[[1]](#footnote-1)

Credit Card Fraud Detection (May 2017)

Sho Miyauchi, Jiangzhou Wang

*Abstract*—Credit card fraud refers to a form of identity theft that takes advantage of an unauthorized taking of another person’s credit card. Given a person’s credit card transaction we attempt to classify this between fraud and non-fraud. Several models including, logistic regression, support vector machine, neural network, k-means, and fuzzy c-means was used for the models.

# INTRODUCTION

C

REDIT card fraud refers to a form of identity theft that takes advantage of an unauthorized taking of another person's credit card in order to charge purchases to the person's account or remove funds from it. Some examples of when this is committed is when a person illegally obtains, uses, buys, or forges someone else's credit card information. Or when someone uses his or her card with knowledge that it is revoked or expired to pay for items. Another example is when someone charges another person with the knowledge that the other's credit card was illegally obtained or used without authorization.

And nowadays, enterprises and public institutions have to face a growing presence of frauds and consequently need automatic systems able to support fraud detection and fight. These systems are essential since it is not always easy for a human analyst to detect fraudulent patterns in transaction datasets, often characterized by a large number of samples.

# Task Description

Our task is to design a model that is capable of classifying between a fraud credit card and a non-fraud credit card given a set of features regarding a person's transaction. We will design, assess, and validate several machine models in order to select the optimum model to complete this task. The models to be designed and assessed are logistic regression, support vector machine, neural network. We will also attempt to cluster the dataset to try and get a better understanding of the problem through unsupervised learning algorithms such as k-means, and fuzzy c-means. Our ultimate goal in performing this task is to form a better understanding of several machine learning practices, methods, and algorithms.

# DATASET

The dataset utilized contains transactions made by credit cards in September 2013 by European cardholders. This dataset presents transactions that occurred in two days, where we have (492) frauds out of (284,807) transactions.

The features in our dataset contain only numerical input variables. This was a result of PCA transformation that was used in order to reduce the dimensions of this large dataset. Theses features are labeled as V1, V2, ... V28. Two more features which were not transformed through PCA transformation were included. These features were Time and Amount. 'Time' refers to the seconds elapsed between each transaction and 'Amount' refers to the transaction amount.

We faced some challenges when using this dataset. First, is this dataset is highly unbalanced. The positive class (frauds) account for only 0.172% of all transactions. Second, because credit card features contain private and sensitive information about the credit card holder, background information about these features were not provided. They were merely numerical features labeled as V1 to V28.

# Preprocessing

We took many approaches to preprocess our dataset. As mentioned earlier, we faced a challenge of a highly unbalanced dataset. To solve this issue, one approach we made was, when we divided our dataset into training and testing set, we wanted to ensure that both sets had an even ratio of fraud classes and non fraud classes. This is because we felt the unbalanced nature of the dataset was significant to our problem. The division was made so the training set consisted of 2/3 of our data, whereas the testing set consisted of the remaining 1/3. To maintain this unbalance, we randomly selected 2/3 of the non fraud classes, and 2/3 of the fraud classes and appended them to produce the training dataset. The remainder was used for the testing set.

Another approach we took was to shuffle the two datasets as before this step, all of the non-fraud classes were indexed at the top of the dataset, where as the fraud was at the bottom. Shuffling ensured there was a random order in these sets.

One last approach we took to preprocess our dataset was reducing the number of features. Luckily our dataset has already gone through PCA transformation to reduce the dimension of our dataset. However, two features, 'Time' and 'Amount' has not gone through this transformation. In order to maintain consistency with the rest of the features we have reduced these two features when we were training and testing our models.

Another form of feature selection we have attempted was reducing features manually through visual analysis. For each feature we graphed the value of that feature with its corresponding classes. This was done in order to see if a specific features’ value had direct impact on whether a sample was fraud or non fraud.

