

University of Montana

ScholarWorks at University of Montana

Graduate Student Theses, Dissertations, &
Professional Papers


Graduate School

2017

K-MER ANALYSIS PIPELINE FOR CLASSIFICATION OF DNA SEQUENCES FROM METAGENOMIC SAMPLES

Russell Kaehler

Follow this and additional works at: <https://scholarworks.umt.edu/etd>

 Part of the [Artificial Intelligence and Robotics Commons](#), [Bioinformatics Commons](#), [Computational Biology Commons](#), [Numerical Analysis and Scientific Computing Commons](#), [Other Ecology and Evolutionary Biology Commons](#), [Other Genetics and Genomics Commons](#), and the [Software Engineering Commons](#)

Let us know how access to this document benefits you.

Recommended Citation

Kaehler, Russell, "K-MER ANALYSIS PIPELINE FOR CLASSIFICATION OF DNA SEQUENCES FROM METAGENOMIC SAMPLES" (2017). *Graduate Student Theses, Dissertations, & Professional Papers*. 10967.

<https://scholarworks.umt.edu/etd/10967>

This Thesis is brought to you for free and open access by the Graduate School at ScholarWorks at University of Montana. It has been accepted for inclusion in Graduate Student Theses, Dissertations, & Professional Papers by an authorized administrator of ScholarWorks at University of Montana. For more information, please contact scholarworks@mso.umt.edu.

**K-MER ANALYSIS PIPELINE FOR CLASSIFICATION OF DNA
SEQUENCES FROM METAGENOMIC SAMPLES**

By

Russell Matthew Kaehler

Bachelor of Science, The Evergreen State College, Olympia, WA, 2010

Thesis

Presented in partial fulfillment of the requirements
for the degree of

Master of Science
in Computer Science

The University of Montana
Missoula, MT

Spring 2017

Approved by:

Scott Whittenburg Ph.D., Dean
Graduate School

Douglas W. Raiford Ph.D., Chair
Computer Science

Alden H. Wright Ph.D.
Computer Science

William E. Holben Ph.D.
Biological Sciences

© COPYRIGHT

by

Russell Matthew Kaehler

2017

All Rights Reserved

K-Mer Analysis Pipeline for Classification of DNA Sequences from Metagenomic Samples

Chairperson: Douglas W. Raiford

Biological sequence datasets are increasing at a prodigious rate. The volume of data in these datasets surpasses what is observed in many other fields of science. New developments wherein metagenomic DNA from complex bacterial communities is recovered and sequenced are producing a new kind of data known as metagenomic data, which is comprised of DNA fragments from many genomes. Developing a utility to analyze such metagenomic data and predict the sampleclass from which it originated has many possible implications for ecological and medical applications.

Within this document is a description of a series of analytical techniques used to process metagenomic data in such a way that it is transformed from the raw sequence information into a reusable data structure that can be processed by feature selection techniques and machine learning algorithms. Analysis and transformation of the data from the raw sequences to a reusable structure is done using k length substrings of DNA, known as k -mers, and storing the count of these observed strings in a Numeric Summarization Vector (NSV).

The technique described herein is offered as a proof of concept for research into analyzing metagenomic data without identifying individual organisms contained within the sample. It is tested using leave-one-out and Monte Carlo cross-validation, while varying numerous parameters and verifying the results by using a large pool of independent experiments initiated with the same starting parameters. The pipeline is validated against multiple data sets using two- and three-class problems. Results are presented showing the accuracy as a function of multiple parameters that can be selected by a user of the pipeline. This work shows that there may be a way to process metagenomic data in near real time to analyze and predict the environmental class of a sample with reasonable accuracy. Consider the difficulty in distinguishing the difference between a healthy and diseased gut microbiome, this approach can classify sample data as belonging to one of those states.

ACKNOWLEDGMENTS

This document is the product of years of coordinated work. Everyone who was a regular member of the Metagenomics research group meetings from 2012 through 2014 gave important feedback and insights to this research. Doctors Holben, Raiford, and Wright all provided guidance and support to ensure that this research expands the overall knowledge of the scientific community.

These words are being written at a much later time than anyone of us would have initially expected. All events in life are extremely complicated and difficult to predict. In my own life, I have walked an interesting road. It is with great joy that I'm able to finish this work and move onto a new voyage in my life. My friends, family, peers, and advisors from all over the world have spent many hours listening to me discuss these topics and now, with the greatest pleasure, I'm able to deliver this work so that we might have some written starting point to continue our discussions in the future.

In all of this, Dr. Douglas Raiford patiently guided and helped me. It was his generosity that initially brought me into the metagenomics world, a place far from where I had expected to swim. Through Dr. Raiford's keen insight and holistic worldview, I have been afforded a deeper understanding of, and communication with, a multidisciplinary research community; this collaboration has been essential for my metagenomics research.

Finally, I would like to take a moment to thank the great institution of the Uni-

versity of Montana - Missoula. Learning here has been a great pleasure. The people and faculty are exceptional. It's been said that raising a child takes a village. I believe something similar might apply in the creation and delivery of a thesis. Being surrounded by an environment of inquisitive minds most certainly increased my own curiosity about this subject and the larger world.

TABLE OF CONTENTS

COPYRIGHT	ii
ABSTRACT	iii
ACKNOWLEDGMENTS	iv
CODE LISTINGS	ix
LIST OF FIGURES	x
LIST OF TABLES	xiii
CHAPTER 1 INTRODUCTION	1
1.0.1 Case Studies & Datasets	3
1.1 Goal	4
1.2 Benefits	6
1.3 Thesis Organization	7
CHAPTER 2 LITERATURE REVIEW	8
2.1 Search Alignment Based Sequence Analysis	9
2.2 Search Alignment free Sequence Analysis	11
2.3 Machine Learning	12
CHAPTER 3 METHODS	16
3.1 Methods Syntax & Conventions	17

3.2	Initial Methods	18
3.3	Exploring Choices for k and Window Overlap	25
3.4	Feature Selection	27
3.5	Machine Learning Algorithms	29
3.5.1	KNN	29
3.5.2	Naïve Bayes	30
3.6	Validation	34
3.7	Data Processing Pipeline	37
3.7.1	Synthetic Data	39
3.7.2	Collecting Raw Records into a Single Pool and Subsequent Random Division	39
3.7.3	Translation of FASTA files into NSVs	42
3.8	Datasets	42
CHAPTER 4 RESULTS		45
4.1	Johnson Grass Dataset	45
4.1.1	k -NN Results	47
4.1.2	Naïve Bayes Classifier Results	51
4.1.2.1	Two Class Problem	51
4.1.2.2	Three Class Problem LOOCV	54
4.1.2.3	7-Mer & 5 Dimensions LOOCV	54
4.1.2.4	Three Class Problem MCCV	56
4.1.2.5	3-Mer & 5 Dimensions	58
4.1.2.6	3-Mer & 10 Dimensions	60
4.1.2.7	4-Mer & 5 Dimensions	62
4.1.2.8	4-Mer & 10 Dimensions	64

4.1.2.9	5-Mer & 5 Dimensions	66
4.1.2.10	5-Mer & 10 Dimensions	68
4.1.2.11	6-Mer & 5 Dimensions	70
4.1.2.12	6-Mer & 10 Dimensions	73
4.1.2.13	7-Mer & 5 Dimensions	75
4.1.2.14	7-Mer & 10 Dimensions	78
4.1.2.15	8-Mer & 5 Dimensions	80
4.1.2.16	3-Mer & 10 Dimensions	83
4.2	Human Gut Dataset	85
4.2.0.1	8-Mer & 5 Dimensions Leave-One-Out	87
CHAPTER 5 DISCUSSION		90
5.1	General Discussion	90
5.1.1	Accuracy as a Function of k -mer Size	91
5.1.2	Dimensions Selection Frequency Across Experiments	92
5.1.3	Dimension Selection and Observed k -Mer Count	93
5.2	Future Directions	93
5.3	Conclusions	95
BIBLIOGRAPHY		97

CODE LISTINGS

3.1	Processing a FASTA File into an NSV	20
3.2	Find the Correct k -Mer Index in an NSV	21
3.3	Normalizing an NSV	22
3.4	All Pairwise Student's T-Test	28
3.5	Calculate Manhattan Distance	30
3.6	Bayes' Rule Classifying Test Records	32
3.7	Bin all Values for all Records	33
3.8	Find Correct Bin for Value in Dimension	34
3.9	Mix Files into Testing and Training Sets by Class	36
3.10	Randomly Split Fragments into Files	41

LIST OF FIGURES

Figure 3.1	Visual Representation of the Data Processing Pipeline	38
Figure 4.1	Confusion Matrix 5-Mers 10 Dimensions 18 Trials SelectKBest Feature Selection	47
Figure 4.2	Bar plot Analyzing Consistency of Feature Selection Frequency of Five Mers & Ten Dimensions	48
Figure 4.3	Confusion Matrix 5-Mers 5 Dimensions 16 Trials	49
Figure 4.4	Bar plot Analyzing Consistency of Feature Selection On 5- mers & 5 Dimensions	50
Figure 4.5	Confusion Matrix 5-Mers 10 Dimensions 25 Trials Two Classes	52
Figure 4.6	Bar plot Analyzing Consistency of Feature Selection On 5- mers & 10 Dimensions Two Classes	53
Figure 4.7	Confusion Matrix 7-Mers 5 Dimensions 20 Trials	54
Figure 4.8	Bar plot Analyzing Consistency of Feature Selection On 7- mers & 5 Dimensions	55
Figure 4.9	Confusion Matrix 3-Mers 5 Dimensions 20 Trials	58
Figure 4.10	Bar plot Analyzing Consistency of Feature Selection On 3- mers & 5 Dimensions	59
Figure 4.11	Confusion Matrix 3-Mers 10 Dimensions 20 Trials	60

Figure 4.12	Bar plot Analyzing Consistency of Feature Selection On 3- mers & 10 Dimensions	61
Figure 4.13	Confusion Matrix 4-Mers 5 Dimensions 20 Trials	62
Figure 4.14	Bar plot Analyzing Consistency of Feature Selection On 4- mers & 5 Dimensions	63
Figure 4.15	Confusion Matrix 4-Mers 10 Dimensions 20 Trials	64
Figure 4.16	Bar plot Analyzing Consistency of Feature Selection On 4- mers & 10 Dimensions	65
Figure 4.17	Confusion Matrix 5-Mers 5 Dimensions 20 Trials	66
Figure 4.18	Bar plot Analyzing Consistency of Feature Selection On 5- mers & 5 Dimensions	67
Figure 4.19	Confusion Matrix 5-Mers 10 Dimensions 20 Trials	68
Figure 4.20	Bar plot Analyzing Consistency of Feature Selection On 5- mers & 10 Dimensions	69
Figure 4.21	Confusion Matrix 6-Mers 5 Dimensions 20 Trials	70
Figure 4.22	Bar plot Analyzing Consistency of Feature Selection On 6- mers & 5 Dimensions	71
Figure 4.23	6-mers & 5 Dimensions Scatter Plot Dimension Frequency vs Raw Count in Dimension	72
Figure 4.24	Confusion Matrix 6-Mers 10 Dimensions 20 Trials	73
Figure 4.25	Bar plot Analyzing Consistency of Feature Selection On 6- mers & 10 Dimensions	74
Figure 4.26	Confusion Matrix 7-Mers 5 Dimensions 20 Trials	75
Figure 4.27	Bar plot Analyzing Consistency of Feature Selection On 7- mers & 5 Dimensions	76

Figure 4.28	7-mers & 5 Dimensions Scatter Plot Dimension Frequency vs Raw Count in Dimension	77
Figure 4.29	Confusion Matrix 7-Mers 10 Dimensions 20 Trials	78
Figure 4.30	Bar plot Analyzing Consistency of Feature Selection On 7- mers & 10 Dimensions	79
Figure 4.31	Confusion Matrix 8-Mers 5 Dimensions 20 Trials	80
Figure 4.32	Bar plot Analyzing Consistency of Feature Selection On 8- mers & 5 Dimensions	81
Figure 4.33	8-mers & 5 Dimensions Scatter Plot Dimension Frequency vs Raw Count in Dimension	82
Figure 4.34	Confusion Matrix 8-Mers 10 Dimensions 20 Trials	83
Figure 4.35	Bar plot Analyzing Consistency of Feature Selection On 8- mers & 10 Dimensions	84
Figure 4.36	Confusion Matrix 8-Mers 5 Dimensions 20 Trials	87
Figure 4.37	Bar plot Analyzing Consistency of Feature Selection On 8- mers & 5 Dimensions	88
Figure 4.38	8-mers & 5 Dimensions Scatter Plot Dimension Frequency vs Raw Count in Dimension	89

LIST OF TABLES

Table 3.1	Rapid growth of dimensions as a function of k	26
Table 3.2	Number of lines in the Human Gut data set.	43
Table 3.3	Saliva samples and associated class values	44
Table 4.1	Number of lines in Johnson Grass Dataset	46
Table 4.2	Varying k -mer size using Student's t-test and the Naive Bayes Classifier	57
Table 4.3	Number of lines in Human Gut Dataset	86

CHAPTER 1 INTRODUCTION

DNA sequencing technology is continually increasing in throughput and availability while simultaneously decreasing in cost. Sequencing is being realized as an essential tool for the research community to study complex environmental and biological problems [1]. Samples containing DNA fragments from many organisms in an ecosystem are known as metagenomic DNA samples. Scientists can rapidly generate genomic data from whole microbial communities. Microbial community analysis is increasingly seen as an additional resource to evaluate a larger system's ecological structures in study systems such as the Human Microbiome Project [2] and the Earth Microbiome Project [3]. Researchers are intently working to assemble comprehensive views of the organisms present in an environment and their corresponding genomic sequences and genomes using rigorous systematic methods.

Predictions of microbial community composition from virtually any environmental sample can be produced by sequence analysis of metagenomic DNA samples. Community models can be developed by using the frequency-of-fragments (FOF) associated with specific organisms as identified by programs such as BLAST [4]. Frequency-of-fragment methods laboriously compare each sequence in a sample to every known organism in a database looking for the best matches between a sample fragment and a known reference genome. Now FOF techniques are the primary method used to build models of microbial communities, but they are currently not used to predict what

type of environmental source produced a given sample. Metagenomic FOF models have been used to study changes in the gut microbiome because of a change of diet [5]. Such results demonstrate that a change in the FOF in a microbiome can be used to clearly demonstrate a shift in the microbial community. Applying soft computing methods and techniques, which are different from traditional computing methods as they allow for inexact or probabilistic solutions, may allow microbial biologists to access portions of the underlying information in their datasets without having to spend as many processing cycles to generate conclusions.

Historically, limitations of computer hardware and software have made applying computer science techniques to analyze biological data sets intractable. Advances in computer physical resources, along with the ability to combine multiple machines into larger groups, allows for rapid analysis of large data sets by distributing the problem through multiple computers or clusters. Machine learning, an artificial intelligence data analysis method, utilizes algorithms that iteratively learn from data. Applications of machine learning are widespread and diverse, including more enigmatic studies such as identifying artistic influences in paintings [6]. By employing machine learning, we now have the necessary elements to begin to study genomic sequences in a much richer way [7].

Bioinformatic research has identified sliding window k -mers, also known as l -, p -, and n -mers, as a useful abstraction for fragments of a genomic sequence to identify the source organism [8] [9] [10]. Existing research publications sharing results from the scientific community reveal that machine learning techniques can successfully classify individual species using metagenomic sequence fragments using k -mer frequency profiles [11]. Such an approach is not designed to answer questions about the kind of environment that produced the metagenomic sample. A different approach is needed to use k -mers in a way that would allow for the classification of a metagenomic sample

back to the kind, or class, of environment that generated the sample.

Both the concept of Numeric Summarization Vectors (NSV)s and k -mers are key components used in this research. A somewhat more technical description of how k -mers and NSVs are related is provided here to facilitate the understanding of the computational complexity dealing with these sequences and vectors at a large scale. K -mers are substrings of a genomic sequence, where k is replaced with an integer number. The number of possible mers is directly related to the choice of k . DNA has four bases and the possible number of mers is 4^K for any integer k . As k increases in size, the possibility that any given mer will be randomly present in a sample dramatically decreases. As k increases, the number of cells that must be allocated in an array to store mer frequency increases exponentially. This is because there is one unique cell in the NSV for each possible mer. The total number of cells or indices in the NSV is the same as 4^K for all choices of k for the k -mer size used to process the metagenomic sample. The mers are stored in the numeric summarization vector (NSV) array of length 4^K using an approach similar to the one pioneered in the R package QuasiAlign [12].

1.0.1 Case Studies & Datasets

In the current work, varying the choice of k in k -mers is explored as a key factor in classification accuracy of metagenomic samples back to the environmental class where the sample was taken. By using multiple feature selection techniques and machine learning algorithms, the change in accuracy is studied in more ways than varying the choice of k alone. The results presented in this document show that varying the number of dimensions used while holding k constant also has an impact on accuracy.

Sorghum halepense (Johnson Grass) is a plant species native to the Mediterranean area, but can now be found as an invasive plant on all continents of the planet

except Antarctica. Johnson Grass can reproduce via both seeds and rhizomes (a subterranean stem capable of producing new roots and stems). Like many invasive species, it uses a combination of methods to overtake an area from native plant species. There is an active research community studying the microbiome of Johnson Grass to understand how the microbial community in and around the rhizosphere assists Johnson Grass in invading a new area [13]. The following research proposes that machine learning and pattern recognition techniques can analyze k -mers from an environmental sample to predict the class of the research site used to produce the sample. The Holben Microbial Ecology Laboratory at the University of Montana-Missoula extracted and sequenced metagenomic DNA from three classes of samples (non-invaded, partially invaded, and fully invaded) obtained from a Johnson Grass invasion study site in Texas. This data set was generously provided by the Holben Microbial Ecology Laboratory for use as one of the data sets for this research. Another dataset used in this research project is a collection of sequences from the Human Gut samples from the Short Read Archive (SRA) operated by the National Center for Biotechnology Information (NCBI) [14] [15]. This dataset is smaller and discussed in greater depth in the Datasets section in Chapter Three.

1.1 Goal

The aim of this thesis is to demonstrate the viability of a new technique using a clear data processing pipeline to rapidly analyze metagenomic DNA samples. The specific goal will be to predict the type of environment that generated a metagenomic sample. This is different from other approaches. Many metagenomic analysis techniques rely on identifying the taxa in the sample and then making comparisons between samples based on the relative difference in taxon abundance. Such approaches are limited in

part by the low number of species that have had their genomes fully sequenced and put into public databases. Consequently, another shortfall of this approach is that it may leave out a large fraction of the sequence fragments as they don't match known taxa in databases. As such, attempting to predict the environmental class of a given sample using all available sequence data without identifying taxa should allow for faster sample analysis using less computational resources.

A critical component of this research relied on the new idea that a Numeric Summarization Vector (NSV), which represents a histogram of k -mers from a metagenomic Fast Adaptive Shrinkage Thresholding Algorithm (FASTA) file may abstract the underlying DNA while providing a useful representation of the data for prediction. In constructing the pipeline, the choice of k is up to the user and can be changed easily to allow for different types of analysis. Once the NSV has been generated, many different machine learning algorithms can be applied to the dataset for classification and study. In many cases, prior to sending the data to the classifier, the NSV will go through a feature selection algorithm to reduce the number of dimensions of the dataset and make classification quicker via reduction of data. By limiting the number of features sent for classification, noise is reduced as well because each feature sent in the final data contains more information about the class of the NSV than what would otherwise exist if the entire dataset had been passed through the process. This step of feature selection is critical when the number of dimensions becomes large, which quickly happens by increasing the size of k . Examples of applying machine learning algorithms to the NSV datasets are detailed along with reports of accuracy and a discussion of why some techniques were more successful than others on this type of dataset.

1.2 Benefits

Before this research, techniques to compare metagenomic DNA samples required extensive computational analysis to identify specific members of the bacterial communities to indicate whether a given sample came from a similar environment or class as another sample. The current approach represents a less computationally intensive and faster way to process the same data, allowing for near real time continuous monitoring of ecosystems at a metagenomic level. Machine learning techniques, a subset of artificial intelligence approaches, can make accurate interpretations of data sets. A review of available literature indicates that up to this point in time, there has not been any research that uses machine learning classifiers to classify a metagenomic sample to an environment strictly using k -mer analysis. Research that is informed by these techniques might have significant applications for the scientific and medical communities such as investigation of ecosystem health and expansion of our understanding of how microbiomes interact with the host environment, or patient diagnosis via fecal sample, oral or skin swab. These are just a few of the possible benefits of a rapid metagenomic sequence classification technique.

1.3 Thesis Organization

The rest of this thesis is organized as follows:

- **Chapter 2** Literature review and overview of the processing pipeline
- **Chapter 3** Overview of datasets and computational methods
- **Chapter 4** Presentation of the results
- **Chapter 5** Discussion and conclusion from results, and future directions for the research

CHAPTER 2 LITERATURE REVIEW

Bioinformatics is the intersection of computer science and biology where insights, techniques, and skills from both disciplines are unified to continue researching current topics in biology using computer science knowledge. High throughput Next Generation Sequencing (NGS) machines produce vast quantities of genomic data [16]. The appeal of studying bacterial communities to provide new insights is easy to understand. For instance, researchers have already started exploring the metagenomic community that exists in wines [17]. The data generated by this new era of bioinformatics has outstripped computational ability to analyze and produce metagenomic data sets, thus leaving researchers with a wealth of valuable, yet, unwieldy information. The scientific community recognizes that the historical methods applied in bioinformatics research must be altered to continue to study metagenomics and keep pace with data creation [18]. Even as the scientific community endeavors to make sense of the results from recent advances, new generations of sequencing technologies are being developed that will make DNA sequencing even more accessible, such as the Oxford Nanopore platform, which will allow sequencing to be done on commodity hardware [19].

Up to this point, the focus in processing DNA sequence information has been to take data from new samples and search against sequences stored in databases to find best matches [4] [20]. Other bioinformatic utilities such as ABySS [21] or MetAMOS [22] seek to assemble DNA sequence fragments from a pure genome or from metagenomic

DNA samples.

Most microbial species are difficult or impossible to culture in a laboratory setting [23]. Approaches have been developed with the aim of assembling the genome sequence of microbes present in a metagenomic sample [24] [25]. Reconstruction of whole genomes from metagenomic samples poses a considerable challenge when acknowledging that more than one strain of a species could be present, multiple identical copies of the same horizontally transferred gene could be present in the sample from different sources, and that the depth of sequencing coverage may not yield enough fragments for full reassembly [26] [27]. Horizontally transferred genes are of interest because they increase functional capability of an environment and allow for multiple species to benefit from one gene without having to independently evolve the gene [28]. Additionally, there may be species of low frequency in the environment and may be difficult to observe as the low frequency species will represent a very small fraction of the resulting sequence data [29]. Finding other ways to interact with metagenomic data, such as classifying the environment that produced the sample, remains an elusive goal of the research community.

2.1 Search Alignment Based Sequence Analysis

Many methods are available for researchers trying to understand metagenomic data. As high throughput NGS technologies make genomic data more accessible for the research community, we are beginning to see a great diversity in efforts to make sense of the data that has been generated. A large section of metagenomic research to date focuses on techniques for evaluating substrings of DNA and matching each substring to a reference sequence thus selecting the most likely organism from which the substring of DNA came.

Clustering 16S ribosomal RNA gene sequence data has been extensively used to identify specific organisms from a metagenomic sample [30]. However, producing 16S sequences requires additional laboratory work that whole genome shotgun (WGS) metagenomic sequence analysis can forgo by utilizing fragments from the entire genome rather than being strictly focused on the 16S rRNA gene [16]. Also, not all 16S sequences will be amplified by any given set of so-called universal primers, and thus there may be some taxa present in a metagenomic sample that fail to be represented in the final analysis [31] [32]. Some of the extra steps required to run 16S metagenomic analysis include amplification and purification of 16S rRNA gene sequences via polymerase chain reaction (PCR) prior to DNA sequence analysis. Another concerning feature is the well documented, and understood, issue that microbiomes aren't identical across individual host organisms; even genetically identical mice show great divergence in gut microbiomes simply as a function of being housed in separate containers [33].

Transforming raw metagenomic DNA sequences into something of greater value has typically been done through sequence composition and similarity analyses. Such techniques require that each individual sequence fragment be compared to all other sequences in a reference database. This kind of work requires a huge number of comparisons simply as the product of unknown sequences to test sequences. This entirely ignores the issue of the substring comparisons that also takes place in these operations. Providing researchers access to public online resources for genomic sequence evaluation has become the standard practice as seen in such utilities as: MEGAN, GreenGenes, RDB, HMMER, and BLAST [4] [34] [35] [36] [37]. However, this doesn't avoid the fact that a large amount of computational resources is required to perform the analyses. Even though large amounts of research have been dedicated to the development and implementation of more efficient algorithms, typically such analyses

still cannot be done on a personal computer and require access to specifically designed research machines or clusters. Thus, these computationally intensive approaches have offloaded the maintenance and cost of the infrastructure components from individual researchers to these agency-supported online resources [38] [39].

2.2 Search Alignment free Sequence Analysis

Next Generation Sequencing technologies are producing vast quantities of genomic data. Unlike the older dideoxy-based sequencing methods, e.g. the Sanger method, NGS fragments are much shorter and more error-prone. Earlier methods for analyzing genomic sequence data are generally challenged by the smaller fragment sizes and error-rich data produced from NGS machines. Alignment-based approaches do not use location information for a sequence within the genome. Sequence location is irrelevant to the problem of substring alignment; the possibility of conserved regions shifting in the genome due to recombination or interactions over large distances in the genome is ignored when looking for substring matches.

Alignment-free sequencing processing techniques have arisen as a method to evaluate genomic data when re-assembly of the fragments is unfeasible or impossible [40]. Phylogeny analysis using alignment-free methods like feature frequency profile (FFP) has arisen using k -mers and algorithms originally developed for text comparison [41]. Initial research for FFP used a range of k -mer sizes, $11 \leq k \leq 20$. Subsequent work using FFP for mammal phylogeny identified optimal k -mer sizes range between $16 \leq k \leq 21$ with the optimal case $k = 18$ [42]. Work on bacterial phylogeny use a larger k -mer where $k = 24$ after experimenting with longer and shorter mer sizes [43].

Alignment-free sequence processing techniques have arisen as a method to evaluate genomic data when re-assembly of the fragments is unfeasible or impossible [40]. Phy-

logenetic analysis using alignment-free methods like feature frequency profile (FFP) using k -mers and algorithms originally developed for text comparison have arisen [41]. Initial research for FFP used a range of k -mer sizes, $11 \leq k \leq 20$. Subsequent work using FFP in mammal phylogeny studies identified optimal k -mer sizes range between $16 \leq k \leq 21$, with the optimal case being $k = 18$ [42]. Work on bacterial phylogenies often use a larger k -mer, e.g. $k = 24$, after experimenting with longer and shorter mer sizes [43].

