Fall 2018

Exploring Machine Learning Techniques To Improve Peptide Identification

Fawad Kirmani

Exploring Machine Learning Techniques To Improve Peptide Identification

by

Fawad Kirmani

Bachelor of Technology
Chandra Shekhar Azad University of Agriculture and Technology, Kanpur, 2010

Submitted in Partial Fulfillment of the Requirements

for the Degree of Master of Science in

Computer Science and Engineering

College of Engineering and Computing

University of South Carolina

2018

Accepted by:

John Rose, Director of Thesis

Marco Valtorta, Reader

Jijun Tang, Reader

Cheryl L. Addy, Vice Provost and Dean of the Graduate School

Dedication

I am dedicating this thesis to my parents.

Acknowledgments

I would like to express my appreciation for Prof. Dr. John Rose, who as my thesis advisor introduced me to the world of proteomics. He was a constant source of guidance, good ideas and encouragement while working on this thesis. He has always been patient with my questions and without his help and knowledge this thesis would not have been completed.

I would like to appreciate all the help that I got from my lab mate Jeremy Lane, who has helped me a lot in data preparation for my experiments. He used to dig deep for the source of the datasets we come across while preparing one more dataset for my experiments.

I would also like to thank Prof. Dr. Marco Valtorta and Prof. Dr. Jijun Tang for agreeing to be part of my thesis committee. They were always ready to spare some time from their busy schedule to answer my queries.

I would also like thank my family and friends who have been source of encouragement for me throughout my life.

Abstract

The goal of this work is to improve proteotypic peptide prediction with lower processing time and better efficiency. Proteotypic peptides are the peptides in protein sequence that can be confidently observed by mass-spectrometry based proteomics. One of the widely used method for identifying peptides is tandem mass spectrometry (MS/MS). The peptides that need to be identified are compared with the accurate mass and elution time (AMT) tag database. The AMT tag database helps in reducing the processing time and increases the accuracy of the identified peptides. Prediction of proteotypic peptides has seen a rapid improvement in recent years for AMT studies for peptides using amino acid properties like charge, code, solubility and hydropathy.

We describe the improved version of a support vector machine (SVM) classifier that has achieved similar classification sensitivity, specificity and AUC on *Yersinia Pestis*, *Saccharomyces cerevisiae* and *Bacillus subtilis str. 168* datasets as was described by Web-Robertson et al. [15] and Ahmed Alqurri [11]. The improved version of the SVM classifier uses the C++ SVM library instead of the MATLAB built in library. We describe how we achieved these similar results with much lesser processing time.

Furthermore, we tested four machine learning classifiers on *Yersinia Pestis*, *Saccharomyces cerevisiae* and *Bacillus subtilis str. 168* data. We performed feature selection from scratch, using four different algorithms to achieve better results from the different machine learning algorithms. Some of these classifiers gave similar or better results than the SVM classifiers with fewer features. We describe the results of these four classifiers with different feature sets.

Preface

This MS thesis is written as completion of my research work at the University of South Carolina. I started to work on this project one year ago with Prof. Dr. John Rose with great enthusiasm. I took this project to learn how to apply different machine learning techniques to a given problem. By working on this project I was introduced to the world of proteomics. I learned about identification of peptides and how it helps in identification of proteins.

In Chapter 2, I describe the data preparation methodology to prepare the proteotypic and non-proteotypic dataset. I have used the datasets available at National Center for Biotechnology Information, MassIVE (University of California at San Diego) and Global Proteome Machine Database (GPMDB) to prepare our dataset. In data preparation, I appreciate all the help I got from Jeremy Lane.

In Chapter 3, I have improved on the support vector classifier (SVM) described by Ahmed Alqurri [11] and Web Robertson et al. [15]. In Chapter 4, I performed different feature selection algorithms from scratch without considering much of previous work. In Chapter 5, I have implemented different machine learning classifiers using feature sets from Chapter 4. In Chapter 3, 4, and 5, I have used *Yersinia Pestis* and *Saccharomyces cerevisiae* datasets to perform machine learning algorithms. In Chapter 6, I have improved on the classification models described in Chapter 5. In Chapter 6, I have added *Bacillus subtilis str. 168* dataset prepared in Chapter 2 to further test our models.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Proteomics is the study of proteins at a very large scale. The goal of proteomics is to identify and quantify proteins in a cell. Proteins unlike genomes are dynamic and are of varying complexity. This is the significant challenge in proteomics. This challenge is overcome by one of the primary approaches in proteomics, Tandem Mass Spectrometry (MS/MS). MS/MS offers high-throughput quantification of the proteome in a biological sample. However, due to the high-throughput capability of MS/MS, the cost of performing this analysis on large datasets is significantly large [15].

As described by Web-Robertson et al. [15], there is a significant amount of effort that goes in to cataloging peptides identified by MS/MS over multiple platforms and database search routines as the information becomes available (Craig et al., [13]; Desiere et al., [5]; Jones et al., [8]; Kiebel et al., [12]). These database are built over time and are very helpful in evaluating proteomes for which data has been amassed. These databases help in reducing cost and time as the search routine to identify proteotypic peptides has to only run on a subset of possible peptide candidates.

The challenge of building these databases for new organisms remains. To overcome the very high cost of building these databases, several algorithmic approaches have been proposed. These algorithms take advantage of the fact that there are many known properties associated with the likelihood of proteotypic peptides, such as polarity, hydrophilicity, hydrophobicity of the peptide. By using the known properties of the peptides, the challenge of predicting proteotypic peptides are significantly reduced. All of these approaches are based on the machine learning algorithms and

model building. In one of the earliest work, Web Robertson et al (2010) [15] used simple sequence-derived properties of peptides for AMT studies to predict proteotypic peptides using support vector machine (SVM) classification. The goal of my work is to improve the prediction of the proteotypic peptides.

In the method described by Web-Robertson et al. [15], the concept of proteotypic peptides is defined as the peptide that has been included in the AMT database at any time that the parent protein is observed [15]. We have incorporated one of the three dataset used by STEPP (Webb-Robertson, 2010) [15] and adopted that definition of proteotypic peptides. Web-Robertson et al [15] used 35 features in predicting proteotypic peptides. Table 1.1 lists 35 features from Web-Robertson et al., 2010 [15].

Table 1.1 Proteotypic peptide features. Features 1-35 are from Web-Robertson et al., 2010 [15] and Feature 36 is from Ahmed Alqurri [11]

Index	**Features**
1	Length
2	Molecular weight
3	Number of non-polar hydrophobic residues
4	Number of polar hydrophilic residues
5	Number of uncharged polar hydrophilic residues
6	Number of charged polar hydrophilic residues
7	Number of positively charged polar hydrophilic residues
8	Number of negatively charged polar hydrophilic residues
9	Hydrophobicity−Eisenberg scale (Eisenberg et al., 1984)
10	Hydrophilicity−Hopp−Woods scale (Hopp and Woods, 1981)
11	Hydrophobicity−Kyte−Doolittle (Kyte and Doolittle, 1982)
12	Hydropathicity−Roseman scale (Roseman, 1988)
13	Polarity−Grantham scale (Grantham, 1974)
14	Polarity−Zimmerman scale (Zimmerman et al., 1968)
15	Bulkiness (Zimmerman et al., 1968)
16−35	Amino acid singlet counts
36	Ordered Amino Acid Usage (3-AAU or 2-AAU)

Ahmed Alqurri, 2017 [11] added one more feature: Ordered Amino Acid Usage

(AAU). He was able to achieve similar results to Web-Robertson with only seven features for the *Yersinia Pestis* dataset. Ordered Amino Acid Usage is an an abstract model of bonds between adjacent amino acids [11]. Ordered amino acid tuples capture the mutual information of these peptide fragments at an abstract level [11]. Alqurri [11] considered both tuples (2-AAU) and triples (3-AAU). We have also adopted that definition of Ordered Amino Acid Usage.

As already described by Ahmed Alqurri [11], some of the STEPP (Webb-Robertson, 2010) features compliment the AAU approach. We verified that and come up with different subsets, combining STEPP features and AAU. These subset of features are slightly different for each of the feature selection methods used in this research. For all these different feature set we experimented with different machine learning techniques to get the optimal results for different datasets. Table 1.1 shows all the proteotypic features we used in our research.

Chapter 2

Data Preparation Methodology

We have incorporated *Yersinia Pestis* dataset from Web-Robertson et al. [15] in our research. The *Saccharomyces cerevisiae* (or *Yeast*) dataset is incorporated from Ahmed Alqurri [11]. In order to verify and test our classification models, we prepared one more dataset. We prepared the dataset for *Bacillus subtilis str. 168*. To prepare the dataset for *Bacillus subtilis str. 168*, first we downloaded three files -

1. Proteomee file in fasta format from National Center for Biotechnology Information (NCBI)
   https://www.ncbi.nlm.nih.gov/genome/665/

2. DeepNovo file in mgf format from MassIVE (University of California, San Diego) [14] *Bacillus subtilis str. 168* from
   ftp://massive.ucsd.edu/MSV000081382/peak/DeepNovo/HighResolution/data/

3. Observed peptides from Global Proteome Machine Database (GPMDB) [4]
   http://peptides.thegpm.org/ /peptides_by_species/

Proteome (fasta) file for *Bacillus subtilis str. 168* had total 4,174 observed proteins. DeepNovo file had total 26,687 observed peptides for *Bacillus subtilis str. 168* after removing modifications. GPMDB had total 54,069 observed peptides. We changed the leucine (L) to isoleucine (I) in the proteome (fasta) file as the DeepNovo file had only isoleucine. We also changed the leucine (L) to isoleucine (I) in the GPMDB peptides file as the DeepNovo file had only isoleucine.

We merged the DeepNovo and GPMDB files. We only kept peptides of length 6 or more. We got 18,959 matching peptides from both the files. There were many smaller peptides which were part of the larger peptides. We checked how many of these smaller peptides which are present as substrings in larger peptides and are present in two or more proteins. We found only 36 substrings (smaller peptides) which were present in two or more proteins, i.e., in one protein they were part of larger peptide, in other protein(s) they were independent of any existing peptide. We included only these 36 smaller peptides (substrings) in our proteotypic file for *Bacillus subtilis str. 168* and removed other substrings (smaller peptides) from our proteotypic file. In the end we had 14,157 proteotypic peptides.

We isolated the GPMDB peptides from Proteome (fasta) file. We digested the pieces left in the proteome (fasta) file after isolating GPMDB peptides. We removed the redundancies and kept the peptides of at least length 6. There were total 42,836 peptides left. We classified these 42,836 peptides as non-proteotypic peptides.

The Table 2.1 shows the list of observed and unobserved peptides from *Yersinia Pestis*, *Saccharomyces cerevisiae* and *Bacillus subtilis str. 168* datasets.

Table 2.1 Bacterial species protein dataset information

Organisms	*Y. Pestis*	*S. cerevisiae*	*B. subtilis*
Total peptides in identified proteins	113,472	21,514	56,993
Proteotypic peptides	8,073	2,121	14,157
Non-proteotypic peptides	105,399	19,393	42,836

Chapter 3

A Fast Peptide Classification Using LIBSVM

A support vector machine (SVM) is a supervised learning algorithm that outputs an optimal hyperplane that categorizes new observations/entries. A SVM can be used for both classification and regression. In the SVM paradigm each data point is an n dimensional vector, where n is the number of features. To achieve good classification, we select a hyperplane (in n dimensional space) that has the largest distance from the training data of all classes. By having the highest margin, the out-of-sample error is reduced.

The LIBSVM [2] library is a simple, easy-to-use, and efficient SVM classification and regression package. We are using the LIBSVM [2] library. The LIBSVM [2] library is available for many programming languages. We performed support vector classification (SVC) using the LIBSVM [2] library for C++ and used LIBSVM's [2] decision function available in the scikit-learn [10] library for Python. The study for peptide identification using Ordered Amino Acids with STEPP was done in MATLAB using it's built in SVM library by Ahmed Alqurri [11]. The MATLAB code takes around 12 hours to run. To reduce the time we wrote the same code in C++ and reduced the time by 7 times.

The LIBSVM [2] has five SVM types:

1. C-SVC (multi-class classification)

2. nu-SVC (multi-class classification)

3. One-class SVM

4. Epsilon-SVR (regression)

5. nu-SVR (regression)

We have used C-SVC and nu-SVC for classification of peptides into proteotypic and non-proteotypic peptides.

The LIBSVM [2] supports five kernel types:

1. Linear

2. Polynomial

3. Radial basis function

4. Sigmoid

5. Precomputed kernel

In general, in machine learning a kernel function is used for pattern analysis. The support vector machine (SVM) is one of the most popular pattern recognition algorithm that employs a kernel function. Kernel function transforms n dimensional feature vector (in an algorithms like SVM) to m dimensional feature space, usually m is much larger than n. Kernel function operates in high dimensional feature space, by adding new features that are the functions of existing features. Kernel functions do not calculate coordinates of the data in that high-dimensional space, instead they calculate the inner products between the images of all the data in that feature space. This approach is called the "kernel trick".

We have used linear kernel function to achieve similar results as described by Web Robertson et al. [15] and Ahmad Alqurri [11].

3.1 Motivation: Performance Issues

In previous work, Ahmad Alqurri [11] used MATLAB [9] to achieve Sensitivity of 90% and Specificity of 81% for the *Yersinia Pestis* dataset. The MATLAB code was very good for prototyping our ideas, but it soon became a bottleneck as we set on improving classification metrics. The MATLAB code for SVC with a linear kernel took approximately 12 hours to run on Intel quad core 2.67GHz processor with 7.8G RAM. Using MATLAB [9] to iterate over our method was a bit time taking. Improving performance became critical as we worked on feature selection and testing different classification algorithms on a number of peptide datasets.

We wrote a C++ code using LIBSVM [2] library to get the same results as the MATLAB [9] implementation of SVM. We have used the normalized datasets for our experiments reported in this chapter. The performance gain that we achieve over the MATLAB [9] implementation is significant. The C++ version of code with nu-SVC reports its results in under 15 minutes. Even for other SVM classifiers mentioned above, LIBSVM's performance is orders of magnitude better compared to MATLAB [9] implementation.

3.2 Specificity and Sensitivity Metrics

Specificity and sensitivity are statistical metrics to measure performance of binary classification algorithms. Sensitivity is the true positive rate (TPR) meaning the percentage of positives correctly identified. Specificity is the true negative rate meaning the percentage of negatives correctly identified. Specificity is also defined as *1 - False Positive Rate*.

Comparison of specificity and sensitivity for peptides classification with SVM using LIBSVM and MATLAB are provided in Figure 3.1. We can observe in the figure that both LIBSVM and MATLAB implementations are able to achieve the similar

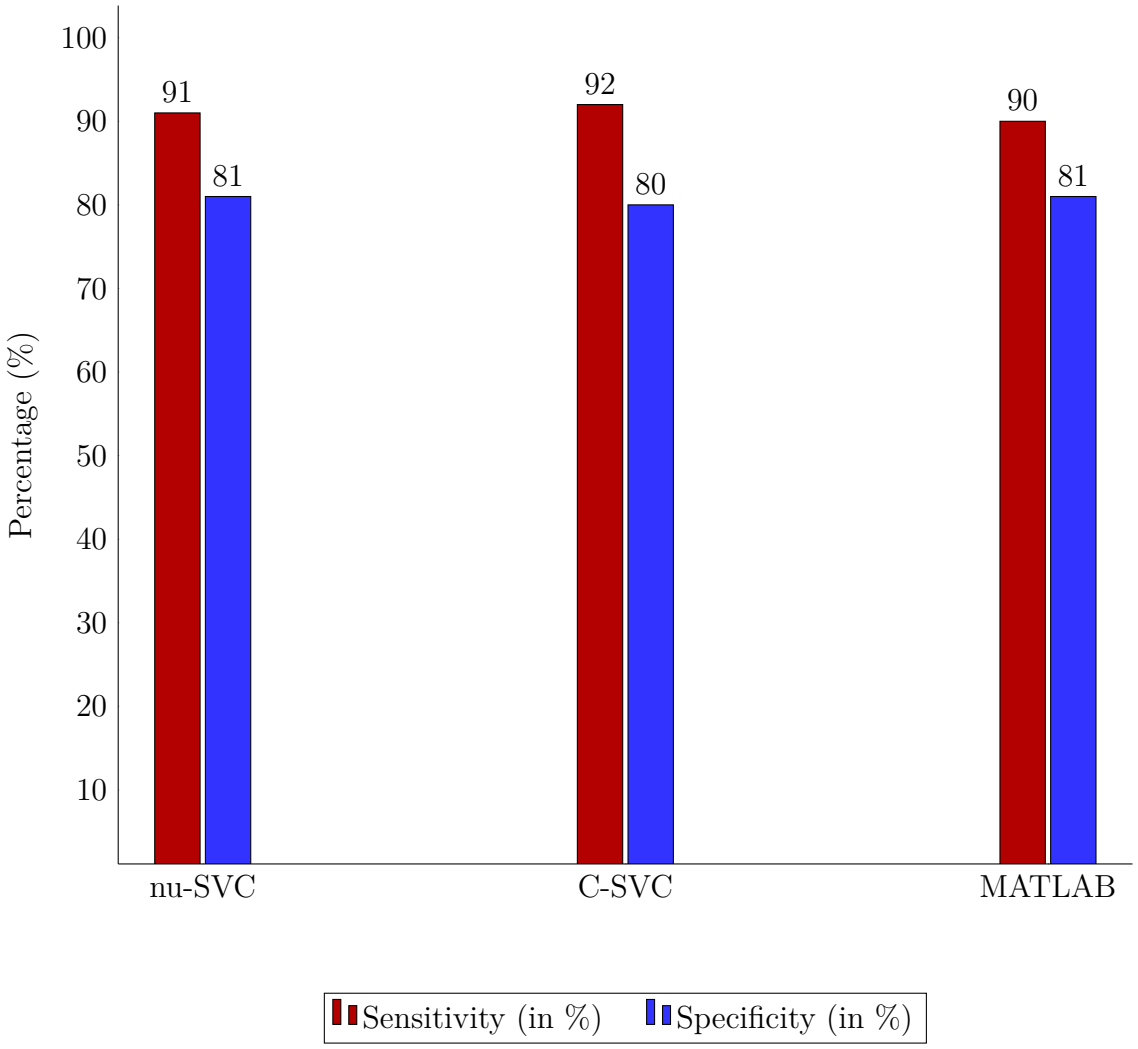


Figure 3.1 Sensitivity and Specificity comparison of LIBSVM [2] and MATLAB [9] SVM classifiers on *Yersinia Pestis* dataset.

specificity and sensitivity on *Yersinia Pestis* dataset. With MATLAB implementation, we were able to achieve a sensitivity of 90% and specificity of 81%. With the LIBSVM's nu-SVC version, we got sensitivity of 91% and specificity of 81%, which are very similar to MATLAB implementation. With the LIBSVM's C-SVC version, we got sensitivity of 92% and specificity of 80%.

3.3 Speedup with C++ code

We achieved significant speedup with C++ code. The MATLAB code took approximately 720 minutes to classify *Yersinia Pestis* dataset. With the LIBSVM's C-SVC classification, the time is reduced to 100 minutes. When we implemented the same LIBSVM's nu-SVC classification, the time was further reduced to approximately 15 minutes.

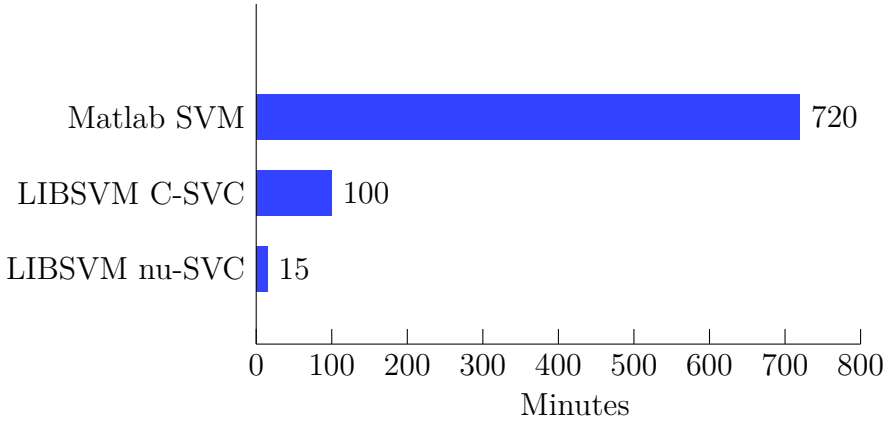


Figure 3.2 Time taken in minutes for SVM on *Yersinia Pestis* dataset using three methods

3.3.1 nu-SVC is faster than C-SVC

We found out that nu-SVC is much faster than C-SVC. nu-SVC takes parameter nu values between 0 and 1. C-SVC takes parameter C values from 0 to infinity. As the nu value for nu-SVC can be very small compared to C value for C-SVC, the processing time for nu-SVC is significantly less than C-SVC. In our model for *Yersinia Pestis* dataset, after doing grid search on parameter nu for nu-SVC and C for C-SVC, we found best results for nu = 0.31 for nu-SVC and C = 1e5 for C-SVC, the sensitivity and specificity metrics values are almost the same for both nu-SVC and C-SVC.

3.3.2 Relative speedups with both algorithms

As shown in the Figure 3.2, nu-SVC takes approximately 15 minutes to complete for *Yersinia Pestis* dataset. Whereas, C-SVC takes 100 minutes to complete for *Yersinia Pestis* dataset.

3.4 Grid Search to get optimal hyper-parameter selection

Hyper-parameters are passed as the arguments to the constructor of the estimator classes. In LIBSVM [2] support vector classifier the kernel type, degree, gamma, cost, nu are some of the examples of hyper-parameters. To get the best cross-validation score, the hyper-parameters are searched and optimized.

Grid search is a method used to search and optimize the hyper-parameters of the estimator classes. Here in case of support vector classifier for identifying peptides using ordered amino acids, we have optimized the parameter C for C-SVC and parameter nu value for nu-SVC.

3.4.1 Tuning nu-SVC

The nu-SVC takes C values in the range of 0 to 1. We started our grid search for C using five values from 0.1 to 0.9 with equal spacing. We found the best results for C = 0.3. We narrowed grid search for C between 0.2 and 0.4. This time we took 11 numbers between 0.2 and 0.4 with equal spacing for grid search. We found best sensitivity and specificity for C = 0.31. The results for Sensitivity and Specificity with C = 0.31 were similar to the Web-Robertson et al. [15] and Ahmed Alqurri [11].

3.4.2 Tuning C-SVC

C-SVC takes C values in the range of 0 to ∞. We started our grid search for C using three values, i.e, 1, 50 and 1e2. We found the best results for C = 1e2. We again did the grid search for C using three 1e2, 5e2 and 1e3. This time we get results for C =

1e3. Now we again did grid search with C equal to 1e3, 1e4 and 1e5. In this iteration we found best results for C = 1e5. The results for Sensitivity and Specificity with C = 1e5 were very close to the Web-Robertson et al. [15] and Ahmed Alqurri [11]. We stopped our grid search right there but for sanity check we did run our model with C = 1e6. With C = 1e6, the Sensitivity and Specificity went down by couple of percentage points. We observed that as we were increasing the value of C for grid search, the speed of the model was becoming slower.

3.5 Incorporating additional datasets

We incorporated the *Saccharomyces cerevisiae* dataset in our experiments and analysis. We trained SVC model on *Saccharomyces cerevisiae* using same hyper parameters as in the *Yersinia Pestis* model. We used normalized dataset (done using min-max scalar) for *Saccharomyces cerevisiae*, similar to *Yersinia Pestis* dataset. We did 10-fold cross validation. The results were even better than *Yersinia Pestis* dataset. We achieved the sensitivity of 97% and specificity of 92% from the *Saccharomyces cerevisiae* dataset.

Further we tested our existing SVM classifier trained on *Yersinia Pestis* dataset on *Saccharomyces cerevisiae* dataset but we were getting sensitivity score of approximately 0%. Similarly when we tested the SVM classifier trained on *Saccharomyces cerevisiae* with *Yersinia Pestis* dataset, the specificity was down to approximately 0%. We assumed that the features which we are using for training SVM classifier here are only giving us the good results for the test sample data taken from the same datasets.

To get the best results, we did the feature selection from scratch. We used four feature selection methods:

1. Univariate Analysis

Table 3.1 SVC trained on sample balanced 3-AAU datasets and tested on 20% unbalanced datasets

Feature Selection	Number of Features	Classifier	Training data	Testing data	Sensitivity	Specificity
Alqurri [11]	7	SVC	Sample YP	20% YP	96%	78%
XGBoost	6	SVC	Sample YP	20% YP	94%	79%
RFE	6	SVC	Sample YP	20% YP	94%	79%
Univariate	6	SVC	Sample YP	20% YP	95%	79%
Alqurri [11]	7	SVC	Sample SC	20% SC	97%	92%
XGBoost	6	SVC	Sample SC	20% SC	98%	93%
RFE	6	SVC	Sample SC	20% SC	97%	93%
Univariate	6	SVC	Sample SC	20% SC	97%	92%

2. Recursive Feature Elimination

3. XGBoost feature importance

4. Principal Component Analysis

These feature selection methods are described in detail in Chapter 4.

We have used Scikit-Learn [10] library for python to do further experiments and analysis. We started off with the *Yersinia Pestis* datasets. We divided the *Yersinia Pestis* data in to train and test data by 80:20 ratio. We then took balanced data from 80% dataset. We did the feature selection by three different ways: Univariate Analysis, Recursive Feature Elimination and XGBoost feature importance. For feature selection we used whole data for *Yersinia Pestis*.

We first trained the SVM classification model using balanced training data. We started off with linear kernel. We again did the grid-search on C using scikit-learn GridSearchCV class with 5 fold cross-validation. We got best results for C = 1e3. We tested our model on unbalanced 20% of the *Yersinia Pestis* dataset as well as on full *Yersinia Pestis* dataset. We repeated similar steps for *Saccharomyces cerevisiae* dataset. The Table 3.1 shows SVC results for model trained on sample balanced 3-

Table 3.2 SVC trained on sample balanced 3-AAU datasets and tested on full unbalanced datasets

Feature Selection	Number of Features	Classifier	Training data	Testing data	Sensitivity	Specificity
Alqurri [11]	7	SVC	Sample YP	Full YP	95%	78%
XGBoost	6	SVC	Sample YP	Full YP	94%	79%
RFE	6	SVC	Sample YP	Full YP	94%	79%
Univariate	6	SVC	Sample YP	Full YP	95%	79%
Alqurri [11]	7	SVC	Sample SC	Full SC	97%	92%
XGBoost	6	SVC	Sample SC	Full SC	97%	93%
RFE	6	SVC	Sample SC	Full SC	97%	93%
Univariate	6	SVC	Sample SC	Full SC	97%	92%

AAU datasets and tested on 20% sample unbalanced datasets. The Figure 3.3 shows ROC curve with AUC scores for the SVC model trained on sample balanced 3-AAU datasets and tested on 20% sample unbalanced datasets.

In the next step, we trained the model on sample balanced *Yersinia Pestis* dataset and tested on full *Yersinia Pestis*s dataset. We have used 5-fold cross-validation. We repeated the same steps for *Saccharomyces cerevisiae*. The Table 3.2 shows SVC on 3-AAU datasets which is trained on sample balanced datasets but tested on full datasets using 5-fold cross-validation. The Figure 3.4 shows ROC curve with AUC scores for the SVC model trained on sample balanced 3-AAU datasets and tested on full datasets.

After getting good results from models trained on sample balanced dataset, we trained the model on unbalanced sample of *Yersinia Pestis* and *Saccharomyces cerevisiae* datasets. For this we trained and tested our models on datasets with 80:20 ratio. We used the same hyper parameters as was used earlier for training sample balanced data. We also tested models with full datasets using 5-fold cross-validation. The Table 3.3 and Table 3.4 shows the SVC models trained on 80% unbalanced datasets and tested on 20% data and full data respectively using cross-validation. The Fig-

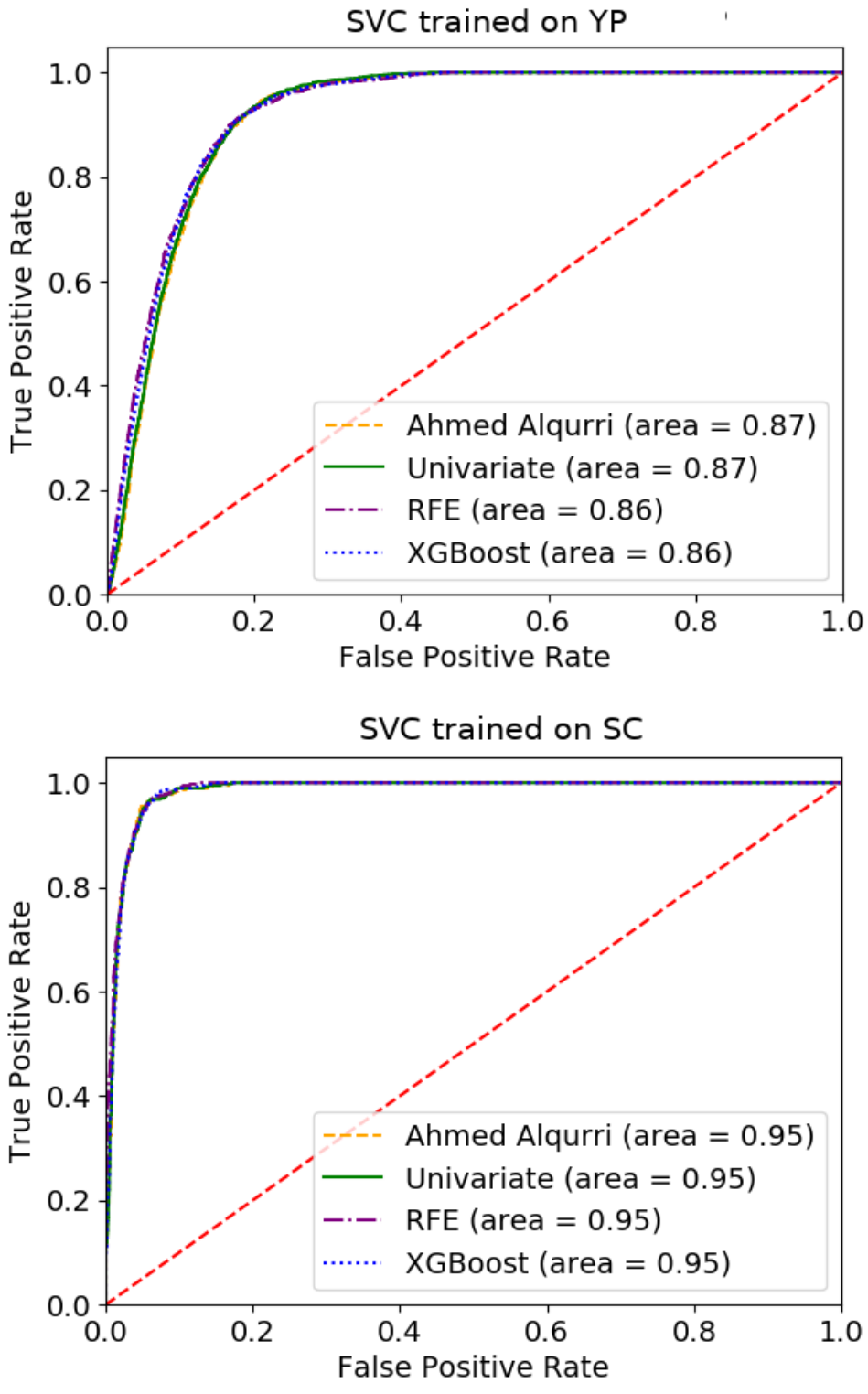


Figure 3.3 ROC curve for SVC trained on sample balanced 3-AAU datasets and tested on 20% unbalanced datasets using four different features set

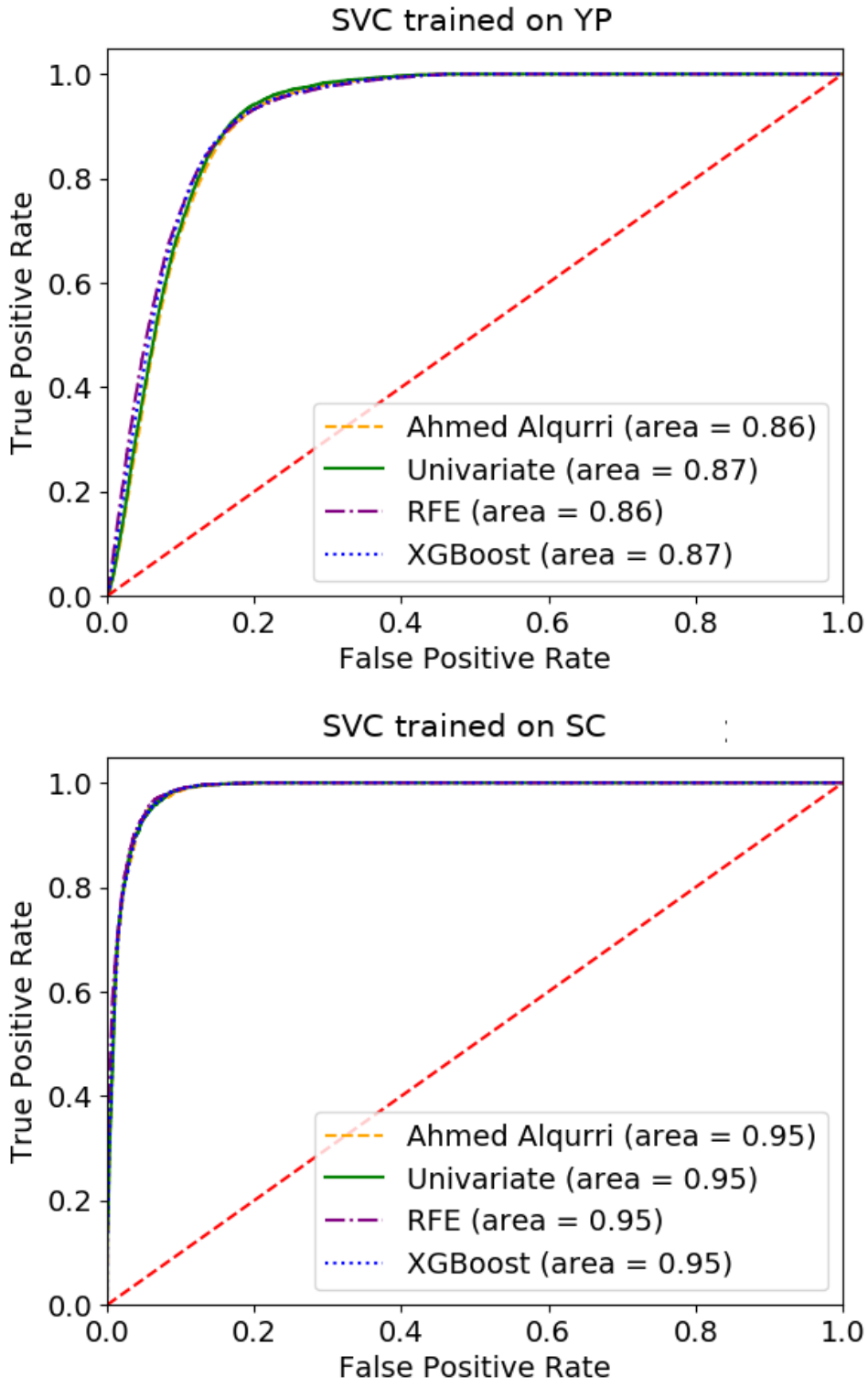


Figure 3.4 ROC curve for SVC trained on sample balanced 3-AAU datasets and tested on full unbalanced datasets using four different features set

Table 3.3 SVC trained on sample unbalanced 3-AAU datasets and tested on 20% unbalanced datasets

Feature Selection	Number of Features	Classifier	Training data	Testing data	Sensitivity	Specificity
Alqurri [11]	7	SVC	Sample YP	20% YP	96%	78%
XGBoost	6	SVC	Sample YP	20% YP	94%	79%
RFE	6	SVC	Sample YP	20% YP	94%	79%
Univariate	6	SVC	Sample YP	20% YP	95%	79%
Alqurri [11]	7	SVC	Sample SC	20% SC	97%	92%
XGBoost	6	SVC	Sample SC	20% SC	98%	93%
RFE	6	SVC	Sample SC	20% SC	97%	93%
Univariate	6	SVC	Sample SC	20% SC	97%	92%

Table 3.4 SVC trained on sample unbalanced 3-AAU datasets and tested on full unbalanced datasets

Feature Selection	Number of Features	Classifier	Training data	Testing data	Sensitivity	Specificity
Alqurri [11]	7	SVC	Sample YP	Full YP	95%	78%
XGBoost	6	SVC	Sample YP	Full YP	94%	79%
RFE	6	SVC	Sample YP	Full YP	94%	79%
Univariate	6	SVC	Sample YP	Full YP	95%	79%
Alqurri [11]	7	SVC	Sample SC	Full SC	97%	92%
XGBoost	6	SVC	Sample SC	Full SC	97%	93%
RFE	6	SVC	Sample SC	Full SC	97%	93%
Univariate	6	SVC	Sample SC	Full SC	97%	92%

ure 3.5 and Figure 3.6 shows ROC curve with AUC scores for the SVC trained on 80% sample unbalanced 3-AAU datasets and tested on 20% sample datasets and full datasets.

