the machine learning model and may therefore increase the complexity of the learning task. For the classification of the number-shapes data set, N_f was empirically set to 10.



(a) Two different temporal signals

(b) The time series segmented into 10 frames

FIGURE 6.6: Two different temporal signals with the same mean, standard deviation and Euclidean norm.

	Signal A			Signal B		
Frame	μ	σ	$\|\|\mathbf{x}\|\|$	μ	σ	$\|\|\mathbf{x}\|\|$
1	-0.558	0.136	5.747	-0.005	0.180	1.801
2	-1.340	0.280	13.694	1.084	0.419	11.619
3	-1.515	0.298	15.442	1.805	0.177	18.145
4	0.008	0.405	4.032	0.450	0.463	6.447
5	0.269	0.165	3.161	-0.198	0.068	2.102
6	-0.198	0.068	2.102	0.269	0.165	3.161
7	0.450	0.463	6.4477	0.008	0.405	4.032
8	1.805	0.177	18.145	-1.515	0.298	15.442
9	1.084	0.419	11.619	-1.340	0.280	13.694
10	-0.005	0.180	1.801	-0.558	0.136	5.747

TABLE 6.2: The mean, standard deviation and Euclidean norm of Signal A and B (as shown in Figure 6.6), segmented into 10 frames.

6.1.6 Frequency Domain Features

Appropriate temporal features can also be extracted from the frequency domain along with the time domain features presented previously. By using a transform function, such as the Fourier transform or the Discrete Wavelet transform, a time series can be converted from the time domain to its representation in the frequency domain. After a time series has been converted to the frequency domain, a number of features can be found such as:

The bin index of the signal's maximum frequency:

$$\Upsilon_{\mathrm{m}} = \arg\max_{i} \quad \theta_i \tag{6.1.10}$$

The amplitude of the maximum frequency:

$$\Upsilon_{\mathrm{ma}} = \max \quad \theta_i \tag{6.1.11}$$

The ratio between the maximum frequencies amplitude and the mean of the spectrum:

$$\Upsilon_{\text{mr}} = \frac{Pma}{\sum_{i=1}^{n} \theta_i} \tag{6.1.12}$$

The top c Fourier coefficients:

$$\Upsilon_{\textbf{coeff}_i} = \theta_i \qquad \text{for} \quad 1 \leq i \leq c \tag{6.1.13}$$

The phase value at the maximum frequency:

$$\Upsilon_{\omega} = \varphi_{P_{max}} \tag{6.1.14}$$

The Euclidean norm of the phase:

$$\Upsilon_{||\omega||} = \sqrt{\sum_{i=1}^{n} \varphi_i^2} \tag{6.1.15}$$

where θ and φ are the magnitude and phase vectors output from the FFT and n is the FFT's window size. These frequency domain features are commonly used in various domains throughout the machine learning literature, such as in speech recognition (Rabiner, 1989) and the recognition of hand gestures captured by an EMG sensor (Kim et al., 2008).

6.2 SVM Experiments

Three experiments were run to validate the classification abilities of the modified SVM algorithm. The first experiment evaluated the classification abilities of the SVM algorithm with the pre-segmented gestures from the numbers-shapes data set (see chapter 5.2). The second experiment evaluated the classification abilities of the SVM algorithm with respect to a minimal amount of training data. Finally, the third experiment tested the SVM algorithm's ability to correctly classify data from the numbers-shapes data set in a continuous stream of data that also contained a number of null gestures. The LIB-SVM (Chang and Lin, 2001) library was used to implement the SVM classifier for all three experiments. A RBF kernel function was used to map the input data into a linearly separable feature space for all three experiments.

6.2.1 SVM Experiment A

This experiment tested the SVM algorithm's ability to correctly classify the pre-segmented data from the numbers-shapes data set. For each participant, a SVM model was trained using 10-fold cross-validation. In each fold, the remaining data not used for training was tested against the model, with the average cross-validation ratio being computed at the end of each test for all 10 participants.

This test was run using three sets of feature data:

1. Time Domain Features (**TD**)
2. Frequency Domain Features (**FD**)
3. Combined Time Domain and Frequency Domain Features (**TDFD**)

The TD features consisted of the mean, standard deviation, Euclidean norm and RMS value for each dimension of data. Each dimension of data was segmented into 10 frames, giving the total number of TD features per training/testing example to 120 (4 features x 10 frames x 3 dimensions). The FD features consisted of the six features given by equations (6.1.10) through (6.1.15) for each dimension of data, with c, the top Fourier coefficients, set at 10 and the FFT window size set at 512. Giving the total number of FD features to 45 (15 features x 3 dimensions). The combined TDFD feature vector therefore consisted of 165 features per training/testing example (120 TD and 45 FD).

6.2.1.1 Results

Table 6.3 shows the ACVR results across all 10 participants for the three conditions, with each condition being run with scaling off and scaling on. With scaling on, all three conditions achieved excellent classification results, with the TD and TDFD conditions both achieving an ACVR value of 99.28%. The FD condition achieved an ACVR value of 97.8%. The results were significantly reduced without scaling the data prior to both training and testing the SVM, with the TD achieving the best ACVR value of 50.34% and the FD and TDFD both achieving the poorest ACVR value of just 24.44%. Figure 6.7 shows the cross-validation results for each particpant for the TD condition with scaling on. For this condition the SVM algorithm achieved an excellent cross-validation result of 98% or higher for all the participants, with the exception of participant 8 for which the algorithm achieved a cross-validation result of 96.4%. The SVM algorithm achieved a 100% recognition rate for participants 1, 5, 6 and 10.

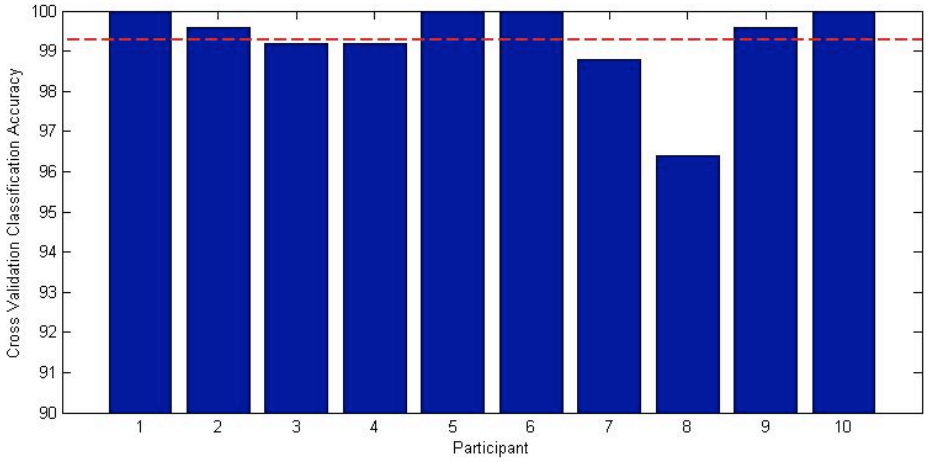


FIGURE 6.7: The cross-validation results for each of the participants (blue) and the overall mean ACVR (dashed red line).

6.2.1.2 Discussion

These results show that the SVM achieves excellent classification results of multivariate temporal data with both features computed in the time domain and features computed in the frequency domain. The TD features slightly outperformed the FD features, achieving very close to a perfect classification result. It is interesting to see that the classification results did not improve when the TD and FD features were combined, suggesting that the FD features are not orthogonal to the TD features. These results suggest that the segmented statistical time domain features are an excellent feature extraction method for multivariate temporal recognition. The time domain features not only provide a high classification result, but are also less computationally demanding to calculate than the frequency domain features. The TD features will therefore be used in the next two experiments which evaluate the classification abilities of the SVM algorithm with respect to a minimal amount of training data and the algorithm's ability to classify the numbers-shapes gestures in a continuous stream of data.

	TD		FD		TDFD	
Scaling	ACVR	σ	ACVR	σ	ACVR	σ
Off	50.34	17.07	27.44	8	27.44	8
On	**99.28**	1.1	97.8	3.03	**99.28**	1.08

TABLE 6.3: The results for each of the three conditions, showing the ACVR and standard deviation.

6.2.2 SVM Experiment B

This experiment tested the classification abilities of the SVM algorithm with respect to a minimal amount of training data. This is an important test for MCI as, if a model can achieve as good a classification result with 5 training examples as it can with 50 training examples, then a performer can save time in both collecting the training data and also in training the model.

For each participant, a Support Vector Machine was trained using η randomly selected training examples from each of the 10 gestures and tested with the remaining data. The SVM was trained using the same settings used in 6.2.1, with time domain features being used as the input to the SVM. η ranged from 3 - 20, starting at 3 as opposed to 1 because at least 3 training examples are required to estimate the threshold value for each template and stopping at 20 to allow at least 5 test examples per trial. To ensure that the results of this test were not weighted by a 'lucky' random selection of the best template from the 25 training samples of each gesture, each test for η was repeated 10 times and the average correct classification ratio was recorded.

6.2.2.1 Results & Discussion

Figure 6.8 shows the ACCR values for each iteration of η. This test showed that the number of training examples significantly effects the classification abilities of the SVM algorithm. With just 3 training examples the SVM only achieved an ACCR value of 3.61%. At 10 training examples the SVM achieved a practical recognition result of 86.67%, however the standard deviation across all 10 participants was still high at 13.35%. With 13 training examples the SVM reached an excellent ACCR value of 95.3% with a moderate standard deviation of 6.81%. Using 20 training examples, the SVM achieved 97.76% with a very small standard deviation of 1.84%. The results of this test suggest that at least 11 training examples are required per-gesture if the user wants to achieve a robust classification result of $> 90\%$.

6.2.3 SVM Experiment C

This experiment tested the SVM's ability to correctly classify data from the numbers-shapes data set in a continuous stream of data that also contains a number of null gestures. This evaluated two important aspects of the SVM algorithm for the recognition of multivariate temporal gestures. Namely the algorithm's ability to correctly classify a set of temporal gestures from a continuous stream of data and also the algorithm's ability to reject any null gesture that is not contained in the model's database.

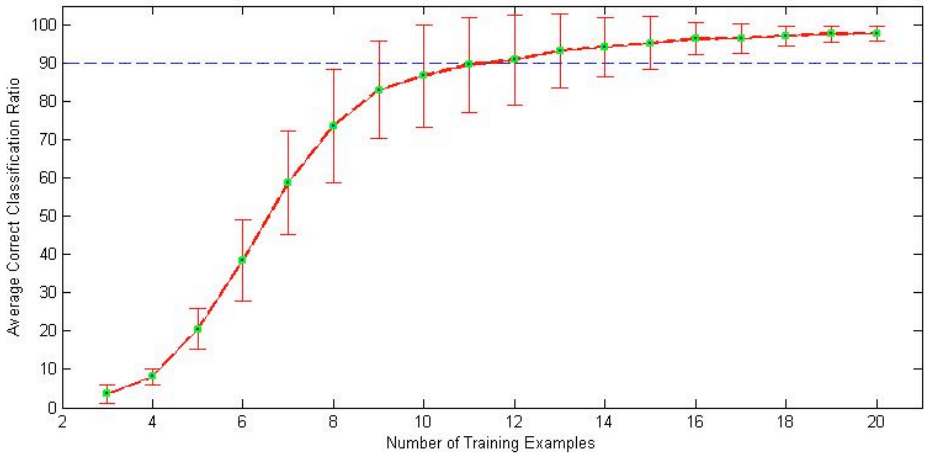


FIGURE 6.8: The ACCR values averaged across all 10 participants for each iteration of η. The horizontal blue line indicates the minimal training examples required to achieve a classification result of $> 90\%$.

For each participant, a SVM model was trained using 10 randomly selected training examples from each of the 10 gestures. After each model had been trained it was tested using a continuous stream of data. The continuous stream of data originated from the data-collection phase of the numbers-shapes database and contained all of the participant's trial recordings. The continuous stream, therefore, contained not only all of the 25 gestures the participant performed (10 of which where used to train the model) but also, importantly, the participant's movements in between each trial along with the periods of rest.

The continuous stream was tested by running a sliding window of size w over the data stream in increments of 5. The window size, w, was individually calculated for each participant by taking the average length of the training examples across the 10 gestures for that participant. For the majority of the participants, w was 306, with the shortest average window length of 250 and the longest average window length of 375. At each increment, the data within the window was given to the SVM model for classification. The gesture zone tag within the data was used to evaluate whether the SVM model had made the correct classification for each window of data. A correct classification result was considered if the correct class label was predicated by the SVM model (i.e. if the kth gesture gave the highest estimated probability and this probability was greater than 0.5). The following classification errors were evaluated for this experiment: ACCR, APR, ARR and ANRR.

6.2.3.1 Results

Figures 6.9 - 6.11 show the ACCR, APR and ARR results respectively. The SVM achieved a poor ACCR value of 38.33% (σ = 10.67%). Participant 7 achieved the maximum ACCR value of 52.34% with participant 9 achieving the minimum ACCR value of 22.34%. The APR and ARR values, which can be viewed in Figures 6.10 and 6.11 respectively, indicate that the SVM had a higher precision ratio than recall ratio. This shows that the SVM made the majority of classification errors by misclassifying gesture i as a null gesture, rather than misclassifying gesture i as gesture j. Table 6.4, which contains the precision ratios for each gesture and each participant, highlights one of the main disadvantages when using a sliding window approach for the recognition of a multivariate temporal gesture within a continuous stream of data. This disadvantage stems from having to choose a fixed window size from which to analyse the incoming stream of data, as some gestures maybe too small for the window and others maybe too large. The results in table 6.4 illustrate this problem as there are a number of null markers (-) that indicate that there were no windows of data for which these gestures had the maximum ID value and the analysis program therefore could not compute the recall and precision values for these gestures. Table 6.4 also illustrates that the SVM achieved an excellent recall ratio for the gestures that the analysis program could test. The precision ratios and recall ratios shown respectively in Figures 6.10 and 6.11 ignore the null markers in the computation of the mean and standard deviation values displayed in both figures. The SVM achieved an excellent ANRR value of 0.97, indicating that the algorithm was successful at classifying 97% of the null-gestures in the continuous data stream.

	G 1	G 2	G3	G 4	G 5	G 6	G 7	G 8	G 9	G10
P 1	1.00	1.00	1.00	1.00	1.00	0.98	1.00	1.00	1.00	1.00
P 2	-	1.00	1.00	1.00	1.00	-	1.00	1.00	1.00	-
P 3	0.55	1.00	1.00	1.00	1.00	-	1.00	1.00	1.00	1.00
P 4	1.00	0.99	1.00	1.00	0.94	1.00	0.97	0.68	-	1.00
P 5	1.00	1.00	-	1.00	1.00	0.93	0.49	1.00	1.00	1.00
P 6	0.98	0.99	0.84	1.00	1.00	-	0.55	1.00	1.00	1.00
P 7	1.00	1.00	1.00	1.00	1.00	0.92	1.00	1.00	1.00	1.00
P 8	0.00	1.00	-	0.50	-	0.71	-	-	1.00	-
P 9	1.00	1.00	1.00	1.00	0.84	1.00	1.00	-	1.00	1.00
P 10	1.00	1.00	1.00	1.00	1.00	-	1.00	1.00	1.00	1.00

TABLE 6.4: The precision ratio results for each of the 10 participants and 10 gestures.

