

# Capstone Project

Randy Jackson

## Table of Contents

1	Definition .....	2
1.1	Project Overview .....	2
1.2	Problem Statement .....	2
1.3	Metrics.....	2
2	Analysis.....	4
2.1	Data Exploration .....	4
2.2	Exploratory Visualization .....	6
2.3	Algorithms And Techniques.....	7
2.4	Benchmark.....	8
3	Methodology .....	11
3.1	Data Preprocessing .....	11
3.2	Implementation .....	13
3.3	Refinement .....	24
4	Results .....	25
4.1	Model Evaluation and Validation .....	25
4.2	Justification.....	26
5	Conclusion .....	26
5.1	Free-Form Visualization.....	26
5.2	Reflection.....	27
5.3	Improvement.....	27
6	References.....	28

## 1 Definition

### 1.1 Project Overview

My capstone project is based on the Home Credit Default Risk competition on Kaggle.

Many people struggle to get loans due to insufficient or non-existent credit histories and, unfortunately, this population is often taken advantage of by untrustworthy lenders. Home Credit strives to broaden financial inclusion for the unbanked population by providing a positive and safe borrowing experience. In order to make sure this underserved population has a positive loan experience, Home Credit makes use of a variety of alternative data—including telco and transactional information—to predict their clients' repayment abilities [1].

Loans default can lead to huge losses for financial firms. Financial firms apply various methods to detect and predict default behaviors of their customers, considerable research and effort has been applied to this problem domain and entire courses are dedicated to this single problem.

### 1.2 Problem Statement

While Home Credit is currently using various statistical and machine learning methods to make these predictions, they're challenging Kagglers to help them unlock the full potential of their data. Doing so will ensure that clients capable of repayment are not rejected and that loans are given with a principal, maturity, and repayment calendar that will empower their clients to be successful [1].

The objective of this competition is to use historical loan application data to predict whether or not an applicant will be able to repay a loan. This is a supervised binary classification task. Supervised, in that the labels are included in the training data and the goal is to train a model to learn to predict the labels from the features. Binary Classification, in that the label is a binary variable, 0 (will repay loan on time), 1 (will have difficulty repaying loan). This value is stored in the TARGET feature. Numerous methods are commonly used for binary classification. For this project we will use the following models: Logistic Regression, Decision trees, Random Forests, Artificial Neural Networks, XGBoost. We will use each method, some with varying approaches to predict the classification of the test set provided for the competition with the goal of surpassing that of our baseline model and while attempting to competition score possible. In order to achieve this goal we will analyze, clean, scale and impute the data, assess feature importance and perform feature selection and attempt to tune and optimize hyperparameters.

### 1.3 Metrics

Inspection of the data indicates that approximately 92% of the data is representative of customers who repay their loans (i.e., 0) and approximately 8% associated with customers who default (i.e., 1)

```
train.TARGET.value_counts()/len(train)
0    0.919271
1    0.080729
Name: TARGET, dtype: float64
```

Figure 1 Class percentage of TARGET variable values

This represents a significant class imbalance. The Class Imbalance Problem is a common problem affecting machine learning due to having disproportionate number of class instances in practice. In learning extremely imbalanced data, the

overall classification accuracy is often not an appropriate measure of performance. A classifier that correctly predicts every case as the majority class can still achieve very high accuracy while performing inadequately against the minority class. In these instances it is best to use other metrics, such as the true negative rate, true positive rate, weighted accuracy, G-mean, precision, recall, and F-measure to evaluate the performance of learning algorithms on imbalanced data [15][7]. To compare solutions, we will include the alternate metrics (True Positive, True Negative, False Positive, False Negative) as well as Accuracy, with the primary metric being Area Under Curve (AUC).

True positive rate (TPR), aka. sensitivity, hit rate, and recall, which is defined as  $TP / (TP + FN)$ . Intuitively this metric corresponds to the proportion of positive data points that are correctly considered as positive, with respect to all positive data points. In other words, the higher TPR, the fewer positive data points we will miss.

