Scalable Subset Selection with Filters and Its Applications

A Thesis
Submitted to the Faculty
of
Drexel University
by
Gregory Charles Ditzler
in partial fulfillment of the
requirements for the degree
of
Doctorate of Philosophy in Electrical & Computer Engineering
April 2015
Dedications

To my father.
Acknowledgements

First of all, I would like to thank my advisor Gail Rosen for allowing me the opportunity to work in the Ecological & Evolutionary Signal Processing and Informatics Lab (EESI). She has been a constant source of encouragement throughout my graduate education. Robi Polikar has also served as a fantastic co-advisor, and I am extremely grateful for his support. I would also like to extend my thanks and gratitude to the additional members of my Ph.D. committee, Gavin Brown, Andrew Cohen, John M. Walsh, and Steve Weber. I would extend a debt of gratitude towards Diamantino Caseiro for his mentorship while I was at AT&T Research and for his suggestions after my PhD research proposal.

I would like to thank the greatest group of labmates that a graduate student could ask for: Cricket Reichenberger, Yemin Lan, Steve Essinger, Calvin Morrison, Stephen Woloszynek, and Steve Pastor. Additionally, I would like to thank David Kosliki and Jean-luc Bouchot for some incredibly insightful discussions when they were doing post-docs at Drexel.

Finally, I thank my family for their continued support.
Abstract

Increasingly many applications of machine learning are encountering large data that were almost unimaginable just a few years ago, and hence, many of the current algorithms cannot handle, i.e., do not scale to, today’s extremely large volumes of data. The data are made up of a large set of features describing each observation, and the complexity of the models for making predictions tend to increase not only with the number of observations, but also the number of features. Fortunately, not all of the features that make up the data carry meaningful information about making the predictions. Thus irrelevant features should be filtered from the data prior to building a model. Such a process of removing features to produce a subset is commonly referred to as feature subset selection. In this work, we present two new filter-based feature subset selection algorithms that are scalable to large data sets that address: (i) potentially large & distributed data sets, and (ii) they are capable of scaling to very large feature sets. Our first proposed algorithm, Neyman-Pearson Feature Selection (NPFS), uses a statistical hypothesis test derived from the Neyman-Pearson lemma for determining if a feature is statistically relevant. The proposed approach can be applied as a wrapper to any feature selection algorithm, regardless of the feature selection criteria used, to determine whether a feature belongs in the relevant set. Perhaps more importantly, this procedure efficiently determines the number of relevant features given an initial starting point, and it fits into a computationally attractive MapReduce model. We also describe a sequential learning framework for feature subset selection (SLSS) that scales with both the number of features as well
as the number of observations. SLSS uses bandit algorithms to process features and form a level of importance for each feature. Feature selection is performed independently from the optimization of any classifier to reduce unnecessary complexity. We demonstrate the capabilities of NPFS and SLSS on synthetic and real-world data sets. We also present a new approach for classifier-dependent feature selection that is an online learning algorithm that easily handles large amounts of missing feature values in a data stream.

There are many real-world applications that can benefit from scalable feature subset selection algorithms; one such area is the study of the microbiome (i.e., the study of micro-organisms and their influence on the environments that they inhabit). Feature subset selection algorithms can be used to sift through massive amounts of data collected from the genomic sciences to help microbial ecologists understand the microbes – particularly the micro-organisms that are the best indicators by some phenotype, such as healthy or unhealthy. In this work, we provide insights into data collected from the American Gut Project, and deliver open-source software implementations for feature selection with biological data formats.
Table of Contents

ABSTRACT ........................................................................... iv
LIST OF TABLES .................................................................... ix
LIST OF FIGURES ................................................................. xi
1. Introduction ........................................................................ 1
   1.1 A Simple and Concise Example............................................. 1
   1.2 The Nature of Data.......................................................... 5
      1.2.1 Large Volumes of Data .............................................. 5
      1.2.2 Data in Life Science ................................................. 5
      1.2.3 Volume, Velocity, Variety, Veracity and Value..................... 6
   1.3 Research Questions .......................................................... 7
   1.4 Contributions ................................................................ 8
   1.5 Broader Impacts ............................................................. 9
   1.6 Organization of the Thesis .................................................. 9
2. Feature Selection and Related Works ............................................ 11
   2.1 Introduction .................................................................. 11
      2.1.1 Feature Subset Selection ............................................ 11
      2.1.2 Wrappers ............................................................. 14
      2.1.3 Embedded Methods ................................................. 15
      2.1.4 Filters ................................................................ 17
      2.1.5 Bandits in Subset Selection ......................................... 23
   2.2 Scaling Feature Selection for Large Data .................................. 27
   2.3 Figures of Merit.............................................................. 28
   2.4 Analyzing Data from the Microbiome ...................................... 30
      2.4.1 An Introduction to the Microbiome ................................ 30
      2.4.2 An Introduction to Metagenomics............................... 31
      2.4.3 Answering Questions with Metagenomic Data .................... 32
      2.4.4 Collecting Samples from the Microbiome ....................... 33
      2.4.5 Representing Data in the Microbiome ....................... 34
      2.4.6 Subset Selection and Its Role in the Analysis of the Microbiome 35
   2.5 Summary ..................................................................... 36
3. Subset Selection with Neyman-Pearson Test .................................... 37
   3.1 Introduction .................................................................. 37
   3.2 Neyman-Pearson Subset Selection .......................................... 37
      3.2.1 Algorithm Description & Derivation ......................... 38
      3.2.2 Theoretical Properties on NPFS .................................. 42
      3.2.3 Advantages of NPFS ............................................... 44
   3.3 Experiments.................................................................... 44
      3.3.1 Data Sets and Testing Procedure .................................. 45
      3.3.2 Results on Synthetic Data Sets ............................... 46
      3.3.3 Results on UCI Data Sets ................................. 49
      3.3.4 Optical Character Recognition ........................... 53
7. Conclusions ........................................................................ 123
    7.1 Contributions of this Work.............................................. 123
    7.2 Broader Impacts ............................................................ 124
    7.3 Recommendations for Future Work ............................... 125
Bibliography ........................................................................... 126
## List of Tables

2.1 Monetary cost of collecting sequences. .................................................. 32
3.1 Mathematical Notations for NPFS ......................................................... 38
3.2 Classification errors of NPFS and benchmarks on the UCI data. .......... 52
4.1 Properties of data sets used in our experiments. ................................. 76
4.2 Average percentage of features selected by algorithms that have the ability to approximate the relevant set size. ........................................ 86
4.3 Rank of an algorithm’s performance when averaged across the data sets shown in Table 4.1 (lower is better). ........................................ 87
4.4 Classification error rates on the real-world data sets. ......................... 88
4.5 Classification F1-measures on the real-world data sets. ..................... 89
5.1 Mathematical Notations for OFSE ......................................................... 94
5.2 Properties of the UCI data sets used in OFS experiments. .................. 103
5.3 Parameter selection for OFS. ............................................................. 103
5.4 Cumulative evaluation time measured in seconds for the ensemble algorithms................................................................. 105
5.5 Cumulative mistake rates and rankings for OFS. ............................... 106
5.6 Statistical significance test for comparing OFS algorithms. ................. 107
6.1 List of the top ranking features for omnivores and vegetarians in the 16S data collected from the American Gut Project detected using JMI within Fizzy. ................................................................. 117
6.2 List of the top ranking features for omnivores and vegetarians in the 16S data collected from the American Gut Project detected using Random Forests. ......................................................... 118
6.3 List of the top ranking features for omnivores and vegetarians in the 16S data collected from the American Gut Project detected using NPFS. ...... 119

6.4 List of the top five ranked Pfams as selected by the Fizzy’s Mutual Information Maximization (MIM) applied to MetaHit ......................... 120
List of Figures

1.1 Binary prediction problem with two variables .............................................. 4

2.1 Example of a feature selection problem. ....................................................... 13

2.2 High-level pseudo code for a wrapper-based feature selection algorithm ... 15

2.3 Entropy of Bernoulli random variables (measured in nats). ....................... 20

2.4 Pseudo code for a filter-based subset selection approach........................... 22

2.5 Pseudo code for the UCB1 Policy [1]. ....................................................... 25

3.1 Pseudo code for NPFS ........................................................................... 43

3.2 Results of the Neyman-Pearson hypothesis test applied to the synthetic uniform data set. ..................................................................................... 47

3.3 Number of features selected by the Neyman-Pearson detector for varying levels of $k$ ......................................................................................... 48

3.4 Variation in the Neyman-Pearson’s test for the value of $k^*$ given that $k$ may have been selected too small ......................................................... 50

3.5 NPFS evaluated on optical character recognition data............................. 51

3.6 Effect of the $\alpha$ and $\beta$ parameters on the critical threshold in NPFS .... 55

3.7 NPFS evaluated with variations in $n$ and $\beta$. ....................................... 56

3.8 Jaccard index for NPFS as a function of the number of bootstraps with early stopping. .............................................................. 58

3.9 The early stopping criteria, $\frac{1}{K}\|\hat{p}_t - \hat{p}_{t-1}\|_1$, for different size feature sets. .. 59

3.10 NPFS evaluated on a data set with 58M+ observations. ......................... 59

3.11 Stability of NPFS on a massive amount of data using the Jaccard and Lustgarten indices as figures of merit. ................................................. 61

4.1 Pseudo code for the SLSS-UCB1 ......................................................... 64
4.2 Pseudo code for the SLSS-Exp3 ................................................ 66
4.3 Pseudo code for the SLSS-Thompson ............................................. 66
4.4 Feature importance detection algorithm for SLSS................................. 68
4.5 Two simulations of the feature importance detection algorithm in Figure 4.4 applied to synthetic rewards from SLSS. ............................... 70
4.6 SLSS implementation with the bag-of-little bootstraps (BLB), which results in a confidence interval around the estimate of SLSS’s feature importance estimation. ............................................................... 71
4.7 Evaluation of the theoretical and empirical regret for SLSS-UCB1. .......... 74
4.8 Cumulative regret of SLSS-\(\epsilon\), SLSS-UCB1, and SLSS-Thompson evaluated on a synthetic data set. ............................................................... 79
4.9 Extension of the average cumulative regret experiment from Figure 4.8 for SLSS-\{\epsilon, UCB1, Thompson\} evaluated in varying \(k^*\) when \(T = 1000\). The solid lines represent \(L = 25\) and the dashed lines represent \(L = 40\). ........ 79
4.10 Visualization of SLSS weights as a function of the learning round.......... 81
4.11 The Lustgarten index and SLSS feature set size as a function of the number of features in a synthetic data set. ................................. 83
4.12 Performance, measured in MSE, of SLSS and SLSS-BLB evaluated on a synthetic data set ............................................................... 85
4.13 SLSS learned feature importance distribution for two data sets collected from microbial ecology. ............................................................... 90
5.1 Online algorithm for learning a linear function with parameters \(w\) ......... 93
5.2 Pseudo code for OFS with Partial Inputs [2] ............................................. 96
5.3 Pseudo code for truncating a parameter vector in OFS [2] ................... 97
5.4 Pseudo code for online bagging feature selectors .................................................. 98
5.5 Pseudo code for online boosting feature selectors .................................................. 99
5.6 Simulations of a Poisson distribution for truncation randomization. ....... 101
5.7 Cumulative mistakes of OFS on UCI data sets. ......................... 105

6.1 Overview of feature selection using data collected from the microbiome. ... 109
6.2 Overview of the Fizzy software tool. ........................................... 111
6.3 Ranked differences between the relative abundances between omnivores and vegetarians as determined from JMI. ................................. 116
6.4 Evaluation times of several feature selection algorithms on biological data. 121
1. Introduction

Supervised pattern recognition and machine learning is the process of taking in raw data and making a decision based on the category or class of the pattern [3], and such a definition focuses itself on the task of prediction or classification. For example, given values from a medical test, is a patient at risk of heart disease? In many tasks of prediction and knowledge discovery, it is commonplace to learn such a function that can make the prediction from a collection observations. Each of observation is made up of this collection of features (i.e., variables), where the number of features could be quite large, as could the number of observations.

Typically, learning such a function has a complexity that scales with the number of observations as well as the number of features in the data set. Furthermore, we often need to consider this complexity before using any machine learning algorithm in practice. Reducing the number of features to a set that contains only the most relevant features can greatly reduce the complexity of the learning algorithm, and allow the algorithm to solely focus on learning from valuable data; hence, not learn the irrelevance of variables.

In this chapter we describe the nature of the problem of feature subset selection by motivating the problem with an intuitive example. We then describe the research problems that this thesis addresses, as well as how the proposed algorithms address them.

1.1 A Simple and Concise Example

Let us consider a recently proposed data science challenge by Kaggle Inc. that was funded by a $3 million prize from the Heritage Provider Network\(^1\) [4]. Approximately

\(^1\)http://www.heritageprovidernetwork.com/
71 million individuals are admitted into hospitals each year, and while many of the patients required a visit to the hospital, there is still approximately $31 billion spent annually on unnecessary visits. Now imagine that we are tasked with predicting if a patient is going to be admitted into the hospital over the next year given information in their medical record. There are an enormous amount of medical records in the US, and having a person manually sift through them on a computer is not a viable option. However, consider that we choose to develop an automated system for making this prediction, so that given a patients’ medical record, an algorithm returns whether the patient is likely to return to the hospital over the next year. Consider that we are provided medical records – or observations – that need to be classified into a category (i.e., visited the hospital or did not visit the hospital). In the context of machine learning, we refer this scenario a supervised learning problem, where we have data (records or observations), and corresponding classes (categories, outcomes or groups). The goal is to use information provided by the features in the medical record to predict the class.

The medical records consist of information about the individual as well as any results from medical tests. Each piece of information in the record can be viewed as a variable, or feature, of interest for predicting the outcome of whether that patient will visit the hospital within the next year. In fact, features are sometimes referred to as predictors in the statistics community. For example, sex, hair color, blood pressure, and weight can all be viewed as features. Typically there is a very large set of features, because a medical record can be represented by millions of features if the record were to contain personal information, images and medical measurements. Fortunately not all of the features are meaningful for making the prediction. Features such as hair color are unlikely to be meaningful for making such a prediction, where blood pressure may be better. Thus if we have a computer program that can make
the prediction, the hair color of a patient is not needed to make the prediction. Recall that many algorithms' complexity that make such predictions scale with the number of variables (also known as the dimensionality of the data). The algorithm then can be made less complex by using only the features that are informative. Not only does the complexity of the algorithm change with the size of the data and feature set, but also the monetary cost to implement the model. For example, if certain medical tests are not informative for making the prediction, then they do not need to be collected/measured. Not requiring features to be collected can impact the cost from the patients' perspective because they do no need to pay for a test. This process of selecting such variables is referred to as feature subset selection.

There are two key points to take note of in this running example. First, there are an enormous number of medical records. In fact, so many that the data may not be stored on the same computer. Many modern computing resources share their data over a network, or use a distributed file system (e.g., Hadoop\(^2\)). Thus, we face our first computational hurdle of addressing if a patient is likely to make a hospital visit. Loading the entire data into memory to build a model can be inefficient or simply infeasible.

Second, we need to develop a framework to filter out the irrelevant features before we build a model. We can view features as either being relevant, redundant or irrelevant. Relevant features are those that are meaningful for predicting the outcome by some predefined measure (discussed in Chapter 2). These features may be medical tests or something that is indicative of the outcome. For example, a feature indicating a serious chronic disease may be very relevant in determining if someone will return to the hospital. Redundant features can be those that carry similar information as another feature. Consider the scenario that if two medical tests provide the same

\(^2\)http://hadoop.apache.org/
outcome (consistently), then both tests do not need to be performed. Irrelevant fea-
tures are those that do not carry any information on predicting the outcome and can 
be left out of the model. Though it is possible that there are two (or more) features 
that are not informative on their own, but jointly are informative. As an example, 
consider Figure 1.1, which shows a binary prediction problem with two variables \( x_1 \) 
and \( x_2 \). Examining \( x_1 \) or \( x_2 \) individually is not very useful. In fact, you may be better 
off flipping a coin to determine the outcome; however, jointly considering \( x_1 \) and \( x_2 \) 
allows us to observe a rule that can predict the class quite reliably. Conceptually, 
these ideas of feature importance are easy to understand; however, the analysis be-
comes more complicated when examined on a feature set on a massive scale for both 
theoretical and algorithmic concerns.

In this example, we have highlighted some basic principles of machine learning 
and feature selection. These concepts are reviewed with more mathematical rigor in 
Chapter 2.
1.2 The Nature of Data

1.2.1 Large Volumes of Data

The nature of the sources generating data today are producing observations and features on a massive scale, such as Giga-, Tera- and even Peta-Bytes of data in some cases. The previous example can be viewed as a very large data set if we look at all records within a healthcare provider’s network. Another example would be CERN’s Large Hadron Collider, which has generated over 100 petabytes of data since its inception. Not only are scientific and experimental studies generating massive data [5–7], but social media companies, such as Yahoo!, Google, Facebook, and YouTube, are generating data a rapid rate that seemed almost unimaginable just several years ago. Data of these proportions are quickly becoming the norm in data analysis to the point where big data may simply be known as data in the not-so-distant future. Therefore, algorithms that can scale to large volumes of data, while still being able to extract the most valuable information, are of utmost importance in machine learning and data mining.

1.2.2 Data in Life Science

Life science is one such area that is known to generate a tremendous amount of data [8, 9] (one such area where there has been a massive influx of interest is the study of the bacterial communities living in the environment and human habitats). This recent interest in the field is being propelled by: (i) reductions in costs associated with sequencing, (ii) improvements to sequencing technology, and (iii) the revelation that bacteria plays an extremely vital role in human health. There are typically thousands to tens of thousands of bacteria detected in a single environmental sample, and microbial ecologists need to have efficient computational methods to uncover differences between phenotypes, such as disease, pH levels, or the biome from
which the sample was collected. In particular, microbial ecologists are interested in finding the bacteria that best differentiate and characterize these phenotypes; hence, feature/variable/subset selection methods can provide useful insights to the experimental microbial ecologist. It is very important that any type of subset selection or extraction that produces a condensed set of variables maintains a physical interpretation (i.e., bacteria, protein family, etc.). Therefore, multi-dimensional scaling and manifold learning methods do not meet the microbial ecologists’ needs because the resulting features cannot be physically interpreted. Unfortunately, the difficulty in applying a feature selection method to metagenomics data is: (i) what objective function for feature selection should be used (should the user choose freely?), and (ii) out of all bacteria detected in the sample, how many important bacteria exist in the collection of sample? The current state of knowledge in feature subset selection contains methods that can typically satisfy (i), but not (ii), or vice versa. One component of this research is developing an algorithm that can satisfy (i) and (ii), and do so in a computationally efficient manner.

1.2.3 Volume, Velocity, Variety, Veracity and Value

Regardless of the source of the data, if it is large, there are typically a few properties of the data. While the number of properties can be debated, we see 5V’s associated with big data: volume, velocity, variety, veracity and value. Big data has traditionally focused volume of the “big” data in term of the number of data records; however, there is a lack of studies on problem of big dimensionality [10]. One of the problems that must be addressed with supervised learning is feature discovery, which does not focus on building a classifier, rather an examination of the importance of variables in a data set. There is certainly no shortage of feature subset selection algorithms that have been developed over the last 20 years and many of them have
been shown to work well – even with data sets of moderate proportions. However, the difficulty lies in applying such methods to very large data sets when: (i) the data cannot be loaded into memory, or (ii) the complexity of the subset selection algorithm simply cannot scale to the size of the data set.

1.3 Research Questions

Most feature selection approaches require to have the number of relevant feature specified in advance [2, 11, 12], which is often a nearly impossible task given any reasonably sized data set. In fact, the set size is typically chosen arbitrarily, or via a grid search. While there are approaches that can provide inference on the feature set size, these approaches require choosing their own free parameters [13], or are tied to a specific classifier [14]. Furthermore, some classifier-independent feature selection approaches can provide inference on the relevant set size, though they fix the objective function for performing the selection. The unfortunate situation arises if users want to: (i) perform classifier-independent subset selection, (ii) have an algorithm detect the relevant set size, and (iii) allow for the choice of any arbitrary feature selection objective function. However, most feature selection research provides users the ability to choose two of the three properties.

The next question of interest is that filter-based subset selection algorithms are typically not used for large data sets out of concerns for scalability. An important contribution would be the development of a framework that can provide properties (i), (ii), and (iii) scaling to large data sets. Furthermore, any such framework should have justifiable theoretical properties.
1.4 Contributions

The primary goal of this research is to develop a framework that: (i) identifies variable relevancy using a decision-theoretic implementation; (ii) sequentially learns subsets of relevant features by solving smaller subset selection problems rather than considering the entire high-dimensional space, (iii) have theoretically justifiable properties derived as a result of this research, (iv) provides a realizable implementation for large scale data, which includes not only high-dimensionality, but also a large number of observations, (v) can be readily implemented in a distributed / parallel computing environment, and (vi) can function in an incremental, or life-long, learning setting, which can seamlessly and continuously allow for processing and learning from new data.

Against this background and motivation, the specific contributions of this thesis include:

- **Neyman-Pearson based feature selection (NPFS) [15, 16]:** A scalable framework for feature subset selection that: (i) is classifier-independent, (ii) provides straightforward model to cope with large data, (iii) has theoretically justifiable properties, and (iv) detects the relevant feature set size. NPFS is highly parallelizable.

- **Sequential Learning Subset Selection (SLSS):** An algorithmic feature subset selection framework that is capable of processing massive data sets using an underlying sequential learning algorithm, which can weigh the relative importance of features in the data set. SLSS can also work with partial information, hence, making it possible to address missing/incomplete data.

- **Online Feature Selection with Bagging & Boosting [17]:** An embedded online ensemble-based feature selection algorithm is presented using bagging and boosting. Our approach can scale to very large data streams and we show
that the complexity of predicting with an ensemble model is equivalent to the complexity of the single model; however, experiments show that the ensemble models yields a much lower error rate.

- **Subset Selection as a Tool for Comparative Metagenomics** [18–21]: We developed feature subset selection tools for biological data, and have applied them to problems in comparative metagenomics. We present novel results that reveal the differentiating taxa between gender and diet.

Implementations of the novel algorithms developed throughout this thesis have been made available to the public.

1.5 Broader Impacts

The impact of this research extends far beyond that of an algorithm and its theoretical properties as the deliverable. There are certainly no shortage of real-world problems that can benefit from a feature subset selection framework that is capable of processing large volumes of data (e.g., health care [4,5], astronomical time series data [6]). The National Research Council has recently laid out the challenges that are being faced in the current big data era [7]. *Feature discovery* is one of the research areas that is of great importance to supervised learning with big/massive data because of its wide applicability to medical care prediction.

1.6 Organization of the Thesis

This thesis is organized as follows:

- Chapter 2 summarizes literature related to feature subset selection as well as other other mathematical concepts, which are important for understanding the contributions. We also discuss related works in metagenomics.

• Chapters 3 and 4 present a formal analysis of NPFS and SLSS, respectively. We provide an experimental analysis in each chapter by evaluating the approaches on synthetic and real-world data sets.

• Chapter 5 presents an online learning ensemble that builds linear feature selection models and combines them with a convex combination. The ensemble model is of the same sparsity as the linear model. We evaluate our approach on several real-world and synthetic data sets.

• Chapter 6 presents experiments of feature subset selection for providing microbial ecologists with tools for comparative metagenomics.

• Chapter 7 provides a discussion, conclusions, and directions for future research.
2. Feature Selection and Related Works

2.1 Introduction

Feature subset selection is a broad research topic that has attracted contributors from fields such as machine learning, mathematics and applied research. In this chapter, we present some works – both recent and foundational – that are related to the work being performed in this thesis. Feature selection is an area in machine learning that has been of interest for the last 20 years; however, our work focuses on modernizing feature selection larger data sets that fit into a convenient computational model.

2.1.1 Feature Subset Selection

Feature subset selection in an important component to the supervised learning process and it enables an algorithm to efficiently learn from data. For example, consider a neural network that is trained on a data set with $K$ features and only a fraction of them are relevant, though we do not know in advance the ones that are relevant. Using all $K$ features during training would require that the neural network learn the features that are irrelevant while also learning the mapping from the features to the labels. Furthermore the complexity of training the neural network is larger using all of the features, whereas a network using just a fraction of features (i.e., those that are relevant) would lead to lower complexity network. The selection of the most informative variables is not only important to reduce the feature set to one that only contains relevant features (i.e., provide a classifier/regressor with only “good” features)\(^1\) [22, 23], but also one that can lead to tighter bounds on uniform

\(^1\)Garbage in, garbage out!
convergence (i.e., the VC-dimension of the hypothesis class $\mathcal{H}$ typically increases with the dimensionality) [24–26], and improve the generalization performance [27]. Not only can subset selection provide nice theoretical properties during learning, but, perhaps most importantly, it has shown to be extremely helpful in areas that have high dimensional data, such as genomics [28], micro-array analysis [29], cancer classification [30], robotics [31], and many other fields.