We then incorporated both the datasets from *Yersinia Pestis* and *Saccharomyces cerevisiae* to test our models. We found that our model trained on only 6 features of *Yersinia Pestis* is not classifying *Saccharomyces cerevisiae* dataset that well for 3-AAU datasets. The AUC score when we were training and testing using datasets from two different species was approximately 50% for both the models. We again

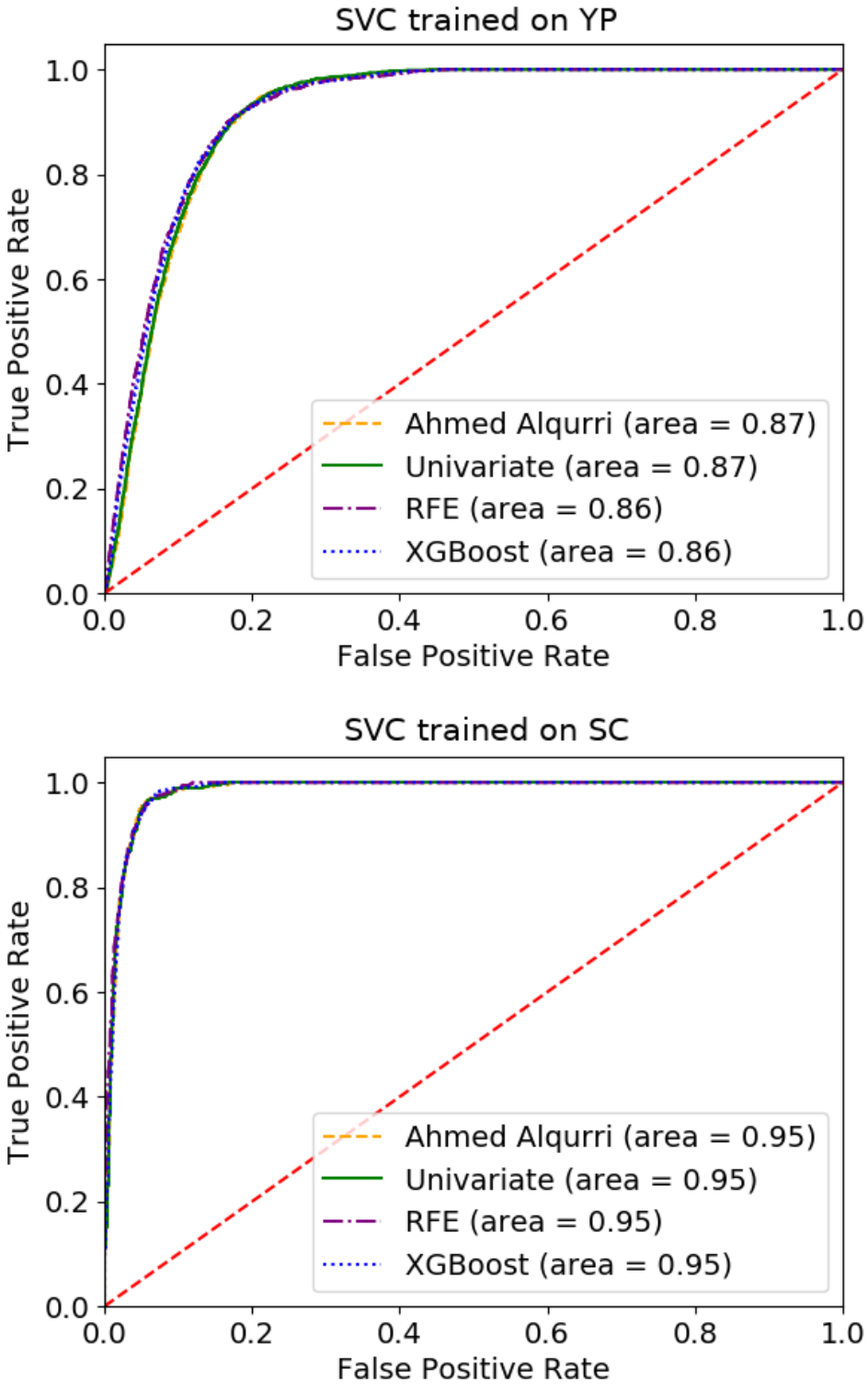


Figure 3.5 ROC curve for SVC trained on sample unbalanced 3-AAU datasets and tested on 20% unbalanced datasets using four different features set

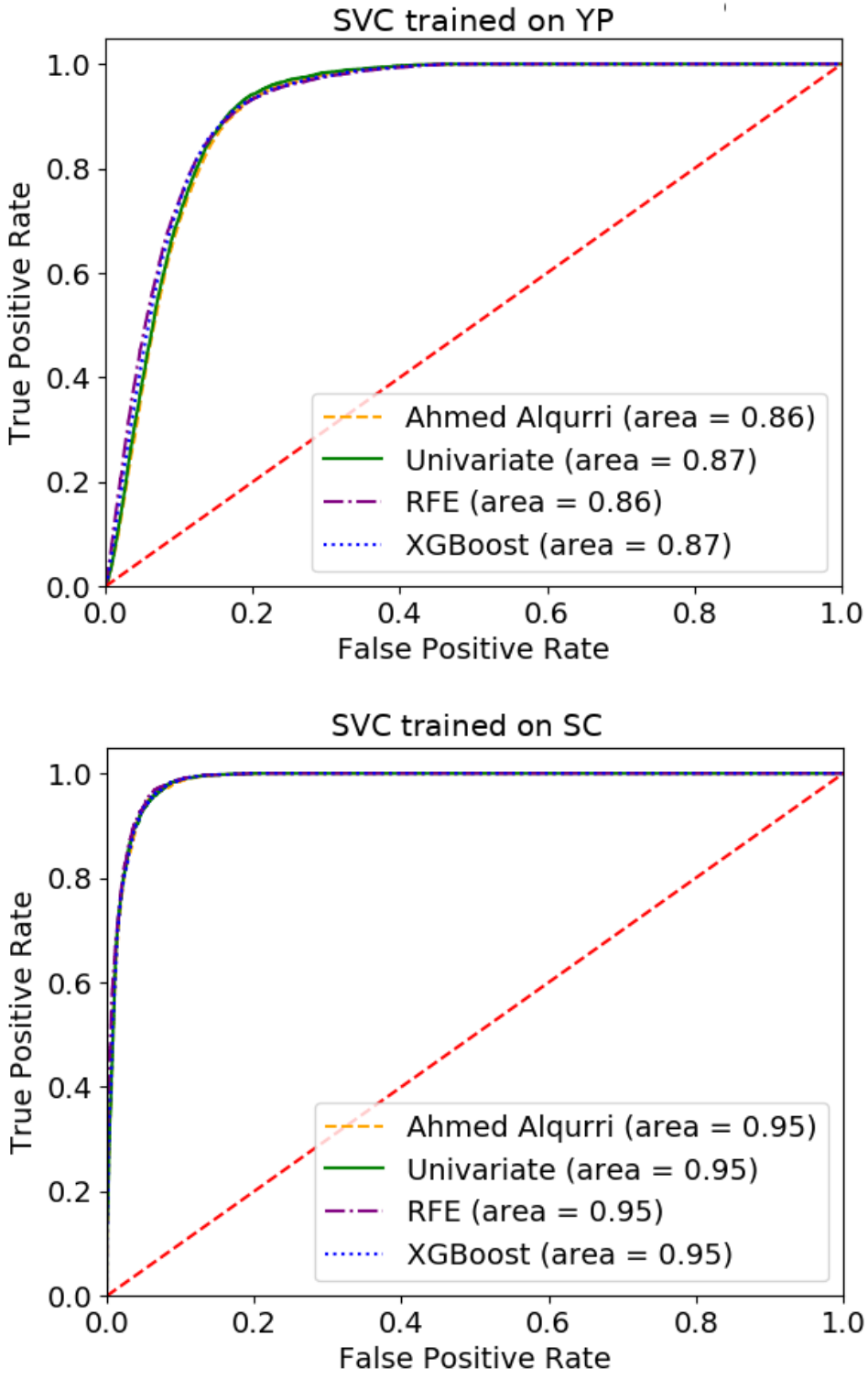


Figure 3.6 ROC curve for SVC trained on sample unbalanced 3-AAU datasets and tested on full unbalanced datasets using four different features set

Table 3.5 SVC on 3-AAU datasets trained and tested on datasets from different species with three different feature selection methods and using RBF kernel

Feature Selection	Number of features	Classifier	Training data	Testing data	Sensitivity	Specificity
XGBoost	13	SVC	Sample YP	Full SC	90%	82%
XGBoost	13	SVC	Sample YP	Full YP	91%	73%
XGBoost	13	SVC	Sample SC	Full SC	95%	92%
XGBoost	13	SVC	Sample SC	Full YP	86%	74%
RFE	13	SVC	Sample YP	Full SC	90%	82%
RFE	13	SVC	Sample YP	Full YP	90%	74%
RFE	13	SVC	Sample SC	Full SC	96%	92%
RFE	13	SVC	Sample SC	Full YP	86%	73%
Univariate	13	SVC	Sample YP	Full SC	86%	83%
Univariate	13	SVC	Sample YP	Full YP	91%	73%
Univariate	13	SVC	Sample SC	Full SC	96%	92%
Univariate	13	SVC	Sample SC	Full YP	87%	71%

performed feature selection using XGBoost feature importance, Univariate and RFE on combined features for *Yersinia Pestis* and *Saccharomyces cerevisiae*. We did this to make sure that both the models are trained on the same feature sets.

We again performed grid search with 5-fold cross-validation on kernel, C and gamma. For both the *Yersinia Pestis* and *Saccharomyces cerevisiae* dataset, we get best results for Radial Basis Function (RBF) kernel with C = 0.1 and gamma = auto. We have also used the 5-fold cross-validation. We have tested the model on the dataset from different species. The Table 3.5 shows Support Vector Classification using different feature selection methods, tested on datasets from different species.

The Figures 3.7, 3.8 and 3.9 show the AUC scores for the SVC done on the normalized datasets using XGBoost feature importance analysis, Recursive feature elimination and Univariate Analysis. We have used 13 features from all three feature selection methods to achieve these results.

We also performed principal component (PCA) analysis for feature reduction on both *Yersinia Pestis* and *Saccharomyces cerevisiae* datasets. In PCA analysis we

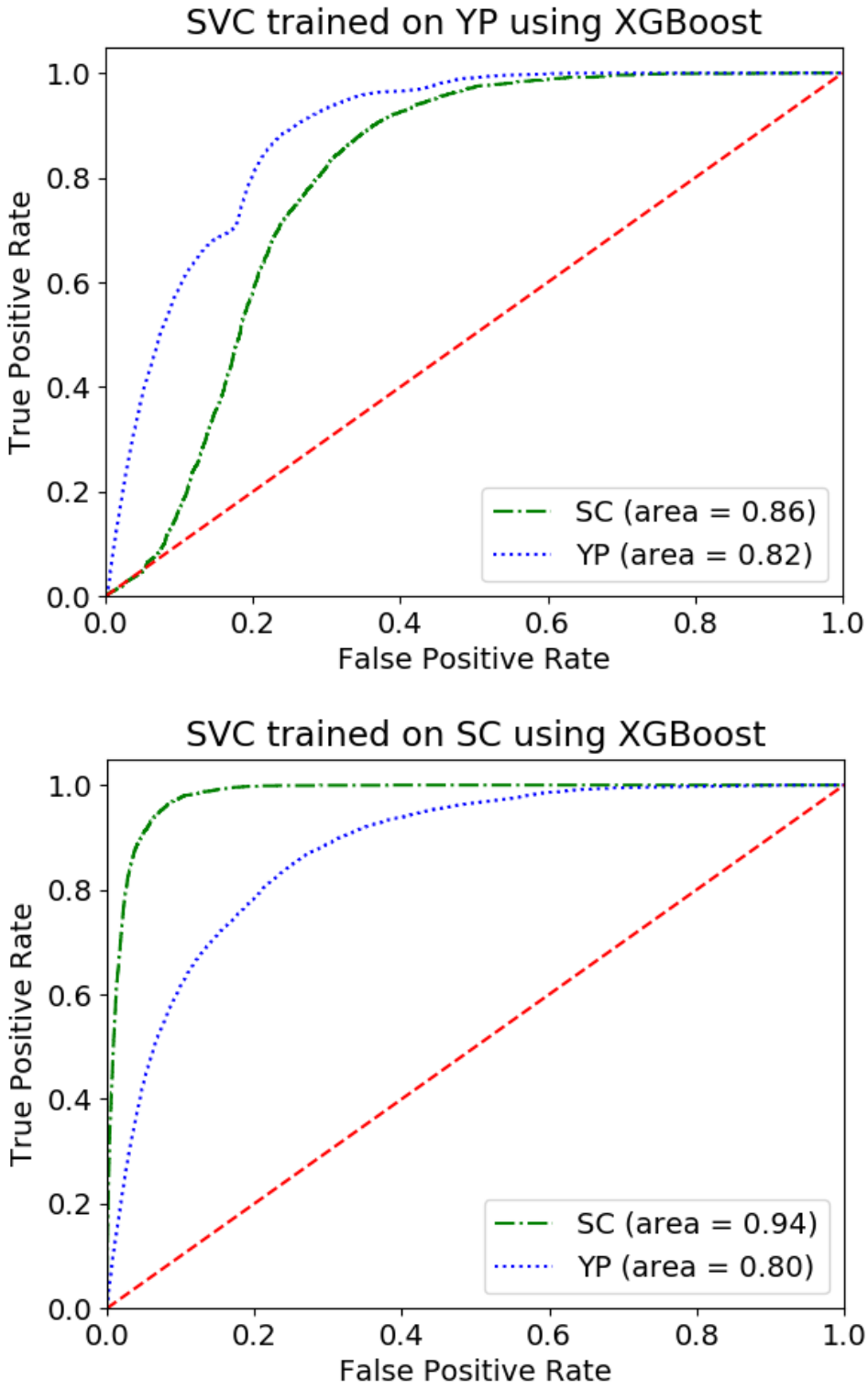


Figure 3.7 ROC curve for SVC trained on sample unbalanced 3-AAU datasets and tested on full unbalanced datasets using XGBoost feature importance analysis

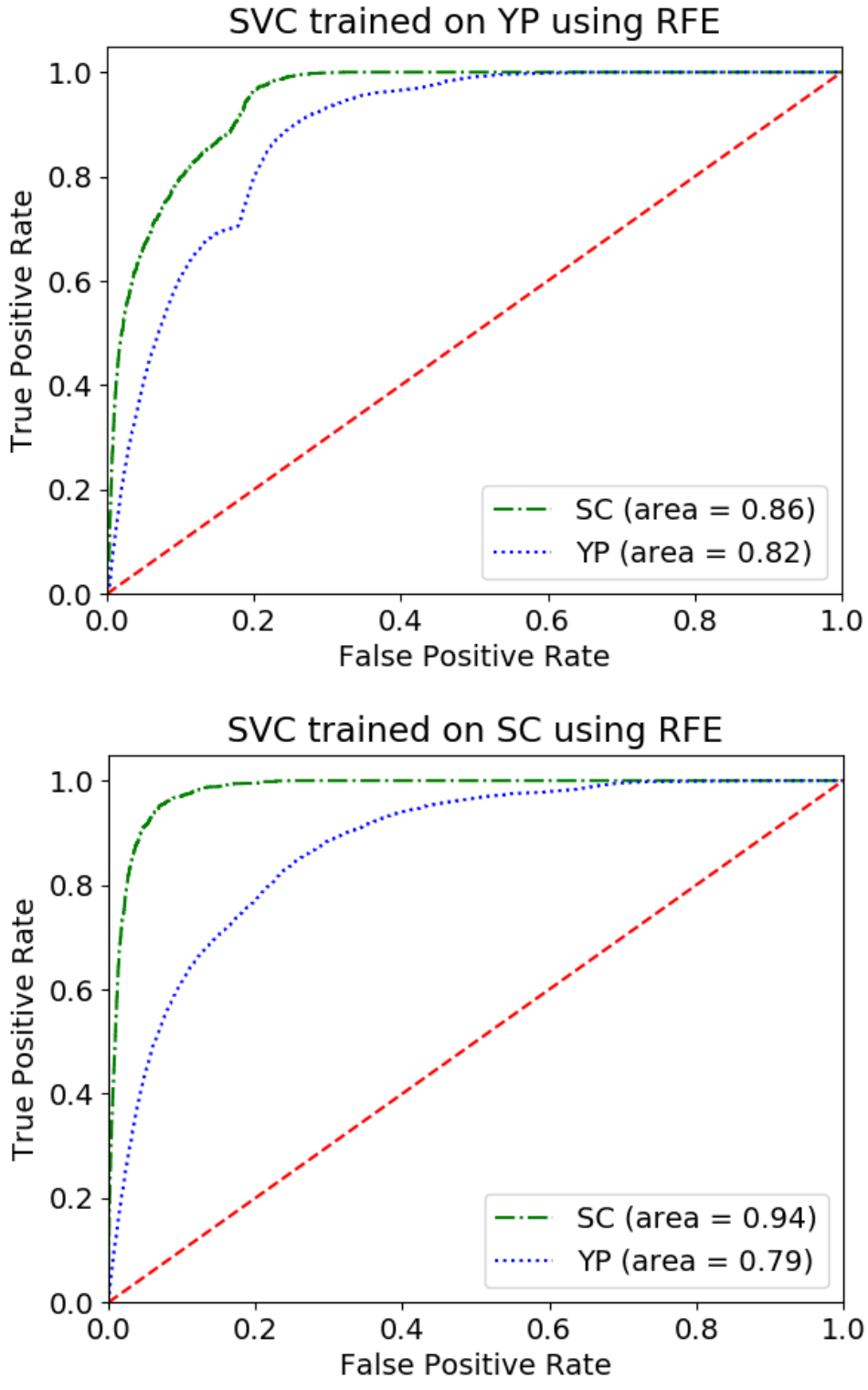
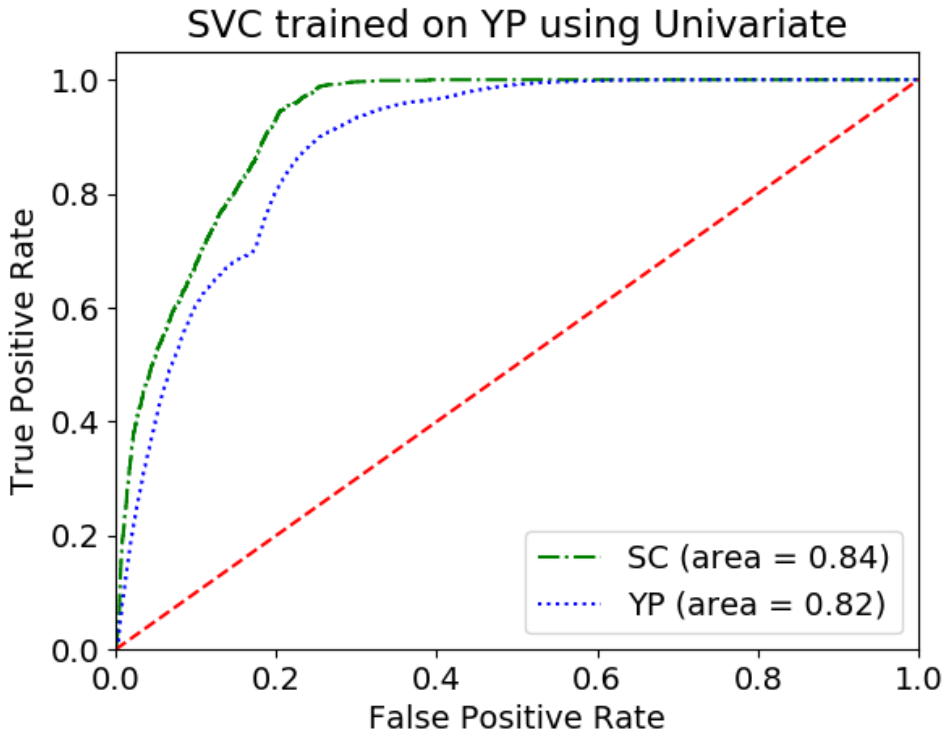


Figure 3.8 ROC curve for SVC trained on sample unbalanced 3-AAU datasets and tested on full unbalanced datasets using Recursive Feature Elimination (RFE) analysis



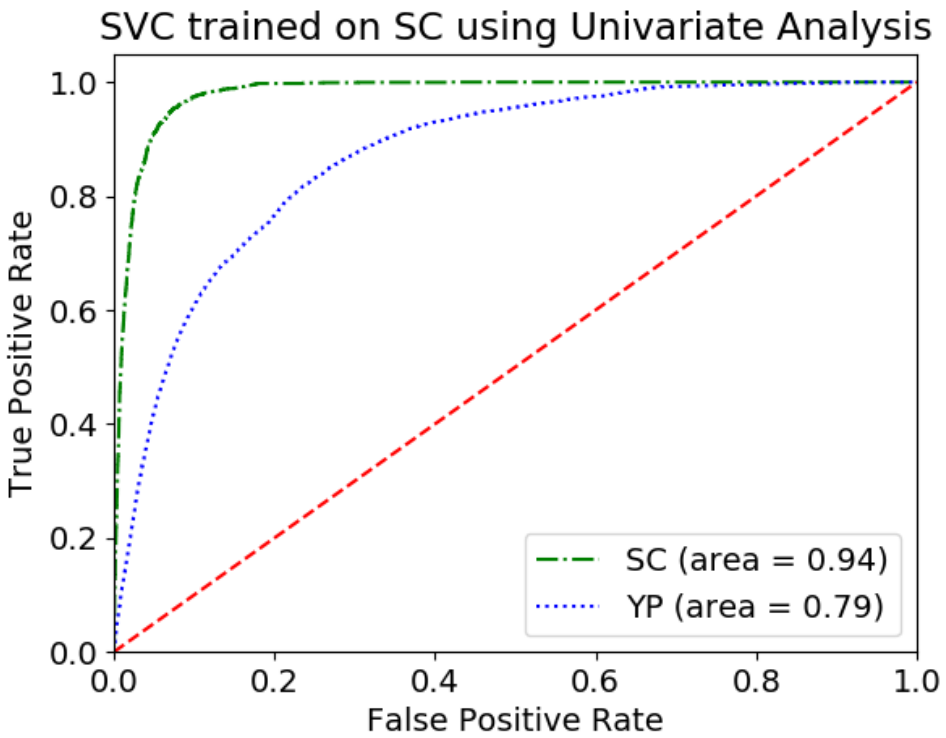


Figure 3.9 ROC curve for SVC trained on sample unbalanced 3-AAU datasets and tested on full unbalanced datasets using Univariate analysis

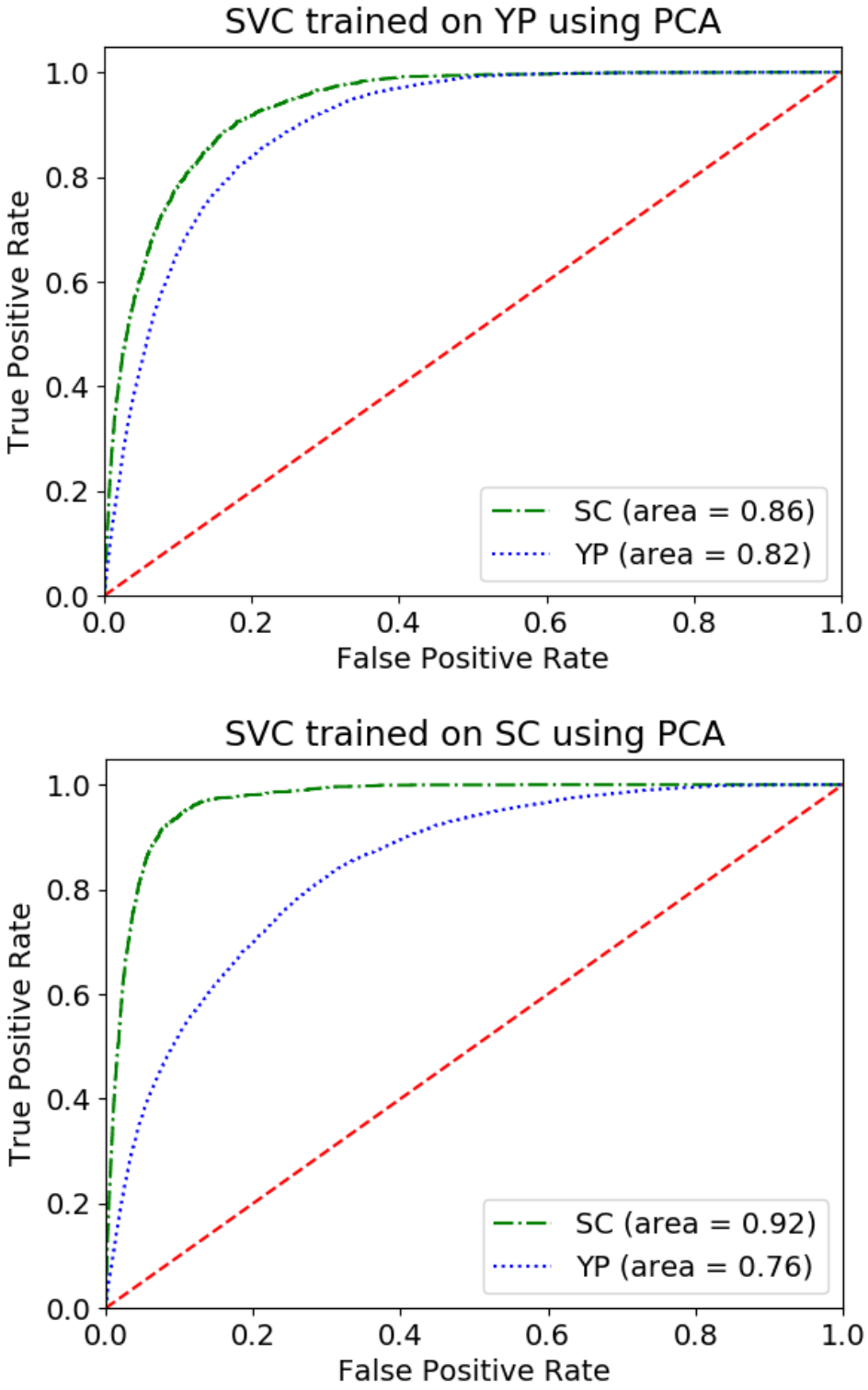


Figure 3.10 ROC curve for SVC using 8 Principal Component Analysis (PCA). Model trained on unbalanced sample 3-AAU datasets and tested on full unbalanced datasets of both the species

Table 3.6 SVC on 3-AAU datasets tested on datasets from different species with 8 principal components and RBF kernel

Feature Selection	Principal Components	Classifier	Training data	Testing data	Sensitivity	Specificity
PCA	8	SVC	Sample YP	Full SC	88%	84%
PCA	8	SVC	Sample YP	Full YP	89%	75%
PCA	8	SVC	Sample SC	Full SC	95%	89%
PCA	8	SVC	Sample SC	Full YP	86%	66%

used all 3-AAU dataset features to get 8 principal components that gave us good results. The 8 principal components from *Yersinia Pestis* model covered 84% of the variance. For the model trained on the *Saccharomyces cerevisiae* dataset, 80% of the variance was covered. For SVM classification, we have used RBF kernel with C = 1 and gamma = auto. We have used the 5-fold cross-validation. We have tested the model with both the datasets.

The Table 3.6 shows Support Vector Classification (SVC) using 8 principal components on 3-AAU datasets and tested on datasets from different species. The Figure 3.10 shows ROC curve with AUC score for Support Vector Classification shown in Table 3.6.

Chapter 4

Feature Selection

Feature selection demonstrates that only small set of features are required for correct prediction of proteotypic peptides. Feature selection is also very important to decrease cost of running models on very big data. Web-Robertson et al. [15] performed support vector classification using all the features but they did provide Fisher Criterion Score (FCS) [1] for each feature. They also mentioned that less number number of features would also provide good prediction of proteotypic peptides. Web-Robertson et al. [15] didn't perform feature selection through algorithms like recursive feature elimination (RFE) because of high cost of computation required to run them on big data.

Ahmed Alqurri [11] performed support vector classification only on 7 features. Alqurri used linear discriminate analysis (LDA) and examined LDA loadings to see the contributions of each feature. After examining LDA loadings, they came up with 7 features. We decided to do features selection exhaustively.

We performed four types of feature selections and did classification using all these sets of features. In this chapter we have used normalized datasets (done through min-max scalar) for feature selection. The four feature selection method we used are as below -

1. Univariate Analysis
2. Recursive Feature Elimination
3. XGBoost feature importance
4. Principal Component Analysis

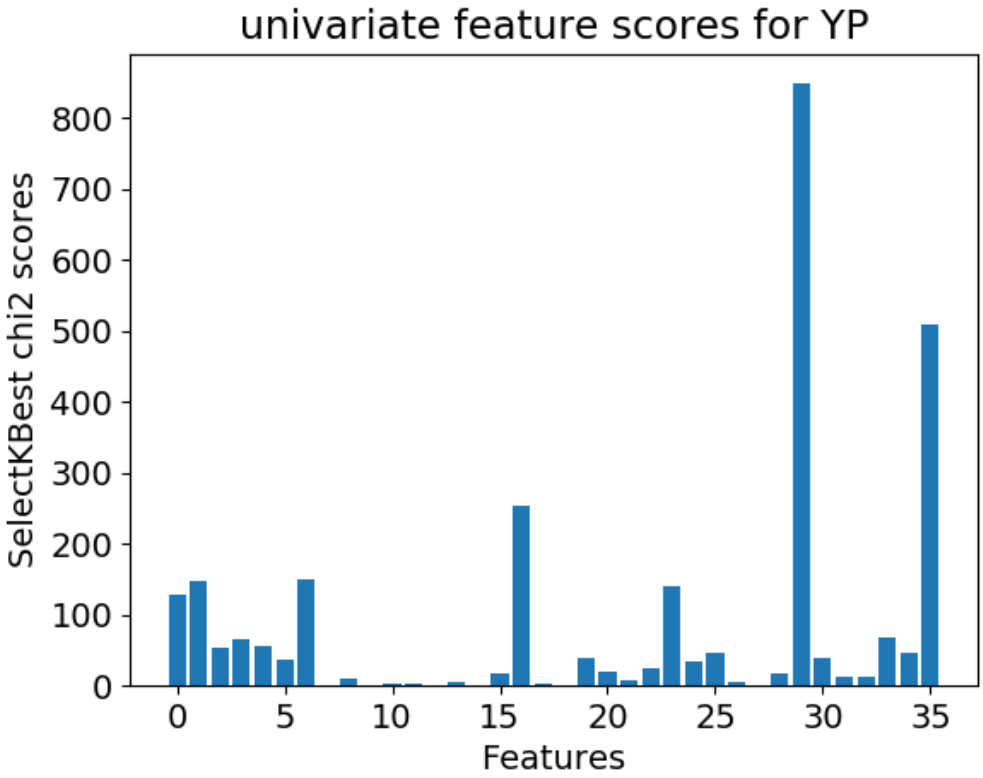


Figure 4.1 Chi2 score for feature selection on 3-AAU *Yersinia Pestis* dataset

Univariate feature selection is based on univariate statistical analysis. Univariate analysis deals with only one variable at a time. It doesn't deal with the relationship between features or variables. Univariate analysis is used to summarize data. The scikit-learn [10] Python library provides *SelectKBest* class that can be used with different statistical tests to select a specific number of features. *SelectKBest* returns a subset of the highest scoring features.