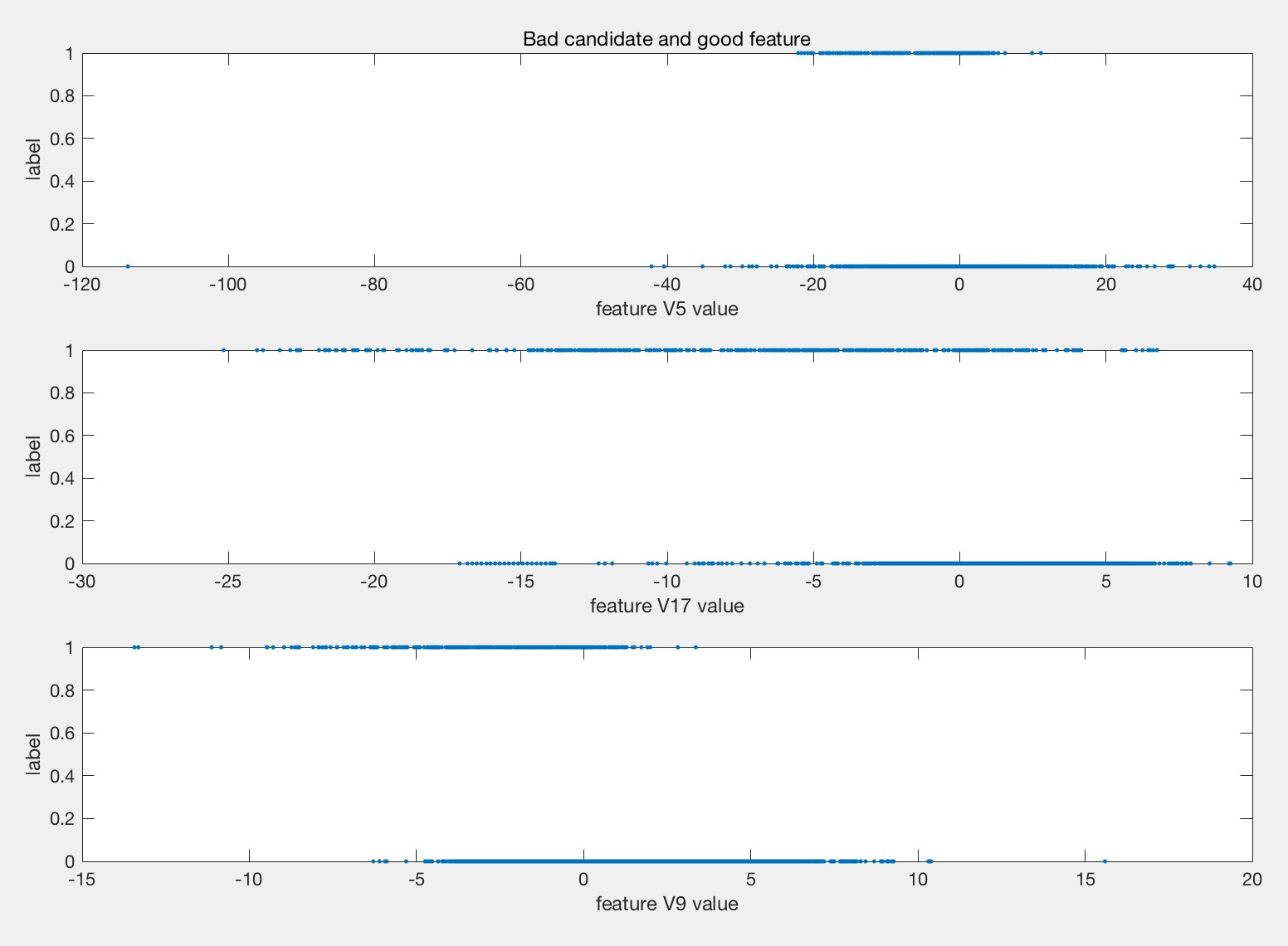


Fig. 1. Bad Candidate And Good Features

To illustrate, in this example we can see that feature V9 is a good selection. Clearly, all of the samples with label 1 (fraud) have a low value of feature V9, whereas all of the samples with label 0 (non-fraud) have high values of feature V9. From this graph we can see that the value of this feature, whether it be low or high value, has a significant impact on what the class is. In contrast to this, we can see that feature V5 is a bad selection. There is no clear divide between label 0 and label 1 in correlation to the value of the feature. Both labels 0 and 1 have a high value of this feature, so we can see that the value of this feature is not significant in classifying this sample. If a feature has slight significance in classifying a label, as in the example of feature V17, we have classified it as a possible candidate in the feature selection. In this example of feature 17, we can see that if the feature is of low value, it has slightly more label 1s, than label 0.

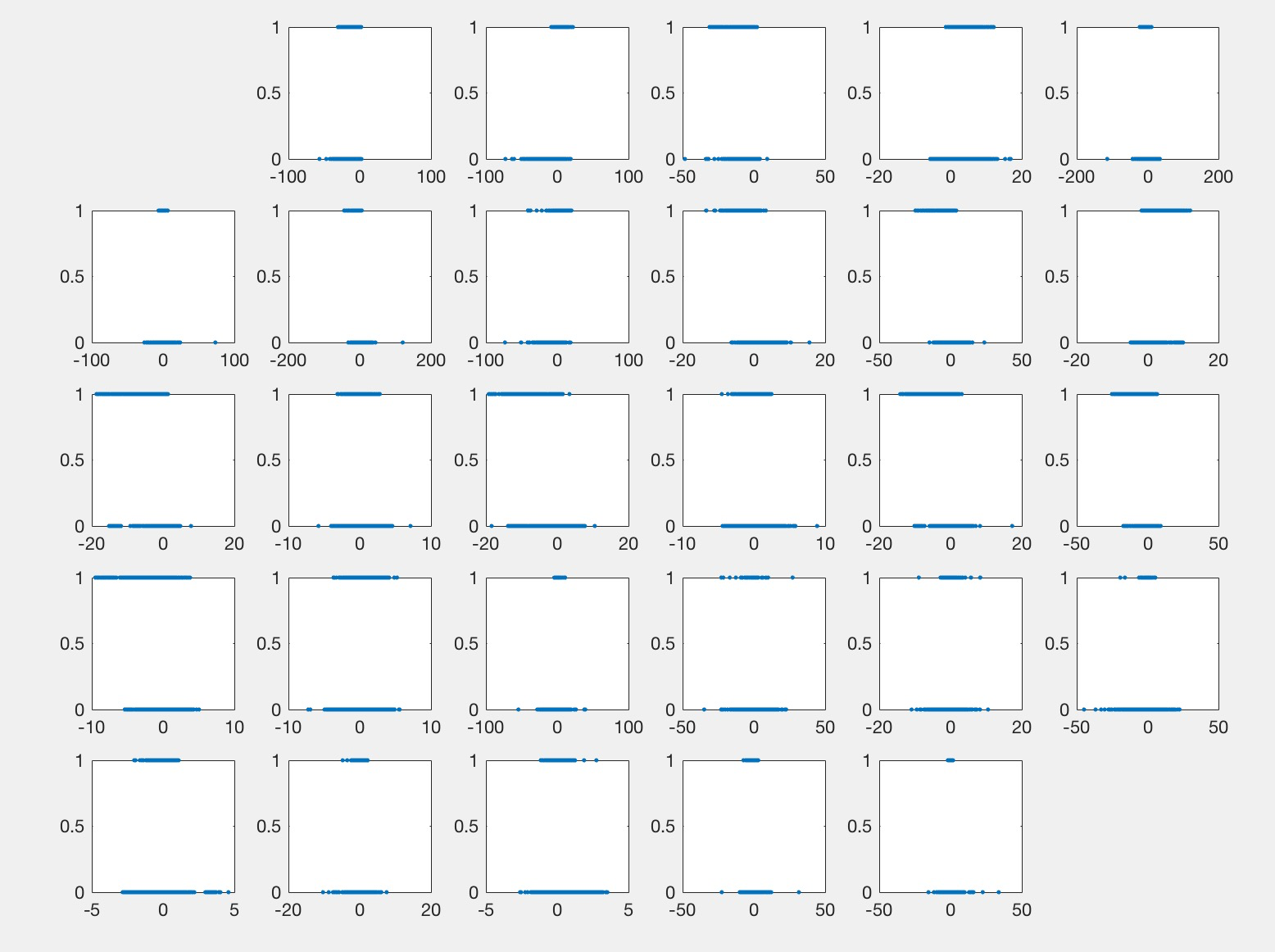
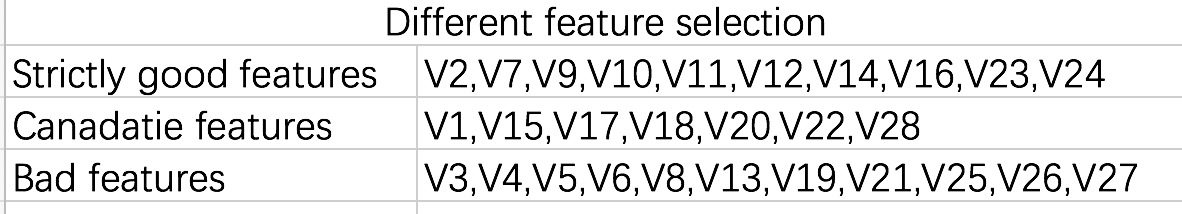


Fig. 2. All Features Values vs Labels

After dividing these features into bad features, candidate features, and good features we tested each model with the following. First, we tested using all every feature (features V1-V28). Then tested using possible candidate features plus good features. And last we tested using strictly the good features.

TABLE 1

FEATURE SELECTION



# Testing and Evaluation

In order to test each of our models the following procedures will be followed. First, we will train each of our models using the training dataset we have generated during our preprocessing step. This testing will be done with three different set of features. First, we will train using all features, followed by training with candidate features plus good features, and lastly we will train using strictly good features. After the model has been trained, we will then test our models using our testing dataset with its respected selected features. From this test we will evaluate each our models through the use of a confusion matrix. From this matrix we will derive four different metrics. We will calculate the error rate, the accuracy, the per class recall rate, and lastly the per class precision rate.

# Models

## Logistic Regressions

### *Implementation*

We have implemented logistic regression through two ways. First we used the matlab function of glmfit. Second, we attempted to create our own function for logistic regression manually in matlab. This was done as logistic regression was limited in the number of parameters we were able to change. We also coded our own logistic regression as the method in glmfit is slightly different in the algorithm to what we have learned in class. In order to test for the optimum threshold we also implemented a way to test through thresholds ranging from 0.001 to 0.999 with increments of 0.001 each iteration. After finding the optimum threshold, this is what we used to test each of the different set of features selected earlier.

