Short Tandem Repeat (STR) Profile Authentication via Machine Learning Techniques

by

Anna Shcherbina

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2012

Author ....
Department of Electrical Engineering and Computer Science
May 21, 2012

Certified by ....
Dr. Anthony Lapadula
MIT Lincoln Laboratory Technical Staff
Thesis Supervisor

Certified by ....
Prof. Manolis Kellis
Associate Professor
Thesis Supervisor

Accepted by ....
Prof. Dennis M. Freeman
Chairman, Masters of Engineering Thesis Committee

Short Tandem Repeat (STR) Profile Authentication via Machine Learning Techniques

by
Anna Shcherbina

Submitted to the Department of Electrical Engineering and Computer Science
on May 21, 2012, in partial fulfillment of the requirements for the degree of
Master of Engineering in Computer Science and Engineering

Abstract

Short tandem repeat (STR) DNA profiles have multiple uses in forensic analysis, kinship identification, and human biometrics. However, as biotechnology progresses, there is a growing concern that STR profiles can be created using standard laboratory techniques such as whole genome amplification and molecular cloning. Such technologies can be used to synthesize any STR profile without the need for a physical sample, only knowledge of the desired genetic sequence. Therefore, to preserve the credibility of DNA as a forensic tool, it is imperative to develop means to authenticate STR profiles. The leading technique in the field, methylation analysis, is accurate but also expensive, time-consuming, and degrades the forensic sample so that further analysis is not possible.

The realm of machine learning offers techniques to address the need for more effective STR profile authentication. In this work, a set of features were identified at both the channel and profile levels of STR electropherograms. A number of supervised and unsupervised machine learning algorithms were then used to predict whether a given STR electropherogram was authentic or synthesized by laboratory techniques. With the aid of the LNKnet machine learning toolkit, various classifiers were trained with the default set of parameters and the full set of features to quantify their baseline performance. Particular emphasis was placed on detecting profiles generated by Whole Genome Amplification (WGA).

A greedy forward-backward search algorithm was implemented to determine the most useful subset of features from the initial group. Though the set of optimal feature values varied by classifier, a trend was observed indicating that the inter-locus imbalance error, stutter count, and range of peak widths for a profile were particularly useful features. These were selected by over two thirds of the classifiers. The signal-to-noise ratio was also a useful feature, selected by seven out of 16 classifiers.

The selected features were in turn used to tune the parameters of machine learning algorithms and to compare their performance. From a set of 16 initial classifiers, the K-nearest neighbors, condensed K-nearest neighbors, multi-layer perceptron, Parzen window, and support vector machine classifiers achieved the best performance. These classification algorithms all attained error rates of approximately ten percent, defined as the percentage of profiles misclassified with the highest performing classifier achieving an error rate of less than eight percent. Overall, the classifiers performed well at detecting artificial profiles but had more difficulty accurately distinguishing natural profiles. There were many false positives for the artificial class, since profiles in this category took on a greater range of feature values. Finally, preliminary steps were taken to form classifier committees. However, combining the top performing classifiers via a majority vote did not significantly improve performance.

The results of this work demonstrate the feasibility of a completely software-based approach to profile authentication. They confirm that machine learning techniques are a useful tool to trigger further investigation of profile authenticity via more expensive approaches.

Thesis Supervisor: Dr. Anthony Lapadula
Title: MIT Lincoln Laboratory Technical Staff

Thesis Supervisor: Manolis Kellis
Title: Associate Professor

This work is sponsored by the Assistant Secretary of Defense for Research and Engineering under Air Force Contract FA8721-05-C-0002. Opinions, interpretations, conclusions and recommendations are those of the authors and are not necessarily endorsed by the United States Government.

Acknowledgments

First and foremost I want to thank my supervisor at Lincoln Laboratory, Dr. Anthony Lapadula. Thank you for providing mentorship and advice throughout my thesis research. Your help and support has been invaluable, and I appreciate your many great ideas that have helped me overcome sticking points over the course of my research. I would also like to thank Dr. Martha Petrovick and Johanna Bobrow for providing the raw STR profile data needed for my research and suggesting features to examine in the feature identification phase of the project. I am also extremely grateful to Edward Wack, group leader of the Biological Engineering Group at MIT Lincoln Laboratory, for accepting me into the group and enabling the funding of my MEng work.

Additionally, I would like to thank Professor Manolis Kellis, my MEng advisor at MIT, for providing great suggestions about machine learning algorithms and techniques. I have learned a lot about the application of machine learning techniques to problems in biology by speaking to you about my thesis work and taking your course on computational biology.

I would furthermore like to thank the staff who run the VI-A program for providing such an amazing opportunity for students like me to obtain valuable industry experience while completing an MEng degree.

Finally, I would like to thank my parents, Tatyana Proshko and Yuri Shcherbina, for working extremely hard to give me extensive opportunities in education and more generally in life. Thank you for encouraging me to pursue my academic interests and for providing emotional (and financial) support as I navigated the frequently uncertain waters of my education at MIT.

Contents

THIS PAGE INTENTIONALLY LEFT BLANK

List of Figures

List of Tables

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 1

Short Tandem Repeat (STR) Profiles: Natural and Synthetic Approaches

Eukaryotic genomes contain many repeated sequences. These vary in size, and are typically identified by the length of the core repeat unit and the number of adjacent repeat units. Alternatively, they can also be designated by the overall length of the repeat region and fall into three categories: long repeat units with several hundred to several thousand bases in the core repeat are referred to as satellite DNA; medium-length repeat units with 10-100 bases in the repeat region are called minisatellites; finally, DNA regions with repeat units that are 2-6 base pairs in length are termed microsatellites, or short tandem repeats (**Figure 1-1**).

Thousands of polymorphic microsatellites have been characterized in human DNA, and there may be more than a million microsatellite loci present [12]. These markers are scattered throughout the genome and occur roughly every 10,000 nucleotides, jointly comprising approximately 3% of the human genome [13]. STRs are of particular interest in human identification and have become a popular DNA marker because

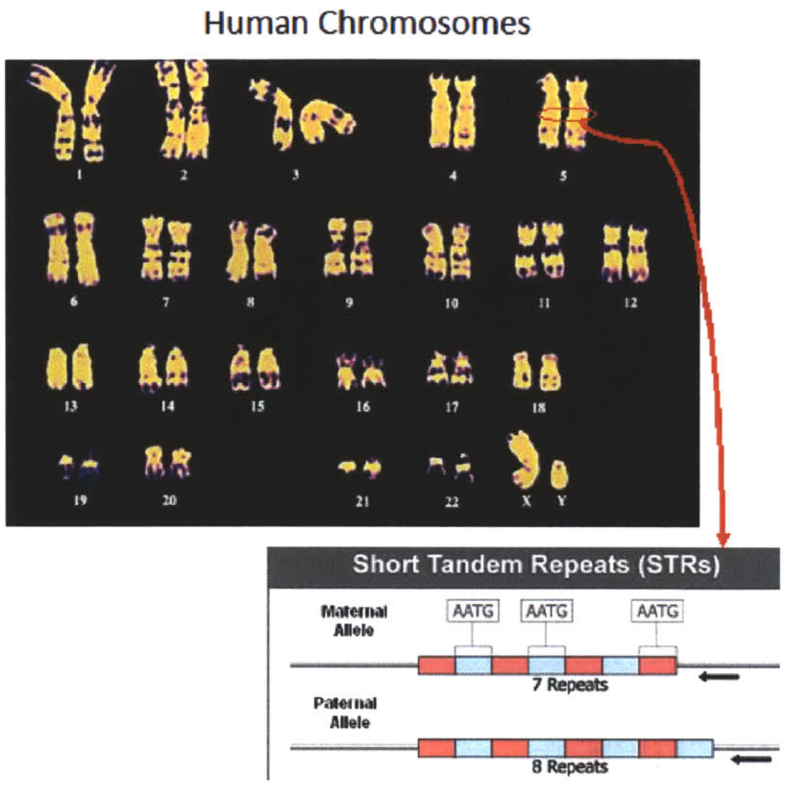


Figure 1-1: *Each person inherits two alleles at an STR locus, one from each parent. The two alleles can differ in the number of base pair repeats.*

they are short and can consequently be easily amplified by the polymerase chain reaction (PCR) with minimal problems caused by differential amplification. Small product sizes are also compatible with degraded DNA, and PCR enables recovery of information from small amounts of material. Additionally, the number of repeats in STR markers is highly variable among individuals, making combinations of STR alleles particularly effective for human identification.

The thirteen loci included in the National DNA Index System (NDIS) are of particular interest for forensics and human identification. NDIS is a DNA database funded by the United States Federal Bureau of Investigation (FBI) that stores DNA profiles created by federal, state, and local crime laboratories in the United States. The associated software, the Combined DNA Index System (CODIS), provides the ability to search the database to assist in the identification of criminal suspects. The 13 loci in the NDIS database provide the bulk of the loci analyzed in this work.

These are CSF1PO, FGA, TH01, TPOX, VWA, D3S1358, D5S818, D7S820, D8S1179, D13S317, D16S539, D18S51, and D21S11. For the full set of 13 loci, the probability of a random match in the profiles of two unrelated individuals is less than one in a trillion [20]. By generating STR profiles that also include Amelogenin (used to determine gender) and two additional loci specific to individual multiplex PCR kits, the random match probability can be further reduced. By October 2008, NDIS had grown to include over 241,685 forensic profiles and 6,384,379 offender profiles [3].

Many techniques exist to generate synthetic STR profiles, such as bacterial cloning and whole genome amplification (WGA). These are summarized in **Figures 1-2** and **1-3** respectively. These and other techniques can be performed using equipment commonly available in Biosafety Level 1 (BL1)[1] and Biosafety Level 2 (BL2)[2] labs. This project focuses primarily on whole genome amplification, with a brief extension to bacterial cloning. Qiagen, the manufacture of a commercials WGA kit, claims that this process is unbiased and that a WGA profile should be indistinguishable from a natural one. However, frequently this is not the case. Quantifying the Qiagen amplified DNA with the Identifiler kit, for example, revealed that WGA underestimates the amount of DNA present.

Currently, bisulfite sequencing is the state-of-the art technique used to authenticate STR profiles. This technique involves performing methylation analysis of the profile in question, based on the concept that human DNA is methylated but bacterial DNA is not [25]. This is effective because most methods to produce artificial DNA employ bacteria as a tool to amplify desired sequences. However, bisulfite sequencing leads to 90% degradation of the DNA due to the need for long incubation times, high temperatures, and elevated bisulfite concentrations. Furthermore, this technique is

[1]This level is suitable for work involving well-characterized agents not known to consistently cause disease in healthy adult humans, and of minimal potential hazard to laboratory personnel and the environment [2].

[2]This level is similar to Biosafety Level 1 and is suitable for work involving agents of moderate potential hazard to personnel and the environment. It includes various bacteria and viruses that cause only mild disease to humans, or are difficult to contract via aerosol in a lab setting [2].

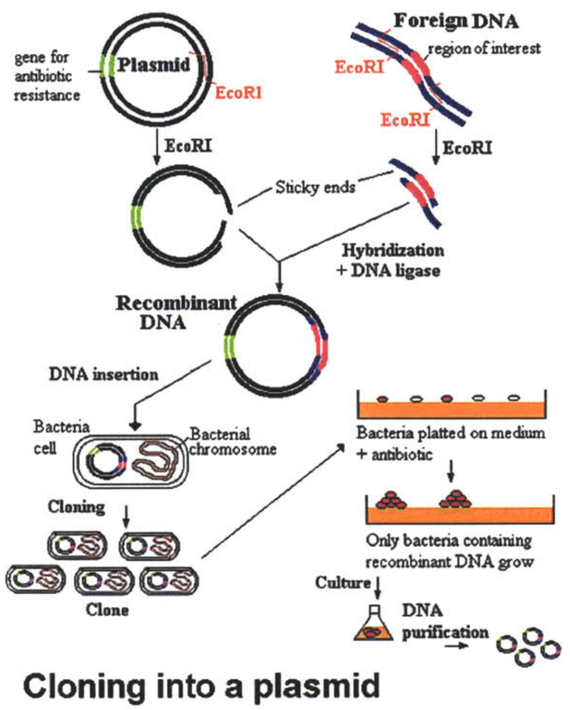


Figure 1-2: *Bacterial cloning summary. For purposes of this project, the pink strands of "foreign DNA" refer to DNA segments that contain the CODIS loci and Amelogenin [24].*

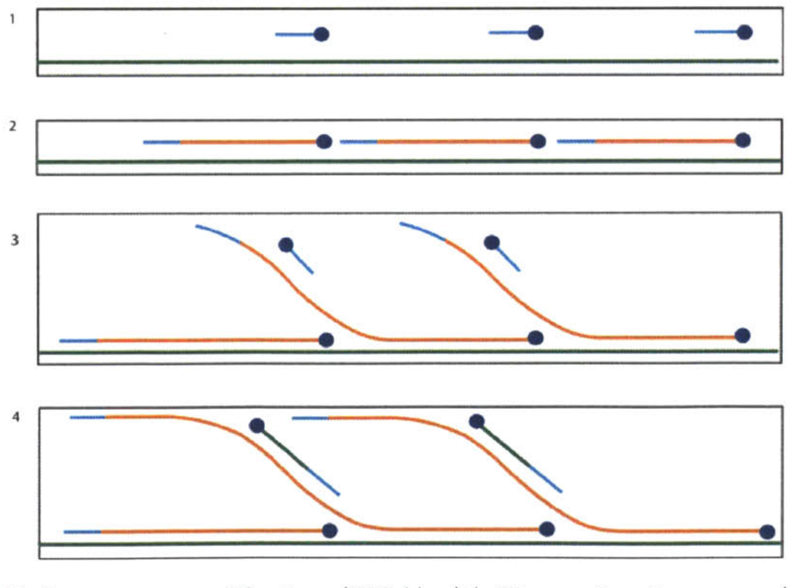


Figure 1-3: *Whole genome amplification (WGA). (1) The random hexamers (represented by a blue line) bind to the denatured DNA (represented by a green line). (2) The DNA polymerase (represented by a blue circle) extends the primers until it reaches newly synthesized double-stranded DNA (represented by an orange line). (3) The enzyme proceeds to displace the strand and continues the polymerization, while primers bind to the newly synthesized DNA. (4) Polymerization starts on the new strands, forming a hyper-branched structure [11].*

expensive and time-consuming.

The shortcomings of bisulfite sequencing suggest the usefulness of a software-based approach based on machine learning models trained on low-cost, readily available STR profiles. Thus, machine learning algorithms provide a valuable tool to detect the bias introduced by WGA and other means of STR profile generation. Although the conclusions inferred by these algorithms should not serve as a definitive test, they provide a useful trigger to run more conclusive and costly tests.

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 2

Data Analysis

2.1 Source STR Profile Availability

Genetic data was obtained in accordance with COUHES protocols from buccal swabs of volunteer donors. Since Identifiler kit data for WGA and natural samples was most readily available, the majority of the analysis was performed on the datasets in the "Identifiler Test Kit" column of **Table 2.1**. The low availability of data for the

	Identifiler Test Kit	**PowerPlex 16 Test Kit**
Natural **40/40 Unique Samples**	1 Sample, 4 Replicates 16 Samples, 0 Replicates 23 Samples, 1 Replicate	10 Samples, 1 Replicate
WGA **58/5 Unique Samples**	1 Sample, 4 Replicates 25 Samples, 0 Replicates 35 Samples, 1 Replicate	5 Samples, 1 Replicate
Bacterial (Cloned)		1 Sample, 16 Replicates
Bacterial(Synthetic)		1 Sample, 16 Replicates

Table 2.1: *Data availability by profile type and test kit.*

bacterial samples in the PowerPlex kit (one unique bacterial clone and one unique sample synthesized from scratch) suggests that machine learning algorithms trained on the bacterial PowerPlex samples should be verified on larger datasets. For many of the donors in the study, several STR profiles were obtained. Thus, "1 sample, 4 replicates" means that one unique donor's profile, plus four additional profiles from the same donor, were obtained for purposes of analysis. See **Chapter 4** for a more detailed discussion of replicate samples.

2.2 Profile Acquisition

STR profiles were obtained via the multiplexed polymerase chain reaction (PCR), a rapid way of amplifying specific DNA sequences (**Figure 2-1**). PCR was performed by adding the DNA to be amplified to a solution containing short tandem primers, the four nucleotides, and DNA polymerase. Three steps were then performed iteratively until a sufficient quantity of the desired sequence has been generated: (1)the DNA was denatured at 94 – 96 °C. (2) annealed to primers at 65 °C(3) elongated at 72 °C. Using this process, it is possible to obtain billion-fold amplification (32 cycles of PCR) in one hour [19]. The 13 CODIS loci, the amelogenin sex-typing marker, and two additional STR loci were co-amplified in a single reaction using existing commercial primer sets.

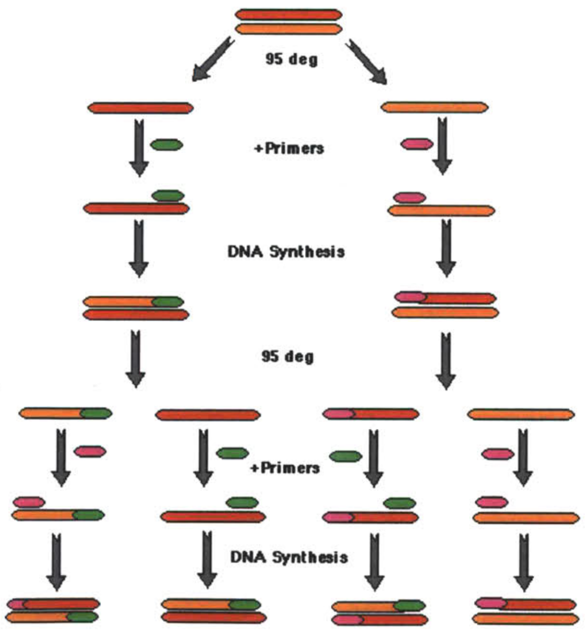


Figure 2-1: *Two cycles of PCR are illustrated. The red and orange lines represent the DNA template to which primers (pink and green ovals) anneal. [19].*

The samples were then subjected to parallel analysis via capillary electrophoresis using an Applied Biosystems 3130 genetic analyzer (ABI3130) [41]. The ABI3130 used amplicon sizing to identify individual alleles. Based on the analysis kit used, different fluorescent dyes were attached to PCR primers that were incorporated into the amplified target region of the source DNA. Amplified STR alleles were represented by peaks in an electropherogram.

One or more allelic ladders were included in each batch of STR profiles analyzed with the ABI3130. An allelic ladder is an artificial mixture of the common alleles present in the human population for a particular STR marker [9]. Such allelic ladders serve as a standard for each STR locus (see **Figure 2-2**). These ladders are generally created with the same primers as test samples and provide a reference DNA size for each allele. They are used to adjust for different sizing measurements obtained from different runs of the ABI3130 instrument. Ladders are constructed by combining locus-specific PCR products from multiple individuals in a population. The samples are then co-amplified to produce an artificial sample containing the common alleles for the STR marker. The allele quantities are balanced by adjusting the input amount of each component so that the alleles are fairly equally represented in the ladder. Internal standards labelled with a different color from the STR alleles were used to perform the DNA size determinations and subsequent correlation with an allelic ladder to obtain an STR genotype.

The rapid processing and multiplex capabilities of the ABI3130 genetic analyzer encourage the development of machine learning techniques to authenticate the output of this technique. For example, both 96-well and 384-well plates of samples can be processed with the ABI 3130. With each run taking 45-60 minutes, a 96-well plate can be analyzed in approximately 5-6 hours [9]. These capabilities make multiplex PCR analysis of STR profiles more attractive to forensics and biometrics laboratories in comparison to more expensive and time-consuming approaches such as methylation analysis.

The Identifiler and PowerPlex kits were chosen for use with the ABI3130 instru-

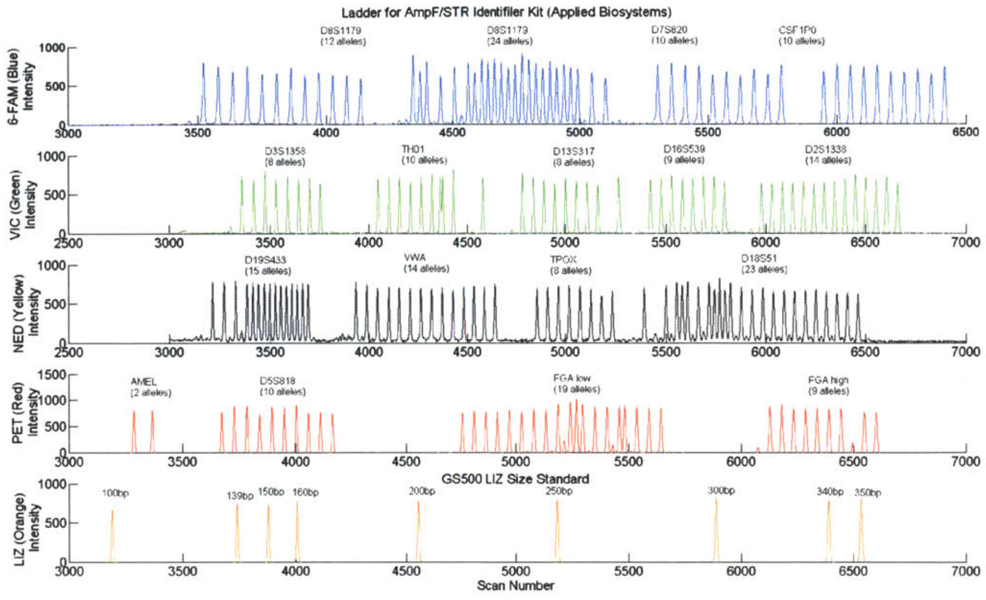


Figure 2-2: *AmpFLSTR Identifiler allelic ladder (Applied Biosystems). A total of 205 alleles are included in this ladder used for genotyping a multiplex PCR reaction involving 15 STR loci and the amelogenin sex-typing test.*

Name	Source	Release Date	STR Loci included
AmpF/STR Identifiler	Applied Biosys-tems	July 2001	CSF1PO, FGA, TPOX TH01, VWA, D3S1358, D5S818, D7S820, D8S1179, D13S317, D16S539, D18S51, D21S11, D2S1338, D19S433, amelogenin
PowerPlex 16	Promega	May 2000	CSF1P0, FGA, TPOX, TH01, VWA, D3S1358, D5S818, D7S820, D8S1179, D13S317, D16S539, D18S5‘, D21S11, Penta D, Penta E, amelogenin

Table 2.2: *Commercially available STR multiplexes used to analyze STR profiles.*

ment because of their widespread usage in the forensics and biometrics communities (**Table 2.2**). Both kits amplify the 13 CODIS loci/amelogenin and are able to identify repeat lengths within similar size ranges. The primary differences between them lie in the additional loci analyzed (Penta E and Penta D for PowerPlex 16; D2S1338 and D19S433 for Identifiler). Additionally, the two kits differ in their dye-labelling strategies: the PowerPlex 16 kit uses four dyes (three channels and a size standard), while the Identifiler kit uses five dyes (four channels and a size standard). A third difference is in the size standards used: ILS600 CXR for PowerPlex 16 and GS500 LIX for Identifiler.

2.3 Extracting Electropherogram Information from .fsa Files

The ABI3130 Genetic Analyzer stores STR profile data in the .fsa format. A Python module was developed to convert the data to human-readable format, extract signals for further analysis, and obtain relevant information about the electrophoresis process. Code was written to identify the fluorescence value of each channel in the electropherogram as a function of scan number (a measure of time). Timestamps for each channel were used to identify the associated ladder. Multiple ladders were included with each PCR sample to account for variations in experimental conditions (i.e. changes in temperature, photo-bleaching) that could effect allele resolutions and introduce artifacts into the STR profile. Consequently, each channel within a profile was analyzed with respect to the nearest ladder. This custom code makes use of the ABIFReader Python module published by Interactive Biosoftware [5, 41].

The signals obtained from the files were then processed in MATLAB to identify allele values for each locus, locate off-ladder alleles, and extract feature values for each sample.Peak alignment techniques were used to overlay the size standard in the bottom channel of an electropherogram over the size standard of the associated allelic ladder. This was done to align the allele peaks in the ladder with corresponding allele peaks in the individual sample and identify the allele value for each locus. An example of a resulting STR profile is presented in **Figure 2-3**. A series of steps was then performed to identify feature values for the sample. The individual features of interest are summarized in **Section 2.4**, and **Figure 2-4** demonstrates an example of an STR profile with annotated alleles and feature values.

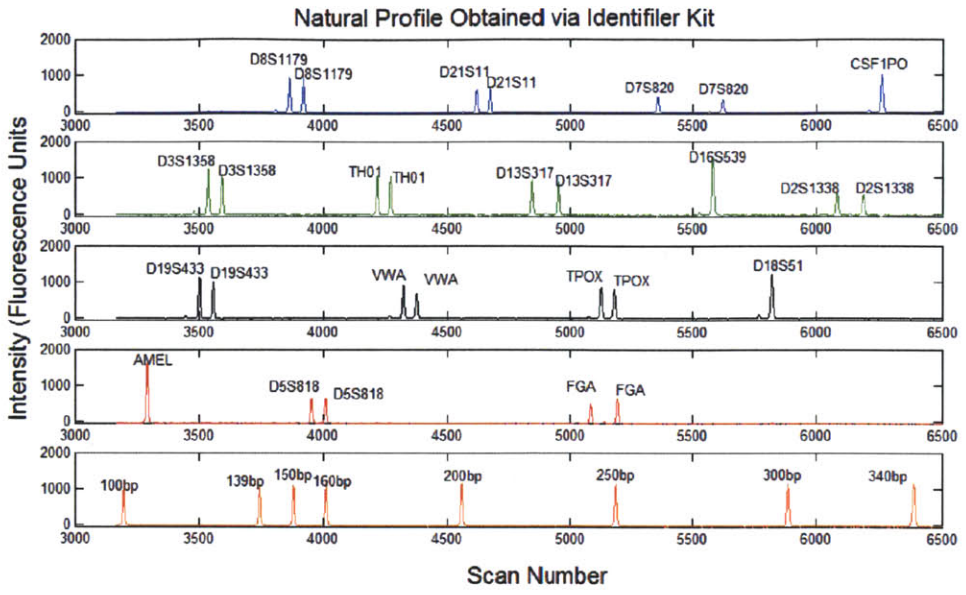


Figure 2-3: *A natural profile obtained via multiplexed PCR with the Identifiler kit. The alleles of interest are indicated by fluorescence peaks. The size standard, in the bottom channel, was aligned with the size standard from a ladder. The aligned sample was then compared with the ladder to identify allele peaks based on their size.*

2.4 Feature Identification

Having obtained annotated STR profile from the MATLAB processing pipeline, the next step was to identify features useful for distinguishing between natural and artificial profiles. These features in combination served as the raw material on which to train a suite of machine learning algorithms to perform authentication. Prior research and examination of the data led to the identification of eleven features. Many of these were based on common biological artifacts of STR markers, such as stutter products, non-template nucleotide addition, microvariants, null alleles, and mutations. The full set of 11 analyzed features included:

- Intra-locus imbalance: Ratio of peak heights between the alleles in a single heterozygous locus (**Figure 2-5**). Intra-locus peak height ratios were calculated for a given locus by dividing the peak height of an allele with a lower fluorescence intensity (shorter peak) by the peak height of an allele with a higher intensity (taller peaks). Theoretically, two alleles for an individual who is heterozygous

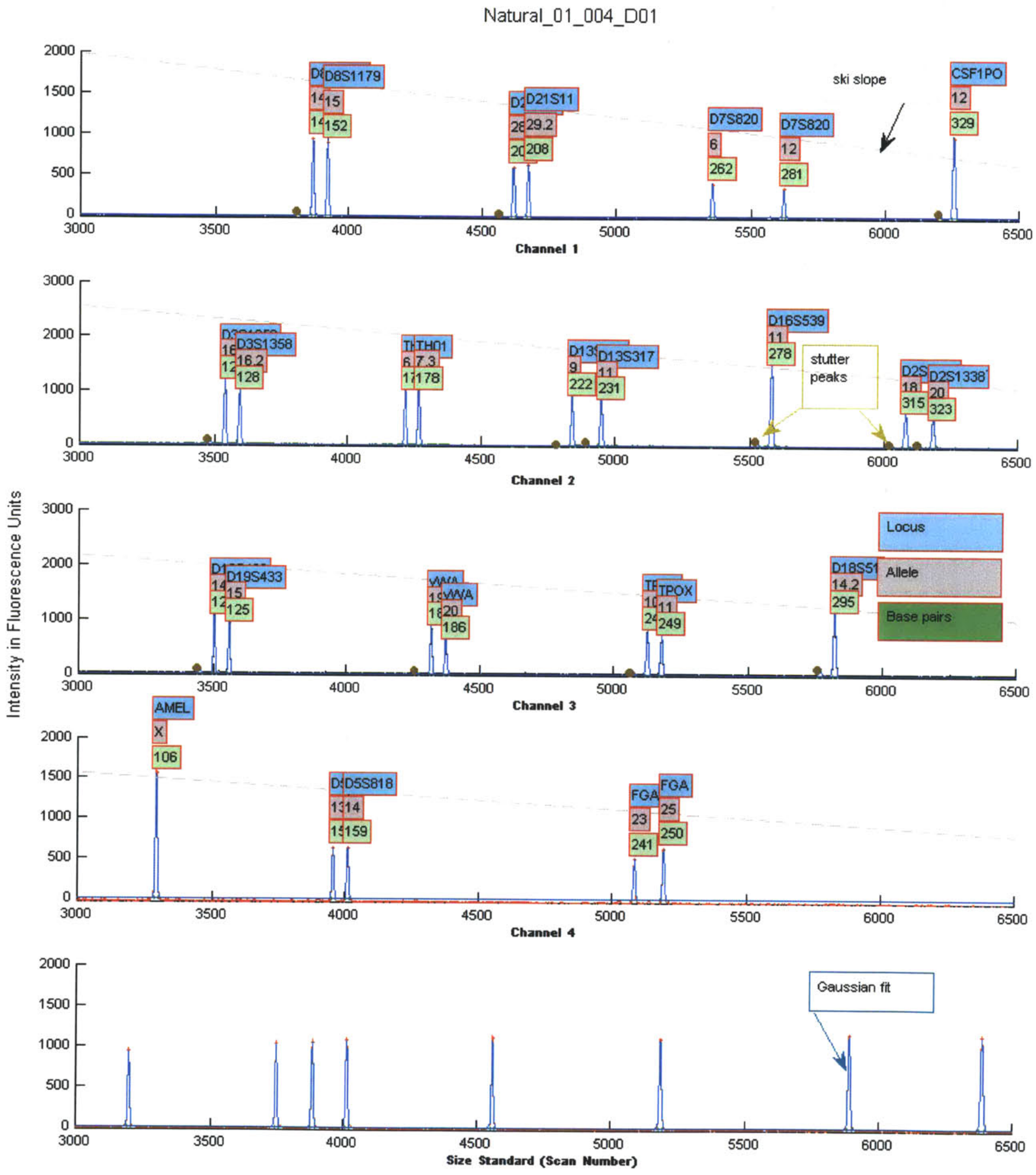


Figure 2-4: *Sample natural STR profile from Identifiler kit with 4 Channels, 16 Loci, and a size standard. Boxed values represent called alleles.*

at a single locus should be present in equal amounts in the genome, amplify equally, and have peak heights are that are approximately equal, with a peak height ratio near 1. In practice, intra-locus imbalance may occur if the DNA source is inhibited, degraded, preferentially amplified, or subject to unequal sampling of true alleles [42]. The latter two conditions are more likely to occur in a laboratory-synthesized profile, suggesting the use of this feature to distinguish between natural and synthetic samples.

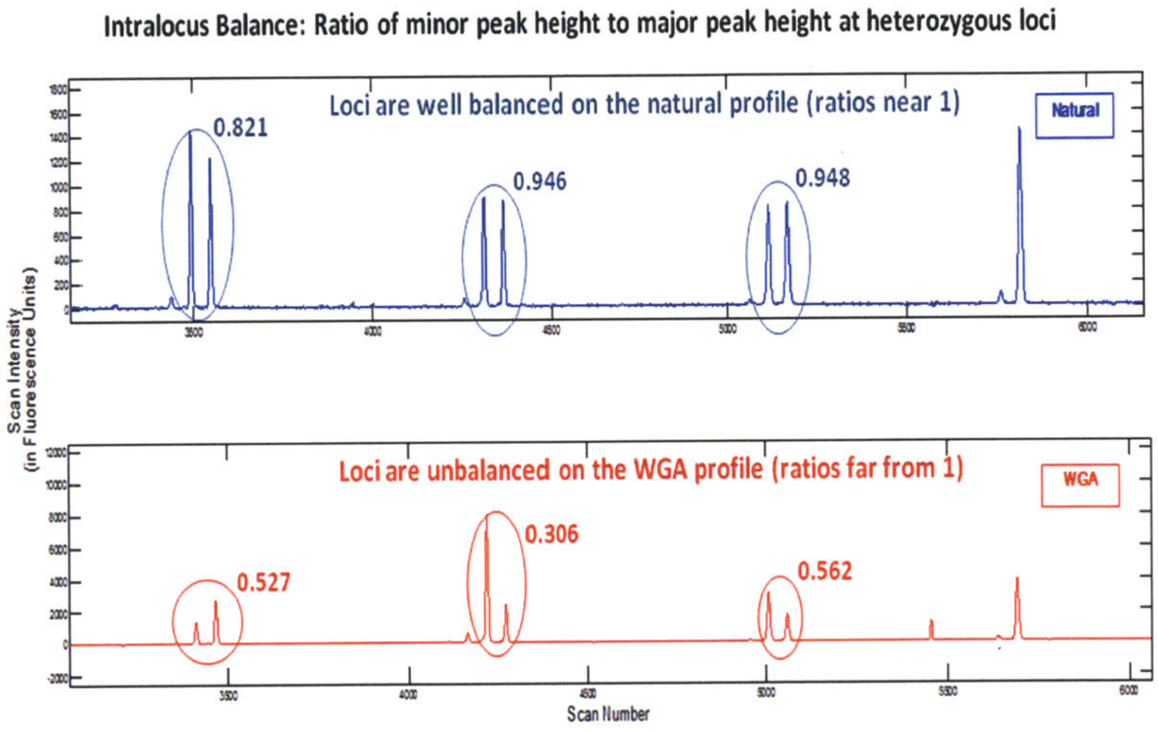


Figure 2-5: *Intra-locus imbalance.*

- Inter-locus imbalance: Ratio of peak heights among adjacent loci in an electropherogram. This feature was calculated in two ways. The first approach involved computing the ratio of tallest locus to shortest locus in a channel. The second approach involved finding the mean squared error between each individual locus and the mean intensity for the channel.

- Fluorescence intensity: Measured by peak height (see **Figure 2-6**).
- Frequency and position of off-ladder alleles: STR microvariants are rare alleles that result from point mutations or insertion/deletions of a block smaller than the locus repeat block size. An off-ladder outside bin allele is a microvariant that does not correspond to any of the standard STR loci included in a test kit (Identifiler/PowerPlex 16); an off-ladder inside bin allele corresponds to a standard locus but does not match any of the standard alleles for that locus [28]. Natural and artificial samples were compared for the presence of both kinds of microvariants. See **Figure 2-6** for an example of a drop-in allele in a sample profile. The feature analysis techniques used to annotated profiles were able to detect allele drop-in, but not allele drop-out, which causes a heterozygous sample to look like a homozygous sample. The feature identification protocol could be extended to accommodate allele dropout by obtaining STR profiles with both the Identifiler and PowerPlex 16 test kits and comparing the heterozygosity of each peak.

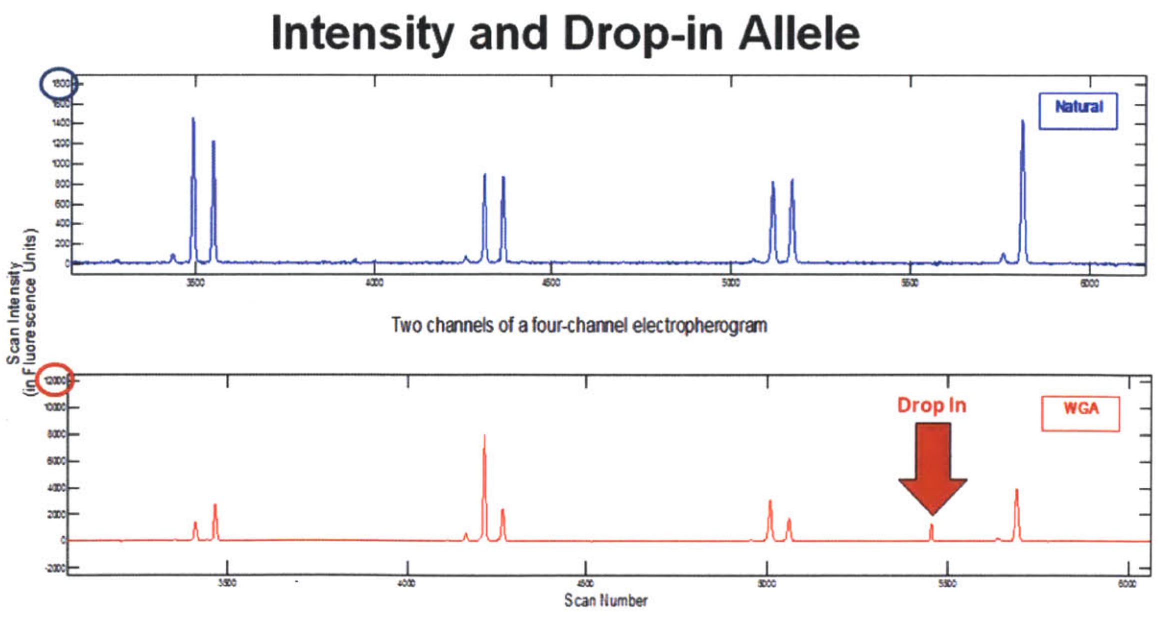


Figure 2-6: *Differences in intensity between natural and WGA profiles are noted with the blue and red circles. A drop-in allele is present in the WGA (lower) channel.*

- Frequency and position of stutter peaks: Stutter product peaks are small peaks that differ in size from an allele peak by one or two repeat units. Stutter products are caused by slip-strand mispairing of the DNA polymerase during replication. Insertion, caused by slippage of the copying strand, leads to a stutter product one repeat unit longer than the main allele. Deletion, caused by slippage of the copied strand, causes a stutter product one repeat unit shorter than the main allele. Since different polymerases are used in natural and synthetic DNA replication, and use of faster polymerase results in fewer stutter products, it is possible that the frequency and position of stutter peaks may differ between natural and artificial samples. Typically, a stutter product is 5-15% of the height of the adjacent allele peak [28]. In **Figure 2-7**, stutter peaks are denoted by small golden dots located near the baseline.

- Ski slope: Biological samples become degraded when exposed to adverse environmental conditions. Since degradation breaks the DNA at random, larger amplified regions are affected first and the height of the peaks in an electropherogram decreases from left to right (**Figure 2-7**). Since artificial DNA samples are less likely to be subject to adverse environmental conditions, it is possible that the ski slope may be a distinguishing feature between the two classes [16].

- Presence of pull-up: Pull-up occurs when the analysis software is unable to discriminate between the different dye colors used for sequencing. If matrix color deconvolution in the fluorescence analysis process does not work properly, a color may bleed from one spectral channel into another, usually because of off-scale peaks [1].

- Signal to noise ratio: The peaks were interpreted as the signal; any non-relevant disturbances in the baseline were interpreted as noise. **Figure 2-8** illustrates the higher noise observedfor some WGA profiles relative to natural profiles.

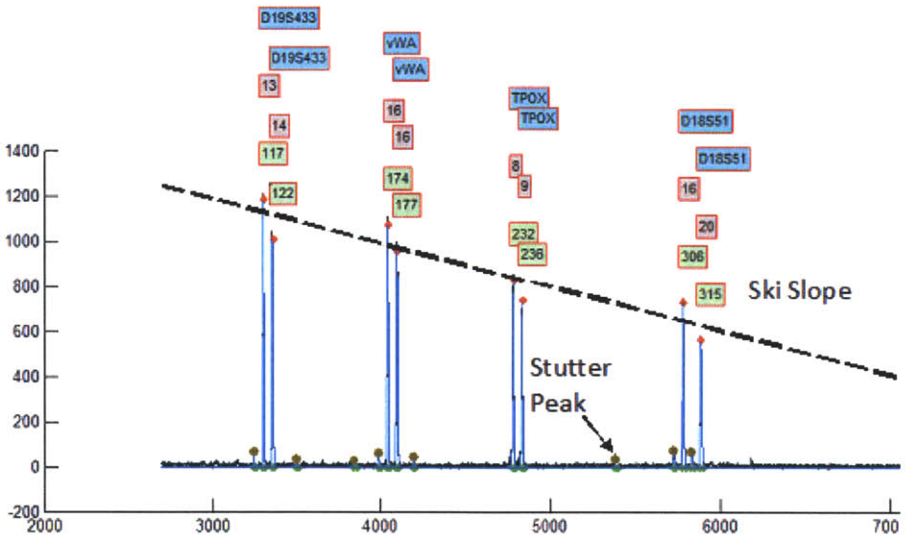


Figure 2-7: *Ski slope and stutter peaks.*

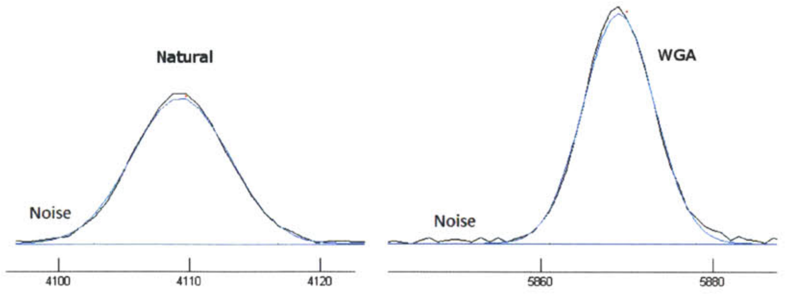


Figure 2-8: *Peak shape and signal-to-noise ratio.*

- Peak shape/area: In natural DNA, peak shape closely approximates a Gaussian curve. Additionally, the peak distribution has a predictable pattern based on the extraction method used and the extent of DNA degradation. Variations in peak shape may be observed due to phenomena such as non-template addition [1]. In the process of adenylation, the Taq polymerase adds an extra adenine nucleotide to the end of a PCR product. Depending on the 5' end of the reverse primer, a guanine can be added to the end of a primer to promote non-template addition. Excess amounts of DNA template in a PCR reaction can result in incomplete adenylation due to insufficient quantities of polymerase, which can be observed on an STR profile as a split peak (**Figure 2-9**). Since this phenomenon depends

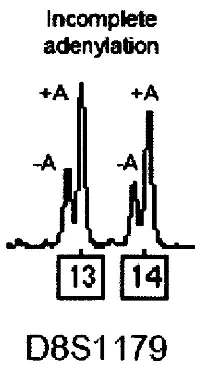


Figure 2-9: *Split peaks due to incomplete adenylation do not follow the expected Gaussian shape [5].*

on both the amount of DNA present and the polymerase used to replicate the DNA, it is possible that the frequency of incomplete adenylation is different for WGA and natural profiles.

2.5 Feature Granularity/Establishing Patterns for Classification

The full set of feature values was calculated for each profile with the goal of combining many weak trends to build an accurate profile classifier. The optimal feature granularity was determined experimentally. In one approach, individual peaks served as the patterns for classifications, and feature values were calculated for every peak. For example, the intra-locus imbalance for a peak was found by calculating the ratio of the two peak heights in a heterozygous allele. This ratio was assigned as the intra-locus imbalance for both peaks in the locus. For homozygotes, the intra-locus imbalance was set to one for the single peak in the locus. When each peak was treated as a separate classification pattern with a full set of feature values, the data shortage problem was avoided, since each sample had a high number of peaks. However, some features, such as ski slope and inter-locus imbalance, are not defined for individual peaks, but rather for combinations of peaks. Multiple peaks must thus be assigned the same feature value. For example, all peaks in a channel would be assigned the same value for the ski slope feature. This approach functions like a smoothing filter and leads to loss of information about the data.

Consequently, to avoid bias, channels were used as patterns for classification rather than individual peaks. **Table 2.3** presents the formulas used to calculate each feature at the channel level. After features were calculated at the channel level, profile statistics were obtained. For each profile, the minimum, maximum, mean, and range of the feature values across the individual channels were calculated. For example, computing the ski slope for the channels in an Identifiler profile results in four values, one for each channel. The minimum of these four is defined as the profile minimum, the maximum among the four is the profile maximum, the difference between the maximum and minimum is defined as the range of the ski slope feature for the profile, and the mean of the four values is the profile mean. Empirically, examining features at the profile level through the min, max, mean, and range statistics led to slightly reduced error rates in the baseline classifier performance (mean reduction 2% across all the classifiers examined). Consequently, all subsequent analysis was performed at the profile level. That is, each sample had 44 unique features – a profile minimum, maximum, mean, and range for each of the 11 features listed in **Table 2.3**.

2.6 High Level Feature Analysis

Once features had been identified and computed for each of the natural, WGA, and bacterial profiles, a high level analysis was performed of the resulting distributions. The scatter plots in **Appendix A** and **Appendix E** show the feature values for the Identifiler and PowerPlex datasets, respectively. The blue dots represent natural profiles, while the red dots represent WGA. In each subplot there are 87 red dots and 37 blue dots, indicative of the 87 WGA and 37 natural profiles that were analyzed with the Identifiler kit. The green dots in **Appendix E** represent bacterial profiles (bacterial clones and synthetic samples were analyzed jointly). The process used to generate the bacterial clones is discussed in **Chapter 1, Figure 1-2**. Although fea-

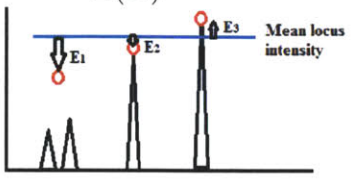
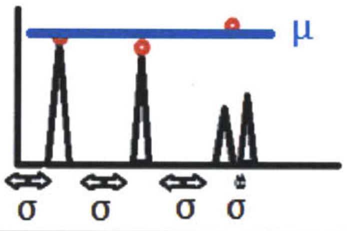
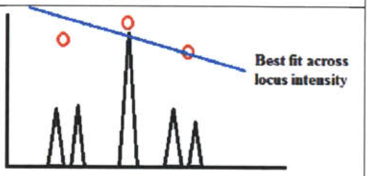
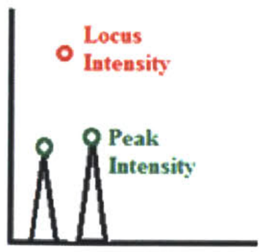
Feature	Value
inter-locus imbalance error	μ across locus intensity Σ (locus intensity - $\mu)^2$ $\Sigma(E)^2$ 
inter-channel intensity	(Σchannel intensity) : (Σprofile intensity)
signal-to-noise ratio	$\frac{\mu = locus\ intensity\ average}{\sigma = non-peak\ variance}$ 
heterozyote intra-locus imbalance	* geometric mean over all heterozygous loci * 1 if all loci in channel are homozygous
inter-locus imbalance ratio	ratio of locus intensity (Max:Min)
peak width	(Σ channel peak widths) : (Σ profile peak widths)
Gaussian error	$\Sigma(\text{error})^2$ over all peaks in channel
ski slope	
off-ladder outside bin	count
off-ladder inside bin	count
stutter	count

Table 2.3: *Identification of features at the channel level.*

Notes/Clarification

- All channel features are propagated to the profiles as Min, Max, Range, Average
- Locus intensity = Σ (Peak Intensity at Locus)



ture values were computed for the bacterial dataset, only one bacterial clone and one bacterial synthetic profile was available for study. Consequently, additional data must be gathered to draw solid conclusions about feature selection for bacterial profiles and to train high-performing classifiers (**Table 2.1**).

