BIRD CALL RECOGNITION WITH ARTIFICIAL NEURAL NETWORKS, SUPPORT VECTOR MACHINES, AND KERNEL DENSITY ESTIMATION.

by Derek J. Ross

A thesis submitted to the Faculty of Graduate Studies in partial fulfillment of the requirements for the degree of

Master of Science

Department of Electrical and Computer Engineering
University of Manitoba
Winnipeg, Canada

I hereby declare that I am the sole author of this thesis.

I authorize the University of Manitoba to lend this thesis to other institutions of individuals for the purpose of scholarly research.

I also authorize the University of Manitoba to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

Derek J. Ross 2006

The University of Manitoba requires the signatures of all persons using or photocopying this thesis. Please sign below, and give your address and the date.

Acknowledgments

First, I would like to thank my advisor Howard Card for skillfully elucidating the inner workings of the concepts of pattern recognition. I would also like to thank my secondary adviser, Dean McNeill, for carrying the torch after Howard went on sabbatical, and for providing valuable advice that helped get this thesis over the finish line. Thank you also to thesis committee members Dr. P. Yahampath and Dr. J. Anderson for their comments and suggestions.

I thank my wife Rowena and children Darwin and Felix, for their patience and support while I spent seemingly countless hours working on this degree.

Thank you to Alex McIlraith for his knowledge of pattern recognition, statistics, and biology, and for practical advice on the art of thesis writing. Thanks go to Rob Berger for his expertise of all things feathered, and a big thank you to both Rob and Alex for providing the audio data used in this project, courtesy of their company, Myrica Systems. Thanks also to Brad Brown of Iders Inc., for giving me some time off to pursue this degree, and for useful advice regarding this work.

Finally, I'd like to thank many friends who have had an impact on this thesis. Gord McGonigal, Cam Mayor, Christian Gan, Andrew McKay, Steve Dueck, Shamir Mukhi, Shelley Gagne, Tam Lam, Richard Burchill, and Chuck Leibert, have all made special contributions.

Abstract

This thesis evaluates artificial neural networks (ANNs), support vector machines (SVMs), and kernel density estimation of probability (KDE) on the task of classifying ten species of birds from audio recordings of their calls.

This project had two primary goals. The first goal was to determine if short-term tonal qualities are adequate for distinguishing bird species. Past research into bird recognition has concentrated on long-term or global characteristics of bird calls, as opposed to short-term qualities.

The second goal was to compare the performance of the three aforementioned pattern recognition algorithms. ANNs have been used for bird recognition in past research, but SVMs and KDE have not been studied in this context.

Recordings were first processed to extract short-term features based on spectral, cepstral, and amplitude characterstics — global features were ignored. Consideration was given to features that would be more resistant to environmental noise.

Three classifiers were trained to recognize a species based on audio recordings that had been separated into frames of 512 samples each. With ANN and SVM, silence and noise frames were rejected by setting a high discrimination threshold, which was determined by finding the optimal point on the receiver operating characteristics (ROC) curve. A discrimination threshold proved problematic with the KDE classifier and was not used.

Recordings from the cross-validation (CV) set were tested by classifying each of the frames as a species, and then processing the collection of votes to determine the likely species of the recording. Two postprocessing methods were used.

The first method, simple voting, counted the number of times each species was selected by a classifier. The species which was most frequently selected was considered to be the winner, and became the species estimate for the entire call. The second method used the chi-squared goodness-of-fit test to match the "confusion row" for a recording to a row in the overall confusion matrix. The row with the lowest χ^2 determined the species.

Both methods gave similar average accuracy results, but the chi-test raised the score of the worst

performing species, in some cases, by significant amounts, and also reduced the variance of accuracy across species. The best average accuracy on the CV set was exhibited by an ANN with 100 hidden neurons, with a score of 82% and an accuracy floor of 46%. A figure of merit consisting of the geometric mean of the average CV accuracy and the CV accuracy floor was used to better evaluate performance. Using this metric, one of the three SVM implementations was the best, with an average CV accuracy of 79% and a floor of 63%. KDE performance was comparable to an ANN with 20 hidden neurons.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Since prehistoric times, people have interacted with birds. They have long been utilised as a source of food. After the invention of agriculture, they were often seen as pests competing for crop resources. The relationship has continued to evolve ever since.

As humanity and technology spreads across the face of the Earth, interactions, both negative and positive, between birds and people grow. In recent years, public sentiment towards birds has changed from something to be killed for fun, food, or profit. Now birds are considered to be deserving of protection.

Because birds come and go as they please, and cannot (generally) be kept out by fences, scientists and engineers seek automated ways to determine their presence. Birds, by and large, are a garrulous lot, so microphones and audio processing equipment could possibly provide this capability. This thesis investigates that possibility.

1.1 Potential Applications

In this section, various applications that would benefit from automated bird recognition technology will be discussed.

1.1.1 The Air Industry

Bird strikes with planes cause more than two billion dollars of damage per year in North America, and sometimes result in accidents that lead to loss of life (Bruder *et al.*, 1998). Birds are a growing problem at airports for several reasons. First, the migratory bird, and "resident" geese populations

have been increasing recently due to conservation efforts. Second, the total number of aircraft flights has been increasing. Third, modern aircraft are being designed to use fewer engines with larger inlets, leading to a more severe situation if one or more engines is lost due to a birdstrike (Bruder *et al.*, 1998).

1.1.2 Electrical Distribution

Birds coming into contact with electrical lines and transformers can cause costly power outages, and kill the birds. Some birds are protected by law, and deaths may result in fines that are comparable to outage costs (Carlton and Harness, 2001). Birds may also nest on transformers, in towers, or in substations. Some species, such as woodpeckers, are capable of boring into wooden utility poles, leading to premature rotting or structural failure (Dennis, 1964).

In fact, the author's indirect participation (graphic user interface development) with an electrical company led to his involvement in bird recognition. In 2000, Manitoba Hydro funded a small research project to develop sound recognition software, along with a unit for distinguishing between background noise and vocalizations, for the purpose of inexpensively monitoring wildlife, especially birds. The project had four goals (Berger, 2005):

- Monitor wildlife in potentially sensitive habitats that require a transmission or distribution line route (i.e., assessment of an area before development);
- Monitor wildlife in problem areas;
- Provide a means for assessing transmission line security risks from potential bird-wire electrocution or collisions;
- Enhance Manitoba Hydro's wildlife monitoring and protection program.

1.1.3 Wind Turbines

Bird collisions with fast-moving turbine blades have be getting a lot of press coverage recently. Wind turbines are increasing in number, and are considered to be an ecologically friendly way to generate power. Perhaps due to this reason, and the fact that they are a "new" technology, more attention is given to bird deaths that they cause. Bird interactions with towers has been investigated by Evans (1998). It seems that birds collide with towers because inclement weather conditions force them to fly low, or poor visibility causes disorientation, leading them to fly towards illuminated objects. An automated detection system could react when birds approach, by adjusting the lighting to make the tower less attractive, or by shutting down the turbines altogether.

1.1.4 Night-Flight Monitoring

Many species of birds give periodic calls when they are migrating at night. Studies with doppler radar show that there exists a weak, but statistically significant, correlation between flight calls and the actual quantity of birds (Farnsworth *et al.*, 2004). Thus, flight calls can be used for ornithological studies and conservation research (Farnsworth, 2005). Some automated recognition in this area has already been implemented. Skyward-pointing microphones with sound-activated recorders can capture the calls of passing birds. Some species classification is done manually by volunteers, but some calls can be detected by heuristic-based processors that scan for short bursts of acoustic energy in specific frequency ranges. Other heuristics are needed to trigger a shutdown mode if continuous false detections are caused by raindrops, insects, frogs or mechanical noises. This research is described in depth by Evans (2005).

1.1.5 Entertainment

Bird watching, or *birding*, is a popular recreational activity. A device that can assist birders with indentifying species might be profitable.

1.2 Other Recognition Efforts

Bird call recognition is not a very active field. After a lengthy search, only about a dozen papers on the topic were found. None of those used either support vector machines or kernel density estimation for recognition. For brevity, here are descriptions of a few of the better ones.

As mentioned earlier, Evans (2005) developed a heuristic based system. It is designed to detect specific frequency bursts that are indicative of broad categories of warblers, sparrows, thrushes, and the Dickcissel. Finer classifications are made by manually inspecting calls with the help of spectrogram software. This appears to be the only bird call recognition system that is being used in a real world application.

Härmä and Somervuo (2004) describe a method of classifying the tonal quality of birds songs based on the presence or absence of harmonics, and the position of the strongest harmonic.

Derégnaucort *et al.*(2001) used artificial neural networks not for classification, but for quantification. They trained a four-layer ANN to distinguish between two purebred subspecies of quail. The feature vector was an entire 128×64 pixel spectrogram image. Once trained, they put calls of various quail hybrids into the network, and used the "analog" values of the output neurons to situate the calls on a two dimensional diagram. This permitted them to explain how a hybrid's call is influenced by its progenitors.

Finally, a paper that was extremely influential in the development of this thesis, and in fact is an often cited pioneering work, is by McIlraith and Card (1997). In that paper, the authors describe two methods of recognition. First, they used the Fourier transform and linear predictive coding to extract features that were then passed to a backpropagation neural network. Second, they segmented songs into periods of sound and silence, then used statistical discriptions of the various elements as features for principal component and quadratic discriminant analysis. Recognition rates were 82–93% correct.

1.3 Musical Instruments

This thesis takes a different approach to the recognition of audio signals. Many other researchers look at a signal as a whole, then try to classify it based on many characteristics including global ones. For the example of bird calls, some pertinent global parameters might be:

- Total call duration;
- Number of separate distinct vocalizations in the call;
- Lengths of each distinct vocalization;
- Global averages, such as average frequency or average energy in the signal.

A decision was made to take the opposite approach. Instead of taking a global view, the local view would be taken, and instantaneous parameters utilized. One reason for this decision was the personal observation that it is possible to broadly categorize bird calls by listening to brief excerpts. For example, if given short fragments of a goose call, a crow's call, and that of a robin, it is clear which species is which. A robin's call is a fairly pure tone without harmonics, a crow's call is harsh and noisy, and a goose has a familiar honking sound. The human ear can categorize these calls based on the tonal qualities of the sound.

1.3.1 Tonal Qualities

The "quality" of a sound can be described with many adjectives. One could say that it is *pure*, or *harsh*, or *dissonant*, or has a particular *timbre*.

Much research has been done on classifying sounds based on their tonal qualities, specifically in the field of musical instruments. The goal of much of this research is to automate the transcription of music.

A good introduction to this topic is Agostini *et al.*, 2001. The authors define six features that are used to classify a recording of a musical instrument. They are:

1. The zero crossing rate, which is an indication of pitch;
2. The spectral centroid, or the center of gravity of the spectrum;
3. Bandwidth, or the sum of the absolute differences of the spectral amplitude from the spectral centroid;
4. The harmonic energy percentage, or the percentage of spectral energy that is contained at multiples of the fundamental frequency, f_0;

5. The inharmonicity, or the sum of the distances between the actual spectral peaks and the expected harmonic (multiples of f_0) peaks;

6. The harmonic energy skewness, which is an extension of the inharmonicity calculation in which the distance is multiplied by the spectral energy.

Using the means and standard deviations of these signals, the authors were able to get excellent recognition (97%) among 27 instruments using Quadratic Discriminant Analysis, and good results ($\approx$ 78%) with Canonical Discriminant Analysis, k-Nearest Neighbor, and Support Vector Machines.

Some other examples of instrument recognition techniques were demonstrated by Martin *et al.*(1998) who used 31 features including vibrato, tremolo, onset, decay and odd-even harmonic characteristics, which were then classified using a hierarchical Gaussian model derived with Fisher multiple discriminant analysis. The accuracy was comparable to human listeners; 99% for instrumental families and 70% for individual instruments, among 15.

Eronen (2001) used 20 features including the amplitude envelope, amplitude modulation characteristics, and onset asynchrony (differences in energy development for different frequencies) and cepstral coefficients, with linear prediction analysis for classification. The results were mediocre: 77% for instrument families and 35% for individual instruments among 16.

Marques *et al.*(1999) used linear prediction features, cepstral and mel-cepstral features, fed into Gaussian Mixture Models and Support Vector Machines, with a resulting 70% accuracy rate overall for a dataset of 8 instruments.

Ideal Conditions

One important aspect of these papers is that the audio recordings used were very optimal. Each recording contained only a single instrument and the signal to noise ratio was very high. The sample databases were CD-quality studio recordings, and were either solo-instrumental performances or, even simpler, a single tone played on the instrument. The inherent clarity and noise-free aspect of these recordings permitted the utilization of features that would be useless in a noisy environment. Two examples are the spectral centroid, which will be biased depending on the "color" of the background noise (see Section B), and the zero crossing rate, which is sensitive to Gaussian noise. As we will see later, the unavoidable noisiness of bird recordings limits the choices for features to use in classification.

1.4 ANN, SVM and KDE

It was decided that this thesis should investigate three pattern recognition methods: artificial neural networks (ANN), support vector machines (SVM), and kernel density estimation (KDE).

ANNs, being a mature field, were chosen to provide a baseline of sorts for comparison. Artificial neural networks had been studied since the 1940s (Hertz *et al.*, 1991), but were revitalized in 1986 with the release of the work by Rumelhart and McClelland (1986), which described the new technique of backpropagation, a computationally efficient and powerful training algorithm.

SVMs are a more recent invention (Vapnik, 1998) which, in many cases, give better results than neural networks (Müller *et al.*, 2001). SVMs were chosen to see how the relatively new field of structural risk minimization (Vapnik, 1998) compared with the more deeply investigated neural networks. SVMs were introduced by Vapnik, (1995, 1998) based on work starting in the late seventies (Vapnik, 1979). Since then, it has proven to be a powerful classifer and has been applied to the problems of text categorization, bioinformatics, optical character recognition, time-series prediction, density estimation, and many others (Müller *et al.*, 2001; Campbell, 2002; Burges, 1998).

KDE, a non-parametric statistical technique with a long history (Scott, 1992), was chosen to see how a simple algorithm compared with more modern classifiers. KDE is usually used to discover non-obvious characteristics of data distributions. It is not generally applied to pattern recognition problems, but its simplicity and clarity of concept make it appealing.

Chapter 2

A Taxonomy of Noises

The data set for this project consisted of some 900 bird sounds, categorized into ten species. The recordings were provided by Alex McIlraith and Rob Berger of Myrica Systems.

To human ears, the quality of the recordings were very good. However, for automated analysis, signal quality has more stringent requirements to be useful in a pattern recognition system. Before any sort of processing was done, the recordings were manually sorted through, and calls that were non-optimal in some respect were discarded. What follows is a list of various elements that might make a recording unsuitable for processing.

2.1 Low SNR

Probably the most common problem seen was a poor signal to noise ratio (SNR). This would be expected when attempting to record a rare or solitary species of bird — it is difficult to get close to the bird, so the call has to be recorded at a great distance, which results in a weak signal that is overpowered by the background noise. Even a strong call can produce a low SNR if the background noise is louder than usual due to conditions such as wind or rain. As an aside, the low-SNR calls gave another example of the capablities of the human brain, as the calls could easily be perceived when conventional pitch detection techniques fail to extract the signal.

2.2 Non-Stationary Noise

Generally, the background noise can be modeled as a white Gaussian noise source that is filtered by characteristics of the environment and recording equipment. This filtered noise can be

described by referring to the noise rainbow and choosing a close match such as pink noise, red noise, green noise, etc. Once the noise spectrum is known, the signal can be cleaned up by applying Weiner filtering (Proakis *et al.*, 1996) or spectral subtraction (Boll, 1979). These methods require that the noise model not change over time, or is *stationary*.

Some recordings exhibited non-stationary noise. Often, the noise spectrum of the birds calls has at least one wide "hump." In most cases the hump stayed put and did not change its center frequency from frame to frame. For the non-stationary signals, the hump was seen in the spectrogram to move quickly (on the order of seconds) up and down the frequency spectrum as time progressed. Physically this can be explained as an effect of the worker changing position during recording. Objects near the microphone, such as the ground, a tree trunk, or the person themself, produce a virtual cavity with its own filtering characteristics. As the person moves, these properties will quickly change, leading to nonstationary background noise.

2.3 High SNR

This was not a problem in this thesis, but in preliminary experimentation, signals which were loud, clear, and almost free of background noise, were categorized separately by a commercial data-clustering tool when looking at spectral characteristics. This observation is being conveyed because it was a counterintuitive result at the time. The "flawless" signal was, in fact, too perfect, and was thus seen as an outlier (which, compared to the other signals, it was).

2.4 Echos and Reverberation

Generally, echos were not a problem. There were two main types: a strong single echo, which created a double image in the spectrogram, and a multi-source echo (reverberation), which caused a smearing out of clear chirps. In early experimentation at isolating individual chirps, it was found that echos were usually interpreted as additional chirps. Reverberation had the effect of blending together a sequence of closely-spaced chirps.

2.5 Environmental Interference

This refers to non-noise sounds that interfere with the bird calls. This was another common problem with the audio samples, and reflected the fact that the birds are not being recorded in a laboratory, but in the real world, where the researcher has little control over the surrounding events at the time of the recording.

Since the recording equipment in this project was operated by people, it is not unexpected that human-generated sounds were captured occasionally. Researchers could be heard discussing birds (among other things), coughing, and changing their position, which resulted in the microphone clunking against things. Movements generated crumpling or swishing sounds of clothing being flexed.

The geographic location of the recording also affects the type of interference. In populated areas, the sounds of machinery become common. Trains, planes, automobiles, lawn mowers, and construction equipment all generate strong signals that interfere with the bird being observed.

In both rural and urban areas you will find cross-species interference, in which the call of one bird is punctuated by the chirps of another bird of a different species, or the buzz or croak of an amphibian or insect. Usually, the interfering call is more distant and weak, so it is obvious to a human listener which call is which. However, an automated pattern recognition system might "perceive" the combined sounds to be that of a single bird.

Finally, there is the issue of interference by birds of the same species, but that will be covered in the section on underclassified calls.

2.6 Equipment Distortion

Usually the recording equipment operated properly. Very few recordings were marred by distortion. Of those that were, some were caused by the recording gain being set too high, causing clipping distortion. This would manifest itself as second order harmonics, and the signal would have a buzzing quality to it.

Some signals showed a strange characteristic of spectral mirroring. When viewed as a spectrogram, a faint vertically mirrored image of the spectrum could be seen along the top, upside-down with the zero-frequency component centered at the sample rate of 22.05 kHz. The source of this unusual distortion has not been fully explained, but in a pure signal processing sense, this effect would be seen if the original signal was being modulated by a weak 22.05 kHz "carrier" wave. Since the analog recording equipment used in the field would have no affinity to that particular frequency, it is suspected that this distortion was introduced when the signal was converted to a digital format. If the analog-to-digital hardware has a gain that was oscillating slightly at 22.05 kHz, then this spectral mirroring would appear.

2.7 Underclassified Calls

Most birds have a *call repertoire*, a variety of distinct sounds that are used in different situations. For example, a bird might issue an *alarm call* if it noticed a predator approaching, which would

alert other birds nearby. Call repertoires are covered in detail by Marler and Slabbekoorn (2004), who describe fifteen basic types of calls (not including variations thereof).

In the data set used, the calls were categorized only by species. Thus, any pattern recognition scheme would have to deduce that each species could generate various styles of calls. In the case of a neural network or SVM approach this adds an extra burden to the learning process. Thus, to simplify things, only one style of call was selected (territorial) and others were removed from the data set. Territorial calls or songs are the most common in the dataset, and are usually the calls that people associate with a species.

Earlier it was mentioned that birds of the same species sometimes interfered with each other. Such sounds are not considered to be flawed and unsuitable for analysis. Some birds are naturally sociable and prone to congregate in flocks. For these species, there is a good chance that a recording will contain intra-species interference. Rather than discard these sounds, it was felt that group calls are as legitimate as solitary calls, and should be separated into "flock" and "solitary" categories.

In summary, the data samples for a species could be classified into the following subcategories in order to simplify the pattern recognition task:

- Calls in repertoire (Marler and Slabbekoorn, 2004)
  - Territorial call;
  - Courtship call;
  - Alarm call;
  - etc.;
- Simultaneous calls of flock.

This is only a subset of sounds a bird may produce. Many species have their own vocabulary of sounds for specific situations and events.

2.8 Other Issues

The data set used in this work had some instances where there were several recordings taken of a specific bird only a few minutes apart, or an exceedingly long sample in which one bird repeated a call several times. Since repeated calls by the same individual does not provide a data set that could be generalized to the species as a whole, these extra calls were removed.

Finally, some recordings might not have a clear specific problem as described in the preceding sections, but rather a combination of minor flaws that sum up to an unusable signal.

Chapter 3

Pitch Determination

With the results of the previous chapter in mind, the next step would be choosing which features to use, and which algorithm should be used to to extract the features.

One popular feature is *pitch*, or *fundamental frequency* (f_0). The usefulness of the pitch component is self-evident, especially for musical instrument detection. Pitch alone can be used to distinguish between many instruments. To give an obvious example, the sounds of a tuba versus a piccolo can be classified solely on pitch information.

3.1 Algorithm Choice

Pitch determination, as simple as the procedure may sound, is actually an unsolved problem (Hess, 1983). A search for "pitch detection" or "pitch determination" on an academic database will reveal dozens if not hundreds of papers on the topic. Even in the last five years alone, several dozen papers have been written on novel pitch determination algorithms (PDAs). This topic is alive and well, for PDAs are important in speech recognition research and human voice bandwidth compression.

A foundational work on PD is (Hess 1983). Although Hess' work may seem out of date, the two main techniques described, autocorrelation PDA and cepstrum PDA, are still in use today, and perform favorably with newer methods (Cheveigné *et al.*, 2001).

3.2 The Human Voice

Before continuing, it is appropriate to describe how pitch is related to human vocalizations, and how they differ from bird sounds. As mentioned earlier, modern PDAs concentrate on human

sounds, so some changes are needed to accomodate bird calls.

In simplification, the human voice is modeled by an excitation signal (from the vocal cords) being passed through a filter (the throat, nose and mouth). The excitation signal is generally seen as a sequence of pulses, at some frequency f_0 (Figure 3.1).

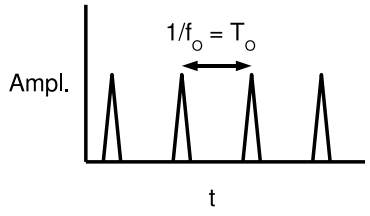

Figure 3.1: Model of human vocal excitation.

The vocal tract response is generally seen as a multi-pole filter. (Figure 3.2)

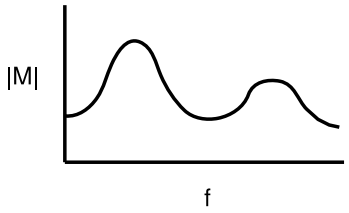

Figure 3.2: Model of human vocal tract response.

The resulting spectrum (Figure 3.3) is the well-known result of passing a pulse train through a filter (Oppenheim *et al.*, 1983).

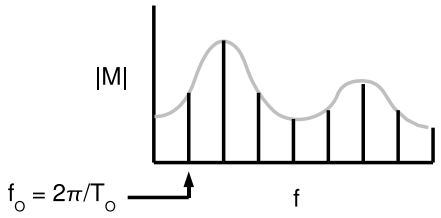

Figure 3.3: Result of passing pulse-train through filter.

Hence, human vocalization has a power spectrum with a comb-like structure. Ideally, the first peak is the fundamental frequency. However, in many situations, the first peak is obscured by noise, or attenuated by extraneous filtering effects. Thus, a simple algorithm that looks for the first peak will often fail. A more robust method is the *double transform*, also known as *cepstral analysis.*

3.3 The Cepstrum

The cepstrum seems intuitive enough. When confronted by a power spectrum that has a repeated periodic sequence of regular peaks, why not just apply the Fourier transform to it to find the "periodicity" of peaks in the frequency domain? Surprisingly, this novel trick (which has no doubt been rediscovered many times over) actually has a theoretical underpinning.

Cepstrum pitch detection was first studied in depth in (Noll, 1964). At the time it was the most reliable PD algorithm, and for many years new PDAs were calibrated against it. Even today, (Cheveigné *et al.*, 2001) show that cepstral methods are very competitive with newer techniques.

Usually, the cepstrum is defined as

$$\text{Cepstrum}\big[s(t)\big] = \mathcal{F}^{-1}\Big[\log\Big|\mathcal{F}\big[s(t)\big]\Big|\Big]. \tag{3.1}$$

Or: the cepstrum is the inverse Fourier transform of the log of the short-term power spectrum of the signal. The *log* operator has the effect of separating the voice source, which is a pulse train, and the vocal tract filter function so that the pulse sequence appears as a peak at a *quefrency* (or *lag*) in the cepstrum, revealing the period of the pulse train, T_0.

In (Hess, 1983) there is much discussion as to whether *log* is the ideal transfer function before the inverse Fourier transform is taken. Hess notes that the fourth root, the square root, and an unaltered magnitude seem to provide better resistance to noise. In this project it found that a linear transfer function produced the best results during experimentation. Thus, the cepstrum PDA used in this thesis is:

$$\text{Cepstrum}'\big[s(t)\big] = \mathcal{F}^{-1}\Big[\Big|\mathcal{F}\big[s(t)\big]\Big|\Big] \tag{3.2}$$

See Figure 3.4 for a flowchart.

3.4 Human versus Bird Sounds

The typical power spectrum of a non-pathological human speaker has two commonalities. First, it is rich in harmonics due to the pulse like signal produced by the vocal cords. This allows cepstral analysis to work effectively. Second, the fundamental frequency ranges from about 50 to 1800 Hz. These two facts motivate much of the development for PDAs, which means that many robust PDAs will degrade when confronted by a bird signal.

Compare to human sounds, bird sounds are unconstrained. the pileated woodpecker or the chickadee produce harmonics-rich sounds. Songbirds produce spectrally pure sinusoids. The great horned owl squawks at a pitch of 300 Hz, and the Blackpoll warbler sings at 10 kHz (Berger, 2005).

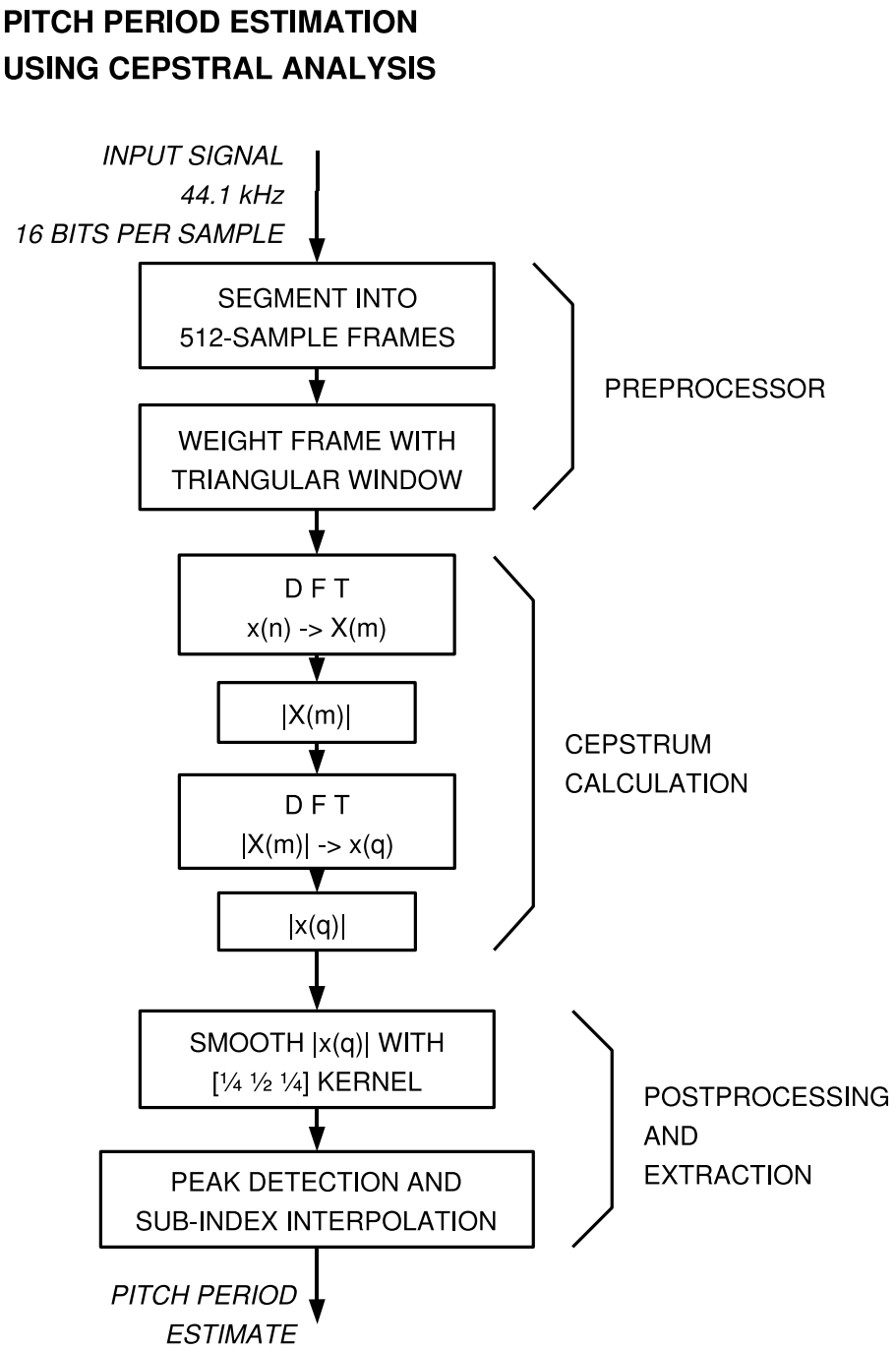

Figure 3.4: Flowchart of the cepstrum pitch detection method with pre- and post- processing.

Thus, an additional PDA is needed, one that is suited to sinusoidal signals.

3.5 Periodicity Determination

Detection of simple periodicity has a long history in signal analysis. There are several common algorithms, each with their own advantages and drawbacks. Some are described in (Proakis *et al.*, 1996) under the subject of *power spectrum estimation*. The technique used in this thesis is the Blackman-Tukey method. Discretely, it is defined as

$$P_{xx}^{BT}(f) = \sum_{m=-(M-1)}^{M-1} r_{xx}(m)w(m)e^{-j2\pi fm} \tag{3.3}$$

Where r_{xx} is autocorrelation, w is the windowing function (usually a triangular Bartlett window), and M is the number of samples. See Figure 3.5 for a flowchart.

Proakis *et al.*(1996) shows that the Blackman-Tukey method gives a higher quality of spectral estimate than either the Bartlett or Welch method. Implementation-wise, the autocorrelation part may be sped up through use of the Wiener-Khintchine theorem:

$$r_{xx}(l) \overset{\mathcal{F}}{\longleftrightarrow} S_{xx}(w) \tag{3.4}$$

or

$$r_{xx}(l) = \mathcal{F}\left[\left|\mathcal{F}\left[s(t)\right]\right|^2\right] \tag{3.5}$$

Where r_{xx} is again the autocorrelation and S_{xx} is the energy spectral density. This allows the full autocorrelation of the signal to be calculated more quickly with the help of two Fourier transforms.