The setting for feature selection can be formulated as follows: we have a labeled data set $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^M$ of $M$ observations, where $x_i \in \mathbb{R}^K$ is the observation, and $y_i \in \Omega$ is the label. The outcome space $\Omega$ is assumed to be discrete (e.g., $\Omega = \{\text{healthy}, \text{disease}\}$), and that $\mathbb{R}^K$ is a high dimensional space and many of the variables in the feature space are assumed to contain little or no information about the space $\Omega$. As discussed in Chapter 1, it is possible for some variables to be redundant with other ones. In some situations, we may seek to eliminate the redundant variables, and particularly if classification is the end goal. The objective is then to find a feature subset of size $k < K$ that provides the most information about the space $\Omega$. Furthermore, in situations where features are relevant or irrelevant, we assume there exists a subset of size $k^*$ that is the size of the true relevant set. Figure 2.1 provides an example of strong, weak and irrelevant features for a very simple problem. The labels are determined by $y = \text{sign}(\theta^T x)$, where $\theta$ is a vector containing the weights tied to each feature. In this example, $\theta \in [0, 1]^K$, where a weight close to zero means the feature carries no information and a weight close to one means the feature is strongly relevant.

Our objective is to examine the subset of $k$ features returned by a feature selection algorithm ($\mathcal{A}$), and provide some inference – post feature subset selection – on the value of $k^*$. We acknowledge that the notion of having a fixed number of $k^*$ becomes ambiguous when there are varying levels of relevance through linear or non-linear
Figure 2.1: Features are weighted with a value show in the $y$-axis. The discriminate function is given by $y = \theta^T x$. Features indexed from -10 to -1 are strongly relevant, -1 to 1 are weakly relevant, and 2 to 10 are irrelevant.

relationships. Therefore, we adopt the following definitions of relevance (or lack thereof) presented by Wu et al. [32].

**Strong Relevance** A feature $X_i$ is strongly relevant for an outcome $Y$ if $\forall \mathcal{X}_i \subset \mathcal{X}\setminus X_i$

$$\mathbb{P}(Y|\mathcal{X}_i) \neq \mathbb{P}(Y|\mathcal{X}_i, X_i)$$  \hfill (2.1)

**Weak relevance** A feature $X_i$ is weakly relevant for an outcome $Y$ if $X_i$ is not strongly relevant and $\exists \mathcal{X}_i \subset \mathcal{X}\setminus X_i$

$$\mathbb{P}(Y|\mathcal{X}_i) \neq \mathbb{P}(Y|\mathcal{X}_i, X_i)$$  \hfill (2.2)

where $\mathcal{X} : \{X_1, \ldots, X_K\}$ is a set of features represented as random variables.

**Irrelevance** A feature $X_i$ is irrelevant to an outcome $Y$ if $\forall \mathcal{X}_i \subset \mathcal{X}\setminus X_i$

$$\mathbb{P}(Y|\mathcal{X}_i) = \mathbb{P}(Y|\mathcal{X}_i, X_i)$$  \hfill (2.3)

However, guaranteeing that equality in (2.3) holds may be too stringent, particu-
larly if there is noise. Therefore, we can relax definition irrelevance to be

\[ |\mathbb{P}(Y|\mathcal{X}_{-i}) - \mathbb{P}(Y|\mathcal{X}_{-i}, X_i)| \leq \delta \]  

(2.4)

for \( \delta \) being some very small number, which implies that there can exist some difference in the posterior if the feature is omitted; however, the difference is rather minuscule.

Feature subset selection algorithms typically fall into one of three broad categories: \textit{wrapper}-based methods, \textit{embedded} methods, and \textit{filter} methods. Though there are hybrid approaches, our focus shall just be on the aforementioned three, which we now describe in more detail.

\subsection{2.1.2 Wrappers}

Wrapper approaches are a classifier-dependent form of subset selection, which forces the user to be limited to a specific classifier. Kohavi & John demonstrated that feature selection wrappers can lead to classifiers that produce a small generalization loss \([33,34]\); however, the Achilles’ heel of these methods is the computational complexity, as the selected features are classifier dependent and a new classifier needs to trained for each selection. A high-level pseudo code can be found in Figure 2.2. The generic algorithm in the figure requires access to a classifier (\textsc{Learn}), which could be a support vector machine \([35,36]\), a data set \( \mathcal{D} \), and a feature set of indices \( \mathcal{F} \subset [K] \), where \(|\mathcal{F}| = k \). The wrapper works in rounds by building a classifier on the data using only the features in \( \mathcal{F} \). Then an error is measured. Based on the error, the feature set is adapted to include a different set of indices in \( \mathcal{F} \). The set \( \mathcal{F} \) can be adapted using genetic algorithms \([37]\), or another suitable evolutionary approach.

The key observation to make for the wrapper is that the score computed from a cross validation error of a predefined classifier. Training and testing the classifier on each round is extremely computationally expensive, which is where the Achilles’ heel
Input: Classifier Learn, data set $\mathcal{D} := \{(x_i, y_i)\}_{i=1}^n$

Initialize: $\mathcal{F}$ is a random set of indices of size $k$.

1: while convergence not met do
2: $h = \text{Learn}(\mathcal{D}, \mathcal{F})$
3: Measure error of $h$
4: Adapt $\mathcal{F}$ based on the error
5: end while

Figure 2.2: High-level pseudo code for a wrapper-based feature selection algorithm

of the wrapper approaches lie. This score can viewed as a measure of quality or value of the set $\mathcal{F}$.

Recent efforts by Bolón-Canedo et al. have attempted to implement a distributed wrapper to improve the feature selection wrapper’s run time [38], but their approach still needs to generate classifiers, which still leads to a computationally complex solution, and optimality of the wrapper cannot be guaranteed [39].

As our focus is primarily on classifier-independent subset selection, and in particular those that can be scaled to large data sets, wrapper-based feature selection algorithms are not of interest.

2.1.3 Embedded Methods

Embedded methods jointly optimize the classifier’s parameters and feature selector simultaneously [22,23]. The parameters are classifier specific and need to be learned. The mechanism that implements the feature selection is dependent on the classifier being used. It may sound that embedded and wrapper methods are the same, but there is a clear difference. The difference between embedded- and wrapper-based approaches, is that the feature selection for an embedded method is controlled by the objective function being optimized, whereas a wrapper is simply a search procedure
for the best features given a classifier. The optimization problem for an embedded approach could be generically written as:

$$\theta^* = \arg \min_{\theta \in \Theta, F \subseteq \mathcal{X}} \text{Loss}(\mathcal{D}, F', \theta) + \text{Penalty}(\theta)$$  \hspace{1cm} (2.5)$$

where $\text{Loss}(\mathcal{D}, F, \theta)$ is the loss of the prediction model with parameters $\theta$ using a feature subset $F$ and $\text{Penalty}(\theta)$ is a term to penalize the complexity of the model. While both embedded and wrapper methods optimize a classifiers, embedded methods tend to be much more desirable due to lower complexity.

The least absolute shrinkage and selection operator (lasso) is a popular embedded feature selection approach [13]. The central concept behind lasso is the minimize the $l_2$-norm of the difference between the predictors $y$ and the prediction $\hat{y} = X^T \theta$ (i.e., measure the error), and add a penalization $\theta$’s $l_1$-norm (i.e., penalize the parameters). The objective function for lasso is formally given by:

$$\theta^* = \arg \min_{\theta \in \Theta} \frac{1}{2M} \|y - X^T \theta\|^2_2 + \lambda_1 \|\theta\|_1$$  \hspace{1cm} (2.6)$$

where $\lambda_1 > 0$, $\|\cdot\|_2$ and $\|\cdot\|_1$ represent the $l_2$- and $l_1$-norm, respectively. The solution of (2.6) tends to be sparse because the of the $l_1$-norm minimization\(^2\). Since lasso’s prediction is given by $y = \theta^T x$, any element where $\theta_j = 0$ would imply that the feature has no influence in the prediction, hence, performing feature selection.

Unfortunately, lasso is not always well suited for problems where there could be data with highly correlated features. Elastic-net was developed to avoid this problem.

\(^2\)The solution is not guaranteed to be sparse, however, setting values to zero in $\theta$ have a magnitude smaller than $\delta > 0$ would force sparsity
(see [40]) by changing the objective to

$$\theta^* = \arg \min_{\theta \in \Theta} \frac{1}{2M} \| y - X^T \theta \|_2^2 + \lambda_1 \| \theta \|_1 + \frac{\lambda_2}{2} \| \theta \|_2^2$$  \hspace{1cm} (2.7)

which is commonly implemented in most software packages as:

$$\theta^* = \arg \min_{\theta \in \Theta} \frac{1}{2M} \| y - X^T \theta \|_2^2 + \alpha \lambda_1 \| \theta \|_1 + (1 - \alpha) \frac{\lambda_2}{2} \| \theta \|_2^2$$  \hspace{1cm} (2.8)

where $\alpha \in [0, 1]$ controls the weight attached to the $l_1$- and $l_2$-norms. If $\alpha = 1$, the objective function is the same as lasso. If $\alpha = 0$, the optimization problem reduces to ridge regression [41] (also known as Tikhonov regularization [42]), otherwise if $\alpha \in (0, 1)$ we have the elastic net\(^3\).

A recent work by Yan et al. began to examine embedded methods for massive data sets [44]. Their approach is an adaptive feature scaling algorithm that, while scaling to very large data sets, limits itself to binary classification problems. Variations of the support vector machine (SVM) have been used in embedded feature selection [45]. However, even with methods such as elastic-net, the end user is limited tuning the features for the prediction model. Thus the set of feature selected by both embedded and wrapper methods may not generalize to other classifiers. Therefore, classifier-independent subset selection is an approach to get aid improving generalization with various classifiers.

### 2.1.4 Filters

As discussed with both wrapper and embedded methods, the scoring function for a feature set is derived from a classifier loss, whether its a 1-0 loss or, in the case of lasso,

\(^3\)The penalization of elastic-nets is sometimes written as $\alpha \| \theta \|_1 + (1 - \alpha) \| \theta \|_2^2$ where $\alpha = \lambda_2/(\lambda_2 + \lambda_1)$ to simplify the problem. The problem of choosing an appropriate value for $\alpha$ is still not an easy task, and must be tuned (see [43] for an example of elastic-net and lasso applied to metagenomic data).
a mean-squared error. Filter methods for subset selection decouple the objective of classification and feature selection by using a scoring function that is not based on a loss. The motivation in decoupling the classifier loss from the feature selection is that the generalization of the filter’s feature set should perform better with different classifiers.

A simple scoring function is the \( \chi^2 \) test, which examines the independence of two events. If the test is rejected for a feature, it indicates that there is dependence between the feature and the class label. \( \chi^2 \) feature selection has been shown to be inaccurate, due to the one degree of freedom, and the test commonly selects more features than are actually relevant [46]. Other approaches, such as Relief, focus on attempting to capture measures of inter- and intra-class dependencies to select features [47].

One of the more popular classes of filter-based subset selection algorithms use information theory and greedy search algorithms [48, 49]. A general form of the information-theoretic methods have objective functions of the form:

\[
J(X, \mathcal{F}) = \text{Relevance}(X, Y) - \text{Redundancy}(X, \mathcal{F})
\] (2.9)

where Relevance\((X, Y)\) measures the relevancy between a feature \(X\) and the class \(Y\), and Redundancy\((X, \mathcal{F})\) measures the redundancy of a variable \(X\) with a collection of other features \(\mathcal{F} \subset \mathcal{X}\). However, before discussing information-theoretic subset selection algorithms, let us review information theory and how it can be used for form the terms in (2.9).

A Review of Information Theory

Information theory is a mathematical theory developed by Claude Shannon to quantify information [50], and the field plays an important role for developing quan-
tities for measuring the relevancy and redundancy of features. While the field of
information theory is quite deep, we shall only discuss the quantities that are of most
importance for discussing feature selection. At the core of information theory lies the
fundamental quantity known as *entropy*, which is formally defined by:

\[ H(X) = - \sum_{x \in \mathcal{X}} p_X(x) \log p_X(x) \]  

(2.10)

The entropy of a random variable \( X \) with support \( \mathcal{X} \) is the measure of uncer-
tainty in the random variable. The choice of the logarithm’s base define the units
of measurement (e.g., \( \log_2 \) is measured in *bits* and \( \log \) is measured in *nats*). Given
the definition of entropy, it is expected that a large entropy means there is a large
amount of uncertainty in a random variable. For example, flipping a biased coin the
probability of heads being 0.98 would be a low entropy random variable because the
outcome is not too uncertain. We know that heads will appear the vast majority
of the coin flips. A fair coin, on the other hand, is a high entropy random variable
because the outcome of the coin flip is the most uncertain. The coin flipping exam-
ple can be viewed as the entropy of a Bernoulli random variable with probability of
success \( p \), which is shown in Figure 2.3.

Another important quantity is the conditional entropy \( H(X|Y) \), which is a mea-
sure of uncertainty in a random variable \( X \) after \( Y \) is known. This quantity is defined
by:

\[ H(X|Y) = - \sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} p_{X,Y}(x,y) \log p_{X|Y}(x|y) \]  

(2.11)

Note that through manipulations of the probabilities, we may write (2.11) differently.
It is important to see that \( H(X|Y) \leq H(X) \), and that both quantities must be
positive. The next major quantity is mutual information, which is defined by:

$$ I(X; Y) = H(X) - H(X|Y) $$  \hspace{1cm} (2.12) 

The mutual information for random variables $X$ and $Y$ is a measure variables' mutual dependence. It measures the amount of uncertainty left after the other random variable is known, thus $I(X; Y) = I(Y; X)$. The final quantity we'll defined is the conditional mutual mutual information.

$$ I(X; Y|Z) = \sum_{z \in Z} p_Z(z)I(X; Y|Z = z) $$  \hspace{1cm} (2.13) 

These few quantities can be used to form a vast number of objective functions that can be used to implement (2.9).
Approaches for Information-Theoretic Filters

Brown et al. state, the filter assumption must be met for these approaches to be effective, i.e., the optimization of the feature set and classifier can be performed in two stages: (i) optimize the feature set, and (ii) use the result from (i) to optimize the classifier. Greedy algorithms are often selected for maximizing the scoring function for filter methods.

Perhaps the simplest objective function to satisfy (2.9) is $J(X) = I(X; Y)$, which drops the redundancy term from the function. This procedure is known as mutual information maximization (MIM) [51]. Feature selection with MIM is a simple as choosing the $k$ features with the largest mutual information.

Peng et al. introduced the minimum-redundancy maximum-relevancy (mRMR) approach to feature selection [52], which has been widely used and benchmarked against. Features are added into a relevant set $F \subset X$ that maximize

$$J(X_j) = I(X_j; Y) - \frac{1}{|F|} \sum_{X \in F} I(X_j; X)$$

(2.14)

where $X$ is a feature in the relevant set $F$. (2.14) shows how the objective function in (2.9) can be designed to control the feature relevancy versus redundancy in the selection process. The first term in (2.14) is simply the MI between the feature and label variable (i.e., how relevant is the feature)? The second term is a penalization for the feature under test having mutual information with features that are already in the relevant feature set $F$ (i.e., how redundant is $X_k$ with the set $F$)?

An objective function such as mRMR’s can be maximized using the greedy algorithm shown in Figure 2.4. The algorithm begins by considering a feature set $X$ with an objective function $J$. The relevant feature set is initialized to the empty set. Then the approach evaluates all the features in $X$ with the objective function $J$, and
Input: Collection of features $\mathcal{X} := \{X_i : i \in [K]\}$, scoring function $J$, and phenotype variables $Y$.

Initialize: $\mathcal{F} = \emptyset$

while $|\mathcal{F}| < k$ do

- Compute the next best feature
  \[ X^* = \arg \max_{X' \in \mathcal{X}} J(X', Y, \mathcal{F}) \] (2.15)

- $\mathcal{F} \leftarrow \mathcal{F} \cup X^*$
- $\mathcal{X} \leftarrow \mathcal{X} \setminus X^*$

end while

Figure 2.4: Pseudo code for search selecting features using a greedy algorithm that attempts to maximize $J$.

chooses the feature that maximizes the objective function. Then add the feature to the relevant set and remove the feature from $\mathcal{X}$. This procedure is repeated $k$ times. Thus, the greedy algorithm in Figure 2.4 can be used to maximizes a very broad class of objective functions of the form in (2.9).

Recall that mRMR’s redundancy term did not use any conditional MI, where the condition would be on the labels (i.e., $I(X_i; X_j | Y)$). Yang and Moody presented the joint mutual information maximization (JMI) objective function for feature subset selection. Unlike mRMR, JMI uses a conditional MI term in the redundancy term, which is given in (2.16).

\[ J_{\text{JMI}}(X_i) = I(X_i; Y) - \frac{1}{|\mathcal{F}|} \sum_{X_j \in \mathcal{F}} (I(X_i; X_j) - I(X_i; X_j | Y)) \] (2.16)

Notice that the sign of the $I(X_i; X_j | Y)$ redundancy term is negative, which with the sign in front of the summing being negative will give $I(X_i; X_j | Y)$ an overall positive contribution to the objective function for JMI. Hence, *not all redundancy is bad*! Such
features that have a large $I(X_i; X_j|Y)$ help reinforce the importance of the joint set of features. There have been many such information-theoretic based approaches that have been gaining popularity [11, 53–55],

There exists a class of functions that are known as submodular [56,57], which have some nice theoretical properties that can get $\epsilon$-close to the optimal solution, when evaluated with a greedy algorithm. Submodularity can be interpreted as a discrete analog of convexity [56]. Submodular functions have been becoming a recent hot-topic in combinatorial optimization [58–60]; however, there is still the issue of when to stop adding to the subset size and you must show the function being used in subset selection is submodular.

Zhou et al. presented a new view of feature selection where the features are streaming and new features are added to the predictor if they are beneficial [61, 62]. Furthermore, they take into consideration the runtime of the approach, which they name streamwise feature selection. Other approaches to subset selection considers correlation of random variables [63], and cost constraints on the features [64].

One of the primary observations to take away from this section is that there is a lack of subset selection implementations that are available to enable user to choose their objective functions for, while at the same time provide inference on $k^*$. The work in the following chapters focuses on the developments of meta selection algorithms that do not restrict the objective function (i.e., MSE, MI, JMI, etc.), and we provide inference of the parameter $k^*$, which is unknown.

2.1.5 Bandits in Subset Selection

Online learning allows efficient analysis of large volumes of data by not considering the entire data set when a classifier is being trained, and these approaches have only recently been considered for feature selection. Wang et al. present an online embedded
subset selection model that is inspired by the perceptron model [2, 65]. They show that the online model is capable of processing large volumes of data; however, their approach is not made classifier-independent and binary prediction problems are the only type addressed. The multi-armed bandits is an online algorithm that can be useful for subset selection.

**Multi-Arm Bandits**

Section 2.1.1 discussed the primary types of subset selection algorithm and there was a heavy focus on the use of classifier-independent (i.e., filters). This section highlights the use of bandit-like approaches in subset selection, however, we do not limit the discussion to only filter-like approaches.

The multi-arm bandit (MAB) addresses the problem of exploration-versus-exploitation (EvE) when a player needs to select a decision (from one of many) that maximizes the reward of the player over time. Robbins was one of the first to address this problem in 1952 while examining the problem of sequential decision making [66], and the MAB problem has become quite popular in the computer science community after Auer et al. derived finite time regret bounds for simple MAB algorithms [1]. The MAB problem is commonly presented as a gambler who is faced with deciding which slot machine lever to pull when she/he is faced with $K$ choices. Each slot machine has a reward distribution. The problem that the gambler faces is which machine’s level should she/he pull such that their reward is maximized after $n$ rounds of play. This is a problem of EvE, where we want to explore new levers (due to viewing the reward as a random variable), but exploit levers we know pay well. A much practical setting for MABs – as opposed to problems that gamblers face on a daily basis – is selecting adds based on users’ prior knowledge and interests [67]. In this setting, the reward could be user clicks (or some function of the # of clicks). In terms of feature subset
The upper confidence bound (UCB1) policy is one such approach bandit algorithm that has shown to be very successful in practice [1]. In this section, we solely focus on the UCB1 implementation (see Figure 2.5 for the pseudo-code). There exists a reward distribution $\nu_j$ over the $j$th arm. At each round $(t)$, a player samples the arm that maximizes (2.17). It is important to note that the player only observes the reward of the arm she/he pulls. The other arms’ rewards at round $t$ that do not maximize (2.17) are never observed.

The UCB1 policy, while simple, achieves sublinear regret. That is to say the “regret” the player feels for not sampling the best arm on each of the $T$ rounds grows sublinearly with $T$. To state this in mathematical terms, define

$$\mu^* = \max_{j \in [K]} \mu_j \quad \text{and} \quad j^* \in \arg \max_{j \in [K]} \mu_j$$

where $\mu^*$ is the reward of the best performing arm. Ideally, the player would pull this

---

**Figure 2.5: Pseudo code for the UCB1 Policy [1].**

```
Input: $K$ arms with rewards $\mu_1, \ldots, \mu_K$
1: for $t = 1, \ldots, T$ do
2:   Play arm $j^*$ where
3:     $j^* = \arg \max_{j \in [K]} \hat{\mu}_j + \sqrt{\frac{2 \log t}{t_j}}$ (2.17)
4:     where $t_j$ is the number of times arm $j$ was played and $\hat{\mu}_j$ is the average reward of arm $j$.
5: end for
```
arm on each of the $T$ rounds because it has the highest expected reward out of the $K$ available arms. The regret a player feels from playing $T$ rounds is defined by:

$$
\text{regret} = \mu^* T - \mathbb{E} \left[ \sum_{t=1}^{T} \mu_{I_t} \right]
$$

(2.18)

where $I_t$ is the arm that was played on round $t$. Given this definition, the regret of the UCB1 policy’s regret is no more than

$$
\text{regret} \leq \left(1 + \frac{\pi^2}{3}\right) \cdot \left(\sum_{j=1}^{K} \Delta_j\right) + 8 \sum_{i: \mu_i < \mu^*} \frac{\log T}{\Delta_i}
$$

(2.19)

after $T$ rounds of play, where $\Delta_j = \mu^* - \mu_j$ [1]. The regret for UCB1 grows sublinearly, actually logarithmically with $T$. Thus the per-round regret grows smaller, which is indicative of the UCB1 policy learning the optimal arm to play.

The UCB1 policy is only one such approach to solving the EvE problem described above. There are many other such approaches to managing EvE; however, for a more formal discussion of bandits, the reader is encouraged to read [68].

**Bandits in Feature Selection**

Busa-Fekete et al. presented a straightforward approaches to using subset selection with Adaboost that uses an adversarial MAB to select features on each round of boosting [69–72]. Their approach used a MAB to identify a set of features (they considered each feature as a set) that was shown to increase the accuracy Adaboost. The MAB would – over time – learn the best set of features to provide to weak-learn. Similar to Adaboost+MAB, many existing online and subset selection methods are classifier-dependent [2,73,74].

MABs are becoming increasingly popular for addressing feature subset selection, though they do not have a widespread adoption to the field. Kalyanakrishnan et al.
have studied MABs that select multiple arms, as opposed to one arm in the gambling example, from a very theoretical perspective [75, 76]. Their works considered not only theoretical quantities of regret that the algorithm experiences (i.e., how close to optimal is the arm selection strategy), but also the time complexity. Kalyanakrishnan et al. did not consider where the reward function for the multiple arms would come from, which is a central issue in feature selection using filters. The work discussed in Chapter 4 develops algorithms and reward qualities for implementing feature selection using MABs.

Gaudel & Sebag examine feature selection using a bandit approach in [77]. As opposed to a strict theoretical perspective, focus on developing an algorithm for implementing feature selection. They use the an upper confidence tree to search the space of feature subsets; however, their reward function for the bandit requires generating a classifier. Ashtiani et al. also examine the use of bandits for subset selection, though, their approach relies on a Euclidean target space with a $k$-NN classifier to score the sets [78]. Furthermore, the scalability of the approach to very large data was not evaluated, though, large scale data was not a central focus of their work.

Again, we find that it is difficult for online and bandit-like subset selection approaches to decouple themselves from a classifier.

### 2.2 Scaling Feature Selection for Large Data

The concept of large and big data has primarily focused on volume of data from the perspective of the number of observations [7]. However, as Zhai et al. point out, there is another component to volume in big data that is commonly overlooked, which is the dimensionality [10], and there is certainly no shortage of problems in big data where subset selection can help [5, 79–82]. Several recent works by Farahat et al. examined an instance subset selection approach, to reduce down the size of
the data, though there could still be an issue of dimensionality [83, 84]. Our goal with this work is to examine an approach to subset select that process subsets of the instances and feature space to learn the relative importance of features in a lightweight computational framework.

2.3 Figures of Merit

We need a way to measure the performance of a feature selection algorithm. The error of a classifier is not necessarily the primary objective when subset selection is used, furthermore, error does not address the consistency or stability of the subset selection method. That is to say, if we had two finite sample data sets \( \mathcal{D}_1 \) and \( \mathcal{D}_2 \) sampled from the probability distribution \( \mathcal{P} \), would the subset selection algorithm choose the same features for \( \mathcal{D}_1 \) and \( \mathcal{D}_2 \)? The answer is: probably not! There are three primary stability indices that can be considered for benchmarking the proposed subset selection algorithms on synthetic and real-world data sets.