Figure A-2 provides an illustrative example of differences in feature values between natural and WGA profiles. The figure illustrates intra-locus imbalance values for each profile. The top row illustrates the intra-locus imbalance value of the four individual channels, while the bottom subplot shows the minimum, maximum, range, and mean of the intra-locus imbalance for the profile. In each of the eight subplots, the WGA datapoints are more spread out than the natural. The blue natural feature values cluster near 0.9, but the red WGA values have no significant clusters. Rather, they are spread fairly uniformly through the range of valid intra-locus imbalance values, from zero to one. This trend is highly evident at the channel level (top row). It is weaker at the profile level: the range and mean values for both WGA and natural profiles are spread throughout the [0-1] range. However, the natural data is more clustered than the WGA data for the profile min and max.

This observation can be generalized for the majority of the features examined. For nearly each feature plot in **Appendix A**, the WGA profiles take on a higher range of values, with a correspondingly higher standard deviation. This phenomenon influences classifier performance and is discussed in **Chapter 6**. The higher spread of feature values for WGA profiles results in a high incidence of false positives for the WGA class, that is, natural profiles misclassified as WGA. Since WGA profiles can take on a greater variety of feature values, classifiers are more likely to mark a profile as WGA than natural. Thus, careful feature selection is necessary to discover the subset of features for each classifier to minimize both false positives and false negatives.

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 3

Machine Learning Algorithms for STR Profile Classification

Once a set of features had been identified to differentiate between natural and synthetic STR profiles, as described in **Chapter 2**, these features were used as inputs to a set of machine learning algorithms. The algorithms operate on the principle that a learner can use example data to identify relationships between observed feature vectors. A major focus of machine learning research is to automatically learn to recognize complex patterns and make intelligent decisions based on data; the difficulty lies in the fact that the set of all possible behaviors given all possible inputs is too large to be covered by the set of observed examples (training data). Consequently, the learner must generalize from the given examples, so as to be able to produce a useful output in new cases (test data).

Machine learning classifiers are trained by partitioning data into two groups: training and test. If sufficient data is available, a third evaluation group may also be created. A classification algorithm uses the training data set to learn the difference between the classes. The evaluation set, if present, is used to evaluate classifier performance and to determine whether further training is necessary. Ultimately, the classification algorithm computes a set of parameters that form a discriminant function between the two classes. Given a new profile, the algorithm applies this function to classify it.

The pattern classification approach is particularly relevant to STR profile authentication for several reasons. First, this approach works well for noisy, complex, or unknown processes, and there is a high level of noise and complexity in the acquired STR profile data. Furthermore, pattern classification is particularly appealing when features can be measured and training data is available, both of which are conditions that hold for the STR profile sample set [38].

3.1 Classifier Training: Developing Accurate Discriminant Functions Between Classes

A classifier consists of a discriminant function that is constructed during the training phase. This function consists of a linear combination of feature values multiplied by weights w (**Figure 3-1**). It has the following mathematical form:

$$y(\mathbf{x}) = f(\sum_{i=0}^{D} x_i * w_i)$$

- $\mathbf{x}$ refers to the input feature vector. In **Chapter 2**, 44 distinct features were identified, so the feature vector for each sample STR profile consists of 44 values.

- $\mathbf{w}$ is the vector of coefficients that produce the desired characteristics in the discriminant function. The focus of the training phase is to optimize this weight vector so that the function y takes on dissimilar values for input samples that belong to different classes and similar values for inputs that belong to the same class.

- D is the number of features used to train the classifier. It is equal to 44 for the STR data.

- f refers to the function that governs the behavior of the machine learning algorithm. Step functions, linear separators, and sigmoids area all commonly used by classification algorithms. Binary trees, support vector machines, and Gaussian mixture models respectively implement these functions. Examples are illustrated in **Figure 3-1**.

The discriminant function takes a feature vector as an input and projects this vector into a higher-dimensional feature space. This feature space is partitioned by decision boundaries, and an input feature vector is classified based on the location of the projection relative to these boundaries [40]. For example, the right half of **Figure 3-1** illustrates the projection of two input feature vectors (A and B) onto a two-dimensional space defined by features x_1 and x_2. Since one vector is projected above the decision boundary and the other one is projected below the boundary, the two vectors get assigned to different classes.

Training a classifier consists of performing discriminant analysis to find the optimal vector of weights w. At each iteration the current discriminant function is used to project the input data to a high-dimensional feature space and to classify them according to their position relative to a set of decision boundaries. The algorithm then identifies training samples that were misclassified. The weight vector w is updated to reduce the number of misclassifications in future iterations. The exact manner in which this vector is updated is algorithm-specific. When the number of misclassified training data samples drops below a pre-defined threshold, training is complete and the weight vector is fixed at its current value. The classifier is said to be trained.

3.2 N-Fold Cross Validation

Due to the small sample sizes for both the Identifiler and PowerPlex test kits, there was insufficient data to split the profiles into training, evaluation, and test sets. To deal with the scarcity of data, four-fold cross validation was performed to select

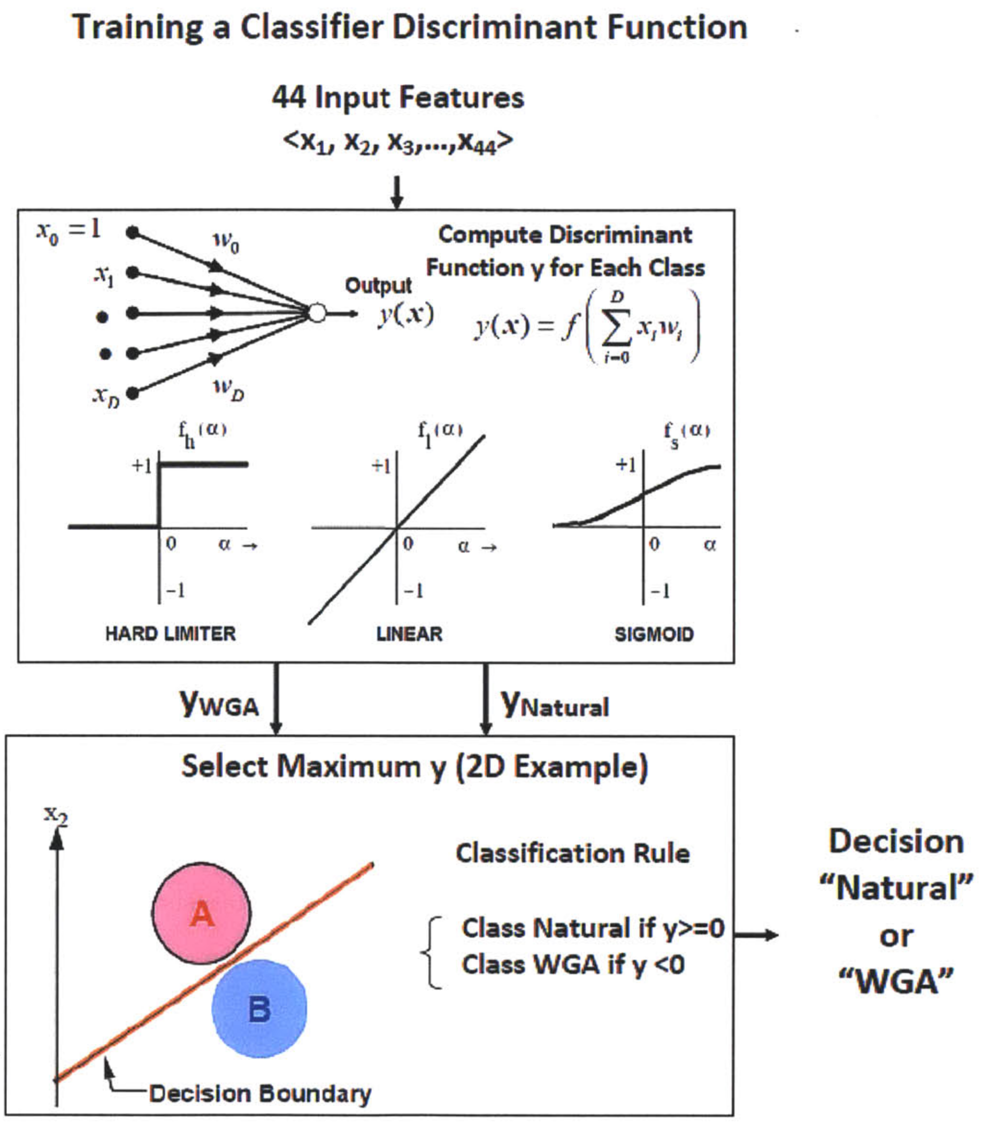


Figure 3-1: *The training phase of classifier development involves creating a discriminant function that forms decision boundaries between the natural and WGA classes in multi-dimensional space.*

features and tune parameters for individual classifiers [42]. The advantage of the cross validation technique is that all available samples were used as both test objects and training objects. To perform four-fold cross validation, samples were separated into four folds, and each was tested against a classifier trained on the data in the other three folds (**Figure 3-2**). The error rates from the individual test folds were summed to obtain the overall error rate of the classifier.

As demonstrated in **Figure 3-2**, classifier training and evaluation via four-fold cross validation can be summarized as follows:

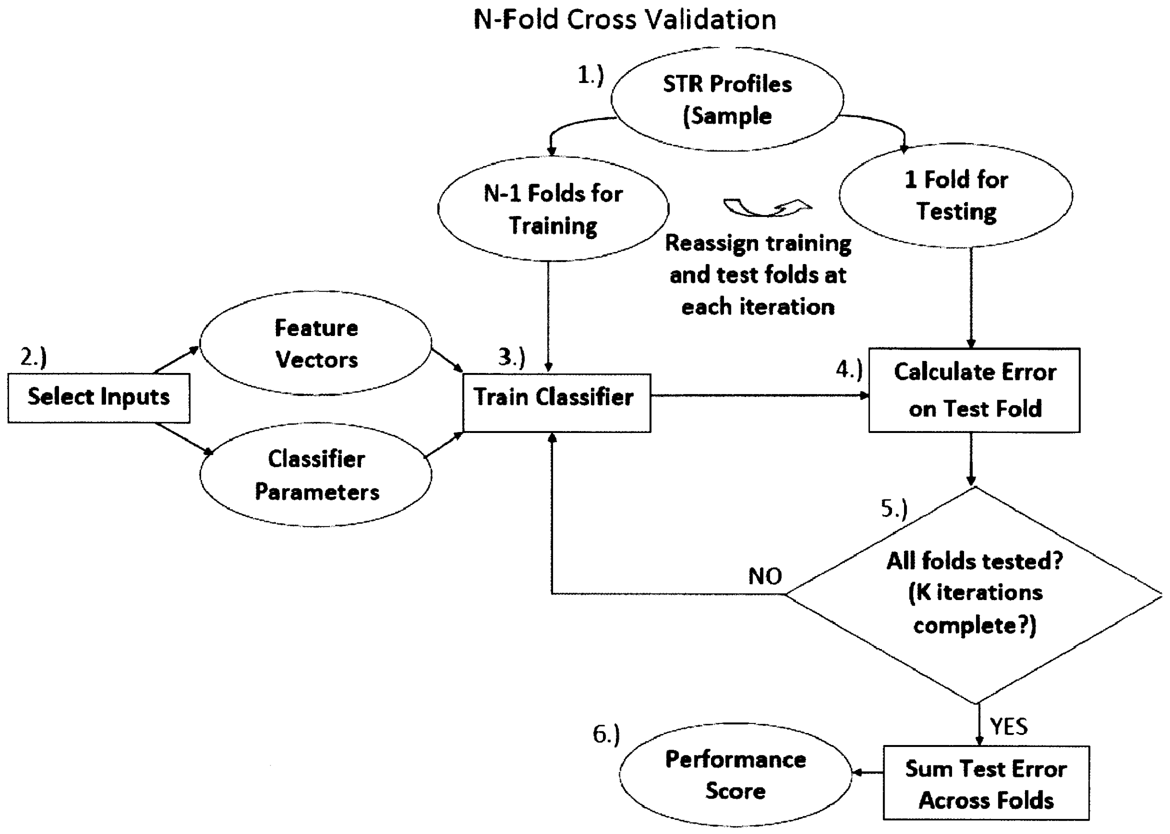


Figure 3-2: *N-fold cross validation was used to combat low data availability by using each profile for both training and testing [42]. Four folds were used (N=4).*

1. The sample STR profiles were randomly assigned to four folds. At each iteration of the validation algorithm, three of the folds were used to train a classifier and the fourth was used to test the classifier performance.

2. The inputs to the classifier were assigned. These consisted of training data (STR profiles), feature vectors calculated for the data, and classifier parameters, discussed in **Chapter 5**.

3. The classifier was then trained in the manner described above with the aim of developing a discriminant function between the natural and WGA classes.

4. The performance of the resulting classifier was evaluated by classifying the test data and calculating the number of misclassifications.

5. The training and test steps were repeated K times. A new fold was selected to serve as the test fold at every iteration. The remaining folds were used as training data.

6. The test error was summed across the folds to produce a final performance score for the classifier. This score is discussed in **Section 5.3**.

3.3 Classifier Taxonomy

A set of 16 supervised and semi-supervised machine learning techniques were used to classify sample profiles as natural or WGA. Additionally, four unsupervised clustering algorithms were used in conjunction with the semi-supervised approaches (**Table 3.1**).

- **Supervised learning** algorithms map labeled input data to desired output classes. The classifiers compute discriminant functions mapping the feature vectors for the input data to classes.

- **Unsupervised learning** algorithms cluster unlabeled input data into groups of similar samples based on the input feature vectors.

- **Semi-supervised algorithms** use combinations of labeled and unlabeled input data.

The classifiers were further grouped by their approaches to pattern classification. Some produced continuous outputs in the form of likelihoods or posterior probabilities. Others produced binary outputs in the form of nearest neighbor assessments or decision boundary calculations. These four approaches are summarized below and illustrated in **Figure 3-3**.

- **Posterior Probability** classifiers estimate the posterior class probabilities of input patterns. Given an input pattern X and two class options "natural"

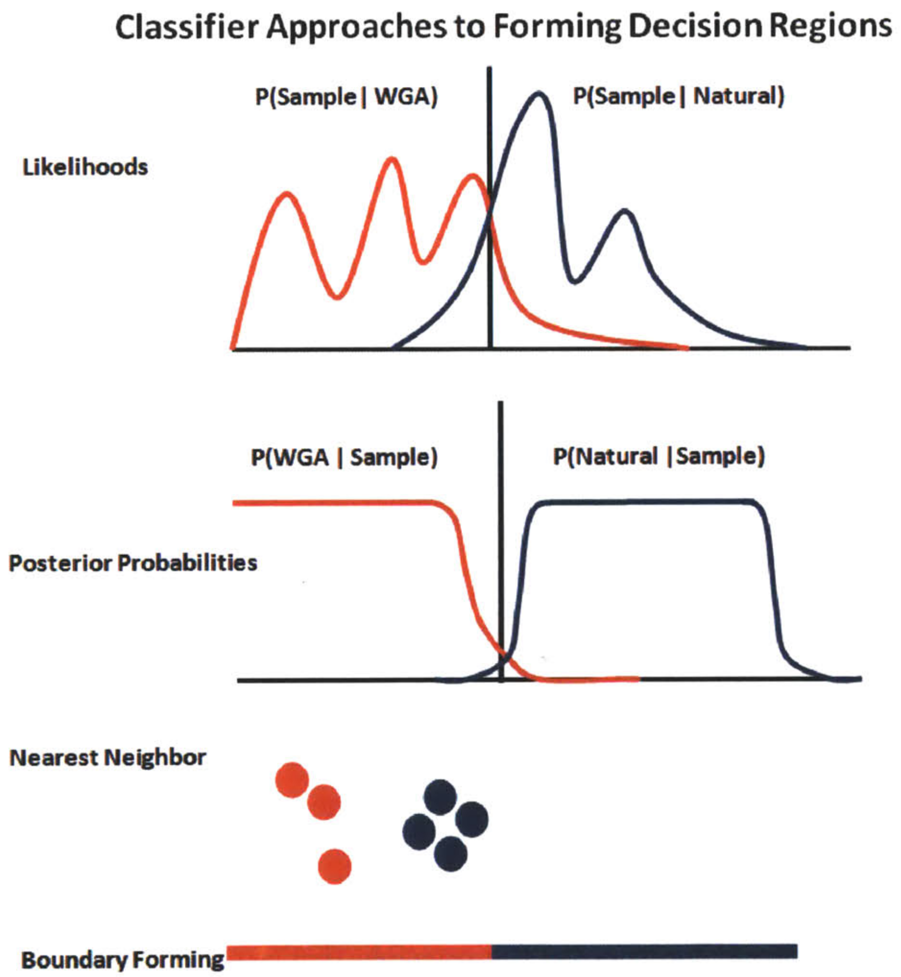


Figure 3-3: *Classifier approaches to forming decision regions [35].*

and "WGA", a posterior probability classifier estimates $p(natural|X)$[1] and $p(WGA|X)$. The classification decision is determined by the higher of these two probabilities. Many neural network classifiers, such as the MLP, IRBF, and RBF calculate posterior probabilities.

- **Likelihood** classifiers estimate a scaled probability density function, or likelihood, for each class. Given an input pattern X and two class options, "natural" and "WGA", a likelihood classifier estimates $p(X|natural)P(natural)$ and $p(X|WGA)p(WGA)$. In these expressions, $p(X|natural)$ and $p(X|WGA)$ are the likelihoods for the two classes, and $p(natural)$, $p(WGA)$ are the prior probabilities that a pattern belongs to either of these two classes. The classification decision is determined by multiplying the class with the highest likelihood by the prior probablity of the class. Consequently, classification decisions are continuous probability distributions that assign X to the natural or WGA class. The Gaussian, Gaussian mixture model, histogram, Naive Bayes, and Parzen window classifiers compute likelihoods.

- **Rule-based** classifiers partition the input space into binary decision regions using threshold logic nodes or rules. They can often be easily implemented in hardware applications. This category includes the support vector machine, the hypersphere classifier, and the binary tree classifier.

- **Nearest neighbor** classifiers work on the principle that a pattern is probably of the same class as those patterns nearest to it when feature vectors are projected into a high-dimensional space. Nearest neighbor classifiers store input patterns during the training phase and compute distances between them. The computation necessary for testing can be prohibitive for large databases. Most enhancements to the algorithm involve reducing the number of patterns

[1]$p(natural|X)$ refers to the probability that the classifier assigns a pattern to the natural class, given the particular set of feature values for that pattern.

stored and used for testing. Nearest neighbor classifiers are simple and easily understood, but do not produce continuous outputs for later analysis and do not generalize well where training and test data differ. Examples of nearest neighbor classifiers are the KNN, condensed KNN, and LVQ algorithms.

	Continuous Output		**Binary Output**	
	Maximum Likelihood Fit to Data	**Squared Error Fit to Posterior Probability**	**Nearest Neighbor**	**Rule Forming**
Supervised	Parzen Window	Multi-Layer Perceptron	Nearest Cluster	Support Vector Machine (SVM)
	Naive Bayes		K-Nearest Neighbors (KNN)	Hypersphere
	Histogram		Condensed K-Nearest Neighbors (CKNN)	Binary Tree
	Gaussian Linear Discriminant			
Combined Supervised / Unsupervised	Gaussian Mixture Model	Radial Basis Function (RBF)	Linear Vector Quantizer (LVQ)	Adaptive Resonance Theory Map (ARTmap)
		Incremental Radial Basis Function (IRBF)		
Unsupervised			Leader Clustering	
			K Means Clustering	
			Expectation Maximization Clustering	
			Random Clustering	

Table 3.1: *Machine learning algorithms used to train profile classifiers [7].*

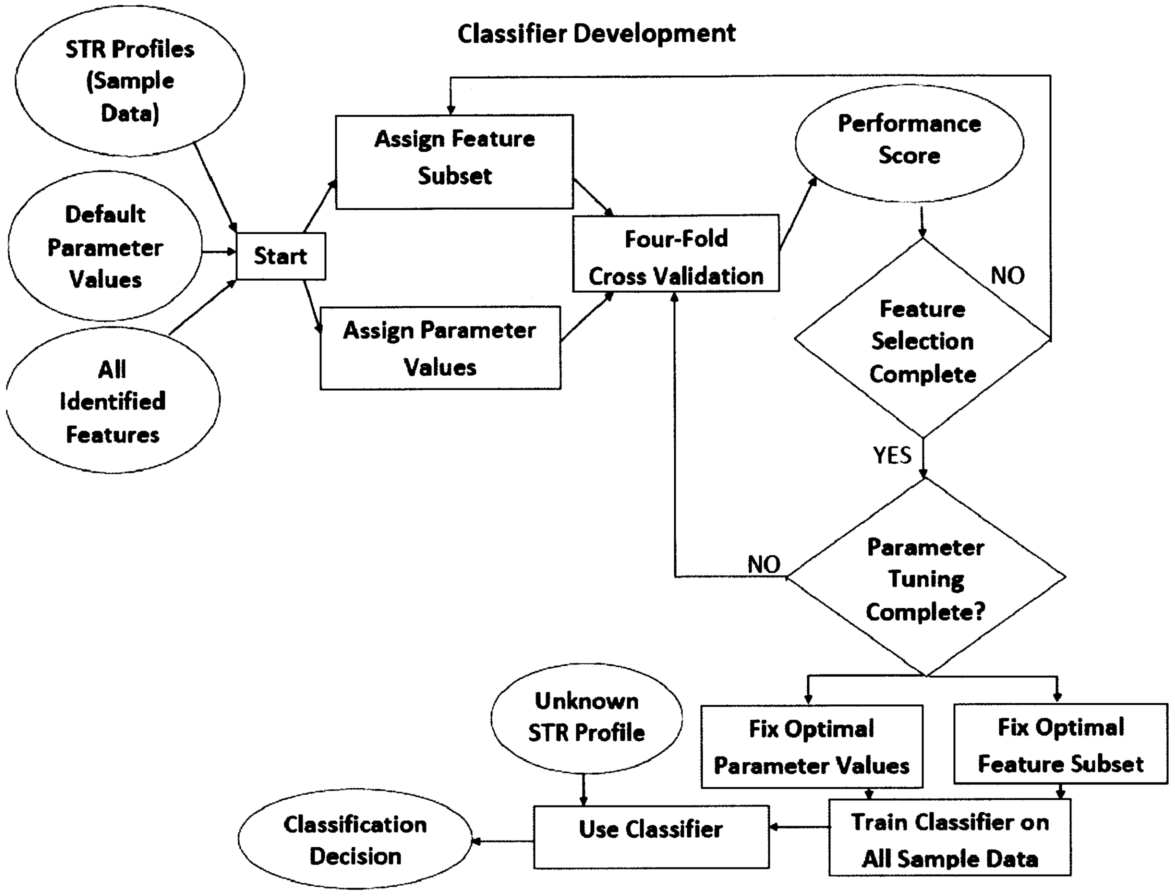


Figure 3-4: *Machine learning classifier development overview.*

3.4 Improving Classifier Performance

The performance of machine learning algorithms is highly influenced by the subset of features used as inputs and by classifier-specific parameters [7]. By selecting the optimal subset of features and tuning a classifier's parameters, performance gains of 10% or more were observed in many cases. **Chapter 4** describes the feature selection process, while **Chapter 5** focuses on parameter tuning. The full classifier development process, incorporating the added steps of feature selection and parameter tuning, is presented in **Figure 3-4**.

Chapter 4

Feature Selection

After generating a set of features for use in training and classification, the next step involved performing selection on this set. Classifier training is an optimization problem in a many-dimensional space. Increasing the dimensionality of the space by adding more features causes an exponential increase in the problem complexity [30]. Consequently, feature selection was performed in order to achieve three goals: improve classifier generalization by reducing classifier complexity, reduce classifier computation requirements, and gain a greater understanding of the problem. In regard to the last goal, classifiers with fewer features are easier to analyze and are more likely to suggest new measures [30].

Feature selection was performed by identifying and eliminating two sets of undesirable features. The first set included features that provided redundant and irrelevant information. Correlated features provide similar information and may be linearly related. Such features make the results challenging to analyze, since omitting one correlated feature does not have an effect on performance. Irrelevant features are undesirable because they add noise to the dataset. The other category of undesirable features are those that add incremental and insufficient information. Incremental features provide a small amount of additional information and may improve classifier performance only with large training datasets.

To eliminate such undesirable feature, selection was done based on actual classifier performance. Forward search (add features incrementally) and backward search (delete features incrementally) were used to identify useful features. These techniques were chosen because they analyze features in combination rather than individually. Such an approach is needed because individual per-feature analysis cannot predict which features may combine to improve performance [30]. Classifier performance in forward and backward search was measured via N-fold cross validation.

4.1 Replicate Analysis

Most samples were examined in replicate. In **Table 2.1**, "1 sample, 4 replicates" indicates that the STR profile was generated from the donor sample five times. Randomly, one of these five profiles was designated as the original, and the remaining four were termed replicates. Replicate analysis was performed to determine whether individual iterations of the multiplex PCR process introduce significant variation in the output profile to influence algorithm development. The initial hypothesis was that replicates should not differ significantly in the features of interest. The results of replicate analysis for a natural profile and a WGA profile are summarized in **Figures 4-1** and **4-2**.

Replicate analysis for a natural profile showed that replicates had similar values for some features, such as inter-channel intensity, peak width, and the presence of off-ladder alleles. However, they differed in other feature values such as the intra-locus imbalance ratio, the inter-locus imbalance rate, SNR, Gaussian error, ski slope, and stutter count. The feature differences among replicates dictated the need to train and test on multiple replicates of a given sample. However, the feature similarities suggested the need to avoid testing on training data. Consequently, though all replicates of a sample were used in classifier development, in the four-fold cross validation process, replicates were always assigned to the same fold so that they jointly served

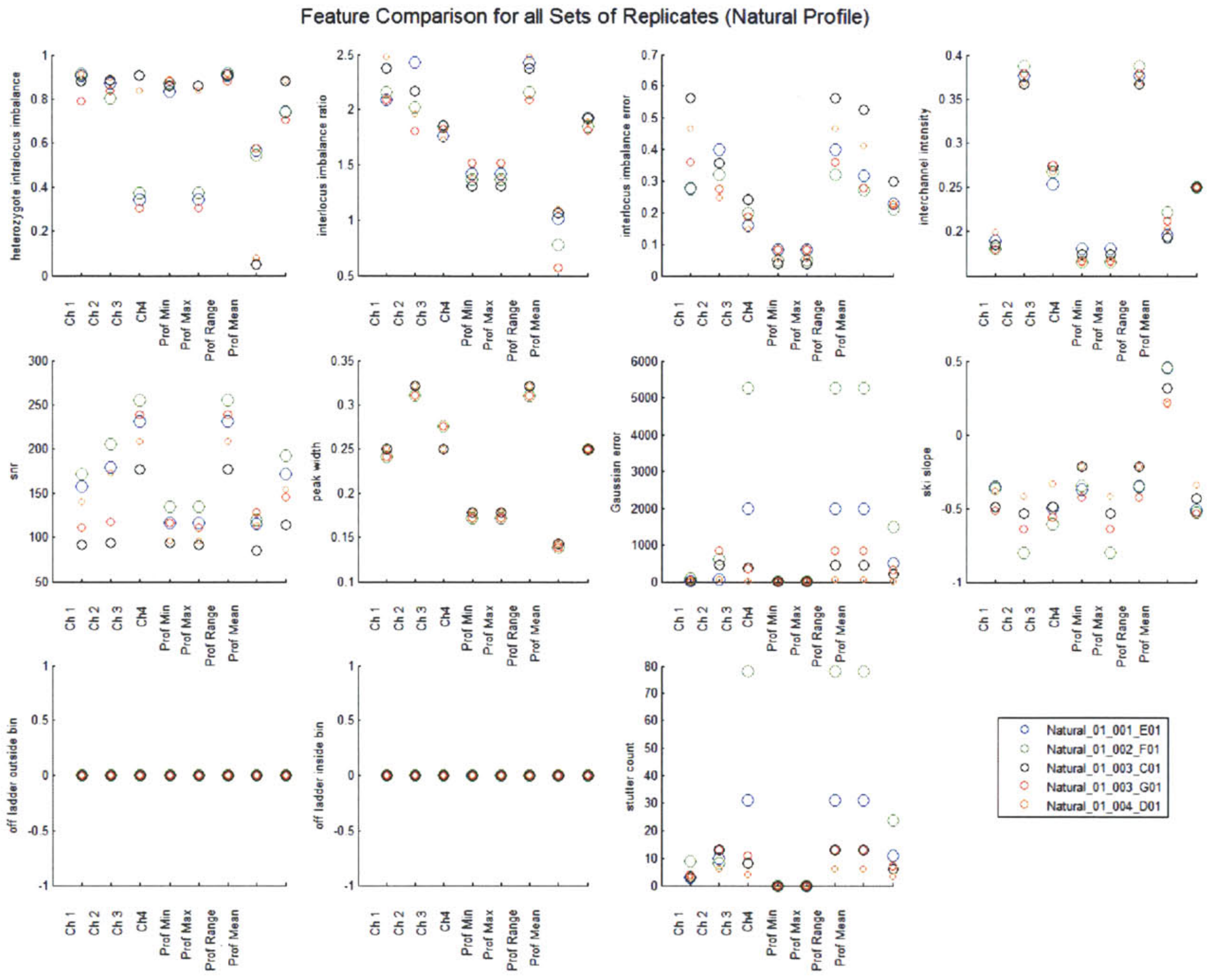


Figure 4-1: *A natural profile with four replicates.*

as either part of the training data or the test data, but not both.

4.2 Feature Selection Algorithms

AS demonstrated in **Figure 4-3**, greedy forward search was used to find the subset of features that led to optimal performance, as measured by a weighted sum of the true positives, true negatives, false positives, and false negatives. All continuous feature values were normalized to zero-mean and unit variance. Outliers, defined as any feature value more than three standard deviations from the mean, were removed from the profiles prior to feature selection. Features were added one at a time in the order that led to the best performance [43]. Performance was measured by four-fold

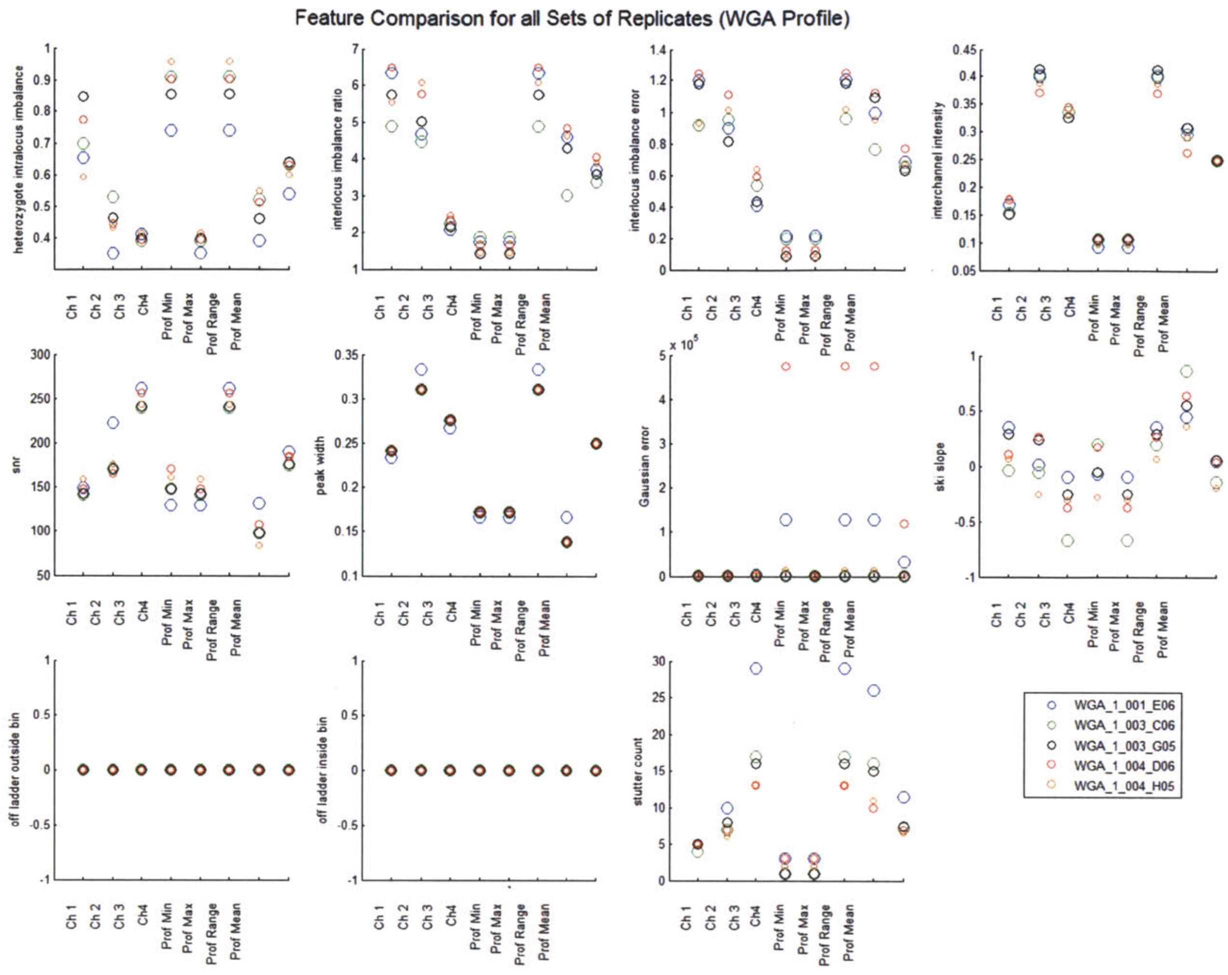


Figure 4-2: *A WGA profile with four replicates.*

cross validation for the chosen feature set. The subset of features that resulted in the lowest error rate was selected. Up to a point, adding features improved performance, but once feature 14 was added, the addition of all subsequent features caused an increase in classification error. This phenomenon is due to over-fitting: the classifier was adjusted to perform well on the training data, but this performance did not generalize to the test data [7]. The overfitting problem arises because the datasets available for classifier training were small. The four-fold cross validation process ameliorates this problem, but is insufficient to completely avoid it.

It is also important to note that the order in which the features were selected had significance. For example, from **Figure 4-3**, it appears that adding features 4 and 6 produced the highest drop in classification error. However, features 4 and 6 were not the first ones selected by the classifier because they produce such a high drop in the error rate only in combination with the previously chosen features.

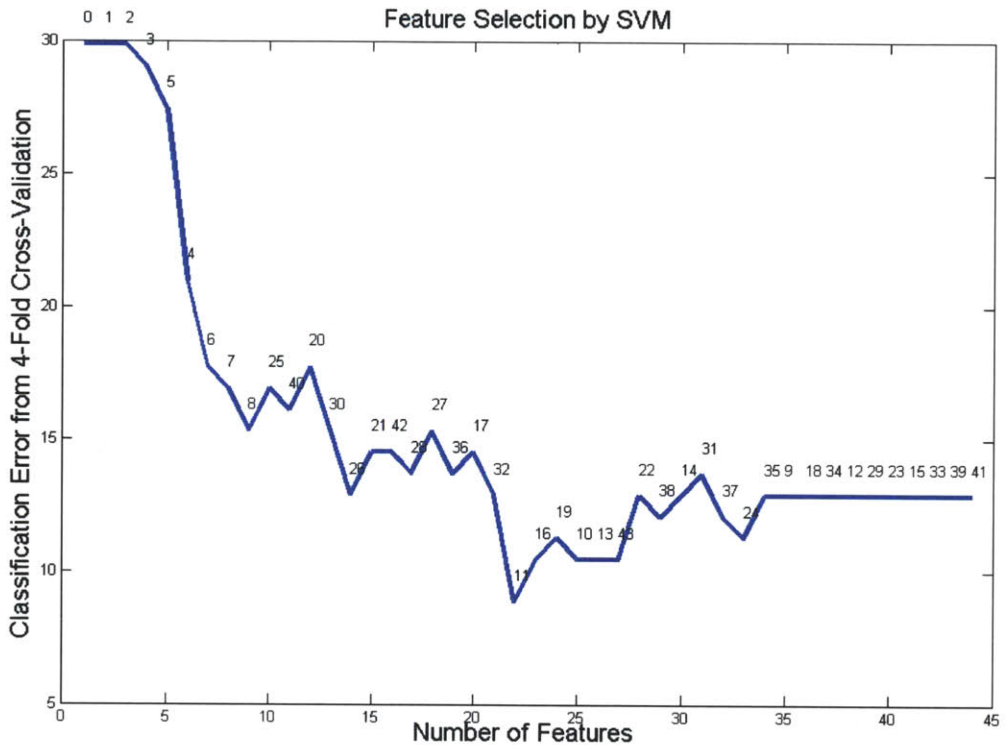


Figure 4-3: *Support vector machine feature selection. The results of feature selection for other classifiers are included in **Appendix B**.*

Several variations on the feature selection algorithm were tested. In one implementation, features were added to the classifier one at a time via forward search. In another implementation, the classifier began with the full set of features and removed one at a time, at each iteration removing the feature that produces the smallest drop in error. Another approach, forward-backward search, used a combination of the two, alternating feature addition with feature removal.

Other variations in the feature selection algorithm dealt with the manner in which optimal performance was measured. In the most straightforward approach, the goal was to minimize overall classification error, as defined by the total fraction of test profiles that were classified incorrectly. In another approach, the aim was to minimize false positives for the WGA class, as defined by $p(wga|nat)$. This quantity refers to the probability that a classifier labels a profile as WGA when, in reality, the sample is a natural profile. In a third approach the goal was to minimize false negatives for the WGA class, defined as $p(nat|wga)$. This converse value refers to the probability that a classifier labels a profile as natural when the sample is in fact a WGA profile.

Figure 4-3 indicates the results of feature selection for the support vector machine classifier. The classifier performed best with a subset of 21 features. The figures in **Appendix B** illustrate the feature selection process for the remaining 15 classifiers.

4.3 Clustering Algorithms

Some of the classifiers described in **Chapter 3**, initialized hidden nodes or other parameters using pre-trained clusters. These include the radial basis function, incremental radial basis function, Gaussian mixture model, nearest cluster classifiers, and learning vector quantizer classifiers. Thus, though clustering algorithms were not a feature proper, they functioned as a feature in the sense that the choice of clustering algorithms strongly influenced classifier performance. Consequently, these algorithms are worth noting. Clusters were trained on labeled data via four-fold cross validation. That is, a separate set of clusters were trained for WGA and natural datasets respectively, each of which had a mean and a diagonal covariance matrix. It was observed that the choice of clustering algorithm influenced classifier performance. Furthermore, the number of clusters generated by each clustering algorithm was varied and also treated as a parameter of the learning algorithm.

The goal of clustering is to group like samples together based on their feature values. Four clustering algorithms were applied: K-means, estimation-maximization, leader clustering, and random clustering.

1. The K-Means clustering algorithm positions a set of K centers in order to minimize the total squared error distance between each training pattern and its

nearest center; the position of that center is then moved to the mean of the patterns assigned to it. Clusters are trained iteratively using a predefined value of K. This clustering algorithm was shown to be effective for linear vector quantization [7].

2. Expectation maximization clustering maximizes the likelihood of the training patterns while training the means, variances, and mixture weights of Gaussian mixture.

3. Leader clustering is a simple fast sequential clustering algorithm. Training patterns are presented one at a time. The first pattern is the first cluster center. Any other pattern that is farther away than delta from an existing cluster center is stored as a new cluster center.

4. The random classifier selects K training patterns to use as the cluster centers. These centers are the first K patterns presented to this clusterer. After centers have been selected, cluster variances are calculated. Similar to the K-means algorithm, the random clustering algorithm assigns each training pattern to the cluster center nearest to it. The cluster is then assigned the variance of its patterns.

4.4 Effects of Feature Selection

Figure 4-4 demonstrates the effects of feature selection on the number of misclassified profiles. Each Identifiler profile was classified with all 16 classifiers and the full set of features, and the number of errors was summed. Feature selection was then performed, and each profile was classified once more with the set of 16 classifiers, but this time considering only the subset of the features identified by feature selection led to an overall 13% reduction in classification error. The effect varied by profile – some profiles were actually more likely to be misclassified post-feature selection, but the majority saw a drop in error. The effect also varied by classifier, as will be discussed in **Chapter 5** and **Appendix B**). Although feature selection did not dramatically improve the performance of all classifiers, individual classifiers such as ARTmap, Parzen, MLP, KNN, and CKNN saw improvements in performance of over 10 percentage points.

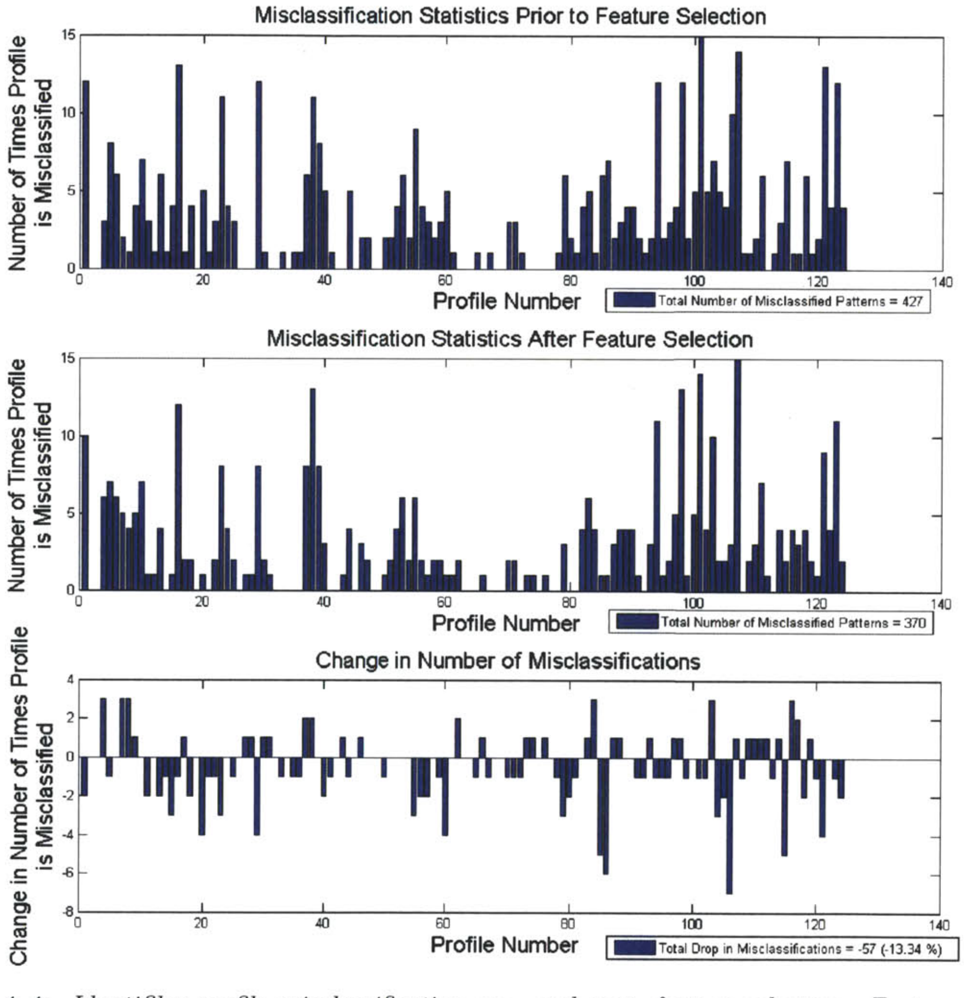


Figure 4-4: *Identifiler profile misclassification pre- and post- feature selection. Feature selection led to a drop of 13 percentage points in the overall number of classification errors.*

Furthermore, the feature selection algorithm did not assign similar sets of features to most of the classifiers studied. **Figure 4-5** and the accompanying **Table 4.1** illustrate the usefulness of individual features. A feature was said to be useful if it was selected by many classifiers during the feature selection process. The most useful feature, the maximum inter-locus imbalance error across the channels, was selected by 15 of the classifiers. Other useful features included the maximum stutter count value for a profile, the range of peak widths, and the mean inter-locus imbalance ratio. These features were chosen by at least three fifths of the classifiers. Other features were demonstrated to be of minimal usefulness, selected by only one or zero classifiers. These include the off ladder inside bin allele count. For most profiles considered, this count was extremely low (generally zero or one), so it is quite possible that the rarity of these alleles makes the off ladder inside bin allele count a poor feature for classifier training.

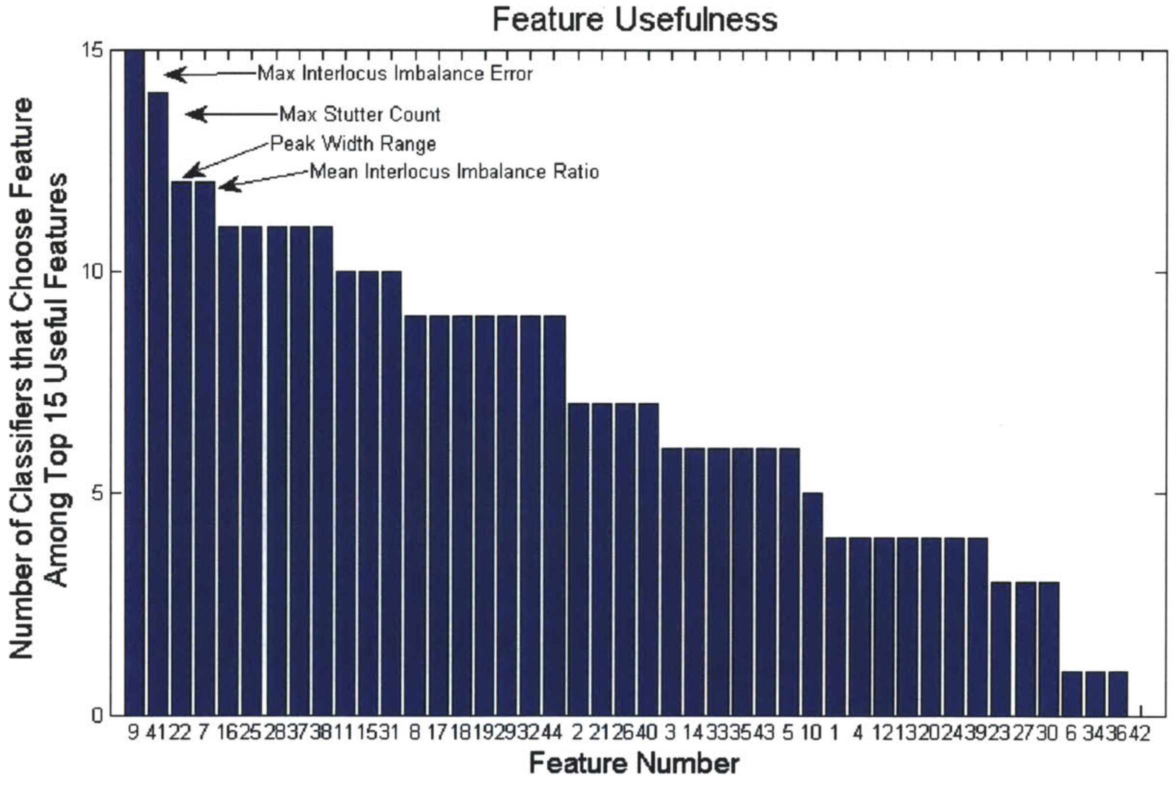


Figure 4-5: *Some features are particularly useful across a variety of the classifiers examined.*

	Prof Min	Prof Max	Prof Range	Prof Mean
Heterozygote intra-locus imbalance	0	1	2	3
Inter-locus Imbalance Ratio	4	5	6	7
Inter-locus Imbalance Error	8	9	10	11
Inter-channel Intensity	12	13	14	15
SNR	16	17	18	19
Peak Width	20	21	22	23
Gaussian error	24	25	26	27
Ski Slope	28	29	30	31
Off ladder inside bin	32	33	34	35
Off ladder outside bin	36	37	38	39
Stutter count	40	41	42	43

Table 4.1: *Feature guide: each number indicates a specific feature value for a profile.*

Chapter 5

Parameter Optimization

The LNKnet machine learning toolkit was used to train and evaluate each classifier via a three-step method [27]. First, to evaluate baseline performance, each classifier was trained on the set of 44 profile-level features, as described in **Section 2.4**, with the default set of parameters included in the LNKnet toolkit. The second step involved performing feature selection via the approach described in **Chapter 4**. The final step involved tuning the individual classifier parameters. The classifiers had between one (KNN) and 10 (MLP) parameters. These were adjusted via a modified version of gradient ascent with the goal of maximizing individual classifier performance.