False positive rate (FPR), aka. fall-out, which is defined as  $FP / (FP + TN)$ . Intuitively this metric corresponds to the proportion of negative data points that are mistakenly considered as positive, with respect to all negative data points. In other words, the higher FPR, the more negative data points will be misclassified. Since to compare two different models it is often more convenient to have a single metric rather than several ones, we compute two metrics from the confusion matrix, which we will later combine into one.

To combine the FPR and the TPR into one single metric, we first compute the two former metrics with many different threshold (for example 0.00;0.01,0.02,...,1.00) for the logistic regression, then plot them on a single graph, with the FPR values on the abscissa and the TPR values on the ordinate. The resulting curve is called ROC curve, and the metric we consider is the AUC of this curve, which we call AUROC. [5]

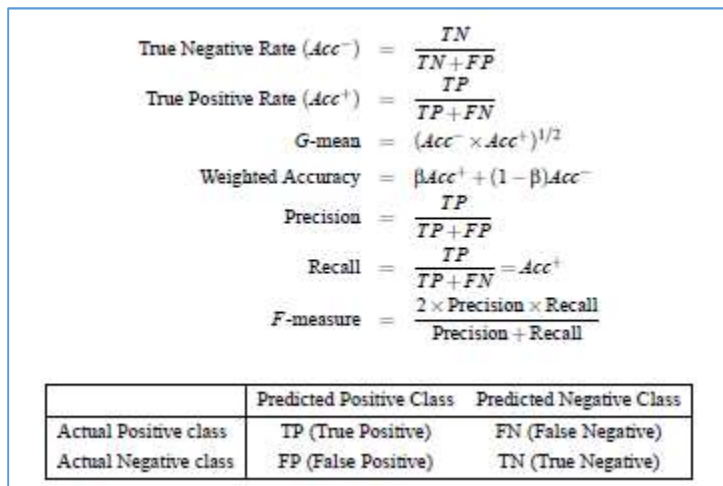


Figure 2 Evaluation Metrics

Due to the prevalence of the unbalanced classification problem many methods have been developed to deal with it. These methods are generally 1) sampling based, or 2) cost function based. Sampling based methods can be further divided into: a) over sampling b) under sampling c) hybrid of oversampling and undersampling [7]. We will investigate the results of using SMOTE and class weighting, these will be described in more detail in later sections.

The Kaggle competition submissions are evaluated based on area under the ROC curve between the predicted probability and the observed target. As previously noted, we will use the ROC curve approach as the primary evaluation criteria. However, for the purpose of supplying a more comprehensive project we will include results using at least two additional evaluation metrics covered in the MLND, such as Accuracy or F-score. **ROC is the metric to use when there is an imbalance and most samples are positive, so the best metric is already selected as a requirement of the project.**

## 2 ANALYSIS

### 2.1 Data Exploration

The data for this project is provided by Home Credit, the company hosting the competition. The training data has 307511 observations (each one a separate loan) and 122 features including the label we want to predict. The testing data has 48744 rows and 121 features. The following data information is taken from the Kaggle competition detail [1].

