PREDICTING HUMAN AND ANIMAL PROTEIN SUBCELLULAR LOCATION

by

Sepideh Khavari

Submitted in Partial Fulfillment of the Requirements
for the Degree of

MASTER OF SCIENCE

in the
Department of Mathematics and Statistics

YOUNGSTOWN STATE UNIVERSITY

August, 2016

Predicting Human and Animal Protein Subcellular Location

Sepideh Khavari

I hereby release this thesis to the public. I understand that this thesis will be made available from the OhioLINK ETD Center and the Maag Library Circulation Desk for public access. I also authorize the University or other individuals to make copies of this thesis as needed for scholarly research.

Signature: _____

Sepideh Khavari, Student Date

Approvals:

_____

Dr. G. Andy Chang, Thesis Advisor Date

_____

Dr. G. Jay Kerns, Committee Member Date

_____

Dr. Xiangjia Jack Min, Committee Member Date

_____

Dr. Salvatore A. Sanders, Dean of Graduate Studies Date

ABSTRACT

An important objective in cell biology is to determine the subcellular location of different proteins and their functions in the cell. Identifying the subcellular location of proteins can be accomplished either by using biochemical experiments or by developing computational predictors that aid in predicting the subcellular location of proteins. Since the former method is both time-consuming and expensive, the computational predictors provide a more advantageous and efficient method of solving the problem. Computational predictors are also ideal in solving the problem of predicting protein subcellular locations since the number of newly discovered proteins have been increasing tremendously as a result of the genome sequencing project.

The main objective of this study is to use several different classifiers to predict the subcellular location of animal and human proteins and to determine which of these classifiers performs the best in predicting protein subcellular location. The data for this study was obtained from The Universal Protein Resource (UniProt) which is a database of protein sequence and annotation. Therefore, by accessing UniProt Knowledgebase (UniProt KB), the human and animal proteins that were manually reviewed and annotated (Swiss-Prot) were chosen for this study.

A reliable benchmark dataset is obtained by following and applying criteria established in earlier studies for predicting protein subcellular locations. After applying the above criteria to the original dataset, the working benchmark dataset includes 2944 protein sequences. The subcellular locations of these proteins are the nucleus (1001 proteins), the cytoplasm (540 proteins), the secreted (436 proteins), the mitochondria (328 proteins), the cell membrane (286 proteins), the endoplasmic reticulum (207 proteins), the Golgi apparatus (86 proteins), the peroxisome (30 proteins), and the lysosome (30 proteins). Therefore, there are 9 different subcellular locations for proteins in this dataset.

The method used for representing proteins in the study is the pseudo-amino acid composition (PseAA composition) adapted from earlier studies. The predictors used to predict the subcellular location of proteins in animal and human include Random Forest, Adaptive Boosting (AdaBoost), and Stage-wise Additive Modeling using a Multi-class Exponential loss function (SAMME), Support Vector Machines (SVMs), and Artificial Neural Networks (ANNs).

The results from this study establish that the SVMs classifier yielded the best overall accuracy for predicting the subcellular location of proteins. Most of the computational classifiers used in this study produced better prediction results for determining the subcellular location of proteins in the nucleus, the secreted, and the cell membrane. The secreted and the cell membrane locations

had high specificity values with all of the classifiers used in this study. The nucleus had the best prediction results, including a high sensitivity and a high MCC value by using the Bagging method.

ACKNOWLEDGEMENT

I am truly grateful to all who have helped me tremendously with this research project. Special thanks are due to Dr. Chang for his great advice on choosing this topic for my Master's Thesis, providing me with ample opportunities to spend time on this project and explore different aspects of this research, and for his everlasting patience while I worked on this paper. I appreciated all of his help and patience greatly. To Dr. Min for providing advice on this research, helping with the data, and answering all of my questions regarding the biology aspects so generously. To Dr. Jay Kerns, for teaching statistics with intuition behind every concept. I would have never developed an interest in this field had it not been for his brilliant intuitive approach to teaching statistics. To Dr. Lucy Kerns, for answering all of my questions and helping me apply statistical theory to applied problems. I have learned so much in her classes. To James Munyon who helped me immensely with R coding for this project and discussed the different aspects of this project. I learned so much about the project and R coding by working alongside James. To the Mathematics and Statistics Department for creating a wonderful and nourishing environment for learning and scholarship. Last, but not the least, to my amazing family for their continued support, encouragement, and understanding. They understood why I had to close the door and work in quiet so many times without interruption. I am grateful to them for everything.

No man is an island entire of itself;
every man is a piece of the continent,
a part of the main.

John Donne

1624

Contents

1 Introduction

A cell is the smallest unit of any life form. The size of a cell can vary from $1 - 5\mu$m, in prokaryotic cells, to a size of greater than 50μm, in eukaryotic cells [32]. Cells perform many diverse and important functions necessary for the survival of all living beings. There are many smaller different components and organelles inside a cell that are responsible for carrying out these diverse and important functions. The organelles inside a cell have specific functions and nearly all of them are surrounded by membranes that have embedded proteins.

A cell's plasma membrane encloses the cytoplasm and acts as a boundary between the inside and outside of the cell. The plasma membrane also controls the passage of molecules and ions in and out of the cell. Various types of proteins such as channel proteins, transport proteins, and receptor proteins are embedded in the plasma membrane and carry out important functions. Eukaryotic cells have a membrane-bounded nucleus in which the genetic material is stored. Ribosomes are organelles that use information sent from the nucleus to build proteins. The Endoplasmic Reticulum (ER) is a system of complex channels and flattened vesicles. Rough ER synthesizes and modifies polypeptides that are used in building protein molecules. It also creates vesicles to transport proteins to other locations in the cell. Smooth ER synthesizes lipids and forms vesicles that carry the molecules to other locations in the cell. The Golgi Apparatus is made up of a stack of slightly curved and flattened vesicles. The Golgi Apparatus receives transport-vesicles from the Rough and Smooth ER. Then, it modifies the molecules within these vesicles. Finally, it sorts and packages the modified molecules into new vesicles for further transport to specific locations. Lysosomes are vesicles that digest molecules and are produced by the Golgi apparatus. Vacuoles are similar to vesicles, but larger in size, and store nutrients and ions. Mitochondria are organelles that are responsible for energy conversion. They break down carbohydrates to produce adenosine triphosphate (ATP) molecules. ATP is the carrier of energy in the cell. Cells use ATP to synthesize and transport molecules and perform specific functions [32].

The functions that are performed by these organelles are vital to the cell's survival. These functions are carried out by the proteins embedded in the organelles or other compartments in the cell. On average, a cell contains 10^9 proteins located in different parts of the cell. Figure 1 illustrates the different compartments and organelles in the cell [21].

The location of proteins in the cell is referred to as the "subcellular location" in the literature. An important objective in cell biology and proteomics is to determine the subcellular location of different proteins and their functions in the cell. The knowledge gained about the location of proteins

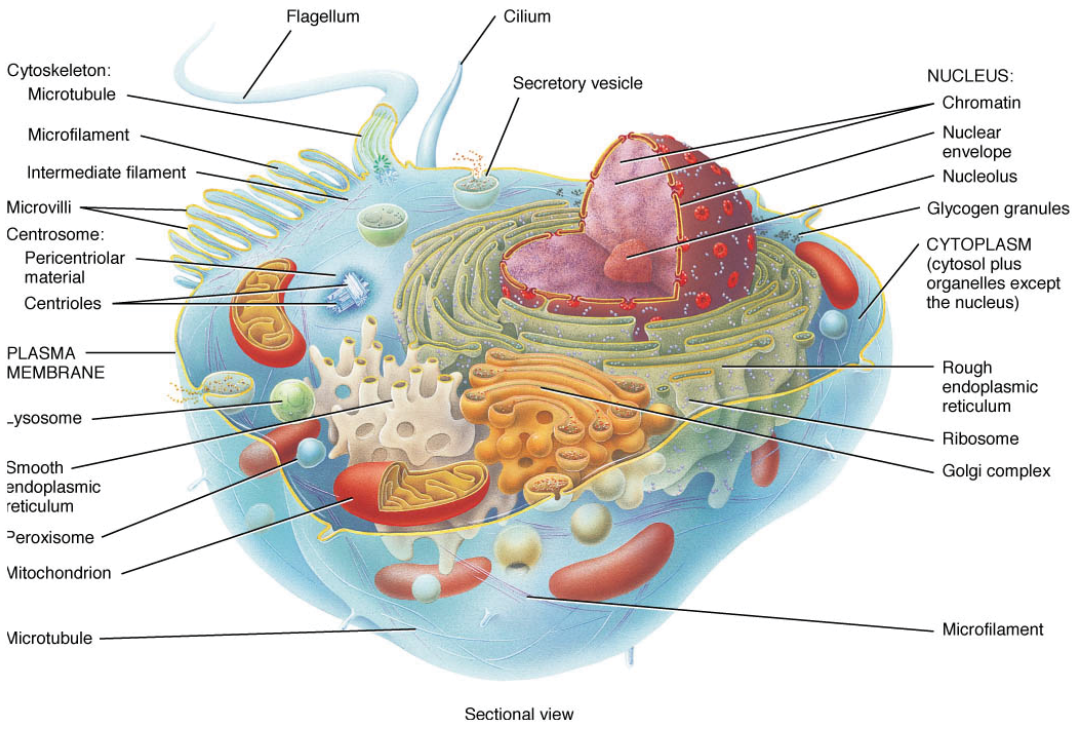

Figure 1: The sectional view of a cell, its organelles and compartments
Source: http://higheredbcs.wiley.com [31]

also can help in determining the specific functions they carry out for the cell's survival [21].

Identifying the subcellular location of proteins can be accomplished by using two methods. The first method is using biochemical experiments in the lab to locate proteins in the cell. However, the drawback to this approach is that it can be both very time-consuming and expensive. Given that the number of newly discovered proteins is increasing so quickly, as a result of genome sequencing project, and that the identification of their subcellular location lags behind, this approach is not a very efficient approach to solving the protein sublocalization problem.

The second method is developing computational predictors that help in predicting the subcellular location of proteins [21]. There are a variety of computational predictors, such as amino acid composition-based methods, which involve machine learning. Neural networks and support vector machines (SVMs) are two methods that utilize amino acid composition to predict the subcellular location of proteins. Another computational method of prediction utilizes different characteristics of proteins to predict their locations in the cell. K-nearest neighbor, SVMs, and Bayesian networks are some examples of this approach. The sequence homology-based method is also another prediction approach to finding the subcellular location of proteins [3]. Therefore, the main objective of this study is to use several different classifiers to predict the subcellular location of animal and

human proteins and to determine which of these classifiers performes the best in predicting protein subcellular location.

2 Background and Literature Review

In the post-genomic era, there has been a significant increase in the number of newly discovered proteins. Given the need to predict the subcellular location of these proteins, there have been many approaches in the literature to address protein sublocalization problem.

Zheng Yuan [39] uses the Markov Chains to predict protein subcellular location. The author points out that by using the Jack-knife test the prediction accuracy is 8% greater than using the neural network method and incorporating amino acid composition [32].

Chou and Shen [21] discuss their review of various methods of prediction available to identify protein subcelluar locations. Their paper examines how the problem of predicting protein sublocation can be viewed in terms of how proteins are represented and what type of algorithm can be used to produce the most accurate prediction. The authors point out that proteins can be represented in various ways such as sequential; non-sequential (amino acid composition and pseudo-amino acid composition [PseAA]); the Functional domain (FunD) discrete model, which is a representation by functional domain since function of a protein is related to its subcellular location; the Gene Ontology (GO) discrete model, which is a representation of protein that is defined in GO database– proteins in a GO database are clustered in such a way that is reflective of their subcellular location; and the hybridization discrete model which is a combination of GO discrete model and PseAA [21].

Next, the authors discuss the algorithms used to predict protein subcellular location and the testing methods incorporated to test the accuracy of each algorithm. The covariant discriminant (CD), the K nearest neighbor (KNN), the optimized evidence-theoretic K nearest neighbor (OET-KNN), and ensemble classifiers were extensively discussed as examples of prediction algorithms. For testing the quality of prediction algorithms, the authors investigated the self-consistency examination, the cross-validation examination, and the Jack-knife examination [21]. A review of protein sequence representation and prediction algorithms for this paper, which is based upon the paper by Chou et. al. [21], will be presented in the following section.

Finally, the authors survey the available prediction methods that have been placed on web servers and are available for free to the general public. The prediction web servers have been classified in terms of the type of eukaryotic organism for which they perform the prediction of protein subcellular location [21].

Cai et al.[38] uses Support Vector Machines (SVMs) to predict the subcellular location of proteins for 12 different locations in the cell. The authors conclude that both the self-consistency and the Jackknife tests have high prediction accuracies, ranging from 75% to 94%, for these locations. Park and Kanehisa [20] also have used SVMs on 12 subcellular locations in eukaryotic cells. They report that, by using the RBF kernel with SVMs, the total accuracy of prediction is 72.4% and the location accuracy is 54.6%. The authors also compare their results to the results obtained by Cai et al. [38]. Park and Kanehisa conclude that, with the use of 5-fold cross validation, their accuracy results are better than the results of the Jackknife test obtained by Cai et al. Another more recent study using SVMs is performed by Dehzangi et al. [1] to predict the subcellular location of proteins in Gram-positive and Gram-negative bacteria. The findings of this study show that, by using 10-fold cross validation, the overall accuracies for prediction of the subcellular locations of proteins in Gram-positive and Gram-negative bacteria are 87.7% and 79.6% respectively.

Reinhardt and Hubbard [29] use neural networks to classify the subcellular location of proteins in prokaryotic and eukaryotic cells. Their results reveal that the accuracy of classification in prokaryotic cells for three subcellular locations is 81%. The accuracy of classification for eukaryotic cells for predicting protein in four subcellular locations is 66%. Singh et al. [4] also use the neural networks classification method to predict plant protein subcellular locations. In their work, they used various methods to represent proteins including amino acid composition and dipeptide composition. For predicting the protein subcellular location, they solve the classification problem by performing multiple binary classification predictions. In the final step, they combine all of the binary classifiers to arrive at a final classifier. They conclude that, with the Pseudo Amino Acid Composition, the overall accuracy of classification using their neural network model is 75%. Furthermore, the overall accuracy of prediction they obtained is more efficacious than the performance of web tools such as YLOC+ and Euk-mPloc for protein subcellular prediction.

Chou and Shen [22] have introduced a web-server called Euk-mPLoc 2.0, which is a hybrid model of gene ontology (GO) information, functional domain information, and sequential evolutionary information, using different forms of pseudo-amino acid composition. The authors point out that this predictor is able to predict the subcellular location of proteins in 22 locations inside the cell. They note that Euk-mPLoc 2.0 is a very powerful predictor in predicting eukaryotic proteins that reside in multiple locations inside the cell [21].

Other papers have also evaluated the current computational methods of prediction that have been proposed to predict protein subcellular location and are available freely for public use. Min [25] discusses the prediction accuracy of various software tools in predicting the secretomes of Eukaryotes.

Prediction accuracy was reflected by using the Mathews' Correlation Coefficient in this paper. His research indicates that there is no one single software that can result in the highest accuracy in protein subcellular location prediction in all Eukaryotes. Min concludes that, among the tools tested, Phobius is best for animal protein prediction, SignalP is best for plant protein prediction, and WoLF PSORT is best for fungal protein prediction [25].

Sprenger et.al. [18] have evaluated prediction methods under the criteria that predictions tools accept large amount of protein sequence, are publicly available, and are also able to predict protein locations in at least nine subcellular locations. The prediction methods surveyed and evaluated included CELLO, MultiLoc, Proteome Analyst, pTARGET and WoLF PSORT. The sources of data used in this study are Swiss Prot and LOCATE. The authors calculate sensitivity and specificity data analyzed by each prediction method and then compare the results to the outcome represented by random chance. They concluded that the prediction methods did not show a level of sensitivity on either data set to prove to be a reliable prediction method for predicting subcellular location of new proteins. In addition, the authors pointed out that the prediction methods produced a lower accuracy for the data from LOCATE [18].