### *Change in Threshold*

We wanted to test different parameters to test different results with our model. The parameter chosen for this model was to change the threshold which decides between a fraud class and a non-fraud class. Below is the graph measuring different thresholds ranging from 0.001 to 0.999 with its corresponding error. From this graph we can conclude that the optimum threshold for our problem would be around 0.2. We assume this is the case as our dataset is highly unbalanced with an overwhelming amount of non-fraud classes in contrast to fraud classes. Because there are such a small number of fraud (0.172%), we must adjust the threshold to a much more lenient value of 0.2 to accurately classify these few fraud classes. In contrast to this, it is much more strict in classifying a non-fraud class as the sample must be lower than 0.2 to be classified as non-fraud. Again, this is done as there is significantly more non-fraud classes than fraud classes.

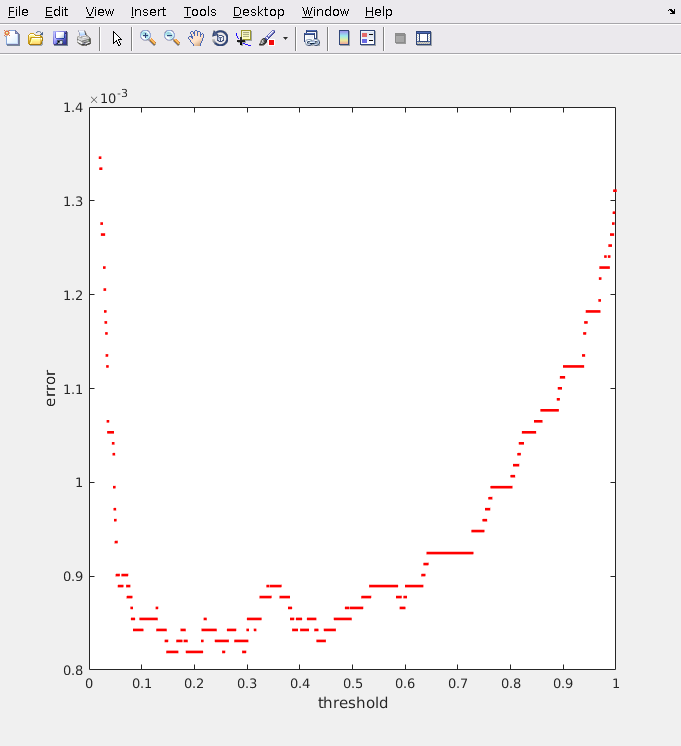
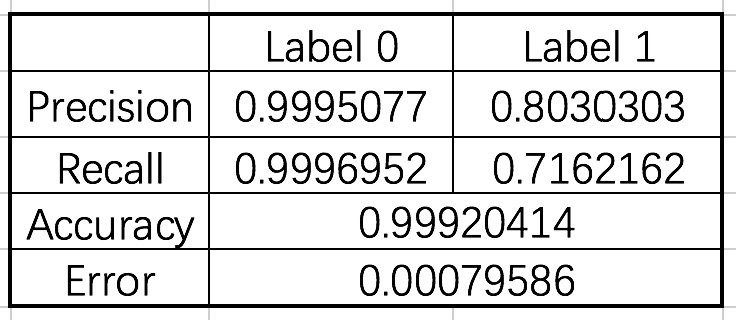


Fig. 3. Threshold vs Error

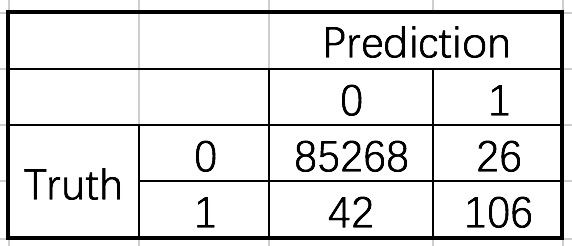
### Evaluation

TABLE 2

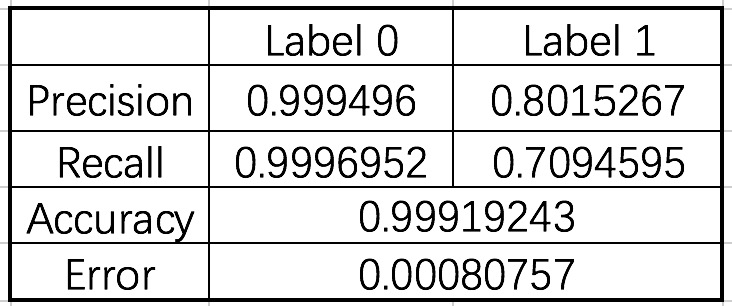
ALL FEATURE STATS



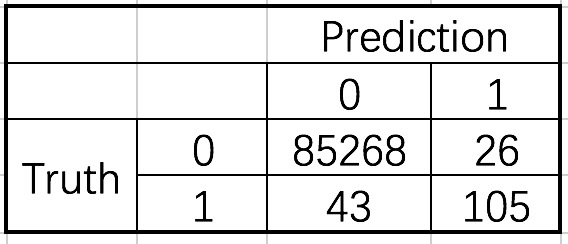
ALL FEATURE CONFUSION MATRIX



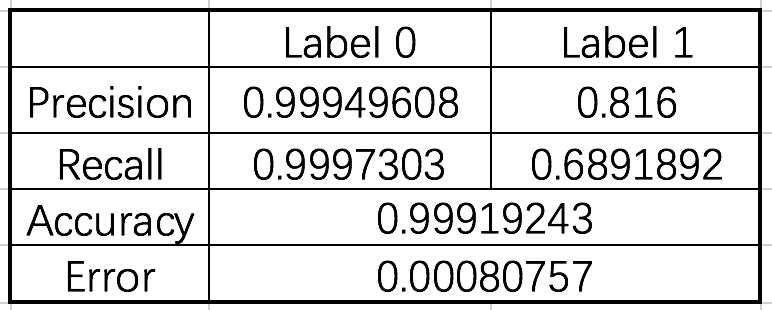
GOOD AND CANDIDATE FEATURES STATS



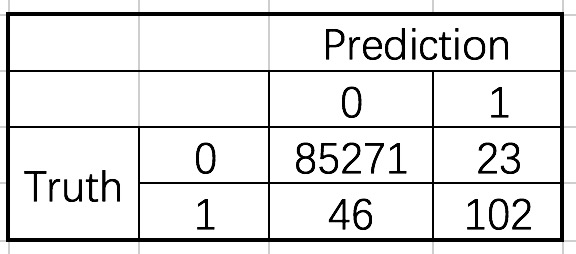
GOOD AND CANADIATE FEATURES CONFUSION MATRIX



STRICTLY GOOD REATURES STATS



STRICTLY GOOD FEATURES CONFUSION MATRIX



As 0.2 produced the most optimum results, this is what we used to test the three selected feature sets. The above set of tables shows the confusion matrix, accuracy, error, per class precision, and per class recall of each set of features with our testing set. From this data, we can deduce that using a different set of selected features, unfortunately did not improve the accuracy of our model. However, one important point to make is that with the good feature set there includes only 11 features whereas using all features would include up to 29. This means we were able to reduce the features to about ⅓ of the size yet produce very accurate results with these features. This would mean we were able to increase efficiency in terms of processing yet able to create accurate results.

