# I. Definition

## Project Overview

Credit cards provide a way for card holders to purchase goods on credit and carry a balance until paid. In exchange, the card holder pays interest on the balance. Banks profit from this interest and other fees paid by the card holder, but they also earn revenue from transactional fees paid by the merchants.

Since credit cards are used for goods that cannot be repossessed (unlike homes and cars), credit card default is a major source of risk for card issuers. “Default” is defined to be not making any payment when it is due[1]. While collection efforts may be able to collect the debt, the account is at risk for repayment and the bank faces loss of the amount in balance. This contributes to the relatively high interest rates (~19% for new card holders) that credit cards have as opposed to home or car loans. It would be valuable to card issuers to more precisely predict default in order to develop approaches to limit risk exposure.

Additionally, the card issuer could offer lower APRs to card holders with low probability of default. According to a research paper by the Boston Fed[2], “Nearly 80 percent of U.S. adults have a credit card, and more than half of them revolve their debt from month to month.” Therefore, bringing the cost of debt lower for low risk cardholders would represent a large opportunity. Such an innovation could improve the reputation of credit card companies. Currently, when obtaining a credit card, the card holder agrees to terms and conditions which include, and lock in, a base APR (variable according to the external factors). These terms allow banks to raise rates for missing payments, but do not include mechanisms to lower rates for good credit behavior. This is a missed opportunity because there is a customer pain point (high APRs) aligned to a business objective: card holders don’t want to carry a balance because it is so expensive and banks want customers to carry a balance as a source of revenue. I believe that, according to supply and demand economics, if the price of credit were to drop, demand would rise. In other words, if a customer’s rate were to decrease over time, based on a low likelihood of default, then the customer would carry a balance more than they do today. This would increase revenue from customers which pay off the balance each month while also managing risk.

## Problem Statement

The problem to be solved is: given the payment history of clients, can we predict default one month in advance? By solving this problem, it would be a step in the direction of predicting likelihood of default in general.

---

1 https://www.consumerfinance.gov/data-research/credit-card-data/know-you-owe-credit-cards/credit-card-contract-definitions/

2 https://www.bostonfed.org/publications/research-department-working-paper/2017/credit-card-utilization-and-consumption-over-the-life-cycle-and-business-cycle.aspx

To solve the problem, I will use supervised classification algorithms to predict default.

## Metrics

Since ~78% of the observations are non-default, a trivial classifier that classifies all observations as non-default, would achieve a 78% accuracy. As a result, accuracy would be a misleading metric to solely characterize the model's performance.

In addition, I'll use a common classification metric known as the Receiver Operating Characteristic Area Under the Curve (ROC AUC). This metric will graph samples of the true positive rate versus the false positive rate and compute the area under the curve. The perfect score would be 1 which would represent that no FNs were found. A score of .5 indicates that the model is no better than chance at classifying between two classes.

# II. Analysis

## Data Exploration

I used a public dataset on Kaggle (originally from UCI)[3]. This dataset was created for research purposes as cited[4].

Below is a random sample of the dataset for reference:

| ID | 11880 | 23449 | 4423 | 24142 | 27725 | 17852 | 19993 | 18523 | 3010 | 20059 |
|---|---|---|---|---|---|---|---|---|---|---|
| **LIMIT_BAL** | 70000 | 30000 | 110000 | 150000 | 30000 | 20000 | 240000 | 100000 | 30000 | 80000 |
| **SEX** | 2 | 2 | 2 | 1 | 2 | 1 | 2 | 2 | 1 | 2 |
| **EDUCATION** | 2 | 6 | 2 | 1 | 3 | 2 | 2 | 1 | 3 | 1 |
| **MARRIAGE** | 2 | 2 | 1 | 2 | 2 | 2 | 1 | 2 | 2 | 1 |
| **AGE** | 24 | 45 | 34 | 27 | 24 | 26 | 34 | 26 | 25 | 37 |
| **PAY_0** | 0 | 0 | 0 | 0 | 1 | 1 | -1 | -1 | 2 | -1 |
| **PAY_2** | 0 | 0 | 0 | 0 | -2 | 2 | -1 | -1 | 2 | -1 |
| **PAY_3** | -2 | 0 | 0 | 0 | -1 | 2 | 0 | -1 | 2 | -1 |
| **PAY_4** | -1 | 0 | 0 | 0 | 0 | 2 | 0 | -1 | 0 | -1 |
| **PAY_5** | -1 | 0 | 0 | 0 | 0 | 0 | -1 | -1 | 0 | -1 |
| **PAY_6** | -1 | 0 | 0 | 0 | 0 | 0 | -1 | -1 | 2 | -1 |
| **BILL_AMT1** | 16366 | 25298 | 24147 | 149340 | 0 | 16167 | 626 | 123 | 22440 | 2028 |

[3] https://www.kaggle.com/uciml/default-of-credit-card-clients-dataset

[4] Yeh, I. C., & Lien, C. H. (2009). The comparisons of data mining techniques for the predictive accuracy of probability of default of credit card clients. Expert Systems with Applications, 36(2), 2473-2480.