6.2.3.2 Discussion

One interesting observation was gained by looking at the log files created during the training and testing program that ran this experiment. The log files showed that, for the gestures that were misclassified as a null gesture, the *a posterior* probabilities for each gesture, including the kth gesture, were all a very small value (commonly below 0.1). This is opposite to the case were a gesture was correctly classified, with the kth gestures *a posterior* probability commonly having a value greater than 0.7 and the summation of all the other gestures around 0.3. In other words, when the SVM made a correct classification the likelihood that a gesture had occurred was high, whereas when the SVM failed to make the correct classification the likelihood that no gesture had occurred was (incorrectly) low. This observation therefore indicates that simply lowering the SVM classification threshold to below 0.5 would not improve the classification results as there is no definitive change in the probability estimates between a gesture that was classified as a null-gesture and an actual null-gesture.

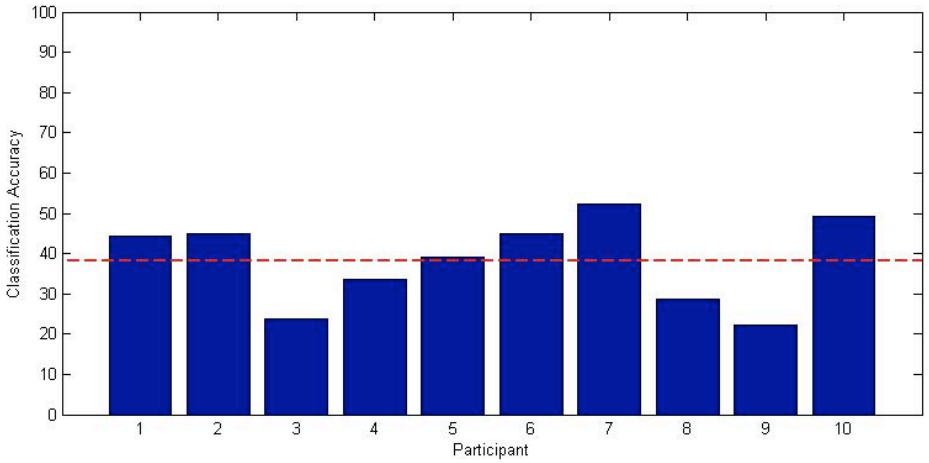


FIGURE 6.9: The correct classification results for each of the participants (blue) and the overall ACCR (dashed red line).

These results suggest that the SVM would be a poor choice for the recognition of multivariate gestures from a continuous stream of data, even though the SVM achieved excellent results with the same data set when the gestures were pre-segmented. Further work is therefore required to enable the SVM algorithm to achieve the same results in a continuous stream of data as it can with the pre-segmented data. In its present state, the SVM algorithm could not be used as a real-time recognition algorithm for the classification of musical gestures from a continuous stream of data, for example, during a live performance, as the performer could not rely on the system to automatically classify a gesture with any great accuracy. The performer would instead be required to manually

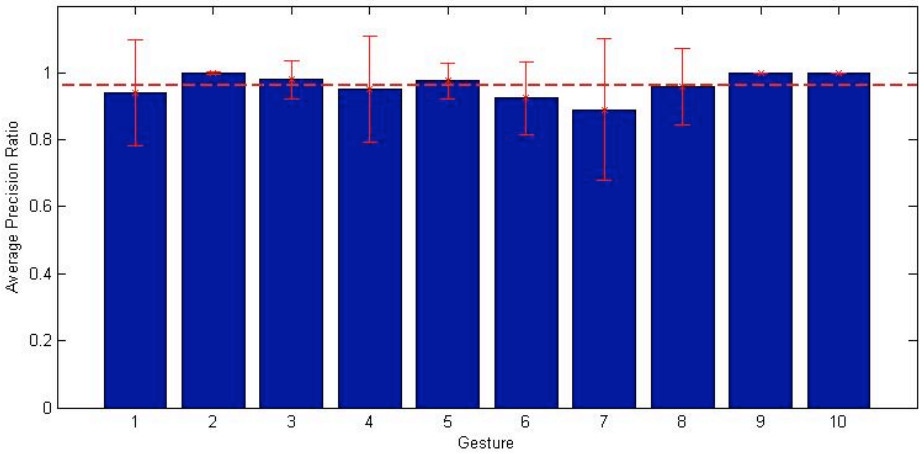


FIGURE 6.10: The APR results for each of the gestures averaged across all 10 participants (blue), along with the mean APR cross all 10 gestures (dashed red line) and standard deviation (red bars).

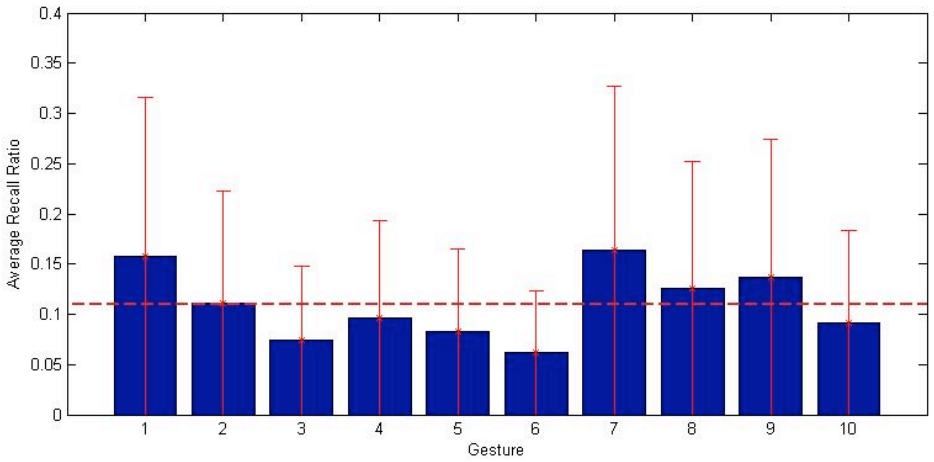


FIGURE 6.11: The ARR results for each of the gestures averaged across all 10 participants (blue), along with the mean ARR cross all 10 gestures (dashed red line) and standard deviation (red bars).

instruct the computer that they have just performed a gesture by, for example, pressing a foot-switch at the start and end of the gesture. The system could then take this segment of data and classify it against one of the gestures in its trained database.

6.3 SVM Summary

Support Vector Machines are a very powerful pattern recognition algorithm that have the following advantages for the recognition of multivariate temporal gestures:

1. Can easily solve non-linear classification problems using the kernel trick
2. The determination of the SVM model parameters corresponds to a convex optimization problem and therefore any local solution is also a global solution (unlike for example an Artificial Neural Network which can commonly get stuck in a local minimum rather than finding the global minimum)
3. The SVM can easily work with very large dimensional feature vectors
4. Even with a large amount of training data with high dimensional feature vectors, the SVM algorithm can still train a model in a very short training time
5. A trained SVM contains a very sparse model (as it uses just the support vectors) and therefore any real-time prediction can be computed efficiently

The work in this chapter has contributed the following aspects to the use of SVMs for the recognition of multivariate temporal musical gestures:

1. Used segmented statistical time domain features to enable basic statistical features, such as mean and standard deviation, to be used to desribe how a temporal signal changes over time
2. Compared these features with a number of frequency domain features, finding that for the recognition of the numbers-shapes data set the time domain features outperformed the frequency domain features and nothing was gained by combing the two features together
3. Applied Platt's posterior probabilities to define a rejection threshold that can be used to reject null gestures, thus alleviating the need to train a null model
4. Shown that the SVM with time domain features provides excellent classification results for pre-segmented multivariate temporal gestures
5. Identified the conditions under which SVMs are likely to provide the best results for the recognition of temporal gestures in a live performance scenario (i.e. using a trigger key for example as opposed to using a continuous sliding window)

6.4 Summary

This chapter has investigated how a powerful machine learning algorithm called Support Vector Machines can be applied for the recognition of multivariate temporal gestures. It has described how a number of feature extraction algorithms were specifically applied to represent multivariate temporal gestures and how the SVM was adapted to classify gestures from a continuous stream of data. The chapter was concluded with a number of experiments designed to evaluate the multivariate temporal classification abilities of the SVM using the numbers-shapes data set. The experiments conducted in sections 6.2.1, 6.2.2 and 6.2.3 have shown the SVM algorithm combined with time-domain features achieves excellent classification results on the numbers-shapes data set when the gestures have been pre-segmented, even with a limited number of training examples. Further work is required however for the adaption of the algorithm for the classification of gestures in a continuous stream of data. Having explored both HMM and SVM as potential methods for recognition of temporal gestures from a continuous stream of data, it was concluded that a novel approach would be required if the problem was to be satisfactorally addressed for the context of MCI. The next chapter therefore presents a novel algorithm that has been specifically designed for the recognition of multivariate temporal musical gestures, even from a continuous stream of data that also contains null gestures.

Chapter 7

Dynamic Time Warping

Everything should be made as simple as possible,
but not simpler.

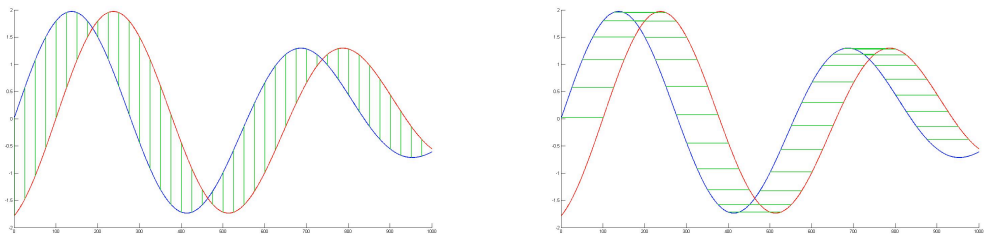
Albert Einstein

The previous chapter described how a Support Vector Machine could be applied to classify multivariate temporal gestures. Although the SVM algorithm achieved an excellent recognition rate of 99.28% on the pre-segmented data from the numbers-shapes data set, it failed to classify the same gestures from a continuous stream of data that also contained null gestures. This chapter presents an algorithm that has been specifically designed to recognise multivariate temporal musical gestures, even from a continuous stream of data. The algorithm is based on *Dynamic Time Warping* and has been extended to classify any N-dimensional signal and automatically compute a classification threshold to reject any data that is not a valid gesture from a continuous stream of data that also contains null gestures. The DTW algorithm is validated using the numbers-shapes data set after which the chapter is concluded by summarising the advantages and disadvantages of all the multivariate temporal recognition algorithms described throughout this thesis.

7.1 Dynamic Time Warping

Dynamic Time Warping is an algorithm that can compute the similarity between two time-series, even if the lengths of the time-series do not match. One of the main issues with using a distance measure (such as Euclidean distance) to measure the similarity between two time-series is that the results can sometimes be very unintuitive. If for example, two time-series are identical, but slightly out of phase with each other, then a

distance measure such as the Euclidean distance will give a very poor similarity measure. Figure 7.1 illustrates this problem. DTW overcomes this limitation by ignoring both local and global shifts in the time dimension (Salvador and Chan, 2007).



(a) Euclidean Distance (b) Dynamic Time Warping

FIGURE 7.1: Two identical time-series, slightly out of phase with each other, matched using Euclidean distance and Dynamic Time Warping.

7.1.1 Related Work

There has been much work over the last two decades in applying DTW to such varying fields as database indexing (e.g., Keogh and Pazzani, 2000, Ding et al., 2008), handwriting recognition (e.g., Vuori et al., 2001) and gesture recognition (e.g., Forbes and Fiume, 2005, Heloir et al., 2006, Leong et al., 2009). Both Merrill and Paradiso (2005) and Bettens and Todoroff (2009) have successfully applied DTW to the recognition of musical gestures. The vast majority of the recent work into DTW has focused on making the algorithm more computationally efficient, such as in the work by Keogh and Pazzani (2000) and Keogh and Pazzani (2001), with the time series in these works all being uni-dimensional signals. Proposed improvements to DTW included constraining the warping path (e.g., Sakoe and Chiba, 1990, Itakura, 1990), lower-bounding (e.g., Keogh and Ratanamahatana, 2005, Lemire, 2009), numerosity reduction (e.g., Xi et al., 2006) and recursive resolution projection (e.g., Salvador and Chan, 2007). It has only been in recent years that research has been conducted into extending DTW to multiple dimensions, with the exception of the early work by Stettiner et al. (1994) who proposed an extension of DTW to multiple dimensions for the application of speech recognition. Vlachos et al. (2003) extended DTW to match two-dimensional time series. In previous work by ten Holt et al. (2007) and also separately by Ko et al. (2008), multi-dimensional DTW was achieved by using a distance function such as the absolute sum, Euclidean distance or cosine correlation coefficient to compute the distance over all the dimensions in the test time series with a template time series for each sample in time. The result of this distance function was used by the standard DTW algorithm to compute the warping cost between the test time series and the template time series. Wullmer et al. (2009) proposed a different approach to multi-dimensional DTW, using a three-dimensional

distance matrix to compute the minimum distance between the input time series and a reference time series. This work used a bimodal input signal (speech data and gesture data captured by a mouse) and would therefore be computationally expensive to expand to an N-dimensional input stream as a large dimensional space would need to be constructed and navigated for each of the G gestures in the database.

7.1.2 One-Dimensional DTW

The foundation algorithm for DTW is as follows. Given two, one-dimensional, time-series, $\mathbf{x} = \{x_1, x_2, ..., x_{|\mathbf{x}|}\}^\intercal$ and $\mathbf{y} = \{y_1, y_2, ..., y_{|\mathbf{y}|}\}^\intercal$, with respective lengths $|\mathbf{x}|$ and $|\mathbf{y}|$, construct a *warping path* $\mathbf{w} = \{w_1, w_2, ..., w_{|\mathbf{w}|}\}^\intercal$ so that $|\mathbf{w}|$, the length of $\mathbf{w}$ is:

$$\max\{|\mathbf{x}|, |\mathbf{y}|\} \leq |\mathbf{w}| < |\mathbf{x}| + |\mathbf{y}| \tag{7.1.1}$$

where the kth value of $\mathbf{w}$ is given by:

$$\mathbf{w}_k = (\mathbf{x}_i, \mathbf{y}_j) \tag{7.1.2}$$

A number of constraints are placed on the warping path, which are as follows:

- The warping path must start at: $\mathbf{w}_1 = (1, 1)$
- The warping path must end at: $\mathbf{w}_{|\mathbf{w}|} = (|\mathbf{x}|, |\mathbf{y}|)$
- The warping path must be continuous, i.e. if $\mathbf{w}_k = (i, j)$ then $\mathbf{w}_{k+1}$ must equal either (i, j), $(i + 1, j)$, $(i, j + 1)$ or $(i + 1, j + 1)$
- The warping path must exhibit monotonic behavior, i.e. the warping path can not move backwards

There are exponentially many warping paths that satisfy the above conditions. However, the only path that needs to be found is the warping path that minimizes the normalised total warping cost given by:

$$\min \quad \frac{1}{|\mathbf{w}|} \sum_{k=1}^{|\mathbf{w}|} DIST(\mathbf{w}_{k_i}, \mathbf{w}_{k_j}) \tag{7.1.3}$$

where $DIST(\mathbf{w}_{k_i}, \mathbf{w}_{k_j})$ is the distance function (typically Euclidean) between point i in time-series $\mathbf{x}$ and point j in time-series $\mathbf{y}$, given by $\mathbf{w}_k$.