3 Methods

3.1 Protein Representation Method

In order to predict the subcellular location of proteins, it is important to employ a method for representing a protein. There are multiple approaches in the literature such as sequential and non-sequential methods, for representing the protein samples. The method used in this study to represent protein data is the pseudo-amino acid composition (PseAA composition) method, which has been adopted from [21]. PseAA composition is a non-sequential method of representation of proteins. It has an advantage over the sequential method of representation in that the sequential method doesn't perform well when a protein of interest has little homology to proteins of known location [21]. It also has an advantage over Amino Acid composition method (AA composition), because in the AA composition method, the order of sequence in the protein is lost [21].The following is a description provided by Chou and Shen on how a protein sequence is presented in PseAA composition methods.

PseAA composition [21]:

Consider a query protein sequence $\mathbf{P}$ that is comprised of L amino acid residues. In the AA composition discrete model this protein would be represented as

$$\mathbf{P} = \mathbf{R_1 R_2 R_3 R_4}, \ldots \mathbf{R_L} \tag{1}$$

In the above equation, $\mathbf{R_1}$ represents the first residue from protein $\mathbf{P}$, $\mathbf{R_2}$ represents the second residue and this pattern continues to $\mathbf{R_L}$ being the Lth residue.

The above protein representation can be expressed according to the Amino Acid (AA)

$$\mathbf{P} = [f_1, f_2, \ldots, f_{20}]^T \tag{2}$$

where $f_u (u = 1, \ldots, 20)$ are the normalized occurrences of the 20 native amino acids.
It is important to note that the disadvantage of the AA composition is that the order of amino acid representation in the original protein is lost. The alternative to the AA composition is the pseudo-amino acid (Pse AA) composition, which preserves the order of amino acids in the original protein.
In the Pse AA composition model the query protein sequence is represented by

$$\mathbf{P} = [p_1, p_2, \ldots, p_{20}, p_{20+1}, \ldots, p_{20+\lambda}]^T, (\lambda < L) \tag{3}$$

In this equation, the $\mathbf{20} + \lambda$ components are given by

$$p_u = \begin{cases} \frac{f_u}{\sum_{i=1}^{20} f_i + w \sum_{k=1}^{\lambda} \tau_k}, & (1 \leq u \leq 20) \\ \frac{w\tau_{u-20}}{\sum_{i=1}^{20} f_i + w \sum_{k=1}^{\lambda} \tau_k}, & (20+1 \leq u \leq 20+\lambda), \end{cases} \tag{4}$$

In the equation above w is described as the weight factor which has been set to equal 0.05 by Chou and Shen [21]. τ_k is the kth tier correlation factor that represents the sequence order correlation between all of the kth most adjacent residues. For a more detailed explanation of the above equation, refer to [21]. It is also important to note that the parameter λ is the number of components in protein's PseAA composition representation. The larger this parameter, the greater is the number of components in a protein's representation. However, the value of λ cannot exceed the number of amino acids in a protein [21].

Therefore, in this paper, the protein sequence $\mathbf{P}$ representation includes the occurrence frequen-

cies of the 20 amino acids, in addition to a predetermined $\lambda = 15$. The value of $\lambda = 15$ was chosen based upon earlier studies [28] and [26].

3.2 Basic Representation of the Problem

For a basic algorithm that helps to find the subcellular location of proteins, this study follows the same method as [21]. If a cell has T proteins, then the proteins can be represented as $\mathbf{P_1}, \mathbf{P_2}, \mathbf{P_3}, \ldots, \mathbf{P_T}$. Also, the cell has K subcellular locations which can be represented at $C = C_1 \cup C_2 \cup C_3, \cup \ldots \cup, C_K$. Note that each subset or subcellular location C_k, where $k = 1, 2, 3, \ldots, K$ is comprised of proteins with the same subcellular location, with the number of proteins in the location being T_k. Hence, we have $T = T_1 + T_2 + T_3 + \ldots + T_K$ proteins. For a query protein P, the objective is to find the subcellular location of P.

3.3 Dataset

The data for this study was obtained from The Universal Protein Resource (UniProt) which is a database of protein sequence and annotation [12]. The proteins of interest in this paper are animal and human proteins. Therefore, by accessing UniProt Knowledgebase (UniProt KB) [12], the human and animal proteins that were manually reviewed and annotated (Swiss-Prot) were chosen for this study.

In order to obtain a reliable benchmark dataset the following criteria were further applied to the obtained data, based upon the suggestions found in the literature for benchmark data set:

- All proteins in this dataset are reviewed and annotated for subcellular location.
- All proteins in the dataset are annotated only for one subcellular location(protein sequences that have multiple subcellular locations have been excluded from the dataset).
- Protein sequences with vague and uncertain labels such as “probable” or “by similarity” have been excluded.
- Proteins that contain unknown amino acids in their sequence and are marked by **X** have been excluded
- Proteins with the label ‘fragment’ are excluded from the dataset (‘fragments’ are those protein sequences that do not start with the letter **M**- referring to the amino acid methionine).

- Proteins in this dataset have at least 100 amino acids in their sequence. Any protein with less than 100 amino acids in their sequence have been excluded from this dataset.

After applying the above criteria to the original dataset, "50/50" BLASTClust [15] is used on the dataset. The purpose of using "50/50" BLASTClust is to cluster similar proteins together and to choose one proteins at random in each cluster for the final dataset. The rule for similarity of proteins is that if two or more proteins are at least 50% similar over at least 50% of their length, then they belong to the same cluster.

After applying "50/50" BLASTClust to the dataset, the working benchmark dataset includes 2949 protein sequences. The subcellular locations of these proteins are the nucleus (1002 proteins), the cytoplasm (540 proteins), the secreted (438 proteins), the mitochondria (329 proteins), the cell membrane (286 proteins), the endoplasmic reticulum (207 proteins), the Golgi apparatus (87 proteins), the peroxisome (30 proteins), and the lysosome (30 proteins). Therefore, there are 9 different subcellular locations for proteins in this dataset.

This study utilizes different computational predictors to predict the subcellular location of human and animal proteins. Any computational predictor is comprised of two components. These components are made up of the benchmark dataset and the prediction algorithm. The benchmark dataset is also composed of two parts: a training (learning) dataset and testing dataset [11].

Prior to partitioning, the benchmark dataset is scanned and 5 proteins are located as having their locations marked as 'NA'. These proteins are removed from the benchmark dataset. Therefore, the benchmark dataset is comprised of 2944 proteins. In the next step, by adopting the [28] and [26] method of partitioning the benchmark dataset, the benchmark dataset is partitioned into a 70% training (learning) dataset and a 30% independent testing dataset. The training dataset includes a total of 2060 proteins. The subcellular locations of these proteins are in the cell membrane (210 proteins), the cytoplasm (369), the endoplasmic reticulum (145 proteins), the Golgi apparatus (63 proteins), the lysosome (21 proteins), the mitochondria (236 proteins), the nucleus (702 proteins), the peroxisome (20 proteins), the secreted (294 proteins). Also, the testing dataset includes a total of 884 proteins. The subcellular locations of these proteins are in the cell membrane (76 proteins), the cytoplam (171), the endoplasmic reticulum (62 proteins), the Golgi apparatus (23 proteins), the lysosome (9 proteins), the mitochondria (92 proteins), the nucleus (299 proteins), the peroxisome (10 proteins), the secreted (142 proteins).

Prior to predicting the subcellular location of proteins in this study, various methods of classification are discussed. In the following sections, methods of Bootstrap Aggregating, Random Forest,

Adaptive Boosting, and Stagewise Additive Model using a Multi-class Exponential loss function, Support Vector Machines, and Artificial Neural Networks are briefly examined. The R software environment and other R packages are used to apply the classification methods to predict the protein subcellular location [33].

3.4 Classification Methods

Classification problems can be solved using a variety of approaches. A popular approach in classification is to construct an ensemble of decision trees, let each tree vote for a class, and finally take the most popular vote as the final class in the classification problem. An example of such an approach is Bootstrap Aggregating, which is also known as Bagging. This method was first introduced by Leo Breiman [5] and [6]. Another method introduced by [14] is when selecting the split at random from k best splits at each node. Amit and Geman [34] start with a large number of geometric features and use a random selection of these features to find the best split at each node. Breiman [7] offers another method in building new training sets by randomizing output in the original data set.

There are also other approaches that do not include a randomization element in solving the classification problems. An example of this type of approach, Adaptive Boosting (AdaBoost), was introduced by [36] which maintains a set of weights on the original data and adjusts these weights as each classifier is learned by the algorithm. Zhu et. al. [16] introduced a similar approach, called Stagewise Additive Model using a Multi-class Exponential loss function (SAMME), which expands the AdaBoost algorithm further.

The classifications methods used in this study include the following ensemble learning methods: Random Forest, AdaBoost, and SAMME. In addition to the ensemble methods, this study uses Support Vector Machines (SVMs) and Artificial Neural Networks (ANNs) to predict the subcellular location of proteins in human and animals. This paper will first provide a brief review of these and related classification methods, present their respective algorithms, and examine each method's advantages and disadvantages. Next, each method of classification is applied in predicting the subcellular location of proteins in this study. Finally, the results obtained from each classification method will be discussed.

3.4.1 Bootstrap Aggregating (Bagging)

Bootstrap Aggregating or Bagging is a machine learning method that uses an ensemble of predictors to solve classification problems. In "Bagging Predictors," Leo Breiman describes Bagging as a method of classification in which multiple versions of a predictor are generated and used to achieve an aggregate predictor. The multiple versions of a predictor are generated by making bootstrap replicates of the training set and using these replicates as new training set. The aggregate predictor averages over the multiple versions when it is predicting a numerical value and searches for a popular (majority) vote to predict a class [5]. Breiman points out that when Bagging is used on real data, it has high accuracy in solving classification problems.

In "out-of-bag estimation," Breiman explains how improvement in classification accuracy is achieved by using the bagging method. He points out that, when constructing bootstrap replicates of the training set, almost 37 percent of examples in the training set do not show up in a bootstrap training set. Therefore, since these examples are not used in the test, they can be used as accurate estimates. Breiman calls these unused examples "out-of-bag estimates." He points out that the out-of-bag estimates can be used to estimate the generalized error, strength, and correlation of classifiers. Breiman also adds that cross-validating bagged predictors requires a lot of computing power. However, out-of-bag estimates are built in the same run that the bagged predictors are being constructed. Therefore, out-of-bag estimates are as far as computing power more efficient [6].It is important to note that the idea of "Bagging" contributed to the development of the random forest classification method.

3.4.2 Random Forests

Random Forest is an ensemble of decision trees each voting for a class. The most popular vote among these trees is then chosen as the final class. Since Random Forest relies on constructing a collection of decision tree predictors and the classification approach of Bagging, a short review of the decision trees introduced by J.R. Quilan [19] is presented.

Decision Trees Quinlan's [19] paper examines a method of building decision trees called Top Down Induction of Decision Trees (TDIDT). To explain this concept, Quinlan considers a universe of objects where each object is described in terms of a set of characteristics or attributes. These attributes measure different and important features of each object. The measurements are discrete and mutually exclusive values [19] . Each object in the universe belongs to a set of mutually exclusive classes. The induction task builds classification rules that classify each object in the universe, based

upon the value of its attributes. The classification rule can be modeled as a decision tree [19].

To determine the class of each object in the universe, one must start with a training set in which the class of each object is known. In the decision tree structure, the nodes of the tree are the attribute-based test with branches that represents possible outcomes. The leaves of the tree represent class names. The procedure of classification starts at the root of the tree by evaluating the test and following the correct branch. It is important to point out that if the attributes are adequate, one can always construct a decision tree that correctly classifies each object in the training set[19].

According to Quinlan, the goal of classifying objects using the training set is to move beyond the training set and classify objects whose class is not known yet. The best decision trees are those with the most simple structure that classify objects correctly. These simple decision trees can discover the underlying relationship between the class of each object and its attribute, even outside the training set, and classify these objects correctly [19].

The 2001 Paper "Random Forest" by Leo Breiman first introduced Random Forest as an ensemble of predictors (decision trees) where the most popular vote among the trees determines the class of an object [8]. Breiman defines Random Forest as "a classifier consisting of a collection of tree-structured classifiers $h(x, \Theta_k), k = 1, ...$ where the Θ_k are independent identically distributed random vectors and each tree casts a unit vote for the most popular class at input x" [8]. Breiman further explains the procedure of Random Forest by noting that "for the kth tree, a random vector Θ_k is generated independent of past vectors $\theta_1, \theta_2, \ldots, \theta_{k-1}$ but with the same distribution. A tree is grown using the training set and the random vector Θ_k which results in a classifier $h(x, \Theta_k)$" [8]. By following this procedure, one can construct a large number of trees. These trees will each vote for a class and the most popular class is chosen as the final class [8].

Breiman proves that, by using the Strong Law of Large Numbers, the generalization error will converge. Therefore, Random Forest does not have the disadvantage of overfitting (when a classification model overfits, it can classify the data in the training set well, but it is not capable of correctly classifying the data in the testing set. The problem of overfitting occurs as a result of the algorithm of the model not truly learning the relationship between the attributes and the class of objects, but only memorizing the predictions in the training set).

Breiman also points out that, although some prior Random Forest models have lower generalization error, Adaboost (which is covered in the next section) has the lowest generalization error. However, he further explains that the accuracy of Random Forest models in his study are improved and are competitive with the accuracy of Adaboost model by having "randomly selected inputs or combinations of inputs at each node to grow each tree" [8]. According to Breiman, the Random

Forest has an accuracy competitive with and sometimes better than Adaboost, its accuracy is not affected by outliers and noise in the data, the algorithms run faster than bagging or boosting, and they generate out-of-bag estimates which include the internal estimate of error, strength, and correlation [8].

3.4.3 Adaptive Boosting (Adaboost)

Yoav Freund and Robert E. Schapire introduce a boosting algorithm called "Adaboost" that improves the accuracy of classifiers immensely [36]. In other words, Adaboost combines many weak classifiers to achieve a strong and very accurate classifier as the final classifier. Adaboost works by starting with a base algorithm and the training data set. Equal weights are assigned to each data point in the data set, and then the algorithm is run which results in a new classifier. In the following step, for all of the data points that were classified incorrectly, the weight is increased and for all of the data points that were correctly classified, the weight is decreased, then the algorithm is run again giving another classifier. This process is repeated until all of the data points in the training data set are classified with 100 percent accuracy. Finally, all of the classifiers in the process are combined as a linear combination to produce a final classifier with a very high accuracy in classification. It is important to note that the Adaboost method is a great for solving two-class problems. If the problem of classification is extended to include multiple-class classification, the problem will be reduced to solving two-class problems using Adaboost.

The following is the algorithm for Adaboost according to Freund and Schapire's paper:

Adaboost Algorithm:

1. Initialize the observation weights $w_i = 1/n, i = 1, 2, ..., n$.
2. For $m = 1$ to M:
   - Fit a classifier $T^{(m)}(x)$ to the training data using weights w_i,
   - Compute $err^{(m)} = \sum_{i=1}^{n} w_i I(c_i \neq T^{(m)}(x_i)) / \sum_{i=1}^{n} w_i$,
   - Compute $\alpha^{(m)} = \log \frac{1-err^{(m)}}{err^{(m)}}$,
   - Set $w_i \longleftarrow w_i \exp\left(\alpha^{(m)}.I(c_i \neq T^{(m)}(x_i))\right)$, $i = 1, 2, ..., n$,
   - Re-normalize w_i.
3. Output $C(x) = \underset{k}{\text{argmax}} \sum_{m=1}^{M} \alpha^{(m)} I(T^{(m)}(x) = k)$.