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| **BILL_AMT2** | 15500 | 26866 | 25750 | 152460 | 0 | 15615 | 1921 | 123 | 24168 | 286 |
| **BILL_AMT3** | 9660 | 27662 | 27012 | 141196 | 1469 | 18047 | 20740 | 123 | 23481 | 14255 |
| **BILL_AMT4** | 6208 | 28474 | 27762 | 105802 | 2432 | 17322 | 21274 | 123 | 23950 | 10056 |
| **BILL_AMT5** | 702 | 28494 | 32126 | 106478 | 827 | 17122 | 888 | 123 | 25659 | 317 |
| **BILL_AMT6** | 4320 | 0 | 37779 | 107096 | 0 | 0 | 360 | 123 | 25098 | 728 |
| **PAY_AMT1** | 1000 | 2000 | 2000 | 10000 | 0 | 0 | 1921 | 123 | 2390 | 286 |
| **PAY_AMT2** | 9660 | 2000 | 2000 | 6040 | 1469 | 3000 | 19000 | 123 | 0 | 14255 |
| **PAY_AMT3** | 6208 | 1500 | 1500 | 5750 | 1008 | 0 | 2624 | 123 | 859 | 10071 |
| **PAY_AMT4** | 702 | 1000 | 5000 | 3000 | 300 | 0 | 888 | 123 | 2097 | 317 |
| **PAY_AMT5** | 4320 | 0 | 6400 | 5000 | 0 | 0 | 360 | 123 | 0 | 728 |
| **PAY_AMT6** | 1650 | 0 | 0 | 35000 | 0 | 0 | 360 | 396 | 1092 | 16312 |
| **default payment next month** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 |

The purpose of the dataset creation was to model the probability of default of credit card holders. There are 25,000 observations in the dataset of which ~22% are in default.

There are 25 variables in the dataset: 23 explanatory variables, one binary target variable (default next month) and one identifier. In the explanatory variables there are two main types: demographic (4) and payment history (19). The demographic variables are all categorical types. The payment history includes a mix of categorical and numerical variables. The payment history was observed in October 2005 from a bank in Taiwan. It includes observations for each card holder from April - September 2005. In each month, it states the repayment status (what happened from prior month, i.e. paid duly or delayed payment for *x* months), the amount on the current month bill statement, and the amount that was paid that month. There are unobserved variables such as payment timing and transaction activity for the month, but those seem to have been encoded in the repayment status, but no data other than the target variable was provided for the month of October. A simple example of how a credit card bill is calculated reveals the relationship between the variables and reveals some unobserved variables:

$$BillAmount = PreviousBalance - Payment + Purchases + Fees + Interest$$

Therefore, without purchase, fee, and interest data, we must assume trust in the encoding of the repayment status and will not be able to validate it.

There are no missing values in the data, but upon inspection there are several values of the categorical variables which were not defined or unclear.

| Variable | Value | Disposition |
|---|---|---|
| Repayment Status (X6-X11) | -2 | Upon inspection, it seems that this indicates 3 possible scenarios: |

| | | |
|---|---|---|
| | | 1. The entire bill statement will be paid in the next month and nothing is due from the prior month (the bill statement amount for month x is >= the payment amount for month x+1)<br>2. All expenses are paid within the month, before being posted to the bill statement. I noted this based on months where there was 0 bill statement but also a payment made, but then the next month it wasn't a negative balance (credit from the payment).<br>3. There was no account activity.<br><br>According to a question on Kaggle, this indicates "no consumption," or the card was not used for new purchases in the month. |
| Repayment Status (X6-X11) | -1 | The bill will be paid in the next month |
| Repayment Status (X6-X11) | 0 | Use of revolving credit, meaning that the minimum was paid on the bill statement balance |
| Repayment Status (X6-X11) | ≥ 1 | Payment will be delayed by *n* months |
| Bill Amount (X12-X16) | <0 | Negative bill amounts are interpreted as credit. This would happen when the card holder makes a payment in excess of the bill amount, likely because they are paying in advance towards purchases not yet posted to the bill.<br><br>I will consider negative bill amount as outliers since it is impossible to default on a credit. |
| Education | 0, 5, 6 | The observed volumes are:<br><br>0 : 14 records<br><br>5: 280 records<br><br>6: 51 records<br><br>Since all records with 0 value do not have default, this may be helpful in prediction, even though its meaning is not clear. I will therefore leave the Education values unchanged even though they have some unclear definition. |
| Marital Status | 0 | Only 54 records. Will change to the "other" category value 3. |