The minimum total warping path can be found by using dynamic programming to fill a two-dimensional ($|\mathbf{x}|$ by $|\mathbf{y}|$) cost matrix $\mathbf{C}$. Each cell in the cost matrix represents the

accumulated minimum warping cost so far in the warping between the time-series $\mathbf{x}$ and $\mathbf{y}$ up to the position of that cell. The value in the cell at $\mathbf{C}_{(i,j)}$ is therefore given by:

$$\mathbf{C}_{(i,j)} = DIST(i,j) + \min\{\mathbf{C}_{(i-1,j)}, \mathbf{C}_{(i,j-1)}, \mathbf{C}_{(i-1,j-1)}\} \tag{7.1.4}$$

which is the distance between point i in the time-series $\mathbf{x}$ and point j in the time-series $\mathbf{y}$, plus the minimum accumulated distance from the three previous cells that neighbor the cell i,j (the cell above it, the cell to its left and the cell at its diagonal).

When the cost matrix has been filled, the minimum possible warping path can easily be calculated by navigating through the cost matrix in reverse order, starting at $\mathbf{C}_{(|\mathbf{x}|,|\mathbf{y}|)}$, until cell $\mathbf{C}_{(1,1)}$ has been reached. At each step, the cells to the left, above and diagonally positioned with respect to the current cell are searched to find the minimum value. The cell with the minimum value is then moved to and the previous three cell search is repeated until $\mathbf{C}_{(1,1)}$ has been reached. Figure 7.2 illustrates the cost matrix and the minimum warping path.

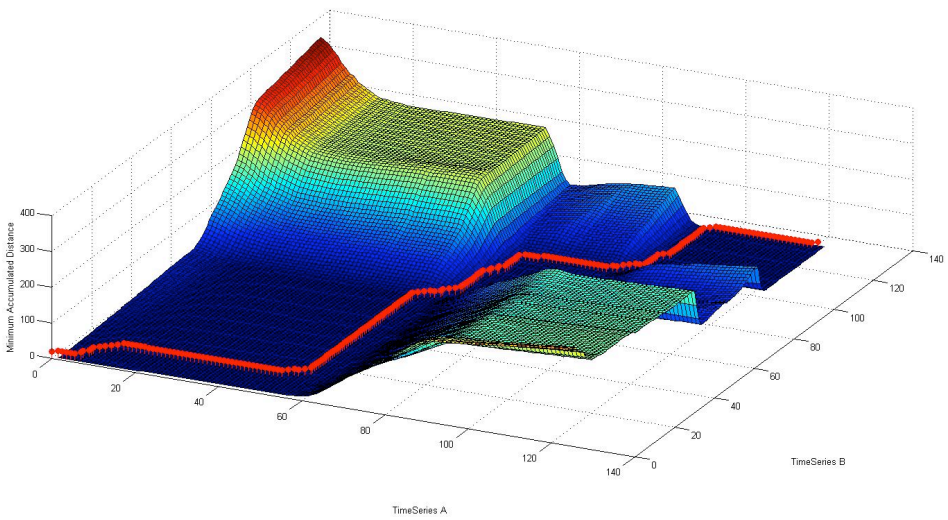


FIGURE 7.2: Cost Matrix and the Minimum Warping Path through it (indicated by the red line).

The warping path then gives the minimum normalised total warping distance between $\mathbf{x}$ and $\mathbf{y}$:

$$DTW(\mathbf{x},\mathbf{y}) = \frac{1}{|\mathbf{w}|}\sum_{k=1}^{|\mathbf{w}|} DIST(\mathbf{w}_{k_i}, \mathbf{w}_{k_j}) \tag{7.1.5}$$

Here, $\frac{1}{|\mathbf{w}|}$ is used as a normalisation factor to allow the comparison of warping paths of varying lengths.

7.1.3 Numerosity Reduction

DTW is a useful tool for computing the distance between two time-series. It is, however, a computationally costly algorithm to use for real-time recognition, as every value in the cost matrix must be filled. Clearly this is unusable for real-time recognition purposes, particularly if the unknown time-series is being matched against a large database of gestures.

To speed up both the training of the gesture templates and the real-time classification of an unknown N-dimensional input time-series, a number of numerosity reduction methods were tested. Perhaps one of the most rudimentary methods for numerosity reduction is to downsample the time-series by a factor of n. To avoid aliasing, the data is filtered using a low-pass FIR filter with a rectangular window and a filter order of n. The maximum cutoff frequency should be set to π/n to ensure the sampling theorem is maintained. Care should be taken when setting the downsample factor as, although using a high downsample factor gives the advantage of greatly reducing the size of the cost matrix, it also has the affect of removing important high frequency information from the signal which may reduce the overall accuracy of the algorithm.

7.1.4 Constraining the Warping Path

Another method commonly adopted for improving the efficiency of DTW is to constrain the warping path so that the maximum warping path allowed cannot drift too far from the diagonal. Controlling the size of this warping window will greatly affect the speed of the DTW computation. If the warping window is small, a large proportion of the cost matrix does not need to be searched or even constructed. The size of the warping window can be controlled by varying the parameter r, given as the percentage of the length of the template time-series. The warping window is then set as the distance, r, from the diagonal to directly above and to the right of the diagonal. This type of global constraint is referred to as the Sakoe-Chiba band (Sakoe and Chiba, 1990), Itakura (1990) has also proposed another global constraint based on a parallelogram.

7.2 N-Dimensional Dynamic Time Warping

Section 7.1.1 described the standard implementation of DTW for two uni-dimensional time-series. It is common, however, in computational fields such as gesture recognition to have time-series that feature multiple-dimensions, such as data captured by a 3-axis accelerometer. It is in this instance that we require an implementation of DTW that can compute the distance between two N-dimensional time-series. The common approach used by ten Holt et al. (2007) and Ko et al. (2008) will be used to compute the distance between two N-dimensional time-series. This takes the summation of distance errors between each dimension of an N-dimensional template and the new N-dimensional time-series. The total distance across all N dimensions is then used to construct the warping matrix $\mathbf{C}$. The Euclidean distance will be used as a distance measure across the N dimensions of the template and new time-series.

$$DIST(i,j) = \sqrt{\sum_{n=1}^{N}(i_n - j_n)^2} \tag{7.2.1}$$

The following section describes our N-Dimensional Dynamic Time Warping (**ND-DTW**) algorithm. In the training stage, an N-dimensional template (ϕ_g) and threshold value (τ_g) for each of the G gestures is computed. In the real-time prediction stage a new N-dimensional time-series is classified against the template that gives the minimum normalised total warping distance between the N-dimensional template and the unknown N-dimensional time-series. Each element of the algorithm will now be discussed in detail.

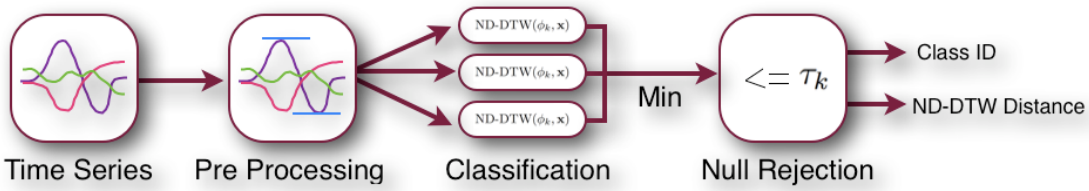


FIGURE 7.3: Real-time classification using ND-DTW.

7.2.1 Training the ND-DTW Algorithm

In order for ND-DTW to be used as a real-time recognition algorithm, a template must first be created for each gesture that needs to be classified. A template can be computed by recording M_g training examples for each of the G gestures that are required to be recognised. After the training data has been recorded, each of the G templates can be found by computing the distance between each of the M_g training examples for the kth

gesture and searching for the training example that gives the minimum normalised total warping distance when matched against the other M_g-1 training examples in that class. The gth template (ϕ_g) is therefore given by:

$$\phi_g = \arg\min_i \quad \frac{1}{M_g - 1} \sum_{j=1}^{M_g} \mathbf{1}\{\text{ND-DTW}(\mathbf{X}_i, \mathbf{X}_j)\} \qquad 1 \leq i \leq M_g \tag{7.2.2}$$

where the $\mathbf{1}\{\cdot\}$ that surrounds the ND-DTW function is the indicator bracket, giving 1 when $i \neq j$ or 0 otherwise and $\mathbf{X}_i$ and $\mathbf{X}_j$ are the ith and jth N-dimensional training examples for the gth gesture in the form of $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_N\}$ with $\mathbf{x}_i = \{x_1, x_2, ..., x_{|\mathbf{x}|}\}^\intercal$. The ND-DTW function in (7.2.2) is simply the extension of the standard DTW algorithm to N-dimensions:

$$\text{ND-DTW}(\mathbf{X}, \mathbf{Y}) = \quad \min \quad \frac{1}{|\mathbf{w}|} \sum_{k=1}^{|\mathbf{w}|} DIST(\mathbf{w}_{k_i}, \mathbf{w}_{k_j}) \qquad DIST(i, j) = \sqrt{\sum_{n=1}^{N} (i_n - j_n)^2} \tag{7.2.3}$$

7.2.2 Multi-Threaded Training

One major advantage of using the DTW algorithm is that each template (i.e. each gesture) can be computed independently from the other templates. This is of particular use on new machines that feature multiple processors as a multi-threaded training approach can be adopted in which each template's training routine is launched in a separate thread. This training approach greatly speeds up the overall training time for a DTW classification system as one template does not need to wait for the previous template to be trained before it can start its own training routine.

The DTW algorithm also has one other advantage in that, if a new gesture is added to an existing trained database or an existing gesture is removed, the entire database does not need to be retrained. Instead, a new template and threshold value only needs to be trained for the new gesture, thus greatly reducing the training time. If an existing gesture is removed from the database then no training is required as the DTW classification system simply removes this template and threshold value from its database. This is not the case for other machine learning algorithms, such as an Artificial Neural Network,

as the entire system would need to be retrained from scratch any time a new gesture is added or removed.

7.2.3 Classification using the ND-DTW Algorithm

After the templates have been created for each gesture in the database, an unknown N-dimensional time-series $\mathbf{X}$ can be classified by computing the normalised total warping distance between $\mathbf{X}$ and each of the G templates. c, the classification index representing the gth gesture is then given by finding the corresponding template that gave the minimum normalised total warping distance:

$$c = \arg\min_{g} \quad \text{ND-DTW}(\phi_g, \mathbf{X}) \qquad 1 \leq g \leq G \tag{7.2.4}$$

7.2.4 Determining the Classification Threshold

Using equation (7.2.4) $\mathbf{X}$, an unknown N-dimensional time-series, can be classified by calculating the distance between it and all the templates in the database. The unknown time-series $\mathbf{X}$ can then be classified against the template that results in the lowest normalised total warping distance. This method will, however, give false positives if the N-dimensional input time-series $\mathbf{X}$ is in-fact not made up of any of the gestures in the database. This false classification problem can be mitigated by determining a classification threshold for each template gesture during the training phase. In the prediction phase, a gesture will only be classified against the template that results in the lowest normalised total warping distance, if this distance is less than or equal to the gesture's classification threshold. If the distance is above the classification threshold, then the algorithm will classify the gesture against a null class, indicating that no match was found:

$$\hat{c} = \begin{cases} c & if(d \leq \tau_g) \\ 0 & \text{otherwise} \end{cases} \tag{7.2.5}$$

where c is given by equation (7.2.4), d is the total normalised warping distance between ϕ_g and $\mathbf{X}$ and τ_g is the classification threshold for the gth template.

The classification threshold for each template can be set as the average total normalised warping distance between ϕ_g and the other $M_g - 1$ training examples for that gesture, plus γ standard deviations:

$$\tau_g = \mu_g + (\sigma_g \gamma) \tag{7.2.6}$$

where

$$\mu_g = \frac{1}{M_g - 1} \sum_{i=1}^{M_g} \mathbf{1}\{\text{ND-DTW}(\phi_g, \mathbf{X}_i)\} \tag{7.2.7}$$

$$\sigma_g = \sqrt{\frac{1}{M_g - 2} \sum_{i=1}^{M_g} \mathbf{1}\{(\text{ND-DTW}(\phi_g, \mathbf{X}_i) - \mu_g)^2\}} \tag{7.2.8}$$

where the $\mathbf{1}\{\cdot\}$ that surrounds the ND-DTW function is the indicator bracket, giving 1 when $i \neq$ the index of the training example that gave the minimum normalised total warping distance when matched against the other M_g-1 training examples in that class (i.e. the template) or 0 otherwise and $\mathbf{X}_i$ is the ith training example for the gth class. γ can be initially set to a number of standard deviations (e.g. 2) during the training phase and later adjusted by the user in the real-time predication phase until a suitable classification/rejection level has been achieved.

It is critical when calculating the classification threshold for each of the g gestures to perform any pre-processing such as scaling or downsampling in the same order as it would be performed during the real-time classification stage. If this is not completed in the same order then the optimal classification threshold will not be found. The various pre-processing options that can be used for ND-DTW will now be discussed.

7.2.5 Pre-processing for ND-DTW

Pre-processing is necessary for ND-DTW if either (a) any of the N-dimensional data originate from a different source range or (b) if invariance to spatial variability and variability of signal magnitude is desired. Each of these scenerios will now be discussed, giving appropriate pre-processing solutions for each.

(a) Varying Input Source Ranges

It is important for each of the N-dimensional data in the time-series $\mathbf{X}$ to originate from a common source range. If this is not the case then one or more of the dimensions may heavily weight the results of the DTW. If each of the N-dimensional data do not

originate from a common source range then each dimension should be scaled using min-max normalisation prior to both the training of the templates and real-time prediction. The minimum and maximum source range values for each of the N-dimensions can be found by searching over all the training data and locating the min and max values for each dimension. Each dimension of the training data (training phase only) or the unknown data (prediction phase only) can then be scaled using these min and max values to a standard target range (i.e. 0.0 to 1.0).

(b) Invariance to Spatial Variability and Variability of Signal Amplitude

Spatial variance and variability in the signal amplitude can be mitigated by first z-normalising both the input time-series and also the recognition templates. Z-normalisation will give both the input and template time-series zero mean and unit variance, therefore removing any affect that spatial variation or variability in the signal amplitude may have had. Keogh and Pazzani (2001) also proposed using the derivative of the input signals to account for similar spatial problems. This method was also used successfully by ten Holt et al. (2007).

7.2.6 Dealing With A Large Gestural Vocabulary

The ND-DTW algorithm has been used in a number of real-time recognition scenarios that allowed a performer to use body movements (such as hand and arm gestures) to control various audio parameters and effects live on stage. These tests all involved relatively small gestural vocabularies, normally consisting of approximately 10 gestures with each gesture lasting around 1-2 seconds. With this small gesture vocabulary the ND-DTW algorithm was able to classify the correct gesture in real-time with insignificant processing delay. If the gestural vocabulary being used consists of a large number of gestures (i.e. $G > 100$), then the processing delay will become unusable for real-time interaction. This problem can be mitigated by grouping the trained templates into clusters and using a search tree to classify an unknown N-dimensional time-series by using ND-DTW to compute the distance between the projected time-series with each node in the current leaf, working down through each child until the best match has been found.

