# Protein Classification

Christopher Mertin  Sam Leventhal

December 19, 2015

## 1 Introduction

The protein classification problem is an ongoing and heavily researched topic in biology, chemistry, pharmacy, and many other academic fields. The wide spread importance of protein research lies in the fact that proteins are fundamental building blocks of organic life which orchestrate the transfer of biological information and as a result govern reproduction, replication, defence, reconstruction, and operation from micro to macro organisms. Knowing which family a protein belongs to provides insights on the behaviors and capabilities a protein exhibits. With new proteins constantly identified and vast holes in our understanding of protein behavior the ability to accurately and efficiently identify protein families allows researchers to explain unknown mechanisms and find new application of known proteins as well as develop or identify capabilities in new proteins.

In order to do this problem, we had to parse many files from the world wide protein database and RCSB [1, 2]. These files were not in the best file format, so even though there are more than 60,000 protein families, only less than 2,000 families were useful. Figure 1a shows the distribution of these $\sim 2000$ families, from which it can easily be seen that it's not a uniform distribution.

Therefore, when training over the data, we opted to only choose the five most common protein families. The choosing of the top five most common was just an arbitrary number that was chosen. In this paper, we explore 3 different avenues for classifying these proteins, which are described below

### 1.1 Feature Vectors

All of the methods that we used required feature vectors to be able to classify the data. Most amino acid chains of proteins are represented in the form

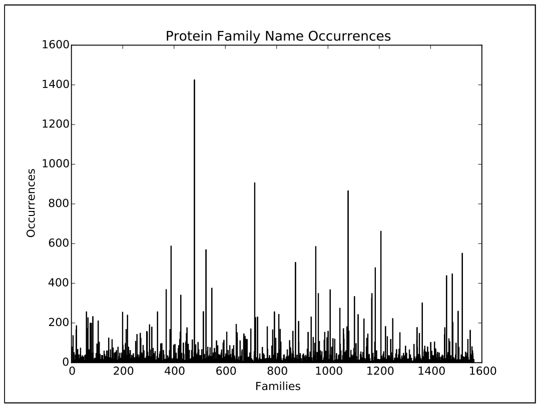
```
ABDRBHCED
```

where we had to create a feature vector from this. To do so, we first created a dimension for every possible amino acid, an empty vector of zeros. Then, to build the feature vector for every protein, what we did was used the number of "counts" that each letter appeared in the chain. *i.e.* in our above example, `A` shows up once, so the feature `A` in our feature vector would have been given a value of 1. `B` shows up twice, so that feature would have a value of 2. These proteins are composed of usually hundreds of amino acids, which would make this very complex.

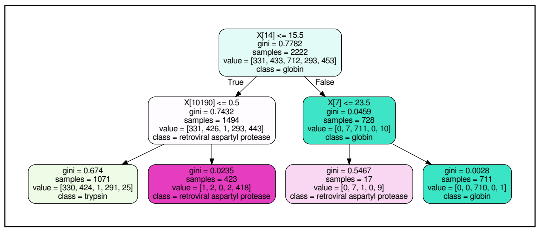
We continued this trend up through 3 combinations of the letters. For example, we checked: `AA`, `AB`, `AC`,. . . , `ZZ`, `AAA`,. . . , `ZZZ`, and made each one a feature in our vector. This resulted in a feature vector if size 12719, which was quite large for our data set, where most of the data was zero since there were so many combinations.

We also created a "baseline" for our machine learning algorithms, where we took the most common label and saw what accuracy we would get if we simply *guessed* that label for all the data. The following is the output from that instance.

```
Size of feature vector: 12,719
Number of families: 5
Baseline Training Accuracy: 0.32043
Baseline Testing Accuracy:  0.32088
```



(a) Distribution of Families



(b) Decision Tree of level 2

Figure 1: Distribution of the protein families and the decision tree forced to a level of 2.