- **application\_{train|test}.csv**
  - This is the main table, broken into two files for Train (with TARGET) and Test (without TARGET).
  - Static data for all applications. One row represents one loan in our data sample.
- **bureau.csv**
  - All client's previous credits provided by other financial institutions that were reported to Credit Bureau (for clients who have a loan in our sample).
  - For every loan in our sample, there are as many rows as number of credits the client had in Credit Bureau before the application date.
- **bureau\_balance.csv**
  - Monthly balances of previous credits in Credit Bureau.
  - This table has one row for each month of history of every previous credit reported to Credit Bureau – i.e the table has (#loans in sample \* # of relative previous credits \* # of months where we have some history observable for the previous credits) rows.
- **POS\_CASH\_balance.csv**
  - Monthly balance snapshots of previous POS (point of sales) and cash loans that the applicant had with Home Credit.
  - This table has one row for each month of history of every previous credit in Home Credit (consumer credit and cash loans) related to loans in our sample – i.e. the table has (#loans in sample \* # of relative previous credits \* # of months in which we have some history observable for the previous credits) rows.
- **credit\_card\_balance.csv**
  - Monthly balance snapshots of previous credit cards that the applicant has with Home Credit.
  - This table has one row for each month of history of every previous credit in Home Credit (consumer credit and cash loans) related to loans in our sample – i.e. the table has (#loans in sample \* # of relative previous credit cards \* # of months where we have some history observable for the previous credit card) rows.
- **previous\_application.csv**
  - All previous applications for Home Credit loans of clients who have loans in our sample.
  - There is one row for each previous application related to loans in our data sample.
- **installments\_payments.csv**
  - Repayment history for the previously disbursed credits in Home Credit related to the loans in our sample.
  - There is a) one row for every payment that was made plus b) one row each for missed payment.
  - One row is equivalent to one payment of one installment OR one installment corresponding to one payment of one previous Home Credit credit related to loans in our sample.
- **HomeCredit\_columns\_description.csv**
  - This file contains descriptions for the columns in the various data files.