7.3 ND-DTW Experiments

Three experiments were run to validate the classification abilities of the ND-DTW algorithm. The first experiment evaluated the classification abilities of the ND-DTW algorithm with the pre-segmented gestures from the numbers-shapes data set (see chapter 5.2). The second experiment evaluated the classification abilities of the ND-DTW algorithm with respect to a minimal amount of training data. Finally, the third experiment evaluated the ND-DTW algorithm's ability to correctly classify data from the numbers-shapes data set in a continuous stream of data that also contains a number of null gestures.

7.3.1 ND-DTW Experiment A

This experiment evaluated the ND-DTW algorithm's ability to correctly classify the pre-segmented data from the numbers-shapes data set. For each participant, a ND-DTW model was trained using 10-fold cross-validation, with the average cross-validation ratio (**ACVR**) taken over all 10 participants being used to evaluate the algorithm. This experiment was run with four conditions (C1) scaling off, z-normalisation off, (C2) scaling on, z-normalisation off, (C3) scaling off, z-normalisation on and (C4) scaling on z-normalisation on. γ was set to 2 for this experiment and downsampling was used with n set to 5.

7.3.1.1 Results

Condition C2 (scaling on, z-normalisation off) achieved the maximum ACVR of 99.37%, however the other conditions also achieved excellent classification results of 98.85% for C1, 98.95% for C3 and 99.37% for C4. The ND-DTW algorithm achieved the maximum cross-validation classification result in condition C2 of 100% for participants 1, 2, 3, 5 and 7. The ND-DTW algorithm achieved the minimum cross-validation classification result with 95.5% for participant 10. Figure 7.4 shows the cross-validation results for each of the 10 participants in condition C2.

7.3.1.2 Discussion

This test shows that the ND-DTW algorithm provides excellent classification results with pre-segmented data, achieving an ACVR value of 99.37%, an improvement of just over 11% when compared with the HMM on the same data set. The algorithm achieved a perfect recognition result of 100% for various participants, with the algorithm achieving

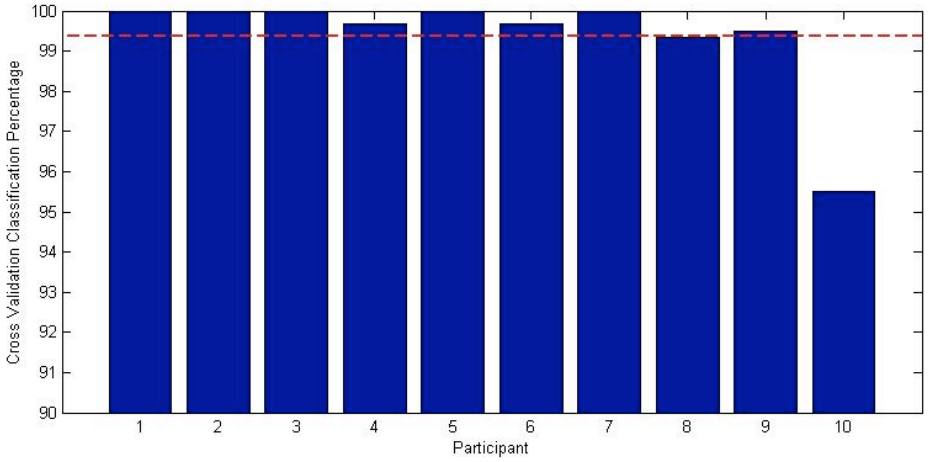


FIGURE 7.4: The cross-validation classification results for each of the 10 participants in condition C2. The ACVR is shown by the horizontal red dashed line.

a classification result of over 99% for all but 1 participant. All four conditions achieved excellent classification results of 98% and above. This suggests that the scaling and normalisation pre-processing methods had little influence on the classification abilities of the algorithm when each gesture was pre-segmented. This is not surprising as the input to the ND-DTW algorithm consisted of a 3-dimensional vector containing the x, y and z position coordinates of the Polhemus sensor that was located on each participant's right hand. Each dimension therefore originated from a common source range and scaling or z-normalising only made a small improvement to the overall classification abilities of the algorithm.

7.3.2 ND-DTW Experiment B

This experiment evaluated the classification abilities of the ND-DTW algorithm with respect to a minimal amount of training data. This is an important test for MCI as, if a model can achieve as good a classification result with 5 training examples as it can with 50 training examples, then a performer can save time in both collecting the training data and also in training the model.

7.3.2.1 Method

For each participant, a ND-DTW model was trained using η randomly selected training examples from each of the 10 gestures and tested with the remaining data. η ranged from 3 - 20, starting at 3 as opposed to 1 because at least 3 training examples are required to estimate the threshold value for each template and stopping at 20 to allow at least 5 test

examples per trial. To ensure that the results of this test were not weighted by a 'lucky' random selection of the best template from the 25 training samples of each gesture, each test for η was repeated 10 times and the average correct classification ratio was recorded. The scaling pre-processing option was used for this experiment, however z-normalisation was not, as this combination of pre-processing options achieved the best results in the ND-DTW experiment A. γ was set to 2 for this experiment and downsampling was used with n set to 5.

7.3.2.2 Results & Discussion

Figure 7.5 shows the ACCR values for each iteration of η. This test suggests that the number of training examples significantly affects the classification abilities of the ND-DTW algorithm. The ND-DTW algorithm achieved a moderate ACCR value of 74.74% with just 3 training examples. With 20 training examples it was able to achieve an ACCR value of 92.19%. It should be noted that the standard deviation over each iteration of η and across all 10 participants was very high. This shows that the classification abilities of the ND-DTW algorithm is heavily dependent on getting the 'best' training examples. An ACCR value of $> 90\%$, for example, was achieved for several participants with just 3 training examples. An ACCR value of $< 70\%$, however, was also achieved for the same participants with 3 training examples, showing that the 'quality' of the training examples heavily influences the results of the classification algorithm. The results of this test suggest that at least 11 training examples are required per-gesture if the user wants to achieve a robust classification result of $> 90\%$.

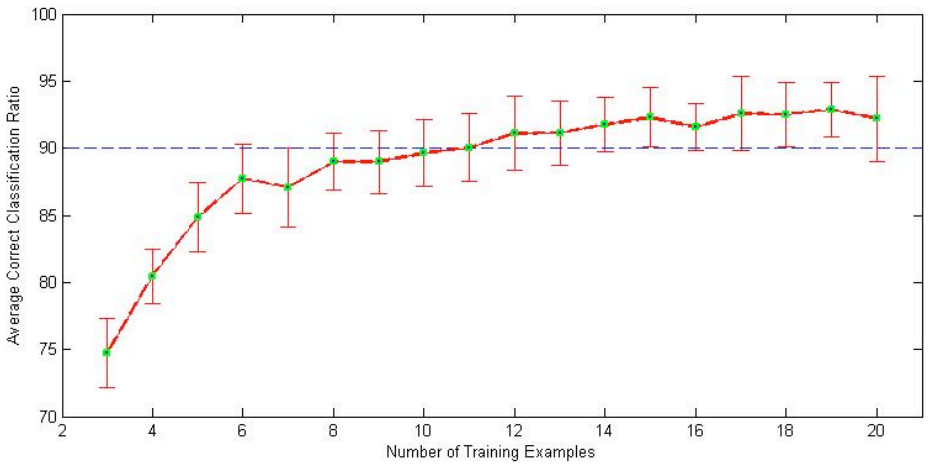


FIGURE 7.5: The ACCR values averaged across all 10 participants for each iteration of η. The horizontal blue dashed line indicates the minimal training examples required to achieve a classification result of $> 90\%$.

7.3.3 ND-DTW Experiment C

This experiment evaluated the ND-DTW algorithm's ability to correctly classify the gestures from the numbers-shapes data set in a continuous stream of data that also contains a number of null gestures. This evaluates two important aspects of ND-DTW for the recognition of multivariate temporal gestures. Namely the algorithm's ability to correctly classify a set of temporal gestures from a continuous stream of data and also the algorithm's ability to reject any null gesture that is not contained in the model's database.

7.3.3.1 Method

For each participant, a ND-DTW model was trained using 12 randomly selected training examples from each of the 10 gestures. After each model had been trained it was tested using a continuous stream of data. The continuous stream of data originated from the data-collection phase of the numbers-shapes database and contained all of the participant's trial recordings. The continuous stream therefore contains not only all of the 25 gestures the participant performed (12 of which were used to train the model) but also, importantly, the participant's movements in between each trial along with the periods of rest.

The continuous stream was tested by running a sliding window of size w over the data stream in increments of 10. The window size, w, was individually calculated for each participant by taking the average length of the 10 ND-DTW templates for that participant. For the majority of the participants, w was 304, with the shortest window length of 248 and the longest window length of 368. At each increment, the data within the window was given to the ND-DTW model for classification. Each sample in the continuous stream had been labelled with the gesture ID tag (0 for a null-gesture or the gth class ID for an actual gesture). This ID tag was used to evaluate if the ND-DTW model had made the correct classification for each window of data. As some windows may cover a section of data that contains half a gesture and non-gestural data, the classification results of a window were only used if the maximum ID count within the window was greater than 80% of the length of the window. The threshold value of 80% was used as this enabled any gesture that was shorter than the window size to still be included in the analysis. A correct classification result was considered if the correct class label was predicated by the ND-DTW model (i.e. if the gth template gave the minimum normalised warping distance and if this value was less than the gth model's classification threshold). The following classification errors were evaluated for this experiment:

	C1		C2		C3		C4	
	ACCR	σ	ACCR	σ	ACCR	σ	ACCR	σ
$\gamma : 2.0$	78.48	11.08	78.03	9.70	62.92	9.78	62.92	9.78
$\gamma : 5.0$	83.31	11.55	**84.18**	9.25	74.15	12.19	74.15	12.19
$\gamma : 10.0$	77.93	9.31	80.69	7.61	74.67	13.11	74.67	13.11

TABLE 7.1: The ACCR results for each value of γ and for each of the four pre-processing conditions.

ACCR, APR, ARR and ANRR. This test was run with same four pre-processing conditions used in the ND-DTW experiment A. This test was also run with three different values of γ to evaluate the affect this parameter has on the continuous classification abilities of the ND-DTW algorithm, with γ set to 2.0, 5.0 and 10.0. Downsampling was used in all conditions with n set to 5.

7.3.3.2 Results

Table 7.1 shows the ACCR results for each setting of γ and each of the four pre-processing conditions. The maximum ACCR value of 84.18% was achieved with a γ value of 5.0 using the C2 pre-processing condition (scaling on, znormalistion off). With these settings, the maximum individual correct classification result of 95.23% was achieved by the algorithm for participant 1, while the algorithm achieved the minimum individual correct classification result of 64.09% for participant 8. Figure 7.6 shows the correct classification results (CCR) for all the participants in the best performing trial. Table 7.2 shows the individual correct classification results for each participant, for each setting of γ and each of the four pre-processing conditions. The type of pre-processing method used had a significant affect on the classification results over all values of γ, with the conditions using znormalisation frequently achieving poorer classification results than those conditions that did not use znormalisation. A possible explanation for the poor performance of the conditions that used znormalisation is that, because znormalisation will give the data zero mean and unit variance, this could reduce the difference between a null gesture and an actual gesture and therefore increase the number of false-positive classification errors.

Table 7.3 shows the APR and ARR results for the best performing condition ($\gamma = 5.0$, C2). The APR and ARR results show that the majority of classification errors were made by in the recall of the algorithm, as opposed to the precision of the algorithm. This shows that the DTW algorithm made the majority of classification errors by misclassifying gesture i as a null gesture, rather than misclassifying gesture i as gesture j. The ANRR

value of 0.88 indicates that the algorithm was successful at rejecting null gestures 88% of the time.

7.3.3.3 Discussion

These results suggest that the ND-DTW algorithm performed well at classifying a number of multivariate temporal gestures from a continuous stream of data that also contained a number of null gestures. The most interesting results from this evaluation came from the affect that the γ parameter had on the classification abilities of the algorithm. Increasing γ from 2.0 to 5.0 increased the overall classification abilities of the algorithm (from a maximum ACCR value of 78.48% with $\gamma = 2.0$ to a maximum ACCR value of 84.18% with $\gamma = 5.0$). Increasing the γ value, however, also had the adverse effect of decreasing the overall precision of the algorithm, with the average APR across all 10 participants decreasing from 0.95 with $\gamma = 2.0$, to 0.90 with $\gamma = 5.0$. The ANRR value also decreased from 0.94 with $\gamma = 2.0$, to 0.88 with $\gamma = 5.0$. These results show that increasing γ, and therefore increasing the classification threshold value for each gesture, increases the likelihood that a gesture will be classified correctly but it will also unfortunately increase the likelihood of false-positive misclassications. If the γ parameter is set too high then the overall classification abilities of the algorithm will start to degrade altogether, as was the case when γ was set to 10.0. This problem illustrates the compromise that a user must make about the sensitivity of their classication system. One performer, for example, may prefer a classification system that always recognises their gestures but also frequently makes false-positive classifications. Another perform could, alternatively, choose to have a classification system that failed to correctly recognise all of their gestures but rarely made any false-postive classification errors. It is for this specific reason that the γ parameter was added to the ND-DTW algorithm as this enables the performer to manually adjust this threshold value themselves until they have reached a satisfactory recognition rate.

One very important observation was made when looking at the prediction log-file for this experiment that suggests that the continuous classification abilities of the ND-DTW are in-fact even better than the ACCR results would imply. The prediction log-file was a data file that contained the analysis results for each window of data that was analysed by the ND-DTW for every participant, each γ value and each condition. The analysis results contained a number of values representing the status of the ND-DTW model at each analysis window, including the known class ID tag, the models predicted ID tag and the ND-DTW distance estimates for the 10 gestures. The algorithm frequently made the correct classification of a gesture when the sliding window reached the 'middle' of the gesture. However, the ND-DTW algorithm would commonly estimate that the data

	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10
γ : 2.0										
C1	86.45	89.26	75.66	76.37	79.92	85.10	88.94	51.51	78.66	72.91
C2	87.20	85.02	72.03	74.10	82.70	86.48	88.87	59.48	76.67	67.78
C3	72.16	65.07	53.47	64.12	64.58	67.99	76.35	41.38	63.78	60.28
C4	72.16	65.07	53.47	64.12	64.58	67.99	76.35	41.38	63.78	60.28
γ : 5.0										
C1	94.23	90.98	83.69	79.95	82.75	91.49	91.28	54.03	85.60	79.09
C2	**95.23**	90.56	84.91	77.76	86.21	89.20	93.89	64.09	81.03	78.89
C3	84.71	79.71	62.20	72.00	79.68	78.92	86.29	45.54	79.42	73.02
C4	84.71	79.71	62.20	72.00	79.68	78.92	86.29	45.54	79.42	73.02
γ : 10.0										
C1	85.44	83.14	81.19	78.95	75.88	84.71	80.68	54.76	83.93	70.60
C2	89.33	86.78	83.84	78.42	84.80	83.60	83.52	63.33	80.38	72.92
C3	91.27	83.67	64.95	71.98	77.97	76.96	75.34	43.60	84.91	76.09
C4	91.27	83.67	64.95	71.98	77.97	76.96	75.34	43.60	84.91	76.09

TABLE 7.2: The individual correct classification results for each of the 10 participants, for each of the three values of γ and each of the four pre-processing conditions. Condition C2 with $\gamma = 5.0$ achieved the maximum ACCR value of 84.18%.