**Definition: Jaccard Stability** [85] The Jaccard stability/consistency index for two subsets \( A \subset X \) and \( B \subset X \) is

\[
C_{\text{Jac}}(A, B) = \frac{|A \cap B|}{|A \cup B|}
\]

(2.20)

The Jaccard index is bounded between \([0, 1]\) where \( C_{\text{Jac}}(A, B) = 1 \) implies that \( A \) & \( B \) are identical sets, and \( C_{\text{Jac}}(A, B) = 0 \) implies that \( A \) & \( B \) do not share any of the same elements. This is a very simple consistency index that does not require that \( A \) & \( B \) are of the same size.

**Definition: Kuncheva Stability** [86] The Kuncheva stability/consistency index for two subsets \( A \subset X \) and \( B \subset X \), such that \( r = |A \cap B| \) and \( |A| = |B| = k \), where
\[ 1 \leq k \leq |\mathcal{X}| = K \]

is

\[ C_{Kun}(A, B) = \frac{rK - k^2}{k(K - k)} \tag{2.21} \]

where \( C_{Kun}(A, B) \in [-1, 1] \) for all subsets \( A \) and \( B \) in \( \mathcal{X} \).

**Definition: Lustgarten et al. Stability [87]** The Lustgarten stability/consistency index for two subsets \( A \subset \mathcal{X} \) and \( B \subset \mathcal{X} \) is

\[ C_{Lus}(A, B) = \frac{|A \cap B| - \frac{|A| \times |B|}{K}}{\min(|A|, |B|) - \max(0, |A| + |B| - K)} \tag{2.22} \]

Alternatively, if a classifier is used in conjunction with the a feature selection method then the error of a classifier can be used as a measure of performance. In practice, error is what applied researchers are interested (i.e., those who just want a software tool to work in the classification pipeline of their data). We use the average classification error, which for a binary classification problem with \( \hat{y}, y \in \{0, 1\} \), is given by

\[ \hat{\text{err}} = \frac{1}{m} \sum_{j=1}^{m} |\hat{y}_j - y_j| = \frac{1}{m} \|\hat{y} - y\|_1 \tag{2.23} \]

Furthermore, if the performance of a classifier is of interested to the end user, we should also consider other figures of merit, such as f-measure [88–90].

Precision, recall, and f-measure are also commonly used as performance measures. All of these measures can be obtained from the confusion matrix. The precision of a classifier, as shown in (2.24), is the ratio of the number of instances that are positive and are classified as positive to the number of instances that were classified as positive. The recall of a classifier, as shown in (2.25), is the ratio of the number of instances that are positive and are classified as positive to the number of instances that were
classified that are truly positive.

\[
\text{precision} = \frac{TP}{TP + FP}
\]  

(2.24)

\[
\text{recall} = \frac{TP}{TP + FN}
\]  

(2.25)

where \( TP \), \( FP \), and \( FN \) are the number of true positives, false positives and false negatives respectively.

The f-measure is a test of accuracy that takes into account the precision and recall. The f-measure or balanced f-score (f1 score) is the harmonic mean of precision and recall, as shown in (2.26). The harmonic mean is proportional to the squared G-mean divided by the arithmetic mean of the precision and recall.

\[
\text{f-measure} = \frac{2 \times \text{recall} \times \text{precision}}{\text{recall} + \text{precision}}
\]  

(2.26)

2.4 Analyzing Data from the Microbiome

2.4.1 An Introduction to the Microbiome

The microbiome is the population of the micro-organisms inhabiting an environment, which could have been collected from a human, soil or water environment. Microbes are absolutely everywhere and drive complex interactions with the places where they inhabit. Microbes are responsible for operations such as the oxygen and carbon-dioxide cycles. Not only are micro-organisms in the environment and throughout our bodies, but they are quite abundant. In fact, there are approximately 10 times more bacterial cells on, and throughout our body than there are our own cells. This realization has lead to a huge thrust to understand the microbes in our body, and
how they can be linked with diseases [91–93], obesity [94,95], and forensics [96].

The study of the microbiome has become a diversity field requiring individuals from multiple backgrounds working together (e.g., life science, mathematics, engineering and computer science). As an example, computer scientists, engineering and biologist are working together to develop a sustainable computational framework and web server to enable an easy to use workflow for researchers to analyze their data using web services such as MGRAST and KBase [8,9,97].

There several questions are of particular importance when the microbiome is being examined. In particular, who is there, how much of them is there, and what are they doing? Some of these questions can be addressed using DNA/RNA sequencing followed by homology and taxonomic classification. Answering such questions requires an advanced pipeline of computational tools to filter, analyze and characterize a sample.

2.4.2 An Introduction to Metagenomics

Metagenomics is the analysis of genetic material extracted from a sample(s) obtained directly from the environment. Each sample is comprised of many microorganisms, as opposed to the traditional study of genomics where an organism has its genome isolated then sequenced. The main difference between the traditional view of DNA analysis and metagenomics is that we are potentially sequencing from hundreds or thousands of organisms in a single sequencing run with metagenomics. While sequencing thousands of organisms sounds like a costly task, improvements to sequencing technology have significantly decreased the cost of obtaining sequences (see Table 2.1).

As mentioned earlier, a sample from a metagenomic study is directly collected from the environment, which can be a gram of soil [98,99], a milliliter of ocean
Table 2.1: The cost-accounting data presented here are summarized relative to two metrics: (1) “Cost per Megabase of DNA Sequence” - the cost of determining one megabase (Mb; a million bases) of DNA sequence of a specified quality; (2) “Cost per Genome” - the cost of sequencing a human-sized genome.

<table>
<thead>
<tr>
<th>Date</th>
<th>Cost per Mb</th>
<th>Cost per Genome</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jan-04</td>
<td>$1,598.91</td>
<td>$28,780,376</td>
</tr>
<tr>
<td>Jan-05</td>
<td>$974.16</td>
<td>$17,534,970</td>
</tr>
<tr>
<td>Jan-06</td>
<td>$699.20</td>
<td>$12,585,659</td>
</tr>
<tr>
<td>Jan-07</td>
<td>$522.71</td>
<td>$9,408,739</td>
</tr>
<tr>
<td>Jan-08</td>
<td>$102.13</td>
<td>$3,063,820</td>
</tr>
<tr>
<td>Jan-09</td>
<td>$2.59</td>
<td>$232,735</td>
</tr>
<tr>
<td>Jan-10</td>
<td>$0.52</td>
<td>$46,774</td>
</tr>
<tr>
<td>Jan-11</td>
<td>$0.23</td>
<td>$20,963</td>
</tr>
<tr>
<td>Jan-12</td>
<td>$0.09</td>
<td>$7,666</td>
</tr>
<tr>
<td>Jan-13</td>
<td>$0.06</td>
<td>$5,671</td>
</tr>
<tr>
<td>Jan-14</td>
<td>$0.04</td>
<td>$4,008</td>
</tr>
</tbody>
</table>

2.4.3 Answering Questions with Metagenomic Data

Once the environmental samples are collected, we can begin to answer basic questions by analyzing a microbial community through its genetic material. The first and second questions microbial ecologists’ like to answer are “who is in the sample”, and “how many of them are there”? In both questions, we use taxonomic classification approaches. The third question is “what are they doing” in the environment. Though, a better way to phrase this question is what is their function, can be answered using whole-genome shotgun data (described in the following section)?
Answering these three questions allows microbial ecologists to make inferences about their data, and address hypotheses about difference populations in an metagenomic sample. For example, assume that we are given DNA sequences collected from a control and a diseased population, what are the most significant differences between these two phenotypes? Here we use the term phenotype to identify a group within a metagenomic study. Examining the differences between phenotypes is a sub-field of metagenomics known as comparative metagenomics. We can borrow from other fields such as machine learning to perform the analysis of the data. For example, finding the bacteria or protein families in metagenomic samples that are collected from a control and a diseased population can be answered through feature selection [19, 22, 28].

2.4.4 Collecting Samples from the Microbiome

Genetic information is collected directly from the environment, and needs to be sequenced to address the aforementioned questions. Biological sequence data are typically derived from one of two popular methods, each of which depends on whether the scientist is interested in targeted genetic loci, or random genetic loci. Targeting a specific loci is achieved using polymerase chain reaction (PCR), and the method is referred to as amplicon sequencing. The 16S rRNA gene is commonly targeted in microbial research because of its slow mutation rate. Amplicon sequencing is ideal for classifying sequences into a taxonomic class, which allows ecologists to see who is in the sample and providing a measure of how abundant they are. The other approach, which randomly targets genetic loci, is the shotgun metagenomic sequencing. Notice that with the shotgun approach to sequencing, we are not targeting a particular gene when sequencing is performed, rather, many genes are being sequenced. Genes provide organisms with functions, and hence, shotgun sequencing allows ecologists to address what is the function within a sample.
One of the advantages to amplicon sequencing is that the monetary cost of acquiring sequences is far less, because only the 16S rRNA gene is being sequenced as opposed to any gene that is collected from the genetic sample. Unfortunately, limiting the gene being sequenced will restrict the functional knowledge that can be obtained from the genetic sample. Though, in either case of sequencing, the general process used for analyzing data can be quite similar.

2.4.5 Representing Data in the Microbiome

There are several pipelines that can be used to convert sequences to vectors in $\mathbb{R}^K$, so that we can apply traditional machine learning and mathematical algorithms to the metagenomic data. One such approach would be to start by removing short and low quality score reads from the entire batch of sequences. The remaining sequences are then clustered, using an algorithm such as CD-HIT, which forms a representative sequence for each cluster [106, 107]. Representative sequences are aligned using NAST [108], and grouped into taxonomic classifications using tools such as Ribosomal Database Project’s (RDP) naïve Bayes classifier (NBC) [109]. Each sample consists of thousands of reads that are classified into taxa or operational taxonomic units (OTUs) then each sample can be represented as a vector $\mathbf{x} \in \mathbb{N}^K_+$, where $K$ is the number of different taxa in the database, and $\mathbb{N}_+$ is the set of positive integers. The $i$th entry of $\mathbf{x}$ is the abundance of taxa $i$, or simply the number of times the $i$th OTU was detected in the sample. Other features, such as $m$-mers can also be considered; however, using $m$-mers leads to an extremely high dimensional space ($K = 4^m$), whereas the OTU features lie in a much smaller feature space.
2.4.6 Subset Selection and Its Role in the Analysis of the Microbiome

Feature subset selection has been used in several recent works related to the study of the microbiome; however, the use of such tools has not yet seen a widespread usage. Microbial ecologists must be able to make comparisons of within (α-diversity), and between (β-diversity) samples. Many of the commonly used software tools that implement these diversity approach are available in the QIIME software suite [110], which has become a de-facto standard when analyzing data collected from 16S (see [98, 111–114]). Many of the β-diversity analysis implemented in QIIME does not go past principal coordinate analysis (PCoA) [115]. The random forest is the only supervised tool within QIIME that can perform feature selection [14]; however, given our previous discussion, a classifier-independent method could be far more desirable.

Knights et al. performed an early study of supervised learning algorithms and their benefits to analyzing data collected from the microbiome [43], and their analysis evaluated multiple classification algorithms, some of which perform embedded feature selection (e.g., random forest and elastic nets). They found that the random forest was consistently one of the better classifiers in their experiments, while a recent work by Fernández-Delgado et al. found that the random forest was able to outperform many other classifiers on 200+ data sets [116]. Kursa and Rudnicki developed specific tools in R for performing feature selection with a random forest [117] (see [118] for microbiome research using the Boruta package). Other recent works have deviated away from random forests to more sophisticated methods, such as genetic algorithms for studying the vaginal microbiome [119]. Unfortunately, all of the recent works are classifier-dependent, and even approaches that use prior knowledge are classifier specific [120–122]. Garbarine et al. couple information-theoretic subset selection with a SVM [123], and more recent work by Lan et al. have studied the use of information-theoretic approaches on age-related feature selection [124].
The random forest has been widely in the study of the microbiome for supervised learning and it has the ability to score features for their importance. However, the value of classifier-independent cannot be understated enough in the field of metagenomics. Given our discussion in this section, developing novel, scalable, and classifier-independent would be not only a benefit to research in the microbiome, but also other related fields.

2.5 Summary

In this chapter we reviewed feature subset selection and categorized these approaches as: filters, embedded or wrapper methods. Furthermore, we gave definitions of relevance, weak relevance and irrelevance in terms of feature selection. Along with general definitions, we described the state-of-the-art approach in subset selection for each of these areas and drew the conclusion that methods are lacking for approaches that are: (i) classifier independent, (ii) objective function independent, (iii) provide an inference on the optimal number of features and (iv) scalable to very large data sets. In addition to subset selection, we provided a brief overview of the microbiome and how subset selection can provide researchers in microbial ecology insights to their data. In the following chapter, we present new methodologies for large-scale subset selection using filters and embedded methods.
3. Subset Selection with Neyman-Pearson Test

3.1 Introduction

This chapter presents a hypothesis test approach for feature selection that works with a generic feature selection/ranking rule (e.g., MIM, JMI, etc.) and the approach is specifically designed for large data sets. Our approach, which is described in detail below, addresses the following problems:

1. **Classifier Independent**: The proposed approach is classifier independent and the focus of the algorithm does not involve constructing a classifier. Instead, filters are used to evaluate feature subsets.

2. **Objective Function Independent**: The selection of the filter’s objective function is not limited in any way. This choice allows a user to define what they deem is relevant, and still use our proposed approach.

3. **Relevant Set Size Detection**: The proposed approach has the user choose the number of features they think are relevant and the hypothesis test returns the number of features that is best represented by the relevant set, even if the user’s choice was incorrect.

4. **Massively Scalable**: The proposed approach can scale to massive data sets that are potentially distributed across a cluster, and the approach fits into a MapReduce model [125].

3.2 Neyman-Pearson Subset Selection

Different feature selection algorithms optimize different objective functions, hence, different assumptions are made about the dispersion or distribution of the data and the meaning of feature importance through the filter. Unfortunately, few methods
Table 3.1: Mathematical Notations for NPFS

<table>
<thead>
<tr>
<th>Notation</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mathcal{X}$</td>
<td>full set of features, $</td>
</tr>
<tr>
<td>$\mathcal{F}$</td>
<td>feature selection algorithm</td>
</tr>
<tr>
<td>$X_l$</td>
<td>Bernoulli random variable indicating if a feature was selected as relevant on the $l$th bootstrap trial</td>
</tr>
<tr>
<td>$Z$</td>
<td>Binomial random variable</td>
</tr>
<tr>
<td>$H_0$</td>
<td>null hypothesis</td>
</tr>
<tr>
<td>$H_1$</td>
<td>alternative hypothesis</td>
</tr>
<tr>
<td>$\zeta_{\text{crit}}$</td>
<td>Neyman-Pearson (critical) threshold</td>
</tr>
<tr>
<td>$k$</td>
<td>number of features selected by $\mathcal{F}$</td>
</tr>
<tr>
<td>$n$</td>
<td>number of bootstraps</td>
</tr>
<tr>
<td>$T(Z)$</td>
<td>sufficient statistic of random variable $Z$</td>
</tr>
<tr>
<td>$\mathcal{A}$</td>
<td>base feature selection algorithm</td>
</tr>
</tbody>
</table>

can offer the dynamic selection of $k$, and fewer yet have the ability to work with other feature selection objective functions (e.g., they already have a specified filter criteria: see feature selection with the $\chi^2$ statistic [126]).

The Neyman-Pearson feature selection (NPFS) hypothesis test is an algorithm-independent meta–approach to determine an appropriate level of $k$. This approach can be used with any feature selection algorithm. Table 3.1 contains the mathematical notations used throughout this manuscript.

### 3.2.1 Algorithm Description & Derivation

A “base” feature selection algorithm, $\mathcal{A}$, is run $n$-times with bootstrap data sets sampled uniformly from $D$. In this setting, data instances – and not the features – that are sampled randomly from a distribution. For each bootstrap data set that is sampled, $\mathcal{A}$ selects $k$ of the $K$ features in the “relevant feature set”. For the moment, we assume there is a $k^*$, the optimal number of relevant features. Ideally, the same
k features would be found by \( \mathcal{A} \) as relevant over each of the \( n \) trials; however, this is rarely the case due to initializations and randomness in the bootstrap sample. A consistency index can be used to measure the stability of the relevant feature sets over these \( n \) trials. This index, however, is not based on a statistical hypothesis test, nor is it designed to determine if a feature is consistently selected as relevant. In fact, by Kuncheva’s formulation, \( C_{\text{Kun}}(A, B) \) is a random variable (this is easy to see in (2.21) since \( R = r \) is a random variable with a hypergeometric distribution).

Let us first consider a hypothesis test being applied to a single feature (the proposed test can be applied to each feature individually). At each bootstrap iteration, \( \mathcal{A} \), returns a set of indices for the relevant feature set. For each feature in the set \( \mathcal{X} \), we mark whether the feature was in the relevant set \( (X_l = 1) \), or not in the set \( (X_l = 0) \), where \( l \in [n] \) is the bootstrap iteration. Note that we have slightly adjusted our definition of \( X_l \) from the previous chapter. The previous chapter viewed \( X_l \) random variable as a feature; however, now it is whether a feature was detected as relevant.

In this situation, we can determine that the random variable \( X_l \) is distributed as a Bernoulli random variable with probability \( p \) (that is yet to be determined). The \( n \) Bernoulli random variables from the \( n \) bootstrap data sets form a Binomial distribution with

\[
Z_n = \sum_{l=1}^{n} X_l
\]
successes \((Z_n = z \text{ be an observation of the random variable } Z_n)\). If a feature is selected by chance, then the probability for such a feature appearing in the relevant feature set is \( p_0 = k/K \). Now, there is the observed probability of a feature appearing in the relevant feature set from the bootstrap trials, which is \( p_1 = z/n \). If all features were equally relevant (or equally irrelevant), we would expect these probabilities to
be equal to one another. Ultimately, we would like to know if \( p_1 > p_0 \), or in other words, if the probability of a feature being in the relevant set is greater than the probability of a feature being selected by random chance. Against this background we have a hypothesis test formulated as follows,

\[
H_0 : p_0 = p_1 \\
H_1 : p_1 > p_0
\]

where \( H_0 \) is the null hypothesis (that all features are equally relevant), and \( H_1 \) is the alternative hypothesis (that some features are more relevant than others). We select the Neyman-Pearson test for several reasons: (i) the likelihood functions under \( H_0 \) and \( H_1 \) can be explicitly computed as shown below, (ii) the solution with the Neyman-Pearson lemma is a simple yet elegant result, and (iii) perhaps most importantly, the Neyman-Pearson test is the most powerful test available for size \( \alpha \) [127]. The Neyman-Pearson lemma states that we reject the null hypothesis if,

\[
\Lambda(z) = \frac{\mathbb{P}(z|H_1)}{\mathbb{P}(z|H_0)} > \zeta_{\text{crit}}
\]  

where \( \mathbb{P}(z|H_0) \) is the probability distribution under the null hypothesis, \( \mathbb{P}(z|H_1) \) is the probability distribution under the alternative hypothesis, and \( \zeta_{\text{crit}} \) is a threshold such that,

\[
\mathbb{P}(T(z) > \zeta_{\text{crit}}|H_0) = \alpha
\]  

where \( \alpha \) is size of the test, and \( T(z) \) is the test-statistic. Using \( \log \Lambda(z) \) would provide equivalent results since taking the logarithm does not affect the solution. Recall that the random variable \( Z \) follows a Binomial distribution. Using (3.1) and the form of
the probability distribution on \( Z \), we apply the Neyman–Pearson lemma:

\[
\frac{\mathbb{P}(Z_n = z|H_1)}{\mathbb{P}(Z_n = z|H_0)} = \frac{\binom{n}{z} p_1^n (1-p_1)^{n-z}}{\binom{n}{z} p_0^n (1-p_0)^{n-z}}
\]

\[
= \left( \frac{p_1}{p_0} \right)^z \left( \frac{1-p_1}{1-p_0} \right)^{n-z}
\]

\[
= \left( \frac{p_1}{p_0} \right)^z \left( \frac{1-p_1}{1-p_0} \right)^n \left( \frac{1-p_0}{1-p_1} \right)^z
\]

\[
= \left( \frac{1-p_1}{1-p_0} \right)^n \left( \frac{p_1}{1-p_1} \right)^z \left( \frac{p_0}{1-p_0} \right)^z
\]

\[
> \zeta_{\text{crit}}
\]

Since \( \left( \frac{1-p_1}{1-p_0} \right)^n \) is simply a constant, which can be moved to the other side of the inequality, resulting in a new threshold \( \zeta'_{\text{crit}} \). Thus,

\[
\left( \frac{p_1 (1-p_0)}{p_0 (1-p_1)} \right)^z > \zeta'_{\text{crit}}
\]

Taking the logarithm gives us

\[
z \log \left\{ \frac{p_1 (1-p_0)}{p_0 (1-p_1)} \right\} > \zeta''_{\text{crit}}
\]

where, again, the logarithm term is simply a constant and it can be removed to find a scaled threshold \( \zeta'''_{\text{crit}} \). Thus, we are seeking

\[
z > \zeta'''_{\text{crit}}
\]

where \( \zeta_{\text{crit}} \) is a critical threshold determined by \( \mathbb{P}(z > \zeta'''_{\text{crit}}|H_0) = \alpha \) (note by definition that \( z \) is a sufficient statistic for \( T(z) \)). Since the probability distribution on the null
hypothesis is known (i.e., Binomial), we may explicitly solve for \( \zeta'''_{\text{crit}} \).

\[
\mathbb{P}(z > \zeta'''_{\text{crit}} | H_0) = 1 - \mathbb{P}(z \leq \zeta'''_{\text{crit}} | H_0) = \alpha
\]

(3.3)

Since \( \mathbb{P}(z \leq \zeta'''_{\text{crit}} | H_0) \) has a closed form expression it can be obtained from a lookup table. Note that \( \alpha \) can be used to control how conservative the hypothesis test will be. That is, if \( \alpha \) is small, it will become more difficult for a feature to be detected as relevant because \( \zeta'''_{\text{crit}} \) will become large. To summarize, NPFS is implemented as follows (also in Figure 3.1):

1. Run a feature selection algorithm \( \mathcal{A} \) on \( n \) independently sampled data sets (sampling instances, not features). The independently sampled data sets can be a result of cross-validation or bootstrap samples. Form a matrix \( \mathbf{X} \in \{0, 1\}^{K \times n} \) where \( \{\mathbf{X}\}_i \) is the Bernoulli random variable for feature \( i \) on trial \( l \).

2. Compute \( \zeta'''_{\text{crit}} \) using (3.3), which requires \( n, p_0 \), and the Binomial inverse cumulative distribution function.

3. Let \( \{z\}_i = \sum_{l=1}^{n} \{\mathbf{X}\}_i \). If \( \{z\}_i > \zeta'''_{\text{crit}} \) then feature belongs in the relevant set, otherwise the feature is deemed non-relevant. Use only the features selected by the Neyman-Pearson detector for learning a classification or regression function.

As eluded in Figure 3.1, NPFS fits into the framework for MapReduce [125], which allows to the subset selection to take place on data sets with a massive number of instances.

3.2.2 Theoretical Properties on NPFS

An important property of the proposed approach is that if \( X_1, \ldots, X_n \sim \text{Bernoulli}(p) \), then we expect the difference between \( p \) and its bootstrap estimate \( \hat{p} \) to become arbi-
Figure 3.1: Pseudo code for NPFS.

Trarily small as $n$ grows large. The probability of the magnitude of difference between $p$ and $\hat{p}$ being greater than some $\epsilon > 0$ can be upper bounded using Hoeffding’s inequality.

**Theorem 3.2.1 (Hoeffding’s Inequality [41])** Let $Y_1, Y_2, \ldots, Y_n$ be independent random observations such that $E[Y] = \mu$, $\bar{Y} = \frac{1}{n} \sum_i Y_i$, and $a \leq Y_i \leq b$. For any $\epsilon > 0$, the following inequality holds,

$$
P(|\bar{Y} - \mu| \geq \epsilon) \leq 2e^{-2n\epsilon^2/(b-a)^2} \quad (3.4)
$$

Hoeffding’s inequality is similar to that of Markov’s inequality; however, it produces a tighter bound for larger deviations. We may use Hoeffding’s inequality with a few assumptions to bound the differences between the bootstrap’s estimate $\hat{p}$, and the true probability $p$. If $X_1, \ldots, X_n \sim \text{Bernoulli}(p)$, then for any $\epsilon > 0$, we have,

$$
P(|\hat{p} - p| \geq \epsilon) \leq 2e^{-2n\epsilon^2} \quad (3.5)
$$

where $\hat{p} = \frac{1}{n}Z_n$. Thus if $X_1, \ldots, X_n \sim \text{Bernoulli}(p)$, then $\hat{p}$ approaches $p$ exponentially quickly as a function of $n$. Chebyshev’s inequality can also be used to find a bound
on $P(|\hat{p} - p| \geq \epsilon)$; however, Hoeffding’s inequality provides a tighter upper bound for larger values of $\epsilon$.