2.3 Machine Learning

Machine learning is a subcategory of artificial intelligence. The study of machine learning is an interdisciplinary field with deep ties to the subjects of mathematics and computer science. Attempting to clearly define the difference between machine learning, statistical learning theory, and data mining is extremely difficult. Machine learning algorithms that rely upon statistical models and have a stochastic nature are known as soft computational techniques [44]. One of many driving reasons that the field of machine learning continues to grow is the need for knowledge discovery in databases (KDD), but this may be less of a factor than the constantly growing drive for understanding and utilizing real-world data. The need for this form of knowledge discovery in an automated way is in large part due to the increased availability of classifiable datasets. The established definition of KDD states that KDD is the nontrivial process of identifying valid, novel, potentially useful, and ultimately understandable patterns in data [45]. Data-Mining is an essential component of KDD and is the part that relies on existing methods developed from statistics and machine learning to extract useful features from the databases.

Extracting knowledge from data can only truly be done through statistical calcula-

tions. No other mechanism of knowledge acquisition from data will result in a specific uncertainty metric when the generation of patterns from the data is applied to a population, thus allowing the researcher to understand the likelihood of the correctness of the hypothesized model [46]. Datasets that have more dimensions than records are said to be high-dimensional datasets. Selecting a set of features or dimensions from the data is one of the steps in the KDD process. Isolating features that help distinguish patterns in high-dimensional data is a challenging process as feature stability, selecting a feature many times in cross-validation experiments, may be lacking due to the high number of features in the data and a low number of total records [47] [48]. Feature selection is an iterative approach and the distillation of features can be a computationally difficult problem requiring large amounts of CPU time. This is especially true in cases of high-dimensional data where only a limited number of feature selection algorithms are available to analyze such datasets [49].

Processing the metagenomic samples into NSVs using k-mer frequencies falls into the category of high-dimensional data [50]. Even with choices as small as $k = 5$ the number of dimensions exceeds one thousand, as shown in Table 3.1. To reduce the amount of total computation required, other researchers have made use of feature selection techniques when dealing with similar open research topics [51]. It is important to recognize that feature selection is not altering the fundamental data; it provides a way to reduce the number of dimensions in the data set sent to the classification algorithm. There are many known ways to perform an analysis of features on metagenomic data including forward sequential, Student's t-test, chi-squared, and Fizzy. Feature analysis by selection criteria allows for the reduction of dimensions and thus a reduction in processing load. They all represent different approaches to reducing the number of dimensions so that the data processing can be done faster and more accurately [52] [53].

The field of statistical learning theory has a clear set of definitions and equations that must be met in order for the application of a machine learning algorithm to generate meaningful results. Questions that can be solved by statistical learning theory frameworks must have the following form: records must be consistent, records must converge over time, the problem must be generalizable, and the data must be structured in a format in such a way that a machine learning algorithm can be constructed to process the data [54]. Consistency means that the data isn't completely random and that selecting or removing features in the data may reduce noise. Convergence can be thought of as settling of the ability of the classifier to maximally distinguish classes per the capacity of the algorithm and dataset. Generalization requires the data be in such a form that when separated into a set of testing and training records, that if the training set is the smallest possible sample size, then the features learned from the training data can accurately classify the test data with maximal accuracy. To construct a learning algorithm for a dataset requires an algorithm that can learn to create a separation hyperplane that will maximize accuracy when exposed to the training set and used to classify the test records. Note that the data and classification algorithms are fundamentally linked together such that the algorithm can learn from features of the data to make meaningful predictions [55] [56].

Machine learning algorithms have received a large amount of support and study in part due to the benefits they offer private and public research interests. Large projects exist and are dedicated to developing general purpose libraries and toolkits for machine learning are longstanding under active development for a number of languages, platforms, and use cases [57] [58] [59] [60] [61] [62]. The study of biology, specifically the subfields of molecular and cellular biology, are some of the principal producers of large data sets. The application of computer science principles to the field of biology is known as bioinformatics. The development of machine learning tools

that can scale to the datasets produced in bioinformatics is the subject of ongoing research and discussion [63] [64]. Examples of some specific predictive algorithms that have been employed to study evolutionary ecology and metagenomics include neural networks and naïve Bayesian classifiers [65] [66] [52]. One of the largest challenges in the field of bioinformatics is the need for a mechanism to transform the raw data into a format that can be classified by a machine learning algorithm in an accurate way.

CHAPTER 3 METHODS

The process established with these methods directly follows the entire KDD process to evaluate the metagenomic data samples [46]. From the entire set of raw data, a selection and division of fragments is drawn without replacement and the new files are isolated into testing and training samples. These samples are processed from nucleic acid sequences into NSVs to extract the environmental information from the data into a numeric form. The numeric data is then filtered using feature selection methods to extract patterns from the data. The data is then ready for analysis by any of a number of machine learning algorithms to quantify the amount of knowledge extracted by an iteration of experiments.

3.1 Methods Syntax & Conventions

Some of the special syntax conventions used in this document are as follows.

- *Scientific Names* Genus names are capitalized and italicized, species names are italicized.
- **Scripts** - bold and Courier font.
- *Variables* - italicized
- **Functions** - bold
- *System Commands* - italicized and underlined

3.2 Initial Methods

Research for this project began as a hypothetical exercise to see if a machine learning approach could be used to distinguish metagenomic samples between environmental classes. A key concept used in this research is the ability to transform metagenomic samples into a Numeric Summarization Vector (NSV).

The method to process metagenomic data in a high-level scripting language is performed using the following steps. Individual fragment reads from a sample were initially analyzed using a k -mer sliding window of some length that shifts one character to the right each iteration. This process generates sequence substrings which are called k -mers, which means a mer of size k . These k -mers are then used to increment the count of k -mers observed for each NSV index as each mer is observed. Each sample is thus transformed into a unique NSV containing the counts of all k -mers observed across all fragments in that sample. The sample is then stored to a file for easy future access. Any substring that contains a non-normal nucleotide, i.e. anything other than A, T, C, or G, is prohibited from incrementing a count in any NSV index.

The metagenomic data files contain both tags and the sequence of nucleotides comprising each sequence read. Each sequence is processed into an NSV by the following approach. Once the integer size of k for the k -mers is selected, starting at the beginning of the sequence the first k -mer is converted into an index in the NSV and that k -mer index is incremented by one. The next k -mer sequence is the next substring where the starting and ending index of the substring may be as discrete as $prev_start + 1, prev_end + 1$, or instead of advancing the indexes by one, the number could intentionally be larger and allow for less observations from each fragment to be stored in the NSV - the choice of substring advancement and this process is referred to

as the 'sliding window'. The choice of sliding window size is always fixed to advance one character per iteration, or the next distinct k -mer from the previously observed k -mer for complete coverage of the sequence fragment. This description is outlined in Code Listing 3.1 where the choices for k and the sliding window number are set by the user. Code Listing 3.1 collapses the mer into the forward and reverse complement, and only builds the forward read version of the sequence and thus, regardless of the direction of the sequence read, the forward sequence from the fragment or the complement of the k -mer is indexed in the NSV using that method. A more direct version of k -mer mapping to an index can be seen in Code Listing 3.2 where the consideration of forward and reverse complement isn't considered. Initial experiments utilized the simpler k -mer to NSV index mapping approach, which consumes more RAM and fewer CPU cycles, and each metagenomic data file produces exactly one NSV. Viewing the pipeline as a diagram can be done in Figure 3.1 at the end of this chapter.

Code Listing 3.1: Processing a FASTA File into an NSV

```

1 def get_mer_count_use_complement(mer_size, file_fragments,
2   sliding_size):
3     """ this function process fragments from a metagenomic file
4     and returns the fragments back as a single NSV using the mer
5     size and sliding window size as adjustable parameters. """
6     mer_counts = {}
7     for fragment in file_fragments:
8         j = 0
9         max_j = len(fragment) - mer_size
10        while j < max_j:
11            mer_frag = fragment[j:j + mer_size]
12            mer_frag = mer_frag.lower()
13            if "n" not in mer_frag:
14                try:
15                    mer_counts[mer_frag] += 1
16                except KeyError:
17                    mer_counts[mer_frag] = 1
18            j += sliding_size
19
20    array_size = (4 ** mer_size) / 2
21    if mer_size % 2 == 0:
22        array_size += 2**(mer_size - 1)
23
24    mer_to_index = make_collapse_map(mer_size)
25
26    my_nsv = [0] * array_size
27    for mer in mer_counts.keys():
28        try:
29            mer_index = mer_to_index[mer]
30        except KeyError:
31            mer_index = mer_to_index[make_complement_mer(mer)]
32
33        my_nsv[mer_index] += mer_counts[mer]
34
35    return my_nsv[:]

```

Code Listing 3.2: Find the Correct k -Mer Index in an NSV

```
def mer_index_finder(my_string):
2   my_string = my_string.lower()
   char_value = {}
4   char_value["a"] = 0
   char_value["t"] = 1
6   char_value["c"] = 2
   char_value["g"] = 3
8   i = 0
   j = 0
10  base_four_string = ""

12  myStrLen = len(my_string)
   while(i < myStrLen):
14     base_four_string += str(char_value[my_string[i]])
       i += 1
16
18  index = int(base4String, 4)

   return index
```

A second set of preprocessing is done to normalize the NSVs and produce a new class representative NSV. Data sample NSVs are normalized so that each column has a value between zero and one. The dimension that contains the highest fragment count has a value of one and the dimension with the fewest fragment counts has a value of zero. Any dimension with a matching score to the minimum or maximum fragment count is set to zero or one respectively. All other dimensions are scaled using (3.1) which assumes float representations of integer numbers *index_value*, *index_count*, and *total_count*. This ensures the total NSV cell sum to be exactly one. The terms in the equation relate to these concepts: *index_value* is the new number that is stored in the averaged NSV; *index_count* represents the count of that specific *k*-mer in the initial NSV at this index; *total_count* is the total number of mers present in the initial NSV. An example of the implementation of the equation into code is provided in Code Listing 3.3 to show how one might code this function.

$$index_value = \frac{index_count}{total_count} \quad (3.1)$$

Code Listing 3.3: Normalizing an NSV

```

1 def normalize_by_size(split_line):
    res_list = []
3     num_mers = sum(split_line)
    for i in range(0, len(split_line)):
5         c_val = split_line[i] / (1.0 * num_mers)
7         res_list.append(c_val)
    return res_list

```

An alternate averaging technique has been developed and tested as well. In (3.2) the terms are: *max* the maximum count of mers in an index across the NSV; *min* the lowest count of mers in any index - typical 0; *current_value* the count of mers present at this index of the NSV; and *new_value* the output of the calculation. Unlike (3.1), in (3.2) the sum of the averaged NSV can be greater or less than one depending on the distribution of the counts of mers. This is problematic for comparing NSVs that are composed of drastically different fragment counts.

$$new_value = \frac{current_value - min}{max - min} \quad (3.2)$$

Given the availability of all the NSVs from the way that the script has been developed, a normalized *k*-mer class representative is also constructed. Each dimension of the normalized representative NSV is assigned the value of the average for the samples in the data set for the class being represented. Each sample is written to a normalized NSV file. The class representative is written to file in a similar manner; however, the class representative doesn't have a class identifier appended to the right-most column. The file name is used as the class identifier for the representative NSVs.

In the final phase of preprocessing, the data, samples are written to a normalized class specific *k*-mer NSV output file where each line is one sample. Additionally, a numeric class identifier is appended to the last (right-most) column. This file is then read into a different script that performs a Student's t-test. For this research, the Student's t-test is used to identify the ten dimensions with the greatest average pairwise statistical variance between classes. A reduced data set containing all samples for the rows and the ten dimensions for the column is then saved to a file.

At this stage, *k*-nearest neighbor (*k*-NN) and linear discriminate analysis (LDA)

are used to classify the reduced dimensional data. The value of k is set to 3 for all the k -NN tests performed in this research. The k in k -NN represents the number of neighbors that are used to classify the test record. Both the k -NN and LDA approach classify the data using leave-one-out cross-validation (LOOCV), where one of the samples in the dataset is randomly selected to be part of the test set. The LOOCV technique has been chosen as there are very few data records. Each sample is tested against the greatest amount of data possible providing a robust test for accuracy. The remaining samples of the data are used as the training set. The system predicts the class of any given sample in the test set based on the learned characteristics of the training set. The accuracy of the system is computed determined during this process.

Classification is tested by using a leave-one-out cross-validation (LOOCV). Each file from the dataset is systematically withheld from the training set and placed into the test set. The classifier is trained using all the other files, which are the current training set. The test file is classified using the model developed from the training set. The system predicts the class of the sample in the test set based on the learned characteristics of the training set. The total accuracy is preserved in the confusion matrix which preserves the way each test set is classified across all folds. The accuracy of the system is determined during this process.

Some challenges with these research methods included the RAM and processor limitations inherent in typical workstations. To mitigate these challenges, the proposed research was run on the most robust machines available. Best practices were followed when building the analysis program to limit the number of sequences in memory, as well as limit the number of multiprocessor cores and threads used, to manage the resource demands independent of machine specifications. Limiting the resource demand directly in the code likely has the greatest impact on resource demand overall. One example of code governing resource demands that can be easily tuned is the choice

of k for both the k -mer size and the k -NN algorithm.

3.3 Exploring Choices for k and Window Overlap

Selecting the size of k for a k -mer depends on several factors. While selecting larger values for k , for example 12-mers, allows greater precision, it also produces a minimally populated NSV. Selecting an especially small number for k , for example 2-mers, produces NSVs that are densely populated and difficult to separate because there is too much overlapping content. Small k values allow for faster computation such that machines with less power can effectively perform the analysis. This is contrasted by larger values for k , which require greater RAM resources. This technique, at least for NSV creation, is strictly limited by RAM and disk resources and varying k will not drastically change the number of CPU cycles required.

This research explored various choices for the size of the mers by using larger and smaller numbers for k . Three is the smallest size of k -mer used in this research, but in theory a k of size one or two could be used. The number `[MAX_NUM]` was selected as the maximum value for k as performance limitations of the research servers wouldn't allow for larger choices.

Table 3.1: The rapid growth of dimensions as a function of k .

Size of K	Number of Dimensions
2	16
3	64
4	256
5	1 024
6	4 096
7	16 384
8	65 536
9	262 144
10	1 048 576
11	4 194 304
12	16 777 216

3.4 Feature Selection

Processing the metagenomic files results in a feature space of very high dimensionality. Creating methods by which to select features and reduce the amount of data per record prior to sending the dataset through the machine learning algorithm allows for the identification of key features in the dataset. By only sending in the most relevant features to a machine learning algorithm during the training process the goal is to prevent the algorithm from learning poor features that can generalize to new records. Overfitting a data set is a known problem in machine learning and happens due to the model being overly complex and typically yields poor predictive capability when a classifier has been overfit to a dataset. Having a model that is overfit to the data makes it difficult for the model to be used on new data sets as the model will be unable to correctly accurately classify the records.

Two feature selection algorithms are included as part of the data processing pipeline. The first feature selection mechanism is an all-pairwise Student's t-test, shown in Code Listing 3.4, where the dimensions with the greatest t-score are selected for classification as they represent the most divergent dimensions between classes using the Student's t-test. The other feature selection algorithm used is information gain, where the selected dimensions are similarly chosen based upon maximal information gain from the data set. One additional dimension reduction technique is included into the pipeline, namely a random selection of dimensions of equal number to the one passed to either of the other two algorithms. Including a smaller, random, set of data in each of the folds of the cross-validation is done to show the benefit of performing the feature selection on the metagenomic data. In addition to the random feature selection, classification using all dimensions for number of choices for k -mer size is also reported in the results section.

Code Listing 3.4: All Pairwise Student's T-Test

```

def find_T_score( dataSet):
2   """ t-test is an all pairwise comparison (upper triangular matrix)
3   """
4   tScores = []
5   pValues = []
6   newData = []
7   if len(dataSet) > 0:
8       #loop over cols make a transposed data set
9       for col in range(0, len(dataSet[0][: -1])):
10          subSetHolder = {}
11          #for levels of classes
12          #take indicies and transform them into subsets of the data
13          for i in range(0, len(dataSet)):
14              classI = dataSet[i][ -1]
15              if classI in subSetHolder.keys():
16                  subSetHolder[ classI ].append( float( dataSet[ i ][ col ] ) )
17              else:
18                  nuList = [ float( dataSet[ i ][ col ] ) ]
19                  subSetHolder[ classI ] = nuList
20
21          myKeys = sorted( subSetHolder.keys() )
22          count = 1.0
23          tmpT = 0.0
24          tmpP = 0.0
25
26          for x in range(0, len(myKeys) - 1):
27              xData = subSetHolder[ myKeys[ x ] ]
28              for y in range( x + 1, len(myKeys) ):
29                  yData = subSetHolder[ myKeys[ y ] ]
30                  #must have the stats package imported from scipy
31                  tRes, pVal = stats.ttest_ind( xData, yData )
32
33                  tRes = math.fabs( tRes )
34                  pVal = math.fabs( pVal )
35                  tmpT += tRes
36                  tmpP += pVal
37                  count += 1.0
38
39          myColT = tmpT / count
40          myColP = tmpP / count
41
42          tScores.append( myColT )
43          pValues.append( myColP )
44
45   return tScores

```

3.5 Machine Learning Algorithms

This section is dedicated to explaining the machine learning algorithms that are used in this research. As there are several algorithms, and they approach classification in a variety of ways, a brief explanation of each algorithm as well as some of the key mathematical concepts that are embedded in them are presented. Where a formula is presented, there will be a short description of all the terms following the equation. Let all terms in these equations be defined the following way: each term is on the left and the definition is on the right, the \triangleq symbol should be read as 'is defined'.

3.5.1 KNN

One machine learning technique that requires essentially no training, in the formal sense, is the k -Nearest Neighbors (k -NN) algorithm. The formal mathematical definition of the algorithm appeared in 1951 as a method to decide group membership based on existing observations, the training data, and a new observation, the test record [67]. Why it can be said that the algorithm requires essentially no training is that all the training records are directly compared across the feature space to a test record. The k in the k -NN algorithm is a variable positive integer number, and the k training records closest in space to the test record then vote to classify the test record. Distance in space can be determined with any number of metrics, however, in this research only Manhattan distance was used. The formal definition of Manhattan distance is presented in (3.3). Implementation to find the Manhattan distance is presented in Code Listing 3.5. Euclidean distance was also considered, but the increased processing demands made it unattractive for preliminary research. In this research, all neighbors for any choice of k are given equal weight.

$$D(a, b) = \|a - b\| = \sum_{i=1}^{i=n} |a_i - b_i| \quad (3.3)$$

- $a, b \triangleq$ as vectors with equal dimensionality, here given as n dimensional space.
- $D(a, b) \triangleq$ distance between the vector a and vector b .
- $\|a - b\| \triangleq$ is the norm of the difference between a and b .
- $\sum_{i=1}^{i=n} |a_i - b_i| \triangleq$ is the sum of the absolute values of the difference of each dimension in vectors a and b for all n dimensions.

Code Listing 3.5: Calculate Manhattan Distance

```

def find_manhattan( point_a , point_b ):
2     dist = 0
    for x in range(0, len(point_a)):
4         dist += math.fabs(float(point_a[x]) - float(point_b[x]))
    return dist

```

3.5.2 Naïve Bayes

Classification by a Naïve Bayes (NB) classifier relies upon the Bayes' theorem which was developed in the middle of the eighteenth century [68]. The statistical theorem uses prior information related to an outcome to predict the likelihood that a specific outcome is observed. The theorem became part of the machine learning canon during the 1950s and 1960s. An assumption that all dimensions within the dataset are independent is explicitly made in the NB algorithm, and as such this classifier is very well suited to be applied to datasets that have a large number of dimensions. Prediction of the class is determined by the maximal product of probabilities across all classes. An example of how this could be implemented is provided in the following code snippets Code Listing 3.6, Code Listing 3.7, and Code Listing 3.8 in practice the size for the bins must be determined prior to attempting to bin the data and

subsequent classification.

$$P(c|d) = \frac{P(d|c)P(c)}{P(d)} \quad (3.4)$$

- $c \triangleq$ class c
- $d \triangleq$ dimension d
- $P(c|d) \triangleq$ probability of class c given the value of dimension d
- $P(d|c) \triangleq$ likelihood of observed value in dimension d given it's associated with class c
- $P(c) \triangleq$ probability of observing class c
- $P(d) \triangleq$ probability of value d being observed in this dimension

Code Listing 3.6: Bayes' Rule Classifying Test Records

```

1 def find_class_probability(class_list, test_data, binned_training_data):
2     final_votes = {}
3     for row in xrange(0, len(test_data)):
4         final_votes[test_data[row][-1]] = {}
5
6     test_cols = len(test_data[0]) - 1
7     for elt in xrange(len(test_data)):
8         elt_bins = []
9         for col in xrange(test_cols):
10            col_bin = find_bin(float(test_data[elt][col]),
11                               max_min_values[col][0], bin_sizes[col],
12                               n_bins)
13            elt_bins.append(col_bin)
14
15        votes_map = {}
16
17        for class_item in class_list:
18            prob = 1.0
19            for dim in xrange(test_cols):
20
21                numerator = \
22
23                binned_training_data[dim][elt_bins[dim]][class_item] + 1
24                numerator = float(numerator)
25
26                denominator = \
27
28                sum(binned_training_data[dim][elt_bins[dim]].values()) \
29                    + m_est
30                denominator = float(denominator)
31
32                prob *= numerator / denominator
33            votes_map[class_item] = prob
34        try:
35            final_votes[test_data[elt][-1]][max(votes_map,
36            key=votes_map.get)] += 1
37        except:
38            final_votes[test_data[elt][-1]][max(votes_map,
39            key=votes_map.get)] = 1

```

Code Listing 3.7: Bin all Values for all Records

```

1 def bin_data(data_set, max_min_matrix, bin_sizes, n_bins, classes):
2     """ for an entire data set correctly assign all records to
3     the correct bins, return a bin matrix as a new data set """
4     nu_data = []
5     n_col = len(data_set[0]) - 1
6
7     for j in xrange(0, n_col):
8         bins = []
9         bin_counter = 0
10        while bin_counter < n_bins:
11            bin_map = {}
12            for class_opt in classes:
13                bin_map[class_opt] = 0
14            bins.append(bin_map.copy())
15            bin_counter += 1
16
17        for i in xrange(0, len(data_set)):
18            ij_bin = find_bin(float(data_set[i][j]),
19                             max_min_matrix[j][0], bin_sizes[j],
20                             n_bins)
21
22            bins[ij_bin][data_set[i][-1]] += 1
23        nu_data.append(bins[:])
24
25    return nu_data[:]

```

Code Listing 3.8: Find Correct Bin for Value in Dimension

```
1 def find_bin(value_to_bin, lower_bound, bin_size, n_bins):  
2     """ given a value and bin information select the correct  
3     bin for the value to be inserted into """  
4     selected_bin = 0  
5  
6     found_bin = False  
7  
8     lhs = lower_bound  
9     rhs = lhs + bin_size  
10    while not found_bin:  
11        if lhs < value_to_bin and value_to_bin <= rhs:  
12            found_bin = True  
13        elif value_to_bin <= lhs and selected_bin == 0:  
14            found_bin = True  
15        elif value_to_bin >= n_bins * bin_size + lower_bound:  
16            found_bin = True  
17        else:  
18            lhs = rhs  
19            rhs += bin_size  
20            selected_bin += 1  
21  
22    return selected_bin
```

3.6 Validation

Initial results were tested and compared using Leave-One-Out Cross-Validation (LOOCV). This posed a problem as the data sets were inherently asymmetric with respect to fragment count per class. Imposing a limit on the number of fragments allowed into the pooled data set partially addressed the asymmetry. One known issue with LOOCV is the tendency to overfit the data [69], which limits the ability of the results to expand from theory to practice. As an alternative to LOOCV, the concept of Monte Carlo Cross-Validation (MCCV) emerged and has subsequently been applied to chemical systems [70] [71] and other areas as well. Adapting some of the ideas from MCCV, an intrinsically random cross-validation technique has been applied to verify the accuracy of the results and prevent bias entering the analysis.

In this research, all primary FASTA files are listed and the class of each file is known at the start of an experiment. In both LOOCV and MCCV all files are grouped together by class. Also in both, one file is held out at a time as the test file. Here the two approaches begin to differ. In LOOCV the grouped training files are processed as a batch using all fragments and the test file is classified per the generated NSVs from the testing files and training file. In MCCV a limited number of random fragments is drawn from each class without repetition. The MCCV process keeps the collection of fragments for each class separate. A random number of fragments is drawn from the test file as well, also without repetition. The pooled training files are divided into ten parts, where each of the ten files has approximately the same number of fragments as the training file. The number of fragments drawn is always less than the total number of fragments in the raw test file. A fold is one iteration of leaving out one file and comparing it to the remaining test files. A trial is a complete collection of folds and in most cases, there are twelve folds in a trial; this is because there are four files of each of the three classes in the Johnson Grass data. This process can be observed in Code Listing 3.9.

Code Listing 3.9: Mix Files into Testing and Training Sets by Class

```

def mix_files(data_start_path, class_options):
2   # get files
   ini_file_list = commonFunctions.GetFiles(data_start_path)
4   split_classes = {}
   # orders = itertools.permutations(ini_file_list)
6   # organize sort
   for item in ini_file_list:
8       for opt in class_options:
           if opt in item:
10          try:
               split_classes[opt].append(item)
12          except:
               split_classes[opt] = [item]
14   # files will be a set of permutations
   # do count of lines
16   rand_orders = {}
   for opt in class_options:
18       print opt
       rand_orders[opt] = []
20       orders = itertools.permutations(split_classes[opt])
       for x in orders:
22           rand_orders[opt].append(list(x))

24   test_files = []
   training_files = []
26   for opt in class_options:
       list_len = len(rand_orders[opt])

28       for index in xrange(list_len):
30           arangement = rand_orders[opt][index]
           # print arangement
32           test_half = arangement[:len(arangement) / 2]
           train_half = arangement[len(arangement) / 2:]
34           try:
               for elt in test_half:
36                 test_files[index].append(elt)
               for alt in train_half:
38                 training_files[index].append(alt)
           except:
40               test_files.append([])
               training_files.append([])
42               for elt in test_half:
                   test_files[index].append(elt)
44               for alt in train_half:
                   training_files[index].append(alt)
46

   return test_files, training_files

```

3.7 Data Processing Pipeline

The methodology that was ultimately established for this research is explained in this section. Many different approaches were explored and tested prior to reaching the set of operations described in the following pages. One of the driving principles for this research was to find a way to rapidly evaluate the class of a bacterial community and can identify how it differs from other communities from similar environments such that they would belong to a different class. Processing the entire set of metagenomic FASTA files on each run from start to finish takes a significant amount of time as identified in the initial methodology section. To circumvent this lengthy process, the final approach adopts techniques to speed up processing while simultaneously increasing accuracy.