We have used the basic chi-squared test as a scoring function with the scikit-learns [10] *SelectKBest* class to select features. The chi-squared statistic or χ^2 test removes features which are mostly independent of the class and therefore irrelevant for classification. The chi-squared statistic or χ^2 test was devised by Karl Pearson

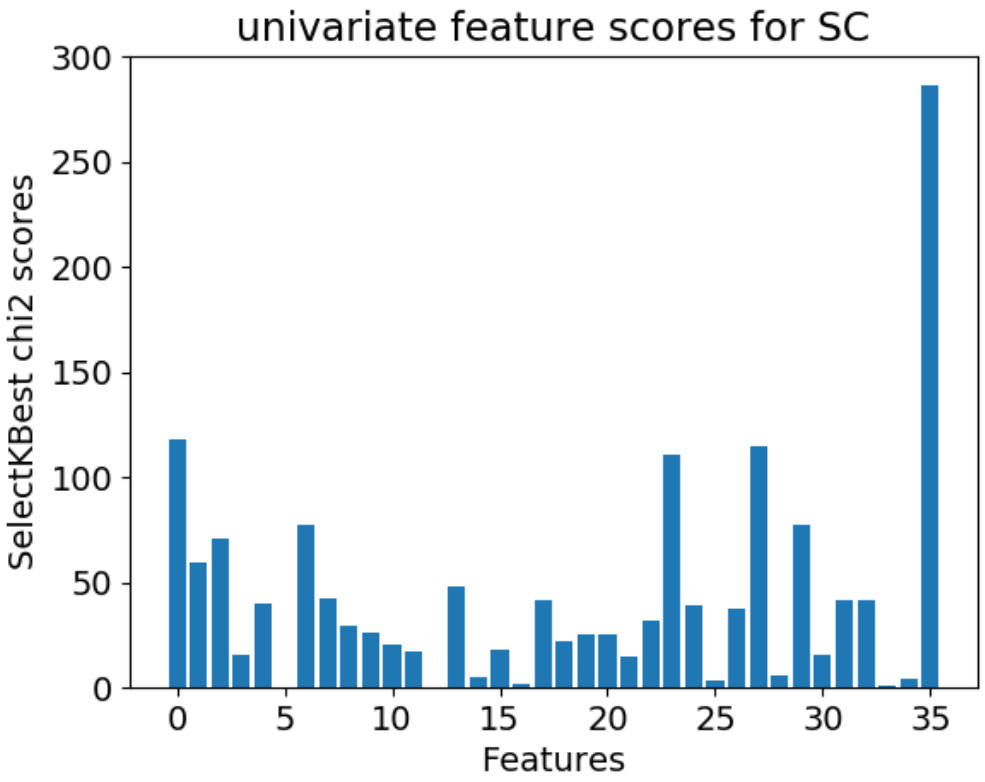


Figure 4.2 Chi2 score for feature selection on 3-AAU *Saccharomyces cerevisiae* dataset

[6] in 1900. The chi-squared test is a statistical hypothesis test where the statistical distribution of the test statistic is a chi-squared distribution where the null hypothesis is true. The null hypothesis in case of a chi-squared test is defined as the hypothesis that states there is no significant difference between expected and observed data.

In our case, we are performing a chi-squared test for independence to test if a particular feature in our data is independent of the class. If that feature is independent of the class then we remove that feature from the classification. The chi-squared statistic is a number that signifies whether the observed value would be significantly different from the expected value if there was no relationship. If the chi-squared statistic is low then it signifies that there is a relationship between a feature and the class. In case of high chi-squared statistic, it signifies there in no relationship between

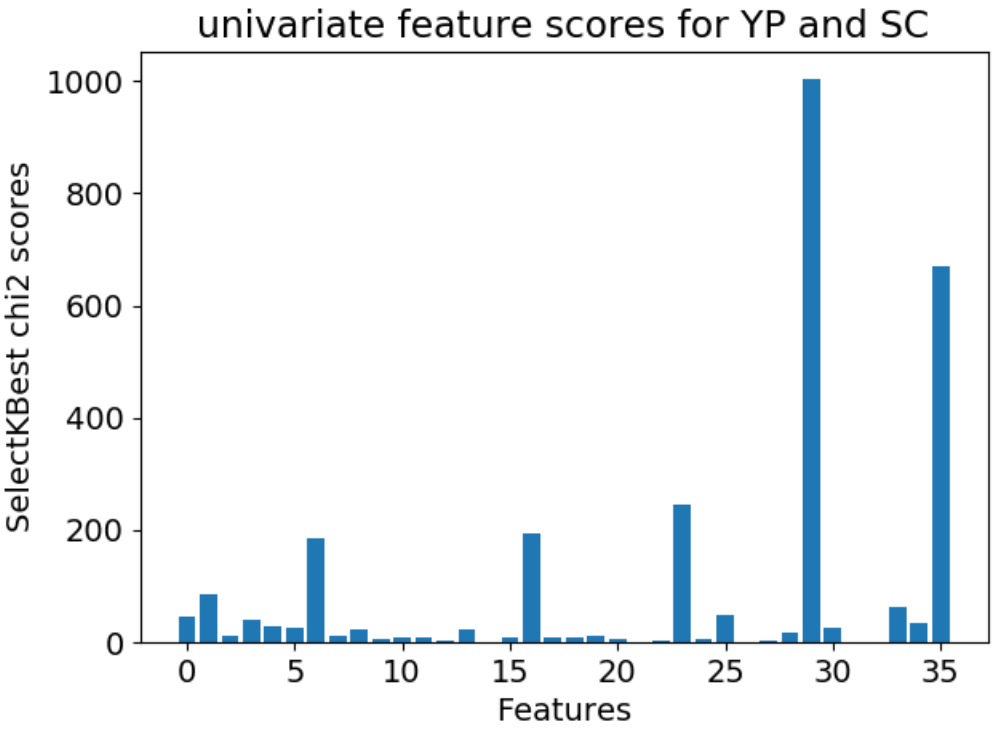


Figure 4.3 Chi2 score for feature selection on 3-AAU *Yersinia Pestis* and *Saccharomyces cerevisiae* dataset

a feature and class.

The chi-squared statistic for the chi-squared test is calculated by the formula -

$$\chi_f^2 = \sum \frac{(O_i - E_i)^2}{E_i} \tag{4.1}$$

where f is the degree of freedom, O is the observed value and E is the expected value.

We performed Univariate analysis for *Yersinia Pestis*, *Saccharomyces cerevisiae* and dataset containing the features from both *Yersinia Pestis* and *Saccharomyces cerevisiae*. The Figures 4.1, 4.2 and 4.3 show the feature selection scores from univariate analysis we have done on three datasets. From the univariate analysis we can observe that number of proline (P) residues and ordered amino acid usage (3-AAU) are the two top features.

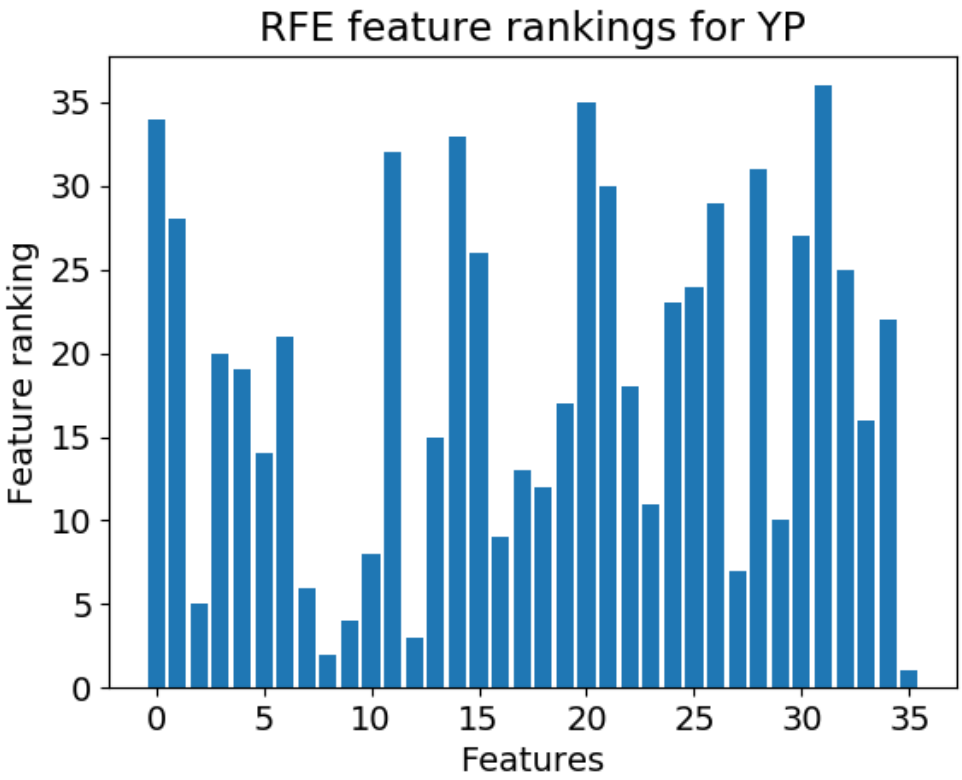


Figure 4.4 RFE feature ranks for 3-AAU *Yersinia Pestis* dataset

Recursive Feature Elimination (RFE) is a multivariate feature selection method. RFE removes features recursively and build the model using the remaining features that are left behind. RFE uses an external estimator to build a model that assigns weights to features. It ranks the features either through coef_ attribute or through feature_importance_ score. The features with least scores are removed from current set of features. This process is repeated recursively with reduced features until the required set of features are selected. As such, this is a greedy algorithm to select the best performing features. Scikit-learn's [10] provides RFE class under feature_selection library. We implemented RFE with logistic regression to rank features by weights.

We did Recursive Feature Elimination (RFE) for *Yersinia Pestis*, *Saccharomyces*

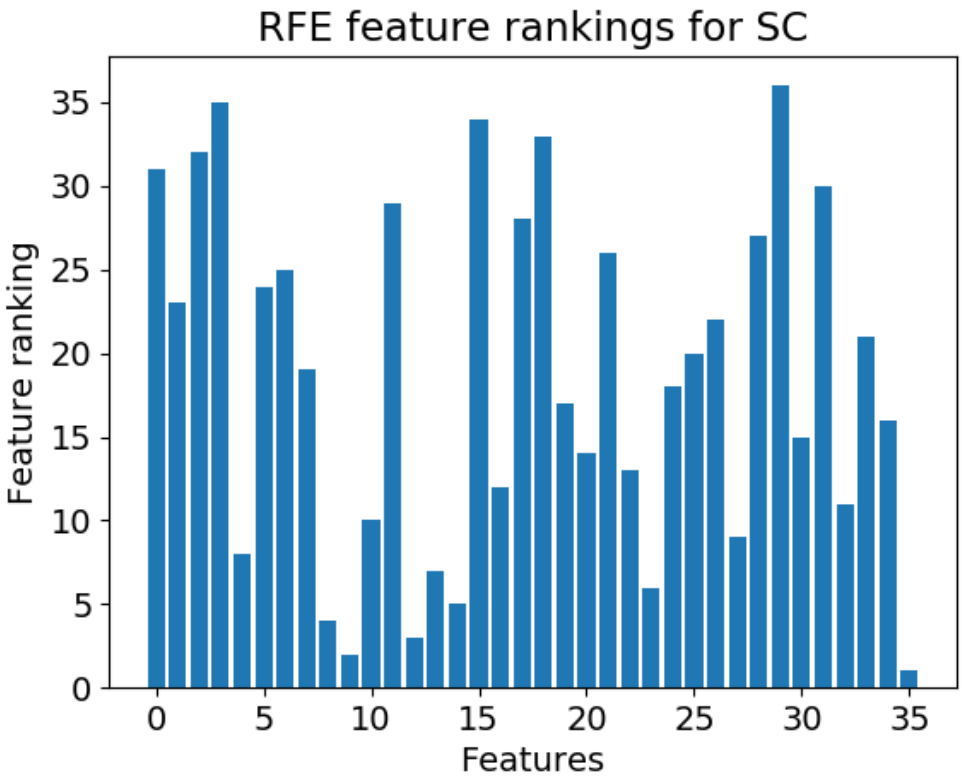


Figure 4.5 RFE feature ranks for 3-AAU *Saccharomyces cerevisiae* dataset

cerevisiae and combination of calculated features from both *Yersinia Pestis* and *Saccharomyces cerevisiae*. The Figures 4.4, 4.5 and 4.6 show the feature rankings for Recursive Feature Elimination (RFE) we have done on three datasets. For both *Yersinia Pestis* and *Saccharomyces cerevisiae*, feature number 35 i.e, Ordered Amino Acid (3-AAU) ranked at the top followed by Hydrophobicity-Eisenberg scale (Eisenberg et al., 1984), Hydrophilicity-Hopp-Woods scale (Hopp and Woods, 1981) and Polarity-Grantham scale (Grantham, 1974). For the combined calculated features dataset for *Yersinia Pestis* and *Saccharomyces cerevisiae*, ordered amino acid (3-AAU) still tops the rank, but second, third and fourth position goes to Number of positively charged polar hydrophilic residues, Hydropathicity-Roseman scale (Roseman, 1988) and Polarity-Grantham scale (Grantham, 1974) respectively.

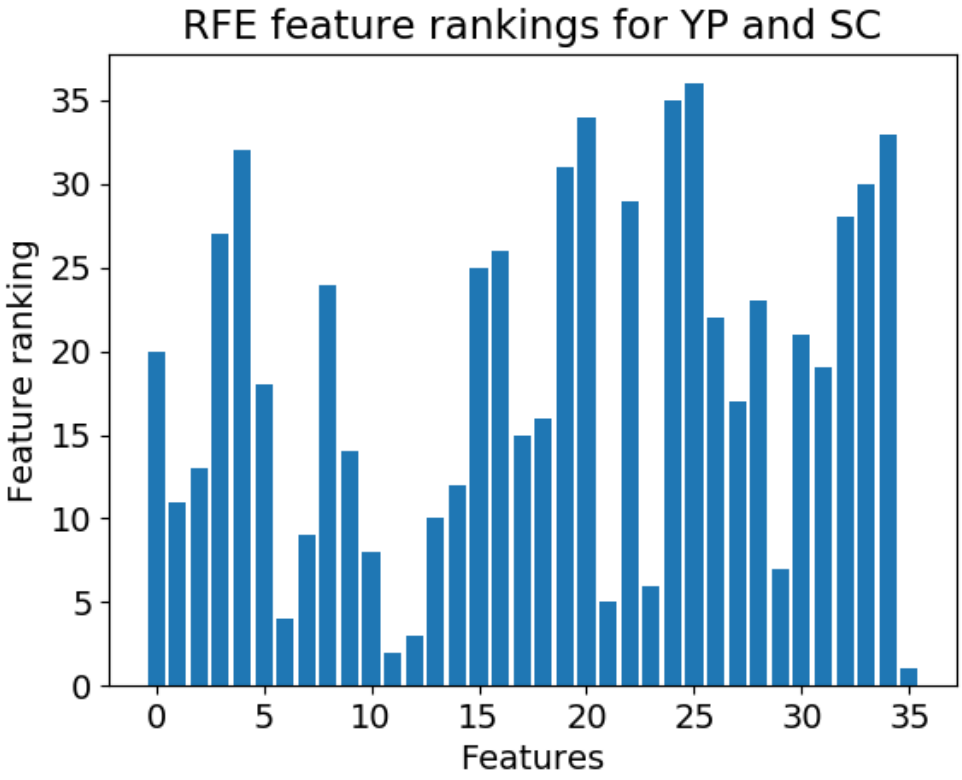


Figure 4.6 RFE feature ranks for on 3-AAU *Yersinia Pestis* and *Saccharomyces cerevisiae* dataset

4.3 XGBoost feature importance

XGBoost stands for Extreme Gradient Boosted trees. XGBoost is a supervised learning technique used for classification and regression. More details are given in section 5.4.

XGBoost library [3] in python provides a very useful function feature_importance_ for trained model. These importance scores are calculated when the model is getting trained. These importance scores are F scores for each feature. These importance scores can be calculated by three types [3]: 'weight', 'gain' and 'cover'. 'weight' is the number of times a feature is used to split the data across all trees [3]. 'gain' is the average gain of the feature when it is used in trees [3]. 'cover' is the average coverage

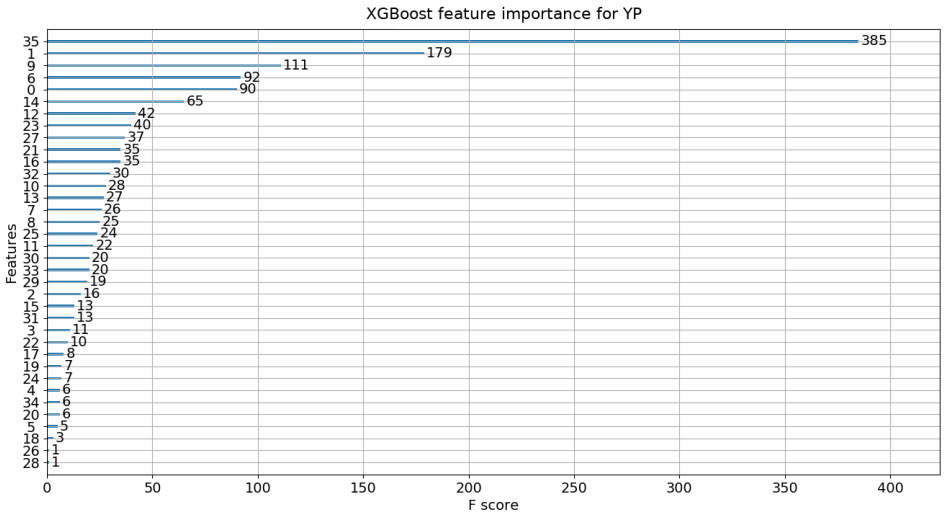


Figure 4.7 XGBoost feature importance for 3-AAU *Yersinia Pestis* dataset

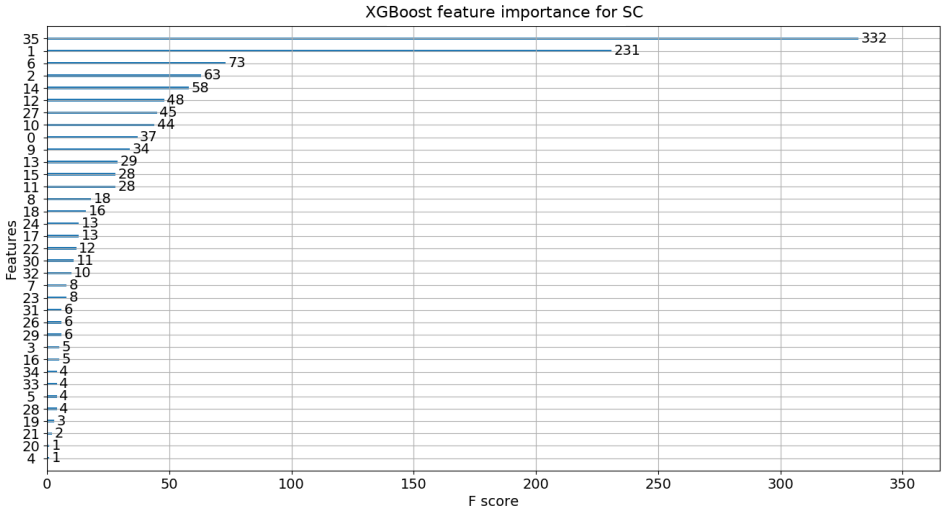


Figure 4.8 XGBoost feature importance for 3-AAU *Saccharomyces cerevisiae* dataset

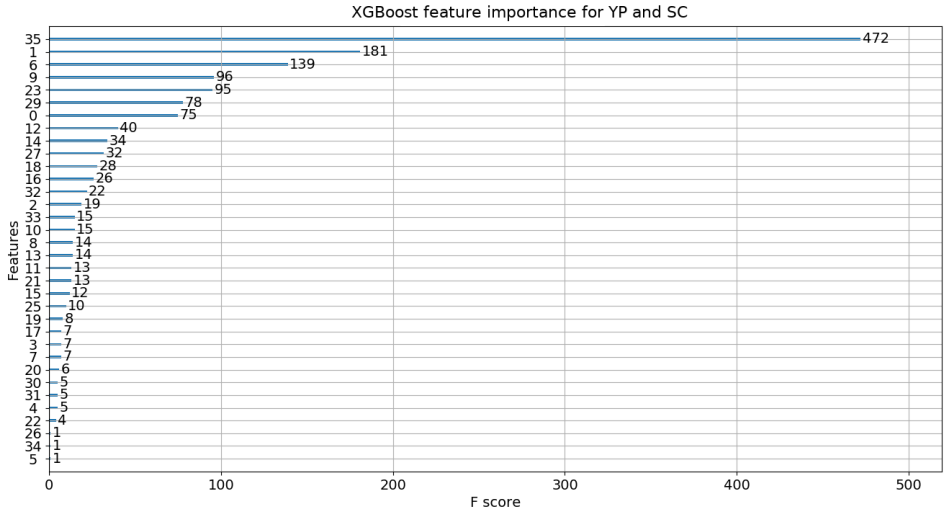


Figure 4.9 XGBoost feature importance for on 3-AAU *Yersinia Pestis* and *Saccharomyces cerevisiae* dataset

of the feature when it is used in trees [3]. We have used 'weight' as the importance type to calculate the feature importance score.

We generated XGBoost feature importance scores for *Yersinia Pestis*, *Saccharomyces cerevisiae* and combined features from both *Yersinia Pestis* and *Saccharomyces cerevisiae* datasets. The Figures 4.7, 4.8 and 4.9 show the feature importance scores from XGBoost models we ran three datasets. For all the three datasets, feature number 35 and 1 i.e, Ordered Amino Acid (3-AAU) and molecular weights of the peptides have the highest scores.

4.4 Principal Component Analysis

Principal Component Analysis (PCA) is an unsupervised learning method used for multivariate analysis. Principal Component Analysis (PCA) is a statistical method that reduces the multivariate dataset in to a set of multiple orthogonal components that explains the maximum amount of variance in the data. PCA reduces dimensions

of data while retaining most of the original information. PCA is mathematical tool that reduces high number of correlated features in to less uncorrelated orthogonal principal components. In other words, PCA is a linear dimension reduction tool that is very useful for data with high correlated variables.

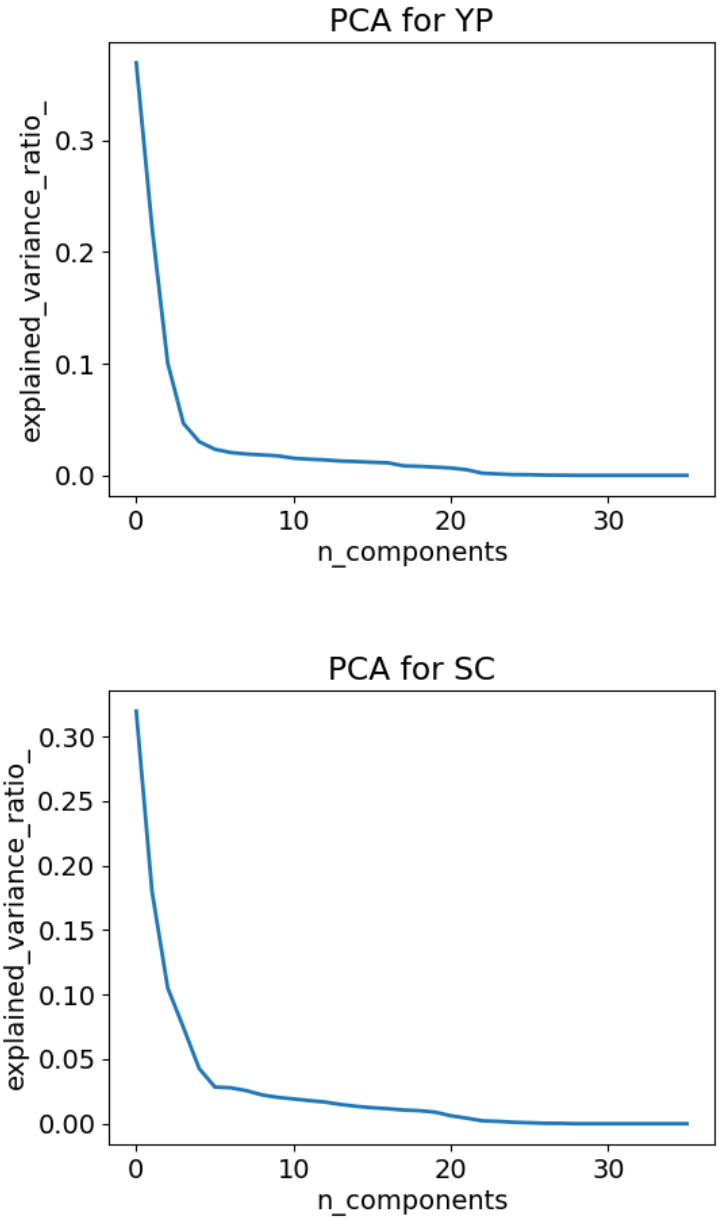


Figure 4.10 PCA feature reduction on 3-AAU *Yersinia Pestis* and *Saccharomyces cerevisiae* dataset

PCA was first invented by Karl Pearson in 1901 [6] and later developed by Harold Hotelling in the 1936 [7]. PCA is one of the simplest eigenvector based multivariate analysis. PCA is mostly used in explanatory data analysis. We have implemented PCA using pythons scikit-learn [10] library. Scikit-learn [10] uses the LAPACK implementation of the full Singular Value Decomposition (SVD) or a randomized truncated SVD by the method of Halko et al. 2009, depending on the shape of the input data and the number of components to extract. It also has option of the scipy.sparse.linalg ARPACK implementation of the truncated SVD [10].

We did Principal Component Analysis (PCA) for *Yersinia Pestis* and *Saccharomyces cerevisiae* datasets. The Figure 4.10 show the explained variance from PCA analysis we have done on two datasets. In the figure 4.10, n_components is the number principal components and explained_variance_ratio_ is the variance explained by each principal component. In figure 4.10, the graphs flattens out from principal components 7. These 7 components covers around 82% variance for *Yersinia Pestis* dataset and 78% variance for *Saccharomyces cerevisiae* dataset.

Chapter 5

Machine Learning techniques for peptide classification

We did experiments using four different machine learning algorithms to verify if Support Vector Machine (SVM) is giving us the best peptide classification. We have incorporated *Saccharomyces cerevisiae* dataset to train and test our models in addition to *Yersinia Pestis* dataset. The following are the machine learning algorithms which we have performed for this research:

1. Logistic Regression
2. Random Forest
3. K-Nearest Neighbor
4. XGBoost

We have also performed three different feature selection techniques to come up with optimal features as described in Chapter 4. The three feature selection methods we performed are Univariate Analysis, Recursive Feature Elimination (RFE) and XGBoost (via it's feature importance method). We have also performed feature reduction algorithm, Principal Component Analysis (PCA). We are reporting the results from each of the classification method using different feature selection methods. We started to run our experiments on normalized datasets. We performed the normalization using min-max scalar.

We have performed our tests on *Yersinia Pestis* and *Saccharomyces cerevisiae* (*Yeast*) normalized datasets. We started off by dividing each of the datasets in to train and test data by 4:1 ratio. We have also tested each of our trained models with both the data sets. We have used grid search with cross-validation to come up with optimal parameters for each of the classifiers. For all these analysis we have used scikit-learn [10] package for Python.

We will go in to each of the classification methods in more detail below:-

5.1 Logistic Regression

Logistic Regression is a supervised learning technique. Logistic regression is a classifier that classifies an observation in one of the two or more classes. Logistic regression can be binomial, multinomial or ordinal. In our case, we are using binomial logistic regression. The logistic function is at the core of logistic regression. The logistic function is a *'S'* shaped sigmoid curve. The equation of logistic function is as, with x as a real value number between $-\infty$ to $+\infty$.

$$f(x) = \frac{e^x}{(1 + e^x)} \tag{5.1}$$

A logistic regression is a model which provides log-odds of the probability of an event in a linear combination of independent or predictor variables. Binary logistic regression is an expression of probability of an event Y = 1, 0 occurring against a set of X = (X_1, X_2, ..., X_k) explanatory variables which can be discrete, continuous, or a combination.

$$logit(Pr(Y_i = 1|X_i = x_i)) = logit(\pi_i) = \beta_0 + \beta_1 x_{i1} + ... + \beta_k x_{ik} \tag{5.2}$$

We started off with the *Yersinia Pestis* datasets. We divided the *Yersinia Pestis* dataset in to train and test data by 80:20 ratio. We then took balanced data from 80% dataset. We did the feature selection by three different ways: Univariate Analysis,

Table 5.1 Logistic Regression trained on sample balanced 3-AAU datasets and tested on 20% unbalanced datasets

Feature Selection	Number of Features	Classifier	Training data	Testing data	Sensitivity	Specificity
XGBoost	6	Logistic Regression	Sample YP	20% YP	94%	80%
RFE	6	Logistic Regression	Sample YP	20% YP	94%	80%
Univariate	6	Logistic Regression	Sample YP	20% YP	94%	80%
XGBoost	6	Logistic Regression	Sample SC	20% SC	96%	94%
RFE	6	Logistic Regression	Sample SC	20% SC	97%	94%
Univariate	6	Logistic Regression	Sample SC	20% SC	97%	94%

Recursive Feature Elimination (RFE) and XGBoost feature importance. For feature selection we used the whole dataset for *Yersinia Pestis*. We trained the Logistic Regression using balanced training data. We tested our model on unbalanced 20% of the *Yersinia Pestis* as well as on full *Yersinia Pestis* dataset. We did the grid search on the trained model to optimize the results. We did grid search with cross-validation on the three parameters: penalty, C and solver. We have also used 10-fold cross-validation while training our models. We repeated similar steps from feature selection to model training on *Saccharomyces cerevisiae* dataset. The Table 5.1 shows Logistic Regression on sample balanced 3-AAU datasets and tested on 20% of the unbalanced datasets. The Figure 5.1 shows ROC-AUC curve for the Logistic Regression on sample balanced 3-AAU datasets and tested on 20% of the unbalanced datasets.

In the next step, we trained the Logistic Regression model on sample balanced *Yersinia Pestis* dataset but tested it on full *Yersinia Pestis*s dataset with 10-fold cross-validation. We repeated the same steps for *Saccharomyces cerevisiae*. For the model trained on *Yersinia Pestis*, feature selection is done using *Yersinia Pestis* dataset. For the model trained on *Saccharomyces cerevisiae*, feature selection is done using *Saccharomyces cerevisiae* dataset. The Table 5.2 shows Logistic Regression on 3-AAU datasets which is trained on sample balanced 3-AAU datasets but tested on full unbalanced datasets. The Figure 5.2 shows ROC-AUC curve for the Logis-

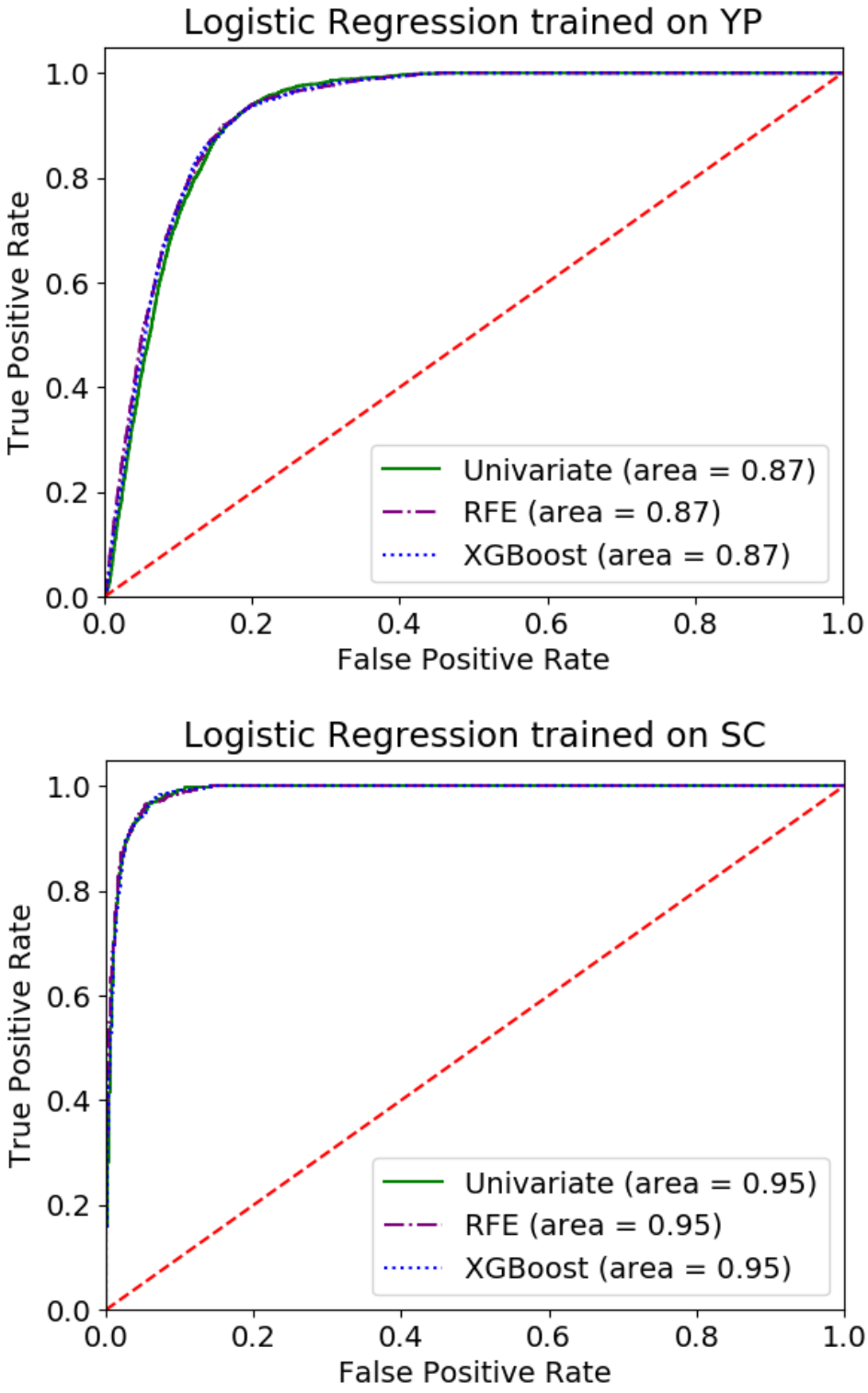


Figure 5.1 ROC curve for Logistic Regression trained on sample balanced 3-AAU datasets and tested on 20% unbalanced datasets using three different feature selection methods

Table 5.2 Logistic Regression trained on sample balanced 3-AAU datasets and tested on full unbalanced datasets

Feature Selection	Number of Features	Classifier	Training data	Testing data	Sensitivity	Specificity
XGBoost	6	Logistic Regression	Sample YP	Full YP	93%	79%
RFE	6	Logistic Regression	Sample YP	Full YP	94%	79%
Univariate	6	Logistic Regression	Sample YP	Full YP	94%	80%
XGBoost	6	Logistic Regression	Sample SC	Full SC	97%	93%
RFE	6	Logistic Regression	Sample SC	Full SC	97%	93%
Univariate	6	Logistic Regression	Sample SC	Full SC	97%	92%

tic Regression on sample balanced 3-AAU datasets and tested on full unbalanced datasets.