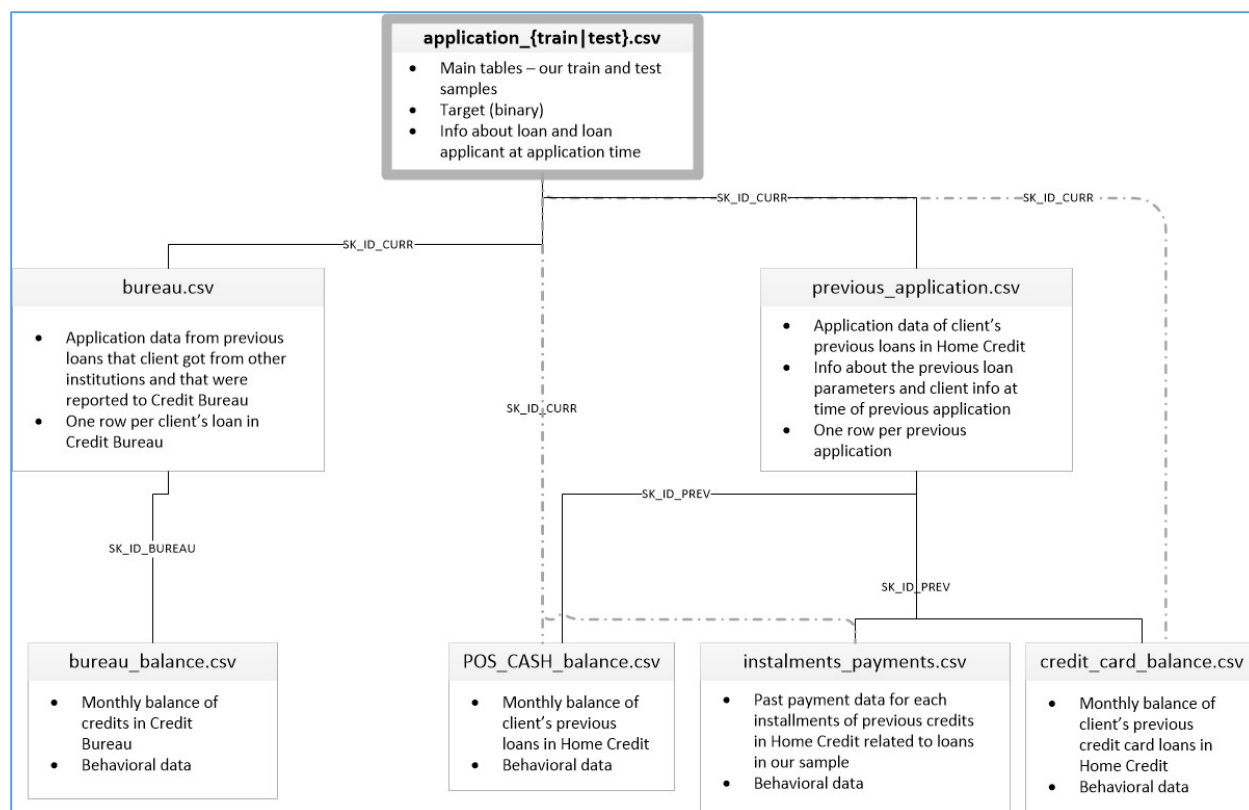


Figure 3 Kaggle Loan Default Competition Data

The primary data is contained in the **application\_{train|test}.csv** files. I have chosen to complete the project in two phases. The first phase of the project attempts to make [predictions based solely on the data contain in application\_{train|test}]. In the second phase of the project we will aggregate and join data from the other supporting files and repeat the prediction using our final model with feature selection. Below is a sample of the **application\_{train|test}** data file.

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT
0	100002	1	Cash loans	M	N	Y	0	202500.0	406500.0
1	100003	0	Cash loans	F	N	N	0	270000.0	129350.0
2	100004	0	Revolving loans	M	Y	Y	0	67500.0	135000.0
3	100006	0	Cash loans	F	N	Y	0	135000.0	312600.0
4	100007	0	Cash loans	M	N	Y	0	121500.0	513000.0