The following list enumerates the languages and libraries used to process the metagenomic data from its raw format to a usable NSV, make predictions, and visualize the results. Python is heavily relied upon for much of the data processing and predictions. The R statistical language is used for additional classifications and predictions, as well as providing the visualization capacity for the results. Both languages are widely used in the scientific community. One of the reasons these two languages, and extension libraries, have been selected is the free open source software licenses they are under, making the replication of this work easier to follow and expand upon.

- **Python** - Programming language used for processing the data and much of the machine learning work [72].
- **R** - Statistical programming language selected for visualizations and more classification algorithms [73].
- **scipy** - An open source scientific library that extends the capacities of Python. [74].
- **scikit-learn** - Large library of tools for data mining and data analysis written in Python. [75]
- **Matplotlib** - Python library to produce visualizations [76].

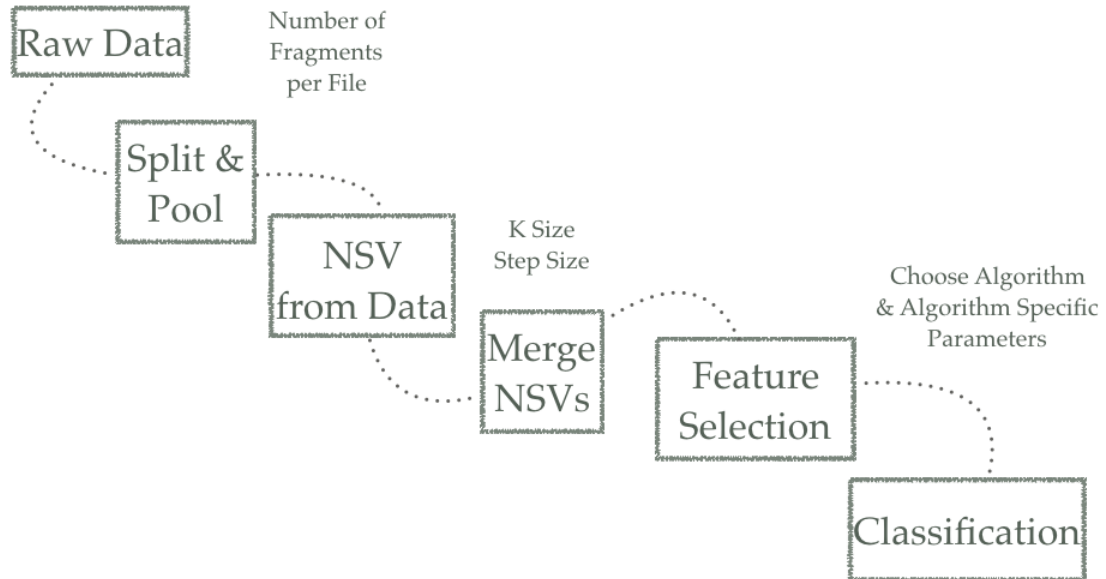


Figure 3.1: A visual Representation of the Data Processing Pipeline. The Data goes through six unique steps before classification can be performed. Each step has multiple options for processing the files in a variety of ways. The smaller text above the boxes shows some of the user parameters that can be selected during those steps.

3.7.1 Synthetic Data

To test new experimental approaches a mechanism to produce synthetic data has been developed to generate files for the pipeline to run unit tests. This research includes such a verification facility that can generate FASTA files with the same desired characteristics, thus emulating the raw data output style of any Next Generation Sequencing (NGS) machine. Such a random sequence generation utility allows researchers to produce new synthetic data which has no unique characteristics, making it theoretically impossible to distinguish between classes. This utility can generate any number of classes, sequences, and vary sequence size. The ability to generate synthetic data with arbitrary properties allows for testing of the pipeline in a way that isn't contingent upon actual data being available.

3.7.2 Collecting Raw Records into a Single Pool and Subsequent Random Division

The pooling and division of data is done for each fold of the cross-validation process. The main purpose of the pooling and splitting of the data is to draw smaller samples from the files to be stochastic in tests, and because the number of fragments drawn for each fold is small with respect of the size of total fragments for each class. Training files are grouped together by class - pooled. Then the small, but representative, number of fragments are drawn and divided into ten class representative files. This is done for each class. Then the same process is done for the test file. Fragments from the test file are randomly drawn without replacement until there is enough to fill ten class representative files with the same number of fragments in the representative file for the test samples as there are in the training files. Feature selection, and classification are performed. Results are updated in the confusion matrix. This process is repeated

for each fold of the cross-validation process.

Collecting all the sequence records from a class into a single dataset is one of the first steps in the data processing pipeline. The idea that similar elements can be grouped together is a well understood concept. Each class is grouped together into a single file containing all records related to the class. This process is repeated for each class until all classes have been combined into one file per class. These are referred to as the 'mega files' representing each class.

A new set of representative files is randomly sampled from the mega file without replacement. The order of the selected fragments is then randomly shuffled, and then divided into ten unique FASTA files. This process is outlined in Code Listing 3.10. These files each contain a given number of fragments from their respective classes. Caution is taken that the number of sequence fragments in each split file is more than the minimal number of records for the split file to be representative of the class. This technique means that strong confidence can be placed in the idea that each split represents the environment from which it was generated. It may seem that using this process that in a trial there would be ten folds. However, this is not the case. The test file is omitted and divided into ten split files. Each of the ten split files for the test records is the same size as the training split files created for each class created earlier. Then in a single fold the ten test split files are classified against all training files.

Code Listing 3.10: Randomly Split Fragments into Files

```

1 def read_mega_file_and_split(inputPath, outputPath, limit, parts):
2     """ take in a mega file and randomly split according to a fragment
3     limit """
4     mega_files = commonFunctions.GetFiles(inputPath)
5     commonFunctions.removeExisitingData(outputPath)
6     for mega in mega_files:
7         tag = ""
8         tagCounts = 0
9         frag = ""
10        fragCounts = 0
11        tagFragDict = {}
12
13        hyphen_location = mega.rfind("-")
14        mega_class = mega[:hyphen_location]
15
16        fileString = inputPath + mega
17        with open(fileString, "r") as myFile:
18
19            for line in myFile:
20                if( line[:1] == ">"):
21                    tag = line
22                    tagCounts += 1
23                if( line[:1] != ">"):
24                    frag = line
25                    fragCounts += 1
26                if( (frag != "") and (tag != "")):
27                    tagFragDict[tag] = frag
28                    tag = ""
29                    frag = ""
30
31            # find number of frags
32            keys = tagFragDict.keys()
33
34            #split frags into n equal parts
35            random.shuffle( keys)
36            if limit:
37                keys = keys[:limit]
38            splits = chunkIt(keys, parts)
39            cnt = 0
40            for split in splits:
41                last_name = outputPath + mega_class + "-split-" + str(cnt) +
42                ".fasta"
43                cnt += 1
44                with open(last_name, "w") as out_file:
45                    for index in split:
46                        out_file.write(index + tagFragDict[index])
47
48    return 1

```

3.7.3 Translation of FASTA files into NSVs

Each of the split files is saved in the standard FASTA format commonly used in bioinformatics research. Every file is turned into a unique Numeric Summarization Vector representing that file. As each fragment is processed by the algorithm, the window size of the k -mer increments a corresponding index in the NSV array, meaning that the specific mer has been seen an additional time in that file. The number of base pairs moved each time the window progresses is a tunable dial. The more units skipped each time, the less unique the NSV becomes, and yet the data will be processed faster due to the decreased number of comparisons. In this research, the window only moves one base pair for each comparison.

3.8 Datasets

Initial research largely made use of metagenomic data of the human gut microbiome available from the publicly available Short Read Archive (SRA) operated by the National Center for Biotechnology Information (NCBI) [14] [15]. Table 3.2 lists all sample identification numbers used for this research. The table also shows how the dataset represents samples from three different disease states of the human gut: Healthy, Ulcerative Colitis, and Crohn's disease. Changing the initial data set from Johnson Grass to human gut metagenomic microbiome samples did not change the theoretical groundwork for the proposed research because the sequence data was similarly divided into three classes.

The Holben Microbial Ecology Laboratory provided data from three different environment classes. The sequence data was obtained from metagenomic DNA recovered from soil samples taken at a Johnson Grass (*Sorghum halepense*) invasion study site in Oklahoma. The intent was to explore how the microbial community shifts as an

effect of invasive Johnson Grass plants overtaking a prairie area. The classes are representations of the following three types of environment: non-invaded, partially invaded, and fully invaded.

Table 3.2: Number of lines in the Human Gut data set.

Line Count	File Name
1218534	Crohns-SRR053016Full.fasta
611178	Crohns-SRR053019Full.fasta
1068952	Crohns-SRR053034Full.fasta
532256	Crohns-SRR053036Full.fasta
2350246	Crohns-SRR054211Full.fasta
1375236	Crohns-SRR054212Full.fasta
805624	Healthy-SRR053013Full.fasta
783114	Healthy-SRR053023Full.fasta
251008	Healthy-SRR053027Full.fasta
443538	Healthy-SRR053031Full.fasta
49714	UC-SRR058718Full.fasta
1246118	UC-SRR059126Full.fasta
1179536	UC-SRR059127Full.fasta
1314246	UC-SRR059128Full.fasta
1336370	UC-SRR059129Full.fasta
25788326	UC-SRR060115Full.fasta
6429190	UC-SRR060116Full.fasta
46783186	TOTAL

Many different species are present in the mouth at any given point of time. Many of the species are helpful or benign, but others are known to facilitate oral diseases such as cavities (caries) [77]. A different study seeking to understand the functional capacity of the oral microbiome produced the first metagenomic sample from the oral cavity without the bias of PCR amplification or cloning [78]. The metagenomic sequence datasets produced by that study have been made publicly available on the MG-RAST server and have been used as an additional dataset in this research project [78] [79]. The eight samples are broken into four classes: i) never had any cavities, ii) had cavities in the past but none or very few active when the sample was taken, iii)

many past cavities and many active cavities when sampled, and iv) the microbiome of an active cavity. The overall health is classified using the World Health Organization carious, absent and obstructed teeth index (CAO index) as established in the Oral health surveys: basic methods [80]. Each of the records and class values from the saliva dataset can be seen in Table 3.3.

Table 3.3: These are the MG-RAST samples used to evaluate the metagenomic classes of the oral cavity [79]. Class status is given as an example for how the classifiers will distinguish the records.

MG-RAST ID	Disease Status	Class
4447192.3	Healthy CAO's Index = 0	Healthy
4447102.3	Healthy CAO's Index = 0	Healthy
4447101.3	Diseased CAO's index = 6	Good
4447103.3	Diseased CAO's index = 8	Good
4447903.3	Diseased CAO's index = 11	Bad
4447943.3	Diseased CAO's index = 25	Bad
4447970.3	Cavity CAO's index = 10	Cavity
4447971.3	Cavity CAO's index = 11	Cavity

CHAPTER 4 RESULTS

The results from several experiments are presented here. Many of the results from the experiments are shown as paired figures where one of the images shows a confusion matrix and the second image shows the consistency across the trials that dimensions were selected aggregated into a bar plot. The k -NN results use a larger number of folds for the MCCV experiment, also all feature selection is done with SelectKBest rather than all pairwise Student's T-test, which is universally used with the Naïve Bayes Classifier experiments.

4.1 Johnson Grass Dataset

The Johnson Grass Dataset is the most evenly divided dataset in terms of total fragments per class. It also benefited from having four samples for each class making the data set well suited to split up for LOOCV and MCCV cross-validation. Many of the experiments made use of this data set for these reasons. The count lines in each file and class values are presented in Table 4.1. To find the number of fragments in a FASTA file divide the line count by two.

Table 4.1: Number of lines in each file and the total number of lines in all files of the Johnson Grass Dataset.

Line Count	File Name
1135814	Invaded_JG_DNA_Index22_CGTACG_L003_joined.join.fasta
364900	Invaded_JG_DNA_Index23_GAGTGG_L003_joined.join.fasta
474416	Invaded_JG_DNA_Index25_ACTGAT_L003_joined.join.fasta
1314180	Invaded_JG_DNA_Index27_ATTCCT_L003_joined.join.fasta
776822	Native_JG_DNA_Index13_AGTCAA_L003_joined.join.fasta
872884	Native_JG_DNA_Index14_AGTTCC_L003_joined.join.fasta
1753014	Native_JG_DNA_Index15_ATGTCA_L003_joined.join.fasta
1669122	Native_JG_DNA_Index16_CCGTCC_L003_joined.join.fasta
465768	Transition_JG_DNA_Index18_GTCCGC_L003_joined.join.fasta
468162	Transition_JG_DNA_Index19_GTGAAA_L003_joined.join.fasta
851044	Transition_JG_DNA_Index20_GTGGCC_L003_joined.join.fasta
1052114	Transition_JG_DNA_Index21_GTTTCG_L003_joined.join.fasta
11198240	total

4.1.1 *k*-NN Results

Results from using the *k*-NN algorithm are generated using the Python scikit learn library and using SelectKBest for feature selection. In Figure 4.2, the ten most selected dimensions are shown. These dimensions were selected using scikit learn SelectKBest as the feature selection mechanism on 5-mers with twenty independent trials. The red bar on the left-hand side of Figure 4.2 shows the maximum number of times any feature could be selected. In the case of this experiment the number of folds run is 24, and with 18 trials the product of the two numbers is 432.

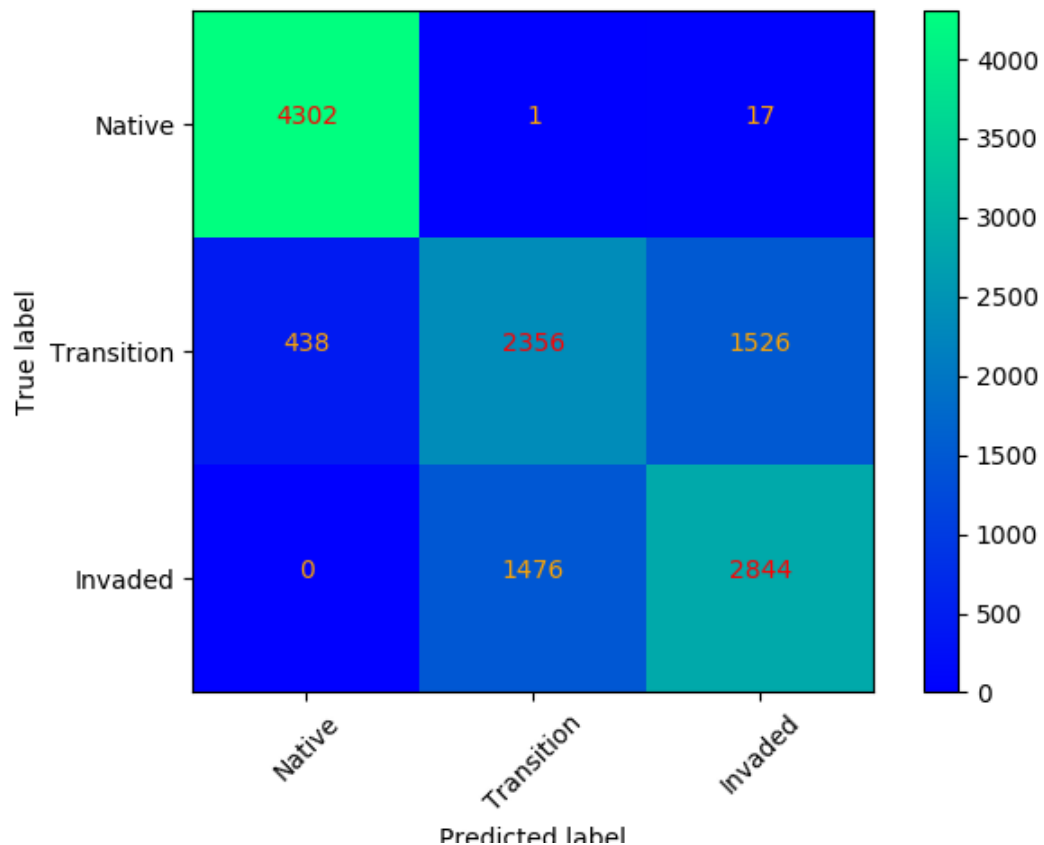


Figure 4.1: Confusion matrix. 5-mers, 10 dimensions, 18 trials SelectKBest feature selection. Each trial has 24 independent folds. This figure is the resulting confusion matrix produced by the KNN algorithm. Total accuracy of this run is 73.31% correct.

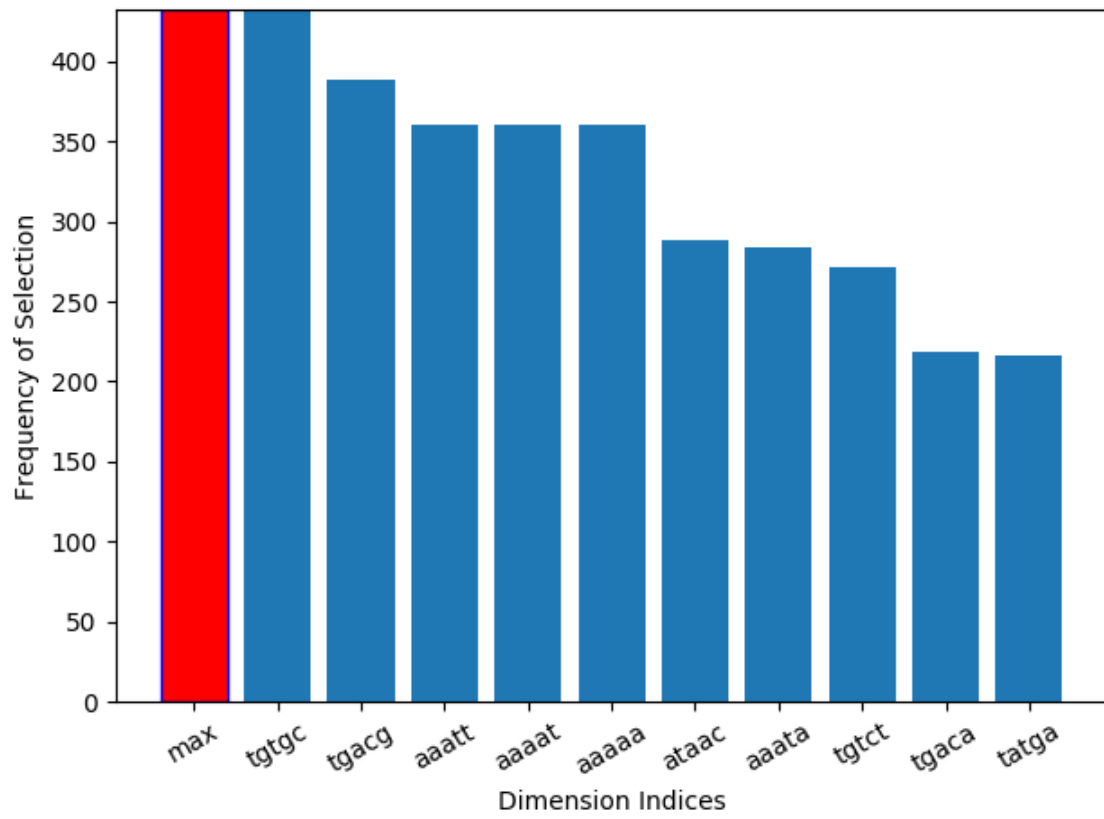


Figure 4.2: Bar plot analyzing consistency of feature selection frequency of 5-mer and Ten Dimensions.

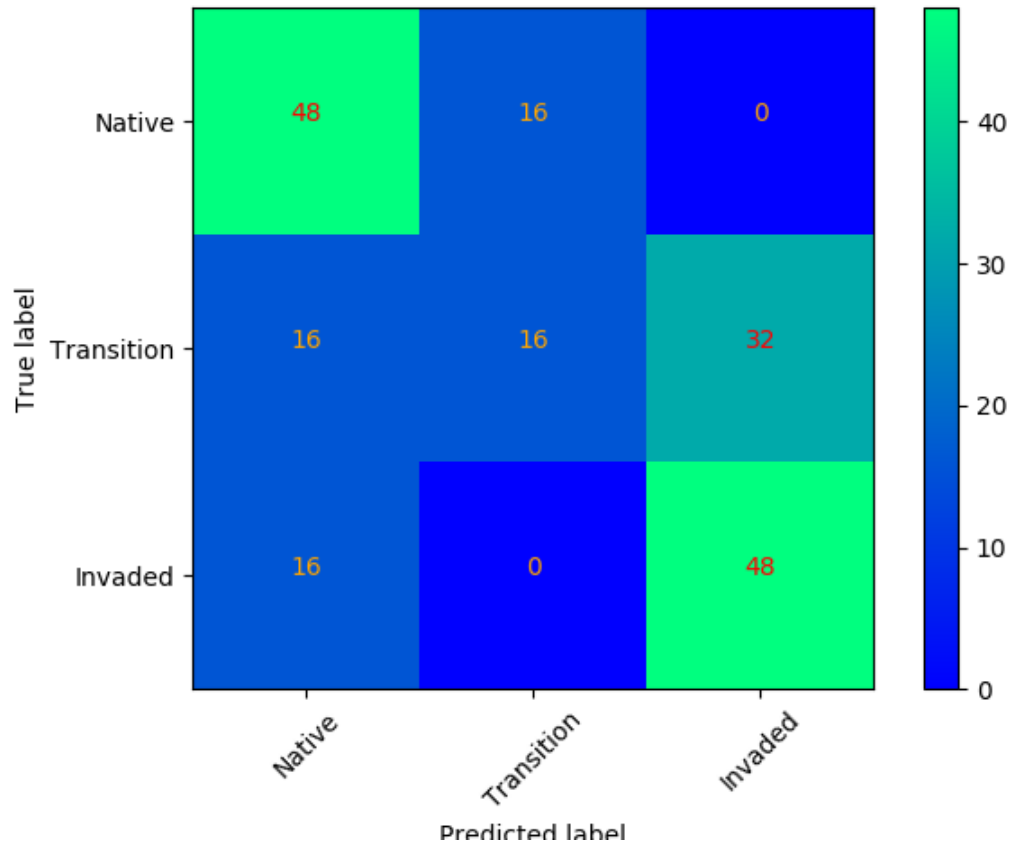


Figure 4.3: Confusion Matrix 5-Mers 5 Dimensions 16 Trials. Each trial uses leave-one-out cross-validation and doesn't pool training data together. Total accuracy of this run is 58.33%.

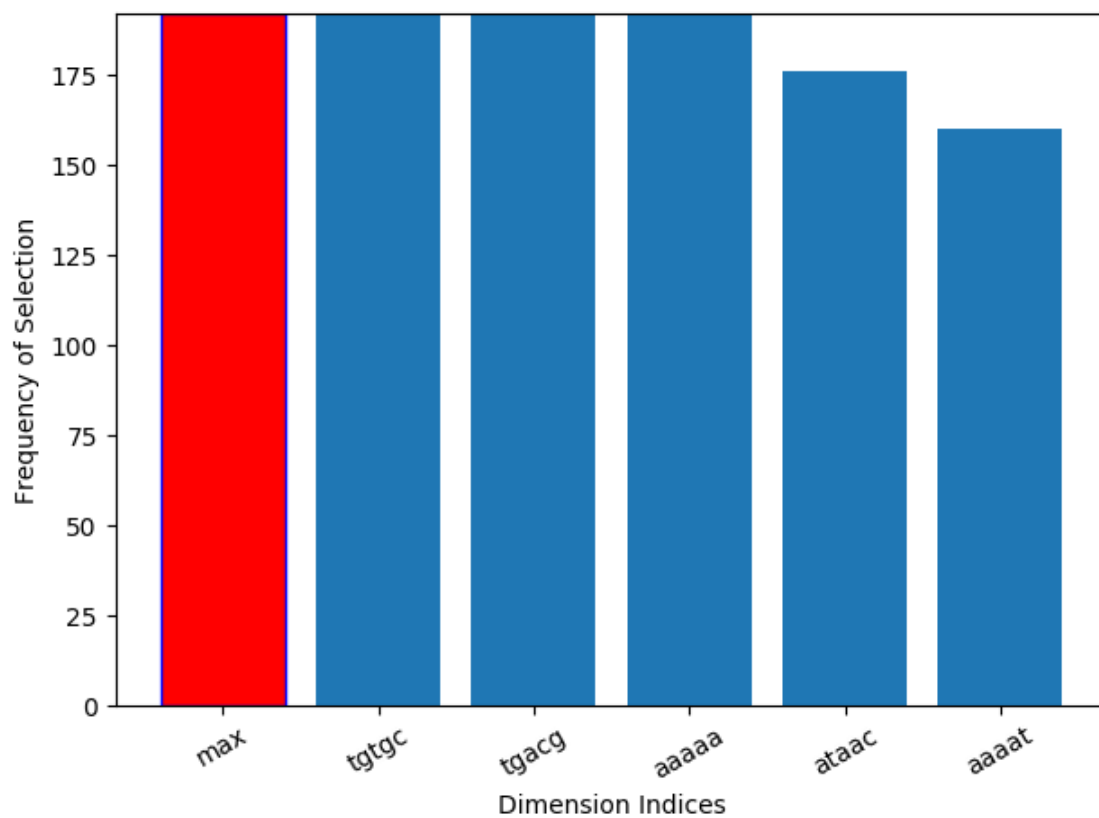


Figure 4.4: Bar plot Analyzing Consistency of Feature Selection On 5-mers & 5 Dimensions.

4.1.2 Naïve Bayes Classifier Results

The core of the experimental pipeline is tested using a Naïve Bayes Classifier. This section outlines the result of many different experiments using slightly different parameters; specifically, the size of k -mers and number of dimensions selected. The classification accuracy results are summarized in Table 4.2.

4.1.2.1 Two Class Problem

Distinguishing between two different environmental classes is an important test to see if the methods can correctly separate the data. Only one experimental run is shown here. This run is different from many other experimental results presented in this chapter as it uses an older classification method that doesn't use normalization or mer collapse. The experiment also uses a higher number of folds and trials, 24 and 25, than the subsequent experiments.