5.1 Motivation for Parameter Tuning

Each classifier has a unique set of parameters that can be finely tuned to maximize performance. The Gaussian classifier, for example, is heavily dependent on the specifications of its covariance matrix. Gaussian linear discriminant classifiers are among the most straightforward and commonly used classification algorithms in machine learning [38]. Consequently, the Gaussian classifier was the first to be applied to the STR profile authentication problem. A Gaussian classifier models each class with a Gaussian distribution centered on the mean of that class. The variance of these Gaussians can be calculated in four ways, as determined by the covariance

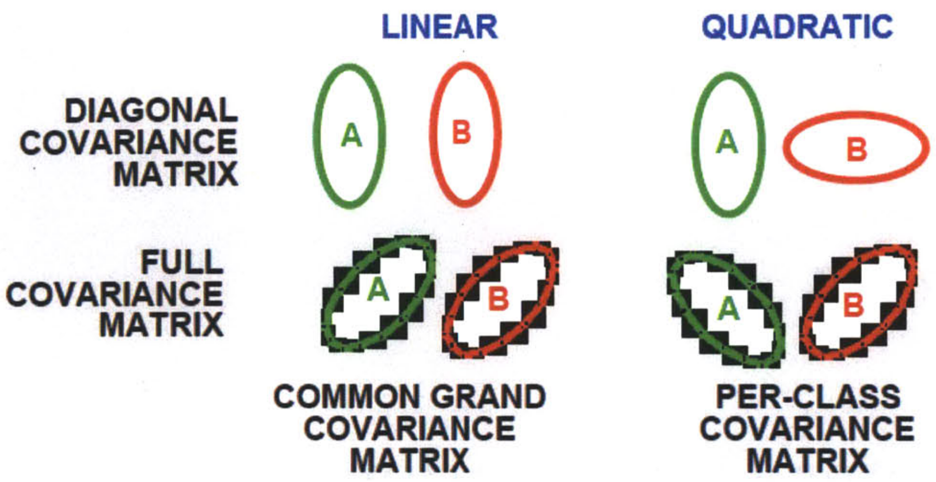

LINEAR
QUADRATIC
DIAGONAL COVARIANCE MATRIX
FULL COVARIANCE MATRIX
A
B
A
B
A
B
A
B
COMMON GRAND COVARIANCE MATRIX
PER-CLASS COVARIANCE MATRIX

Figure 5-1: *The covariance matrix of a Gaussian classifier can be calculate in four ways [38].*

matrix (**Figure 5-1**). The variance of each class can be calculated separately, referred to as a per-class covariance matrix. Alternatively, individual class variances can be averaged to give a common grand variance used for all classes. The covariance matrix can also be either diagonal or full. Diagonal covariance matrices constrain equal-probability ellipses to have major axes parallel to the input feature axes. This implies one variance calculation for each dimension of the input data. Full covariance matrices, on the other hand, allow equal probability ellipses to have any orientation. When there are many input features, full covariance Gaussian classifiers have many more parameters than diagonal covariance classifiers and may perform worse with limited training data. In addition, the variance can be limited to be above a minimum value to prevent numerical problems when input features are unchanged across training patterns. A linear discriminant classifier is a Gaussian classifier with grand variances, where variances are the same for all classes. The simplest linear discriminant classifier uses the same diagonal covariance matrix for each class. A quadratic classifier is a Gaussian classifier with separate variances for each class [43].

It is difficult to visualize the results of the Gaussian classifier for the STR profile dataset in a three-dimensional rendering due to the high dimensionality of the data, but a simpler example for vowel data classification illustrates similar effects of parameter values on performance (**Figure 5-2**). In this example, spoken vowels were classified into 10 classes based on the first (x-axis) and second (y-axis) formants. Each decision region is formed by projecting a three-dimensional Gaussian representation of the class. The ellipses in the figure correspond to these projections. The figure indicates that classification results vary based on the covariance matrix used in the Gaussian classifier, underscoring the importance of tuning classifier parameters to achieve optimal classification results [32].

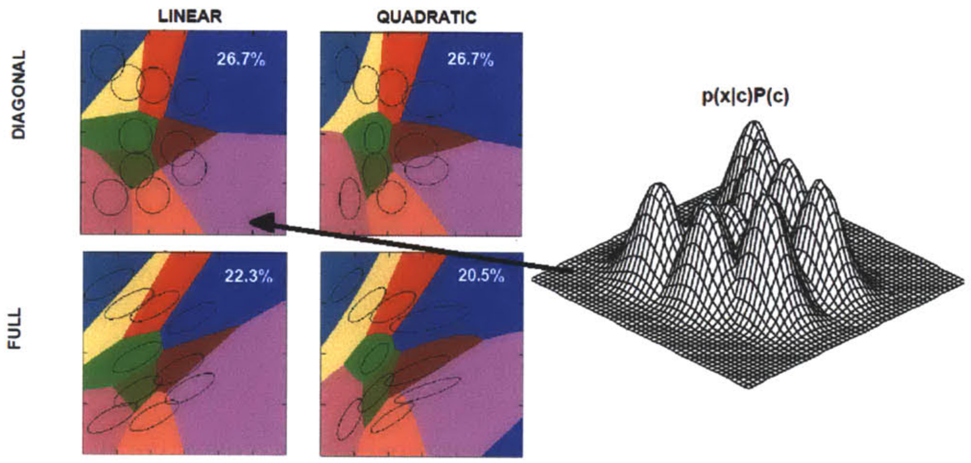


Figure 5-2: *The performance of the Gaussian classifier depends on its covariance matrix [38].*

5.2 Parameter Tuning Algorithm

For each classifier, including the Gaussian model summarized above, parameters were tuned via gradient ascent. The algorithm can be summarized as follows:

- For each parameter
  - Sweep through the range of possible values while holding the other parameters constant at the LNKnet defaults.
  - Perform cross validation and calculate the performance score (**Section 5.3**) corresponding to the current set of parameter values.
- Compare the highest performance score achieved for each parameter and choose

the maximal value. In case of ties in performance score, choose the parameter value that is closest to the LNKnet default.

- Fix the value of the corresponding parameter.
- Repeat this process for the remaining free parameters.

This greedy process is not guaranteed to yield globally optimal results, and has the potential to converge to local extrema of the performance score function.

5.3 Performance Evaluation

Classifier performance was measured by the formula:

$$\boxed{\begin{gathered}\text{score=}\\ w1*p(nat|nat)+w2*p(synth|synth)-w3*p(synth|nat)-w4*p(nat|synth)-w5*p(fail)\end{gathered}}$$

If the goal of classification is to detect synthetic profiles:

- $p(nat|nat)$ is a true negative that refers to the probability of a classifier declaring that a test profile is natural given that the profile actually is natural.
- $p(synth|synth)$ is a true positive that refers to the probability of a classifier declaring that a test profile is synthetic given that the profile actually is synthetic. WGA and bacterial profiles were tested separately against natural profiles. Thus, $p(synth|synth) = p(WGA|WGA)$ or $p(synth|synth) = p(bacterial|bacterial)$, depending on the type of synthetic profile used to tune the classifier. No combinations of bacterial, WGA and natural profiles were included in the same data set due to the greater challenges associated with multi-class classification. However, multi-class classification is a potential direction to pursue in the future and is described in **Chapter 8**.
- $p(synth|nat)$ is a false positive that refers to the probability of a classifier declaring that a profile is synthetic when the profile is actually natural.

- $p(nat|synth)$ is a false negative that refers to the probability of a classifier declaring that a test profile is natural given that the profile actually is synthetic.

- $p(fail)$ refers to the probability that the training algorithm did not converge in the specified number of iterations or that some other type of error occurred during the four-fold cross validation process.

The weights w1 through w5 were assigned manually and adjusted to place varying levels of emphasis on correctly identifying natural profile, avoiding false positives, or avoiding false negatives. The weights could also be adjusted to reflect the non-uniform prior probabilities. For example, in the Identifiler dataset, roughly two-thirds of the profiles are WGA and one third are natural. Thus, there is inherently a higher probability of a classifier declaring that a profile is WGA. By increasing the values of w1 and w3, greater emphasis can be placed on the correct classification of natural profiles, helping to counteract the initial skew in the classifier.

5.4 Parameter Tuning Examples for the Gaussian and ARTmap classifiers

Parameter tuning for the Gaussian classifier is summarized in **Figure 5-3**. The top left subplot illustrates the default performance of the classifier, with the full set of features and the default parameter values (a full grand covariance matrix). As indicated in the legend, the combined area of the green bars refers to the probability that a profile is classified correctly (light green indicates the probability of classifying a natural profile correctly, while dark green indicates the probability of correctly classifying a WGA profile). The combined red area represents the probability of classifying a profile incorrectly (light red refers to the probability of stating that a WGA profile is natural, while dark red refers to the probability of stating that a natural profile is WGA). The baseline performance of the classifier was poor: only 65 percent of profiles were classified correctly. The top right subplot represents the improvement in performance due to feature selection: the probability of correct classification increases from 0.65 to 0.70.

The gradient ascent algorithm was then used to determine the optimal covariance matrix for the classifier. As described in **Section 5.1**, the covariance matrix of the Gaussian classifier is defined in terms of two parameters. These are referred to as *full* and *per_class.* Each of these parameters can take two values: a value of 0 for the *full* parameter indicates a diagonal covariance matrix, while a value of 1 indicates a full covariance matrix. A value of 0 for the *per_class* parameter indicates a grand covariance matrix, while a value of 1 indicates a per class covariance matrix. Thus, in the context of the Gaussian classifier, a parameter sweep simply involves calculating the performance scores for the two possible values of the given parameter (0 and 1).

In accordance with the gradient ascent algorithm, sweeps were performed independently over both the *full* and *per_class* parameters. In each case the non-target parameter was held at the LNKnet default. The maximum performance scores were calculated for each parameter. Comparing the two maximum performance scores indicated that the score was higher for the full parameter. Consequently, this parameter was fixed at the optimal value of 0. A sweep was then performed over the values of the remaining parameter, *per_class*, while holding the *full* parameter at the optimized value. This sweep revealed that setting the *per_class* value to 0 yielded the highest performance score.

The Fuzzy ARTmap classifier (**Figure 5-4**) provides a more complex example of parameter tuning. This supervised neural network classifier depends on five parameters: alpha, beta, alpha training vigilance, alpha test vigilance, and beta vigilance. The adaptive resonance theory (ART) system is an semi-supervised learning model that consists of a comparison field and a recognition field. It is composed of neurons, a vigilance parameter, and a reset module. The vigilance parameters have considerable influence on the system: higher vigilance produces highly detailed memories (many

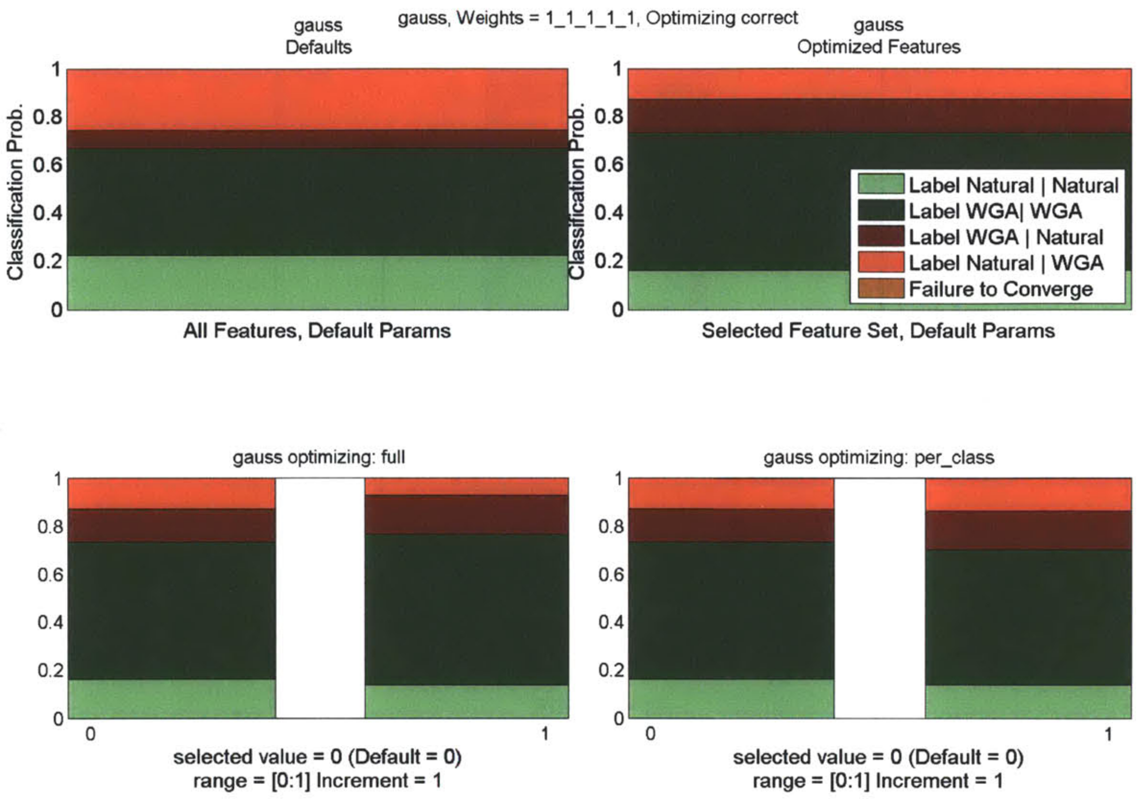


Figure 5-3: *Gaussian classifier with parameters full, per_class*

fine-grained categories), while lower vigilance results in more general memories (fewer more-general categories). The comparison field takes a feature vector and transfers it to its best match in the recognition field. Its best match is the single neuron whose weight vector most closely matches the feature vector. The alpha parameter indicates the degree of recoding in the neural network that is used to make this comparison. There are two basic methods of training ART-based neural networks: slow and fast. The learning rate is indicated by the beta parameter. In the slow learning method, the weight updates for the recognition neurons are continuous values calculated via differential equations. These updates depend on the length of time that an input feature vector is presented. With fast learning, simpler algebraic equations are used to calculate weight adjustments, and more coarsely sampled values of time are used [24].

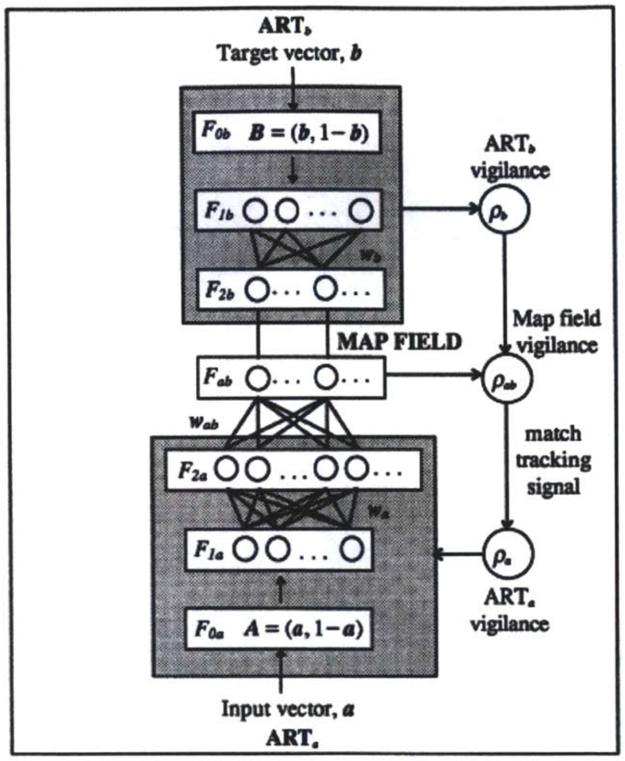


Figure 5-4: *Fuzzy ARTmap classifier schematic [24].*

As demonstrated in **Figure 5-5**, the alpha, beta, and vigilance parameters were tuned using the modified gradient ascent algorithm. Among the classifier parameters *alpha*, *beta*, *alpha_train_vigil*, *alpha_test_vigil*, and *beta_vigil*, four of the parameters were set to the default LNKnet values, and the remaining fifth parameter was swept through the full range of possible values. The optimal parameter value was determined by identifying the highest performance score, and performance scores were compared for the individual parameters to determine which parameter value should be fixed at the new value. The process was then repeated for each of the remaining ARTmap parameters until all four remaining parameters were tuned. The parameters were fixed in this order: *alpha_test_vigil*, *alpha_train_vigil*, *alpha*, *beta*, *beta_vigil*.

Figure 5-5 demonstrates that the *alpha_test_vigil* parameter can take on values ranging from 0 to 0.8 with no effect on performance. In such a situation, the parameter value closest to the LNKnet default was selected: for the ARTmap classifier the *alpha_test_vigil* parameter was set to 0.8 because that value was closest to the default of 0.9. Additionally, turning the parameters **alpha**, **beta**, and **beta_vigil** did not affect classifier performance. Consequently, the tuned values of these parameters were kept at the LNKnet defaults.

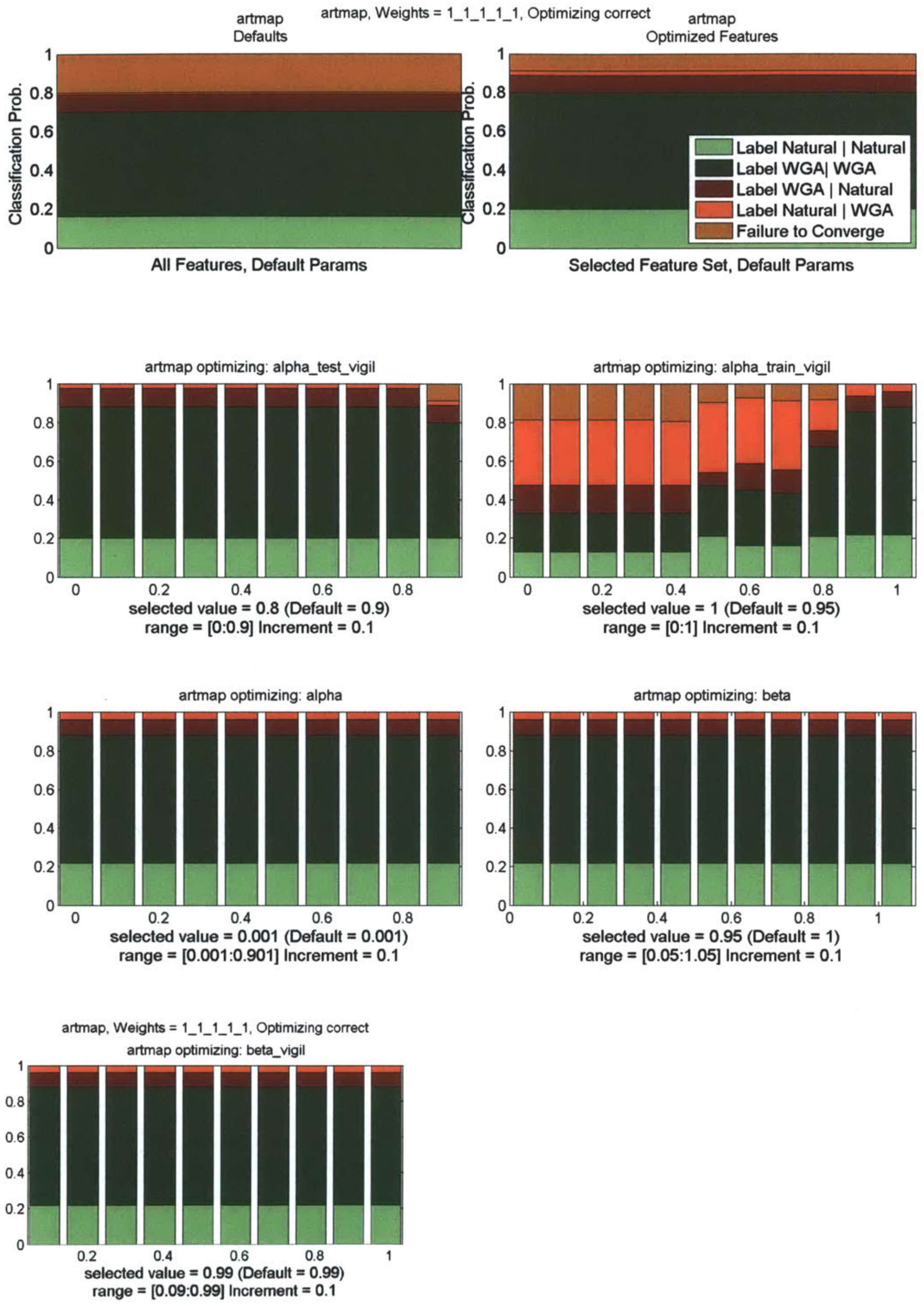


Figure 5-5: *Artmap classifier with parameters alpha_test_vigil, alpha_train_vigil, alpha, beta, beta_vigil.*

The parameter-tuning process for the highest performing classifiers is shown in **Chapter 6**. Additional classifiers are summarized in **Appendix C**.

5.5 Full Parameter Sweeps

The drawback to the gradient ascent approach is its failure to capture interdependencies between parameters. If the performance metric is viewed as a function over classifier parameters, the modified gradient ascent approach may produce a result that is a local maximum but not a global maximum of the function. The risk of converging to a local maximum in the performance score can be reduced by replacing the gradient ascent approach with a full parameter sweep, although this imposes a large performance penalty. Such a parameter sweep is illustrated in **Figure 5-6** for the Gaussian classifier. The top plot indicates the ratio of true positives (dark green) to false positives (dark red) for the WGA class. The bottom plot presents this ratio for the natural class. The center plot summarizes the total fraction of profiles that fall into each of the four categories $p(nat|nat)$, $p(wga|wga)$, $p(wga|nat)$, $p(nat|wga)$. This plot helps to explain why the default values of 0 were chosen for both the **full** and **per_class** parameters: though the individual ratios of true positives to true negatives are higher for both the WGA and natural classes when the full parameter is set to 1, the combined ratio of correct to incorrect profiles is higher when the parameter is set to 0. This combined value was obtained by summing the height of the dark and light green surfaces in the center plot and comparing the result to the sum of the heights of the light and dark red surfaces.

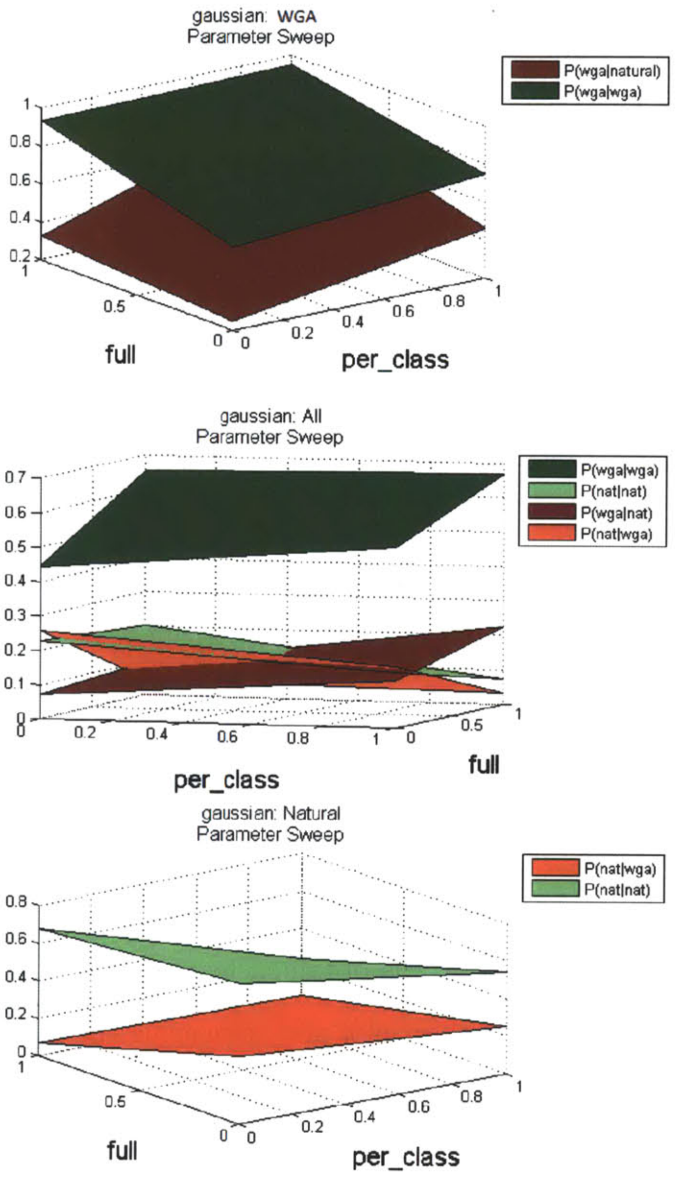


Figure 5-6: *Parameter sweep for the Gaussian classifier with covariance matrix determined by parameters full and per_class.*

The Gaussian classifier has only two tunable parameters, so a full parameter sweep is computationally feasible. However, the high cost of this approach is evident in **Figure 5-7**, which shows a full parameter sweep for the ARTmap classifier. For ARTmap, it was determined that the parameters **alpha** and **beta_vigil** do not play a significant role in determining the performance score, so to minimize computation time, only **beta**, **alpha_train_vigil**, and **alpha_test_vigil** were used for the surface sweep. The scatterplot reveals the drop in performance for **alpha_test_vigil** = 0.9, as well as the increase in performance for high values of the **alpha_train_vigil** parameter. However, the scatter plot also indicates that best performance is achieved for low values of the **beta** parameter, while the gradient ascent approach suggested that the value of the **beta** parameter does not have a significant effect on the classifier performance. This discrepancy shows the limitations of the gradient ascent approach and the effects of local maxima. However, due to computational costs associated with performing full parameter sweeps, especially on classifiers with multiple parameter values, the gradient ascent approach alone was used to evaluate the performance of all other classifiers.

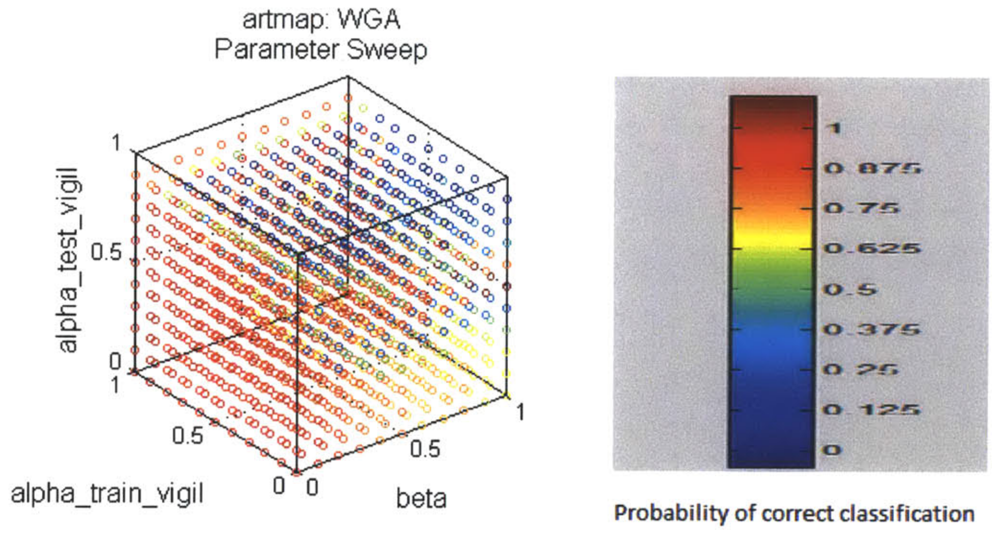


Figure 5-7: *Parameter sweep tuning for the ARTmap classifier with parameters **alpha_train_vigil**, **alpha_test_vigil**, and **beta**.*

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 6

Results of Feature Selection and Parameter Tuning: Determining Optimal Classifier Behavior

Once feature selection and parameter tuning were performed for the individual classifiers, their performance was compared to determine the overall optimal combination of features, parameters, and classifiers for the STR profile authentication problem. To evaluate performance, the weights w1 through w5 in the performance score were all set to 1. This gave a score metric of

$$\text{performance} = p(wga|wga) + p(nat|nat) - p(wga|nat) - p(nat|wga) - p(fail)$$

Other score metrics are presented in **Section 6.3**.

6.1 Classifier Comparison with Performance Score Weights = [1,1,1,1,1]

Figure 6-1 shows the performance of all classifiers on the Identifiler data set when all weights in the performance score were set to 1.

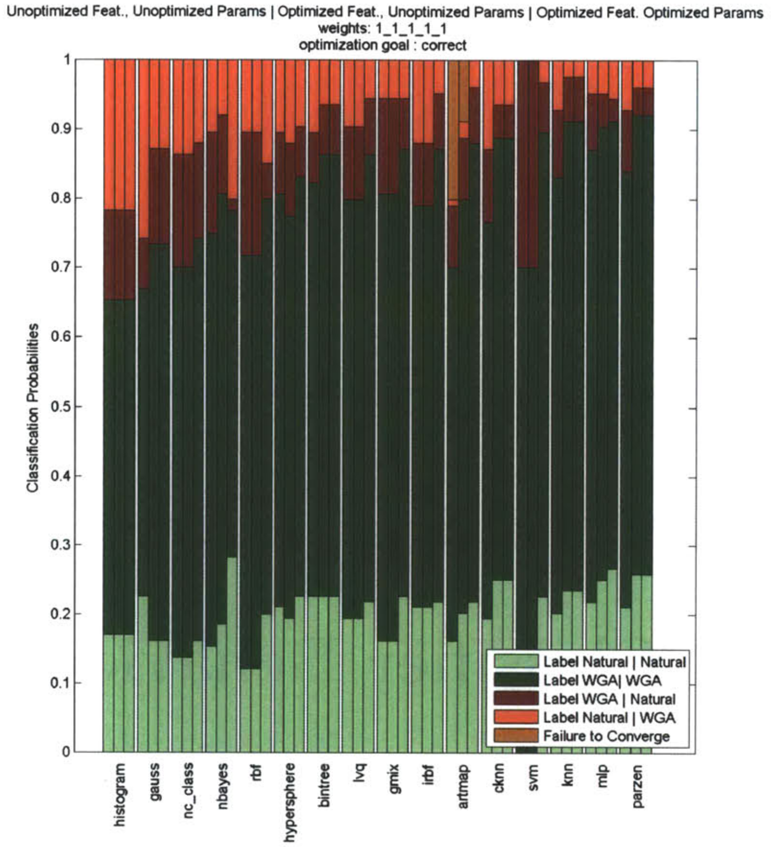


Figure 6-1: *Identifiler, weights = [1,1,1,1,1], column 1: default performance, column 2: performance after feature selection, column 3: performance after parameter tuning.*

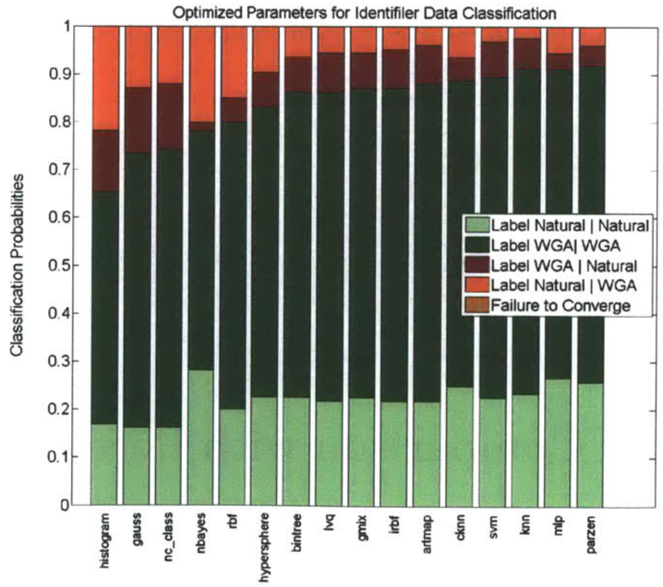


Figure 6-2: *Identifiler, weights = [1,1,1,1,1]. Performance after parameter tuning is shown.*

In the figure, each classifier is summarized in three bars. The first bar measures the default performance of the classifier, the second bar measures the performance after feature selection, and the third bar measures performance after both feature selection and parameter tuning. The classifiers are sorted in ascending order by final performance. **Figure 6-2** focuses on only the final, tuned performance of each classifier. These figures suggest that all classifiers performed reasonably well – most achieved error rates below 20 percent, and even the worst performing classifier (histogram) correctly classified 64 percent of the samples. Of the classifiers analyzed, the condensed K-nearest neighbors, K-nearest neighbors, multi-layer perceptron, Parzen window, and support vector machine classifiers performed particularly well, all achieving error rates near ten percent. Their performance is summarized in more detail in **Table 6.1**. The parameter tuning process for these classifiers is summarized in **Figures 6-3** through **6-7**.

All five of the top performers achieved higher values for $p(wga|wga)$ compared to $p(nat|nat)$. Furthermore, all were more likely to misclassify a WGA profile as natural rather than misclassify a natural profile as WGA. These results are unsurprising: the scatter plots in **Appendix A** indicate that most features take on a higher range of values, with a higher standard deviation of values from the mean, for the WGA samples. Thus, the features of a natural sample could have values that fall in the WGA range, but it is less likely that a WGA sample would have features that fall into the natural range.

Classifier	**p(correct)**	**p**$(nat\|nat)$	**p**$(wga\|wga)$	**p**$(wga\|nat)$	**p**$(nat\|wga)$
cknn	0.8871	0.8378	0.9080	0.1622	0.0920
svm	0.8952	0.7568	0.9540	0.2432	0.0460
knn	0.9113	0.7838	0.9655	0.2162	0.0345
mlp	0.9113	0.8919	0.9195	0.1081	0.0805
parzen	0.9194	0.8649	0.9425	0.1351	0.0575

Table 6.1: *Performance of top classifiers: CKNN, SVM, KNN, MLP, Parzen.*

6.2 High-Performing Classifiers

When all weights were set to one, the highest-performing classifiers were CKNN, KNN, MLP, Parzen, and SVM. As demonstrated in **Sections 6.3** and **6.4**, these classifiers continued to perform well when the performance score weight vector was varied: for a weight vector of [1,1,1,10,1], the highest-performing classifiers in descending order were Parzen, MLP, KNN, CKNN, ARTmap, and SVM. With the exception of the ARTmap classifier, these are the same as the top performers for the [1,1,1,1,1] vector. Similarly, for the weight vector [3,1,1,3,1], the top performers were KNN, MLP, SVM, Gaussian, Gaussian mixture model, and CKNN. The Gaussian and Gaussian mixture model classifiers are new to the top five, and the Parzen classifier did not perform as well as in the other test cases. Finally, using the [1,3,3,1,1] metric rendered the top performers as KNN, Parzen, MLP, SVM, and ARTmap. These overlap with the previous cases, with the exception of the missing CKNN classifier. This suggests that a subset of the classifiers consistently perform well, and these are summarized below in greater detail.

6.2.1 K-Nearest Neighbors (KNN)

During the training phase, a K-nearest neighbors classifier stores all the patterns presented to it. In the subsequent test phase, it uses a Euclidean distance measure to iterate through all stored patterns and identify the K neighboring patterns that are closest to a pattern of interest. A vote is taken among the K neighbors and the class that occurs the most is assigned to the test pattern. Ties are broken randomly. As demonstrated in the top right plot of **Figure 6-3**, feature selection has a powerful effect on the performance of this classifier, improving the probability of correct classification from 82% to 91%. Additional performance improvements were achieved by tuning the K parameter, which determines the number of nearest neighbors to consider [36]. As demonstrated in **Figure 6-3**, the classifier performed best for K

=1, and performance declined nearly linearly as the value of K increased.

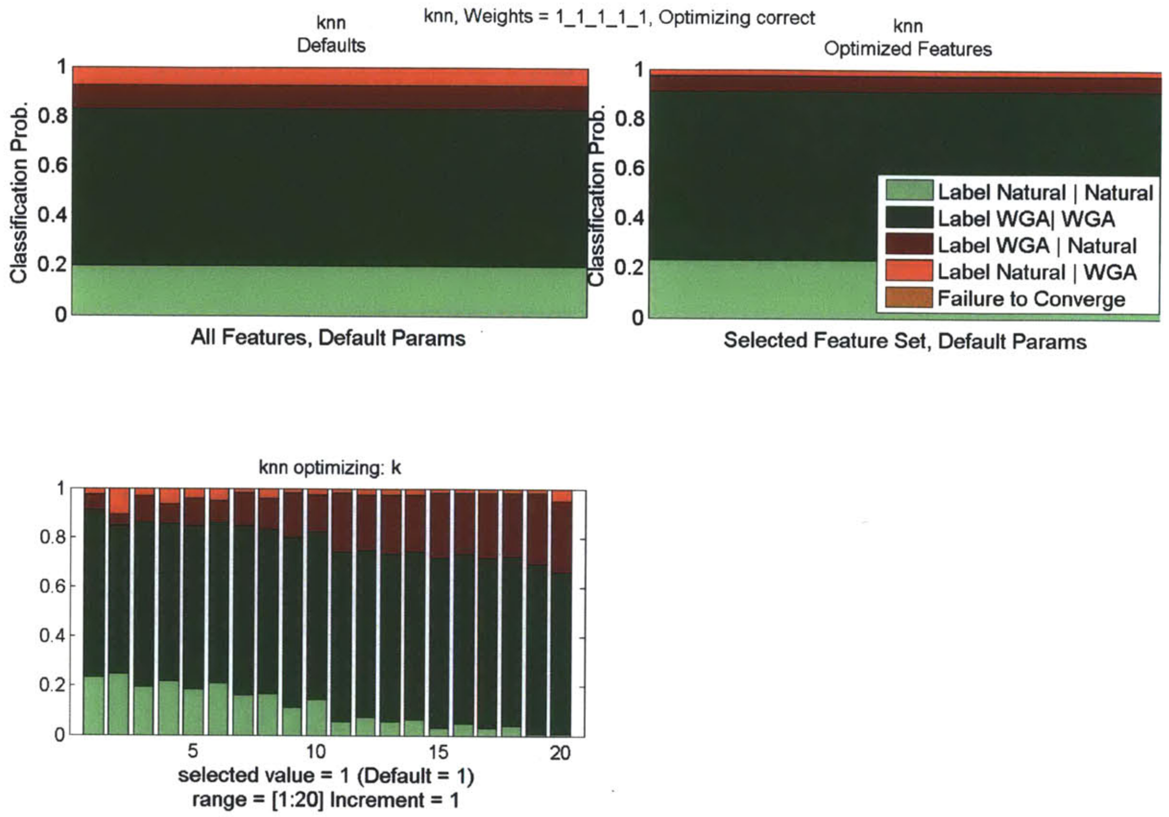


Figure 6-3: *K nearest neighbors with parameter K.*

6.2.2 Condensed K-Nearest Neighbors (CKNN)

The condensed K-nearest neighbor classifier (CKNN) functions similarly to the KNN classifier, but stores fewer patterns. During the training phase, CKNN examines training patterns successively. It stores only those patterns that are classified incorrectly during testing. As in the case of the KNN classifier, feature selection improved the performance of CKNN by approximately 10 percentage points. In **Figure 6-4**, the parameter *epochs* refers to the number of times the patterns were examined during training [38]. Adding epochs of training improved performance up to a point: five epochs yielded higher performance than one epoch. This occurred because each time the patterns were examined, the classification boundaries were adjusted to reduce the number of misclassifications. However, beyond five epochs, adding further iterations had no significant effect. Unlike the KNN classifier, the CKNN classifier was not strongly affected by the value of the K parameter. Setting K to one still gave the highest performance score, but higher values of K did not hurt performance as significantly as for KNN. This is due to the fact that the CKNN classifier places more emphasis on neighbors that are misclassified rather than on the entire set of neighbors.

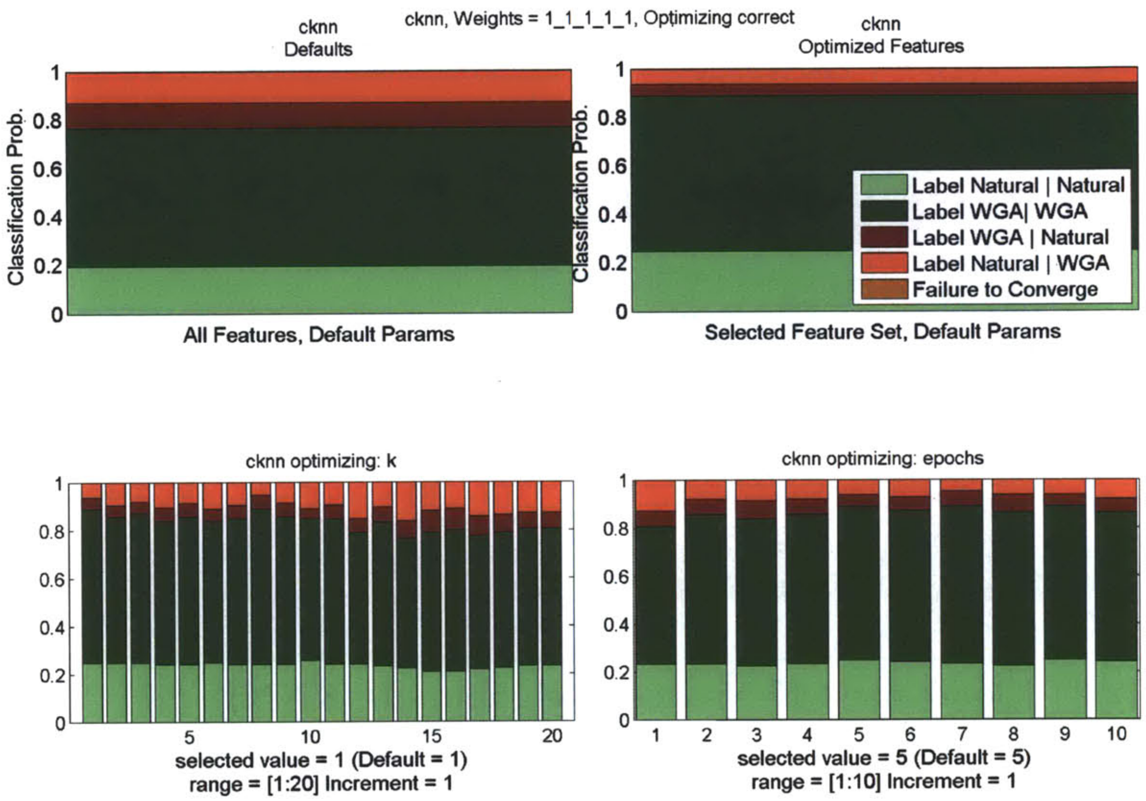


Figure 6-4: *K nearest neighbors with parameters K and epochs*

6.2.3 Multi-Layer Perceptron (MLP)

The multi-layer perceptron (MLP) classifier (**Figure 6-5**) fits sigmoid discriminant functions to the input space. Weighted connections between layers of the MLP are used to create hyperplanes that partition the input space into half spaces. The half spaces correspond to decision boundaries that separate the natural class from the WGA class. The structure of the classifier is defined by the number of hidden layers, the number of nodes in each layer, and output node processing steps. The decision made by the classifier is a weighted sum of the output nodes. If two classes are separable, the MLP algorithm is guaranteed to converge in finite time and to find a separating hyperplane [7].

A back propagation gradient descent algorithm was used to train the weights between the perceptron nodes [23]. If the MLP algorithm misclassified a training pattern, the weights in the discriminant function corresponding to that pattern were adjusted so as to minimize the probability of future errors. Ultimately, the combined magnitude of these weights determined the behavior of the classifier. 10 parameters governed the behavior of the MLP. The first to be tuned was *epochs*, which refers to the number of times the data was examined. The next parameter was *alpha*, which refers to the weights of the connections between nodes in the hidden layer of the network (**Figure 6-5**). With small weights (low alpha), the network behaved linearly. With medium weights, smooth nonlinearity was observed, approximating Gaussian posterior probabilities. Large weights induced threshold logic: the network performed logical "AND" and "OR" operations. The MLP classifier performed best for medium values of *alpha* in the 0.5 - 0.9 range; 0.5 was selected as the optimal value. The number of nodes in the hidden layer of the network was then optimized to a value of 25.

A gradient descent algorithm needs a step size, which is a multiplier applied to the gradient when the weights are updated. For the MLP this is the *etta* parameter. When set to zero, the *etta_change_type* parameter denoted a constant step size for gradient descent; when set to one, it denoted a step size inversely proportional to the number of steps taken. According to **Figure 6-5**, performance did not depend strongly on the value of *etta*, but varying the *etta_change_type* led to a 30% range in performance. This indicates that, for the STR data set, the classifier performed much better when a constant step size was used for gradient descent. This result is surprising, as the literature suggests that calculating an optimal step size at each iteration generally leads to improved performance when compared to using a constant step size [7]. It is likely that a confounding factor lead to better performance for the constant step size.

The gradient descent algorithm was performed using a cost function collectively defined by the parameters *hfunction*, *ofunction*, and *cost_fun*. Five cost functions types were evaluated: the squared-error function (parameter value 0), maximum likelihood function, the cross-entropy function, the perceptron convergence procedure, and the top-two difference function. Of these, the squared error and perceptron convergence functions both led to correct classificantion rates greater than 90%. It is unsurprising that the squared error function lead to optimal behavior, as this function promotes small weights and simpler or smoother solutions. The choice of this function was in line with the choice of the *alpha* parameter (see above), as both choices indicate Gaussian distributions and smooth nonlinearity in the input data set [31].

The behavior of the squared error function in turn depended on several other parameters. One of these was the choice of the output function that is applied to the weighted sum calculated for the output layer. Output functions could take the form of a standard sigmoid (parameter value 0), a symmetric sigmoid (parameter value 1), or a linear weighted sum (parameter value 2). The tuning algorithm selected the value of 0 for the *ofunction* parameter, indicating that a standard sigmoid function led to the best performance.

The final parameters to be tuned were *init_mag* and *sigmoid_param*. *init_mag* refers to the maximum magnitude of the initial weights, and the *sigmoid_param* parameter adjusts the steepness of the sigmoid cost function. A higher steepness value sharpens the decision region boundaries. Parameter tuning revealed that optimal performance was achieved when the sigmoid steepness was set to one. Performance was significantly worse for values of *sigmoid_param* in the 0.1 - 0.6 range, and dropped gradually for values of the parameter greater than one. This suggests that the sharpness of the decision boundaries is important, and that the classifier prefers decision boundaries that are neither too smooth nor too sharp.