## 2 Multi-class Classification with SVM's

The method that we used for this section was a one-versus-all method. One-versus-all builds a weight vector for each classification, and builds these weight vectors by simply treating all labels in that class as positive, and the rest as negative. Other algorithms can do this, but this was the easiest to attempt to implement.

However, we ran into problems when attempting to implement this data, so we opted to use the one-versus all algorithm from `scikit-learn`, which we were told we were allowed to use as long as we weren't "blindly" plugging our data into it and using it. We looked at the source code and the algorithm to get a general idea on *exactly* how it works, and it is a normal SVM for each weight vector in the code.

### 2.1 Results

One of the main reasons why we wanted to use a Mutli-class SVM, is because when initially testing our data with the decision tree, we were getting very high accuracies and we wanted to see if it was a bug in our code. The decision tree algorithm results and implementation can be seen in Section 3. However, even though we used this Multi-class Classification method, we also got an absurdly high accuracy for this method as well. One downside from using this library is that we were not able to see the parameters that the classifier used, which could have been helpful in watching how it learned. However, we did tell it to use cross validation over the training set, which from the results, seems to have been successful.

```
One vs All Results
Learned with an accuracy of: 0.99640
Tested with an accuracy of:  0.99415
```

## 3 Decision Trees

We implemented a standard decision tree algorithm which minimized the Gini Coefficient for our data, much like we did for the first homework assignment. The results are discussed below.

### 3.1 Results

Initially, when building our decision tree, we got very accurate results, which can be seen below. So what we opted to do was to force our decision tree to only be a height of 2 and see what the accuracy was as well, to see if our results would change. The resulting decision tree of height 2 can be seen in Figure 1b, while the final, unrestricted, decision tree can be seen in Figure 2. In these figures, you can see the Gini coefficients at each step, and also see the labels that were used. In the figures, the `values` array represents the number of values in each class that are at that node, while $X[14] \leq 15.5$ is the "break point" for that node, such that it will go to the left if true and the right if false, where it's looking at the 14$^{th}$ index of the feature vector.