5 rows x 122 columns

Figure 4 Data Example

## 2.2 Exploratory Visualization

Visualizations to support the above



Figure 5 Distribution of payers vs defaulters

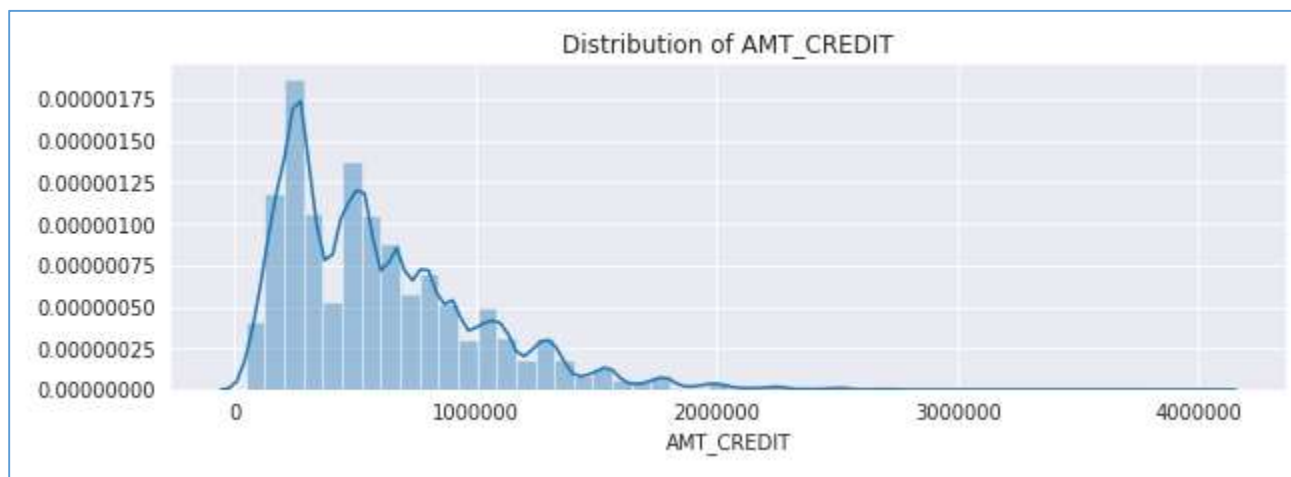


Figure 6 Distribution of Credit Amount

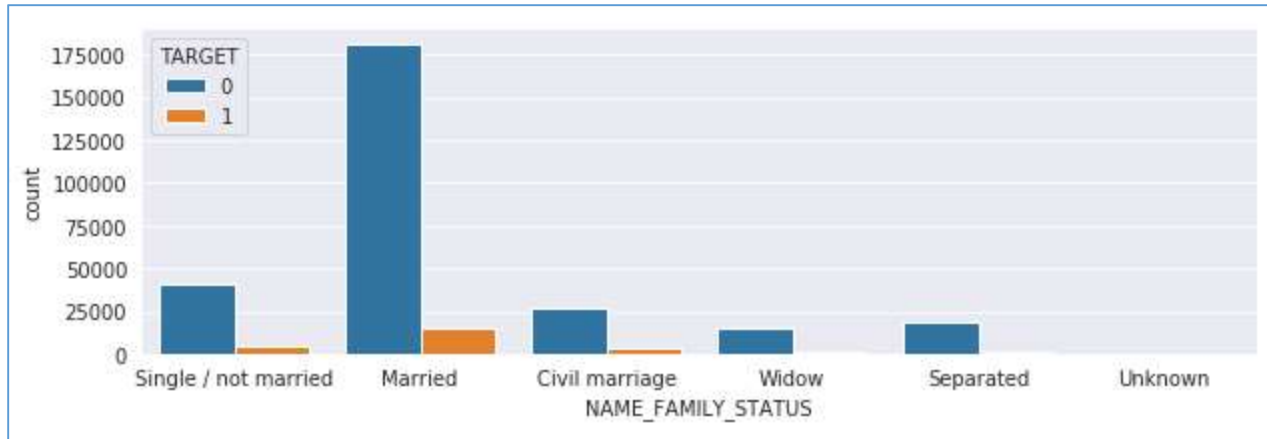


Figure 7 Distribution of Family Status vs Pay & Default

## 2.3 Algorithms And Techniques

First we'll start by describing the Algorithms we chosen to use and why:

### Logistic Regression

- Strengths: Outputs have a nice probabilistic interpretation, and the algorithm can be regularized to avoid overfitting. Logistic models can be updated easily with new data using stochastic gradient descent. [ref: Strengths and Weaknesses of ML Algorithms](#)
- Weaknesses: Logistic regression tends to underperform when there are multiple or non-linear decision boundaries. They are not flexible enough to naturally capture more complex relationships.
- Justification: Predictions are mapped to be between 0 and 1 through the logistic function, which means that predictions can be interpreted as class probabilities.

### Decision Trees

- Strengths: Decision trees implicitly perform variable screening or feature selection and require relatively little effort from users for data preparation. Nonlinear relationships between parameters do not affect tree performance • [4 Key Decision Tree Advantages](#) • [Decision Tree Advantages](#) • [Decision Trees in ML](#)
- Weaknesses: Decision trees are relatively easy to understand when there are few decisions and outcomes included in the tree. Large trees that include dozens of decision nodes (spots where new decisions are made) can be convoluted and may have limited value. The more decisions there are in a tree, the less accurate any expected outcomes are likely to be • [Decision Tree Advantages/Disadvantages](#) • [Decision Tree Disadvantages](#)
- Justification: A major decision tree analysis advantage is its ability to assign specific values to problem, decisions, and outcomes of each decision. This reduces ambiguity in decision-making. This problem lends itself to this major advantage.

### Random forests

- Strength: Random Forest is one of the most accurate learning algorithms available. For many data sets, it produces a highly accurate classifier [20] Random Forest also runs efficiently on large databases and can handle a large number of features before requiring variable deletion. It support feature importance and has methods for balancing error in class population unbalanced data sets.

- Weaknesses: Random forests have been observed to overfit for some datasets with noisy classification/regression tasks.
- Justification: We have a very large set of data and the need for determining feature importance

### Neural networks

- Strength: ANNs have the ability to learn and model non-linear and complex relationships, which is really important because in real-life, many of the relationships between inputs and outputs are non-linear as well as complex. ANNs can generalize - After learning from the initial inputs and their relationships, it can infer unseen relationships on unseen data as well, thus making the model generalize and predict on unseen data[21].
- Weaknesses: Unexplained behavior of the network: This is the most important problem of ANN. When ANN produces a solution (or fails to produce one), it does not give a clue as to why and how. There is no specific rule for determining the structure of artificial neural network, success is achieved primarily through trial and error.
- Justification: The competition test data set is only about 20% of the actual test data, the competition winners will ultimately be judged on a very large set of unseen data.