After getting good results from logistic regression models trained on sample balanced data, we trained the model on unbalanced sample of *Yersinia Pestis* and *Saccharomyces cerevisiae* datasets. For this we trained and tested our models on 80:20 ratio. We used the same hyper parameters as used earlier for training sample balanced data. For unbalanced datasets, we have used class weights. Like previously we also tested our models on full datasets using 10-fold cross-validation. The Tables 5.3 and 5.4 shows the Logistic Regression results trained on 80% unbalanced datasets and tested on 20% data and full datasets respectively. The Figures 5.3 and 5.4 shows the ROC-AUC curve for the Logistic Regression results trained on 80% unbalanced datasets and tested on 20% data and full datasets respectively.

We then incorporated both the datasets from *Yersinia Pestis* and *Saccharomyces cerevisiae* to test our logistic regression models. We found that our model trained on only 6 features of *Yersinia Pestis* dataset is not able to classify *Saccharomyces cerevisiae* dataset that well for 3-AAU datasets. Similarly, the model trained on *Saccharomyces cerevisiae* was not able to classify the *Yersinia Pestis* dataset that well. The AUC score when we were training and testing using datasets from two different species was approximately 50% for both the models. We again performed feature se-

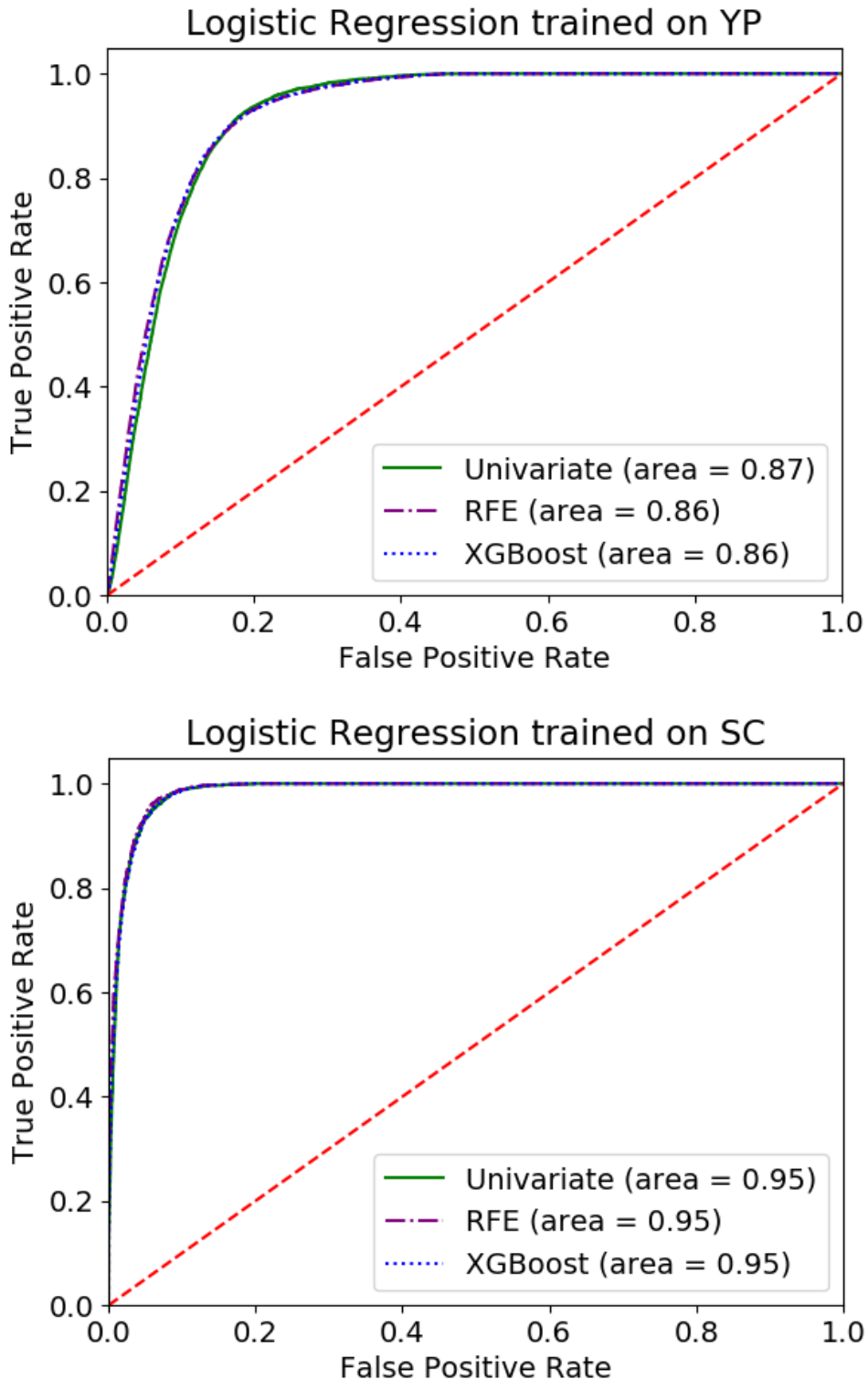


Figure 5.2 ROC curve for Logistic Regression trained on sample balanced 3-AAU datasets and tested on full unbalanced datasets using three different feature selection methods

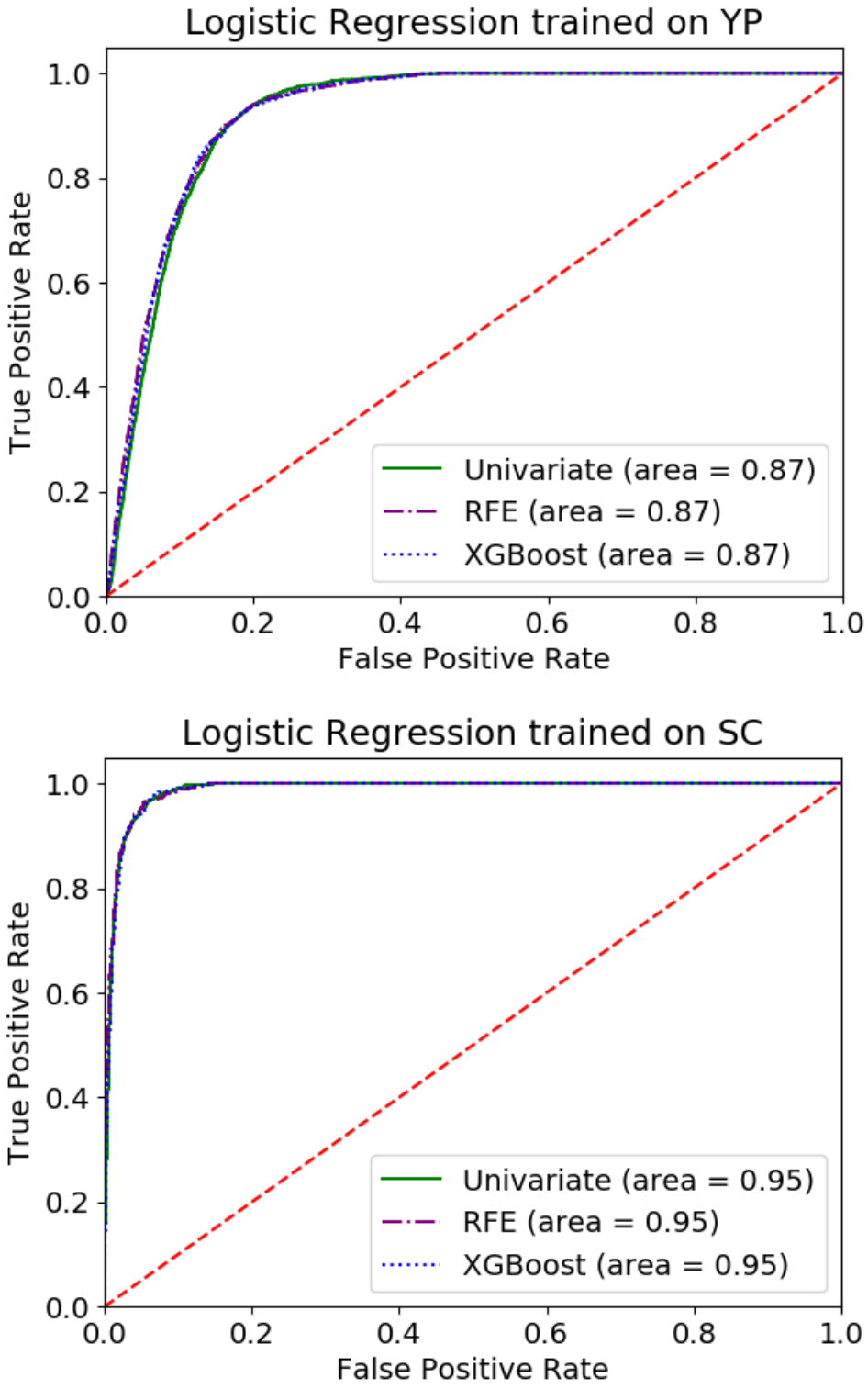


Figure 5.3 ROC curve for Logistic Regression trained on sample unbalanced balanced 3-AAU datasets and tested on 20% unbalanced datasets using three different feature selection methods

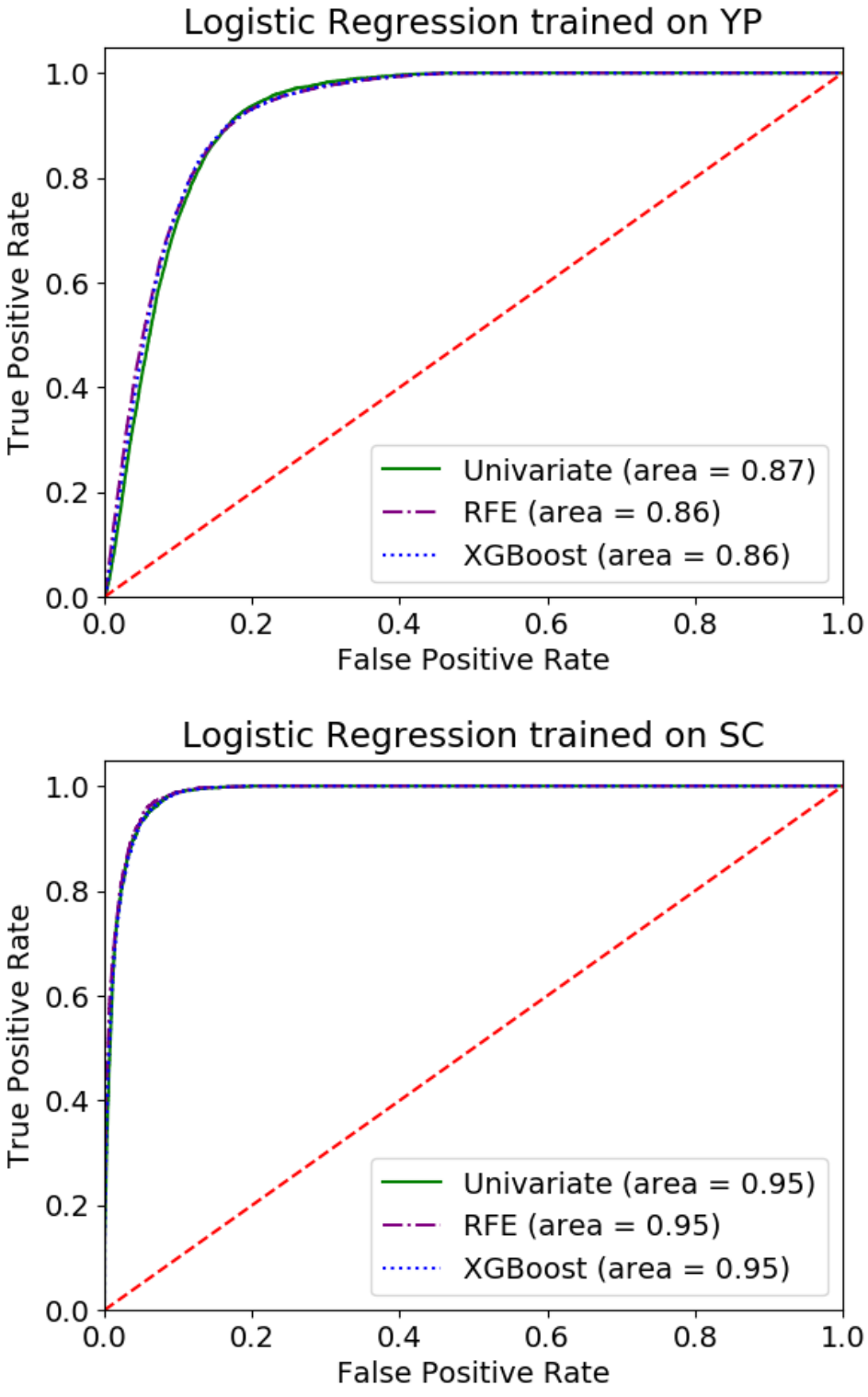


Figure 5.4 ROC curve for Logistic Regression trained on sample unbalanced balanced 3-AAU datasets and tested on full unbalanced datasets using three different feature selection methods

Table 5.3 Logistic Regression trained on sample unbalanced 3-AAU datasets and tested on 20% unbalanced datasets

Feature Selection	Number of Features	Classifier	Training data	Testing data	Sensitivity	Specificity
XGBoost	6	Logistic Regression	80% YP	20% YP	94%	80%
RFE	6	Logistic Regression	80% YP	20% YP	94%	80%
Univariate	6	Logistic Regression	80% YP	20% YP	94%	80%
XGBoost	6	Logistic Regression	80% SC	20% SC	96%	94%
RFE	6	Logistic Regression	80% SC	20% SC	97%	94%
Univariate	6	Logistic Regression	80% SC	20% SC	97%	94%

Table 5.4 Logistic Regression trained on sample unbalanced 3-AAU datasets and tested on full unbalanced datasets

Feature Selection	Number of Features	Classifier	Training data	Testing data	Sensitivity	Specificity
XGBoost	6	Logistic Regression	80% YP	Full YP	93%	79%
RFE	6	Logistic Regression	80% YP	Full YP	94%	79%
Univariate	6	Logistic Regression	80% YP	Full YP	94%	80%
XGBoost	6	Logistic Regression	80% SC	Full SC	97%	93%
RFE	6	Logistic Regression	80% SC	Full SC	97%	93%
Univariate	6	Logistic Regression	80% SC	Full SC	97%	92%

lection using XGBoost feature importance, Univariate and RFE on combined features for both the *Yersinia Pestis* and *Saccharomyces cerevisiae* datasets. We did this to make sure that both the model are trained on the same feature sets. The results for 3-AAU datasets were not good, the AUC scores were pretty low.

We created the 2-AAU datasets for *Yersinia Pestis* and *Saccharomyces cerevisiae* for further testing. We again performed feature selection using XGBoost feature importance, Univariate and RFE on combined features for *Yersinia Pestis* and *Saccharomyces cerevisiae* 2-AAU normalized datasets. We again performed grid search with 5-fold cross-validation on C and penalty. We have used class weights. We got decent results only with the features set from RFE analysis. For the model trained with the 2-

Table 5.5 Logistic Regression trained on sample unbalanced 2-AAU datasets tested on full unbalanced datasets from different species. The feature selection is done using features from both the *Yersinia Pestis* and *Saccharomyces cerevisiae* datasets.

Feature Selection	Number of Features	Classifier	Training data	Testing data	Sensitivity	Specificity
RFE	13	Logistic Regression	Sample YP	Full SC	92%	50%
RFE	13	Logistic Regression	Sample YP	Full YP	72%	74%
RFE	13	Logistic Regression	Sample SC	Full SC	85%	77%
RFE	13	Logistic Regression	Sample SC	Full YP	69%	75%

Table 5.6 Logistic Regression trained on sample unbalanced 3-AAU datasets and tested on full unbalanced datasets from different species

Feature Selection	Principal Components	Classifier	Training data	Testing data	Sensitivity	Specificity
PCA	8	Logistic Regression	Sample YP	Full SC	81%	78%
PCA	8	Logistic Regression	Sample YP	Full YP	85%	75%
PCA	8	Logistic Regression	Sample SC	Full SC	91%	80%
PCA	8	Logistic Regression	Sample SC	Full YP	83%	68%

AAU *Yersinia Pestis* dataset, we got decent results with C=0.00012 and penalty='l2'. For the model trained with the 2-AAU *Saccharomyces cerevisiae* dataset, we got decent results with C=0.001 and penalty='l2'. We have used the 5-fold cross-validation to validate the models when testing with the same dataset. We have also tested the models on the dataset from different species. The Table 5.5 shows logistic regression classification on 2-AAU datasets using RFE analysis, tested on full datasets of both the *Yersinia Pestis* and *Saccharomyces cerevisiae*. The Figure 5.5 shows the ROC-AUC scores for logistic regression model trained on 2-AAU datasets of *Yersinia Pestis* and *Saccharomyces cerevisiae*.

We also performed principal component analysis (PCA) for feature reduction on both *Yersinia Pestis* and *Saccharomyces cerevisiae* datasets. In PCA analysis we used all 3-AAU dataset features to get 8 principal components that gave us good results. These 8 principal components covered 84% variance for *Yersinia Pestis* dataset and

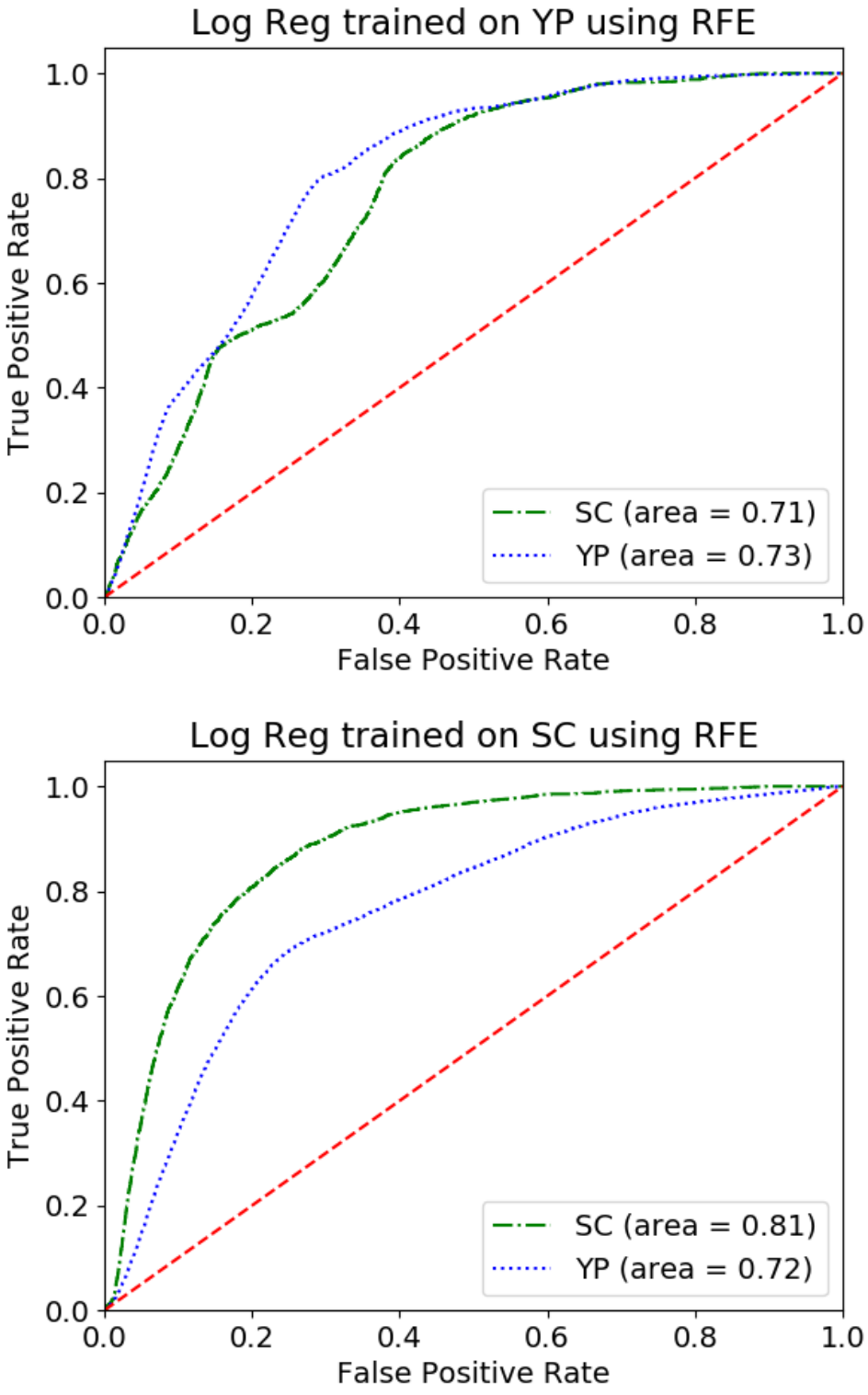


Figure 5.5 ROC curve for Logistic Regression using the 13 feature from RFE analysis. Model trained on unbalanced sample 2-AAU datasets and tested on full unbalanced datasets of both the species

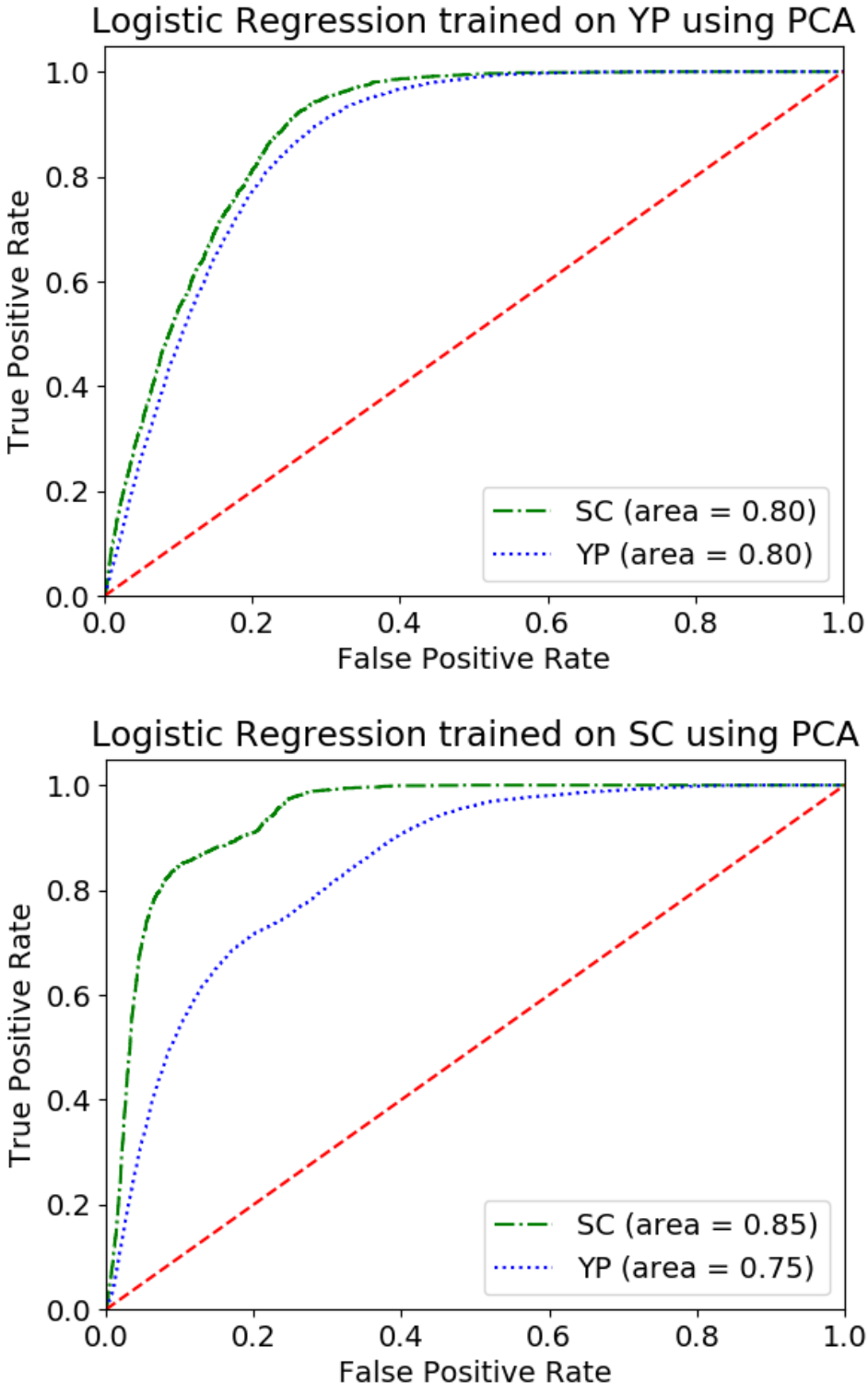


Figure 5.6 ROC curve for Logistic Regression using the 8 principal components. Model trained on unbalanced sample 3-AAU datasets and tested on full unbalanced datasets of both the species

80% variance for *Saccharomyces cerevisiae* dataset. We have used these 8 principal components for the logistic regression model to train on *Saccharomyces cerevisiae* and tested on *Yersinia Pestis*. We have performed grid search with 5-fold cross-validation on the two parameters: penalty and C. We have used class weights in our models. We have used the 5-fold cross-validation to validate the model trained and tested on the same dataset. We have tested the models with the datasets from different species. The Table 5.6 shows Logistic Regression using PCA on 3-AAU datasets which is trained on 80% sample datasets but tested on full datasets for both the species. The Figure 5.6 shows ROC-AUC curve for the Logistic Regression using PCA on 3-AAU datasets which is trained on 80% sample datasets and tested on full datasets for both the species.

In summary, for the logistic regression classification, we got best results with the models trained and tested on the datasets from the same species. We got best AUC score of 87% for the model trained and tested with *Yersinia Pestis* datasets using features set from Univariate analysis. We got AUC score of 95% for the model trained and tested with *Saccharomyces cerevisiae* datasets using features set from all three feature selection methods we performed. However, these models gave AUC of only 50% when tested with datasets from different species. Hence, we have created 2-AAU normalized datasets to get better results for the models trained and tested on datasets from different species. For the 2-AAU datasets, using RFE features set, we got decent results. For the model trained on 2-AAU *Yersinia Pestis* dataset, we got AUC score of 73% and 71% when testing with *Yersinia Pestis* and *Saccharomyces cerevisiae* datasets respectively. For the model trained on 2-AAU *Saccharomyces cerevisiae* dataset, we got AUC score of 72% and 81% when testing with *Yersinia Pestis* and *Saccharomyces cerevisiae* datasets respectively. With PCA, using 8 principal components for 3-AAU datasets, the model trained with the *Yersinia Pestis* dataset gave AUC score of 80% when tested with both the *Yersinia Pestis* and *Saccharomyces*

cerevisiae datasets. With PCA, using 8 principal components for 3-AAU datasets, the model trained with the *Saccharomyces cerevisiae* dataset gave AUC score of 85% and 75% when tested with the *Saccharomyces cerevisiae* and *Yersinia Pestis* datasets respectively.

5.2 Random Forest

Random Forests is a supervised learning technique used for classification and regression. Random Forests is an ensemble of many number of decision trees created from randomly selected subset of training data. While splitting the node during construction of trees, the split that is picked is the best split among a random subset of the features. Random Forests outputs the class that is the mode of classes (classification) or mean prediction (regression) of the individual trees. Random Forests follow divide and conquer approach to increase performance.

We started off with the *Yersinia Pestis* datasets. We divided the *Yersinia Pestis* dataset in to train and test data by 80:20 ratio. We then took balanced sample data from 80% dataset. We did the feature selection by three different ways: Univariate Analysis, Recursive Feature Elimination (RFE) and XGBoost feature importance. For feature selection we have used whole dataset for *Yersinia Pestis*. We trained the Random Forest using sample balanced training data. We tested our model on unbalanced 20% of the *Yersinia Pestis* dataset as well as on full *Yersinia Pestis* dataset. We have used grid search with cross-validation to get optimal hyper parameters values for Random Forest. We have used 5-fold cross-validation while training our models. We repeated similar steps for feature selection and model training on *Saccharomyces cerevisiae* dataset. The Table 5.7 shows Random Forest classification on sample balanced 3-AAU datasets and tested on 20% unbalanced datasets.

The Figure 5.7 shows the ROC-AUC curve for the Random Forest classification using three different feature selection methods. The models are trained on sample

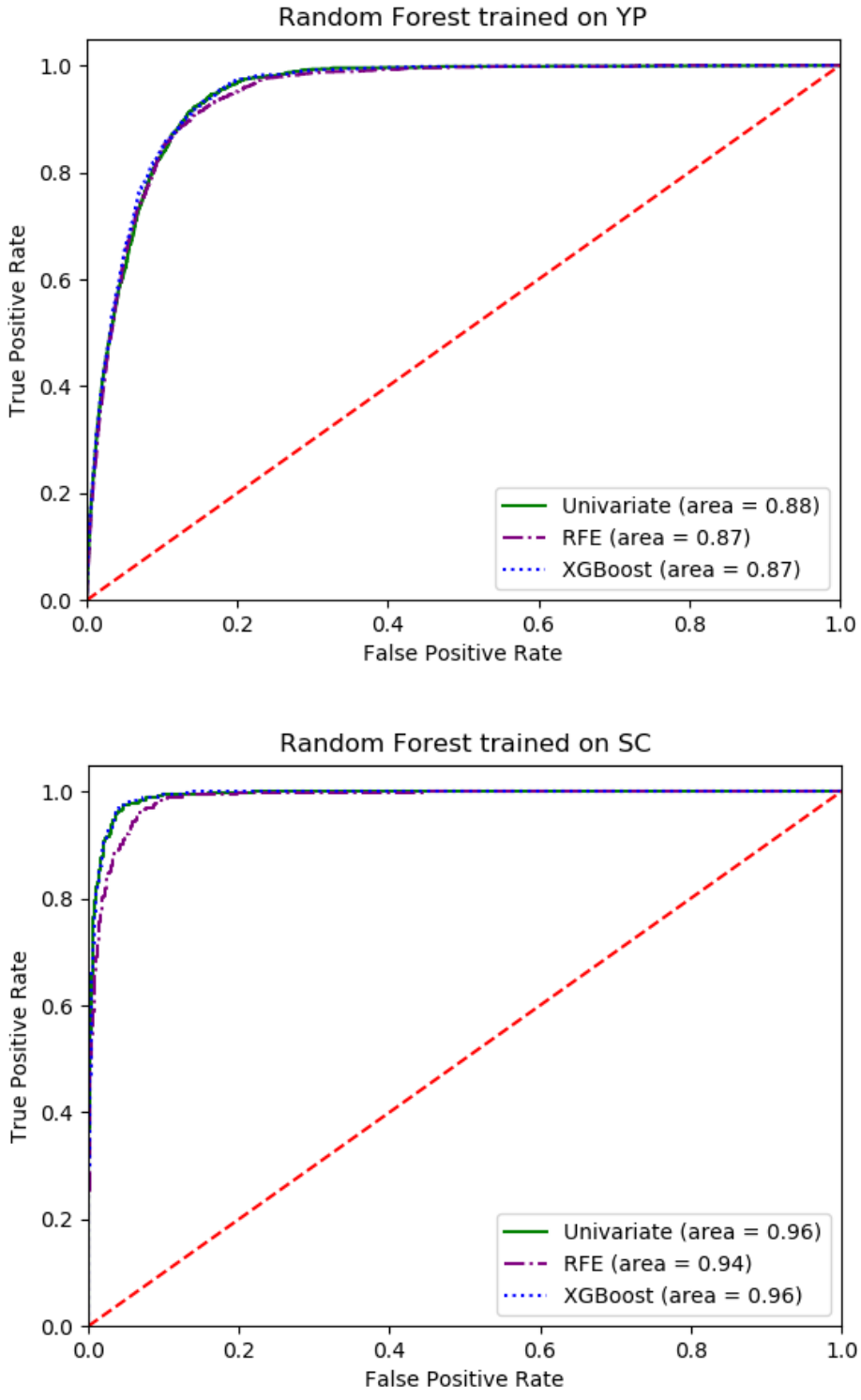


Figure 5.7 ROC curve for Random Forest trained on sample balanced 3-AAU datasets and tested on 20% unbalanced datasets using three different feature selection methods

Table 5.7 Random Forest trained on sample balanced 3-AAU datasets and tested on 20% unbalanced datasets

Feature Selection	Number of Features	Classifier	Training data	Testing data	Sensitivity	Specificity
XGBoost	6	Random Forest	Sample YP	20% YP	83%	91%
RFE	6	Random Forest	Sample YP	20% YP	83%	90%
Univariate	6	Random Forest	Sample YP	20% YP	87%	88%
XGBoost	6	Random Forest	Sample SC	20% SC	96%	96%
RFE	6	Random Forest	Sample SC	20% SC	93%	95%
Univariate	6	Random Forest	Sample SC	20% SC	96%	96%

balanced 3-AAU *Yersinia Pestis* and *Saccharomyces cerevisiae* datasets and tested on 20% unbalanced *Yersinia Pestis* and *Saccharomyces cerevisiae* datasets. For the model trained on *Yersinia Pestis*, feature selection is done using *Yersinia Pestis* dataset. For the model trained on *Saccharomyces cerevisiae*, feature selection is done using *Saccharomyces cerevisiae* dataset.