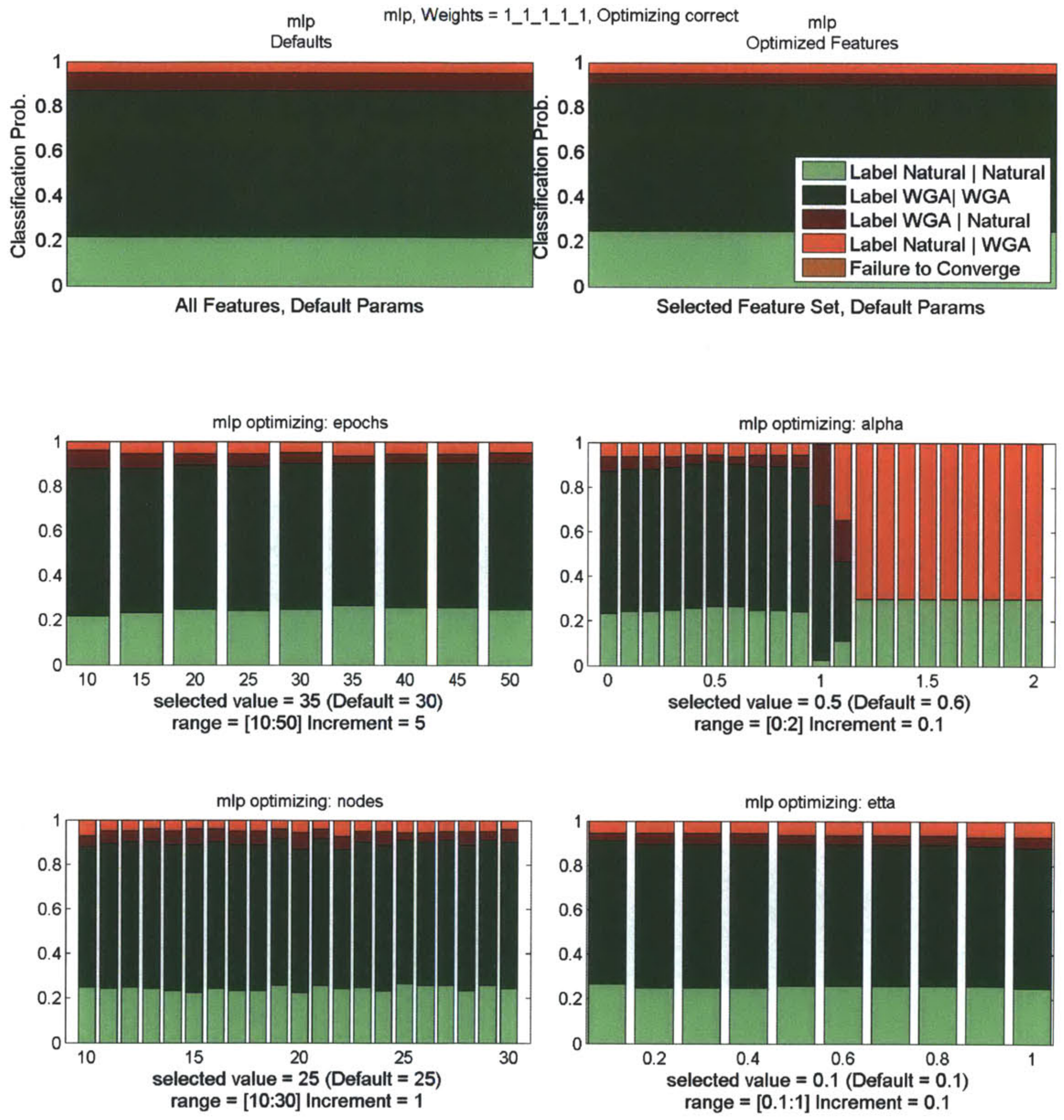


Figure 6-5: *Multi-layer perceptron with parameters epochs, alpha, nodes, etta, etta_change_type, hfunction, ofunction, cost_fun, init_mag, sigmoid_param.*

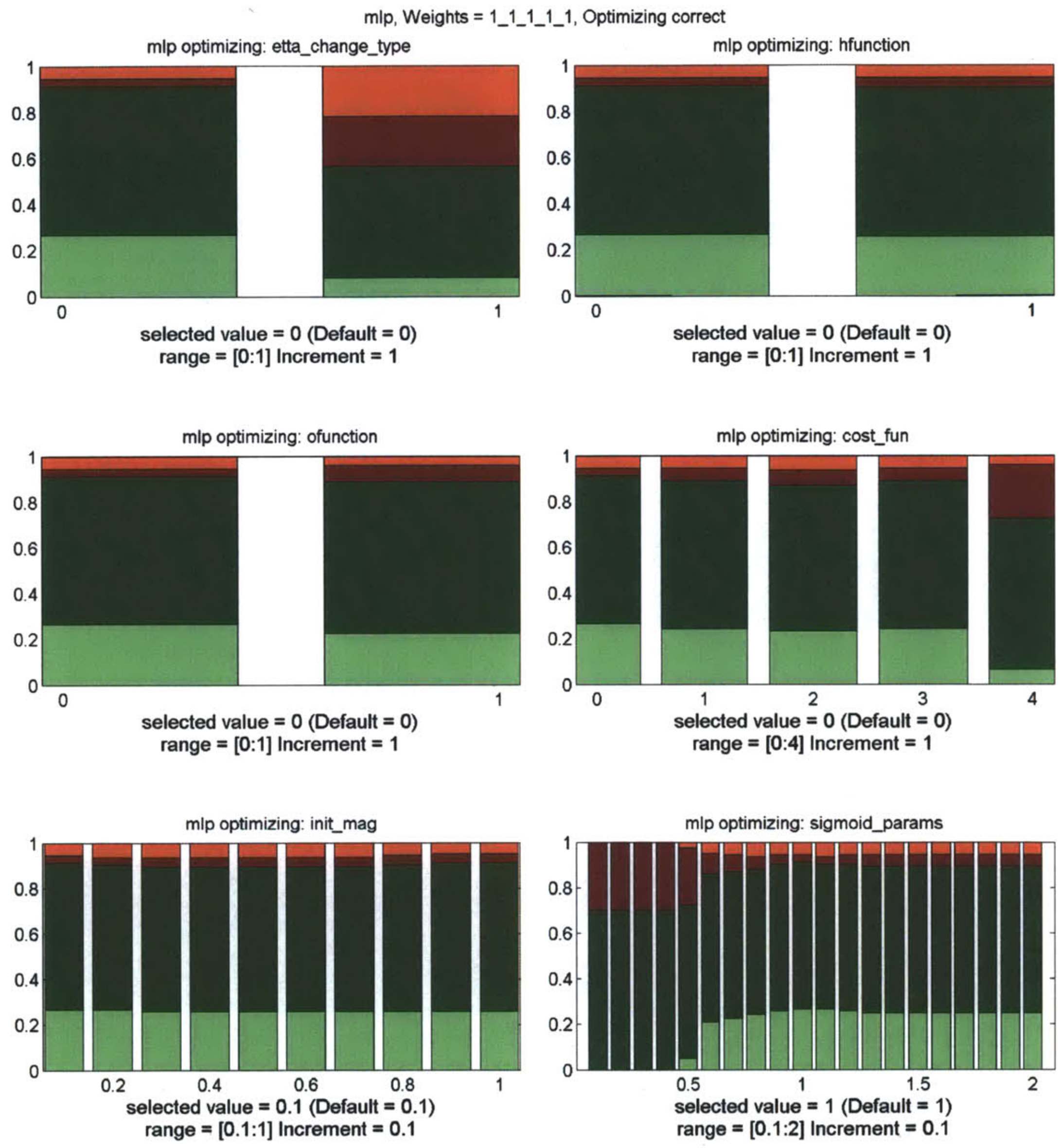


Figure 6-5: *(continued).*

6.2.4 Parzen Window

Tuning of the Parzen window classifier is summarized in **Figure 6-6**. This classifier places kernel functions over each training pattern. The classifier calculates the likelihood that an input sample is WGA or natural by summing across the likelihoods for each kernel function in the class and subsequently normalizing by the number of training patterns in the class. Kernel functions can be either Gaussians or rectangular pulse functions [13], as noted in the *linear* parameter. This parameter was set to 0 for Gaussian functions and 1 for the rectangular pulse function. The classifier performed better with Gaussian kernel functions, achieving a correct classification rate of 93%, compared to only 60% with rectangular pulse functions. The strong tendency to prefer Gaussian functions correlates with the preference for Gaussian parameters by the MLP classifier. The sigma value of this kernel, modelled by the *spread* parameter, strongly influenced performance, with sigma = 1.5 leading to optimal results. Higher values of sigma led to increased classifier complexity and ultimately to poorer results [13].

Additionally, Parzen kernel functions can be uniform (circular or square functions), or the length of each side can be proportional to the variance of each input feature (elliptical or rectangular). The *per_class* parameter regulated this and was set to 0 for uniform functions and 1 for feature-based functions. **Figure 6-6** demonstrates that uniform functions were strongly preferred. The preference for uniform kernel functions correlates with the preference of the Gaussian classifier discussed in **Chapter 5** for grand and full covariance matrices. The shape of the kernel was determined by the *per_input* parameter, which was set to 0 to indicate that all kernel functions have the same shape, or 1 to indicate separate kernel function shapes for each class. The value of this parameter did not strongly affect the performance of the classifier.

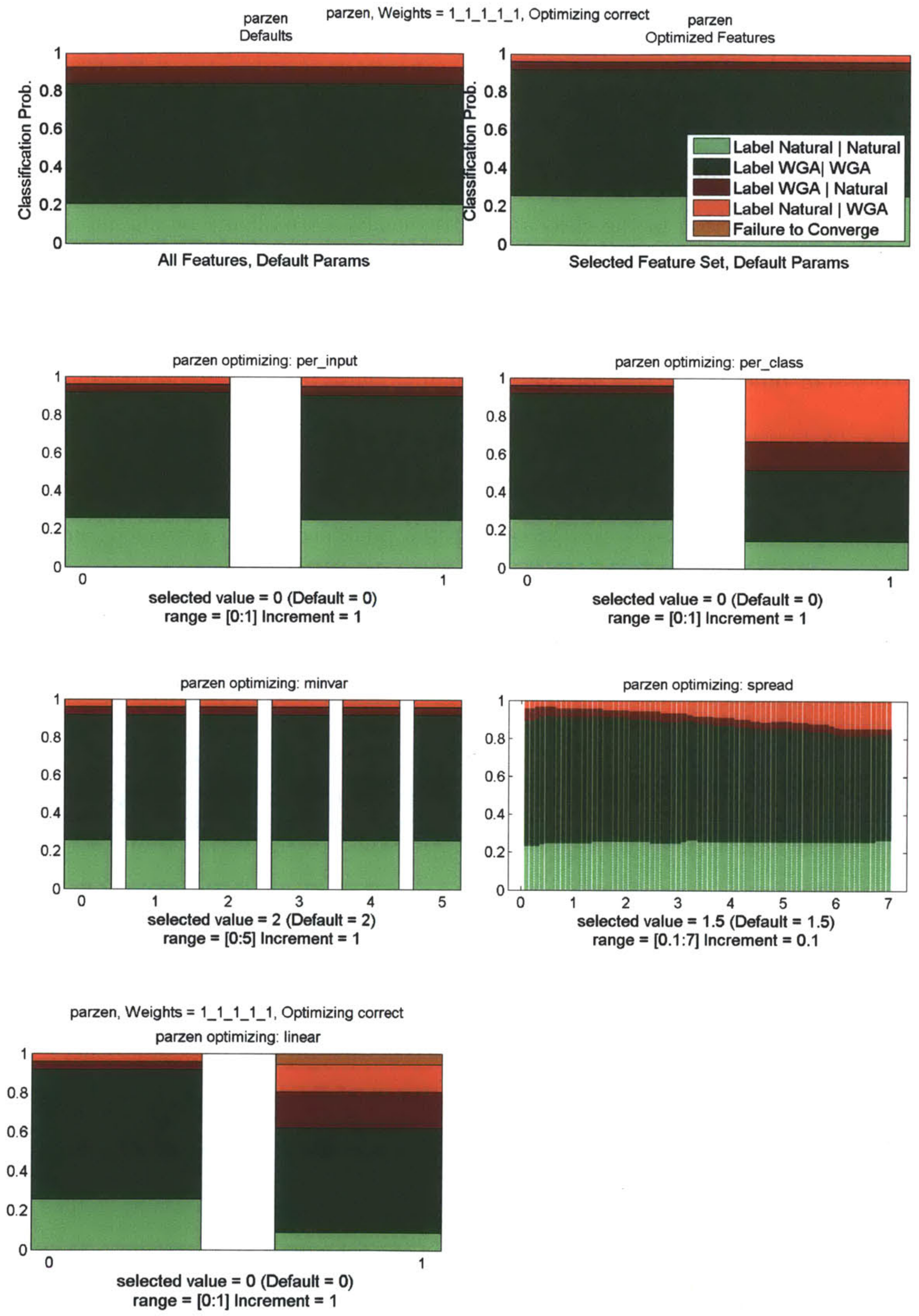


Figure 6-6: *Parzen classifier with parameters per_input, per_class, minvar, spread, linear.*

6.2.5 Support Vector Machine (SVM)

A support vector machine separates data into two or more classes using a hyperplane. The separating hyperplane is positioned to maximize the margin, which is defined as the separating distance from the patterns in two classes. The SVM implementation used by the LNKnet software utilizes sequential minimal optimization [23]. The algorithm begins with zero Lagrange multipliers. It then sweeps through all training patterns to find those where the Kuhn Tucker (KT) optimality condition is not satisfied. The KT requirement states that only support vectors can have non-zero dual variables [43]. The Lagrange multipliers are then adapted so that all patterns satisfy the KT conditions.

A support vector machine with a Gaussian kernel was used, and the performance of this machine depended on the Gaussian standard deviation of the kernel (*sigma*) as well as the Lagrange Multiplier Upper Bound (*cbound*). *Sigma* determined the width of the Gaussian kernel. **Figure 6-7** indicates that the classifier performed best when *sigma* was set to 4.9. This suggests that the SVM classifier had high generalizability, as the order of the optimal kernel was low. In general, higher order kernels reduce errors on the training data but do not generalize well. Conversely, low-order kernels generalize well, but tend to make errors on the training data. [43].

The parameter *cbound* controls the trade off between errors of the SVM on training data and margin maximization; $cbound = 0$ yields a hard margin SVM. If the *cbound* is chosen to be too large, the result is a high penalty for non-separable datapoints, which ultimately results in many support vectors and overfitting. If it is too small, underfitting will occur instead. The soft margin solution ($cbound > 0$) was used because the WGA and natural classes are not linearly separable, as can be observed from the feature scatter plots in **Appendix A**. The optimal value of *cbound* was 11. This value lay toward the center of the range, suggesting that neither underfitting nor overfitting were serious problems.

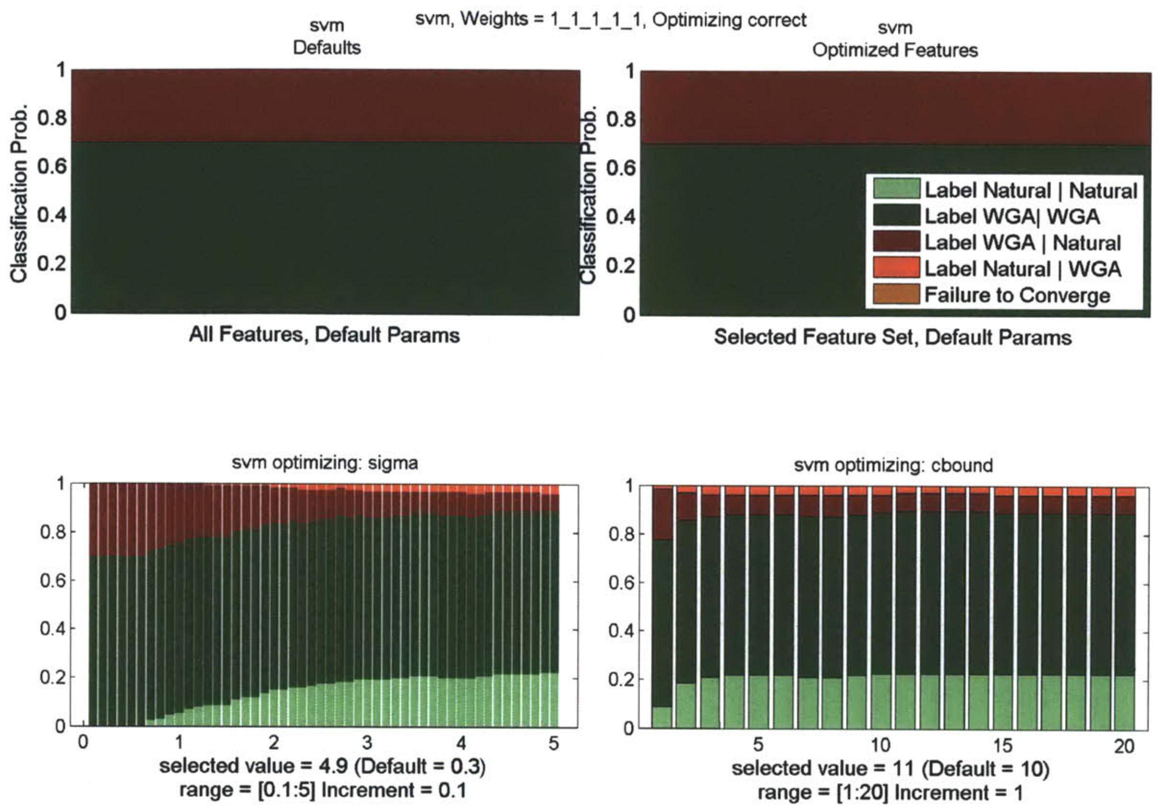


Figure 6-7: *SVM classifier with parameters sigma and cbound.*

6.3 Emphasis on Minimizing False Negatives for WGA Class

For purposes of forensic analysis, synthetic STR profiles are particularly noteworthy, as they bring into question the validity of forensic evidence. Thus, in this context, the performance score for a classifier should penalize the case $p(nat|wga)$ more severely than any of the other cases, as this denotes the probability that the classifier labels a WGA profile as Natural. Parameter tuning was thus performed for all classifiers using this performance score, which has a weight vector of [1,1,1,10,1]:

$$\text{performance} =$$
$$p(wga|wga) + p(nat|nat) - p(wga|nat) - 10 * p(nat|wga) - p(fail)$$

Note the higher weighting of the $p(nat|wga)$ term to emphasize the high cost of false positives for the natural class. The results are demonstrated in **Figure 6-8**. This figure includes the default performance, performance after feature selection, and performance after parameter tuning in columns one through three for each classifier. **Figure 6-9** includes only the tuned performance sorted in ascending order. This performance is compared to the tuned performance of all classifiers with the default weight vector [1,1,1,1,1] (**Figure 6-10**).

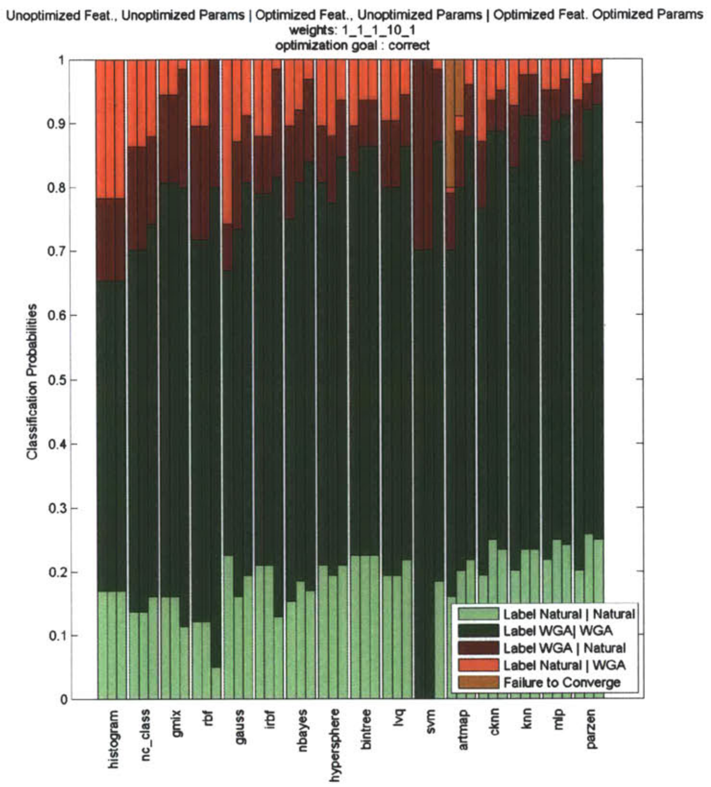


Figure 6-8: *Identifiler, weights = [1,1,1,10,1]. p(nat|wga) was assigned a cost 10 times higher than any other error. Column 1: default performance, column 2: feature selection, column 3: parameter tuning.*

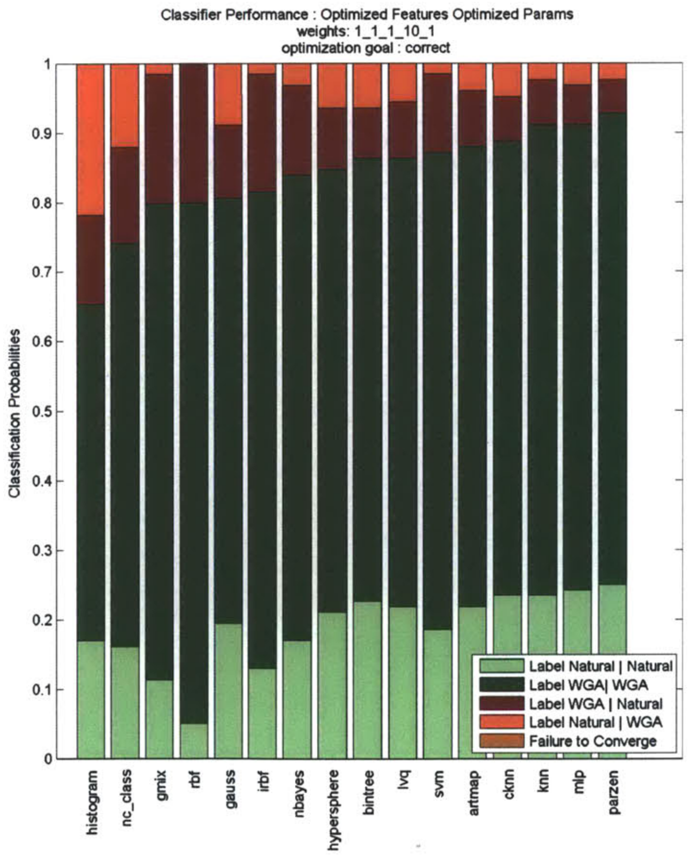


Figure 6-9: *Identifiler, weights = [1,1,1,10,1], tuned parameters. p(nat|wga) was assigned a cost 10 times higher than other errors.*

For the two sets of weight vectors, the combined area of the dark green bars (true positives) and light green bars (true negatives) is identical and equal to 13.41 units. This was calculated by summing the areas of all green bars (one set of bars per classifier). The area of the red and green bars must sum to one for each classifier (and consequently to 16 for the full set of classifiers examined). The combined red areas were consequently also the same for each set of weights, equal to 2.59 units. However, as indicated in **Table 6.2**, the area corresponding to $wga|wga$ is relatively higher for the classifiers with weights [1,1,1,10,1] compared to the classifiers with weights [1,1,1,1,1]. Similarly, the area corresponding to $nat|wga$ is cumulatively lower for classifiers with weights [1,1,1,10,1]. Thus, if performance is averaged across classifiers with weight vector [1,1,1,10,1], the overall probability of classifying a profile correctly doesn't change, but there is a higher likelihood of correctly classifying profiles that belong to the WGA class.

The same phenomenon is observed if the five high performing classifiers are analyzed separately. For both the [1,1,1,1,1] and [1,1,1,10,1] weightings, the top performing classifiers are KNN,CKNN, MLP, Parzen, and SVM. Combining the total green area across all classifiers yields a value of 4.52 for both sets of weights.This suggests that for the top five classifiers, as for the entire set of classifiers as a whole, the probability of classifying a profile correctly does not change when the weight vector is altered. However, the total dark green area $p(wga|wga)$ increases from 3.29 to 3.36 and the total light red area $p(nat|wga)$ decreases from 0.22 to .15 for the top performers with weights = [1,1,1,10,1]. This suggests a slightly higher probability of identifying a WGA profile correctly (**Table 6.2**), which may be of value in a forensics application.

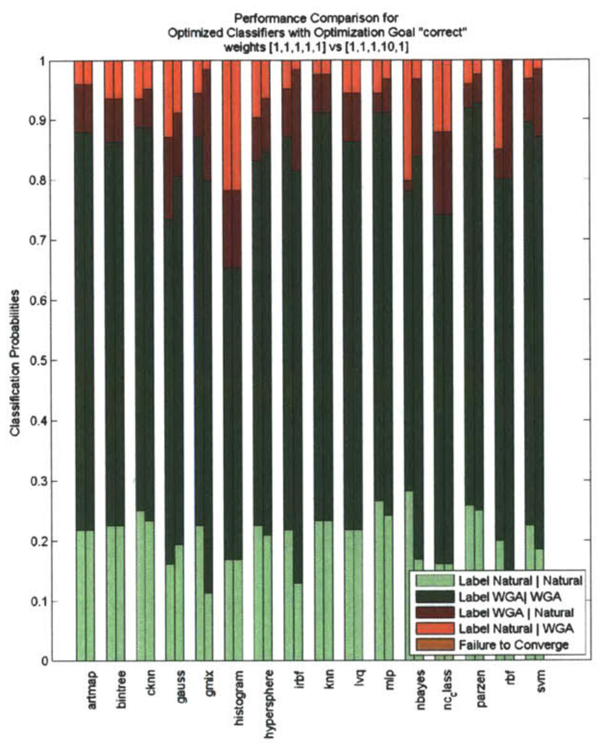


Figure 6-10: *Identifiler, weights = [1,1,1,1,1] compared with weights [1,1,1,10,1].*

	Area $nat\|nat$	**Area** $wga\|wga$	**Area** $wga\|nat$	**Area** $nat\|wga$
Sum of Classifiers with Weights =[1,1,1,1,1]	3.54	9.87	1.19	1.40
Sum of Classifiers with Weights =[1,1,1,10,1]	3.00	10.41	1.72	0.86
Sum of Top 5 with Weights = [1,1,1,1,1]	1.23	3.29	0.26	0.22
Sum of Top 5 with Weights = [1,1,1,10,1]	1.15	3.36	0.35	0.15

Table 6.2: *Comparison of classifier performance with w=[1,1,1,1,1] and [1,1,1,10,1] by profile category.*

6.4 Emphasis on Correctly Classifying Natural vs. Correctly Classifying WGA

In another experiment, the performance score was altered in two ways to place varying emphasis on correct classification of natural and WGA profiles. In the first approach, the w1 variable in the score was set to 3 to place heavier emphasis on the

$p(nat|nat)$ term, which denotes true positives for the natural class. w4 in the score was also set to 3 to place a heavier penalty on the $p(nat|wga)$, false positives for the natural class, yielding a weight vector of [3,1,1,3,1]. Thus, the modified performance score was

$$\text{performance} = 3^{*}p(nat|nat) + p(wga|wga) - p(wga|nat) - 3^{*}p(nat|wga) - p(fail)$$

In the second approach, the opposite was done: w2 and w3 were set to 3 to place greater emphasis on $p(wga|wga)$ and a higher penalty on $p(wga|nat)$, the false positive and false negative rates for the WGA class, yielding a weight vector of [1,3,3,1,1]. The score in this case was:

$$\text{performance} = p(nat|nat) + 3^{*}p(wga|wga) - 3^{*}p(wga|nat) - p(nat|wga) - p(fail)$$

Here, there is a higher emphasis on classifying WGA profiles correctly, and a higher penalty on false positives for WGA. The purpose of these two weightings was to explore the behavior of the classifiers when the importance of correct classification is class-dependent.

Additionally, by adjusting the weights for different subsets of parameters, the prior probabilities of classification can be altered. In the Identifiler data set, there are roughly twice as many WGA profiles as there are natural profiles. By weighing $p(nat|nat)$ higher and penalizing $p(wga|nat)$, the goal was to determine whether these unequal priors could be adjusted for in the performance score.

The classification results for the two sets of weights are presented in **Figures 6-11** and **6-12**. It can be concluded from these figures that the same set of classifiers that performed well with the weight vector [1,1,1,1,1] also perform well with the skewed weights. For both the [3,1,1,3,1] and [1,3,3,1,1], the top five performers included KNN, MLP, SVM. However, some unexpected results were observed: the Parzen classifier had been the top performer for the classifiers scored with weights [1,1,1,1,1].

This classifier was also the second best performer with weights [3,1,1,3,1], achieving a correct performance rate around 92%. However, it did quite poorly on the [1,3,3,1,1] dataset and classified only 75% of the sample profiles correctly. A reverse phenomenon was observed for the CKNN classifier: it classified 85% of the profiles in the [1,3,3,1,1] set correctly, but classified only 70% of the profiles correctly in the [3,1,1,3,1] set. This result is surprising because the Parzen classifier and the CKNN algorithm are both adaptations of the KNN algorithm, so it was expected that the he performance of one would correlate with the other [37]. However, as indicated above, the opposite effect was observed: CKNN performed well when Parzen performed poorly and vice versa.

Furthermore, the feature selection algorithm did not improve performance for either the [1,3,3,1,1] classifiers nor the [3,1,1,3,1] classifiers as significantly as for the default case with weights [1,1,1,1,1]. In the default case, feature selection led to a slight drop in performance for the hypersphere classifier, did not significantly change the performance of six of the classifiers, and significantly improved the performance of the nine remaining classifiers (**Figure 6-1**). However, for the [1,3,3,1,1] case, feature selection led to a performance loss for four of the classifiers, no change in performance for six classifiers, and a performance gain for 6 of the classifiers (**Figure 6-12**). Thus, a net performance gain was still observed, but it was not as pronounced as for the default case. Similarly, when feature selection was used with the weight vector [3,1,1,3,1], there was a performance decline for three of the classifiers, no change for seven of the classifiers, and an improvement for the remaining six (**Figure 6-11**).

In conclusion, the feature selection algorithm is not robust to different weight vectors used to score classifier performance. This algorithm was developed with the default set of weights [1,1,1,1,1], and leads to a high performance improvement for most classifiers with this weight vector. However, the performance gain is much lower when other weight vectors are used to score classifier performance.

Finally, it is important to note that for all three weight vectors considered, the classifiers had nearly identical default performance. Among the three sets of weight vectors, the sum of the areas of the bars corresponding to $p(WGA|WGA)$ was in the range [2.78,2.79], the sum of the bars for $p(nat|nat)$ was in the range [9.44,9.45], the sum of the bars for $p(wga|nat)$ was in the range [1.63,1.64], and the sum of the bars for $p(nat|WGA)$ was equal to 0.2016. This serves as a check for classifier development: classifiers are expected to perform equally well with default feature and parameter sets since the influence of the weight vector isn't expressed until the feature selection step.

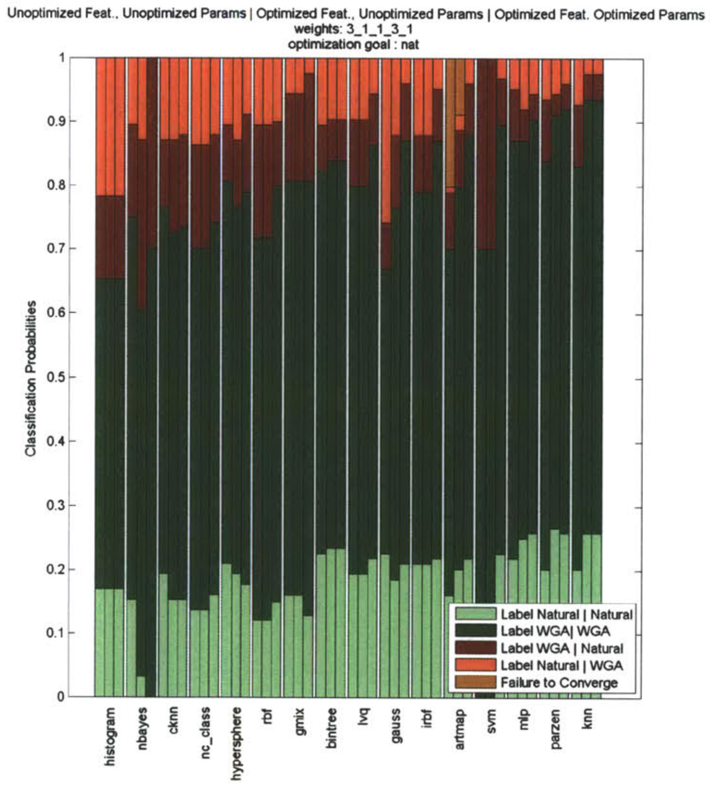


Figure 6-11: *Identifiler, weights = [3,1,1,3,1], features optimized for natural profile classification.*

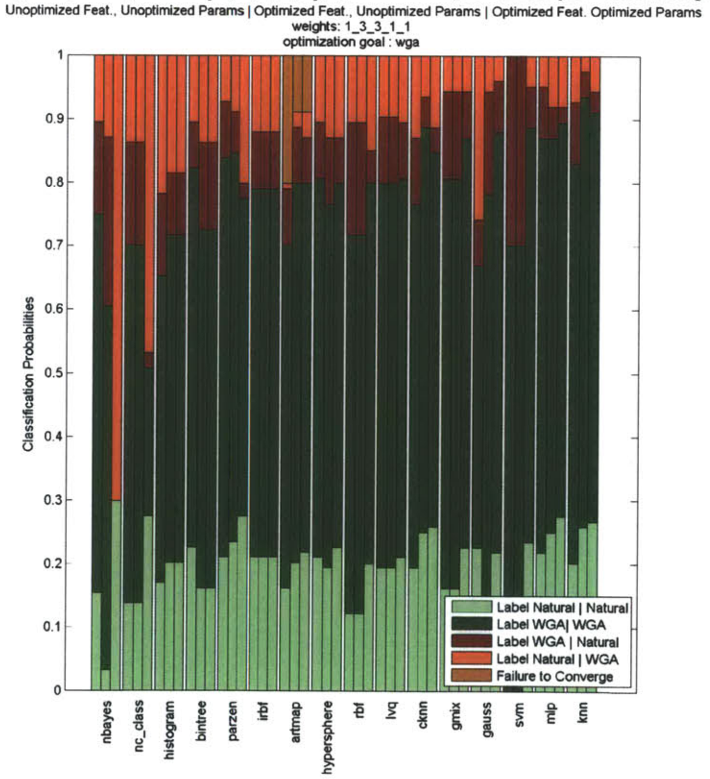


Figure 6-12: *Identifiler, weights = [1,3,3,1,1], features optimized for WGA profile classification.*

The tuned performance of classifiers with weight vector [1,3,3,1,1] and classifiers with weight vector [3,1,1,3,1] is compared in **Figure 6-13** and **Table 6.3**. From this data, it appears that the cumulative probability of correct classification across all 16 classifiers is slightly higher for the [1,3,3,1,1] weight vector than the [3,1,1,3,1] vector (the sum of the green bars is 13.21 for the former and 12.31 for the latter). This difference in overall performance is fairly small, and even smaller when only high-performing classifiers are considered. As expected, the [1,3,3,1] classifiers are better at classifying WGA profiles, while [3,1,1,3,1] classifiers are better at classifying natural profiles. The performance difference between [1,3,3,1] and [1,1,1,1,1] classifiers is higher than between [3,1,1,3,1] and [1,1,1,1,1] classifiers. This can be explained by the fact that WGA profiles take on a higher range of feature values than natural profiles, a difference that is amplified by the [1,3,3,1,1] weighting.

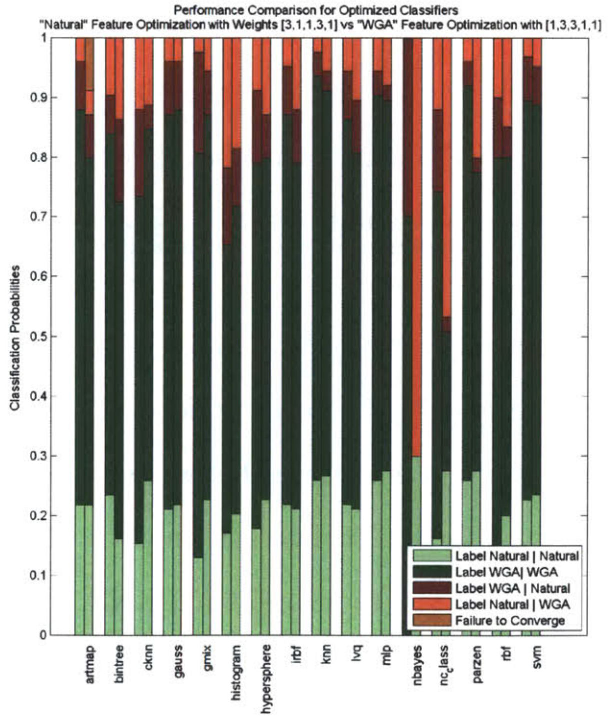


Figure 6-13: *Identifiler, comparison between classifiers optimized for identifying natural profiles (weight vector [3,1,1,3,1]), and classifiers optimized for identifying WGA (weight vector [1,3,3,1,1]).*

	Area $nat\|nat$	**Area** $wga\|wga$	**Area** $wga\|nat$	**Area** $nat\|wga$
Sum of Classifiers with Weights =[3,1,1,3,1]	3.04	10.17	1.69	1.11
Sum of Classifiers with Weights =[1,3,3,1,1]	3.75	8.56	0.97	2.63
Sum of Top 5 with Weights = [3,1,1,3,1]	1.15	3.23	0.34	0.27
Sum of Top 5 with Weights = [1,3,3,1,1]	1.31	3.01	0.19	0.50

Table 6.3: *Comparison of classifier performance with w=[3,1,1,3,1] and [1,3,3,1,1] by profile category.*

6.5 Machine Learning on the PowerPlex STR Typing Kit

The full process of feature selection and parameter tuning was repeated for the STR PowerPlex kit. Fewer samples were available for PowerPlex analysis compared to Identifiler (**Table 2.1**), so the results should be considered preliminary. However, the results illustrate the degree to which the techniques of feature selection and parameter tuning can be generalized, and are summarized in **Figures 6-14** and **6-15**. Overall, the classification algorithms were less likely to classify PowerPlex samples correctly than to classify Identifiler samples correctly. 10 of the 16 classifiers had error rates below 20%. The Gaussian and MLP classifiers performed particularly well on the PowerPlex data set, as both had error rates below 10%. However, of the five high performing classifiers for the Identifiler data set, only the MLP and Parzen were in the top five for the PowerPlex set, though CKNN, KNN, and SVM achieved error rates below 18 percent and were ranked 6th, 7th, and 8th respectively.

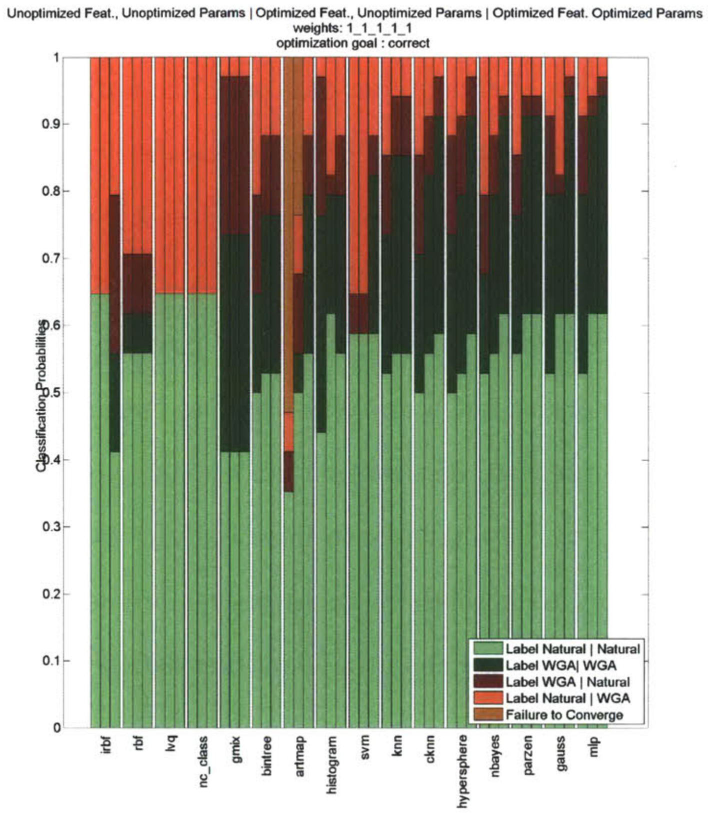


Figure 6-14: *PowerPlex, natural vs WGA, weights = [1,1,1,1,1], column 1: default performance, column 2: performance after feature selection, column 3: performance after parameter tuning.*

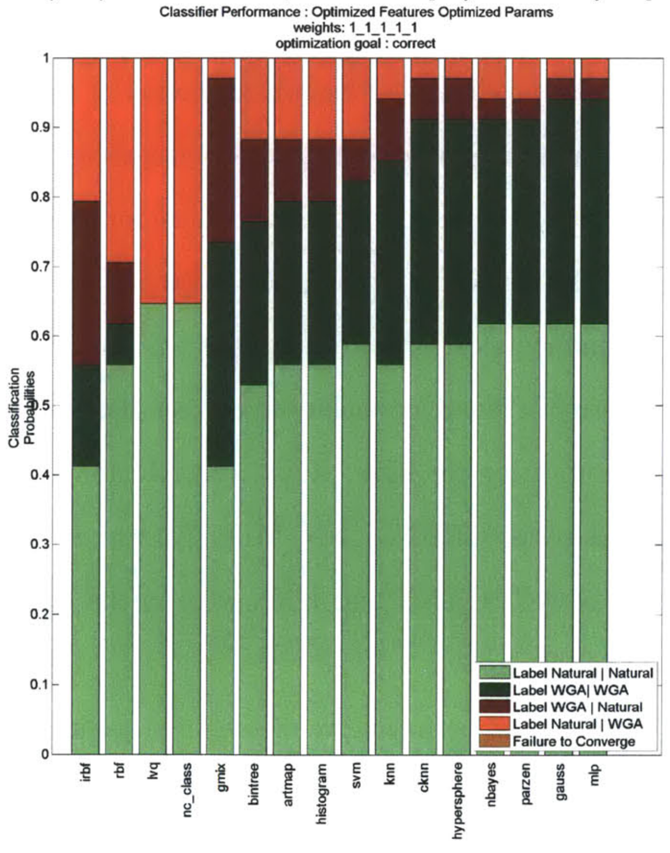


Figure 6-15: *PowerPlex, natural vs. WGA, weights = [1,1,1,1,1]. Performance after parameter tuning.*

Several of the classifiers were more likely to classify a profile as natural for the PowerPlex data set than for the Identifiler data set. The radial basis function, linear vector quantizer, and nearest cluster algorithms classified profiles as natural more than 90% of the time. Two factors may explain this trend. First, the Powerplex data set had a 3:2 ratio of natural profiles to WGA profiles. This unequal distribution of data likely biased the classifiers in favor of natural profiles. The reverse effect was observed for the Identifiler dataset, were WGA profiles outnumbered natural profiles by a ratio of 5:4. Since the data distribution among the two classes is more skewed for the PowerPlex dataset, the tendency to misclassify a WGA profile as natural is higher than the tendency to misclassify an Identifiler profile as WGA. Second, the PowerPlex dataset is much smaller than the Identifiler data set, so worse performance is expected.

6.6 Distinguishing Natural Profiles from Bacterial Clones

Two types of artificial profiles were examined, bacterial synthetic, and bacterial cloned. **Chapter 1** summarizes the bacterial cloning process used to obtain these. Though the results must be reproduced on a larger size for higher credibility, preliminary performance data are presented in **Figures 6-16** and **6-17**. In this preliminary study, classifier performance scores where high, but this may be an artifact of small sample size. The majority of classifiers achieved correct classification rates near 90%, with 100% correct classification for RBF, KNN, and ARTmap. The high-performing classifiers from previous tests (CKNN, KNN, MLP, Parzen, SVM) performed well on the bacterial samples as well: CKNN and KNN were in the top five, and the others followed closely in 6th-8th place.

Figure 6-18 compares the performance of the 16 classifiers studied on the Identifiler data, the PowerPlex WGA dataset, and the PowerPlex bacterial dataset. The figure demonstrates that, with a few exceptions, most classifiers performed better on the Identifiler data rather than the PowerPlex data when comparing natural profiles to WGA. Additionally, for most classifiers, performance was highest on the PowerPlex bacterial dataset compared to both WGA datasets. This difference in performance may indicate that profiles derived via bacterial cloning are less similar to natural profiles than are WGA-derived profiles. However, the difference in performance may also be an artifact of the small sample size of bacterial clones. More bacterial cloning data must be analyzed to determine which of these explanations is legitimate.

Additionally, despite performance differences among the three datasets, for all three datasets the CKNN, SVM, MLP, Parzen, and SVM classifiers performed well, each achieving error rates around 10%.

In summary, for the majority of the 16 machine learning algorithms analyzed in this work, classification performance was significantly improved by tuning classifier parameters via a greedy gradient ascent approach. Most classifiers performed reasonably well; all achieved error rates below 35% after tuning had been performed. Five classifier outperformed the rest: the Parzen window, multi-layer perceptron, support vector machine, K-nearest neighbors, and condensed K-nearest neighbors all achieved error rates near 10%. Altering the weight vectors from the default value of $[1, 1, 1, 1, 1]$ to emphasize particular classification outcomes did not significantly effect overall classifier performance. For example, heavily emphasizing false negatives for the WGA class $p(nat|wga)$ did not change overall classifier performance, though the error due to false negatives for WGA was relatively smaller as compared to the same error for the classifiers with weight vector $[1, 1, 1, 1, 1]$. Using sample data generated with the PowerPlex kit as well sample data generated by bacterial cloning suggests that the results obtained with Identifiler data generalize fairly well to other datasets.