3.4.4 Stagewise Additive Model using a Multi-class Exponential loss function (SAMME)

Adaboost is a great classification ensemble for two-class classification problems. However, when dealing with multi-class problems, Adaboost approaches solving the classification by breaking down the multi-class problem into multiple two-class classification problems. The Stagewise Additive Model using a Multi-class Exponential loss function or in short SAMME, improves on Adaboost by solving the multi-class classification problems without condensing them into multiple two-class problems. SAMME combines weak classifiers and requires the performance of each classifier to be better than random guessing [16].

The algorithm for SAMME is presented below [16]:

SAMME Algorithm:

1. Initialize the observation weights $w_i = 1/n, i = 1, 2, ..., n$.

2. For $m = 1$ to M:

   - Fit a classifier $T^{(m)}(x)$ to the training data using weights w_i,
   - Compute $err^{(m)} = \sum_{i=1}^{n} w_i I(c_i \neq T^{(m)}(x_i))/\sum_{i=1}^{n} w_i$,
   - Compute $\alpha^{(m)} = \log \frac{1-err^{(m)}}{err^{(m)}} + \log(K-1)$,
   - Set $w_i \longleftarrow w_i \exp\left(\alpha^{(m)}.I(c_i \neq T^{(m)}(x_i))\right), i = 1, 2, ..., n$,
   - Re-normalize w_i.

3. Output $C(x) = \underset{k}{\operatorname{argmax}} \sum_{m=1}^{M} \alpha^{(m)} I(T^{(m)}(x) = k)$,

The algorithm for SAMME is very similar to the algorithm for Adaboost except that in SAMME's algorithm in part c, $\log(K-1)$ is added to the equation. Note that if $K = 2$, then the algorithm for SAMME is the same as the algorithm of Adaboost. Also, the advantage of this algorithm is that, when adding weight to data points that were not classified correctly, it adds more weight to misclassified data points that Adaboost would. In addition, SAMME algorithm combines weak classifiers somewhat differently than Adaboost algorithm [16].

3.4.5 Support Vector Machines (SVMs)

Support Vector Machines (SVMs) classify data into groups by finding the optimal hyperplane that maximizes the margin of separation between classes. Figure 2 illustrates a visual representation of the optimal hyperplane that accomplishes the noted goal in the SVMs algorithm. Although SVMs

are binary computational predictors, the binary predictors can be combined to achieve a final multi-class classifier [10].

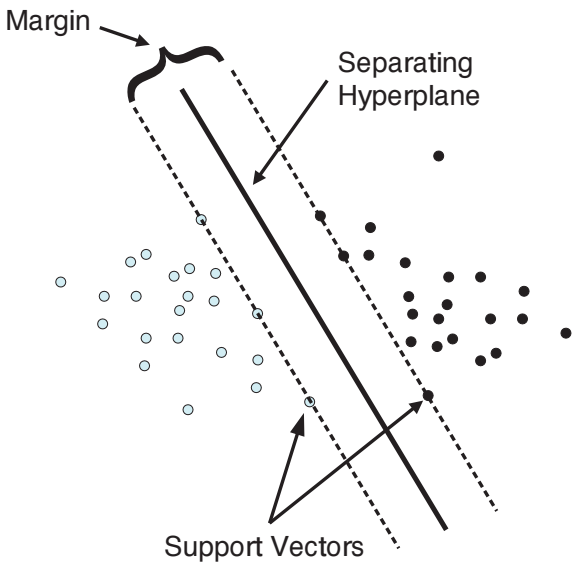

Figure 2: Classification using SVM
Source: Support Vector Machines by David Meyer page 2

In a classification problem, the dataset can be partitioned into training and testing datasets. Each data point in the training set has specific features, referred to as "attributes", and belongs to a specific class, referred to as “target value”. The goal of SVMs is to teach the data in the training set so that the SVM algorithm will be able to predict the correct “target value” for each data point in the testing set [9].

Hsu et. al [9] explain the SVMs approach in classification problems in a very concise and clear manner. In order to find an optimal hyperplane that maximizes the margin of separation between classes, the process starts with the training set. The data points in the training set are represented by $(\mathbf{x}_i, y_i), i = 1, ..., l$ with $\mathbf{x}_i \in \mathbb{R}^n$ and $\bar{y} \in \{-1, 1\}$. Next, SVM solves the following optimization problem

$$\min_{\mathbf{w},b,\xi} \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{i=1}^{l}\xi_i, \tag{5}$$

$$\text{subject to } y_i(\mathbf{w}^T\phi(\mathbf{x}_i) + b) \geq 1 - \xi_i, \text{ where } \quad \xi_i \geq 0. \tag{6}$$

The function ϕ maps vectors $\bar{x}_i$ into higher dimensional spaces. The goal of SVMs is to find a separating hyperplane, with maximal margin in the higher dimension. The "C" in the above optimization problem represents the penalty parameter of the error term. Also, $K(\bar{x}_i, \bar{x}_j) = \phi(\bar{x}_i)\phi(\bar{x}_j)$ is called the "kernel function" [9].

The basic kernel types are linear, polynomial, radial basis function (RBF), and sigmoid:

- Linear: $K(\bar{x}_i, \bar{x}_j) = \bar{x}_i^T \bar{x}_j$
- Polynomial: $K(\bar{x}_i, \bar{x}_j) = (\gamma \bar{x}_i^T \phi \bar{x}_j + r)^d, \quad \gamma > 0$
- Radial basis function (RBF): $K(\bar{x}_i, \bar{x}_j) = \exp(-\gamma \|\bar{x}_i - \bar{x}_j\|^2), \quad \gamma > 0$
- Sigmoid: $K(\bar{x}_i, \bar{x}_j) = \tanh(\gamma \bar{x}_i^T \bar{x}_j + r)$

Note that $\gamma, r,$ and d are kernel parameters.

The SVMs algorithm requires that all of the data values be represented as vectors of real numbers. In addition, the SVMs algorithm performs optimally when data points are scaled. There are two advantages in scaling the dataset before running the SVM algorithm. First, when scaling is performed, the attributes with higher numeric values do not overpower the attributes with lower numeric values. Next, in the SVM algorithm, kernel values are computed by the inner product of the feature vectors, resulting in situations in which large attribute values may cause computational and numeric difficulties. Scaling the data attributes to ranges $[-1, 1]$ or $[0, 1]$ have been suggested as a good choice for scaling. It is important to note that both the training data and the testing data must be scaled using the same scaling range [9].

As far as the model selection for SVMs used to classify the protein subcellular locations in this paper, the RBF kernel was chosen as the kernel function in accordance with the reasoning provided by [9]. The reasons underlying this choice of kernel include, but are not limited to:

- RBF uses non-linear mapping to map each data point to a higher dimension. Therefore, RBF can perform admirably even if non-linear relationships exist between attributes and classes.
- The number of hyper parameters, which are very important in designing the overall model complexity, are fewer with the RBF kernel than those with the use of the polynomial kernel.
- There are fewer numerical difficulties using the RBF kernel compared to the other kernels.

For choosing the best parameter values for C and γ for the RBF kernel, the method of k-fold cross validation is recommended by [9]. In k-fold cross validation, the entire dataset is divided into k equal subsets. Next, $k-1$ subsets are used as the training dataset to train the algorithm. One subset is used as a testing dataset, which tests the trained classifier. Therefore, each data point is predicted by the classifier once. The accuracy of the entire dataset is calculated by the percentage of correctly classified data points. This approach helps to alleviate the overfitting problem in which a classifier learns the relationship between features and classes in the training data and performs with high accuracy. However, since the classifier has only memorized the relationship patterns, it fails to perform well on the testing dataset [9].

Hsu et al. [9] recommends using a "grid search," in combination with cross-validation, to find the best parameter values for C and γ. The benefits of grid search include taking advantage of an exhaustive method to examine a wide range of parameter values, in order to find the optimal one for the algorithm. In addition, although there are more advanced methods for finding these parameter values, the time and computing power required to do the grid search is very close to those of the advanced methods.

3.4.6 Artificial Neural Networks (ANNs)

A branch of artificial intelligence, artificial neural networks (ANNs) or neural networks, are inspired by the biological brain and the nervous system [30]. A neural network is comprised of a set of artificial neurons that are inter-connected in a manner similar to the neural connections in the human brain. ANNs are used for classification problems in areas such as bankruptcy detection, speech recognition, product inspection, and fault detection [30]. ANNs can find patterns that are hidden deep within a dataset. Therefore, an ANN learns by examining many examples and discovering the underlying patterns and relationships (similarities and differences) between classes in a dataset.

ANNs were first formulated by McCulloch and Pitts in 1943. In the 1960's Rosenblatt introduced the Perceptron Convergence Theorem, while Minsky and Papert worked on showing the limitation of a simple perceptron. In the 1980's, the interest in ANNs was renewed by Hopefield's research and findings in this area. In addition, the back propagation learning algorithm for multilayer perceptron by Webros reinvigorated the field furthermore reinvigorated interest in ANNs [2].

An artificial neural network can be also be described as a weighted directed graph. In such a scheme, the artificial neurons are the nodes of the graph and the connections between neurons' inputs and outputs are the directed edges (these directed edges also include weights) [2].

As far as the architecture of an ANN, a standard neural network is comprised of three layers.

An input layer, a hidden layer, and an output layer. All neurons in the network are identical in structure and include a sum and a function unit. When inputs are fed into the neural network, the network assigns weight for each input. Next, the weighted inputs are all summed up. The result of this step is used as an input for the transfer or activation function in the neuron. The output from the activation function is the output of the neuron. There are different types of activation functions that can be used in a neural network. The most common function is the sigmoid or logistic function, which has the following form

$$f(x) = \frac{1}{1 + e^{-x}} \tag{7}$$

The graph of the sigmoid function is represented in Figure 3. It is important to note that the sigmoid function takes the input values and produces output results that range between zero and one [23].

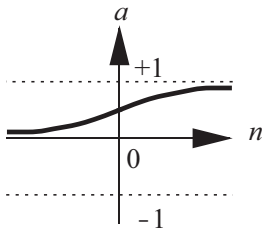

Figure 3: The log-sigmoid transfer function
Source: Neural Network Design by Hagan, Demuth, Beale, and De Jesus [23].

In Figure 4, a single input neuron is illustrated. The network is comprised of three layers: the

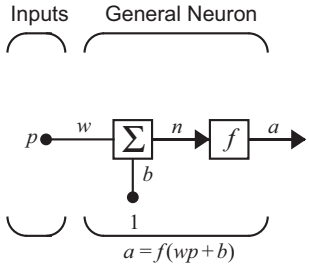

Figure 4: Single Input Neuron
Source: Neural Network Design by Hagan, Demuth, Beale, and De Jesus [23].

input layer, the hidden layer (one neuron), and the output layer. A single input p is fed to the neuron. The weight w is assigned to the input (p is multiplied by w). Next, pw and a bias quantity together form an input for the transfer function. The output of the transfer function f is the output of the neuron referred to as a.

In Figure 5, a single neuron is handling multiple inputs. Each input is assigned a different weight. The process follows the same steps with the inputs being multiplied by their respective assigned weights. The result is added to the bias value. Next, the result of this step is used as an input for the transfer function f. The output a of function f is the output of the neuron.

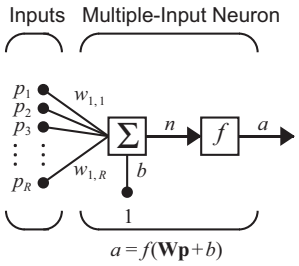

Figure 5: Multiple Input Neuron
Source: Neural Network Design by Hagan, Demuth, Beale, and De Jesus [23].

Finally, Figure 6, illustrates a more sophisticated neural network. There are multiple inputs and multiple neurons. The network still has three layers: input layer, hidden layer (neurons), and the output layer. The process works very similarly to the previous two examples.

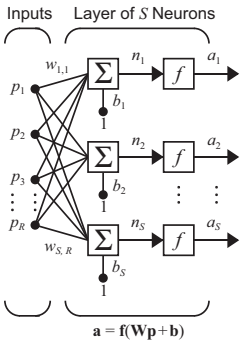

Figure 6: Multiple Input and multiple layer neural network
Source: Neural Network Design by Hagan, Demuth, Beale, and De Jesus [23].

The goal of training an ANN is to find the best weights for inputs so that the overall classification

accuracy can be maximized. Since all of the neurons are similar in structure, the difference in the assigned weight for each input is the key for the ANN to differentiate between different classes. A feature (input) can be assigned a very large weight feeding into one neuron and be assigned a very small weight feeding into another neuron, thereby changing its contribution to the summation value. If a feature has a large weight, it has a greater influence on the neuron. Similarly, with a smaller weight, a feature has a smaller influence on a neuron [23].

Once the ANN has classified all of the inputs from the training dataset (the class of each data point is known), the class predicted by the ANN is compared to the actual class in the training data set for each input. The difference between the predicted class and the actual class is used as an error signal to re-assign weights for the inputs. This method is know as the "back propagation method." A training iteration is completed when the ANN has been shown every data point in the dataset one time. However, training continues until the weights have reached a steady value or a maximum number of training iterations has been reached [2].

4 Results

4.1 Key Terms

In analyzing the results of this study, there are several important key terms that will be presented in this section and incorporated later when discussing the results of this study. Therefore, this section will first define these terms before proceeding to present the results.

In order to analyze the performance of each prediction or classification method, the term "accuracy" is employed. The "accuracy" of a prediction or classification method describes the number or the percentage of correctly classified object in a dataset. Furthermore, there are various methods for assessing performance of a predictor or classifier such as sensitivity, specificity, balanced accuracy, and Mathew's correlation coefficient (MCC). Before presenting these measurement methods, several terms that are incorporated in these measurement methods are defined below:

- TP (True Positives) = the number of true positives (proteins that are predicted to be in a subcellular location and are actually in that location)
- FN (False Negatives) = the number of false negatives (proteins that are not predicted to a subcellular location and are actually in that location)
- FP (False Positives) = the number of false positives (proteins that are predicted to be in a

subcellular location but they are not actually in that location)

- TN (True Negatives) = the number of true negatives (proteins that are not predicted to be in a subcellular location and they are not actually in that location)

The following are the formulas for sensitivity, specificity, balanced accuracy, and Mathew's correlation coefficient [25]. Both "balanced accuracy" and the "Mathew's correlation coefficient" are used in measuring the performance in classification in machine learning. "Balanced accuracy" is the arithmetic mean of sensitivity and specificity. While MCC considers true and false positives and negatives and can be used with classes that have different sizes. The MCC values range between -1 and 1 with the MCC value of 1 indicating perfect prediction [24].

$$\text{Sensitivity } \% = [TP/(TP + FN)] * 100$$

$$\text{Specificity } \% = [TN/(TN + FP)] * 100$$

$$\text{Balanced Accuracy } \% = [(\text{Sensitivity} + \text{Specificity})/2] * 100$$

$$\text{Mathew's Correlation Coefficient } \% = \frac{TP*TN-FP*FN}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}} * 100$$

4.2 Bagging model

Table 1: Overall Statistics

Accuracy	0.44
95% CI	$(0.40, 0.47)$
No Information Rate	0.34
P-Value [Acc > NIR]	8.1×10^{-10}
Kappa	0.22

Table 2: Statistics by Class

	Sen	Spe	PPV	NPV	Pre	DR	DP	BA	MCC	NoP
Cell membrane	0.39	0.94	0.41	0.94	0.09	0.04	0.09	0.66	0.39	76
Cytoplasm	0.1	0.97	0.46	0.82	0.19	0.02	0.04	0.54	0.1	171
Endoplasmic Reticulum	0.29	0.98	0.54	0.94	0.07	0.02	0.04	0.63	0.29	62
Golgi Apparatus	0	1	NA	0.97	0.03	0	0	0.5	0	23
Lysosome	0	1	NA	0.99	0.01	0	0	0.5	0	9
Mitochondria	0.19	0.96	0.35	0.91	0.11	0.02	0.06	0.57	0.19	92
Nucleus	0.93	0.35	0.42	0.91	0.34	0.31	0.74	0.64	0.93	299
Peroxisome	0	1	NA	0.99	0.01	0	0	0.5	0	10
Secreted	0.16	0.99	0.79	0.87	0.15	0.02	0.03	0.58	0.16	142

Key for Abbreviated Measures:

Sen: Sensitivity, Spe: Specificity, PPV: Positive Predicted Value, NPV: Negative Predicted Value, Pr: Prevalence, DR: Detection Rate, DP: Detection Prevalence, BA: Balanced Accuracy, MCC: Matthews Correlation Coefficient, NoP: Number of Proteins

The two tables above summarize the results from the bagging model. Table 1 shows that the model can predict proteins to their subcellular location with a 44% accuracy. The Table 2 illustrates statistics by class and indicates the performance of the model using the testing data. The locations with a sensitivity value of 0 and a specificity value of 1 indicate that there are no proteins predicted to these subcellular locations.