In the next step, we trained the Random Forest model on sample balanced *Yersinia Pestis* dataset and tested on full *Yersinia Pestis*s dataset. We repeated the same steps for *Saccharomyces cerevisiae*. The Table 5.8 shows Random Forest classification on 3-AAU datasets which is trained on sample balanced datasets but tested on full datasets. The Figure 5.8 shows the AUC scores for the Random Forest classifier using three different feature selection methods. The models are trained on sample balanced 3-AAU *Yersinia Pestis* and *Saccharomyces cerevisiae* datasets and tested on full unbalanced *Yersinia Pestis* and *Saccharomyces cerevisiae* datasets.

After getting good results from models trained on sample balanced data, we trained the models on unbalanced sample of *Yersinia Pestis* and *Saccharomyces cerevisiae*. For this we divided the datasets in 80:20 ratio for training and testing respectively. We used the same hyper parameters as used earlier for training sample balanced data. Like previously, we have also tested the models with full datasets.

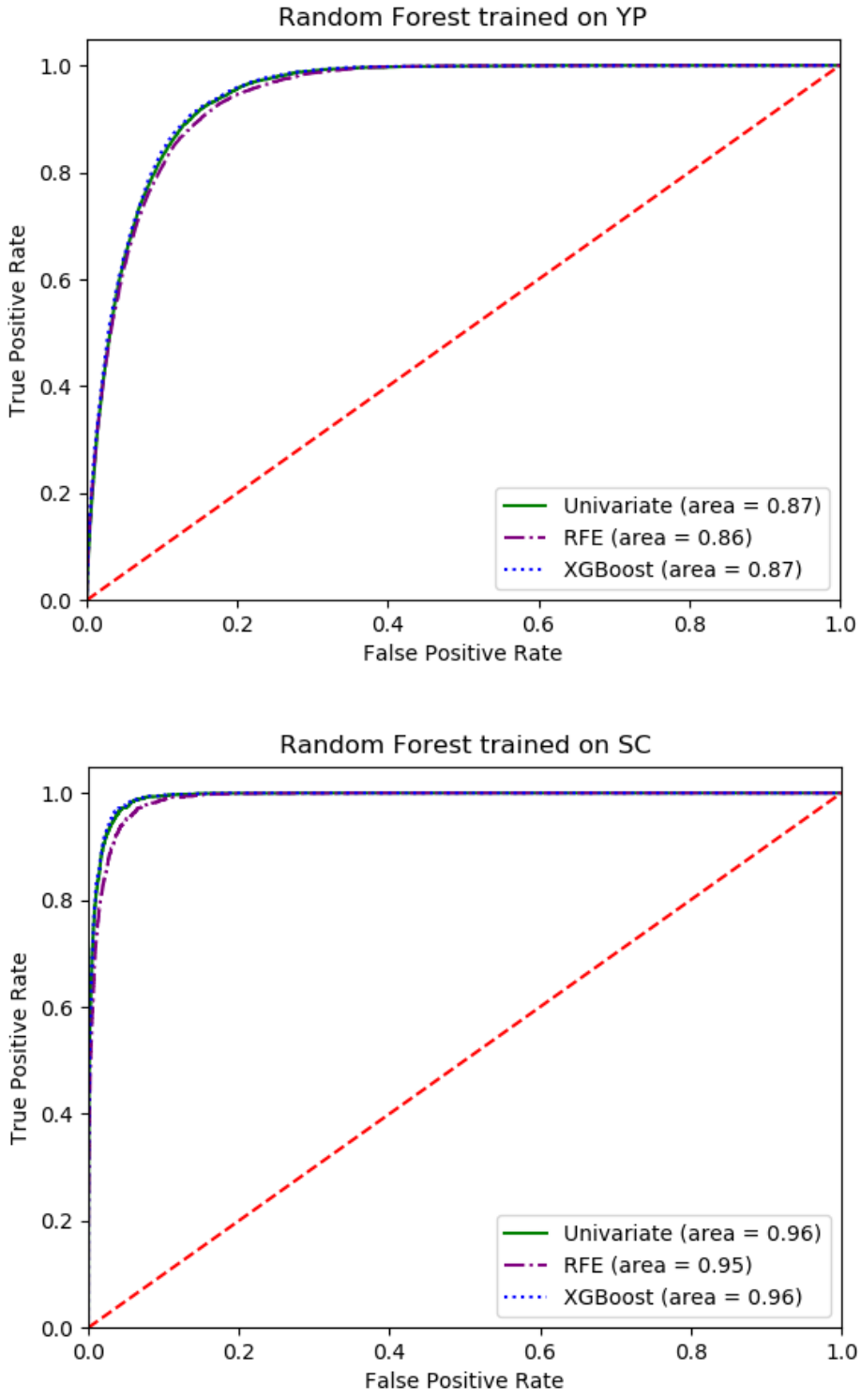


Figure 5.8 ROC curve for Random Forest trained on sample balanced 3-AAU datasets and tested on full unbalanced datasets using three different feature selection methods

Table 5.8 Random Forest trained on sample balanced 3-AAU datasets and tested on full unbalanced datasets

Feature Selection	Number of Features	Classifier	Training data	Testing data	Sensitivity	Specificity
XGBoost	6	Random Forest	Sample YP	Full YP	83%	90%
RFE	6	Random Forest	Sample YP	Full YP	82%	90%
Univariate	6	Random Forest	Sample YP	Full YP	86%	88%
XGBoost	6	Random Forest	Sample SC	Full SC	97%	96%
RFE	6	Random Forest	Sample SC	Full SC	95%	95%
Univariate	6	Random Forest	Sample SC	Full SC	97%	96%

Table 5.9 Random Forest trained on sample unbalanced 3-AAU datasets and tested on 20% unbalanced datasets

Feature Selection	Number of Features	Classifier	Training data	Testing data	Sensitivity	Specificity
XGBoost	6	Random Forest	80% YP	20% YP	83%	91%
RFE	6	Random Forest	80% YP	20% YP	83%	90%
Univariate	6	Random Forest	80% YP	20% YP	87%	89%
XGBoost	6	Random Forest	80% SC	20% SC	96%	96%
RFE	6	Random Forest	80% SC	20% SC	93%	95%
Univariate	6	Random Forest	80% SC	20% SC	96%	96%

The Tables 5.9 and 5.10 shows the Random Forest results for the model trained on unbalanced sample and tested on 20% sample datasets and full datasets respectively.

Figure 5.9 shows the AUC scores for the Random Forest using three different feature selection methods. The models are trained on 80% unbalanced 3-AAU *Yersinia Pestis* and *Saccharomyces cerevisiae* datasets and tested on 20% unbalanced *Yersinia Pestis* and *Saccharomyces cerevisiae* datasets. For the model trained on *Yersinia Pestis*, feature selection is done using *Yersinia Pestis* dataset. For the model trained on *Saccharomyces cerevisiae*, feature selection is done using *Saccharomyces cerevisiae* dataset.

Figure 5.10 shows the ROC-AUC curve for the Random Forest using three differ-

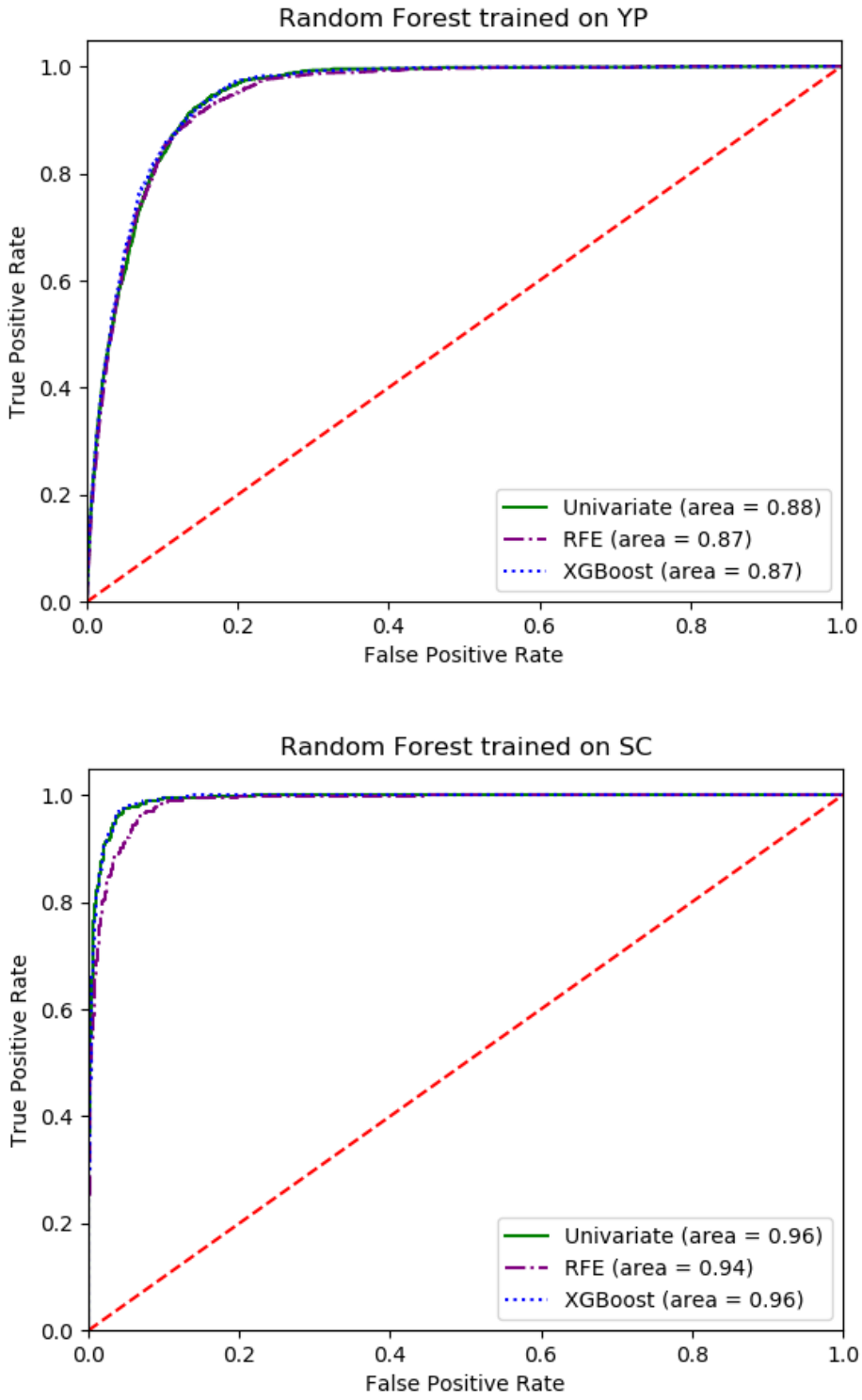


Figure 5.9 ROC curve for Random Forest trained on 80% unbalanced 3-AAU datasets and tested on 20% unbalanced datasets using three different feature selection methods

Table 5.10 Random Forest trained on sample unbalanced 3-AAU datasets and tested on full unbalanced datasets

Feature Selection	Number of Features	Classifier	Training data	Testing data	Sensitivity	Specificity
XGBoost	6	Random Forest	80% YP	Full YP	83%	90%
RFE	6	Random Forest	80% YP	Full YP	82%	90%
Univariate	6	Random Forest	80% YP	Full YP	86%	88%
XGBoost	6	Random Forest	80% SC	Full SC	97%	96%
RFE	6	Random Forest	80% SC	Full SC	95%	95%
Univariate	6	Random Forest	80% SC	Full SC	97%	96%

Table 5.11 Random Forest trained on sample unbalanced 3-AAU datasets and tested on full unbalanced datasets from different species

Feature Selection	Number of Features	Classifier	Training data	Testing data	Sensitivity	Specificity
PCA	7	Random Forest	Sample YP	Full SC	87%	81%
PCA	7	Random Forest	Sample YP	Full YP	76%	85%
PCA	7	Random Forest	Sample SC	Full SC	92%	91%
PCA	7	Random Forest	Sample SC	Full YP	77%	76%

ent feature selection methods. The models are trained on 80% unbalanced 3-AAU *Yersinia Pestis* and *Saccharomyces cerevisiae* datasets and tested on full unbalanced *Yersinia Pestis* and *Saccharomyces cerevisiae* datasets.

We then incorporated both the datasets from *Yersinia Pestis* and *Saccharomyces cerevisiae* to test our models. We found that our model trained on only 6 features of *Yersinia Pestis* is not classifying *Saccharomyces cerevisiae* dataset that well for 3-AAU datasets. We again did feature selection using XGBoost feature importance, Univariate and RFE on combined calculated features for *Yersinia Pestis* and *Saccharomyces cerevisiae*. But still the classification was way off, we were getting only 50% AUC.

Like in case of Logistic Regression, we again worked with 2-AAU datasets to

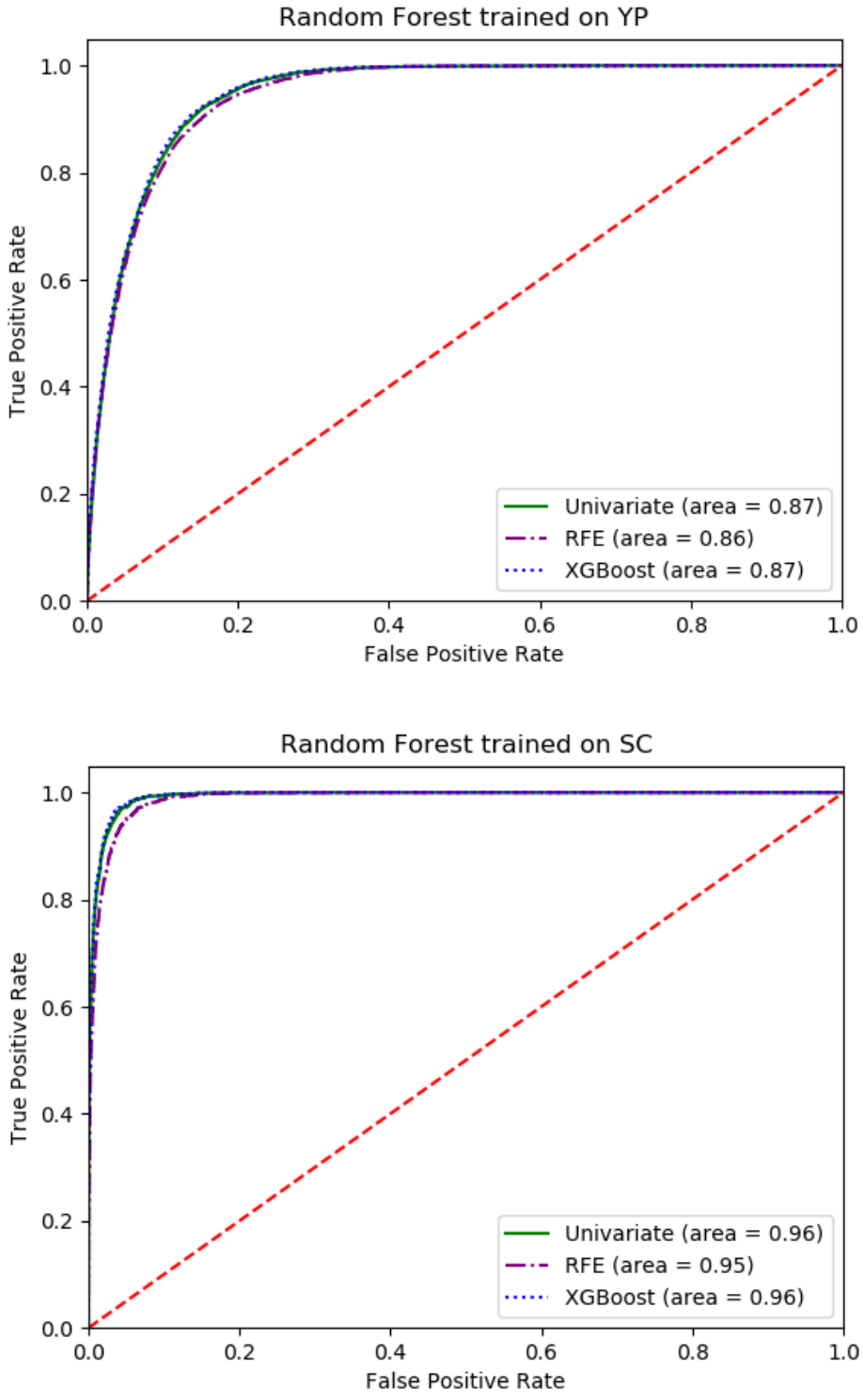


Figure 5.10 ROC curve for Random Forest trained on 80% unbalanced 3-AAU datasets and tested on full unbalanced datasets using three different feature selection methods

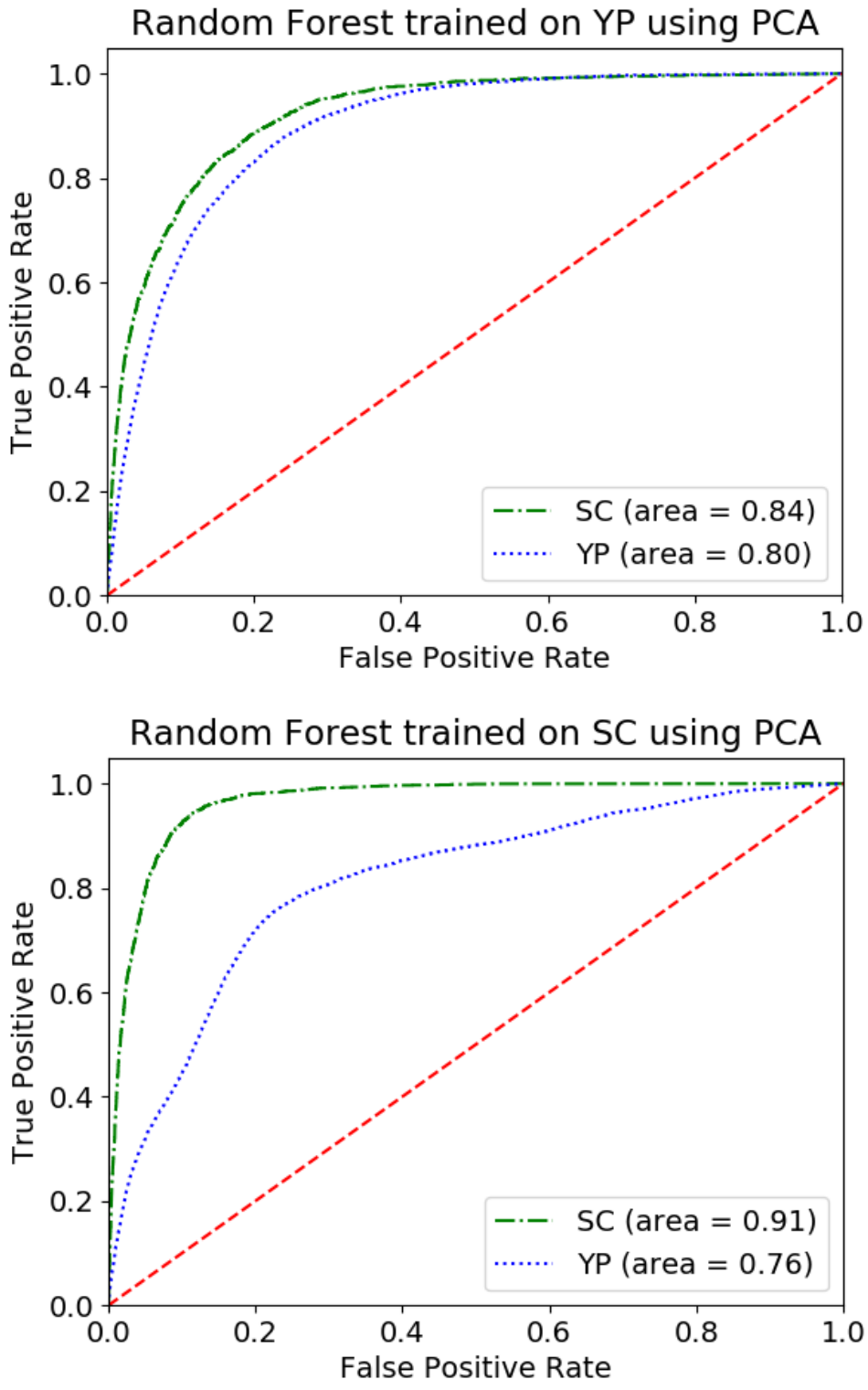


Figure 5.11 ROC curve for Random Forest using 7 Principal Component Analysis (PCA). Model trained on unbalanced sample 3-AAU datasets and tested on full unbalanced datasets of both the species

get decent results for Random Forest classification but we were getting pretty low AUC scores. We then performed principal component analysis (PCA) for feature reduction on both *Yersinia Pestis* and *Saccharomyces cerevisiae* datasets. In PCA analysis we used all 3-AAU dataset features to get 7 principal components that gave us decent results. These 7 principal components covered 82% variance for *Yersinia Pestis* dataset and 78% variance for *Saccharomyces cerevisiae* dataset. We have also used 5-fold cross-validation while training our models. The Table 5.11 shows Random Forest classification using 7 principal components on 3-AAU datasets, trained on 80% sample datasets but tested on full datasets for both the species. The Figure 5.11 shows ROC-AUC curve for the Random Forest classification using 7 principal components on 3-AAU datasets, trained on 80% sample datasets but tested on full datasets for both the species.

In summary, for the random forest classification, we got best results with the models trained and tested on the datasets from the same species. We got best AUC score of 88% for the model trained on 80% of the *Yersinia Pestis* dataset and tested with 20% of *Yersinia Pestis* dataset using features set from Univariate analysis. We got best AUC score of 96% for the model trained and tested with the *Saccharomyces cerevisiae* datasets using features set from Univariate analysis and XGBoost feature importance. However, these models gave AUC of only 50% when tested with datasets from different species. Hence, we have created 2-AAU normalized (using min-max scalar) datasets to get better results for the models trained and tested on datasets from different species. But for the 2-AAU datasets as well we didn't get any decent results. We were still getting very low AUC scores with 2-AAU datasets. With PCA, using 7 principal components for 3-AAU datasets, the model trained with the *Yersinia Pestis* dataset gave AUC score of 80% and 84% when tested with both the *Yersinia Pestis* and *Saccharomyces cerevisiae* datasets respectively. With PCA, using 7 principal components for 3-AAU datasets, the model trained with the *Sac-*

charomyces cerevisiae dataset gave AUC score of 91% and 76% when tested with the *Saccharomyces cerevisiae* and *Yersinia Pestis* datasets respectively.

5.3 K-Nearest Neighbor

The k-nearest neighbors (KNN) is a non-parametric machine learning algorithm used for classification and regression. Non-parametric techniques do make any assumption on the underlying data distribution which in sense resembles very closely to the real world problems. K-nearest neighbor doesn't have any explicit training phase hence training phase is quite fast. In KNN, a positive number k is specified which is generally small. Along with k, a new sample is provided. A database of with k entries is created which closely resemble our new sample. In this database we gave the most common classification of these entries to the new sample.

In KNN, selecting k is most important, as we increase k, the classification boundaries become smoother. For example, in binary classification, if we increase k to infinity, all the entries will either in one class on in the other depending upon the total majority.

We started KNN with the *Yersinia Pestis* datasets. We divided the *Yersinia Pestis* data in to train and test data by 80:20 ratio. We then took sample balanced data from 80% dataset. We did the feature selection by three different ways: Univariate Analysis, Recursive Feature Elimination (RFE) and XGBoost feature importance. For feature selection we used whole data for *Yersinia Pestis*. We trained the k-nearest neighbor model using balanced training data. We tested our model on unbalanced 20% of the *Yersinia Pestis* as well as on full *Yersinia Pestis* dataset with 4-fold cross-validation. We repeated similar steps from feature selection to model the trained with *Saccharomyces cerevisiae* dataset. The Table 5.12 shows k-nearest neighbor on sample balanced 3-AAU datasets and tested on 20% unbalanced datasets. The Figure 5.12 shows ROC-AUC curve for the k-nearest neighbor on sample balanced 3-AAU

Table 5.12 k-nearest neighbor trained on sample balanced 3-AAU datasets and tested on 20% unbalanced datasets

Feature Selection	Number of Features	Classifier	Training data	Testing data	Sensitivity	Specificity
XGBoost	6	KNN	Sample YP	20% YP	93%	70%
RFE	6	KNN	Sample YP	20% YP	94%	69%
Univariate	6	KNN	Sample YP	20% YP	90%	79%
XGBoost	6	KNN	Sample SC	20% SC	95%	89%
RFE	6	KNN	Sample SC	20% SC	96%	85%
Univariate	6	KNN	Sample SC	20% SC	96%	89%

Table 5.13 k-nearest neighbor trained on sample balanced 3-AAU datasets and tested on full unbalanced datasets

Feature Selection	Number of Features	Classifier	Training data	Testing data	Sensitivity	Specificity
XGBoost	6	KNN	Sample YP	Full YP	99%	7%
RFE	6	KNN	Sample YP	Full YP	99%	3%
Univariate	6	KNN	Sample YP	Full YP	98%	27%
XGBoost	6	KNN	Sample SC	Full SC	97%	71%
RFE	6	KNN	Sample SC	Full SC	98%	68%
Univariate	6	KNN	Sample SC	Full SC	98%	76%

datasets and tested on 20% unbalanced datasets.

In the next step, we trained the model on sample balanced *Yersinia Pestis* dataset and tested on full *Yersinia Pestis*s dataset using 4-fold cross-validation. We repeated the same steps for *Saccharomyces cerevisiae*. The Table 5.13 shows k-nearest neighbor on 3-AAU datasets which is trained on sample datasets but tested on full datasets. We got pretty low specificity scores while testing with full datasets using 5-fold cross-validation.

We also trained the KNN model on unbalanced sample of *Yersinia Pestis* and *Saccharomyces cerevisiae*. For this we divided the datasets in 80:20 ratio for training

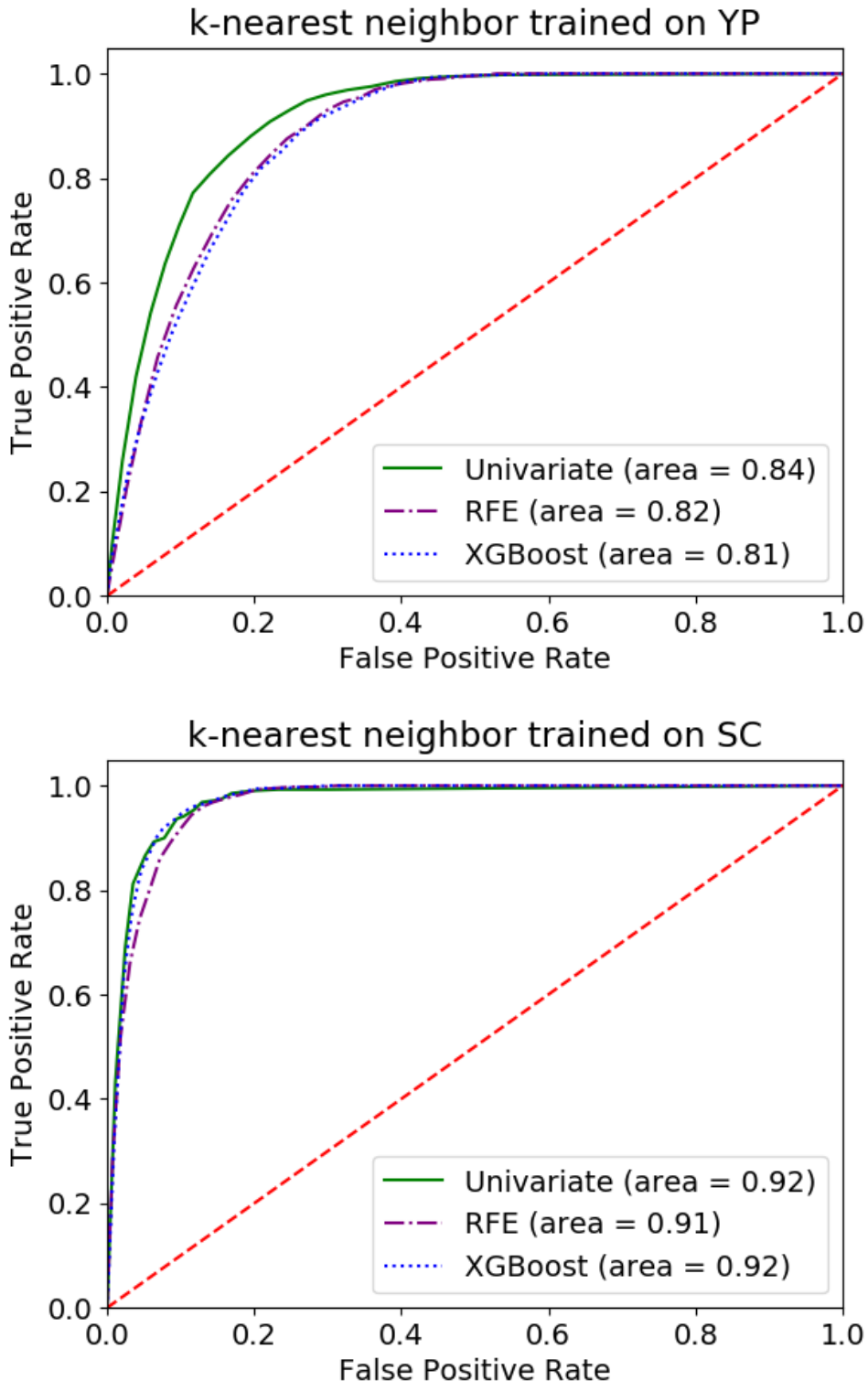


Figure 5.12 ROC curve for k-nearest neighbor trained on sample balanced 3-AAU datasets and tested on 20% unbalanced dataset using three different feature selection methods

Table 5.14 k-nearest neighbor trained on sample unbalanced 3-AAU datasets and tested on 20% unbalanced datasets

Feature Selection	Number of Features	Classifier	Training data	Testing data	Sensitivity	Specificity
XGBoost	6	KNN	Sample YP	20% YP	8%	99%
RFE	6	KNN	Sample YP	20% YP	5%	99%
Univariate	6	KNN	Sample YP	20% YP	23%	99%
XGBoost	6	KNN	Sample SC	20% SC	71%	98%
RFE	6	KNN	Sample SC	20% SC	70%	98%
Univariate	6	KNN	Sample SC	20% SC	78%	98%

Table 5.15 k-nearest neighbor trained on sample unbalanced 3-AAU datasets and tested on full unbalanced datasets

Feature Selection	Number of Features	Classifier	Training data	Testing data	Sensitivity	Specificity
XGBoost	6	KNN	Sample YP	Full YP	8%	99%
RFE	6	KNN	Sample YP	Full YP	5%	99%
Univariate	6	KNN	Sample YP	Full YP	23%	99%
XGBoost	6	KNN	Sample SC	Full SC	72%	97%
RFE	6	KNN	Sample SC	Full SC	69%	97%
Univariate	6	KNN	Sample SC	Full SC	76%	98%

and testing respectively. We again performed the grid search. Like previously, we have also tested our model with full datasets with 5-fold cross-validation. We didn't get good results with KNN classification. When we tested our KNN model trained on unbalanced datasets with full datasets, the model took almost all of computer memory and took lot of time. Cross-validation takes most of the memory of the system and takes lot of time. Still we again tried grid search to get better results but we didn't good results with KNN classification. We realized that KNN may not be suitable for this kind of problem. The Tables 5.14 and 5.15 show the k-nearest neighbor results for the model trained on 80% unbalanced datasets and tested with 20% sample datasets and full datasets respectively.

Table 5.16 k-nearest neighbor trained on sample balanced 3-AAU datasets and tested on full unbalanced datasets from different species

Feature Selection	Number of Features	Classifier	Training data	Testing data	Sensitivity	Specificity
XGBoost	8	KNN	Sample YP	Full SC	90%	80%
XGBoost	8	KNN	Sample YP	Full YP	89%	75%
XGBoost	8	KNN	Sample SC	Full SC	96%	86%
XGBoost	8	KNN	Sample SC	Full YP	85%	66%

As the KNN classification model was not performing well for the unbalanced as well as larger datasets, we just incorporated both the datasets from *Yersinia Pestis* and *Saccharomyces cerevisiae* to test our models trained on balanced and smaller datasets. We found that our model trained on only 6 features of *Yersinia Pestis* is not classifying *Saccharomyces cerevisiae* dataset that well for 3-AAU datasets. Similarly the model trained on *Saccharomyces cerevisiae* dataset with 6 features was not classifying the *Yersinia Pestis* dataset that well. The AUC while testing both the models with different species was approximately only 50%. We again did feature selection using combined features for *Yersinia Pestis* and *Saccharomyces cerevisiae*. We found best results with 8 features using XGBoost feature importance analysis. In summary we can say that KNN classification did perform reasonably well when trained on balanced smaller dataset. However, when trained on unbalanced larger dataset, KNN classification didn't perform well at all. The Table 5.16 shows k-nearest neighbor on 3-AAU datasets which is trained on sample datasets but tested on full datasets from both the species.

5.4 XGBoost

XGBoost stands for extreme gradient boosting. XGBoost is scalable tree boosting system designed and developed by Tianqi Chen [3]. It implements machine learning

algorithms under the Gradient Boosting framework. The model of choice for XGBoost is decision tree ensembles. XGBoost is an implementation of gradient boosted decision trees designed for speed and performance.

Decision tree ensembles consists of a set of classification and regression trees (CART). Decision tree ensembles are widely used in Gradient Boosting Methods, Random Forests, etc. Tree ensemble methods learn higher order interaction between features and are scalable. It is intractable to learn all the trees in a decision tree ensembles at once. Hence an additive strategy is applied to train decision trees rather than taking gradient as in other supervised learning techniques. At each step, we take the tree which optimizes over objective function. XGBoost follows greedy algorithm at each tree split.

We started XGBoost model training with the *Yersinia Pestis* dataset. We divided the *Yersinia Pestis* dataset in to train and test data by 80:20 ratio. We then took balanced data from 80% dataset for training. We did the feature selection by three different ways: Univariate Analysis, Recursive Feature Elimination (RFE) and XGBoost. For feature selection we used whole data for *Yersinia Pestis* dataset. We then trained the XGBoost model using balanced training dataset. We tested our model on unbalanced 20% of the *Yersinia Pestis*. We repeated similar steps for *Saccharomyces cerevisiae* dataset. The Table 5.17 shows XGBoost classification trained on sample balanced 3-AAU datasets and tested on 20% unbalanced data. The Figure 5.13 shows ROC-AUC curve for the XGBoost classification trained on sample balanced 3-AAU datasets and tested on 20% unbalanced data.