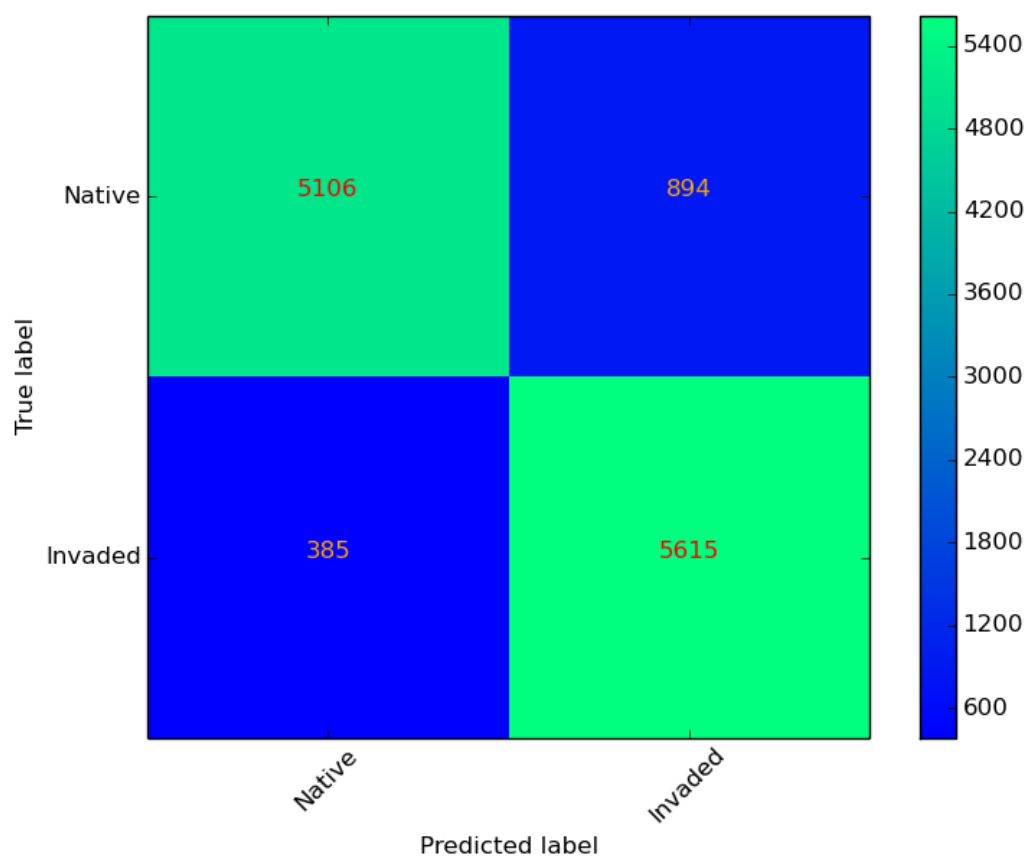


Figure 4.5: Confusion Matrix. 5-mers, 10 dimensions, 25 trials, 2 classes. After twenty five independent trials where each trial has 24 independent folds run using MCCV, this is the resulting confusion matrix. Total accuracy of this run is 89.34%.

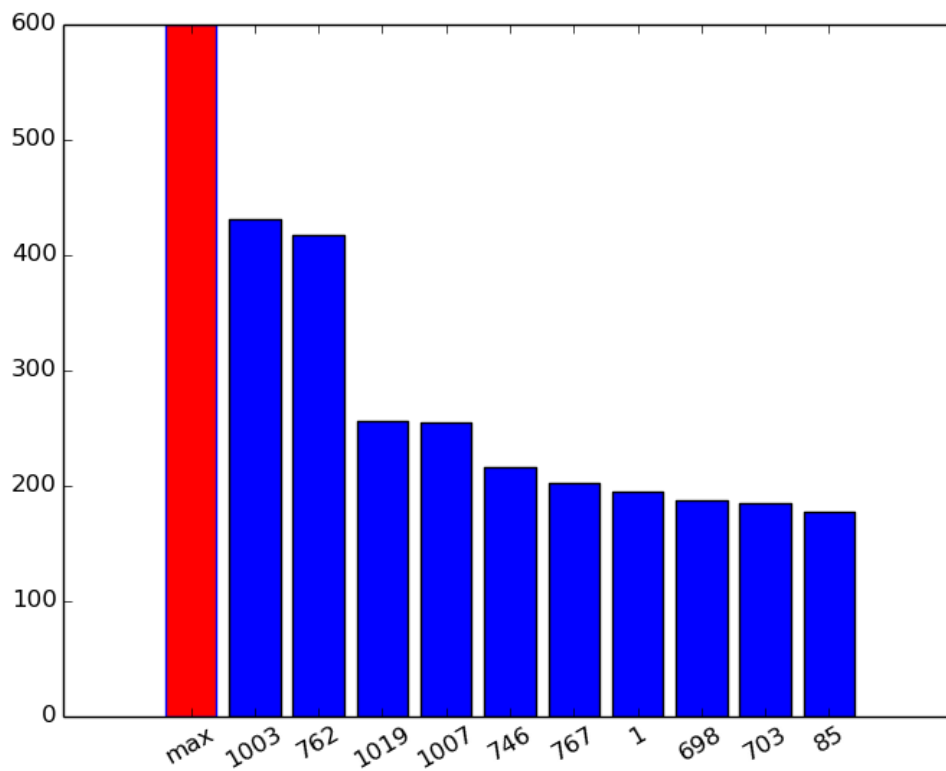


Figure 4.6: Bar plot analyzing consistency of feature selection on 5-mers, 10 dimensions and two classes.

4.1.2.2 Three Class Problem LOOCV

Leave-one-out cross-validation is used in this experiment against the Johnson Grass Dataset. One large experiment was performed using only five dimensions and 7-mers as other informed by the results of other experiments which suggest these parameters will produce good results. Figures 4.7 and 4.8 show the results of the experiment using the Nave Bayes Classifier and the Student's t-test as the feature selection algorithm, respectively.

4.1.2.3 7-Mer & 5 Dimensions LOOCV

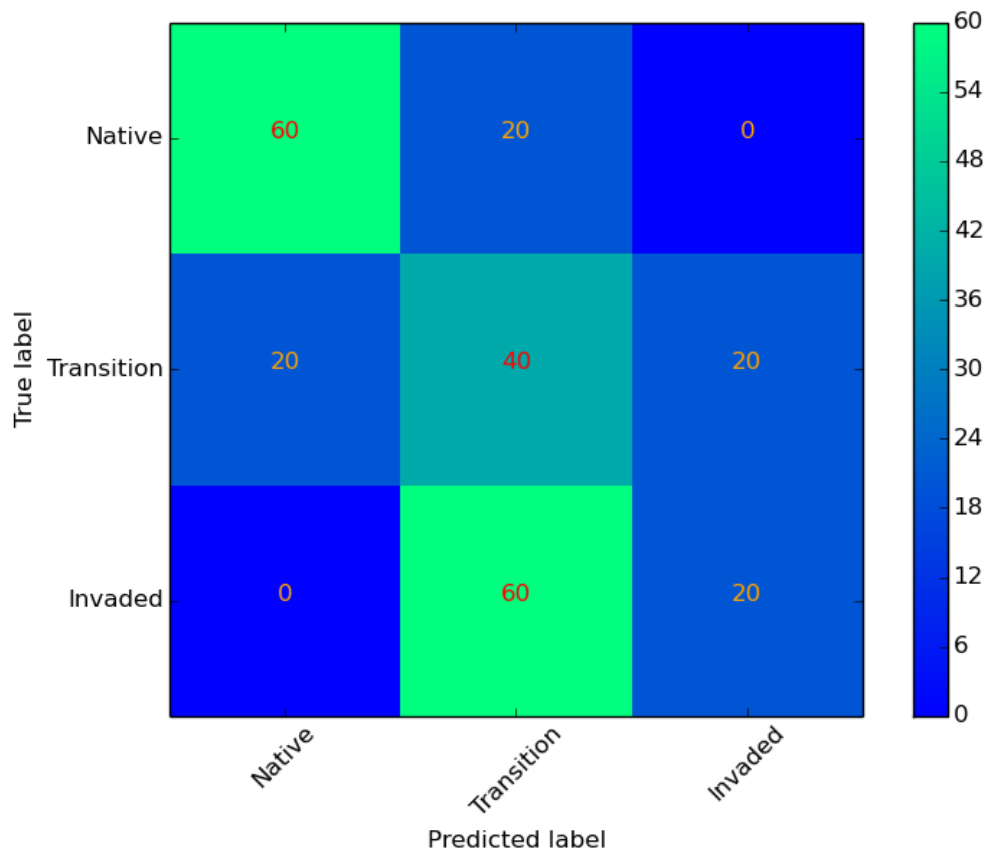


Figure 4.7: Confusion Matrix. 7-mers, 5 dimensions, 20 trials. Each trial uses leave-one-out cross-validation. Total accuracy of this run is 50.00%.

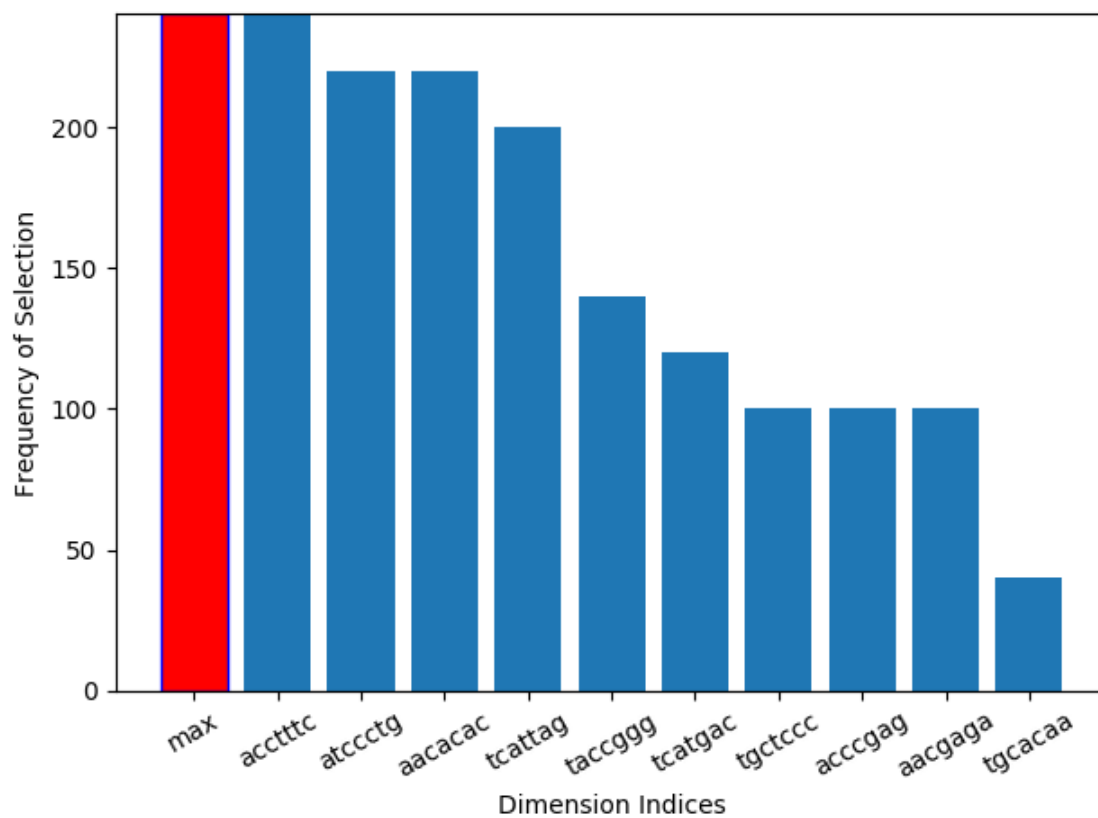


Figure 4.8: Bar plot analyzing consistency of feature selection on 7-mers and 5 Dimensions.

4.1.2.4 Three Class Problem MCCV

Comparing a test sample to training data from three classes is a more complex problem than the two-class case, requiring more data processing time to run the experiments. Results from varying the size of the k-mer and number of selected dimensions can be seen in Table 4.2. Figures 4.9, 4.11, 4.13, 4.15, 4.17, 4.19, and 4.21. Figures 4.24, 4.26, 4.29, 4.31, and 4.34 are directly related to the values seen in the Table 4.2. Each experiment is divided into its own sub-subsection.

In addition to the confusion matrix for each experiment shown in Table 4.2, the frequency of the most selected n dimensions is shown in Figures 4.10, 4.12, 4.14, 4.16, 4.18, 4.20, 4.22, 4.25, 4.27, 4.30, 4.32, and 4.35. These plots show dimensions that were most commonly selected by the feature selection mechanism of an all-pairwise Student's t-test to identify dimensions that contribute the most to discriminatory power.

Figures 4.23, 4.28, and 4.38 are scatter plots where the count of k-mers in a specific dimension of the NSV are graphed against the frequency of dimension selection. The points on these plots correspond to exactly one index in the NSV, and only dimensions that were selected at least once are shown in the plots. The summation of dimension frequency selection is across all trials of the experiment related to each respective plot. Each of these figures explores the possible connection between count and dimension frequency to assess whether there is an apparent relationship between these two aspects of the data.

Table 4.2: Varying k-mer size using Student's t-test and the Naive Bayes Classifier. Confusion matrix accuracies for the three-class problem are presented in this table.

Size of k	Top 5 Dimensions	Top 10 Dimensions
3	35.77	41.13
4	36.05	34.79
5	39.91	40.77
6	41.30	40.52
7	46.01	43.33
8	50.12	49.10

4.1.2.5 3-Mer & 5 Dimensions

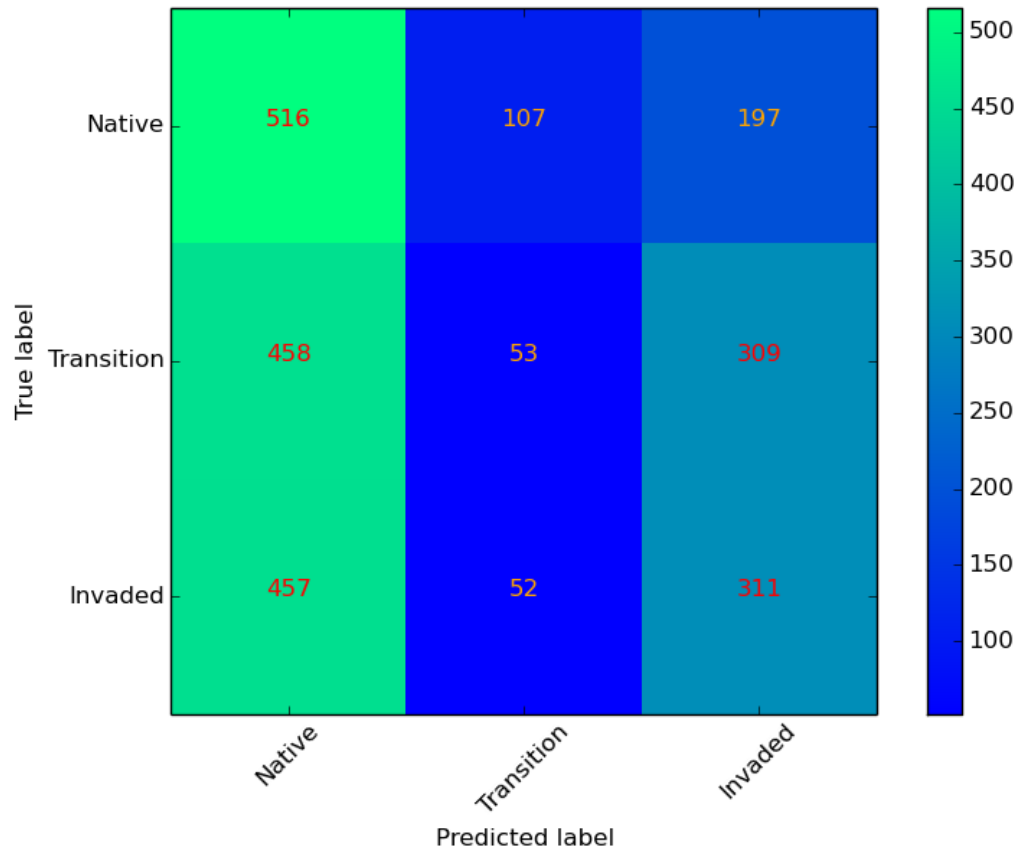


Figure 4.9: Confusion Matrix. 3-mers, 5 dimensions, 20 trials, and 3 classes. Each trial uses leave-one-out cross-validation and pools training data together. Total accuracy of this run is 35.77%.

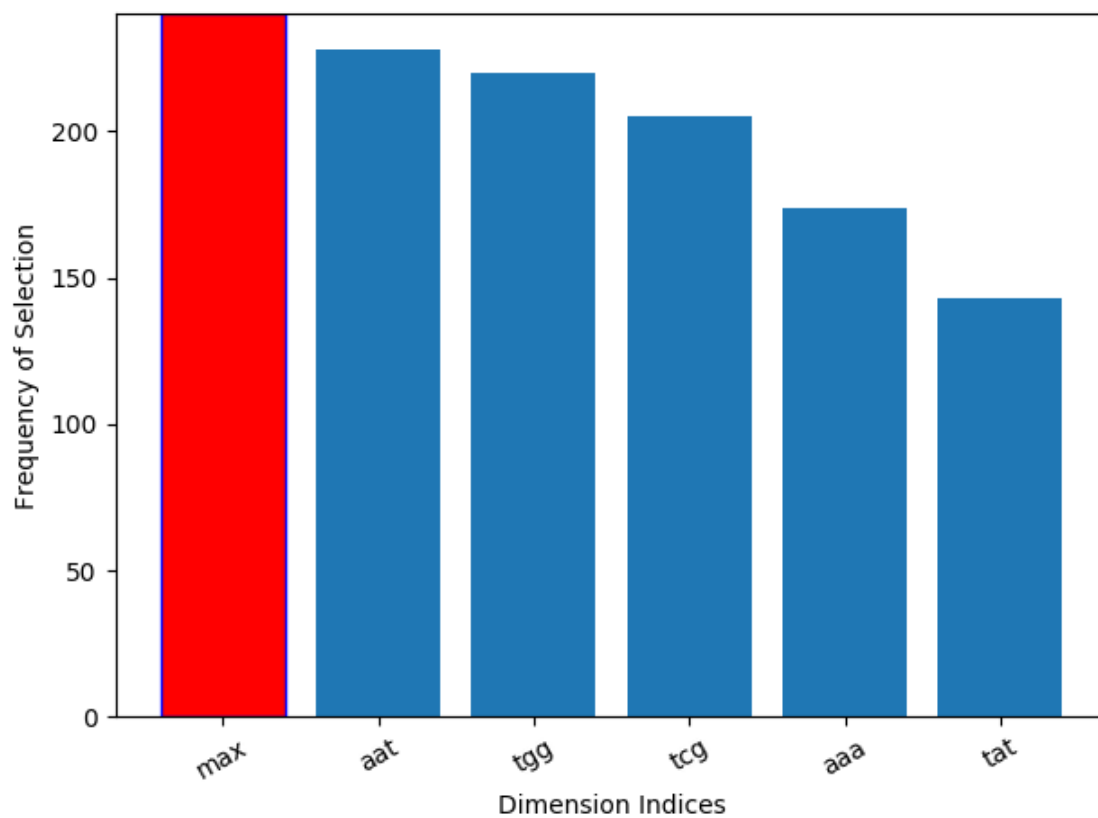


Figure 4.10: Bar plot analyzing consistency of feature selection on 3-mers and 5 Dimensions.

4.1.2.6 3-Mer & 10 Dimensions

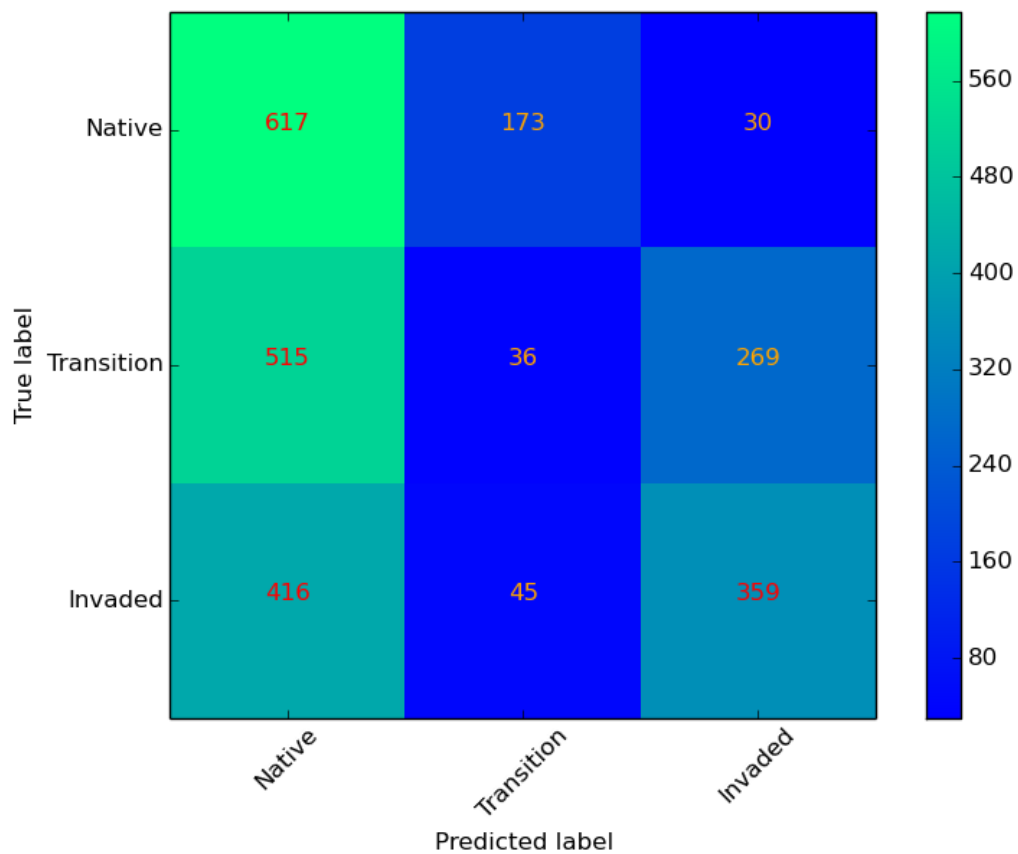


Figure 4.11: Confusion Matrix. 3-mers, 10 dimensions, 20 trials, and 3 classes. Each trial uses leave-one-out cross-validation and pools training data together. Total accuracy of this run is 41.13%.

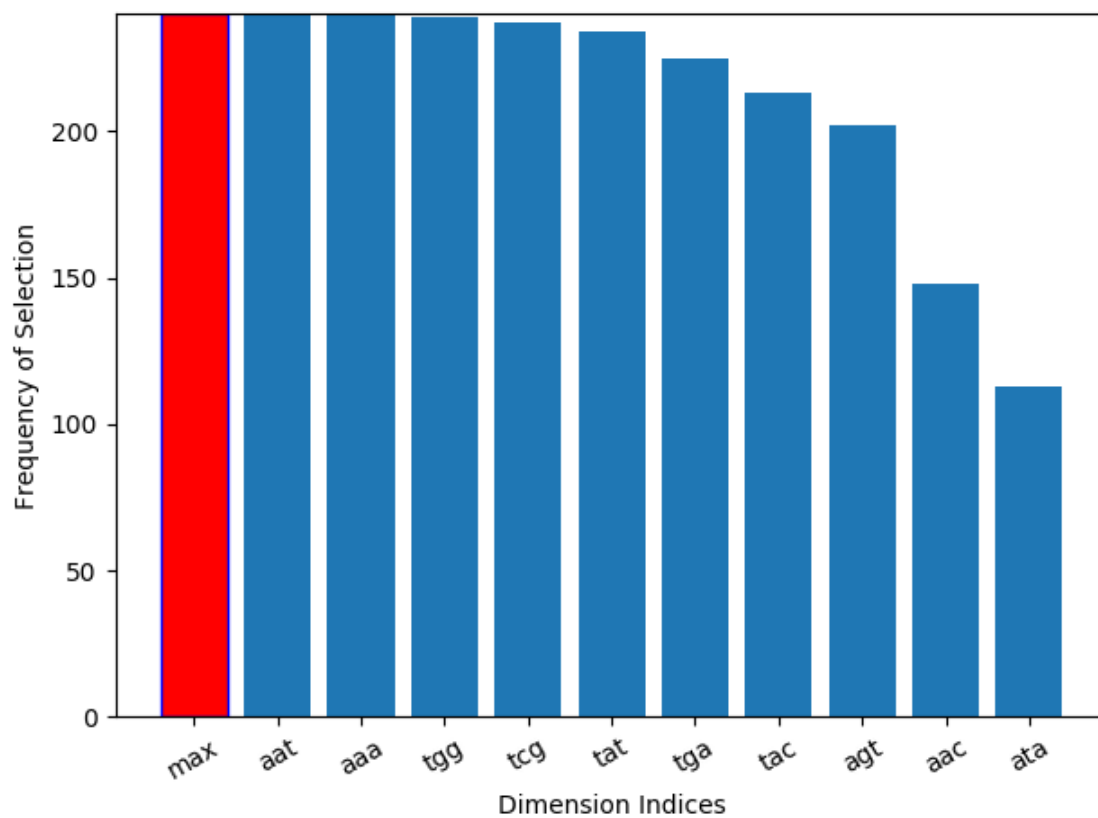


Figure 4.12: Bar plot analyzing consistency of feature selection on 3-mers and 10 Dimensions.