3.6 Combining PDAs

Now, we have two pitch determination algoritms: the cepstral method, which handles signals rich in harmonics, and the Blackman-Tukey method, for signals with few spectral peaks. Since bird sounds could be either one of these types of signals (or somewhere in between), we need some way to choose a method depending on the signal, or to combine the two methods together in a logical and useful manner. For this work, very little additional pre-processing was chosen. Instead, both the spectral and cepstral results were passed to the pattern recognition algorithm, with the hope that it would find something useful in the information. Specifically, the following pitch-related parameters were used:

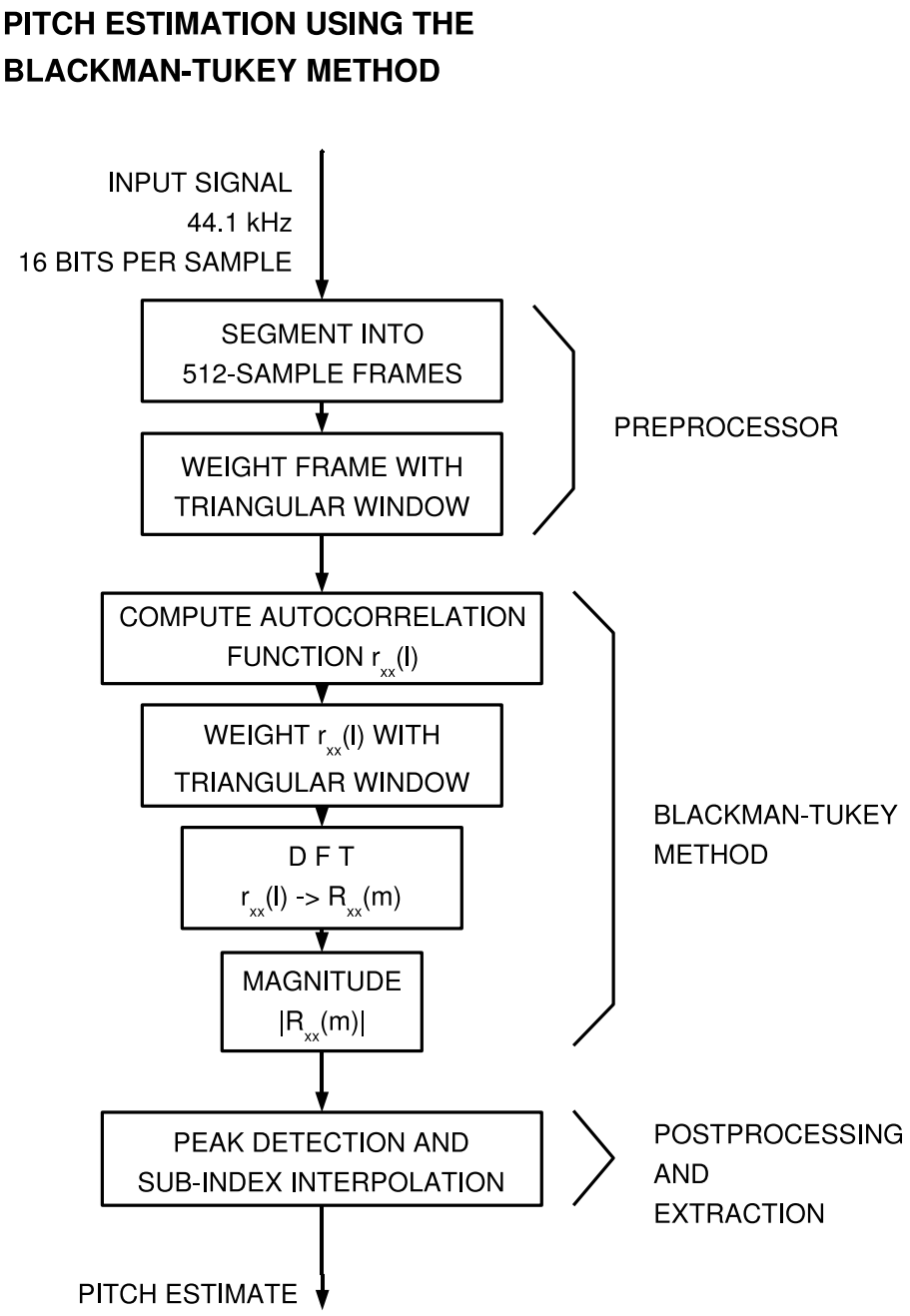

Figure 3.5: Flowchart of the Blackman-Tukey method with pre- and post- processing.

- Spectral characteristics;
  - Total energy of the spectrum;
  - Peak characteristics;
    - Frequency at the peak;
    - Energy in the peak (+/- one bin);
    - Normalized energy in peak (peak energy divided by total energy);
- Cepstral characterstics;
  - Energy of the cepstrum;
  - Cepstral peak characteristics;
    - Quefrency (lag) of the peak;
    - Energy in the peak (+/- one bin);
    - Normalized energy in peak (peak energy divided by total energy).

Normalization and Confidence

The normalized energy parameters are important because they provide a sort of "confidence" metric to the spectral and cepstral peak estimates. In the best case, the peak would contain all the energy, and the normalized energy would be 1. In the worst case, say, a signal of white noise, the normalized peak energy would be close to zero, because most of the energy is spread elsewhere in the spectrum of cepstrum. In fact, by limiting our observations only to the normalized energies, we can situate different types of signals in a two dimensional space based on tonal quality, as shown in Figure 3.6. This parameter simultaneously informs us of the purity, harmonicity, and noisiness of the signal, and thus may be useful. The tonal quality analysis was restricted to this level of detail, because the signals available were far too noisy to apply some of the finer techniques used for musical instrument recognition.

3.7 Other Features Used

3.7.1 Derivatives

All of the pitch-related features discussed previously are numerically differentiated, and these derivatives are passed on to the pattern recognition algorithm. The motivation for this is based on the fact that for some birds, the rate of variation of a parameter is important. For example, the pitch of a warbler's song is expected to increase and decrease rapidly.

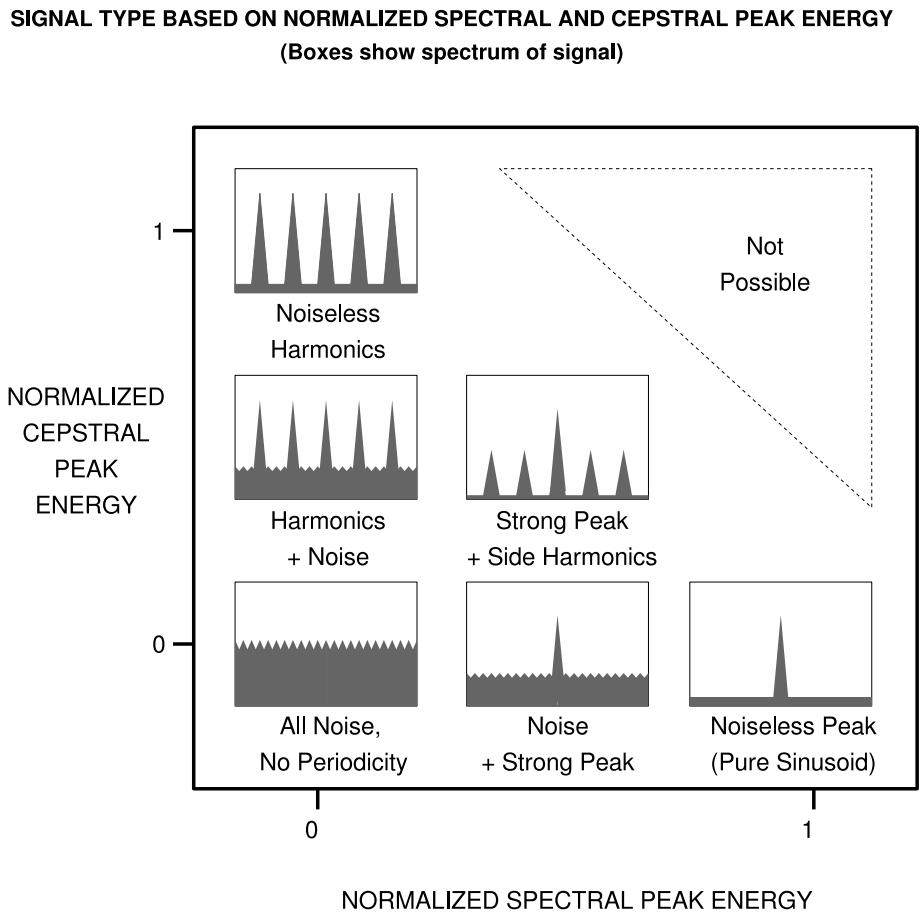

Figure 3.6: Signal type based on spectral and cepstral energy.

3.7.2 Amplitude Envelope Frequency

Until now, all the features mentioned were instantaneous values which were extracted from small fragments of the signal, on the order of 10ms in duration. There were no intra-frame features that gave some information of the (relatively) long-term structure of the bird call. The amplitude envelope frequency is one way to convert this long-term information into more digestible single-valued parameters. Figure 3.7 explains the details of the algorithm.

This was motivated by the observation that some birds produce calls which are smooth in loudness, whereas others are stocatto and repetitive. By looking at a 1.5 second interval of a call, and determining the periodicity of amplitude variations within that range, we can extract useful information about the overall structure of the call. Like the pitch detection algorithms, the envelope frequency includes the normalized peak energy to give a confidence estimate of the extracted frequency.

As far as the author knows, this feature and algorithm has not been used before in bird (or other animal) recognition, and is a novel invention in this project.

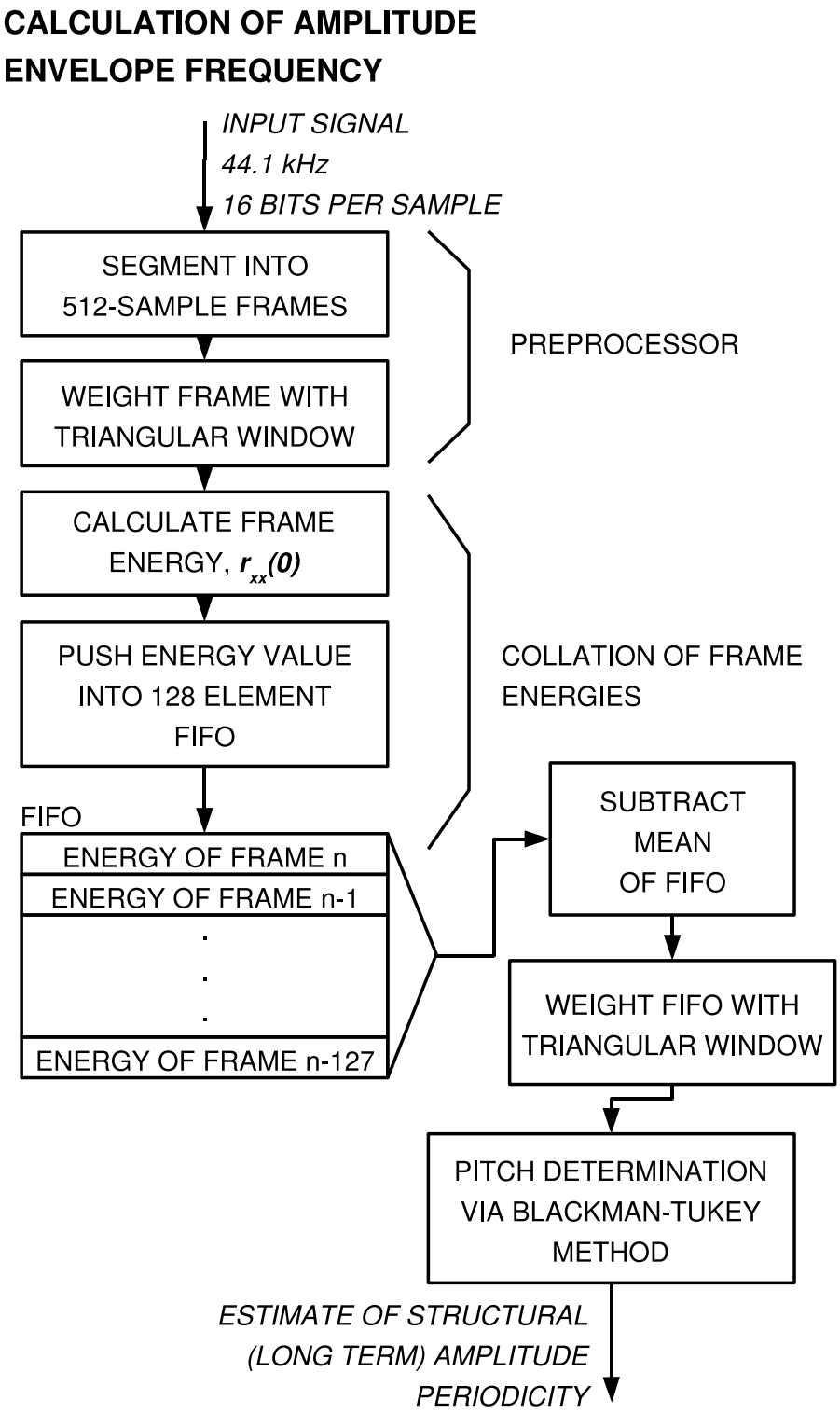

Figure 3.7: Determination of envelope periodicity, also referred to as structural amplitude frequency.

FEATURES EXTRACTED FROM AUDIO SIGNAL

Description	Formula	$\frac{d}{dt}$ *
Frequency Domain Parameters		
Power in frequency peak plus two adjacent bins	$P^{\mathrm{BT}}_{\mathrm{peak}} = \sum_{i=-1}^{+1} P^{\mathrm{BT}}_{xx}(f^{\mathrm{BT}}_{\mathrm{peak}} + i)$ ‡	Yes
Total power in frame	$P_{\mathrm{total}} = r_{xx}(0)$	Yes
Normalized peak power	$P^{\mathrm{BT}}_{\mathrm{peak}} / P_{\mathrm{total}}$	Yes
Frequency at peak	$f^{\mathrm{BT}}_{\mathrm{peak}} = \arg\max_f P^{\mathrm{BT}}_{xx}(f)$ †	Yes
Cepstral Domain Parameters§		
Sum of gamnitude in peak and adjacent bins	$G_{\mathrm{peak}} = \sum_{i=-1}^{+1} G(q_{\mathrm{peak}} + i)$	Yes
Total gamnitude in cepstrum	$G_{\mathrm{total}} = \sum_q G(q)$	Yes
Normalized peak gamnitude	$G_{\mathrm{peak}} / G_{\mathrm{total}}$	Yes
Quefrency at peak	$q_{\mathrm{peak}} = \arg\max_q (G(q))$	Yes
Quefrency at peak back-converted to frequency	$f^{\mathrm{CEP}}_{\mathrm{peak}} = N/(2q_{\mathrm{peak}})$	Yes
Multi-Frame Amplitude Envelope Parameters		
Amplitude envelope frequency	$f^{\mathrm{BT}}_{\mathrm{envel}} = \arg\max_f P^{\mathrm{BT}}_{\mathrm{envel}}(f)$	No
Normalized envelope peak power	$P^{\mathrm{BT}}_{\mathrm{envel}_{\mathrm{peak}}} / P^{\mathrm{BT}}_{\mathrm{envel}_{\mathrm{total}}}$	No

∗ If "Yes", then an additional feature was created by taking the numerical derivative between frames.
† The function $\arg\max f(x)$ returns x that gives the maximum value of $f(x)$.
‡ BT indicates the Blackman-Tukey method of power spectrum estimation.
§ *Quefrency* and *gamnitude* are the cepstral analogues to frequency and magnitude.

Table 3.1: Features passed to the pattern recognition system.

Chapter 4

Linear Learning

This section provides an introduction to learning systems through a discussion of linear learning machines. Not only are these easy to understand, easy to train, and useful in their own right, but they also lay a foundation for artificial neural networks and support vector machines. Linear learning is well covered in (Cristianini *et al.*, 2000) and (Bishop, 1995).

4.1 What is Learning?

There are two types of learning usually discussed in the field of pattern recognition: unsupervised and supervised (Haykin, 1994).

Unsupervised learning involves observation of a dataset and consequent extraction of groups (or clusters) in that data, with no external input specifying the expected classification of any of the elements of the dataset (Haykin, 1994). Unsupervised learning will not be investigated in this thesis.

A supervised learning system takes a dataset of observation vectors, and finds a mapping, or function, that will enable the deduction of some elements of the vectors based on the values of the other elements. Usually a system is described as taking input, or training, vectors, and corresponding output, or target, vectors, as a training dataset. Once a system is trained it ideally will possess the ability to process a new input vector (in the absence of a target vector), and correctly predict the values of the target. See Figure 4.1 for a schematic representation (Haykin, 1994).

A popular form of pattern recognition is simple binary, or *yes/no* classification of an input vector. An input vector, $\mathbf{x}$, might be classified as *true* if some real function $y = f(\mathbf{x})$ gives a result ≥ 0 , and *false* otherwise. In the case where $f(\mathbf{x})$ is linear, the classifier can be written as

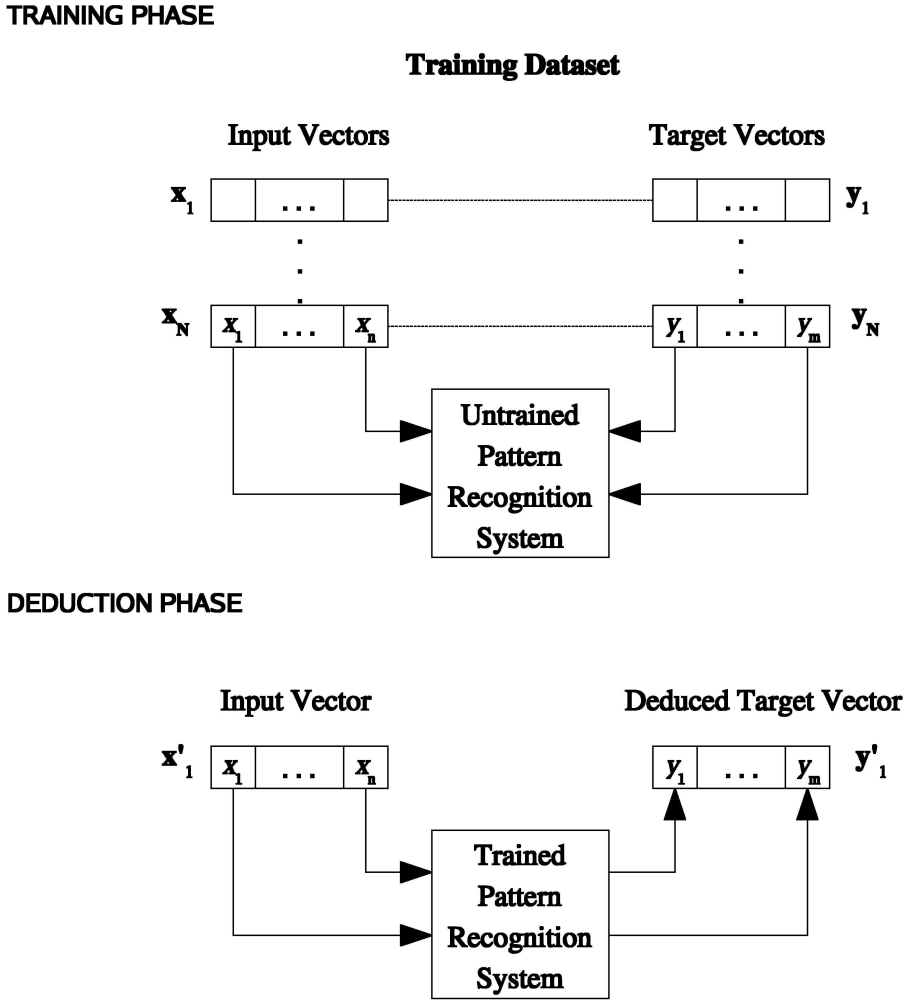

Figure 4.1: Phases of Supervised Learning. This diagram shows N observations, n elements in each input vector, and m elements in each output vector (Haykin, 1994).

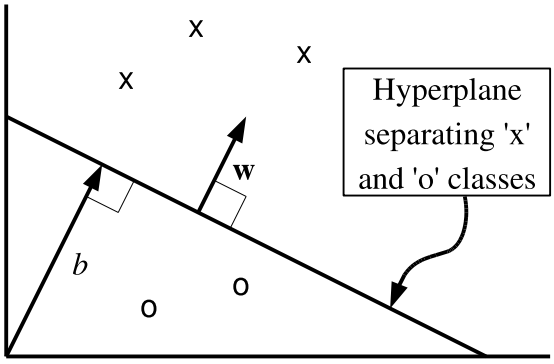

Figure 4.2: A two-dimensional space separated by a one-dimensional hyperplane specified by **w** and b (Cristianini *et al.*, 2000).

$$\begin{aligned} f(\mathbf{x}) &= \mathbf{w} \cdot \mathbf{x} + b \\ &= \sum_{i=1}^{n} w_i x_i + b \end{aligned} \tag{4.1}$$

Geometrically, this function cuts the n-dimensional hyperspace along an $(n-1)$ dimensional hyperplane, where $f(\mathbf{x}) = 0$. See Figure 4.2 for a two-dimensional example that shows a hyperplane whose normal is defined by **w**, and which has an offset defined by b. Typically **w** and b are known as the *weight vector* and *bias* (Haykin, 1994).

Rosenblatt showed in 1956 that such a hyperplane can always classify a *linearly separable* dataset. (The definition of linearly separable being "that which can be separated by a hyperplane.") The hyperplane can be discovered by the iterative *Perceptron Algorithm* (Haykin, 1994).

A parameter of interest related to the Perceptron Algorithm is the *margin* of a point **x** with respect to the hyperplane $(\mathbf{w}, b)$, and is defined as

$$\gamma_i = (y_i)(\mathbf{x} \cdot \mathbf{w} + b). \tag{4.2}$$

Where $\gamma_i > 0$ implies correct classification, and y_i is either $+1$ or -1 depending on the class to which **x** belongs. The margin γ_i is simply the distance from the point **x** to the hyperplane $(\mathbf{w}, b)$ (Cristianini *et al.*, 2000).

The smallest margin between all points in a training set **X** and a hyperplane $(\mathbf{w}, b)$ is sometimes referred to as the *margin of the hyperplane* with respect to the training set (Burges, 1998).

The margin M of a training set **X** is the greatest possible margin over all hyperplanes (Burges, 1998). A hyperplane producing this maximum is the *maximal margin*, or *optimal* hyperplane. Its margin will be positive for a linearly separable training set (see Figure 4.3).

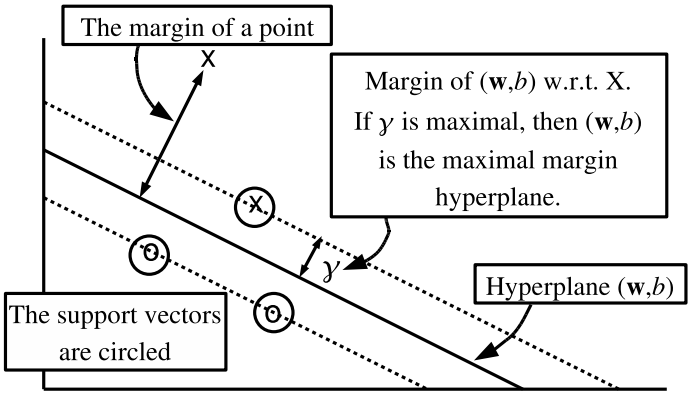

Figure 4.3: Hyperplane and Margins (Cristianini *et al.*, 2000).

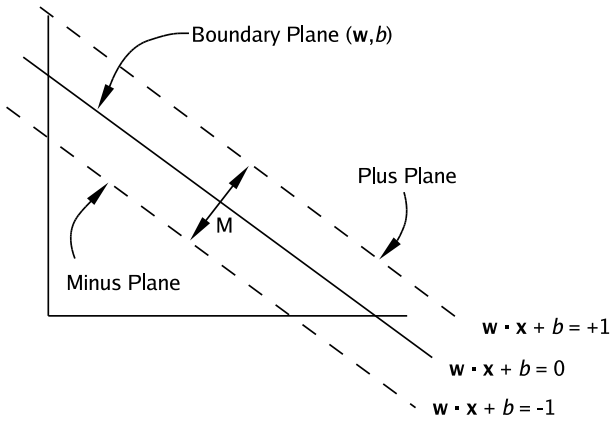

Figure 4.4: Determining margin implicitely (Moore, 2001).

The *support vectors* are the subset of the dataset which lie adjacent to the optimal hyperplane at a distance of the margin (Burges, 1998).

(As an aside, a statistician might note that such an optimal hyperplane, as defined, depends solely on outliers in the dataset.)

4.2 Determining the Optimization Problem

To continue with our analysis, we will need to know the margin width of a hyperplane $(\mathbf{w}, b)$. An interesting fact is that the margin is implicitly defined by requiring that $\mathbf{w} \cdot \mathbf{x} + b$ returns either greater than +1 or less than -1 depending on the class. This extends the concept of a hyperplane dividing the two classes by adding a "built in" margin (Moore, 2001).

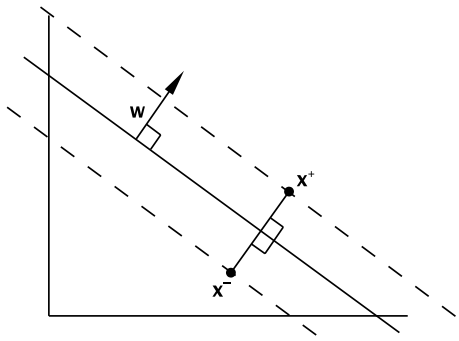

Figure 4.5: Location of nearest points on opposite margins (Moore, 2001).

We would like to find M, the margin for $(\mathbf{w}, b)$, as shown in Figure 4.4. We know that

$$\mathbf{w} \cdot \mathbf{x}^+ + b = +1 \tag{4.3}$$

and

$$\mathbf{w} \cdot \mathbf{x}^- + b = -1. \tag{4.4}$$

Assume that $\mathbf{x}^+$ is on the plus plane, and $\mathbf{x}^-$ is on the minus plane. Then, the distances from $\mathbf{x}^+$ and $\mathbf{x}^-$ to the hyperplane boundary are

$$\gamma^+ = (+1)(\mathbf{x}^+ \cdot \mathbf{w} + b) \tag{4.5}$$

$$\gamma^- = (-1)(\mathbf{x}^- \cdot \mathbf{w} + b) \tag{4.6}$$

Assume also that $\mathbf{x}^+$ is the closest point on the plus plane to $\mathbf{x}^-$ (see Figure 4.5). The line from $\mathbf{x}^+$ to $\mathbf{x}^-$ is perpendicular to the dividing boundary $(\mathbf{w}, b)$, so to get from $\mathbf{x}^-$ to $\mathbf{x}^+$ we have to move in the direction of $\mathbf{w}$. Thus, $\mathbf{x}^+ = \mathbf{x}^- + \lambda \mathbf{w}$ for some value of λ.

Now we can take the equation

$$\mathbf{x}^+ = \mathbf{x}^- + \lambda \mathbf{w} \tag{4.7}$$

together with Equation 4.3

$$\mathbf{w} \cdot \mathbf{x}^+ + b = +1 \tag{4.8}$$

to get

$$\mathbf{w} \cdot (\mathbf{x}^- + \lambda \mathbf{w}) + b = +1 \tag{4.9}$$

which reduces down to

$$\lambda = \frac{2}{\mathbf{w} \cdot \mathbf{w}}. \tag{4.10}$$

Now that we know λ, we can find the actual margin M. From (4.7) we get

$$\lambda \mathbf{w} = \mathbf{x}^+ - \mathbf{x}^- \tag{4.11}$$

then

$$M = |\mathbf{x}^+ - \mathbf{x}^-| = |\lambda \mathbf{w}| \tag{4.12}$$

$$= \lambda |\mathbf{w}| = \lambda \sqrt{\mathbf{w} \cdot \mathbf{w}}. \tag{4.13}$$

Replacing λ with the result of (4.10) gives

$$M = \frac{2\sqrt{\mathbf{w} \cdot \mathbf{w}}}{\mathbf{w} \cdot \mathbf{w}} = \frac{2}{\sqrt{\mathbf{w} \cdot \mathbf{w}}}. \tag{4.14}$$

Thus, we can determine the margin width of this classifier solely with $\mathbf{w}$. To find the optimal hyperplane, we want to maximize M (or minimize $\mathbf{w} \cdot \mathbf{w}$) while making sure that all points in the plus class are at or beyond the plus boundary, such that

$$\mathbf{x}^+ \cdot \mathbf{w} + b \geq +1 \tag{4.15}$$

and similarly, for points in the minus class,

$$\mathbf{x}^- \cdot \mathbf{w} + b \leq -1 \tag{4.16}$$

4.3 Quadratic Programming

The numeric technique known as quadratic (or nonlinear) programming (or optimization) is normally used to find the minimum value of a second order polynomial which is constrained by inequalities. QP, like its cousin, linear programming, has a long history, having been investigated since the 1960s (Wright 2004). Many algorithms have been developed and much contemporary research continues in this field.

One typical formulation of a QP problem is to satisfy the following constraints:

$$\text{minimize} \quad \frac{1}{2}\mathbf{w}^T \mathbf{Q} \mathbf{w} - \mathbf{k}^T \mathbf{w} \tag{4.17}$$

$$\text{subject to} \quad \mathbf{X}\mathbf{w} \leq \mathbf{c} \tag{4.18}$$

Where $\mathbf{Q}$ is a positive definite $n \times n$ matrix, $\mathbf{k}$ is an n-vector, $\mathbf{c}$ is an m-vector, $\mathbf{w}$ is the unknown, and $\mathbf{X}$ is an $m \times n$ matrix. The optimal hyperplane problem can be converted to a QP problem by

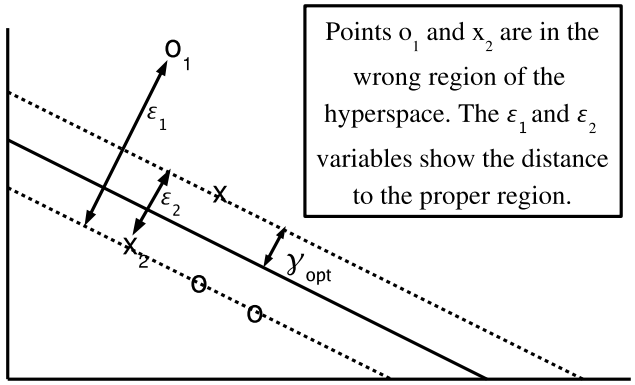

Figure 4.6: Dataset that is not linearly separable (Moore, 2001).

satisfying (Cristianini *et al.*, 2000):

$$\text{minimize} \quad \mathbf{w} \cdot \mathbf{w} - \mathbf{k}^T \mathbf{w} \tag{4.19}$$

$$\text{subject to} \quad (y_i)(\mathbf{x} \cdot \mathbf{w} + b) \geq 1 \tag{4.20}$$

4.3.1 Optimization Problem with Noise

QP will produce an answer for this objective function and constraints only if the dataset is fully linearly separable. If some points are on the wrong side of the hyperplane $(\mathbf{w}, b)$, then additional steps must be taken to make this QP problem work. Figure 4.6 shows a noisy training set with two points in the wrong region of the hyperspace. The training set is not linearly separable. Fortunately, there is a way around this. We can introduce a *cost* parameter that is based on how far the errant datapoints are from their proper margin. Figure 4.6 (Moore, 2001) shows two datapoints at distances ε_1 and ε_2 (known as *slack variables*) from the proper margin. These distances can be multiplied by a cost parameter C, which produces a new objective function and constraints for the QP problem:

$$\text{minimize} \quad \frac{1}{2}\mathbf{w} \cdot \mathbf{w} + C \sum_{i=1}^{m} \varepsilon_i \tag{4.21}$$

$$\text{subject to} \quad (y_i)(\mathbf{x} \cdot \mathbf{w} + b) \geq 1 - \varepsilon_i \tag{4.22}$$

$$\text{and} \quad \varepsilon_i \geq 0 \ \text{ for all } i \tag{4.23}$$

However, this formulation is not the best for QP tools. The system can be optimized more quickly if it is converted to the equivalent *Wolfe dual* problem (Burges 1998) which becomes:

$$\text{maximize} \underset{dual\ lagrangian}{L_D} \equiv \sum_i \alpha_1 - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j \, y_i y_j \, \mathbf{x}_i \cdot \mathbf{x}_j \tag{4.24}$$

$$\text{subject to} \quad 0 \leq \alpha_i \leq C \tag{4.25}$$

$$\text{and} \quad \sum_i \alpha_i y_i = 0 \tag{4.26}$$

Once all α_1 and y_i have been found, $\mathbf{w}$ and b can be found by

$$\mathbf{w} = \sum_{i=1}^{N_S} \alpha_i y_i \, \mathbf{x}_i \tag{4.27}$$

and

$$b = y_I(1 - \varepsilon_I) - \mathbf{x}_I \cdot \mathbf{w}_I \quad \text{where} \quad I = \arg\max_i \alpha_i \tag{4.28}$$

Where N_S is the number of support vectors.

In prediction phase, an input vector $\mathbf{x}$ can be classified by taking the sign of

$$f(\mathbf{x}) = \sum_{i=1}^{N_S} \alpha_i y_i \, \mathbf{s}_i \mathbf{x} + b \tag{4.29}$$

where $\mathbf{s}_i$ are the support vectors.

Thus, a linear learning machine is able to find an optimal hyperplane for a noisy training set that is not strictly linearly separable. This is the technique used for linear support vector machines (LSVMs).

Chapter 5

Nonlinear Support Vector Machines

Now that we have described LSVMs, the next question is, can LSVMs be augmented to handle datasets where the optimal decision boundary is nonlinear? The answer is *yes*, and the simplicity of the solution is striking. It relies on an old kernel equality commonly known as the *kernel trick* (Aizerman *et al.*, 1964).