In the next step, we trained the XGBoost model on sample balanced *Yersinia Pestis* dataset and tested on full *Yersinia Pestis*s dataset. We repeated the same steps for *Saccharomyces cerevisiae*. The Table 5.18 shows XGBoost classification on 3-AAU datasets which is trained on sample balanced datasets and tested on full unbalanced datasets. The Figure 5.14 shows ROC-AUC curve for the XGBoost classification

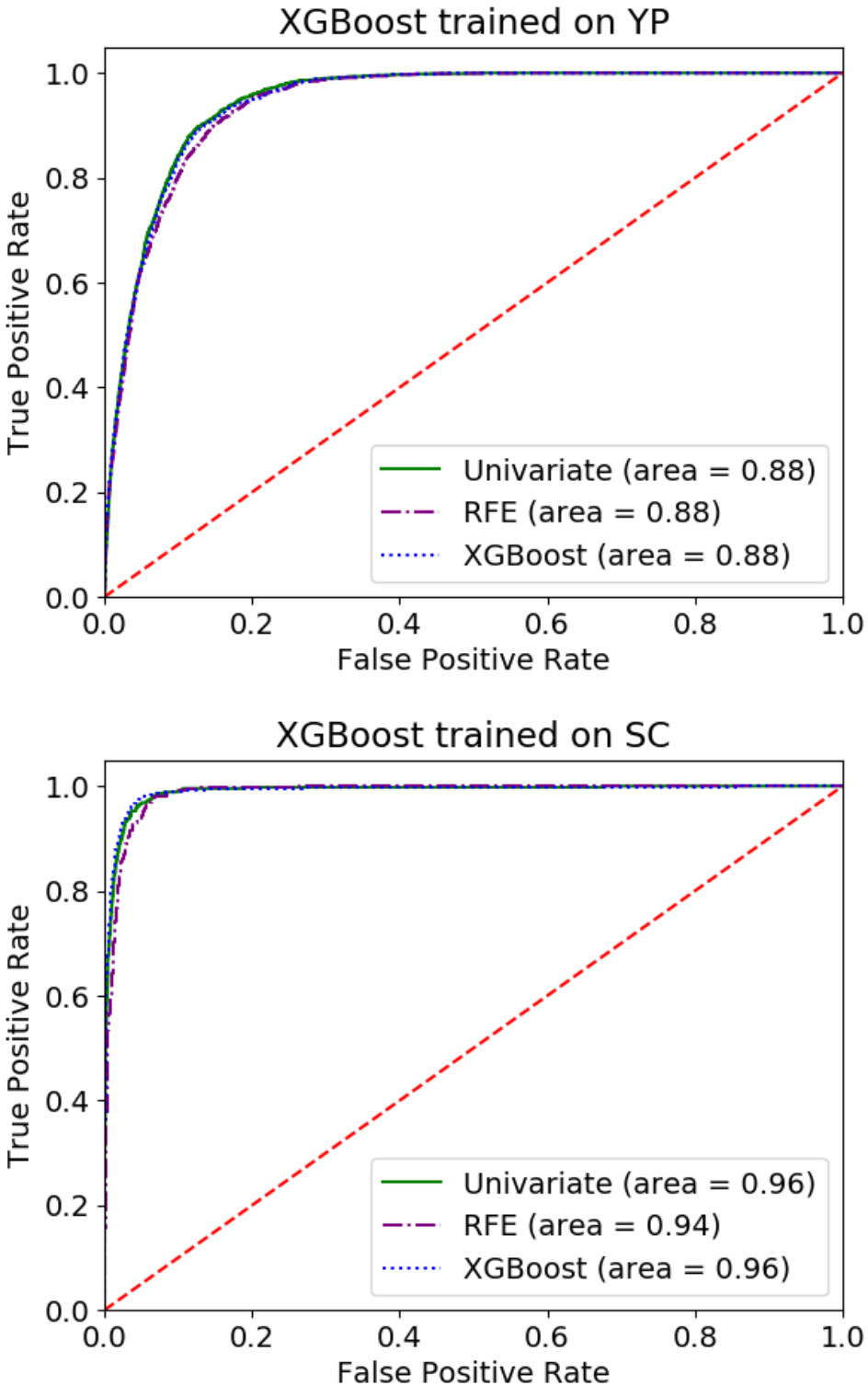


Figure 5.13 ROC curve for XGBoost trained on sample balanced 3-AAU datasets and tested on 20% unbalanced datasets using three different feature selection methods

Table 5.17 XGBoost trained on sample balanced 3-AAU datasets and tested on 20% unbalanced datasets

Feature Selection	Number of Features	Classifier	Training data	Testing data	Sensitivity	Specificity
XGBoost	6	XGBoost	Sample YP	20% YP	90%	86%
RFE	6	XGBoost	Sample YP	20% YP	90%	85%
Univariate	6	XGBoost	Sample YP	20% YP	91%	86%
XGBoost	6	XGBoost	Sample SC	20% SC	95%	97%
RFE	6	XGBoost	Sample SC	20% SC	93%	96%
Univariate	6	XGBoost	Sample SC	20% SC	95%	96%

Table 5.18 XGBoost trained on sample balanced 3-AAU datasets and tested on full unbalanced datasets

Feature Selection	Number of Features	Classifier	Training data	Testing data	Sensitivity	Specificity
XGBoost	6	XGBoost	Sample YP	Full YP	92%	85%
RFE	6	XGBoost	Sample YP	Full YP	92%	84%
Univariate	6	XGBoost	Sample YP	Full YP	93%	84%
XGBoost	6	XGBoost	Sample SC	Full SC	97%	96%
RFE	6	XGBoost	Sample SC	Full SC	96%	95%
Univariate	6	XGBoost	Sample SC	Full SC	97%	96%

trained on sample balanced datasets and tested on full unbalanced datasets.

After getting good results from models trained on sample balanced data, we trained the model on unbalanced sample of *Yersinia Pestis* and *Saccharomyces cerevisiae*. For this we trained and tested our models on 80:20 ratio. We used the same hyper parameters as used earlier for training sample balanced data. Like previously we also tested on full datasets. The Table 5.19 and Table 5.20 shows the XGBoost classification results for the model trained on 80% sample unbalanced datasets and tested on 20% sample datasets and full datasets respectively. The Figure 5.15 and Figure 5.16 shows the ROC-AUC curve for the XGBoost classification results for the

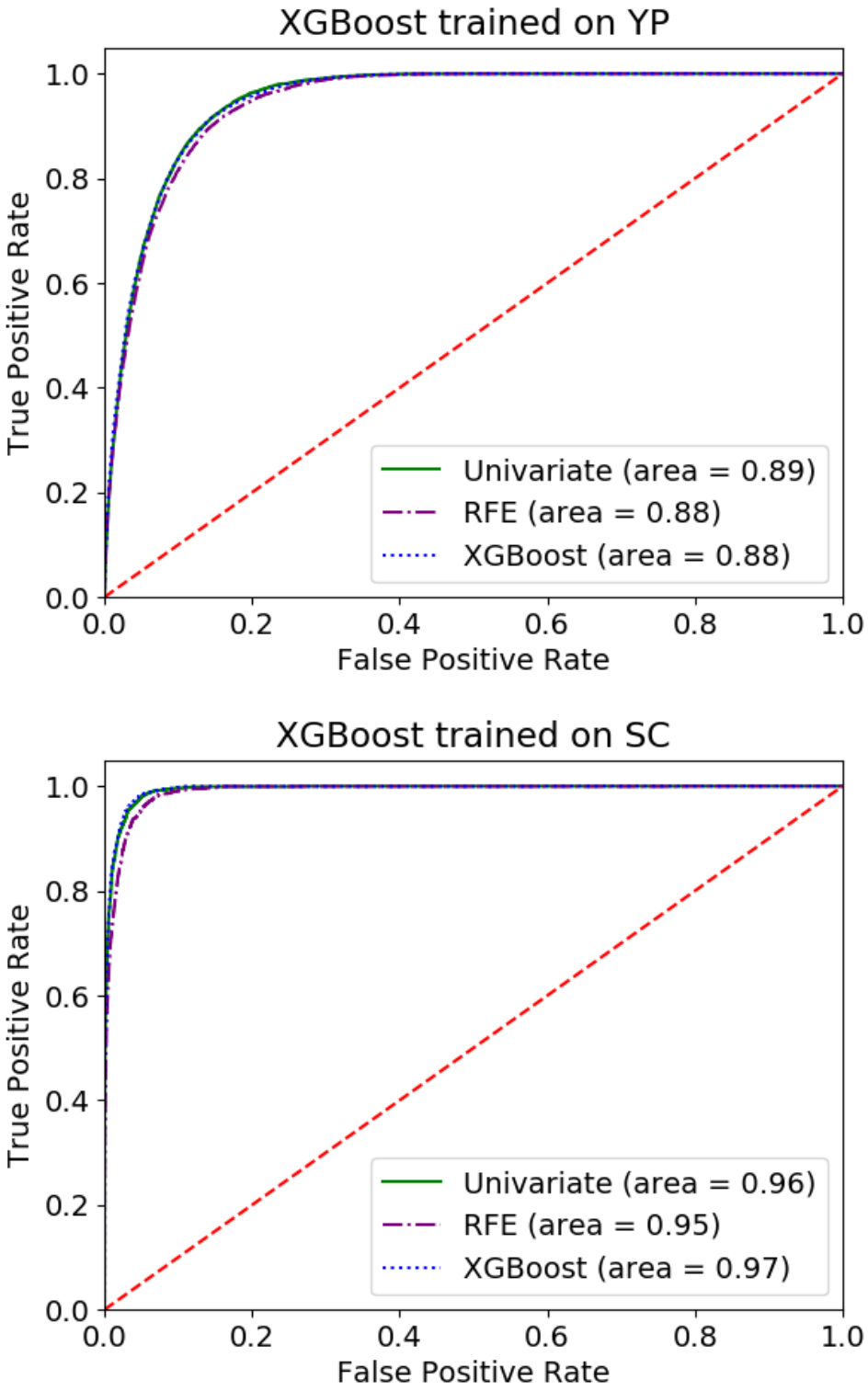


Figure 5.14 ROC curve for XGBoost trained on sample balanced 3-AAU datasets and tested on full unbalanced datasets using three different feature selection methods

Table 5.19 XGBoost trained on sample unbalanced 3-AAU datasets and tested on 20% unbalanced datasets

Feature Selection	Number of Features	Classifier	Training data	Testing data	Sensitivity	Specificity
XGBoost	6	XGBoost	80% YP	20% YP	90%	86%
RFE	6	XGBoost	80% YP	20% YP	90%	85%
Univariate	6	XGBoost	80% YP	20% YP	91%	86%
XGBoost	6	XGBoost	80% SC	20% SC	95%	97%
RFE	6	XGBoost	80% SC	20% SC	93%	95%
Univariate	6	XGBoost	80% SC	20% SC	95%	96%

Table 5.20 XGBoost trained on sample unbalanced 3-AAU datasets and tested on full unbalanced datasets

Feature Selection	Number of Features	Classifier	Training data	Testing data	Sensitivity	Specificity
XGBoost	6	XGBoost	80% YP	Full YP	92%	85%
RFE	6	XGBoost	80% YP	Full YP	92%	84%
Univariate	6	XGBoost	80% YP	Full YP	93%	84%
XGBoost	6	XGBoost	80% SC	Full SC	97%	96%
RFE	6	XGBoost	80% SC	Full SC	96%	95%
Univariate	6	XGBoost	80% SC	Full SC	97%	96%

model trained on 80% sample unbalanced datasets and tested on 20% sample datasets and full datasets respectively.

We then incorporated both the datasets from *Yersinia Pestis* and *Saccharomyces cerevisiae* to test our already trained models. We found that our model trained on only 6 features of Yersinia Pestis is not classifying Yeast dataset that well for 3-AAU datasets. We again did feature selection using XGBoost feature importance, Univariate and RFE on combined calculated features for *Yersinia Pestis* and *Saccharomyces cerevisiae*. But still the classification was way off, we were getting only 50% AUC. Then we employed principal component analysis (PCA) for feature reduction on both

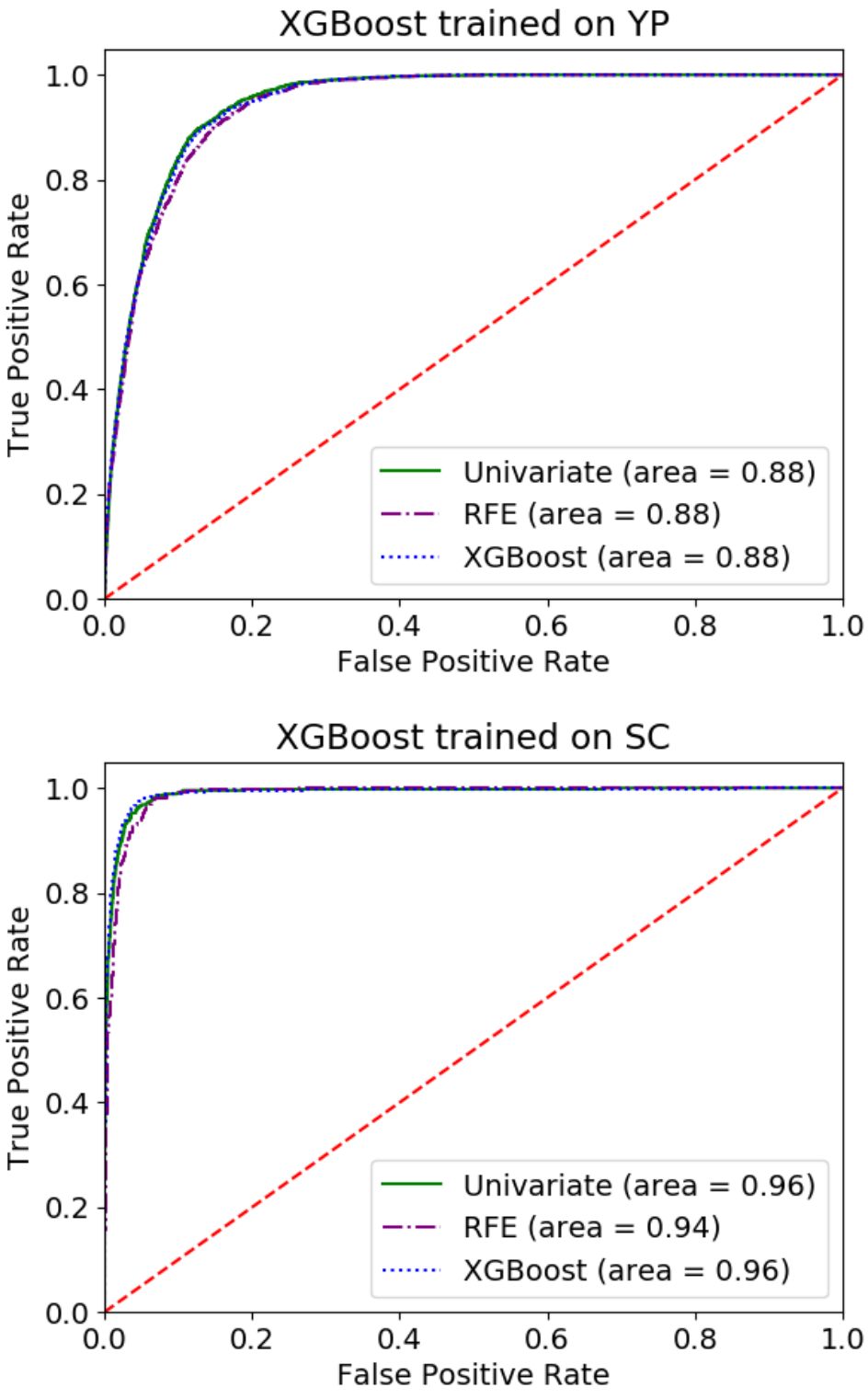


Figure 5.15 ROC curve for XGBoost trained on sample 80% unbalanced 3-AAU datasets and tested on 20% unbalanced datasets using three different feature selection methods

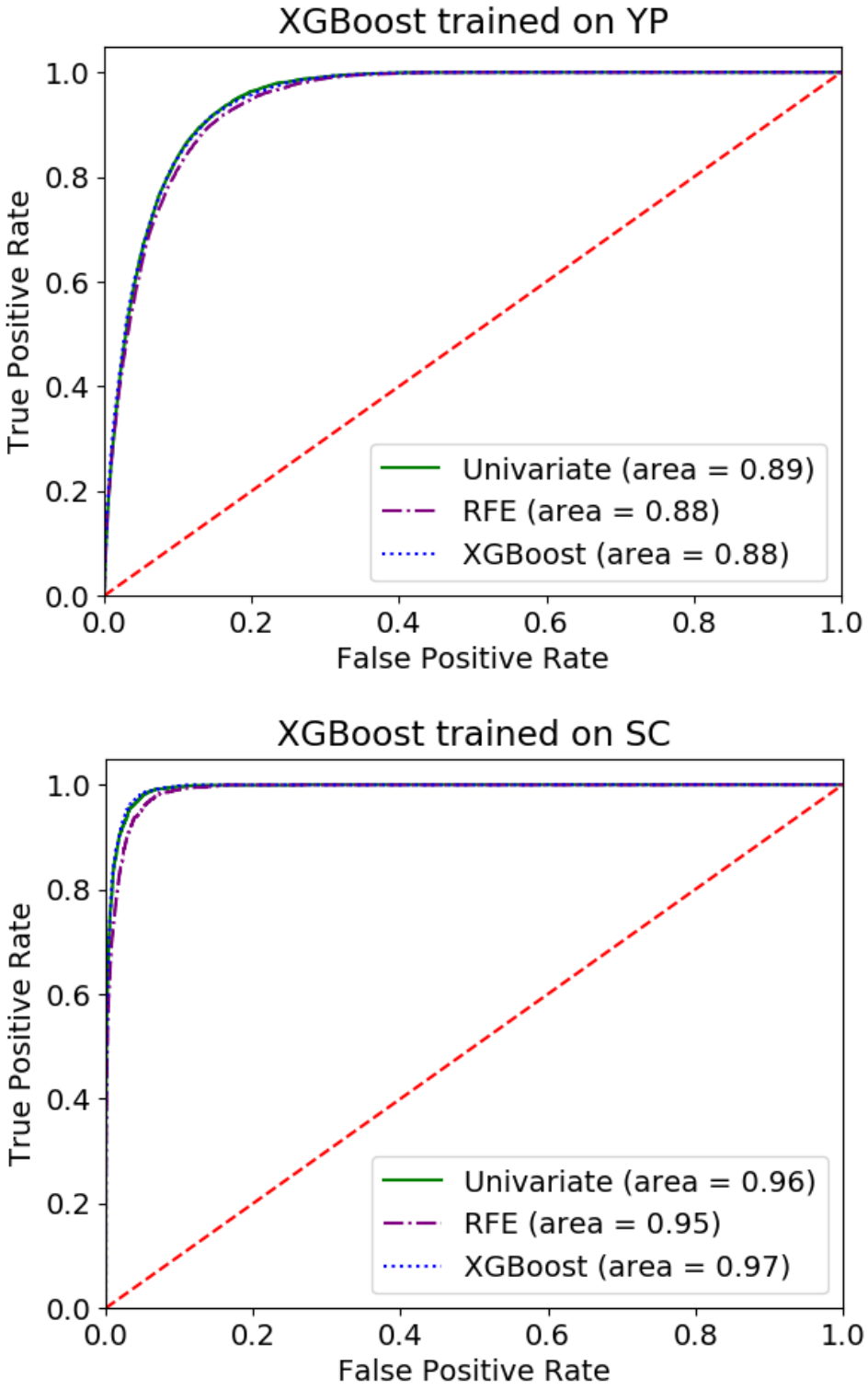


Figure 5.16 ROC curve for XGBoost trained on sample 80% unbalanced 3-AAU datasets and tested on full unbalanced datasets using three different feature selection methods

Table 5.21 XGBoost trained on sample unbalanced 3-AAU datasets and tested on full unbalanced datasets from different species

Feature Selection	Principal Components	Classifier	Training data	Testing data	Sensitivity	Specificity
PCA	7	XGBoost	Sample YP	Full SC	88%	76%
PCA	7	XGBoost	Sample YP	Full YP	84%	80%
PCA	7	XGBoost	Sample SC	Full SC	90%	94%
PCA	7	XGBoost	Sample SC	Full YP	77%	74%

the *Yersinia Pestis* and *Saccharomyces cerevisiae* datasets.

In the PCA analysis we used all the 36 3-AAU dataset features to get 7 principal components that gave us good results. The 7 principal components from *Yersinia Pestis* model covered 82% of the variance. For the model trained on the *Saccharomyces cerevisiae* dataset, 78% of the variance was covered. We have also used 5-fold cross-validation while training our models.

The Table 5.21 shows XGBoost on 2-AAU datasets which is trained on 80% sample unbalanced datasets and tested on full datasets for both the species. The figure 5.17 shows the ROC-AUC curves for the models trained on the *Yersinia Pestis* and *Saccharomyces cerevisiae* datasets using 7 principal components.

In summary, for the XGBoost classification, we got best results with the models trained and tested on the datasets from the same species. We got best AUC score of 89% for the model trained and tested with *Yersinia Pestis* datasets using features set from Univariate analysis. We got best AUC score of 97% for the model trained and tested with *Saccharomyces cerevisiae* datasets using features set from XGBoost feature importance method. However, these models gave AUC of only 50% when tested with the datasets from different species. Hence, we have created 2-AAU normalized datasets to get better results for the models trained and tested on datasets from different species. But for the 2-AAU datasets as well we didn't get any decent results.

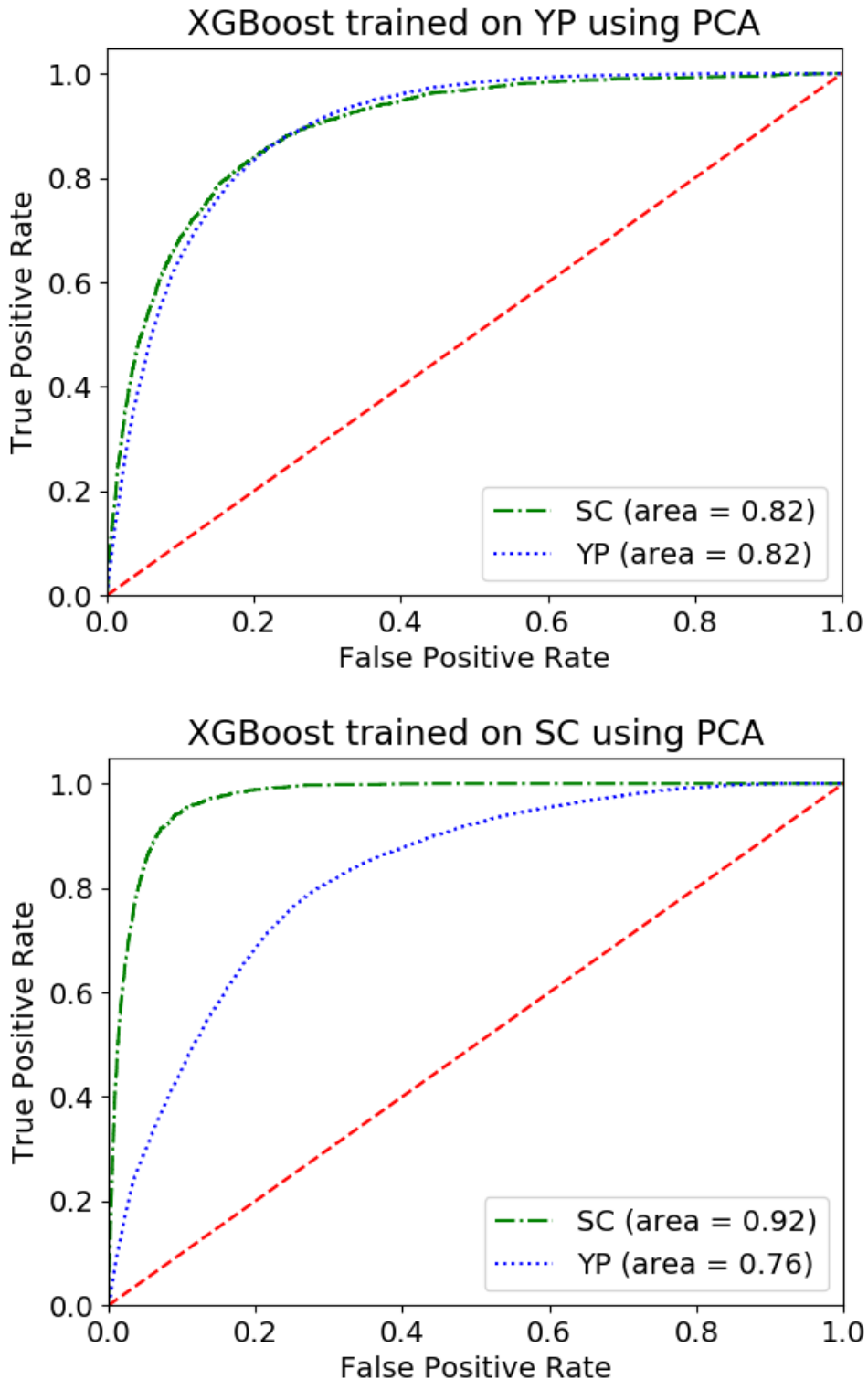


Figure 5.17 ROC curve for XGBoost using the 7 Principal Component Analysis (PCA). Model trained on unbalanced sample 3-AAU datasets and tested on full unbalanced datasets of both the species

We were still getting very low AUC scores with 2-AAU datasets. With PCA, using 7 principal components for 3-AAU normalized datasets, the model trained with the *Yersinia Pestis* dataset gave AUC score of 82% when tested with both the *Yersinia Pestis* and *Saccharomyces cerevisiae* datasets. With PCA, using 7 principal components for 3-AAU datasets, the model trained with the *Saccharomyces cerevisiae* dataset gave AUC score of 92% and 76% when tested with the *Saccharomyces cerevisiae* and *Yersinia Pestis* datasets respectively.

As we observed with our experiments in this chapter that all the models trained on dataset from one organism and tested on the datasets from other organisms were not performing well. We realized that it was happening because of the way we performed the normalization. The min-max scalar was not suitable for our experiments. In Chapter 6, we rerun the experiments on non-normalized datasets.

Chapter 6

Results on non-normalized datasets

We saw in our previous experiments in Chapter 5, to get good results for the model trained and tested on datasets from different organisms we had to employ more features. Also, by using more features we were not getting the same results as we were getting while working on single dataset. We also observed that tree based machine learning algorithms like Random Forest and XGBoost were not performing as well as SVC and logistic regression when trained and tested on datasets from different organism. We realized this may be happening because of the normalization which we did for our datasets. We were using min-max scalar for normalization. We realized min-max scalar was not the right approach to do normalization for these datasets. We decided to do the experiments with the non-normalized datasets. In this chapter. we have included one more dataset to further test our experiments. We have prepared dataset for *Bacillus Subtilis str. 168*. The data preparation methodology is explained in detail in Chapter 2.

We ran our C++ code for C-SVC on the 3-AAU non-normalized datasets. We used the same features that were used by Ahmad Alqurri [11]. We used the same hyper-parameters as described in section 3.4. We ran the model for all three dataset, i.e., *Yersinia Pestis*, *Saccharomyces cerevisiae* and *Bacillus Subtilis str. 168* datasets. The model trained on the sample *Bacillus Subtilis str. 168* dataset was giving low sensitivity score.

We reworked on our C++ code for C-SVC model. We changed the value of C to 5e0. We again trained and tested our model with *Yersinia Pestis* dataset. This time

SVC model trained and tested on *Yersinia Pestis* dataset gave sensitivity of 94% and specificity of 80% with 10-fold cross-validation. The *Yersinia Pestis* model tested with *Saccharomyces cerevisiae* dataset was giving sensitivity of 99% and specificity of 85%. The *Yersinia Pestis* model tested with *Bacillus Subtilis str. 168* dataset was giving sensitivity of 98% and specificity of 77%. The time it took for model to train was approximately 100 minutes.

The model trained on the sample *Saccharomyces cerevisiae* dataset was giving sensitivity of 97% and specificity of 92% with 10-fold cross-validation. The *Saccharomyces cerevisiae* model tested with *Yersinia Pestis* dataset was giving sensitivity of 89% and specificity of 69%. The *Saccharomyces cerevisiae* model tested with *Bacillus Subtilis str. 168* dataset was giving sensitivity of 96% and specificity of 83%.

The model trained on the sample *Bacillus Subtilis str. 168* dataset was giving sensitivity of 92% and specificity of 90% with 10-fold cross-validation. The *Bacillus Subtilis str. 168* model tested with *Yersinia Pestis* dataset was giving sensitivity of 81% and specificity of 62%. The *Bacillus Subtilis str. 168* model tested with *Saccharomyces cerevisiae* dataset was giving sensitivity of 93% and specificity of 94%. The Figure 6.1 shows the time taken (in minutes) to run the SVC model on the three datasets.

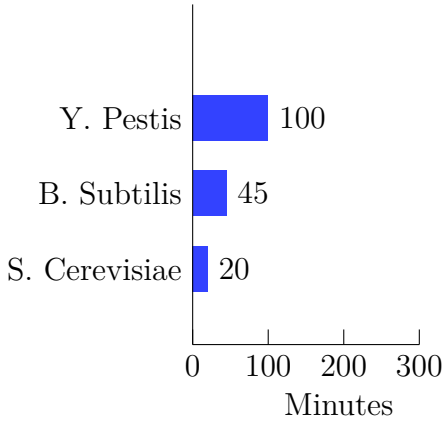


Figure 6.1 Time taken in minutes for C-SVC on three non-normalized datasets

Table 6.1 SVC with Alqurri's [11] features set, trained on sample unbalanced non-normalized 3-AAU datasets and tested on full unbalanced non-normalized 3-AAU datasets from different species

Feature Selection	Number of Features	Classifier	Training data	Testing data	Sensitivity	Specificity
Alqurri [11]	7	SVC	Sample YP	Full SC	99%	88%
Alqurri [11]	7	SVC	Sample YP	Full YP	94%	80%
Alqurri [11]	7	SVC	Sample YP	Full BS	98%	77%
Alqurri [11]	7	SVC	Sample SC	Full SC	97%	92%
Alqurri [11]	7	SVC	Sample SC	Full YP	89%	69%
Alqurri [11]	7	SVC	Sample SC	Full BS	96%	83%
Alqurri [11]	7	SVC	Sample BS	Full SC	93%	94%
Alqurri [11]	7	SVC	Sample BS	Full YP	81%	62%
Alqurri [11]	7	SVC	Sample BS	Full BS	92%	90%

As shown in the Table 6.1, the SVC model trained on the peptides dataset without any normalization from one organisms was able to classify the peptides from another organism in this case. This was great improvement from the situation in Chapter 3, where the SVC model (written in C++) trained on the *Yersinia Pestis* dataset was giving sensitivity of approximately 0% when tested with *Saccharomyces cerevisiae* dataset with Alqurri's [11] feature set.

6.1 Feature rankings using non-normalized datasets

To improve the results for each of the machine learning techniques we have performed feature selection for our experiments in chapter 3 and chapter 5. For non-normalized datasets, we again performed the feature selection using all three datasets, i.e., *Yersinia Pestis*, *Saccharomyces cerevisiae* and *Bacillus Subtilis str. 168*. We have performed feature selection using the same methods described in Chapter 4, i.e., Univariate Analysis, Recursive Feature Elimination (RFE) and XGBoost feature importance method. We made a small change for univariate analysis from Chapter 4, as chi-squared test doesn't take negative values, we have performed ANOVA analysis

for univariate feature selection.

The figures 6.2 and 6.3 show the feature rankings for the combined 3-AAU datasets of *Yersinia Pestis*, *Saccharomyces cerevisiae* and *Bacillus Subtilis str. 168* using Univariate analysis, RFE and XGBoost feature importance. We performed feature reduction through the principal component analysis (PCA). Only one principal component was able to explain 99% of the variance for both the *Yersinia Pestis* and *Saccharomyces cerevisiae* datasets. This was happening because the PCA doesn't perform well with non-normalized datasets. We didn't run our classification experiments using PCA because of this reason with non-normalized datasets.

The AUC score with the univariate features set from the combined three datasets were not giving good results for the model trained on *Bacillus Subtilis str. 168* dataset. To overcome this issue, we have performed the univariate feature analysis with the datasets from *Yersinia Pestis* and *Saccharomyces cerevisiae* only. We have used univariate features set from the univariate analysis on *Yersinia Pestis* and *Saccharomyces cerevisiae* datasets for all the classification algorithms. The Figure 6.4 show the univariate feature scores for the combined non-normalized 3-AAU datasets of *Yersinia Pestis* and *Saccharomyces cerevisiae*.