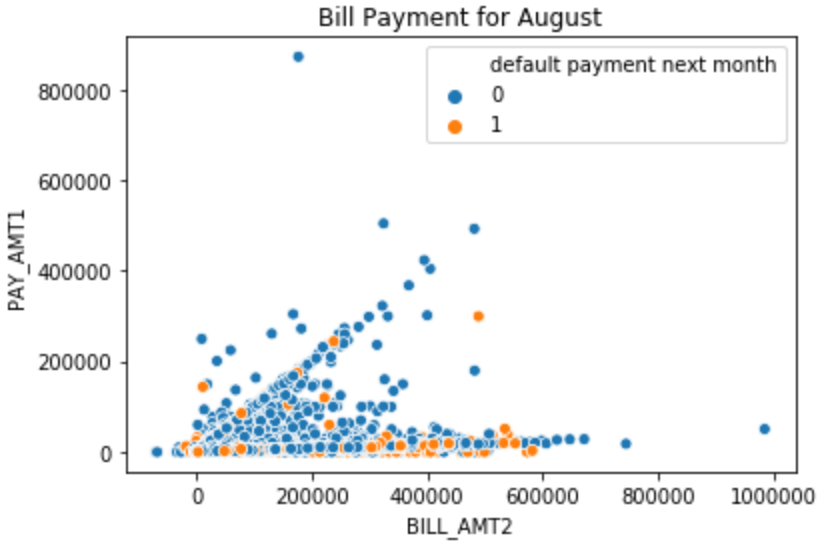
This data is based on Taiwan customer default payments from April - September 2005. As a result, this dataset should not be assumed to create a model that would generalize well for a current US bank.

# Exploratory Visualization

The dataset includes time based information that was collected over a 5 month period. As mentioned above, there is an inherent temporal relationship between the variables that is a key characteristic to explore. Since

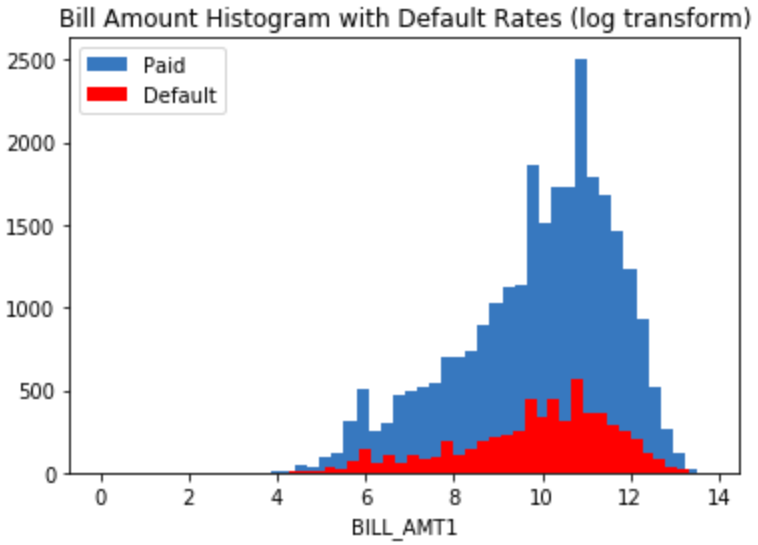
$$Bill\text{-}Amount = Previous\text{-}Balance - Payment + Purchases + Fees + Interest$$

I would assume that the bill amount and the paid amount would appear triangular in a scatterplot. The following plot explores the relationship between the `BILL_AMT` and `PAY_AMT` variables.



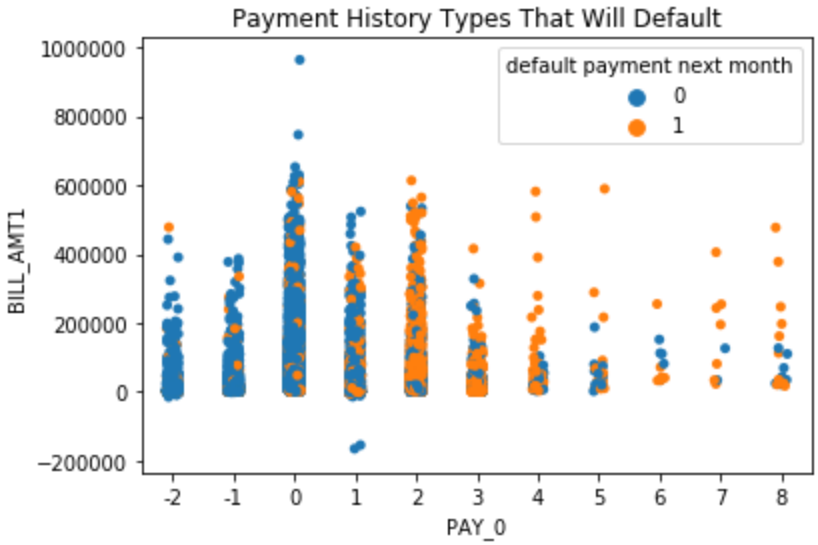

- The data which is **on the diagonal line** $y = x$ represents cardholders that pay their total balance before the next bill statement.
- The data **below the line** shows when the cardholder pays less than the previous balance and thus begins paying interest.
- There is also the presence of payments **above the bill amount** and in such a case the cardholder would be given credit the next month. This may be the cardholder paying the posted amount for purchases which have not yet appeared on a bill.
- Another interesting observation that this plot shows is that many cardholders that will default are already not paying the bill in September.
- Lastly, there is a stronger correlation seen between higher bill amounts and default.