Notice first how data is used in the LSVM problem. Individual values are never used — only the dot products are taken. Now imagine a mapping from the d-dimensional data space to a higher dimensional *feature space*, $\mathcal{H}$, of f dimensions where $f > d$. This mapping will be called Φ (Burges, 1998). Thus

$$\Phi : \mathbb{R}^d \rightarrow \mathcal{H}, \quad \text{where } \mathcal{H} \in \mathbb{R}^f \text{ and } f > d \tag{5.1}$$

If this mapping is applied to a pair of vectors $\mathbf{w}$ and $\mathbf{x}$, then the dot product would be $\Phi(\mathbf{w}) \cdot \Phi(\mathbf{x})$.

Direct calculation of $\Phi(\mathbf{w}) \cdot \Phi(\mathbf{x})$ may be intractible in a computational sense if Φ maps to an extremely high dimensional space. This is alleviated because in some cases, $\Phi(\mathbf{w}) \cdot \Phi(\mathbf{x})$ may be represented by a kernel K that is mathematically equivalent, yet requires only the dot product of the vectors in the original d-dimensional space. Thus,

$$K(\mathbf{w}, \mathbf{x}) = \Phi(\mathbf{w}) \cdot \Phi(\mathbf{x}) \tag{5.2}$$

This kernel trick, in effect, eliminates the need to explicitely calculate a high-dimensional space with Φ. In fact it allows mapping of the input space to an *infinite dimensional* feature space.

For example, if $d = 2$ with the kernel

$$K(\mathbf{w}, \mathbf{x}) = (\mathbf{w} \cdot \mathbf{x})^2 \tag{5.3}$$

it can be shown, by expanding the dot product above and solving for $\Phi(\cdot)$, to correspond to the

3-dimensional mapping

$$\Phi(\mathbf{x}) = (x_1^2, x_2^2, \sqrt{2}\, x_1 x_2). \tag{5.4}$$

To demonstrate, evaluate

$$\begin{aligned} \Phi(\mathbf{w}) \cdot \Phi(\mathbf{x}) &= (w_1^2, w_2^2, \sqrt{2} w_1 w_2) \cdot (x_1^2, x_2^2, \sqrt{2} x_1 x_2) \\ &= x_1^2 w_1^2 + x_2^2 w_2^2 + 2 x_1 w_1 x_2 w_2 \end{aligned} \tag{5.5}$$

also evaluate

$$\begin{aligned} K(\mathbf{w}, \mathbf{x}) &= (\mathbf{w} \cdot \mathbf{x})^2 \\ &= (w_1 x_1 + w_2 x_2)^2 \\ &= w_1^2 x_1^2 + 2 w_1 x_1 w_2 x_2 + w_2^2 x_2^2. \end{aligned} \tag{5.6}$$

This is equivalent to the earlier expansion, thus demonstrating the kernel trick.

To give an example of an infinite dimensional feature mapping, consider a 1-dimensional (for simplicity) data vector $\mathbf{x}$ being mapped to an infinite dimensional space:

$$\Phi(\mathbf{x}) = \left(1, \frac{x}{\sqrt{1!}}, \frac{x^2}{\sqrt{2!}}, \frac{x^3}{\sqrt{3!}}, \ldots\right) \tag{5.7}$$

Evaluating the dot product we get

$$\Phi(\mathbf{w}) \cdot \Phi(\mathbf{x}) = \left(1, \frac{w}{\sqrt{1!}}, \frac{w^2}{\sqrt{2!}}, \ldots\right) \cdot \left(1, \frac{x}{\sqrt{1!}}, \frac{x^2}{\sqrt{2!}}, \ldots\right) \tag{5.8}$$

$$= 1 + \frac{wx}{1!} + \frac{(wx)^2}{2!} + \frac{(wx)^3}{3!} + \ldots \tag{5.9}$$

Inspection reveals that this mapping is equivalent to the kernel

$$K(\mathbf{w}, \mathbf{x}) = \exp(\mathbf{w} \cdot \mathbf{x}) = \exp(wx) \tag{5.10}$$

Thus the kernel trick can work for infinite dimensional feature spaces.

5.1 Common Kernels

The first kernels investigated with SVMs were the following (Burges, 1998):

Polynomial of degree p,

$$K(\mathbf{w}, \mathbf{x}) = (\mathbf{w} \cdot \mathbf{x} + 1)^p \tag{5.11}$$

Gaussian radial basis function,

$$K(\mathbf{w}, \mathbf{x}) = \exp\left\{-\frac{\|\mathbf{w} - \mathbf{x}\|^2}{2\sigma^2}\right\} \tag{5.12}$$

and the hyperbolic tangent or sigmoid,

$$K(\mathbf{w}, \mathbf{x}) = \tanh(\kappa \mathbf{w} \cdot \mathbf{x} - \delta) \tag{5.13}$$

The hyperbolic tangent is interesting in that it lets the SVM simulate a neural network. Plugging the tanh kernel into the trained SVM classifer (Equation 5.21) we get

$$f(\mathbf{x}) = \sum_{i=1}^{N_S} \alpha_i y_i \tanh(\kappa \mathbf{s}_i \cdot \mathbf{x} - \delta) + b \tag{5.14}$$

where N_S is the number of support vectors and $\mathbf{s}_i$ are the support vectors. This is the same as a neural network with N_S hidden neurons, each of which has d weights where d is the dimensionality of the dataset, passed to a linear output neuron with N_S weights.

Not every function can be used in an SVM in this manner. It must meet Mercer's condition (Burges, 1998) to be a valid kernel. In particular, there exists a mapping and an expansion

$$K(\mathbf{w}, \mathbf{x}) = \sum_i \Phi(\mathbf{w})_i \Phi(\mathbf{x})_i \tag{5.15}$$

if and only if, for any $g(\mathbf{w})$ such that

$$\int g(\mathbf{w})^2 d\mathbf{w} \quad \text{is finite} \tag{5.16}$$

then

$$\int K(\mathbf{w}, \mathbf{x})\, g(\mathbf{w}) g(\mathbf{x})\, d\mathbf{w} d\mathbf{x} \geq 0. \tag{5.17}$$

The hyperbolic tangent does not always satisfy Mercer's condition, but still may be used in an SVM. If a given training set results in a kernel matrix that is positive semidefinite Hessian, then the SVM will converge perfectly well.

5.2 Kernel Trick and LSVMs

The kernel trick lets the QP algorithm be modified so that, whenever a dot product is taken, it can be replaced with the kernel. Thus, the QP problem becomes

$$\text{minimize} \quad \frac{1}{2} K(\mathbf{w}, \mathbf{w}) + C \sum_{i=1}^{m} \varepsilon_i \tag{5.18}$$

$$\text{subject to} \quad (y_i)(K(\mathbf{w}, \mathbf{x}) + b) \geq 1 - \varepsilon_i \tag{5.19}$$

$$\text{and} \quad \varepsilon_i \geq 0 \text{ for all } i. \tag{5.20}$$

Now, instead of finding the optimal hyperplane in the original data space (which might not be linearly separable) an optimal hyperplane can be found in the higher dimensional space. The

kernel trick keeps the QP computations tractible (Moore, 2001).

5.3 Support Vectors and Classification

In an LSVM, the support vectors are the subset of the dataset which lie adjacent to the optimal hyperplane at a distance of the margin, and, if slack variables are being used, those that lie in the wrong section of the hyperspace. The support vectors alone are enough to define the hyperplane and margin. (These vectors could be thought of as supporting the hyperplane.) Slack parameters are needed because, otherwise, it might be impossible to find the optimal hyperplane in a high dimensional feature space. We can use the same method to classify non-training data points as was used earlier, but with kernels instead:

$$f(\mathbf{x}) = \sum_{i=1}^{N_S} \alpha_i y_i \, K(\mathbf{s}_i, \mathbf{x}) + b \tag{5.21}$$

were N_S is the number of support vectors, and $\mathbf{s}_i$ are the support vectors. Taking the sign of $f(\mathbf{x})$ gives the class (Burges, 1998).

5.4 SVM Implementation

The quadratic programming aspect of the SVM algorithm can be solved by straight-forward application of many pre-existing QP packages that use Newton's method, conjugate gradient, or primal dual methods. For small training sets, the QP problem can be solved analytically, which has a worst case computational complexity of order N_S^3 (inversion of the Hessian), where N_S is the number of support vectors (Burges, 1998). For larger training sets, numeric methods must be used. One problem with using conventional QP techniques is that the entire kernel matrix, which grows quadratically with the number of training samples, must be stored in memory. For example, the kernel matrix for the training data in this project would require over 120 MB of memory using these methods.

As a result, novel SVM techniques have been created that reduce computational complexity. The major ones will be describe here.

Recall that the optimization problem (Cristianini *et al.*, 2000) is:

$$\text{maximize} \quad W(\boldsymbol{\alpha}) = \sum_{i=1}^{\ell} \alpha_i - \frac{1}{2} \sum_{i}^{\ell} \sum_{j}^{\ell} \alpha_i \alpha_j \, y_i y_j \, K(\mathbf{x}_i, \mathbf{x}_j) \tag{5.22}$$

$$\text{subject to} \quad \sum_{i=1}^{\ell} \alpha_i y_i = 0 \tag{5.23}$$

$$\text{and} \quad 0 \leq \alpha_i \leq C, \quad i = 1...\ell \tag{5.24}$$

Where y specifies the class and is either -1 or +1, $\mathbf{x}$ is an input vector, ℓ is the size of the training set, and $C < \infty$ for the 1-norm soft margin case.

5.4.1 Naive Solution: Gradient Ascent

Simple numerical gradient descent can be used to follow the parabolic surface towards the unique global maximum (Cristianini *et al.*, 2000). A learning rate η is needed to update the vector $\boldsymbol{\alpha}$ for each iteration, and must be properly sized to allow for timely convergence, yet prevent oscillatory behavior. The step size for individual elements of $\boldsymbol{\alpha}$ is given by

$$\delta\alpha_i^t = \eta \frac{\partial W(\boldsymbol{\alpha}^t)}{\partial \alpha_i} \tag{5.25}$$

Where t indicates the iteration, and the i subscript shows that the multi-dimensional problem is reduced to a sequence of single-dimensional ones. This technique is not optimal with respect to speed, but it works well for datasets up to a few thousand points. The linear constraint $\sum \alpha_i y_i = 0$ causes a problem. At least two α values must be changed simultaneously to keep the system from leaving the feasible region. This is the basis of *sequential minimal optimization* (SMO).

5.4.2 Chunking and Decomposition

The *chunking* and *decomposition* techniques reduce memory requirements by breaking the problem down into smaller sub-problems (Cristianini *et al.*, 2000).

Chunking

The simplest heuristic is known as "chunking." Here, a small subset, or "chunk," of the training set is used to train an SVM. Once trained, the support vectors are kept, and the remainder of the training data is tested with the SVM. The points that give the worst results are kept, then a new SVM is trained using these points and the previous support vectors as the training set. The process is repeated until some stopping criteria is reached.

Decomposition

Chunking might fail because the kernel matrix may grow too large. Decomposition is a more sophisticated method inspired by chunking. This algorithm updates a fixed number of α_i values while the rest are kept constant. A small subset of training points are used as the *active set*.

Whenever a new training point is added to the active set, another one is removed. The goal is to optimize the global problem by looking at only a subset of the training data at any time.

Chunking and decomposition have not been theoretically proven to converge, but in practise, they work well and permit training sets of tens of thousands of points.

5.4.3 SMO: Sequential Minimal Optimization

Sequential minimal optimization (SMO) is an extreme reduction of the decomposition method to a minimal subset of only two points per iteration (Platt, 1998). The benefit of this is that this minimal subset can be solved analytically without the need of a QP solver.

With each iteration, SMO uses a simple heuristic (based on constraint violation) to choose two points, α_i and α_j, to optimize. All other parameters are assumed to be fixed, and new optimal values of α_i and α_j are determined analytically, after which $\boldsymbol{\alpha}$ is updated.

Compared to other methods, SMO needs more iterations to converge, but each iteration is so fast that the algorithm shows a speedup of orders of magnitude. Other qualities of this method are that it does not need a kernel matrix stored in memory, and also does not need any sort of QP package included as part of the algorithm.

The LIBSVM software package used in this project is based on the SMO algorithm (Chang and Lin, 2005).

Chapter 6

Artificial Neural Networks

A great deal of the research into pattern recognition has been inspired by the workings of the human brain. As such, there have been many attempts to artificially simulate the biological processes that lead to intelligent behavior. At one end of the artificial intelligence spectrum is the field of symbolic reasoning, which attempts to synthesize intelligence through manipulation of high-level cognitive concepts. At the other end, you have the reductionists — those who believe that intelligence can be created by emulating the brain at the level of the smallest building blocks: namely, the neurons.

6.1 Biological Neurons

Neurons, the fundamental structure of the brain, were first described by Cajál in 1911 (Sdorow, 1990). Since then, scientists have discovered many different types of neurons, each of which has a distinct purpose in the brain. In general, a neuron is composed of the following parts (Sdorow, 1990):

- The cell body (or soma), the structural and biochemical center of the neuron;
- The dendrites, branch like receptors on the cell body;
- The axon, an extremely long (relatively speaking) conduit for channeling signals;
- And, the synaptic terminals, a branch-like structure extruding from the end of the axon. These terminals transmit messages to other neurons, muscles, etc. Figure 6.1 shows a simplified diagram of a neuron.

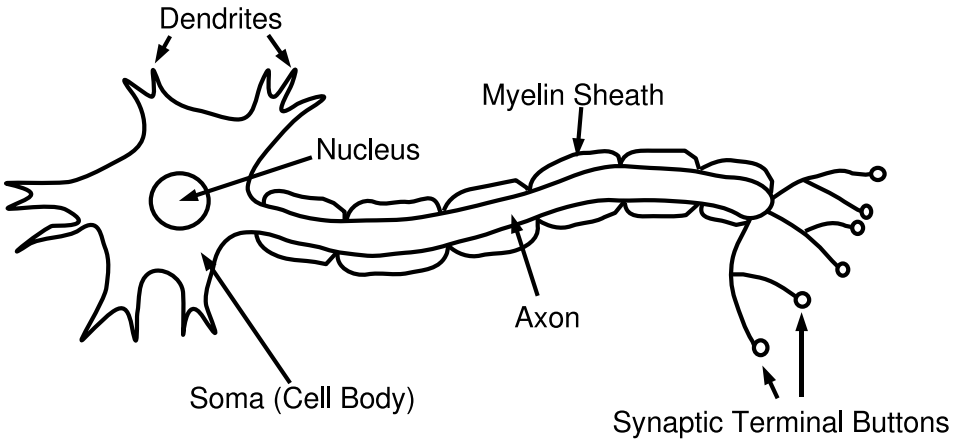

Figure 6.1: Simplified diagram of a biological neuron (Sdorow, 1990).

A neuron acts as an integrator of its inputs. It receives multiple excitatory and inhibitory inputs at the cell body and dendrites, the cell body applies some sort of transfer function to those combined inputs, and sends the result down the axon to other neurons and muscles, in the form of a pulse. This is a simplified explanation though. Ongoing research has shown that neurons are far from simple and are also affected by non-synaptic influences such as biochemicals and drugs.

6.2 Artificial Neurons

A simplified neural model, one that was influential for decades to come, was proposed by McCullogh and Pitts in 1943. In this model, a neuron multiplies the inputs by weights, calculates the sum, and applies a threshold. The result of this computation would then be transmitted to subsequent neurons. The McCullogh-Pitts neuron has been generalized (Hertz *et al.*, 1991) to

$$y_i = f\left(\sum_k w_{ik} x_k + \mu_i\right) \tag{6.1}$$

where x_k are inputs to the neuron i, w_{ik} are weights attached to the inputs, μ_i is a threshold, offset or bias, $f(\cdot)$ is a transfer function and y_i is the output of the neuron. Figure 6.2 shows a diagram of a generalized McCullogh-Pitts neuron.

The transfer function $f(\cdot)$ can be anything. It could be linear, i.e., $y = x$, in which case the neuron simply takes a dot product $\mathbf{x} \cdot \mathbf{w}$. Several such neurons in a layer will calculate a matrix multiplication. The function $f(\cdot)$ could be a non-smooth nonlinear function, such as a unit step (Figure 6.3), which was the basis of the original McCullogh-Pitts neuron. This transfer function, as

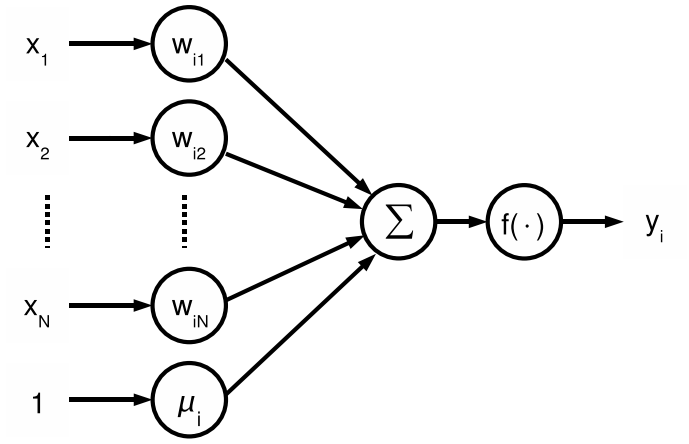

Figure 6.2: Generalized McCullogh-Pitts neuron (Hertz *et al.*, 1991).

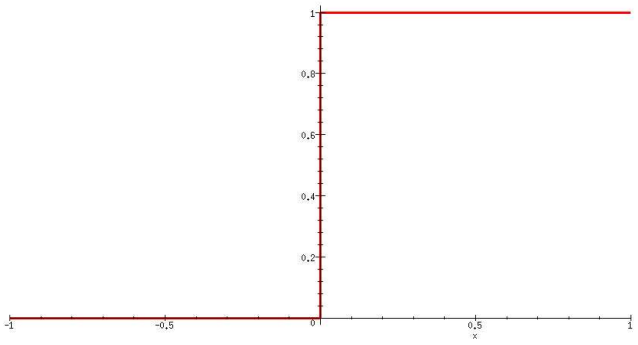

Figure 6.3: The Heaviside unit step function (Hertz *et al.*, 1991).

simple as it may be, gives these neurons the ability to emulate basic boolean functions such as AND, OR, NAND, etc. The function $f(\cdot)$ could also be piecewise linear, like a unit step with a ramp segment connecting the 0 and 1 parts (Figure 6.4). The disadvantage of these functions is that they are not differentiable, and thus not suited to automatic learning.

The most useful transfer functions for artificial neurons are nonlinear and differentiable. The iconic example is the sigmoid or logistic function, $y = 1/(1 + \exp(-x))$ (see Figure 6.5). The hyperbolic tangent, $y = \tanh(x)$ (Figure 6.6) is also commonly used, and in fact it is just a scaled and shifted sigmoid function.

These functions can easily be used for training with error-backpropagation (Section 6.8) because there is a derivative everywhere. The sigmoid curve also has a "squishing" effect, so that extremely high or low input values have only a minor effect on the output, which is asymptotically limited to the range $(0, 1)$.

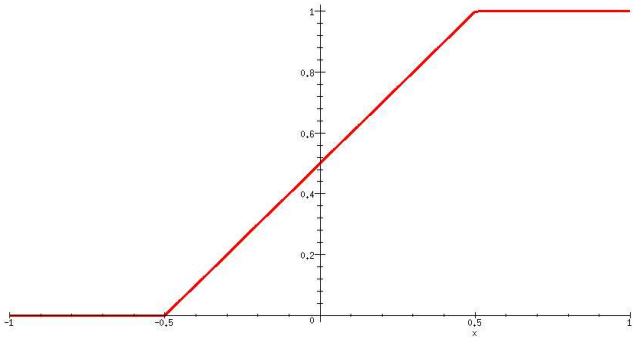

Figure 6.4: A piecewise linear function combining a ramp with a unit step.

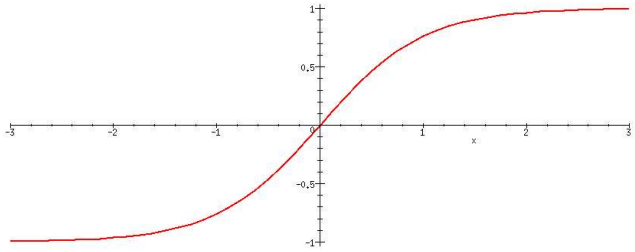

Figure 6.5: The logistic function $y = 1/(1 + \exp(-x))$.

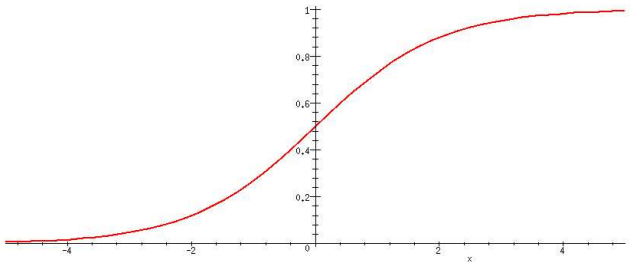

Figure 6.6: The hyperbolic tangent $y = \tanh(x)$.

6.3 Single Neuron Computation

What kinds of computation is a single artificial neuron capable of? (For the remainder of this section, Attention will be given to neurons with a sigmoid activation function. A McCullogh-Pitts neuron can be approximated as a sigmoidal neuron by multiplying the result of the summation by a large number.)

Functionally, a neuron takes an n-dimensional input vector and maps it to a single real value. Specifically

$$y_i = f\left(\sum_x w_{ik}x_k + \mu_i\right) = f(\mathbf{w}_i \cdot \mathbf{x} + \mu_i) \tag{6.2}$$

Provided that $f(\cdot)$ has an inverse, we can rewrite that as

$$f^{-1}(y_i) = \mathbf{w}_i \cdot \mathbf{x} + \mu_i. \tag{6.3}$$

The geometrical consequence of this is that the input space is bisected by an $(n-1)$ dimensional plane. See Section 4 for more information. When $f(\cdot)$ is a sigmoid, the separating hyperplane is commonly considered to be a plane along where $y_i = 1/2$, which is the value of the sigmoid function at the origin. In the earlier discussion, the hyperplane was seen as a strict delimiter of membership. Points at opposite sides of the hyperplane are seen as occupying different classes. The sigmoid function allows a more statistical type of classification, rather than a hard binary "yes/no" or "true/false." Intuitively, this makes sense; on the boundary points of the hyperplane, membership is $1/2$, which implies that points in that area are equally likely to be a member of either class. Near the boundary, the sigmoid function gives values near $1/2$, indicating that the point belongs in one of the classes — but the uncertainty is high. Finally, the farther you get from the hyperplane, the closer the sigmoid function goes to either 0 or 1, indicating a high confidence. This observation has a basis in statistics, and is known as logistic discrimination (Bishop, 1995).

6.4 Logistic Descrimination

Consider two classes that have multivariate Gaussian distributions, and identical covariance matrices $\mathbf{\Sigma}$. The probability that a point $\mathbf{x}$ is a member of class $\mathcal{C}_k$ is

$$p(\mathbf{x}|\mathcal{C}_k) = \frac{1}{(2\pi)^{d/2}|\mathbf{\Sigma}|^{1/2}} \exp\left\{\frac{1}{2}(\mathbf{x}-\mu_k)^{\mathrm{T}}\mathbf{\Sigma}^{-1}(\mathbf{x}-\mu_k)\right\} \tag{6.4}$$

where d is the dimension of the data and μ_k is the mean of the distribution.

We can apply Bayes' theorem to find the posterior probability $\mathbf{x}$ is a member of class $\mathcal{C}_1$:

$$P(\mathcal{C}_1|\mathbf{x}) = \frac{p(\mathbf{x}|\mathcal{C}_1)P(\mathcal{C}_1)}{p(\mathbf{x}|\mathcal{C}_1)P(\mathcal{C}_1) + p(\mathbf{x}|\mathcal{C}_2)P(\mathcal{C}_2)} \tag{6.5}$$

$$= \frac{1}{1 + \exp(-a)} \tag{6.6}$$

where

$$a = \ln \left[\frac{p(\mathbf{x}|\mathcal{C}_1)P(\mathcal{C}_1)}{p(\mathbf{x}|\mathcal{C}_2)P(\mathcal{C}_2)} \right]. \tag{6.7}$$

This result justifies the behavior of a neuron with a sigmoid activation function. It shows that for a common type of discrimination problem, a neuron can produce an output that is identical to the posterior probability of an input vector belonging to a specific class (Bishop, 1995).

6.5 Training a Neuron

For simple problems, it is possible to scrutinize the data and manually determine which weights are needed to give the desired behavior. For example, this process can be done to create boolean logic functions with McCullogh-Pitts neurons. For more complicated neural processing problems, the optimal weights are not obvious. An automatic procedure is needed which will configure the neuron based on a training dataset.

6.6 Perceptrons

A perceptron is the simplest form of artificial neural network capable of classifying linearly separable patterns, and was first studied by Rosenblatt in 1958 and 1962. Rosenblatt's perceptron was based on the McCullogh-Pitts neuron, in which a weighted sum of inputs is subjected to a step function such as the unit step (Heaviside) or sign function. Rosenblatt developed an automatic learning procedure known as the perceptron algorithm. This simple method can be explained as follows (Haykin, 1994).

The Perceptron Algorithm

1. A training vector $\mathbf{x}$ is presented to the perceptron. One of the elements of $\mathbf{x}$ is always 1, which acts as a bias or offset value.
2. The output value of the neuron is evaluated.
3. If $\mathbf{x}$ is properly classified, then no changes are made to the weight vector $\mathbf{w}$.
4. If $\mathbf{x}$ is misclassified, then $\mathbf{w}$ is updated using the rule:
   (a) $\mathbf{w}_{new} = \mathbf{w}_{old} - \eta\mathbf{x}$ if $\mathbf{w} \cdot \mathbf{x} > 0$ and $\mathbf{x}$ belongs to class $\mathcal{C}_0$;
   (b) $\mathbf{w}_{new} = \mathbf{w}_{old} + \eta\mathbf{x}$ if $\mathbf{w} \cdot \mathbf{x} \leq 0$ and $\mathbf{x}$ belongs to class $\mathcal{C}_1$.

Where the classes $\mathcal{C}_0$ of $\mathcal{C}_1$ are indicated when the output of the neuron is 0 or 1 respectively, when using a step activation function. The value of η is the learning rate parameter. It is a small value that controls the size of changes to the weight vectors during training.

Rosenblatt proved, in his *perceptron convergence theorem*, that the perceptron algorithm is guaranteed to converge and find a solution — but only if the classes in the training set are linearly separable. See Figure 6.7 for an example.

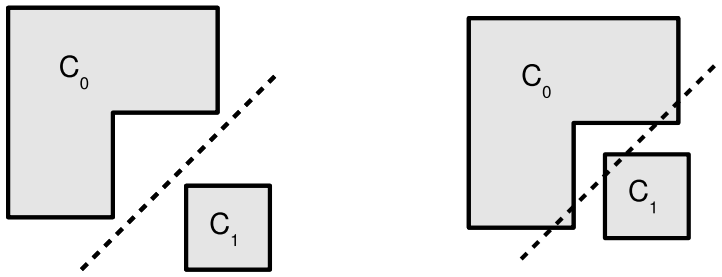

Figure 6.7: Linearly separable vs. non-linearly separable classes.

The perceptron created a stir in the field of machine learning in the 1960's. However, the inability to classify patterns that are not linearly separable was its fatal flaw. Finally, in 1969, Minsky and Papert elucidated the limitations of the perceptron in their book *Perceptrons*. They also conjectured that multilayer perceptrons would be a "sterile" area of research. History has proven them wrong, but their influential book set ANN research back by several years.

6.7 Steepest Descent and the LMS Algorithm

The least mean square (LMS) algorithm is a very simple technique for linear adaptive filters. It is also known as the *delta rule* or *Widrow-Hoff rule* (Widrow and Hoff, 1960). The simplicity and reliability of the LMS algorithm has made it the standard for comparing other adaptive filtering algorithms. The well-developed field sets the stage for other training algorithms for nonlinear ANNs, and is relevant to linear neurons, which are identical to linear filters.

6.7.1 Steepest Descent

The basic problem statement with linear filters is (Haykin, 1994):

> Given a set of corresponding observations $(y_i, \mathbf{x}_i)$ where y_i is a real value and $\mathbf{x}_i$ is a vector of reals, can we find an optimal weight vector $\mathbf{w}$ that will allow us to predict y_i

with

$$y_i = \mathbf{w} \cdot \mathbf{x}_i \tag{6.8}$$

such that the error of the predicted y_i is minimized?

If d_i is the desired response, then the error is

$$e_i = d_i - y_i \tag{6.9}$$

and an overall performance measure is the mean squared error (MSE), J,

$$J = \frac{1}{2}E[e^2] \tag{6.10}$$

where E is the statistical expectation operator, and the factor $\frac{1}{2}$ is used for convenience. The optimal $\mathbf{w}$ which minimizes J is known as a *Weiner filter*.

For certain types of systems, such as a spatial filter, the MSE J surface takes on a multidimensional "bowl" shape with respect to the vector $\mathbf{w}$. There is only one global (and no local) minima on this surface, at which point $\mathbf{w}$ is optimal. This minima can be solved directly by means of the *Weiner-Hopf equations* (Haykin, 1994) which requires matrix inversion of autocorrelation values. It can also be found by the iterative *method of steepest descent*, which involves traveling along the error surface in the direction of the gradient

$$\nabla_{w_k} J = \frac{\partial J}{\partial w_k}, \quad k = 1, 2, ..., p \tag{6.11}$$

which is the differentiation of the cost function J w.r.t. $\mathbf{w}$. Expanding the equation,

$$\nabla_{w_k} J = -r_{dx}(k) + \sum_{j=1}^{p} w_j r_x(j,k) \tag{6.12}$$

where r_{dx} is the autocorrelation and r_x is the cross-correlation. Numerically, the weight vector w can be adjusted slightly by means of the rule

$$w_{k_{new}} = w_k + \delta w_k, \quad k = 1, 2, ..., p \tag{6.13}$$

where

$$\delta w_k = \eta \nabla_{w_k} J \tag{6.14}$$

where η is a positive constant called the *learning rate parameter*. Since steepest descent requires correlation values to be known, it is not suitable for some environments, such as those where data samples are disordered. In these situations, we need to use estimates of r_x and r_{dx}. The least mean square algorithm provides these.

6.7.2 The Least Mean Square Algorithm

The least mean square (LMS) algorithm uses instantaneous estimates for the autocorrelation r_x and the cross-correlation r_{xd} (Haykin, 1994). The estimates are

$$\widehat{r}_x(i,j) = x_j x_k \tag{6.15}$$

$$\widehat{r}_{dx}(k) = x_k d \tag{6.16}$$

When these are plugged back into the the steepest descent algorithm, the rule for updating $\mathbf{w}$ becomes

$$\widehat{w}_{k_{new}} = \widehat{w}_k + \eta[d - y]x_k, \quad k = 1, 2, ..., p \tag{6.17}$$

In effect, LMS minimizes the instantaneous squared error.

LMS is also known as the *stochastic gradient descent* algorithm, because $\widehat{w}$, the estimate of w, follows a random trajectory, unlike the steepest descent method which follows a smooth one.

An added benefit of LMS is that it does not rely on the signal being stationary. It can "track" a signal whose statistics change over time, because it uses instantaneous estimates.

6.8 Multilayer Perceptrons and Back Propagation

Multilayer perceptrons (MLPs) have been used successfully in a wide variety of applications. They commonly use *error back-propagation* as the training algorithm, which is a many-layered generalization of the ubiquitous LMS algorithm.

Backpropagation consists of a *forward pass* and a *backward pass*.

In the forward pass, the input vector (training pattern) is presented to the hidden layer. The outputs are calculated and passed on to the output layer which applies another activation function to produce the final output vector. In the forward pass, no weights are changed (Haykin, 1994).

The next phase, the backward pass, is where training occurs and the weights are adjusted slightly. In this phase, the outputs from the forward pass are subtracted from the desired or target values. This error signal is passed back through the network, and the weights are adjusted incrementally to reduce the error. This is called *error back-propagation* (Haykin, 1994).

6.9 Characteristics of Multi-Layer Perceptrons

MLPs have three distinguishing characteristics (Haykin, 1994).

1. The neural activation functions are *nonlinear* and *smooth* (differentiable everywhere). A common function is the logistic curve $y = 1/(1 + \exp(-x))$;
2. The network contains one or more hidden layers, that lie between the input and output layers of the network. These hidden layers allow the network to progressively extract more meaningful features, thus permitting complex recognition tasks;
3. The neurons in the network are highly interconnected, such that there are a large number of weights relative to the number of neurons. Typically, every neuron in a layer is connected to every neuron of the subsequent layer.

These three characteristics, plus the ability to learn, gives ANNs great power. The same characteristics makes theoretical analysis difficult, because of the nonlinearity and the great degree of interconnections. Hidden neurons also make the processes harder to observe and visualize. Furthermore, ANNs do not have unlimited capabilities — the Curse of Dimensionality requires that preprocessing be done to reduce the dimensionality of the input space (Bishop, 1995).