### 3.2.3 Advantages of NPFS

The proposed method for post-analysis of feature selection offers several capabilities. Let us assume that $k$ was selected to be too large compared to the true number of relevant features, $k^*$. How can we determine a more accurate value of $k$? The proposed approach provides a natural solution: simply use the features that Neyman-Pearson detector returns as being relevant. Note that the number of features returned by the Neyman-Pearson detector need not be $k$: if $k$ were too large, we expect the test to return fewer relevant features. Having such an inference on $k$ can reduce the complexity of the classifier or the regression function. We can also ask the opposite question: what if $k$ – provided as a user-input to the feature selection algorithm – was selected too small? Could we apply this hypothesis test to determine the subset of $K$ features that are relevant even though $A$ never selects all of them because $k$ was smaller than $k^*$?

### 3.3 Experiments

Our proposed methodology for feature relevance using NPFS was implemented using joint mutual information (JMI), unless otherwise stated, as the baseline feature selection objective function. In this section, we seek to determine the behavior of the hypothesis testing procedure through several experiments on synthetic and real-world data. We wish to answer the following questions:

1. Given a controlled data set, can NPFS correctly identify the truly relevant features?
2. If $k$ were selected too large, can NPFS identify the subset of the $k$ features that should be used instead of the set of $k$ features?

3. If $k$ were selected too small, can NPFS identify all the relevant features that could not be identified as relevant due to $k$ being too small?

We provide a Matlab implementation of NPFS under the GNU GPLv3.

### 3.3.1 Data Sets and Testing Procedure

The proposed Neyman-Pearson hypothesis testing methodology (NPFS) for any given feature selection algorithm was tested on a synthetic data set, and a collection of data obtained from the UCI machine learning repository [128] (see Table 3.2). The synthetic data, described below, allows us to tailor experiments to test the strengths and weaknesses of the proposed approach.

#### Description of the Uniform Data

$M$ observations are generated with features that are independently and identically distributed (iid) uniform random variables in the interval $[0, 10]$. This data set is referred to as $\mathcal{D}_{uni}$. Each feature vector $\mathbf{x}_m$ for $m \in [M]$ has $K$ features. The true labeling function, unknown to any algorithm, is given by,

$$y_m = \begin{cases} 
1, & \sum_{i=1}^{k^*} x_m(i) \leq 5 \cdot k^* \\
0, & \text{otherwise}
\end{cases}$$

Hence, only the first $k^*$ features carry information for determining the label $y_m$ of a feature vector $\mathbf{x}_m$. Our goal is to identify, using our hypothesis test, those features (indices $i \in [k^*]$) that are relevant to the classification problem. Note that the threshold for determining the class label is the statistical expectation of the linear

\footnote{http://github.com/EESI/NPFS}
combination of the first \( k^* \) feature variables (this is easily shown using the properties of the expectation of a linear function). Such a threshold sets the prior probability on each of the classes to approximately \( \frac{1}{2} \) for a randomly sampled data set.

There are \( n \) bootstrap data sets drawn from \( \mathcal{D}_{\text{uni}} \), and the JMI feature selection algorithm is run independently on each sampled bootstrap set. \( k \) of \( K \) features are selected for each bootstrap data set, and a vector with binary indicators representing whether or not the feature was selected is produced. The \( n \) vectors form a \( K \times n \) matrix with binary entries (i.e., \( X \)). Each row, corresponding to a feature, is the sequence of Bernoulli experiments of success and failures used in NPFS.

### 3.3.2 Results on Synthetic Data Sets

Let us start with our questions on appropriate selection of \( k \): if \( k \) is selected too large, can \( k^* \) be found such that \( k^* < k \), and what is approximately the ideal value of \( k \) given the results from the \( n \) bootstraps? In this experiment, 5 features were considered relevant out of 25 features (recall that the features are uniform random variables). The value of \( k \) was varied from 10 to 24. For these cases, there are (at least) 5 to 19 irrelevant features are incorrectly selected as relevant at any given bootstrap iteration. We apply the Neyman-Pearson test after 100 bootstraps. Figure 3.2 shows that the Neyman-Pearson test can identify when irrelevant features are being selected by JMI. In this figure, the matrix \( X \) is visualized with white entries indicating features selected by JMI at different bootstrap iterations. The orange rows highlight the features that Neyman-Pearson method identifies as being relevant. Note that features \( \{1, 2, 3, 4, 5\} \) are the only relevant features for this problem. Clearly, the inference provided by the Neyman-Pearson test allows us the ability to reduce \( k \) to achieve a much smaller subset of relevant features. In each of these experiments, we find that there are a few features being detected as relevant, which are actually
It is possible to tune $n$ and $\alpha$ such that in every experiment only features 1 through 5 are being detected as relevant. In every experiment, however, the proposed method is always recommending the use of fewer features, because many of the features JMI selects at each bootstrap iteration are irrelevant.

![Figure 3.2](image)

Figure 3.2: Results of the Neyman-Pearson hypothesis test applied to the synthetic uniform data set for different cardinalities of the relevant feature set. The Neyman-Pearson hypothesis test recovers the original 5 relevant features (first 5 rows of each plot) with only a few additional irrelevant features in the set. This is a visualization of $X$, where black segments indicate $X_l = 0$, white segments $X_l = 1$, and the orange rows are the features detected as relevant by the Neyman-Pearson test.
Figure 3.3: Number of features selected by the Neyman-Pearson detector for varying levels of $k$ (too large & too small) when there are 15 relevant features ($k^*$) in the synthetic data set. The number of features selected by the proposed approach appears to be converging to 15 when $k$ is initially selected too small. Even though the number of selected features diverges when $k$ is selected too big, they undershoot the original guess while the too small $k$’s overshoot their original guesses.

The second key question is: can the value of $k^*$ be recovered if $k$ was initially chosen too small, and if so, how many bootstraps are needed? To examine this situation, three more synthetic uniform data sets were generated. All synthetic data sets’ features are uniform random variables with 15 relevant features; however, the data set have 50, 100, or 250 features. We apply our Neyman-Pearson test with the number of bootstraps varying between 1 and 500. Furthermore, $k \in \{3, 5, 10, 15, 25\}$ are examined. Figure 3.3(a) shows that the value $k^*$ selected by the Neyman-Pearson algorithm is approaching the true value for various selections of $k$. We should note that we can improve these results by increasing the number of observations in the data set. However, if $k$ were too large, there are still a few features left in the relevant set as determined by the Neyman-Pearson detector (as observed previously in Figure 3.2). Figure 3.3(c) shows the effect of using 250 features rather than 50 features. Again, if $k$ were selected too small, the Neyman-Pearson detector finds approximately $k^*$ features; however, the method still unable to completely recover all of them with 500 bootstraps.
3.3.3 Results on UCI Data Sets

In this section, we present the classification error using a base classifier trained on: (i) all features, (ii) trained on the top 10 features selected by JMI, and (iii) trained on features selected by the proposed approach. The data sets are obtained from the UCI machine learning repository [128], and [52] mRMR paper. The naïve Bayes (nb) and CART algorithms are used as baseline classifiers [3, 129]. We use the following notation to denote the classifier and the feature selection algorithm: nb (naïve Bayes trained on all features), nb-npfs (naïve Bayes trained with features identified by JMI and the proposed NPFS), and nb-jmi (top 10 features selected with JMI). It is important to note that we do not have access to the (true) $k^*$ or the degree of feature relevancy for these data sets, therefore, we must examine the performance of a classifier to evaluate the methods effectiveness.

Table 3.2 presents each classifier’s error and its rank (see [130]). The proposed approach for both the naïve Bayes and CART produces the best average rank. Unfortunately, there is not enough statistical evidence to suggest that the proposed approach provides uniformly the lowest error rate. There is, however, statistical significance between CART-NPFS and CART-JMI, with CART-NPFS out performing CART-JMI with an $\alpha$-level of 0.1 using the Wilcoxon’s signed rank test. The average number of features being selected by the Neyman-Pearson test after 10,000 bootstraps can be found in Figure 3.4. The UCI data sets do not allow us to control the level of feature relevancy as we did with the synthetic data and it is worth noting that we do not observe NPFS detecting all features as relevant even when the number of bootstraps is quite large.
Figure 3.4: Variation in the Neyman-Pearson’s test for the value of $k^*$ given that $k$ may have been selected too small. $x$-axis represents the data set under test and the $y$-axis is the predicted $k^*$ by the proposed approach using 10,000 bootstraps.
Figure 3.5: **Top row:** 16 × 16 image from the OCR data set corrupted with noisy pixels. The actual OCR images are 8 × 8 and take a 4-bit value. **Bottom row:** Irrelevant features marked by the Neyman-Pearson test are indicated in black. Note ONLY black pixels are irrelevant feature and not the “actual” value of the pixel (i.e., we have scaled the pixel to assure there were not black pixels). The Neyman-Pearson test selects a subset of 52 features in the 16 × 16 image that are relevant.
Table 3.2: Classification errors of a Naïve Bayes and CART tested on the UCI data sets (see section 3.3.3) and rank after 10-fold cross validation. The errors in the table have been truncated; however, the ranks are determined via the untruncated values.

<table>
<thead>
<tr>
<th>data set</th>
<th>instances</th>
<th>features</th>
<th>nb</th>
<th>nb-jmi</th>
<th>nb-npfs</th>
<th>cart</th>
<th>cart-jmi</th>
<th>cart-npfs</th>
</tr>
</thead>
<tbody>
<tr>
<td>breast</td>
<td>569</td>
<td>30</td>
<td>0.069 (3)</td>
<td>0.055 (1.5)</td>
<td>0.055 (1.5)</td>
<td>0.062 (3)</td>
<td>0.056 (2)</td>
<td>0.041 (1)</td>
</tr>
<tr>
<td>congress</td>
<td>435</td>
<td>16</td>
<td>0.097 (3)</td>
<td>0.088 (1)</td>
<td>0.088 (2)</td>
<td>0.051 (3)</td>
<td>0.051 (1.5)</td>
<td>0.051 (1.5)</td>
</tr>
<tr>
<td>heart</td>
<td>270</td>
<td>13</td>
<td>0.156 (1)</td>
<td>0.163 (2)</td>
<td>0.174 (3)</td>
<td>0.244 (3)</td>
<td>0.226 (2)</td>
<td>0.207 (1)</td>
</tr>
<tr>
<td>ionosphere</td>
<td>351</td>
<td>34</td>
<td>0.117 (3)</td>
<td>0.091 (2)</td>
<td>0.091 (1)</td>
<td>0.077 (3)</td>
<td>0.068 (1)</td>
<td>0.074 (2)</td>
</tr>
<tr>
<td>krvskp</td>
<td>3196</td>
<td>36</td>
<td>0.122 (3)</td>
<td>0.108 (1)</td>
<td>0.116 (2)</td>
<td>0.006 (1)</td>
<td>0.056 (3)</td>
<td>0.044 (2)</td>
</tr>
<tr>
<td>landsat</td>
<td>6435</td>
<td>36</td>
<td>0.204 (1)</td>
<td>0.231 (2.5)</td>
<td>0.231 (2.5)</td>
<td>0.161 (1)</td>
<td>0.173 (2)</td>
<td>0.174 (3)</td>
</tr>
<tr>
<td>lungcancer</td>
<td>32</td>
<td>56</td>
<td>0.617 (3)</td>
<td>0.525 (1)</td>
<td>0.617 (2)</td>
<td>0.542 (2)</td>
<td>0.558 (3)</td>
<td>0.533 (1)</td>
</tr>
<tr>
<td>parkinsons</td>
<td>195</td>
<td>22</td>
<td>0.251 (3)</td>
<td>0.170 (1.5)</td>
<td>0.170 (1.5)</td>
<td>0.133 (1.5)</td>
<td>0.138 (3)</td>
<td>0.133 (1.5)</td>
</tr>
<tr>
<td>pengcolon</td>
<td>62</td>
<td>2000</td>
<td>0.274 (3)</td>
<td>0.179 (2)</td>
<td>0.164 (1)</td>
<td>0.21 (1)</td>
<td>0.226 (2.5)</td>
<td>0.226 (2.5)</td>
</tr>
<tr>
<td>pengleuk</td>
<td>72</td>
<td>7070</td>
<td>0.421 (3)</td>
<td>0.029 (1)</td>
<td>0.043 (2)</td>
<td>0.041 (2)</td>
<td>0.027 (1)</td>
<td>0.055 (3)</td>
</tr>
<tr>
<td>penglung</td>
<td>73</td>
<td>325</td>
<td>0.107 (1)</td>
<td>0.368 (3)</td>
<td>0.229 (2)</td>
<td>0.337 (1)</td>
<td>0.530 (3)</td>
<td>0.504 (2)</td>
</tr>
<tr>
<td>penglymp</td>
<td>96</td>
<td>4026</td>
<td>0.087 (1)</td>
<td>0.317 (3)</td>
<td>0.140 (2)</td>
<td>0.357 (3)</td>
<td>0.312 (2)</td>
<td>0.311 (1)</td>
</tr>
<tr>
<td>pengnci9</td>
<td>60</td>
<td>9712</td>
<td>0.900 (3)</td>
<td>0.600 (2)</td>
<td>0.400 (1)</td>
<td>0.667 (2)</td>
<td>0.617 (1)</td>
<td>0.783 (3)</td>
</tr>
<tr>
<td>semeion</td>
<td>1593</td>
<td>256</td>
<td>0.152 (1)</td>
<td>0.456 (3)</td>
<td>0.387 (2)</td>
<td>0.25 (1)</td>
<td>0.443 (3)</td>
<td>0.355 (2)</td>
</tr>
<tr>
<td>sonar</td>
<td>208</td>
<td>60</td>
<td>0.294 (3)</td>
<td>0.279 (2)</td>
<td>0.241 (1)</td>
<td>0.259 (2)</td>
<td>0.263 (3)</td>
<td>0.201 (1)</td>
</tr>
<tr>
<td>soybean</td>
<td>47</td>
<td>35</td>
<td>0.000 (2)</td>
<td>0.000 (2)</td>
<td>0.000 (2)</td>
<td>0.020 (2)</td>
<td>0.020 (2)</td>
<td>0.020 (2)</td>
</tr>
<tr>
<td>spect</td>
<td>267</td>
<td>22</td>
<td>0.210 (2)</td>
<td>0.206 (1)</td>
<td>0.232 (3)</td>
<td>0.187 (1)</td>
<td>0.210 (2)</td>
<td>0.229 (3)</td>
</tr>
<tr>
<td>splice</td>
<td>3175</td>
<td>60</td>
<td>0.044 (1)</td>
<td>0.054 (2)</td>
<td>0.055 (3)</td>
<td>0.085 (3)</td>
<td>0.070 (2)</td>
<td>0.066 (1)</td>
</tr>
<tr>
<td>waveform</td>
<td>5000</td>
<td>40</td>
<td>0.207 (3)</td>
<td>0.204 (2)</td>
<td>0.202 (1)</td>
<td>0.259 (3)</td>
<td>0.238 (2)</td>
<td>0.228 (1)</td>
</tr>
<tr>
<td>wine</td>
<td>178</td>
<td>13</td>
<td>0.039 (2.5)</td>
<td>0.039 (2.5)</td>
<td>0.034 (1)</td>
<td>0.079 (3)</td>
<td>0.068 (1.5)</td>
<td>0.068 (1.5)</td>
</tr>
<tr>
<td>average</td>
<td></td>
<td></td>
<td>2.275</td>
<td>1.900</td>
<td>1.825</td>
<td>2.075</td>
<td>2.1250</td>
<td>1.800</td>
</tr>
</tbody>
</table>
3.3.4 Optical Character Recognition

Our final experiment uses the optical character recognition data set collected from UCI Machine Learning Repository. Each image in the experiment consists of 64 pixels represented by 4-bits (i.e., an $8 \times 8$ image); however, each image has been corrupted by adding noisy pixels. The final image is $16 \times 16$. Just as before, we run 100 bootstrap trials with the JMI feature selection algorithm and apply the Neyman-Pearson hypothesis test. In this experiment $k = 64$ and $K = 256$. Each noisy pixel is sampled from a uniform probability mass function taking possible values $\{1, \ldots, 16\}$.

Figure 3.5 presents the NPFS results on OCR data set. The top row of Figure 3.5 shows the $16 \times 16$ images corrupted with noisy pixels. Note that the original OCR images can be observed as they are embedded within the noise. The bottom row of Figure 3.5 shows the irrelevant features marked in black by the Neyman-Pearson test. Note that only the black pixels are irrelevant features and not the “actual” value of the pixel (i.e., we have scaled the pixel to assure there were not black pixels). The Neyman-Pearson test selects a subset of 52 features in the $16 \times 16$ image that are relevant. Thus the Neyman-Pearson test is suggesting that there is a subset of features, fewer than 64, that are relevant for the discrimination between the characters in the image.

3.3.5 Biasing NPFS to Control False Positives

One of the issues that was observed with our method is that if $n$ is increased NPFS may select more features than are actually relevant. We attribute this to have the null hypothesis of the NPFS set to assume that all features can fall into the relevant set uniformly at random which is an unrealistic assumption. One way to address this
problem is to regularize the hypothesis with a bias term as follows:

\[
\begin{align*}
H_0 : p_0 + \beta &= p_1 \\
H_1 : p_1 &> p_0 + \beta
\end{align*}
\]

where \( H_0 \) is the null hypothesis, \( H_1 \) is the alternative hypothesis and \( \beta \in (0, 1 - p_0] \) is a bias term. However, a practical selection of \( \beta \) should be small. We omit the derivation of the test statistic and simply note that the resulting test is equivalent to NPFS, with the exception that the calculation of the critical threshold using (3.2) assumes that \( Z \) is Bernoulli with probability \( p_0 + \beta \) rather than \( p_0 \). This biased algorithm is referred to as NPFS-\( \beta \). The effect of the \( \beta \) heuristic is examined in the experimental results section.

Figure 3.6(a) shows the effect that \( \beta \) has on the critical threshold \( \zeta_{\text{crit}}' \) for a fixed value of \( n \) and size of the hypothesis test (for convenience we write \( \zeta_{\text{crit}}' \) as \( \zeta \)). For this experiment \( n = 100 \) and \( \alpha = 0.01 \). As desired increasing \( \beta \) increases the critical threshold; hence, making it more difficult to select a feature as being relevant, which leads to a more conservative test for selecting features. Blindly increasing \( \beta \) to be arbitrarily large (e.g., \( \beta = 0.3 \)) more than doubles the number of times a feature needs to be detected as relevant by \( \mathcal{A} \) to be considered relevant by NPFS-\( \beta \).

However, why not tune the value of \( \alpha \), which can also be used to control the size/conservativeness of the test? Figure 3.6(b) show \( \alpha \) varying from 0.001 to 0.2. This figure shows that we have more control over how conservative the test is by changing \( \beta \) than \( \alpha \). Furthermore, \( \beta \), like \( \alpha \), has an intuitive sense of control over the conservativeness of the test. Also, \( \alpha \), as we would expect, has the most control of \( \zeta \) when \( \alpha \) is small. Unfortunately, this makes it more difficult to control the conservatives of the test when such small changes in \( \alpha \) correspond to exponential changes in \( \zeta \).
Figure 3.6: (left) Increasing the value of $\beta$ increases the critical threshold $\zeta$ for several different values of $p_0$ ($n = 100$ and $\alpha = 0.01$ for this experiment). (right) Effect that $\alpha$ has on the critical threshold $\zeta$.

Figure 3.7 examines the effects of $\beta$ and $n$ on NPFS when evaluated on the synthetic data set described above. In this experiment $\beta \in [0, \frac{1}{5}]$, $n \in [10, 500]$, $M = 1,000$, $K = 50$, and MIM is the base feature subset selection method $\mathcal{A}$ [51]. The Jaccard index is our performance figure of merit.

Figure 3.7(a), 3.7(b), 3.7(c), and 3.7(d) fixes the relevant feature set to $\{1, \ldots, 10\}$ and MIM selects 3, 5, 15, 25 features, respectively. Note that in the case of $k = \{3, 5\}$, MIM selects fewer than the optimal number of features (i.e., $k^* = 10$), and for $k = \{15, 25\}$, MIM selects more than the optimal number of features. NPFS works better with $\beta = 0$ when $k < k^*$, which goes along with our intuition about how $\beta$ affects NPFS. $\beta$ increases the critical threshold for the hypothesis test, which makes it more difficult for a feature to be detected as relevant; hence, the lower Jaccard indices for larger values of $\beta$ in the setting where $k < k^*$. When $k > k^*$, we observe that $\beta$ improves the Jaccard index for NPFS. Similar observations can be made for the remaining subplots in Figure 3.7, which shows the same experiment for $k^* = 15$.

Clearly, $\beta$ can have a very large impact on NPFS even on this synthetic data set,
Figure 3.7: Jaccard index of NPFS measured against the set \( \{1, \ldots, k^*\} \) on a synthetic uniform data set. The number of bootstraps are varied from 10 → 100 and the bias parameter is varied from 0 → \( \frac{1}{5} \).
how should we proceed to choose $\beta$? In most settings, empirical results suggest to use a smaller value of $k$ with a small $\beta < \frac{k}{K}$ then increase the number of bootstraps. This observation can be made in Figures 3.7(a), 3.7(b), 3.7(e), 3.7(f), and 3.7(g). These experiments demonstrate that $\beta$ must be chosen very carefully and not naively, similar to choosing the learning rate for gradient descent (see Duda et al.’s discussion on learning rates and overshooting [3]).

3.3.6 Convergence Checking for $\hat{p}_t$

The number of bootstraps, or random calls to a collection of databases, is a free parameter of NPFS. Therefore, in this section we examine how implementing early stopping with NPFS can avoid unnecessary computation. Let $\hat{p}_t \in [0, 1]^K$ be a vector containing the ratio of the number of times a feature was detected as relevant over the number of trials that have been performed until time $t$. A straightforward stopping criteria is to define a bound on the average difference between two consecutive time stamps $t$ and $t - 1$, which is given by

$$
\frac{1}{K} \| \hat{p}_t - \hat{p}_{t-1} \|_1 \leq \xi
$$

(3.6)

where $\xi > 0$ is some small number to define the minimal change needed to continue running NPFS. While the selection $\xi$ must be chosen somewhat carefully, we argue that this stopping criteria is a quick way to perform early stopping and has been used for early stopping in neural network training [42].

We evaluated implementation of early stopping on the synthetic data set described in the beginning of the section, where $K = 100$, $k^* = 25$, $k = 10$, and we average our results over 100 randomly sampled data sets. We set the maximum number of bootstraps to $n = 1000$, which given the properties of the Binomial RVs in $\mathbb{Z}$, such a value of $n$ should be sufficient to reach convergence. We set $\xi = \{0.001, 0.0005, 0.0001\}$
and evaluated NPFS using the Jaccard index.

Figure 3.8 shows the Jaccard index for NPFS as a function of the number of bootstraps. We find that if early stopping is applied that the average stopping times for the three different thresholds are given by $T_{\xi=0.001} = 81.56$, $T_{\xi=0.0005} = 156.06$, and $T_{\xi=0.0001} = 758.12$. Similar to other areas that use early stopping, there are generally small decreases in performance; however, the advantage is the amount of time the algorithm takes to run is significantly lower with early stopping. NPFS was observed to converge in the Jaccard and Lustgarten rather quickly, and this can be explained by the quick convergence of $\frac{1}{K}\|\hat{p}_t - \hat{p}_{t-1}\|_1$ to zero. Figure 3.9 shows $\frac{1}{K}\|\hat{p}_t - \hat{p}_{t-1}\|_1$ for $\xi = 0.0005$ applied to varying levels of dimensionality. For all selections of $K$, NPFS stops running before the maximum number of bootstraps is reached when early stopping is implemented.
Figure 3.9: The early stopping criteria, $\frac{1}{K}||\hat{p}_t - \hat{p}_{t-1}||_1$, for different size feature sets.

Figure 3.10: (left) Runtime of NPFS versus lasso on a large synthetic data set. (right) NPFS evaluated on a very large data set. The NPFS times can be interpreted as the amount of time it takes to complete a data set of size $X$ GB.
3.3.7 Big Data Trials

We have also evaluated NPFS on a massive synthetic data set to prove its scalability for data that have a massive \( M \) and a large \( K \) (i.e., \( M = 58,340,000 \) and \( K = 10,000 \)). The entire data set was separated into many smaller files (\( \approx 100\text{MB} \)) to allow for our computing nodes to load portions of the data into memory without crashing\(^2\). Lasso and NPFS were implemented in Matlab, where NPFS uses MIM as the base-subset selection algorithm. NPFS takes advantage of parallel computing in its implementation. We process about 100MB of data at a time. We also increase the number of bootstraps as the larger data sets are processed to take advantage of parallelism within NPFS.

Figure 3.10(a) shows the evaluation time of lasso and NPFS on the synthetic data set with increasing size. Clearly, NPFS is a far better performer in terms of the evaluation time as the size of the data set increases, and the size of the data sets being tested are – at the moment – not too large. This figure shows that is difficult to make lasso perform well in a reasonable amount of time with a large data set (i.e., larger than 1GB). Figure 3.10(b) shows NPFS applied on a 100GB data set; however, we did not include lasso because the software implementation would not run on data sets over \( \approx 1\text{GB} \). We observe that NPFS’s runtime is increasing approximately linearly with the size of the data it is processing. In the previous example, lasso saw a large jump in the runtime when the data set became quite large in size. Note that NPFS times could be further improved upon by increasing the number of parallel workers; however, due to software limitations, we must limit the number of parallel processes NPFS can use at once.