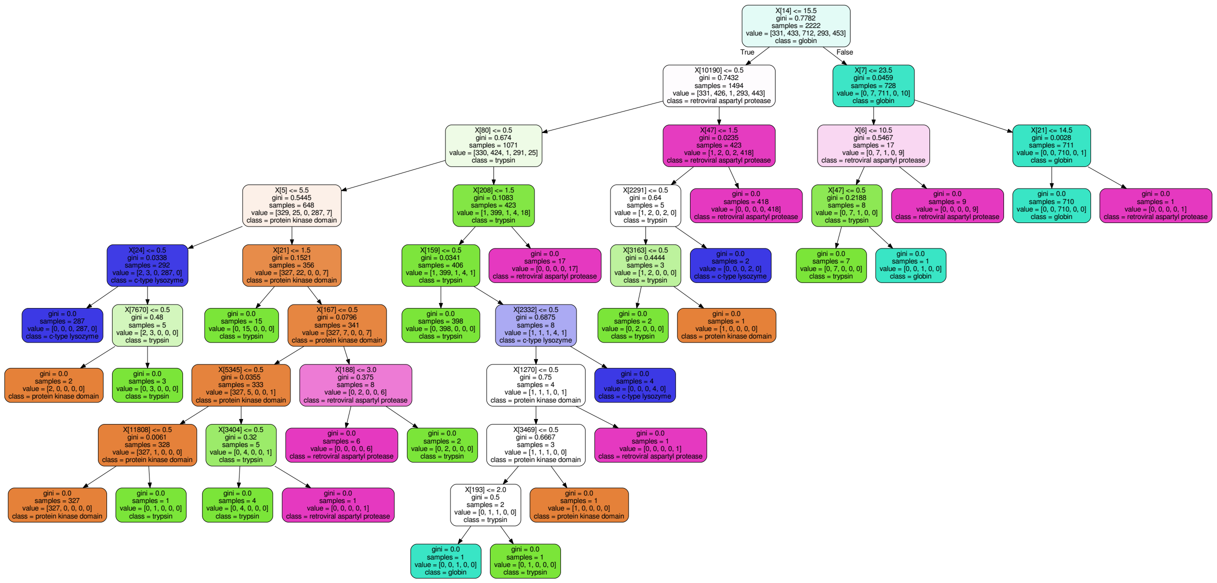

X[14] <= 15.5 gini = 0.7782 samples = 2222 value = [331, 433, 712, 293, 453] class = globin
True
False
X[10190] <= 0.5 gini = 0.7432 samples = 1494 value = [331, 426, 1, 293, 443] class = retroviral aspartyl protease
X[7] <= 23.5 gini = 0.0459 samples = 728 value = [0, 7, 711, 0, 10] class = globin
X[80] <= 0.5 gini = 0.674 samples = 1071 value = [330, 424, 1, 291, 25] class = trypsin
X[47] <= 1.5 gini = 0.0235 samples = 423 value = [1, 2, 0, 2, 418] class = retroviral aspartyl protease
X[6] <= 10.5 gini = 0.5467 samples = 17 value = [0, 7, 1, 0, 9] class = retroviral aspartyl protease
X[21] <= 14.5 gini = 0.0028 samples = 711 value = [0, 0, 710, 0, 1] class = globin
X[5] <= 5.5 gini = 0.5445 samples = 648 value = [329, 25, 0, 287, 7] class = protein kinase domain
X[208] <= 1.5 gini = 0.1083 samples = 423 value = [1, 399, 1, 4, 18] class = trypsin
X[2291] <= 0.5 gini = 0.64 samples = 5 value = [1, 2, 0, 2, 0] class = trypsin
gini = 0.0 samples = 418 value = [0, 0, 0, 0, 418] class = retroviral aspartyl protease
X[47] <= 0.5 gini = 0.2188 samples = 8 value = [0, 7, 1, 0, 0] class = trypsin
gini = 0.0 samples = 9 value = [0, 0, 0, 0, 9] class = retroviral aspartyl protease
gini = 0.0 samples = 710 value = [0, 0, 710, 0, 0] class = globin
gini = 0.0 samples = 1 value = [0, 0, 0, 0, 1] class = retroviral aspartyl protease
X[24] <= 0.5 gini = 0.0338 samples = 292 value = [2, 3, 0, 287, 0] class = c-type lysozyme
X[21] <= 1.5 gini = 0.1521 samples = 356 value = [327, 22, 0, 0, 7] class = protein kinase domain
X[159] <= 0.5 gini = 0.0341 samples = 406 value = [1, 399, 1, 4, 1] class = trypsin
gini = 0.0 samples = 17 value = [0, 0, 0, 0, 17] class = retroviral aspartyl protease
X[3163] <= 0.5 gini = 0.4444 samples = 3 value = [1, 2, 0, 0, 0] class = trypsin
gini = 0.0 samples = 2 value = [0, 0, 0, 2, 0] class = c-type lysozyme
gini = 0.0 samples = 7 value = [0, 7, 0, 0, 0] class = trypsin
gini = 0.0 samples = 1 value = [0, 0, 1, 0, 0] class = globin
gini = 0.0 samples = 287 value = [0, 0, 0, 287, 0] class = c-type lysozyme
X[7670] <= 0.5 gini = 0.48 samples = 5 value = [2, 3, 0, 0, 0] class = trypsin
gini = 0.0 samples = 15 value = [0, 15, 0, 0, 0] class = trypsin
X[167] <= 0.5 gini = 0.0796 samples = 341 value = [327, 7, 0, 0, 7] class = protein kinase domain
gini = 0.0 samples = 398 value = [0, 398, 0, 0, 0] class = trypsin
X[2332] <= 0.5 gini = 0.6875 samples = 8 value = [1, 1, 1, 4, 1] class = c-type lysozyme
gini = 0.0 samples = 2 value = [0, 2, 0, 0, 0] class = trypsin
gini = 0.0 samples = 1 value = [1, 0, 0, 0, 0] class = protein kinase domain
gini = 0.0 samples = 2 value = [2, 0, 0, 0, 0] class = protein kinase domain
gini = 0.0 samples = 3 value = [0, 3, 0, 0, 0] class = trypsin
X[5345] <= 0.5 gini = 0.0355 samples = 333 value = [327, 5, 0, 0, 1] class = protein kinase domain
X[188] <= 3.0 gini = 0.375 samples = 8 value = [0, 2, 0, 0, 6] class = retroviral aspartyl protease
X[1270] <= 0.5 gini = 0.75 samples = 4 value = [1, 1, 1, 0, 1] class = protein kinase domain
gini = 0.0 samples = 4 value = [0, 0, 0, 4, 0] class = c-type lysozyme
X[11808] <= 0.5 gini = 0.0061 samples = 328 value = [327, 1, 0, 0, 0] class = protein kinase domain
X[3404] <= 0.5 gini = 0.32 samples = 5 value = [0, 4, 0, 0, 1] class = trypsin
gini = 0.0 samples = 6 value = [0, 0, 0, 0, 6] class = retroviral aspartyl protease
gini = 0.0 samples = 2 value = [0, 2, 0, 0, 0] class = trypsin
X[3469] <= 0.5 gini = 0.6667 samples = 3 value = [1, 1, 1, 0, 0] class = protein kinase domain
gini = 0.0 samples = 1 value = [0, 0, 0, 0, 1] class = retroviral aspartyl protease
gini = 0.0 samples = 327 value = [327, 0, 0, 0, 0] class = protein kinase domain
gini = 0.0 samples = 1 value = [0, 1, 0, 0, 0] class = trypsin
gini = 0.0 samples = 4 value = [0, 4, 0, 0, 0] class = trypsin
gini = 0.0 samples = 1 value = [0, 0, 0, 0, 1] class = retroviral aspartyl protease
X[193] <= 2.0 gini = 0.5 samples = 2 value = [0, 1, 1, 0, 0] class = trypsin
gini = 0.0 samples = 1 value = [1, 0, 0, 0, 0] class = protein kinase domain
gini = 0.0 samples = 1 value = [0, 0, 1, 0, 0] class = globin
gini = 0.0 samples = 1 value = [0, 1, 0, 0, 0] class = trypsin