Like some other important scientific discoveries, backpropagation was discovered independently by several people. It was first mentioned in Werbos' Ph. D. thesis in 1975, and rediscovered by Rumelhard, Hinton and Williams in 1986. Similar generalizations were discovered separately by both Parker and LeCun in 1985 (Haykin, 1994).

Backpropagation was a landmark because it was computationally efficient. Although it is not guaranteed to find a solution to all solvable problems, it works well enough to show that Nimsky and Papert's pessimistic prediction of multilayer perceptrons was wrong.

6.10 Derivation of the Backpropagation Algorithm

In backpropagation (Rumelhart *et al.*, 1986; Masters, 1993; Haykin, 1994; Bishop, 1995), the error signal of an output neuron j at the n^{th} training pattern is defined as

$$e_j = d_j(n) - y_j(n) \tag{6.18}$$

where d_j is the desired value and y_i is the actual output of the neuron. The instantaneous value for the squared error of neuron j is $\frac{1}{2}e_j^2(n)$.

The instantaneous value $\mathcal{E}(n)$ is the sum of the squared errors obtained by summing the squared errors of the output layer:

$$\mathcal{E}(n) = \frac{1}{2}\sum_{j \in C} e_j^2(n) \tag{6.19}$$

where C is the set of all output neurons. The output neurons are the only neurons where errors can be directly calculated.

If N is the total number of patterns (training examples) in the training set, the *average squared error* is obtained by summing $\mathcal{E}(n)$ over all n and normalizing for N:

$$\mathcal{E}_{\text{av}} = \frac{1}{N}\sum_{n=1}^{N} \mathcal{E}(n) \tag{6.20}$$

$\mathcal{E}_{\text{av}}$ is a "cost function" for a given training set. The goal of training is to adjust the free parameters of the network to minimize this cost function. This minimization can be done through a process similar to the LMS algoritms. A common form of backpropagation is to update the weights after each pattern is presented to the network. The adjustments are calculated according to the error for a particular training pattern. The average of all these incremental changes is an estimate of the true change that would occur if all weights were adjusted at once to minimize the cost function for the entire training set.

6.11 The Mathematics of the Output Layer

Given a neuron j receiving inputs from the previous layer of neurons, $i = 1$ to P, the summation value v_j at the neuron (before nonlinearity) is

$$v_j(n) = \sum_{i=1}^{P} w_{ji}(n) y_i(n) \tag{6.21}$$

(The threshold or bias input is assumed to be one of the elements of y_i with a constant value.)

Thus the output of neuron j at iteration n is

$$y_j(n) = \varphi_j(v_j(n)), \quad \text{where } \varphi \text{ is a nonlinear function.} \tag{6.22}$$

If we apply the lessons learnt from the LMS algorithm, we can find an equation that gives an incremental adjustment $\Delta w_{ji}(n)$ for the weight w_{ji}, which is proportional to the instantaneous gradient

$$\frac{\partial \mathcal{E}(n)}{\partial w_{ji}(n)}. \tag{6.23}$$

Using the chain rule, the derivation (Haykin, 1994) follows:

$$\frac{\partial \mathcal{E}(n)}{\partial w_{ji}(n)} = \frac{\partial \mathcal{E}(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} \frac{\partial v_j(n)}{\partial w_{ji}(n)}. \tag{6.24}$$

This gradient represents a sensitivity factor. It shows how influential a specific weight w_{ji} is on the error, and determines the direction of gradient traversal in weight space for w_{ji} (Haykin, 1994).

Differentiating 6.19 w.r.t. $e_j(n)$ we get

$$\frac{\partial \mathcal{E}(n)}{\partial e_j(n)} = e_j(n) \tag{6.25}$$

Differentiating 6.18 w.r.t. $y_j(n)$ we get

$$\frac{\partial e_j(n)}{\partial y_j(n)} \tag{6.26}$$

Differentiating 6.22 w.r.t. $v_j(n)$ we get

$$\frac{\partial y_j(n)}{v_j(n)} = \varphi'(v_j(n)) \quad \text{where } \varphi' \text{ is the derivative of } \varphi. \tag{6.27}$$

Finally, differentiate 6.21 w.r.t. w_{ji} to get

$$\frac{\partial v_j(n)}{\partial w_{ji}(n)} = y_i(n). \tag{6.28}$$

Combining equations 6.25 to 6.28 into 6.24 gives

$$\frac{\partial \mathcal{E}(n)}{\partial w_{ij}(n)} = -e_j(n)\ \varphi_j'\left(v_j(n)\right)\ y_i(n) \tag{6.29}$$

The incremental correction $\Delta w_{ji}(n)$ is defined by the *delta rule* (Rumelhart *et al.*, 1986)

$$\Delta w_{ji}(n) = -\eta \frac{\partial \mathcal{E}(n)}{\partial w_{ji}(n)} \tag{6.30}$$

where η is the *learning rate parameter* of the backpropagation algorithm. The minus sign indicates that we want to descend the gradient in weight space, to move towards lower error. Furthermore, using 6.29 in 6.30 gives

$$\Delta w_{ji}(n) = \eta \delta_j(n) y_i(n) \tag{6.31}$$

where the local gradient $\delta_j(n)$ is defined as

$$\delta_j(n) = -\frac{\partial \mathcal{E}(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_i(n)} \frac{\partial y_i(n)}{\partial v_j(n)} = e_j(n)\ \varphi'\left(v_j(n)\right). \tag{6.32}$$

From 6.32 we see that the local gradient is the product of the error signal $e_j(n)$ and the derivative $\varphi'\left(v_j(n)\right)$ of the associated activation function.

Hence, this is the core of the backpropagation algorithm for a single neuron. Application to output layer neurons is straightforward since these neurons know what the expected value of $y_j(n)$ is and thus the error signal $e_j(n)$.

The problem becomes more difficult for hidden layer neurons, because the error signal is not obvious.

6.12 Gradient Descent for Hidden Neurons

Because the error signal for a hidden neuron is not directly known, it must be deduced by recursively scrutinizing all the neurons to which it is connected. As Haykin (1994) says, "this is where the development of the back-propagation algorithm gets complicated."

Given Equation 6.32, we can redefine the local gradient $\delta_j(n)$ for hidden neuron j as

$$\delta_j(n) = \frac{\partial \mathcal{E}(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} \tag{6.33}$$

$$= -\frac{\partial \mathcal{E}(n)}{\partial y_j(n)} \varphi_j'\left(v_j(n)\right) \tag{6.34}$$

Applying Equation 6.19 to find the instantaneous error for the hidden layer neurons, we get

$$\mathcal{E} = \frac{1}{2} \sum_{k \in C} e_k^2(n) \tag{6.35}$$

(This is after we have recalibrated our frame of reference so that hidden neurons are indicated by j and output neurons by k.)

Differentiating 6.35 w.r.t. the function signal $y_j(n)$ we get

$$\frac{\partial \mathcal{E}(n)}{\partial y_j(n)} = \sum_k e_k \frac{\partial e_k(n)}{\partial y_j(n)}. \tag{6.36}$$

Apply the chain rule for the partial derivative $\frac{\partial e_k(n)}{\partial y_j(n)}$ to get the equivalent form (Haykin, 1994)

$$\frac{\partial \mathcal{E}(n)}{\partial y_j(n)} = \sum_k e_k(n) \frac{\partial e_k(n)}{\partial v_k(n)} \frac{\partial v_k(n)}{\partial y_j(n)}. \tag{6.37}$$

Since we are dealing with the hidden layers, the error

$$e_k(n) = d_k(n) - y_k(n) \tag{6.38}$$

becomes

$$e_k(n) = d_k(n) - \varphi_k\left(v_k(n)\right) \text{ where neuron } k \text{ is an output node.} \tag{6.39}$$

Hence,

$$\frac{\partial e_k(n)}{\partial v_k(n)} = -\varphi'\left(v_k(n)\right). \tag{6.40}$$

For neuron k, the summation of the inputs is

$$v_k(n) = \sum_{j=0}^{q} w_{kj}(n) y_j n \tag{6.41}$$

where q is the total number of inputs (excluding the threshold) applied to neuron k. The weight w_{k0} is equal to the threshold $\theta_k(n)$ applied to neuron k, and the corresponding input y_0 is fixed at -1.

Differentiating Equation 6.41 gives

$$\frac{\partial v_k(n)}{\partial y_i(n)} = w_{kj}(n). \tag{6.42}$$

Then, using Equations 6.40 and 6.37 we get the desired partial derivative:

$$\frac{\partial \mathcal{E}(n)}{\partial y_j(n)} = -\sum_k e_k(n) \varphi_k'\big(v_k(n)\big) w_{kj}(n) \tag{6.43}$$

$$= -\sum_k \delta_k(n) w_{kj}(n) \tag{6.44}$$

Finally, plugging Equation 6.44 into 6.34, we get the local gradient $\delta_j(n)$ for a hidden neuron j (Haykin, 1994):

$$\delta_j(n) = \varphi_j'\left(v_j(n)\right) \sum_k \delta_k(n) w_{kj}(n). \tag{6.45}$$

6.13 The Two Passes

The feed-forward and back-propagation phases do not happen simultaneously. Training occurs in two stages. For the forward pass, a training pattern is presented to the input layer, all the activations of the neurons are calculated, and the final error at the output layer is calculated. In the back-propagation phase the errors are used to determine the adjustments to the weights in the network, and the weights are changed. Back-propagation has a variety of parameters that can be tweaked for different scenarios. In the following sections, a brief explanation is given on fine-tuning backprop and its cousins.

6.14 Nonlinear Activation Function

The ability of a backpropagation network to be trained to emulate a nonlinear function relies on the activation function of the neurons being nonlinear and differentiable. Linear functions would be pointless — the network would simply become a sequence of matrix multiplications (which could be boiled down to a single matrix). Differentiablity is required for the backpropagation algorithm to succeed, which rules out the old-fashioned perceptron functions such as the unit step or the clipped ramp.

The most popular activation function is the logistic function, $y = \varphi(x) = 1/(1 + \exp(-x))$ (see Figure 6.6). This function also happens to have a numerically efficient calculation for the derivative:

$$\varphi'(x) = \frac{\exp(-x)}{\left[1 + \exp(-x)\right]^2} = \varphi(x)\left[1 - \varphi(x)\right]. \tag{6.46}$$

This would have helped increase the logistic function's popularity in the early days of backpropagation research, when CPU cycles were much harder to come by.

The hyperbolic tangent (Figure 6.5) is another popular activation function:

$$\varphi(x) = a \tanh(bv) = \frac{2a}{1 + \exp(-bx)} - a \tag{6.47}$$

As you can see, tanh is a scaled and shifted version of the logistic function. An ANN may learn faster when using tanh, because odd functions, such that $f(-x) = -f(x)$, can generate both

positive and negative values, unlike the logistic function, which gives only positive values. Although, a logistic neuron can be made odd merely by setting its bias input to $-\frac{1}{2}$, and the bias weight to 1.

Theoretically, any nonlinear differentiable function could be used. A parabola, such that $\varphi(x) = x^2$ could be used in a backpropagation network. So could the exponential $\varphi(x) = e^x$. However, these would not be stable in a neural network, as they could generate numerically enormous output values which would change by large amounts even if the preceding weights were adjusted by a very small amount. Sigmoid functions prevent this instability because they have a "squishing" characteristic. The point of greatest variability is were the derivative reached its maximum, at $x = 0$. As the input value increases or decreases to infinity the output converges to an asympotic maximum or minimum. According to Rumelhart *et al.*(1986a), this feature contributes to the stability of the algorithm.

6.15 The Learning Rate

Traversal of the error surface in backpropagation is implemented numerically. The parameter $\Delta w_{ij}(n)$ provides an estimate of the gradient, which is multiplied by the learning rate parameter η, to give a discrete step with which to modify the weight w_{ij}. Decreasing this step size by reducing η seems to be an obvious way to improve the smoothness and accuracy of the gradient descent. However, one tradeoff is that training will take longer with smaller η. This slowdown might be acceptable in some applications, but for others, timely training might be needed.

Increasing η will decrease the training time of the network, but if it is increased too much, the weights will become unstable and never converge to a good solution. If you imagine that the weights are attempting to reach a global minimum on the error surface, but the steps are too big, the minimum will never be reached. The weight vector will overshoot it repeatedly. This is known as *oscillatory* behavior (Haykin, 1994).

The learning rate could also be scheduled to change during the learning process. This would allow a larger learning rate at the onset of training, which would speed the traversal of the weight vector over the error surface. Then the learning rate could be reduced to prevent the weight vector from overshooting the minimum on the error surface. The learning rate could be scheduled in any manner. Distinct values could be used at different times, or the rate could be tied to a linear or exponential curve. The point is to decrease the learning rate near the end of training to decrease oscillations around whatever minima has been found by that time (Haykin, 1994).

6.16 Pattern and Batch Mode

In the *pattern mode* of training, the weights of the network are updated each time a pattern is presented. Processing all patterns in the training set is called an *epoch*. Usually, a network is trained over multiple epochs until a stopping criteria is reached. In pattern mode, it is desirable to randomize the order of patterns between epochs to avoid the possibility of cycles in the evolution of the weight vectors.

In *batch mode* training, the weights are left untouched during an epoch, then updated all at once between epochs. The weight deltas are determined by descending the gradient of the $\mathcal{E}_{\text{av}}$ average error surface (as opposed to the instantaneous error $\mathcal{E}(n)$ for pattern mode training).

The weight adjustment for batch mode is

$$\Delta w_{ji} = -\frac{\eta}{N} \sum_{n=1}^{N} e_j(n) \frac{\partial e_j(n)}{\partial w_{ji}} \tag{6.48}$$

where N is the number of training patterns.

Each mode has its advantages and disadvantages. Pattern mode needs less storage for weight values, is more stochastic and less prone to settling on a local minimum. Batch mode provides a better estimate of the error gradient. In general, the effectiveness of each depends on the problem at hand (Hertz *et al.*, 1991).

6.17 Stopping Criteria

In a theoretical sense, the back-propagation algorithm has not been shown to converge (Bishop, 1995). Nor does it converge numerically in practise. Because of the stochastic nature of training, continued presentation of training patterns to a network will always result in changes to the weights — waiting for the weights to stop changing would be an inadequate stopping criteria. There are many types of stopping criteria; this section describes some common ones.

6.17.1 Gradient Convergence

One method, due to Kramer and Sangiovanni-Vincentelli (1989), is to stop when the Euclidean norm of the gradient vector reaches a small enough threshold value. The thinking behind this is, as the weight vector approaches the minimum on the error surface, it gets nearer to the "flat" part of the minimum. At the minimum of the surface, the derivative, and therefore the gradient, ought to be zero. This criteria may require long training times, and it needs additional computation of the gradient vector.

6.17.2 Accuracy Convergence

Another criteria is based on accuracy convergence. Training continues until the absolute change in the average error, $\mathcal{E}_{\text{av}}$, reaches a small enough value.

6.17.3 Error Target

A third criteria is to train until $\mathcal{E}_{\text{av}}$ reached a target error rate τ. This method might never halt if the network is incapable of achieving $\mathcal{E}_{\text{av}} \leq \tau$. On the other hand, if the network does reach $\mathcal{E}_{\text{av}} \leq \tau$, it could possibly be a suboptimal network that is capable of a much lower error rate if the training were allowed to continue.

6.17.4 Hybrid Criteria

A hybrid criteria proposed by Kramer and Sangiovanni-Vincentelli (1989) combines Euclidean norm of the gradient with the targeted $\mathcal{E}_{\text{av}}$ both mentioned above. With this method, the training stops if either one of these criteria are met. This allows training to end even if one of the indicators is "stuck."

6.17.5 Peak Generalization

This criteria incorporates one of the most important properties of a pattern recognition machine — the ability to generalize. After each training epoch, the network is tested with a cross-validation (CV) data set. So long as the network is improving in its generalization capability, the error rate of the CV data set will continue to decrease. Once overtraining starts to take place, $\mathcal{E}_{\text{CV}_{\text{av}}}$ will reach a minimum and then start increasing, at which time the training will be halted (Bishop, 1995).

6.17.6 Constant Training Time

The simplest stopping criteria is to train the network for a fixed number of epochs, or for a certain duration, then stop. Obviously, this method is guaranteed to halt. This criteria would be useful in a real-time environment were network training is subject to strict time constraints. Overtraining might be a problem with this method, but it can be alleviated by combining with the "peak generalization" method above.

6.17.7 Noise Issues

In practise, the parameters mentioned above do not change smoothly from epoch to epoch. There is usually an element of noise thrown in, so that, say, the $\mathcal{E}_{\text{av}}$ value, will not appear to be smoothly decreasing over time, but will be somewhat jagged. Usually, simple filtering is applied to the parameter, such as requiring the criteria to be "true" for a certain number of sequential epochs.

6.18 Initialization

Ideally, we would like to start training a neural network with an initial set of weights that will speed up and reduce the total training time. However, since the whole exercise of training is to determine the weights, guessing them at the onset is unlikely. It might be possible to use prior information to embed some preconceived notions into the weights, but that in itself is an unsolved problem, plus you run the risk of initializing the network in a suboptimal way so that the global error minimum is never reached.

For these reasons, neural networks are usually initialized with pseudo-random values (which in itself is a pre-conceived notion.) These values are uniformly distributed within a small range. Care must be taken here, because a poor initialization might lead to *premature saturation* (Lee *et al.*, 1991). This is a condition where too many neurons have an overly positive or negative sum of inputs, and are giving outputs that are far into the "flat" regions of the sigmoid function where it is approaching its asymptotic limits. Here, the derivative of the sigmoid curve is very low, and the delta rule (which relies on the derivative to determine step size) will adjust the weight values by an extremely small quantity. The symptom of this will be a time after the onset of training when the error rate $\mathcal{E}_{\text{av}}$ changes exceedingly slowly. On the error surface, the weights are situated in a nearly flat plateau area.

Lee *et al.*(1991) and Russo (1991) give advice in reducing the probability of premature saturation. For this thesis, the neural weights were initialized with random values from $-1/2$ to $+1/2$, which gave good results.

6.19 Variations on the Delta Rule

Bishop (1995) notes that the old delta rule for backpropagation is not the most effective means of gradient descent. Many modifications have been proposed over the years. Most are *ad hoc*, but some are theoretically well founded.

None of the alternative techniques was implemented in this thesis. The goal here was to compare backpropagation to SVM and KDE, not to compare it to itself. Nevertheless, here are some variations on the theme, briefly described.

6.19.1 Momentum

The simple method adds "inertia" to the trajectories on the error surface, which has the effect of increasing the learning rate along directions where the gradient is smooth and monotonic. This keeps the weight vector from getting bogged down in a long narrow trough on the error surface. The old delta rule will cause the weight vector to bounce from side to side in the "ditch," making very slow progress along the overall gradient. A momentum term, μ, helps by increasing η along the length of the ditch (Plaut, *et al.*1986).

6.19.2 Bold Driver

This dramatically named technique (Vogl *et al.*1988) continually adjusts the learning rate depending on the error performance of the network. If the error has increased after a step, then it is assumed that the step overshot a minimum. The step is then undone, the learning rate η is reduced by a factor σ, and training is continued. If a learning iteration reduces the error, then the learning rate η is increased by a factor ρ.

6.19.3 Quickprop

This technique, due to Fahlmann (1988), uses two sequential error evaluations and a gradient evaluation to interpolate and model the error surface as a hyper-parabola. Then, the weight vector is adjusted to coincide with the minimum of the parabola. Bishop (1995) notes that several fixes are needed to get quickprop to work in practise. It is not too difficult to imagine an error surface that would confound an algorithm that expects a parabolic shape.

6.19.4 Many η's

The "long narrow trough" (a.k.a. "ditch") mentioned earlier is such a common problem that much work has been done to try to minimize the negative effects it has on gradient descent. Intuitively, it might be a good idea to use a larger learning rate along the long shallow part of the valley, and a smaller rate along the steep narrow part, to reduce oscillations. Jacobs (1988) investigated some schemes in which each weight value in the network was given its own learning rate. In effect, this gives each dimension of the error surface a different η, and thus, things like long narrow valleys might benefit, because the axes of the valley might be oriented orthogonally. Jacobs redefined the delta rule to handle multiple η's, and also developed the *delta-delta* and *delta-bar-delta* methods, which are attempts to incorporate heuristics into gradient descent and reduce oscillations in the error surface. The delta-delta method does not work very well in practise, and the delta-bar-delta (which is the delta-delta rule with some tweaks) works better, but has four parameters that the

user has to supply. The method also assumes that the weights are independent, when in practise some of them are strongly coupled.

6.19.5 Summary of Delta Rule Alternatives

The delta rule is still the simplest of the gradient descent algorithms, and works well for many situations. The alternative rules are useful when confonted with a problem that converges exceedingly slowly with the delta rule.

From a practical perspective, if you are doing research on a problem, and you are not sure about how to set the free parameters (such as η), you have to run trials with different parameter values to see which works best. Say, for example that you want to try three values for the learning rate η: low, medium and high. Then you have to run three trials. If you add a momentum term, and you want to test three values for that, then you have $3^2 = 9$ trials to run. If you are using the bold driver technique, with four free parameters, and you wish to cover the parameter space with three values for each parameter, then you will have to run $3^4 = 81$ trials. The number of trials increases exponentially with the number of parameters. The Curse of Dimensionality strikes in unexpected places. (These examples ignore the fact that the user must choose the number of neurons in the hidden layer, which creates yet another dimension on the parameter space.)

6.20 Neurons in the Hidden Layer

Changing the number of neurons in the hidden layer is a way to adjust the precision of the network with respect to the training set. At one extreme, you could have a hidden layer of only a single neuron, which would merely bisect the input space into two classes along a hyperplane (with a smooth transition due to the sigmoid activation function) (Bishop, 1995). The other extreme is a hidden layer that is so large that it actually emulates a look-up table of the training set, with perfect accuracy (Haykin, 1994). Either extreme is useless for real applications, so the number of hidden neurons must be determined by experimentation.

One strategy is to train a network with ever-increasing numbers of neurons, then testing the performance on a *cross-validation* (CV) set, an additional data set that was not used in training (Bishop, 1995). The optimum is determined to be the maximum number of neurons that yields a reduction in the CV error. When the CV error increases, this indicates that the network is behaving like a lookup table of the training set and is not generalizing well to the CV set (Bishop, 1995).

Pragmatics must also be taken into account. Adding neurons to the hidden layer will increase the computation time. This time increases as $\mathcal{O}(n)$ (Bishop, 1995). In other words, double the hidden layer neurons and you double the training time. A complexity of $\mathcal{O}(n)$ is efficient, but increasing the hidden layer still reduces the number of trials that an experimenter can run.

Chapter 7

Other Statistical Techniques

A variety of well-known statistical techniques are used for different parts of this project.

KDE, or *kernel density estimation*, is used as a classifier and compared to SVMs and ANNs.

The *chi-square test* is used as a post-processing step for comparing classification results to rows in a confusion matrix.

The *ROC*, or *receiver operating characteristic*, is used to set the discrimination threshold that is used to discard weakly classified results with the ANN and SVM.

The *confusion matrix* is used to describe multi-class performance, and, as mentioned above, is used during post-processing.

In this chapter, each of these is described in more detail.

7.1 Kernel Density Estimation

Kernel density estimation, or *KDE*, is a simple method for estimating the probability density function given a set of points sampled from an unknown distribution (Bishop, 1995).

KDE is an offshoot of the *sliding histogram* technique, in which a fine-grained estimate is found by calculating a regular histogram repeatedly, but with the bin boundaries varying over the range of a bin-width along an axis (Scott, 1992). It turns out that this is equivalent to convolving the data with a cube-shaped kernel known as a *Parzen window* (Scott, 1992).

The Parzen window gives the number of points in a cube-shaped region, and may be defined as

$$H(\mathbf{u}) = \begin{cases} 1 & |u_j| < \frac{1}{2}, \quad j = 1, ..., d \\ 0 & \text{otherwise} \end{cases} \tag{7.1}$$

where d is the dimensionality of the data. This kernel corresponds to a hypercube of unit size centered on the origin.

The total number of points falling inside the hypercube is

$$K = \sum_{n=1}^{N} H\left(\frac{\mathbf{x} - \mathbf{x}^n}{h}\right) \tag{7.2}$$

where $\mathbf{x}$ is the point where the estimate is desired, $\mathbf{x}^n$ are the datapoints, N is the number of points, and h is the "bandwidth" or smoothing parameter, which, for a cubical kernel, is the length of each side.

The probability density can thus be estimated by

$$\widetilde{p}(\mathbf{x}) = \frac{1}{N} \sum_{n=1}^{N} \frac{1}{h^d} H\left(\frac{\mathbf{x} - \mathbf{x}^n}{h}\right) \tag{7.3}$$

which is the count, K, normalized so that $\widetilde{p}(\mathbf{x}) \leq 1$.

Since the boundaries of the cubical kernel are discontinuous steps, the resulting estimate will also be discontinuous. This can be eliminated by using a kernel with smooth boundaries, such as the Gaussian kernel (Bishop, 1995)

$$\widetilde{p}(\mathbf{x}) = \frac{1}{N} \sum_{n=1}^{N} \frac{1}{(2\pi h^2)^{d/2}} \exp\left\{-\frac{\|\mathbf{x} - \mathbf{x}^n\|^2}{2h^2}\right\} \tag{7.4}$$

The width parameter, h, is a free variable that must be chosen by the user. If it is too large, then detail will be lost. If it is too small, the estimate will be too rough with too much fine structure, and will not generalize well. In the worst case, an estimate with an overly small width will behave like a lookup table of the dataset.

Even if a good width parameter is found, KDE has a fundamental "flaw." Rosenblatt (1956) has shown that, for a finite dataset, there is no non-negative estimator which is unbiased for all continuous density functions.

Nevertheless, KDE is a well-investigated (if lacking in consensus) field of endeavour. In statistical research, density estimation is a powerful methodology for gaining insight into data, for example, revealing multimodal distributions (Jones *et al.*, 1996).

The primary problem in KDE research is to automatically determine the bandwidth value. A multitude of techniques exist and are discussed in depth in (Scott, 1992). To summarize, these can be separated into two classes: first generation, and second generation (Jones *et al.*, 1996).

The old "first generation", or classical, methods are:

- Visual inspection;
- Rules of thumb;
- Least squares cross validation;
- Biased cross validation.

The more modern "second generation" methods are:

- "Solve the equation" plug-in approach;
- Smoothed bootstrap.

Jones *et al.*(1996) advise that the plug-in bandwidth selector is the best. Loader (1999) "challenges the claimed superiority of the plug-in methods on several fronts." Bowman *et al.*(1998) conclude that the simple reference bandwidth is quite effective.

Because of this lack of consensus in the literature and also the fact that most density estimation is usually applied to low dimensional (i.e. one dimensional) datasets, it was decided to use the simple reference rule, which minimizes the *asymptotic mean integrated squared error* (*AMISE*) for a KDE. For a Gaussian kernal, the reference rule is

$$h_{\mathbf{ref}} \quad = \quad (4/3)^{1/5}\,\sigma\,N^{-1/5} \quad \approx \quad 1.06\,\widetilde{\sigma}\,N^{-1/5} \tag{7.5}$$

Where σ and $\widetilde{\sigma}$ are the population and sample standard deviations of the dataset.

Scant research has been done on KDE for high-dimensional data. Scott and Wand (1991) showed that synthetic ten-dimensional data could be modeled reasonably well, and conclude that the biggest problem with the curse of dimensionality was as much the lack of full rank as the sparseness of data.

7.2 Chi-Square Test

Given two sets of data, statisticians (and other people) often want to know: are the sets drawn from the same distribution function, or from different distribution functions?

Data can be either continuous or binned. A dataset can be compared to a known distribution, or two equally unknown datasets can be tested to see if they are both from the same distribution. In this project a confusion vector is compared to rows in a confusion matrix, hence, the data is binned and equally unknown.

The accepted test for comparing binned distributions is the *chi-square test*. For binned, equally unknown distributions, the chi-square statistic is

$$\chi^2 = \sum_i \frac{\left(\sqrt{\frac{S}{R}}R_i - \sqrt{\frac{R}{S}}S_i\right)^2}{R_i + S_i} \tag{7.6}$$

where

$$R \equiv \sum_i R_i \quad \text{and} \quad S \equiv \sum_i S_i \tag{7.7}$$

and R_i and S_i are the number of events in bin i for data sets **R** and **S** (Press *et al.*, 1992). A larger value of χ^2 shows that it is unlikely that two distributions are drawn from the same population. The χ^2 statistic can further be used with the *chi-square probability function* to determine a confidence level of the two distributions being equivalent. That was not done here, as the goal was to simply find the "nearest" row in the confusion matrix.

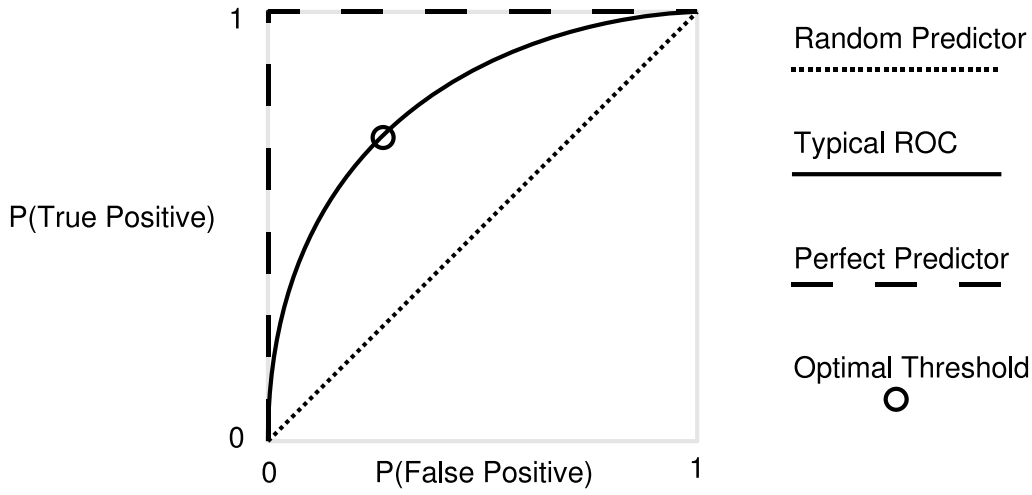

Figure 7.1: Some examples of ROC curves.

7.3 Receiver Operating Characteristics

The concept of the *receiver operating characteristics* (ROC) curve was developed in the 1940's to study systems for detecting airplanes in radar signals. In the 1960's they were used in psychophysics to assess subjective detection of weak signals (Egan, 1975). More recently, ROCs have been used in pattern recognition systems for selection of optimal discrimination thresholds.

The ROC is a plot of the *probability of false positives*, $P(\text{FP})$, versus the *probability of true positives*, $P(\text{TP})$, for a binary classifier as the discrimination threshold is varied.

Figure 7.1 shows some common ROC curves. The random predictor is a diagonal line emanating from the origin at a 45° angle. It shows that, regardless of the threshold, an equal number of false and true positive are accepted.

The perfect predictor appears as a horizontal line at $P(\text{TP}) = 1$. It shows that classification is error-free for any threshold value.

A typical ROC appears as a curve that is situated somewhere between the random predictor and perfect predictor curves. In practise, the curve sometimes appears a little lumpier. It shows that, as the discrimination threshold is changed, the ratio of false to true positives changes. In the case of neural networks, for example, the threshold would be the cut-off value applied to the output neurons, below which any results are discarded. In Figure 7.1, the top-right point of the curve shows classification with no thresholding applied, the bottom left, maximum thresholding. The circle shows an "optimal" threshold that improves the ratio of true to false positives. The selection of this point is not straightforward. In general, increasing the threshold (making it more restrictive) improves the $P(\text{TP})/P(\text{FP})$ ratio. The tradeoff is that more classifications are rejected

as being too weak.

In this work, a simple heuristic was developed for automatic selection of a good threshold value.

1. Choose a minimum $P(TP)$ value that would give an acceptable "reject" rate. A value of $P_{\mathrm{TP}_{\min}} = 0.25 = 25\%$ was chosen, which gives a rejection rate of about 75% (not counting false positives), which is still useful for the overall scheme used here.
2. Find the point on the ROC that is greater than $P_{\mathrm{TP}_{\min}}$, such that $P(\mathrm{TP}) > P_{\mathrm{TP}_{\min}}$, which also maximizes $P(\mathrm{TP})/[P(\mathrm{FP}) + 1]$. This is almost like a simple ratio, but the $+1$ prevents the denominator from being zero, as well as giving a higher value to higher $P(\mathrm{TP})$s, for equal ratios of $P(\mathrm{TP})/P(\mathrm{FP})$.