4.1.2.7 4-Mer & 5 Dimensions

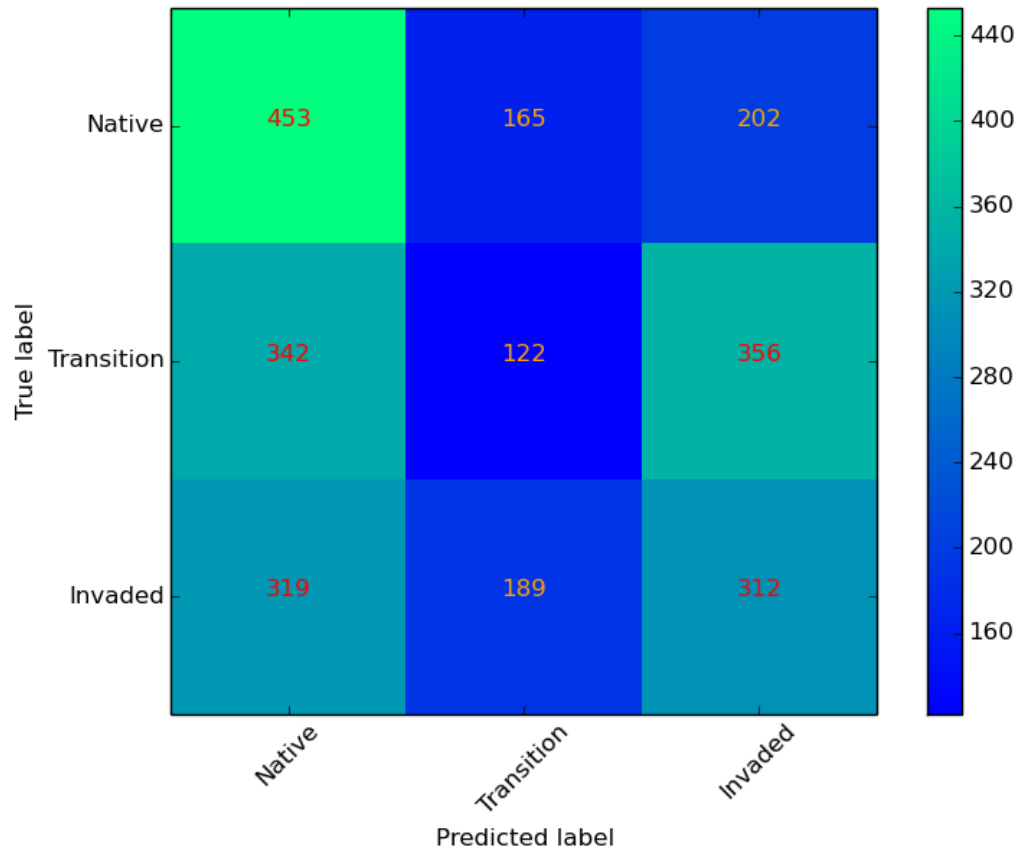


Figure 4.13: Confusion Matrix. 4-mers, 5 dimensions, 20 trials, and 3 classes. Each trial uses leave-one-out cross-validation and pools training data together. Total accuracy of this run is 36.05%.

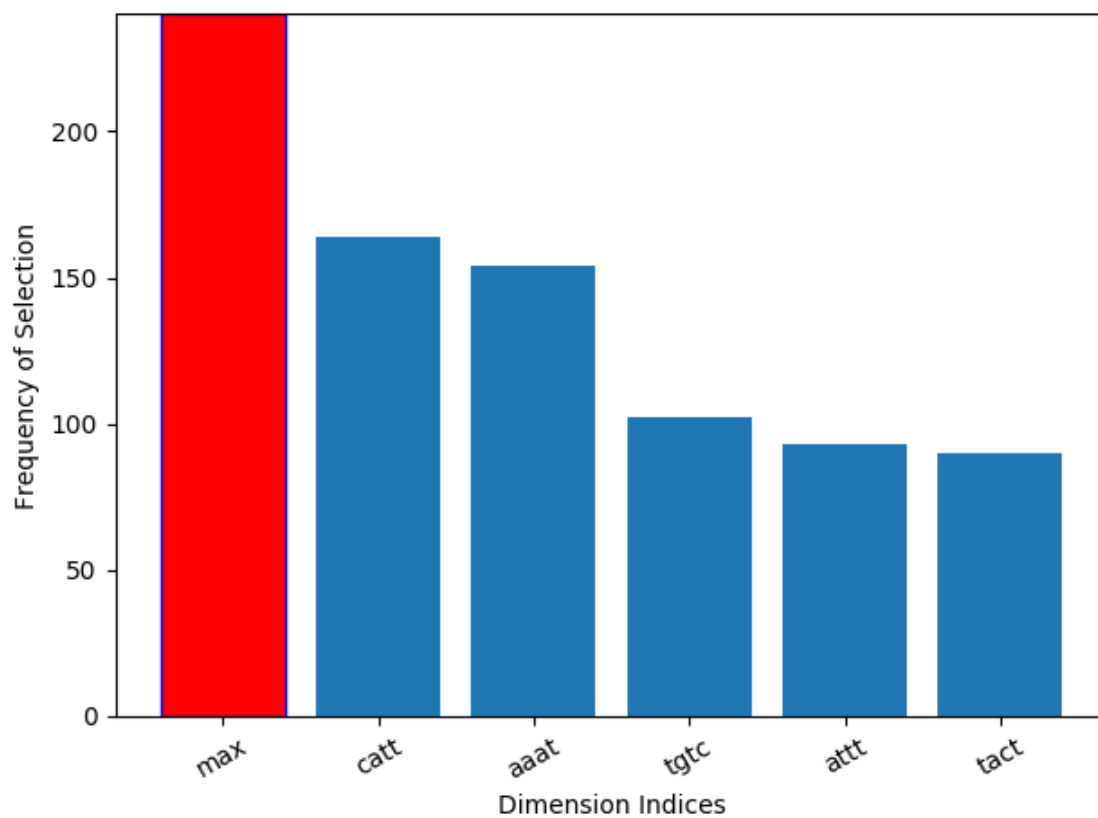


Figure 4.14: Bar plot analyzing consistency of feature selection on 4-mers and 5 Dimensions.

4.1.2.8 4-Mer & 10 Dimensions

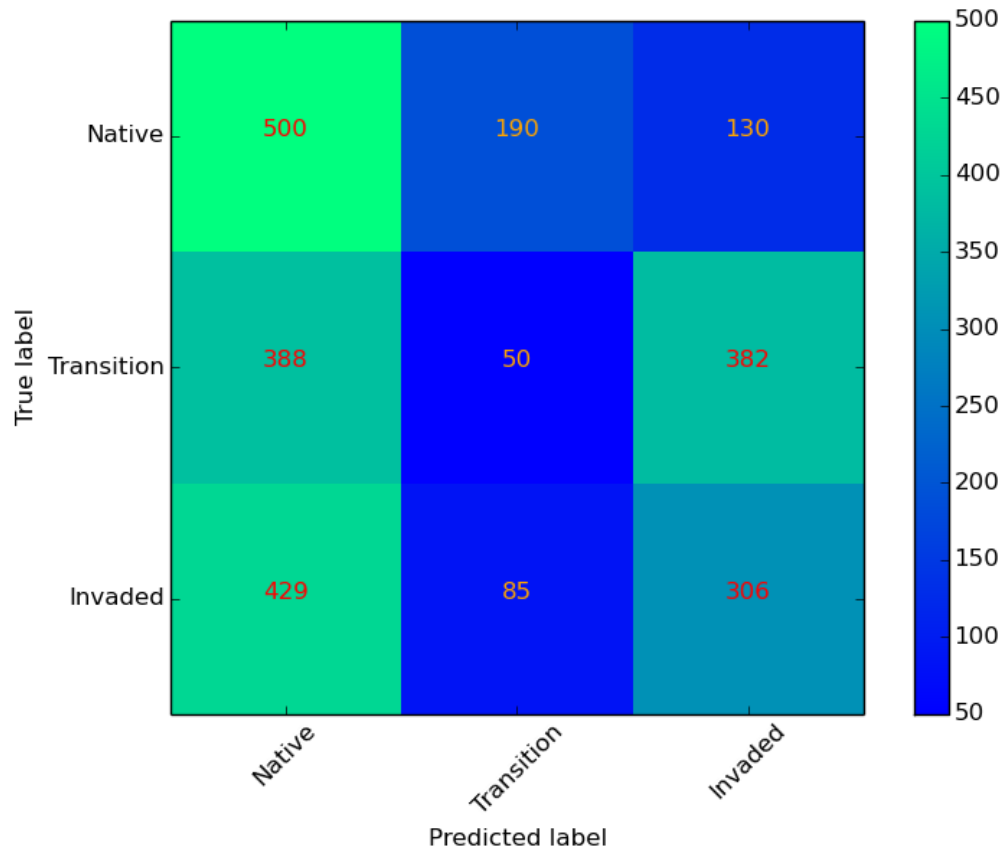


Figure 4.15: Confusion Matrix. 4-mers, 10 dimensions, 20 trials, and 3 classes. Each trial uses leave-one-out cross-validation and pools training data together. Total accuracy of this run is 34.79%.

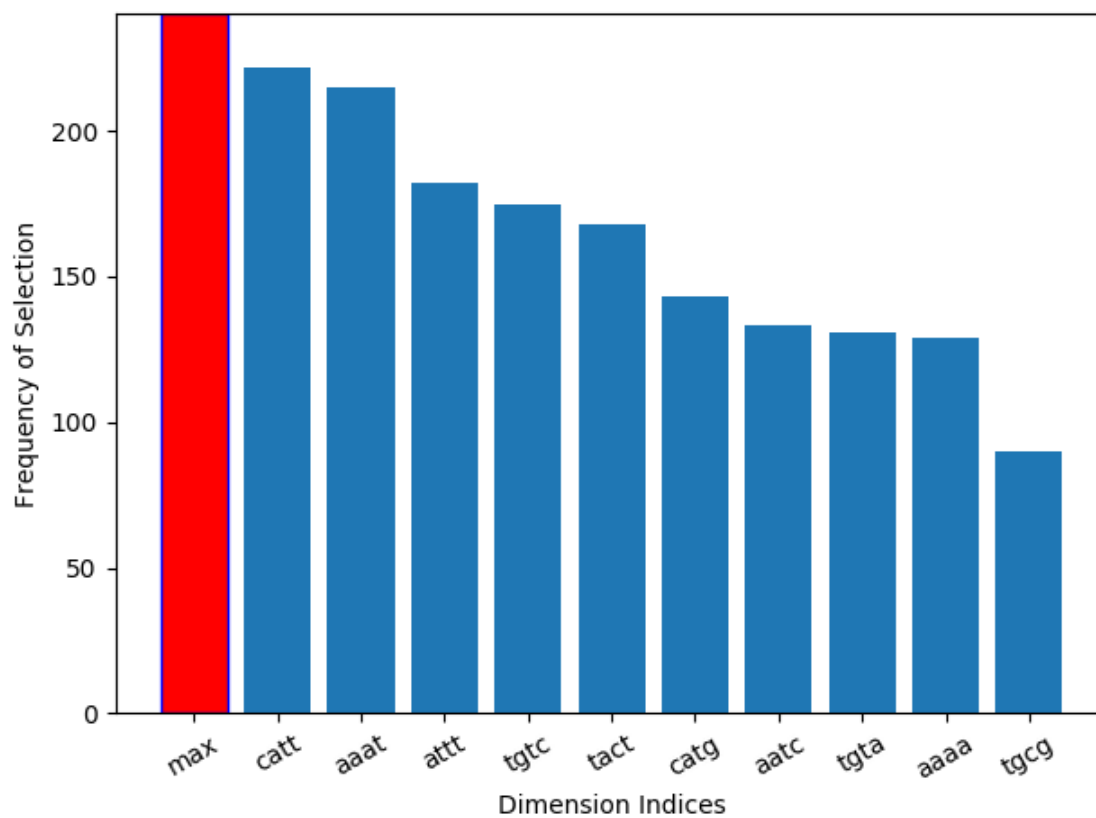


Figure 4.16: Bar plot analyzing consistency of feature selection on 4-mers and 10 Dimensions.

4.1.2.9 5-Mer & 5 Dimensions

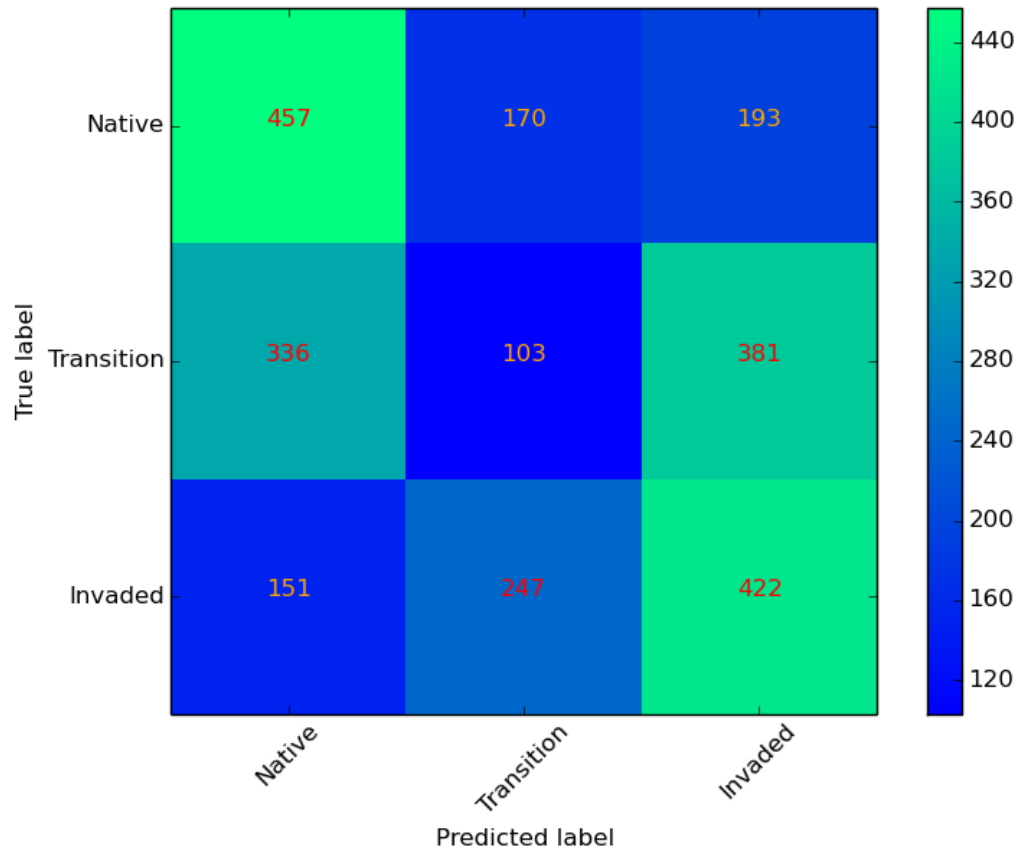


Figure 4.17: Confusion Matrix. 5-mers, 5 dimensions, 20 trials, and 3 classes. Each trial uses leave-one-out cross-validation and pools training data together. Total accuracy of this run is 39.91%.

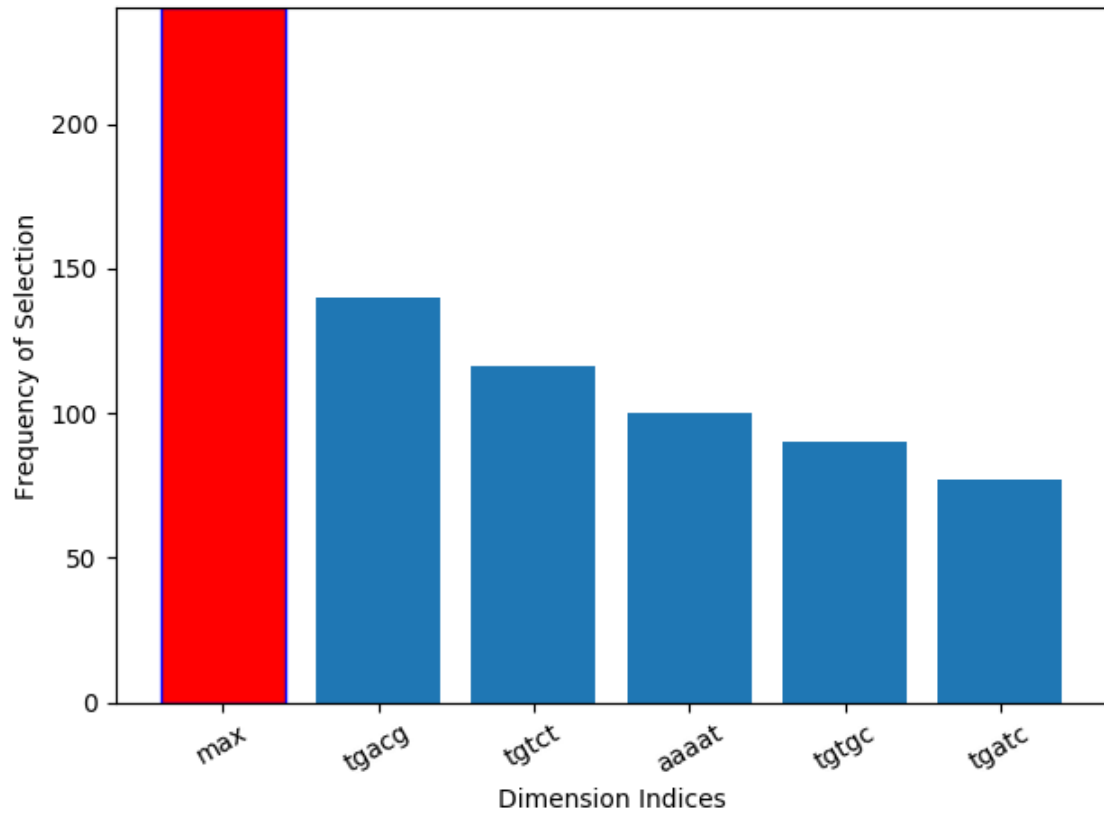


Figure 4.18: Bar plot analyzing consistency of feature selection on 5-mers and 5 Dimensions.

4.1.2.10 5-Mer & 10 Dimensions

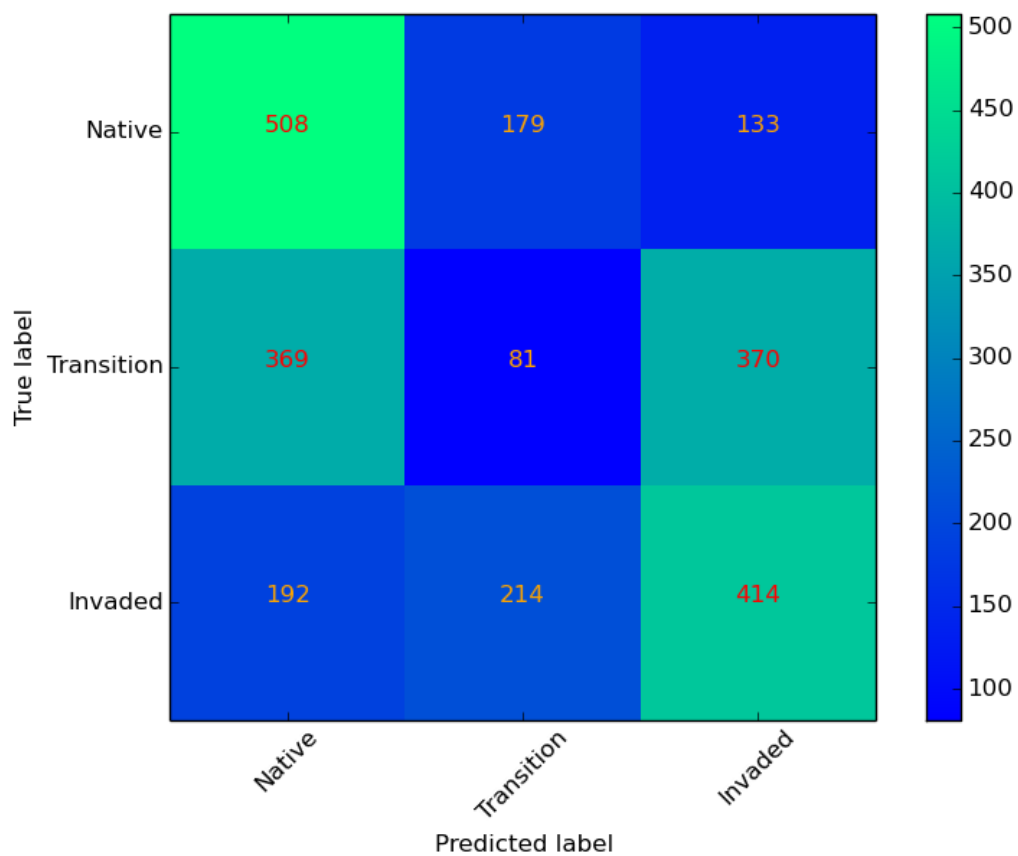


Figure 4.19: Confusion Matrix. 5-mers, 10 dimensions, 20 trials, and 3 classes. Each trial uses leave-one-out cross-validation and pools training data together. Total accuracy of this run is 40.77%.

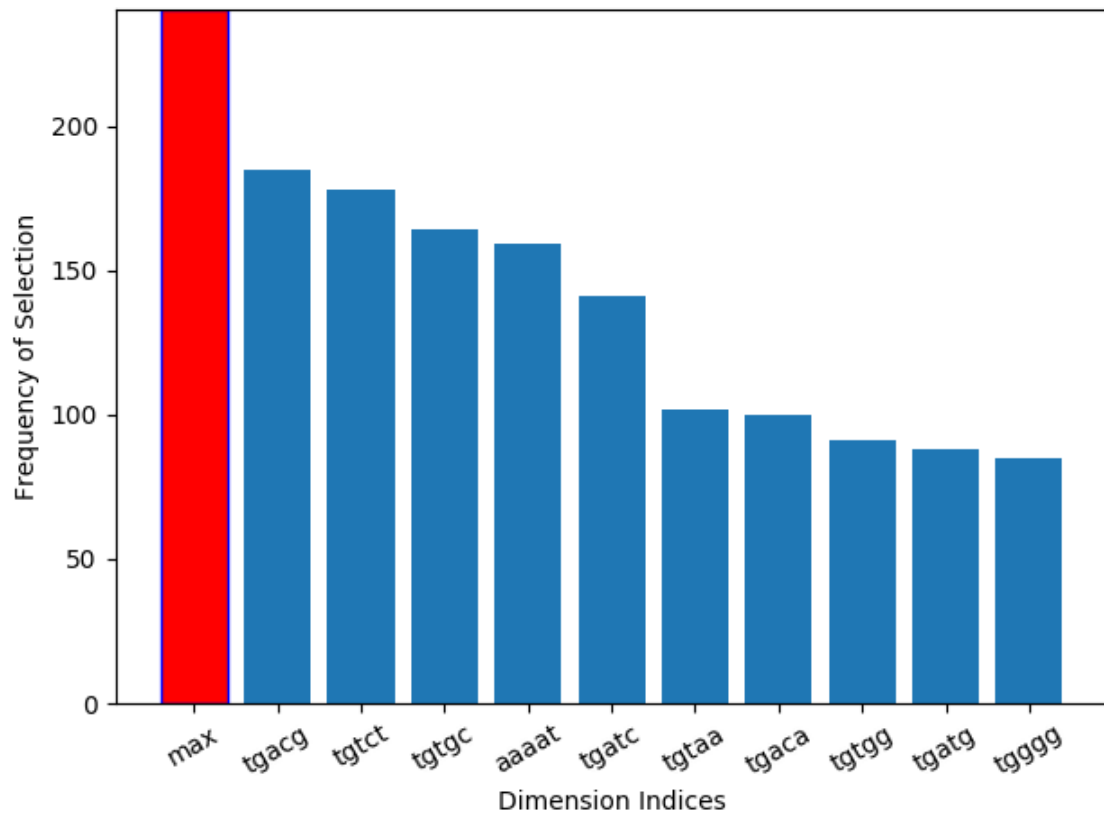


Figure 4.20: Bar plot analyzing consistency of feature selection on 5-mers and 10 Dimensions.

4.1.2.11 6-Mer & 5 Dimensions

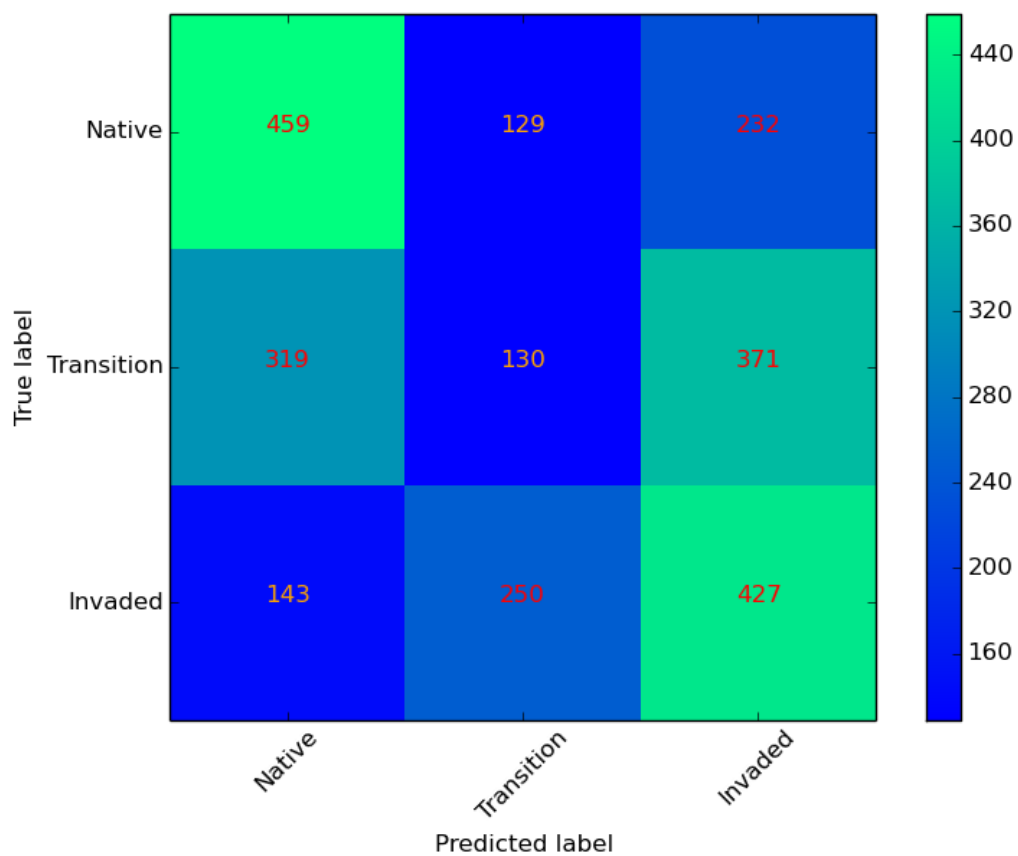


Figure 4.21: Confusion Matrix. 6-mers, 5 dimensions, 20 trials, and 3 classes. Each trial uses leave-one-out cross-validation and pools training data together. Total accuracy of this run is 41.30%.