Figure 2: Resulting Decision Tree (zoom in to see)

```
Decision Tree (height = 2)
Trained with an accuracy of: 0.70252
Tested with an accuracy of:  0.69649
Decision Tree (no height restrictions)
Trained with an accuracy of: 1.00000
Tested with an accuracy of:  0.99236
```

## 4 Convolutional Neural Net Feature for Transfer Learning

Much effort has gone into the development of 3D representations of proteins. From these images domains and topologies can be identified. The development of image processing offers these vast data banks as new means for protein classification.

Our approach was a feed forward method which uses layers within a pre-trained convolutional neural net as input for other learning algorithms as previously discussed such as support vector machines. Our feature representations for the training a classifier are vectors of sigmoid activation values associated to various layer of the ConNet. Layers of interest were those following 2 dimensional convolutions, max pooling, heavily connected dense layers with drop out, and the last output layer. Our choice of classier were support vector machines.

The model used for our convolutional neural net is symbolized in Figure 3

In order to capture all structures within a protein we wrote a jmol script to manipulate and record five perspectives of each protein: front view, top view, bottom view, left view, and right view. Our CNN each image of size $516 \times 639$ as input and first performs a convolution with 96 filters of size $5 \times 5$ followed by two convolutions with 96 filters of size $3 \times 3$. Our incentive for decreasing filter size was in order to preserve attributes and hopefully correlations between structures (such as domains) within the image. This constitutes the first layer where we extract a vector feature representation containing values of the activation function. Notice the massive decrease in size of our feature vector as we feed forward in the CNN.