We also examined the stability of NPFS in the previous experiment using the Jaccard and Lustgarten indices. Recall that the Lustgarten is a variation of Kuncheva’s

\(^2\)The entire data set is nearly 100GB, which is far larger than the memory on any of our computing nodes.
consistency index [86]; however, Lustgarten’s index does not require that $|A| = |B| = k$, which is assumed by Kuncheva. Figure 3.11 shows the Jaccard and Lustgarten index of NPFS as data are processed. This figure suggests that NPFS can detect the optimal feature set after a very small portion of the data set was processed.

3.4 Summary

In this chapter, we presented a wrapper methodology for validating the selection of $k$ given a feature selection algorithm using the Neyman-Pearson hypothesis test – uniformly the most powerful hypothesis test. There are no assumptions made about the distribution of the data that the base feature selection algorithm would not already be making. The approach is easily integrated with existing feature selection methods, and can be used as a post-hoc test to determine the size of the relevant set given an initial starting point at $k$. We demonstrated, on synthetic data sets, that NPFS is cable of identify the correct number of relevant features even when the base feature
selection method does not select \( k^* \) features for each bootstrap, and that NPFS works well in practice on UCI data sets.

We also examined biasing the NPFS to reduce false positives and a method of early stopping. The primary observations that we found are:

- Early stopping can be quite effective at reducing the number of rounds that NPFS requires; however, the selection of \( \xi \) can degrade the performance of stability indices. Furthermore, we lose the parallelism by needing to check for convergence of \( \|\hat{\mathbf{p}}_t - \hat{\mathbf{p}}_{t-1}\|_1 \).

- Biasing the hypothesis test in NPFS by some amount \( \beta \) can reduce the false positives when \( k \) was chosen larger than \( k^* \). The best suggestion for choosing \( k \) and \( \beta \) would be to set \( k \) small with a small value of \( \beta \). Increasing the bootstraps then tended to detect \( k^* \).

- NPFS can easily be scaled to massive data sets, which may not have been possible to process with traditional feature subset selection algorithms.

Furthermore, we demonstrated NPFS’s feasibility on massive data sets (100GB+) and presented how it fits into the MapReduce computational model.
4. Sequential Learning for Subset Selection

4.1 Introduction

The previous chapter has presented a simple – yet effective – approach for inferring $k^*$ with any generic subset selection algorithm ($A$) that returns $k$ of $K$ features. NPFS uses $A$ to evaluate all of the feature variables at each iteration, and this evaluation could be too computationally intensive. If at all possible we should try to move away from processing the all the features in data at once. Therefore, in this chapter we propose a family of algorithms inspired from bandit literature for processing subsets of the overall feature set at any iteration. In this chapter, we present a new approach to performing subset selection using multi-arm bandits.

4.2 Forming Feature Selection as a Bandit Problem

The problem of selecting $k$ of $K$ features using a generic feature selection algorithm $A$ can be computationally very difficult, and particularly if the software implementation of $A$ has strict limitations on the memory usages, time requirements, or cost of running the algorithm (e.g., RAM-hours). Similar to NPFS, our proposed approach *sequential learning for subset selection* (SLSS), uses a base-subset selection algorithm $A$ to select smaller subsets of features and evaluating the set for importance features. However, how should SLSS select variable subsets such that as the sequential search continues? We eluded to the use of bandits to solve this problem in Chapter 2. We now continue this discussion by introducing a simple bandit algorithm for feature subset selection.

As discussed earlier, the MAB addresses the problem of EvE when a player needs to select a decision (from on of many) that maximizes the reward of the player over
Input: data set \( D \), \( T \) rounds, select \( k \), subset size \( \ell \), weak reward \( \eta \in [0, 1] \), and set weights \( w_j^0 = 0 \) for \( j \in [K] \).

Initialize: \( T_j, r_j = 0 \) for \( j \in [K] \)

Exploration: Explore all features using subsets of size \( \ell \) until all features have been tested (i.e., \( T_j > 0 \)).

1: \( \text{for } t = 1, \ldots, T \text{ do} \)
2: Choose \( \ell \) indices that maximize

\[
\hat{w}_j^t = w_j^t + \sqrt{\frac{2 \log t}{T_j}}
\]  

Refer to this set of indices as \( \mathcal{K}_t \).

3: \( \mathcal{I}_t = \mathcal{A}(D(\mathcal{K}_t), k) \)
4: \( T_j \leftarrow T_j + 1 \) for \( j \in \mathcal{K}_t \)
5: \( r_j \leftarrow r_j + 1 \) for \( j \in \mathcal{I}_t \)
6: \( r_{j'} \leftarrow r_{j'} + \eta \) for \( j' \in \mathcal{K}_t \setminus \mathcal{I}_t \)
7: Update weights \( w_j^{t+1} \) for \( j \in \mathcal{K}_t \)
8: \( \text{end for} \)

Figure 4.1: Pseudo code for the SLSS-UCB1

4.3 Sequential Learning for Subset Selection

4.3.1 SLSS Description

There are three primary implementations of the SLSS family of feature selection algorithms, which are covered in detail in this section. Figure 4.1 shows the pseudo code for SLSS-UCB1. As indicated by its name, SLSS-UCB1 uses the UCB1 confidence term to select features to test (see [1]). The key difference between UCB1 and SLSS is that UCB1 traditionally selects only one arm. In our setting, multiple arms (i.e., features) are going to be evaluated in each learning round. SLSS-UCB1 holds...
a set of weights, $w^t_j$, over the features $j \in [K]$ at learning round $t$. The variables $T_j$ and $r_j$ represent the total number of times feature $j$ was evaluated (i.e., tested), and its cumulative reward, respectively. Both of these variables are initialized to zero.

Then for $T$ rounds, SLSS-UCB1 begins sorting the weights in (4.1) then adding the features with the $\ell$ largest values to a set $\mathcal{K}_t$. (4.1) is the upper confidence term from UCB1 and it controls the EvE. Next the base algorithm $\mathcal{A}$ performs feature selection on the subset of features in $\mathcal{K}_t$ (i.e., feature selection is not performed using all $K$ features). The features $\mathcal{A}$ selects from $\mathcal{K}_t$ is denoted by $\mathcal{I}_t$ (i.e., $\mathcal{I}_t \subset \mathcal{K}_t$). A Bernoulli-style reward is assigned to the features in $\mathcal{K}_t$, such that the features that $\mathcal{A}$ selected as important receive reward “1”, and those features not selected receive reward $0 \leq \eta < 1$ at time $t$. All features tested (i.e., those in $\mathcal{K}_t$) have their $T_j$ incremented. The features $\mathcal{X} \setminus \mathcal{K}_t$ do not have $r_j$, $T_j$ or $w^t_j$ updated. A weak reward is provided to the features tested, but not selected. Thus features that were not selected by $\mathcal{A}$ are provided with some positive feedback just for being considered in the set $\mathcal{K}_t$.

SLSS-UCB1 uses a stochastic bandit for the choosing features to test in $\mathcal{K}_t$, which assumes that the rewards on each of the arms are sampled from a fixed probability distribution. However, since the reward of a feature could depend on the other features in $\mathcal{K}_t$, the assumptions of the stochastic bandit could be too strict. Therefore, we evaluate an adversarial bandit, which does not assume the reward distribution to remain fixed. Figure 4.2 shows the pseudo code for SLSS-Exp3. The adversarial MAB using the Exp3 style selection rule [131], which chooses the $\ell$ largest features from:

$$\hat{p}^j_t = (1 - \gamma) \cdot \left( \frac{w^t_j}{\sum_{i=1}^{K} w^t_i} \right) + \frac{\gamma}{K}$$

where $\gamma \in [0, 1]$ is an egalitarian factor that interpolates between a uniform dis-
**Input:** data set $D$, $T$ rounds, select $k$, subset size $\ell$, weak reward $\eta \in [0,1]$, Exp3 parameter $\gamma \in [0,1]$, and set weights $w^t_j = \frac{1}{K}$ for $j \in [K]$.

**Exploration:** Explore all features using subsets of size $\ell$ until all features have been tested.

1: **for** $t = 1, \ldots, T$ **do**
2: Choose $\ell$ indices that maximize

$$\hat{p}_j = (1 - \gamma) \frac{w^t_j}{\sum_{i=1}^{K} w^t_i} + \frac{\gamma}{K} \quad (4.2)$$

Refer to this set of indices as $\mathcal{K}_t$.

3: $\mathcal{I}_t = A(D(\mathcal{K}_t), k)$
4: Set:

$$w^{t+1}_j = w^t_j \exp \left( \frac{\gamma r^t_j}{\ell} \right) \quad (4.3)$$

5: **end for**

**Figure 4.2:** Pseudo code for the SLSS-Exp3

---

**Input:** data set $D$, $T$ rounds, select $k$, subset size $\ell$, set success/failures $s_j, f_j = 0$ for $j \in [K]$, and choose $\alpha$ & $\beta$ for a Beta distribution.

**Exploration:** Explore all features using subsets of size $\ell$ until all features have been tested.

1: **for** $t = 1, \ldots, T$ **do**
2: Set

$$\theta_j = \text{Beta}(s_j + \alpha, f_j + \beta) \quad (4.4)$$

Choose $\ell$ indices of the largest of values as $\mathcal{K}_t$.

3: $\mathcal{I}_t = A(D(\mathcal{K}_t), k)$
4: $s_j \leftarrow s_j + 1$ \hspace{1cm} for $j \in \mathcal{I}_t$
5: $f_{j'} \leftarrow f_{j'} + 1$ \hspace{1cm} for $j' \in \mathcal{K}_t \setminus \mathcal{I}_t$
6: **end for**

**Figure 4.3:** Pseudo code for the SLSS-Thompson
tribution, and the probability distribution formed by the weights. The traditional Exp3 algorithm only would consider the feature with the largest value, not the $\ell$ largest as SLSS-Exp3 does. This term tunes Exp3’s desire to uniformly choose features at random or use the weight distribution. The updated weights then become $w_{j}^{t+1} = w_{j}^{t} \exp \left( \gamma \frac{r_{j}^{t}}{t} \right)$ for the $j \in K_t$ indices, where $r_{j}^{t}$ is the reward assigned to the $j$th feature evaluated at time $t$.

Chapelle and Li demonstrated that Thompson sampling, which is an old heuristic for balancing exploration & exploitation, can be a highly competitive compared to other bandit approaches [132]. Therefore, we also implemented an SLSS implementation of Thompson sampling for feature subset selection, whose pseudo code can be found in Figure 4.3. The implementation is similar to SLSS-UCB1 and SLSS-Exp3; however, the set $K_t$ is determined by sampling from a Beta distribution with parameters $\alpha$ and $\beta$. The rewards assigned as a success ($s_j$) or failure ($f_{j'}$), which can be viewed as a Bernoulli random variable. As Chapelle and Li point out, Thompson sampling configures different features to use different Beta distributions as their priors (i.e., the prior are controlled by the successes and failures of a feature being selected).

Finally, we also implemented an $\epsilon$-Greedy search, which we refer to as SLSS-$\epsilon$. This implementation selects a random set of indices $K_t \subset \{1, \ldots, K\}$ with probability $\epsilon$, and exploits the $\ell$ maximizing weights with probability $1 - \epsilon$. We can reduce the level of unnecessary exploration by implementing annealing, which sets $\epsilon_t = \kappa^t \epsilon$ for $\kappa \in (0, 1)$.
**Input:** SLSS feature weights $w^T$ where $w^T(j) = w_j^T$, and number of simulations $M$.

1: for $m = 1, \ldots, M$ do
2: Choose $I_m$ to be a bootstrap sample with replacement from indices $[K]$ using $w^T$
3: $k_m^* = \text{Unique}(I_m)$
4: end for

**Output:** Size of feature importance set

$$\hat{k}^* = \left\lceil \frac{1}{M} \sum_{m=1}^{M} k_m^* \right\rceil,$$ (4.5)

or the floor operator can be used in place of the ceiling operator.

---

4.3.2 Selecting Features with SLSS

The SLSS family of feature subset selection approaches output a set of reward weights over the features, $w^T$ where $w^T(j) = w_j^{T1}$; however, we still need to convert a set of weights to the selection of a feature subset. To produce a feature set, we propose a feature sampling algorithm in Figure 4.4. The algorithm begins normalizing the weights, $w^T$, such that they form a distribution. Then a bootstrap sample of size $K$ is drawn with replacement from the set $\{1, \ldots, K\}$, where each feature is weighted using the normalized weights, which represent the rewards given by the MAB algorithm. Hence, higher rewards yield a higher probability of being sampled. $k_m^*$ is the number of unique indices in the bootstrap sample of indices, where $m$ denotes the $m$th sample drawn. The relevant feature set size is given by (4.5), which is the average unique set size from the bootstrap samples. The important features are the chosen as the $\hat{k}^*$ indices with the largest reward.

---

Note that $T$ here refers to the number of iterations, and is not a transposition (which is denoted with $T$ instead)
We also implemented an alternate procedure to that described in Figure 4.4 to increase the robustness of the feature selection phase of SLSS by seeking to decrease the number of false positives. The weights are preprocessed by reducing the weight of a feature that are less than \( w_j^T + \hat{\sigma}_w^T \), where \( \hat{\sigma}_w^T \) is the sample standard deviation of the weights. Then the sampling procedure continues as shown in the algorithm in Figure 4.4.

To demonstrate the capabilities and effectiveness of the SLSS feature selection algorithm, we provide two simulations of rewards from SLSS and show how Figure 4.4 can be used to detect important features. A random 50-dimensional vector is sampled from \([0, 1]^{50}\), and a logistic sigmoid is applied to give some features inherently more relevance than others. We then provide this vector to our feature important sampling algorithm in Figure 4.4. Figures 4.5(a) and 4.5(c) show the rewards for the two simulations with the detection approach with and without the weight reduction, respectively. The features that are detected as relevant by SLSS are colored in red, while irrelevant features are colored in blue in Figures 4.5(a) and 4.5(c). Figures 4.5(b) and 4.5(d) show the distribution of the expected set size after running the selection algorithm 1000 times. We observe in the simulation that the feature set size is relatively consistent, and that using the ceiling or floor operator in Figure 4.4 has little effect on the set size outcome. Furthermore, the variances of the expected set size is significantly reduced by performing the set size reductions.

### 4.3.3 Scaling to Large Data Sets

The implementation of SLSS, while capable of processing relatively large data sets, cannot feasibly be evaluated on data sets with a large number of observations. The reasons for this is that the bootstrap data sets are the same size as the original data set, which may not be feasible due to strict memory limitations. To get around this
Figure 4.5: Two simulations of the feature importance detection algorithm in Figure 4.4 applied to synthetic rewards from SLSS. The bottom row implements the weight reduction preprocessing step described in Section 4.3.2.
issue of large sample sizes, one could *aggressive subsample* the observations, which is suggested in for estimating a bandits reward function [77]. The aggressive subsample is, unfortunately, very heuristic. To avoid this heuristic, Kleiner et al. presented a procedure for performing a bootstrap with a very large data set [133] (a.k.a, the big data bootstrap), while being supported by sound theory and statistical correctness.

Figure 4.6 shows an implementation of SLSS with the bag-of-little bootstraps (BLB), as described below. A large data set \( D \) is subsampled from the observations with replacement to form \( M \) randomly sampled data subsets. SLSS is called \( r \) times, using bootstrap samples, on each of the bootstrap subsamples from \( D \). The results from SLSS applied to each of these \( r \) bootstraps reduces to a statistic \( \xi_v \) for \( v \in [M] \). The statistic \( \xi_v \) contains a point estimate for the rewards for each feature from SLSS, along with a set of error bars to denote the confidence in the prediction. The \( M \) statistics composed of \( \xi_v \) are reduced to a single statistic \( \xi^* \), which has a point estimate and a set of final error bars. Its important to note that this process is *extremely* parallelizable.
4.3.4 Theoretical Analysis

SLSS-UCB1 and SLSS-Exp3 can have their regret against an optimal policy upper bounded, and show that the regret is sublinear with $T$. To motivate the relationship between the regret for SLSS-UCB1 and the traditional UCB1 algorithm let $f(w) \in [0,1]$ be a reward function. Assume that after running SLSS-UCB1 there are $L$ features selected from the feature indices $\{1, \ldots, K\}$, where $L < K$. Thus there are $\binom{K}{L}$ possible feature sets that SLSS-UCB1 could choose. We can then view SLSS-UCB1 as a UCB1 bandit with $\binom{K}{L}$ arms. Given this connection, we now bound the regret of SLSS-UCB1 and SLSS-Exp3 using existing works in the literature on multi-arm bandits. The regret of SLSS-UCB1 and SLSS-Exp3 can be examined by generalizing the results on regret in [1,134].

**Theorem 4.3.1** (Regret of SLSS-UCB1) Define an optimal reward to be $f(w_{true}^*) = 1_L w_{true}^*$, where $w_{true} \in [0,1]^K$ are the optimal weights, $1_L \in \{1\}^L$ is a vector of ones, and $w_{true}^*$ to be a vector containing only the optimal reward values in the set $I^*$. Define $\Delta_i = f(w_{true}^*) - f(w_{true}^I_i)$ to be the difference in rewards for the maximum reward and the reward of a set of features $I_i$. Then the regret of SLSS-UCB1 is upper bounded by:

$$
\left(1 + \frac{\pi^2}{3}\right) \cdot \left(\sum_{i=1}^{\binom{K}{L}} \Delta_i + \sum_{i: f(w_{true}^I_i) < f(w_{true}^*)} \frac{8 \log(T)}{\Delta_i}\right)
$$

Furthermore, we can examine similar bounds for SLSS-Exp3.

**Theorem 4.3.2** (Regret of SLSS-Exp3) For SLSS-Exp3 being run with $\gamma \in [0,1)$ and any stopping time $T > 0$, then the regret of SLSS-Exp3 feels a regret no greater than

$$
T \gamma (e - 1) 1_L w_{true}^* + \frac{K' \log K'}{\gamma}
$$
where $K' = \binom{K}{L}$.

Unfortunately theorems 4.3.1 and 4.3.2 are purely analytical interpretations since $\mathcal{I}^*$ is never actually known. Furthermore, Exp3 is not sublinear in regret, because the approach is suited for an adversarial setting, which can be important if SLSS is applied to nonstationary data streams [135].

The theoretical pseudo-regret and the empirical regret are evaluated for a synthetic data sets of a relatively small $K$ because of the $\left(\begin{array}{c} K \\ L \end{array}\right)$ term. Figure 4.7 shows the theoretical versus empirical regret of SLSS-UCB1. The solid lines represent the theoretical upper bound on regret for different choices of $K$ and the dotted lines represent the empirical regret terms. First, the regret can be calculated for this problem because the data are synthetic, and we precisely know the optimal solution (i.e., policy). Second, we choose to examine $\log(\sum_t \text{regret}(t))$ for reasons of presentation. It can be clearly observed that the upper bound on regret is quite loose; however, we do observe that the empirical regret follows the behavior of the theoretical regret.

4.4 Experiments

We now present an empirical analysis of the SLSS family feature selection algorithms, and compare our approaches to other feature selection approaches. The experiments include a number of carefully designed synthetic data sets, as well as real-world problems. The synthetic data sets are particularly useful when we want to understand the behavior of a feature selection approach under very specific conditions and characteristics of a data set. Such data can allow us to demonstrate the strengths and weaknesses of an approach, which can not as easily be performed on real-world data. We also evaluate SLSS approach with and without the BLB to show the importance of parallelization with SLSS on large data sets. Furthermore, we have evaluated SLSS and other widely used feature subset selection approaches on several
real-world data sets, some of which are commonly used as benchmarks and others that are collected from the domain of microbial ecology.

4.4.1 Description of Synthetic & Real World Data

For synthetic data, $M$ observations are generated with features that are independently and identically distributed (iid) uniform random variables in the interval $[0,10]$. Each feature vector $x_m, m \in [M]$ has $K$ features. The true labeling function, unknown to any algorithm, is given by,

$$y_m = \begin{cases} 1, & \sum_{j=1}^{K} \alpha_j x_m(j) \leq \nu \\ 0, & \text{otherwise} \end{cases}$$

where $\alpha_j, \nu \in \mathbb{R} \forall j$. Given this formulation of $y_m$ we may use the weights $\alpha_j$ to control the importance level of a feature. For example, if $\alpha_j = 1$ for $j \in [k^*]$ and
all other values of $\alpha_i$ equal zero then only the first $k^*$ features carry information for determining the label $y_m$ of a feature vector $x_m$ and the remaining $K - k^*$ features are irrelevant. Our goal is to identify those features (indices $i \in [k^*]$) that are relevant to the classification problem. In this setting if $\nu = 5k^*$ then such a threshold sets the prior probability on each of the classes to approximately $\frac{1}{2}$ for a random sample of size $m$. This can be determine by observing that the threshold is the sum of $k^*$ iid random variables with mean 5. Thus, by linearity of expectation the mean for the sum would be $5k^*$. Furthermore, the synthetic data allows us to choose $\alpha_j$ however we want. We also experiment with the coefficients $\alpha_j$ forming a logistic sigmoid as shown in Figure 2.1.

Most of the real-world data are collected from the UCI machine learning repository [128], and the original mRMR manuscript [52]. Table 4.1 contains properties of the UCI data sets and data sets listed below.

Furthermore, we also evaluate data collected from the American Gut Project (AGP)$^2$, which is the study of the human microbiome. The AGP data are represented by the abundance of thousands of bacterial species (i.e., species) detected in fecal samples collected from individuals. The data are collected from 1900+ people of different genders, diet types, and age groups. We extracted 24k+ taxa abundances as the features and the diet type is label for each sample. The diets are categorized as either omnivore or vegetarian. Another microbial study of the gut microbiome was collected by Caporaso et al. [101], and used in our evaluation. With this data, there are 15k+ taxa and the gender of the sample is used as the class label. The data for both the Caporaso’s study and the AGP studies were preprocessed by removing low abundance taxa, and binning the features to allow us to compute information-theoretic quantities for the base-subset selection algorithm used by SLSS.

$^2$http://americangut.org/
Table 4.1: Properties of data sets used in our experiments.

<table>
<thead>
<tr>
<th>data set</th>
<th>instances</th>
<th>features</th>
</tr>
</thead>
<tbody>
<tr>
<td>Caporaso (Cap)</td>
<td>467</td>
<td>1193</td>
</tr>
<tr>
<td>American Gut (AG)</td>
<td>2905</td>
<td>3110</td>
</tr>
<tr>
<td>Colon</td>
<td>62</td>
<td>2000</td>
</tr>
<tr>
<td>Lung</td>
<td>73</td>
<td>325</td>
</tr>
<tr>
<td>NCI9</td>
<td>60</td>
<td>9712</td>
</tr>
<tr>
<td>Leuk</td>
<td>72</td>
<td>7070</td>
</tr>
<tr>
<td>Sido0</td>
<td>12678</td>
<td>4932</td>
</tr>
</tbody>
</table>

4.4.2 Algorithms for Comparison & Figures of Merit

We use several state-of-the-art algorithms to compare their ability to learn from moderate to very high dimensional data, and we also evaluate data sets with a very large cardinality. We have considered two standard information-theoretic subset selection approaches, and a more recent approach to address data sets with a large number of observations. We evaluate the following feature subset selection approaches:

- **Base**: no feature selection is performed, and a classifier is trained using all features.
- **minimum Redundancy Maximum Relevancy (mRMR)**: Information-theoretic approach that maximizes the mutual information function, $I(X_j; Y)$, with a scaled penalization on redundancy, $I(X_j; X \in \mathcal{F})$ [52].
- **Mutual Information Maximization (MIM)**: Information-theoretic approach that maximizes the mutual information function, $I(X_j; Y)$ [51].
- **Neyman-Pearson Feature Selection (NPFS)**: Similar to SLSS, NPFS uses a base-subset selection approach then applies a hypothesis test to detect the relevant
feature set size [15]. However, NPFS needs to evaluate the entire feature space, whereas SLSS evaluates subspaces.

- \textit{SLSS-}\{\epsilon, \text{UCB1, Thompson}\}: The parameters \( \ell = \lceil K/8 \rceil, k = \lceil K/16 \rceil, \epsilon = \frac{1}{10} \) and \( \eta = \frac{1}{10} \) are fixed for all data sets, and the base scoring algorithm (\( A \)) is MIM. The greedy search is annealed.

While there are several parameters in SLSS, we argue that many of them are straightforward to choose and have little effect with a large \( T \).

We use a distance weighted 5-NN classifier with the Euclidean distance, and CART. All experiments are evaluated using 5-fold cross validation. Our implementations of mRMR and MIM are in the FEAST feature selection toolbox [11]. The 1-0 classification error and f1-measure, which is given by:

\[
\text{f1} = \frac{2 \times \text{precision} \times \text{recall}}{\text{precision} + \text{recall}}
\]

are used as figures of merit for all classification tasks [89].