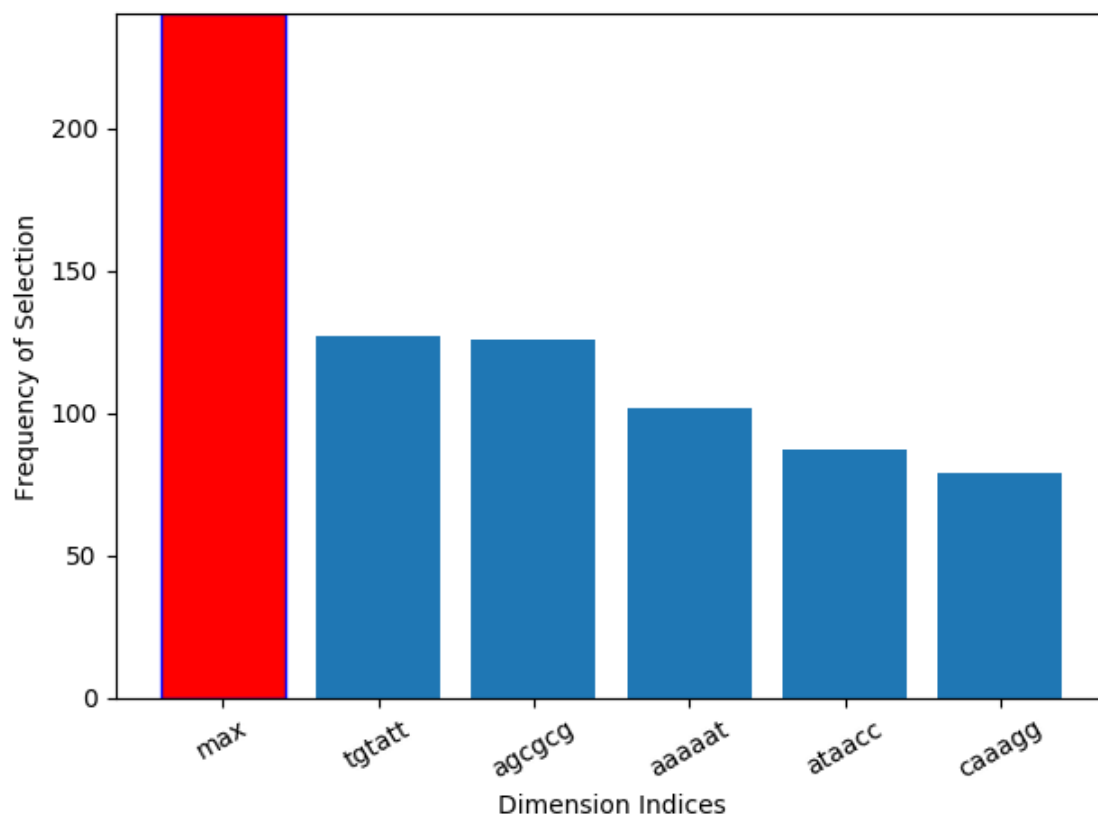


Figure 4.22: Bar plot analyzing consistency of feature selection on 6-mers and 5 Dimensions.

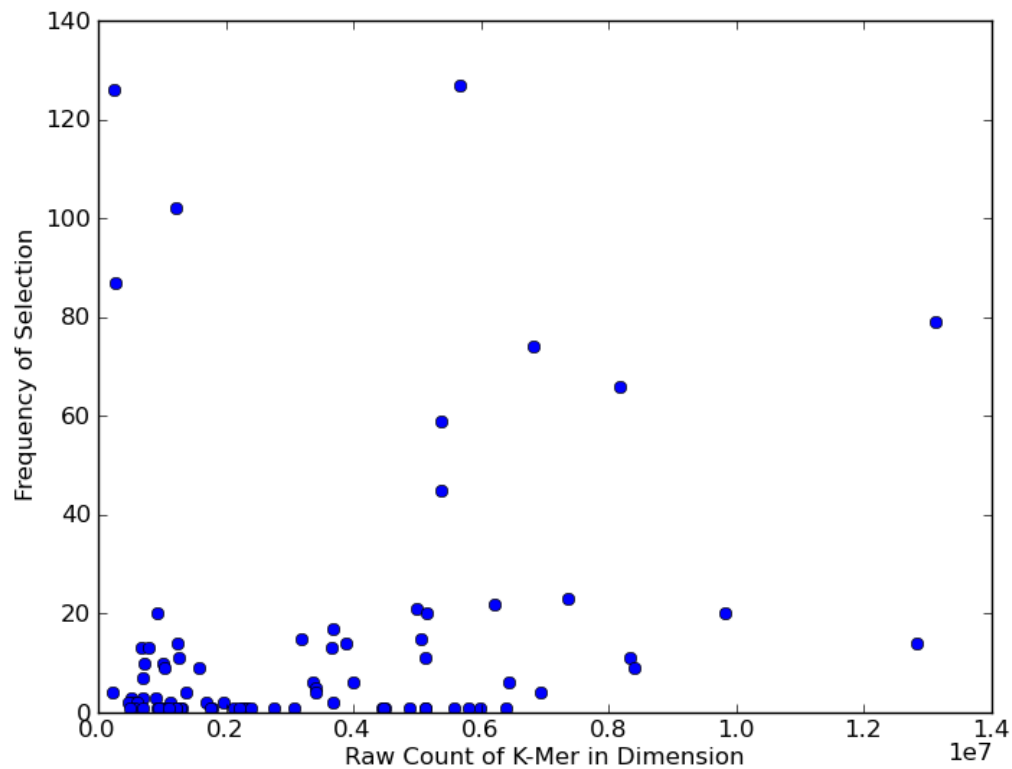


Figure 4.23: 6-mers and 5 dimensions scatter plot dimension frequency vs raw count in dimension. This scatter plot shows the frequency at which the dimensions are selected on the Y-axis and the raw count in that dimension in all NSVs from the experiment prior to normalization on the X-axis this is for the experiment with 6-mers and 5 dimensions using the data from all 20 trials.

4.1.2.12 6-Mer & 10 Dimensions

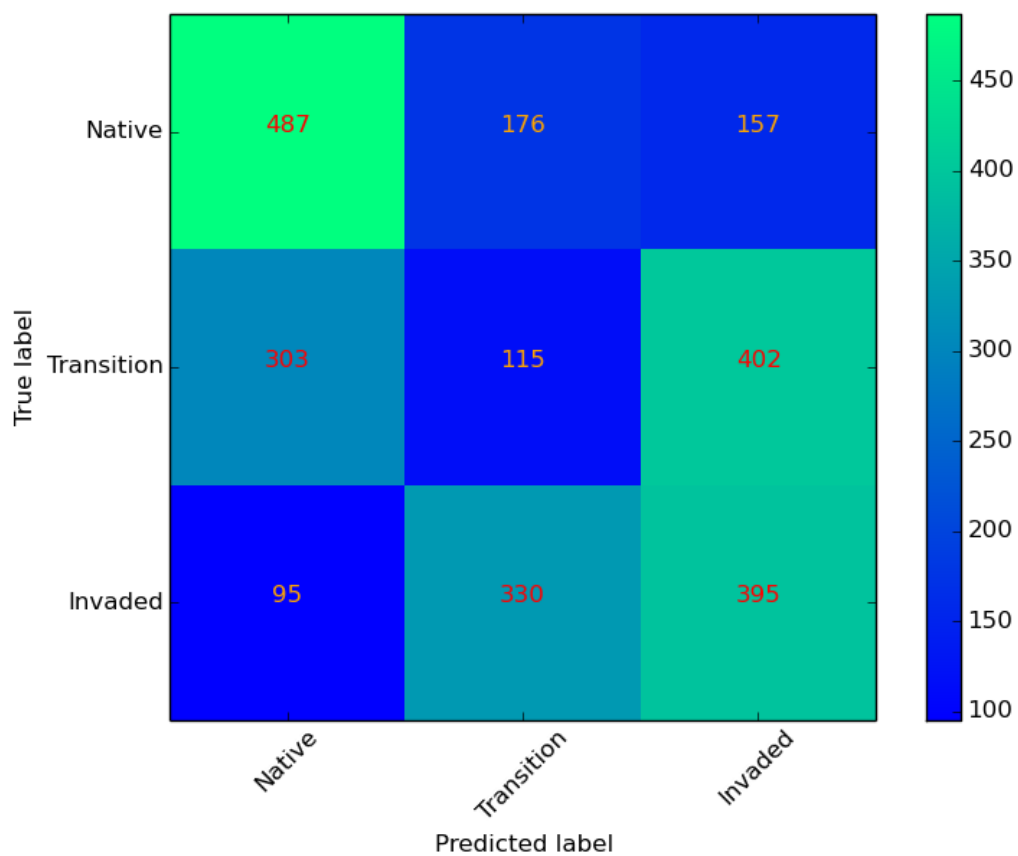


Figure 4.24: Confusion Matrix. 6-mers, 10 dimensions, 20 trials, and 3 classes. Each trial uses leave-one-out cross-validation and pools training data together. Total accuracy of this run is 40.52%.

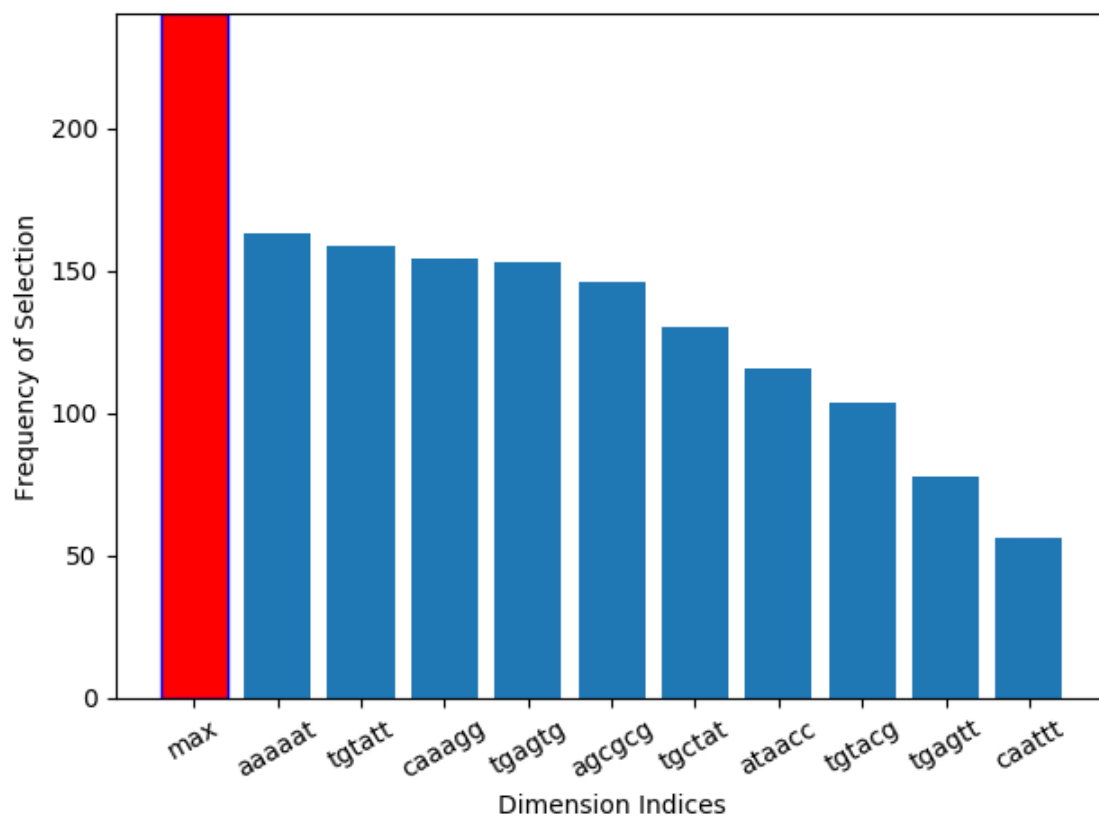


Figure 4.25: Bar plot analyzing consistency of feature selection on 6-mers and 10 Dimensions.

4.1.2.13 7-Mer & 5 Dimensions

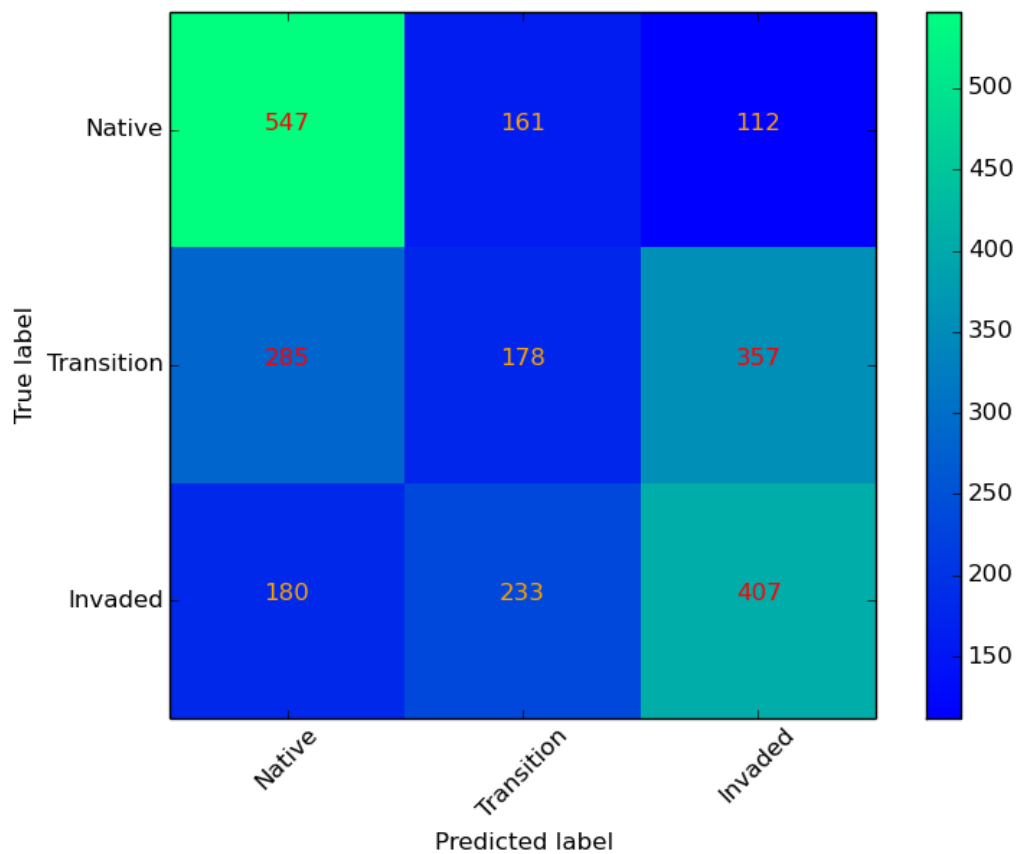


Figure 4.26: Confusion Matrix. 7-mers, 5 dimensions, 20 trials, and 3 classes. Each trial uses leave-one-out cross-validation and pools training data together. Total accuracy of this run is 46.01%.

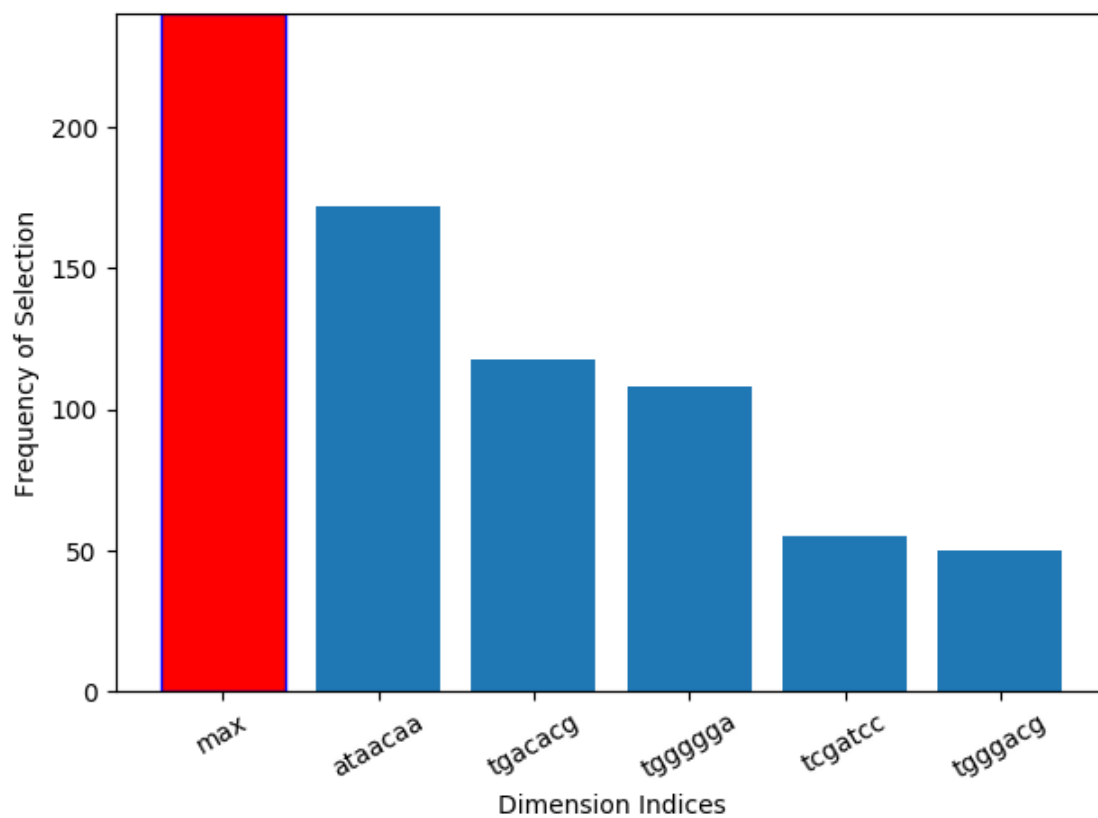


Figure 4.27: Bar plot analyzing consistency of feature selection on 7-mers and 5 Dimensions.

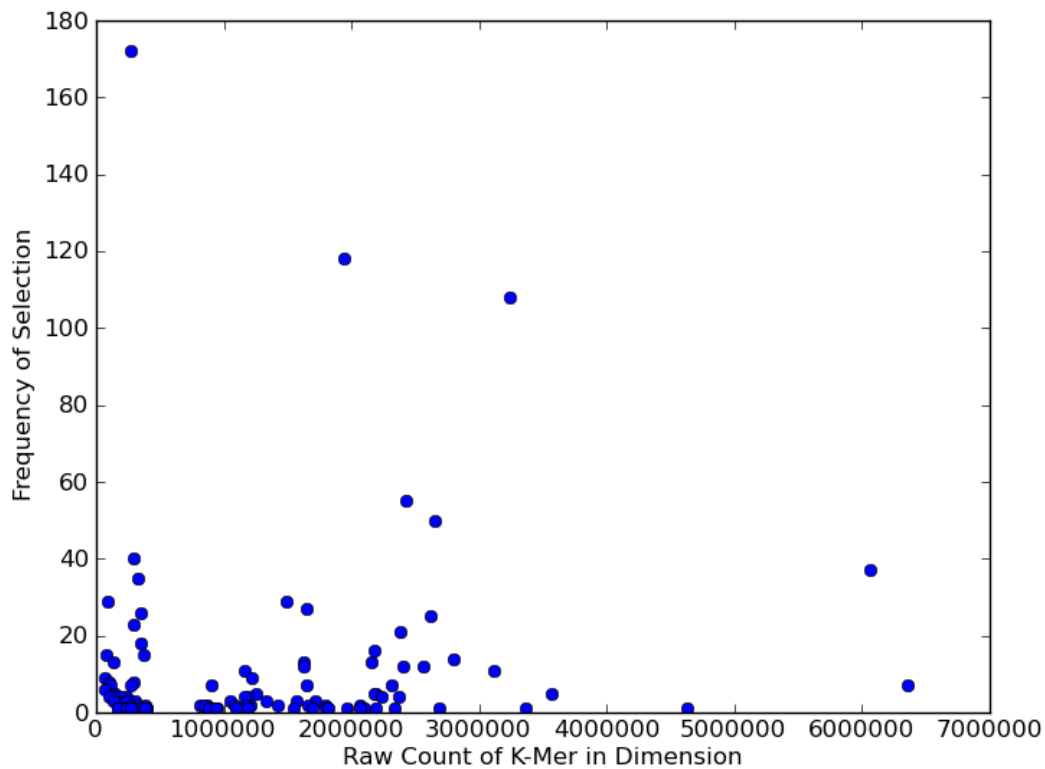


Figure 4.28: 7-mers and 5 dimensions scatter plot dimension frequency vs raw count in dimension. Frequency at which the dimensions are selected on the Y-axis and the raw count in that dimension in all NSVs from the experiment prior to normalization on the X-axis this is for the experiment with 7-mers and 5 dimensions using the data from all 20 trials.

4.1.2.14 7-Mer & 10 Dimensions

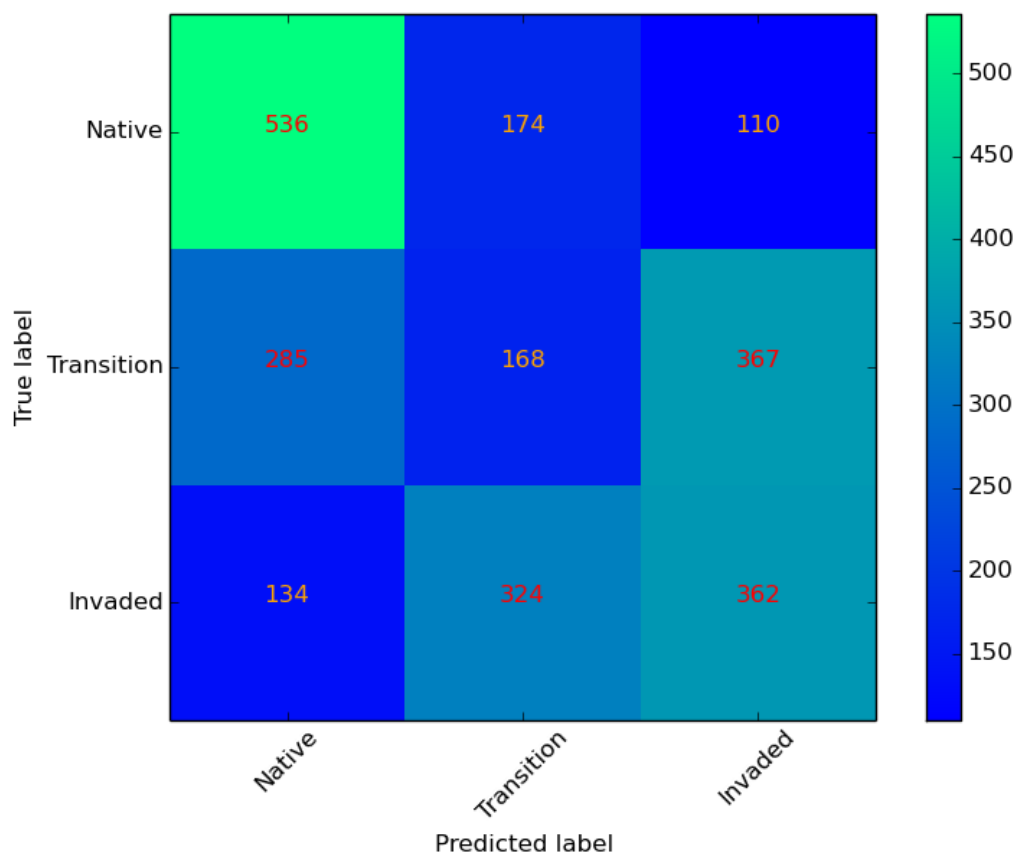


Figure 4.29: Confusion Matrix. 7-mers, 10 dimensions, 20 trials, and 3 classes. Each trial uses leave-one-out cross-validation and pools training data together. Total accuracy of this run is 43.33%.

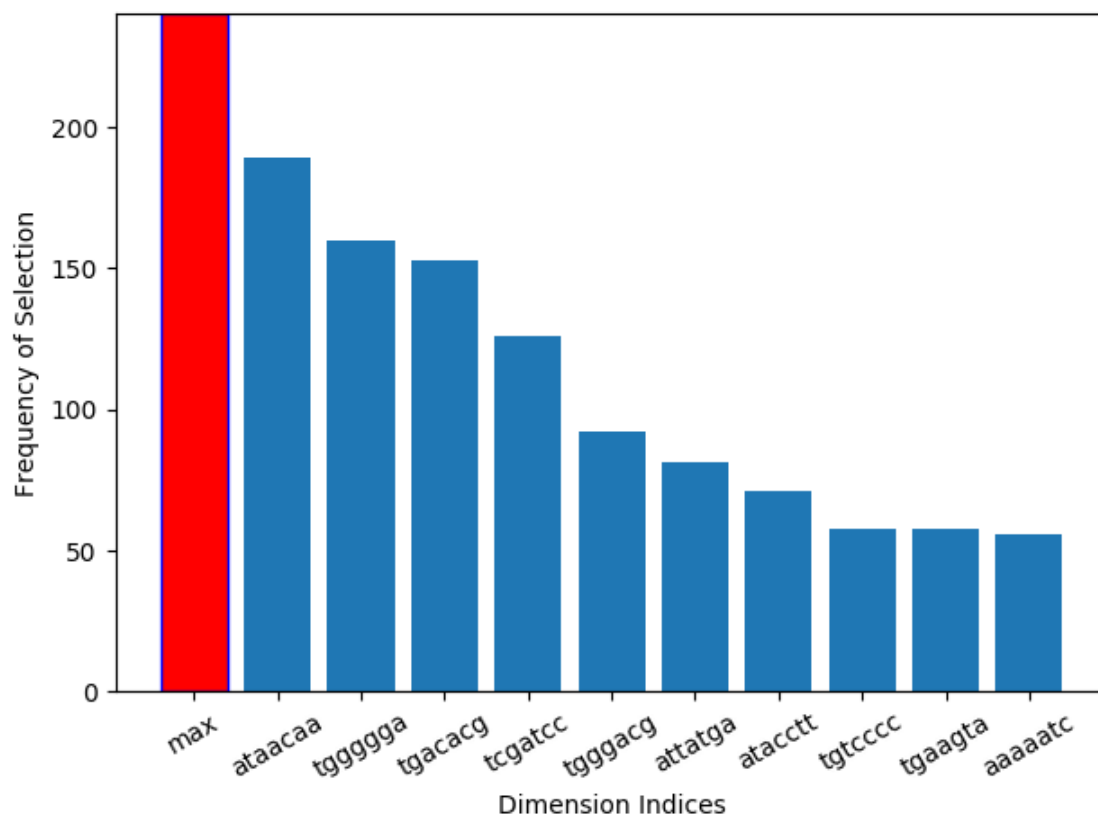


Figure 4.30: Bar plot analyzing consistency of feature selection on 7-mers and 10 Dimensions.