It was our goal to exploit the convolutional neural network's ability to exaggerate different structural properties at different layers. Even between one convolution and pooling we can see separate and informative features.

Unfortunately, due to lengthy computational time and heavy memory consumption we were not able to allocate the amount of time we had hoped for in order to treat the number as well as combinations of

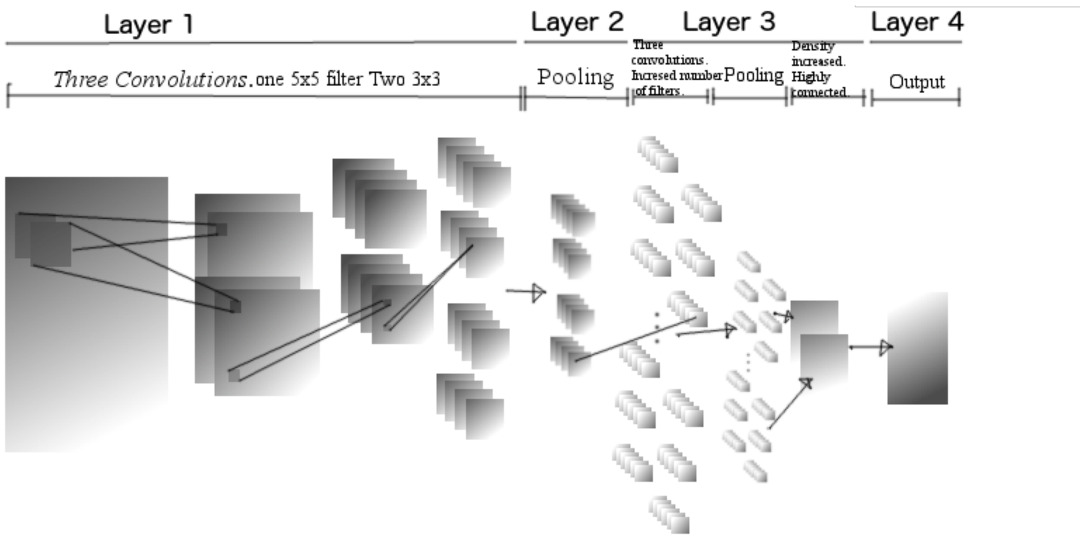

Layer 1
Layer 2
Layer 3
Layer 4
Three Convolutions.one 5x5 filter Two 3x3
Pooling
Three convolutions. Incresed number of filters.
Pooling
Density increased. Highly connected.
Output

Figure 3: Feature Layers and ConNet Model

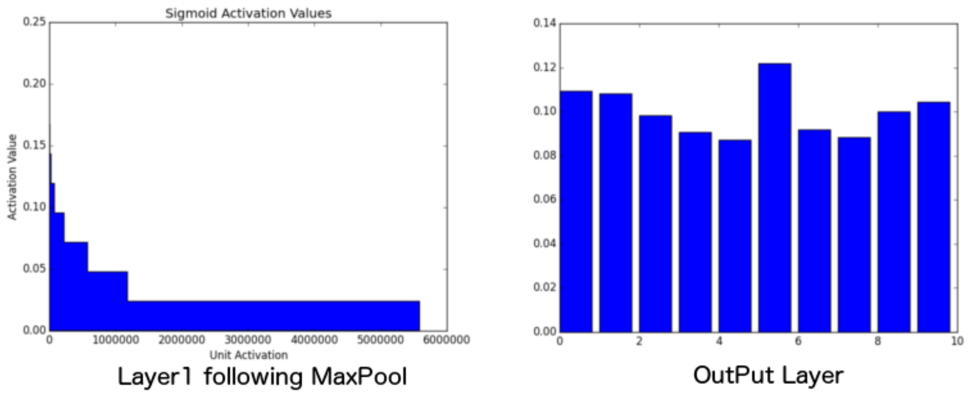


Figure 4: Sigmoid Activation. Left: Three convolutions and max-pooling Right: Output following dropout