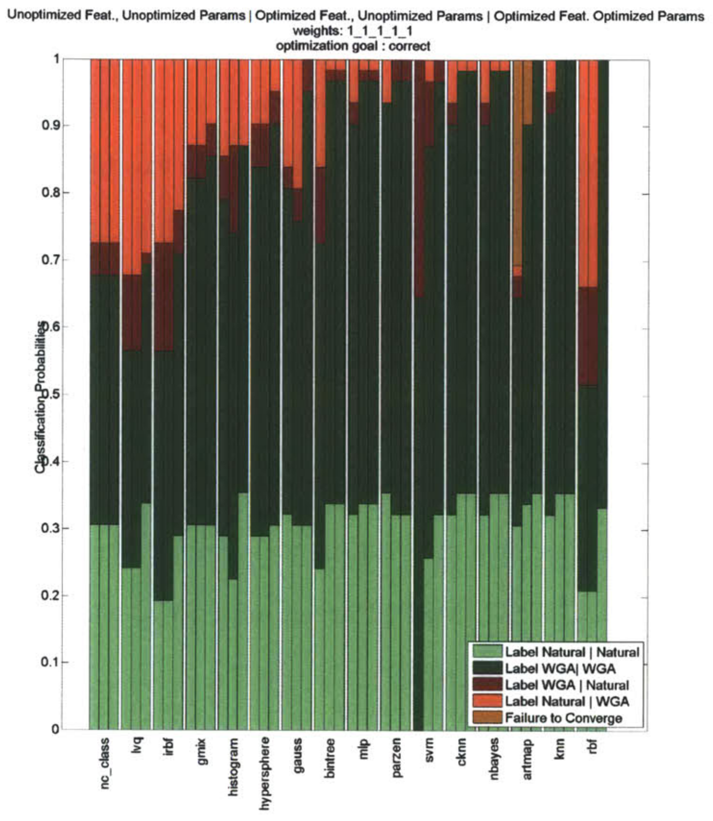


Figure 6-16: *PowerPlex, natrual vs. bacterial cloned/synthetic, weights = [1,1,1,1,1], column 1: default performance, column 2: feature selection, column 3: parameter tuning.*

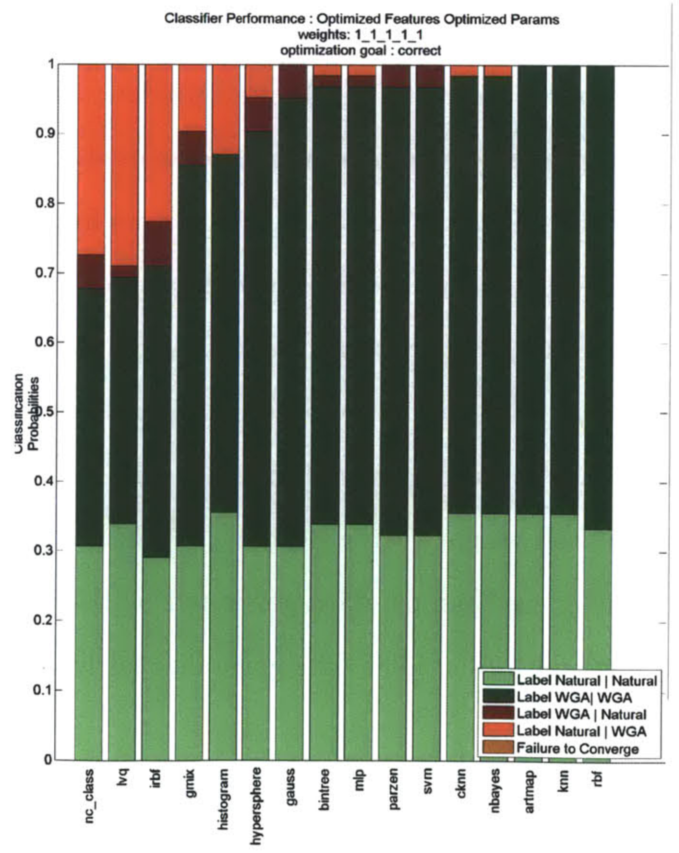


Figure 6-17: *PowerPlex, natural vs bacterial, tuned parameters, weights = [1,1,1,1,1].*

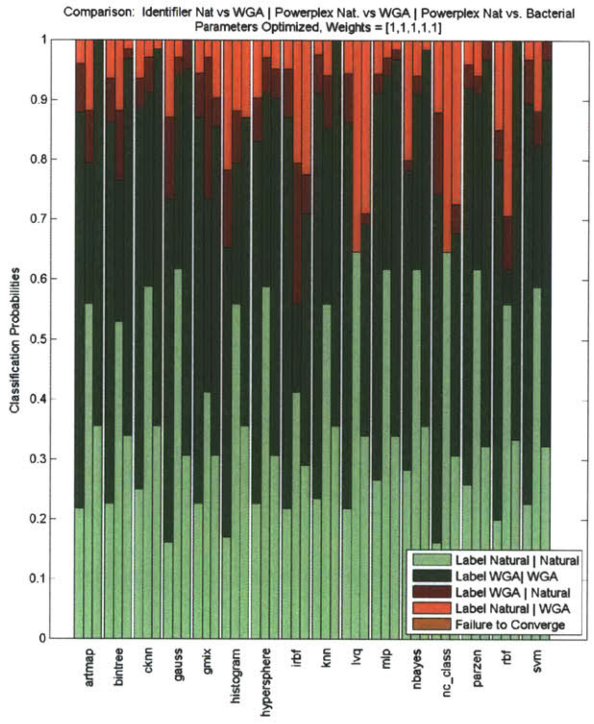


Figure 6-18: *Parameters optimized, weights = [1,1,1,1,1]. The three columns in each bar represent Identifiler natural vs. WGA, PowerPlex natural vs. WGA, PowerPlex natural vs. bacterial.*

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 7

Committee Classifiers

A single classifier often does not provide the best performance. In many cases, better performance is attained by a committee. Committees may consist of different types of classifiers. Alternatively, they may be built with one type of classifier but using different samplings of the training data. Committees improve performance because averaging several high-performing classifiers cancels out the biases inherent in each. Furthermore, some classifiers, such as binary trees, are susceptible to noise, and averaging across many trees by building a random forest can reduce variance while not introducing additional bias [29]. If VC dimension theory is used to quantify the complexity of a classifier committee, it is often the case that the VC dimension of a committee with many members is small if all the members combine to minimize a global loss function [42].

Consequently, several committee classifiers were used to combine the outputs of the trained classifiers described in **Chapter 5** to return a final combined classification decision. Committee classification results indicated that improvements were greatest for uncorrelated classifiers with large variance.

7.1 Committee Generation by Stacking

Stacking refers to combinations of heterogenous classifiers [42]. This technique was performed on various combinations of the classifiers described in **Chapter 3**. The same training data (124 Identifiler profiles) and output classes (WGA, natural) were used for all committee members. All members used features at the profile level, but the actual features for each member were determined by the feature selection algorithm described in **Chapter 4** and varied among the classifiers. To form committees, all member classifiers were tuned as described in **Chapter 3**, using four-fold cross validation. Each was then trained and tested independently on the Identifiler profile data: the data was separated into training (60 percent), evaluation (20 percent), and test (20 percent) samples. The resulting outputs were concatenated via the LNKnet software and used as an input to a higher level committee classifier. Three types of committee classifiers were formed by taking a majority vote, mean result, and median result of the constituent members.

To form the committees, the classifiers were separated by type (**Table 7.1**). This was done to ensure that average and median comparisons were performed on compatible datasets. For example, for the mean and median classifiers, it was necessary to ensure that all constituent classifiers estimate either posterior probabilities or likelihoods. Furthermore, nearest neighbor classifiers were excluded from the mean and median committees because they do not produce continuous outputs.

Posterior Probability	Likelihood	Rule-Based	Nearest Neighbor	High Performers
MLP	Gaussian	Binary Tree	KNN	KNN
RBF	Gmix	SVM	CKNN	CKNN
IRBF	Histogram	Hypersphere	LVQ	MLP
ARTmap	Naive Bayes		Nearest Cluster	Parzen
	Parzen			SVM

Table 7.1: *Six committees were formed by taking different combinations of individual classifiers. In addition to the five committees presented, a sixth committee was formed by taking a majority vote of all the classifiers.*

7.1.1 Majority Vote Results

For each of the six committees in **Table 7.1**, a majority vote was taken among the constituent members. That is, the results of the individual classifiers were compared, and a majority vote among these classifiers determined the class of each test profile. As indicated in **Figure 7-1** this technique did not lead to improved performance. A majority vote among the nearest neighbor classifiers actually led to poor performance, as the committee concluded that all test profiles belonged to the natural class (only light red and light green bars are present in the figure). The highest-performing committee was formed by taking a majority vote among the top five performing classifier. This committee is further summarized below.

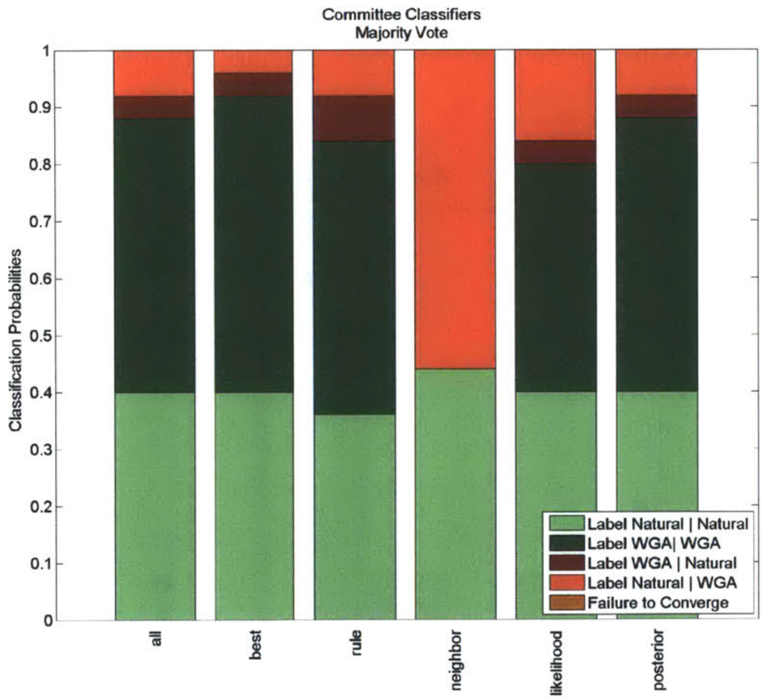


Figure 7-1: *Majority vote committees.*

In addition to analyzing overall committee performance it is useful to analyze the number of times each test profile was misclassified. This data is presented in **Figure 7-2**. The question of interest is whether different committees are likely to misclassify the same profiles, or, alternatively, whether the errors made by individual committees are uncorrrelated. The figure suggests that most test profiles were misclassified by one or two committees, pointing to the conclusion that errors made by the different committees are uncorrelated. However, two of the test profiles were misclassified by five committees, and one was misclassified by all six committees, suggesting that a profile may have characteristics that make it more susceptible to misclassification. The raw profile data for profile number 15, a natural profile that all six committees classified as WGA, is presented in **Figure 7-3**. For comparison, **Figure 7-4** presents a natural profile from the same testing set that was classified correctly by all of the majority vote committees. Visually, there do not appear to be major differences between the feature values for the two profiles, so it is not trivial to conclude why one performs well in testing and the other performs poorly. Further investigation of the classifier performance is necessary to correlate specific profile properties with the probability of misclassification.

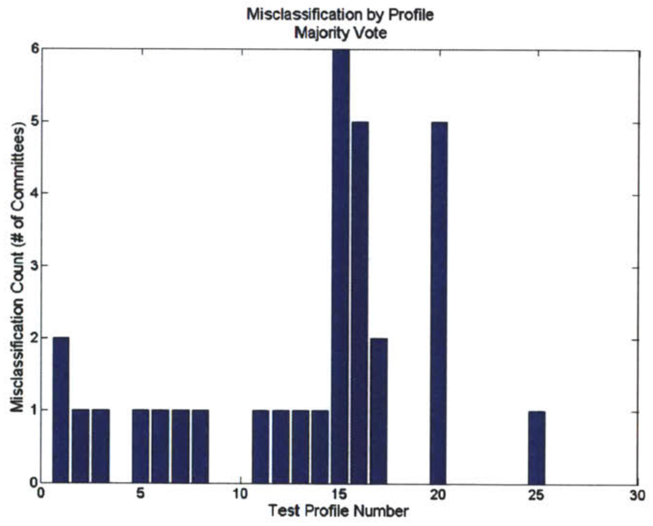


Figure 7-2: *Majority vote committee test data misclassifications.*

7.1.2 Majority Vote of High-Performing Classifiers

As indicated by Table 7.2 and Figure 7-5, taking a majority vote does not improve performance when compared to the five top-performing classifiers individually. When performing four-fold cross validation on 124 Identifiler profiles, taking a majority vote

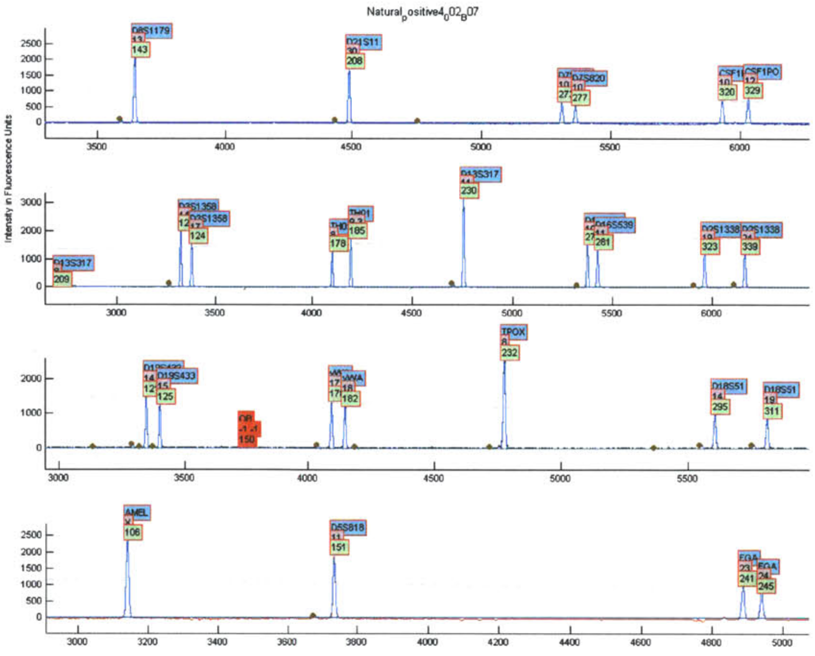


Figure 7-3: *Natural profile that was classified as WGA by all six majority vote committees.*

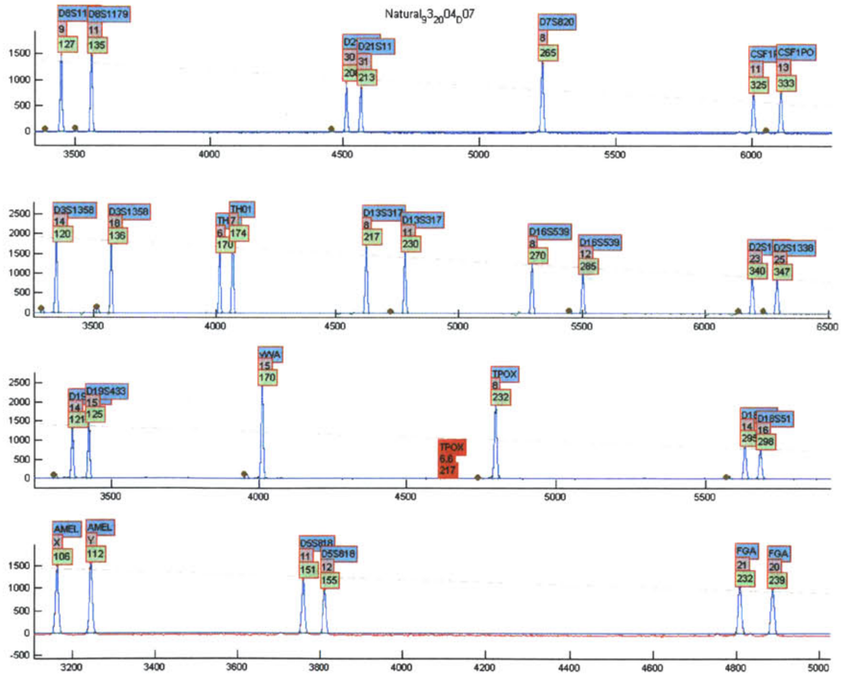


Figure 7-4: *Natural profile that was classified correctly by all six majority vote committees.*

among the classifiers led to a combined error rate of 0.081, equivalent to the error rate for the Parzen classifier. Other metrics, such as the rate of false positives $p(wga|nat)$ and the rate of false negatives $p(nat|wGA)$ were also comparable for the individual top-performing classifiers and the majority vote. For example, the majority vote gave $p(wga|nat) = 0.1622$ and $p(nat|wga) = 0.0460$, equal to the values given by the CKNN and SVM classifiers. These values are within once percent of the values obtained by the overall highest-performing Parzen classifier $p(wga|nat) = 0.1622$, $p(nat|wga) = 0.0575$. These statistics suggest that using a committee formed by combining the decisions of the best classifiers in a simple vote does not significantly improve performance. It is possible that more sophisticated approaches, such as boosting classifier performance with the Adaboost algorithm, would yield more fruitful results.

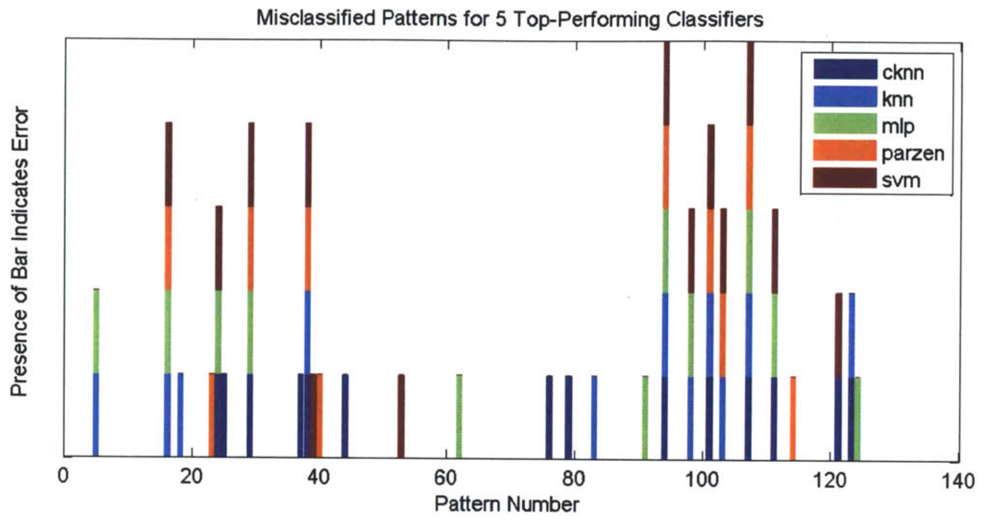
majority vote	**0.9194**	**0.8378**	**0.9540**	**0.1622**	**0.0460**
Classifier	**p(correct)**	**p($nat|nat$)**	**p($wga|wga$)**	**p($wga|nat$)**	**p($nat|wga$)**
cknn	0.8871	0.8378	0.9080	0.1622	0.0920
svm	0.8952	0.7568	0.9540	0.2432	0.0460
knn	0.9113	0.7838	0.9655	0.2162	0.0345
mlp	0.9113	0.8919	0.9195	0.1081	0.0805
parzen	0.9194	0.8649	0.9425	0.1351	0.0575

Table 7.2: *Majority vote among high-performing classifiers: CKNN, KNN, MLP, Parzen, SVM.*

7.1.3 Average of Classifier Results

Figure 7-6 demonstrates the performance of three committees that were formed by averaging the results of rule-based classifiers, likelihood classifiers, and posterior probability-based classifiers. None of these committees was able to achieve an error rate below 14%. This is a higher error rate than that of the individual constituent members: the likelihood committee includes the Parzen classifier, which by itself was able to achieve an error rate of only 7%. However, the likelihood committee as a whole had an error rate of nearly 40%. These results suggest that, for the STR profile authentication problem, averaging classifier results is not beneficial.

Figure 7-7 illustrates the number of errors made by the average results committee on each of the test profiles. As for the majority vote committee, the errors appear to be well distributed among the test data: most profiles were classified incorrectly by



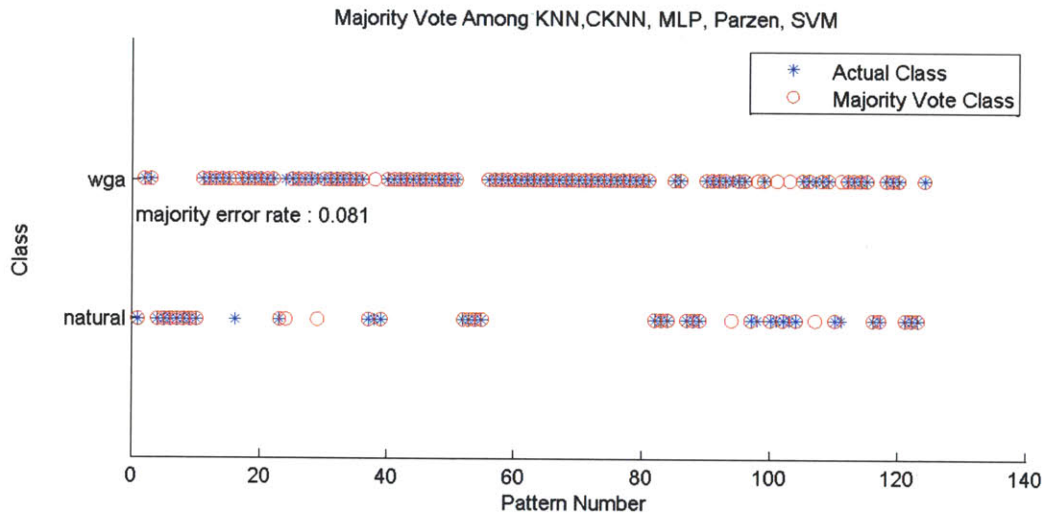


Figure 7-5: *Majority Vote of Five Top-Performing Classifiers: CKNN, KNN, MLP, Parzen, SVM*

only one of the three committees. However, two of them, profile 16 and profile 20, were classified incorrectly by all 3 committees. It is interesting to note that profile 20 was classified correctly by all 6 of the majority vote committees. This indicates that taking a majority vote among the component classifiers produces a different set of errors than taking the average of the classifier results.

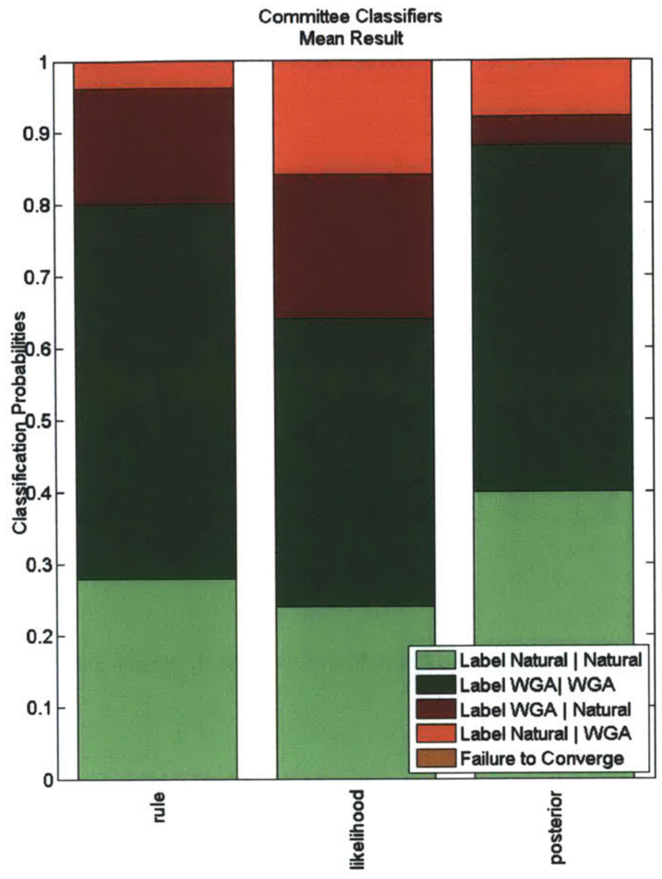


Figure 7-6: *Committees formed by averaging individual classifier results.*

7.1.4 Median of Classifier Results

A committee was formed by taking the median result of the individual classifier decisions. The committee decision was formed by ordering class probabilities for all the component classifiers and choosing the class that corresponded to the median probability. Though the median committees performed slightly better than the corresponding average-based committees, their performance was still no better than that of the best individual member.

The error distribution among the test profiles is different for the median committees than for the majority vote and average-based committees. As illustrated in **Figure 7-9**, most profiles that were misclassified were misclassified by at least two of the three committees, suggesting a higher clustering of errors than for the other two

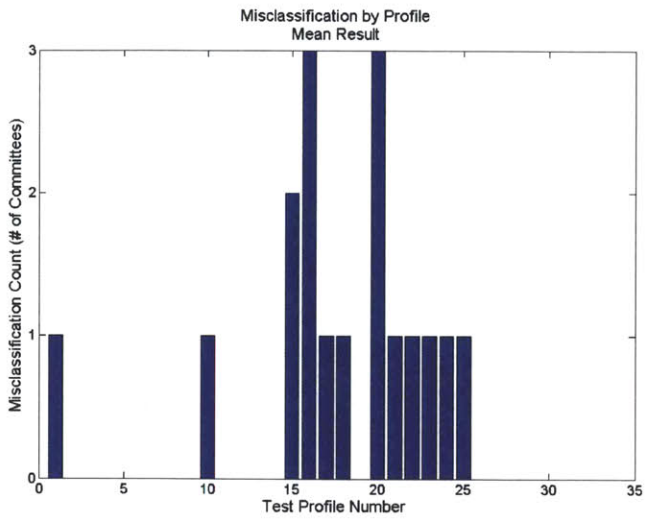


Figure 7-7: *Mean committee test data misclassifications.*

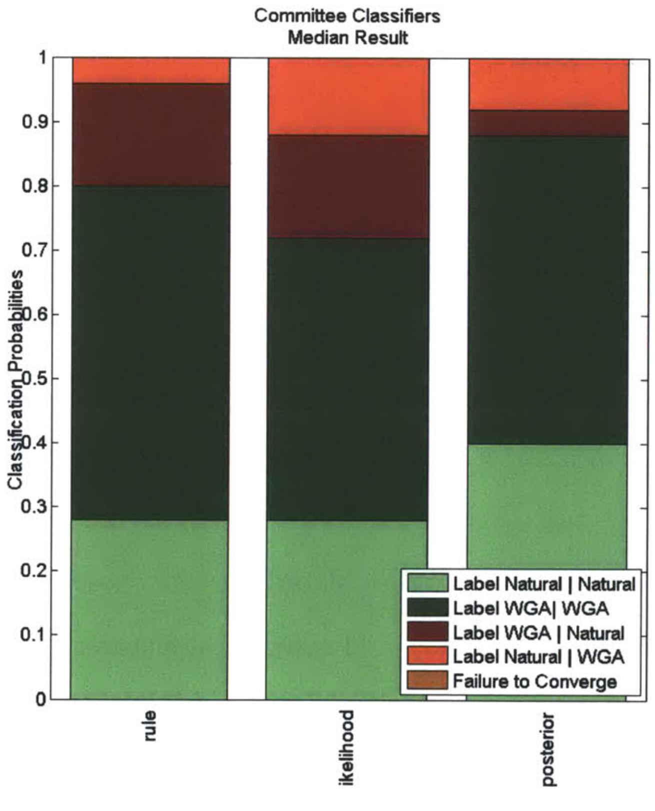


Figure 7-8: *Committees formed by taking the median of individual classifier results.*

committees.

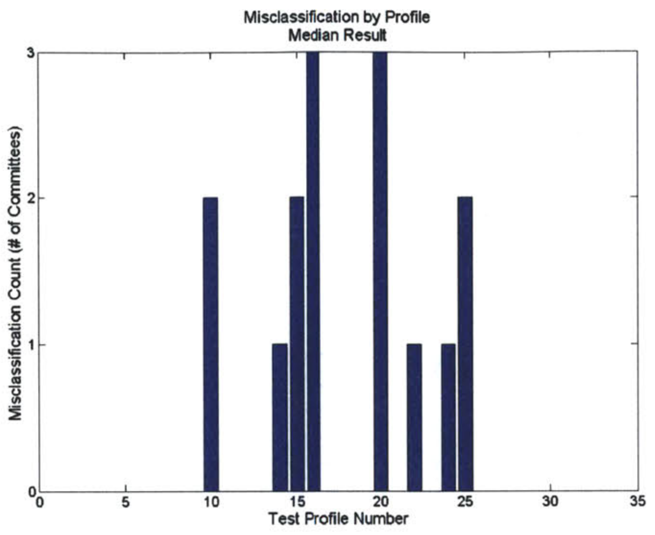


Figure 7-9: *Median committee test data misclassifications.*

7.2 Random Forest

A random forest classifier was constructed as another approach to forming committees to reduce classification error. A random forest consists of a number of binary trees whose classification decisions are averaged together to produce an aggregate decision. Binary trees have high standard deviation (and consequently a high variance) in their output, but the standard deviation can be reduced by averaging the results of many trees in a forest, which in many cases leads to a lower probability of misclassification [23].

A second reason the random forest algorithm was chosen is its effectiveness in performing regression on both binary and categorical features. Though most features in the examined dataset are continuous, some can take on only discrete values and are best treated as categorical for purposes of classification. These include the stutter count, the off-ladder outside bin count, and the off-ladder inside bin count, which were empirically observed to take on a limited range of discrete values.

The MATLAB Machine Learning Toolkit was used to construct binary trees, which were then used as an input to the TreeBagger Bootstrap aggregation algorithm to build ensembles of decision trees. Each tree in this ensemble was grown on an independently-drawn bootstrap replica of the input data. Bootstrap sampling was used to make the component trees uncorrelated. This process involved sampling with replacement from the full set of Identifiler profiles to create uniformly sized training sets. Each set contained 60 percent of the data (75 samples), selected randomly. The unselected samples in each set were used for testing and evaluation, and are referred to as "out-of-bag" observations. Thus, the term "out-of-bag classification error" refers to the probability of misclassification on the test data. The bootstrap-sampled training sets were used to construct one hundred binary trees, each of which was built to maximum size to reduce bias [13].

The TreeBagger algorithm was then used to compute the forest classification decisions. Trees were added to the forest one at a time at each iteration of the algorithm, with the goal of discovering the optimal forest size. **Figure 7-10** compares the average out-of-bag error of an individual tree in the forest with the out-of-bag error of the forest as a whole. The first metric, illustrated by the red asterisk symbols in the figure, was obtained by classifying the test samples with each tree in the forest and calculating the out-of-bag error for each individual tree. These error metrics where then averaged. The result constituted an estimate for the out-of-bag error of a representative tree in the forest. The cumulative out-of-bag error, on the other hand, is represented by a solid red line in the figure. This metric was obtained by using each tree in the forest to classify the test data, but with the added step of averaging the classification decisions of the individual trees to produce a single aggregate classification decision. The out-of-bag error was then calculated based only on the aggregate classification decision (not on the decisions of the individual trees).

As the size of the forest was increased, the error estimates for a single representative tree centered around a mean of .20 (20% of the out-of-bag profiles were misclassified), with a standard deviation of 0.0284 (**Table 7.3**). There was no corre-

Calculation Method	Standard Deviation
Individual,Non-Pruned	0.0284
Individual,Pruned	0.0298
Cumulative, Non-Pruned	0.0092
Cumulative, Pruned	0.0201

Table 7.3: *Standard deviation in out-of-bag error for individual trees and forests.*

lation between forest size and the out-of-bag error of a representative tree: the error of the single tree in a forest of size one was 0.17. The error of a representative tree in a forest of size 90 was also 0.17. However, the out-of-bag error of the forest as a whole followed a different trend, decreasing exponentially as the size of the forest increased. It was at its highest value (0.17) for a forest of size one. As trees were added, the forest out-of-bag error declined exponentially until the forest reached a size of 20 trees. Adding more trees lead to a slow decline in error; once the forest reached a size of 80 trees, the out-of-bag error plateaued at 11%, and adding additional trees had little effect. The overall standard deviation of the forest out-of-bag error was 0.0092, an order of magnitude lower than the standard deviation of a single representative tree in the forest. Thus, the advantages of a forest committee classifier over a single representative tree are two-fold: a reduced out-of-bag error rate as well as a lower standard deviation in the error rate.

Figure 7-10 also analyzes the effect of pruning on random forest classifiers. Blue asterisks represent individual pruned trees and a solid blue line represents the pruned forest. The trees were pruned via Chi-square pruning, an algorithm that determines whether a node is statistically relevant to the classification during tree induction. The literature suggests that pruning should be avoided for random forests because tree nodes with few cases are unlikely to pass the significance test. This causes trees to become over-pruned and consequently leads to poor generalization [29]. **Figure 7-10** supports this theory, as the out-of-bag error for the unpruned forest was consistently lower than the error for the pruned forest.

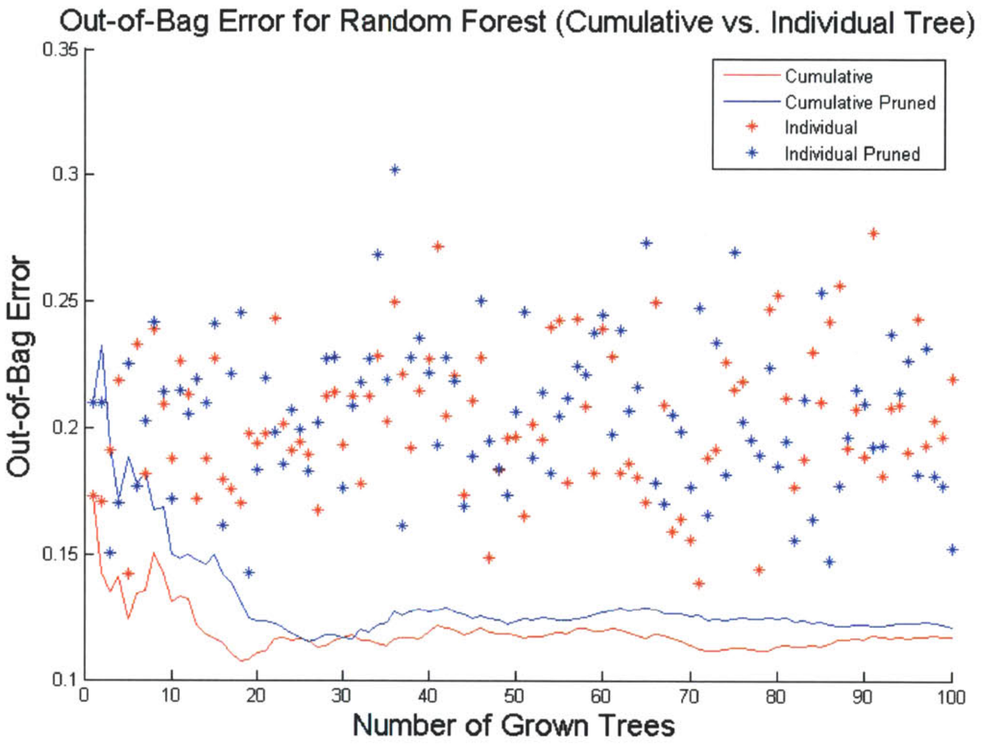


Figure 7-10: *Random forest out-of-bag error: individual trees compared with cumulative forest.*

Ultimately, the committees explored in this work were found to provide no significant performance improvement over individual high-performing classifiers. Of the multiple approaches to committee construction that were examined, the only one that yielded an error rate below 10 percentage points was the majority vote among the high performing classifiers (**Figure 7-2**). This committee had a misclassification rate of 8.01%, only one percentage point lower than the constituent members. Since one of the aims in machine learning is to reduce classifier complexity, the added complexity of committee classifiers formed via stacking techniques and the random forest approach does not outweigh the very slight performance gain.

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 8

Project Extensions

The results of this effort to authenticate STR profiles via machine learning techniques suggest a set of features and classifiers that are useful in accomplishing this task. However, in addition to answering questions about the most effective way to distinguish natural profiles from synthetic ones, the results raise a number of new questions about performance, multi-class capabilities, and classification costs that would be useful to investigate further. Some of these are summarized below.

8.1 Classifier Committees via Boosting

The committees used in this project consisted of stacking approaches (majority voting, averaging individual results, and computing the median of individual results) and bootstrap sampling to create a random forest. Boosting techniques, such as the Adaboost algorithm, are another avenue to explore. Though slightly more complex than the above-mentioned techniques, Adaboost can improve performance for classifiers whose error rates are uncorrelated [29]. In the Adaboost algorithm, classifiers that make the fewest errors on the test data receive higher weights. Patterns that are misclassified by high-performing classifiers are given more attention by subsequent classifiers. The Adaboost algorithm generally reduces variability of the final decision more than bagging, the approach used to construct the random forest in **Chapter**

7 [33].

8.2 Multi-Class Classification

In this project, the PowerPlex kit was used to generate natural, WGA, and bacterial-derived STR profiles. Although there were three categories of data, they were compared in a pairwise manner using two-class classification: natural profiles were separately compared with WGA profiles and bacterial profiles. Ideally, the problem should be approached using multi-class classification, which would entail creating training and test datasets that contain natural, WGA, cloned bacterial, and synthetic bacterial profiles. Some classifiers have built-in functionality for multi-class classification, but others, such as the SVM, are binary in nature. However, many strategies exist that enable the use of these classifiers for multi-class classification. In the "one versus all" strategy, a single classifier is trained for each class to distinguish that class from all of the other classes. Testing and prediction are performed by each classifier individually, and the prediction with the highest confidence score is selected. In the case of ties, outputs for each class are scanned across all pairwise classifiers that include that class, and the minimum is selected. These minimum values are compared to find the class with highest minimum value. The final classification decision corresponds to that class [21].

8.3 Identify the Source of Classifier Errors

As an extension of the project, it would be interesting to identify the source of classification errors made by the individual and committee classifiers. Errors typically arise when the data are non-separable, or when outliers are present. The feature scatterplots in **Appendix A** and **Appendix E** indicate that both criteria hold for the sample data. It would be interesting to determine profiles that are more likely to be misclassified than other profiles, and the aspects of those profiles that make them difficult to classify correctly. The misclassification results for committee classifiers in

Chapter 7 suggest that some profiles are indeed more likely to be misclassified by multiple classifiers but do not explain why that is the case.

8.4 Determining ROC Curves

In the DNA authentication problem, costs and class prior probabilities are variable and sometimes unknown. For example, it is difficult to determine the prior probability that a genetic profile can be "faked" because few data currently exist. Additionally, the cost of false positives $p(nat|wga)$ and false negatives $p(nat|wga)$ varies greatly based on the context in which DNA authentication is being performed. To limit the impact of these unknowns, it would be useful to develop Receiver Operating Characteristic (ROC) curves for each of the classifiers and committees analyzed [43]. **Figure 8-1** demonstrates an approach for calculating these curves. In a ROC curve, the threshold for classifier results is varied to trace out different rates of detection and false-alarms. To produce such a curve, the outputs of posterior probability classifiers can be used directly. For likelihood classifiers, a posterior probability can be calculated, or the likelihood ratio can be used. In an ROC curve, a classifier dominates other classifiers if its performance is above and to the left of other classifiers. Thus, to determine the best classifier for a given problem, regardless of the weights used in the performance score or the prior probabilities of the data distribution, one can use the dominant classifier in an ROC curve, if such a classifier exists, or selected the best classifier over the desired operating range. Such a range can be defined as the acceptable rate of false positives for the classification problem.

8.5 Other Cost Metrics: Time and Memory

For the purposes of this work, highest performing classifiers were defined as classifiers that made the fewest mistakes on the test data. Other metrics, such as memory and time considerations, were not evaluated. However, these metrics are important to consider in real-world applications and should be examined in greater detail.

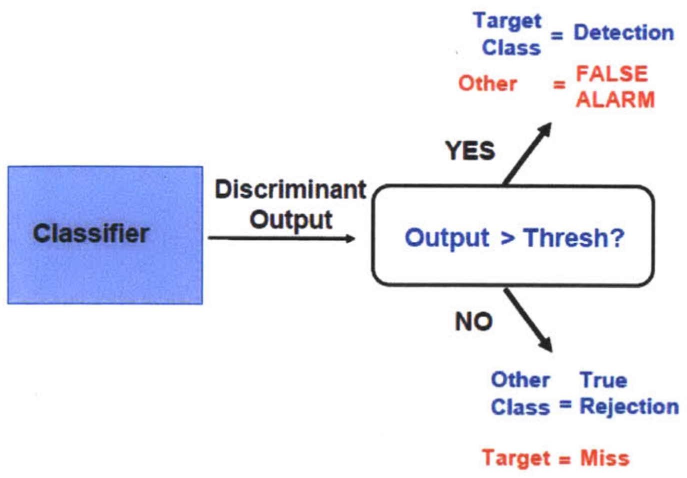


Figure 8-1: *Method to generate receiver operating characteristic (ROC) curve [43].*

For example, feature selection may be the critical component for obtaining rapid training [35]. For a multi-layer perceptron with linearly separable classes, a single layer of the network can classify simple non-overlapping class distributions via three logical functions: "and", "or", and "majority" (**Figure 8-2**). These algorithms all result in equal correctness, but training is fastest if input features are selected to require "or" function learning, so this function should be selected for overall optimal performance.

The difference in performance of the K-nearest neighbors classifier and other neighbor-based classifiers provides another reason to consider cost metrics other than accuracy. Both classifiers were high performers and had similar error rates on most of the data sets analyzed. However, the K-nearest neighbors algorithm has large memory and computation requirements. Other classifiers operate on the same principle as KNN, but reduce the number of samples that need to be stored to perform training [31]. For example, the CKNN classifier stores only training patterns that fall near decision borders. The nearest cluster classifier clusters training patterns and measures the distance to cluster centers. The learning vector quantizer moves patterns to improve classification accuracy, and the hypersphere classifier adds hyperspheres to form decision regions instead of exemplars. Of these alternatives to KNN, CKNN

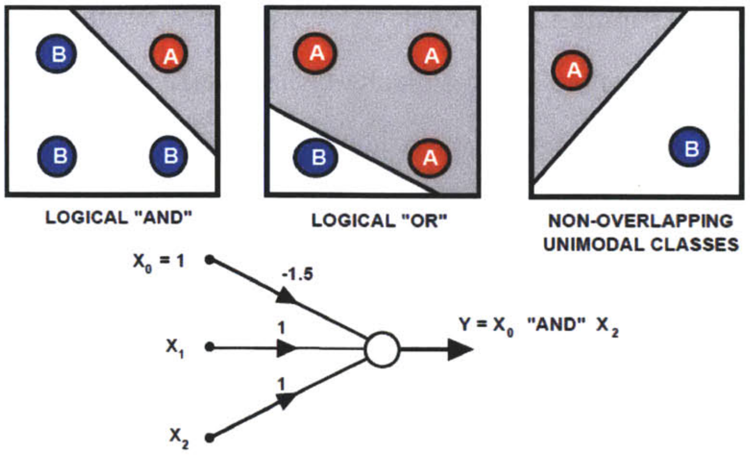


Figure 8-2: *A single layer of the MLP algorithm on a separable data set can be implemented via logical "and", "or", "majority" functions, which all return correct outputs, but differ in training and test time [35].*

generally provides comparable performance. Additionally, for some applications, the savings in time and memory of the LVQ, hypersphere, and nearest cluster algorithms may provide more important than slightly lower correctness. These tradeoffs are useful to investigate in future extensions of the project.

Finally, though the Parzen classifier was the top performer for most of the experiments, this classifier is not frequently used in practice [34]. This classifier is computationally intensive and models class densities with more computation than is typically required to achieve high accuracy.

8.6 Quantitative Estimate of Classifier Generalization

A common problem in machine learning is the tendency to overfit classifier parameters to the training data. This results in good performance on the training data, but poor generalization to test data. Small data samples, such as the one used in this project, are particularly vulnerable to the overfitting problem [7]. Techniques like feature selection and four-fold cross validation help to reduce overfitting, but it would be useful to obtain a quantitative measure of classifier generalization. This can be achieved via the Vapnik-Chervonenkis (VC) dimension, which provides a worst-case upper bound for generalization error. Other metrics of generalization include the minimum description length approach and regularization theory [7]. These techniques measure predicted generalization as a function of the training error and a penalty term related to classifier complexity. Less complex classifiers, such a KNN and CKNN, have a lower penalty than more complex classifiers such as non-linear SVMs. The generalization of the tuned classifiers can further be improved, if necessary, using a number of techniques. These include reducing the number of input features by performing more aggressive feature selection, principal component analysis (PCA), or linear discriminant analysis (LDA). For the SVM and KNN classifiers, the internal smoothing can be increased by altering values of the sigma and K parameters, respectively. For the MLP classifier as well as the random forest, the number of nodes can be pruned. The ranges of parameter values may also be constricted to smaller values than used for the tuning algorithms in **Chapter 5**. Other potential techniques include stopping stochastic training early and sharing parameters across classes ("grand" or "pooled" covariance matrices) [35].

Chapter 9

Summary of Major Conclusions

This project aimed to address the short tandem repeat (STR) authentication problem via an *in silico* approach. Currently, bisulfite sequencing is the state-of-the-art method most often used to establish STR profile authenticity, but the cost of this approach is high in both time and resources. The goal of this work was to facilitate STR profile authentication by developing a set of machine learning algorithms to differentiate between STR profiles and synthetic profiles generated by standard laboratory techniques such as whole genome amplification and bacterial cloning.

Toward this end, sample profiles were obtained and amplified via the commercial Identifiler and PowerPlex analysis kits. A set of promising features, described in **Chapter 2**, were identified and a set of 16 machine learning classifiers were chosen. **Table 9.1** summarizes each of the features that was analyzed. The default performance of each classifier was measured and quantified via a performance score:

$$\text{performance score} = w1 * p(nat|nat) + w2 * p(synth|synth) - w3 * p(synth|nat) - w4 * p(nat|synth) - w5 * p(fail)$$

Greedy feature selection was performed on each classifier to determine the optimal subset of features to use for improved performance. This selection was performed at the level of individual peaks, profile channels, and entire profiles. It was found that the best performance was achieved when feature selection was performed at the profile level, considering the minimum, maximum, and average values of a given feature for a profile, as well as the range of feature values for that profile. Though the optimal feature set differed for each classifier, certain trends emerged during feature selection and are summarized in **Figure 9-1** (repeated from **Chapter 4**). In particular, the inter-locus imbalance, stutter count, and peak width features were selected by many classifiers.