In short, the rule to find the optimal threshold point in the ROC is

$$\text{maximize} \quad \left(\frac{P(\mathrm{TP})}{P(\mathrm{FP}) + 1}\right) \left[P(\mathrm{TP}) > P_{\mathrm{TP}_{\min}}\right] \tag{7.8}$$

where the $[\cdots]$ notation is an *Iverson bracket* that evaluates to 1 or 0 if its contents are true or false, respectively.

Figures 7.2 and 7.3 show the ROC curves for a NN-100 classifier trained on all species, and the same classifier with only the alder flycatcher.

The first figure (7.2) is an example of a classic ROC curve. It shows that, with a low discrimination threshold of 0.2, the accuracy is 71%, and no data is rejected. As the threshold is increased, so does the ratio of true positives to false positives. The optimal threshold is determined to be 0.85, at which point the accuracy is 98%, but the rejection rate is 78%.

The ROC curve for the alder flycatcher (Figure 7.3) does not have the well-defined curvature of the "all species" curve. It is somewhat closer to a straight line, which means that changing the threshold will not have a significant effect on non-thresholded accuracy of 76%. Still, around the optimal point of 0.65, the slope of the curve increases, giving an accuracy rate of 94% and a rejection rate of 76%.

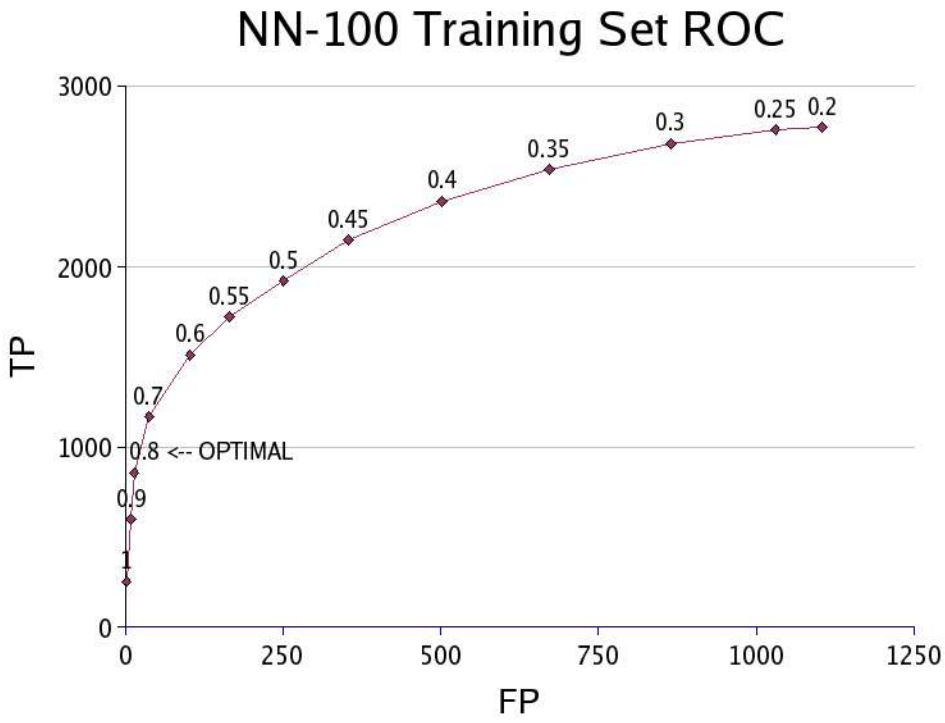

Figure 7.2: The ROC curve for the NN-100 classifier. The values on the curve indicate the discrimination threshold. A threshold of 0.2 produced a 0% rejection rate.

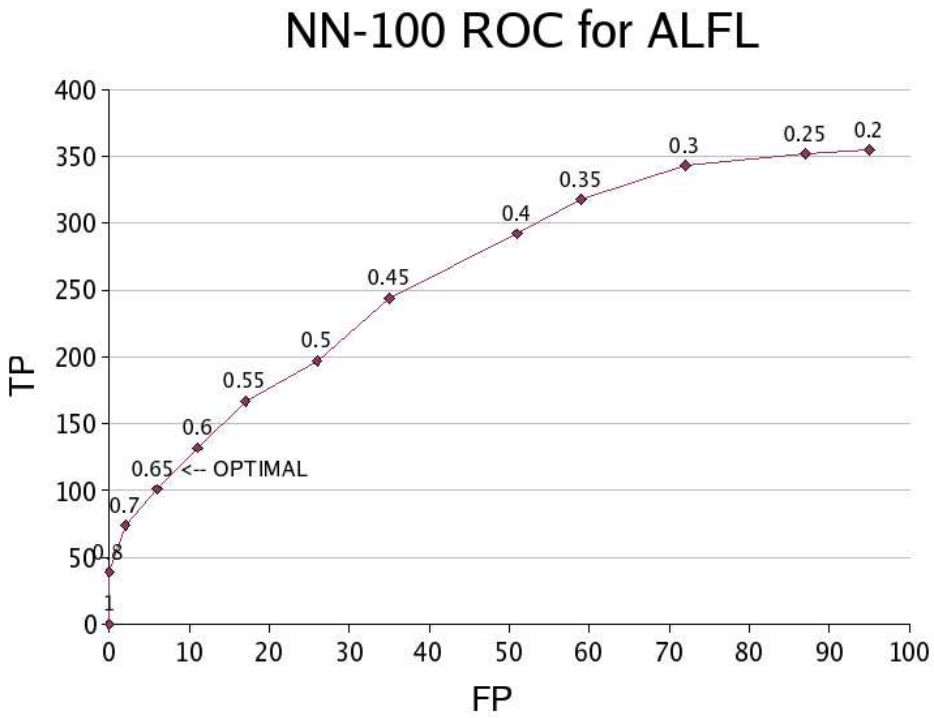

Figure 7.3: The ROC curve for the alder flycatcher with the NN-100 Classifier. The values on the curve indicate the discrimination threshold. A threshold of 0.2 produced a 0% rejection rate.

7.4 The Confusion Matrix

A *confusion matrix* (CM) is a well-known construction in the field of pattern recognition and statistics in general. It is a form of *contingency table.* It provides a simple and intuitive indication of the accuracy of a multi-category classifier (Kohavi *et al.*, 1998).

Each row in a CM represents the *actual* category to which an input vector belongs. Each column represents the category which the classifier *deduced* to be the proper category. The contents of each cell may be counted quantities, but percentages can be used to give a clearer understanding of accuracy without the need to do mental arithmetic. An additional "unclassified" column may be present for classifiers that have the option of deciding that an input vector does not fall under any of the categories provided. Perfect, error-free classification would give a CM with positive values only along the diagonal, with all other cells at zero.

Example of a Confusion Matrix

Here is a simple example to illustrate some of the salient features of a CM. Given a hypothetical system that is designed to distinguish between ducks and crows when given some feature vector, the following CM might be generated after several runs:

		DEDUCED CATEGORY		
		crow	duck	Unclassified
ACTUAL CATEGORY	crow	80%	15%	5%
	duck	30%	50%	20%

This confusion matrix quickly tells us that:

- When the classifier is given an input from the crow category, it produces the correct result 80% of the time, it thinks the vector belongs to the duck category 15% of the time, and 5% of the time, the classifier has decided that the input fits into none of the categories and is unclassified.
- When an input from the duck category is given, the system is correct 50% of the time, thinks it was a crow 30% of the time, and was unable to classify the input 20% of the time.

Chapter 8

Pattern Recognition Implementation

In this section an overview is given of the various stages of the pattern recognition process. The transformation from a raw audio signal to a species estimate has many steps, and two of those, classification and postprocessing, each have a few variations. However, at a higher level, the procedure can be explained in simpler terms:

- An audio signal is converted to digital form;
- The digital signal is broken into smaller frames;
- Each frame is processed to extract a variety of features;
- The feature vector for each frame is passed to a classifier to obtain a species estimate;
- The collection of frames for a call is postprocessed to determine a species estimate for the entire call.

Figure 8.1 gives a general flowchart of how the audio signal was converted to a result.

8.1 Bird Species

Ten species were analyzed. As can probably be guessed by looking at the names in Table 8.1, they were chosen by taking the first ten species from a CD-ROM in alphabetical order. The following table shows the class ID used internally in the software, the BBL (Bird Banding Lab) four-letter codes, and the species name. Photographs and spectrograms of each species may be found in Appendix A.

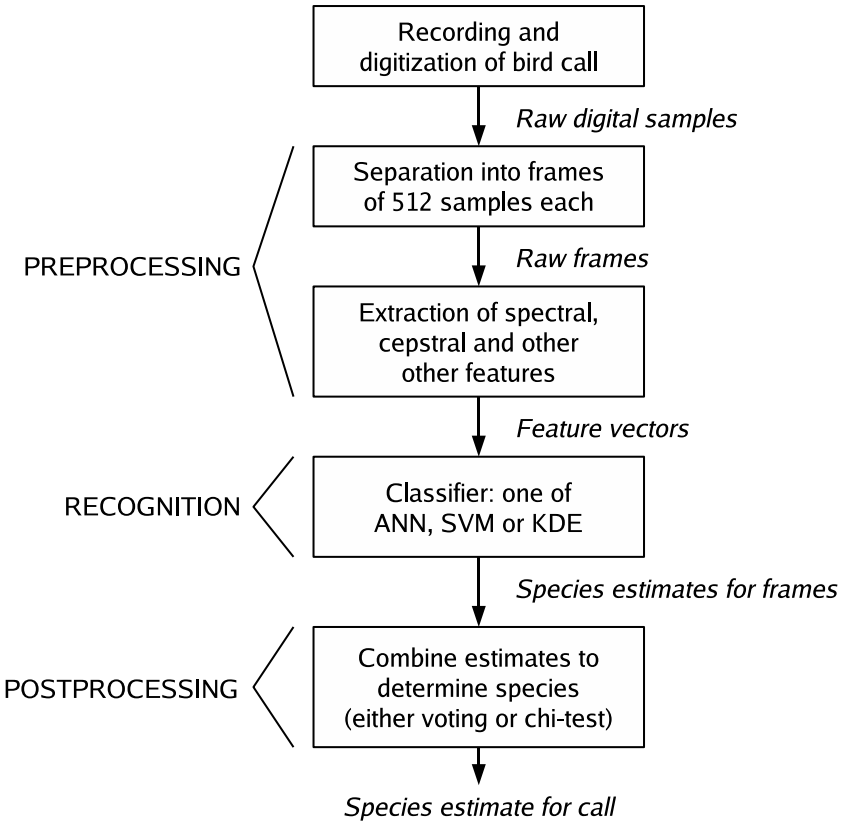

Figure 8.1: Flowchart of the overall recognition process. Blocks represent algorithms, and words in italic represent data passed to and from algorithms.

Class ID	BBL Code	Species Name
0	ALFL	Alder Flycatcher
1	AMCR	American Crow
2	AMGO	American Goldfinch
3	AMRE	American Redstart
4	AMRO	American Robin
5	BAOR	Baltimore Oriole
6	BCCH	Black-Capped Chickadee
7	BCTI	Black-Crested Titmouse
8	BDOW	Barred Owl
9	BLJA	Blue Jay

Table 8.1: Bird species used for recognition.

8.2 Preprocessing

The preprocessing stages were explained in detail in Section 3. In short, the following steps were taken:

- The audio recording was digitized at 44.1 kHz;
- The digitized signal was segmented into frames of 512 samples;
- The frames were transformed into the frequency and quefrency (cepstral) domains;
- Twenty features were extracted. Most were local to each frame, but two features involving the relatively long-term (1.5 seconds) amplitude of multiple frames were also calculated.

Table 3.1 gives mathematical formulae for the features.

8.3 Data Sets

Three datasets were extracted from a body of 160712 frames of 512 audio samples each. The data was divided into training and cross-validation parts.

The *training superset* (which will be referred to as the "superset") is 110193 frames in size. This was far too big for timely analysis with the various algorithms used in this project, so a much smaller *training set* of 3887 frames was randomly sampled from the superset. The optimal ROC based output thresholds are found with the training set. The confusion matrices, which are used later in the chi-squared goodness of fit test, are determined with the superset. This is to give a larger sample size for better confusion matrices.

The *cross-validation* (or *CV*) set is 50524 frames in size.

Data Set	Frame Count	Call Count	Avg. Calls per Species	Calls per Species: Range
Training Superset	110193	404	40	23–67
Training Set	3887	403	40	23–67
Cross-validation Set	50524	193	19	10–32

Efforts were taken to ensure that each species represented had approximately the same number of frames per data set. This would eliminate the need to handle differences in prior probabilities between species. Frames were separated into training and CV data sets in a *per-call* instead of a per-frame basis to prevent the possibility that different data sets might contain different frames from the same call. Unfortunately, frames and calls are different things, and calls can differ wildly in duration. Even though the frame count was consistent per species, the per-species call count

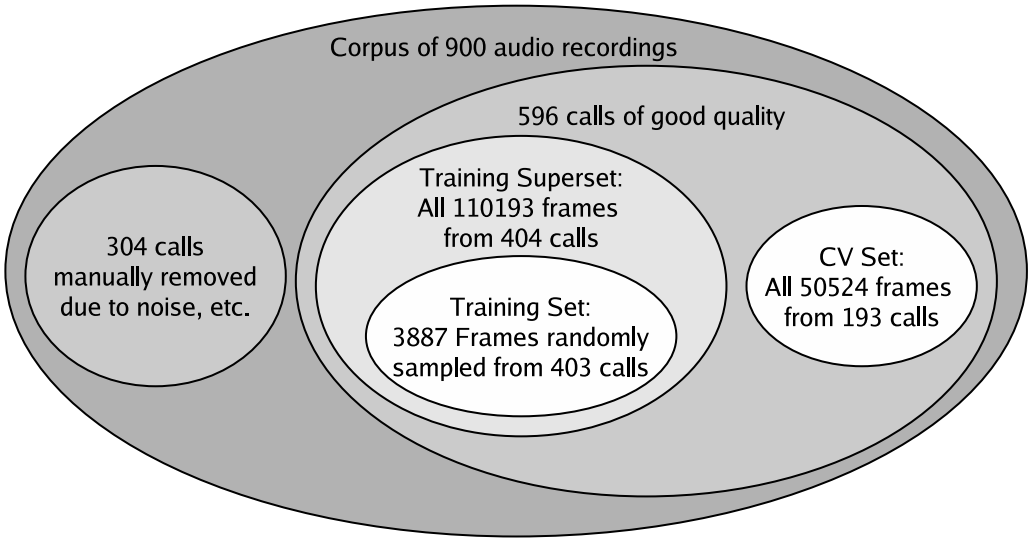

Figure 8.2: Structure of the datasets used in this project. The training set has one fewer call than the training superset because the random sampling coincidentally discarded all the frames of one of the calls.

varied from 23 to 67 in the training set. See Figure 8.2 for a graphical portrayal of the datasets used in this project.

Of the 900 calls available, 304 were discarded for being inadequate in some respect. Section 2 lists the various reasons for removing a call from the dataset. It is important to note that these calls were not individually removed because they were found to be misclassified in preliminary experimentation. Rather, all calls were listened to, and a simple checklist system was used to decide if a call should stay or go. The purpose of this was to ensure that the classifiers would learn a specific call of a bird (see Section 2.7) instead of recognizing extraneous sounds. For example, some species, such as the American robin, are common in urban areas. In many robin recordings, mechanical sounds such as traffic and lawn-mowers can be heard in the background. In effect, there is a correlation between these sounds and the robin. If recordings with background sounds were not removed, there was the possibility that a classifier might learn to recognize a species based partially on extraneous (if associated) sounds, rather than the bird's vocalization itself.

8.4 Pattern Recognition

Three pattern recognition systems were tested: artificial neural networks, support vector machines, and kernel density estimation.

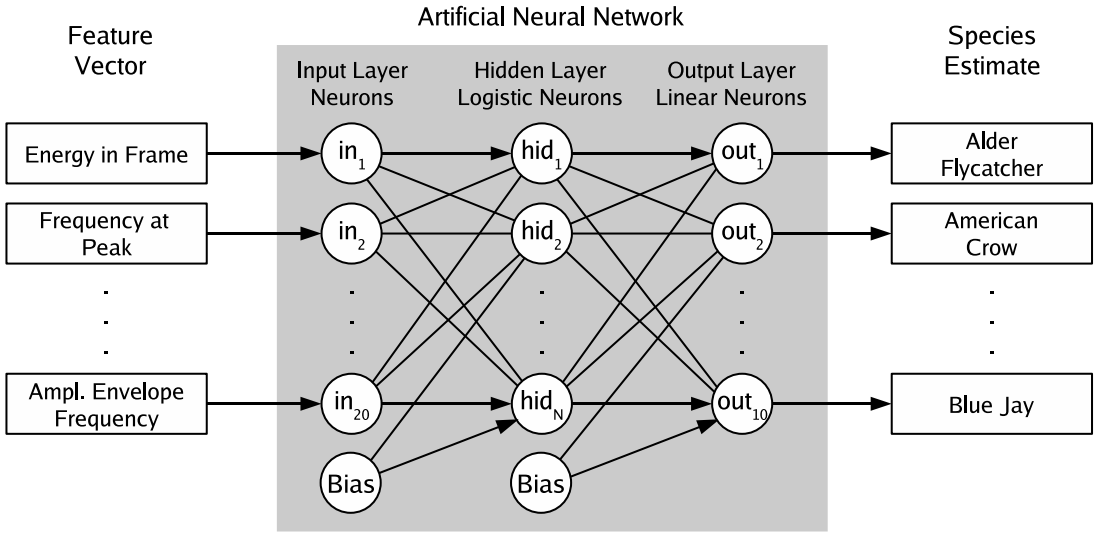

Figure 8.3: Organization of the neural network used in this project.

8.4.1 Artificial Neural Network

The ANN software was hand coded in GNU C++ and ran on a Pentium 4, 2.4 GHz computer.

The backpropagation algorithm was based on the explanation provided by Haykin (1994). The only embellishment added to the delta rule was a variable learning rate η during training. The user could specify an initial and final ηs, and the code would decrease it exponentially as the epochs progressed.

The network is composed of three layers, with an input layer, hidden layer and output layer. Logistic neurons were used in the hidden layer, and linear neurons for the output layer (Figure 8.3). As is recommended by ANN practitioners, the training set was randomly scrambled after each epoch (Haykin, 1994; Masters, 1993). The number of hidden neurons, training epochs, and learning rates varied. The following table encapsulates the various configurations of the neural network.

Hidden Neurons	Training Epochs	Start η	Stop η	CPU Time
20	100000	0.0001	1×10^{-7}	$\approx$ 1 hour
100	100000	0.0001	1×10^{-7}	$\approx$ 5 hours
500	50000	0.0001	3×10^{-6}	$\approx$ 13 hours

8.4.2 Support Vector Machines

For the SVM, the freely available LIBSVM library by Chang and Lin (2001) was used.

The LIBSVM training application `svm-train` has a multitude of options, with the following

combination being chosen:

- The SVM type is C-SVC, or *C-support vector classification*;
- The kernel is the radial basis function, $\exp\left[-\gamma|\mathbf{u}-\mathbf{v}|^2\right]$;
- Probability estimates, based on (Platt, 1999), were enabled to give smooth membership values to permit output threshold determination with ROC curve;
- Internal cross-validation was set to four-fold;
- The termination threshold, ϵ, was left at the default of 0.0001.

The LIBSVM training algorithm is based on *sequential minimal optimization* (SMO).

SVM is a inherently a binary classifier. LIBSVM works around this by using what is called the "one-against-one" approach. Here, $k(k-1)/2$ classifiers are trained, one for each pair of classes. During the prediction phase, each binary classifier "votes" for a class, and the winner is taken as the result. Chang and Lin (2005) chose this approach because it trains more quickly.

Grid Search

For this SVM setup, there are two free parameters that must be tweaked for an optimal model: γ for the kernel, which is analogous to $1/\sigma^2$ in a normal distribution, and the C, or *cost* parameter, a "penalty" value for misclassified points. A typical way of finding the best combination is to run a two-dimensional *grid search* to find the (C, γ) coordinate the gives the best CV accuracy.

After some initial experimentation with a subsampled training set, an 11×11 grid was chosen, using four-fold CV. SVM training at each cell of the grid was limited to 10 minutes, after which time that training process was aborted, and the search continued. The entire process took 12 CPU hours. Figure 8.4 shows the result of a grid search. The top plot shows accuracy with black cells indicating that training was aborted for taking too long. The bottom plot shows the time required. The region in the lower-right of each plot is where the SVM did not converge quickly enough and timed out. Perhaps not coincidentally, the highest accuracy is observed to be along the non-convergence zone. An optimal (C, γ) should maximize accuracy, but with better training accuracy, you run the risk of overtraining. Using this reasoning, three (C, γ) pairs were chosen, centered along the visible ridge of accuracy, one adjacent to the dead-zone, and two a little farther away:

Distance to Dead-Zone	C	γ
Far	$2^6 = 64$	$2^{-10} = 0.000977$
Midrange	$2^9 = 512$	$2^{-11} = 0.000488$
Near	$2^{12} = 4096$	$2^{-12} = 0.000244$

These points are indicated by **N**, **M**, and **F** in Figure 8.4. Further investigation of the non-convergence zone showed that, even if the timeout was increased, `svm-train` still would not converge.

In theory, SVM with slack variables should always have a solution, except where numerical roundoff error prevents it. In iterative algorithms such as SMO, a termination threshold, ϵ, is needed to determine when convergence has stopped. This value was set to 0.0001 for LIBSVM, which may be too small for certain combinations of data and parameters (Lin, 2005).

Support Vectors

After training, the results were as follows.

Distance to Dead-Zone	C, γ	Total SVs	SVs per Class (Range)	Average SVs per Binary Classifier
Near	$2^{12}, 2^{-12}$	2770	193–367	62.6
Midrange	$2^{9}, 2^{-11}$	2860	195–379	63.6
Far	$2^{6}, 2^{-10}$	3020	206–404	67.1

The *Average SVs* column shows the result of dividing *Total SVs* by the number of classifiers used in the *one-against-one* method, which in this case is 45. The increase in the number of support vectors might be due to the increase of γ, which would reduce the "width" of the radial basis functions in the model, therefore allowing a more complicated separation between classes.

Prediction

Finally, the `svm-predict` application was used to classify the CV and test datasets using the model created by `svm-train`.

8.4.3 Kernel Density Estimation

The KDE algorithm is, by far, the simplest algorithm of the three mentioned in this section. It can be expressed as a single formula (Bishop, 1995):

$$\widetilde{p}(\mathbf{x}) = \frac{1}{N} \sum_{n=1}^{N} \frac{1}{h^d} H\left(\frac{\mathbf{x} - \mathbf{x}^n}{h}\right) \tag{8.1}$$

where $\widetilde{p}(\mathbf{x})$ is the model density, N is the number of points in the training set, h is a bandwidth parameter, H is a kernel that acts on the difference between two points, x_n are the training points,

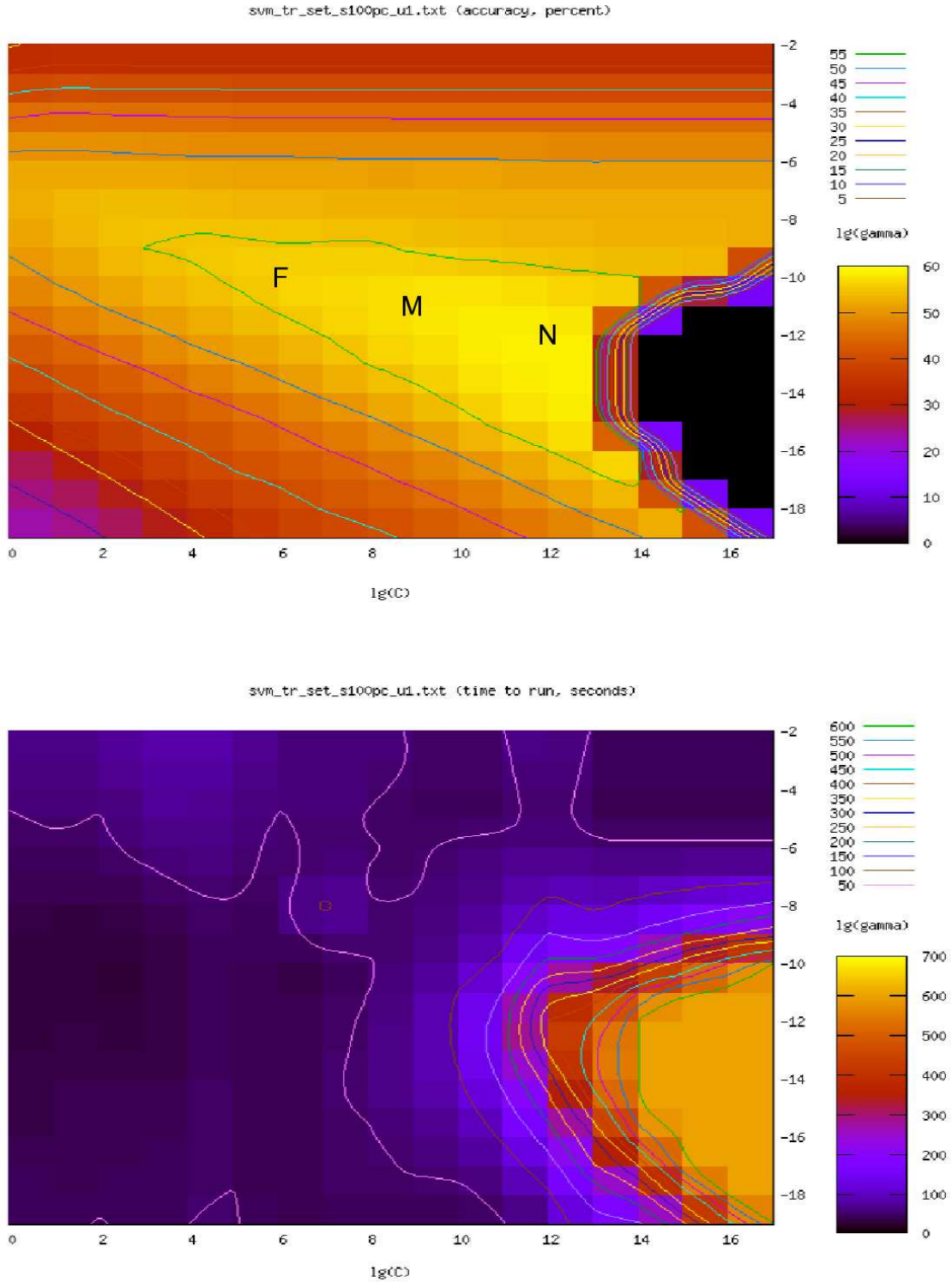

Figure 8.4: Grid search accuracy (top) and duration (bottom). Black cells in (top) indicate that the SVM model did not converge and timed out. The **N**, **M** and **F** show (C, γ) pairs that are near, midrange, and far in relation to the non-convergence zone.

and d is the dimension of each point. In this project, a multivariate normal kernel (RBF) is used for H, which gives the formula:

$$\widetilde{p}(\mathbf{x})_{\mathcal{C}} = \frac{1}{N_{\mathcal{C}}} \sum_{n=1}^{N_{\mathcal{C}}} \frac{1}{(2\pi h_{\mathcal{C}}^2)^{d/2}} \exp\left\{ -\frac{\|\mathbf{x} - \mathbf{x}_{\mathcal{C}}^n\|^2}{2h_{\mathcal{C}}^2} \right\} \tag{8.2}$$

where the $\mathcal{C}$ subscript indicates data specific to a particular class. The implementation of this was hand-coded with GNU C++.

8.4.4 Bandwidth Selection

KDE does not have a separate training phase like ANNs and SVMs. Rather, the entirety of the training data is accessed for each estimate produced. A process that might be similar to training is *bandwidth selection*, whereby a value of h is determined that gives the best results. Bowman *et al.*(1998) show that a simple reference bandwidth is effective when compared to more elaborate methods of bandwidth selection.

In KDE terminology, the *reference rule* is a value of h that will minimize the *mean integrated squared error* (MISE) for a KDE, as N approaches infinity (Scott, 1992). For a Gaussian (normal) kernel the reference rule is

$$h_{\mathbf{ref}} \quad = \quad (4/3)^{1/5}\, \sigma\, N^{-1/5} \quad \approx \quad 1.06\, \widetilde{\sigma}\, N^{-1/5} \tag{8.3}$$

Where σ and $\widetilde{\sigma}$ are the population and sample standard deviations of the data set, respectively. The standard deviation of the training set was determined by calculating σ along each dimension, then averaging them.

A covariance matrix was not used. For simplicity, a single σ value was used for all dimensions. Other values of h, both above and below $h_{\mathbf{ref}}$, were experimented with, but the reference rule gave the best results.

8.4.5 Recognition

To calculate the output vector for an input $\mathbf{x}$, $\widetilde{p}(\mathbf{x})_{\mathcal{C}}$ is calculate for each of the ten classes. The same postprocessing used with the ANN and SVM methods could then be applied. The outputs of the KDE were lower than those of the ANN and SVM models, which are around the range of 0–1. They appeared to be exponentially distributed, from 0 to about 1×10^{-16}, and increased by about one decade for every tenth percentile of the distribution. The values were recalibrated by taking the log and then scaling and shifting to bring the $\widetilde{p}(\mathbf{x})$ values closer to a uniformly distribution from 0–1. This worked fine for the training set, but the thresholds proved to be too high for the CV set: the majority of the samples were rejected, because of KDE being a biased

estimator and inherently giving higher values to anything in the training set. In the end, the thresholds were removed altogether for KDE.

These tiny values for $\widetilde{p}(\mathbf{x})$ are expected. Remember that the integral of a probability density function will be only 1 (Gersho and Gray, 1992). Combined with the high dimensionality of the space, the density at any point is expected to be low. For example, consider a uniform probability distribution contained in a d-dimensional cube with sides of length h. The integral of the cube with density ρ must equal to 1. For this simple cube-shaped distribution, $1 = \rho h^d$, or, $\rho = 1/h^d$. If we consider the hypothetical scenario that the feature set for the bird data occupies a cube-shaped space, a rough estimate of $h = 10$ with $d = 20$ gives $\rho = 1 \times 10^{-20}$.

8.5 Postprocessing

Finally, when an audio frame has been processed, the output must be interpreted to determine the species. A simple solution is to take the classifier output with the highest value and make that the winner. However, there are two flaws with this.

The first is that the many of the frames are recordings of the silence between bird chirps. It would be nonsensical to try to classify silence as a species. This can be remedied by setting a high threshold on the output activations. Then, any frames that do not confidently belong to one species will be rejected. The thresholds are selected to improve the ratio of true positives over false positives as described in Section 7.3.

The second flaw is that we are actually trying to classify entire recordings, not just individual frames. Thus, all the frames in the recording as a whole have to be combined, and a decision made on the aggregate.

8.5.1 Simple Voting

One solution to this is to employ a voting algorithm. The maximum element in the output vector (if any) increments a counter for the corresponding species. When all frames in the call are finished, the species with the most votes wins.

8.5.2 Confusion Matching

As with the voting method, the output vector increments a counter for the corresponding species. After the call has been processed, the vote tally forms what would be a single row from a confusion matrix, or a *confusion row*.

What can be done with this confusion row? Inspection of the data has shown that the confusion rows for the same species often appear similar. Perhaps a confusion row can be compared to rows in the training confusion matrix, and the closest one can be found, which would imply the species. Since the elements of the confusion row represent probabilities of a *multinomial distribution*, a *chi-squared goodness-of-fit test* can be used to determine if two rows are drawn from the same distribution. For binned data, the chi-squared statistic is:

$$\chi^2 = \sum_i \frac{\left(\sqrt{\frac{S}{R}}R_i - \sqrt{\frac{R}{S}}S_i\right)^2}{R_i + S_i} \tag{8.4}$$

where

$$R \equiv \sum_i R_i \quad \text{and} \quad S \equiv \sum_i S_i \tag{8.5}$$

and R_i and S_i are the number of events in bin i for data sets **R** and **S** (Press *et al.*, 1992). Lower values of χ^2 indicate a better goodness-of-fit. The chi-squared statistic of a confusion row is calculated against all rows of a confusion matrix, and the winner is the row that gives the lowest value.

Figure 8.5 shows a detailed flowchart of the overall recognition procedure.