### *Challenges and Solutions*

One challenge we faced in this model is our inability to adjust several parameters with glmfit. To solve this issue we attempted to produce our own logistic regression function. The goal of implementing our own was to be able to set our own learning rate and our desired maximum iteration. This way we would have more control over our training as well as have a better understanding of how the algorithm functions. However, our implementation proved to be too slow to process all of our data. We assume this is the case as we implemented the algorithm using several for loops rather than through vectors and matrices. One solution for this is for future implementations, try to take advantage of vector and matrix manipulation rather than relying on these loops.

## Logistic Regressions

### *Implementation*

Initially, the svm model was implemented manually through code from previous coding assignments. However, this failed to work for reasons which will be discussed later in this section. As an alternative we have implemented the matlab function of fitcsvm. With this model we implemented three different kernels to use with svm. The linear kernel, rbf kernel, and polynomial kernel. To find optimum values of the rbf kernel, we tested several different gammas to get the best result. After testing the best kernel out of the three, the one with the lowest error was chosen to test our custom features which we selected previously

### *Change in Parameters*

TABLE 3

Kernel v Error

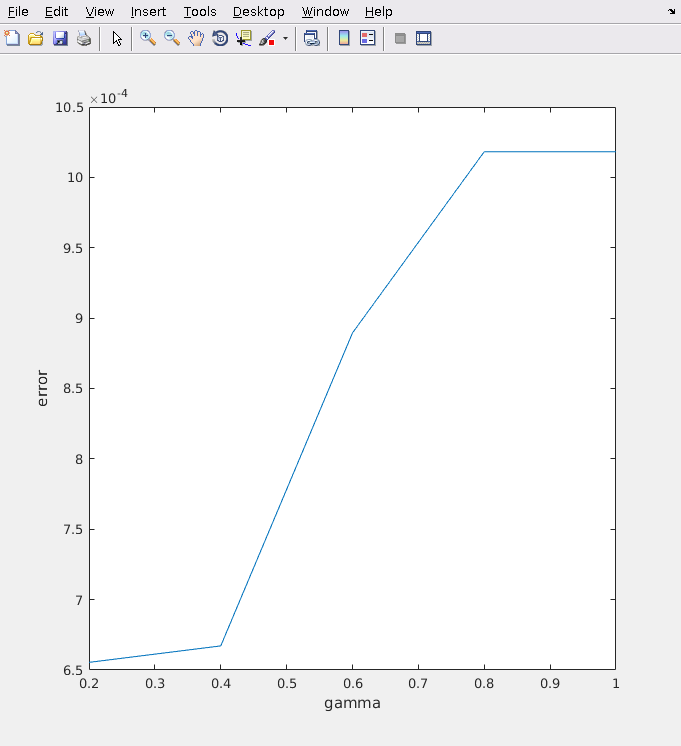
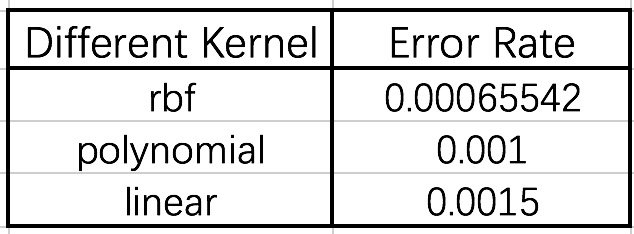


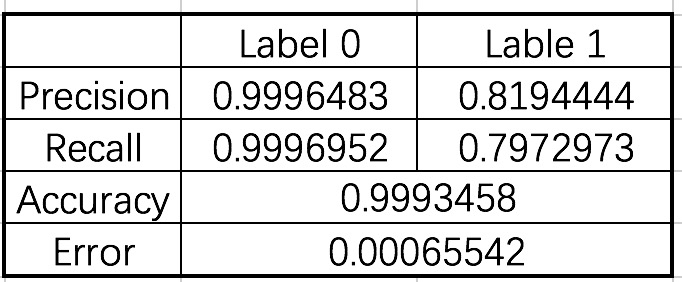
Fig. 4. Gammas vs Error for RBF

With the svm model we were able to change several conditions of the model. First, we were able to choose between different kernels to measure the similarities between our features and the landmarks. The three kernels we were able to choose from was 'linear', 'rbf', and lastly 'polynomial.' In addition to this, in regards to the 'rbf' kernel, we were also able to adjust the different levels of gamma. As we can see from our graph, the lower the gamma selected, the lower the error rate. This is most likely due to the small value of gamma giving us a much lower bias with a high variance to better fit our data. Because of this we have selected a gamma of 0.2 for the following tests. The following graph shows different error rates produced by using the three different kernels. The linear kernel produced the worst error rate out of the three with 0.0015. The best error rate was produced by the rbf kernel with an error rate of 6.5542e-04. And lastly, in between these three the polynomial kernel produced an error rate of 0.0010. From this data, we can deduct that, while our data can be linearly separated, separating through a more complex dimensions produce a more accurate division.

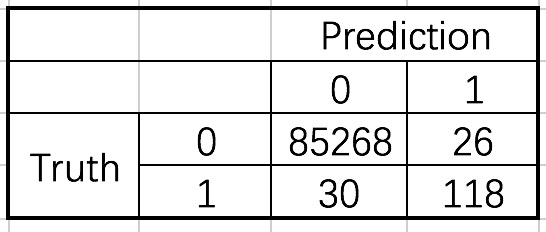
### *Evaluation*

TABLE 4

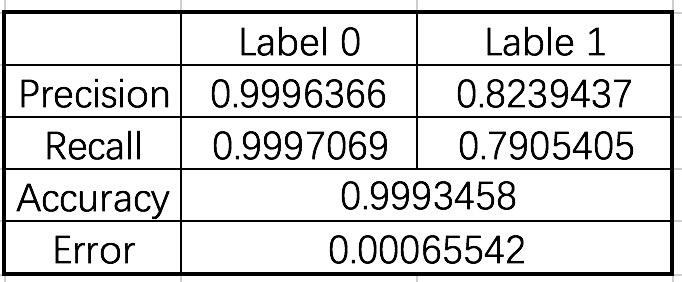
ALL FEATURE STATS



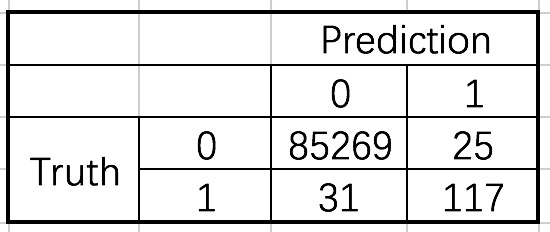
ALL FEATURE CONFUSION MATRIX



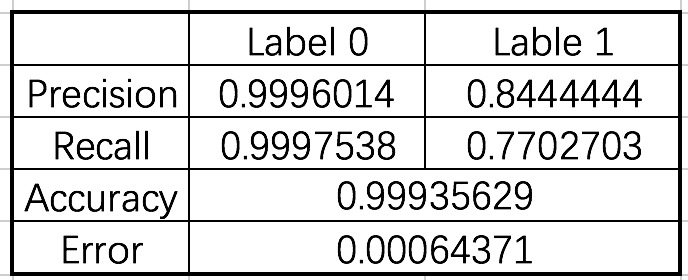
GOOD AND CANDIDATE FEATURES STATS



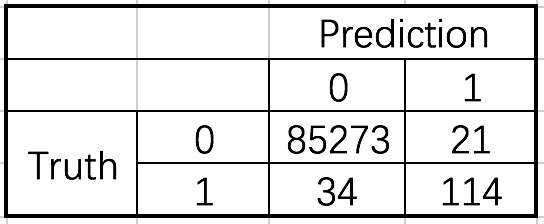
GOOD AND CANDIDATE FEATURES CONFUSION MATRIX