The following plot explores if the bill amount may correlate with default.




This plot shows the relative (transformed) shape of the bill amount distribution stacked against the default = 1 distribution. The closer the distributions are in shape, the stronger the correlation. In this plot it is observed that they do resemble each other, but the default distribution does not reach as high of a percent of the bill amount in the most common bill amounts. This means that it would appear default is more likely with both small and large bills than the mode bill amount. My intuition would have guessed a higher default rate for large bills, but I was a bit surprised to see it at a higher rate for the smaller bills.

This leads me to explore the correlation between `PAY_0` and `default next month`.




Looking at the categories of payment history, it is clear that a history of deferred payments (indicated by the encoded Pay_0 where 1 = delayed for one month, etc) increases the likelihood of default payment the next month. This can be seen by observing the orange tones dominating the categories starting at 2.

This shows that both the bill amount and payment history will correlate with `default payment next month`.

# Algorithms and Techniques

These classification algorithms were used to predict the test data. As mentioned, this dataset has both categorical and numerical features and as a consequence of some algorithms selected, data preprocessing was needed.

### **Logistic Regression**:

- ***Why good candidate?:*** This is a classic algorithm for this sort of classification problem. It will optimize weights to each of the features to minimize the loss function.
- ***Strengths:*** Very interpretable and gives probabilities for observations which means that without changing the model, it is possible to change the probability thresholds to result in different classifications. Other algorithms here output discrete classifications.
- ***Weaknesses:*** Requires preprocessing of the data (categorical variables into dummies and normalized numerical variables). Doesn't perform well with a large feature space, for example in image recognition.
- ***Parameters***: `solver = liblinear` Used default solver since it was recommended by scikit learn for small dataset with a binary target class. `penalty = L1` since L1 regularization is robust to outliers and there are outliers in the financial variables (outlier defined as over 1.5 x σ away from the mean). L2 regularization was tested as well but did not perform as well in grid search. `C=2` smaller values specify stronger regularization. This value was found to perform better than the default of 1.
- ***References:*** sci-kit-learn; wikipedia

### ADABoost

- ***Why good candidate?*** The decision tree upon which AdaBoost will operate can handle both categorical and continuous data. AdaBoost is an ensemble method that minimizes the risk that a singular decision tree only considers the information gain of splitting a feature at the node level and not within the context of the data as a whole (as the weights essentially force it to do).
- ***Strengths:*** Builds upon "weak classifiers," such as a decision tree, by giving extra weight to misclassified data and iteratively fitting upon weighted versions of the data. This addresses an issue with decision trees where after a split is made in the data, the subsequent splitting decisions do not take all data from the prior split into consideration. It is therefore not very susceptible to over-fitting (as compared to decision trees).
- ***Weaknesses*** The algorithm must operate in a sequence of applications of the weak learner, so it cannot be parallelized. If the data were noisey, it could weight the "misclassified" noise more strongly and therefore noise would have more of an impact. Less interpretable than a regular decision tree.
- ***Parameters***: Left all parameters in the default settings mainly `n_estimators=50` and `learning_rate=1`

- ***References*** sci-kit-learn; Boosting with AdaBoost and Gradient Descent; viola-jones wikipedia; AdaBoost

### XGBoost

- ***Why good candidate?:*** Similar to ADABoost, but strong performance in kaggle competitions
- ***Strengths:*** Decision tree based classifier that is gradient based and designed for speed and performance.
- ***Weaknesses:*** Require preprocessing of categorical variables (one hot encoding) because it will assume an ordinal relationship for the label encoding already used in the dataset. Lots of hyperparameters.
- ***Parameters***: `booster = gbtree` sets the algorithm to use a decision tree based model. `objective = binary:logistic` recommended for binary classification. `eta = 0.5` learning rate reduction used in update to prevent overfitting. `gamma = 0` Minimum loss reduction required to make a further partition on a leaf node of the tree. The larger gamma is, the more conservative the algorithm will be. `min_child_weight = 1` like gamma, the larger the weight the more conservative the algorithm will be because it will partition the data less. `max_depth = 3` Increasing this value will increase complexity and likelihood of overfitting.
- ***References:*** XGBoost Example; Introduction to XGBoost; Data Preprocessing for XGBoost; XGBoost whitepaper

### Artificial Neural Network

- ***Why good candidate?:*** Shown to have good performance on this dataset in the original paper cited earlier.
- ***Strengths:*** Can handle non-linear relationships if present in data (it's been proven that neural networks can approximate any function).
- ***Weaknesses:*** Convergence to global minimum is not guaranteed. Difficult to configure with many hyperparameters. Training time is much longer than other methods used. Loss of interpretability of the model. Also requires preprocessing of the data: categorical variables (one hot encoding) because it will assume an ordinal relationship for the label encoding already used in the dataset and normalization of the numerical variables helps convergence in gradient descent.
- ***Parameters***: The architecture used was 2 hidden layers with 64 nodes and 32 nodes respectively. Relu activation was used in the hidden layers and softmax activation on the output later. In compiling the model, `optimizer=rmsprop` provides momentum to the step sizes in gradient descent by dividing the gradient by a moving average of it magnitude. `loss=categorical_crossentropy` used for binary classification. `metrics=accuracy` since accuracy is a metric used here to compare models and ROC is not an available metric. `batch_size=32` kept the default value for the number of samples per gradient update.
- ***References:*** Universal Function Approximation theorem; keras documentation

# Benchmark

As mentioned above, a trivial classifier could achieve a 78% accuracy.