4.1.2.15 8-Mer & 5 Dimensions

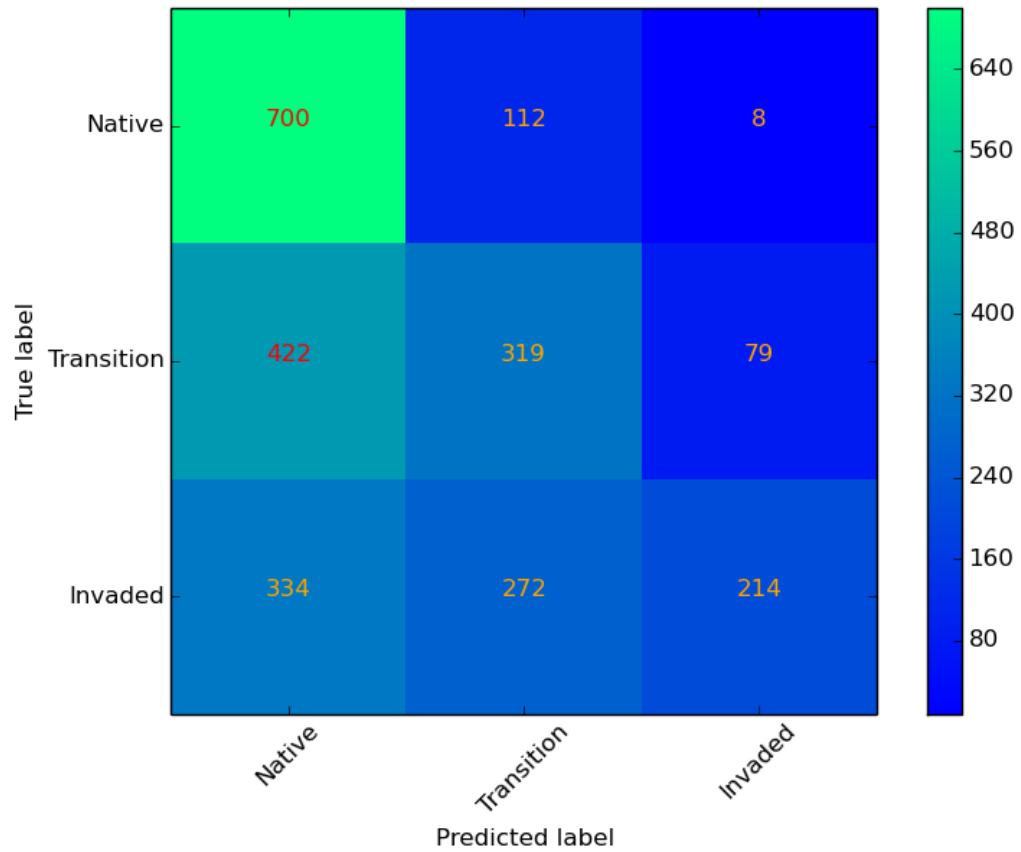


Figure 4.31: Confusion Matrix. 8-mers, 5 dimensions, 20 trials, and 3 classes. Each trial uses leave-one-out cross-validation and pools training data together. Total accuracy of this run is 50.12%.

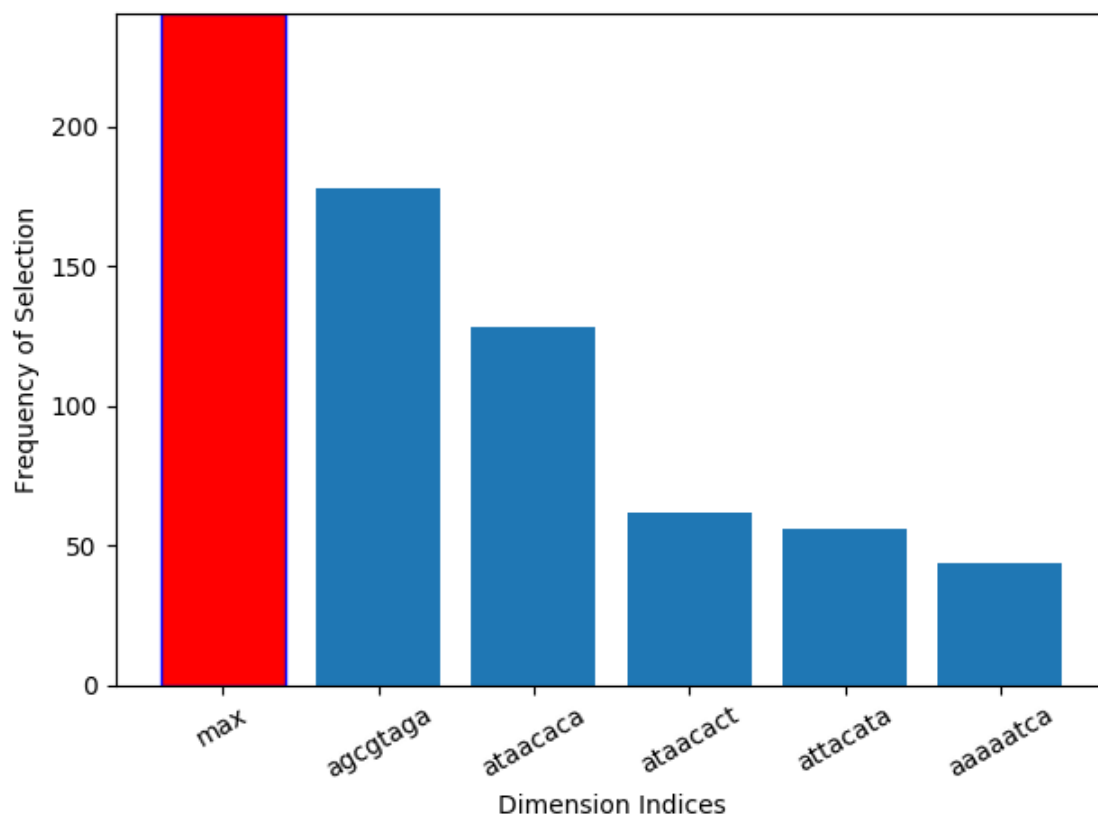


Figure 4.32: Bar plot analyzing consistency of feature selection on 8-mers and 5 Dimensions.

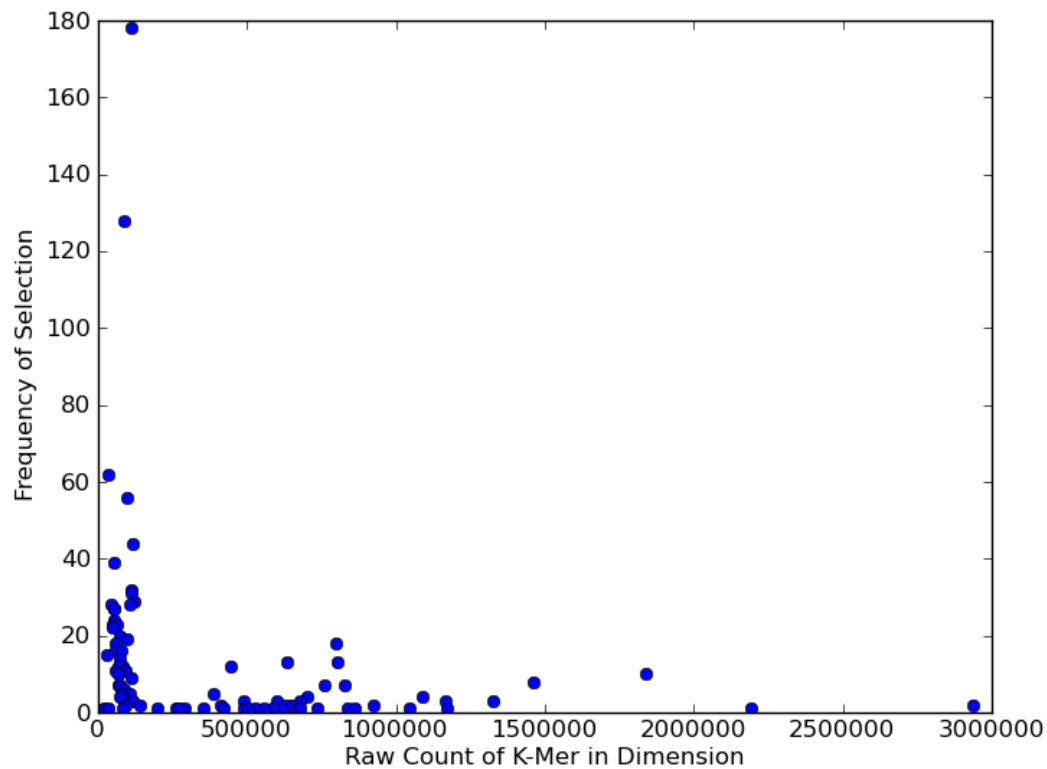


Figure 4.33: 8-mers and 5 dimensions scatter plot dimension frequency vs raw count in dimension. Frequency at which the dimensions are selected on the Y-axis and the raw count in that dimension in all NSVs from the experiment prior to normalization on the X-axis this is for the experiment with 8-mers and 5 dimensions using the data from all 20 trials.

4.1.2.16 3-Mer & 10 Dimensions

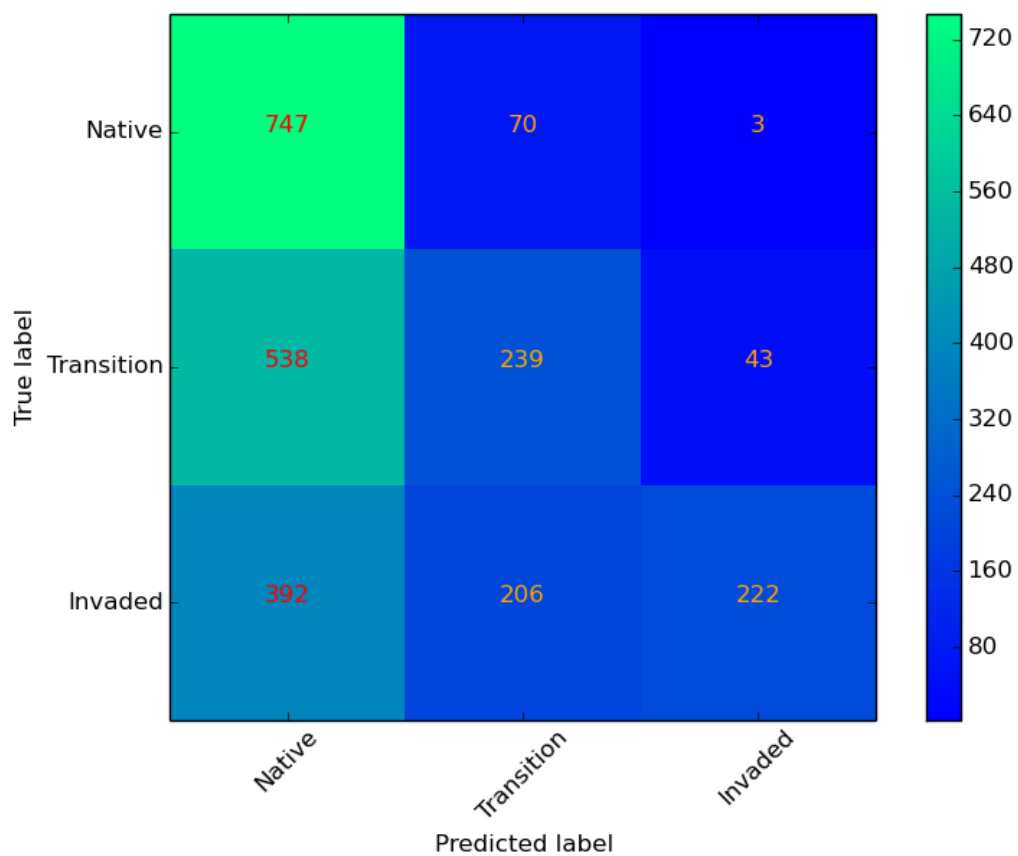


Figure 4.34: Confusion Matrix. 8-mers, 10 dimensions, 20 trials, and 3 classes. Each trial uses leave-one-out cross-validation and pools training data together. Total accuracy of this run is 49.10%.

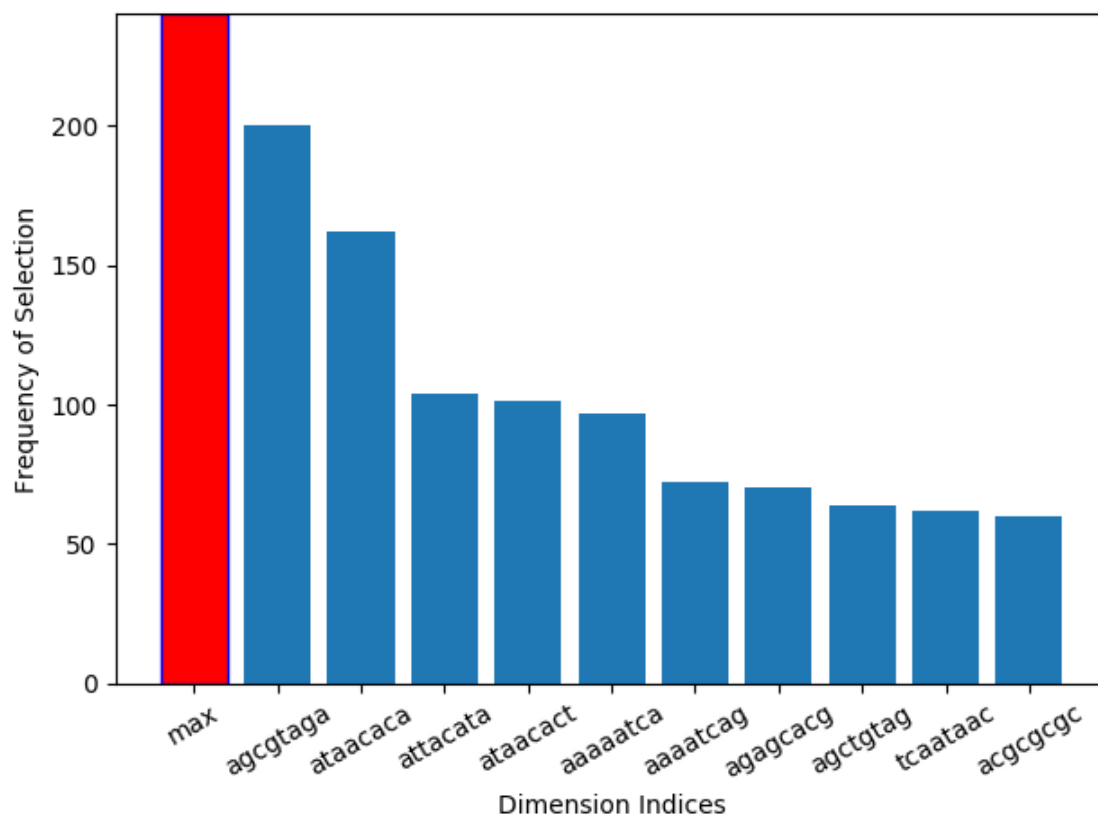


Figure 4.35: Bar plot analyzing consistency of feature selection on 8-mers and 10 Dimensions.

4.2 Human Gut Dataset

The Human Gut Dataset has great diversity of class representation. The distribution of fragments for each class is uneven - Table 4.3. The experiment uses a Monte Carlo cross-validation technique and limits the number of fragments for testing and training to be an even number amongst the classes. All fragments from each class were pooled together and the experiment ran in a similar way to the experiment performed with the Johnson Grass dataset. Only one experiment is performed using with the Human Gut dataset using parameters given the information from the previous experiment. Figures 4.37 4.37 4.38 show the results of the experiment using the Naïve Bayes classifier with Monte Carlo cross-validation and with the Human Gut dataset.

The Human gut dataset has great diversity of class representation. The distribution of fragments for each class is uneven as shown in Table 4.3. The experiment used a Monte Carlo cross-validation technique and limited the number of fragments for testing and training to be an even number amongst the classes. All fragments from each class were pooled together and the experiment was run in a similar way to the experiment performed with the Johnson Grass dataset. Only one experiment is performed with the Human Gut dataset using parameters given the information from the previous experiment which suggest these parameters will produce good results. Figures 4.37, 4.37, and 4.38 show the results of the experiment using the Nave Bayes Classifier with Monte Carlo Cross-Validation with the Human Gut dataset.

Table 4.3: Number of lines in each file and the total number of lines in all files of the Human Gut Dataset.

Line Count	File Name
1218534	Crohns-SRR053016Full.fasta
611178	Crohns-SRR053019Full.fasta
1068952	Crohns-SRR053034Full.fasta
532256	Crohns-SRR053036Full.fasta
2350246	Crohns-SRR054211Full.fasta
1375236	Crohns-SRR054212Full.fasta
805624	Healthy-SRR053013Full.fasta
783114	Healthy-SRR053023Full.fasta
251008	Healthy-SRR053027Full.fasta
443538	Healthy-SRR053031Full.fasta
49714	UC-SRR058718Full.fasta
1246118	UC-SRR059126Full.fasta
1179536	UC-SRR059127Full.fasta
1314246	UC-SRR059128Full.fasta
1336370	UC-SRR059129Full.fasta
25788326	UC-SRR060115Full.fasta
6429190	UC-SRR060116Full.fasta
46783186	TOTAL

4.2.0.1 8-Mer & 5 Dimensions Leave-One-Out

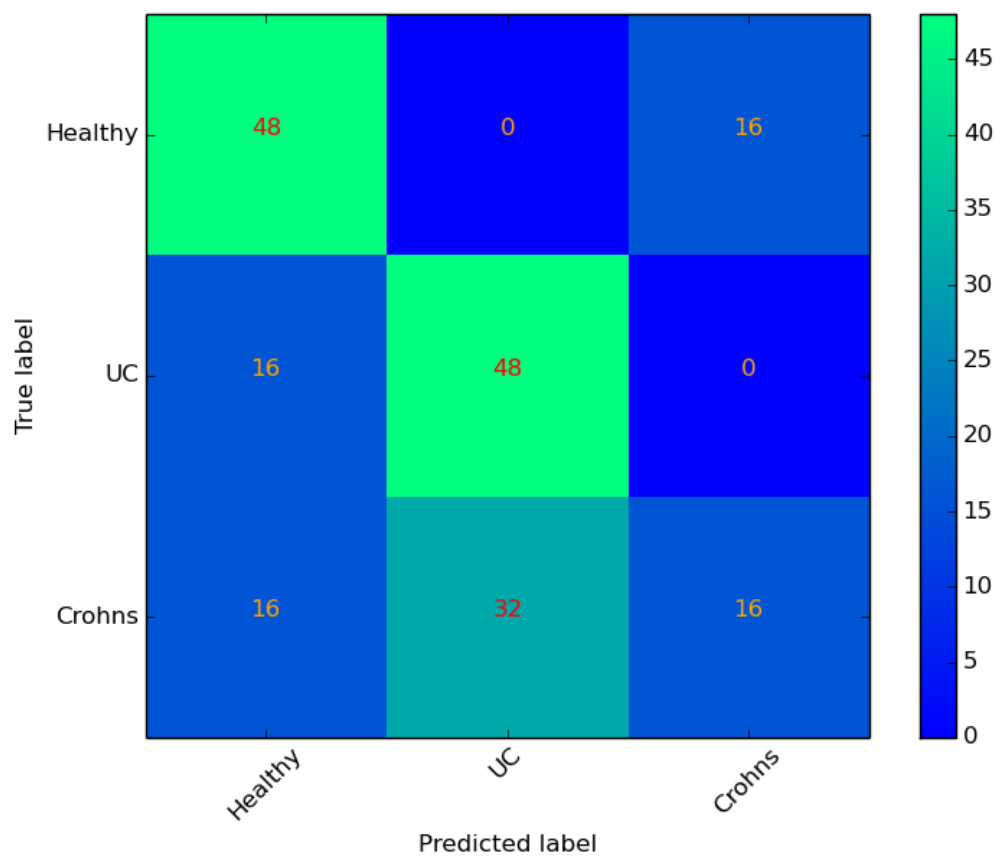


Figure 4.36: Confusion Matrix. 8-mers, 5 dimensions, 20 trials, and 3 classes. Each trial uses leave-one-out cross-validation and pools training data together. Total accuracy of this run is 58.33%.

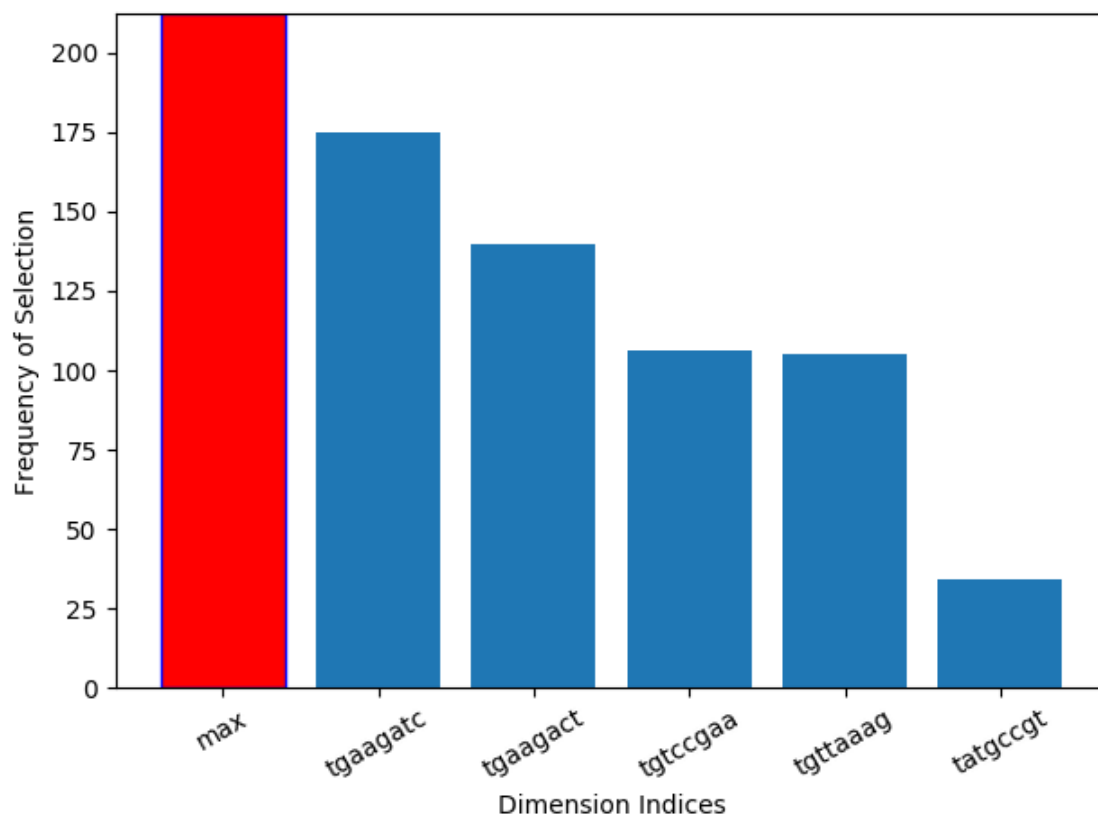


Figure 4.37: Bar plot analyzing consistency of feature selection on 8-mers and 5 Dimensions.

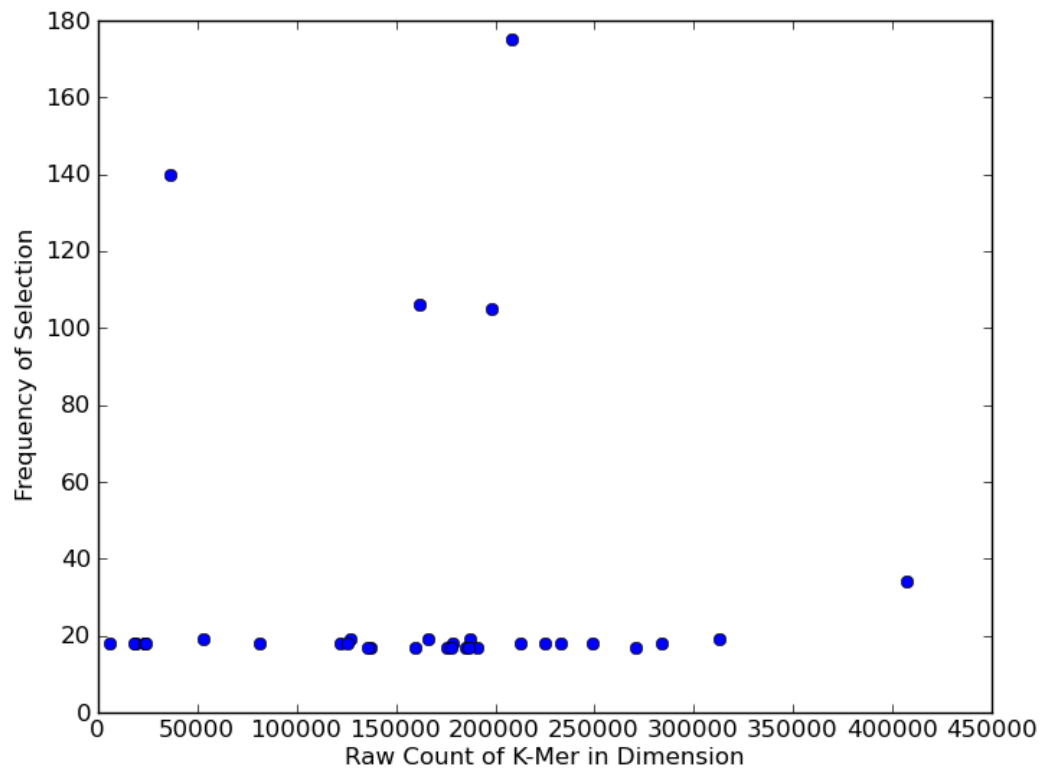


Figure 4.38: 8-mers and 5 dimensions scatter plot dimension frequency vs raw count in dimension. Frequency at which the dimensions are selected on the Y-axis and the raw count in that dimension in all NSVs from the experiment prior to normalization on the X-axis this is for the experiment with 8-mers and 5 dimensions using the data from all 20 trials.

CHAPTER 5 DISCUSSION

The k -mer analysis pipeline for the classification of metagenomic samples presented herein provides a mechanism to process metagenomic datasets into a reusable data structure. Additionally, there are scripts that visualize classification logs and other utilities to explore the results. This repository, https://bitbucket.org/russell_kaehler/sequence-free-analysis, contains all the scripts related to the work presented in this thesis.

5.1 General Discussion

Prior to the development of this technique, understanding the nature of a metagenomic sample required a great amount of known bacterial sequence information to be accessible. These genomes would serve as a reference set to compare the fragments of a new sample to known sequences. However, with this technique one can imagine going to a new environment and rapidly being able to monitor the fluctuations of an environment without necessarily knowing all the members of the bacterial community.

As the results show, this novel approach allows for rapid analysis of a new metagenomic sample dataset to identify the environmental class that generated the sample with reasonable accuracy. This type of approach is new to the scientific community. Using this technique to explore microbial communities can now be used by others in the research community to begin to study changes to a microbial community – such

as from a healthy to disease state.

Compressing metagenomic files into NVSs of varying mer sizes and storing those files allows for rapid re-processing of large datasets and the comparison of newly processed files to existing records. It should be noted that the information stored in an NSV is a compression of the metagenomic records and does not allow for full reconstruction of the sequences or genomes.