The Lustgarten index is used to measure the stability of a feature selection algorithm [87], which was designed to expand Kuncheva’s stability index [86]. For two sets, \( A \) and \( B \) that have are of size no greater than \( K \), then the Lustgarten index is given by

\[
L(A, B) = \frac{|A \cap B| - \frac{|A||B|}{K}}{\min(|A|, |B|) - \max(0, |A| + |B| - K)}
\]

which lies in the range \( \pm 1 \). For the purposes of our experiments, \( A \) is the optimal feature set and \( B \) is the feature set that SLSS selects. Ideally, we would want \( L(A, B) = 1 \). This measure can only be applied on the synthetic data since the optimal feature set is unknown for real-world data sets. The Jaccard index was not evaluated.
because of how rapidly the index drops off and becomes ineffective at coping with a reasonable number of false positives [85].

4.4.3 Examination of Regret

In our first experiment, we evaluate the regret of SLSS, and its variants, on the synthetic data described above. Note that other approaches such as NPFS and mRMR cannot be evaluated for regret for several reasons: (i) SLSS’s regret can be evaluated as a function of $T$ whereas other approaches do not work in learning rounds, and (ii) some of the state-of-the-art approaches require that $k$ be fixed in its final selection of features and, hence, cannot adapt to an optimal size. We use synthetic data because we can compute the reward from the optimal policy, and set $\alpha_j = 1$ for $j \in [k^*]$ and all other weights to zero. The regret on the $T$th round is measured by

$$\text{regret}(T) = 1_L^T w_{1:L}^{\text{true}} - 1_L^T w_{T}^{\text{true}}$$

where $w_{1:L}^{\text{true}}$ is an $L$-dimensional vector of the optimal rewards, $w_{T}^{\text{true}}$ is an $L$-dimensional vector of the rewards chosen by SLSS, and $1_L$ is an $L$-dimensional vector of ones. The value of $L$ is not a parameter of SLSS, rather it is only used to compute the regret. For this experiment we set $T = 5000$, $k = 5$, $\ell = 10$, $\eta = \frac{1}{10}$, $L = 25$, and $K = 100$, and we evaluate the cumulative regret. The cumulative regret for four values of $k^*$ is shown in Figure 4.8. Before examining the regret, observe that as $k^*$ increases the optimal feature set is becoming less sparse (i.e., more features are relevant). We examine $L$ that there are setting when the selection of $L \leq k^*$ and $L > k^*$.

We observe a clear sub-linear cumulative regret in Figures 4.8(a) and 4.8(a), which occur when $k^*/K$ is small. We have omitted the results of SLSS-Exp3 because it is known from Theorem 4.3.2 that the cumulative regret will not be sub-linear because of the $T$ term. We observe that for both SLSS-UCB1 and SLSS-Thompson that the
Figure 4.8: Cumulative regret of SLSS-$\epsilon$, SLSS-UCB1, and SLSS-Thompson evaluated on a synthetic data set with $T = 5000$, $k = \lfloor K/16 \rfloor$, $L = 25$, $\ell = \lfloor K/8 \rfloor$, $\eta = \frac{1}{10}$, and $K = 100$. We choose the optimal reward to be the sum of the top 25 feature weights (i.e., $k^*/K$). We observe that when $k^*$ are close to each other that the per round regret approach zero quickly. The shaded region represents a 95% confidence interval around the mean cumulative regret.

Figure 4.9: Extension of the average cumulative regret experiment from Figure 4.8 for SLSS-$\{\epsilon, UCB1, Thompson\}$ evaluated in varying $k^*$ when $T = 1000$. The solid lines represent $L = 25$ and the dashed lines represent $L = 40$. 
feature importance vector is approaching the optimal reward vector quite quickly. Furthermore, the larger regret for SLSS-$\epsilon$ can be explained by the search policy randomly searching the feature space with probability $\epsilon$. Though one way to attempt to improve SLSS-$\epsilon$ would be to perform annealing of the exploration rate. However, for situations where the optimal feature set does not have a small $k^* / K$, i.e., where most of the features are relevant, the sub-linear behavior of SLSS is more difficult to show even after a large $T$ (see Figures 4.8(c) and 4.8(d)). Having observed this, we know that changing $k$ and $\ell$ can further improve the regret if chosen with knowledge of $k^*$; however, such inference is commonly unknown. These results suggest that SLSS works best in situations where the optimal feature subset is quite small, which is not a very limiting assumption. Furthermore, we observe that SLSS-$\epsilon$ consistently obtains a higher regret than SLSS-Thompson and SLSS-UCB1. Another observation that is observed from the cumulative regret is that the regret is generally growing with increasing $k^*$ in Figures 4.8(a), 4.8(b), and 4.8(c); however, the cumulative regret drops off in Figure 4.8(d).

This change in regret can be explained by examining the cumulative regret for a fixed $T$ and $L$, which is shown in Figure 4.9. Two selection of $L$, 25 & 40, are represented by solid and dashed lines, respectively. There is a clear peak in the maximum regret at $k^* = L$, which is semi-intuitive if we consider the experiment. Recall that all relevant features carry the same weight, so if $k^*$ is less than $L$ then SLSS only needs to select the $k^*$ relevant indices and the remaining indices selected carry no weight in the regret measurement. Furthermore, if $k^*$ is large than it becomes more likely (and easier) for SLSS to identify a relevant features because it only needs to identify $L$ of the $k^*$ features to incur no regret. The most difficult selection happens at $k^* = L$ because SLSS must select its feature set to be precisely the optimal one.
4.4.4 Results on Synthetic Data

We now examine SLSS and its variants on different configurations of the synthetic data set discussed in section 4.4.1. We designed our experiments to seek answers to the following key questions:

1. Does $w^T$ converge after $T$ rounds of learning; and how does the overall level of feature importance affect the convergence?

2. How does the dimensionality affect the performance of SLSS, particularly if we make a naïve choice of $\ell$ and $k$? How does SLSS react to the scenario of an increasing dimensionality with a decrease ratio of $k^*/K$?

3. How stable is SLSS’s feature selection using the algorithm in Figure 4.4?

We first evaluate the convergence of $w^T$. To do so, we examine two methods for determining the labels of the synthetic data. In the first approach, $\alpha_j = 1$ for $j \in [k^*]$
and all other values of $\alpha_i$ equal to zero for $i \in [K] \setminus [k^*]$. Hence, features either carry importance, or not (i.e., a given feature is either (strongly) relevant or not relevant at all, as discussed in Section 2.1.1). In the second approach $\alpha_j$ is a logistic function similar to that shown in Figure 2.1, where features contain different levels relevance. The SLSS implementations have $T = 500$, $\ell = 25$, $M = 5000$, and $k = 10$. The data set has 500 features; however, only 50 features are relevant (i.e., $K = 500$ and $k^* = 50$).

Figure 4.10 shows the convergence of SLSS’s feature weight vector over time for the first synthetic data where each feature is relevant or non-relevant. For each graph, each column is a learning round and each row is a feature. The plot shows the log($w^t$) to better highlight the differences between the SLSS variants. Let us begin by examining the top row of Figure 4.10. The key observation is that each of the SLSS implementation favors the first 50 features (see Figures 4.10(a), 4.10(b), 4.10(c), & 4.10(d)) by giving them a larger weight than any of the other features. Recall from (4.6) that the first 50 features are indeed the relevant ones. A second observation from the top row is that SLSS-$\{\epsilon, UCB1, Thompson\}$ learns the true relevant features in fewer iterations than SLSS-Exp3, which Exp3 is an adversarial bandit is unlike $\epsilon$-greedy, UCB1, and Thompson sampling.

The bottom row of Figure 4.10 shows the convergence of SLSS’s feature weight vector over time for the synthetic data using a logistic function to determine the importance of a feature (approximately the first 200 features carry a high or moderate weight). We observe, again, that with the exception of the Exp3 bandit that the SLSS variants quickly learn the importance distribution, with SLSS-Thompson providing a very clear transition from strong to weak relevance.

We now move on to address the 2nd and 3rd questions proposed in this section. To address these questions jointly, we designed a synthetic data set with $k^* = 25$ and
$K \in \{100, 200, \ldots, 2000\}$. In such a setting, the ratio of the optimal feature set size to the total number of features is becoming increasing smaller, and would evaluate the ability of SLSS to identify the optimal feature set even if the selection of $k$ and $\ell$ inaccurate. For SLSS, we choose the heuristic of $k = \lfloor K/16 \rfloor$, and $\ell = \lfloor K/8 \rfloor$. Figure 4.11 shows the stability of SLSS (via the Lustgarten index), and the average set size determined by SLSS’s sampling procedure. Figure 4.11(a) has the lowest Lustgarten index of all the SLSS algorithms, which is caused by SLSS-Exp3 giving larger weights to features that are not relevant (see Figure 4.11(b)). Another important note is that SLSS almost identifies the correct feature set size using the sampling procedure, and there are only a few incorrectly selected variables. Notice that using the heuristic of choosing $k$ and $\ell$ worked quite well until about $k = 500$, where only 5% of the features are relevant. SLSS-Exp3 tends to select more features than the other SLSS approaches, which explains why SLSS-Exp3 has the worst Lustgarten index. False positives are unavoidable and they’ll generally increase with the total number of features (similar results were found in [15]); which explains the slight decrease in the
stability. It is important to note here that we are focusing on problems where an overwhelming number of features are irrelevant or very weakly relevant.

### 4.4.5 Evaluation with the Bag-of-Little Bootstraps

We now empirically evaluate SLSS with the bag-of-little-bootstraps implementation to run on a large synthetic data set (see Section 4.4.1 with $M = 150k$ and $K = 4000$). We evaluated the proposed approach on a single cluster node with 64-cores and 256GB of memory. For the implementation without the BLB, SLSS is sequentially evaluated on sequentially chunks of data. Thus, SLSS can be used in incremental learning problems where the importance of a variable can be learned over multiple batches of data.

We monitor the mean-squared error (MSE) between the optimal weight vector and the weight vector produced by SLSS and SLSS-BLB, and the total time of evaluation. Figure 4.12, which was motivated from an evaluation of the BLB in [133], shows the evaluation time versus MSE for SLSS and SLSS-BLB. We find that SLSS-BLB produces a smaller error in a fraction of the time, which can be further sped up with access to more cores on a physical computing node.

### 4.4.6 Results on Real-World Data

Unlike the synthetic data sets, we do not know the correct number of relevant features for the real-world data. Therefore, we cannot use the Lustgarten index as we did in the previous experiments. In this section we evaluate the SLSS and the state-of-the-art approaches with two different classifiers.

Table 4.2 presents the proportion of features that are being selected by the algorithms. We only present the algorithms that are capable of identifying the relevant feature set size. We observe that SLSS-\{UCB1, $\epsilon$, Thompson\} select a smaller per-
Figure 4.12: Performance, measured in MSE, of SLSS and SLSS-BLB evaluated on a synthetic data set. We only show a single point for SLSS-BLB because it can be fully parallelized and the MSE can be evaluated after all subsamples of $\mathcal{D}$ are processed.

The error and f1-measure are averaged from 5-fold cross validation. The averages are ranked so that we can make comparisons among algorithms. The ranks can range from (1) to (8), where (1) is the best and (8) is the worst performing algorithm. We then use the Friedman test as described in [130], which has the null hypothesis that all algorithms are performing equally, to make formal statistical comparisons between classifiers over multiple data sets.

Table 4.3 shows the ranks averaged over all of the data sets for the 5-NN and CART (the full error and f-measure tables can be found in Tables 4.4 and 4.5, respectively). For the 5-NN, we find that the Friedman test rejects the null hypothesis that the error rates are equal; however, there is not enough statistical significance to state there is a difference in the f1-measure. Using Table 4.3 and apply the Friedman test, we find that...
Table 4.2: Average percentage of features selected by algorithms that have the ability to approximate the relevant set size.

<table>
<thead>
<tr>
<th>data set</th>
<th>NPFS</th>
<th>UCB1</th>
<th>Thompson</th>
<th>$\epsilon$-greedy</th>
<th>Exp3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cap</td>
<td>11.5</td>
<td>6.9</td>
<td>6.7</td>
<td>6.6</td>
<td>38.5</td>
</tr>
<tr>
<td>AGP</td>
<td>17.1</td>
<td>11.9</td>
<td>11.3</td>
<td>10.2</td>
<td>30.5</td>
</tr>
<tr>
<td>Colon</td>
<td>16.6</td>
<td>11.5</td>
<td>10.8</td>
<td>9.7</td>
<td>17.9</td>
</tr>
<tr>
<td>Lung</td>
<td>19.4</td>
<td>14.9</td>
<td>13.4</td>
<td>12.9</td>
<td>21.3</td>
</tr>
<tr>
<td>NCI</td>
<td>19.6</td>
<td>13.5</td>
<td>13.1</td>
<td>12.1</td>
<td>20.7</td>
</tr>
<tr>
<td>Leuk</td>
<td>16.6</td>
<td>11.2</td>
<td>10.2</td>
<td>9.2</td>
<td>19.1</td>
</tr>
</tbody>
</table>

the null hypothesis is rejected for both the CART and 5-NN classifiers. Upon further investigation for the 5-NN, we find that SLSS-Thompson is outperforming base, MIM and mRMR, and other improvements are observed with SLSS-UCB1 (mRMR), and SLSS-$\epsilon$ (base, mRMR, Exp3). For CART, significant improvements are observed with the error of SLSS-UCB1 (base, MIM and NPFS), SLSS-Thompson (base and NPFS), and SLSS-$\epsilon$ (base). It is worth noting that the implementation of mRMR did not run due to the size of the data set. This is one of the key advantages of SLSS as it does not need to evaluate the entire feature set, rather, process small regions of the feature space at a time.

We now take a closer look at the results of SLSS on data collected from the American Gut Project and the Caporaso data. Figure 4.13 shows the weights of SLSS-{$\epsilon$, UCB1, Thompson}. The x-axis represents indices corresponding to a particular bacteria, which have been sorted in descending order according to the UCB1 weights. We notice that bacteria are being approximately ranked in the same order. If we order the weights in increasing order (i.e., weight attached to a particular bacteria) we find that a linear fit achieves $R^2 = 0.9997$, which demonstrates that the ranking of the bacteria are in very high agreement between the 3 different SLSS algorithms. Taking a closer look at the bacteria being highly ranked, we find that the families
Table 4.3: Rank of an algorithm’s performance when averaged across the data sets shown in Table 4.1 (lower is better).

<table>
<thead>
<tr>
<th>algorithm</th>
<th>5-NN error</th>
<th>f1</th>
<th>CART error</th>
<th>f1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base</td>
<td>6.21</td>
<td>5.86</td>
<td>5.86</td>
<td>6.57</td>
</tr>
<tr>
<td>MIM</td>
<td>4.93</td>
<td>4.00</td>
<td>5.21</td>
<td>4.14</td>
</tr>
<tr>
<td>mRMR</td>
<td>6.21</td>
<td>4.93</td>
<td>3.50</td>
<td>4.29</td>
</tr>
<tr>
<td>NPFS</td>
<td>5.14</td>
<td>6.14</td>
<td>5.86</td>
<td>5.43</td>
</tr>
<tr>
<td>SLSS-UCB1</td>
<td>3.71</td>
<td>4.79</td>
<td>3.00</td>
<td>3.57</td>
</tr>
<tr>
<td>SLSS-Thom</td>
<td>2.36</td>
<td>2.93</td>
<td>3.50</td>
<td>4.14</td>
</tr>
<tr>
<td>SLSS-(\epsilon)</td>
<td>3.29</td>
<td>3.43</td>
<td>3.50</td>
<td>2.14</td>
</tr>
<tr>
<td>SLSS-Exp3</td>
<td>4.14</td>
<td>3.93</td>
<td>5.57</td>
<td>5.71</td>
</tr>
</tbody>
</table>
Table 4.4: Classification error rates on the real-world data sets.

<table>
<thead>
<tr>
<th>data set</th>
<th>Base</th>
<th>MIM</th>
<th>mRMR</th>
<th>NPFS</th>
<th>SLSS-UCB1</th>
<th>SLSS-Thom</th>
<th>SLSS-$\epsilon$</th>
<th>SLSS-Exp3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Colon</td>
<td>35.1</td>
<td>21</td>
<td>22.4</td>
<td>35.5</td>
<td>22.8</td>
<td>26.2</td>
<td>24.4</td>
<td>20.6</td>
</tr>
<tr>
<td>Leuk</td>
<td>8.2</td>
<td>7</td>
<td>5.4</td>
<td>8.4</td>
<td>4.2</td>
<td>5.5</td>
<td>4.2</td>
<td>11.2</td>
</tr>
<tr>
<td>Lung</td>
<td>48.4</td>
<td>50.7</td>
<td>37</td>
<td>42.7</td>
<td>42.5</td>
<td>42.5</td>
<td>46.9</td>
<td>46.5</td>
</tr>
<tr>
<td>NCI</td>
<td>75</td>
<td>71.7</td>
<td>71.7</td>
<td>68.3</td>
<td>65</td>
<td>63.3</td>
<td>71.7</td>
<td>75</td>
</tr>
<tr>
<td>Cap</td>
<td>1.3</td>
<td>0.6</td>
<td>1.1</td>
<td>1.9</td>
<td>0.2</td>
<td>1.9</td>
<td>0</td>
<td>2.1</td>
</tr>
<tr>
<td>AGP</td>
<td>19.2</td>
<td>19.4</td>
<td>15.7</td>
<td>19.3</td>
<td>18</td>
<td>17.2</td>
<td>17.8</td>
<td>17.7</td>
</tr>
<tr>
<td>Sido0</td>
<td>3.6</td>
<td>3.6</td>
<td>–</td>
<td>3.6</td>
<td>3.6</td>
<td>3.6</td>
<td>3.6</td>
<td>3.6</td>
</tr>
</tbody>
</table>
Table 4.5: Classification F1-measures on the real-world data sets.

<table>
<thead>
<tr>
<th>data set</th>
<th>Base</th>
<th>MIM</th>
<th>mRMR</th>
<th>NPFS</th>
<th>SLSS-UCB1</th>
<th>SLSS-Thom</th>
<th>SLSS-ϵ</th>
<th>SLSS-Exp3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Colon</td>
<td>61.1</td>
<td>74.4</td>
<td>69.2</td>
<td>57</td>
<td>73</td>
<td>70.7</td>
<td>71.4</td>
<td>75.4</td>
</tr>
<tr>
<td>Leuk</td>
<td>90.5</td>
<td>92.2</td>
<td>93.7</td>
<td>90.7</td>
<td>92.5</td>
<td>93.7</td>
<td>94.7</td>
<td>87.3</td>
</tr>
<tr>
<td>Lung</td>
<td>41.3</td>
<td>39.5</td>
<td>57.2</td>
<td>49.6</td>
<td>47.5</td>
<td>46.1</td>
<td>49.6</td>
<td>48.1</td>
</tr>
<tr>
<td>NCI</td>
<td>27</td>
<td>33.5</td>
<td>35.3</td>
<td>28.5</td>
<td>31</td>
<td>28.9</td>
<td>29.1</td>
<td>20.8</td>
</tr>
<tr>
<td>Cap</td>
<td>98.4</td>
<td>99.2</td>
<td>98.6</td>
<td>97.6</td>
<td>99.8</td>
<td>97.6</td>
<td>100</td>
<td>97.3</td>
</tr>
<tr>
<td>AGP</td>
<td>50.6</td>
<td>51.2</td>
<td>51</td>
<td>52.3</td>
<td>53.9</td>
<td>53.9</td>
<td>54.4</td>
<td>52.9</td>
</tr>
<tr>
<td>Sido0</td>
<td>49.1</td>
<td>49.1</td>
<td>–</td>
<td>49.1</td>
<td>49.1</td>
<td>49.1</td>
<td>49.1</td>
<td>49.1</td>
</tr>
</tbody>
</table>
Figure 4.13: SLSS learned feature importance distribution for two data sets collected from microbial ecology.

*Lachnospiraceae, Ruminococcaceae, Erysipelotrichaceae* and *Verrucomicrobiaceae* are commonly ranked with large weights, which indicates a high level of importance. These families are all known indicators of either having – or not having – a vegetarian diet [136, 137], which supports our intuition about how we should expect SLSS to behave on this well understood life science application.

### 4.4.7 Summary

In this work we discussed the difficulties of scaling up state-of-the-art feature subset selection algorithms to data that contain a large number of features and observations, particularly with an emphasis on filter-based methods, which are largely ignored with large-scale subset selection. Large data sets, which are common throughout the fields such as life sciences, raise both theoretical and algorithmic concerns in performing subset selection. Two benefits of the filter approaches is that the scoring function can be evaluated quite quickly, and they are classifier independent. Furthermore, implementing subset selection with multiple classes is rather straightforward with filter-based feature selection approaches, whereas many embedded approaches
assume binary learning problems. However, the issue of long evaluation times can arise for very large feature sets. As a solution, we proposed a family of feature subset selection algorithms, namely sequential learning for subset selection (SLSS), that only need to consider a portion of the feature and observation space in a series of small feature selection tasks. SLSS searches regions of the feature subspace to learn the importance of variables over a fixed number of learning rounds. This strategy is motivated from optimization literature where it is common to break a large optimization problem into smaller optimization problems [138,139]. Such a strategy can greatly improve the applicability of an algorithm in an environment where it is simply infeasible to evaluate the entire feature and observation space simultaneously. SLSS was evaluated on several synthetic settings to understand the dynamics of the approach when applied to problems of varying levels of difficulty. Furthermore, we also evaluated SLSS on real-world data and received promising insights on the data collected from microbial ecology. The results of SLSS applied to the data from microbial ecology agreed with results previously explored in the literature, which makes subset selection an attractive tool for microbial data analytics.

Furthermore, SLSS easily fits into an incremental learning framework that does not require all data to be available at the time of learning. This was demonstrated in Section 4.4.5 where an incremental learning version of SLSS was compared to SLSS-BLB. As expected the incremental approach does not yield a smaller MSE, as batch algorithms tend to perform slightly better than batch or online algorithms [12].
5. Online Feature Selection Ensemble

5.1 Introduction

The research from the previous chapters has focused on the use of filters for subset selection and all the data were assumed to be available at the time of learning. While SLSS is capable of incremental learning, it is not an online learning algorithm (i.e., batches of data are not needed rather only a single instance). One of the key advantages to a pure online algorithm is that only a single data instance needs to be loaded into memory, thus, analyzing a massive data stream (e.g., billions of observations) is straightforward. Furthermore, this chapter focuses on an embedded approach to feature selection rather than a filter-based approach.

Our online learning approach to feature selection addresses three of the five V’s with big data, namely: volume, value and velocity. Both NPFS and SLSS can easily handle extracting value for a large volume of data, however, data that are arriving in a stream (i.e., velocity) are not incorporated into their design. One observation to make about SLSS is that it only examines a portion of the feature space at each learning round, which we refer to as learning with partial information because not all features need to be available. Such a learning setting is useful when there is a large amount of missing data, though most algorithms do not incorporate the possibility that there could be missing data. It is against this background that we develop an ensemble classifier that: (i) implements feature selection, (ii) incorporate new data into the model when it becomes available and (iii) learns a prediction model using partial information.
**Input** Distribution $\mathcal{D}$, learning rule $\text{Update}$, parameter vector $\mathbf{w} \sim \mathcal{N}(0, 1)^{K \times 1}$ for $t = 1, 2, \ldots$

1. Receive $(\mathbf{x}_t, y_t) \sim \mathcal{D}$
2. Update $\mathbf{w} = \text{Update}(\mathbf{w}, \mathbf{x}_t, y_t)$
3. Receive test instance $\mathbf{x} \sim \mathcal{D}$
4. Receive $y$ and measure the loss $\ell(\mathbf{w}^T \mathbf{x}, y)$

Figure 5.1: Online algorithm for learning a linear function with parameters $\mathbf{w}$.

### 5.2 Learning Setting

Our learning setting is formulated as follows (summarized in Figure 5.1 with common mathematical notations provided in Table 5.1): a data pair $(\mathbf{x}_t, y_t) \in \mathbb{R}^K \times \{\pm 1\}$ are presented at some time $t$ that are sampled from a joint distribution $\mathcal{D}$ (i.e., $\mathcal{D} := \mathbb{P}(X,Y)$). The objective is to learn a linear model $\mathbf{w}$ such that it produces a minimal loss on a function $\ell(\mathbf{w}^T \mathbf{x}, y)$, where $\ell(\cdot, \cdot)$ is convex in its first argument. The model $\mathbf{w}$ is adjusted accordingly via an update rule when each new data pair arrive. Since the model is linear, the output is given by $\hat{y} = \text{sign}(\mathbf{w}^T \mathbf{x})$. Feature selection is performed by forcing some, preferably many, of the elements in $\mathbf{w}$ to zero. This approach to learning is an online learning paradigm with delayed labeling because we receive some level of feedback on our predictions (i.e., a loss can be measured) after the prediction is made. Furthermore, unlike traditional online learning algorithms that assume all $K$ features are available when the update is performed, our learning setting assumes that only $B$ feature values are needed for learning ($B < K$). We describe below in detail, on how these $B$ variables are selected.