6.2 Support Vector Classification

In this chapter, we have performed Support Vector classification (SVC) using the non-normalized datasets. We performed SVC using the three features sets from Univariate analysis, Recursive feature elimination (RFE) and XGBoost feature importance analysis. We did feature selection using all the three datasets. Here in Chapter 5, with non-normalized datasets we are able to get better results with SVC. The feature set from univariate analysis, for the model trained on *Bacillus Subtilis str. 168* and tested on *Yersinia Pestis* and *Saccharomyces cerevisiae* datasets was giving sensitivity of only 54% and 56% respectively. To improve the results of SVC classification

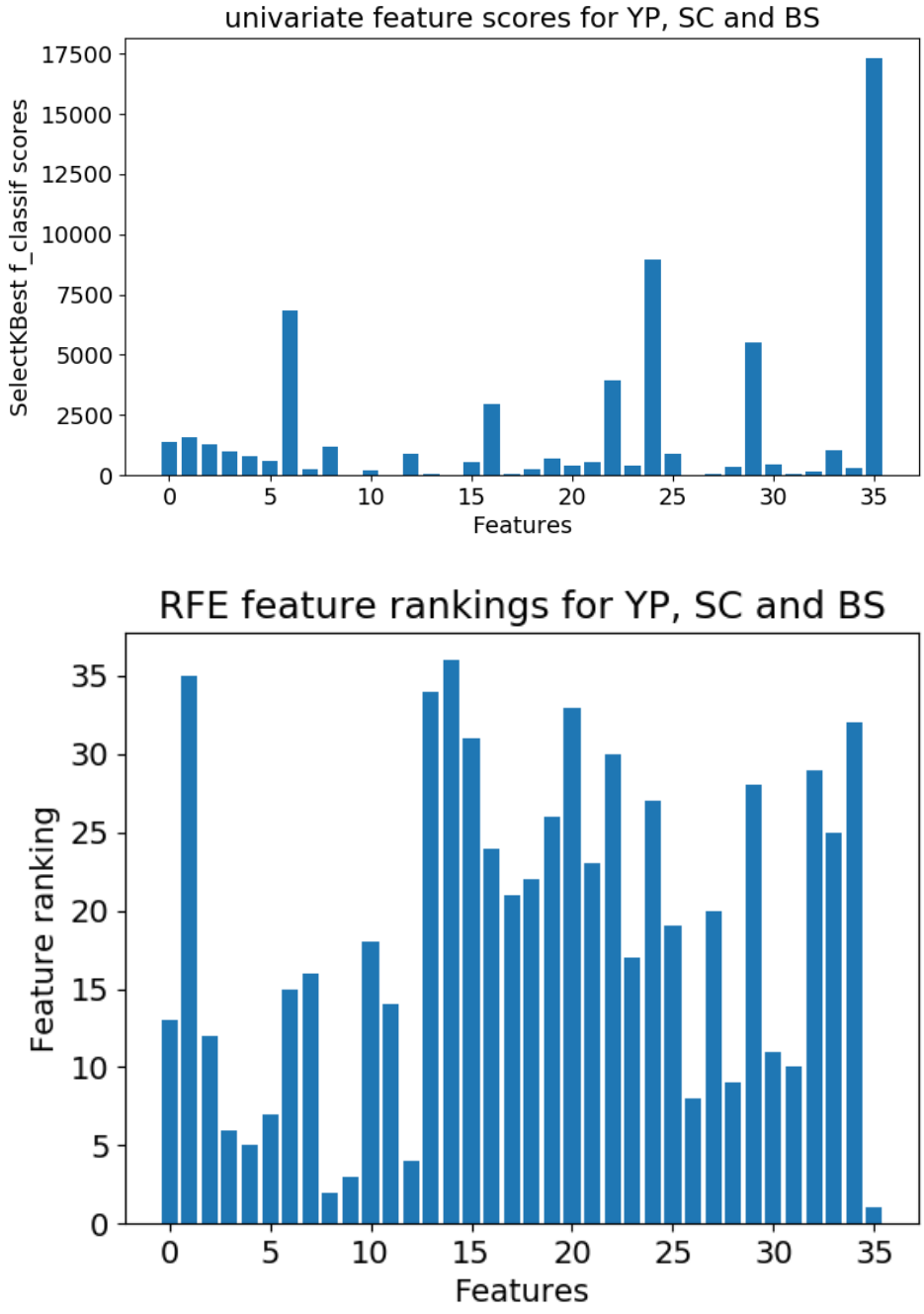


Figure 6.2 Feature ranking through Univariate analysis using ANOVA F-score and Recursive Feature Elimination (RFE) on combined 3-AAU non-normalized datasets of *Yersinia Pestis*, *Saccharomyces cerevisiae* and *Bacillus Subtilis str. 168*

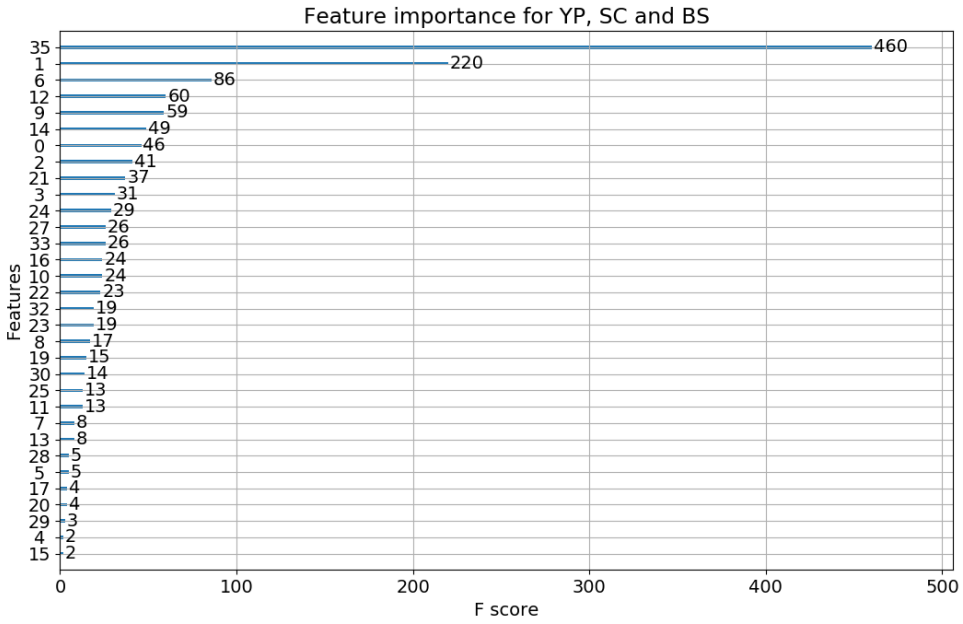


Figure 6.3 Feature ranking using XGBoost feature importance on combined 3-AAU non-normalized datasets of *Yersinia Pestis*, *Saccharomyces cerevisiae* and *Bacillus Subtilis str. 168*

with univariate features set, we have used only *Yersinia Pestis* and *Saccharomyces cerevisiae* datasets for univariate feature analysis.

We employed the same hyper-parameters which were used in the SVC model to train and test *Yersinia Pestis* dataset in Table 3.4. With those same hyper-parameters, we didn't get good results with non-normalized datasets. We again performed the grid search with 5-fold cross-validation. We have used class weights for all the models as the datasets are unbalanced. For the features set from Univariate analysis, we got the best SVC classification results with C = 1e2, gamma = auto or 10 and kernel = rbf. For the features set from RFE analysis, we got best results with C = 15, gamma = auto and kernel = rbf. For the features set from XGBoost feature importance analysis, we got best results with C = 1e7, gamma = 1e-06 and kernel = rbf.

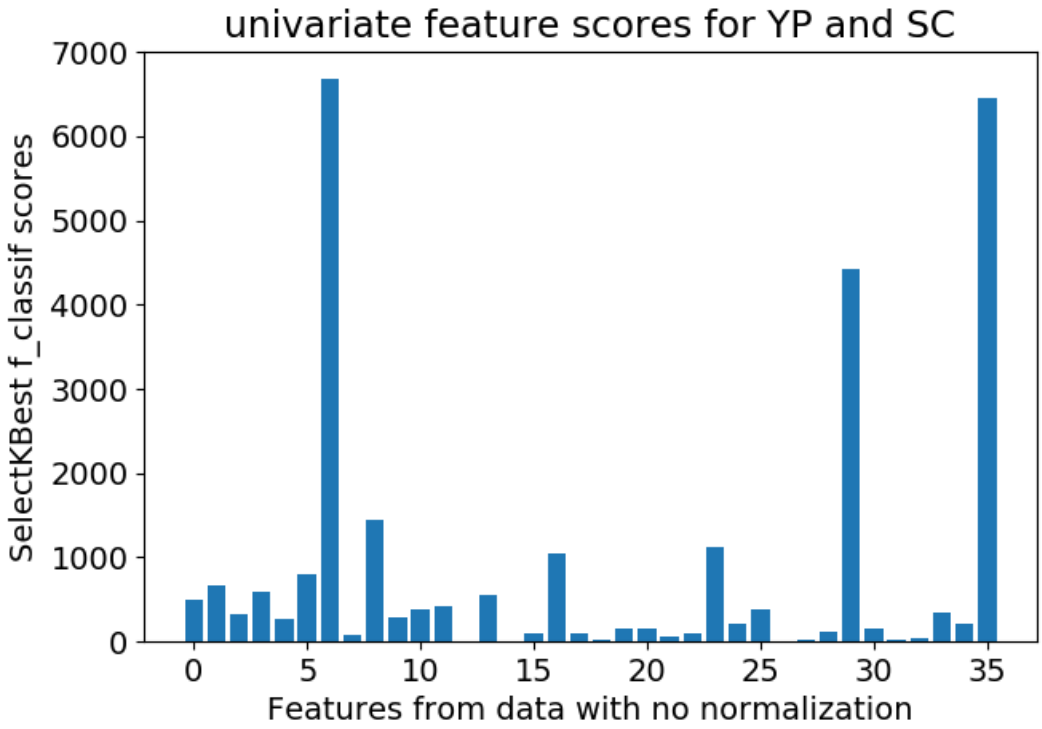


Figure 6.4 Feature ranking through Univariate analysis using ANOVA F-score on combined 3-AAU non-normalized datasets of *Yersinia Pestis* and *Saccharomyces cerevisiae*

We got the best AUC scores for the SVC model using feature set from the XGBoost feature importance analysis. For the model trained on the sample *Yersinia Pestis* dataset and tested on full *Yersinia Pestis*, *Saccharomyces cerevisiae* and *Bacillus Subtilis str. 168* datasets, we got AUC scores of 86%, 94% and 90% respectively. For the model trained on the sample *Saccharomyces cerevisiae* dataset and tested on full *Yersinia Pestis*, *Saccharomyces cerevisiae* and *Bacillus Subtilis str. 168* datasets, we got AUC scores of 81%, 96% and 91% respectively. For the model trained on the sample *Bacillus Subtilis str. 168* dataset and tested on full *Yersinia Pestis*, *Saccharomyces cerevisiae* and *Bacillus Subtilis str. 168* datasets, we got AUC scores of 80%, 83% and 92% respectively.

The Table 6.2 shows the sensitivity and specificity scores for the Support Vector

classification (SVC) model performed on *Yersinia Pestis*, *Saccharomyces cerevisiae* and *Bacillus Subtilis str. 168* non-normalized 3-AAU datasets. The Figures 6.5, 6.6 and 6.7 show the ROC curve with AUC scores for the Support Vector classification (SVC) classification model performed on the three non-normalized 3-AAU datasets using XGBoost feature importance analysis, Recursive feature elimination (RFE) and Univariate analysis respectively.

Table 6.2 SVC trained and tested on non-normalized 3-AAU datasets from different species with three different feature selection methods and using RBF kernel

Feature Selection	Number of features	Classifier	Training data	Testing data	Sensitivity	Specificity
XGBoost	6	SVC	Sample YP	Full SC	99%	89%
XGBoost	6	SVC	Sample YP	Full YP	97%	75%
XGBoost	6	SVC	Sample YP	Full BS	99%	81%
XGBoost	6	SVC	Sample SC	Full SC	99%	93%
XGBoost	6	SVC	Sample SC	Full YP	91%	71%
XGBoost	6	SVC	Sample SC	Full BS	92%	90%
XGBoost	6	SVC	Sample BS	Full SC	99%	86%
XGBoost	6	SVC	Sample BS	Full YP	94%	66%
XGBoost	6	SVC	Sample BS	Full YP	97%	87%
RFE	6	SVC	Sample YP	Full SC	96%	93%
RFE	6	SVC	Sample YP	Full YP	93%	81%
RFE	6	SVC	Sample YP	Full BS	98%	83%
RFE	6	SVC	Sample SC	Full SC	97%	94%
RFE	6	SVC	Sample SC	Full YP	76%	74%
RFE	6	SVC	Sample SC	Full BS	94%	91%
RFE	6	SVC	Sample BS	Full SC	97%	86%
RFE	6	SVC	Sample BS	Full YP	95%	91%
RFE	6	SVC	Sample BS	Full BS	94%	61%
Univariate	6	SVC	Sample YP	Full SC	86%	84%
Univariate	6	SVC	Sample YP	Full YP	94%	79%
Univariate	6	SVC	Sample YP	Full BS	79%	64%
Univariate	6	SVC	Sample SC	Full SC	95%	92%
Univariate	6	SVC	Sample SC	Full YP	87%	80%
Univariate	6	SVC	Sample SC	Full BS	89%	84%
Univariate	6	SVC	Sample BS	Full SC	92%	90%
Univariate	6	SVC	Sample BS	Full YP	87%	67%
Univariate	6	SVC	Sample BS	Full BS	97%	82%

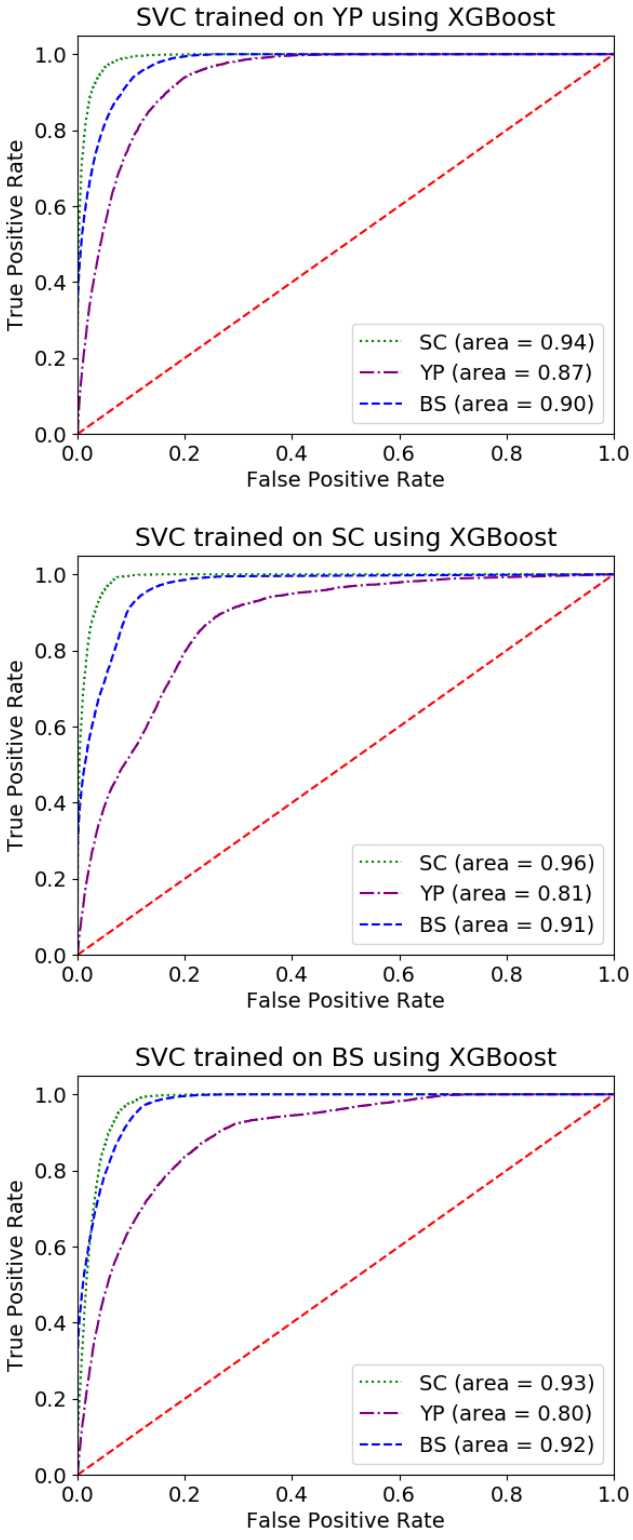


Figure 6.5 ROC curve for Support Vector classification trained on sample unbalanced non-normalized 3-AAU datasets and tested on full non-normalized datasets using XGBoost feature importance analysis

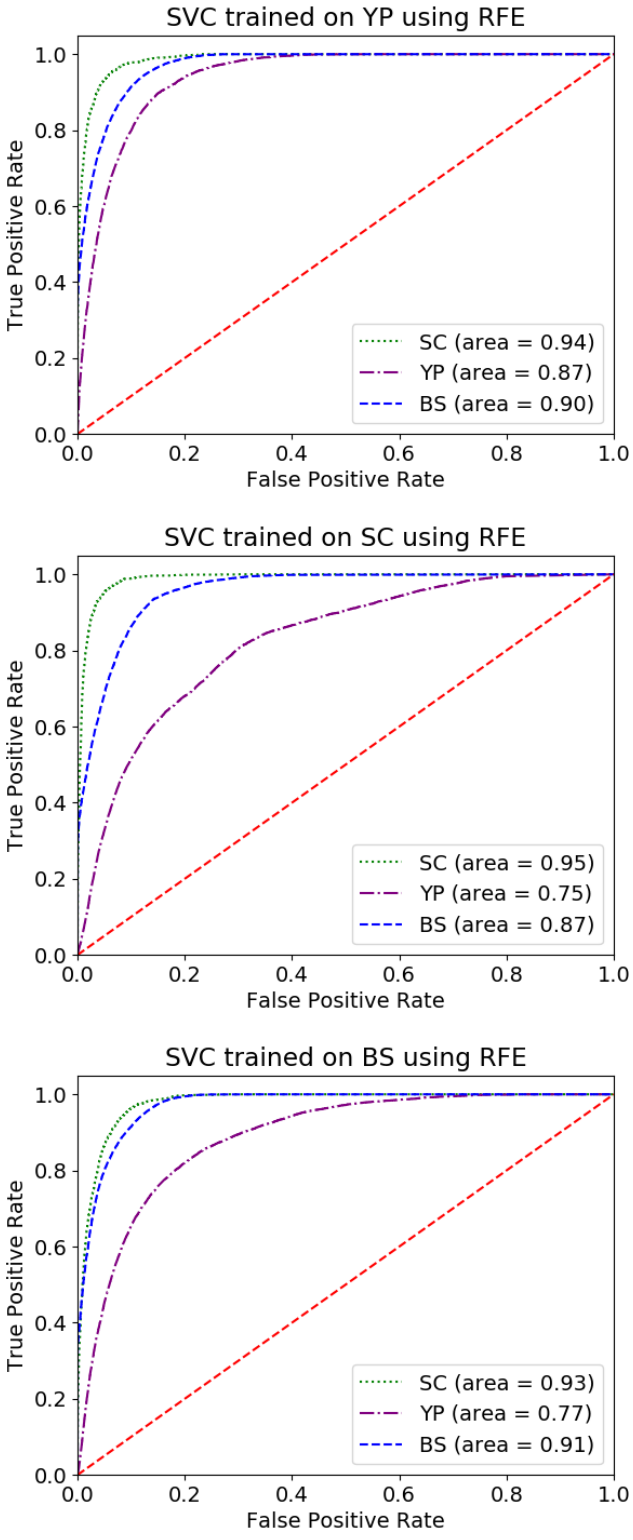
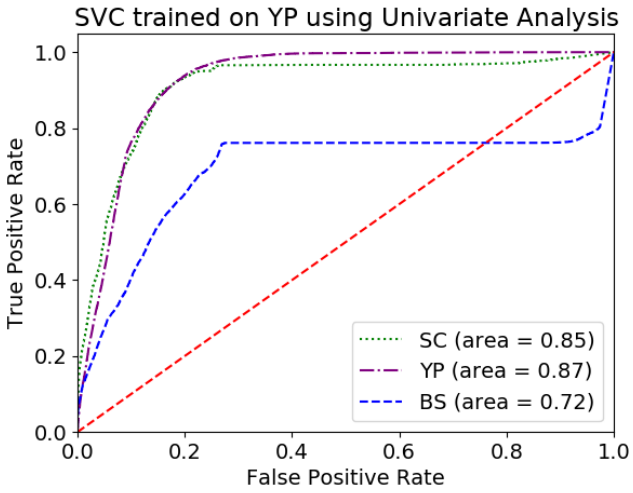
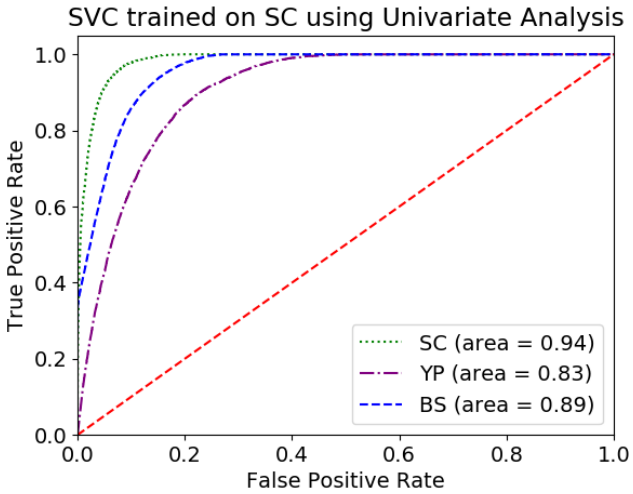


Figure 6.6 ROC curve for Support Vector classification trained on sample unbalanced non-normalized 3-AAU datasets and tested on full non-normalized datasets using RFE analysis





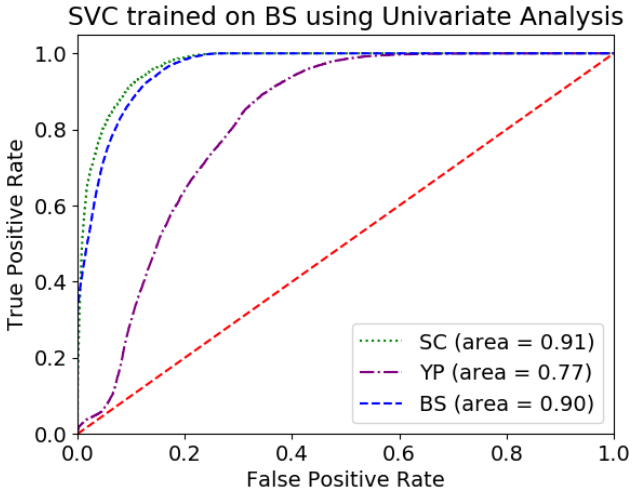


Figure 6.7 ROC curve for Support Vector classification trained on sample unbalanced non-normalized 3-AAU datasets and tested on full non-normalized datasets using Univariate analysis

Table 6.3 Logistic Regression classification trained and tested on non-normalized 3-AAU datasets from different species with three different feature selection methods.

Feature Selection	Number of features	Classifier	Training data	Testing data	Sensitivity	Specificity
XGBoost	6	Logistic Regression	Sample YP	Full SC	98%	89%
XGBoost	6	Logistic Regression	Sample YP	Full YP	94%	79%
XGBoost	6	Logistic Regression	Sample YP	Full BS	98%	77%
XGBoost	6	Logistic Regression	Sample SC	Full SC	97%	93%
XGBoost	6	Logistic Regression	Sample SC	Full YP	99%	57%
XGBoost	6	Logistic Regression	Sample SC	Full BS	99%	73%
XGBoost	6	Logistic Regression	Sample BS	Full SC	97%	91%
XGBoost	6	Logistic Regression	Sample BS	Full YP	82%	67%
XGBoost	6	Logistic Regression	Sample BS	Full BS	95%	88%
RFE	6	Logistic Regression	Sample YP	Full SC	96%	92%
RFE	6	Logistic Regression	Sample YP	Full YP	93%	79%
RFE	6	Logistic Regression	Sample YP	Full BS	98%	77%
RFE	6	Logistic Regression	Sample SC	Full SC	97%	93%
RFE	6	Logistic Regression	Sample SC	Full YP	97%	65%
RFE	6	Logistic Regression	Sample SC	Full BS	99%	76%
RFE	6	Logistic Regression	Sample BS	Full SC	98%	90%
RFE	6	Logistic Regression	Sample BS	Full YP	83%	67%
RFE	6	Logistic Regression	Sample BS	Full BS	95%	87%
Univariate	6	Logistic Regression	Sample YP	Full SC	98%	89%
Univariate	6	Logistic Regression	Sample YP	Full YP	94%	80%
Univariate	6	Logistic Regression	Sample YP	Full BS	98%	77%
Univariate	6	Logistic Regression	Sample SC	Full SC	97%	92%
Univariate	6	Logistic Regression	Sample SC	Full YP	89%	71%
Univariate	6	Logistic Regression	Sample SC	Full BS	96%	82%
Univariate	6	Logistic Regression	Sample BS	Full SC	93%	91%
Univariate	6	Logistic Regression	Sample BS	Full YP	88%	57%
Univariate	6	Logistic Regression	Sample BS	Full BS	95%	85%

6.3 Logistic Regression

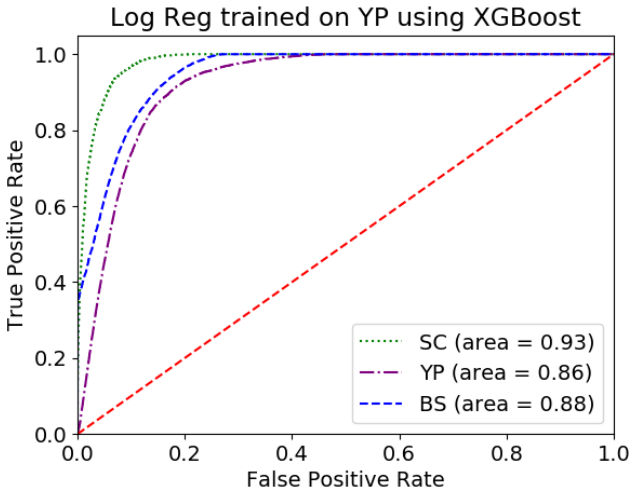
In this chapter, we have performed the logistic regression classification with non-normalized datasets. Our model described in Chapter 5 was not performing well with the non-normalized dataset. This time we did the feature selection using all three datasets. We again performed the grid search for the logistic regression model. We have used 5-fold cross-validation for our experiments. For the features set obtained from XGBoost feature importance and RFE, all the models gave good results for C

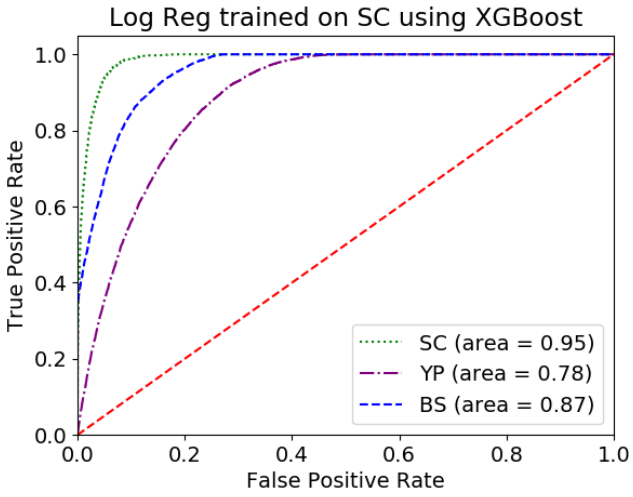
= 1e10, solver = liblinear and multi-class = ovr. The feature set from Univariate analysis for the model trained on *Bacillus Subtilis str. 168* and tested on *Yersinia Pestis* and *Saccharomyces cerevisiae* datasets was giving AUC of only 54% and 57% respectively. For Univariate feature analysis we have only used *Yersinia Pestis* and *Saccharomyces cerevisiae* datasets.

We got best AUC scores for the Logistic Regression classification using feature set from Recursive Feature Elimination (RFE) analysis. For the model trained on the sample *Yersinia Pestis* dataset and tested on full *Yersinia Pestis*, *Saccharomyces cerevisiae* and *Bacillus Subtilis str. 168* datasets, we got AUC scores of 86%, 94% and 88% respectively. For the model trained on the sample *Saccharomyces cerevisiae* dataset and tested on full *Yersinia Pestis*, *Saccharomyces cerevisiae* and *Bacillus Subtilis str. 168* datasets, we got AUC scores of 81%, 95% and 87% respectively. For the model trained on the sample *Bacillus Subtilis str. 168* dataset and tested on full *Yersinia Pestis*, *Saccharomyces cerevisiae* and *Bacillus Subtilis str. 168* datasets, we got AUC scores of 75%, 94% and 91% respectively.

The Table 6.3 shows the results for logistic regression for both the *Yersinia Pestis* and *Saccharomyces cerevisiae* non-normalized datasets. If we observe the sensitivity and specificity in the Table 6.3, we see that results from features selected through XGBoost feature importance, RFE and Univariate analysis are much better for the non-normalized datasets compared to the normalized datasets. The Figures 6.8, 6.9 and 6.10 show the ROC curve with AUC scores for the logistic regression model performed on the non-normalized 3-AAU datasets using XGBoost feature importance analysis, Recursive feature elimination and Univariate analysis respectively.

The Logistic Regression classification model trained on *Yersinia Pestis* dataset took approximately 0.5 minute to train and cross-validate, which is almost 320 times faster than C++ version of SVM classification model described earlier in this chapter.





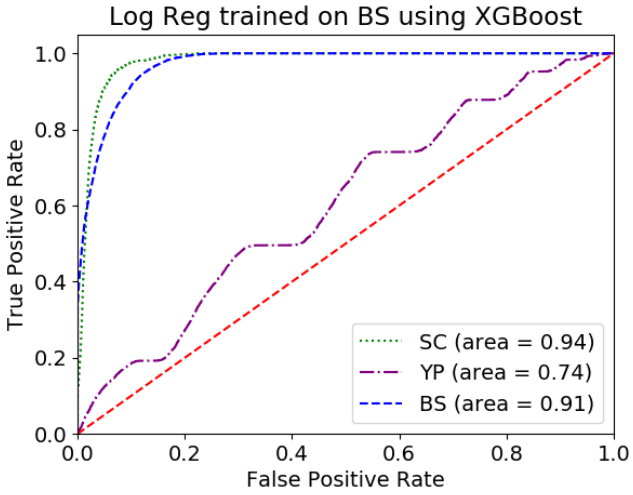


Figure 6.8 ROC curve for Logistic Regression classification trained on sample unbalanced non-normalized 3-AAU datasets and tested on full datasets using XGBoost feature importance analysis

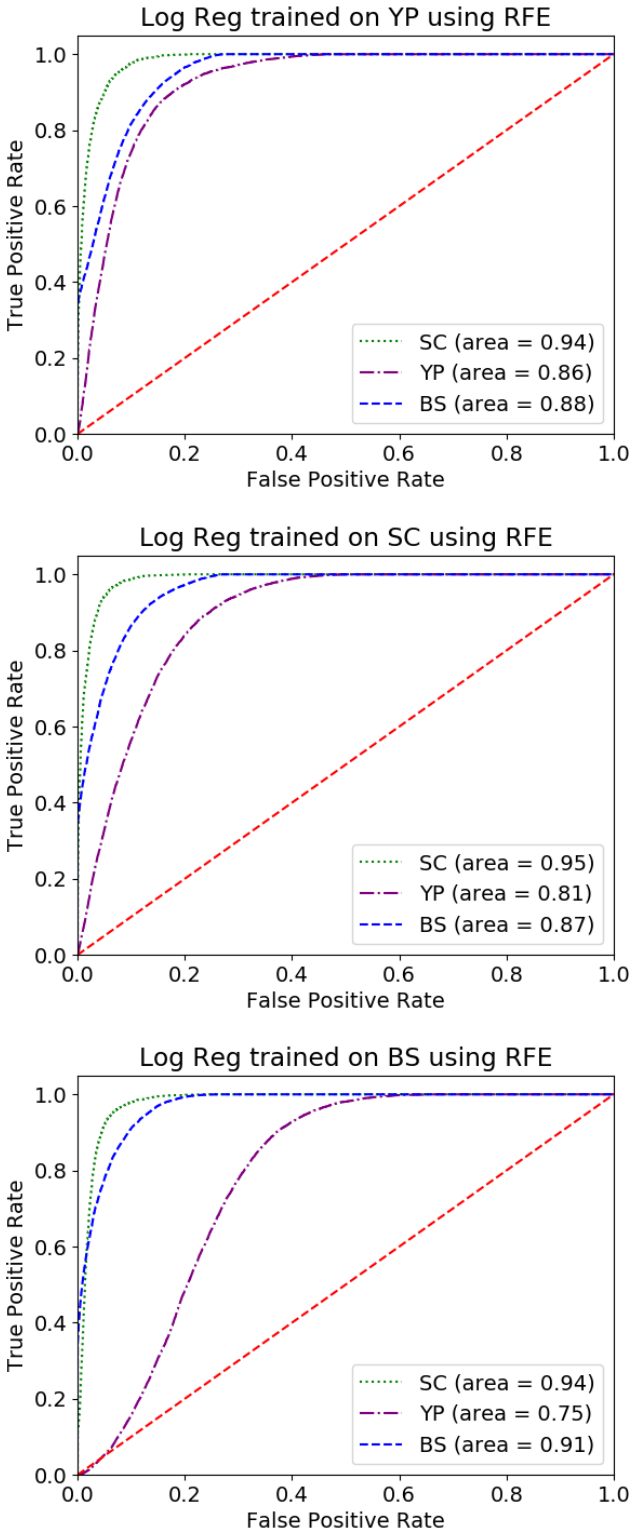
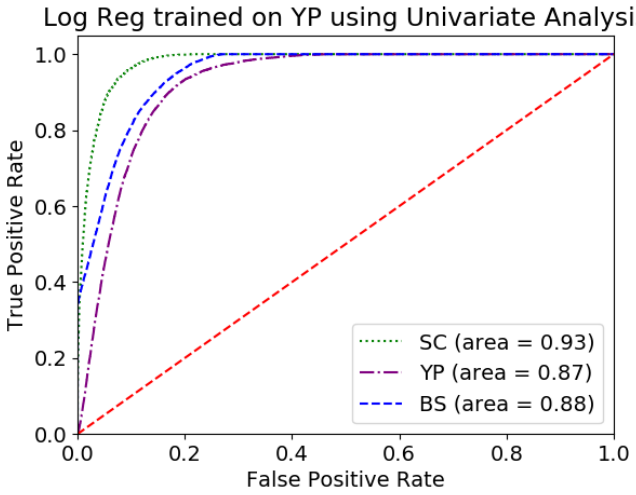
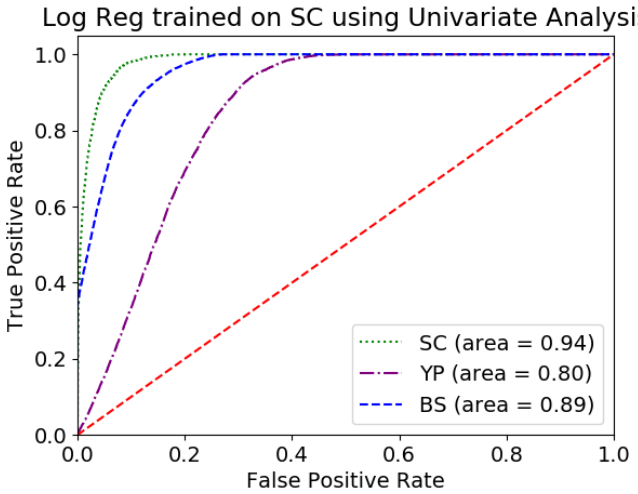


Figure 6.9 ROC curve for Logistic Regression classification trained on sample unbalanced non-normalized 3-AAU datasets and tested on full datasets using RFE analysis





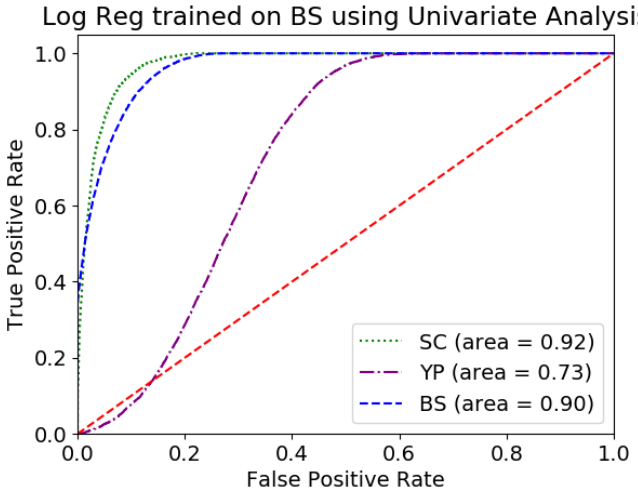


Figure 6.10 ROC curve for Logistic Regression classification trained on sample unbalanced non-normalized 3-AAU datasets and tested on full datasets using Univariate analysis

6.4 Random Forest

In this chapter, we have performed Random Forest classification using non-normalized datasets. We performed Random Forest classification using the three features sets from univariate analysis, recursive feature elimination (RFE) and XGBoost feature importance analysis. We did feature selection using all the three datasets. Unlike in Chapter 5, here in case of non-normalized datasets, we are able to get good results.