The subcellular location with the best prediction in the model is the nucleus with an MCC of 0.93, a sensitivity of 93% and a specificity of 35%. The sensitivity measure for the nucleus indicates that the model correctly predicted nucleus location 93% of the time. The specificity measure conveys the fact that the model correctly predicted the absence of proteins in the nucleus 35% of the time. The balanced accuracy value for this location is 64%. Although the bagging method predicts the nucleus proteins with high sensitivity, further work is needed to improve the value of specificity. In addition, other subcellular locations have not had good predictions with this method.

4.3 Bagging with Cross Validation model

Table 3: Overall Statistics

Accuracy	0.42
95% CI	$(0.40, 0.44)$
No Information Rate	0.34
P-Value [Acc > NIR]	2.2×10^{-16}
Kappa	0.22

Table 4: Statistics by Class

	Sen	Spe	PPV	NPV	Pre	DR	DP	BA	MCC	NoP
Cell membrane	0.3	0.95	0.38	0.93	0.1	0.03	0.08	0.62	0.27	76
Cytoplasm	0.08	0.95	0.27	0.82	0.18	0.01	0.05	0.52	0.05	171
Endoplasmic Reticulum	0.3	0.97	0.45	0.95	0.07	0.02	0.05	0.64	0.33	62
Golgi Apparatus	0	1	NA	0.97	0.03	0	0	0.5	0	23
Lysosome	0	1	NA	0.99	0.01	0	0	0.5	0	9
Mitochondria	0.15	0.98	0.44	0.9	0.11	0.02	0.04	0.56	0.21	92
Nucleus	0.89	0.38	0.43	0.87	0.34	0.3	0.71	0.64	0.28	299
Peroxisome	0	1	NA	0.99	0.01	0	0	0.5	0	10
Secreted	0.25	0.95	0.46	0.88	0.15	0.04	0.08	0.6	0.26	142

Key for Abbreviated Measures:

Sen: Sensitivity, Spe: Specificity, PPV: Positive Predicted Value, NPV: Negative Predicted Value, Pr: Prevalence, DR: Detection Rate, DP: Detection Prevalence, BA: Balanced Accuracy, MCC: Matthews Correlation Coefficient, NoP: Number of Proteins

Table 3 shows that the model can predict proteins to their subcellular location with a 42% accuracy. Table 4 illustrates statistics by class and indicates the performance of the model using the testing data. The locations with a sensitivity value of 0 and a specificity value of 1 indicate that there are no proteins predicted to these subcellular locations.

The ER location has the highest MCC value of 0.33, with a sensitivity of 30% and a specificity of 97%. Next, the nuclear subcellular location has an MCC value of 0.28. The sensitivity and

specificity values for the nucleus are 89% and 38%. The balanced accuracy value for both of these locations are 64%. Similar to the bagging methods, other subcellular locations failed to perform as well as they had performed in the other models.

4.4 Random Forests Model

The results for Random Forest model is summarized in the following tables. The package "randomForest" in R which is based upon the Random Forest algorithm proposed by Leo Brieman was utilized. The results are based upon partitioning the dataset into a 70% training dataset and a 30% testing dataset. This "random forest" is comprised of 500 trees. (Each tree is built by using a random selection of variables).

Table 5: Overall Statistics

Accuracy	0.51
95% CI	$(0.49, 0.56)$
No Information Rate	0.34
P-Value [Acc $>$ NIR]	2.2×10^{-6}
Kappa	0.36

Table 6: Statistics by Class

	Sen	Spe	PPV	NPV	Pre	DR	DP	BA	MCC	NoP
Cell membrane	0.49	0.95	0.49	0.95	0.09	0.04	0.09	0.72	0.44	76
Cytoplasm	0.27	0.92	0.45	0.84	0.19	0.05	0.12	0.6	0.24	171
Endoplasmic Reticulum	0.21	0.99	0.62	0.94	0.07	0.01	0.02	0.6	0.34	62
Golgi Apparatus	0	1	NA	0.97	0.03	0	0	0.5	0	23
Lysosome	0	1	NA	0.99	0.01	0	0	0.5	0	9
Mitochondria	0.33	0.95	0.42	0.92	0.1	0.03	0.08	0.64	0.3	92
Nucleus	0.89	0.58	0.52	0.91	0.34	0.3	0.58	0.73	0.45	299
Peroxisome	0	1	NA	0.99	0.01	0	0	0.5	0	10
Secreted	0.51	0.96	0.73	0.91	0.16	0.08	0.11	0.74	0.55	142

Key for Abbreviated Measures:

Sen: Sensitivity, Spe: Specificity, PPV: Positive Predicted Value, NPV: Negative Predicted Value, Pr: Prevalence, DR: Detection Rate, DP: Detection Prevalence, BA: Balanced Accuracy, MCC: Matthews Correlation Coefficient, NoP: Number of Proteins

The two tables above summarize the results from the Random Forest model. Table 5 shows that the model can predict proteins to their subcellular location with a 51% accuracy. Table 6 illustrates statistics by class and indicates the performance of the model using the testing data. The locations with a sensitivity value of 0 and a specificity value of 1 indicate that there are no proteins predicted to these subcellular locations.

The secreted location has the highest MCC value of 0.55, with a sensitivity of 51% and a specificity of 96%. This location has a balanced accuracy of 74%. Next, the nuclear subcellular location has an MCC value of 0.45. The sensitivity, specificity, and balanced accuracy values for the nucleus are 89%, 58%, and 73% . Finally, the cell membrane location has an MCC value of 0.44. The sensitivity and specificity values for the nucleus are 49% and 95%. The balanced accuracy value for this location is 72%.

4.5 ADA model

Table 7: Overall Statistics

Accuracy	0.49
95% CI	$(0.45, 0.52)$
No Information Rate	0.34
P-Value [Acc $>$ NIR]	2.2×10^{-6}
Kappa	0.31

Table 8: Statistics by Class

	Sen	Spe	PPV	NPV	Pre	DR	DP	BA	MCC	NoP
Cell membrane	0.53	0.94	0.47	0.95	0.09	0.05	0.1	0.73	0.44	76
Cytoplasm	0.25	0.9	0.38	0.83	0.19	0.05	0.13	0.58	0.18	171
Endoplasmic Reticulum	0.21	0.98	0.43	0.94	0.07	0.01	0.03	0.59	0.27	62
Golgi Apparatus	0	1	NA	0.97	0.03	0	0	0.5	0	23
Lysosome	0	1	NA	0.99	0.01	0	0	0.5	0	9
Mitochondria	0.3	0.94	0.39	0.92	0.1	0.03	0.08	0.62	0.28	92
Nucleus	0.84	0.54	0.48	0.87	0.34	0.28	0.59	0.69	0.36	299
Peroxisome	0	1	NA	0.99	0.01	0	0	0.5	0	10
Secreted	0.38	0.99	0.86	0.89	0.16	0.06	0.07	0.68	0.53	142

Key for Abbreviated Measures:

Sen: Sensitivity, Spe: Specificity, PPV: Positive Predicted Value, NPV: Negative Predicted Value, Pr: Prevalence, DR: Detection Rate, DP: Detection Prevalence, BA: Balanced Accuracy, MCC: Matthews Correlation Coefficient, NoP: Number of Proteins

The above tables summarize the results from the Adaptive Boosting (ADA) model. Table 7 indicates that the ADA model predicts the subcellular location of proteins with an accuracy of 49%. Therefore the overall accuracy of the ADA model in predicting protein subcellular location is slightly lower than Random Forest model.

Table 8 presents the statistics by class and indicates the performance of the model, using testing data. The secreted location has the highest MCC value of 0.53, with a sensitivity of 38%, and a specificity of 99%. This location has a balanced accuracy of 68%. Next, the cell membrane subcellular location has an MCC value of 0.44. The sensitivity, specificity, and balanced accuracy values for the nucleus are 53%, 94%, and 73%.

4.6 ADA model with Cross Validation

Table 9: Overall Statistics

Accuracy	0.42
95% CI	$(0.40, 0.44)$
No Information Rate	0.34
P-Value [Acc > NIR]	2.2×10^{-6}
Kappa	0.19

Table 10: Statistics by Class

	Sen	Spe	PPV	NPV	Pre	DR	DP	BA	MCC	NoP
Cell membrane	0.3	0.95	0.38	0.93	0.1	0.03	0.08	0.62	0.27	286
Cytoplasm	0.07	0.96	0.28	0.82	0.18	0.01	0.04	0.51	0.05	540
Endoplasmic Reticulum	0.29	0.98	0.48	0.95	0.07	0.02	0.04	0.63	0.34	207
Golgi Apparatus	0	1	NA	0.97	0.03	0	0	0.5	0	87
Lysosome	0	1	NA	0.99	0.01	0	0	0.5	0	30
Mitochondria	0.13	0.98	0.45	0.9	0.11	0.01	0.03	0.56	0.2	329
Nucleus	0.9	0.37	0.42	0.87	0.34	0.31	0.72	0.63	0.28	1002
Peroxisome	0	1	NA	0.99	0.01	0	0	0.5	0	30
Secreted	0.26	0.95	0.46	0.88	0.15	0.04	0.08	0.6	0.26	438

Key for Abbreviated Measures:

Sen: Sensitivity, Spe: Specificity, PPV: Positive Predicted Value, NPV: Negative Predicted Value, Pr: Prevalence, DR: Detection Rate, DP: Detection Prevalence, BA: Balanced Accuracy, MCC: Matthews Correlation Coefficient, NoP: Number of Proteins

The above tables summarize the results from the Adaptive Boosting (ADA) with cross validation model. Table 9 indicates that the ADA model predicts the subcellular location of proteins with an accuracy of 42%. Therefore the overall accuracy of the ADA with cross validation model in predicting protein subcellular location is lower than Random Forest model.

Table 10 presents the statistics by class and indicates the performance of the model, using testing data. The ER location has the highest MCC value of 0.34, with a sensitivity of 29%, and

a specificity of 98%. This location has a balanced accuracy of 63%. Next, the nuclear subcellular location has an MCC value of 0.28. The sensitivity, specificity, and balanced accuracy values for the nucleus are 90%, 37%, and 63%. Finally, the cell membrane location has an MCC value of 0.27. The sensitivity and specificity values for the nucleus are 30% and 95%. The balanced accuracy value for this location is 62%.

The cell membrane and secreted subcellular locations failed to perform as well as they had performed in the Random Forest and ADA models. The predictions for the nuclear location are also lower than the predictions for this location using the Bagging method.

4.7 SAMME Model

Table 11: Overall Statistics

Accuracy	0.51
95% CI	$(0.48, 0.59)$
No Information Rate	0.34
P-Value [Acc $>$ NIR]	2.2×10^{-6}
Kappa	0.37

Table 12: Statistics by Class

	Sen	Spe	PPV	NPV	Pre	DR	DP	BA	MCC	NoP
Cell membrane	0.53	0.95	0.48	0.96	0.09	0.05	0.09	0.74	0.45	76
Cytoplasm	0.41	0.84	0.39	0.86	0.19	0.08	0.2	0.63	0.25	171
Endoplasmic Reticulum	0.21	0.98	0.48	0.94	0.07	0.01	0.03	0.6	0.29	62
Golgi Apparatus	0	1	0	0.97	0.03	0	0	0.5	-0.01	23
Lysosome	0	1	NA	0.99	0.01	0	0	0.5	0	9
Mitochondria	0.36	0.93	0.38	0.93	0.1	0.04	0.1	0.65	0.3	92
Nucleus	0.77	0.71	0.57	0.86	0.34	0.26	0.45	0.74	0.45	299
Peroxisome	0	1	NA	0.99	0.01	0	0	0.5	0	10
Secreted	0.49	0.95	0.67	0.91	0.16	0.08	0.12	0.72	0.51	142

Key for Abbreviated Measures:

Sen: Sensitivity, Spe: Specificity, PPV: Positive Predicted Value, NPV: Negative Predicted Value, Pr: Prevalence, DR: Detection Rate, DP: Detection Prevalence, BA: Balanced Accuracy, MCC:

Matthews Correlation Coefficient, NoP: Number of Proteins

The two tables above summarize the results from the Stagewise Additive Model using a Multi-class Exponential loss function (SAMME) model. Table 11 indicates that the SAMME model predicts the subcellular location of proteins with an accuracy of 48%.

Table 12 indicates that the subcellular locations of the secreted, the cell membrane, and the nucleus have the highest MCC values. These results follow the same pattern of prediction from the previous models, where these three subcellular locations have had the highest values for MCC. The sensitivity values of these locations, in order, are 49%, 53%, 77%. The specificity value of these locations, in order, are 95%, 95%, 71%. The MCC values for these locations are 0.51, 0.45, and 0.45 respectively.

4.8 SVMs Model

Table 13: Overall Statistics

Accuracy	0.55
95% CI	$(0.53, 0.57)$
No Information Rate	0.34
P-Value [Acc $>$ NIR]	2.2×10^{-16}
Kappa	0.42

Table 14: Statistics by Class

	Sen	Spe	PPV	NPV	Pre	DR	DP	BA	MCC	NoP
Cell membrane	0.59	0.95	0.57	0.96	0.1	0.06	0.1	0.77	0.54	286
Cytoplasm	0.34	0.87	0.38	0.85	0.18	0.06	0.17	0.61	0.22	540
Endoplasmic Reticulum	0.35	0.98	0.54	0.95	0.07	0.02	0.05	0.66	0.4	207
Golgi Apparatus	0.03	1	0.5	0.97	0.03	0	0	0.52	0.13	86
Lysosome	0	1	NA	0.99	0.01	0	0	0.5	0	30
Mitochondria	0.48	0.94	0.52	0.94	0.11	0.05	0.1	0.71	0.44	328
Nucleus	0.76	0.76	0.62	0.86	0.34	0.26	0.42	0.76	0.5	1001
Peroxisome	0	1	NA	0.99	0.01	0	0	0.5	0	30
Secreted	0.63	0.92	0.57	0.93	0.15	0.09	0.16	0.77	0.52	436

Key for Abbreviated Measures:

Sen: Sensitivity, Spe: Specificity, PPV: Positive Predicted Value, NPV: Negative Predicted Value, Pr: Prevalence, DR: Detection Rate, DP: Detection Prevalence, BA: Balanced Accuracy, MCC: Matthews Correlation Coefficient, NoP: Number of Proteins

Table 13 shows that the model can predict the subcellular location of proteins with a 55% accuracy. The subcellular location with the best prediction results in the model was the cell membrane, with a sensitivity of 59% and a specificity of 95%. Other locations with good model performance, in the context of these two measurements, include the secreted, with a sensitivity of 63% and a specificity of 92% and the nucleus, with a sensitivity of 76% and a specificity of 76%.

The three subcellular locations of the cell membrane, the secreted, and the nucleus also have the three highest values of balanced accuracy and MCC. The cell membrane has a balanced accuracy of 77% and an MCC of 0.54 , the secreted has a balanced accuracy of 77% and an MCC of 0.52, and the nucleus which has a balanced accuracy of 76% and an MCC of 0.50.

