**Team 11 Preliminary Progress Report**

**Data Preprocessing**

Data preprocessing was performed using the *preprocess.py* file, which read the source data in a csv format, pruned to the 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 two csv files, one with a header and index, and one without. 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. Later, we will also modify the file to make several smaller subsamples for plotting the experimental result of accuracy vs sample size.

**Algorithms**

***Principle Component Analysis***

Our version of the PCA algorithm can be found in *pcalearn.py*, with a full implementation in *pcamain.py.* We then use *pcaproj.py* to create a projection matrix of the model. The algorithm is fully implemented, but lacks any hyperparameter optimizations (with the dimensionality reduction and the validation set), so for now it uses a constant F=10 for training and testing.

***K-Nearest Neighbors***

Our version of the KNN algorithm is in the *knnpredict.py* file, and we can change the k-value parameter in the *knnmain.py* file. This algorithm is also fully implemented, but similar to the PCA, it lacks any hyperparameter optimizations within the validation set, so for now sits with a constant k=1 for training and testing.

**Cross-Validation**

As far as cross-validation with our training/validation/testing sets, we have been able to achieve our first instances with our Principal Component Analysis algorithm. Since we have yet to enforce hyperparameter tuning with our validations sets, we considered a random dimensionality reduction of F=10, reaching ~8.6% error when our model was placed against our testing set. For the K-Nearest Neighbors algorithm, we had a similar case without any hyperparameter tuning through the validation set, but were left with an error ~16.4% with k=1 against the testing set.