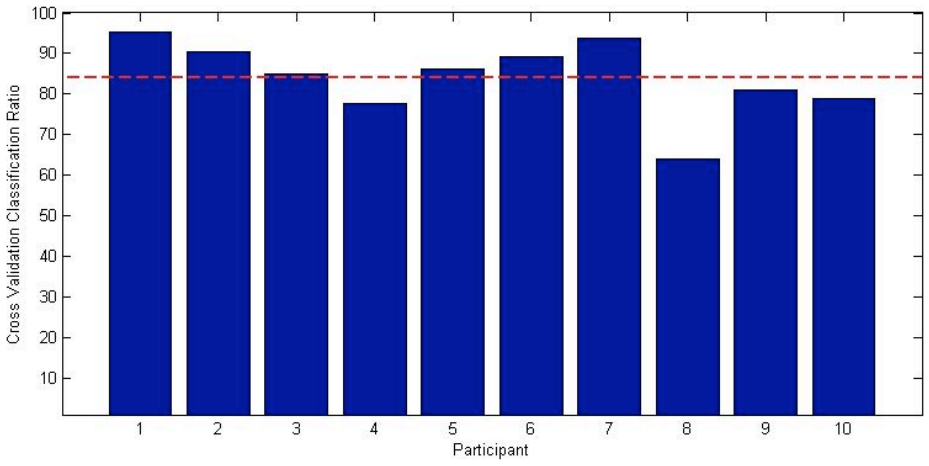


FIGURE 7.6: The correct classification results for each of the 10 participants with $\gamma = 5.0$ and using the pre-processing condition C2 (scaling on and z-normalisation off). The ACCR value is shown by the horizontal red dashed line.

	G1	G2	G3	G4	G5	G6	G7	G8	G9	G10
APR	0.91	0.89	0.80	0.79	0.92	0.95	0.83	0.96	0.95	0.96
ARR	0.93	0.78	0.85	0.81	0.82	0.66	0.94	0.93	0.92	0.90

TABLE 7.3: The average precision ratio and average recall ratio for each gesture with $\gamma = 5.0$ and using the pre-processing condition C2 (scaling on and z-normalisation off).

in the sliding window was a null gesture when in-fact the window contained a majority of the start or end of a gesture. Figures 7.7 and 7.8 illustrate this problem. The significance of this observation is that the ND-DTW algorithm may not have correctly classified every single window of data, however, it was able to consistently classify each gesture correctly over the course of a number of consecutive windows, commonly when the sliding window had reached the 'middle' of a gesture. This means that if a performer used a post-processing class ID filter (that only allowed a gesture ID to be acted upon if n consecutive gesture IDs were predicted for example) on the predicted gesture ID of the ND-DTW algorithm then the classification abilities of the algorithm would be even more robust than the ACCR result suggests.

7.4 ND-DTW Summary

The ND-DTW algorithm provides a formidable algorithm for the recognition of multivariate temporal gestures due to its applicability to classifying temporally variant signals. The experiments conducted in sections 7.3.1, and 7.3.3 have shown that the ND-DTW algorithm achieves excellent classification results on the numbers-shapes data set when the gesture has been pre-segmented and with a limited number of training examples. The algorithm also achieved moderate classification results when used to recognise the numbers-shapes gestures from a continuous stream of data that also contained null gestures, something that neither the Hidden Markov Models or Support Vector Machines achieved.

The following modifications to the DTW algorithm were contributed to optimize it for the recognition of musical gestures:

1. Adopted a multi-threaded training approach for each gesture to minimise the total training time of the algorithm
2. Computed the template for each gesture using equation (7.2.2)
3. Added a threshold classification value for each gesture, thus alleviating the need for training a null gesture model
4. Enabled the threshold value for each gesture to be manually adjusted by the user in the real-time prediction phase, thus allowing the performer to choose their own suitable classification/rejection level

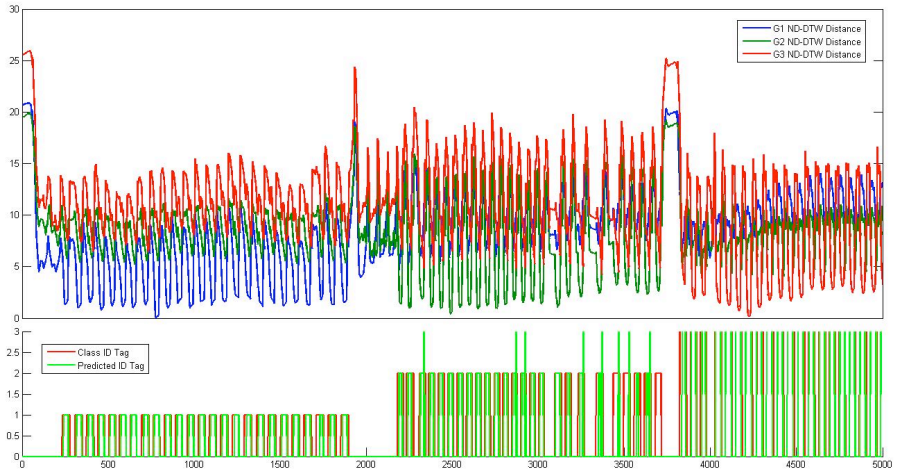


FIGURE 7.7: An illustration of the prediction abilities of the ND-DTW algorithm. The three waveforms in the top most graph show the ND-DTW distance between a window of data and each of the first three gesture templates. The bottom graph shows the class ID tag and overall predicted ID tag from the ND-DTW algorithm at the same time window as the top graph.

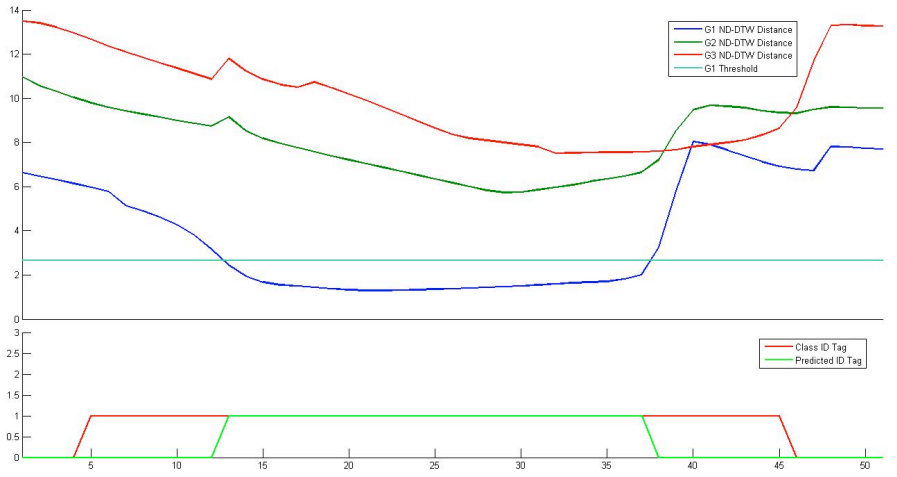


FIGURE 7.8: An illustration of the continuous classification abilities of the ND-DTW algorithm. This image contains a small segment of data that was shown in Figure 7.7. In addition to the three distance measures for each of the first three gesture templates, the top graph now also shows the classification threshold for the first gesture (solid horizontal cyan line). Note, by looking at the predicted ID tag in the lower graph, how the algorithm correctly estimates that gesture 1 has occurred but only from the windows that cover the center of the gesture and not the start or end windows.

7.5 Multivariate Temporal Recognition Algorithm Summary

This chapter and the previous two chapters have all presented algorithms that can be used for the real-time recognition of multivariate temporal musical gestures. This chapter is concluded by summarising the advantages and disadvantages of each algorithm and suggest scenarios where a performer may want to favor one algorithm over another.

Reiterating the design criteria outlined in chapter 3.2, a machine learning algorithm used for musician-computer interaction should provide the following functionality:

1. The input to the algorithm should be flexible and not constrained to one type of sensor or feature
2. The algorithm should be able to be quickly trained
3. The algorithm should not require hundreds of training examples in order to train a robust model
4. The algorithm should be able to recognise a gesture from a continuous stream of data and also reject null gestures without having to first train a noise/null gesture model
5. The algorithm should provide a low intra-personal generalisation error

With these goals in mind the three multivariate temporal recognition algorithms described throughout this chapter and the previous two chapters have been grouped into table 7.4.

	HMM	**SVM**	**ND-DTW**
Flexible N-Dimensional Input	Yes*	Yes	Yes
Training Time	Moderate	Low	Low
Training Size Required	Moderate	Low	Low
Applicability For Pre-segmented Recognition	Moderate	High†	High
Applicability For Unsegmented Recognition	Low	Low	High
Intra-personal Classification Rate	Moderate	High	High
†: when appropriate feature extraction is used such as the time domain or frequency domain features presented in chapter 6.1.4.			
$\star$: when appropriate feature extraction is used such as k-means and SAX			

Table 7.4: A summary of the advantages and disadvantages of the HMM, SVM and ND-DTW algorithms for the real-time classification of multivariate temporal musical gestures.

7.5.1 Choosing Which Algorithm To Use When

The HMM, SVM and ND-DTW algorithms can all be applied to the recognition of multivariate temporal musical gestures. There are instances, however, where a performer may want to favor one algorithm over another. The results of the experiments on the numbers-shapes data set suggest that a performer might want to first try either the SVM or ND-DTW algorithm before trying the HMM algorithm as the SVM and ND-DTW algorithms consistently outperformed the HMM algorithm in several scenarios. But when should a performer use the SVM algorithm and when should they try the ND-DTW algorithm? To answer this question, a number of possible scenarios where one algorithm would be more appropriate to try before the other have been highlighted.

7.5.1.1 Limited Number Of Training Examples

What if a performer only wants to record a very limited number of training examples for each gesture to train their recognition system (i.e. 1 or 2 examples at most). Which algorithm would be best to use in this scenario? Both the SVM and ND-DTW algorithm performed well with a limited number of training examples, with both algorithms achieving a classification result of 90% with just 11 training examples. The SVM algorithm, however, performed poorly with a very limited number of training examples, achieving an ACCR value of less than 10% when just 4 training examples were used to train the algorithm. The ND-DTW algorithm, alternatively, performed significantly better with a very limited number of training examples, i.e. achieving an ACCR value of over 80% with only 4 training examples. The ND-DTW algorithm could in-fact be trained with just one training example per gesture, however, the user would need to empirically set the classification threshold value for each gesture as the algorithm can not estimate this with only one training example. The ND-DTW algorithm would therefore be a better algorithm for a performer to first use to attempt to classify their multivariate temporal musical gestures if they intend to train the algorithm with only a limited number of training examples.

7.5.1.2 Substantial Number Of Training Examples

What if a performer has prototyped their gestural vocabulary and is happy with each action-sound relationship and now wants to create the most robust recognition system possible? In this instance the performer might be willing to record hundreds of training examples for each gesture in their gestural vocabulary. If an ample number of training

examples are available (i.e. hundreds or even thousands of training examples per gesture) which algorithm would be best to use? Both the SVM algorithm and ND-DTW algorithm achieved excellent classification results of 99% on the numbers-gestures data set when the majority of the data set was used to train the algorithms. One advantage that the SVM algorithm has over the ND-DTW algorithm is that, due to the way it is trained, a larger training set will commonly result in a more robust model. Increasing the size of the data set should therefore always improve the generalisation abilities of the SVM algorithm. The ND-DTW algorithm on the other hand may not benefit as much from a very large training-set. This is because the algorithm picks 'the best' training example from the data set to use as the gth template. The algorithm, however, only uses one training example as the template for each gesture regardless of the number of training examples available. Therefore, increasing the size of the data set may only marginally improve the generalisation abilities of the algorithm. The ND-DTW algorithm could take advantage of a large data set if more than one template was chosen for each gesture and a k-nearest neighbor voting scheme was used to classify an unknown gesture against one of the G gestures in the trained model. This technique increases the complexity of the model however and therefore increases both the training and prediction times of the algorithm so it might be unsuitable for certain real-time applications. The SVM algorithm would therefore be a better algorithm for a performer to first use to attempt to classify their multivariate temporal musical gestures if they intend to train the algorithm with a substantial number of training examples.

7.5.1.3 Adding & Removing Gestures From A Trained Model

What if a performer has trained a machine learning algorithm with a data set and then decides to either remove a gesture or add a new gesture to their gestural vocabulary? In this scenario, particularly if there are a substantial number of training examples for each gesture, the performer might not want to have to re-train an algorithm from the start but instead simply have to update the model with the gestures that have been changed. In this instance the SVM algorithm might not be the best algorithm to use as, due to the way it is trained and how it handles a multi-class classification problem, the entire model would have to be trained from the start. The ND-DTW algorithm would be more suitable in this scenario as, because each template is separate from every other template in the model, an unwanted gesture would simply need to be deleted from the ND-DTW model or alternatively if a new gesture was to be added then only that gesture template and classification threshold would need to be trained, as opposed to the entire model.

7.5.1.4 Automatic Recognition

What if a performer is unable to use a 'trigger key' due to practical or aesthetic reasons? In this scenario the performer should use the ND-DTW algorithm as it was the only algorithm out of the three multivariate temporal recognition algorithms that achieved a useable classification result when recognising a gesture from a continuous stream of data using the sliding window technique.

7.6 Summary

This chapter has presented the ND-DTW algorithm and validated its ability to classify multivariate temporal musical gestures. The experiments in this chapter showed that the ND-DTW algorithm achieved excellent classification results on the gestures in the numbers-shapes data set when the gestures were pre-segmented and also from a continuous stream of data that also contained null gestures. The experiments also showed that the ND-DTW algorithm could achieve excellent classification results with a limited number of training examples. The chapter was concluded by summarising the advantages and disadvantages of all the multivariate temporal recognition algorithms described throughout this thesis, suggesting scenarios where a performer may want to favor one algorithm over another. The next and final chapter concludes this thesis, summarising the work described throughout the thesis and highlighting the contributions achieved within it.

Chapter 8

Conclusion

The important thing is not to stop questioning.
Curiosity has its own reason for existing.

Albert Einstein

The work presented in this thesis has set out to investigate how machine learning can be applied to the recognition of musical gestures. Whereas the majority of previous work in the recognition of musical gestures has focused on the continuous mapping of a movement to a sound or control parameter; the work in this thesis has focused on the discrete classification of a musical gesture. The main research question of this thesis was: ***how can machine learning be applied to the recognition of musical gestures?*** The work in this thesis has shown that by adopting a machine learning approach a musician can quickly train a computer to accurately recognise even complex multivariate temporal gestures. However, to achieve this required a paradigm shift from the common design, development, training and evaluation strategies applied throughout many areas of machine learning, in HCI and beyond.

The main objectives of this thesis were to:

1. Evaluate what differences, if any, there are between the application of machine learning for the recognition of musical gestures from that of the recognition of other gestures used throughout various fields within human-computer interaction.
2. Test the applicability of the machine learning algorithms that are potentially the most appropriate for the recognition of musical gestures.
3. Once existing algorithms have been evaluated, to develop new algorithms specifically for musician-computer interaction should they be required.

4. Develop software tools that can facilitate real-time musician-computer interaction for any user; even those with no knowledge of machine learning or who have limited computer skills.