STRICTLY GOOD REATURES STATS



STRICTLY GOOD FEATURES CANDIDATE CONFUSION MATRIX



For our evaluation of SVM, we have selected the model using a rbf kernel with a gamma of 0.2. This was tested with the three sets of selected features. In contrast to the results we have found from logistic regression, with SVM we have actually managed to achieve a higher accuracy using strictly the good features. However, the features we have selected as candidate and good produced the same accuracy as using every feature. Although, the candidate and good features resulted in a slightly different precision and recall rates. This high accuracy was most likely produced through, again, selecting a kernel that does not rely on simple linear separation, and selecting an optimum gamma value for proper regularization.

### Challenges and Solutions

We faced a few challenges when trying to construct this model. First, with this model we were initially planning on changing several parameters such as margin value and gamma values. To do this, our initial attempt at implementing this model was done through using the code developed during the mini programming assignments, as the code was much more flexible in this regard. However, when we attempted to use this model with our dataset we have faced a stack overflow. This is due to our dataset having a large number of 284,807 samples. To solve this issue, we resorted to using the matlab function of fitcsvm. As a consequence, we were unable to see the different changes in the different values of the margin. Although we would assume that having too small of a margin would lead to a lack of proper regularization, causing outlier features to heavily influence our data. Unfortunately, we were not able to find this optimum value for the margin.

## Logistic Regressions

### *Implementation*

The model for our neural network was designed using matlab's neural network toolbox. With our neural network we were able to change several parameters. This includes the number of hidden layers, as well as the number of units in each hidden layer. For each architecture we have implemented, we have calculated the errors of each and chose the best to test with our selected feature sets. In addition to this, similar to the logistic regression approach, we were also able to adjust the threshold for the final layer. Again the optimum threshold for this model will be chosen for the tests. Lastly we also adjust different learning rates for our model.

### *Change in Parameters*

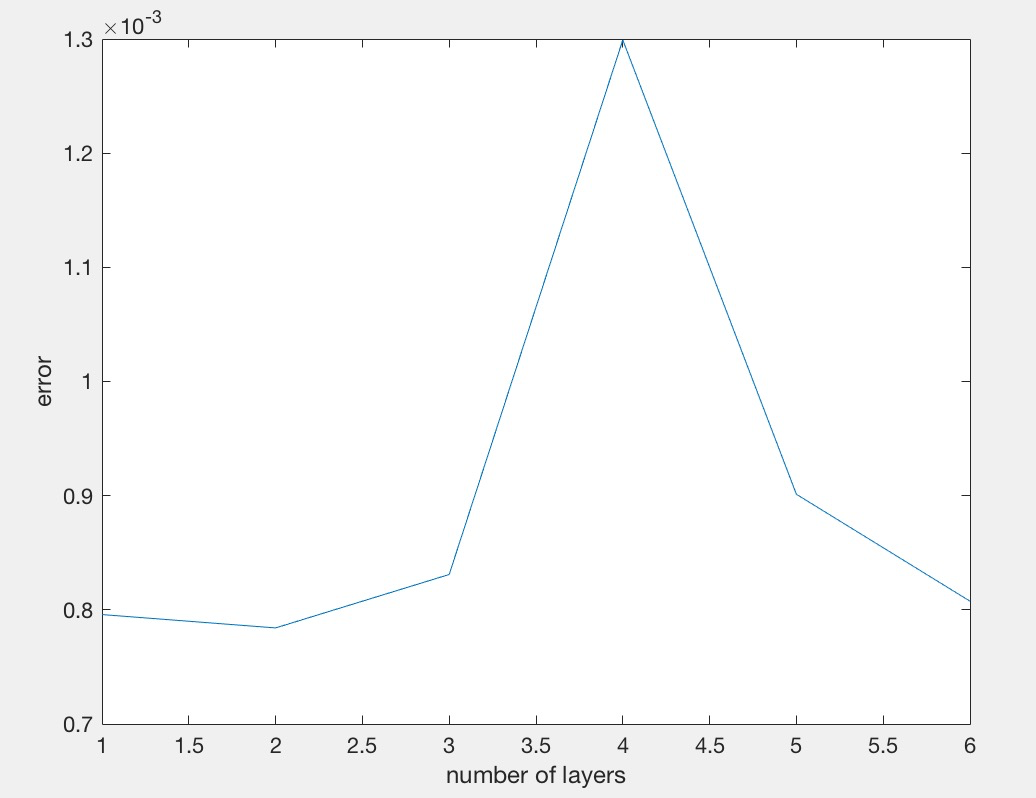


Fig. 5. Number of Layers (20 units each) vs Error

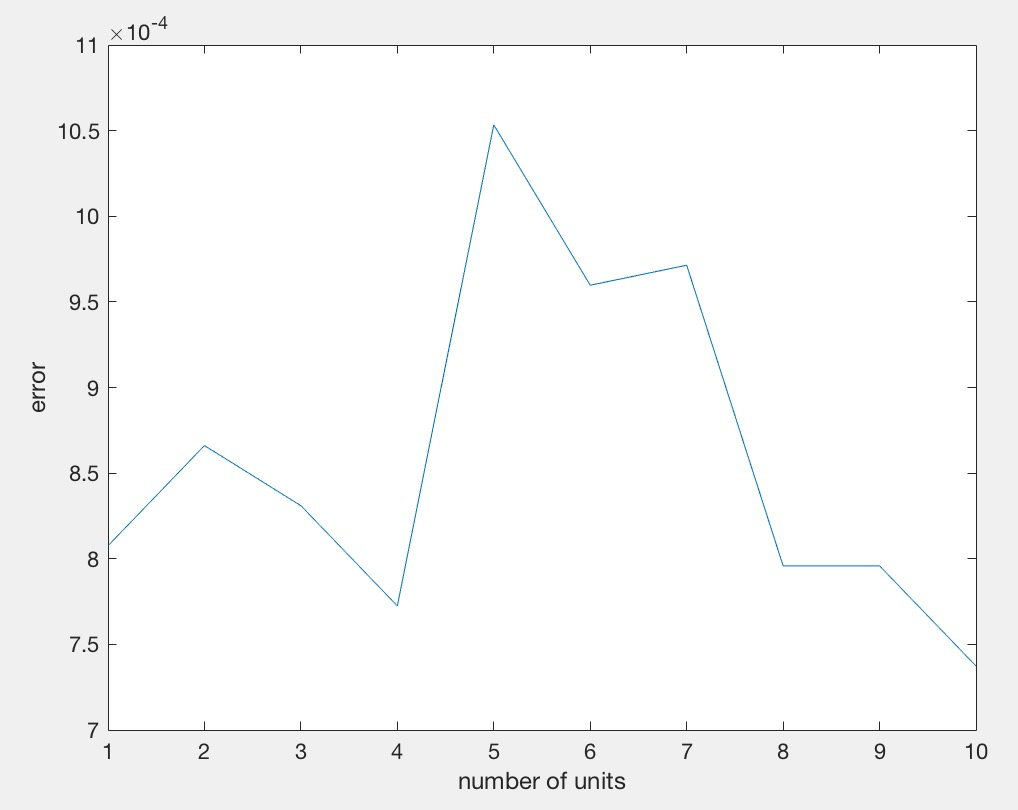


Fig. 6. Number of Units in 1 Layer vs Error

From trying out various different architectures we found it difficult to reach a proper conclusion to any trends within the number of layers of number of units within each layer. As the graph shows, there appears to be almost no correlation between layers and errors. Additionally, there appears to be no correlation with the number of each units in a layer with its corresponding error rate. These results reflect what we have learned in class of how there is no true method or algorithm towards constructing optimum architectures with neural networks.