There are three research papers that I was able to find that use the “Taiwan” dataset[5][6][7]. In the research of [5] the authors used accuracy and area ratio in the lift chart to measure different algorithms in order to approximate the real probability of default. These experiments were found to give at best an 84% accuracy and 54% area ratio on validation data. In [6] the authors use different algorithms and achieve an accuracy of 81.96% and AUC of 64.63% Lastly, in [7] the authors augment the dataset with transactional purchase data (an unobserved variable as I noted above) from another dataset. Using this method they were able to achieve 95.8% accuracy, 94.8% precision, 85.9% recall, and .9104 F1 score.

Since most of this research used accuracy as the metric, I will compute this as well so that I have an additional benchmark to compare the results of my work.

# III. Methodology

## Data Preprocessing

Preprocessing was needed on the data to transform the data as required by the algorithms I used.

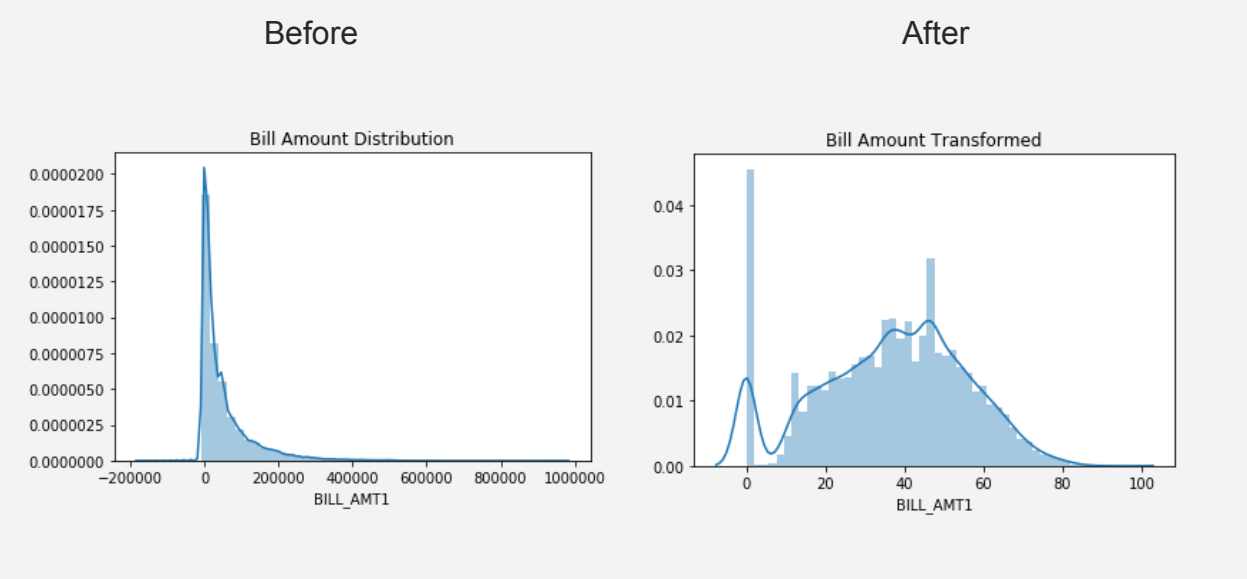
For the categorical variables, I used one hot encoding to transform them into a new feature space since there is no correlation between the numeric values in which there are encoded in the original data (i.e. where male = 1 and female = 2, algorithms will essentially interpret that as degrees of the same scale).

For the numerical variables, since they are skewed, I transformed them using a Yeo-Johnson power transformation. This transformation is advantageous over a log or Box-Cox transform, in that it doesn’t require that the data be positive. This is important since I will map negative bill statements to 0 as mentioned above. Normally distributed data helps gradient descent in convergence.

As an example of the transformation:

---

[5] Yeh, I. C., & Lien, C. H. (2009). The comparisons of data mining techniques for the predictive accuracy of probability of default of credit card clients. Expert Systems with Applications, 36(2), 2473-2480.
[6] Lu, Hongya, Haifeng Wang, and Sang Won Yoon. "Real Time Credit Card Default Classification Using Adaptive Boosting-Based Online Learning Algorithm." IIE Annual Conference. Proceedings. Institute of Industrial and Systems Engineers (IISE), 2017.
[7] Islam, Sheikh Rabiul & Eberle, William & Khaled Ghafoor, Sheikh. “Credit Default Mining Using Combined Machine Learning and Heuristic Approach,” 2018.