8.0.1 Objective 1

This thesis has proposed that the application of machine learning for the recognition of musical gestures requires a paradigm shift from the traditional training, testing, deployment and evaluation strategies used throughout other areas of HCI. Chapter 3 proposed that the goal of a gesture recognition system for MCI, particularly in a live performance scenario, should be to achieve a low intra-personal generalisation error, as opposed to the low inter-personal generalisation error goal that is common in other areas of HCI. This user-specific generalisation goal was proposed due to the three main aspects that differentiate a gesture recognition system for MCI from that within other areas of HCI - namely, the ambiguous input to a recognition system, the uncertainty of what the recognition system will be controlling combined with the user-specific gestural vocabularies that can be found throughout MCI. The requirement for a user-configurable recognition system for MCI therefore inspired the design criteria for a recognition system that should have a middleware design architecture (and thus not be dependent on any one piece of audio hardware or software), be completely user-configurable and also have the functionality to be quickly trained with examples from the user's own gestural vocabulary.

8.0.2 Objective 2 & 3

The design, development and deployment strategies for a gesture recognition system for MCI outlined in chapter 3 also inspired us to test the intra-generalisation abilities of the existing classification algorithms in the machine learning literature. It was found that many of the existing classification algorithms did not meet the requirements set out in chapter 3.2, i.e. they could not work with any N-dimensional signal, could not classify a gesture from a continuous stream of data that also contained non-gestural data or could not be trained with a limited number of training examples. A large part of the work in this thesis has therefore been to extend a number of existing algorithms, such as Hidden Markov Models, Support Vector Machines and Dynamic Time Warping, to enable the algorithms to be used for MCI. As an existing algorithm that enabled the adaptive classification of semiotic musical gestures could not be found, a novel algorithm called the Adaptive Naïve Bayes Classifier was developed which can be found in chapter 4.

The experiments documented in chapters 4 to chapters 7, have shown some interesting results for MCI. The results of the Air Makoto study in chapter 4 suggest that the classification abilities of the ANBC algorithm are significantly improved when the adaptive online training function is used. The classification results of the multivariate temporal recognition algorithms in chapters 5, 6 and 7 have shown that all three modified algorithms achieved excellent classification results when a gesture has been pre-segmented, with N-Dimensional Dynamic Time Warping and Support Vector Machines both achieving an ACVR result of 99%. The ND-DTW algorithm was also successful at classifying a number of multivariate temporal gestures from a continuous stream of data that contained null gestures. This is a significant achievement as the challenge of 'gesture spotting' is one area of research across many areas within HCI that has seen little progress despite numerous attempts at solving the problem (Junker et al., 2008b). The addition of a classification threshold to all of the algorithms presented throughout this thesis has therefore been our contribution to solving a problem which has particular relevance for the classification of temporal gestures for MCI within the context of a live performance context.

8.0.3 Objective 4

Finally, the SEC and all of the algorithms encapsulated within it has provided a novel software tool that enables a musician, regardless of their technical skills or prior knowledge of machine learning, to use a number of powerful machine learning algorithms to recognise their musical gestures in real-time.

8.1 Research Contributions

This thesis has made the following contributions to the field of musician-computer interaction:

- *Definition of Musician-Computer Interaction*

  This thesis defined the term **musician-computer interaction** as a specific subfield of the larger research area of HCI. MCI was defined to help differentiate the design and research goals for musical interaction from that of other areas of HCI. This is because music provides a number of interesting research and design challenges over and above the more general field of HCI as, due to its real-time musical application, a low-latency highly robust user-configurable system is required. This definition can be found in chapter 3.1.

- *Adopting A Machine Learning Approach For MCI*

  This thesis presented the motivations for a performer to adopt a machine learning approach to enable the automatic recognition of musical gestures to be used for MCI. Chapter 3 proposed that the application of machine learning for the recognition of musical gestures requires a paradigm shift from the common training, testing, deployment and evaluation strategies used throughout other areas of HCI that also use gesture recognition. Evidence was presented that the goal of a gesture recognition system for MCI should be to achieve a low intra-personal generalisation error, as opposed to the inter-personal generalisation error goal that is common in other areas of HCI. The MCI machine learning design, development and deployment strategies were then applied to create a design criteria for a recognition system for musical gestures which in turn provided the motivation for a new software tool to enable the recognition of discrete gestures for MCI. These arguments can be found in chapter 3.1.3.

- *Recognition of Static Musical Gestures*

  This thesis presented a novel algorithm called the **Adaptive Naïve Bayes Classifier** (**ANBC**) which can be used for the recognition of static musical gestures. The ANBC algorithm has five significant advantages for the classification of static musical gestures:

  1. The input to the ANBC algorithm consists of an N-dimensional vector, resulting that the input to the algorithm has not been constrained to only work with one type of sensor, such as a mouse or camera.
  2. An N-dimensional weighting coefficients vector enables the user to specify which of the N dimensions of input data are salient for a particular gesture. This enables one general classifier to be used in scenarios were several classifiers would have been required.
  3. The ANBC algorithm can be rapidly trained with a small number of training examples.
  4. The ANBC algorithm can be used to recognise static musical gestures in a continuous stream of data that may also contain non-gestural data without having to first train a null-model, such as a noise model that is used in speech recognition.
  5. The ANBC algorithm can automatically adapt itself as a performer adapts their own gestures over, for example, the course of a rehearsal period.

  The ANBC algorithm can be found in chapter 4.

- *Recognition of Multivariate Temporal Musical Gestures*

  This thesis has made some significant contributions to the discrete classification of multivariate temporal gestures for both MCI and for the wider HCI community. Three existing machine learning algorithms have been specifically adapted for the recognition of musical gestures. Each algorithm has been extended to:

  1. Classify any N-dimensional signal.
  2. Be rapidly trained with a small number of training examples.
  3. Recognise temporal musical gestures in a continuous stream of data that may also contain non-gestural data without having to first train a null-model.

  These machine learning algorithms included **Hidden Markov Models**, **Support Vector Machines** and **Dynamic Time Warping**. Each algorithm was tested using a specifically captured data set of temporal musical gestures with all three modified algorithms achieving excellent recognition results, some even achieving 100% recognition for specific participants and 99% on average across all ten participants. The advantages and disadvantages of each algorithm have been summarized for their potential application in MCI. The algorithms, results and summaries can be found in chapters 5 to 7.

- *A Flexible Gesture Recognition System For MCI*

  One of the major contributions of this thesis is that all of the algorithms developed, design concepts proposed and training strategies suggested have all been encapsulated within the SEC, a highly flexible and user-configurable machine learning toolbox that enables composers, performers and researchers to actually use this work to recognise their musical gestures. The algorithms within the SEC, which was presented in chapter 3.3, have been designed to operate independently from any one specific piece of sensor device or audio software. The SECs' main advantage for MCI is that it enables any performer, regardless of their programming abilities, to quickly train a computer to recognise their musical gestures using a number of powerful machine learning algorithms. The flexible middleware design of the SEC facilitates even a performer with the most rudimentary technical skills to quickly adapt an example patch that has been designed using the two-dimensional input of a mouse for example and replace this with the N-dimensional input of the user's own sensor device. The user can then quickly retrain the machine learning algorithm at the core of the adapted patch and then use this trained model to recognise their musical gestures. The modular design of each pre-processing function, machine learning algorithm and post-processing function enables a user with a higher level of technical skills or knowledge of machine learning to fully

customise their own recognition system. EyesWeb also enables researchers or performers with more advanced technical skills to create their own blocks if they want to test a new form of feature extraction or recognition algorithm and quickly integrate this with some of the SEC or EyesWeb blocks. This is a useful feature for the application of the SEC blocks in future research as a number of researchers may want to test the applicability of a new feature extraction method as input to a standard machine learning algorithm, such as a Hidden Markov Model or Support Vector Machine. EyesWeb and the SEC facilitates a researcher to quickly test their new feature extraction algorithm in a real-time scenario without having to first create an online recognition system or develop common machine learning algorithms such as an HMM or SVM from scratch. The modular design of each algorithm facilitates a researcher to integrate their own feature extraction algorithm as an individual block within EyesWeb, which can then be easily connected to any of the SEC machine learning algorithms.

8.2 Future Research

8.2.1 Continuous Real-Time Recognition

Out of the four algorithms presented in this thesis, only two (the ANBC and ND-DTW algorithms) were successful at recognising a gesture from a continuous stream of real-time data that also contains non-gestural data. The HMM algorithm, for example, achieved an ACCR value of 87.53% on pre-segmented gestures but achieved a poor ACCR value of 44.80% on the same gestures in a continuous stream of data. The SVMs performance was perhaps even more disappointing, as although it achieved an excellent ACVR value of 99.28% on the pre-segmented data, it only achieved an ACCR result of 38.88% on the same gestures in a continuous stream of data. The SVMs main error was in classifying gesture i as a null-gesture (i.e. it had a high recall error), however, the algorithm did achieve an excellent average precision ratio of 0.96 and also achieved an excellent ANRR value of 0.97. These results indicate that the algorithm performed well at rejecting null gestures and also performed well at not misclassifying gesture i as gesture j. It may be the case then that the SVM algorithm *can* classify a number of multivariate temporal gestures from a continuous stream of data *if* it gets enough training data to create a robust generalisable model. This should receive further investigation as the SVM is a powerful recognition algorithm and it would be beneficial for MCI if it can be used for the real-time recognition of multivariate temporal gestures from a continuous stream of data that may also contain null gestures.

8.2.2 Coherent Classification Feedback

One of the key observations commonly found in the real-time application of the algorithms in this thesis through both live performances, installations and experiments is the importance and necessity of *coherent classification feedback*. This is the feedback, such as audio or visual, that informs the user about what the recognition system *thinks* the user is trying, or importantly not trying, to do. This is particularly salient for performer's playing a virtual musical instrument who desire fine-grain control as they lack the common forms of feedback and reference points, such as frets, strings or keys, that can be found in most digital and acoustic musical instruments. Kratz and Ballagas (2009) found, for example, that simply by adding visual feedback (in the form of a filtered waveform representing the user's most recent gesture) improved recognition rates by over 34%. Recognition rates improved by such a significant amount because the visual feedback enabled the users to understand any sensor inaccuracies or uncertainties in the recognition system. This feedback then allowed the user to adapt their movements to overcome any inaccuracies or uncertainties in the recognition system.

8.3 Concluding Remarks

The predominant driving motivation for the work in this dissertation has been the goal of making gestural interaction as viable an interaction paradigm between a musician and a computer as it currently is between two performers. It was shown that, by adopting a machine learning approach, a musician can quickly teach a machine to recognise their musical gestures in the same way a performer might teach a new musician in an ensemble the gestural vocabulary used by that group. By adopting a machine learning approach a musician can simply 'show a computer' a number of examples of the movement the performer wants to make and instruct the computer as to what should be triggered, controlled or manipulated with that gesture. Although the work in this thesis has only scratched the surface of the automatic recognition of musical gestures it has made some significant contributions towards the use of gestures as an interaction medium between a musician and a computer. Combining this work with the abundance of cheap novel sensors and real-time audio software leads us towards exciting new interaction paradigms facilitating a performer to use their own body movements to control a machine even when their hands are busy playing an instrument.

Bibliography

Abe, S. (2005). *Support vector machines for pattern classification.* Springer-Verlag New York Inc.

Abou-Moustafa, K. T., M. Cheriet, and C. Y. Suen (2004). On the structure of hidden markov models. *Pattern Recognition Letters 25*(8), 923–931.

Al-Muhtaseb, H. A., S. A. Mahmoud, and R. S. Qahwaji (2008). Recognition of off-line printed arabic text using hidden markov models. *Signal Processing 88*(12), 2902 – 2912.

Al-Rajab, M., D. Hogg, and K. Ng (2008). A comparative study on using zernike velocity moments and hidden markov models for hand gesture recognition. *Articulated Motion and Deformable Objects*, 319–327.

Angesleva, J., S. O'Modhrain, I. Oakley, and S. Hughes (2003). Body mnemonics. In *In: Mobile HCI Conference 2003.*

Awaidah, S. M. and S. A. Mahmoud (2009). A multiple feature/resolution scheme to arabic (indian) numerals recognition using hidden markov models. *Signal Processing 89*(6), 1176 – 1184.

Babu, R. V., S. Suresh, and A. Makur (2010). Online adaptive radial basis function networks for robust object tracking. *Computer Vision and Image Understanding 114*(3), 297 – 310.

Bartlett, J. F. (2000, may/jun). Rock 'n' scroll is here to stay [user interface]. *Computer Graphics and Applications, IEEE 20*(3), 40 –45.

Baum, L. and J. Eagon (1967). An inequality with applications to statistical estimation for probabilistic functions of markov processes and to a model for ecology. *Bull. Amer. Math. Soc 73*(3), 360–363.

Baum, L. and G. Sell (1968). Growth transformations for functions on manifolds. *Pacific Journal of Mathematics 27*(2), 211–227.

Baum, L. E., T. Petrie, G. Soules, and N. Weiss (1970). A maximization technique occurring in the statistical analysis of probabilistic functions of markov chains. *The Annals of Mathematical Statistics 41*(1), 164–171.

Benbasat, A. and J. Paradiso (2002). An inertial measurement framework for gesture recognition and applications. In I. Wachsmuth and T. Sowa (Eds.), *Gesture and Sign Language in Human-Computer Interaction*, Volume 2298 of *Lecture Notes in Computer Science*, pp. 77–90. Springer Berlin / Heidelberg.

Benzeghiba, M., R. D. Mori, O. Deroo, S. Dupont, T. Erbes, D. Jouvet, L. Fissore, P. Laface, A. Mertins, C. Ris, R. Rose, V. Tyagi, and C. Wellekens (2007). Automatic speech recognition and speech variability: A review. *Speech Communication 49*(10-11), 763 – 786. Intrinsic Speech Variations.

Bertini, G. and P. Carosi (1993). Light baton system: A system for conducting computer music performance. *Journal of New Music Research 22*(3), 243–257.

Bettens, F. and T. Todoroff (2009). Real-time dtw-based gesture recognition external object for max/msp and puredata. *Proceedings of SMC 2009 - 6th Sound and Music Computing Conference, 23-25 july 2009, Porto - Portugal*.

Bevilacqua, F., F. Guédy, N. Schnell, E. Fléty, and N. Leroy (2007). Wireless sensor interface and gesture-follower for music pedagogy. In *NIME '07: Proceedings of the 7th international conference on New interfaces for musical expression*, New York, NY, USA, pp. 124–129. ACM.

Bevilacqua, F., R. Muller, and N. Schnell (2005). Mnm: a max/msp mapping toolbox. In *Proceedings of the 2005 International Conference on New Interfaces for Musical Expression (NIME05), Vancouver, BC, Canada*.

Bevilacqua, F., B. Zamborlin, A. Sypniewski, N. Schnell, F. Guédy, and N. Rasamimanana (2009). Continous realtime gesture following and recognition. *Lecture Notes in Cimputer Science (LNCS), Gesture Embodied Communication and Human-Computer Interaction*.

Bishop, C. M. (1995). *Neural Networks for Pattern Recognition*. Oxford: Oxford University Press.

Bishop, C. M. (2006). *Pattern recognition and machine learning*. Science and Business Media, Springer.

Bolt, R. A. (1980, July). "put-that-there": Voice and gesture at the graphics interface. *SIGGRAPH Comput. Graph. 14*, 262–270.

