**Team 11 Final Report**

**Data Preprocessing**

Data preprocessing was performed using the *preprocess.py* file, which read the source data in a csv format, pruned the top 2000 rows, created a new Boolean column ‘default’ representing whether or not the person defaulted on their loan (which was represented by a positive number in the ‘loss’ column) and removed all but the first 50 features. This was then outputted into three pairs of csv files, training, validation, and testing, one with a header and index, and one without.

In addition, it also produces training data subsets for use in a later experimental result where we measure the effectiveness of our models relative to the sample size used for training. These subsets scale in a rounded logarithmic manner, with each subset approximately half the size of the last, 500, 250, 100, and 50 row files.

One minor change from the original project plan was that we did not end up using the provided test data for our validation and testing due to the fact that they did not include loss labels and we are thus unable to measure the effectiveness of our model with it. Instead, we simply used some of the remaining rows in the original training set as the validation and testing sets, but otherwise maintained the original project plan, and there should be nearly no effective difference between the two.

**Algorithms**

***Principal Component Analysis***

The Principal Component Analysis algorithm was implemented in *pcamain.py*, which sample the csv output files of *preprocess.py* (including train50.csv, train100.csv, train250.csv, train500.csv, train.csv, validation.csv, and test.csv), to perform cross-validation under training/validation/testing (with a presumed 50/25/25 distribution). To run this algorithm, we used *pcalearn.py* to create the projection matrix, and *pcaproj.py*, to project the sample data towards the smaller dimensionality.

Alongside the training/validation/testing cross-validation I discussed, we also assessed a hyperparameter optimization for the value F, or the number of principle components. This process involved taking the output projection matrix of *pcalearn.py* using the validation set (with F = 30), finding the covariance matrix, getting the eigenvalues and eigenvectors of the output, and calculating a list of the explained variance. This could list could then be mapped to individual principal component indices, that are shown on the plot (a scree plot) in the cross-validation section for hyperparameter tuning. When analyzing the output plot (output as *scree.png*), F = 6 was identified as an optimal elbow for the principal component index.

Furthermore, we assessed each algorithm’s error rate against the number of training samples. For this example, the training sets ranged from 50 samples to 1000 samples while using the optimum value of 6 for F as proposed above. This plot was also output in the local directory as *accuracy\_sample.png*. Any sample less than or equal to 250 units in this case showed the lowest error rate of 8.6% with F = 6. While we already assessed the scree plot for the optimum based on explained variance, we also held the hyperparameter against error rate (output as *accuracy\_hyper.png*). This assessment interestingly yielded different results than the scree plot, showing F values equal to 1, or greater than 11 had the lowest error rates of 8.6% (similar to the previous assessment against sample size).

***K-Nearest Neighbors***

The K Nearest Neighbor algorithm is implemented in *knnmain.py* which uses the csv output files of *preprocess.py* to perform cross-validation in a training/validation/testing setting. Two helper functions are used to create this algorithm, the *knnrun.py* and *knnpredict.py* functions. The knnrun.py function implements cross-validation and knnpredict.py is where the actual KNN algorithm is created and used.

One of the parameters we assessed was the hyperparameter optimization for the k-value. This was where we used the *knnrun.py* to test various values for k against the validation data. By doing so, we were able to find an optimal solution for the hyperparameter k, and as shown in the plot, the value for k was optimized at the value k = 2, and k = 4. It is at these points that the error rate was the lowest it could be, and algorithm was optimized in respect to K.

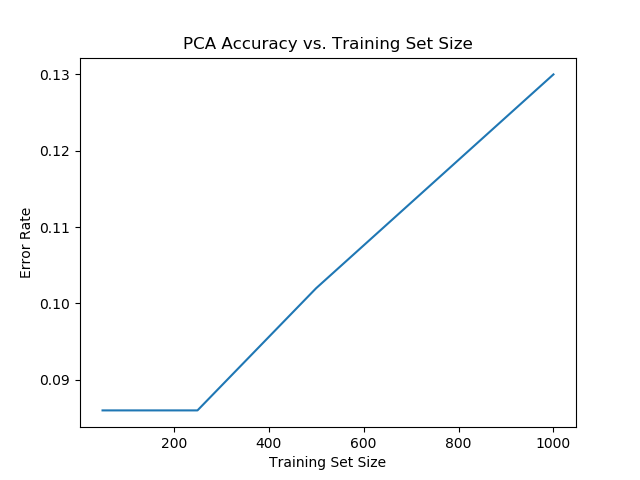
In addition to optimizing the k-value, we also looked into optimizing the sample size of our training data. To do this, we arranged our data into subsamples of 50, 100, 250, 500, and 1000, and trained and tested our algorithms with these subsets of data. We did this in our *knnmain.py* file, and we were also able to plot out these datapoints in a graph. As seen there, with all except the smallest sample size of 50, the plot seems to be constant, possibly due to the fact that all of our data subsets were from the same larger dataset. Nevertheless, it seems that any sample greater than or equal to 100 had the lowest error rate of 0.1.