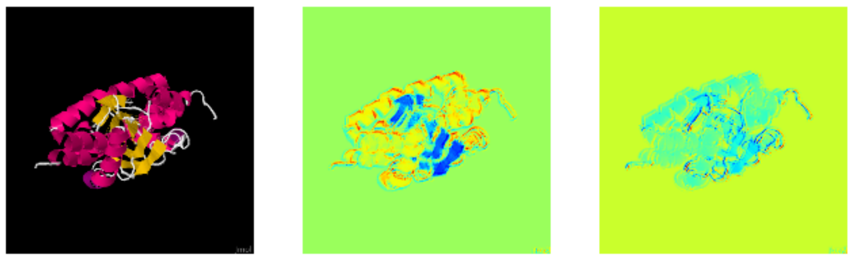


Figure 5: 2D convolution and pooling on protein 4EAR

convolutions, maxpooling, and hidden layers as hyperparameters. Though it may not be the most accurate due to the small sample size we were able to use 4-fold cross validation between random sets of 12 images. In this manner we were able to estimate the CNN model which we are now using as more likely to generate feature vectors more accurately classified by our SVM.

On the second occurrence a multinomial logistic regression is performed as a function of linear and non-linear transformations. By performing gradient decent we were then able to optimize our hyper-parameter bias and learning rate within our nueral net's back propagation.

| Transformation | size |
|---|---|
| First Convolution | 5x5 |
| Second Convolution | 3x3 |
| Third Convolution | 3x3 |
| MaxPool | 2x2 |
| Fourth Convolution | 3x3 |
| Fifth Convolution | 3x3 |
| Sixth Convolution | 3x3 |
| MaxPool | 2x2 |
| Training Set Size | Accuracy |
| 120 | 2.1% |
| 100 | 0% |

Notice how the accuracy of our implementation decreases dramatically as the size of our training set decreases. This is due to the fact that with a smaller training set we dramatically reduce the number of known labels. As a result when training we have an extremely high probability of attempting to label a data point whose labeling is unknown and has never been trained with.

We chose to train with support vector machines due to their being well studied in the one vs all and all vs all case interpretation for multi-class labeling. What is more support vector machines handle large arrays as examples well due to being able to form hyper-planes separating higher dimensional data. What is more, SVM's perform well with multi-class classification that the intersection of hyper-planes between classes form region corresponding to a label within the higher dimensional space.

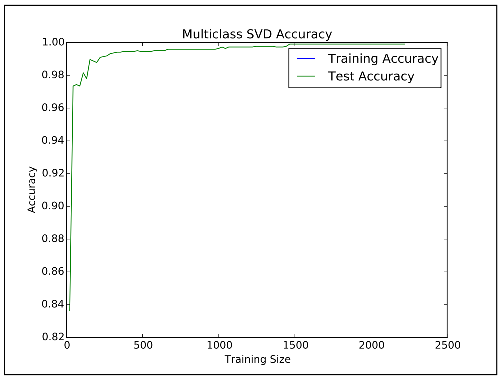
The non-linearity of neural nets is an obvious indicator that optimizing the hyper parameters is not sufficient for improving classification success. Though not done here it would be beneficial to parametrise the weight values used rather than allowing the weights to only update depending on observed data. Parametrization of the weights would require an assumed prior which imposes some dependence between weights as apposed to the current assumption of independent. We could then approximate a posterior distribution relating weights and hyper parameters. Investigating potential posterior distributions would be a novel direction of study as we introduce in the next section.