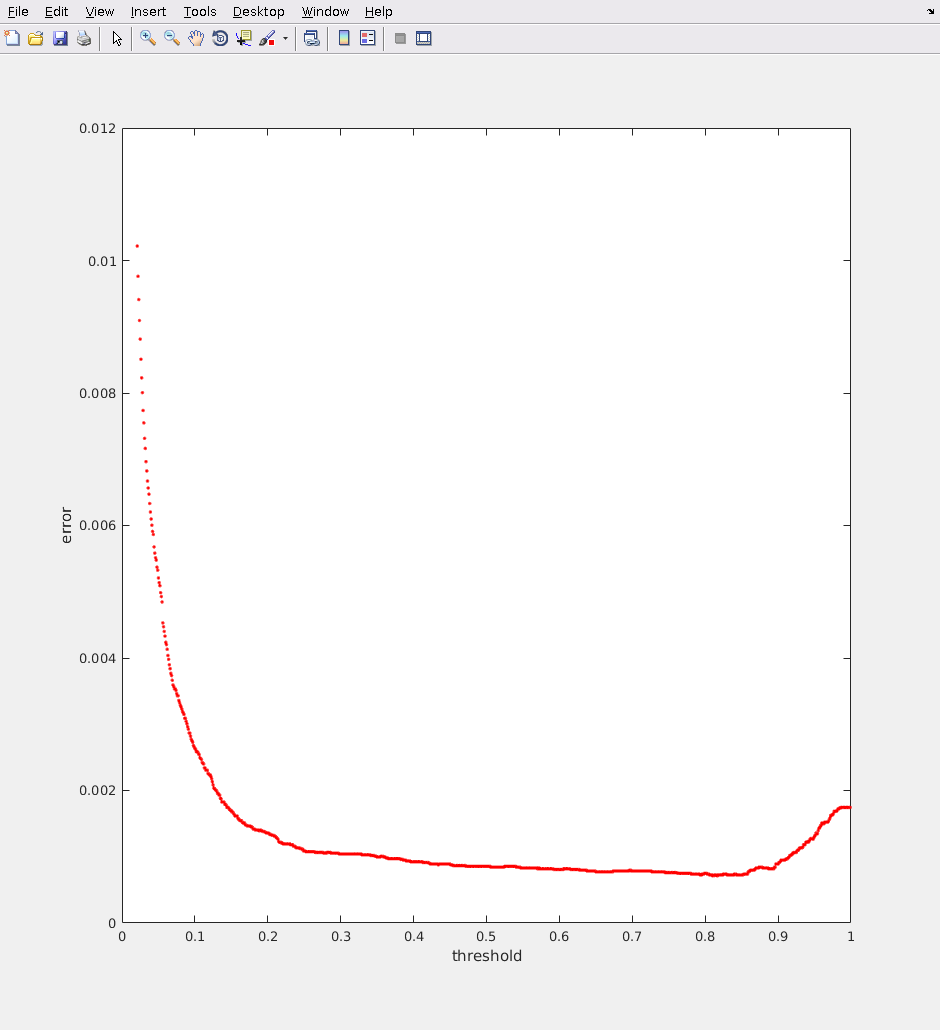


Fig. 7. Threshold vs Error

Surprisingly, from changing different levels of thresholds, the results appeared to be different than that of logistic regression. Where the optimum threshold for logistic regression ranged from around 0.2, the optimum for neural network seemed to range anywhere from 0.3 to 0.8. However both logistic regression and neural network is consistent in how extremely low thresholds and extremely high thresholds tend to generate higher errors.

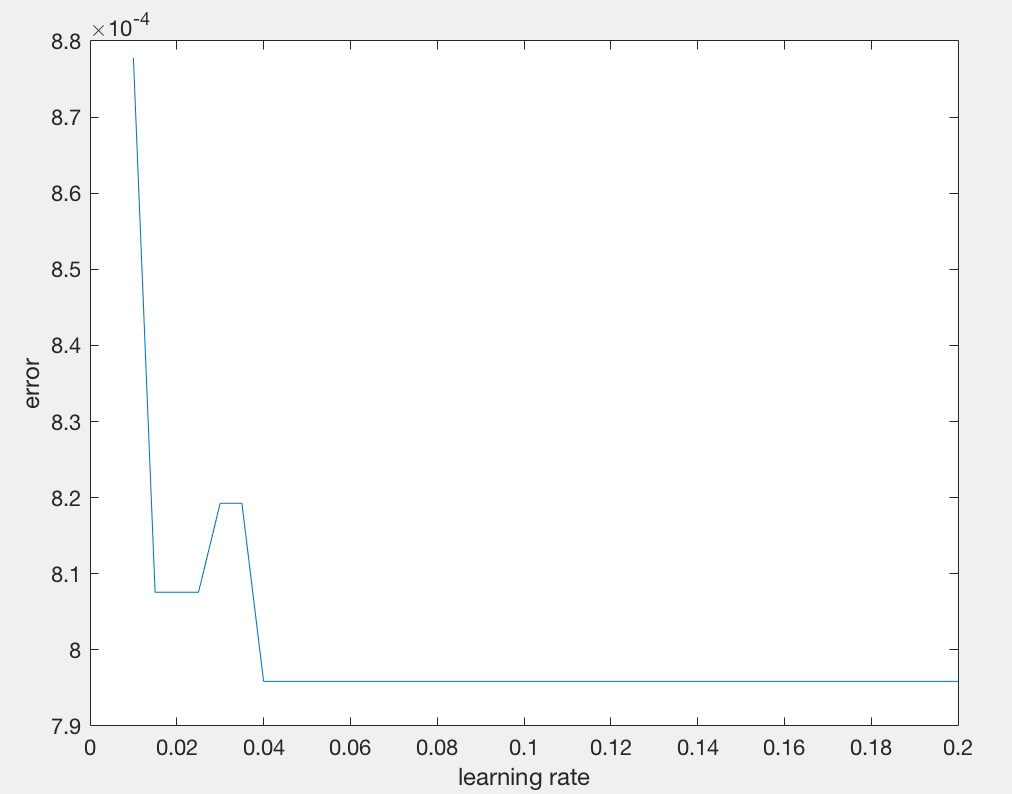


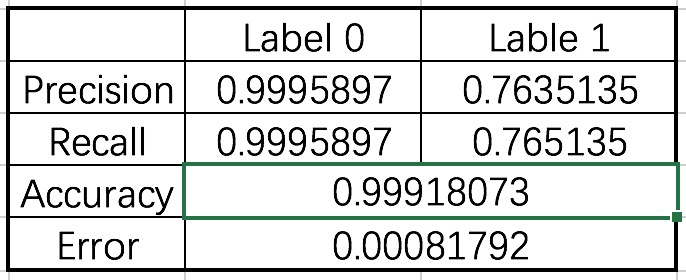
Fig. 8. Learning Rate vs Error

Lastly, from plotting the different error rates using different learning rates, we have concluded that the value of 0.04 suits our model with minimum error.