# Implementation

All implementation was carried out using Python 3.6 in a Jupyter Notebook. Scikit Learn was used to implement Logistic Regression and ADABoost. XGBoost was implemented with the Python API. The Neural network was implemented using Keras.

During implementation I found that in order to calculate both Accuracy and ROC AUC metrics, it was required to make two different prediction datasets for each algorithm: one with a binary prediction was needed for the accuracy metric and one with probabilistic predictions was needed for ROC AUC. All of the algorithms supported predictions of both types.

One complication in implementing the Neural Network was that cross-entropy required a categorical encoding of the target variable. Keras provided a method to handle this and it was used to modify both the training and testing target data.

# Refinement

Grid Search was used to refine the models implemented with Scikit learn. Experimentation was used to refine the Neural Network model.

Initial Logistic Regression model parameters:

```
clf = LogisticRegression(random_state=42, solver='liblinear', penalty='l1', \
                                        max_iter=10000).fit(X_train, y_train)
```

This model achieved a 81.98% accuracy and 77.37% ROC AUC on test data. After running grid search this was found to be the best LR model:

```
LogisticRegression(C=2, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=10000,
                   multi_class='warn', n_jobs=None, penalty='l1',
```

```
                    random_state=42, solver='liblinear', tol=0.0001, verbose=0,
                    warm_start=False)
```

This shows that the model performed best when regularization was kept smaller (as shown by the C value greater than the default=1 and L1 versus L2 regularization). I believe that this was due to the feature scale differences in the data. This final LR model achieved an Accuracy of 81.95% and ROC AUC of 77.33% on the test data. This is actually slightly poorer performance on the test data which may indicate that the parameters which performed best in validation did not generalize to the test data.

The initial ADA Boost model used the default parameters:

```
clf_ADA = AdaBoostClassifier(random_state=42)
```

This model achieved an 81.65% accuracy and 77.37% ROC AUC on the test data. After running grid search on the learning_rate and n_estimators parameters, this was found to be the best ADA Boost model:

```
AdaBoostClassifier(algorithm='SAMME.R', base_estimator=None, learning_rate=1,
                   n_estimators=50, random_state=42)
```

This shows that the default parameters were already optimal for this algorithm.

The initial XG Boost model used the default parameters:

```
from xgboost import XGBClassifier
#Instantiating the classifier
clf_XGB = XGBClassifier()
```

This model performed the best so far with an accuracy of 81.93% and ROC AUC of 78.27% on the test data. After running grid search on a number of parameters, this was found to be the best XG Boost model:

```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0,
              learning_rate=0.5, max_delta_step=0, max_depth=3,
              min_child_weight=1, missing=None, n_estimators=25, n_jobs=1,
              nthread=None, objective='binary:logistic', random_state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
              silent=None, subsample=1, verbosity=1)
```

This model somewhat improved performance to an accuracy of 82.08% and ROC AUC of 77.98%.

Neural networks require significantly more design with more possibility for refinement than the other algorithms used. To get started, the initial neural network architecture had only one hidden layer, slightly larger than the input size, and a 50% dropout. Relu activation was used in the hidden layer and softmax in the output layer.

Initial Neural Network Architecture:

```
clf_NN = Sequential()
clf_NN.add(Dense(100, activation='relu', input_shape=(94,)))
```

```
clf_NN.add(Dropout(.5))
clf_NN.add(Dense(2, activation='softmax'))
```

This achieved a strong comparative performance to the other algorithms with 81.52% accuracy and 77.02% ROC AUC on the test data. I tried adding another hidden layer and reducing the size of first hidden layer to about ⅔ of the input layer.

```
clf_NN = Sequential()
clf_NN.add(Dense(64, activation='relu', input_shape=(94,)))
clf_NN.add(Dropout(.3))
clf_NN.add(Dense(32, activation='relu'))
clf_NN.add(Dropout(.3))
clf_NN.add(Dense(2, activation='softmax'))
```

This did improve performance to Accuracy of 82.05% ROC AUC of 77.35% on the test data. I tried making a deeper network with another hidden layer, this time with no dropout:

```
clf_NN = Sequential()
clf_NN.add(Dense(64, activation='relu', input_shape=(94,)))
clf_NN.add(Dense(32, activation='relu'))
clf_NN.add(Dense(32, activation='relu'))
clf_NN.add(Dense(2, activation='softmax'))
```

Performance started to degrade on the test data which I believe is due to overfitting the training data (ROC AUC test score: 76.07% and Neural Network Test Accuracy: 81.23%).

```
clf_NN = Sequential()
clf_NN.add(Dense(64, activation='relu', input_shape=(94,)))
clf_NN.add(Dropout(.3))
clf_NN.add(Dense(32, activation='relu'))
clf_NN.add(Dense(32, activation='relu'))
clf_NN.add(Dropout(.3))
clf_NN.add(Dense(2, activation='softmax'))
```

Adding in the dropout did in fact help on the test data a little bit (ROC AUC score: 76.82% and Accuracy: 81.78%). Since it seems that deeper networks are losing performance and more difficult to train, I'm opting to go back to the second version of the network I tried above with two hidden layers. On a final run, this achieved ROC AUC score: 77.24% and Accuracy: 81.83%.