Finally, we use an ensemble setting, and hence, at each time stamp $t$ we update not one model, but rather a finite set of models. Our motivation is rooted in the
Table 5.1: Mathematical Notations for OFSE

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mathcal{D}$</td>
<td>joint probability distribution for sampling data</td>
</tr>
<tr>
<td>$(x_t, y_t)$</td>
<td>data pair in the space $\mathbb{R}^K \times {-1, 1}$, $(x, y) \sim \mathcal{D}$ at time $t$</td>
</tr>
<tr>
<td>$C_t$</td>
<td>feature subset that is required to be measured in $x_t$</td>
</tr>
<tr>
<td>$\tilde{x}_t$</td>
<td>partial view of $x_t$ containing features in the set $C_t$</td>
</tr>
<tr>
<td>$\hat{x}_t$</td>
<td>scaled version of $\tilde{x}_t$</td>
</tr>
<tr>
<td>$w_t^{(j)}$</td>
<td>$j$th single OFS model in an ensemble with $J$ models at time $t$</td>
</tr>
<tr>
<td>$p$</td>
<td>ensemble model (i.e., linear combination of $w_t^{(j)}$)</td>
</tr>
<tr>
<td>$B$</td>
<td>sparsity level</td>
</tr>
<tr>
<td>$R$</td>
<td>maximum bound on $|w_t^{(j)}|_2$</td>
</tr>
<tr>
<td>$I(\tau)$</td>
<td>1 if $\tau = \text{True}$ otherwise 0</td>
</tr>
<tr>
<td>$[K]$</td>
<td>the set ${1, \ldots, K}$</td>
</tr>
<tr>
<td>$\mathcal{N}(\mu, \sigma)$</td>
<td>Gaussian distribution with parameters $\mu$ and $\sigma$</td>
</tr>
<tr>
<td>$\text{Poisson}(\lambda)$</td>
<td>Poisson distribution with parameter $\lambda$</td>
</tr>
</tbody>
</table>

well-established result that ensemble models under relatively mild conditions, have better generalization performance than a single model [140, 141]. The ensembles are constructed using variants of online bagging and boosting. Furthermore, since we use linear models as the base classifier, the ensemble model is no more complex than any single model. That is, the ensemble prediction is given by:

$$\hat{y} = \sum_{j=1}^{J} x^T \alpha_j w_t^{(j)} = x^T p$$ (5.1)

where $p$ is the ensemble model that is a linear combination of the single models $w_t^{(j)}$. We later show that the sparsity level of $w_t^{(j)}$ and $p$ can be forced to be equal (i.e., $\|w_t^{(j)}\|_0 \leq B$ and $\|p\|_0 \leq B$), however, $p$ results in a smaller number of mistakes.
5.3 Online Feature Selection with Partial Inputs

Online feature selection (OFS) was presented by Wang et al. for learning feature importance via a sparse projection [2]. In their work, they proposed a full and partial information approach for OFS. Full information assumes that all of the feature measurements in $x_t$ are available at the time of training or testing. Partial information assumes that $B$ variables are available in $x_t$, where $B < K$.

Figure 5.2 presents OFS with partial information. OFS requires, as inputs, a limit placed on the $l_2$-norm of $w$ ($R$), a sparsity level ($B$), a search probability ($\tau$), and learning rate ($\eta$) be specified prior to learning. A parameter vector $w_1$ is initialized to a zero-vector (other initializations can be considered). For each time $t$, a Bernoulli random variable is sampled with probability of success $\tau$. If the random variable is a success then $B$ variables are chosen uniformly at random from the set $[K]$, otherwise the non-zero entries in $w_t$ are chosen. Call this set $C_t$ and only require these indices from $x_t$, which we call $\tilde{x}_t$. Let all the other entries $[K] \setminus C_t$ be equal to zero. Hence, $K - B$ variables need not be measured, and depending on the selection of $B$, OFS can work well even when there are a large number of missing variables (i.e., because $K - B$ variables are not required for learning). If the prediction on $\tilde{x}_t$ is correct, then there is no update to $w_t$. However, if the prediction is incorrect then $\tilde{x}_t$ is scaled in (5.2) and a gradient descent update is performed with a learning rate of $\eta$, which the result of the update is then projected onto an $l_2$-ball to bound the Euclidean norm. The vector $w_t$ is then truncated to have at most $B$ non-zero elements. The truncation procedure is shown in Figure 5.3. We refer to lines 4 through 15 of Figure 5.2 as “OFS_Update”.
1: **Input**

- $R$: maximum $l_2$ magnitude
- $B$: OFS truncation parameter
- $\epsilon$: search parameter
- $\eta$: learning rate

2: **Initialization**

- $w_1 = 0$

3: for $t = 1, \ldots, T$ do

4: if Bernoulli($\tau$) == 1 then

5: Randomly choose $B$ attributes $C_t$ from $[K]$

6: else

7: Choose the attributes that have non-zero values in $w_t$, i.e., $C_t := \{i : [w_t]_i \neq 0\}$

8: end if

9: Receive $\tilde{x}_t$ by only requiring the attributes in $C_t$, and $y_t$ from $\mathcal{D}$

10: if $y_t\tilde{x}_t^Tw_t < 0$ then

11: Compute $\hat{x}_t (i \in [K])$

12: \[
[\hat{x}_t]_i = \frac{[\tilde{x}_t]_i}{\frac{B}{R}\epsilon + I([w_t]_i \neq 0)(1 - \epsilon)}
\] (5.2)

13: $\tilde{w}_t \leftarrow w_t + \eta y_t\hat{x}_t$

14: $\hat{w}_t \leftarrow \min \left\{ 1, \frac{R}{\|w_t\|_2} \right\} \tilde{w}_t$

15: $w_{t+1} \leftarrow \text{Truncate}(\hat{w}_t, B)$

16: end if

end for

Figure 5.2: Pseudo code for OFS with Partial Inputs [2]
1: Input
   • \( w \in \mathbb{R}^K \): weight vector
   • \( B \): OFS truncation parameter

2: if \( \|w\|_0 > B \) then
3:   Set \( \hat{w} \) equal to the \( w \) and the \( K - B \) elements with the smallest magnitude equal to zero
4: else
5:   \( \hat{w} = w \)
6: end if
7: Return \( \hat{w} \)

Figure 5.3: Pseudo code for truncating a parameter vector in OFS [2]

5.4 Online Feature Selection Ensembles

In this section, we describe our proposed integration of online feature selection with online bagging and boosting. While, the integration of these two methodologies is rather straightforward, the impact is quite significant. As discussed in the previous section, OFS is a linear prediction approach, i.e., \( \hat{y} = \text{sign}(w^T x) \), where \( w \) is \( B \)-sparse with \( \|w\|_0 \leq B \). By forming an ensemble model of a linear \( B \)-sparse models, the ensemble model can typically have a lower error rate while remaining \( B \)-sparse. Note that we are assuming only partial information is adequate as discussed in Section 5.3. In this section, we limit our discussion to integrating online bagging and boosting mechanisms into an online feature selection framework. Interested readers are encouraged to read Oza’s work for a more detailed explanation and motivation for a general framework of online ensemble algorithms [142].

5.4.1 Algorithm Descriptions
1: **Input**
   - $B$: OFS truncation parameter
   - $R$: maximum $l_2$ magnitude
   - $J$: ensemble size

2: **Initialization**
   - $w^{(j)}_0 \sim \mathcal{N}(0, 1)$ \hspace{1cm} $\forall j \in [J]$
   - $p_0 \sim \mathcal{N}(0, 1)$

3: for $t = 1, \ldots, T$ do
4:   $(x_t, y_t) \sim D$
5:   for $j = 1, \ldots, J$ do
6:     $Z \sim \text{Poisson}(1)$
7:     for $z = 1, \ldots, Z$ do
8:         $w^{(j)}_t \leftarrow \text{OFS\_Update}(w^{(j)}_t, x_t, y_t)$
9:     end for
10:   end for
11:   $p_t = \frac{1}{J} \sum_{j=1}^{J} w^{(j)}_t$
12:   $p_t \leftarrow \text{Truncate}(p_t, B)$
13: end for

Figure 5.4: Pseudo code for online bagging feature selectors
1: Input
   • $B$: OFS truncation parameter
   • $R$: maximum $l_2$ magnitude
   • $J$: ensemble size

2: Initialization
   • $w_0^{(j)} \sim \mathcal{N}(0, 1) \quad \forall j \in [J]$
   • $\lambda_{j}^{sc} \leftarrow 0 \quad \forall j \in [J]$
   • $\lambda_{j}^{sw} \leftarrow 0 \quad \forall j \in [J]$
   • $p_0 \sim \mathcal{N}(0, 1)$

3: for $t = 1, \ldots, T$ do
   4: $(x_t, y_t) \sim \mathcal{D}$
   5: $\lambda_t = 1$
   6: for $j = 1, \ldots, J$ do
      7: $Z \sim \text{Poisson}(\lambda_t)$
      8: for $z = 1, \ldots, Z$ do
         9: $w_t^{(j)} \leftarrow \text{OFS}_\text{Update}(w_t^{(j)}, x_t, y_t)$
      10: end for
      11: if $y_t x_t^T w_t^{(j)} < 0$ then
         12: $\lambda_{j}^{sw} \leftarrow \lambda_{j}^{sw} + \lambda_t$
         13: $\lambda_t \leftarrow \lambda_t \left( \frac{t}{2\lambda_{j}^{sw}} \right)$
      14: else
         15: $\lambda_{j}^{sc} \leftarrow \lambda_{j}^{sc} + \lambda_t$
         16: $\lambda_t \leftarrow \lambda_t \left( \frac{t}{2\lambda_{j}^{sc}} \right)$
      17: end if
   18: end for
   19: Set $\epsilon_j = \frac{\lambda_{j}^{sw}}{\lambda_{j}^{sw} + \lambda_{j}^{sc}} \quad \forall j \in [J]$
   20: $p_t = \sum_{j=1}^{J} \log \left( \frac{1-\epsilon_j}{\epsilon_j} \right) w_t^{(j)}$
   21: $p_t \leftarrow \text{Truncate}(p_t, B)$
   22: end for

Figure 5.5: Pseudo code for online boosting feature selectors
Online Bagging

We begin by presenting online bagging for feature selectors (OFS-Bag), whose pseudo code can be found in Figure 5.4. OFS-Bag requires one additional input, the ensemble size $J$. To better promote diversity, the single OFS models, $w_t^{(j)}$, are initialized using a $K$ dimensional random Gaussian vector, as opposed to a zero-vector as done in the traditional OFS, at time $t$. Furthermore, initialized vectors are truncated using Algorithm 5.3, so that the models are $B$-sparse. At each time $t$, a data pair are sampled from the distribution $D$. For each OFS model in the ensemble, a random variable is then sampled from a Poisson($1$) distribution. The single model is updated with $(x_t, y_t)$ the number of times indicated by the Poisson random variable. Once all OFS models are updated, they are averaged and truncated. Hence, the prediction with the ensemble is no more complex than a prediction with a single model, and the ensemble is still $B$-sparse because of the truncation at line 12. Predictions can be made on a sample $x \sim D$ after line 12 using $\hat{y} = \text{sign}(p_t^T x)$.

Online Boosting

The pseudo code for online boosting with OFS (OFS-Boo) is shown in Algorithm 5.5. Similar to OFS-Bag, OFS-Boo requires one extra input, the size of the ensemble. Some of the key differences between OFS-Boo and OFS-Bag are that the Poisson sampling parameter is fixed for OFS-Bag, whereas OFS-Boo initializes the Poisson parameter, $\lambda_t$, to 1 then adjusts the parameter based on the ability of a single model to make the prediction with partial information.

Online boosting is similar to online bagging; however, unlike online bagging, the Poisson parameter does not remain fixed. Rather, it is adjusted for each data sample. The parameter is updated as follows, if an instance is misclassified by a single OFS model then the Poisson parameter, $\lambda_t$, is increased when the observation is presented
to the next OFS model in the ensemble. This is analogous to AdaBoost increasing
the increasing the instances weights in batch of data after a weak hypothesis has been
produced. If the single OFS model predicted $x_t$ correctly then the Poisson parameter
$\lambda_t$ is decreased. Note that line 11 of Figure 5.5 can use as many – or few – feature
measurements available in $x_t$. The OFS models are now combined via a weighted sum
rather than a simple average. The ensemble model $p_t$ is then truncated to assure that
it is $B$-sparse.

5.4.2 Promoting Diversity

Diversity within an ensemble is generally known to be a useful quality [143–146].
We experiment with two different approaches to promote varying levels of diversity
within the ensemble. First, at each time $t$ the parameter vectors, $w_{ij}^{(t)}$, of the ensem-
bles are updated, where with probability $\tau$ a random set of indices in the parameter
vector are evaluated. Rather than keeping the set of indices fixed for all parameters at
each time $t$ (i.e., $C_t$), we let each update to the parameter vector sample a different set
of indices. Thus, each classifier has the opportunity to sample different features, but limited to sampling at most $B$. This measure of diversity is implemented in all of our ensemble algorithms. In situations where there is a high percentage of missing data, this sampling procedure may not be feasible; however, we leave this investigation to future work. Another measure to promote diversity is to let the truncation parameter be a random variable sampled from $B \sim \text{Poisson}(\beta K)$, where $\beta \in (0, 1)$. Of course $B < K$, which is a check that must be performed. Figure 5.6 shows four Poisson distributions for a data set with $K = 100$. Setting a reasonable Poisson parameter allows the OFS models to have varying levels of diversity. We refer to the implementations that use the random truncation for OFS as OFS-Bag-R and OFS-Boo-R.

5.5 Experimental Results

5.5.1 Data Sets & Benchmarks

We evaluated our proposed ensemble approaches on data sets collected from the UCI machine learning repository [128]. All benchmarks selected are binary prediction problems whose feature set size and cardinality are shown in Table 5.2. A test-then-train strategy is used to evaluate the data stream [147], where an observation is tested before it is used for training any model. We measure the cumulative number of mistakes made by our ensemble based prediction algorithms and compare them to the single model methods as a benchmark. At each time step, the partial information in $\mathbf{x}_t$ is used for training; however, the full information in $\mathbf{x}_t$ is used to evaluate the mistake rate. Each data set is run 25 times, where the data stream is randomly permuted on each run. The average of the trials is reported. OFS free parameters were selected based on recommendations by the authors in [2] (see Table 5.3 for parameter values). All data sets have had their features standardized.
Table 5.2: Properties of the UCI data sets used in our experiments. All data sets are binary prediction problems.

<table>
<thead>
<tr>
<th>Name</th>
<th>Instances</th>
<th>Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>adult</td>
<td>32,561</td>
<td>123</td>
</tr>
<tr>
<td>german</td>
<td>1,000</td>
<td>24</td>
</tr>
<tr>
<td>ionosphere</td>
<td>351</td>
<td>33</td>
</tr>
<tr>
<td>ovarian</td>
<td>216</td>
<td>4,000</td>
</tr>
<tr>
<td>sido0</td>
<td>12,678</td>
<td>4,932</td>
</tr>
<tr>
<td>spam</td>
<td>4,601</td>
<td>56</td>
</tr>
<tr>
<td>splice</td>
<td>3,175</td>
<td>60</td>
</tr>
<tr>
<td>svmguide</td>
<td>1,243</td>
<td>21</td>
</tr>
</tbody>
</table>

Table 5.3: Parameter selection for OFS following guidelines from Wang et al. [2].

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\tau$</td>
<td>$\frac{1}{5}$</td>
</tr>
<tr>
<td>$J$</td>
<td>25</td>
</tr>
<tr>
<td>$\eta$</td>
<td>$\frac{1}{5}$</td>
</tr>
<tr>
<td>$R$</td>
<td>10</td>
</tr>
<tr>
<td>Poisson($\beta K$)</td>
<td>Poisson($\frac{1}{4} K$)</td>
</tr>
<tr>
<td>$B$</td>
<td>$\lfloor \frac{1}{4} K \rfloor$</td>
</tr>
</tbody>
</table>

Finally, we have made the code publicly available for reproducibility\footnote{Note that precise reproducibility of the evaluation times depends on the speed and capabilities of the CPU.}: \url{https://github.com/gditzler/OFSE}. The code was implemented in Matlab.

\subsection*{5.5.2 Real-World & UCI Experiments}

The evaluation times and mistake rates can be found in Table 5.4 and 5.5, respectively. There are several observations we can make from these experiments. First, the cumulative evaluation time of each ensemble method is approximately the same.
(time only includes the evaluation), which agrees with our intuition since there is little difference in the complexity. Note that evaluation of the ensemble model would be equal to the single model because the ensembles are linear models computed with a dot product. That is the ensemble prediction is given by \( \hat{y} = \mathbf{x}^T \mathbf{p}_t \) and the single model is \( \hat{y} = \mathbf{x}^T \mathbf{w}_t^{(j)} \). The only difference in timing between the single model and ensemble model would be the cumulative training times.

The cumulative mistake rates, on the other hand (see Table 5.5 and Figure 5.7), are significantly lower for the ensemble models than they are on the single OFS model on all data sets evaluated. The number in parenthesis is the performance rank of the algorithm (lower is better). In fact, OFS-Boo and OFS-Boo-R are generally ranked very well even compared to their bagging counterparts. The improvements can clearly be observed by examining the cumulative mistakes made over time (see Figure 5.7). We also begin to observe convergence in the mistake rate for some of the ensemble models evaluated on the data streams, despite the fact that these algorithms are only allowed to observe \( B \) of the \( K \) features. The randomization of \( B \) appears to provide some improvement for online bagging; however, the differences are not statistically significant.

We find that OFS-Boo and OFS-Boo-R – in general – performs the best, though not always with statistical significance (see Table 5.6 for \( p \)-values derived from the rank statistics [130]). Table 5.6 can be interpreted as follows: each element represents the \( p \)-value that “row algorithm” is performing better than the corresponding “column algorithm”. The Friedman rank test rejects the hypothesis that all algorithms are performing equally \( (p_F = 9.1831 \times 10^{-5} \) with the size of the test set to \( \alpha = 0.05 \)). Furthermore, upon further investigation of the family-wise analysis of the \( p \)-values derived from the mistakes, we find that each of the ensemble approaches presented in this work performs better – with statistical significance – than the single OFS
Figure 5.7: Cumulative mistakes made by a single OFS model and the proposed online bagging and boosting models. Note that at each time step only a fraction \( \frac{B}{K} \) of the total number of features are required for learning.

Table 5.4: Cumulative evaluation time (i.e., testing time) measured in seconds for the ensemble algorithms.

<table>
<thead>
<tr>
<th>Data Set</th>
<th>OFS-Bag</th>
<th>OFS-Boo</th>
<th>OFS-Bag-R</th>
<th>OFS-Boo-R</th>
</tr>
</thead>
<tbody>
<tr>
<td>adult</td>
<td>0.362</td>
<td>0.407</td>
<td>0.369</td>
<td>0.411</td>
</tr>
<tr>
<td>german</td>
<td>0.010</td>
<td>0.011</td>
<td>0.010</td>
<td>0.011</td>
</tr>
<tr>
<td>ionosphere</td>
<td>0.004</td>
<td>0.004</td>
<td>0.004</td>
<td>0.004</td>
</tr>
<tr>
<td>ovarian</td>
<td>0.010</td>
<td>0.009</td>
<td>0.010</td>
<td>0.009</td>
</tr>
<tr>
<td>sido0</td>
<td>0.720</td>
<td>0.658</td>
<td>0.722</td>
<td>0.660</td>
</tr>
<tr>
<td>spambase</td>
<td>0.045</td>
<td>0.052</td>
<td>0.045</td>
<td>0.050</td>
</tr>
<tr>
<td>splice</td>
<td>0.032</td>
<td>0.035</td>
<td>0.032</td>
<td>0.036</td>
</tr>
<tr>
<td>svmguide3</td>
<td>0.012</td>
<td>0.014</td>
<td>0.012</td>
<td>0.014</td>
</tr>
</tbody>
</table>
Table 5.5: Cumulative mistake rates made by the proposed online learning algorithms and a single OFS model. The number in parenthesis represents the order rank of the performance on the algorithm.

<table>
<thead>
<tr>
<th>Data Set</th>
<th>Single</th>
<th>OFS-Bag</th>
<th>OFS-Boo</th>
<th>OFS-Bag-R</th>
<th>OFS-Boo-R</th>
</tr>
</thead>
<tbody>
<tr>
<td>adult</td>
<td>0.34768 (5)</td>
<td>0.28876 (1)</td>
<td>0.31086 (3)</td>
<td>0.28915 (2)</td>
<td>0.29911 (4)</td>
</tr>
<tr>
<td>german</td>
<td>0.42981 (5)</td>
<td>0.37911 (4)</td>
<td>0.33674 (2)</td>
<td>0.38432 (3)</td>
<td>0.31231 (1)</td>
</tr>
<tr>
<td>ionosphere</td>
<td>0.28810 (5)</td>
<td>0.22400 (3)</td>
<td>0.15943 (1)</td>
<td>0.22419 (4)</td>
<td>0.16667 (2)</td>
</tr>
<tr>
<td>ovarian</td>
<td>0.21734 (5)</td>
<td>0.17736 (4)</td>
<td>0.13457 (2)</td>
<td>0.18450 (3)</td>
<td>0.13364 (1)</td>
</tr>
<tr>
<td>siso0</td>
<td>0.50575 (5)</td>
<td>0.50311 (3)</td>
<td>0.42115 (2)</td>
<td>0.49908 (4)</td>
<td>0.38612 (1)</td>
</tr>
<tr>
<td>spambase</td>
<td>0.17244 (5)</td>
<td>0.12559 (1)</td>
<td>0.12972 (3)</td>
<td>0.12588 (2)</td>
<td>0.13412 (4)</td>
</tr>
<tr>
<td>splice</td>
<td>0.23686 (5)</td>
<td>0.19389 (4)</td>
<td>0.18003 (1)</td>
<td>0.19433 (3)</td>
<td>0.18794 (2)</td>
</tr>
<tr>
<td>svmguide3</td>
<td>0.39052 (5)</td>
<td>0.30258 (4)</td>
<td>0.25980 (3)</td>
<td>0.27424 (1)</td>
<td>0.24745 (2)</td>
</tr>
</tbody>
</table>

rank  5.000  3.000  2.125  2.750  2.125

model. For statistical significance in the family-wise comparison we must require that $p \leq \alpha/4$ to account for the multiple comparisons [148]. Given this definition of statistical significance it becomes clear that the ensemble are clearly outperforming the single OFS model (see Table 5.6).

5.6 Summary

In this chapter, we addressed online learning of streaming data in a high-dimensional space, where many of the features carry little or no information, and only portions of the feature space are available at the time of learning. We presented two online ensemble-based approaches, where once trained, are of the same complexity as a single OFS model to evaluate unseen data, while maintaining the same level of sparsity. We also presented two approaches, specific for the online ensemble of feature selections, to promote diversity within the ensemble. OFS-Bag and OFS-Boo were evaluated on several data sets collected from the UCI machine learning repository to demonstrate the ability of the ensemble approach with single OFS models. The key advantage to
Table 5.6: Statistical significance test ($p$-values) for determining if an algorithm is performing better than any other. Note that for $\alpha = 0.05$, the $p$-value must be less than 0.01 to account for multiple comparisons.

<table>
<thead>
<tr>
<th></th>
<th>Single</th>
<th>OBag</th>
<th>OBoo</th>
<th>OBag-R</th>
<th>OBoo-R</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single</td>
<td>–</td>
<td>0.9943</td>
<td>0.9999</td>
<td>0.9978</td>
<td>0.9999</td>
</tr>
<tr>
<td>OBag</td>
<td>0.0057</td>
<td>–</td>
<td>0.8658</td>
<td>0.6241</td>
<td>0.8658</td>
</tr>
<tr>
<td>OBoo</td>
<td>0.0001</td>
<td>0.1342</td>
<td>–</td>
<td>0.2146</td>
<td>0.5000</td>
</tr>
<tr>
<td>OBag-R</td>
<td>0.0022</td>
<td>0.3759</td>
<td>0.7854</td>
<td>–</td>
<td>0.7854</td>
</tr>
<tr>
<td>OBoo-R</td>
<td>0.0001</td>
<td>0.1342</td>
<td>0.5000</td>
<td>0.2146</td>
<td>–</td>
</tr>
</tbody>
</table>

Identify is that OFS-Bag and OFS-Boo typicality yield smaller mistake rates – with statistical significance – than any single linear model, while maintaining the same level of sparsity.
6. Feature Selection Tools for Genomics

6.1 Introduction

Some of the current software tools for comparative metagenomics provide microbial ecologists the ability to investigate and explore bacterial communities using $\alpha$- & $\beta$-diversity (i.e., within and between sample diversity, respectively) tests. Feature subset selection can also provide a unique insight into the differences between metagenomic phenotypes. In particular, feature subset selection methods can obtain the operational taxonomic units (OTUs), or functional features, that have the most influence on the condition being studied. For example, in a previous studies have begun to examine information-theoretic feature selection to understand the differences between protein family abundances that best discriminate between different age groups in the human gut microbiome [124].

In this chapter we present a new Python command line tool, which is compatible with the widely adopted BIOM format [149], for microbial ecologists that implements information-theoretic subset selection methods for biological data formats. We demonstrate the software tools capabilities on publicly available datasets.