Figures 4.3, 4.5, 4.9, 4.11, 4.13, 4.15, 4.17, 4.19, 4.21, 4.24, 4.26, 4.29, 4.31, and 4.34 show that the use of machine learning classification algorithms can be applied directly to normalized NSVs in a pooled MCCV method with results that are better than chance. If LOOCV is used instead of MCCV, the results shown in Figures 4.7 and 4.36 show that LOOCV also performs better than chance and even performs slightly better than MCCV in some cases. Additional feature selection methods and classifiers are available in the pipeline shown in Figure 4.1.

5.1.1 Accuracy as a Function of k -mer Size

Based on the results presented herein, there is a clear relationship that by increasing the size of k for the k -mers accuracy increases. Table 4.2 shows the most consistent accuracy increase when 5 dimensions are selected. However, when ten dimensions are selected there's two cases where accuracy decreases slightly; specifically $K = 4$ and $K = 6$. Code Listing 3.1 shows in lines 19 – 21, that even k -mer sizes will have a larger NSV, which means that the information is slightly less densely packed than in cases where k is odd. This is because even-sized k -mers are unable to have a reverse complements of themselves, in some cases, due to how reverse complements are formed. It may be possible that the larger NSV generated from an even-sized k -mer and using more dimensions, each with less discriminatory power, that the extra information included did not contribute to discriminatory power and thus produces

incorrect results when classifying with the Naïve Bayes Classifier. The general trend of accuracy increasing with the selection of a larger values of k was expected from the literature and confirmed by the experiments. By increasing the size of k , the execution time required also increases and, as such, results are limited to small values of $3 \leq k \leq 8$. Future experiments with much larger values for k , including cases where $k > 20$, should be explored when resources are available. Limitations of system memory are the greatest concern with large k-mers. A selection of $k = 15$ will generate an NSV of length 1,073,741,824 and storing the NSV in memory would require more than 8.5 Gigabytes to hold a single array.

Research done previously in the University of Montana Microbial Ecology Research group has attempted to circumvent the issue of memory intensive array sizes by using a hashing approach. Other software applications such as Kracken [81] also use a hash approach to avoid storing the entire array in memory. These approaches work well for sparsely populated arrays, but have in practice been unable to deliver increased performance on data sets with relatively low counts such as the metagenomic data used in this research.

5.1.2 Dimensions Selection Frequency Across Experiments

Selecting dimensions using either of the feature selection algorithms across all sizes of k , whether selecting 5 or 10 dimensions, seemed to indicate that having only a few dimensions provided strong discriminatory power for the purposes of classification. This observation is supported in Figures 4.2, 4.4, 4.10, 4.12, 4.14, 4.16, 4.18, 4.20, 4.22, 4.25, 4.27, 4.30, 4.32, 4.35, and 4.37, where the first few dimensions are selected very often and then the remaining dimensions were selected less frequently and the graphs all resembled a decay curve.

The x axis of the bar plots showed that the feature selection methods selected NSV

dimensions which correspond to k -mers that repeat characters often and produce strings that are considered biologically unlikely. This may suggest an issue in the software pipeline in that the strings are not the least common or most common strings within the datasets. Nor is the distribution of the strings heavily biased toward one class—more discussion on this topic is continued in Section 5.1.3.

5.1.3 Dimension Selection and Observed k -Mer Count

Figures 4.23, 4.28, and 4.33 show that the count of a k -mer occurring in a normalized NSV and the frequency that the dimension gets selected appear to have some relationship. Specifically, in Figure 4.33 the most frequently selected dimensions of the NSV all have low counts in the NSV. This may suggest that less frequently occurring k -mers allow for the distinction between environmental classes to be identified by the feature selection methods. Figures 4.23, 4.28, and 4.33 only show points where the dimension is selected at least once in one trial, so low count may not be the only factor in dimension selection. Another point to consider is that the counts displayed are from the un-normalized NSV and so they may graph differently when viewed as the normalized real number.

5.2 Future Directions

Table 4.2 displays the relationship that by increasing k -mer size, classification accuracy also increases. Profiling the code and re-writing sections in C for speed optimizations may reduce running time. If such measures were taken, then the ability to perform these same experiments with much larger sizes of k -mers should be possible, as there are now cloud instances that have more than 1TB of memory available.

Like many other tools available to researchers in the metagenomics community, a

web based interface would allow many more people to operate the pipeline. This would allow for more data sets to be incorporated into the system for greater classification power. Allowing trusted researchers to add to the reference data sets would allow for a crowdsourced approach to greater understanding of the world of microbial ecology.

Open sourcing the code allows for more developers to add additional features and continue to maintain the pipeline. In the future, as Python 2 is deprecated, it will become necessary to port the core components of the pipeline into Python 3. Similarly, the visualization tools in R will necessarily also need to be upgraded to keep up with the latest versions as they age out.

The no free lunch theorem demonstrates that the performance of machine learning algorithms is a function of the data set they are being tested against [82]. Thus, it should not be assumed that the algorithms used in this research offer optimal performance upon metagenomic NSV data. Many more machine learning algorithms should be tested using the data that the pipeline generates. It is unknown whether the algorithms presented herein produce the best results for this data. Whether these selected algorithms are in fact optimal is irrelevant as the results can speak to the viability of the existing technique.

Other future options could include moving the entire pipeline onto a cloud provider platform. This would allow for a quick scaling of the available infrastructure such that as new samples need to be processed, the amount of resources could be intelligently governed such that there is sufficient processing power when needed, but excess computing time is not allocated unnecessarily. This is a new field of computing and will likely become a growing part of all bioinformatics work going forward.

Optimizations that might boost performance for KNN and Naïve Bayes Classifiers are readily available in the literature. Modifying the distance metric in KNN to either a Euclidean distance or an adaptive distance metric as presented by Wang [83]. As

an alternative to the Manhattan distance used in both normalized NSV records and raw NSV counts given to the Naïve Bayes Classifiers the distance metrics discussed by Hsu in their article could be explored [84]. Many additional examples exist in the literature that could be considered in future research to increase the speed and performance of the classification and data processing mechanisms presented here.

5.3 Conclusions

A functioning pipeline to classify metagenomic samples as a function of their originating environments without the need to identify individual taxa comprising the metagenome has been presented in this research. A new set of software developed to process metagenomic files and deliver reusable NSV formatted datasets for research purposes is perhaps the major output of this research project. The entire pipeline has more utilities than the munging of metagenomic data; additionally, it can change machine learning algorithms, feature selection mechanisms, and use different kinds of cross validation to show consistency across experiments. The code has been heavily profiled and linted to ensure a standardization when reading and maintaining the codebase. The feasibility of processing metagenomics files into abstract NSVs and using the heavily compressed data to accurately classify complicated bacterial communities and tease out the distinction between environmental classes has been demonstrated in this research. Post-pipeline analysis scripts have been crafted to process the results and log files generated by the pipeline. This pipeline and the other utilities presented are all stored in the repository listed at the beginning of this chapter and the pipeline is available for use in addition to the extra utilities to generate plots and post processing evaluation of log files generated in the processing of the metagenomic data.

After varying the size of k -mers and conducting a large series of experiments, the success of discriminating the class of metagenomic communities by turning the count of k -mers into NSVs has been demonstrated. Accurate predictions can be made using small-sized k -mers; here the smallest tested were 3-mers, in near linear processing time with respect to the total number of fragments. Accuracy of predictions was increased by increasing the size of k -mers used in the experiment. Continued research into the topic of classification of metagenomic samples to environmental class using NSVs should be continued with the use of larger mer sizes, a database to store counts of NSVs, or possibly processing the initial fragments with a GPU.

BIBLIOGRAPHY

- [1] J. A. Eisen, “Environmental shotgun sequencing: Its potential and challenges for studying the hidden world of microbes,” *PLoS Biology*, 2007.
- [2] B. A. Methé and *et al*, “A framework for human microbiome research,” *Nature*, 2012.
- [3] J. A. Gilbert, F. Meyer, D. Antonopoulos, P. Balaji, C. T. Brown, C. T. Brown, N. Desai, J. A. Eisen, D. Evers, D. Field, W. Feng, D. Huson, J. Jansson, R. Knight, J. Knight, E. Kolker, K. Konstantindis, J. Kostka, N. Kyrpides, R. Mackelprang, A. McHardy, C. Quince, J. Raes, A. Sczyrba, A. Shade, and R. Stevens, “Meeting report: The terabase metagenomics workshop and the vision of an earth microbiome project,” *Standards in Genomic Sciences*, 2010.
- [4] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman, “Basic local alignment search tool,” *Journal of molecular biology*, 1990.
- [5] P. J. Turnbaugh, R. E. Ley, M. A. Mahowald, V. Magrini, E. R. Mardis, and J. I. Gordon, “An obesity-associated gut microbiome with increased capacity for energy harvest,” *Nature*, 2006.
- [6] B. Saleh, K. Abe, R. S. Arora, and A. M. Elgammal, “Toward automated discovery of artistic influence,” *CoRR*, vol. abs/1408.3218, 2014. [Online]. Available: <http://arxiv.org/abs/1408.3218>

- [7] M. C. Schatz, B. Langmead, and S. L. Salzberg, “Cloud computing and the dna data race,” *Nat Biotech*, 2010.
- [8] R. M. Kotamarti, M. Hahsler, D. Raiford, M. McGee, and M. H. Dunham, “Analyzing taxonomic classification using extensible markov models,” *Bioinformatics*, vol. 26, no. 18, pp. 2235–2241, 2010. [Online]. Available: <http://bioinformatics.oxfordjournals.org/content/26/18/2235.abstract>
- [9] S. Vinga and J. Almeida, “Alignment-free sequence comparison a review,” *Bioinformatics*, vol. 19, no. 4, pp. 513–523, 2003. [Online]. Available: <http://bioinformatics.oxfordjournals.org/content/19/4/513.abstract>
- [10] K. Song, J. Ren, G. Reinert, M. Deng, M. S. Waterman, and F. Sun, “New developments of alignment-free sequence comparison: measures, statistics and next-generation sequencing,” *Briefings in Bioinformatics*, 2013. [Online]. Available: <http://bib.oxfordjournals.org/content/early/2013/11/25/bib.bbt067.abstract>
- [11] G. Rosen, E. Garbarine, D. Caseiro, R. Polikar, and B. Sokhansanj, “Metagenome fragment classification using n-mer frequency profiles,” *Advances in Bioinformatics*, 2008.
- [12] A. Nagar and M. Hahsler, “Quasialign: Position sensitive p-mer frequency clustering with applications to genomic classification and differentiation,” *Southern Methodist University*, 2012.
- [13] M. E. Rout, T. H. Chrzanowski, T. K. Westlie, T. H. DeLuca, R. M. Callaway, and W. E. Holben, “Bacterial endophytes enhance competition by invasive plants,” *American Journal of Botany*, 2013.

- [14] L. Rasko, H. Sugawara, and M. Shumway, "The sequence read archive," *Nucleic Acids Research*, 2011.
- [15] L. Y. Geer, A. Marchler-Bauer, R. C. Geer, L. Han, J. He, S. He, C. Liu, W. Shi, and S. H. Bryant, "The ncbi biosystems database," *Nucleic Acids Research*, 2010.
- [16] J. G. Caporaso, C. L. Lauber, W. A. Walters, D. Berg-Lyons, J. Huntley, N. Fierer, S. M. Owens, J. Betley, L. Fraser, M. Bauer, N. Gormley, J. A. Gilbert, G. Smith, and R. Knight, "Ultra-high-throughput microbial community analysis on the illumina hiseq and miseq platforms," *ISME J*, 2012. [Online]. Available: <http://dx.doi.org/10.1038/ismej.2012.8>
- [17] Y. Liu, S. Rousseaux, R. Tourdot-Maréchal, M. Sadoudi, R. Gougeon, P. Schmitt-Kopplin, and H. Alexandre, "Wine microbiome, a dynamic world of microbial interactions," *Critical Reviews in Food Science and Nutrition*, 2015.
- [18] V. Kunin, A. Copeland, A. Lapidus, K. Mavromatis, and P. Hugenholtz, "A bioinformaticians guide to metagenomics," *Microbiology and Molecular Biology Reviews*, 2008.
- [19] C.-S. Ku and D. H. Roukos, "From next-generation sequencing to nanopore sequencing technology: paving the way to personalized genomic medicine," *Expert Review of Medical Devices*, 2013.
- [20] S. R. Eddy, "Multiple alignment using hidden markov models," *Intelligent Systems for Molecular Biology*, 1995.
- [21] J. T. Simpson, K. Wong, S. D. Jackman, J. E. Schein, S. J. Jones, and I. Birol, "Abyss: A parallel assembler for short read sequence data," *Genome Research*, 2009.

- [22] T. J. Treangen, S. Koren, D. D. Sommer, B. Liu, I. Astrovskaya, B. Ondov, A. E. Darling, A. M. Phillippy, and M. Pop, “Metamos: a modular and open source metagenomic assembly and analysis pipeline,” *Genome Biology*, 2013.
- [23] P. Hugenholtz, “Exploring prokaryotic diversity in the genomic era,” *Genome Biology*, 2002.
- [24] J. Laserson, V. Jojic, and D. Koller, “Genovo: de novo assembly for metagenomes,” *Journal of Computational Biology*, 2011.
- [25] A. Charuvaka and H. Rangwala, “Evaluation of short read metagenomic assembly,” *BMC Genomics*, 2011.
- [26] M. Pignatelli and A. Moya, “Evaluating the fidelity of de novo short read metagenomic assembly using simulated data,” *PLoS ONE*, 2011.
- [27] S. Garcia-Vallvé, A. Romeu, and J. Palau, “Horizontal gene transfer in bacterial and archaeal complete genomes,” *Genome Research*, vol. 10, no. 11, pp. 1719–1725, 2000.
- [28] T. Nogueira, D. J. Rankin, M. Touchon, F. Taddei, S. P. Brown, and E. P. Rocha, “Horizontal gene transfer of the secretome drives the evolution of bacterial cooperation and virulence,” *Current Biology*, vol. 19, no. 20, pp. 1683–1691, 2009.
- [29] J. C. Wooley, A. Godzik, and I. Friedberg, “A primer on metagenomics,” *PLoS Comput Biol*, 2010.
- [30] X. Hao, R. Jiang, and T. Chen, “Clustering 16s rRNA for OTU prediction: a method of unsupervised bayesian clustering,” *Bioinformatics*, 2011.

- [31] U. Nübel, F. Garcia-Pichel, and G. Muyzer, “Pcr primers to amplify 16s rrna genes from cyanobacteria.” *Applied and environmental microbiology*, vol. 63, no. 8, pp. 3327–3332, 1997.
- [32] A. Engelbrektson, V. Kunin, K. C. Wrighton, N. Zvenigorodsky, F. Chen, H. Ochman, and P. Hugenholtz, “Experimental factors affecting pcr-based estimates of microbial species richness and evenness,” *The ISME journal*, vol. 4, no. 5, pp. 642–647, 2010.
- [33] G. B. Rogers, J. Kozłowska, J. Keeble, K. Metcalfe, M. Fao, S. E. Dowd, A. J. Mason, M. A. McGuckin, and K. D. Bruce, “Functional divergence in gastrointestinal microbiota in physically-separated genetically identical mice,” *Scientific Reports*, 2014.
- [34] D. H. Huson, A. F. Auch, J. Qi, and S. C. Schuster, “Megan analysis of metagenomic data,” *Genome Research*, 2007.
- [35] T. Z. DeSantis, I. Dubosarskiy, S. R. Murray, and G. L. Andersen, “Comprehensive aligned sequence construction for automated design of effective probes (cascade-p) using 16s rdna,” *Bioinformatics*, vol. 19, no. 12, pp. 1461–1468, 2003. [Online]. Available: <http://bioinformatics.oxfordjournals.org/content/19/12/1461.abstract>
- [36] J. R. Cole, Q. Wang, J. A. Fish, B. Chai, D. M. McGarrell, Y. Sun, C. T. Brown, A. Porras-Alfaro, C. R. Kuske, and J. M. Tiedje, “Ribosomal database project: data and tools for high throughput rrna analysis.” *Nucleic acids research*, 2014.
- [37] T. J. Wheeler and S. R. Eddy, “nhmmer: Dna homology search with profile hmms,” *Bioinformatics*, vol. 29, no. 19, pp. 2487–2489, 2013. [Online]. Available: <http://bioinformatics.oxfordjournals.org/content/29/19/2487.abstract>

- [38] J. R. Cole, Q. Wang, E. Cardenas, J. Fish, B. Chai, R. J. Farris, A. Kulam-Syed-Mohideen, D. M. McGarrell, T. Marsh, G. M. Garrity *et al.*, “The ribosomal database project: improved alignments and new tools for rrna analysis,” *Nucleic acids research*, vol. 37, no. suppl 1, pp. D141–D145, 2009.
- [39] J. G. Caporaso, J. Kuczynski, J. Stombaugh, K. Bittinger, F. D. Bushman, E. K. Costello, N. Fierer, A. G. Peña, J. K. Goodrich, J. I. Gordon *et al.*, “Qiime allows analysis of high-throughput community sequencing data,” *Nature methods*, vol. 7, no. 5, pp. 335–336, 2010.
- [40] C.-K. K. Chan, A. L. Hsu, S.-L. Tang, and S. K. Halgamuge, “Using growing self-organising maps to improve the binning process in environmental whole-genome shotgun sequencing,” *BioMed Research International*, vol. 2008, 2007.
- [41] G. E. Sims, S.-R. Jun, G. A. Wu, and S.-H. Kim, “Alignment-free genome comparison with feature frequency profiles (ffp) and optimal resolutions,” *Proceedings of the National Academy of Sciences*, 2009. [Online]. Available: <http://www.ncbi.nlm.nih.gov/pmc/articles/PMC2634796/>
- [42] G. A. W. Gregory E. Sims, Se-Ran Jun and S.-H. Kim, “Whole-genome phylogeny of mammals: Evolutionary information in genic and nongenic regions,” *Proceedings of the National Academy of Sciences*, 2009. [Online]. Available: <http://www.pnas.org/content/106/40/17077.abstract>
- [43] G. E. Sims and S.-H. Kim, “Whole-genome phylogeny of escherichia coli/shigella group by feature frequency profiles (ffps),” *Proceedings of the National Academy of Sciences*, 2011. [Online]. Available: <http://www.ncbi.nlm.nih.gov/pmc/articles/PMC3100984/>

- [44] E. Hüllermeier, “Fuzzy methods in machine learning and data mining: Status and prospects,” *Fuzzy sets and Systems*, vol. 156, no. 3, pp. 387–406, 2005.
- [45] U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, *Advances in knowledge discovery and data mining*. AAAI press Menlo Park, 1996, vol. 21.
- [46] U. Fayyad, G. Piatetsky-Shapiro, and P. Smyth, “From data mining to knowledge discovery in databases,” *AI magazine*, vol. 17, no. 3, p. 37, 1996.
- [47] É. Perthame, C. Friguet, and D. Causeur, “Stability of feature selection in classification issues for high-dimensional correlated data,” *Statistics and Computing*, 2016. [Online]. Available: <http://dx.doi.org/10.1007/s11222-015-9569-2>
- [48] H.-P. Kriegel, P. Kröger, and A. Zimek, “Clustering high-dimensional data: A survey on subspace clustering, pattern-based clustering, and correlation clustering,” *ACM Transactions on Knowledge Discovery from Data (TKDD)*, vol. 3, no. 1, p. 1, 2009.
- [49] J. Hua, W. D. Tembe, and E. R. Dougherty, “Performance of feature-selection methods in the classification of high-dimension data,” *Pattern Recogn*, 2009. [Online]. Available: <http://dx.doi.org/10.1016/j.patcog.2008.08.001>
- [50] A. Kalousis, J. Prados, and M. Hilario, “Stability of feature selection algorithms: a study on high-dimensional spaces,” *Knowledge and Information Systems*, 2006.
- [51] A.-C. Haury, P. Gestraud, and J.-P. Vert, “The influence of feature selection methods on accuracy, stability and interpretability of molecular signatures,” *PLoS ONE*, 2011.
- [52] H. Saghir and D. B. Megherbi, “An efficient comparative machine learning-based metagenomics binning technique via using random forest,” in *Computational In-*

telligence and Virtual Environments for Measurement Systems and Applications (CIVEMSA), 2013 IEEE International Conference on, 2013.

- [53] G. Ditzler, J. C. Morrison, Y. Lan, and G. L. Rosen, “Fizzy: feature subset selection for metagenomics,” *BMC Bioinformatics*, 2015.
- [54] V. N. Vapnik, “An overview of statistical learning theory,” *IEEE transactions on neural networks*, vol. 10, no. 5, pp. 988–999, 1999.
- [55] V. Cherkassky and Y. Ma, “Another look at statistical learning theory and regularization,” *Neural Networks*, vol. 22, no. 7, pp. 958–969, 2009.
- [56] J. M. Buhmann, “Statistical learning theory,” 2015. [Online]. Available: https://www.inf.ethz.ch/personal/alexeygr/slt15/tutorials/SLT_Script_150804.pdf
- [57] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg *et al.*, “Scikit-learn: Machine learning in python,” *Journal of Machine Learning Research*, vol. 12, no. Oct, pp. 2825–2830, 2011.
- [58] I. H. Witten, E. Frank, L. E. Trigg, M. A. Hall, G. Holmes, and S. J. Cunningham, “Weka: Practical machine learning tools and techniques with java implementations,” 1999.
- [59] R. Kohavi, G. John, R. Long, D. Manley, and K. Pflieger, “Mlc++: A machine learning library in c++,” in *Tools with Artificial Intelligence, 1994. Proceedings., Sixth International Conference on*. IEEE, 1994, pp. 740–743.
- [60] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg,

- D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, “TensorFlow: Large-scale machine learning on heterogeneous systems,” 2015, software available from tensorflow.org. [Online]. Available: <http://tensorflow.org/>
- [61] (2017) Amazon machine learning documentation. [Online]. Available: <https://aws.amazon.com/documentation/machine-learning/>
- [62] (2017) Azure machine learning documentation. [Online]. Available: <https://docs.microsoft.com/en-us/azure/machine-learning/>
- [63] H. Bhaskar, D. C. Hoyle, and S. Singh, “Machine learning in bioinformatics: A brief survey and recommendations for practitioners,” *Computers in biology and medicine*, vol. 36, no. 10, pp. 1104–1125, 2006.
- [64] A. C. Tan and D. Gilbert, “An empirical comparison of supervised machine learning techniques in bioinformatics,” in *Proceedings of the First Asia-Pacific bioinformatics conference on Bioinformatics 2003-Volume 19*. Australian Computer Society, Inc., 2003, pp. 219–222.
- [65] D. A. Power, R. A. Watson, E. Szathmáry, R. Mills, S. T. Powers, C. P. Doncaster, and B. Czapp, “What can ecosystems learn? expanding evolutionary ecology with learning theory,” *Biology Direct*, 2015.
- [66] P. E. Larsen, D. Field, and J. A. Gilbert, “Predicting bacterial community assemblages using an artificial neural network approach.” *Nature Methods*, 2012.

- [67] E. Fix and J. L. Hodges Jr, “Discriminatory analysis-nonparametric discrimination: consistency properties,” California Univ Berkeley, Tech. Rep., 1951.
- [68] M. Bayes and M. Price, “An essay towards solving a problem in the doctrine of chances. by the late rev. mr. bayes, frs communicated by mr. price, in a letter to john canton, amfrs,” *Philosophical Transactions (1683-1775)*, pp. 370–418, 1763.
- [69] H. A. Martens and P. Dardenne, “Validation and verification of regression in small data sets,” *Chemometrics and intelligent laboratory systems*, vol. 44, no. 1, pp. 99–121, 1998.
- [70] Q.-S. Xu and Y.-Z. Liang, “Monte carlo cross validation,” *Chemometrics and Intelligent Laboratory Systems*, vol. 56, no. 1, pp. 1–11, 2001.
- [71] R. R. Picard and R. D. Cook, “Cross-validation of regression models,” *Journal of the American Statistical Association*, vol. 79, no. 387, pp. 575–583, 1984.
- [72] G. Rossum, “Python reference manual,” Amsterdam, The Netherlands, The Netherlands, Tech. Rep., 1995.
- [73] R Development Core Team, *R: A Language and Environment for Statistical Computing*, R Foundation for Statistical Computing, Vienna, Austria, 2008, ISBN 3-900051-07-0. [Online]. Available: <http://www.R-project.org>
- [74] E. Jones, T. Oliphant, P. Peterson *et al.*, “SciPy: Open source scientific tools for Python,” 2001–, [Online; accessed 2016-09-22]. [Online]. Available: <http://www.scipy.org/>
- [75] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos,

- D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.
- [76] J. D. Hunter, "Matplotlib: A 2d graphics environment," *Computing In Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007.
- [77] P. D. Marsh, "Microbiology of dental plaque biofilms and their role in oral health and caries," *Dental Clinics of North America*, vol. 54, no. 3, pp. 441–454, 2010.
- [78] P. Belda-Ferre, L. D. Alcaraz, R. Cabrera-Rubio, H. Romero, A. Simón-Soro, M. Pignatelli, and A. Mira, "The oral metagenome in health and disease," *The ISME journal*, vol. 6, no. 1, pp. 46–56, 2012.
- [79] F. Meyer, D. Paarmann, M. D'Souza, R. Olson, E. M. Glass, M. Kubal, T. Paczian, A. Rodriguez, R. Stevens, A. Wilke *et al.*, "The metagenomics rast server—a public resource for the automatic phylogenetic and functional analysis of metagenomes," *BMC bioinformatics*, vol. 9, no. 1, p. 1, 2008.
- [80] W. H. Organization, *Oral health surveys: basic methods*. World Health Organization, 2013.
- [81] D. E. Wood and S. L. Salzberg, "Kraken: ultrafast metagenomic sequence classification using exact alignments," *Genome Biology*, vol. 15, no. 3, 2014. [Online]. Available: <http://www.ncbi.nlm.nih.gov/pmc/articles/PMC4053813/>
- [82] D. H. Wolpert and W. G. Macready, "No free lunch theorems for optimization," *IEEE Transactions of Evolutionary Computation*, 1997.

- [83] J. Wang, P. Neskovic, and L. N. Cooper, “Improving nearest neighbor rule with a simple adaptive distance measure,” *Pattern Recognition Letters*, vol. 28, no. 2, pp. 207–213, 2007.
- [84] C.-N. Hsu, H.-J. Huang, and T.-T. Wong, “Why discretization works for naïve bayesian classifiers,” in *Proceedings of the Seventeenth International Conference on Machine Learning, Morgan Kaufmann, San Francisco, CA*. Citeseer, 2000, pp. 399–406.