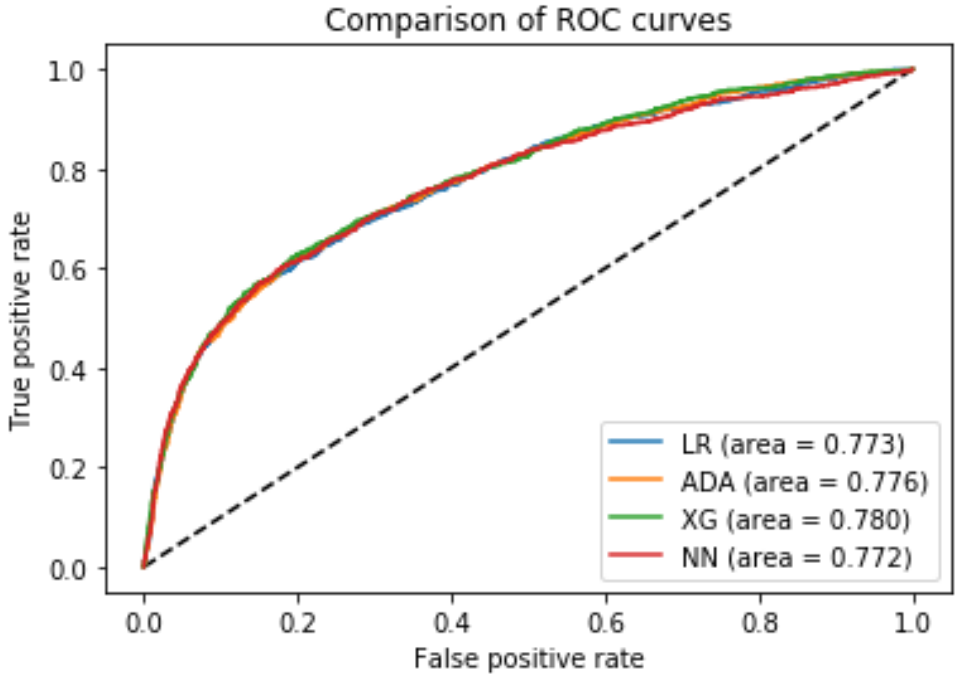
# IV. Results

## Model Evaluation and Validation

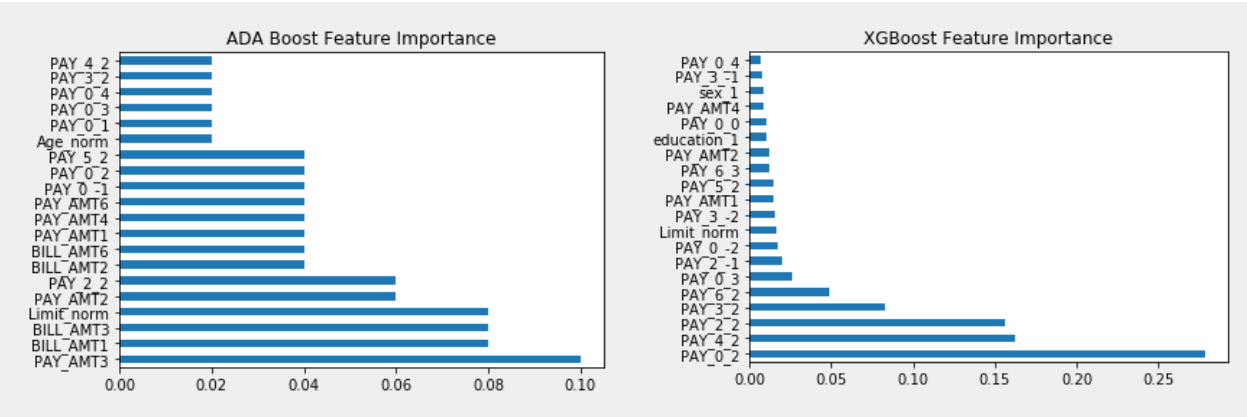
Ultimately, all of the models developed performed comparably. The below table summarizes the results obtained on the test data set:

| | Test Data Accuracy | Test Data ROC AUC |
|---|---|---|
| **Logistic Regression** | 81.95% | 77.33% |
| **ADA Boost** | 81.65% | 77.37% |
| **XG Boost** | 82.08% | 77.98% |
| **Artificial Neural Network** | 82.05% | 77.35% |

The following plot shows the similarity in ROC curves for all models.




Nonetheless, XGBoost performed the best in terms of both accuracy and ROC AUC. Interestingly, it seems that even though ADABoost and XGBoost are both boosted tree based algorithms, they derived very different feature importances:



ADA Boost seemed to weight more behavioral type variables, such as spend and payment amounts, whereas XG Boost seemed to favor the encoded payment history status of 2 months delayed payment. Intuitively, I believe that XGBoost's model is more understandable because it could be described as saying customers that have delayed payment 2 months at some point in 6 months are more likely to default payment next month. This also shows that this model is stable and will not be sensitive to small changes in the data since most of the data is not a strong predictor of default.