Bradshaw, D. and K. Ng (2008). Analyzing a conductors gestures with the wiimote. In *Proceedings of EVA London 2008: the International Conference of Electronic Visualisation and the Arts*, pp. 22–24.

Brecht, B. and G. Garnett (1995). Conductor follower. In *Proceddings of the 21st. International Computer Music Conference (ICMC), Baniff Canada.*

Bruegge, B., C. Teschner, P. Lachenmaier, E. Fenzl, D. Schmidt, and S. Bierbaum (2007). Pinocchio: conducting a virtual symphony orchestra. In *Proceedings of the international conference on Advances in computer entertainment technology*, ACE '07, New York, NY, USA, pp. 294–295. ACM.

Burges, C. J. C. (1998). A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery 2*, 121–167.

Cadoz, C. (1988). Instrumental gesture and musical composition. In *Proceedings of the International Computer Music Conference*, pp. 1–12.

Cadoz, C. and M. Wanderley (2000). Gesture-music. *Trends in Gestural Control of Music*, 71–93.

Camurri, A., P. Coletta, A. Massari, B. Mazzarino, M. Peri, M. Ricchetti, A. Ricci, and G. Volpe (2004). Toward real-time multimodal processing: Eyesweb 4.0. In *Proc. AISB.*

Camurri, A., P. Coletta, G. Varni, and S. Ghisio (2007). Developing multimodal interactive systems with eyesweb xmi. In *NIME07*, pp. 305–308. ACM.

Camurri, A., B. Mazzarino, M. Ricchetti, R. Timmers, and G. Volpe (2004). Multimodal analysis of expressive gesture in music and dance performances. *Gesture-based communication in human-computer interaction*, 357–358.

Camurri, A., G. Volpe, G. De Poli, and M. Leman (2005, jan.-march). Communicating expressiveness and affect in multimodal interactive systems. *Multimedia, IEEE 12*(1), 43 – 53.

Chang, C. and C. Lin (2001). *LIBSVM: A library for support vector machines.* Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm.

Chen, F. S., C. M. Fu, and C. L. Huang (2003). Hand gesture recognition using a real-time tracking method and hidden markov models. *Image and Vision Computing 21*(8), 745–758.

Cho, S., Y. Sung, R. Murray-Smith, K. Lee, C. Choi, and Y. Kim (2007). Dynamics of tilt-based browsing on mobile devices. *CHI*.

Christensen, C. (1968). An example of the manipulation of directed graphs in the ambit/g programming language. *Interactive Systems for Experimental Applied Mathematics*.

Cont, A., T. Coduys, and C. Henry (2004). Real-time gesture mapping in pd environment using neural networks. In *Proceedings of the 2004 Conference on New Interfaces for Musical Expression (NIME04), Hamamatsu, Japan.*

Cowie, R., E. Douglas-Cowie, N. Tsapatsoulis, G. Votsis, S. Kollias, W. Fellenz, and J. Taylor (2001, January). Emotion recognition in human-computer interaction. *IEEE Signal Processing Magazine*, 32–80.

Delalande, F. (1988). La gestique de gould. *Glenn Gould Pluriel*, 85–111.

Dillon, R., G. Wong, and R. Ang (2006). Virtual orchestra: An immersive computer game for fun and education. In *Proceedings of the 2006 international conference on Game research and development*, CyberGames '06, Murdoch University, Australia, Australia, pp. 215–218. Murdoch University.

Ding, H., G. Trajcevski, P. Scheuermann, X. Wang, and E. Keogh (2008). Querying and mining of time series data: Experimental comparison of representations and distance measures. *Proceedings of the VLDB Endowment 1*(2), 1542–1552.

Dipietro, L., A. M. Sabatini, and P. Dario (2008, july). A survey of glove-based systems and their applications. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on 38*(4), 461 –482.

Domingos, P. and M. Pazzani (1997). On the optimality of the simple bayesian classifier under zero-one loss. *Machine Learning 29*, 103–130.

Duda, R. O., P. E. Hart, and D. G. Stork (2001). *Pattern classification.* Citeseer.

Ellis, T. O., J. F. Heafner, W. L. Sibley, and R. C. S. M. CALIF. (1969). *The grail project: An experiment In man-machine communications.* Rand.

Ergovic, V., S. Tonkovic, and V. Medved (2009). Human gait data mining by symbol based descriptive features. In *World Congress on Medical Physics and Biomedical Engineering, September 7-12, 2009, Munich, Germany*, pp. 460–463. Springer.

Erol, A., G. Bebis, M. Nicolescu, R. D. Boyle, and X. Twombly (2007). Vision-based hand pose estimation: A review. *Computer Vision and Image Understanding 108*(1-2), 52 – 73. Special Issue on Vision for Human-Computer Interaction.

Eslambolchilar, P., J. Williamson, and R. Murray-Smith (2004). Multimodal feedback for tilt controlled speed dependent automatic zooming. *ACM Symposium on User Interface Software and Technology*.

Farella, E., S. O'Modhrain, L. Benini, and B. Ricco (2006). Gesture signature for ambient intelligence applications: A feasibility study. *PERVASIVE 2006*.

Fels, S. (1995). Glove-talkii: A neural network interface which maps gestures to parallel formant speech synthesizer controls. *CHI'95*.

Fiebrink, R. (2011). *Real-time human interaction with supervised learning algorithms for music composition and performance*. Ph. D. thesis, School of Computer Science, Princeton University.

Fiebrink, R., P. R. Cook, and D. Trueman (2009). Play-along mapping of musical controllers. *The 2009 International Computer Music Conference (ICMC)*.

Fiebrink, R., D. Trueman, and P. R. Cook (2009). A meta-instrument for interactive, on -the-fly machine learning. *NIME09*.

Fitz-Walter, Z., S. Jones, and D. Tjondronegoro (2008). Detecting gesture force peaks for intuitive interaction. In *Proceedings of the 5th Australasian Conference on Interactive Entertainment*, IE '08, New York, NY, USA, pp. 2:1–2:8. ACM.

Forbes, K. and E. Fiume (2005). An efficient search algorithm for motion data using weighted pca. In *Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pp. 76. ACM.

Freeman, W. and C. Weissman (1995). Television control by hand gestures. In *Proc. of Intl. Workshop on Automatic Face and Gesture Recognition*, pp. 179–183. Citeseer.

Friberg, A. (2005). Home conducting: Control the overall musical expression with gestures. In *Proceedings of the 2005 International Computer Music Conference*, pp. 479–482.

Gibet, S. (1987). *Codage, représentation et traitement du geste instrumental: Application à la synthèse de sons musicaux par simulation de mécanismes instrumentaux*. Ph. D. thesis, Institut National Polytechnique de Grenoble, Institut de Mathématiques Appliquées de Grenoble.

Gillian, N., R. B. Knapp, and S. O'Modhrain (2011a). An adaptive classification algorithm for semiotic musical gestures. In *the 8th Sound and Music Computing Conference (SCM2011), Padova, Italy*.

Gillian, N., R. B. Knapp, and S. O'Modhrain (2011b). A machine learning toolbox for musician computer interaction. In *Proceedings of the 2011 International Coference on New Interfaces for Musical Expression (NIME11), Oslo, Norway*.

Gillian, N., R. B. Knapp, and S. O'Modhrain (2011c). Recognition of multivariate temporal musical gestures using n-dimensional dynamic time warping. In *Proceedings of the 2011 International Coference on New Interfaces for Musical Expression (NIME11), Oslo, Norway.*

Gillian, N., S. O'Modhrain, and G. Essl (2009). Scratch-off: A gesture based mobile music game with tactile feedback. In *Proceedings of the 2009 Conference on New Interfaces for Musical Expression (NIME09), Pittsburgh, USA.*

Hall, M., E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten (2009). The weka data mining software: An update. *ACM SIGKDD Explorations Newsletter 11*(1), 10–18.

Harrison, B. L., K. P. Fishkin, A. Gujar, C. Mochon, and R. Want (1998). Squeeze hold tilt me! an exploration of manipulative user interfaces. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, CHI '98, New York, NY, USA, pp. 17–24. ACM Press/Addison-Wesley Publishing Co.

Hashimoto, S. (1997). Kansei as the third target of information processing and related topics in japan. In *Proceedings of the International Workshop on "KANSEI: The technology of emotion", AIMI (Italian Computer Music Association) and DIST-University of Genova, pp101-104.*

Heloir, A., N. Courty, S. Gibet, and F. Multon (2006). Temporal alignment of communicative gesture sequences. *Computer Animation and Virtual Worlds 17*(3-4), 347.

Hinckley, K., J. Pierce, M. Sinclair, and E. Horvitz (2000). Sensing techniques for mobile interaction. In *Proceedings of the 13th annual ACM symposium on User interface software and technology*, UIST '00, New York, NY, USA, pp. 91–100. ACM.

Hofer, A., A. Hadjakos, and M. Muhlhauser (2009). Gyroscope-based conducting gesture recognition. *In Proceedings of the 2009 Conference on New Interfaces for Musical Expression (NIME09).*

Hoffman, M., P. Varcholik, and J. J. LaViola (2010). Breaking the status quo: Improving 3d gesture recognition with spatially convenient input devices. In *Virtual Reality Conference (VR), 2010 IEEE*, pp. 59–66. IEEE.

Ilmonen, T. (2003). Tools and experiments in multimodal interaction. *System 11*(2), 240–247.

Inoue, T., R. Nakagawa, M. Kondou, T. Koga, and K. Shinohara (2011). Discrimination between mothers' infant-and adult-directed speech using hidden markov models. *Neuroscience Research.*

Itakura, F. (1990). Minimum prediction residual principle applied to speech recognition. *Readings in speech recognition*, 154.

Jensenius, A. R., M. M. Wanderley, R. I. Godøy, and M. Leman (2009). *Gestural affordances of musical sound*. Routledge.

Junker, H., O. Amft, P. Lukowicz, and G. Troster (2008a). Gesture spotting with body-worn inertial sensors to detect user activities. *Pattern Recognition 41*(6), 2010–2024.

Junker, H., O. Amft, P. Lukowicz, and G. Troster (2008b). Gesture spotting with body-worn inertial sensors to detect user activities. *Pattern Recognition 41*(6), 2010–2024.

Just, A. and S. Marcel (2009). A comparative study of two state-of-the-art sequence processing techniques for hand gesture recognition. *Computer Vision and Image Understanding 113*(4), 532–543.

Karam, M. (2006, November). *A framework for research and design of gesture-based human computer interactions*. Ph. D. thesis, Faculty of Engineering, Science and Mathematics, School of Electronics and Computer Science.

Kasten, E. P., P. K. McKinley, and S. H. Cage (2007). Automated ensemble extraction and analysis of acoustic data streams. *Proceedings of the First International Workshop on Distributed Event Processing, Systems and Applications (DEPSA), in conjunction with ICDCS 2007, Toronto, Ontario Canada, June*.

Keir, P., J. Payne, J. Elgoyhen, M. Horner, M. Naef, and P. Anderson (2006, march). Gesture-recognition with non-referenced tracking. In *3D User Interfaces, 2006. 3DUI 2006. IEEE Symposium on*, pp. 151 – 158.

Kendon, A. (2004). *Gesture: Visible action as utterance*. Cambridge Univ Pr.

Keogh, E. and C. A. Ratanamahatana (2005). Exact indexing of dynamic time warping. *Knowledge and Information Systems 7*(3), 358–386.

Keogh, E. J. and M. J. Pazzani (2000). Scaling up dynamic time warping for datamining applications. In *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 285–289. ACM.

Keogh, E. J. and M. J. Pazzani (2001). Derivative dynamic time warping. In *First SIAM international conference on data mining*. Citeseer.

Kim, J., S. Mastnik, and E. André (2008). Emg-based hand gesture recognition for realtime biosignal interfacing. *Proc ACM IUI 08, 30 39*.

Ko, M., G. West, S. Venkatesh, and M. Kumar (2008). Using dynamic time warping for online temporal fusion in multisensor systems. *Information Fusion 9*(3), 370–388.

Kolesnik, P. and M. Wanderley (2004). Recognition, analysis and performance with expressive conducting gestures. In *Proceedings of the International Computer Music Conference*.

Kong, A., D. Zhang, and M. Kamel (2009). A survey of palmprint recognition. *Pattern Recognition 42*(7), 1408 – 1418.

Kratz, L., M. Smith, and F. J. Lee (2007). Wiizards: 3d gesture recognition for game play input. In *Proceedings of the 2007 conference on Future Play*, Future Play '07, New York, NY, USA, pp. 209–212. ACM.

Kratz, S. and R. Ballagas (2009). Unravelling seams: improvoing mobile gesture recognition with visual feedback techniques. In *Proceedings of the 27th international conference on Human factors in computing systems*, CHI '09, New York, NY, USA, pp. 937–940. ACM.

Kratz, S. and M. Rohs (2010). The $3 recognizer: simple 3d gesture recognition on mobile devices. In *Proceeding of the 14th international conference on Intelligent user interfaces*, IUI '10, New York, NY, USA, pp. 419–420. ACM.

Kurtenbach, G. and E. Hulteen (1990, May). *The Art and science of interface design*. Gestures in Human-Computer Communication. Addison-Wesley Publishing Co.

Lee, J. C. (2008, July). Hacking the nintendo wii remote. *IEEE Pervasive Computing 7*, 39–45.

Lee, M., A. Freed, and D. Wessel (1992). Neural networks for simultaneous classification and parameter estimation in musical instrument control. *Adaptive Learning Systems 1706*, 244–255.

Lemire, D. (2009). Faster retrieval with a two-pass dynamic-time-warping lower bound. *Pattern Recognition 42*(9), 2169–2180.

Leong, T. S., J. Lai, J. Panza, P. Pong, and J. Hong (2009). Wii want to write: An accelerometer based gesture recognition system. In *International Conference on Recent and Emerging Advanced Technologies in Engineering*.

Li, Y. (2010). Protractor: A fast and accurate gesture recognizer. In *Proceedings of the 28th international conference on Human factors in computing systems*, CHI '10, New York, NY, USA, pp. 2169–2172. ACM.

Li, Y. and R. Anderson-Sprecher (2006). Facies identification from well logs: A comparison of discriminant analysis and naïve bayes classifier. *Journal of Petroleum Science and Engineering 53*(3-4), 149 – 157.

Licsar, A. and T. Sziranyi (2005). User-adaptive hand gesture recognition system with interactive training. *Image and Vision Computing 23*(12), 1102 – 1114.

Lin, H., C. Lin, and R. Weng (2007). A note on platt's probabilistic outputs for support vector machines. *Machine Learning 68*(3), 267–276.

Lin, J., E. Keogh, S. Lonardi, and B. Chiu (2003). A symbolic representation of time series with implications for streaming algorithms. *In Proceedings of the 8th ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*.

Lin, J., E. Keogh, L. Wei, and S. Lonardi (2007). Experiencing sax: A novel symbolic representation of time series. *Data Mining and Knowledge Discovery 15*, 107–144.

Liu, J., L. Zhong, J. Wickramasuriya, and V. Vasudevan (2009). uwave: Accelerometer-based personalized gesture recognition and its applications. *Pervasive and Mobile Computing 5*(6), 657 – 675. PerCom 2009.

Liu, M. (2010). Fingerprint classification based on adaboost learning from singularity features. *Pattern Recognition 43*(3), 1062 – 1070.