4.9 ANN Model

Table 15: Overall Statistics

Accuracy	0.49
95% CI	$(0.46, 0.52)$
No Information Rate	0.34
P-Value [Acc $>$ NIR]	2.2×10^{-16}
Kappa	0.33

Table 16: Statistics by Class

	Sen	Spe	PPV	NPV	Pre	DR	DP	BA	MCC	NoP
Cell membrane	0.39	0.95	0.43	0.94	0.09	0.03	0.08	0.67	0.36	76
Cytoplasm	0.12	0.95	0.35	0.82	0.19	0.02	0.07	0.53	0.11	171
Endoplasmic Reticulum	0.21	0.98	0.43	0.94	0.07	0.01	0.03	0.59	0.27	62
Golgi Apparatus	0	1	NA	0.97	0.03	0	0	0.5	0	23
Lysosome	0	1	NA	0.99	0.01	0	0	0.5	0	9
Mitochondria	0.5	0.89	0.35	0.94	0.1	0.05	0.15	0.7	0.34	92
Nucleus	0.85	0.65	0.55	0.89	0.34	0.29	0.52	0.75	0.47	299
Peroxisome	0	1	NA	0.99	0.01	0	0	0.5	0	10
Secreted	0.49	0.91	0.52	0.9	0.16	0.08	0.15	0.7	0.42	142

Key for Abbreviated Measures:

Sen: Sensitivity, Spe: Specificity, PPV: Positive Predicted Value, NPV: Negative Predicted Value, Pr: Prevalence, DR: Detection Rate, DP: Detection Prevalence, BA: Balanced Accuracy, MCC: Matthews Correlation Coefficient, NoP: Number of Proteins

Table 15 shows that the model can predict the subcellular location of proteins with a 49% accuracy. The subcellular location with the best prediction in the model was the nucleus, with a sensitivity of 85% and a specificity of 65%. The next best result was seen with the secreted location also has a sensitivity of 49% and a specificity of 91%.

The subcellular locations of the nucleus and the secreted also have the two highest values of balanced accuracy and MCC. The nucleus has a balanced accuracy of 75% and an MCC of 0.47 while the secreted has a balanced accuracy of 70% and an MCC of 0.42.

5 Analysis

The main objective of this study was to use several different classifiers to predict the subcellular location of animal and human proteins and to determine which of these classifiers performed the best in predicting protein subcellular location. The working benchmark dataset includes 2944 protein sequences. The subcellular locations of these proteins are the nucleus (1001 proteins), the cytoplasm (540 proteins), the secreted (436 proteins), the mitochondria (328 proteins), the cell membrane (286 proteins), the endoplasmic reticulum (207 proteins), the Golgi apparatus(86 proteins), the

peroxisome (30 proteins), and the lysosome (30 proteins). Therefore, there are 9 different subcellular locations for proteins in this dataset.

The method used for representing proteins in the study is the pseudo-amino acid composition (PseAA composition), adapted from earlier studies. The computational predictors that are used to predict the subcellular location of proteins in animals and humans in this study include Bagging, Bagging with cross validation, Random Forest, the Adaptive Boosting (AdaBoost), AdaBoost with cross validation, the Stage-wise Additive Modeling using a Multi-class Exponential loss function (SAMME), Support Vector Machines (SVMs), and Artificial Neural Networks (ANNs).

The results from this study demonstrate that the SVM model has the best overall accuracy of 55% for predicting the subcellular location of proteins. However, since the dataset for this study is imbalanced (the number of proteins in each subcellular location varies greatly), the overall accuracy is not a good measure of performance of the computational predictors. Therefore, for the analysis of the prediction results, the measures of sensitivity, specificity, and MCC are chosen to gauge how well a predictor can predict the correct subcellular locations of proteins.

After analyzing the results, the predictions for the three subcellular locations of the nucleus, cell membrane, and secreted demonstrate the best outcomes out of the nine subcellular locations in this study. The Table 17 displays the performance of the various predictors in predicting the subcellular location of proteins for these three locations.

The nucleus proteins are best predicted by the Bagging method, with a sensitivity of 0.93, a specificity of 0.93, and an MCC of 0.93. The next best predictor for the nucleus proteins is the SVMs with, an MCC value of 0.50, which is the second highest MCC value. The sensitivity and specificity values are 0.76 and 0.76. respectively.

The cell membrane proteins are best predicted by the AdaBoost, with cross validation with a sensitivity of 0.30, a specificity of 0.95, and an MCC value of 0.62. The SVMs method has an outcome with a sensitivity of 0.59, a specificity of 0.95, and an MCC of 0.54. The SAMME predictor can predict cell membrane protein with a sensitivity of 0.53, a specificity of 0.95, and an MCC of 0.45. For cell membrane protein prediction, the predictors fail to produce higher sensitivity results compared to the results observed for the nucleus location. Therefore, the future work will include efforts to improve the sensitivity metric for the cell membrane protein location prediction. Finally, the MCC value for each of the mentioned predictors is low, as a result of the lower sensitivity values in each predictor.

The secreted proteins are best predicted by Random Forest, with a sensitivity of 0.51, a specificity of 0.96, and an MCC of 0.55. Next, the AdaBoost predictor shows a sensitivity of 0.38, a

Table 17: Protein Subcellular Location Prediction Results Using Various Predictors

Subcellular Locations	Predictors	Sensitivity	Specificity	Balanced Accuracy	MCC
Nucleus	Bagging	0.93	0.35	0.64	0.93
	Bagging with CV	0.89	0.38	0.64	0.28
	Random Forest	0.89	0.58	0.73	0.45
	AdaBoost	0.84	0.54	0.69	0.36
	AdaBoost with CV	0.90	0.37	0.63	0.28
	SAMME	0.77	0.71	0.74	0.45
	SVMs	0.76	0.76	0.76	0.50
	ANNs	0.85	0.65	0.75	0.47
Cell Membrane	Bagging	0.39	0.94	0.66	0.39
	Bagging with CV	0.30	0.95	0.62	0.27
	Random Forest	0.49	0.95	0.72	0.44
	AdaBoost	0.53	0.94	0.73	0.44
	AdaBoost with CV	0.30	0.95	0.27	0.62
	SAMME	0.53	0.95	0.74	0.45
	SVMs	0.59	0.95	0.77	0.54
	ANNs	0.39	0.95	0.67	0.36
Secreted	Bagging	0.16	0.99	0.58	0.16
	Bagging with CV	0.25	0.95	0.60	0.26
	Random Forest	0.51	0.96	0.74	0.55
	AdaBoost	0.38	0.99	0.68	0.53
	AdaBoost with CV	0.26	0.95	0.60	0.26
	SAMME	0.49	0.95	0.72	0.51
	SVMs	0.63	0.92	0.77	0.52
	ANNs	0.49	0.91	0.70	0.42

specificity of 0.99, and MCC of 0.53. The SVMs method has a sensitivity of 0.63, a specificity of 0.92, and an MCC of 0.52. Similar to the prediction of the cell membrane subcellular location, the sensitivity results from each predictor need to be further improved. The specificity results are optimal and high. The lower MCC level in each case mentioned above is as a result of lower sensitivity value from each predictor.

Therefore, in general, it can be concluded that of the methods used in this study to predict the location of the nucleus proteins, the Bagging method produces the best prediction results (with a sensitivity of 0.93, specificity of 0.93, and MCC of 0.93). For the prediction of cell membrane protein,

the AdaBoost with cross validation produces the best results with an MCC of 0.62, a sensitivity of 0.30, and a specificity of 0.95. Finally, to predict the location of secreted proteins, Random Forest has the best results with an MCC of 0.55, a sensitivity of 0.51, and a specificity of 0.96.

6 Comparison of the Prediction Results Using the R Software with the Results of Meinken et al. (2015) Using Web Server Predictors

The 2015 paper published by Meinken et al. [17] has used the available computational predictors available on web servers to predict the subcellular location of proteins in human and animals. The total number of proteins in the Meinken et al. study is $18,874$, with eleven subcellular locations. The computational tools used in the Meinken et al. study are the SignalP (version 3.0 and version 4.0), Phobius, WoLF PSORT, TargetP, TMHMM(version 2.0), and Scan-Prosite (PS-Scan) [17]. The algorithms used with these web-based computational predictors are the following: the K-Nearest Neighbor Classifier for WoLF PSORT, the Neural Network for TargetP, the Hidden Markov Model for Phobius, the Neural Network and the Hidden Markov Model for SignalP, and the Hierarchal Agglomerative Clustering and the Hidden Markov Model for ScanProsite.

In comparison, the dataset for this study has 2944 proteins and nine subcellular locations. The computational predictors were included in the packages written for the R software. Since, both of these studies focus on human and animal proteins, comparing the results produced by each study will highlight the efficacy of the R software and the packages provided for computational biology and/or classification purposes in this software compared to the computational tools available for free on web servers.

By comparing the results of Table 18 and Table 19, for the mitochondrial protein subcellular location, the best prediction pointed out by Meinken et al. [17] has a sensitivity of 0.42, a specificity of 0.98 and an MCC value of 0.53. The best predictor for the mitochondria location in this paper is SVMs, producing results with a sensitivity of 0.48, a specificity of 0.94 and an MCC value of 0.44. Therefore, the predictors used by Meinken et al. [17] definitely does a better job of predicting mitochondrial proteins.

Table 18: Prediction Accuracy from Meinken et al. Study [17]

Subcellular Locations	Sensitivity	Specificity	MCC
(a) Mitochondrial proteins			
TargetP	49.73	94.28	43.70
WoLF PSORT	49.20	97.17	52.80
TargetP AND WoLF PSORT	42.46	98.46	53.20
TaregetP OR WoLF PSORT	56.47	92.93	45.50
(b) Secreted proteins			
Secreted (4 out of 4)	87.77	97.90	87.60
Highly likely secreted (3 of 4)	93.47	96.03	88.90
Likely secreted (2 of 4)	94.57	93.96	86.60
Weakly likely secreted (1 of 4)	95.04	88.88	80.1
(C) ther subcellular locations			
Cytoplasm	55.56	93.35	46.40
Cytoskeleton	27.56	99.65	45.00
ER	32.91	98.97	41.90
Golgi	4.46	99.89	12.30
Lysosome	0.52	99.96	2.20
Nucleus	78.58	94.08	72.10
Peroxisome	3.33	99.46	3.00
Plasma membrance	82.7	95.8	77.90
Vacuole	0.0	100.0	-

Next, for the secreted subcellular location, the method employed by Meinken et al. [17] had a sensitivity of 0.88, a specificity of 0.98 and an MCC value of 0.88. Meanwhile, the best predictor in this research is the Random Forest method producing a sensitivity of 0.51, a specificity of 0.96 and an MCC value of 0.55. Therefore, the predictors used by Meinken et al. [17] outperform the “random forest" by large margins.

The cytoplasm location in Meinken et al. paper [17] reported prediction values with a sensitivity of 0.56, a specificity of 0.93 and an MCC value of 0.46. The SAMME predictor produces results with a sensitivity of 0.41, a specificity of 0.84 and an MCC value of 0.25. Therefore, these values of predictions are lower than the method employed by Meinken et al. [17] for this location as well.

Table 19: Protein Subcellular Location Prediction Results Using Various Predictors

Subcellular Locations	Predictors	Sensitivity	Specificity	MCC
Mitochondria	Bagging	0.19	0.96	0.19
	Bagging with CV	0.15	0.98	0.21
	Random Forest	0.33	0.95	0.30
	AdaBoost	0.30	0.94	0.28
	AdaBoost with CV	0.13	0.98	0.20
	SAMME	0.36	0.93	0.30
	SVMs	0.48	0.94	0.44
	ANNs	0.50	0.89	0.34
Secreted	Bagging	0.16	0.99	0.16
	Bagging with CV	0.25	0.95	0.26
	Random Forest	0.51	0.96	0.55
	AdaBoost	0.38	0.99	0.53
	AdaBoost with CV	0.26	0.95	0.26
	SAMME	0.49	0.95	0.51
	SVMs	0.63	0.92	0.52
	ANNs	0.49	0.91	0.42
Cytoplasm	Bagging	0.10	0.97	0.10
	Bagging with CV	0.08	0.95	0.05
	Random Forest	0.27	0.92	0.24
	AdaBoost	0.25	0.90	0.18
	AdaBoost with CV	0.07	0.96	0.05
	SAMME	0.41	0.84	0.25
	SVMs	0.34	0.87	0.22
	ANNs	0.12	0.95	0.11
ER	Bagging	0.29	0.98	0.29
	Bagging with CV	0.30	0.97	0.33
	Random Forest	0.21	0.99	0.34
	AdaBoost	0.21	0.98	0.27
	AdaBoost with CV	0.29	0.98	0.34
	SAMME	0.21	0.98	0.29
	SVMs	0.35	0.98	0.40
	ANNs	0.21	0.98	0.27
Golgi	Bagging	0	1	0
	Bagging with CV	0	1	0
	Random Forest	0	1	0
	AdaBoost	0	1	0
	AdaBoost with CV	0	1	0
	SAMME	0	1	0
	SVMs	0.03	1	0.13
	ANNs	0	1	0
Lysosome	Bagging	0	1	0
	Bagging with CV	0	1	0
	Random Forest	0	1	0
	AdaBoost	0	1	0
	AdaBoost with CV	0	1	0
	SAMME	0	1	0
	SVMs	0	1	0
	ANNs	0	1	0
Nucleus	Bagging	0.93	0.35	0.93
	Bagging with CV	0.89	0.38	0.28
	Random Forest	0.89	0.58	0.45
	AdaBoost	0.84	0.54	0.36
	AdaBoost with CV	0.90	0.37	0.28
	SAMME	0.77	0.71	0.45
	SVMs	0.76	0.76	0.50
	ANNs	0.85	0.65	0.47
Peroxisome	Bagging	0	1	0
	Bagging with CV	0	1	0
	Random Forest	0	1	0
	AdaBoost	0	1	0
	AdaBoost with CV	0	1	0
	SAMME	0	1	0
	SVMs	0	1	0
	ANNs	0	1	0
Cell Membrane	Bagging	0.39	0.94	0.39
	Bagging with CV	0.30	0.95	0.27
	Random Forest	0.49	0.95	0.44
	AdaBoost	0.53	0.94	0.44
	AdaBoost with CV	0.30	0.95	0.62
	SAMME	0.53	0.95	0.45
	SVMs	0.59	0.95	0.54
	ANNs	0.39	0.95	0.36

The Golgi apparatus, the lysosome, and the peroxisome locations were not predicted or predicted extremely poorly by the classifiers in this paper. Although the results illustrated in the Meinken et al. [17] table have much lower prediction values compared to the other locations, they predict better values than the classifiers in this study.

The ER location in Meinken et al. paper [17] has prediction values with a sensitivity of 0.33, a specificity of 0.99 and an MCC value of 0.42. The SVMs predictor produces results with a sensitivity of 0.35, a specificity of 0.98 and an MCC value of 0.40. Therefore, for this location, the values of predictions from both studies are very close to each other.

Next, for the nucleus subcellular location, the method employed by Meinken et al. [17] predicts with a sensitivity of 0.79, a specificity of 0.94 and an MCC value of 0.72. Meanwhile, the best predictor in this research is the bagging predictor produced a sensitivity of 0.93, a specificity of 0.35 and an MCC value of 0.93. Therefore, the predictors used by this paper outperform the predictors used by Meinken et al. [17].

Finally, for the cell (plasma) membrane, the prediction results by Meinken et al. [17] predicts a sensitivity of 0.83, a specificity of 0.96 and an MCC value of 0.78. Meanwhile, the best predictor in this research is the AdaBoost with cross validation predictor producing a sensitivity of 0.30, a specificity of 0.95 and an MCC value of 0.62. Therefore, the values of prediction are lower than the Meinken et al. [17] for this location as well.