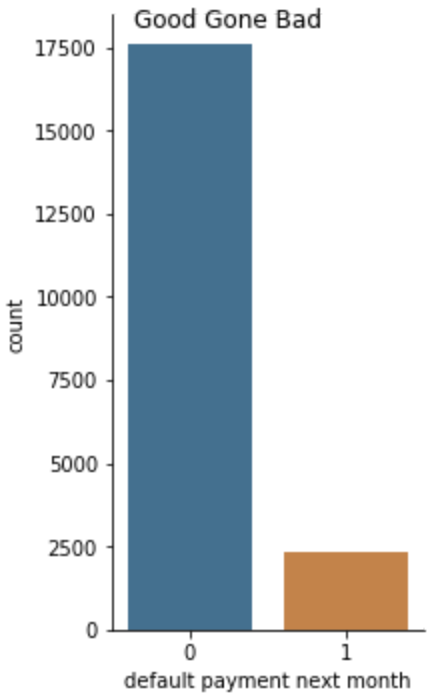
## Justification

The XGBoost model performed reasonably well against the benchmarks obtained from research and significantly better than a trivial classifier. While only [6] reported an AUC, the XGBoost model appears to have performed better along that metric: 77.98% versus 64.63%. In terms of accuracy, the XGBoost model did not perform as well as the best one seen in research: 82.08% versus 95.8%[7]. This hints at an area for improvement: the authors of [7] augmented the dataset. Comparing the XGBoost model to research that did not use augmented data, the accuracy is comparable (84% in [5] and 81.96% in [6]).

# V. Conclusion

## Free-Form Visualization

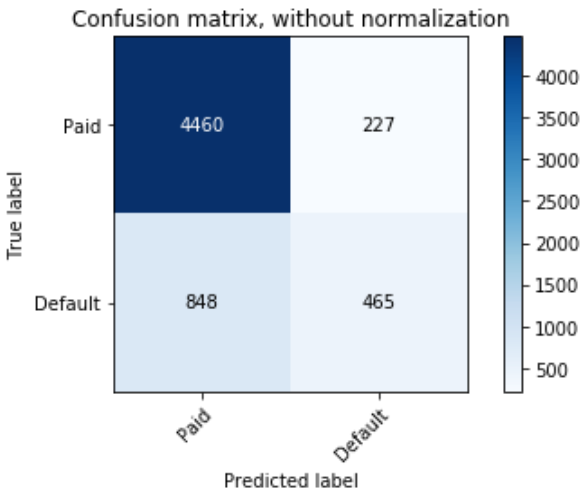
One key characteristic of the problem that can be observed in the data is that there is an inherent randomness in behavior due to many unobserved externalities. This plot shows clients that for the past 6 months had good payment history, then suddenly they default payment the next month. I do not believe that it is possible to predict these 2500 observations (almost 10% of the total data) with the data provided. To the extent that the demographic variables provide any explanatory value in a model to these observations, I do not believe that such a model will generalize well in different cultures or timeframes.



# Reflection

Reflecting on the solution obtained, I do not believe that is sufficiently solves the problem with enough certainty to be used in a business context. This is expected, though, because this dataset did not provide nearly the amount of information that would be available to a credit card issuer. At the same time, I think that what this solution shows is that even with a small and simplistic dataset, it is possible to reasonably predict default payment. I found it interesting that two boosted tree methods would split the data so differently. In the data exploration, it was observed that both bill amount/payment information and payment history correlated with default, but it seems that ADABoost and XGBoost each diverged into which of those features was more important. Thinking back to the beginning of the project, I found it difficult to deal with the undefined encoded values of categorical variables, but it was interesting to research them and make decisions about how to interpret them.

Going back to the business opportunity considered in the introduction, false negatives would cost a credit card issuer revenue. If a lower rate were given to someone who defaulted, then the impact is now greater. Observing the confusion matrix for XGBoost, the model's recall is not very strong when 65% of defaults in the test data were predicted as paid. At scale, this could be very costly to a card issuer. This is why I do not believe that problem is solved with enough certainty to be used in a business context.




# Improvement

I can see three areas for improvement where I think a better solution exists. One was identified in research where this dataset could be augmented by joining in additional data that would be included in credit card statements, primarily spending data. This was an unobserved variable that I believe lead to seeming randomness in the payment history encoding.

This inspires a second idea for improvement. To increase the amount of data, the data could be restructured such that each observation is only for one month. The payment history labels that were provided for each period could then become target data after being made into a binary variable. It seems that only paid duly, no consumption or revolving credit would be considered non-default, but this is an assumption.

Lastly, I do not believe that the algorithms used will perform as well without a concept of the sequence of time between the observation in each period. Further research into a baysian-like algorithm that would use prior knowledge could be promising.