To preserve time we record feature representations after having initialized our neural net for a given example. By building upon our feature record during training it allows for an interesting method of testing and further training. Since our data set is so large, essentially infinite, we almost have a hold out set to train on. We are therefore able to train our model and then test on a hold out set randomly taken from the total data. During training we also update our training set with new examples and unseen labels. This saves time in that reformatting and transformations such as convolution over images is only performed on the smaller hold out set. By updating our learned features with the hold out set we are therefore training our model during testing yet not overfitting. In effect, the massive size of data and labels allows us to turn out neural net into an on-line algorithm that updates as it sees new examples.

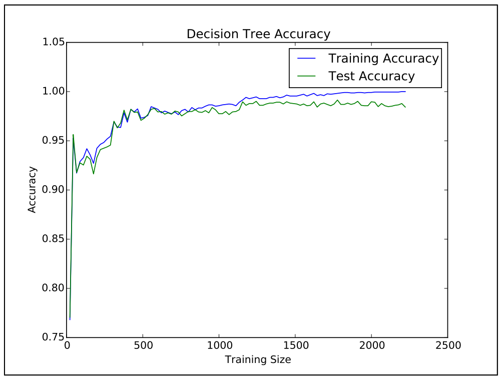
## 5 Expectation Minimization

The abundance of data and labels is a glaring obstacle in protein classification. Future directions must therefore aim to exploit techniques which lower the risk of misclassification despite being forced to train on small subsets of examples as well as the having numerous unobserved labels.

One known method of training resulting in a lower probability of error would be optimizing weights and hyper parameters based on expectation minimization (EM). Our aim would be to train hyper-parameters as



(a) Training and Test Error for SVM



(b) Training and Test Error for Decision Tree

Figure 6: Training and Test Error for both the Multiclass SVM and the Decision Tree. Note, in (a), the training accuracy was 100%.

well as develop a methodology for choosing a sub-sample of the data for training with confidence our sub-sample accurately represents the data as a whole. As required by the EM assumptions we must rationalize a probability distribution over the data such that the probability of observing some unknown example coincides with the maximum likelihood associated to our observed data. From our work it would be possible to calculate the maximum likelihood of unseen examples through the logistic loss function used within the CNN as a function over all linear and non-linear transformations between layers. However it may prove more fruitful to study the probability of domains, their combinations, and their structures to occur.

## 6 Conclusion

Finally, as can be seen in Figure 6, we took the training and test error for the decision tree and for the multi-class SVM, and plotted their training and testing accuracy. For this set, we started with a low number of training examples $\sim 20$, and worked our way up by increments of 20 until we got to the maximum size. As you can see in the figures, they start off with low accuracy, but it grows rapidly over the entire data set.

We are unsure of why the accuracy of these methods is so high. We checked to make sure that we were not training on the test set, and a few other things such as making sure that the labels were not in our features. We analyzed the distributions of classifications on the training and test sets, and they were almost exactly 50%. One of the guesses that we have is that there was something that happened in parsing the data from [1, 2], but by randomly sampling the files, this doesn't seem to be the case.

With more time, we would have tried to figure out *why* these classification rates are so high, and if it was just that we got lucky with the protein families that we had chosen and they were extremely separable. We would have also liked to attempt to implement a Hidden Markov Model (HMM) into the mix to see how that performs, as that is what is used in building some of these databases, but the use of the HMM requires you to "align" the amino chains, which is something that after hours of research we couldn't figure out what it meant. Therefore, with more time, we may have been able to find someone with enough domain knowledge who could help describe these to us.

The main thing that we learned from this project is that data is *invaluable* and it can make or break your project. Sadly, we do not know how accurate our project truly is, but we learned a lot about ways to transform your data into features which can be learnable, even if on the surface it doesn't look like it.

## References

[1] World Wide Protein Data Bank, `http://www.wwpdb.org/`

[2] Research Collaboratory for Structural Bioinformatics Protein Data Bank, `http://www.rcsb.org`