7 Future Work

In conclusion, the results show that the computational predictors used by Meinken et al. [17] do a better job at predicting the subcellular location of proteins in every case except for two locations. For the nuclear location, this study predicts protein location with a better sensitivity and MCC value. Also, for the ER location, the results from this paper are very competitive with the results from the Meinken et al. [17] study. It is important to point out that the lower levels of MCC in this study were the result of the lower values of sensitivity measures in each subcellular location. By looking at both tables, it is clear that the specificity results obtained by using different computational predictors in this paper are either higher or tied with the specificity values from the Meinken et al. study (except for the nuclear location) [17]. Therefore, further work will include efforts on improving the sensitivity measures for each subcellular location.

The future direction for this project includes exploring other available predictors in order to determine whether the overall accuracy of protein subcellular location prediction can be increased.

Also, other methods can be explored to attempt to obtain a higher predictive value for other subcellular locations than the three locations discussed in this paper. Examples of these methods include the covariant discriminant algorithm and random walk on graphs.

Another issue that arose from this research was how to analyze imbalanced data. In this paper, the measures of sensitivity, specificity, and MCC were used to analyze the results. However, additional attention could be brought to finding better ways to prepare the data set (over-sampling or under-sampling), before using the computational predictors. The R software offers a package for "Synthetic Sampling", which can be used on this dataset for future work. In addition, using "penalized SVM", in which additional costs are imposed on the model for making mistakes in classification, can also be applied to this work to improve the overall analysis of the data.

References

[1] Abdollah Dehzangi, Rhys Heffernan, A. S. J. L. K. P. Gram-positive and gram-negative protein subcellular localization by incorporating evolutionary-based descriptors into chou's general pseaaceneral pseaac. *Journal of Theoretical Biology 364* (2015), 284–294.

[2] Anil K. Jain, Jianchang Mao, K. M. Artificial neural networks: A tutorial. *IEEE Computer Society 29*, 3 (1996), 31–44.

[3] Anubha Dubey, U. C. Subcellular localization of proteins. *Archives of Applied Science Research 3*, 6 (2011), 392–401.

[4] Ashutosh Kumar Singh, S. S. S., and Mishra, A. Sub-cellular localization prediction using machine learning approach. *Nucleus 734*, 568 (2015), 63.

[5] Breiman, L. Bagging predictors. *Machine Learning 24* (1996a), 123–140.

[6] Breiman, L. Out-of-bag estimation. `ftp://ftp.stat.berkeley.edu/pub/users/breiman/OOBestimation.ps.Z.`, 1996b.

[7] Breiman, L. Using adaptive bagging to debias regressions. `https://www.stat.berkeley.edu/~breiman/adaptbag99.pdf.`, 1999.

[8] Breiman, L. Random forests. *Machine Learning 45* (2001), 5–32.

[9] Chih-Wei Hsu, C.-C. C., and Lin, C.-J. A practical guide to support vector classification. 4 2010.

[10] Chih-Wei Hsu, C.-J. L. A comparison of methods for multi-class support vector machines. *IEEE Transactions on Neural Networks 13*, 2 (2002), 415–425.

[11] Chou KC, Z. C. Prediction of protein structural classes. *Critical Reviews in Biochemistry and Molecular Biology 30* (1995), 275–349.

[12] consortium, T. U. Uniprot knowledgebase (uniprotkb), 11 2015.

[13] D. Charif, J. L. Sequinr: A contributed pacage to the r project for statistical computing devoted to biological sequence retrieval and analysis.

[14] Dietterich, T. G. An experimental comparison of three methods for constructing ensembles of decision tress: Bagging, boosting, and randomization. *Machine Learning 40* (2000), 139–157.

[15] for Developmental Biology, M.-P. I.

[16] Ji Zhu, Saharon Rosset, H. Z. T. H. Multi-class adaboost. http://www.researchgate.net/profile/Trevor_Hastie/publication/228947999_Multi-class_adaboost/links/0c960521b946de42a9000000.pdf., 2006.

[17] John Meinken, Gary Walker, C. R. C., and Min, X. J. Metazseckb: the human and animal secretome and subcellular proteome knowledgebase. *Database: The Journal of Biological Databases and Curation Article ID bav077* (2015).

[18] Josefine Sprenger, J Lynn Fink, R. D. T. Evaluation and comparison of mammalian subcellular localization prediction methods. *BMC Bioinfomatics 7*, (Suppl 5):S3 (2006).

[19] J.R.QUINLAN. Induction of decision trees. *Machine Learning 1* (1986), 81–106.

[20] Keun-Joon Park, M. K. Prediction of protein subcellular locations by support vector machines using compositions of amino acids and amino acid pairs. *Bioinformatics 19* (2003), 1656–1663.

[21] Kou-Chen Chou, H.-B. S. Recent progress in protein subcellular location prediction. *Analytical Biochemistry 370* (2007), 1–6.

[22] Kou-Chen Chou, H.-B. S. A new method for predicting the subcellular localization of eukaryotic proteins with both single and multiple sites: Euk-mploc 2.0. *PLOSONE 5*, 4 (2010).

[23] Martin T. Hagan, Howard B. Demuth, M. H. B., and Jesús, O. D. *Neural Network Design*, 2 ed. Martin Hagan, 2014.

[24] Matthews, B. Comparison of the predicted and observed secondary structure of t4 phage lysozyme. *Biochim. Biophys. Acta 405* (1975), 442–451.

[25] Min, X. J. Evaluation of computational methods for secreted protein prediction in different eukaryotes. *Journal of Proteomics and Bioinformatics 3*, 5 (2010).

[26] Munyon, J. Predicting fungal protein subcellular location. May 2015.

[27] N. Xiao, Q.S. Xu, D. C. protr: Generating various numerical representation scheme of protien sequence.

[28] Neizer-Ashun, K. A. Prediction of plant protein subcellular location. 7th International Conference on Bioinformatics and Computational Biology (BICoB-2015).

[29] Reinhardt, A., and Hubbard, T. Using neural networks for prediction of the subcellular location of proteins. *Nucleic Acids Research 26*, 9 (1998), 2230–2236.

[30] Saravanan K, S. S. Review on classification based on artificial neural networks. *International Journal of Ambient Systems and Applications (IJASA) 2*, 4 (2014), 11–18.

[31] Sons, J. W. . Typical structures found in body cells.

[32] Sylvia S. Mader, M. W. *ESSENTIALS OF BIOLOGY*, fourth edition ed. MC GRAW HILL EDUCATION, 2015.

[33] Team, R. C. R: A language and environment for statistical computing, 11 2015.

[34] Yali Amit, D. G. Shape quantization and recognition with randomized trees. *Neural Computation 9*, 1545-1588 (1997).

[35] Yoav Freund, R. E. S. Experiments with a new boosting algorithm. *Machine Learning: Proceedings of the Thirteenth International Conference* (1996).

[36] Yoav Freund, R. E. S. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences 55* (1997), 119–139.

[37] Yoav Freund, R. E. S. A short introduction to boosting a short introduction to boosting. *Journal of Japanese Society for Artificial Intelligence, 14*, 5 (1999), 771–780.

[38] Yu-Dong Cai, Xiao-Jun Liu, X.-b. X., and Chou, K.-C. Support vector machines for prediction of protein subcellular location by incorporating quasi-sequence-order effect. *Journal of Cellular Biochemistry 84*, 343-348 (2002).

[39] Yuan, Z. Prediction of protein subcellular location using markov chain model. *FEBS letters 451*, 23-26 (1999).