Lu, S., D. Chiang, H. Keh, and H. Huang (2010). Chinese text classification by the naïve bayes classifier and the associative classifier with multiple confidence threshold values. *Knowledge-Based Systems*.

MacQueen, J. (1967). Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, Number 281-297 in 1, pp. 14. California, USA.

Marrin, T. (1996). *Toward an understanding of musical gesture: Maping expressive intention with the digital baton*. Ph. D. thesis, Massachusetts Institute of Technology.

Mathews, M. V. (1991a). The radio baton and conductor program, or: Pitch, the most important and least expressive part of music. *Computer Music Journal*, 37–46.

Mathews, M. V. (1991b). The radio baton and conductor program, or: Pitch, the most important and least expressive part of music. *Computer Music Journal*, 37–46.

McNeill, D. (1992). *Hand and mind*. University of Chicago Press.

McNeill, D. (2000). *Language and gesture*. Cambridge University Press Cambridge, UK.

Merrill, D. J. and J. A. Paradiso (2005). Personalization, expressivity, and learnability of an implicit mapping strategy for physical interfaces. *Proceedings of CHI 2005 Conference on Human Factors in Computing Systems*.

Mitra, S. and T. Acharya (2007). Gesture recognition: A survey. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on 37*(3), 311–324.

Modler, P., T. Myatt, and M. Saup (2003). An experimental set of hand gestures for expressive control of musical parameters in realtime. In *Proceedings of the 2003 conference on New interfaces for musical expression*, NIME '03, Singapore, Singapore, pp. 146–150. National University of Singapore.

Moeslund, T. B., A. Hilton, and V. Kruger (2006). A survey of advances in vision-based human motion capture and analysis. *Computer Vision and Image Understanding 104*(2-3), 90 – 126. Special Issue on Modeling People: Vision-based understanding of a person's shape, appearance, movement and behaviour.

Morales-Mazanares, R., E. F. Morales, and D. Wessel (2005). Combining audio and gesture for a real-time improviser. *in International Computer Music Conference, (Barcelona, 2005), ICMA*.

Mori, S., C. Suen, and K. Yamamoto (1992, jul). Historical review of ocr research and development. *Proceedings of the IEEE 80*(7), 1029 –1058.

Morita, H., S. Hashimoto, and S. Ohteru (1991). A computer music system that follows a human conductor. *Computer 24*, 44–53.

Mulder, A. (1994). Virtual musical instruments: Accessing the sound synthesis universe as a performer. In *Proceedings of the First Brazilian Symposium on Computer Music*, pp. 243–250. Citeseer.

Mulder, A. (2000). Towards a choice of gestural constraints for instrumental performers. *Trends in Gestural Control of Music*.

Murphy, D., T. H. Andersen, and K. Jensen (2004). Conducting audio files via computer vision. In *Gesture-Based Communication in Human-Computer Interaction*, Volume 2915 of *Lecture Notes in Computer Science*, pp. 101–102. Springer Berlin / Heidelberg.

Murray-Smith, R. and S. Strachan (2008). Rotional dynamics for design of bidirectional feedback during manual interaction. *Fun and Games*, 1–10.

Nakra, T. M. (1999). *Inside the conductor's jacket: Analysis, interpretation and musical synthesis of expressive gesture*. Ph. D. thesis, Massachusetts Institute of Technology.

Nash, C. and A. Blackwell (2008). Realtime representation and gestural control of musical polytempi. *Proceedings of the 2008 Conference on New Interfaces for Musical Expression (NIME08)*, 28–33.

O'Modhrain, S. (2004). Touch and godesigning haptic feedback for a hand-held mobile device. *BT technology journal 22*(4), 139–145.

Overholt, D., J. Thompson, L. Putnam, B. Bell, J. Kleban, B. Sturm, and J. Kuchera-Morin (2009). A multimodal system for gesture recognition in interactive music performance. *Computer Music Journal 33*(4), 69–82.

Park, C. and S. Lee (2011). Real-time 3d pointing gesture recognition for mobile robots with cascade hmm and particle filter. *Image and Vision Computing 29*(1), 51 – 63.

Patel, K., N. Bancroft, S. M. Drucker, J. Fogarty, A. J. Ko, and J. Landay (2010). Gestalt: integrated support for implementation and analysis in machine learning. In *Proceedings of the 23nd annual ACM symposium on User interface software and technology*, UIST '10, New York, NY, USA, pp. 37–46. ACM.

Patel, S. N., J. S. Pierce, and G. D. Abowd (2004). A gesture-based authentication scheme for untrusted public terminals. In *Proceedings of the 17th annual ACM symposium on User interface software and technology*, UIST '04, New York, NY, USA, pp. 157–160. ACM.

Plamondon, R. and S. Srihari (2000, jan). Online and off-line handwriting recognition: a comprehensive survey. *Pattern Analysis and Machine Intelligence, IEEE Transactions on 22*(1), 63 –84.

Platt, J. C. (1999). Probabilistic outputs for support vector machines and comparison to regularized likelihood methods. *Advances in Large Margin Classifiers*, 61–74.

Press, W. H., S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery (2007). *Numerical recipes: the art of scientific computing.* Cambridge Univ Pr.

Pritchard, B. and S. Fels (2006). Grassp: Gesturally-realized audio, speech and song performance. *Proceedings of the 2006 International Coference on New Interfaces for Musical Expression (NIME06)*, 272–271.

Pylvänäinen, T. (2005). Accelerometer based gesture recognition using continuous hmms. *Pattern Recognition and Image Analysis*, 639–646.

Quek, F. K. H. (1994). Toward a vision-based hand gesture interface. In *Proceedings of the conference on Virtual reality software and technology*, River Edge, NJ, USA, pp. 17–31. World Scientific Publishing Co., Inc.

Rabiner, L. R. (1989). A tutorial on hidden markov models and selected applications in speech recognition. *in Proceedings of the IEEE 77*, 257–286.

Rasamimanana, N., E. Fléty, and F. Bevilacqua (2006). Gesture analysis of violin bow strokes. *Gesture in Human-Computer Interaction and Simulation*, 145–155.

Rehm, M., N. Bee, and E. André (2008). Wave like an egyptian: accelerometer based gesture recognition for culture specific interactions. In *Proceedings of the 22nd British HCI Group Annual Conference on People and Computers: Culture, Creativity, Interaction - Volume 1*, BCS-HCI '08, Swinton, UK, UK, pp. 13–22. British Computer Society.

Rimé, B. and L. Schiaratura (1991). Gesture and speech.

Rish, I. (2001). An empirical study of the naïve bayes classifier. In *IJCAI-01 workshop on "Empirical Methods in AI"*.

Sakoe, H. and S. Chiba (1990). Dynamic programming algorithm optimization for spoken word recognition. *Readings in speech recognition*, 159.

Salvador, S. and P. Chan (2007). Toward accurate dynamic time warping in linear time and space. *Intelligent Data Analysis 11*(5), 561–580.

Savage, N. S., S. R. Ali, and N. E. Chavez (2010). Mmmmm: A multi-modal mobile music mixer. In *Proceedings of the 2010 Conference on New Interfaces for Musical Expression (NIME10), Syndey, Australia.*

Sawada, H. and S. Hashimoto (1997). Gesture recognition using an acceleration sensor and its application to musical performance control. *Electronics and Communications in Japan (Part III: Fundamental Electronic Science) 80*(5), 9–17.

Sebe, N., M. S. Lew, I. Cohen, A. Garg, and T. S. Huang (2002). Emotion recognition using a cauchy naïve bayes classifier. *Pattern Recognition, International Conference on 1*, 10017.

Sreedharan, S., E. S. Zurita, and B. Plimmer (2007). 3d input for 3d worlds. In *Proceedings of the 19th Australasian conference on Computer-Human Interaction: Entertaining User Interfaces*, OZCHI '07, New York, NY, USA, pp. 227–230. ACM.

Stettiner, Y., D. Malah, and D. Chazan (1994). Dynamic time warping with path control and non-local cost. In *Proceedings of the 12th IAPR International Conference on Pattern Recognition, 1994. Vol. 3-Conference C: Signal Processing*, pp. 174–177.

Strachan, S., R. Murray-Smith, and S. O'Modhrain (2007). Bodyspace: Inferring body pose for natural control of a music player. *CHI*.

Sturman, D. J. and D. Zeltzer (2002). A survey of glove-based input. *Computer Graphics and Applications, IEEE 14*(1), 30–39.

Sturman, D. J., D. Zeltzer, and S. Pieper (1989). Hands-on interaction with virtual environments. In *Proceedings of the 2nd annual ACM SIGGRAPH symposium on User interface software and technology*, UIST '89, New York, NY, USA, pp. 19–24. ACM.

Sutherland, I. E. (1964). Sketch pad a man-machine graphical communication system. In *Proceedings of the SHARE design automation workshop*, pp. 6–329. ACM.

Tanaka, A. and R. B. Knapp (2002). Multimodal interaction in music using the electromyogram and relative position sensing. In *NIME '02: Proceedings of the 2002 conference on New interfaces for musical expression*, Singapore, Singapore, pp. 1–6. National University of Singapore.

Tarabella, L. (2005). Handel, a free-hands gesture recognition system. In U. K. Wiil (Ed.), *Computer Music Modeling and Retrieval*, Volume 3310 of *Lecture Notes in Computer Science*, pp. 139–148. Springer Berlin / Heidelberg.

Teitelman, W. (1964). Real time recognition of hand-drawn characters. In *Proceedings of the October 27-29, 1964, fall joint computer conference, part I*, AFIPS '64 (Fall, part I), New York, NY, USA, pp. 559–575. ACM.

ten Holt, G. A., M. J. T. Reinders, and E. A. Hendriks (2007). Multi-dimensional dynamic time warping for gesture recognition. In *Thirteenth annual conference of the Advanced School for Computing and Imaging*.

Turner, D. (2007). The nintendo wii: A game console with underwhelming graphics wins with neat controllers. *TECHNOLOGY REVIEW-MANCHESTER NH- 110*(4), 22.

Vlachos, M., M. Hadjieleftheriou, D. Gunopulos, and E. Keogh (2003). Indexing multidimensional time-series with support for multiple distance measures. *Proceedings of the 9th ACM SIGKDD int. conf. on Knowledge discovery and data mining*.

Vuori, V., J. Laaksonen, E. Oja, and J. Kangas (2001). Experiments with adaptation strategies for a prototype-based recognition system for isolated handwritten characters. *International Journal on Document Analysis and Recognition 3*, 150–159.

Wachs, J. P., M. Kölsch, H. Stern, and Y. Edan (2011, February). Vision-based hand-gesture applications. *Commun. ACM 54*, 60–71.

Waisvisz, M. (1985). The hands, a set of remote midi-controllers. In *Proceedings of the International Computer Music Conference*, pp. 313–318.

Wanderley, M. and M. Battier (Eds.) (2000). *Trends in gestural control of music*. Ircam - Centre Pompidou.

Wanderley, M. and P. Depalle (2004a). Gestural control of sound synthesis. *Proceedings of the IEEE 92*(4), 632–644.

Wanderley, M. M. (1999). Non-obvious performer gestures in instrumental music. *Gesture-based communication in human-computer interaction*, 37–48.

Wanderley, M. M. and P. Depalle (2004b). Gestural control of sound synthesis. *Proceedings of the IEEE 92*(4), 632–644.

Wang, C., Z. Liu, and S. Fels (2010). Everyone can do magic: An interactive game with speech and gesture recognition. In H. Yang, R. Malaka, J. Hoshino, and J. Han (Eds.), *Entertainment Computing - ICEC 2010*, Volume 6243 of *Lecture Notes in Computer Science*, pp. 32–42. Springer Berlin / Heidelberg.

Ward, J. A., P. Lukowicz, and G. Troster (2005). Gesture spotting using wrist worn microphone and 3-axis accelerometer. In *Proceedings of the 2005 joint conference on Smart objects and ambient intelligence: innovative context-aware services: usages and technologies*, pp. 99–104. ACM.

Wexelblat, A. (1995, September). An approach to natural gesture in virtual environments. *ACM Trans. Comput.-Hum. Interact. 2*, 179–200.

Whitehead, A. and K. Fox (2009). Device agnostic 3d gesture recognition using hidden markov models. In *Proceedings of the 2009 Conference on Future Play on@ GDC Canada*, pp. 29–30. ACM.

Williamson, J., R. Murray-Smith, and S. Hughes (2007). Shoogle: Excitatory multimodal interaction on mobile devices. *Computer/Human Interaction*.

Wilson, A. D. and A. F. Bobick (2000). Realtime online adaptive gesture recognition. In *Pattern Recognition, 2000. Proceedings. 15th International Conference on*, Volume 1, pp. 270 –275 vol.1.

Wobbrock, J. O., A. D. Wilson, and Y. Li (2007). Gestures without libraries, toolkits or training: a $1 recognizer for user interface prototypes. In *Proceedings of the 20th annual ACM symposium on User interface software and technology*, UIST '07, New York, NY, USA, pp. 159–168. ACM.

Wong, S. F. and R. Cipolla (2006). Continuous gesture recognition using a sparse bayesian classifier. *Pattern Recognition 1*, 1084–1087.

Wong, T. and L. Chang (2011). Individual attribute prior setting methods for naïve bayesian classifiers. *Pattern Recognition 44*(5), 1041 – 1047.

Wright, M. and A. Freed (1997). Open sound control: A new protocol for communicating with sound synthesizers. In *International Computer Music Conference*, Thessaloniki, Hellas, pp. 101–104. International Computer Music Association.

Wu, J., G. Pan, D. Zhang, G. Qi, and S. Li (2009). Gesture recognition with a 3-d accelerometer. In D. Zhang, M. Portmann, A.-H. Tan, and J. Indulska (Eds.), *Ubiquitous Intelligence and Computing*, Volume 5585 of *Lecture Notes in Computer Science*, pp. 25–38. Springer Berlin / Heidelberg.

Wullmer, M., M. Al-Hames, F. Eyben, B. Schuller, and G. Rigoll (2009). A multidimensional dynamic time warping algorithm for efficient multimodal fusion of asynchronous data streams. *Neurocomputing*.

Xi, X., E. Keogh, C. Shelton, L. Wei, and C. A. Ratanamahatana (2006). Fast time series classification using numerosity reduction. In *Proceedings of the 23rd international conference on Machine learning*, pp. 1040. ACM.

Yoon, H. S., J. Soh, Y. J. Bae, and H. Seung Yang (2001). Hand gesture recognition using combined features of location, angle and velocity. *Pattern Recognition 34*(7), 1491–1501.

Young, D. (2008). Classification of common violin bowing techniques using gesture data from a playable measurement system. *Proceedings of the 2008 Conference on New Interfaces for Musical Expression (NIME08)*, 44–48.

Zhang, X. and Y. Gao (2009). Face recognition across pose: A review. *Pattern Recognition 42*(11), 2876 – 2896.

Zhao, L. and N. Badler (2001). Synthesis and acquisition of laban movement analysis qualitative parameters for communicative gestures. *University of Pennsylvania, Philadelphia, PA*.

Zhao, W., R. Chellappa, P. J. Phillips, and A. Rosenfeld (2003, December). Face recognition: A literature survey. *ACM Comput. Surv. 35*, 399–458.

Zimmerman, T. G., J. Lanier, C. Blanchard, S. Bryson, and Y. Harvill (1986, May). A hand gesture interface device. *SIGCHI Bull. 17*, 189–192.