6.2 Motivation

There is an immense amount of sequence data being collected from the next generation sequencers. Sequences from bacterial communities are collected from whole genome shotgun (WGS) or amplicon sequencing runs, and the analysis of such data allows researchers to study the functional or taxonomic composition of a sample. Microbial ecologists represent the composition in the form of an abundance matrix (see Section 2.4.5), which usually holds counts of operational taxonomic units (OTUs), but
can also hold counts of genes/metabolic pathway occurrences if the data are collected from WGS (see Figure 6.1). Furthermore, collections of metagenomic samples contains different factors, groups, or phenotypes, such as environmental pH and salinity values, or a health related status [94,150].

In this chapter, we introduce software tools for microbial ecologists that implement feature subset selection routines for biological data formats, which we refer to as Fizzy. Prior to feature selection, we assume that the raw sequences from the environmental samples have already been classified into operational taxonomic units (OTUs), or functional features. The raw OTU counts or functional features are stored in a matrix $X \in \mathbb{N}_+^{K \times M}$, where $\mathbb{N}_+$ is the set of positive natural numbers, $K$ is the number of OTU clusters, and $M$ is the number of samples collected (i.e., abundance profile in Figure 6.1). The $M$ samples contain a significant amount of meta-data describing...
the sample, which is where we obtain phenotypes describing the sample. While there may be many different meta-data, we shall only focus on one piece of meta-data at a time. For example, a sample may contain the sex, age, and height of the person from where a sample was collected, and the analysis would only use one of those fields. That is we could use \( X \) to build a predictive model of sex. Both the data matrix and meta-data can be found for hundreds of publicly available datasets though pioneering projects such as MG-RAST [8], KBase [9], the Human Microbiome Project [151], and the Earth Microbiome Project [152].

A natural question to ask about studies with multiple phenotypes is: “which OTUs or functions are important for differentiating the phenotypes?” Answering such a question can be useful for understanding which conditions are driving/being affected by differences in composition and function across samples. Subset selection can produce a feature subset that not only removes irrelevant features (i.e., features that do not carry information about the phenotype), but also does not contain features that are redundant (i.e., features carry the same information). This process of reducing the feature set offers a rapid insight into uncovering the differences between multiple populations in a metagenomic study and can be performed as complementary analysis to \( \beta \)-diversity methods, such as principal coordinate analysis (PCoA) [115].

6.3 Fizzy: Feature Selection for Biological Data Formats

Fizzy is a suite of feature subset selection tools for biological data formats and, in particular, we use the Biom\(^1\) format standard [149], because of its wide usage throughout microbial ecology (e.g., HMP\(^2\), EMP\(^3\), etc). Commonly used software in microbial ecology, such as Qiime [110], typically requires a Biom file containing the

\(^1\)The Biom format follows the standard for JSON.
\(^2\)http://hmpdacc.org/
\(^3\)http://www.earthmicrobiome.org/
16S data and a map file contain the meta-data of the samples within the Biom file. However, Fizzy allows users to store the meta-data in the Biom file directly, thus avoiding requirements for both a Biom and map file. Figure 6.2 shows a high-level overview of the software tool, which is discussed below.

The Fizzy software suite implements information-theoretic subset selection, NPFS (see Chapter 3), and lasso. The core of Fizzy is based on the FEAST C feature selection library [11], which is used to implement all of the information theoretic methods. FEAST was selected for two primary reasons: (i) the library contains a large selection of information-theoretic feature selection objective functions, and (ii) the run-time of FEAST is typically faster than other feature selection libraries because it is written in a compiled language. We implemented a Python interface for FEAST to use within Fizzy, which is available to the public\(^4\). The Fizzy tool

\(^4\)http://github.com/EESI/PyFeast
requires a Biom format OTU table (sparse or dense), a mapping file in tab-delimited (TSV) format, a metagenomic phenotype column in the map file, and an output file path be specified. Furthermore, Fizzy allows the user to specify the number of taxonomic units to select as well as the feature selection objective function. The current implementation of Fizzy has nine subset selection objective functions, which are all based on information theory (see Brown et al. for the mathematical details about the objective functions [11]). We also provide an implementation of the NPFS module, which can infer on the number of relevant features given any subset selection methods in FEAST [15], which is described in Chapter 3. NPFS has a parallel implementation where the user can control the number of cores used by the program. The lasso implementation within Fizzy uses Scikit-Learn [153].

6.4 Benchmark Data Sets

We consider two publicly available data sets for benchmarking the suite of Fizzy’s feature selection tools.

6.4.1 MetaHit

As a case study, let’s examine a metagenome data set collected by [150], which is widely referred to as the MetaHit data set. The data are collected from Illumina-based metagenomic sequencing of 124 fecal samples of 124 European individuals from Spain and Denmark. Among the 124 individuals in the database, 25 are from patients who have inflammatory bowel disease (IBD), and 42 patients are also obese. It is interesting to note that only three of the individuals who have IBD are also obese. Let us consider two different labeling schemes for the data: IBD and obesity, both of which are binary prediction problems. The sequences from each individual are functionally annotated using the Pfam database [154], in a recent study that utilized
the MetaHit data set for feature selection on patient age [124]. There are a total of 6,343 unique functional features detected in the data set.

6.4.2 American Gut Project

The American Gut Project (AGP)\textsuperscript{5} is a project that is a joint effort between several institutes that aims at characterizing the gut microbiome of individuals across the US. The samples are collected via volunteers who collect the samples and send them to the institutions running the project for sequencing and analysis. In a way, the AGP can be seen as a way of crowdsourcing science to get individuals involved in the research by providing sample, which are kept anonymous. Furthermore, which the project focuses on the “gut” as indicated by the name, the AGP data base contains samples from different body sites (e.g., skin or oral cavity).

Once the samples are received by the Knight Lab at the University of California, San Diego, they are processed using 16S amplicon sequencing to produce an taxonomic abundance matrix (see Section 2.4.5). The AGP data are represented by the abundance of thousands of operational taxonomic units (OTUs) detected during sequencing and we examine only the the fecal samples. The final fecal data are represented by 2.9k+ people (of both male & females), diet types, and age groups. We extracted 24k+ taxa abundances as the features and the diet type was chosen label for each sample. The diets are categorized as either omnivore or vegetarian.

6.5 Experimental Results

6.5.1 American Gut Project

Fizzy was run using joint mutual information maximization (JMI) on 2.9k+ samples collected from the AG Project and feature were selected using the diet type as

\textsuperscript{5}http://americangut.org/
the predictor variable. The diets are are broken down into omnivore and vegetarians, where subcategories of omnivore and vegetarians (e.g., omnivore but does not eat red meat) is simply categorized as omnivore. Table 6.1 shows the top ranking OTUs as selected for differentiate omnivores versus vegetarians in the AG Project data. Both Bacteroides and Prevotella were selected by Fizzy (note that Prevotella is not shown in the table because it was not ranked within the top 15 OTUs), which have been hypothesized as being important differentiators of diet [155]. NPFS detected 27 OTUs of the Prevotella genus and the relative abundances were larger for the vegetarians when examining the largest differences, which coincides with results in the literature [156].

We also compare our Fizzy to Qiime’s random forests [14], because the random forest within Qiime has become a commonly used benchmark in microbial ecology. The top ranked features for random forests are found in Table 6.2. Similar to approaches such as mRMR and JMI, a threshold for the number of features to select must be chosen in advance for the random forest. We find some overlap between the results of Fizzy (using JMI) and the random forests. The Bacteroides genus was detected as relevant several times for both Fizzy and random forests. We find the Bacteroides has been found to be an indicator of diet [136,137,157].

Figure 6.3 shows the largest differences between the omnivores and vegetarians in the top 500 OTUs feature selected by JMI. The numerical values on the x-axis that correspond to the OTU given by:

1. (F148) Bacteroidetes, Bacteroidia, Bacteroidales, Bacteroidaceae, Bacteroides uniformis (GGID1733364): -6.20923
2. (F4) Bacteroidetes, Bacteroidia, Bacteroidales, Bacteroidaceae, Bacteroides (GGID197367): 5.14587
3. (F127) Firmicutes, Clostridia, Clostridiales, Lachnospiraceae (GGID340761):
4. (F223) *Firmicutes, Clostridia, Clostridiales, Ruminococcaceae* (GGID180285): -4.11038

5. (F291) *Bacteroidetes, Bacteroidia, Bacteroidales, Bacteroidaceae, Bacteroides ovatus* (GGID180606): -3.96605

6. (F206) *Firmicutes, Clostridia, Clostridiales, Ruminococcaceae* (GGID352347): -3.65923

7. (F195) *Bacteroidetes, Bacteroidia, Bacteroidales, Bacteroidaceae, Bacteroides* (GGID3465233): 3.34877

8. (F60) *Firmicutes, Clostridia, Clostridiales* (GGID173876): -2.49844

9. (F458) *Firmicutes, Clostridia, Clostridiales, Lachnospiraceae* (GGID193477): -2.28077

10. (F113) *Bacteroidetes, Bacteroidia, Bacteroidales, Rikenellaceae* (GGID4453609): 1.94571

11. (F463) *Firmicutes, Clostridia, Clostridiales, Lachnospiraceae, Ruminococcus gnavus* (GGID191755): 1.32321

12. (F310) *Bacteroidetes, Bacteroidia, Bacteroidales, Porphyromonadaceae, Parabacteroides* (GGID847228): 1.30030

13. (F276) *Firmicutes, Clostridia, Clostridiales, Lachnospiraceae, Coprococcus* (GGID2740950): -1.12856

14. (F257) *Bacteroidetes, Bacteroidia, Bacteroidales, Bacteroidaceae, Bacteroides* (GGID190913): 0.89408

15. (F106) *Firmicutes, Clostridia, Clostridiales, Lachnospiraceae* (GGID176306): -0.58509

where the difference is $\times 10^{-3}$ and a negative value means that the average relative abundance was higher in the vegetarians.
Figure 6.3: Joint Mutual Information (JMI) was configured to select 500 features from the 25k+ OTUs in the American Gut Project’s fecal samples. The diet of the sample is the dependent variables. The selected Greengenes (GG) OTUs are sorted by the absolute difference between the omnivores and vegetarians. The numerical values on the x-axis that correspond to an OTU can be found the the text.
Table 6.1: List of the top ranking features for omnivores and vegetarians in the 16S data collected from the American Gut Project detected using JMI within Fizzy. The number followed by “F” indicates the order Fizzy selected the OTU and the “GGID” contains the Greengenes OTU ID from the taxonomic classification.

<table>
<thead>
<tr>
<th>(feature rank)</th>
<th>Operation Taxonomic Unit Classification (OTU ID)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(F1)</td>
<td>Firmicutes, Clostridia, Clostridiales, Lachnospiraceae (GGID4329132)</td>
</tr>
<tr>
<td>(F2)</td>
<td>Firmicutes, Clostridia, Clostridiales, Ruminococcaceae (GGID185584)</td>
</tr>
<tr>
<td>(F3)</td>
<td>Bacteroidetes, Bacteroidia, Bacteroidales, Bacteroidaceae, Bacteroides (GGID177150)</td>
</tr>
<tr>
<td>(F4)</td>
<td>Bacteroidetes, Bacteroidia, Bacteroidales, Bacteroidaceae, Bacteroides (GGID197367)</td>
</tr>
<tr>
<td>(F5)</td>
<td>Bacteroidetes, Bacteroidia, Bacteroidales, Bacteroidaceae, Bacteroides (GGID199716)</td>
</tr>
<tr>
<td>(F6)</td>
<td>Bacteroidetes, Bacteroidia, Bacteroidales, Bacteroidaceae, Bacteroides (GGID188887)</td>
</tr>
<tr>
<td>(F7)</td>
<td>Bacteroidetes, Bacteroidia, Bacteroidales, Bacteroidaceae, Bacteroides (GGID312140)</td>
</tr>
<tr>
<td>(F8)</td>
<td>Bacteroidetes, Bacteroidia, Bacteroidales, Bacteroidaceae, Bacteroides (GGID4401110)</td>
</tr>
<tr>
<td>(F9)</td>
<td>Bacteroidetes, Bacteroidia, Bacteroidales, Bacteroidaceae, Bacteroides (GGID198449)</td>
</tr>
<tr>
<td>(F10)</td>
<td>Firmicutes, Bacilli, Bacillales, Paenibacillaceae, Paenibacillus (GGID4470837)</td>
</tr>
<tr>
<td>(F11)</td>
<td>Firmicutes, Clostridia, Clostridiales, Ruminococcaceae, Faecalibacterium prausnitzii (GGID359314)</td>
</tr>
<tr>
<td>(F12)</td>
<td>Bacteroidetes, Bacteroidia, Bacteroidales, Bacteroidaceae, Bacteroides (GGID2859978)</td>
</tr>
<tr>
<td>(F13)</td>
<td>Firmicutes, Clostridia, Clostridiales (GGID197832)</td>
</tr>
<tr>
<td>(F14)</td>
<td>Bacteroidetes, Bacteroidia, Bacteroidales, Bacteroidaceae, Bacteroides (GGID205904)</td>
</tr>
<tr>
<td>(F15)</td>
<td>Firmicutes, Clostridia, Clostridiales, Ruminococcaceae, Faecalibacterium prausnitzii (GGID520413)</td>
</tr>
</tbody>
</table>
Table 6.2: List of the top ranking features for omnivores and vegetarians in the 16S data collected from the American Gut Project detected using Random Forests. The number followed by “F” indicates the order Fizzy selected the OTU and the “GGID” contains the Greengenes OTU ID from the taxonomic classification.

<table>
<thead>
<tr>
<th>Operation Taxonomic Unit Classification (OTU ID)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(F1) Bacteroidetes, Bacteroidia, Bacteroidales, Bacteroidaceae, Bacteroides ovatus (GGID180606)</td>
</tr>
<tr>
<td>(F2) Bacteroidetes, Bacteroidia, Bacteroidales, Bacteroidaceae, Bacteroides fragilis (GGID4386507)</td>
</tr>
<tr>
<td>(F3) Firmicutes, Clostridia, Clostridiales, Lachnospiraceae, Roseburia (GGID4335815)</td>
</tr>
<tr>
<td>(F4) Actinobacteria, Actinobacteria, Actinomycetales, Corynebacteriaceae, Corynebacterium simulans (GGID912997)</td>
</tr>
<tr>
<td>(F5) Bacteroidetes, Bacteroidia, Bacteroidales, Rikenellaceae (GGID175375)</td>
</tr>
<tr>
<td>(F6) Firmicutes, Clostridia, Clostridiales, Lachnospiraceae (GGID194112)</td>
</tr>
<tr>
<td>(F7) Firmicutes, Clostridia, Clostridiales, Ruminococcaceae (GGID189924)</td>
</tr>
<tr>
<td>(F8) Bacteroidetes, Bacteroidia, Bacteroidales, Bacteroidaceae, Bacteroides (GGID1105984)</td>
</tr>
<tr>
<td>(F9) Bacteroidetes, Bacteroidia, Bacteroidales, Bacteroidaceae, Bacteroides (GGID197367)</td>
</tr>
<tr>
<td>(F10) Firmicutes, Clostridia, Clostridiales, Ruminococcaceae (GGID174818)</td>
</tr>
<tr>
<td>(F11) Firmicutes, Clostridia, Clostridiales, Ruminococcaceae (GGID4324040)</td>
</tr>
<tr>
<td>(F12) Firmicutes, Clostridia, Clostridiales, Ruminococcaceae (GGID197204)</td>
</tr>
<tr>
<td>(F13) Bacteroidetes, Bacteroidia, Bacteroidales, Bacteroidaceae, Bacteroides (GGID1944498)</td>
</tr>
<tr>
<td>(F14) Firmicutes, Clostridia, Clostridiales, Ruminococcaceae (GGID196307)</td>
</tr>
<tr>
<td>(F15) Firmicutes, Clostridia, Clostridiales, Ruminococcaceae, Ruminococcus flavefaciens (GGID1122673)</td>
</tr>
</tbody>
</table>
Table 6.3: List of the top ranking features for omnivores and vegetarians in the 16S data collected from the American Gut Project detected using NPFS. The number followed by “F” indicates the order Fizzy selected the OTU and the “GGID” contains the Greengenes OTU ID from the taxonomic classification.

<table>
<thead>
<tr>
<th>Operation Taxonomic Unit Classification (OTU ID)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(F1) Firmicutes, Clostridia, Clostridiales, Lachnospiraceae, Shuttleworthia (GGID4424924)</td>
</tr>
<tr>
<td>(F2) Cyanobacteria, Oscillatoriophycideae, Chroococcales, Xenococcales, Chroococcidiopsis (GGID649518)</td>
</tr>
<tr>
<td>(F3) Proteobacteria, Betaproteobacteria, Gallionellales, Gallionellaceae, Gallionella (GGID3239358)</td>
</tr>
<tr>
<td>(F4) Firmicutes, Clostridia, Clostridiales (GGID176062)</td>
</tr>
<tr>
<td>(F5) Firmicutes, Bacilli, Gemellales, Gemellaceae (GGID967433)</td>
</tr>
<tr>
<td>(F6) Firmicutes, Erysipelotrichi, Erysipelotrichales, Erysipelotrichaceae, Erysipelothrix (GGID4478325)</td>
</tr>
<tr>
<td>(F7) Firmicutes, Clostridia, Clostridiales, Lachnospiraceae (GGID183576)</td>
</tr>
<tr>
<td>(F8) Firmicutes, Clostridia, Clostridiales, Clostridiaceae, Clostridium (GGID174688)</td>
</tr>
<tr>
<td>(F9) Firmicutes, Clostridia, Clostridiales, Clostridiaceae (GGID1137375)</td>
</tr>
<tr>
<td>(F10) Firmicutes, Clostridia, Clostridiales, Lachnospiraceae, Blautia (GGID305997)</td>
</tr>
<tr>
<td>(F11) Firmicutes, Clostridia, Clostridiales, Lachnospiraceae (GGID288682)</td>
</tr>
<tr>
<td>(F12) Proteobacteria, Gammaproteobacteria, Pasteurellales, Pasteurellaceae, Haemophilus (GGID995893)</td>
</tr>
<tr>
<td>(F13) Bacteroidetes, Bacteroidia, Bacteroidales, Bacteroidaceae, Bacteroides (GGID4450198)</td>
</tr>
<tr>
<td>(F14) Firmicutes, Clostridia, Clostridiales (GGID267502)</td>
</tr>
<tr>
<td>(F15) Bacteroidetes, Bacteroidia, Bacteroidales, Bacteroidaceae, Bacteroides (GGID531722)</td>
</tr>
</tbody>
</table>
Table 6.4: List of the top five ranked Pfams as selected by the Fizzy’s Mutual Information Maximization (MIM) applied to MetaHit

<table>
<thead>
<tr>
<th>Rank</th>
<th>IBD features</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>feature 1 ABC transporter (PF00005)</td>
</tr>
<tr>
<td>2</td>
<td>feature 2 Phage integrase family (PF00589)</td>
</tr>
<tr>
<td>3</td>
<td>feature 3 Glycosyl transferase family 2 (PF00535)</td>
</tr>
<tr>
<td>4</td>
<td>feature 4 Acetyltransferase (GNAT) family (PF00583)</td>
</tr>
<tr>
<td>5</td>
<td>feature 5 Helix-turn-helix (PF01381)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Rank</th>
<th>Obese features</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>feature 1 ABC transporter (PF00005)</td>
</tr>
<tr>
<td>2</td>
<td>feature 2 MatE (PF01554)</td>
</tr>
<tr>
<td>3</td>
<td>feature 3 TonB dependent receptor (PF00593)</td>
</tr>
<tr>
<td>4</td>
<td>feature 4 Histidine kinase-, DNA gyrase B-, and HSP90-like ATPase (PF02518)</td>
</tr>
<tr>
<td>5</td>
<td>feature 5 Response regulator receiver domain(PF00072)</td>
</tr>
</tbody>
</table>

6.5.2 MetaHit

The top Pfams that maximize the mutual information for the MetaHit data set are shown in Table 6.4. It is known in IBD patients, the expression of ABC transporter protein (PF00005, the first feature MIM selected for classifying IBD vs. no IBD samples) is decreased which limits the protection against various luminal threats [158]. The feature selection for IBD also identified glycosyl transferase (PF00535), whose alternation is hypothesized to result in recruitment of bacteria to the gut mucosa and increased inflammation [159, 160]. And the genotype of acetyltransferase (PF00583) plays an important role in the pathogenesis of IBD, which is useful in the diagnostics and treatment of IBD [93]. It is not surprising that ABC transporter (PF00005) is also selected for obesity, which is known to mediate fatty acid transport that is associated with obesity and insulin resistant states [161], and ATPases (PF02518) that catalyze dephosphorylation reactions to release energy.
6.5.3 Evaluation Times

Figure 6.4 shows the evaluation time of five feature selection algorithms and the number of features they select. Both lasso and NPFS can select size of the relevant set, which is why they are represented as a single point. An interesting observation to make is that lasso selects very few features (compared to NPFS) and the evaluation time is nearly triple that of NPFS. MIM, as expected, has the fast evaluation time because there is no calculation for redundancy, and the approaches that use redundancy (JMI and mRMR) take significantly longer to run.

6.6 Summary

Feature subset selection provides an avenue for rapid insight to the taxonomic or functional differences that can be found between different metagenomic or 16S phenotypes in an environmental study. We have presented an information-theoretic feature subset selection, and lasso for biological data formats in Python that are compatible with those used with the software Qiime package. Furthermore, we have
compared the results of our subset selection implementations on real-world 16S and metagenomic data, and we have compared our results to recent literature to ensure biological importance.
7. Conclusions

The primary scope of this thesis was to develop new scaleable feature subset selection algorithms that are capable of processing large volumes of features and observations. There were three families of algorithms that were presented for addressing such problems, namely: NPFS, SLSS, and OFSE, where NPFS and SLSS use filters and OFSE is an embedded approach for subset selection.

7.1 Contributions of this Work

Neyman-Pearson based feature selection (NPFS) is an approach for performing large scale – and potentially – distributed feature selection. This framework for feature subset selection (i) has theoretically justifiable properties, (ii) is classifier-independent, (iii) provides computationally straightforward model to analyze large data, and (iv) can automatically detect the relevant feature set size. NPFS is highly parallelizable, and it can fit into the MapReduce computational model. Another advantage of NPFS is that it does not restrict the user to using any particular objective function, rather the user can choose the function.

Sequential Learning Subset Selection (SLSS) is developed to sequentially search the subspace of features using a multi-arm bandit algorithm to determine the importance of the features. SLSS is motivated by the fact that NPFS uses generic subset selection approach to evaluate all feature variables at each iteration, and this evaluation can be too computationally intensive. Therefore, SLSS still uses a generic subset selection approach, however, only a subspace is considered. SLSS can be implemented in parallel on distributed architectures such as Hadoop using the bag-of-little bootstraps. SLSS can also work with partial information, hence, making it
possible to address missing/incomplete data.

**Online Feature Selection with Bagging & Boosting** is developed with the motivation that real-world applications are commonly associated with scenarios where: (i) not all data instances are available at the time of training, (ii) not all data can fit into memory during training, and/or (iii) not all feature values (e.g., measurements) are available for all data instances. Therefore, we extended the Online Feature Selection (OFS) approach [2], a recently introduced approach that uses partial feature information, by developing an online ensemble to make predictions. The OFS approach assumes a linear model, which allows the $l_0$-norm of the parameter vector to be constrained to perform feature selection. We show that the ensemble model (OFSE) typically yields a smaller mistake rate than any single linear model, while maintaining the same level of sparsity. Thus, a $B$-sparse model ($\|w\|_0 \leq B$ for a model $w$) for an ensemble yields lower error rates than a $B$-sparse single model.

**Feature Selection Tools for Biological Data Formats** are implemented in Python for performing feature selection with the widely used Biom and map file formats. We implemented an interface to the Feast C library for information-theoretic feature selection. Furthermore, we provide implementations of NPFS and lasso for microbial ecologists to use with their data. All software implementations have been made available to the public.

### 7.2 Broader Impacts

The primary application area of this thesis was to study the complex interactions within microbial communities, and understand how they differ under various environmental conditions. Though at the core of analysis was the feature subset selection algorithms presented throughout this thesis. NPFS, SLSS, and OFSE are all approaches for performing feature subset selection that can be applied to a broad
spectrum of applications. For example, algorithms presented in this thesis can be applied to astronomical time series analysis [6], health care [4, 5, 79], proteomics [81], and text analysis [162].

7.3 Recommendations for Future Work

While this work provided some basic families of large scale filter-based subset selection approaches, there is still work that needs to be explored. The theoretical bounds on the SLSS regret are quite loose and do not involve $k$ or $\ell$. It would be worthwhile to understand the influence of these parameters on the regret. Furthermore, we observed that SLSS can appropriately identify the correct feature set size with our sampling procedure even with a poor choice of $\ell$ and $k$ (see Figure 4.11). It would be advantageous to explore how MapReduce can be used for filter-based subset selection – particularly SLSS – to extend the algorithms to large-scale distributed data.

Embedded subset selection for massive data is typically more common due to improvements to optimization [2, 10, 12]; though, implementing filter-based subset selection approaches on massive data is still largely under-explored. Improvements to optimization techniques has the potential to move embedded subset selection to even larger scale data sets than we have evaluated in this thesis. Furthermore, optimizing $B$ for OFSE would remove a free parameter and potentially lead to lower mistake rates.