8 Appendix: R Code

```
MASTER'S PROJECT



### First - install and load packages with install.packages()

library(seqinr)
library(protr)
library(party)
library(BioSeqClass)
library(randomForest)
library(lme4)
library(caret)
library(adabag)
library(tree)
library(rpart)
library(DMwR)

#modification to featurepseudoAAComp because of problems
myfeaturePseudoAAComp=function(seq,d,w=.05){
  H1 = c(0.62, -2.53, -0.78, -0.09, 0.29, -0.85, -0.74, 0.48,
         -0.4, 1.38, 1.53, -1.5, 0.64, 1.19, 0.12, -0.18, -0.05,
         0.81, 0.26, 1.8)
  names(H1) = c("A", "R", "N", "D", "C", "Q", "E", "G", "H",
                "I", "L", "K", "M", "F", "P", "S", "T", "W", "Y", "V")
  H2 = c(-0.5, 3, 0.2, 3, -1, 0.2, 3, 0, -0.5, -1.8, -1.8,
         3, -1.3, -2.5, 0, 0.3, -0.4, -3.4, -2.3, -1.5)
  names(H2) = c("A", "R", "N", "D", "C", "Q", "E", "G", "H",
                "I", "L", "K", "M", "F", "P", "S", "T", "W", "Y", "V")
  M = c(71.03711, 156.10111, 114.04293, 115.02694, 103.00919,
        128.05858, 129.04259, 57.02146, 137.05891, 113.08406,
        113.08406, 128.09496, 131.04049, 147.06841, 97.05276,
        87.03203, 101.04768, 186.07931, 163.06333, 99.06841)
  names(M) = c("A", "R", "N", "D", "C", "Q", "E", "G", "H",
               "I", "L", "K", "M", "F", "P", "S", "T", "W", "Y", "V")
  H1 = (H1 - mean(H1))/sd(H1)
  H2 = (H2 - mean(H2))/sd(H2)
  M = (M - mean(M))/sd(M)
  R <- function(aa1, aa2) {
    ((H1[aa1] - H1[aa2])^2 + (H2[aa1] - H2[aa2])^2 + (M[aa1] -
```

```
                                                    M[aa2])^2)/3
  }
  compo = featureFragmentComposition(seq, 1, elements("aminoacid"))
  pseudo = sapply(seq, function(x) {
    y = unlist(strsplit(x, split = ""))
    nTotal = length(y)
    sapply(1:d, function(i) {
      mean(sapply(1:(nTotal - i), function(j) {
        R(y[j], y[j + i])
      }))
    })
  })


  # OUR CODE TO FIX THE PROBLEMS
  protlist=lapply(X=1:length(data),FUN=function(z){
    a=unlist(compo[1,z])
    b=pseudo[,z]
    b=b/sum(b)
    c=c(a,b)
    c=c/sum(c)
    return(c)
  })

  # making data frame with each row being a protein
  paa=data.frame()

  for(i in 1:length(data)){
    paa=rbind(paa,protlist[[i]])
  }

  # fixing row names and column names
  rownames(paa)=names(seq)
  colnames(paa)=c("A", "R", "N", "D", "C", "Q", "E", "G", "H",
                  "I", "L", "K", "M", "F", "P", "S", "T", "W", "Y", "V
                       ","1","2","3","4","5","6","7","8","9","10","11","12","13","14","15")
  return(paa)

}
```

```
### Read in protein data, already mostly subsetted properly using online database
### (readFASTA may not work; may have to use read.fasta (they're from different packages))
    :

data = readFASTA("/Users/Podkayne/Documents/YSU/Master's Thesis/benchmark.fasta")#,seqtype
     = "AA", as.string = TRUE)

### Remove any proteins containing an X (unknown amino acid) in their sequences:
xprot = NULL
for(i in 1:length(data)){
  if(grepl("X", data[i]) == TRUE){
    xprot = c(xprot, i)
  }
}

data= data[-xprot]
### Shorten protein names to just their length 6 identifiers:
names(data) = sapply(names(data), function(x){
  substring(x, 4, 9)
})
### Set a seed for reproducability (since random sampling is coming up):
set.seed(1357)
### Read in output text file from "50/50" BLASTClust, where the one or more proteins on
    each line represent a specific cluster:
##sep is \n newline character
scan = scan("/Users/Podkayne/Documents/YSU/Master's Thesis/clusters.txt", what = character
    (), sep = "\n")
### Create an R list, each list element a cluster-giving it a structure
list = list()
for(i in 1:length(scan)){
  list[[i]] = unlist(strsplit(scan[i], " "))
}

### Shorten protein names to just their length 6 identifiers:
for(i in 1:length(list)){
  for(j in 1:length(list[[i]])){
    list[[i]][j] = substring(list[[i]][j], 4, 9)
  }
}

### For each cluster in the list, randomly select just one protein from the cluster:
```

```
proteins = NULL
for(i in 1:length(list)){
  proteins[i] = list[[i]][sample(1:length(list[[i]]), 1)]
}

### Remove proteins that weren't previously selected:
wasteprot = NULL
for(i in 1:length(data)){
  if((names(data)[i] %in% proteins) == FALSE){
    wasteprot = c(wasteprot, i)
  }
}
#throwing away not needed proteins and keeping the rest
data = data[-wasteprot]

### Read in list of known protein locations:
locations = read.csv("/Users/Podkayne/Documents/YSU/Master's Thesis/metaz_subloc_trim1_325
    .csv", header = TRUE, col.names = c("Protein", "Location"))
### Transform protein sequences by means of Pseudo Amino Acid Composition:
#PAA = featurePseudoAAComp(data, d = 15, w = 0.05)

# the line above didn't work, running the below line

PAA = myfeaturePseudoAAComp(data, d = 15, w = 0.05)

### Save results to machine:
write.csv(PAA, "/Users/Podkayne/Documents/YSU/Master's Thesis/50.50paabench.csv")
### Read results back in:
PAA = read.csv("/Users/Podkayne/Documents/YSU/Master's Thesis/50.50paabench.csv", header =
     TRUE, row.names = 1)

#re-order data alphabetize paa
PAA=PAA[order(row.names(PAA)),]

#re-order locations alphabetically

locations=locations[order(locations),]

### Remove location listings for proteins that aren't now in our dataset:
subcellular = rep("word", nrow(PAA))
a = NULL
```

```
for (i in 1:nrow(locations)){
  if (locations[i,1] %in% rownames(PAA) == FALSE){
    a = c(a,i)
  }
}
locations = locations[-a,]

### Combine known locations and the PAA values into a data frame:
subcellular = as.factor(as.character(locations$Location))
PAA = data.frame(subcellular, PAA)
### Save data frame to machine:
write.csv(PAA, "/Users/Podkayne/Documents/YSU/Master's Thesis/50.50paabench.csv")
### End of preprocessing-- data completely cleanded up.
save.image("/Users/Podkayne/Documents/YSU/Master's Thesis/cleanedup.RData")
###End of preprocessing#############################################################

###BEGINNING OF ANALYSIS
###Refresh Seed
set.seed(1357)
#read data back in
DATA = read.csv("/Users/Podkayne/Documents/YSU/Master's Thesis/50.50paabench.csv", header
    = TRUE, row.names = 1)
### for loop for removing for 5 NA's
dumpvector=NULL
for(i in 1:nrow(DATA)){
  if("NA"%in% as.character(DATA[i,])){
    dumpvector=c(dumpvector,i)
  }
}
DATA=DATA[-dumpvector,]
### Take a sample of 70% of the numbers in 1 through the number of proteins in the dataset
    :
nums = sample(1:nrow(DATA), floor(.7*nrow(DATA)))
### (need to do this so models can know that each location is a "possibility"):
traindata = DATA[nums,]
while (0 %in% summary(traindata$subcellular) == TRUE){
  nums = sample(1:nrow(DATA), floor(.7*nrow(DATA)))
  traindata = DATA[nums,]
}
### Create 30% testing set:
testdata = DATA[-nums,]
```

```
### Random Forests
### Create Random Forest model with training data - 500 trees is default:
RFOREST = randomForest(formula = subcellular ~ ., data = traindata, importance = TRUE,
    proximity = TRUE,na.action=na.omit)
### Variable importance plot:
VarImpPlot_RFOREST = varImpPlot(RFOREST)
### Obtain predictions for testing data:
predicted_RFOREST = predict(RFOREST, newdata = testdata)
### Save actual locations of testing data:
actual = testdata[,1]
### Calculate error rate of classification:1-percentage of right= percentage of wrong
error_RFOREST = 1 - sum(predicted_RFOREST == actual)/nrow(testdata)
### Create confusion matrix and performance statistics for each location:
confusion_RFOREST = confusionMatrix(data = predicted_RFOREST, reference = actual)


### Adaptive Boosting
### Set some "appropriate" parameter options for the algorithm:
cntrl = rpart.control(maxdepth = 6, minsplit = 0, cp = -1)
### Create adaptive boosting model with training data, using 500 trees:
ada = boosting(formula = subcellular ~ ., data = traindata, mfinal = 500, coeflearn = "
    Freund", boos = TRUE, control = cntrl)
### Variable importance plot:
varImp_ada = barplot(ada$importance[order(ada$importance, decreasing = TRUE)], ylim = c
    (0,20), main = "Variables Relative Importance", col = "lightblue")
### Obtain predictions for testing data:
predicted_ada = predict(object = ada, newdata = testdata, newmfinal = 500)
### Calculate error rate of classification:
error_ada = 1 - sum(predicted_ada$class == actual)/nrow(testdata)
### Create confusion matrix and performance statistics for each location:
predicted_ada_class = as.factor(predicted_ada$class)
levels(predicted_ada_class) = c(levels(predicted_ada_class), levels(actual))
predicted_ada_class = factor(predicted_ada_class, levels(predicted_ada_class)[order(levels
    (predicted_ada_class))])
confusion_ada = confusionMatrix(data = predicted_ada_class, reference = actual)


### Adaptive Boosting with ten-fold cross-validation
### Create adaptive boosting with ten-fold cross-validation model with full dataset, using
     (50 trees in each "fold")*(10 "folds") = 500 total trees:
```

```
ada_cv = boosting.cv(formula = subcellular ~ ., data = DATA, v = 10, boos = TRUE, mfinal =
     50, coeflearn = "Freund", control = rpart.control(maxdepth = 10))
### Variable importance plot (gives an error for James, ran okay with mine):
#varImp_ada_cv = barplot(ada_cv$importance[order(ada_cv$importance, decreasing = TRUE)],
    ylim = c(0,20), main = "Variables Relative Importance", col = "lightblue")
### Save actual locations of full dataset:
ACTUAL = DATA[,1]
### Calculate error rate of classification:
error_ada_cv = 1 - sum(ada_cv$class == ACTUAL)/nrow(DATA)
### Create confusion matrix and performance statistics for each location:
ada_cv_class = as.factor(ada_cv$class)
levels(ada_cv_class) = c(levels(ada_cv_class), levels(actual))
ada_cv_class = factor(ada_cv_class, levels(ada_cv_class)[order(levels(ada_cv_class))])
confusion_ada_cv = confusionMatrix(data = ada_cv_class, reference = ACTUAL)



### SAMME
### Create SAMME model with training data, using 500 trees
samme = boosting(formula = subcellular ~ ., data = traindata, mfinal = 500, coeflearn = "
    Zhu", boos = TRUE, control = cntrl)
### Variable importance plot:
varImp_samme = varImp_samme = barplot(samme$importance[order(samme$importance, decreasing
    = TRUE)], ylim = c(0,20), main = "Variables Relative Importance", col = "lightblue")
### Obtain predictions for testing data:
predicted_samme = predict(object = samme, newdata = testdata, newmfinal = 500)
### Calculate error rate of classification:
error_samme = 1 - sum(predicted_samme$class == actual)/nrow(testdata)
### Create confusion matrix and performance statistics for each location:
predicted_samme_class = as.factor(predicted_samme$class)
levels(predicted_samme_class) = c(levels(predicted_samme_class), levels(actual))
predicted_samme_class = factor(predicted_samme_class, levels(predicted_samme_class)[order(
    levels(predicted_samme_class))])
confusion_samme = confusionMatrix(data = predicted_samme_class, reference = actual)
################################################################################

###SAVE RANDOM FOREST RESULTS TO MACHINE

write.csv(as.data.frame.matrix(confusion_RFOREST$table),file = "/Users/Podkayne/Documents/
    YSU/Master's Thesis/DataResults/RANDOMFOREST_confusion.csv")
write.csv(as.data.frame.matrix(confusion_RFOREST$byClass),file = "/Users/Podkayne/
```

```
    Documents/YSU/Master's Thesis/DataResults/ RANDOMFOREST_stats.csv")
write.table(error_RFOREST,"/Users/Podkayne/Documents/YSU/Master's Thesis/DataResults/
    RANDOMFOREST_stats.csv")
### I added this part to round the values of confusion matrix and save a csv file
confusion_RFOREST$byClassRounded <- round(confusion_RFOREST$byClass, 2)
write.csv(as.data.frame.matrix(confusion_RFOREST$byClassRounded),file = "/Users/Podkayne/
    Documents/YSU/Master's Thesis/DataResults/ RANDOMFORESTROUNDED.csv")

### Set c as a confusion matrix, previously calculated and stored
### (this confusion matrix has rows as predictions and columns as actuals):
c = as.matrix(as.data.frame.matrix(confusion_RFOREST$table))
### Initialize length 16 MCC vector:
MCC = vector(mode = "numeric", length = 9)
### For each location:
for(k in 1:9){
  A = as.double(c[k,k]) ### A = #(true positives)
  B = as.double(sum(c[k,]) - c[k,k]) ### B = #(false positives)
  C = as.double(sum(c[,k]) - c[k,k]) ### C = #(false negatives)
  D = as.double(sum(c) - A - B - C) ### D = #(true negatives)
  if((((A + B)*(A + C)*(D + B)*(D + C))^.5) == 0){ ### if denominator is zero, arbitrarily
       set as one - MCC will come out to be zero
    MCC[k] = (A*D) - (B*C)
  }
  else{MCC[k] = ((A*D) - (B*C))/(((A + B)*(A + C)*(D + B)*(D + C))^.5)} ### MCC value by
      formula
  rm(A,B,C,D)
}

### Rounding, setting as column vector:
MCC = round(matrix(MCC, nrow = 9, ncol = 1), digits = 2)
### Set appropriate column name:
colnames(MCC) = "MCC"
### Save results to machine (to be appended to pre-existing statistics by class table):
write.table(MCC, file = "/Users/Podkayne/Documents/YSU/Master's Thesis/DataResults/RF_MCCs
    .csv", sep = ",", row.names = FALSE, col.names = TRUE)
rm(c, MCC)



###SAVE ADAPTIVE BOOSTING RESULTS TO MACHINE
```

```
write.csv(as.data.frame.matrix(confusion_ada$table), file = "/Users/Podkayne/Documents/YSU
    /Master's Thesis/DataResults/ADA_confusion.csv")
write.csv(as.data.frame.matrix(confusion_ada$byClass), file = "/Users/Podkayne/Documents/
    YSU/Master's Thesis/DataResults/ADA_stats.csv")
write.csv(error_ada,file = "/Users/Podkayne/Documents/YSU/Master's Thesis/DataResults/
    ADA_stats.csv")

### I added this part to round the values of confusion matrix and save a csv file
confusion_ada$byClassRounded <- round(confusion_ada$byClass, 2)
write.csv(as.data.frame.matrix(confusion_ada$byClassRounded),file = "/Users/Podkayne/
    Documents/YSU/Master's Thesis/DataResults/ ADAROUNDED.csv")


### Set c as a confusion matrix, previously calculated and stored
### (this confusion matrix has rows as predictions and columns as actuals):
c = as.matrix(as.data.frame.matrix(confusion_ada$table))
### Initialize length 16 MCC vector:
MCC = vector(mode = "numeric", length = 9)
### For each location:
for(k in 1:9){
  A = as.double(c[k,k]) ### A = #(true positives)
  B = as.double(sum(c[k,]) - c[k,k]) ### B = #(false positives)
  C = as.double(sum(c[,k]) - c[k,k]) ### C = #(false negatives)
  D = as.double(sum(c) - A - B - C) ### D = #(true negatives)
  if((((A + B)*(A + C)*(D + B)*(D + C))^.5) == 0){ ### if denominator is zero, arbitrarily
       set as one - MCC will come out to be zero
    MCC[k] = (A*D) - (B*C)
  }
  else{MCC[k] = ((A*D) - (B*C))/(((A + B)*(A + C)*(D + B)*(D + C))^.5)} ### MCC value by
      formula
  rm(A,B,C,D)
}

### Rounding, setting as column vector:
MCC = round(matrix(MCC, nrow = 9, ncol = 1), digits = 2)
### Set appropriate column name:
colnames(MCC) = "MCC"
### Save results to machine (to be appended to pre-existing statistics by class table):
write.table(MCC, file = "/Users/Podkayne/Documents/YSU/Master's Thesis/DataResults/
    ada_MCCs.csv", sep = ",", row.names = FALSE, col.names = TRUE)
rm(c, MCC)
```

```
###SAVE ADAPTIVE BOOSTING with 10 FOLD CROSS VILAIDATION RESULTS TO MACHINE

write.csv(as.data.frame.matrix(confusion_ada_cv$table), file = "/Users/Podkayne/Documents/
    YSU/Master's Thesis/DataResults/ADA_CV_confusion.csv")
write.csv(as.data.frame.matrix(confusion_ada_cv$byClass), file = "/Users/Podkayne/
    Documents/YSU/Master's Thesis/DataResults/ADA_CV_stats.csv")
write.csv(error_ada_cv,file = "/Users/Podkayne/Documents/YSU/Master's Thesis/DataResults/
    ADA_CV_stats.csv")

### I added this part to round the values of confusion matrix and save a csv file
confusion_ada_cv$byClassRounded <- round(confusion_ada_cv$byClass, 2)
write.csv(as.data.frame.matrix(confusion_ada_cv$byClassRounded),file = "/Users/Podkayne/
    Documents/YSU/Master's Thesis/DataResults/ ADACVROUNDED.csv")



### Set c as a confusion matrix, previously calculated and stored
### (this confusion matrix has rows as predictions and columns as actuals):
c = as.matrix(as.data.frame.matrix(confusion_ada_cv$table))
### Initialize length 16 MCC vector:
MCC = vector(mode = "numeric", length = 9)
### For each location:
for(k in 1:9){
  A = as.double(c[k,k]) ### A = #(true positives)
  B = as.double(sum(c[k,]) - c[k,k]) ### B = #(false positives)
  C = as.double(sum(c[,k]) - c[k,k]) ### C = #(false negatives)
  D = as.double(sum(c) - A - B - C) ### D = #(true negatives)
  if((((A + B)*(A + C)*(D + B)*(D + C))^.5) == 0){ ### if denominator is zero, arbitrarily
       set as one - MCC will come out to be zero
    MCC[k] = (A*D) - (B*C)
  }
  else{MCC[k] = ((A*D) - (B*C))/(((A + B)*(A + C)*(D + B)*(D + C))^.5)} ### MCC value by
      formula
  rm(A,B,C,D)
}

### Rounding, setting as column vector:
MCC = round(matrix(MCC, nrow = 9, ncol = 1), digits = 2)
### Set appropriate column name:
```

```
colnames(MCC) = "MCC"
### Save results to machine (to be appended to pre-existing statistics by class table):
write.table(MCC, file = "/Users/Podkayne/Documents/YSU/Master's Thesis/DataResults/
    ada_cv_MCCs.csv", sep = ",", row.names = FALSE, col.names = TRUE)
rm(c, MCC)


###SAVE SAMME RESULTS TO MACHINE

write.csv(as.data.frame.matrix(confusion_samme$table),file = "/Users/Podkayne/Documents/
    YSU/Master's Thesis/DataResults/SAMME_confusion.csv")
write.csv(as.data.frame.matrix(confusion_samme$byClass),file = "/Users/Podkayne/Documents/
    YSU/Master's Thesis/DataResults/SAMME-stats.csv")
write.table(error_samme,file = "/Users/Podkayne/Documents/YSU/Master's Thesis/DataResults/
    SAMME_stats.csv")

### I added this part to round the values of confusion matrix and save a csv file
confusion_samme$byClassRounded <- round(confusion_samme$byClass, 2)
write.csv(as.data.frame.matrix(confusion_samme$byClassRounded),file = "/Users/Podkayne/
    Documents/YSU/Master's Thesis/DataResults/ SAMMEROUNDED.csv")

### Set c as a confusion matrix, previously calculated and stored
### (this confusion matrix has rows as predictions and columns as actuals):
c = as.matrix(as.data.frame.matrix(confusion_samme$table))