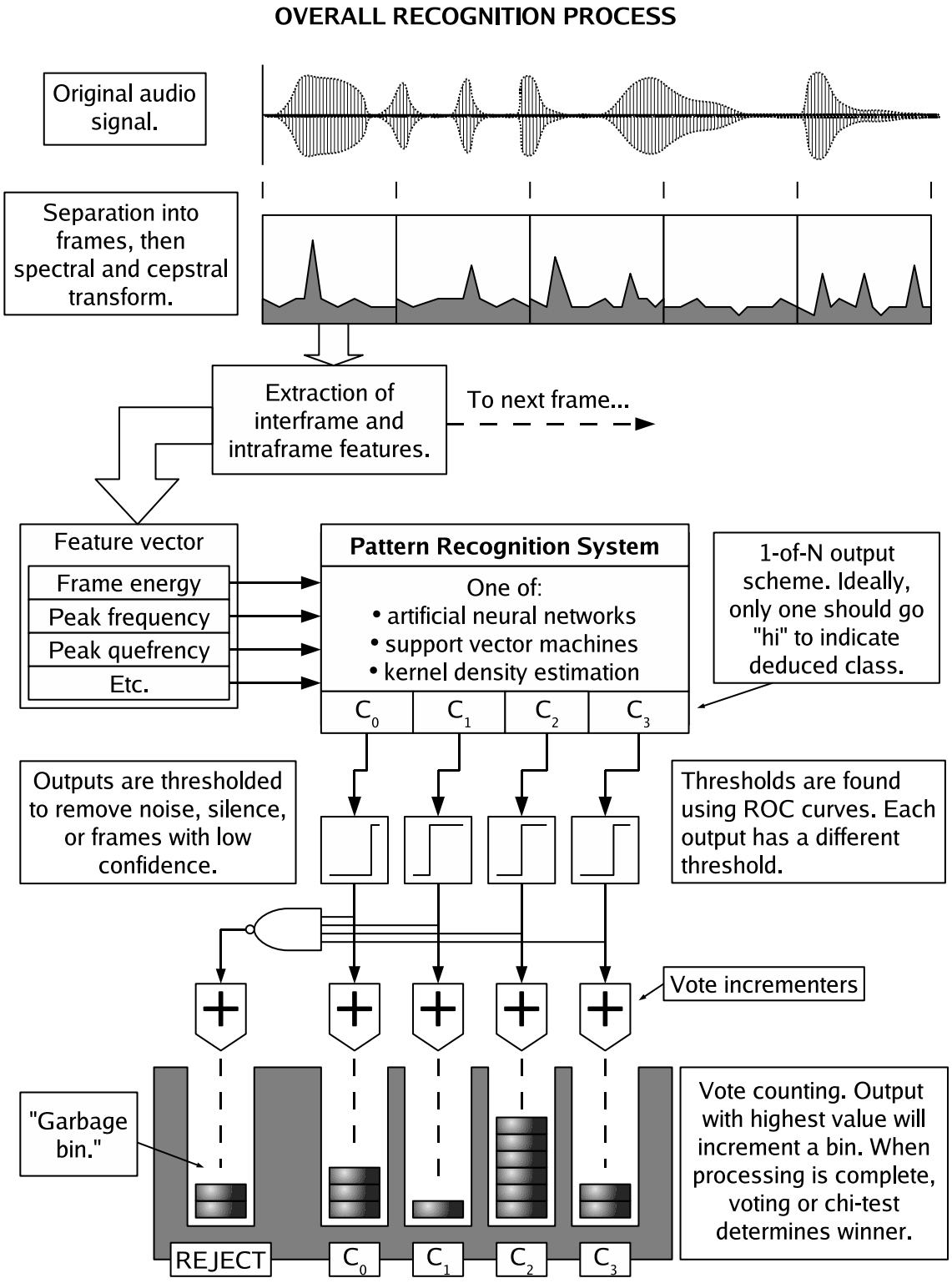

Figure 8.5: Detailed flowchart of the overall recognition process.

Chapter 9

Results

9.1 Introduction

In this chapter, the experimental results will be put forth and interpreted.

First, the numeric results of the trials will be presented with minimal interpretation.

Second, some high-level comments will be made reguarding accuracy rates, data sets, unclassified data, and interpretation of results. Some issues that are unique to neural networks will also be discussed.

Third, the results for both single-frame and entire-call data will be investigated in more depth.

Finally, a meta-analysis and comparison of the different classifiers will be attempted.

9.1.1 Numeric Results

Table 9.1 summarizes the results of single-frame recognition. Each row represents a classification trial with a specific data set (as described in Section 8.3) and recognition system. The systems NN-20, NN-100, and NN-500 indicate artificial neural networks with 20, 100, and 500 hidden neurons. The SVM-FAR, SVM-MID, and SVM-NEA labels indicate support vector machines with C (cost) and γ parameters that are far, midrange, and near in relation to a non-convergence region of a grid search (Section 8.4.2). The KDE label indicates a kernel density estimation classifier, with the bandwidth set according to a reference rule (Section 8.4.4).

The "accuracy" column shows the average recognition accuracy over all species for a trial. "Rejections" shows the percentage of frames that were not classified because none of the classifier

Data Set	System	Frame Accuracy (%)	Rejections (%)	Accuracy Floor (%)
Training Superset	NN-20	67	53	36
	NN-100	**87**	63	**61**
	NN-500	85	62	34
	SVM-FAR	82	64	47
	SVM-MID	83	68	48
	SVM-NEA	78	57	40
	KDE	40	0	12
Training Set	NN-20	66	46	35
	NN-100	93	65	73
	NN-500	**98**	58	**91**
	SVM-FAR	89	61	58
	SVM-MID	93	66	64
	SVM-NEA	89	57	52
	KDE	74	0	63
CV Set	NN-20	64	54	29
	NN-100	**83**	68	**53**
	NN-500	79	67	28
	SVM-FAR	76	65	36
	SVM-MID	79	69	42
	SVM-NEA	74	58	30
	KDE	38	0	8

Table 9.1: Results for frame recognition. The best accuracy and accuracy floor values for a particular data set and postprocessing method are shown in bold.

outputs met the discrimination thresholds that were derived from the ROC curve for each species (Section 7.3). The "accuracy floor" column shows the lowest accuracy rate of the ten species for a given trial. This metric was added as an indication of worst case performance. This was selected as a performance measure, instead of median or maximum accuracy, because it is the type of requirement that would be needed in an engineering specification.

Table 9.2 shows the frame accuracies for each of the ten species. The lowest value in each row is the "accuracy floor" that is given in Table 9.1. Table 9.3 shows the rejection rates for all species in a similar layout.

Table 9.4 summarizes the results of recognition for entire calls. It has one more column than the "frames" table (9.1), that being "postprocessing," which indicates whether the voting or the chi-test postprocessor (Section 8.5) was used to convert the multiple species estimates for a call into a single estimate.

Table 9.5 shows the call accuracies for each of the ten species. The lowest value in each row is the "accuracy floor" for a given trial.

Data Set	System	Frame Accuracy (%) for Each Species									
		ALFL	AMCR	AMGO	AMRE	AMRO	BAOR	BCCH	BCTI	BDOW	BLJA
Training Superset	NN-20	85	95	64	62	57	39	36	36	75	60
	NN-100	87	99	81	92	80	72	61	86	96	78
	NN-500	86	99	83	90	79	73	34	81	95	74
	SVM-FAR	69	90	84	86	87	47	73	77	88	64
	SVM-MID	84	98	89	90	77	61	48	82	89	71
	SVM-NEA	77	97	70	81	83	40	45	71	88	58
	KDE	43	12	53	62	43	27	48	61	46	31
Training Set	NN-20	83	96	59	69	64	47	35	41	77	62
	NN-100	96	99	86	98	92	88	73	94	100	90
	NN-500	100	100	98	99	98	96	91	100	100	95
	SVM-FAR	90	94	95	94	95	58	88	90	95	92
	SVM-MID	99	99	98	98	92	78	64	97	96	95
	SVM-NEA	96	99	95	94	97	52	61	95	98	87
	KDE	65	64	81	82	82	63	73	87	71	71
CV Set	NN-20	76	91	72	61	62	34	35	33	74	29
	NN-100	66	98	83	89	78	61	62	88	94	53
	NN-500	61	97	80	89	72	60	28	86	94	59
	SVM-FAR	38	77	81	86	84	36	63	85	83	66
	SVM-MID	54	95	87	89	73	47	42	89	86	73
	SVM-NEA	49	95	67	78	85	30	42	75	85	42
	KDE	35	8	51	60	33	23	48	63	49	26
Stats	Minimum	35	8	51	60	33	23	28	33	46	26
	Mean	73	86	79	83	77	54	55	77	85	65
	Median	77	96	81	89	80	52	48	85	88	66
	Maximum	100	100	98	99	98	96	91	100	100	95
	Std. Dev.	20	27	14	13	17	20	18	20	15	21

Table 9.2: Frame recognition accuracy for each species. The species abbreviations are explained in Table 8.1. The "Stats" section gives a variety of statistics for each column.

Data Set	System	Frame Rejections (%) for Each Species									
		ALFL	AMCR	AMGO	AMRE	AMRO	BAOR	BCCH	BCTI	BDOW	BLJA
Training Superset	NN-20	32	39	51	16	62	69	29	70	70	48
	NN-100	81	42	77	41	79	82	76	81	54	77
	NN-500	73	49	71	42	78	77	84	82	48	78
	SVM-FAR	77	56	70	44	61	69	54	77	71	69
	SVM-MID	80	49	67	42	79	75	73	81	75	80
	SVM-NEA	66	40	69	38	53	62	61	73	63	72
	KDE	0	0	0	0	0	0	0	0	0	0
Training Set	NN-20	29	40	52	15	58	67	29	68	66	46
	NN-100	77	45	78	35	76	82	74	77	50	75
	NN-500	61	49	58	29	71	74	81	73	40	71
	SVM-FAR	73	60	71	41	42	69	29	74	72	64
	SVM-MID	74	47	62	38	77	77	76	76	74	78
	SVM-NEA	60	39	74	40	39	67	63	68	62	75
	KDE	0	0	0	0	0	0	0	0	0	0
CV Set	NN-20	37	42	50	20	68	58	22	69	70	54
	NN-100	80	48	77	43	86	83	72	83	50	82
	NN-500	74	55	71	45	85	77	83	84	45	81
	SVM-FAR	79	64	70	45	68	63	55	71	66	56
	SVM-MID	83	55	67	44	82	72	68	76	72	76
	SVM-NEA	67	44	68	40	64	58	57	70	57	73
	KDE	0	0	0	0	0	0	0	0	0	0
Stats	Minimum	0	0	0	0	0	0	0	0	0	0
	Mean	57	41	57	31	58	61	52	64	53	60
	Median	73	45	68	40	68	69	61	73	62	72
	Maximum	83	64	78	45	86	83	84	84	75	82
	Std. Dev.	29	19	25	16	28	26	29	27	24	27

Table 9.3: Frame rejection rate for each species. The species abbreviations are explained in Table 8.1. The "Stats" section gives a variety of statistics for each column.

Data Set	Postproc.	System	Call Accuracy (%)	Rejections (%)	Accuracy Floor (%)
Training Superset	Voting	NN-20	76	0	0
		NN-100	**89**	0	**48**
		NN-500	**89**	0	57
		SVM-FAR	86	0	37
		SVM-MID	88	0	44
		SVM-NEA	85	0	23
		KDE	62	0	0
	Chi-Test	NN-20	84	0	62
		NN-100	89	0	70
		NN-500	**91**	0	**79**
		SVM-FAR	87	0	76
		SVM-MID	88	0	70
		SVM-NEA	89	0	63
		KDE	85	0	70
Training Set	Voting	NN-20	64	8	0
		NN-100	75	20	39
		NN-500	82	16	**48**
		SVM-FAR	79	12	47
		SVM-MID	78	16	**48**
		SVM-NEA	81	9	40
		KDE	**88**	0	0
	Chi-Test	NN-20	71	0	44
		NN-100	77	0	44
		NN-500	84	0	48
		SVM-FAR	83	0	60
		SVM-MID	78	0	52
		SVM-NEA	83	0	48
		KDE	**93**	0	**83**
CV Set	Voting	NN-20	69	0	0
		NN-100	**82**	2	27
		NN-500	81	2	27
		SVM-FAR	77	0	7
		SVM-MID	81	0	**40**
		SVM-NEA	79	0	13
		KDE	59	0	0
	Chi-Test	NN-20	76	0	36
		NN-100	**82**	0	46
		NN-500	79	0	53
		SVM-FAR	79	0	**63**
		SVM-MID	81	0	56
		SVM-NEA	78	0	56
		KDE	72	0	36

Table 9.4: Results for call recognition. The best accuracy and accuracy floor values for a particular data set and postprocessing method are shown in bold.

Data Set	Postproc.	System	Call Accuracy (%) for Each Species									
			ALFL	AMCR	AMGO	AMRE	AMRO	BAOR	BCCH	BCTI	BDOW	BLJA
Training Superset	Voting	NN-20	0	99	60	85	92	37	93	3	96	52
		NN-100	89	100	80	97	92	57	69	84	100	48
		NN-500	80	100	90	96	96	67	69	74	100	57
		SVM-FAR	37	97	70	96	94	67	90	68	96	57
		SVM-MID	77	100	80	96	90	60	79	74	96	44
		SVM-NEA	54	97	80	97	98	23	72	48	96	39
		KDE	0	73	0	6	73	0	0	6	100	0
	Chi-Test	NN-20	74	97	73	96	79	83	62	74	96	70
		NN-100	97	96	87	94	90	70	76	81	98	78
		NN-500	86	97	90	96	92	80	79	84	100	96
		SVM-FAR	86	88	83	97	85	80	76	81	96	87
		SVM-MID	89	97	80	96	81	70	79	81	96	87
		SVM-NEA	91	93	87	96	90	63	83	84	96	87
		KDE	86	88	70	96	79	77	76	90	96	70
Training Set	Voting	NN-20	0	89	53	84	71	47	86	29	68	30
		NN-100	100	88	67	96	58	60	45	65	86	39
		NN-500	100	86	90	99	71	73	55	77	89	48
		SVM-FAR	80	76	63	96	90	47	97	65	75	74
		SVM-MID	97	88	80	97	60	60	48	77	75	78
		SVM-NEA	91	91	80	93	92	40	45	68	84	65
		KDE	0	80	0	63	83	67	0	71	96	0
	Chi-Test	NN-20	71	88	57	90	63	80	52	45	71	44
		NN-100	91	97	67	94	58	63	52	65	86	44
		NN-500	97	99	90	97	69	77	55	77	89	48
		SVM-FAR	91	76	83	96	85	60	93	71	77	87
		SVM-MID	97	86	77	97	56	63	52	81	75	78
		SVM-NEA	97	91	97	93	85	53	48	84	84	74
		KDE	83	99	100	97	85	100	86	90	89	91
CV Set	Voting	NN-20	0	94	59	80	90	19	90	8	100	9
		NN-100	73	97	86	90	90	50	70	75	100	27
		NN-500	40	94	86	83	87	50	50	58	100	27
		SVM-FAR	7	88	68	87	87	38	80	67	100	36
		SVM-MID	40	88	91	87	90	50	80	75	100	46
		SVM-NEA	27	94	86	87	93	13	60	50	100	18
		KDE	0	56	0	7	47	0	0	8	93	0
	Chi-Test	NN-20	53	91	64	87	80	63	80	67	100	36
		NN-100	93	88	91	83	83	69	70	75	100	46
		NN-500	53	91	86	80	80	75	60	75	100	64
		SVM-FAR	87	75	77	90	80	63	70	67	93	73
		SVM-MID	60	84	86	87	77	56	80	75	100	100
		SVM-NEA	60	91	86	83	77	56	60	67	93	82
		KDE	73	69	82	87	57	56	80	83	93	36
	Stats	Minimum	0	56	0	6	47	0	0	3	68	0
		Mean	65	89	73	87	80	57	65	65	92	54
		Median	79	91	80	93	84	60	70	74	96	50
		Maximum	100	100	100	99	98	100	97	90	100	100
		Std. Dev.	34	9	23	19	13	22	23	23	9	27

Table 9.5: Call recognition accuracy for each species. The species abbreviations are explained in Table 8.1. The "Stats" section gives a variety of statistics for each column.

9.1.2 Caveat on Interpreting Results

One must be cautious of making absolute statements about relative performance based on the data presented here. Many of the results for different systems are quite close: less than 5 percentage points apart, in some cases. It is expected that if trials were re-run with slightly different training and cross-validation sets, the trends would look similar, but the precise values would be different. Also, in the cross-validation "calls" set, each species has, on average, only $193/10 \approx 19$ calls. Given this quantity, the misclassification of a single call can lead to an accuracy variation of $(100\%)/19 \approx 5\%$. For these reasons, one can only comment on obvious trends rather than marginal differences.

9.1.3 Absence of a Test Set

Originally, a separate "test" set was created, but then was merged with the cross-validation (CV) set to increase the CV set's size. In general, in some training scenarios, a test set is desired. This is because, if a researcher runs many trials and tweaks parameters in an effort to maximize the CV accuracy, he is indirectly incorporating the CV set into the training set. One way to see if this has happened is to process, when all the training and testing with the CV set is done, an additional test set. If overtraining on the CV set has occurred, then the test set will perform more poorly than the CV set.

In this project, very little fine-tuning was done. Some preliminary experimentation was needed to get some ballpark estimates of various parameters, then a few trials were run. No effort was made to optimize the CV results. Thus, CV overtraining is not expected to be an issue with this project.

9.1.4 Rejections and Accuracy

Initially, it was not clear how best to interpret unclassified frames. Eventually it was decided that accuracy should be tallied after ignoring rejected frames or calls, because silent segments in the audio recordings are expected to create unclassified frames. Also, any bird recording could have a potentially unlimited duration of silence preceding or following its call, which may or may not be removed by the person editing and digitizing it. Thus, it would be best to ignore rejects altogether. This is why the "accuracy" and "rejections" columns in Tables 9.1 and 9.4 usually add up to more than 100%.

9.1.5 Neural Network Training

One characteristic that distinguishes ANNs from SVMs and KDE is the transparency of the training process. Back-propagation is an iterative procedure; during training, the accuracy and

mean squared error may be easily inspected. This permits observation of the effect known as overtraining.

Overtraining, a phenomenon in which a neural network acts like a lookup table for the training data, was observed only in the NN-500 network (500 hidden units). Figure 9.1 shows the mean squared error (the average of squared differences between the outputs and the targets) reaching a minimum at about 5000 epochs, after which it slowly begins rising, finally leveling off at about 40000 epochs. This shows that generalization from the training set to the CV set is degrading. The NN-20 (not plotted) and NN-100 networks show no overtraining — the MSE continues decreasing, and finally levels off.

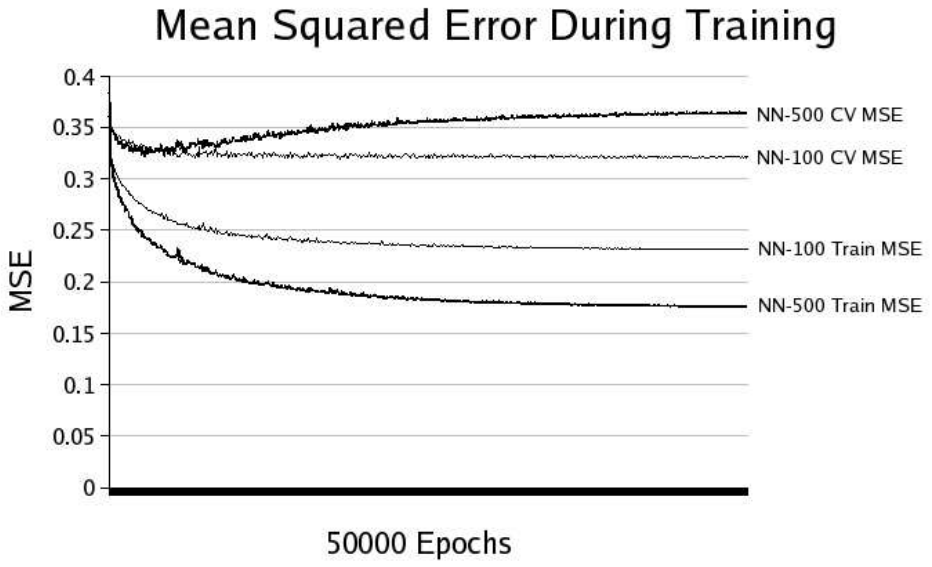

Figure 9.1: Mean squared error during training for both the training and cross-validation sets.

Oddly enough, overtraining was not seen when looking at the accuracy rate (Figure 9.2). An output vector is considered "accurate" if the maximum element matches the target. Another interpretation is that the maximum element is set to 1, and all other elements are set to 0. (Note: accuracy during training is calculated before optimal thresholds are calculated as explained in Section 7.3.) This would eliminate intermediate values that would contribute to the MSE.

The MSE evidence for overtraining agrees with the accuracy floor observations (Table 9.1; Figure 9.5), which show that the NN-500 network has a stellar training set accuracy, but a poor CV accuracy, when compared to the NN-100 results.

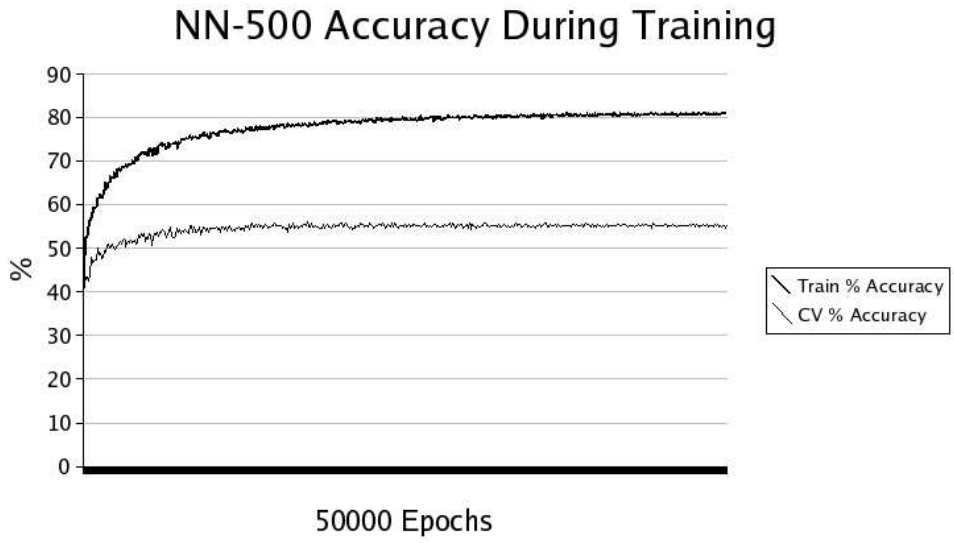

Figure 9.2: Accuracy during training.

9.2 Single-Frame Accuracy

As observed in Figure 9.3, the best performers for single-frame accuracy are NN-100, NN-500, SVM-FAR, -MID and -NEA. They gave a training set accuracy of 89–98%, and a CV accuracy of 74–83%.

The NN-500 system shows symptoms of overtraining. It has the best training set score, at 98%, but both the superset and CV scores are lower than those of the simpler NN-100 system.

All three SVM results are within 5%, but the highest score is for SVM-MID, which used cost and γ parameters that were halfway between high and low scoring points in the grid search (Figure 8.4).

The NN-20 results are unique in that they are almost the same for all three datasets, varying by only 3 percentage points, from 64% to 67%. This appears to be, but is not an example of generalization. Good generalization occurs when the bias and variance have reached an optimal trade-off point. Here, *variance* is low, as indicated by the similar error rates for the different data sets. However, the lower scores indicate that *bias* is high, and the network function is, on average, different from the unknown classification function (Bishop, 1995). A high bias shows that a classifier has too little flexibility and is unable to fit well with the actual classification function (Bishop 1995). This corresponds to the fact that NN-20 has only 20 hidden neurons. Since each hidden neuron acts as a simple binary classifier that divides the feature space in two, 20 neurons classifying a 20-dimensional feature space would not be capable of distinguishing fine-grained detail.

The KDE results really show it to be a *biased estimator*. The training set score of 74% is not bad, but

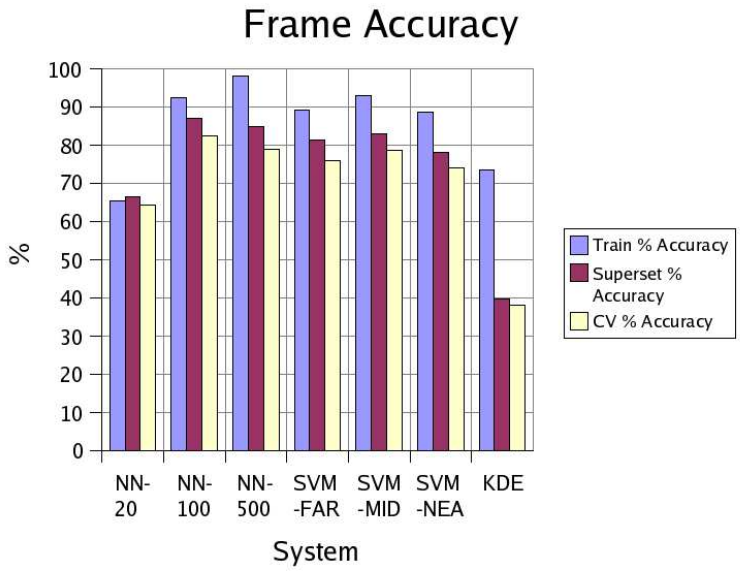

Figure 9.3: Single frame accuracy.

the superset and CV scores of 40% and 38% hardly come close to the training score.

9.2.1 Frame Rejection

Figure 9.4 shows that there is a weak correlation of 0.51 between rejected frames and accuracy. KDE was not included because it was necessary to set the rejection threshold to zero.

9.2.2 Frame Accuracy Floor

The accuracy floors (lowest species score) for all classifiers are laid out in Figure 9.5. It has a similar structure to Frame Accuracy (Figure 9.3), but with greater variation, which is expected since the minimum accuracy is an outlier value. The NN-500 and KDE systems are notable for the range in scores. The NN-500 has a difference of 63 percentage points between the training and CV sets, and the KDE has a difference of 55 percentage points. The CV floor for the KDE is actually less than chance. As with the frame accuracy chart, these observations hint at overtraining.

9.3 Call Accuracy

When analyzing entire calls, an extra bit of processing has to be done to determine the species based on a collection of frames. As mentioned earlier, either a simple vote count, or a chi-squared

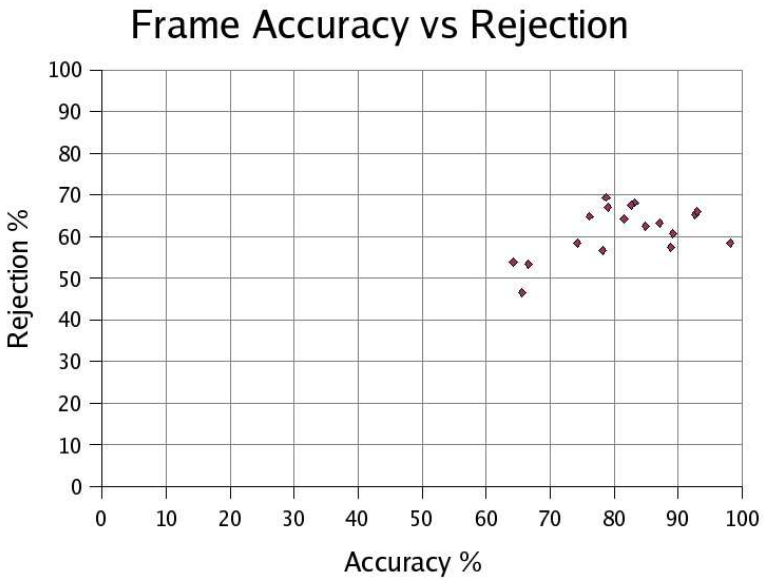

Figure 9.4: Single frame accuracy vs. rejection.

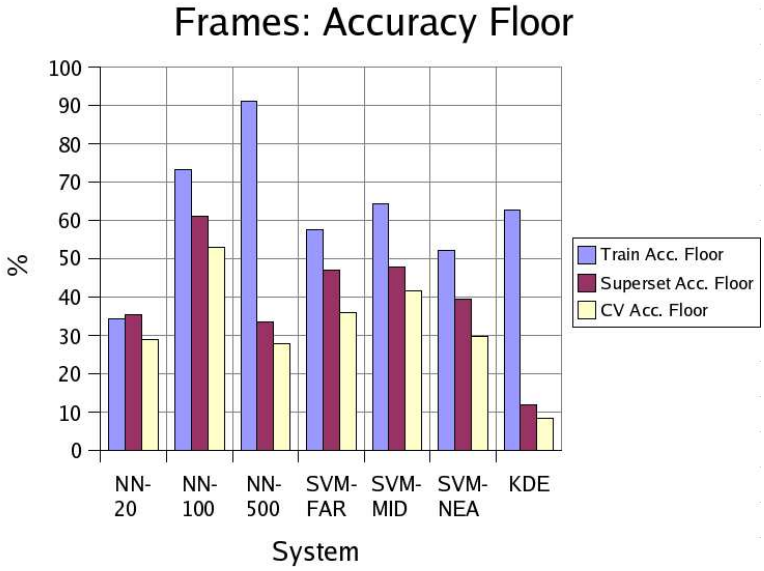

Figure 9.5: Single frame accuracy floor.

goodness-of-fit test can be used to achieve this. In this section, analysis concentrates on comparing performance between these two methods.

Figures 9.6 and 9.7 show recognition accuracy for both types of postprocessing. In general, the training set performs more poorly than the larger superset from which it was extracted. This might appear paradoxical, but it is expected, for the calls in the training set do not have enough frames (only 10, on average) to give a statistically good estimate of a confusion row, which itself has 10 columns.

Call Rejection

Frames have a higher rejection rate than calls. For a call to be unclassified, all frames composing it also have to be unclassified. The rejection rate for frames will be approximately proportional to the amount of silence or noise in the signal. Unless a call is completely filled with low-confidence frames, it is unlikely to be rejected. Thus, rejection rates of calls are not plotted because they are negligible. Superset and CV set rejection is, at most, 2%. Training set rejection is higher, up to 20%. This is not surprising because the training set is a small subsampled fraction (about 4%) of the superset. On average, each call in the training set is composed of $3887/403 \approx 10$ frames, so there is a significant probability that they might all be rejected.

Calls being postprocessed with the chi-test will have zero rejects — every call is guaranteed to be classified, because the nearest matching row in the confusion matrix is used.

For clarity, reference will be made to the score of only the CV dataset, as that is the one that is most indicative of generalization ability.

9.3.1 Call Accuracy Comparison

Figures 9.6 and 9.7 show accuracy with both postprocessors. With voting, the best performer is NN-100 with a score of 82%. With the chi-test, the best is also NN-100, with a score of 82%.

With voting, accuracy is fairly even, except for the NN-20 and KDE classifiers, which give results similar to those of single frame accuracy (Figure 9.3). The calls have less variation and higher accuracy than frames, which is probably due to the averaging effect of voting, and the fact that there are more datapoints with which the classifier can make its decision.

Figure 9.7 shows accuracy with a chi-test postprocessor. The scores of all systems have become more equalized. Even the two weaklings, NN-20 and KDE, now appear competitive with the best NN and SVM classifiers. The chi-test seems to have improved accuracy. But, by how much? Figure 9.8 shows the accuracy difference between the two postprocessors. For the best performers, NN-200, NN-500, and all SVMs, the chi-test offers only a negligible improvement, and in some

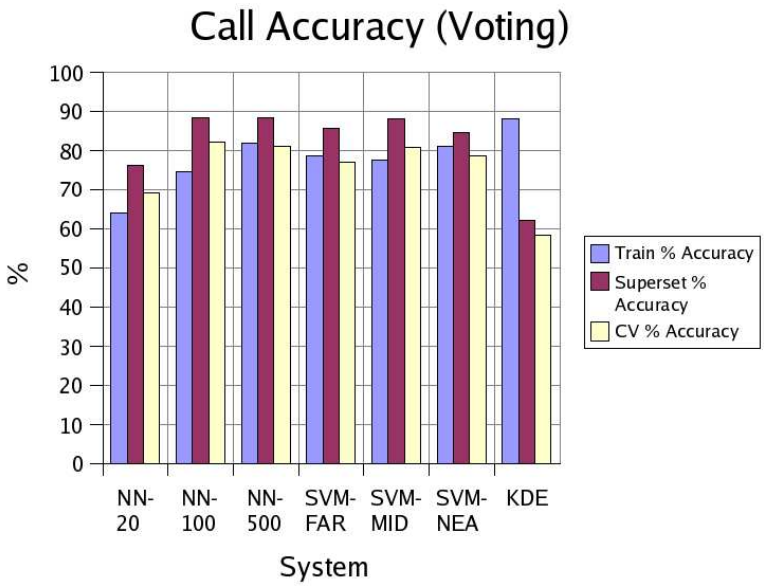

Figure 9.6: Call accuracy with a voting postprocessor.