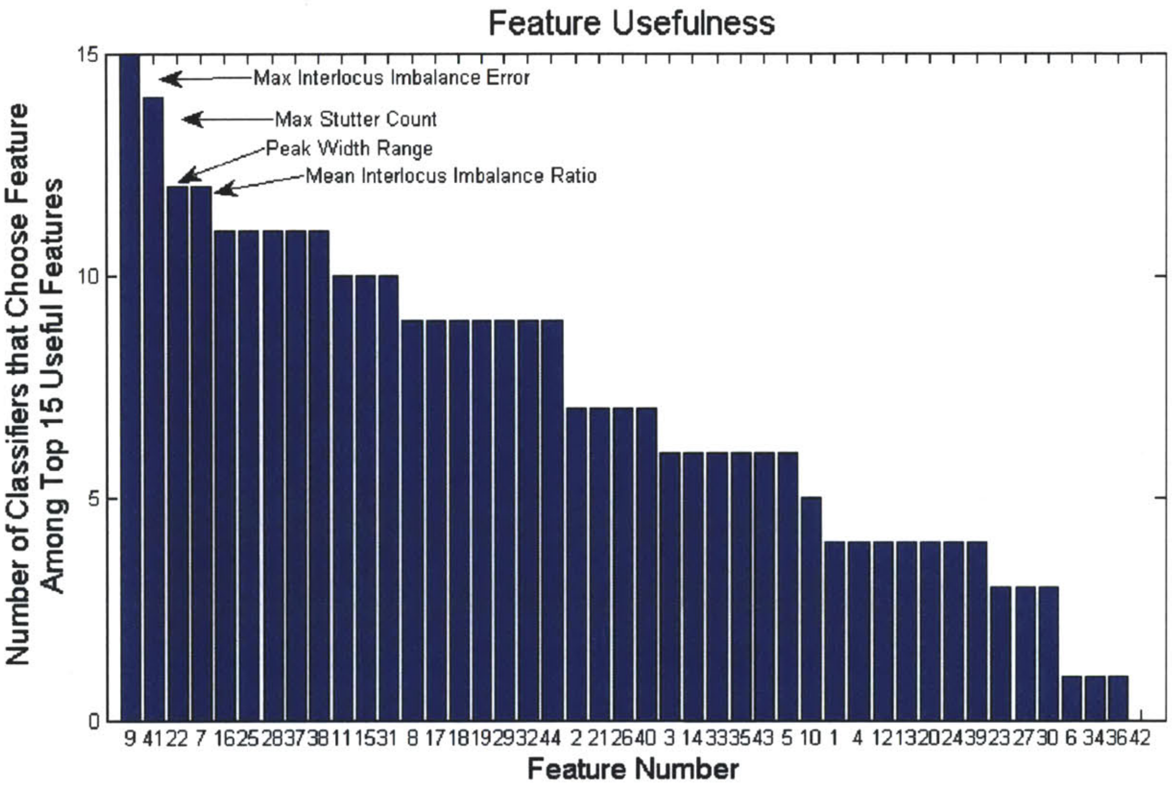


Figure 9-1: *Some features are particularly useful across a variety of the classifiers examined.*

Once feature selection had been performed for each classifier, the selected features were used to tune classifier parameters. Each classifier had a given number of tunable parameters, ranging from one for the KNN algorithm to 10 for the MLP. Consequently, since a full feature sweep was infeasibly expensive, a gradient ascent algorithm was used to select the best value for each parameter from a supplied range of choices. In case of ties, values closest to the LNKnet default were used. The final

	Prof Min	Prof Max	Prof Range	Prof Mean
Heterozygote inter-locus imbalance	0	1	2	3
Inter-locus Imbalance Ratio	4	5	6	7
Inter-locus Imbalance Error	8	9	10	11
Inter-channel Intensity	12	13	14	15
SNR	16	17	18	19
Peak Width	20	21	22	23
Gaussian error	24	25	26	27
Ski Slope	28	29	30	31
Off ladder inside bin	32	33	34	35
Off ladder outside bin	36	37	38	39
Stutter count	40	41	42	43

Table 9.1: *Feature guide: each number indicates a specific feature value for a profile. In total, 44 features were examined.*

tuned classifier performance was quantified and is presented in **Figures 9-2** and **9-3** (reproduced from **Chapter 6**). In **Figure 9-2**, the first column for each classifier represents its default performance, the second column represents performance after feature selection, and the third demonstrates performance after parameter tuning. **Figure 9-3** illustrates only the tuned performance of all classifiers, presenting in ascending order. All classifiers achieved overall error rates below 35%. The top five performers were: CKNN, KNN, MLP, Parzen, and SVM. Each of these achieved an overall error rate below 10% percent, with the Parzen classifier performing best of all.

Subsequently, an effort was made to improve classifier performance by forming committees via stacking techniques (taking a majority vote of constituent member results, averaging the results, or calculating the median value of the results). In another approach, binary tree classifiers were combined to create a random forest.

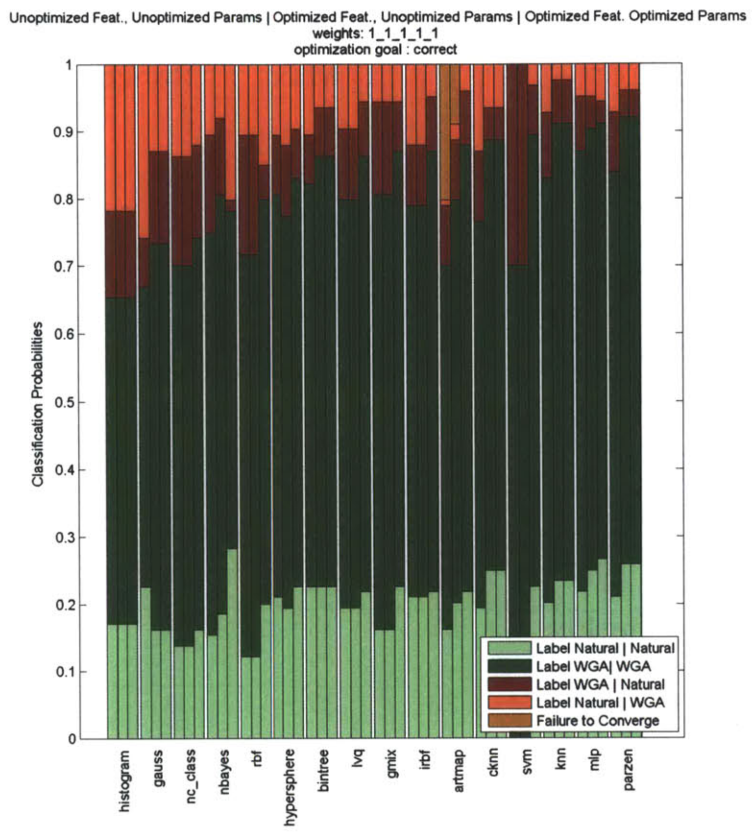


Figure 9-2: *Identifiler, weights = [1,1,1,1,1], features optimized, column 1: baseline performance, column 2: feature selection, column 3: parameter tuning.*

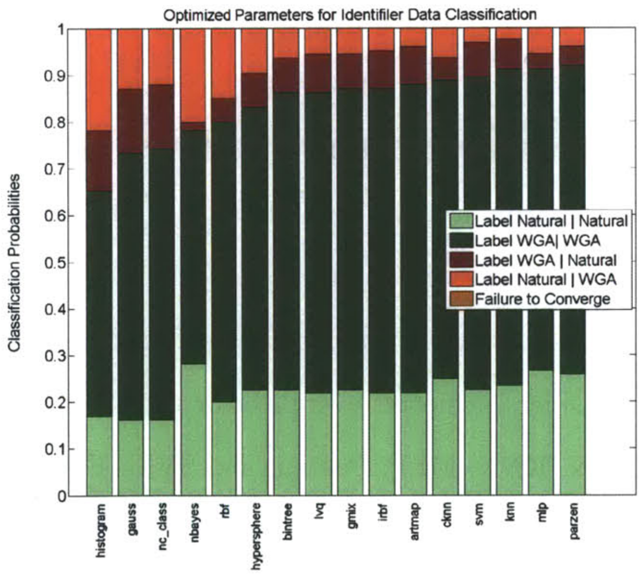


Figure 9-3: *Identifiler, weights = [1,1,1,1,1], fine-tuned parameters.*

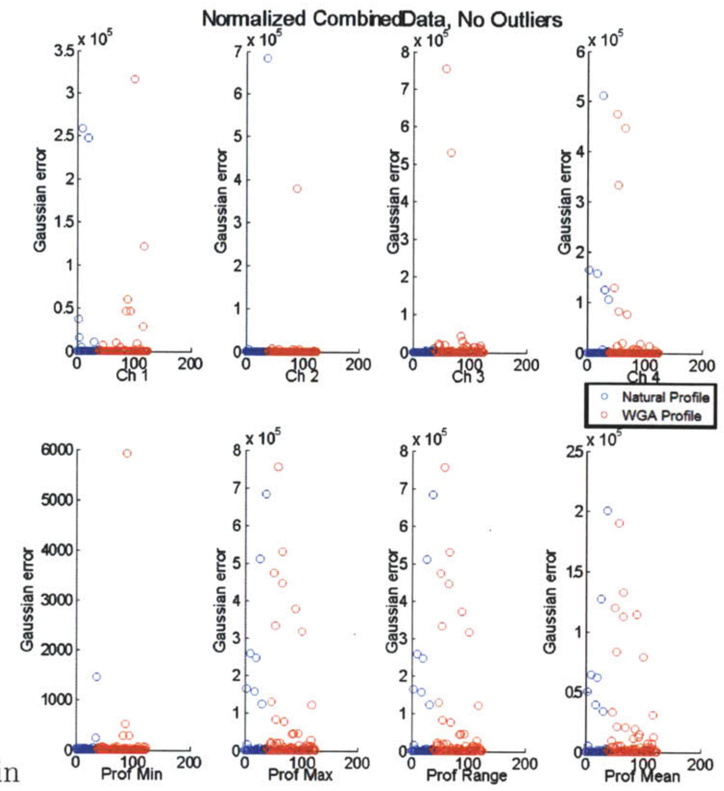
Ultimately however, committee performance was no better than that of the top-performing members.

In conclusion, the machine learning approach to STR profile authentication is currently not robust enough to definitively authenticate sample profiles for real-world applications. However, it can serve as a highly valuable preliminary assessment tool used to trigger the need for further profile analysis via more costly and complex techniques.

THIS PAGE INTENTIONALLY LEFT BLANK

Appendix A

Raw Feature Data for Identifiler Kit



inin

Figure A-1: *Gaussian Error For Each Channel and Profile (Normalized, No Outliers)*

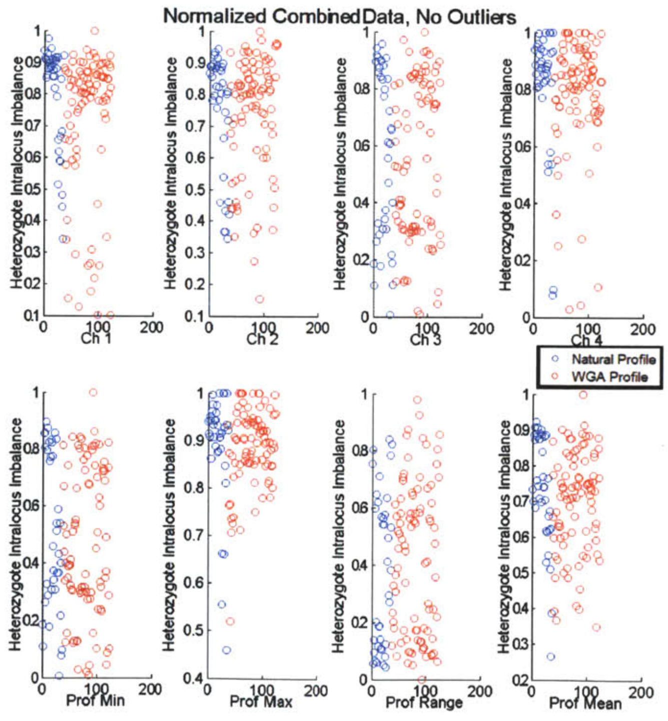


Figure A-2: *Heterozygote Intralocus Imbalance For Each Channel and Profile (Normalized, No Outliers)*

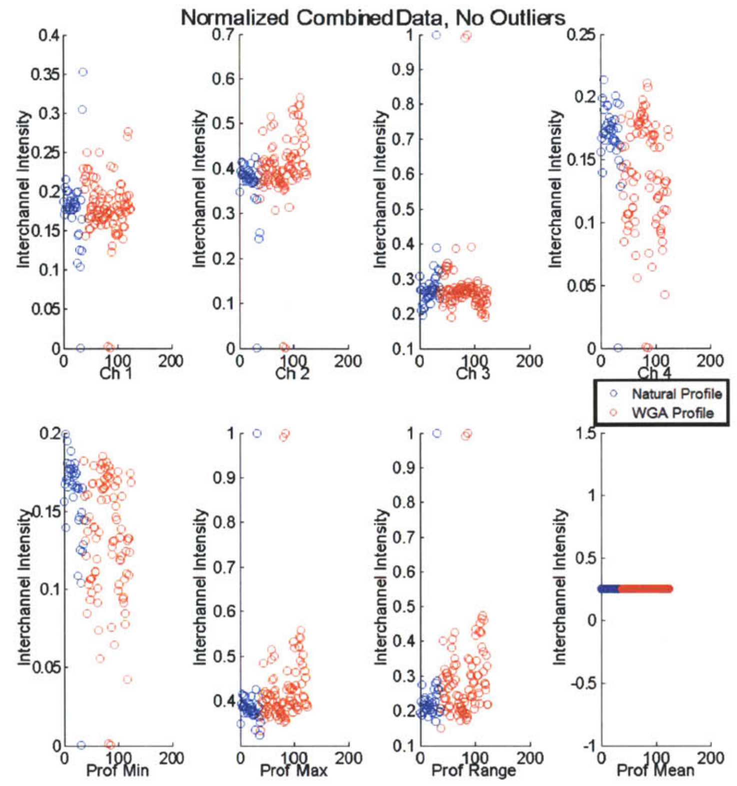


Figure A-3: *Interchannel Intensity For Each Channel and Profile (Normalized, No Outliers)*

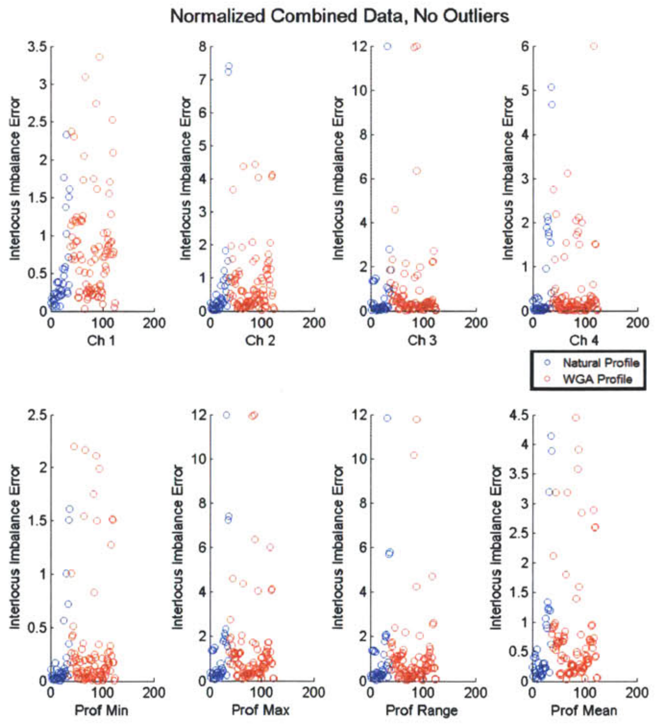


Figure A-4: *Interlocus Imbalance Error For Each Channel and Profile (Normalized, No Outliers)*

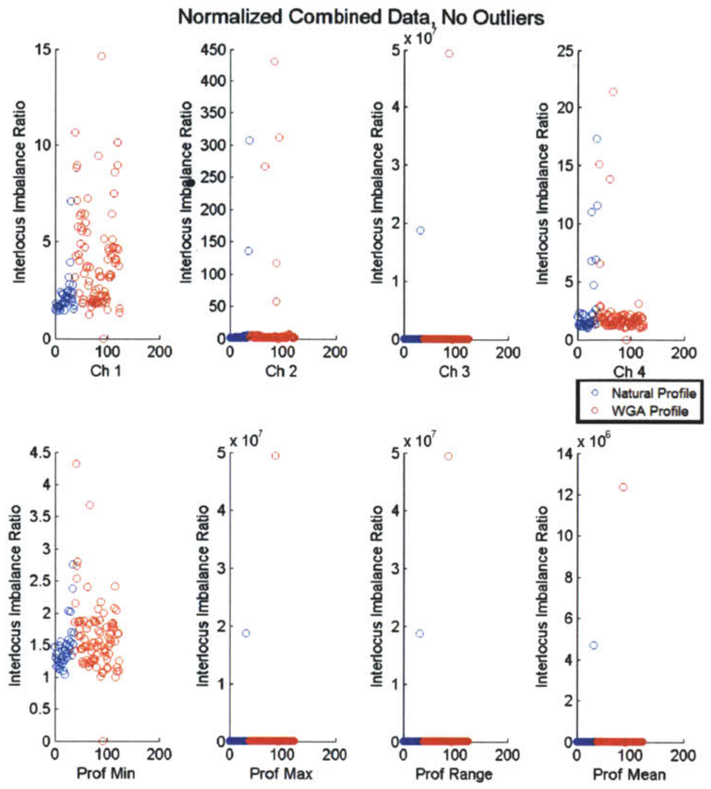


Figure A-5: *Interchannel Intensity For Each Channel and Profile (Normalized, No Outliers)*

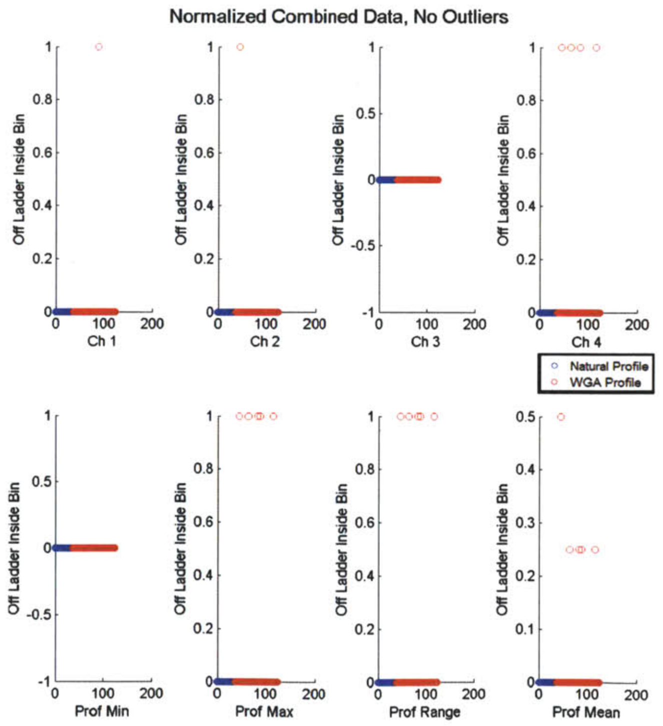


Figure A-6: *Off Ladder Inside Bin For Each Channel and Profile (Normalized, No Outliers)*

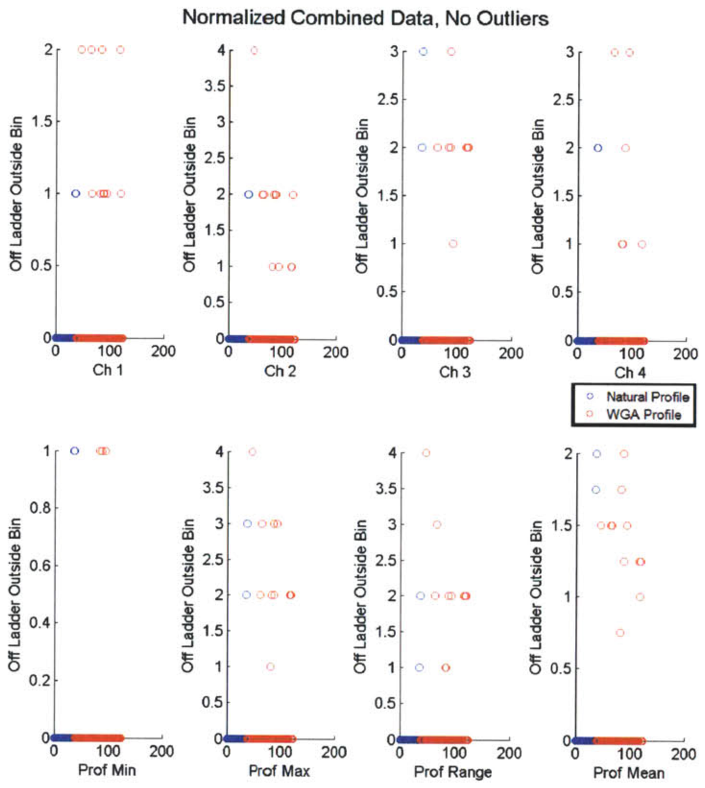


Figure A-7: *Off Ladder Outside Bin For Each Channel and Profile (Normalized, No Outliers)*

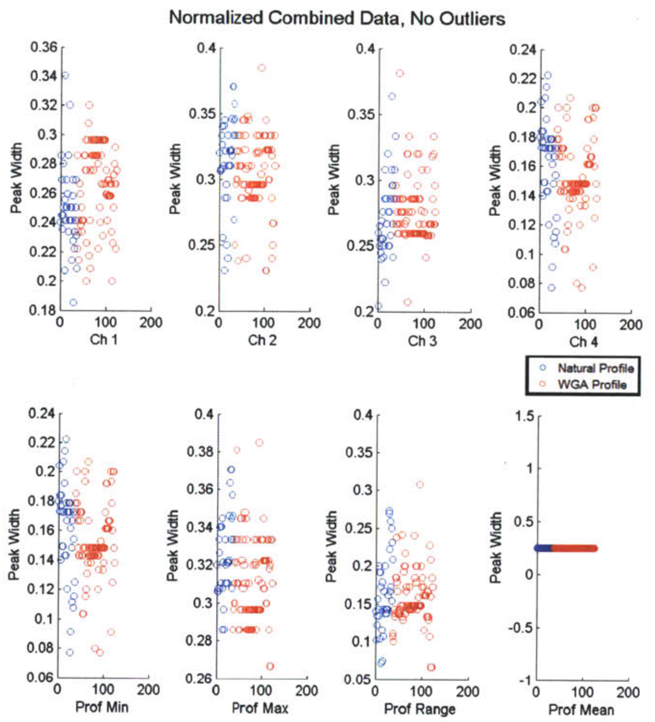


Figure A-8: *Peak Width For Each Channel and Profile (Normalized, No Outliers)*

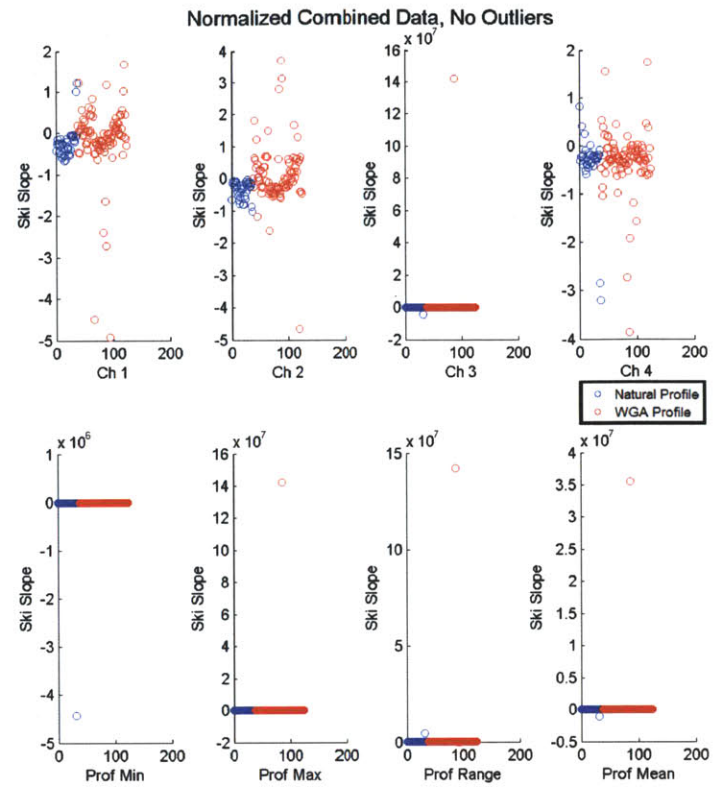


Figure A-9: *Ski Slope For Each Channel and Profile (Normalized, No Outliers)*

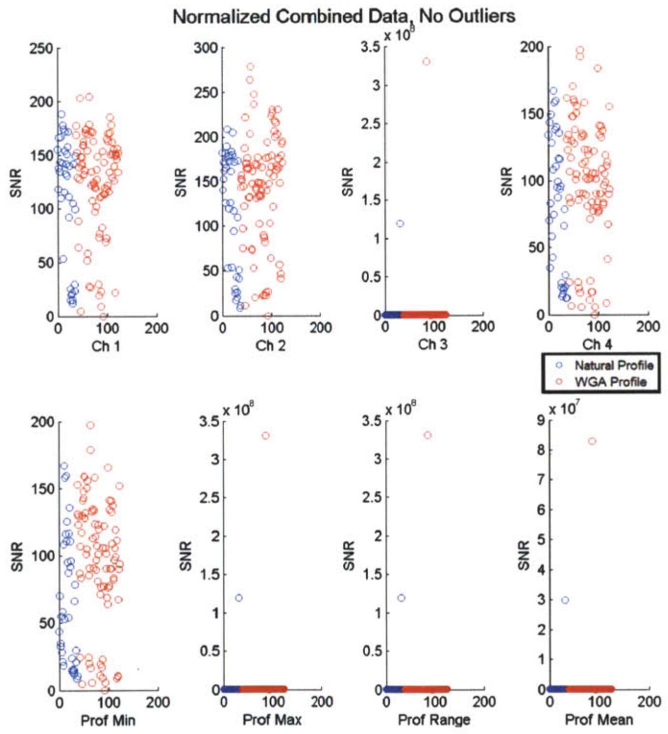


Figure A-10: *SNR For Each Channel and Profile (Normalized, No Outliers)*

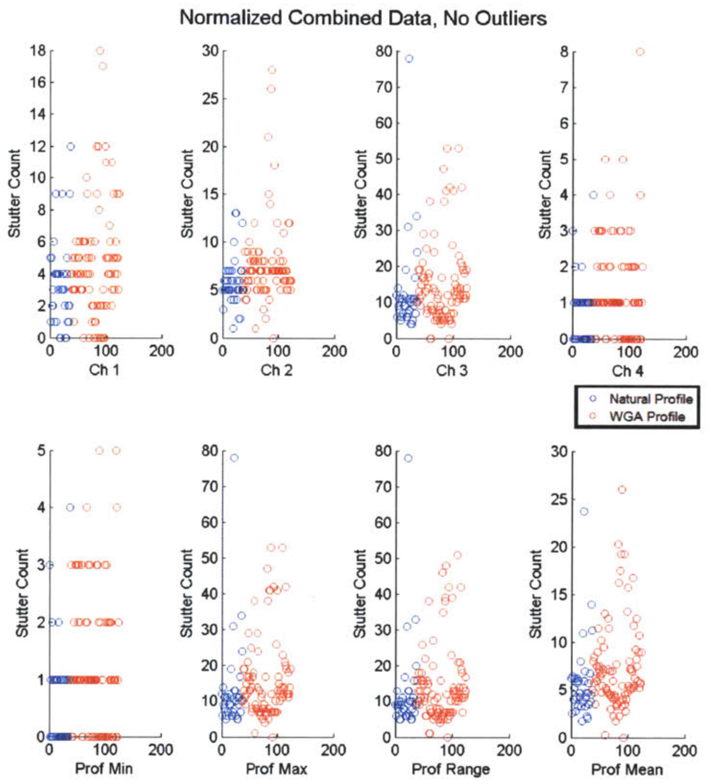


Figure A-11: *Stutter Count For Each Channel and Profile (Normalized, No Outliers)*