### Initialize length 16 MCC vector:
MCC = vector(mode = "numeric", length = 9)
### For each location:
for(k in 1:9){
  A = as.double(c[k,k]) ### A = #(true positives)
  B = as.double(sum(c[k,]) - c[k,k]) ### B = #(false positives)
  C = as.double(sum(c[,k]) - c[k,k]) ### C = #(false negatives)
  D = as.double(sum(c) - A - B - C) ### D = #(true negatives)
  if((((A + B)*(A + C)*(D + B)*(D + C))^.5) == 0){ ### if denominator is zero, arbitrarily
      set as one - MCC will come out to be zero
    MCC[k] = (A*D) - (B*C)
  }
  else{MCC[k] = ((A*D) - (B*C))/(((A + B)*(A + C)*(D + B)*(D + C))^.5)} ### MCC value by
      formula
  rm(A,B,C,D)
}
```

```
### Rounding, setting as column vector:
MCC = round(matrix(MCC, nrow = 9, ncol = 1), digits = 2)
### Set appropriate column name:
colnames(MCC) = "MCC"
### Save results to machine (to be appended to pre-existing statistics by class table):
write.table(MCC, file = "/Users/Podkayne/Documents/YSU/Master's Thesis/DataResults/
    samme_MCCs.csv", sep = ",", row.names = FALSE, col.names = TRUE)
rm(c, MCC)
##########################################################################################################

##########################################################################################################

# SUPPORT VECTOR MACHINES:

library(e1071)
### "Refresh" seed - not really necessary:
set.seed(0)
### Read in ready-to-go data:
DATA = read.csv("/Users/Podkayne/Documents/YSU/Master's Thesis/50.50paabench.csv", header
    = TRUE, row.names = 1)

### for loop for removing for 5 NA's
dumpvector=NULL
for(i in 1:nrow(DATA)){
  if("NA"%in% as.character(DATA[i,])){
    dumpvector=c(dumpvector,i)
  }
}
DATA=DATA[-dumpvector,]


### Set a seed for reproducability (since random sampling is coming up withinthe cross-
    validation):
  set.seed(1234)
### Determine optimal SVM parameters using ten-fold cross-validation


tune = tune.svm(x = DATA[,-1], y = DATA[,1],
                gamma = 2^(seq(from = -15, to = 3, by = 2)),
                cost = 2^(seq(from = -5, to = 15, by = 2)))
                tunecontrol = tune.control(sampling = c("cross"))
```

```
### Save the optimal parameters:
parameters = unlist(tune$best.parameters)
### Save a list of numbers, from 1 to the number of proteins in the dataset:
nums = 1:nrow(DATA)
### Create (2949) SVM models, each one using all but one left-out protein
### Then predict the left-out protein?s location
### (this is jackknife validation):
svms = lapply(nums, function(z){
  a = svm(x = DATA[-z,-1], y = DATA[-z,1], cost = parameters["cost"], gamma =
            parameters["gamma"])
  b = predict(object = a, newdata = DATA[z,-1])
  return(prediction = b)
})
### Unlist svms (turn R object from a list to a vector):
svms = unlist(svms)
### Save actual locations of full dataset:
44

ACTUAL = DATA[,1]
### Calculate error rate of classification:
error_svms = 1 - sum(svms == ACTUAL)/nrow(DATA)
### Create confusion matrix and performance statistics for each location:

confusion_svms = confusionMatrix(data = svms, reference = ACTUAL)

#### My SOLUTION TO CONFUSIONMATRIX FUNCTION ERROR (The line above worked on the third try
    )

# Create a confusion matrix from the given outcomes, whose rows correspond
# to the actual and the columns to the predicated classes.
createConfusionMatrix <- function(ACTUAL, svms) {
  # You've mentioned that neither actual nor predicted may give a complete
  # picture of the available classes, hence:
  numClasses <- max(ACTUAL, svms)
  # Sort predicted and actual as it simplifies what's next. You can make this
  # faster by storing `order(act)` in a temporary variable.
 # pred <- pred[order(act)]
 # act  <- act[order(act)]
  #sapply(split(pred, act), tabulate, nbins=numClasses)
}
```

```
# Generate random data since you've not provided an actual example.
#actual    <- sample(1:4, 1000, replace=TRUE)
#predicted <- sample(c(1L,2L,4L), 1000, replace=TRUE)

print( createConfusionMatrix(ACTUAL, svms) )


### Save SVM results to machine
write.csv(as.data.frame.matrix(confusion_svms$table), file = "svms_confusion.csv")
write.csv(as.data.frame.matrix(confusion_svms$byClass), file = "svms_stats.csv")
write.table(error_svms, file = "svmserror_stats.csv")
### Save R workspace
save.image("/Users/Podkayne/Documents/YSU/Master's Thesis//DataResultsSVM_results_from_lab
    .RData")

### I added this part to round the values of confusion matrix and save a csv file
confusion_svms$byClassRounded <- round(confusion_svms$byClass, 2)
write.csv(as.data.frame.matrix(confusion_svms$byClassRounded),file = "/Users/Podkayne/
    Documents/YSU/Master's Thesis/DataResults/ SVMSROUNDED.csv")

### Set c as a confusion matrix, previously calculated and stored
### (this confusion matrix has rows as predictions and columns as actuals):
c = as.matrix(as.data.frame.matrix(confusion_svms$table))
### Initialize length 16 MCC vector:
MCC = vector(mode = "numeric", length = 9)
### For each location:
for(k in 1:9){
  A = as.double(c[k,k]) ### A = #(true positives)
  B = as.double(sum(c[k,]) - c[k,k]) ### B = #(false positives)
  C = as.double(sum(c[,k]) - c[k,k]) ### C = #(false negatives)
  D = as.double(sum(c) - A - B - C) ### D = #(true negatives)
  if((((A + B)*(A + C)*(D + B)*(D + C))^.5) == 0){ ### if denominator is zero, arbitrarily
       set as one - MCC will come out to be zero
    MCC[k] = (A*D) - (B*C)
  }
  else{MCC[k] = ((A*D) - (B*C))/(((A + B)*(A + C)*(D + B)*(D + C))^.5)} ### MCC value by
      formula
  rm(A,B,C,D)
}
```

```
### Rounding, setting as column vector:
MCC = round(matrix(MCC, nrow = 9, ncol = 1), digits = 2)
### Set appropriate column name:
colnames(MCC) = "MCC"
### Save results to machine (to be appended to pre-existing statistics by class table):
write.table(MCC, file = "/Users/Podkayne/Documents/YSU/Master's Thesis/DataResults/
    SVMS_MCCs.csv", sep = ",", row.names = FALSE, col.names = TRUE)
rm(c, MCC)

##########################################################################################################

##########################################################################################################

# BAGGING:

### Bagging (Bootstrap Aggregating)
### Create bagging model with training data, using 500 trees:
set.seed(2345)
bagging = bagging(formula = subcellular ~ ., data = traindata, mfinal = 500)
### Variable importance plot:
varImp_bagging = barplot(bagging$importance[order(bagging$importance,
                         decreasing = TRUE)], ylim = c(0,20), main = "Variables Relative
                             Importance",
                         col = "lightblue")
### Obtain predictions for testing data:
predicted_bagging = predict(object = bagging, newdata = testdata, newmfinal = 500)
###save actual locations of testing data (I have added to free the error)
actual=testdata[,1]
### Calculate error rate of classification:
error_bagging = 1 - sum(predicted_bagging$class == actual)/nrow(testdata)
### Create confusion matrix and performance statistics for each location:
predicted_bagging_class = as.factor(predicted_bagging$class)
levels(predicted_bagging_class) = c(levels(predicted_bagging_class), levels(
  actual))
predicted_bagging_class = factor(predicted_bagging_class, levels(
  predicted_bagging_class)[order(levels(predicted_bagging_class))])
confusion_bagging = confusionMatrix(data = predicted_bagging_class, reference =
                                      actual)
```

```
### Save Bagging results to machine
write.csv(as.data.frame.matrix(confusion_bagging$table), file = "bagging_confusion.csv")
write.csv(as.data.frame.matrix(confusion_bagging$byClass), file = "bagging_stats.csv")
write.table(error_bagging, file = "bagging_statserror.csv")

### Save R workspace
save.image("/Users/Podkayne/Documents/YSU/Master's Thesis//
    DataResultsBAGGING_results_from_lab.RData")

### I added this part to round the values of confusion matrix and save a csv file
confusion_bagging$byClassRounded <- round(confusion_bagging$byClass, 2)
write.csv(as.data.frame.matrix(confusion_bagging$byClassRounded),file = "/Users/Podkayne/
    Documents/YSU/Master's Thesis/DataResults/ BAGGINGROUNDED.csv")



### Set c as a confusion matrix, previously calculated and stored
### (this confusion matrix has rows as predictions and columns as actuals):
c = as.matrix(as.data.frame.matrix(confusion_bagging$table))
### Initialize length 16 MCC vector:
MCC = vector(mode = "numeric", length = 9)
### For each location:
for(k in 1:9){
  A = as.double(c[k,k]) ### A = #(true positives)
  B = as.double(sum(c[k,]) - c[k,k]) ### B = #(false positives)
  C = as.double(sum(c[,k]) - c[k,k]) ### C = #(false negatives)
  D = as.double(sum(c) - A - B - C) ### D = #(true negatives)
  if((((A + B)*(A + C)*(D + B)*(D + C))^.5) == 0){ ### if denominator is zero, arbitrarily
       set as one - MCC will come out to be zero
    MCC[k] = (A*D) - (B*C)
  }
  else{MCC[k] = ((A*D) - (B*C))/(((A + B)*(A + C)*(D + B)*(D + C))^.5)} ### MCC value by
      formula
  rm(A,B,C,D)
}

### Rounding, setting as column vector:
MCC = round(matrix(MCC, nrow = 9, ncol = 1), digits = 2)
### Set appropriate column name:
colnames(MCC) = "MCC"
### Save results to machine (to be appended to pre-existing statistics by class table):
```

```
write.table(MCC, file = "/Users/Podkayne/Documents/YSU/Master's Thesis/DataResults/
    BAGGING_MCCs.csv", sep = ",", row.names = FALSE, col.names = TRUE)
rm(c, MCC)


set.seed(1234)
### Bagging with ten-fold cross-validation
### Create bagging with ten-fold cross-validation model using full dataset,
#using (50 trees in each "fold")*(10 "folds") = 500 total trees:
  bagging_cv = bagging.cv(formula = subcellular  ~ ., data = DATA, v = 10, mfinal =
                            50, control = rpart.control(maxdepth = 10))
### Variable importance plot (gives an error):
varImp_bagging_cv = barplot(bagging_cv$importance[order(bagging_cv$importance,
decreasing = TRUE) ], ylim = c(0,20), main = "Variables Relative Importance")

### Calculate error rate of classification:
### Save actual locations of full dataset:
ACTUAL = DATA[,1]
error_bagging_cv = 1 - sum(bagging_cv$class == ACTUAL)/nrow(DATA)
### Create confusion matrix and performance statistics for each location:
bagging_cv_class = as.factor(bagging_cv$class)
levels(bagging_cv_class) = c(levels(bagging_cv_class), levels(actual))
bagging_cv_class = factor(bagging_cv_class, levels(bagging_cv_class)[order(
  levels(bagging_cv_class))])
confusion_bagging_cv = confusionMatrix(data = bagging_cv_class, reference =
                                         ACTUAL)


### Save Bagging with ten-fold cross-validation results to machine
write.csv(as.data.frame.matrix(confusion_bagging_cv$table), file = "bagging_cv_confusion.
    csv")
write.csv(as.data.frame.matrix(confusion_bagging_cv$byClass), file = "bagging_cv_stats.csv
    ")
43
write.table(error_bagging_cv, file = "bagging_cv_statserror.csv")


### Save R workspace
save.image("/Users/Podkayne/Documents/YSU/Master's Thesis//
    DataResultsBAGGING_CV_results_from_lab.RData")
```

```
### I added this part to round the values of confusion matrix and save a csv file
confusion_bagging_cv$byClassRounded <- round(confusion_bagging_cv$byClass, 2)
write.csv(as.data.frame.matrix(confusion_bagging_cv$byClassRounded),file = "/Users/
    Podkayne/Documents/YSU/Master's Thesis/DataResults/ BAGGING_CVROUNDED.csv")


### Set c as a confusion matrix, previously calculated and stored
### (this confusion matrix has rows as predictions and columns as actuals):
c = as.matrix(as.data.frame.matrix(confusion_bagging_cv$table))
### Initialize length 16 MCC vector:
MCC = vector(mode = "numeric", length = 9)
### For each location:
for(k in 1:9){
  A = as.double(c[k,k]) ### A = #(true positives)
  B = as.double(sum(c[k,]) - c[k,k]) ### B = #(false positives)
  C = as.double(sum(c[,k]) - c[k,k]) ### C = #(false negatives)
  D = as.double(sum(c) - A - B - C) ### D = #(true negatives)
  if((((A + B)*(A + C)*(D + B)*(D + C))^.5) == 0){ ### if denominator is zero, arbitrarily
       set as one - MCC will come out to be zero
    MCC[k] = (A*D) - (B*C)
  }
  else{MCC[k] = ((A*D) - (B*C))/(((A + B)*(A + C)*(D + B)*(D + C))^.5)} ### MCC value by
      formula
  rm(A,B,C,D)
}

### Rounding, setting as column vector:
MCC = round(matrix(MCC, nrow = 9, ncol = 1), digits = 2)
### Set appropriate column name:
colnames(MCC) = "MCC"
### Save results to machine (to be appended to pre-existing statistics by class table):
write.table(MCC, file = "/Users/Podkayne/Documents/YSU/Master's Thesis/DataResults/
    BAGGING_CV_MCCs.csv", sep = ",", row.names = FALSE, col.names = TRUE)
rm(c, MCC)


##########################################################################################################

##########################################################################################################

### NEURAL NETWORK##########################################
```

```
#### NORMALIZING DATA


#install.packages("nnet")

library(nnet)
set.seed(1357)

### Take a sample of 70% of the numbers in 1 through the number of proteins in the dataset
    :
nums = sample(1:nrow(DATA), floor(.7*nrow(DATA)))
### (need to do this so models can know that each location is a "possibility"):
traindata = DATA[nums,]
while (0 %in% summary(traindata$subcellular) == TRUE){
  nums = sample(1:nrow(DATA), floor(.7*nrow(DATA)))
  traindata = DATA[nums,]
}
### Create 30% testing set:
testdata = DATA[-nums,]


### Centering both traindata and testdata
library(caret)

#removing the protein names since preProcess works only with numeric values
normtrain <- traindata[,-1]

#saving location names
subcellular2 <- traindata[,1]

procValues <- preProcess(normtrain, method = c("center", "scale"))
scaledtraindata <-  predict(procValues, normtrain)
traindatanorm <- cbind(subcellular2,scaledtraindata)

#removing the protein names since preProcess works only with numeric values
normtest<- testdata[,-1]

#saving location names
subcellular2 <- testdata[,1]
```

```
scaledtestdata <-  predict(procValues, normtest)
testdatanorm <- cbind(subcellular2,scaledtestdata)

#### Centering Done#############

###### FIXING LENGTH PROBLEM###################################
set.seed(1357)
#read data back in
DATA = read.csv("/Users/Podkayne/Documents/YSU/Master's Thesis/50.50paabench.csv", header
    = TRUE, row.names = 1)
keepvector = complete.cases(DATA)
DATA2 = DATA[keepvector,]  ## removed 5 NA's and have 2944 rows
subcellular2 = subcellular[keepvector]  ## re-sized the length by removing NA's
####### RUNNING NEURAL NETWORK#######################
set.seed(1357)
nnetmodel = nnet(subcellular2~., data = traindatanorm, size = 8, decay = .2,
                 linout = FALSE, entropy = TRUE, maxit=1000)
nnetmodel
nnetpred1 = predict(nnetmodel, newdata = traindatanorm, type = "class")
nnetpred2 = predict(nnetmodel, newdata = testdatanorm, type = "raw")
#this gives error for length
table(nnetpred1,testdatanorm$class)


### Save SVM results to machine
write.csv(as.data.frame.matrix(confusion_nnetmodel$table), file = "nnetmodel_confusion.csv
    ")
write.csv(as.data.frame.matrix(confusion_nnetmodel$byClass), file = "nnetmodel_stats.csv")
write.table(error_svms, file = "nnetmodelerror_stats.csv")
### Save R workspace
save.image("/Users/Podkayne/Documents/YSU/Master's Thesis//
    DataResultsNNETMODEL_results_from_lab.RData")

### I added this part to round the values of confusion matrix and save a csv file
confusion_nnetmodel$byClassRounded <- round(confusion_nnetmodel$byClass, 2)
write.csv(as.data.frame.matrix(confusion_nnetmodel$byClassRounded),file = "/Users/Podkayne
    /Documents/YSU/Master's Thesis/DataResults/ NNETMODELROUNDED.csv")

### Set c as a confusion matrix, previously calculated and stored
### (this confusion matrix has rows as predictions and columns as actuals):
c = as.matrix(as.data.frame.matrix(confusion_nnetmodel$table))
```

```
### Initialize length 16 MCC vector:
MCC = vector(mode = "numeric", length = 9)
### For each location:
for(k in 1:9){
  A = as.double(c[k,k]) ### A = #(true positives)
  B = as.double(sum(c[k,]) - c[k,k]) ### B = #(false positives)
  C = as.double(sum(c[,k]) - c[k,k]) ### C = #(false negatives)
  D = as.double(sum(c) - A - B - C) ### D = #(true negatives)
  if((((A + B)*(A + C)*(D + B)*(D + C))^.5) == 0){ ### if denominator is zero, arbitrarily
       set as one - MCC will come out to be zero
    MCC[k] = (A*D) - (B*C)
  }
  else{MCC[k] = ((A*D) - (B*C))/(((A + B)*(A + C)*(D + B)*(D + C))^.5)} ### MCC value by
      formula
  rm(A,B,C,D)
}

### Rounding, setting as column vector:
MCC = round(matrix(MCC, nrow = 9, ncol = 1), digits = 2)
### Set appropriate column name:
colnames(MCC) = "MCC"
### Save results to machine (to be appended to pre-existing statistics by class table):
write.table(MCC, file = "/Users/Podkayne/Documents/YSU/Master's Thesis/DataResults/
    NNETMODEL_MCCs.csv", sep = ",", row.names = FALSE, col.names = TRUE)
rm(c, MCC)



########## MY SECOND TRY WITH A DIFFERENT APPROACH
set.seed(1357)
nnetmodel = nnet(subcellular2~., data = traindatanorm, size =2, decay = 15e-4,
                 linout = FALSE, entropy = TRUE, maxit=1000)
## Use TEST data for testing the trained model
test.nnet<-predict(nnetmodel,testdatanorm,type=("class"))
## MisClassification Confusion Matrix
table(testdatanorm$subcellular2,test.nnet)
## One can maximize the Accuracy by changing the "size" while training the neural network.
     SIZE refers to the number of nodes in the hidden layer.
which.is.max(test.nnet)  ## To Fine which row break ties at random (Maximum position in
    vector)

table(test.nnet,subcellular2)
```

```
#### CODE TO GET ERROR RATE
actual = testdatanorm[,1]
### Calculate error rate of classification:1-percentage of right= percentage of wrong
error_nnetmodel = 1 - sum(test.nnet == actual)/nrow(testdatanorm)
```