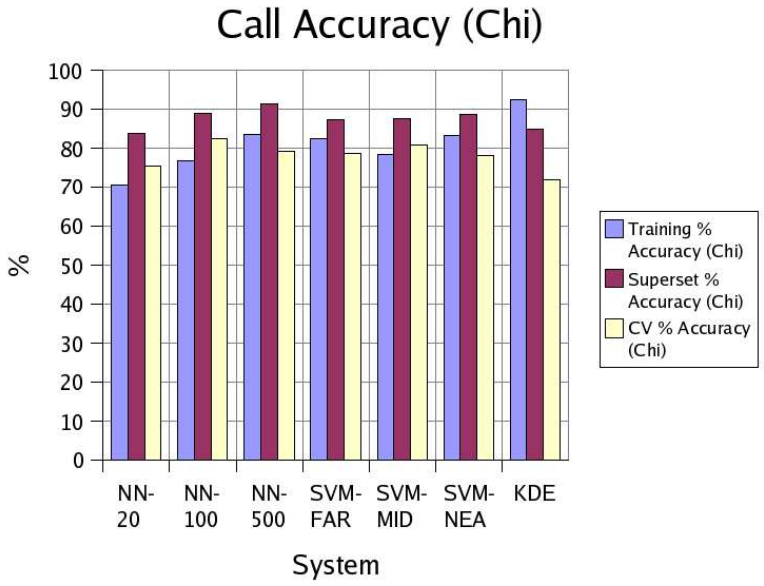

Figure 9.7: Call accuracy with a chi-test postprocessor.

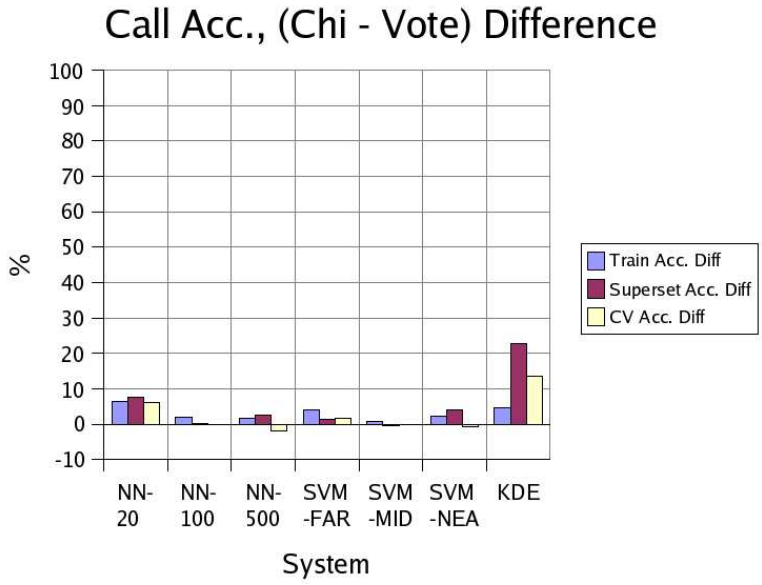

Figure 9.8: The difference between call accuracy when using voting as opposed to chi-test.

cases, a minor reduction of up to 2 percentage points. It improves the CV scores of NN-20 and KDE by 6 and 14 percentage points, a moderate improvement, but nothing spectacular.

For overall accuracy, the chi-test gives only a moderate improvement for weak classifiers.

9.3.2 Call Accuracy Floor

Figures 9.9 and 9.10 show the accuracy floor, which is the score of the single worst performing species. The accuracy floor with a voting postprocessor is shown in Figure 9.9. The scores have degraded significantly from the average accuracy (Figure 9.3). NN-20 and KDE show that one species, ALFL, has a 0% accuracy rate for all three datasets (see also Table 9.5). SVM-FAR has a score of 7% for ALFL — less than chance. Further inspection of Tables 9.5 and 9.3 shows that ALFLs accuracy scores have a greater standard deviation that other species (Table 9.5), and it has a slightly higher median rejection rate (Table 9.3), but ALFL does not appear to be a species that is inherently difficult to classify.

The best scorers are NN-100 and NN-500 at 27% each, still less than half their respective accuracies of 82% and 81%.

The chi-test postprocessor improves the accuracy floor scores, as shown in Figure 9.10. Here, the low scores are for NN-20 and KDE, both at 36%. The best is SVM-FAR, at a respectable 63%. The chi-test postprocessor clearly raises the accuracy floor. Figure 9.11 shows the difference between chi-test and voting postprocessing. The chi-test improves the floor by at least 16 percentage points

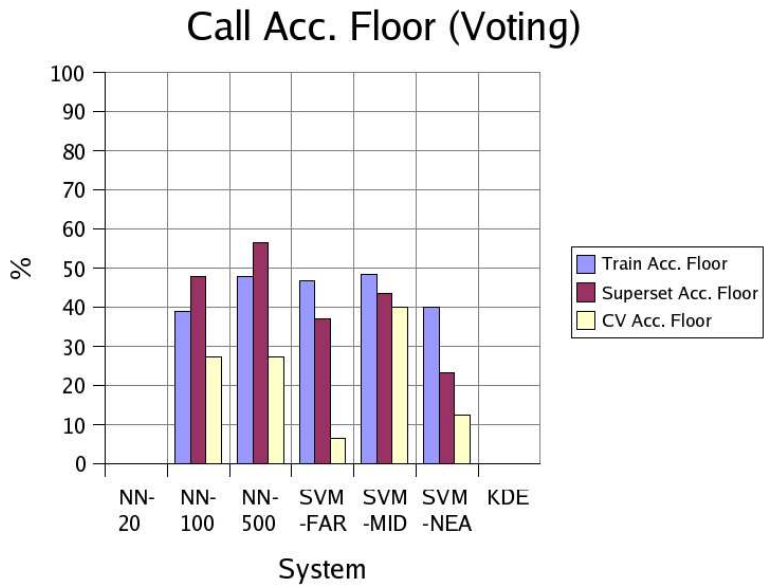

Figure 9.9: Call accuracy floors when using a voting postprocessor.

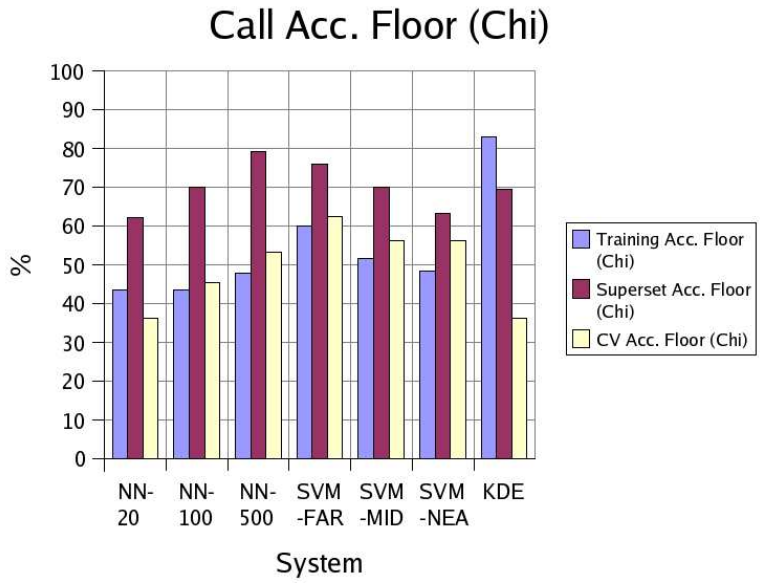

Figure 9.10: Call accuracy floors when using chi-test postprocessor.

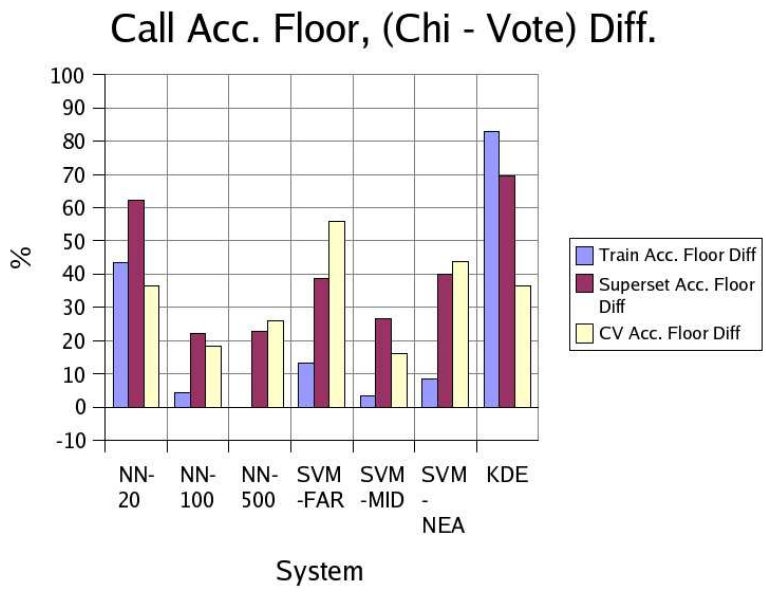

Figure 9.11: Difference in accuracy floor with voting as opposed to chi-test.

for SVM-MID, and up to 56 points for SVM-FAR. This chart has no negative values; on average, the chi-test increases the floor by 33 points.

Accuracy Variance

You might have noticed that, although the accuracy floor has increased with the chi-test, the average has changed very little (Figure 9.8). Does this say anything about how the *maximum* accuracy value is affected by the chi-test? After all, if the minimum increases, but the average remains the same, then that implies that the maximum value would have to come down to keep the average at the same spot. Figure 9.12 shows that this is indeed the case. The chart shows the standard deviation of the CV accuracy scores for each classifier. With voting, the σ value averages 29.6; with the chi-test, it is reduced by half, to 15.4. Thus, the chi-test postprocessor decreases the variance of classifier accuracy, making performance more predictable.

9.4 Median Confusion Matrices

The confusion matrices in Figures 9.13 and 9.14 show the medians of the CV frame and call accuracies across all classifiers. As such, it gives a meta-analysis of classifier performance. The cells are grayscale coded, so that higher values are lighter in color, which makes problem areas more obvious.

After perusing the CMs, some problem species pop out:

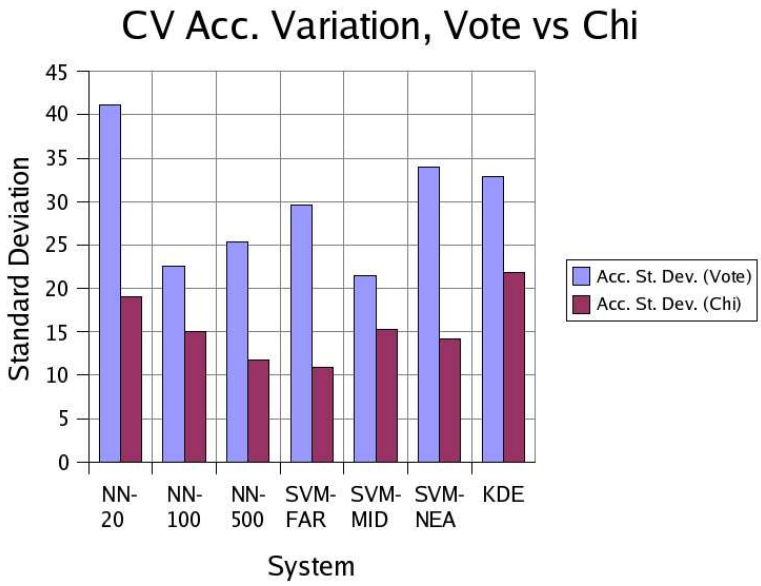

Figure 9.12: Variance of accuracy, voting vs chi-test

Species	ALFL	AMCR	AMGO	AMRE	AMRO	BAOR	BCCH	BCTI	BDOW	BLJA
ALFL	57	0	0	12	0	0	7	6	0	3
AMCR	0	93	1	0	1	0	0	0	0	5
AMGO	7	4	64	0	8	3	3	0	0	0
AMRE	16	0	0	82	0	0	1	0	0	0
AMRO	9	0	6	0	69	7	0	5	0	0
BAOR	7	0	8	0	27	40	0	4	0	0
BCCH	48	0	4	11	0	0	24	0	0	0
BCTI	11	0	0	0	0	0	0	82	0	0
BDOW	0	0	0	2	2	0	0	0	90	0
BLJA	5	6	0	0	13	0	0	0	0	52

Figure 9.13: Median confusion matrix of all classifiers when recognizing frames.

Species	ALFL	AMCR	AMGO	AMRE	AMRO	BAOR	BCCH	BCTI	BDOW	BLJA
ALFL	93	0	0	0	0	0	0	7	0	0
AMCR	0	100	0	0	0	0	0	0	0	0
AMGO	0	0	86	0	0	0	0	0	0	0
AMRE	3	0	3	87	3	0	0	0	0	0
AMRO	0	0	7	0	83	0	0	3	0	0
BAOR	0	0	6	0	38	44	0	0	0	0
BCCH	30	0	10	20	0	0	40	0	0	0
BCTI	8	0	0	0	17	0	0	67	0	0
BDOW	0	0	0	0	0	0	0	0	100	0
BLJA	0	9	9	0	27	0	0	0	0	36

Figure 9.14: Median confusion matrix of all classifiers when recognizing calls.

- The BCCH (black capped chickadee) is often mistaken for an ALFL (alder flycatcher).
- The BAOR (Baltimore oriole) is often mistaken for an AMRO (American robin).
- The BLJA (bluejay) is often mistaken for an AMRO (American robin).

Some of these errors can be explained by looking at the spectrograms and listening to the recordings and observing that the calls are similar, such as with the BAOR and AMRO. With some other errors, such as the BLJA being confused for an AMRO, it is not clear where the source of the problem is. The calls do not seem all that similar. Maybe the frequency ranges are about the same, and the overall phrasing is close, but it is still clear (to a person) that they are different.

Since this confusion is common across all recognition methods, it must show a deficiency with something outside of the recognizers, in particular, with the preprocessing stage, which never changes. Recall that, in a way, preprocessing "compresses" data and extracts a small set of features, and necessarily discards data that is not deemed important. Recognizer errors will occur if the ability to distinguish two species depends on some quality that was culled.

A solution to this would be to simply add more features, especially ones that are good at distinguishing the BLJA from the AMRO. However, with adding features, you increase the complexity of the problem, and come closer to colliding with the Curse of Dimensionality. Another solution would be to train an auxiliary network that is been trained to only distinguish robins and bluejays. This network would be called into action whenever one of these species is encountered, to add an extra "opinion" to the system. Yet again, that adds complexity. Since this thesis was intended to investigate, rather than solve, a problem, the BLJA and AMRO inadequacies are left untouched, to serve as a datapoint to ponder.

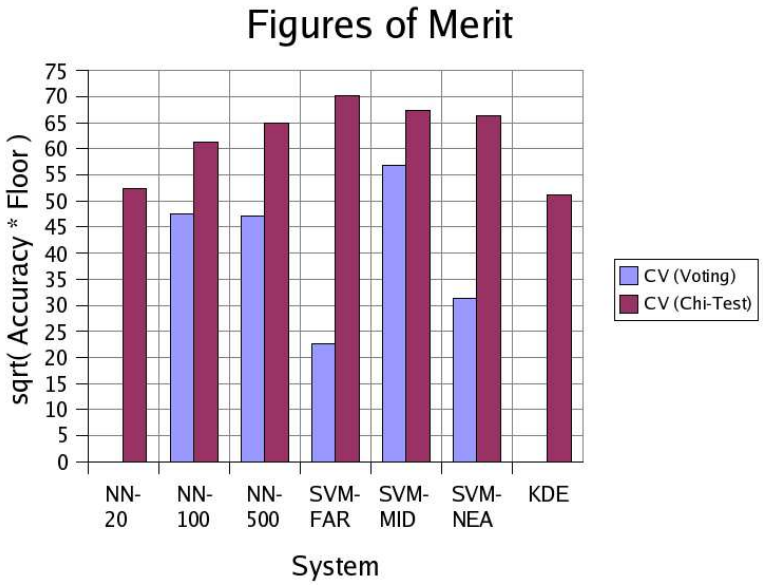

Figure 9.15: Figures of merit for classifiers.

9.5 A Final Figure of Merit

Given all these numbers, is there some way to combine a few of them to create an informative *figure of merit* (FOM)? A good FOM would probably ignore the scores of the training set and superset — the CV set is the one that everyone is interested in. It would have to include both the accuracy and accuracy floor. Adding them together would not be very useful; you would get the same value for (100,0) and (50,50). A good FOM should penalize very low accuracy or floor values. Multiplying the two gives a better result. Thus, the figure of merit used here is the geometric mean, or

$$\text{FOM} = \sqrt{(\text{CV Accuracy}) \cdot (\text{CV Accuracy Floor})} \tag{9.1}$$

Figure 9.15 shows the CV FOMs of all classifiers for both voting and chi-test postprocessing. As shown earlier, chi-test works better than voting. However, if you were forced to choose a classifier that used voting, then SVM-MID is the best with a FOM of 57, and NN-100 is second with a score of 47. With chi-test, the three SVM classifiers score better than the rest, with SVM-FAR having the highest score at 70, which is the geometric mean of 79% for accuracy and 63% for the floor. This is not the best accuracy of the lot (it comes third) but it is the highest floor.

In summary, NN-100, NN-500, and all three SVMs performed the best. NN-20 and KDE were worse, yet far better than a random predictor, with average accuracies of seven times chance.

9.6 Speed Issues

The previous portion of this chapter ignored the practical matter of classifier speed. In this section, computational requirements are briefly analyzed.

9.6.1 Training and Classification Speed

Both the NN and SVM models required several hours of training to produce the best results. After training, the classification time was longer with the SVM. The KDE model has no training phase whatsoever, but, like the SVM, was also slow.

Using the parameters that gave the best results, an estimate can be made for classification times. In the following table, A is the time required for an addition operation, M is the time for a multiplication, and φ is the time for an $\exp(\cdot)$. Experimentation showed that A, M and φ relatively took times of 1, 1 and 10 approximately.

System	Complexity	Values at Best Accuracy	Total
NN-100	$(M+A)(n_{hid}(n_{inp}+n_{out}))+n_{hid}\varphi$	$2(100(20+10))+1000$	7000
SVM-MID	$M(n_{SV}(2+n_{inp}))+A(2*n_{SV})+n_{SV}\varphi$	$3000(22)+6000+60000$	132000
KDE	$n_{train}\varphi+An_{train}+(M+A)(n_{train}+n_{inp})$	$39000+3900+2(3900+20)$	50740

The neural network classifier is faster than an SVM or KDE, by far. This is just a crude approximation though. Optimizations, such as using a *fast Gauss transform* (Greengard and Strain, 1991) could speed up the KDE, as explained in (Elgammal *et al.*, 2003), and possibly also the SVM.

9.6.2 Backpropagation Speed

The standard delta rule worked well for NNs. No embellishments were needed to speed up the training. Perhaps speed tweaking of the delta rule is a relic of the early days of backpropagation, because of the increasing power of computers. A quick calculation with Moore's law* shows that the computers of today (2005) are ≈ 6000 times more powerful than the computers when Rumelhart and McClelland released *Parallel Distributed Processing* in 1986. To put this in perspective, a 1 CPU-hour trial today would have taken 8 CPU-months in 1986. Perhaps the days of backpropagation tweaks are numbered, because increasing CPU speeds will make plain backpropagation fast enough for most applications.

*A common interpretation of Moore's law is that CPU speeds double every 18 months.

Chapter 10

Conclusion

This thesis investigated the ability of pattern recognition techniques to make an automated determination of bird species based on audio recordings of calls. Inspiration was drawn from earlier research on musical instrument recognition.

In this project, short-term tonal characteristics alone were used as features, as opposed to global qualities that have been commonly used in earlier research.

Each bird call was separated into frames of 512 samples. Well-known spectral and cepstral pitch characteristics, as well as the short term amplitude envelope, were extracted and used as features. These features were chosen in part because of their resistance to noise. Spectral and cepstral analysis together allowed pitch determination of signals ranging from pure sinusoids to those rich in harmonics. Global characteristics, such as the duration, structure, and order of sounds, were ignored.

Three pattern classification techniques were evaluated: artificial neural networks (ANN) with backpropagation; the more recently invented support vector machines (SVM); and kernel density estimation (KDE), an old statistical technique.

Each of these recognizers were trained to convert a single frame into a species estimate. A high discrimination threshold was selected by automatic inspection of the receiver operating characteristic (ROC) curve which allowed the ANN and SVM to reject low-confidence frames. Since an entire bird call is composed of dozens of frames, two postprocessing methods were used to condense a groups of estimates into a single estimate for the complete call.

The first method was simple voting. Each frame, when processed, gave a “most likely” species. When all frames of a call had been processed, the species with the most votes was selected as the winner.

The second method relied on the chi-squared goodness-of-fit test. After the species votes were tallied for all frames in a call, what remained was a multinomial distribution (or "confusion row") that could be matched to a row in the confusion matrix of the training set. The row that gave the lowest χ^2 value was the winner.

Both methods gave similar average accuracy scores, with the chi-test giving moderate improvements for the weaker classifiers.

The beneficial effects of the chi-test was seen when examining the accuracy floor, which is the accuracy of the worst-performing species for a classifier. The accuracy floor is an indication of worst-case performance expectations. The chi-test raised the floor for all classifiers. For the weaker classifiers, the increase was significant. The chi-test also reduced the variance of per-species accuracy, making performance more predictable.

To evaluate overall performance, the geometric mean of average accuracy and accuracy floor was used as a figure of merit. Based on this figure, the best classifiers were the support vector machines (with SVM-FAR having a CV accuracy of 79% and a floor of 63%) followed by the neural networks and kernel density estimation.

In summary, the three most interesting results of this thesis are:

- Short-term tonal qualities, which ignore global characteristics of a call, are adequate for species recognition;
- Multiple species estimates from a call can be combined using a voting algorithm or goodness-of-fit test to give a good accuracy score;
- The chi-square goodness-of-fit test can be used to improve the accuracy of weak classifiers, and also reduces the variance of accuracy across classifiers.

10.1 Future Directions

During the development of this project, there were many occasions when variations could have been applied to the techniques being evaluated. Unfortunately, due to time constraints, these could not be investigated here — those journeys will have to wait for future research projects. Following is a description of some of the potential areas of investigation.

10.1.1 More Species

Currently, this project investigates the classification of only ten species. What would happen if this were increased to fifty (a number which was the original goal) or more? How will the current

classifiers degrade with additional species? If they are made more powerful by adding neurons or support vectors, how will performance change? What will happen if hundreds of species are trained for? Will recognition be tenable, or will a hierarchical system be needed that does a crude classification first, then one or more additional classifications to fine-tune the recognition?

10.1.2 Different Features

There is always an urge to use a large number of features for the classifiers, but restraint was deliberately exercised here in an attempt to keep dimensionality under control. One particular feature which came to light after the fact was a frequency modulation parameter. This would be an analysis of the short-term pitch by taking the peak frequency and determining its periodicity. This might be a useful feature for birds that emitted warbling sounds.

One of the goals of this work was to use only local or instantaneous features. An obvious question is then, how would performance improve if global characteristics, such as syllable durations, inter-syllable spacings, and total call duration, were used?

Finally, are any of the current features redundant or useless for recognition? Maybe pruning algorithms can be used to determine which features are salient.

10.1.3 Preprocessing Robustness

Good features are of utmost importance. ANNs and SVMs may be powerful, but they are not telepathic; if given a feature that is no different from a random variable due to noise, there are limits to what can be deduced from it.

Features were chosen based on many criteria, but one aspect was robustness against noise. Do there exist other types of local or global features that are as robust (if not more) than the ones used here?

10.1.4 Musical Instruments and Beyond

The features used in this project were inspired by research into recognition of musical instruments. How would the system developed here fare against other classifiers for musical instruments?

How would this system perform when recognizing other sounds altogether, such as non-bird animals, insects, etc., or mechanical sounds?

10.1.5 Continuous Processing

The current system classifies birds on a per-file basis. Could it be modified to operate continually, as would be required in a real-world monitoring situation? Perhaps a several-second "sliding window" would work.

10.1.6 More KDE

Although KDE was shown to be a poor performer, the sheer simplicity of the method is, in itself, appealing. Investigating the various bandwidth selection techniques could be an interesting project unto itself.

Appendix A

Species Description

Ten species of North American birds were used in this thesis. All these species can be found in Manitoba, with the exception of the black-crested titmouse, which is limited to Texas. This section shows pictures of the birds, along with a spectrogram to give an impression of how they might sound. Audio files of most of these calls can be found on the internet, at `http://www.antiquark.com/birds`. The images in this section were obtained from Wikipedia: The Free Encyclopedia, at `http://www.wikipedia.org`.