THIS PAGE INTENTIONALLY LEFT BLANK

Appendix B

Features Selected by a Variety of Classifiers

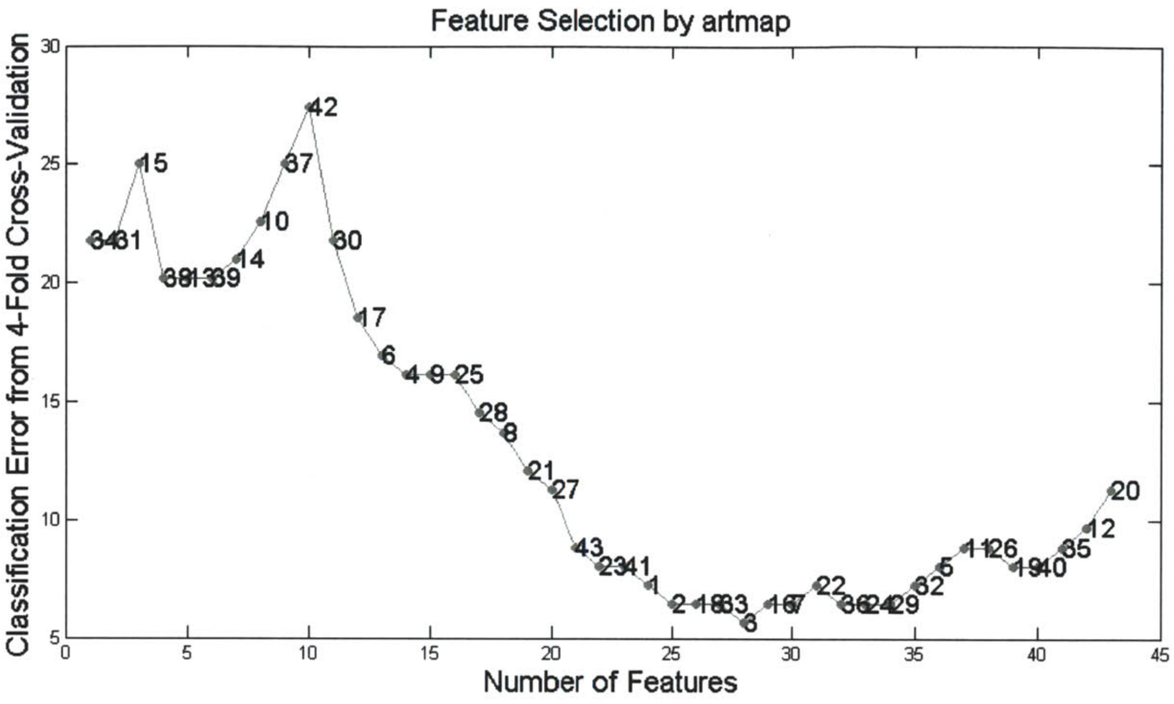


Figure B-1: *Artmap Classifier Feature Selection*

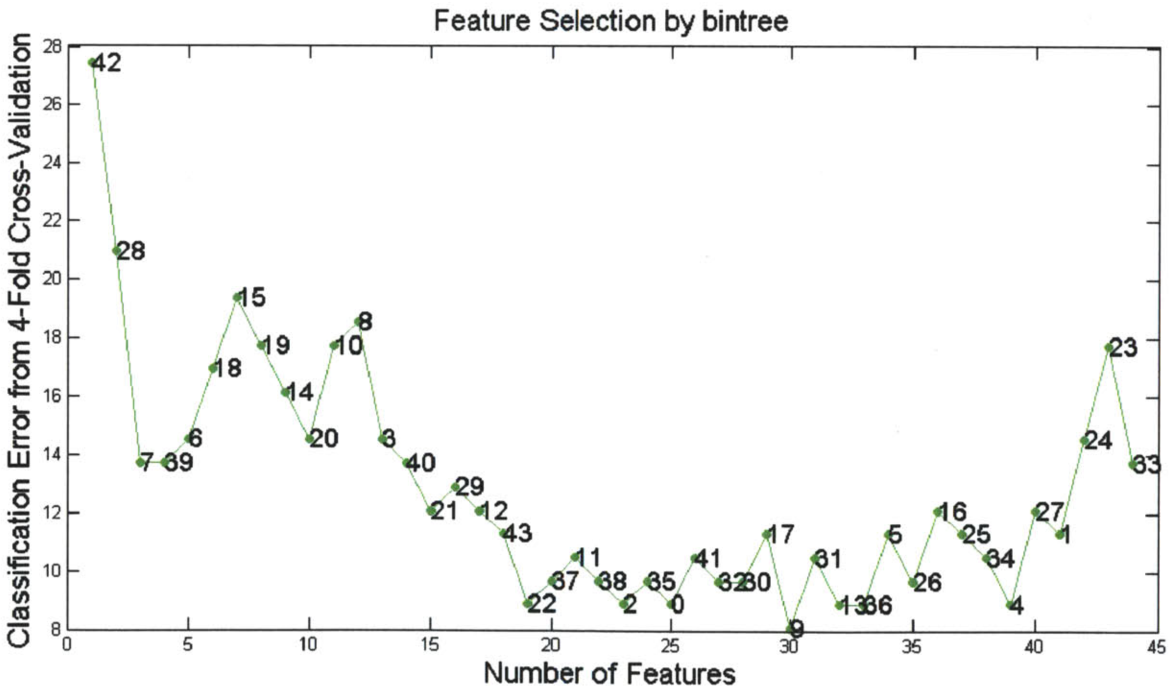


Figure B-2: *Bintree Classifier Feature Selection*

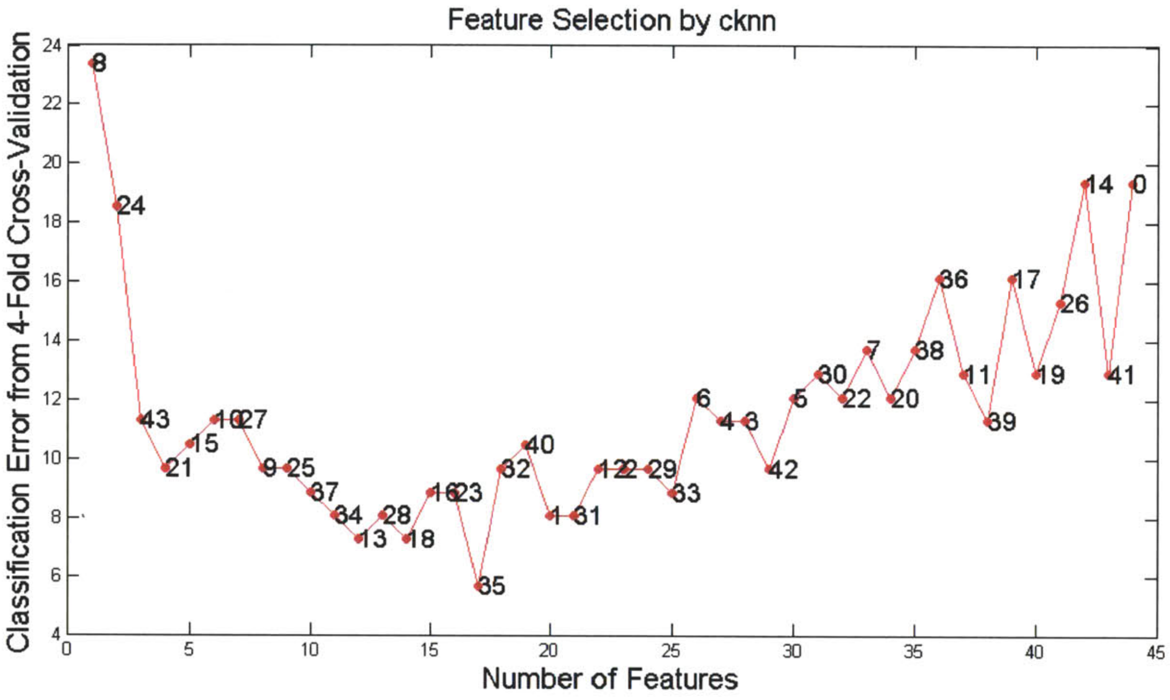


Figure B-3: *Condensed K Nearest Neighbors Classifier Feature Selection*

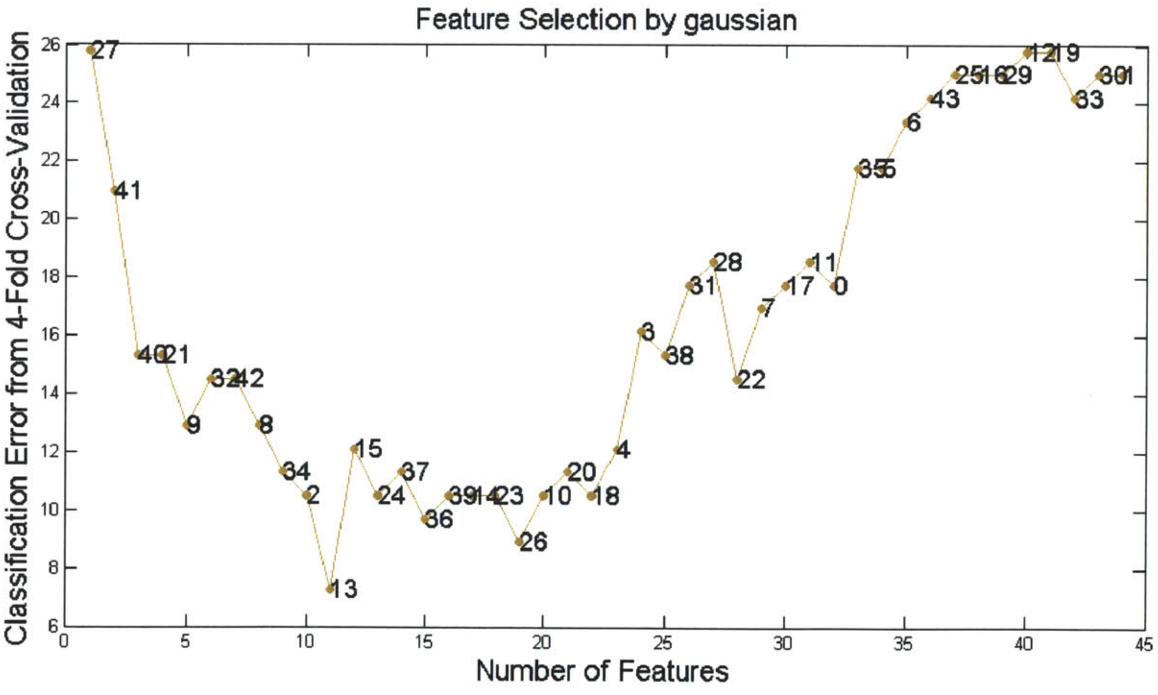


Figure B-4: *Gaussian Classifier Feature Selection*

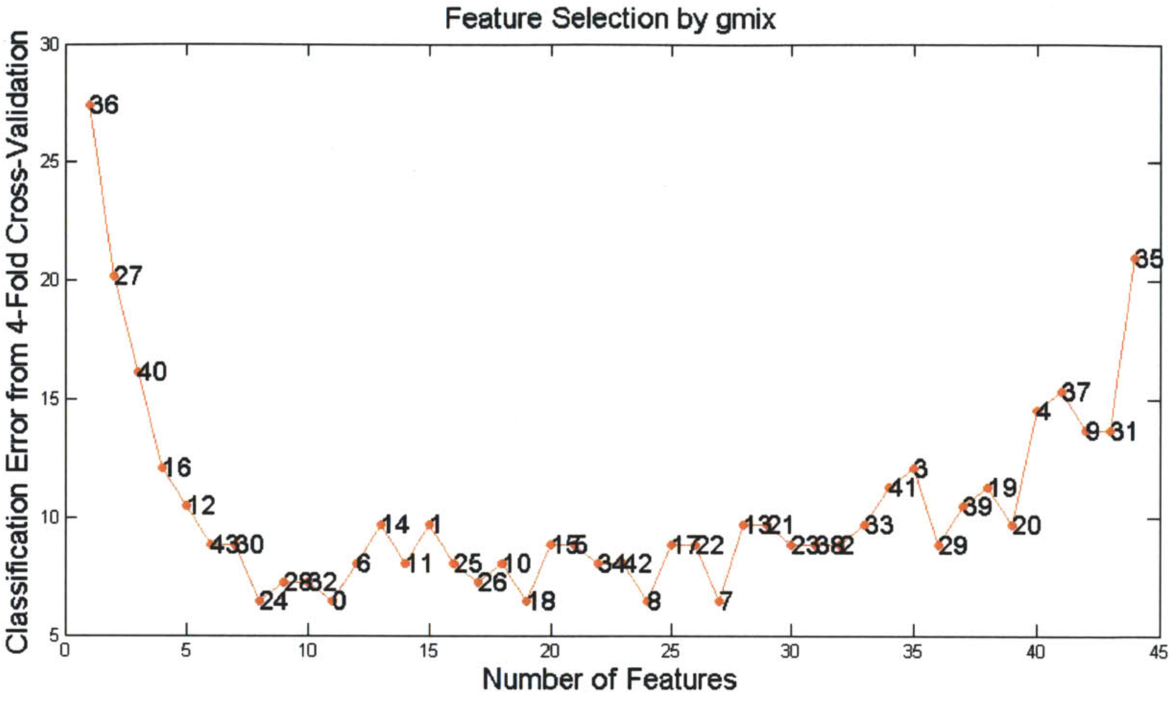


Figure B-5: *Gaussian Mixture Model Feature Selection*

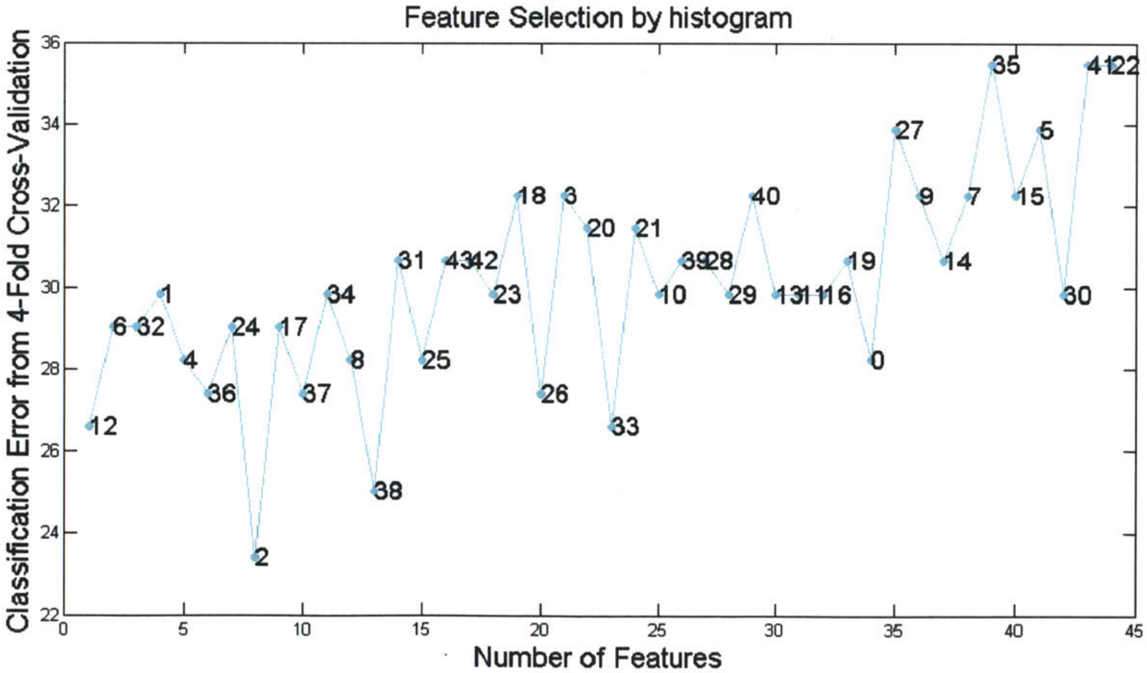


Figure B-6: *Histogram Classifier Feature Selection*

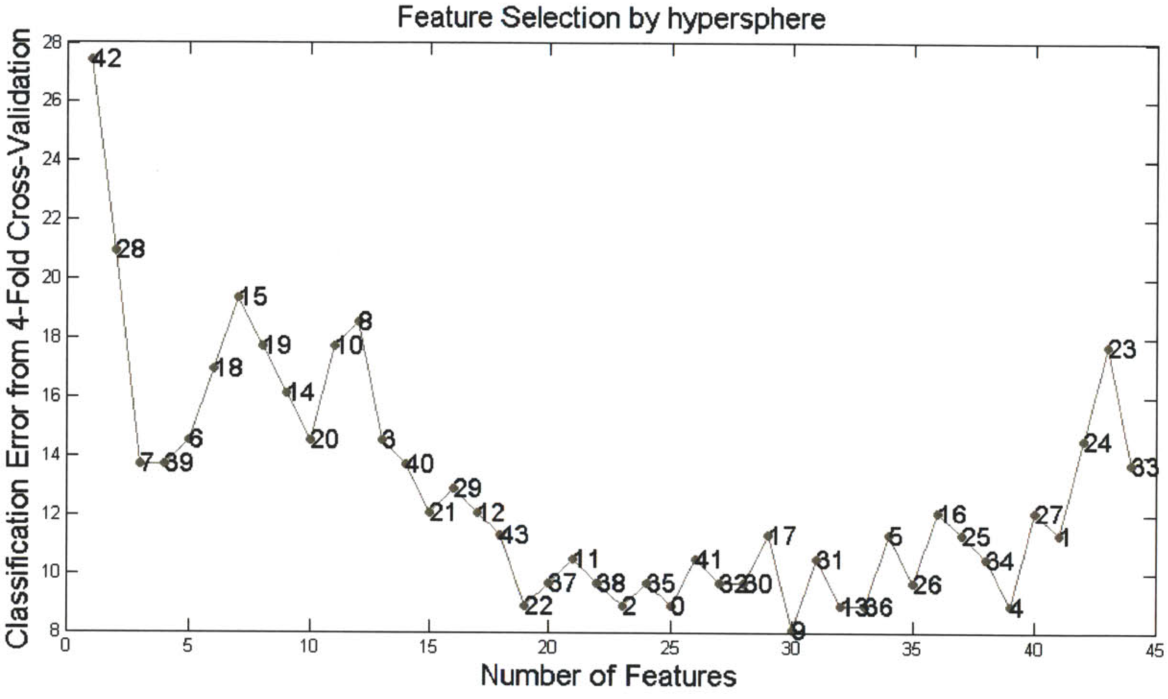


Figure B-7: *Hypersphere Classifier Feature Selection*

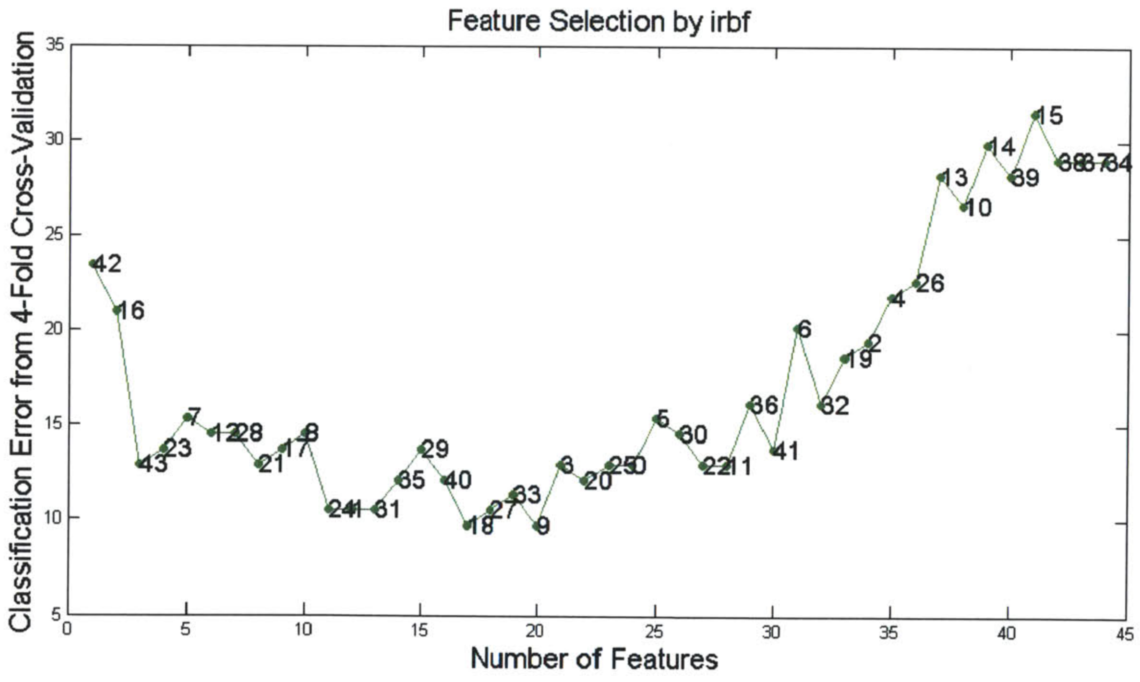


Figure B-8: *Incremental Radial Basis Function Classifier Feature Selection*

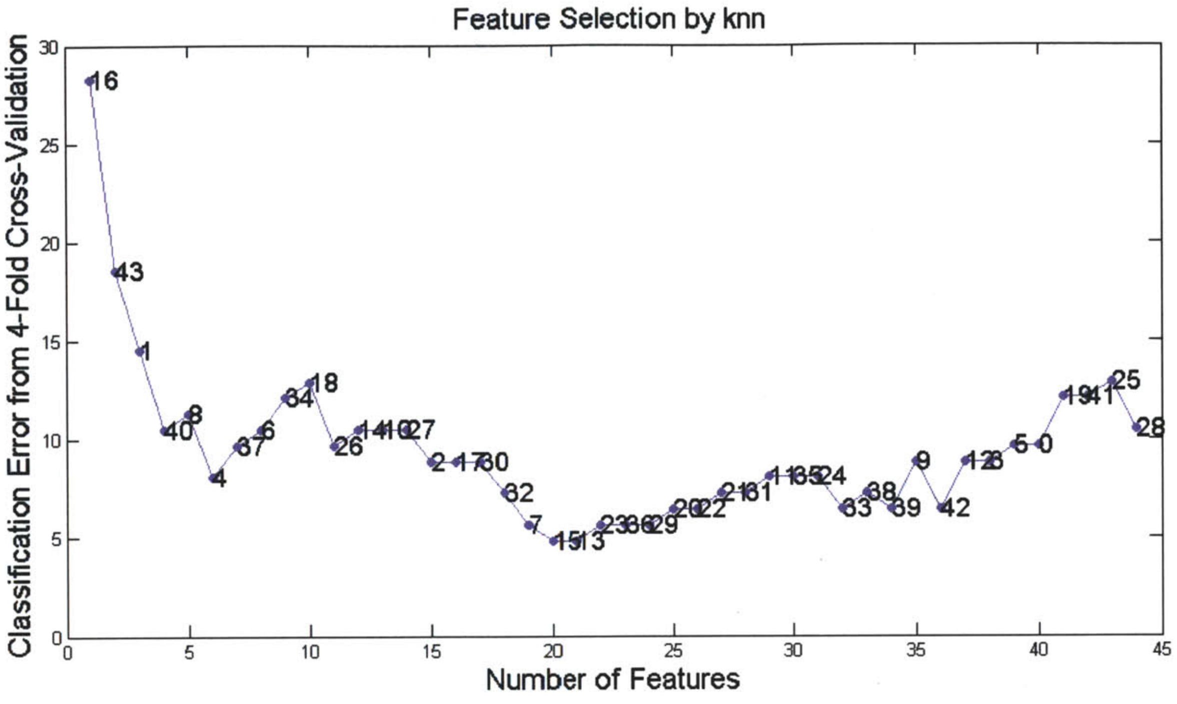


Figure B-9: *K Nearest Neighbors Classifier Feature Selection*

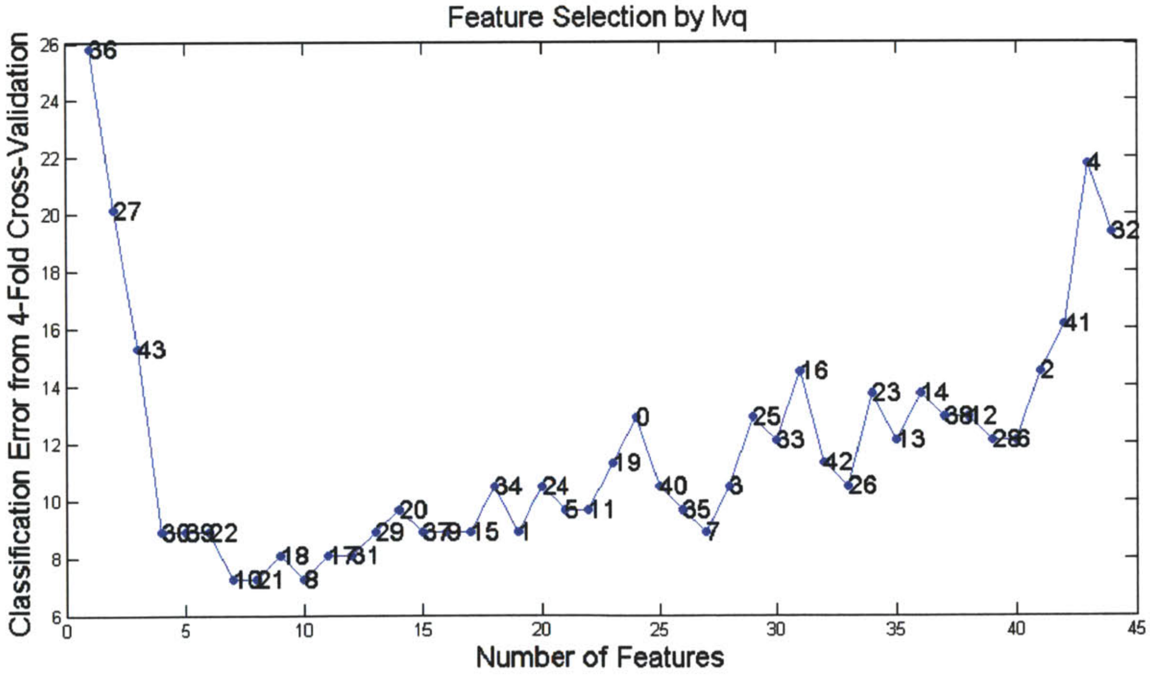


Figure B-10: *Linear Vector Quantizer Classifier Feature Selection*

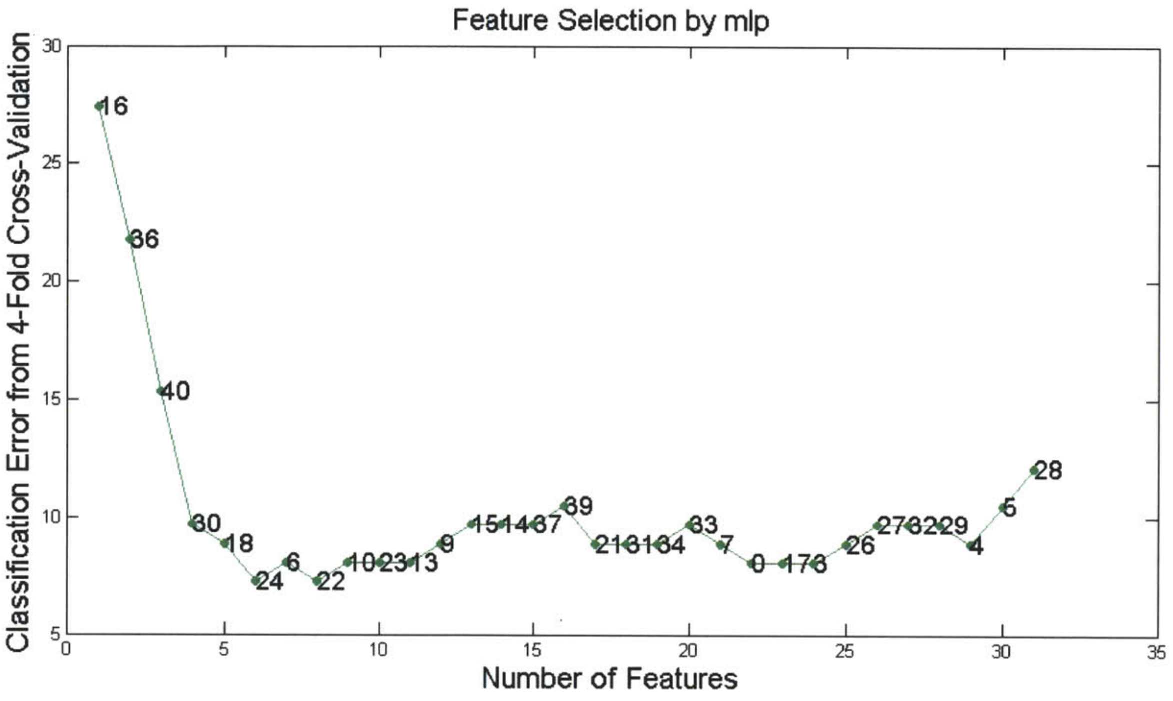


Figure B-11: *Multi-Layer Perceptron Feature Selection*

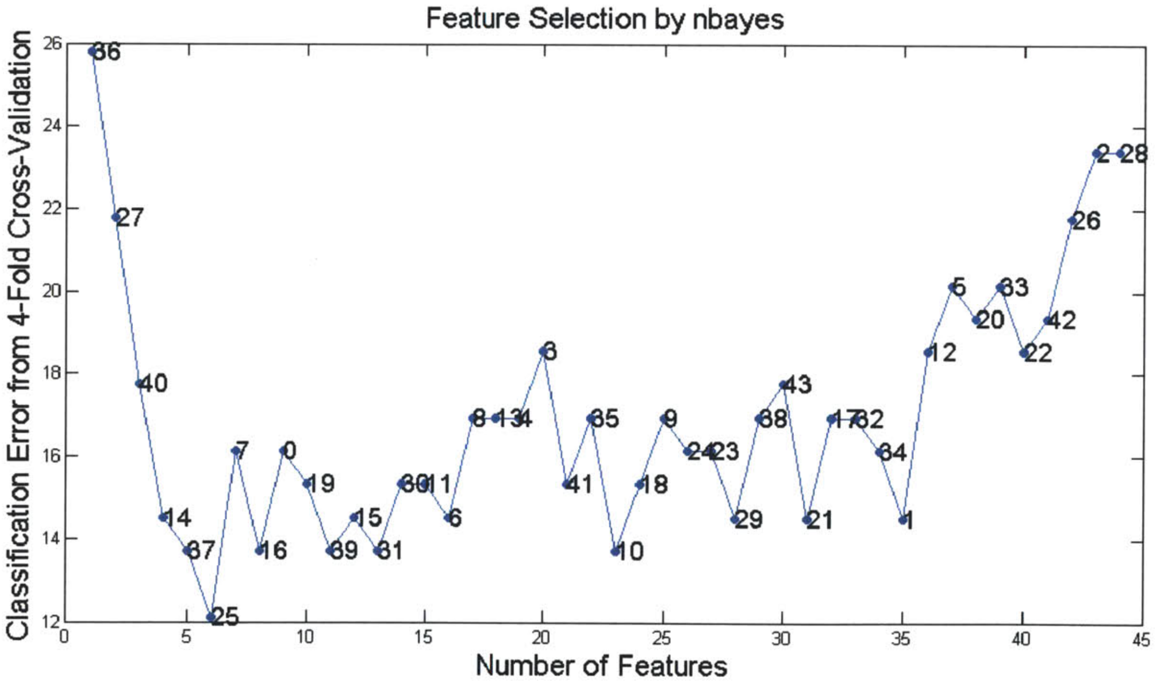


Figure B-12: *Naive Bayes Classifier Feature Selection*

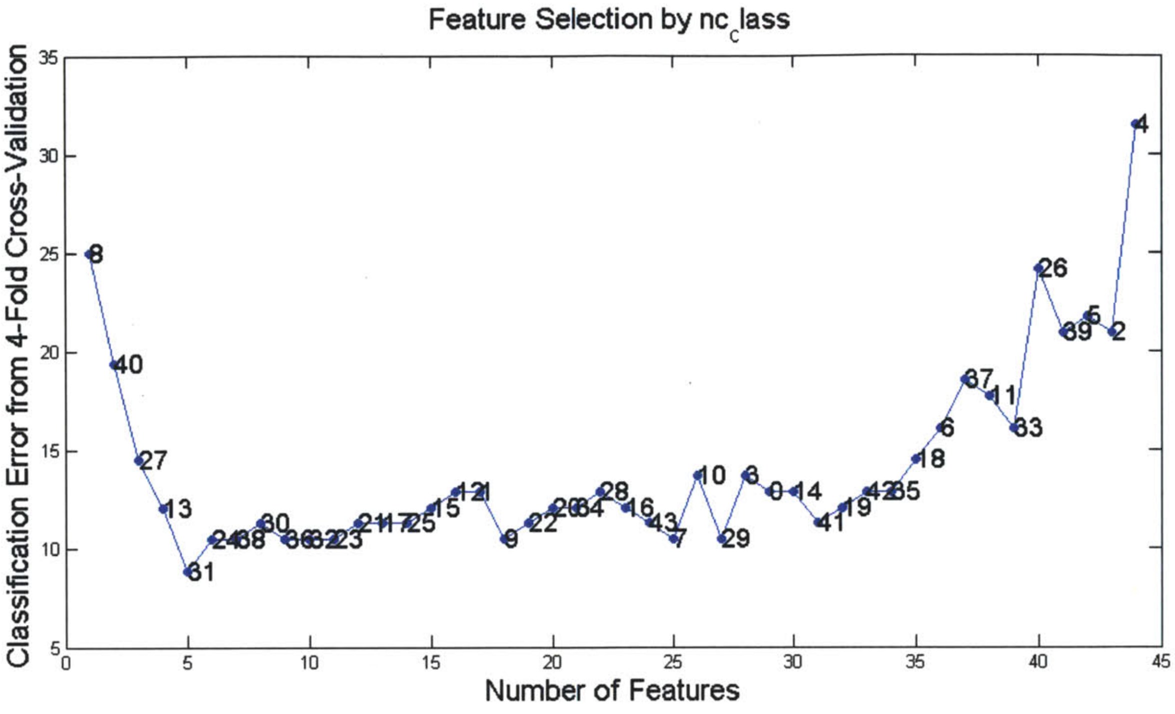


Figure B-13: *Nearest Cluster Classifier Feature Selection*

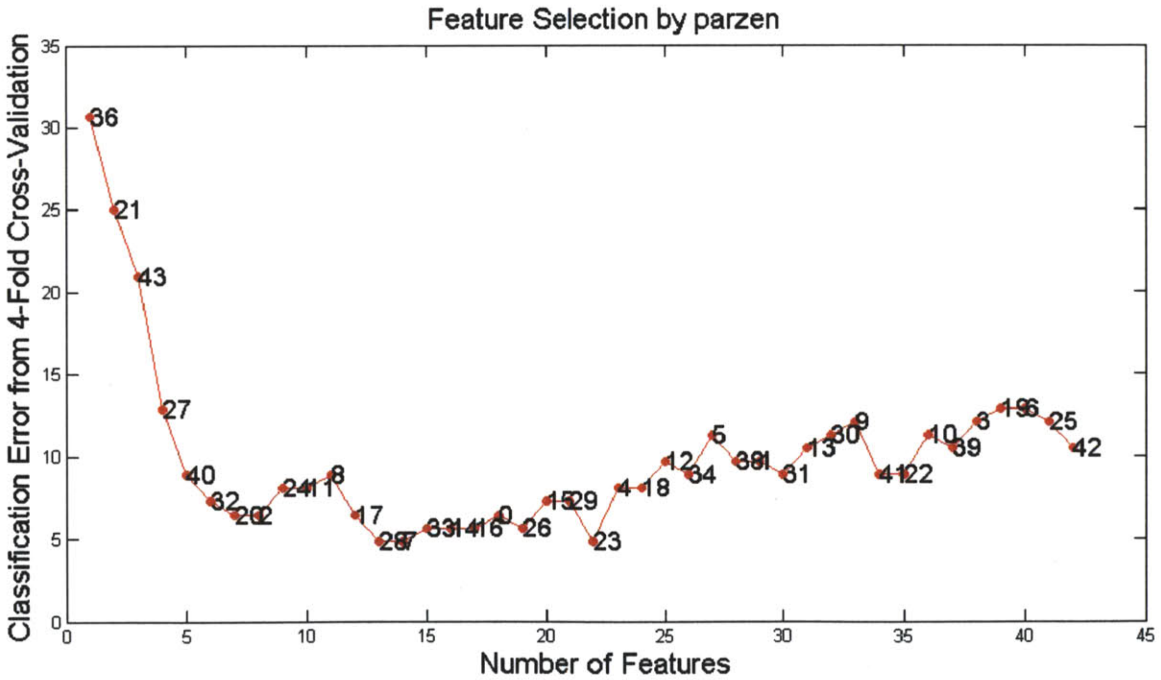


Figure B-14: *Parzen Classifier Feature Selection*

Appendix C

Individual Classifier Tuning

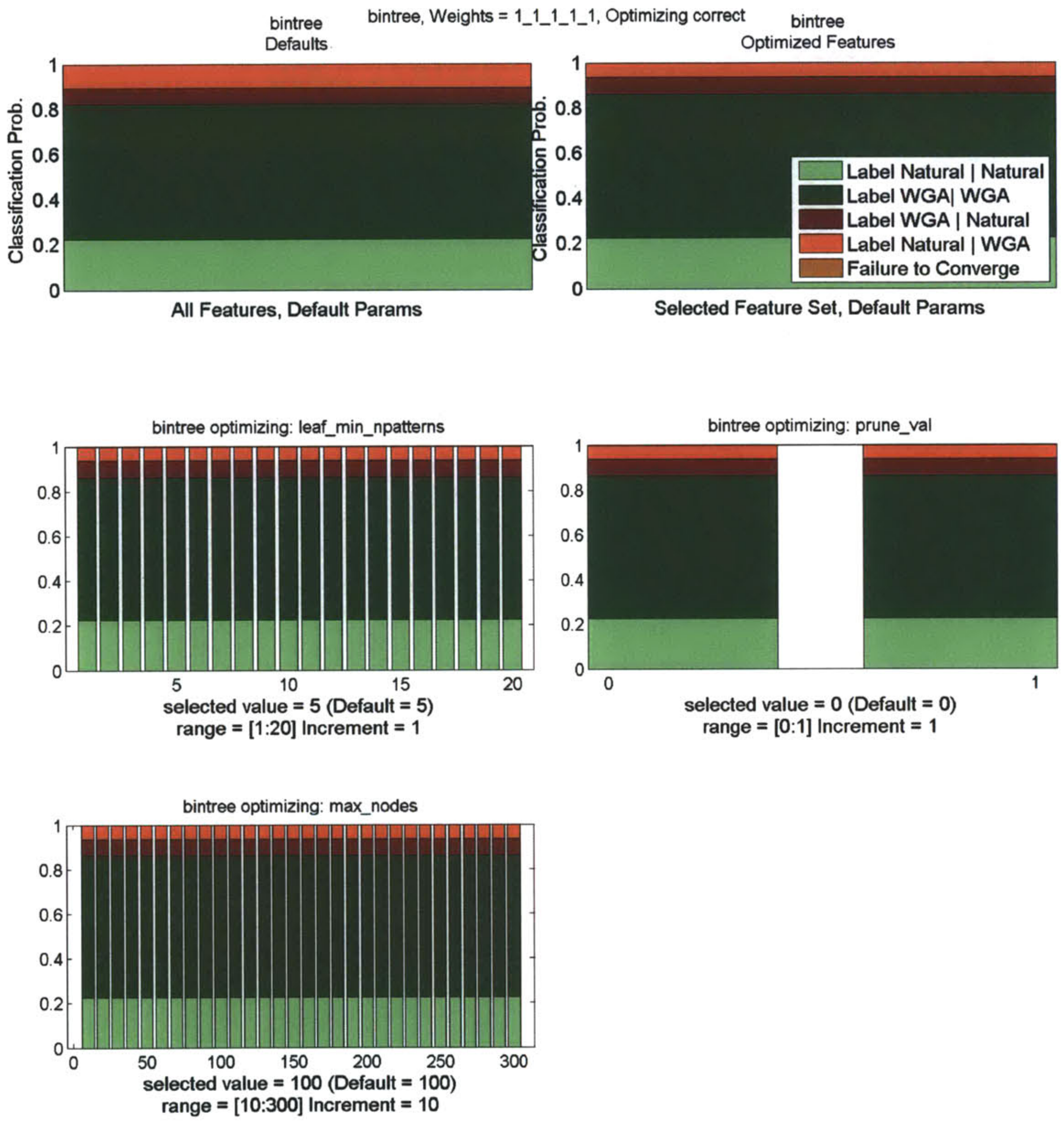


Figure C-1: *Binary Tree Classifier with parameters leaf_min_npatterns, prune_val, max_node.*

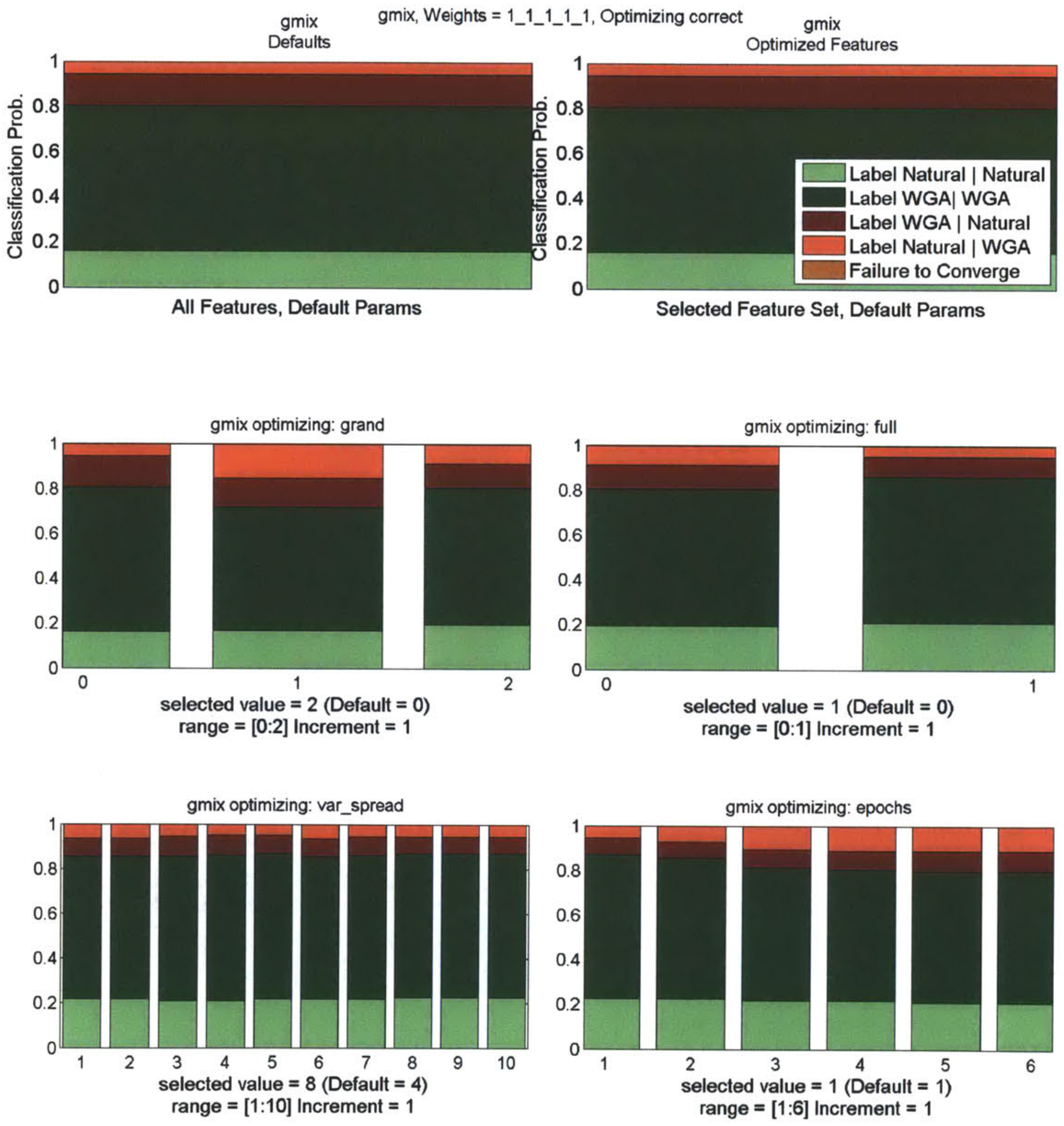


Figure C-2: *Gaussian Mixture Classifier with parameters grand, full, var_spread, epochs.*

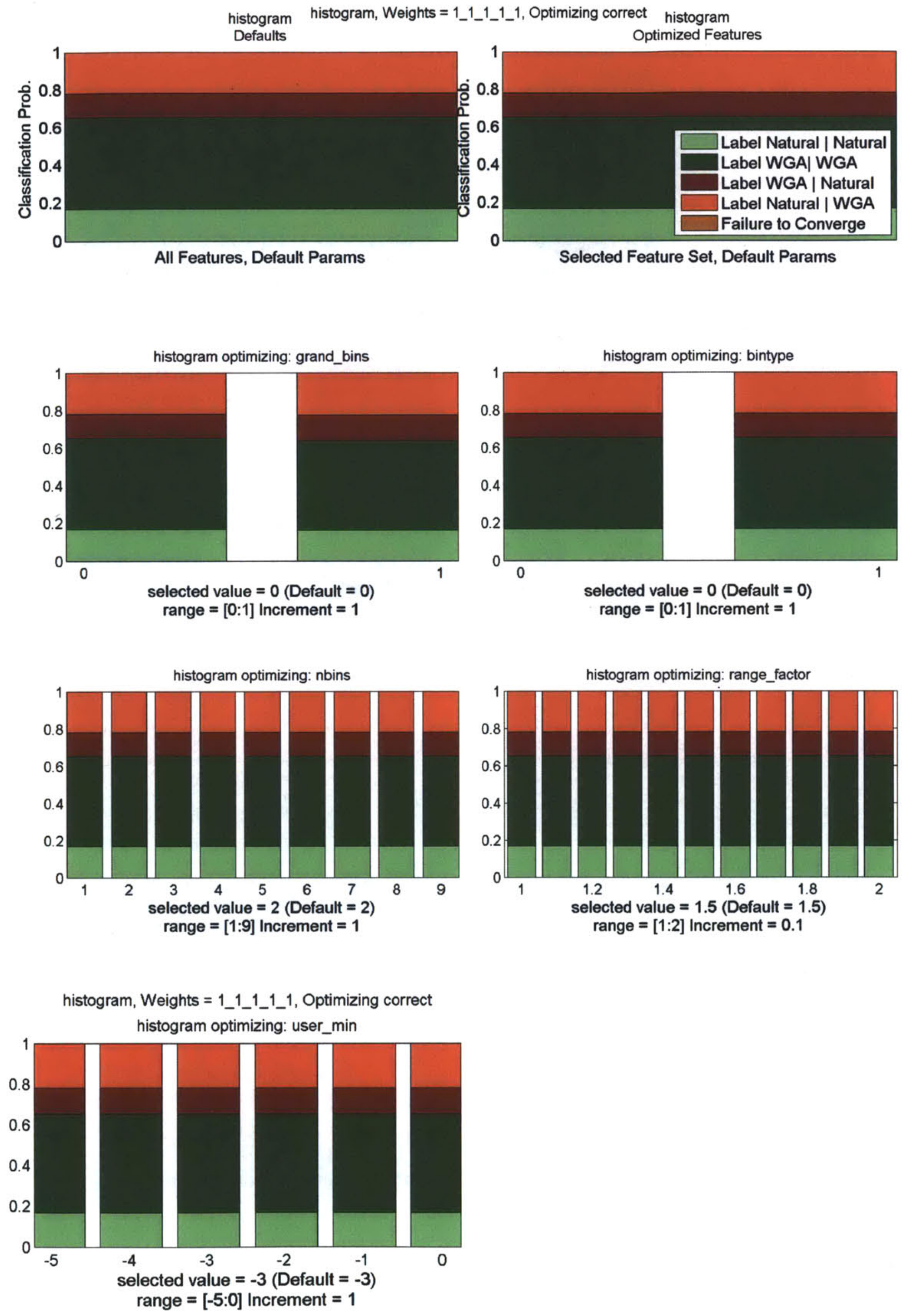


Figure C-3: *Histogram Classifier with parameters grand_bins, bintype, nbins, range_factor.*

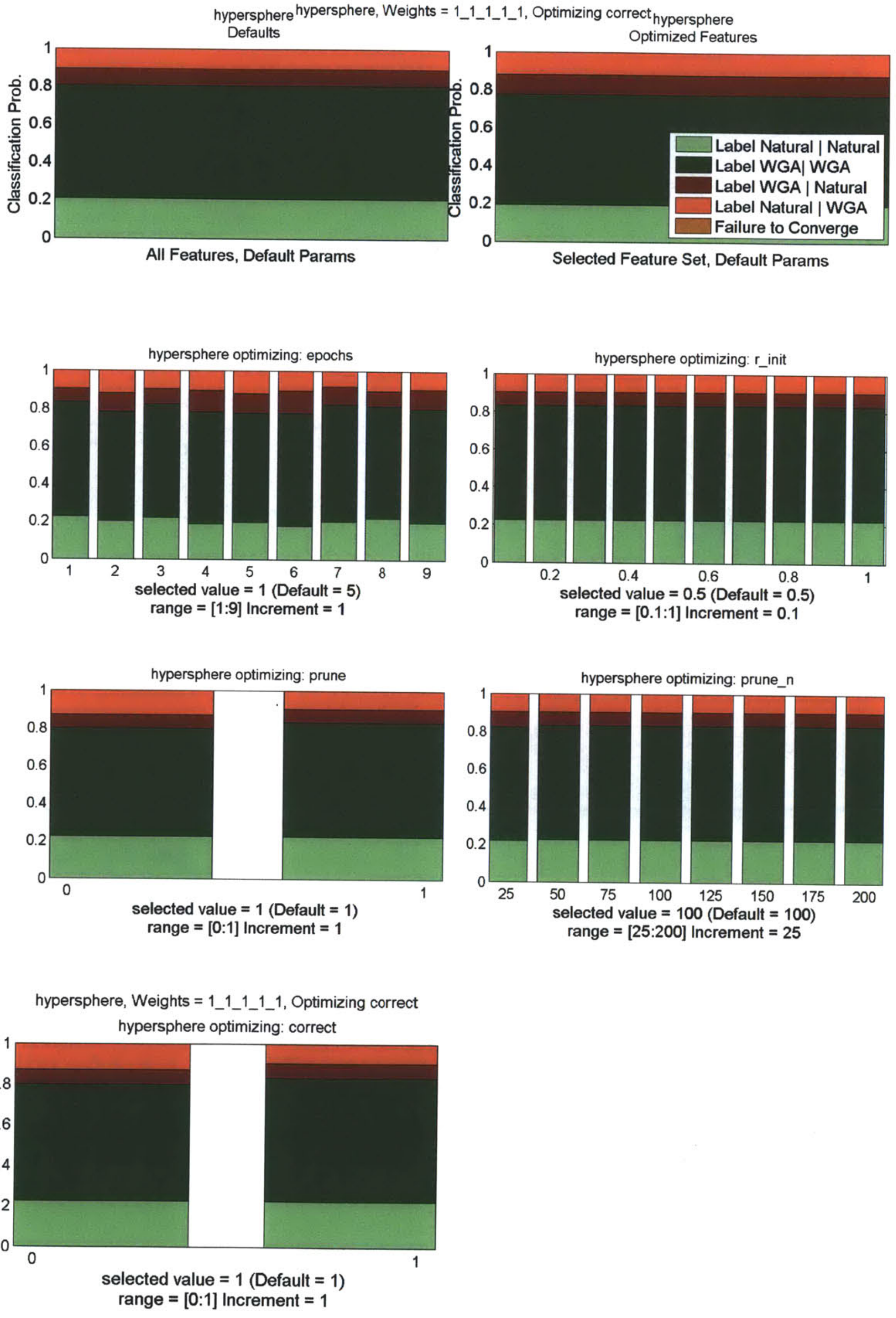


Figure C-4: *Hypersphere Classifier with parameters grand_bins, bintype, nbins, range_factor.*

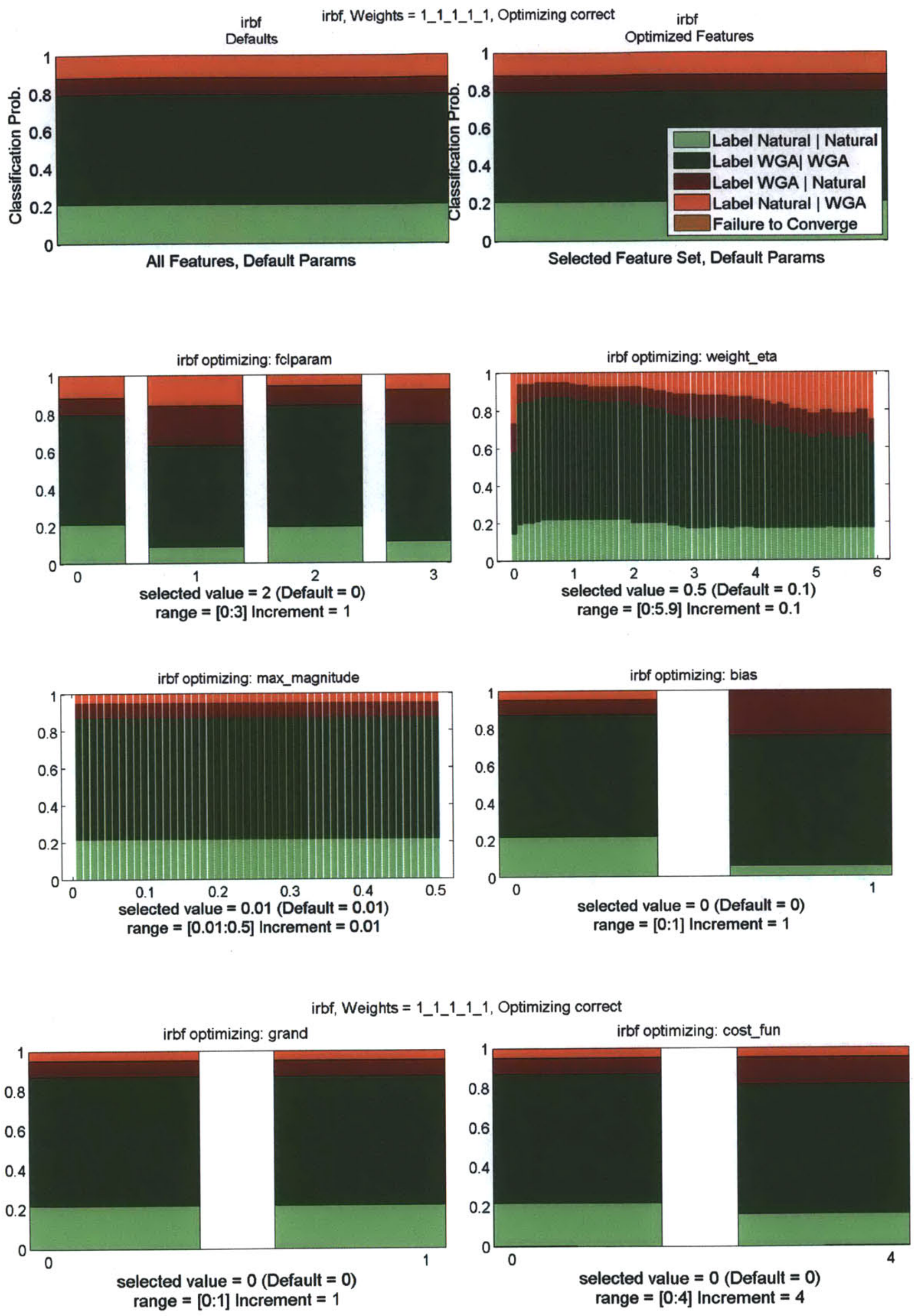


Figure C-5: *Incremental Radial Basis Function with parameters fclparam, weight_eta, max_magnitude, bias, grand, cost_fun.*

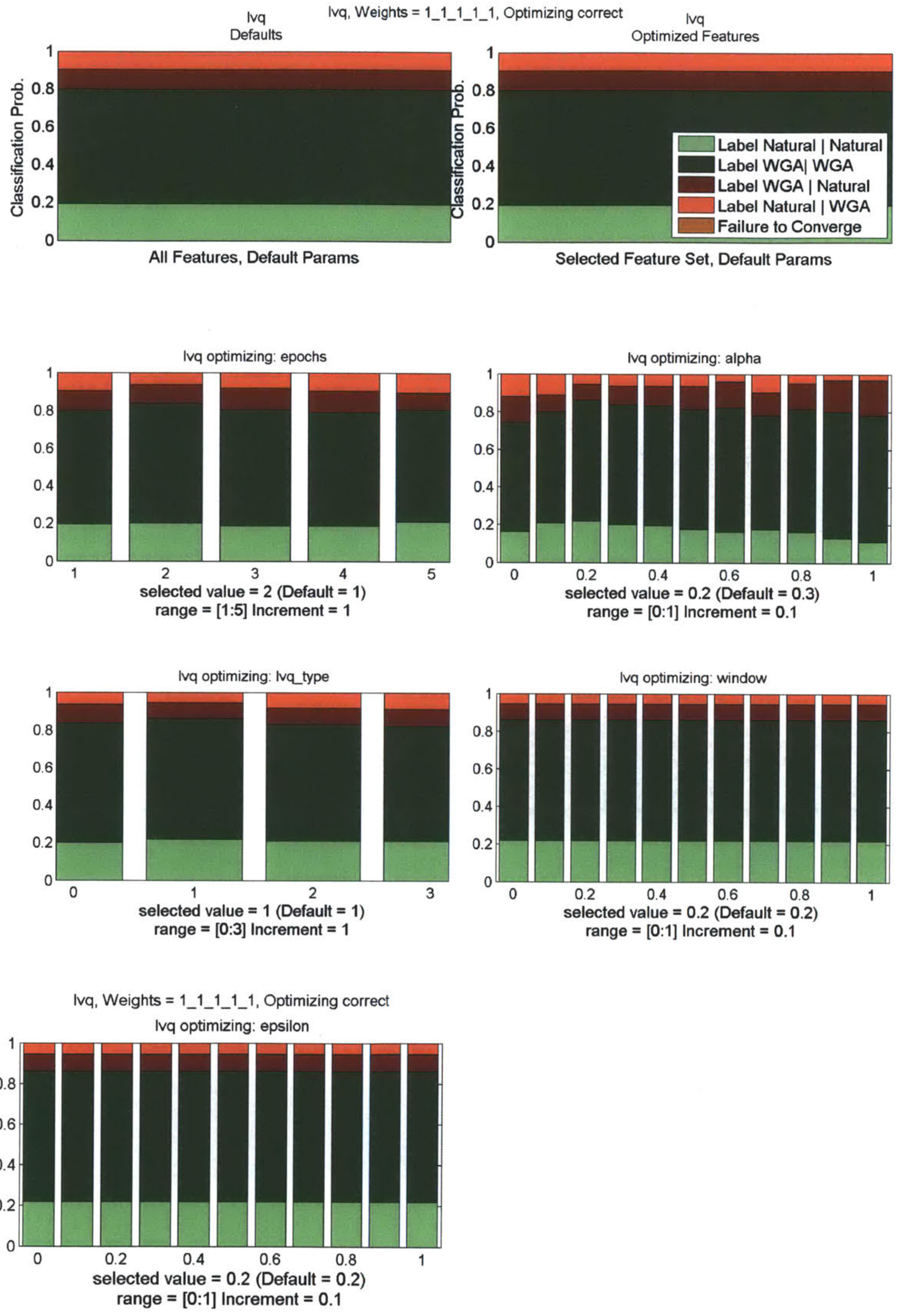


Figure C-6: *Learning Vector Quantizer with parameters epochs, alpha, lvq_type, window, epsilon.*

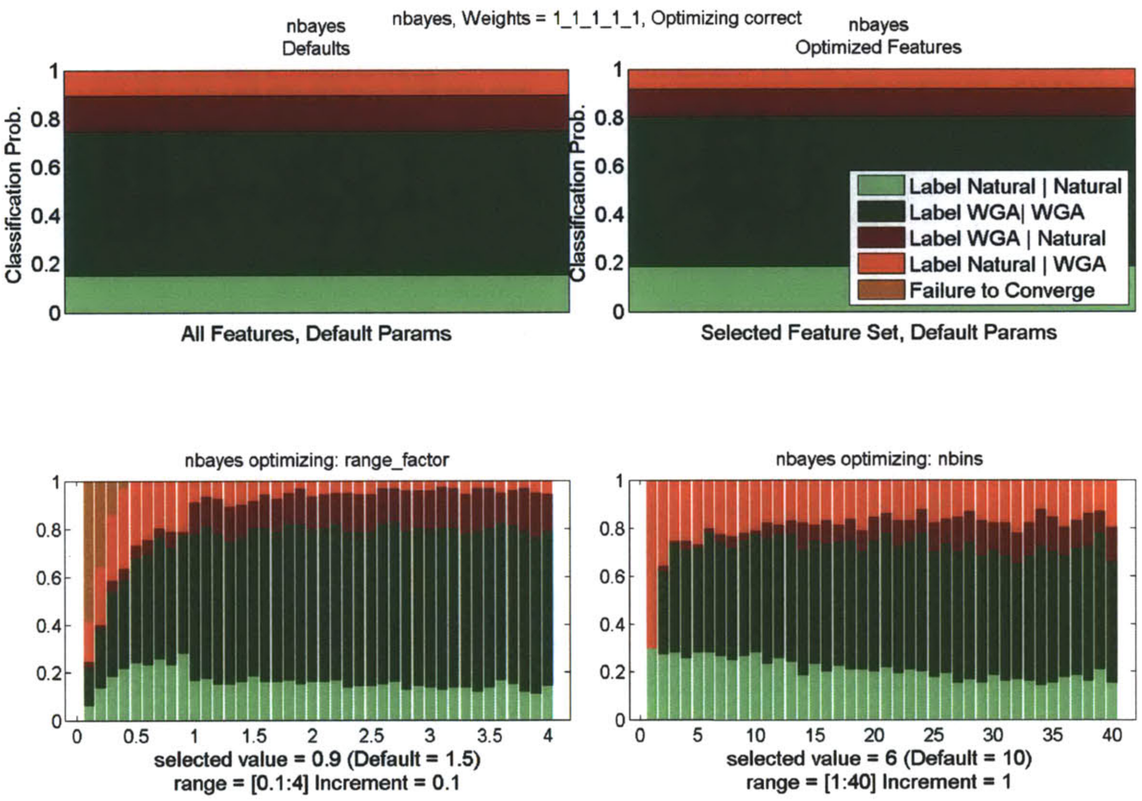


Figure C-7: *Naive Bayes Classifier with parameters range_factor, bins.*

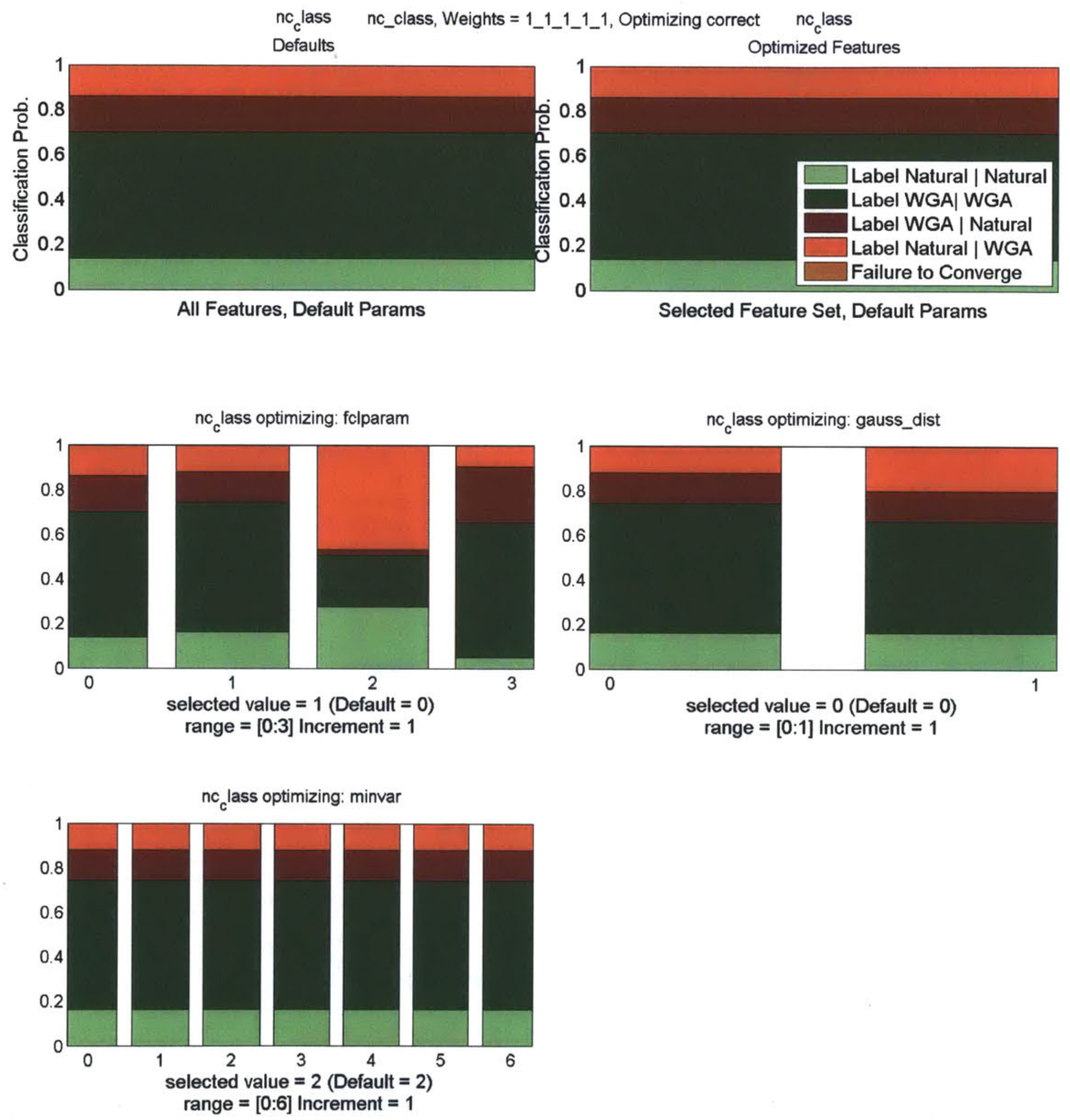


Figure C-8: *Nearest Cluster Classifier with parameters fclparam, gauss_dist, minvar.*

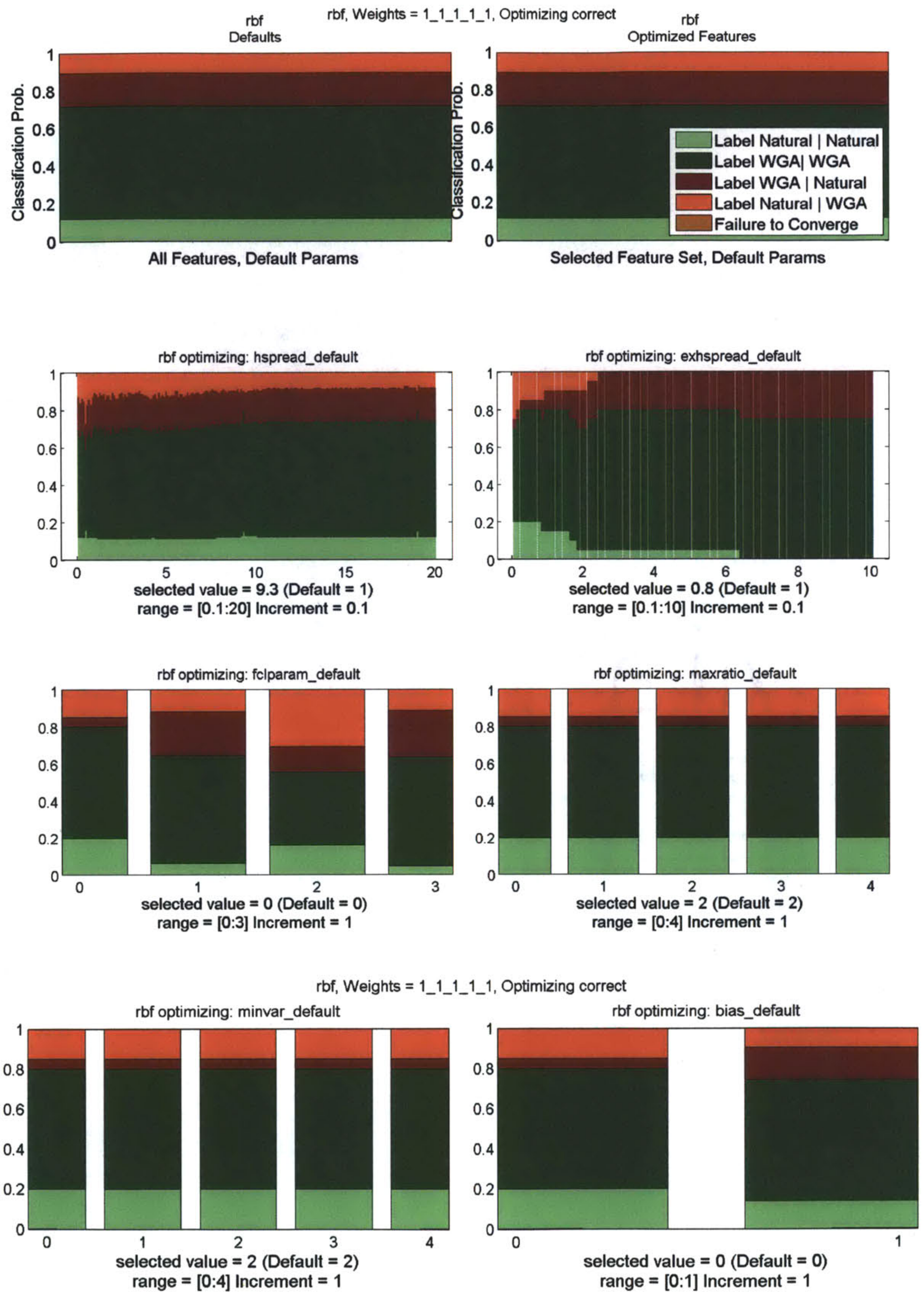


Figure C-9: *Radial Basis Function with parameters hspread_default, exhspread_default, fclparam_default, maxratio_default, minvar_default, bias_default.*