**Cross-Validation**

**Results:**

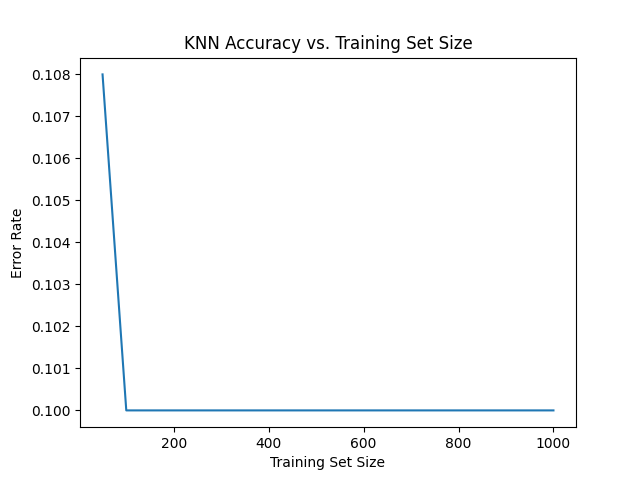
***Sample Size:***

***Principle Component Analysis:***



Take note to observe that the relationship appears to be mostly linear for training sets with >250 samples, with only a slight “bump” occurring at 500 samples.

***K-Nearest Neighbors***



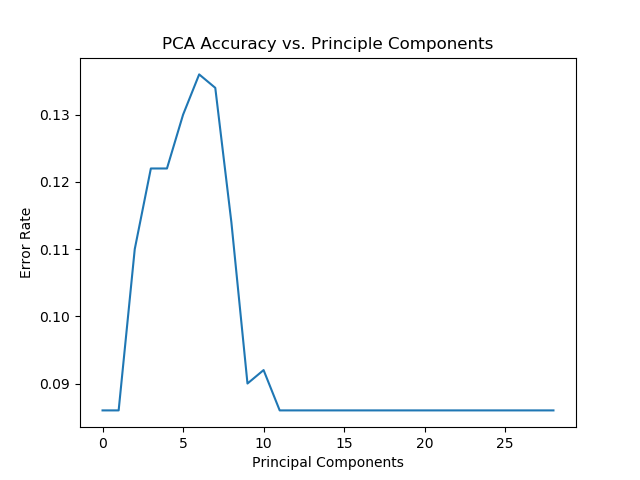
With the exception of the smallest (50) samples training set size, all other subsets had the same error rate. This is likely caused by the fact that the data subsets were, as their name implies, subsets of the same data, and thus would form similar or even identical clusters.

***Hyperparameter Tuning:***

***Principle Component Analysis (F-dimensionality):***

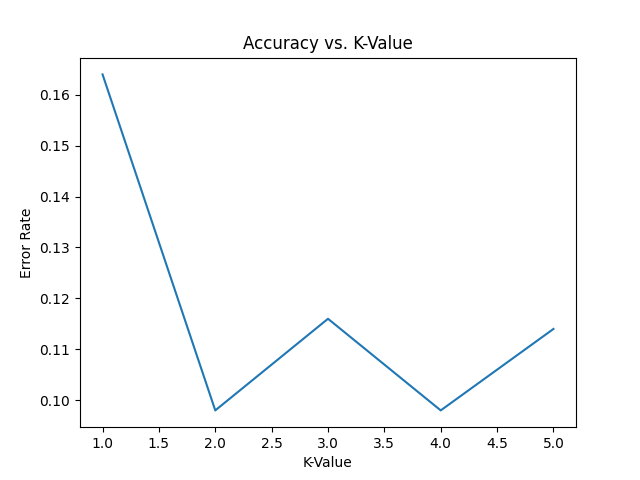


Observing the distribution of the explained variance ratio for each principal component index (up to index 30), the strongest change, or the “elbow”, in explained variance can be seen with the 6th principle component, serving as the optimum balance for dimensionality in the projection matrix, with F = 6.



Unlike what was portrayed with the scree plot for the explained variance of the principle component indices, the optimal value of F was not 6, but rather a value of 1 or greater than 11 resulted in a minimized rate of error (highest accuracy).

***K-Nearest Neighbors (k-value):***



For this model and our data, we were able to minimize the error rate (highest accuracy) for K = 2 and 4.