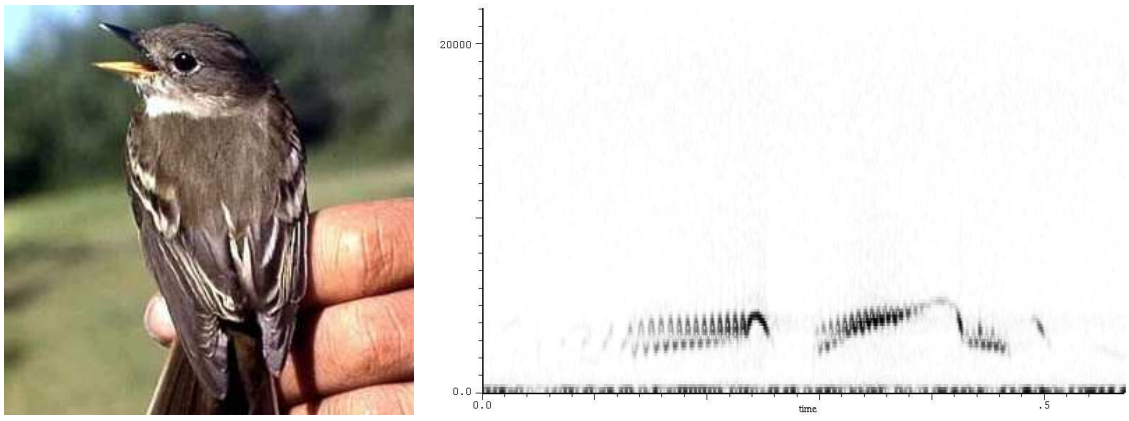

Figure A.1: Alder Flycatcher (ALFL) Picture and Spectrogram

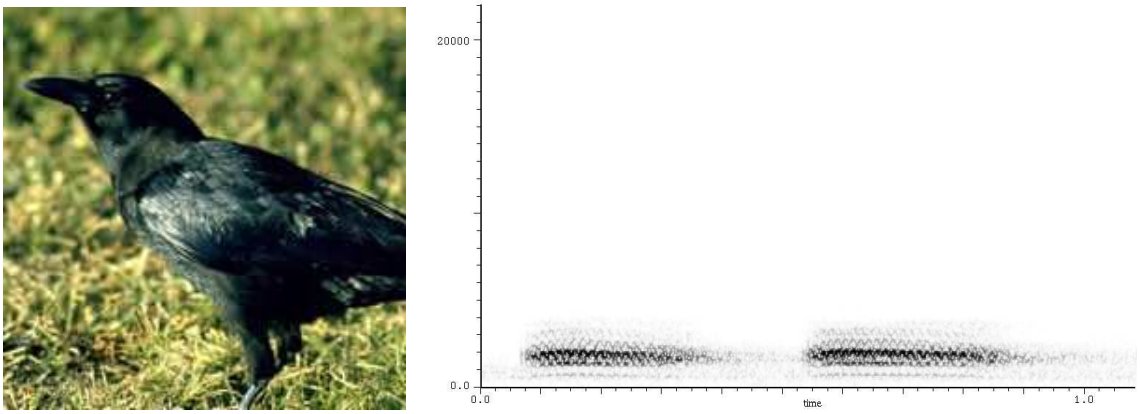

Figure A.2: American Crow (AMCR) Picture and Spectrogram

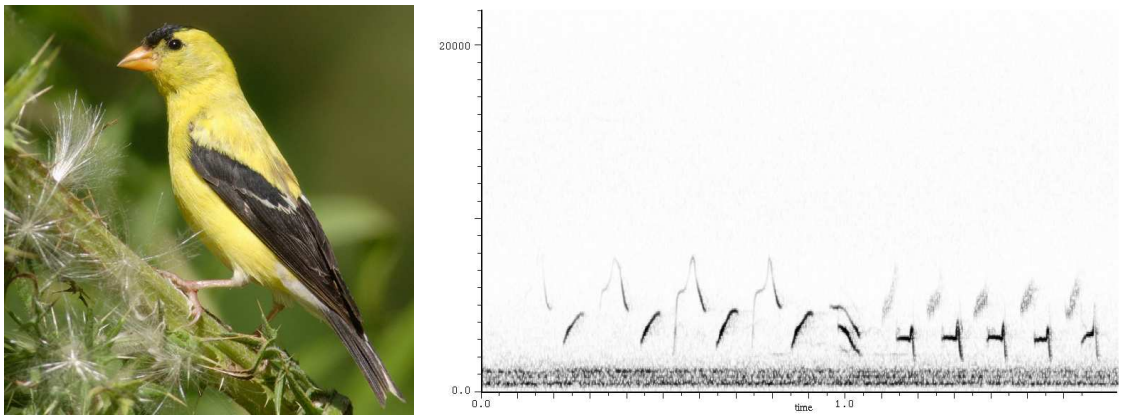

Figure A.3: American Goldfinch (AMGO) Picture and Spectrogram

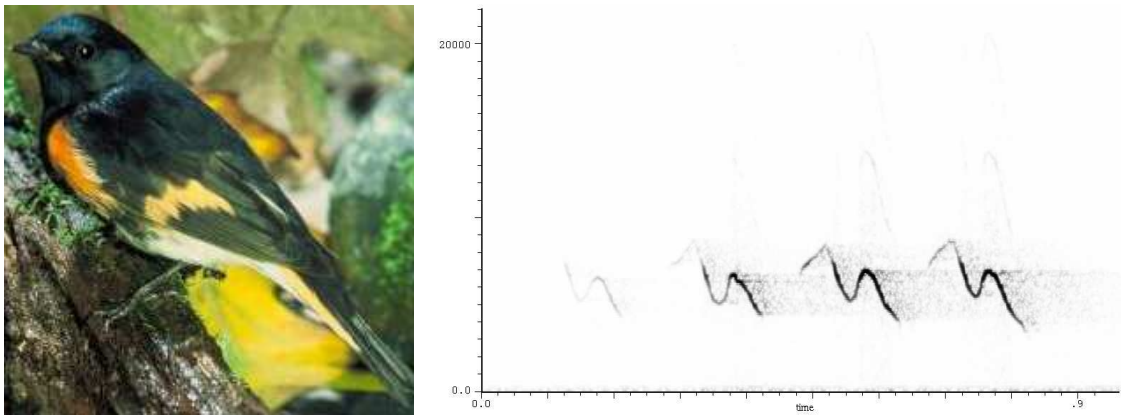

Figure A.4: American Redstart (AMRE) Picture and Spectrogram

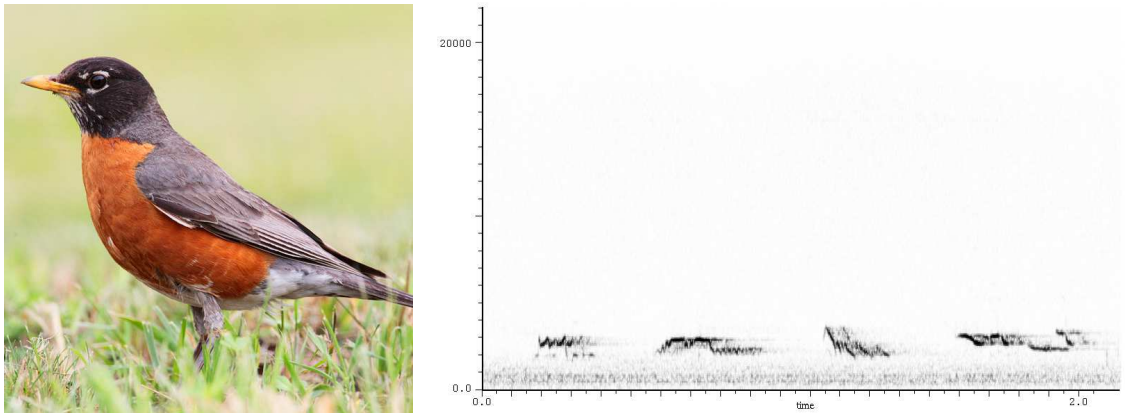

Figure A.5: American Robin (AMRO) Picture and Spectrogram

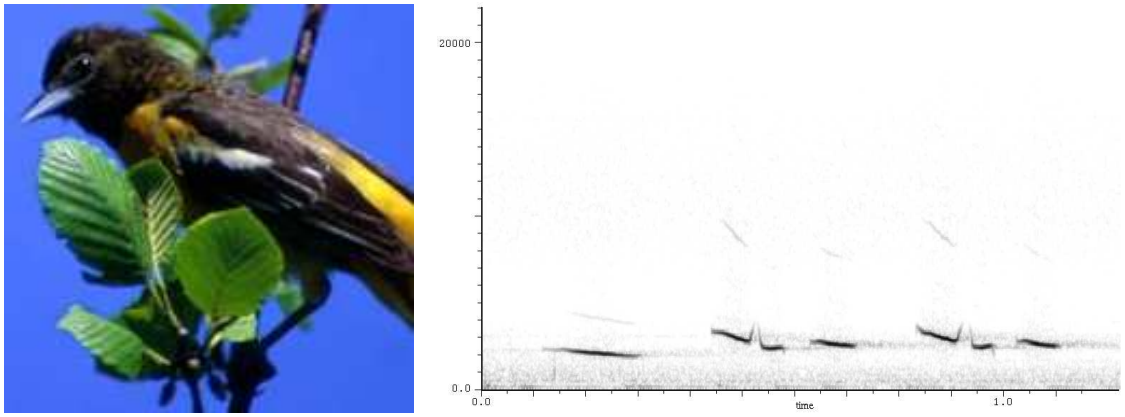

Figure A.6: Baltimore Oriole (BAOR) Picture and Spectrogram

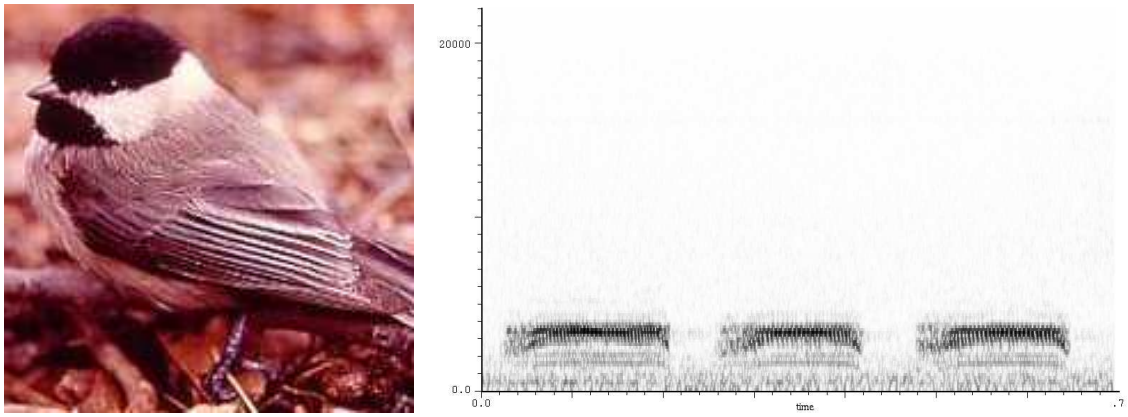

Figure A.7: Black-Capped Chickadee (BCCH) Picture and Spectrogram

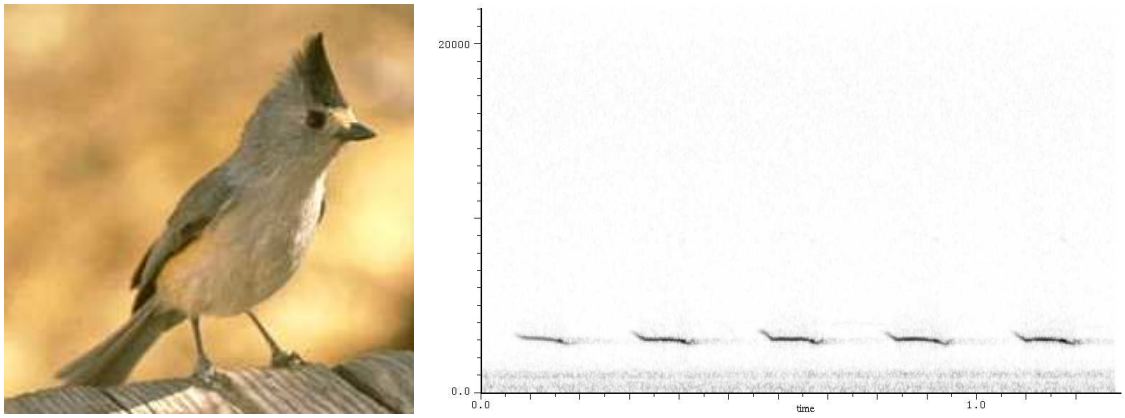

Figure A.8: Black-Crested Titmouse (BCTI) Picture and Spectrogram

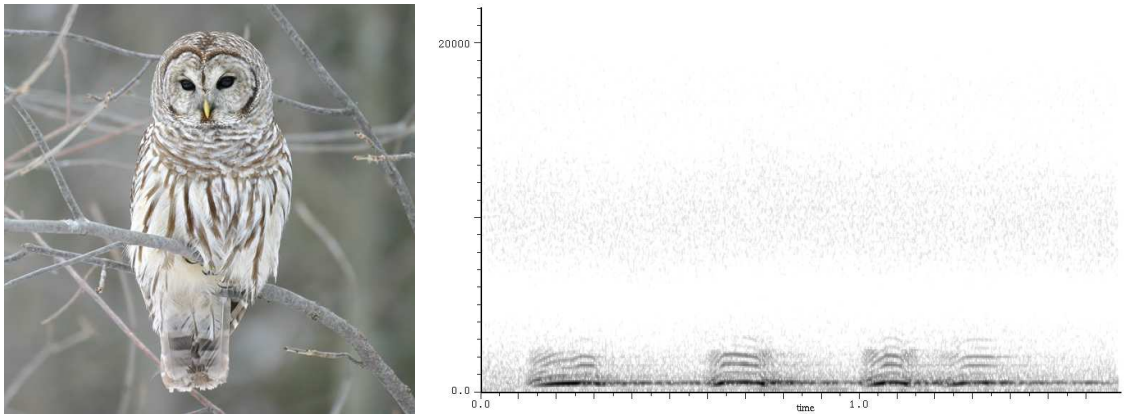

Figure A.9: Barred Owl (BDOW) Picture and Spectrogram

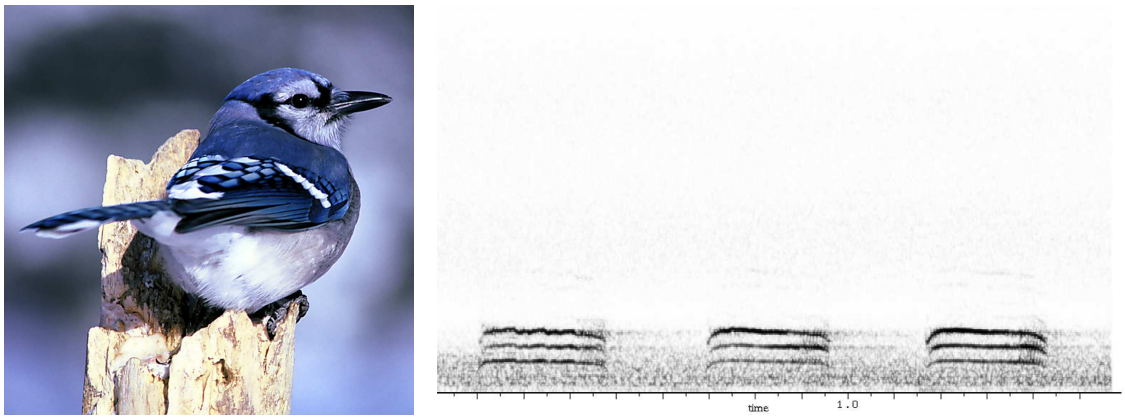

Figure A.10: Blue Jay (BLJA) Picture and Spectrogram

Appendix B

Choosing Features

The background work for this thesis, involved experimentation with many preprocessing methods to try to determine which ones were best suited for use in a pattern recognition context. Most of the experiments failed, but in the process several rules of thumb were built up, to help qualify and choose features. Here, for the reader's interest, is a list of these rules, in no particular order.

B.1 Use Well-Known Pre-Processing Methods

You should use tried-and-true techniques for preprocessing. It might be possible to invent a novel and innovative feature, but if you do so, then it is necessary to analyze this new technique, prove that it works, then implement it and show that the implementation works. Transferring your creativity and inventiveness to the problem that comes after the pre-processing stage will simplify the overall task (Chen, 1973).

B.2 Noise Rejection

A feature should be immune (or at least resistant) to variations in background noise or other extraneous forms of interference (Devijver and Kittler, 1982).

B.3 Features Should be Reversible

Given a set of features, you should be able to reconstruct (to some extent) the original signal (Devijver and Kittler, 1982). This shows that the features you have chosen retain much of the key

information of the original, just in a different format. A perfect example of this is the Fourier transform, which can be inverted to produce the original signal.

B.4 Dimensionality Reduction

You should make efforts to reduce the number of dimensions of the input signal. This will help the subsequent processing stages avoid the Curse of Dimensionality (Bishop, 1995). Some literature indicates that about 10–20 dimensions is the maximum useful dimensionality for a feature space (Scott, 1992). An obvious solution would be to simply use fewer features. Another workaround is to project a number of features onto a lower-dimensional space, a technique that has its share of problems (Scott, 1992).

B.5 Invariance to Amplitude Changes

A feature should be invariant to changes in amplitude Both weak and strong signals of the same category should produce the same value for a given feature (Winston, 1984). (Although, in the case of this thesis, the signals were largely normalized for amplitude because of auto-gain circuitry in the recording devices, and manual gain control by the people operating the equipment.)

B.6 Input/Output Space Smoothness and Continuity

A transformation that converts a point in the input space to a feature should be smooth and continuous. Similar points in the input space should produce features with similar values (Devijver and Kittler, 1982). To further illustrate, here is an example.

Assume that your input space is a series of samples from an audio signal. If you were to apply the Fourier transform as a means to find the peak frequency component, you might get a result as shown in Figure B.1. Here, similar input frequencies will produce similar value for the extracted peak frequency.

Consider the case where you now have a more complicated signal with multiple peaks in the frequency domain, as shown in Figure B.2.

If the three peaks in the frequency domain are close in magnitude, then a simple peak detection algorithm could produce one of several results, depending on small variations in the original signal caused by noise. For example, if the peaks were 100 Hz apart, then the detected frequency could be either 100, 200 or 300 Hz.

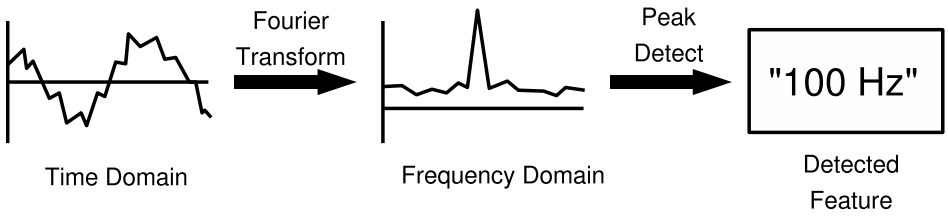

Figure B.1: Simple spectral feature extractor.

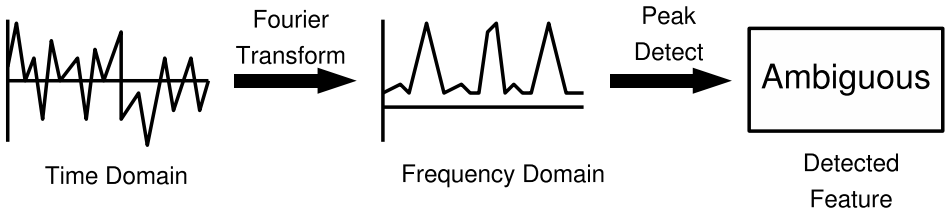

Figure B.2: Simple spectral feature extractor with ambiguous result.

The solution to this particular problem is to add another transform and find the *cepstrum* of the signal (Section 3.3). The cepstrum will convert a comb-like structure in the frequency domain into a single peak (Figure B.3).

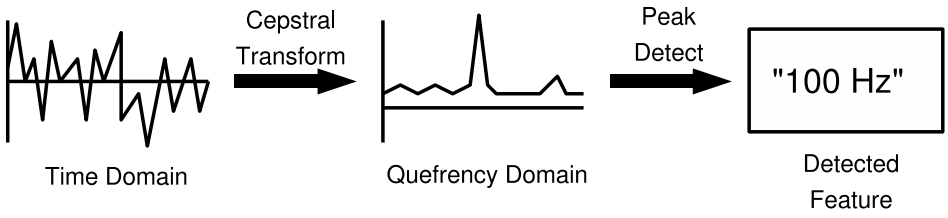

Figure B.3: Simple cepstral feature extractor.

This peak value, 100 Hz, is much more resistant to noise. The magnitude of the peaks after the initial Fourier transform can vary, but the cepstral estimate of the spacing between the peaks will remain stable.

B.7 Avoid Binning or Thresholds

This is related to the previous item on smoothness and continuity (Devijver and Kittler, 1982). It might appear to be simplifying things if data is broken down into bins before being passed to a

classifier. For example, a signal level could be separated into "weak" and "strong" bins before processing. Information is lost if this is done. A "strong" signal that is almost straddling the boundary will be in the same bin as the strongest signal in the dataset. It is better to simply pass the signal level to the classifier.

B.8 Ease of Implementation

A feature extractor is worthless if you cannot get it to operate properly. If you plan to code an algorithm from scratch, it is a good idea to choose one that can be implemented and tested in a timely manner, and with some confidence that it actually works (Chen, 1973).

B.9 Avoid Conceptual Cross-Contamination

If you plan to compare the performance of, say, neural networks to SVMs, then the preprocessor should contain neither NNs nor SVMs. If it does, then it is more difficult to isolate performance differences. This rule-of-thumb came to mind when investigating pitch detection methods for the preprocessing stage, and it was discovered that neural networks have been used for that purpose.

B.10 Confirm Assumptions with Experiment

It is possible to propose features which, at first glance, and even after a longer, harder look, appear ideal for extracting useful information. Only through subsequent testing on real data do the limitations of the proposed method become apparent. One example of such a case is using the standard deviation of the frequency spectrum. The standard deviation (or σ) should provide an indication of the overall "shape" of the spectrum. A spectrum with a sharp, well-defined peak would have a small σ, but a spectrum with a spread-out peak, or with multiple harmonics, would give a larger σ. Intuitively, this appears to be a useful measure.

In reality however, the σ of the spectrum is useless for describing the shape of the peak. In a laboratory environment, σ would indeed be useful, but in real-life signals with background noise, σ turns out to be little better than a random variable.

In an outdoor environment, the background noise typically has more energy at lower frequencies, and is similar to pink or green noise which has a $1/f$ distribution. Figure B.4 shows the type of spectrum that might be found in a real signal.

As can be seen by inspecting the resulting spectrum of Figure B.4, the σ value will vary depending on:

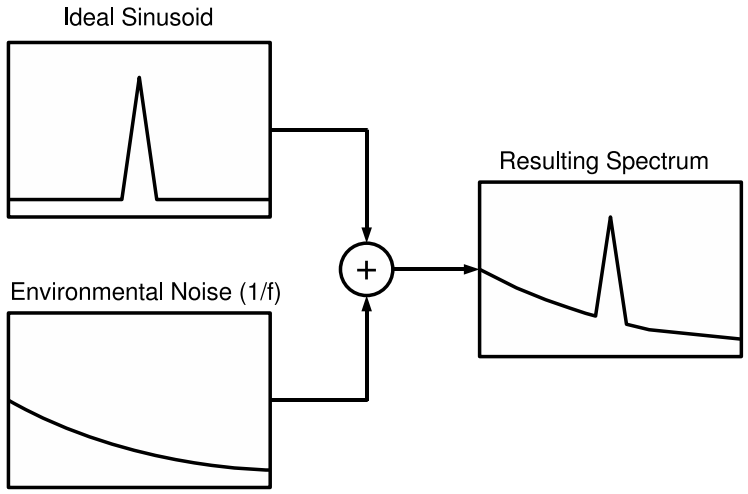

Figure B.4: Effect of ambient noise on signal.

- The frequency of f_o;
- The amplitude of f_o;
- The amplitude of the background noise;
- The harmonics of f_o.

To belabour this point: one should not *assume* that a feature is useful; one should *show* that a feature is useful.

B.11 Confidence Metric

Every good feature should have an extra value that gives an indication of how "reliable" it is. For example, if you have a frequency estimate of some signal, you would like to know if it is the result of a strong component, or merely a random blip of noise. The magnitude of the component is a good confidence metric, but that would fail if the signal is composed only of loud noise. A better metric, one that is used in this thesis, is to normalize the peak energy by dividing it by the energy of the frame. This indicates the magnitude of the peak relative to the rest of the signal, and will produce a lower value if noise is present.

B.12 Automated Feature Selection

There are many methods for automatically selecting a subset of features from a large features set. An exhaustive search is impossible, as it increases as $d!$ where d is the size of the feature set. Heuristics have to be used instead. These methods were not utilized, as they require the classifier(s) to be retrained dozens of times, which, for this project, would have taken months of CPU time. A good comparison of feature selection algorithms is by Jain and Zongker (1997).

Bibliography

Agostini, G., Longari, M., and Pollastri, E. (2001a). Content-based classification of musical instrument timbres. In *Proc. of Content-Based Multimedia Indexing, IEEE Multimedia Processing TC*, pp. 159–166. Università degli Studi di Brescia, Brescia, Italy.

Agostini, G., Longari, M., and Pollastri, E. (2001b). Musical instrument timbres classification with spectral features. In *IEEE Fourth Workshop on Multimedia Signal Processing*, pp. 97–102. Cannes, France.

Agresti, A. (1996). *An Introduction to Categorical Data Analysis*. Toronto: Wiley.

Aizerman, A., Braverman, E. M., and Rozoner, L. I. (1964). Theoretical foundations of the potential function method in pattern recognition learning. *Automation and Remote Control* **25**, 821–837.

Berger, R. (2005). Personal communication.

Bishop, C. M. (1995). *Neural Networks for Pattern Recognition*. Oxford: Clarendon Press.

Boll, S. F. (1979). Suppression of acoustic noise in speech using spectral subtraction. *IEEE Trans. on Acoustics, Speech and Signal Processing* **27**, 113–120.

Bowman, A., Hall, P., and Prvan, T. (1998). Bandwidth selection for the smoothing of distribution functions. *Biometrika* **85** (4), 799–808.

Bruder, J. A., Cavo, V. N., and Wicks, M. C. (1998). Bird hazard detection with airport surveillance radar. In *8th Annual Meeting, Bird Strike Committee*, pp. 160–163. Cleveland, Ohio: Internet Center for Wildlife Damage Management.

Burges, C. J. C. (1998). A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery* **2**, 121–167.

Campbell, C. (2002). Kernel methods: a survey of current techniques. *Neurocomputing* **48**, 63–84.

Chang, C.-C. and Lin, C.-J. (2005). *LIBSVM: a library for support vector machines*. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm` as of Jan. 2006.

Chen, C.-H. (1973). *Statistical Pattern Recognition*. Rochelle Park, New Jersey: Hayden Book Company, Inc.

Chiu, S.-T. (1996). A comparative review of bandwidth-selection for kernel density estimation. *Statistica Sinica* **6**, 129–145.

Cristianini, N. and Shawe-Taylor, J. (2000). *An Introduction to Support Vector Machines and other kernel based learning methods*. Cambridge: Cambridge University Press.

Cucker, F. and Smale, S. (2001). On the mathematical foundations of learning. *Bulletin (New Series) of the American Mathematical Society* **39** (1), 1–49.

de Cheveigné, A. and Kawahara, H. (2001). Comparative evaluation of F_0 estimation algorithms. In *Eurospeech 2001 - Scandinavia*.

Dennis, J. V. (1964). Woodpecker damage to utility poles: With special reference to the role of territory and resonance. *Bird-Banding: A Journal of Ornithological Investigation* **35** (4), 225–253.

Derégnaucort, S., Guyomarc'h, J.-C., and Richard, V. (2001). Classification of hybrid crows in quail using artificial neural networks. *Behavioral Processes* **56**, 103–112.

Devijver, P. A. and Kittler, J. (1982). *Pattern Recognition: A Statistical Approach*. Englewood Cliffs, New Jersey: Prentice/Hall International.

Devore, J. L. (1987). *Probability and Statistics for Engineering and the Sciences*. Monterey, California: Brookes/Cole.

Dubnow, J. J., Schafer, R. W., and Rabiner, L. R. (1976). Real-time digital hardware pitch detector. *IEEE Trans. on Acoustics, Speech and Signal Processing* **24** (1), 2–8.

Egan, J. P. (1975). *Signal Detection Theory and ROC Analysis*. New York: Academic Press.

Elgammal, A., Duraiswami, R., and Davis, L. S. (2003). Efficient kernel density estimation using the fast gauss transform with applications to color modeling and tracking. *IEEE Trans. on Pattern Analysis and Machine Intelligence* **25** (11), 1499–1504.

Eronen, A. (2001). Comparison of features for musical instrument recognition. In *2001 IEEE Workshop on Applications of Signal Processing to Audio and Acoustics*, pp. 19–22. New Paltz, New York.

Evans, W. R. (1998). Applications of acoustic bird monitoring for the wind power industry. In *National Avian - Wind Power Planning Meeting III*, pp. 141–152. San Diego, California: National Wind Coordinating Committee.

Evans, W. R. (2005). Monitoring avian night flight calls—the new century ahead. *The Passenger Pigeon* **67** (1), 15–24.

Everitt, B. S. (1992). *The Analysis of Contingency Tables*. New York: Chapman and Hall.

Fahlman, S. E. and Lebiere, C. (1990). The cascade-correlation learning architecture. In Touretzky, D. S. (Ed.) , *Advances in Neural Information Processing Systems 2*, pp. 524–532. San Mateo, CA: Morgan Kaufman.

Farnsworth, A. (2005). Flight calls and their value for future ornithological studies and conservation research. *The Auk* **122** (3), 733–746.

Farnsworth, A., Gauthreaux, S. A., and van Blaricom, D. (2004). A comparison of nocturnal call counts of migrating birds and reflectivity measurements on doppler radar. *Journal of Avian Biology* **35**, 365–369.

Gersho, A. and Gray, R. M. (1992). *Vector Quantization and Signal Compression*. Boston: Kluwer.

Greengard, L. and Strain, J. (1991). The fast gauss transform. *SIAM J Sci Stat Comput* **12** (1), 79–94.

Greenwood, P. E. and Nikulin, M. S. (1996). *A Guide to Chi-Squared Testing*. Toronto: Wiley.

Härmä, A. and Somervuo, P. (2004). Classification of the harmonic structure in bird vocalization. In *ICASSP 2004*, pp. 701–704. IEEE.

Harness, R. and Carlton, R. (2001a). Automated systems for monitoring avian interactions with utility structures and evaluating the effectiveness of mitigative measures. In *Power Engineering Society Winter Meeting 2001*, Volume 1, pp. 359–362. IEEE.

Harness, R. and Carlton, R. (2001b). New solutions for bird collision and electrocution outage problems. In *Power Engineering Society Winter Meeting 2001*, Volume 1, pp. 341–354. IEEE.

Haykin, S. (1994). *Neural Networks: A Comprehensive Foundation*. New York: Macmillan College Publishing Company.

Hearst, M. A. (1998). Support vector machines. *IEEE Intelligent Systems* pp. 18–28. July/August 1998.

Hertz, J., Krogh, A., and Palmer, R. G. (1991). *An Introduction to the Theory of Neural Computation*. Redwood City, CA: Addison Wesley.

Hess, W. (1983). *Pitch Determination of Speech Signals*. New York: Springer-Verlag.

Hsu, C.-W. and Lin, C.-J. (2002). A comparison of methods for multiclass support vector machines. *IEEE Trans. on Neural Networks* **13** (2), 415–425.

Ivakhnenko, A. G. (1971). Polynomial theory of complex systems. *IEEE Trans. on Systems, Man, and Cybernetics* **1** (4), 364–378.

Jacobs, R. A. (1988). Increased rates of convergence through learning rate adaptation. *Neural Networks* **1**, 295–307.

Jain, A. and Zongker, S. (1997). Feature selection: Evaluation, application and small sample performance. *IEEE Trans. on Pattern Analysis and Machine Intelligence* **19** (2), 153–158.

Jain, A. K., Duin, R. P. W., and Mao, J. (2000). Statistical pattern recognition: A review. *IEEE Trans. on Pattern Analysis and Machine Intelligence* **22** (1), 4–37.

Jardine, E. (1996). *Bird Song Identification Made Easy*. Toronto, Ontario: Natural Heritage / Natural History Inc.

Jones, M. C., Marron, J. S., and Sheather, S. J. (1996). A brief survey of bandwidth selection for density estimation. *Journal of the American Statistical Association* **91** (433), 401–407.

Kanji, G. K. (1999). *100 Statistical Tests*. London: SAGE Publications.

Kecman, V. (2001). *Learning and Soft Computing: Support Vector Machines, Neural Networks, and Fuzzy Logic Models*. Cambridge, Massachusetts: MIT Press.

Kemerait, R. C. and Childers, D. G. (1972). Signal detection and extraction by cepstrum techniques. *IEEE Trans. on Information Theory* **18** (6), 745–759.

Khazanie, R. (1986). *Elementary Statistics In a World of Applications* (Second ed.). USA: Scott, Foresman and Company.

Kohavi, R. and Provost, F. (1998). Glossary of terms. *Machine Intelligence* **30** (2), 271–274.

Kramer, A. H. and Sangiovalli-Vincentelli, A. (1989). Efficient parallel learning algorithms for neural networks. In Touretzky, D. S. (Ed.) , *Advances in Neural Information Processing Systems*, Volume 1, pp. 40–48. San Mateo, CA: Morgan Kaufman.

Kulkarni, S. R., Lugosi, G., and Venkatesh, S. S. (1998). Learning pattern classification—a survey. *IEEE Trans. on Information Theory* **44** (6), 2178–2206.

Kuo, F. Y. and Sloan, I. H. (2005). Lifting the curse of dimensionality. *Notices of the AMS* **52** (11), 1320–1328.

Kwak, N. and Choi, C.-H. (2002). Input feature selection for classification problems. *IEEE Trans. on Neural Networks* **13** (1), 143–159.

Lee, Y., Oh, S., and Kim, M. (1991). The effect of initial weights on premature saturation in back-propagation learning. In *International Joint Conference on Neural Networks*, Volume 1, pp. 765–770. Seattle, WA.

Lin, C.-J. (2005). Personal communication.

Loader, C. R. (1999). Bandwidth selection: classical or plug-in? *The Annals of Statistics* **27** (2), 415–438.

Marler, P. and Slabbekoorn, H. (2004). *Nature's Music: The Science of Birdsong*. San Diego, California: Elsevier Academic Press.

Marques, J. and Moreno, P. J. (1999). A study of musical instrument classification using gaussian mixture models and support vector machines. Cambridge Research Laboratory, Technical Report Series. CRL 99/4, June 1999.

Martin, K. D. (1998). Toward automatic sound recognition: Identifying musical instruments, pp. 1–6. Presented at the NATO Computational Hearing Advanced Study Institute. Il Ciocco, Italy.

Martin, K. D. and Kim, Y. E. (1998). Musical instrument recognition: A pattern-recognition approach, pp. 1–12. Presented at the 136th meeting of the Acoustical Society of America.

Masters, T. (1993). *Practical neural network recipes in C++*. San Diego, USA: Academic Press Professional, Inc.

McIlraith, A. L. (1996). Identification of birdsong using artificial neural computing. Master's thesis, University of Manitoba.

McIlraith, A. L. and Card, H. C. (1997). Birdsong recognition using backpropagation and multivariate statistics. *IEEE Trans. on Signal Processing* **45** (11), 2740–2748.

Moore, A. W. (2001). Support vector machines. URL is `http://www.cs.cmu.edu/~awm` as of Jan. 2006. Unpublished.

Müller, K.-R., Mika, S., Rätsch, G., Tsuda, K., and Schölkopf, B. (2001). An introduction to kernel-based learning algorithms. *IEEE Trans. on Neural Networks* **12** (2), 181–201.

Noll, A. M. (1964). Short-time spectrum and 'cepstrum' techniqes for vocal-pitch detection. *Journal of the Acoustical Society of America* **36** (2), 269–302.

Oppenheim, A. V. and Schafer, R. W. (2004). From frequency to quefrency: A history of the cepstrum. *IEEE Signal Processing Magazine*. Sept. 2004, pp. 95–106.

Oppenheim, A. V., Willsky, A. S., and Young, I. T. (1983). *Signals and Systems*. New Jersey: Prentice Hall.

Platt, J. C. (1998). Sequential minimal optimization: A fast algorithm for training support vector machines. Technical report, Microsoft Research. MSR-TR-98-14.

Platt, J. C. (1999). Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. In A. J. Smola, P. Bartlett, B. Schölkopf, D. Schuurmans (Eds.), *Advances in Large Margin Classifiers*. MIT Press.

Poggio, T. and Smale, S. (2003). The mathematics of learning: Dealing with data. *Notices of the AMS* **50** (5), 537–544.

Press, W. H., Teukolsky, S. A., Vetterling, W. T., and Flannery, B. P. (1992). *Numerical Recipes in C: The Art of Scientific Computing* (Second ed.). New York: Cambridge University Press.

Proakis, J. G. and Manolakis, D. G. (1996). *Digital Signal Processing: Principles, Algorithms and Applications* (Third ed.). New Jersey: Prentice Hall.

Rabiner, L. R. (1977). On the use of autocorrelation analysis for pitch detection. *IEEE Trans. on Acoustics, Speech and Signal Processing* **25** (1), 24–33.

Rabiner, L. R., Cheng, M. J., Rosenberg, A. E., and McGonegal, C. A. (1976). A comparative performance study of several pitch detection algorithms. *IEEE Trans. on Acoustics, Speech and Signal Processing* **24** (5), 399–418.

Rosenblatt, M. (1956). Remarks on some nonparametric estimates of a density function. *Annals of Mathematical Statistics* **27**, 832–837.

Ross, M. J., Shaffer, H. L., Cohen, A., Freudberg, R., and Manley, H. J. (1974). Average magnitude difference function pitch extractor. *IEEE Trans. on Acoustics, Speech and Signal Processing* **22** (5), 353–362.

Ruiz, A. and López-de-Teruel, P. E. (2001). Nonlinear kernel-based statistical pattern analysis. *IEEE Trans. on Neural Networks* **12** (1), 16–32.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning internal representations by error propagation. In Rumelhart, D. E., McClelland, J. L., and the PDP Research Group (Eds.) , *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, Volume 1: Foundations, pp. 318–362. Cambridge, MA: MIT Press.

Rumelhart, D. E. and McClelland, J. L. (Eds.) (1986). *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, Volume 1. Cambridge, MA: MIT Press.

Russo, A. P. (1991). Neural networks for sonar signal processing, Tutorial No. 8. In *IEEE Conference on Neural Networks for Ocean Engineering*. Washington, DC.

Schölkopf, B., Burges, C. J. C., and Smola, A. J. (Eds.) (1999). *Advances in Kernel Methods: Support Vector Learning*. Cambridge, Massachusetts: MIT Press.

Schölkopf, B. and Smola, A. J. (2002). *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. Cambridge, Massachusetts: MIT Press.

Scott, D. W. (1992). *Multivariate Density Estimation: Theory, Practise, and Visualization*. New York: John Wiley and Sons.

Scott, D. W. and Wand, M. P. (1991). Feasibility of multivariate density estimates. *Biometrika* **78** (1), 197–205.

Sdorow, L. (1990). *Psychology*. Dubuque, Indiana: Wm. C. Brown Publishers.

Shilton, A., Palaniswami, M., Ralph, D., and Tsoi, A. C. (2005). Incremental training of support vector machines. *IEEE Trans. on Neural Networks* **16** (1), 114–131.

Sondhi, M. M. (1968). New methods of pitch extraction. *IEEE Trans. on Audio and Electroacoustics* **16** (2), 262–266.

Vapnik, V. (1979). *Estimation of Dependences Based on Empirical Data [In Russian]*. Moscow: Nauka. (English translation: Springer-Verlag, New York, 1982).

Vapnik, V. (1995). *The Nature of Statistical Learning Theory*. New York: Springer-Verlag.

Vapnik, V. (1998). *Statistical Learning Theory*. New York: John Wiley and Sons.

Vapnik, V. N. (1999). An overview of statistical learning theory. *IEEE Trans. on Neural Networks* **10** (5), 988–999.

Widrow, B. and Hoff, M. E. (1960). Adaptive switching circuits. In *IRE WESCON Convention Record*, pp. 96–104.

Winston, P. H. (1984). *Artificial Intelligence* (Second ed.). Reading, Massachusetts: Addison Wesley.

Wright, M. H. (2004). The interior-point revolution in optimization: History, recent development, and lasting consequences. *Bulletin (New Series) of the American Mathematical Society* **42** (1), 39–56.