Appendix D

Combined Classifier Performance as a Function of Scoring Weights and Feature Optimization Parameters

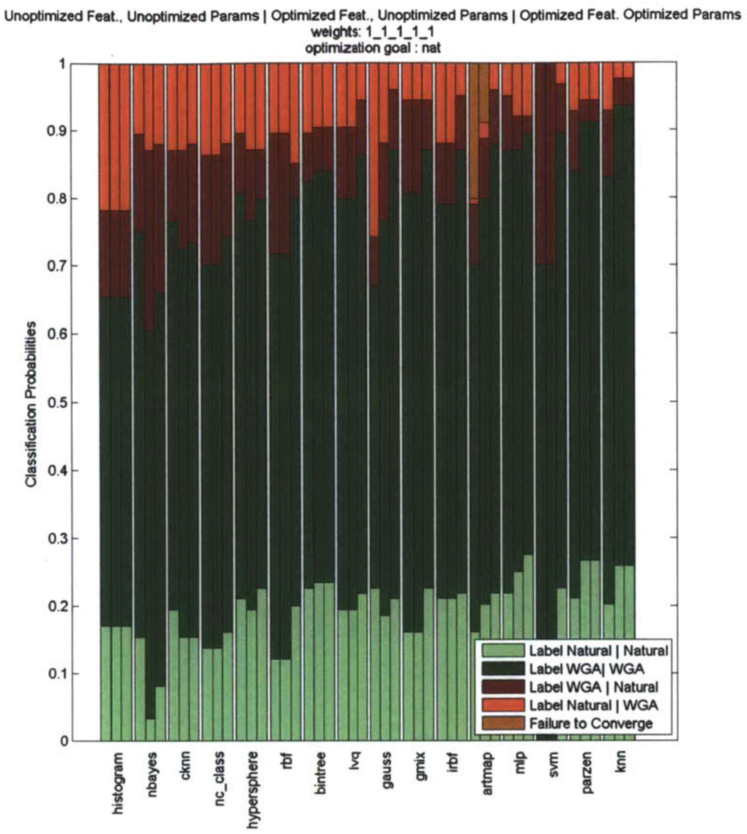


Figure D-1: *Identifiler, weights = [1,1,1,1,1], features optimized for "natural"*

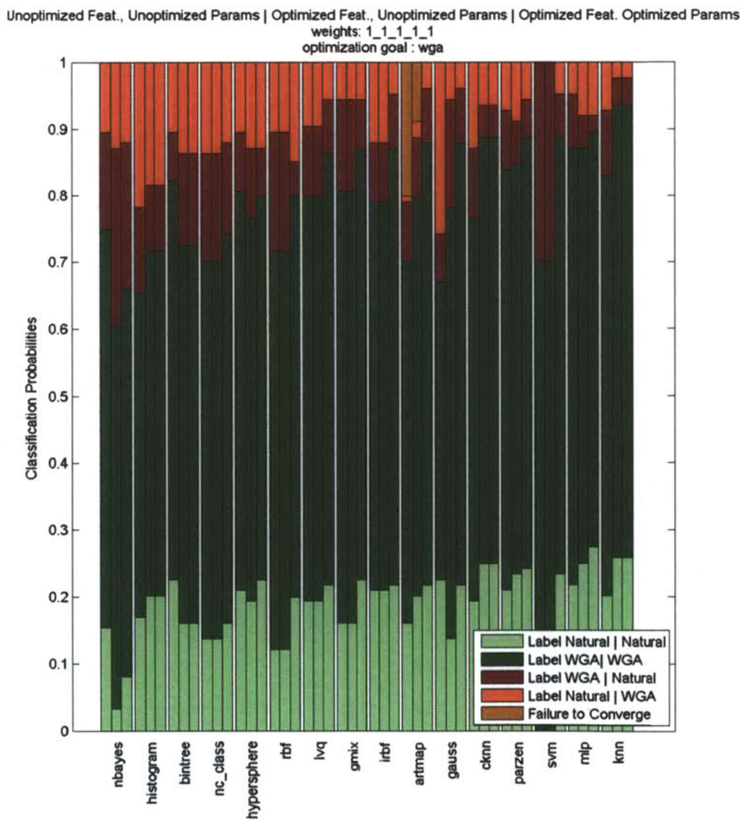


Figure D-2: *Identifiler, weights = [1,1,1,1,1], features optimized for "wga"*

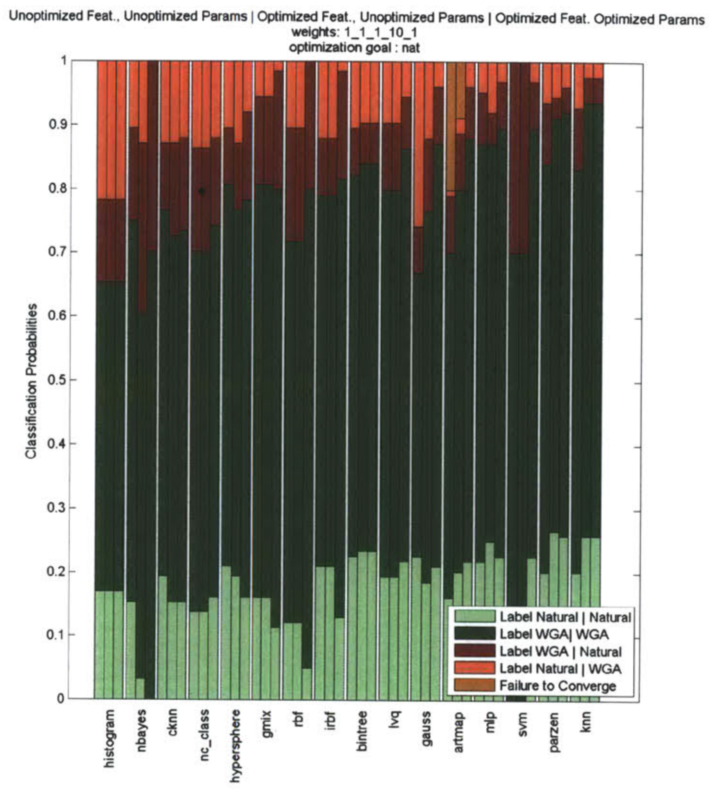


Figure D-3: *Identifiler, weights = [1,1,1,10,1], features optimized for "natural"*

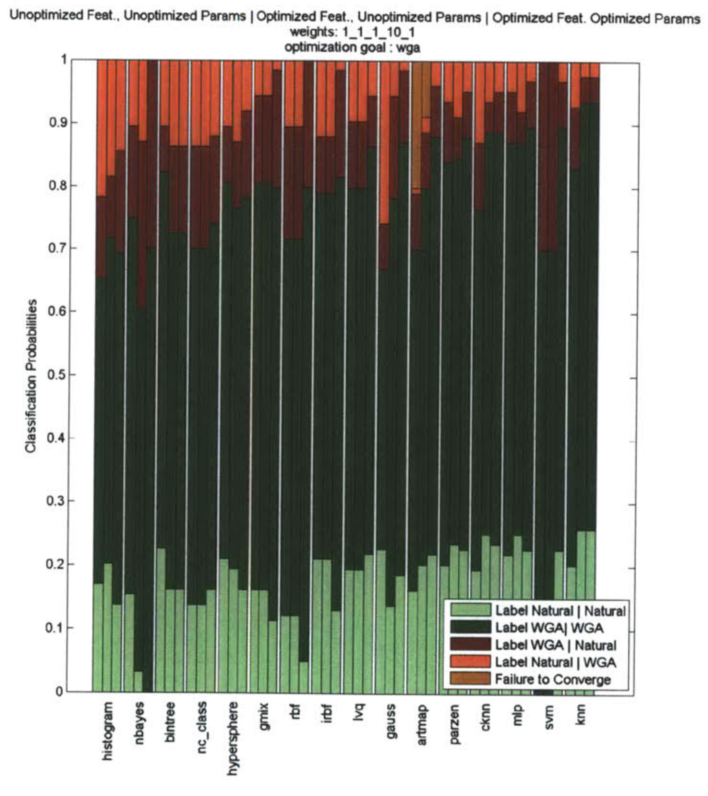


Figure D-4: *Identifiler, weights = [1,1,1,10,1], features optimized for "wga"*

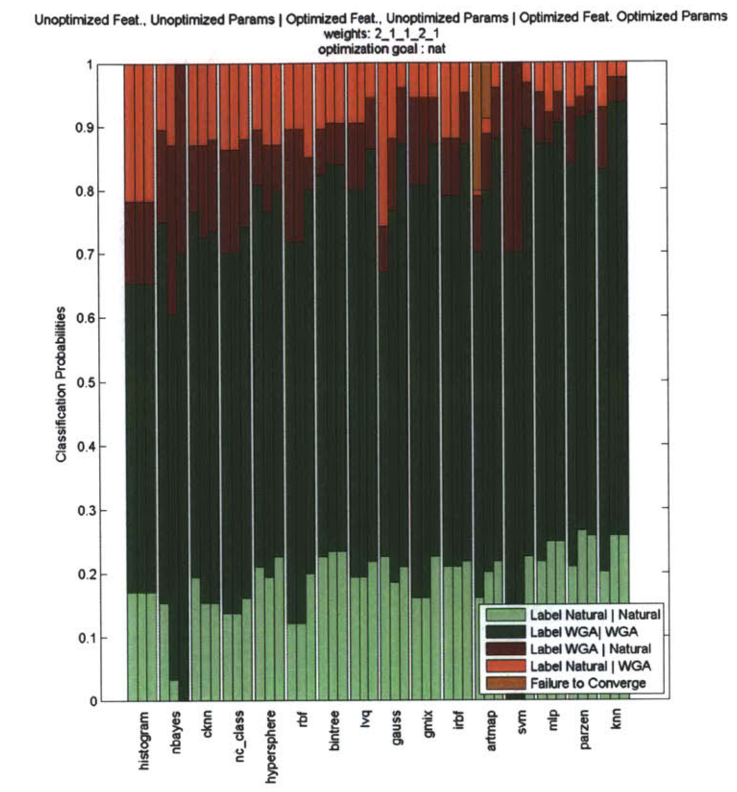


Figure D-5: *Identifiler, weights = [2,1,1,2,1], features optimized for "natural"*

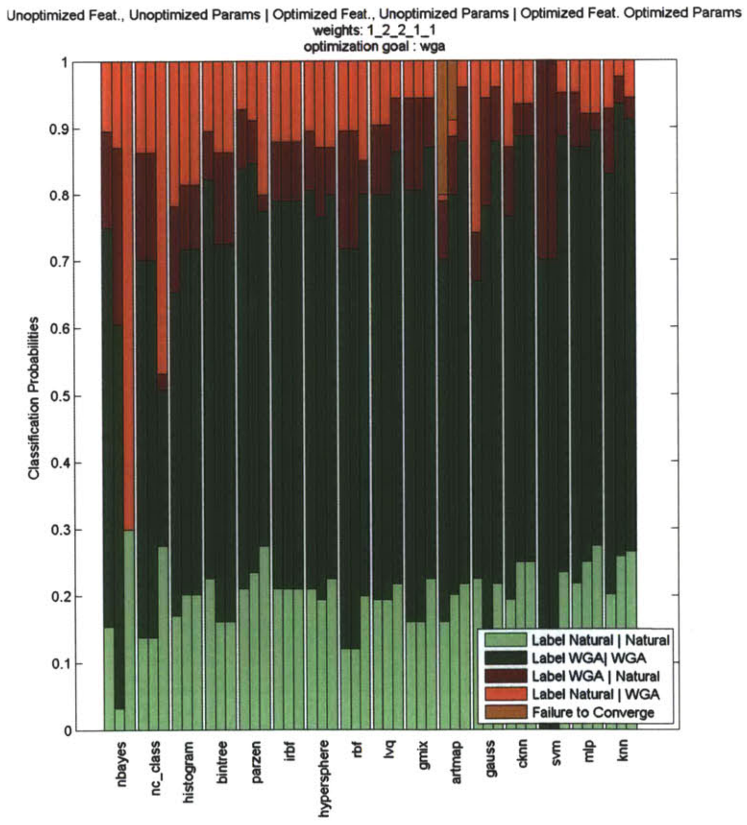


Figure D-6: *Identifiler, weights = [1,2,2,1,1], features optimized for "wga"*

Appendix E

Raw Feature Data for Powerplex Kit

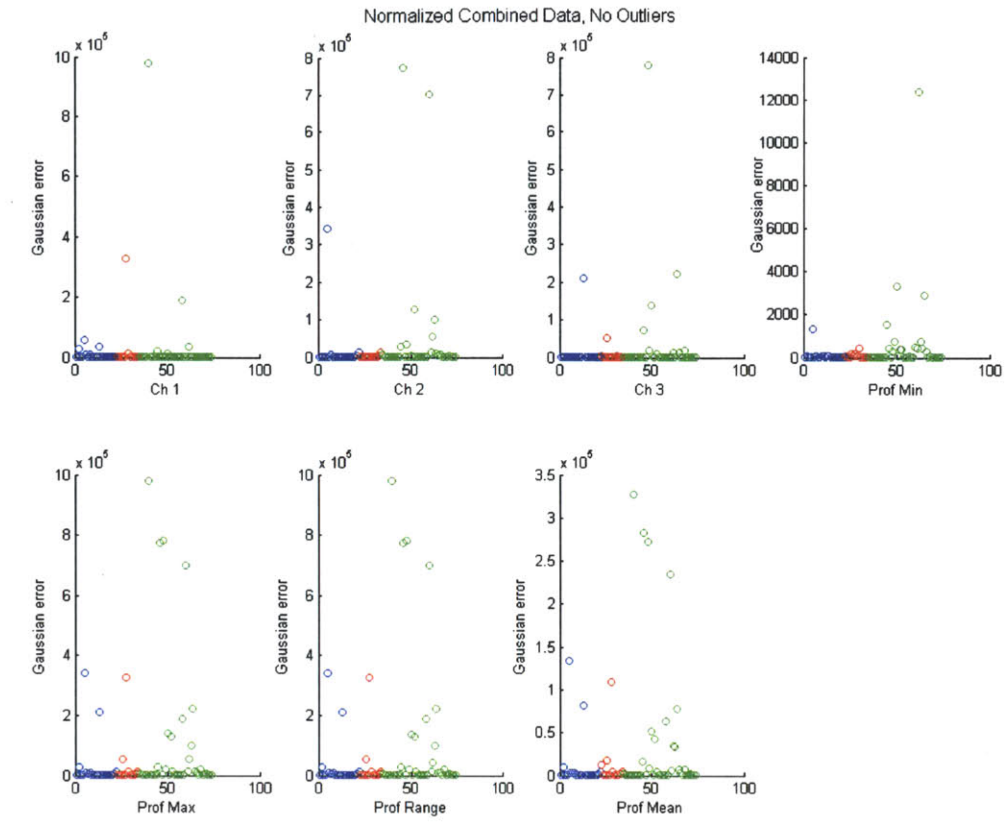


Figure E-1: *Gaussian Error For Each Channel and Profile (Normalized, No Outliers)*

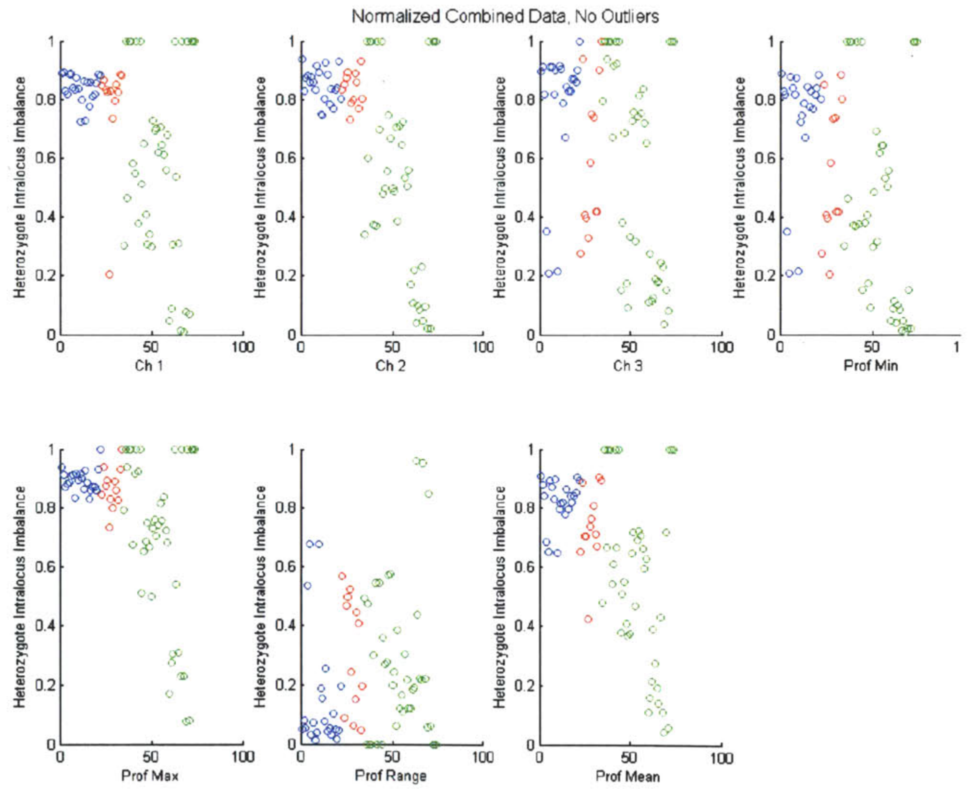


Figure E-2: *Heterozygote Intralocus Imbalance For Each Channel and Profile (Normalized, No Outliers)*

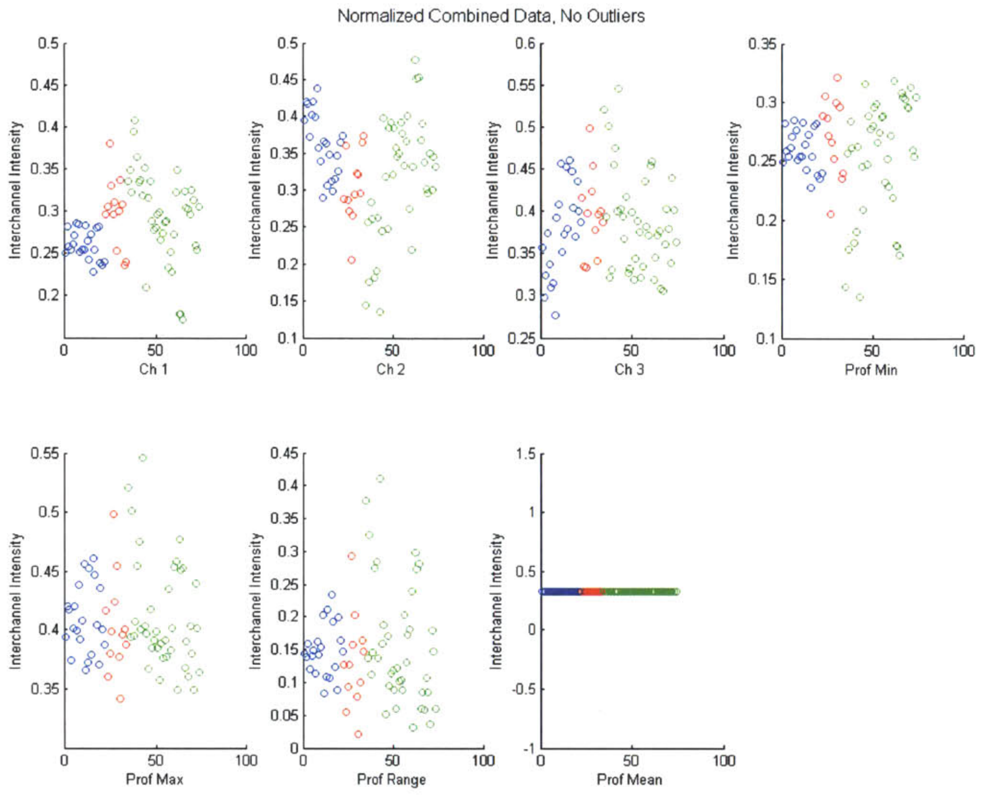


Figure E-3: *Interchannel Intensity For Each Channel and Profile (Normalized, No Outliers)*

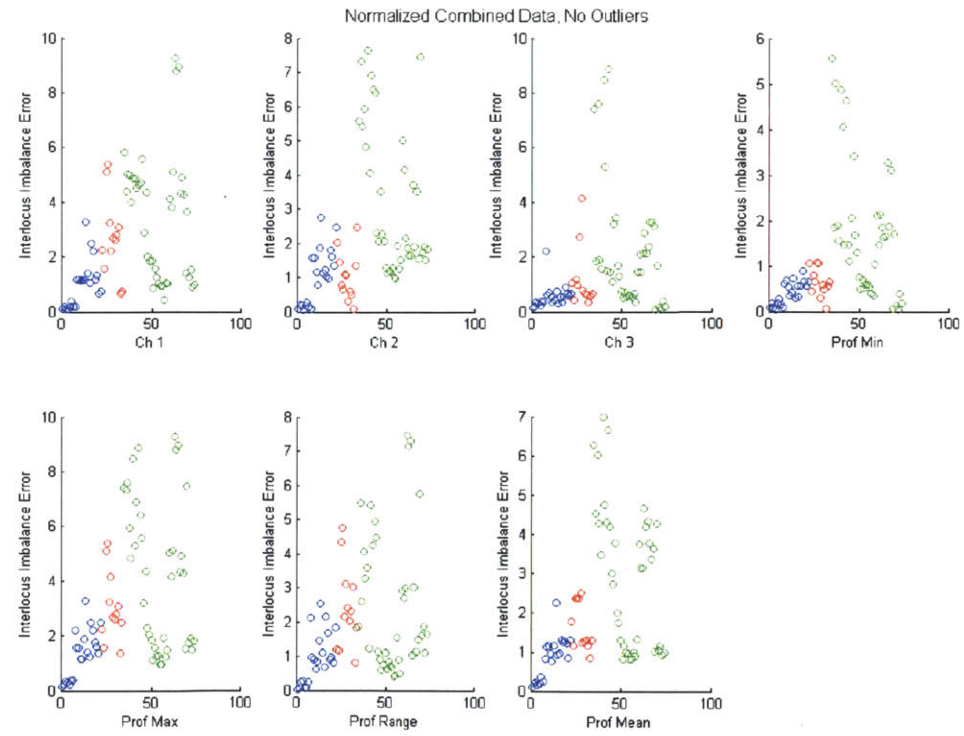


Figure E-4: *Interlocus Imbalance Error For Each Channel and Profile (Normalized, No Outliers)*

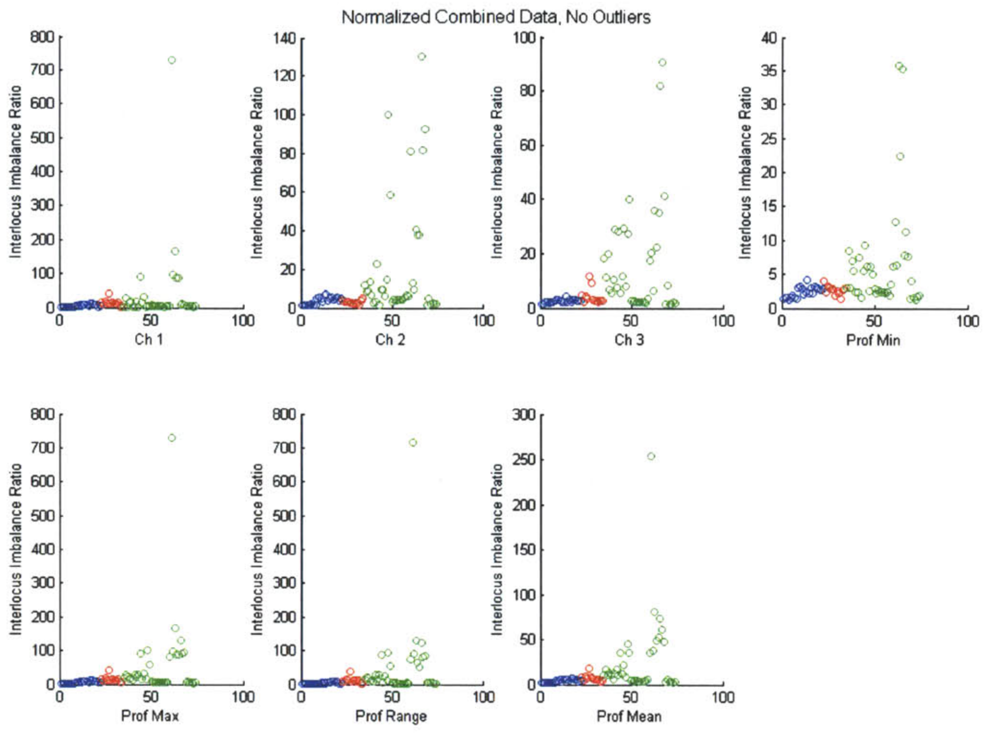


Figure E-5: *Interchannel Intensity For Each Channel and Profile (Normalized, No Outliers)*

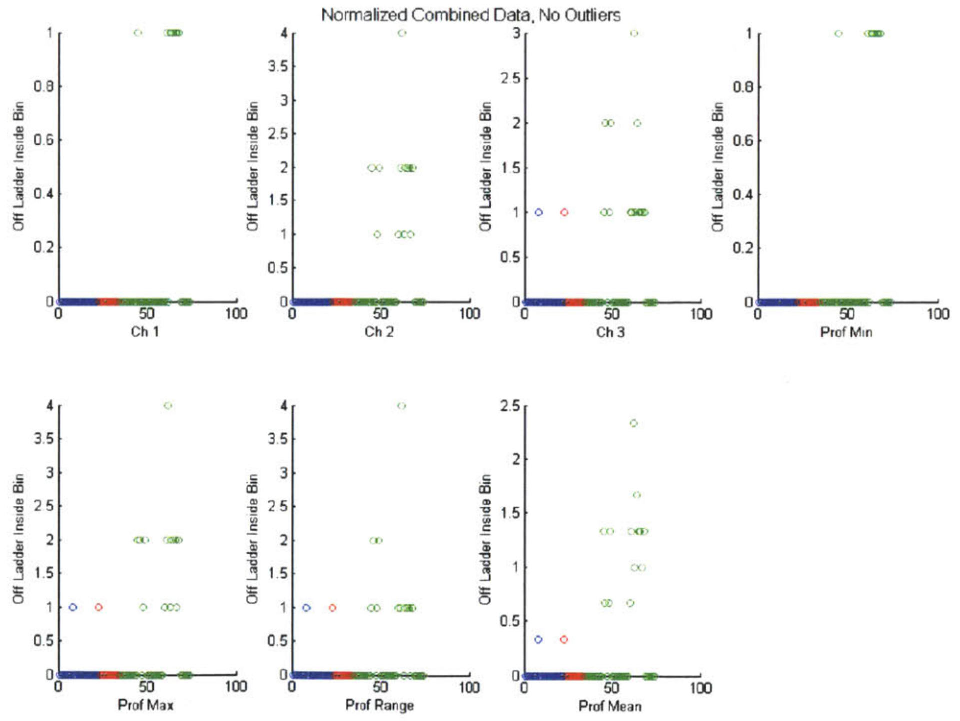


Figure E-6: *Off Ladder Inside Bin For Each Channel and Profile (Normalized, No Outliers)*

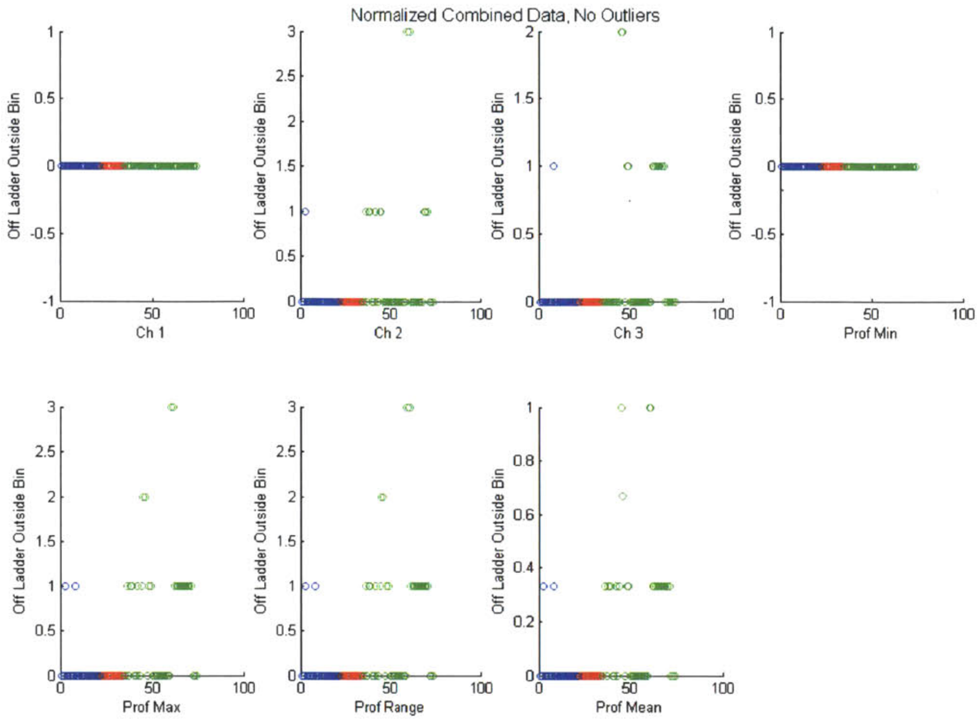


Figure E-7: *Off Ladder Outside Bin For Each Channel and Profile (Normalized, No Outliers)*

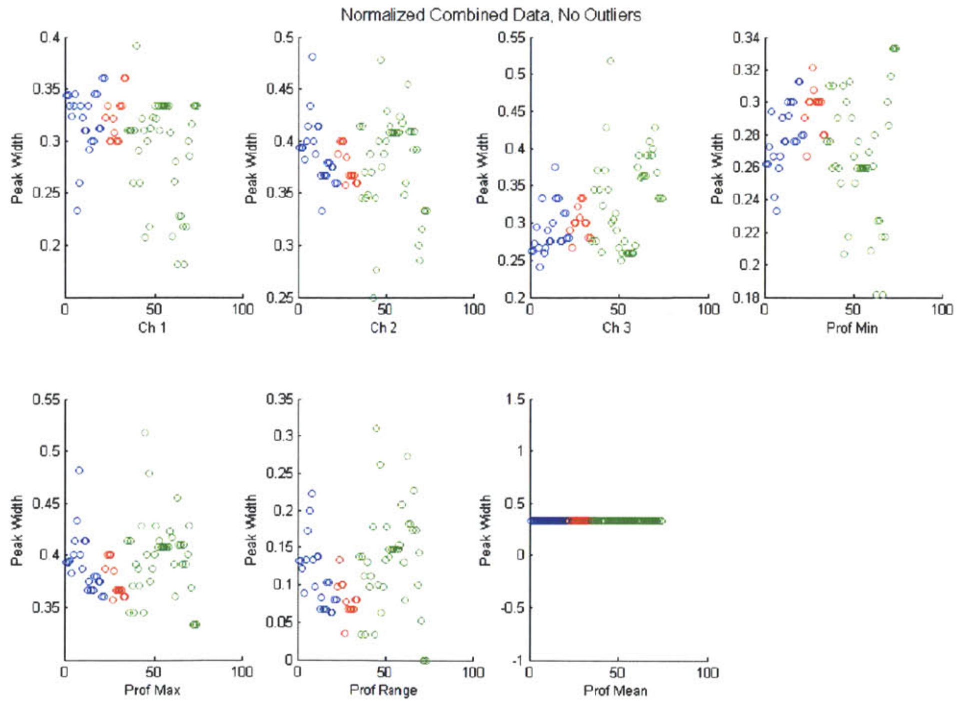


Figure E-8: *Peak Width For Each Channel and Profile (Normalized, No Outliers)*

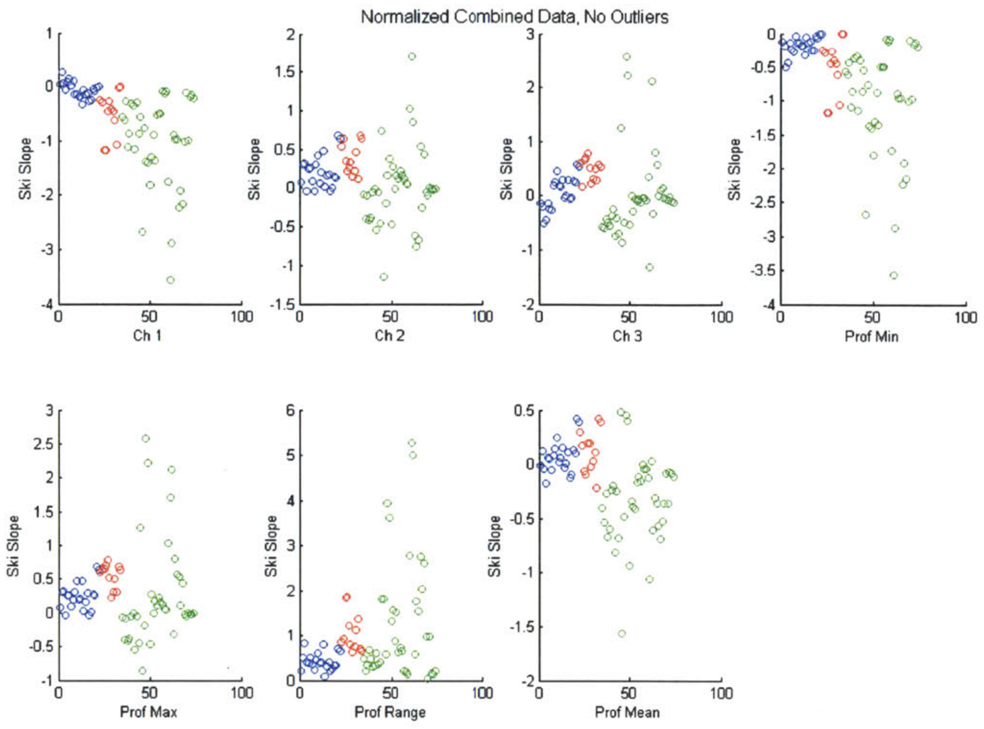


Figure E-9: *Ski Slope For Each Channel and Profile (Normalized, No Outliers)*

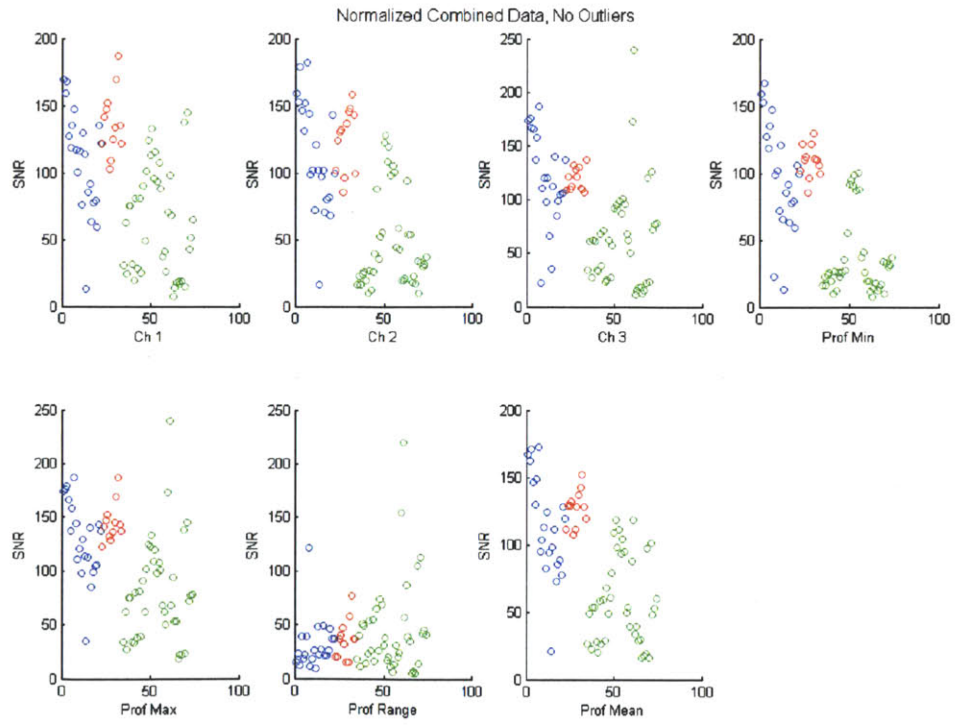


Figure E-10: *SNR For Each Channel and Profile (Normalized, No Outliers)*

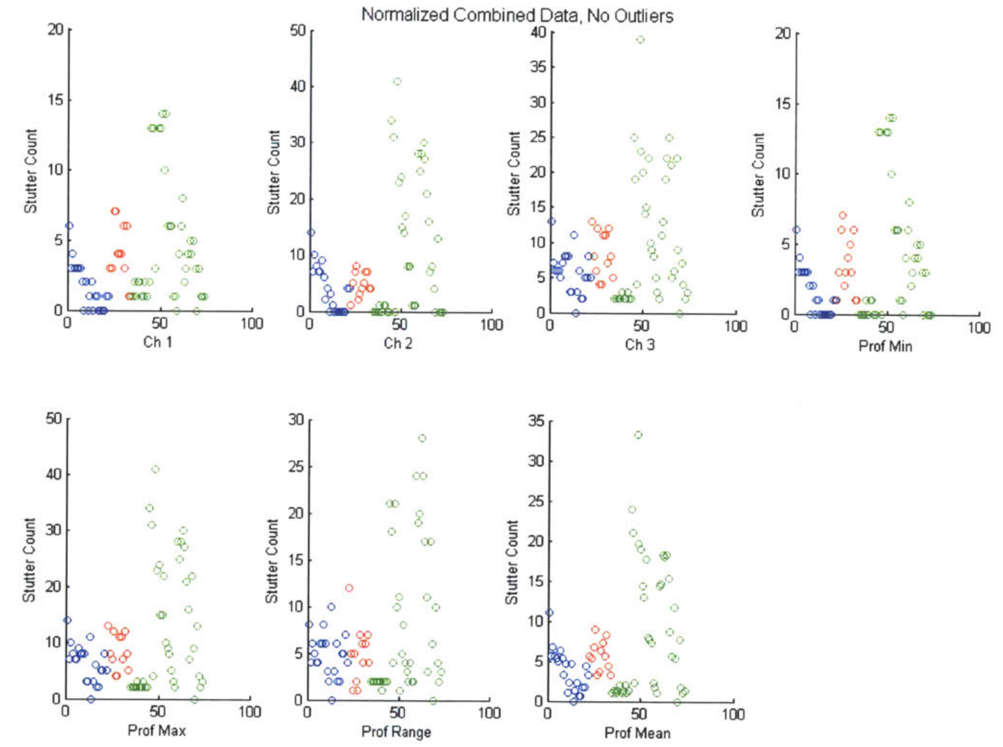


Figure E-11: *Stutter Count For Each Channel and Profile (Normalized, No Outliers)*

THIS PAGE INTENTIONALLY LEFT BLANK

Bibliography

[1] DNA Analyst Training. *NFSTC: Science Serving Justice*, Aug 2011. http://www.nfstc.org/pdi/Subject00/pdi_s00.htm .

[2] Biosafety. Web site, Mar 2012. http://www.cdc.gov/biosafety/.

[3] Combined DNA Index System (CODIS). Web site, Jan 2012. http://www.fbi.gov/about-us/lab/codis.

[4] *DNA Typing Protocols : Molecular Biology and Forensic Analysis.* Eaton Publishers, c2000.

[5] Applied Biosystems. *Applied Biosystems Genetic Analysis Data File Format*, Sep 2009.

[6] F Ausubel. *Short Protocols In Molecular Biology.* New York, 1992.

[7] C M Bishop. *Pattern Recognition and Machine Learning.* 2006. ISBN 0-387-31073-8.

[8] J M Butler. Allele Frequencies for 15 Autosomal STR Loci on U.S. Caucasian, African American, and Hispanic Populations. *J Forensic Sci*, 48:908–911, 2003.

[9] J M Butler. *Forensic DNA Typing: Biology, Technology, and Genetics of STR Markers.* Academic Press, 2 edition, 2005.

[10] J M Butler. Software Developed by the NIST Forensic/Human Identity Project Team. *Short Tandem Repeat DNA Internet Database*, Sep 2010. http://www.cstl.nist.gov/biotech/strbase/software.htm.

[11] M De Rycke L Van Haute A Van Steirteghem I Liebaers K Sermon C Spits, C Le Caignec. Whole-Genome Multiple Displacement Amplification from Single Cells. *Nature Protocols*, 1:1965 – 1970, 2006.

[12] D L Deuwer and J M Butler. Multiplex QA: an Exploratory Quality Assessment Tool for Multiplexed Electrophoretic Assays. *Electrophoresis*, 27:3735–3746, 2006.

[13] H Ellegren. Nature Reviews Genetics. 5:435–445, 2004.

[14] B Leclair et al. Systematic Analysis of Stutter Percentages and Allele Peak Height and Peak Area Ratios at Heterozygous STR Loci for Forensic Casework and Database Samples. *J Forensic Sci*, 49(5):1–12, Sep 2004.

[15] D Frumkin et al. Authentication of Forensic DNA Samples. *Forensic Sci. in. Genet.*, 2009. doi: 10.1016/j.fsigen.2009.06.009.

[16] D Shinde et al. Taq DNA Polymerase Slippage Mutation Rates Measured by PCR and Quasi-Likelihood Analysis: (CA/GT)n and (A/T)n Microsatellites. *Nucleic Acids Res.*, 31:974–980, 2003.

[17] F B Dean et al, editor. *Comprehensive Human Genome Amplification Using Multiple Displacement Amplifications*, number 99 in Proc. Natl. Acad. Sci. USA, 2002.

[18] J Gilder et al. Magnitude-Dependent Variation in Peak Height Balance at Heterozygous STR Loci. *International Journal of Legal Medicine.* doi 10.1007/s00414-009-0411-2.

[19] J Watson et al. *Recombinant DNA.* WH Freeman, NY, 1992.

[20] R Chakroborty et al. Electrophoresis. pages 1682–1696, 1999.

[21] Richard Lippmann et al. LNKnet: Neural Network, Machine-Learning, and Statistical Software for Pattern Classification. *The Lincoln Laboratory Journal*, 6(2):249–268, 1993.

[22] Robert Pinard et al. Assessment of Whole Genome Amplification-Induced Bias Through High-Throughput Massively Parallel Whole Genome Sequencing. *BMC Genomics*, 7:7:216, 2006.

[23] William Thompson et al. Evaluating Forensic DNA Evidence: Essential Elements of a Competent Defense Review. *The Champion*, 27(3):16–25, 2003.

[24] N Markuzon J Reynolds D Rosen G A Carpenter, S Grossberg. Fuzzy ARTMAP: A neural network architecture for incremental supervised learning of analog multidimensional maps. *IEEE Transactions on Neural Networks*, 3:698–713, 1992.

[25] Y. K. P. Lanlan Shen. Genome-Wide Profiling of DNA Methylation Reveals a Class of Normally Methylated CpG Island Promoters. *PLoS Genetics*, (10), 2007.

[26] R S Lasken and M Egholm. Whole Genome Amplification: Abundant Supplies of DNA from Precious Samples of Clinical Specimens. *Trends in Biotechnology*, 21:531–535, 2003.

[27] M Levitt. Forensic Databases: Benefits and Ethical and Social Costs. *Br. Med. Bull.*, 83:235–248, 2007.

[28] Jason Linville. Biology of STRs. Online Powerpoint, Aug 2011. www.dpo.uab.edu/ jglinvil/JS674web/JS674SP07Istrbiology.ppt.

[29] Richard Lippmann. Committees, Bagging, Random Forests. Pattern Classification and Machine Learning Class, May 2011.

[30] Richard Lippmann. Feature Selection and Projection. Pattern Classification and Machine Learning Class, march 2011.

[31] Richard Lippmann. Local Neural Net Classifiers. Pattern Classification and Machine Learning Class, Apr 2011.

[32] Richard Lippmann. Pattern Classification and Machine Learning. Pattern Classification and Machine Learning Class, Feb 2011.

[33] Richard Lippmann. Selecting and Comparing Classifiers. Pattern Classification and Machine Learning Class, May 2011.

[34] Richard Lippmann. Statistical Classifiers. Pattern Classification and Machine Learning Class, Feb 2011.

[35] Richard Lippmann. Training, Tuning, and Regularization. Pattern Classification and Machine Learning Class, Mar 2011.

[36] A Durand L Duponchel J Huvenne O Devos, C Ruckebush. Support vector machines (SVM) in near infrared (NIR) spectroscopy: Focus on parameters optimization and model interpretation. *Chemometrics and Intelligent Laboratory Systems*, 96(1):27 - 33, 2009.

[37] N Riedel P Zheng, J Peng. Finite Sample Error Bounds for Parzen Windows. *Journal of Machine Learning Research*, 1(48), 2000.

[38] D Stork R O Duda, P E Hart. *Pattern Classification.*

[39] Stephen Sherry and Lisa Forman. Expert Systems for Forensic Sample Analysis: Incorporating Lessons Learned Into Next Generation Software. Online Powerpoint, Aug 2011. ftp://ftp.ncbi.nlm.nih.gov/pub/forensics/CHI_2004_forensics.ptt .

[40] D Storti. UNESCO and Information Processing Tools: IDAMS Statistical Software. Web site, Mar 2011. http://portal.unesco.org/ci/en/ev.php.

[41] Nicola Vitacolonna. Bio-Trace-ABIF-1.05: Perl Extension for Reading and Parsing ABIF (Applied Biosystems, Inc. Format) Files. Computer Software, Feb 2010.

[42] C Word. Peak Height Ratios. Online Powerpoint, Oct 2010. http;//www.cstl.nist.gov/biotech/strbase/training.htm.

[43] Z R Yang. *Machine Learning Approaches to Bioinformatics.* World Scientific, Hackensack, N.J., 2010.