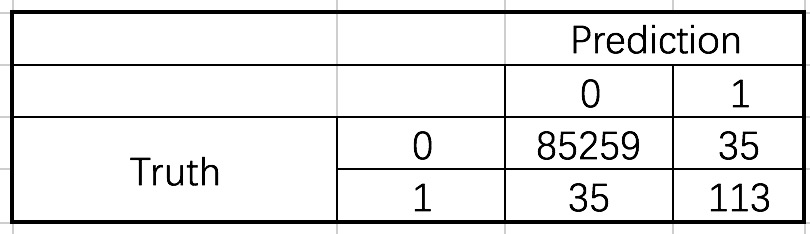
### *Evaluation*

TABLE 5

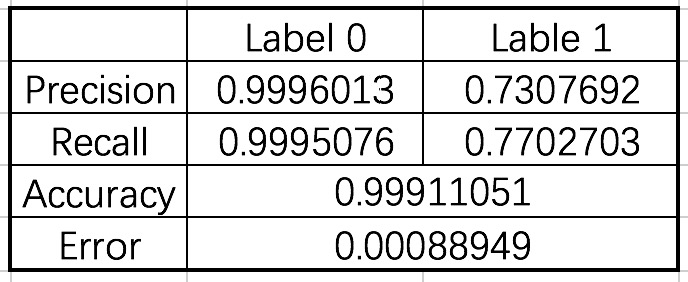
ALL FEATURE STATS



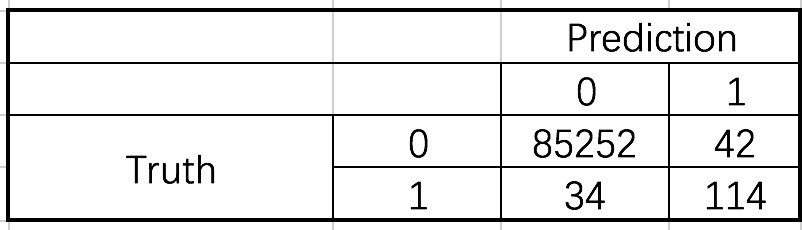
ALL FEATURE CONFUSION MATRIX



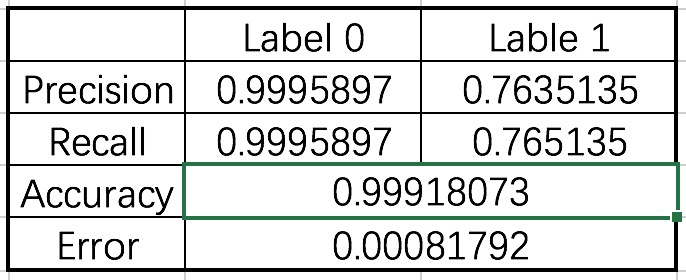
GOOD AND CANDIDATE FEATURES STATS



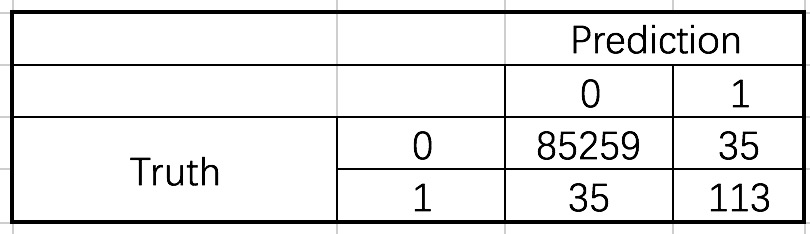
GOOD AND CANDIDATE FEATURES CONFUSION MATRIX



STRICTLY GOOD REATURES STATS



STRICTLY GOOD FEATURES CANDIDATE CONFUSION MATRIX



Similar to with logistic regression, our feature selection failed to improve the accuracy with our neural network model. However we were able to increase efficiency by cutting down the number of features as with logistic regression. Again we were able to produce high accuracies with the highest being 0.9992.

### *Challenges and Solutions*

Similar to with SVM, our first attempt with implementing this model was with using code from our mini assignments. However, the classes predicted with the assignments did not match with the classes in our dataset. Ultimately we resulted to using the matlab function. This meant that we lost the ability to change features such as output functions and activation functions.

## K-Means

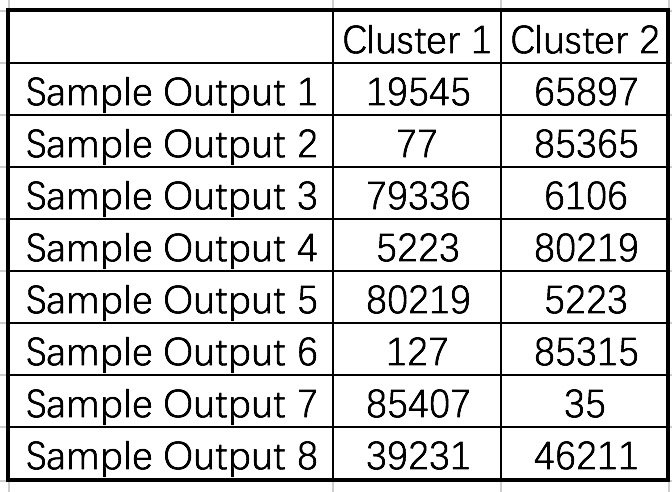
### *Implementation*

The k-means algorithm was implemented through matlab’s kmeans function. We have selected a K of two and fed the function the features of our testing set. This was to attempt to cluster the dataset into two groups. One group would have consisted of all of the non-fraud classes, whereas the other group would consist of fraud classes. After grouping we counted the two groups to see if they have been grouped correctly.

### *Evaluation*

TABLE 6

K-MEANS SAMPLE OUTPUTS



From implementing k-means with our dataset with a k of 2, we failed to produce any meaningful groups. From testing, various results were produced. For example, for one test, the two groups may have been clustered evenly with 50% for each group. Whereas in another case there are an overwhelming amount of one class compared to the other. The latter result is what we would ideally want as we have an overwhelming amount of non-fraud compared to fraud. However, with such inconsistent results we have concluded that using the k-means model would not provide any meaningful results. Although we can change the number of clusters, this does not tell us anything about our data.

### *Challenges and Solutions*

The challenge we faced with this model is our highly imbalanced dataset. This is most likely the cause of such poor results. Since clustering is performed using a distance measure between samples, where we assume that clusters share certain properties with certain boundaries. Since we have largely unequal classes with very different distances between classes. This could be the cause of such poor results.