The feature set from Univariate analysis, for the model trained on *Bacillus Subtilis str. 168* and tested on *Yersinia Pestis* and *Saccharomyces cerevisiae* datasets was giving AUC of only 72% and 79% respectively. To further improve the results of classification using Univariate feature set, we have used only *Yersinia Pestis* and *Saccharomyces cerevisiae* datasets for Univariate feature analysis.

We got best AUC scores for Random Forest classification using feature set from XGBoost feature importance method. For the model trained on sample *Yersinia Pestis* and tested on full *Yersinia Pestis*, *Saccharomyces cerevisiae* and *Bacillus Subtilis str. 168* we got AUC scores of 85%, 95% and 91% respectively. For the model trained on sample *Saccharomyces cerevisiae* and tested on full *Yersinia Pestis*, *Saccharomyces cerevisiae* and *Bacillus Subtilis str. 168* we got AUC scores of 80%, 96% and 91% respectively. For the model trained on sample *Bacillus Subtilis str. 168* and tested on full *Yersinia Pestis*, *Saccharomyces cerevisiae* and *Bacillus Subtilis str. 168* we got AUC scores of 81%, 95% and 92% respectively.

The Table 6.4 shows the results for Random Forest classification for the *Yersinia Pestis*, *Saccharomyces cerevisiae* and *Bacillus Subtilis str. 168* non-normalized datasets using XGBoost feature importance analysis, Recursive feature elimination and Univariate analysis. The Figures 6.11, 6.12 and 6.13 shows the AUC scores for the Random Forest classification performed on the non-normalized datasets using XGBoost feature importance analysis, Recursive feature elimination and Univariate analysis respectively.

Table 6.4 Random Forest classification trained and tested on non-normalized 3-AAU datasets from different species with three different feature selection methods.

Feature Selection	No. of features	Classifier	Training data	Testing data	Sensitivity	Specificity
XGBoost	6	Random Forest	Sample YP	Full SC	94%	96%
XGBoost	6	Random Forest	Sample YP	Full YP	78%	91%
XGBoost	6	Random Forest	Sample YP	Full BS	91%	90%
XGBoost	6	Random Forest	Sample SC	Full SC	96%	97%
XGBoost	6	Random Forest	Sample SC	Full YP	68%	91%
XGBoost	6	Random Forest	Sample SC	Full BS	89%	92%
XGBoost	6	Random Forest	Sample BS	Full SC	96%	94%
XGBoost	6	Random Forest	Sample BS	Full YP	74%	89%
XGBoost	6	Random Forest	Sample BS	Full BS	93%	91%
RFE	6	Random Forest	Sample YP	Full SC	94%	94%
RFE	6	Random Forest	Sample YP	Full YP	78%	91%
RFE	6	Random Forest	Sample YP	Full BS	91%	89%
RFE	6	Random Forest	Sample SC	Full SC	94%	96%
RFE	6	Random Forest	Sample SC	Full YP	67%	91%
RFE	6	Random Forest	Sample SC	Full BS	84%	92%
RFE	6	Random Forest	Sample BS	Full SC	94%	92%
RFE	6	Random Forest	Sample BS	Full YP	75%	85%
RFE	6	Random Forest	Sample BS	Full BS	92%	90%
Univariate	6	Random Forest	Sample YP	Full SC	92%	92%
Univariate	6	Random Forest	Sample YP	Full YP	81%	88%
Univariate	6	Random Forest	Sample YP	Full BS	93%	85%
Univariate	6	Random Forest	Sample SC	Full SC	93%	94%
Univariate	6	Random Forest	Sample SC	Full YP	74%	86%
Univariate	6	Random Forest	Sample SC	Full BS	94%	85%
Univariate	6	Random Forest	Sample BS	Full SC	92%	88%
Univariate	6	Random Forest	Sample BS	Full YP	72%	93%
Univariate	6	Random Forest	Sample BS	Full BS	92%	88%

The Random Forest classification model trained on *Yersinia Pestis* dataset took approximately 4 minutes to train and cross-validate, which is almost 25 times faster than C++ version of SVM classification described earlier in this chapter. Also, the Random Forest classifier is the only classifier that is giving us higher specificity than sensitivity.

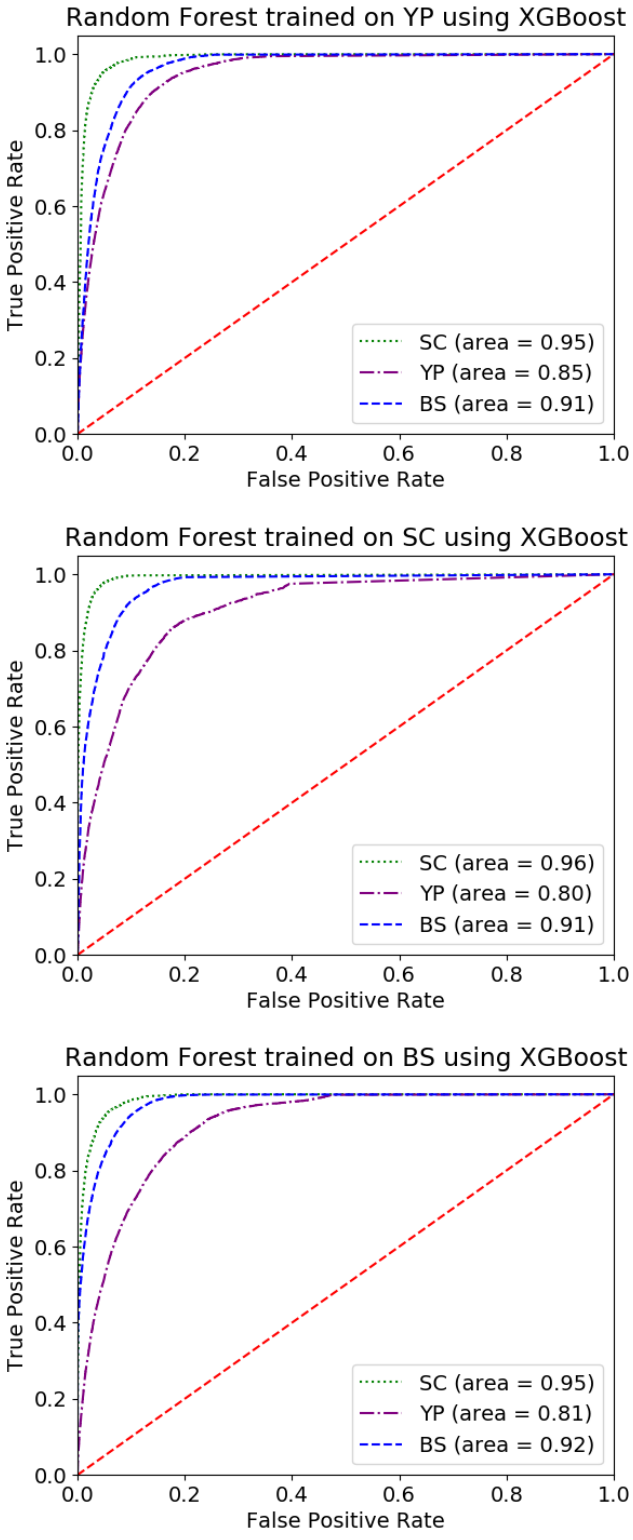


Figure 6.11 ROC curve for Random Forest classification trained on sample unbalanced non-normalized 3-AAU datasets and tested on full datasets using XGBoost feature importance analysis

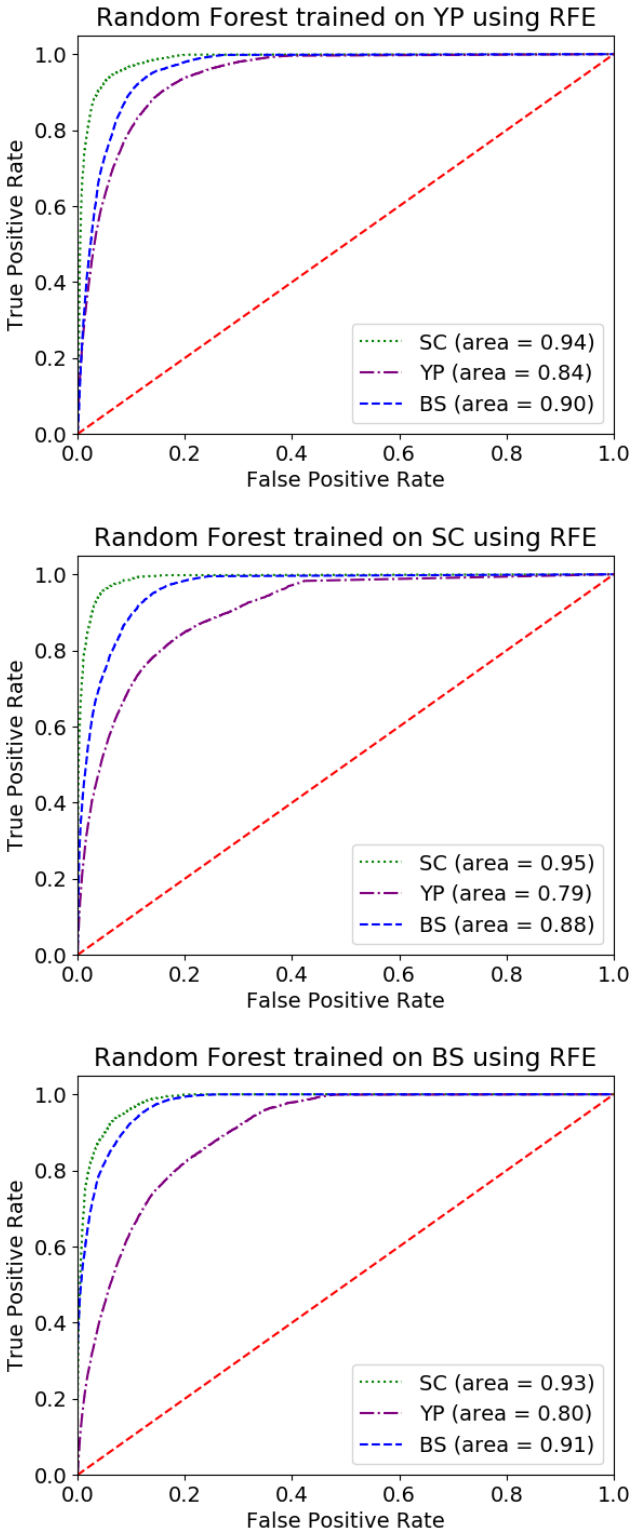


Figure 6.12 ROC curve for Random Forest classification trained on sample unbalanced non-normalized 3-AAU datasets and tested on full datasets using RFE analysis

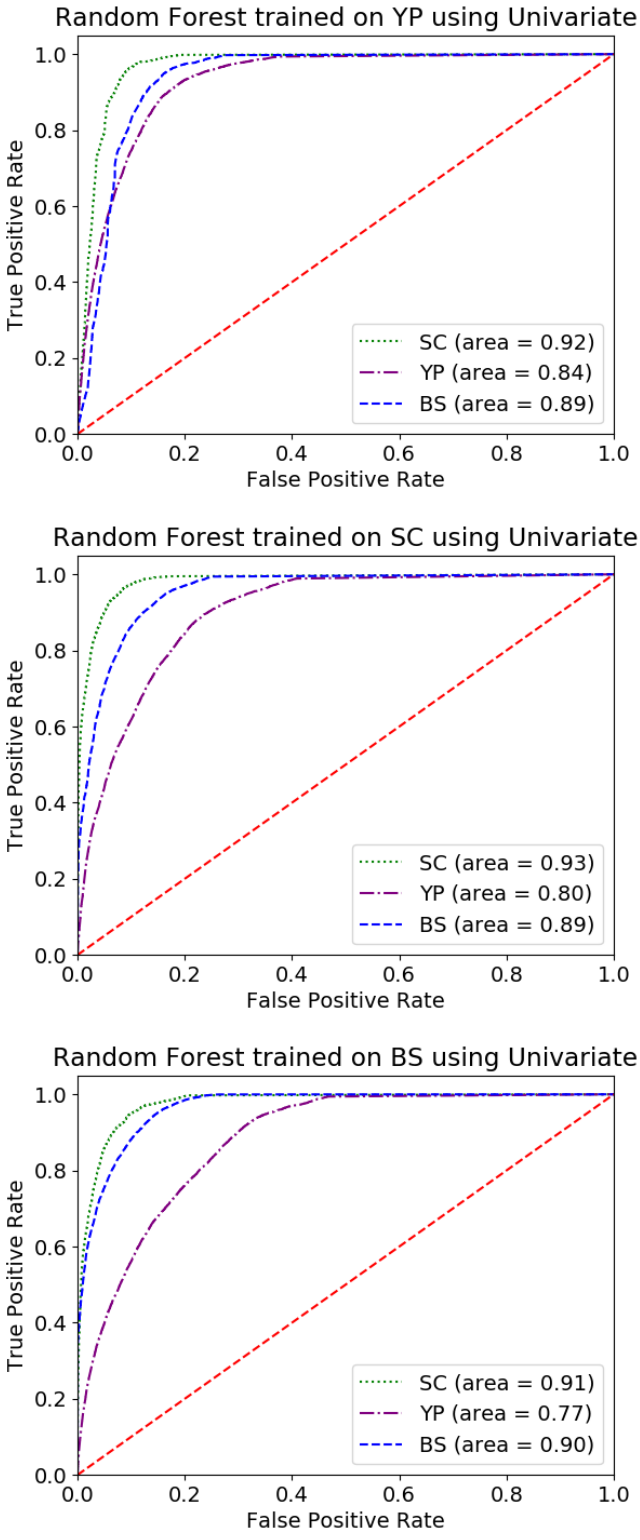


Figure 6.13 ROC curve for Random Forest classification trained on sample unbalanced non-normalized 3-AAU datasets and tested on full datasets using Univariate analysis

6.5 XGBoost

In this chapter, we performed XGBoost classification on non-normalized datasets. We performed XGBoost classification using the three features sets from Univariate analysis, RFE and XGBoost feature importance analysis. We did feature selection using all the three datasets. Unlike in Chapter 5, here with non-normalized datasets we are able to get good results with XGBoost classification. The features set from univariate analysis, for the model trained on *Bacillus Subtilis str. 168* dataset and tested on *Yersinia Pestis* and *Saccharomyces cerevisiae* datasets was giving sensitivity of only 63% and 67% respectively. To improve the results of classification with univariate features set, we have used only *Yersinia Pestis* and *Saccharomyces cerevisiae* datasets for univariate feature analysis.

We got the best AUC scores for XGBoost classification using features set from XGBoost feature importance method. For the model trained on sample *Yersinia Pestis* dataset and tested on full *Yersinia Pestis*, *Saccharomyces cerevisiae* and *Bacillus Subtilis str. 168* datasets, we got AUC scores of 88%, 96% and 91% respectively. For the model trained on sample *Saccharomyces cerevisiae* dataset and tested on full *Yersinia Pestis*, *Saccharomyces cerevisiae* and *Bacillus Subtilis str. 168* datasets, we got AUC scores of 87%, 96% and 91% respectively. For the model trained on sample *Bacillus Subtilis str. 168* dataset and tested on full *Yersinia Pestis*, *Saccharomyces cerevisiae* and *Bacillus Subtilis str. 168* datasets, we got AUC scores of 83%, 96% and 92% respectively.

The Table 6.5 shows the results for XGBoost classification on *Yersinia Pestis*, *Saccharomyces cerevisiae* and *Bacillus Subtilis str. 168* non-normalized 3-AAU datasets using three feature selection methods. The Figures 6.14, 6.15 and 6.16 show the ROC curve with AUC scores for the XGBoost classification performed on the non-normalized 3-AAU datasets using XGBoost feature importance analysis, Recursive feature elimination and Univariate Analysis respectively.

The XGBoost classification model on *Yersinia Pestis* dataset took approximately 10 minutes to train and cross-validate. This is almost 10 times faster than C++ version of SVM classification described earlier in this chapter. Overall, we are getting best AUC scores with the XGBoost classifier with all the three feature selection methods that we have performed.

Table 6.5 XGBoost classification on 3-AAU datasets trained and tested on non-normalized 3-AAU datasets from different species with three different feature selection methods.

Feature Selection	No. of features	Classifier	Training data	Testing data	Sensitivity	Specificity
XGBoost	6	XGBoost	Sample YP	Full SC	99%	93%
XGBoost	6	XGBoost	Sample YP	Full YP	92%	84%
XGBoost	6	XGBoost	Sample YP	Full BS	97%	86%
XGBoost	6	XGBoost	Sample SC	Full SC	98%	95%
XGBoost	6	XGBoost	Sample SC	Full YP	91%	82%
XGBoost	6	XGBoost	Sample SC	Full BS	94%	89%
XGBoost	6	XGBoost	Sample BS	Full SC	98%	94%
XGBoost	6	XGBoost	Sample BS	Full YP	91%	75%
XGBoost	6	XGBoost	Sample BS	Full BS	96%	88%
RFE	6	XGBoost	Sample YP	Full SC	98%	89%
RFE	6	XGBoost	Sample YP	Full YP	92%	83%
RFE	6	XGBoost	Sample YP	Full BS	98%	90%
RFE	6	XGBoost	Sample SC	Full SC	97%	93%
RFE	6	XGBoost	Sample SC	Full YP	81%	84%
RFE	6	XGBoost	Sample SC	Full BS	92%	89%
RFE	6	XGBoost	Sample BS	Full SC	96%	91%
RFE	6	XGBoost	Sample BS	Full YP	84%	78%
RFE	6	XGBoost	Sample BS	Full BS	96%	87%
Univariate	6	XGBoost	Sample YP	Full SC	97%	89%
Univariate	6	XGBoost	Sample YP	Full YP	93%	82%
Univariate	6	XGBoost	Sample YP	Full BS	97%	81%
Univariate	6	XGBoost	Sample SC	Full SC	97%	91%
Univariate	6	XGBoost	Sample SC	Full YP	89%	80%
Univariate	6	XGBoost	Sample SC	Full BS	96%	81%
Univariate	6	XGBoost	Sample BS	Full SC	93%	92%
Univariate	6	XGBoost	Sample BS	Full YP	83%	77%
Univariate	6	XGBoost	Sample BS	Full BS	96%	84%

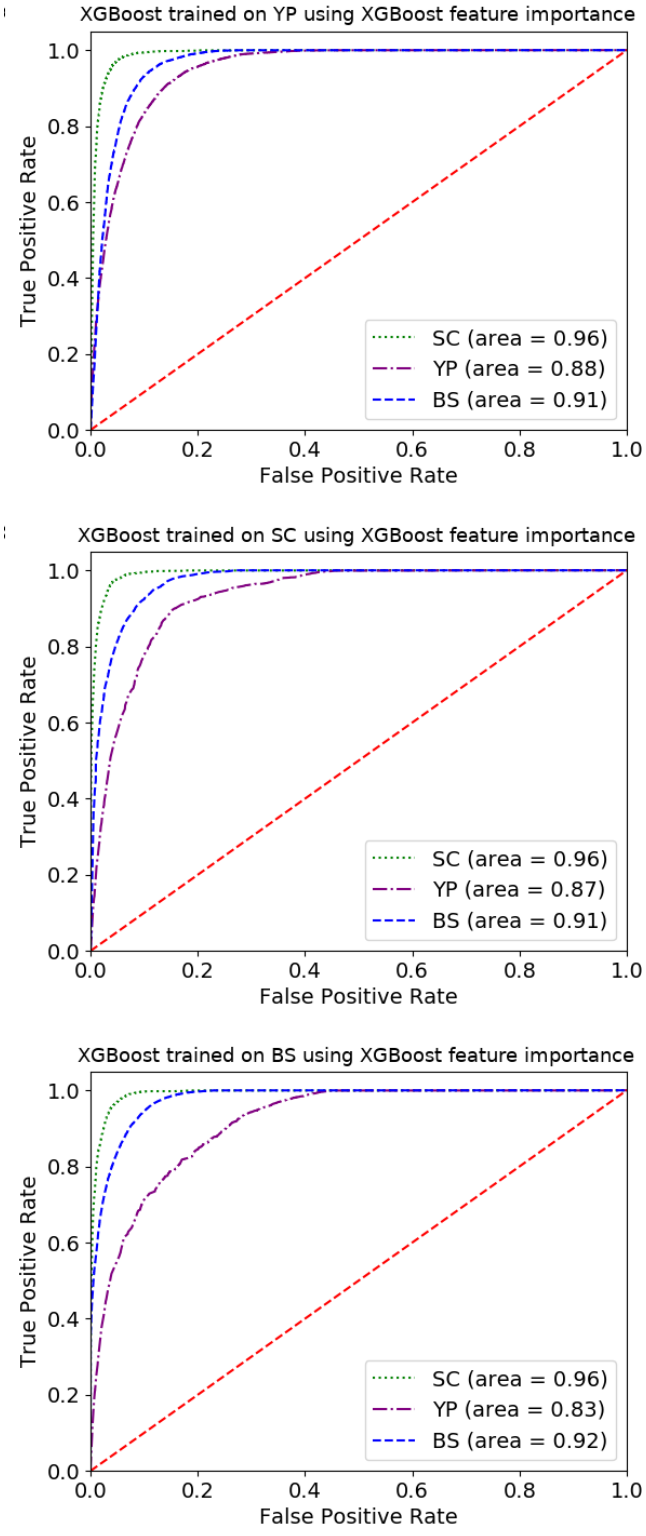


Figure 6.14 ROC curve for XGBoost classification trained on sample unbalanced non-normalized 3-AAU datasets and tested on full non-normalized datasets using XGBoost feature importance analysis

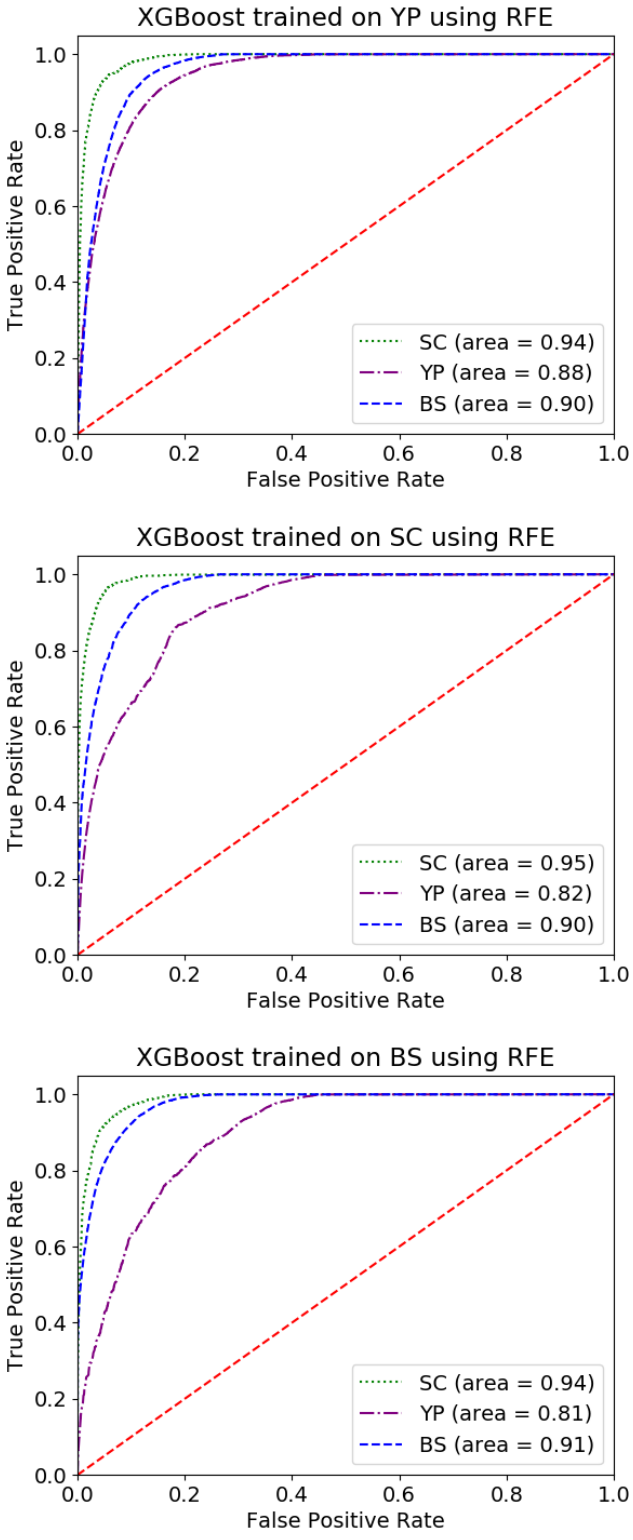


Figure 6.15 ROC curve for XGBoost classification trained on sample unbalanced non-normalized 3-AAU datasets and tested on full non-normalized datasets using RFE analysis

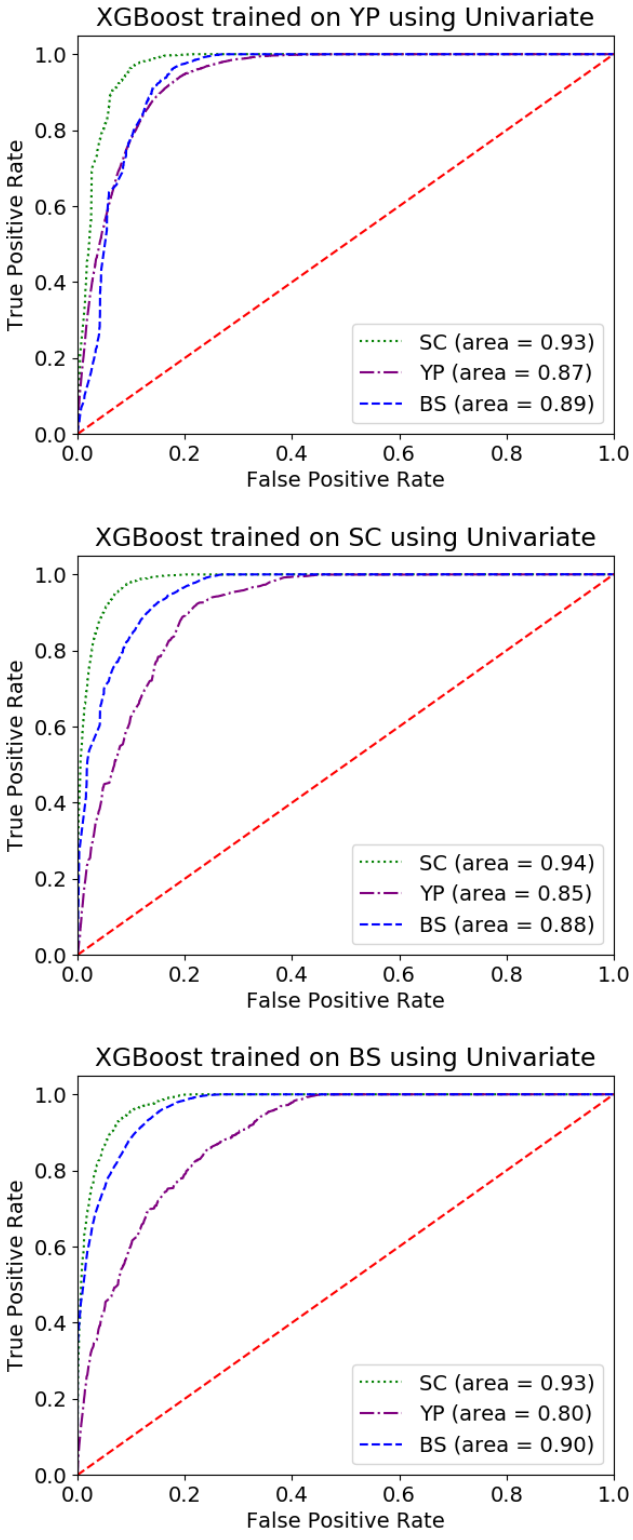


Figure 6.16 ROC curve for XGBoost classification trained on sample unbalanced non-normalized 3-AAU datasets and tested on full non-normalized datasets using Univariate analysis

Chapter 7

Conclusion

From the experiments we have run, we can say that we have achieved comparable or some times better results than Web Robertson et al. [15] with only 6 features. For the SVM classifier written in C++ with 7 features from Ahmad Alqurri [11], we have achieved better sensitivity (94%) and almost similar specificity (80%) than Ahmad Alqurri [11]. Our SVM classifier written in C++ is at least 7 times faster than MATLAB version.

We have achieved similar or better results using faster machine learning algorithms like Logistic Regression, Random Forest and XGBoost. We have achieved better sensitivity and specificity scores than Ahmad Alqurri [11] with only 6 features (from XGBoost feature importance analysis) using XGBoost classifier. The XGBoost classifier written in Python is almost 10 times faster than SVM classifier written in C++. The Random Forest classifier written in Python is almost 25 times faster than SVM classifier written in C++.

The Ordered Amino Acid Usage (AAU) feature is the most significant feature overall that helps in improving the results for all the classifiers. In other words, AAU feature helps in improving the prediction of proteotypic peptides with different machine learning techniques.

Bibliography

[1] Christopher M. Bishop. *Neural Networks for Pattern Recognition*. New York, NY, USA: Oxford University Press, Inc., 1995. ISBN: 0198538642.

[2] Chih-Chung Chang and Chih-Jen Lin. "LIBSVM: a library for support vector machines". In: *ACM transactions on intelligent systems and technology (TIST)* 2.3 (2011), p. 27.

[3] Tianqi Chen and Carlos Guestrin. "XGBoost: A Scalable Tree Boosting System". In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '16. San Francisco, California, USA: ACM, 2016, pp. 785–794. ISBN: 978-1-4503-4232-2. DOI: 10.1145/2939672.2939785. URL: http://doi.acm.org/10.1145/2939672.2939785.

[4] Robertson Craig, John P. Cortens, and Ronald C. Beavis. "Open Source System for Analyzing, Validating, and Storing Protein Identification Data". In: *Journal of Proteome Research* 3.6 (2004). PMID: 15595733, pp. 1234–1242. DOI: 10.1021/pr049882h. eprint: https://doi.org/10.1021/pr049882h. URL: https://doi.org/10.1021/pr049882h.

[5] Frank Desiere et al. "The PeptideAtlas project". In: *Nucleic Acids Researc* 34 (2006), pp. D655–D658. DOI: 10.1093/nar/gkj040. eprint: /oup/backfile/content_public/journal/nar/34/suppl_1/10.1093/nar/gkj040/2/gkj040.pdf. URL: http://dx.doi.org/10.1093/nar/gkj0401.

[6] Karl Pearson F.R.S. "LIII. On lines and planes of closest fit to systems of points in space". In: *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* 2.11 (1901), pp. 559–572. DOI: 10.1080/14786440109462720. eprint: https://doi.org/10.1080/14786440109462720. URL: https://doi.org/10.1080/14786440109462720.

[7] Harold Hotelling. "Relations Between Two Sets of Variates". In: *Biometrika* 28.3/4 (1936), pp. 321–377. ISSN: 00063444. URL: http://www.jstor.org/stable/2333955.

[8] Philip Jones et al. "PRIDE: a public repository of protein and peptide identifications for the proteomics community". In: *Nucleic Acids Research* 34 (2006),

pp. D659–D663. DOI: 10.1093/nar/gkj138. eprint: /oup/backfile/content_public/journal/nar/34/suppl_1/10.1093/nar/gkj138/2/gkj138.pdf. URL: http://dx.doi.org/10.1093/nar/gkj138.

[9] *MATLAB Optimization Toolbox*. The MathWorks, Natick, MA, USA. 2017.

[10] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[11] Ahmed AL-Qurri. "Improving Peptide Identification by Considering Ordered Amino Acid Usage". In: (2017).

[12] Kiebel Gary R. et al. "PRISM: A data management system for high-throughput proteomics". In: *PROTEOMICS* 6.6 (), pp. 1783–1790. DOI: 10.1002/pmic.200500500. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/pmic.200500500. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/pmic.200500500.

[13] Craig Robertson, Cortens John P., and Beavis Ronald C. "The use of proteotypic peptide libraries for protein identification". In: *Rapid Communications in Mass Spectrometry* 19.13 (), pp. 1844–1850. DOI: 10.1002/rcm.1992. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/rcm.1992. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/rcm.1992.

[14] Ngoc Hieu Tran et al. "De novo peptide sequencing by deep learning". In: *Proceedings of the National Academy of Sciences* 114.31 (2017), pp. 8247–8252. ISSN: 0027-8424. DOI: 10.1073/pnas.1705691114. eprint: http://www.pnas.org/content/114/31/8247.full.pdf. URL: http://www.pnas.org/content/114/31/8247.

[15] Bobbie-Jo M. Webb-Robertson et al. "A support vector machine model for the prediction of proteotypic peptides for accurate mass and time proteomics". In: *Bioinformatics* 26.13 (2010), pp. 1677–1683. DOI: 10.1093/bioinformatics/btq251. eprint: /oup/backfile/content_public/journal/bioinformatics/26/13/10.1093/bioinformatics/btq251/2/btq251.pdf. URL: http://dx.doi.org/10.1093/bioinformatics/btq251.