## Fuzzy C-Means

### *Implementation*

The fuzzy c-means algorithm was implemented using the code from the homework. With fuzzy c means, we tried to cluster into groups of two. In addition to this we attempted to plot the different groups after our clustering.

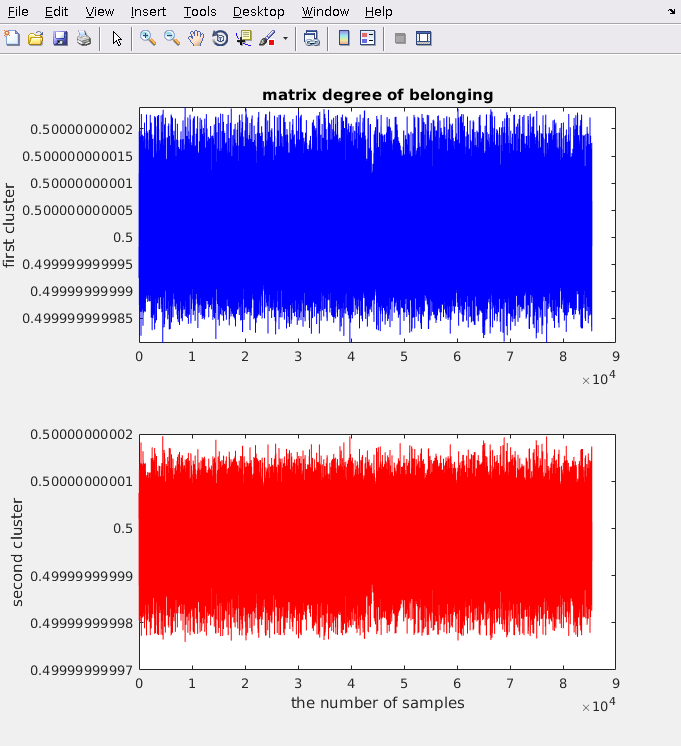


Fig. 9. Cluster Grouping

### *Evaluation*

After plotting the various groups, we were able to visually analyze that there appeared to be no significant difference between the two clusters. This similarity fails to tell us anything about the groups in our dataset in correlation to its features. From this and from using the previous unsupervised learning algorithm of k-means, we have concluded that we are not able to properly cluster our dataset.

### *Challenges and Solutions*

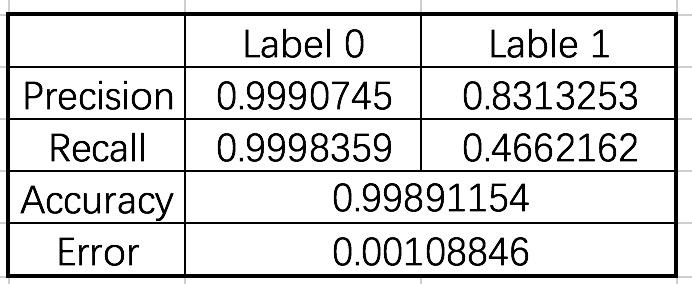
Similar with k-means, we faced the challenge of a highly unbalanced dataset. For future implementations, we should attempt a method of fixing this imbalance or use a dataset with more equal number between groups.

# CONCLUSION

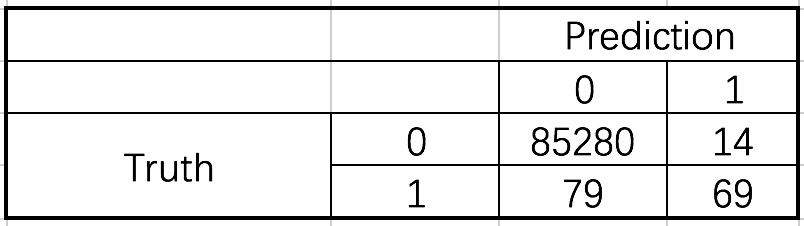
In this experiment, our goal was to design a model that is able to solve the task of classifying between fraud and non-fraud credit card transactions. As a metric of what is the best model for this task, we have initially decided to measure each of the accuracies to choose between the many models. However after testing and analyzing our many results, we have found that all of the models in exclusion to the unsupervised learning models were able to classify between the classes with very high accuracy. Most achieved an accuracy above 0.999. Because of this we have also compared other data. Considering the context of the problem in how we are dealing with a very serious problem of credit card fraud, we wanted a model that was able to classify as much fraud classes as possible. This is because we found it more important for our model to classify more fraud classes than it was able to classify non-fraud classes. In order to do this we have also compared the many models recall rates for class 1 (fraud). With these metrics in mind we have concluded that the best model for this task was the SVM model using an rbf kernel with 0.2 as the gamma and using all features.

TABLE 7

STRICTLY BAD FEATURES STATS



STRICTLY BAD FEATURES CONFUSION MATRIX



Another conclusion we have reached is in regards to our feature selection method. For most of the models we have tested, we have unfortunately resulted in our feature selection method not making a significant impact on the results. However, we do not feel as this method of feature selection is ineffective. To test if our method was effective or not, we have tested the chosen model (svm with rbf kernel and 0.2 gamma) using strictly the bad features. As a result, this set of features produced an error rate which was higher than all of the working models tested earlier with an error rate of 0.0018. Because of this we believe that this form of feature selection is not completely ineffective. We assume our method was not as effective due to the highly unbalanced dataset. Further testing is required with a more even data set to see if our method is truly effective.

We have faced several challenges in this experiment as we have discussed before. First we had to adjust features to ensure consistency in regards to PCA transformation. Next we had to shuffle our dataset as all of the classes were grouped in the top or bottom of the set. Another challenge is most of the code we tried to implement ourselves were not efficient or failed to work due to stack overflows. Lastly, as we had to deal with a highly imbalanced dataset.

For future works we hope research into different models which have better fits for our dataset. Conventional algorithms are often biased towards the majority class and the minority classes usually will be treated as outliers. To solve this issue we hope to find a model that is more optimal towards highly imbalanced datasets such as our own. We would also like to research into different methods or tricks to handle this problem of imbalanced datasets, which was not a topic discussed in class. Also we would like to study different methods of evaluation, as we feel accuracy and a confusion matrix does not effectively represent our results. Instead we would like to study on the ROC curve, or a precision-recall curve. Additionally we hope with future works to implement more of the functions ourselves than rely on matlab functions. This way we have more control over different parameters to better fit our problem.

We hope this report provides more information in how these models and algorithms apply with the problem of credit card fraud.

References

[1] Andrea Dal Pozzolo, Olivier Caelen, Reid A. Johnson and Gianluca Bontempi. Calibrating Probability with Undersampling for Unbalanced Classification. In Symposium on Computational Intelligence and Data Mining (CIDM), IEEE, 2015

Acknowledgements

We would like to thank Professor Liu for providing us with a proper foundation in the topic of Machine Learning. With this knowledge and newly developed passion we hope to pursue further into this topic.

We would also like to thank Yuan Wang (not enrolled in CS596) for assisting us with our feature selection method.

We would also like to thank each of our two group members. Although all the other members left us, we stuck to this group and helped each other in this project. We indeed learned a lot and put lot heart in this project. Meantime, we both will leave SDSU after this semester, we do hope this project will be a good ending of our university life in SDSU.

1. [↑](#footnote-ref-1)