### XGBoost

- Strengths: Has similar advantages as Gradient Boosting (which it is a dialect of) plus the addition of higher execution speed and model performance. It handles missing data, weighted classification, and provides feature importance. It has a reputation as the winning algorithm used in most competitions.
- Weaknesses: They are prone to overfitting
- Justification: Speed and performance necessitated by a large dataset and the need for determining feature importance.

Logistic Regression was chosen primarily to provide a baseline. All of the selected algorithms support classification and have some method of dealing with imbalanced classification, they all support class weighting. We also leveraged oversampling with SMOTE primarily for the sake of comparison to the class weighting approach. Each has at least one notable advantage that is applicable to the problem domain or data.

## 2.4 Benchmark

A champion / challenger approach can be used when doing benchmarking. The approach uses the current model as the champion and current benchmark, which is then challenged by a new model. If the challenger model beats the champion in performance, then it can become the new champion and the new benchmark. This way, models are continuously challenged and further perfected. We started by defining a naive solution as a very simple model using Logistic as the initial champion, the model was used with the default parameters with the exception of the C parameter.

The model performed with high accuracy, almost 92%, but as noted earlier this did not tell the full story



Fit_Report(best_model, XTrain, YTrain, XTest, YTest)				
	precision	recall	f1-score	support
0	0.92	1.00	0.96	56554
1	0.00	0.00	0.00	4949
avg / total	0.85	0.92	0.88	61503
Predictor: [Accuracy score: 0.9192]				

Figure 8 Classification report for Logistic (first run)

We can see from the Confusion Matrix that the model did a poor job with the 1's. No 1's were predicted and, true 1's were predicted as 0. The model is not catching the 1's, this is a result of the **class imbalance problem** where the minority class is not being predicted by the model. The high accuracy value is based on the majority class, and is misleading. There are only .08 of 1's, in the data set as noted above in the metrics section. A good model needs to predict that 8%.

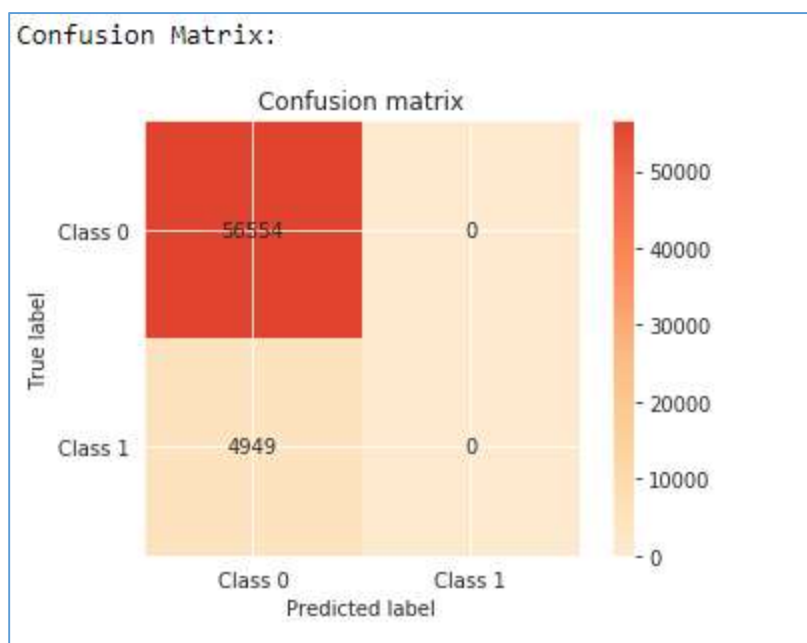


Figure 9 Confusion matrix for Logistic (first run)

The initial Logistic Regression Model yielded an AUC of .68 (note that this prior to accounting for class imbalance)

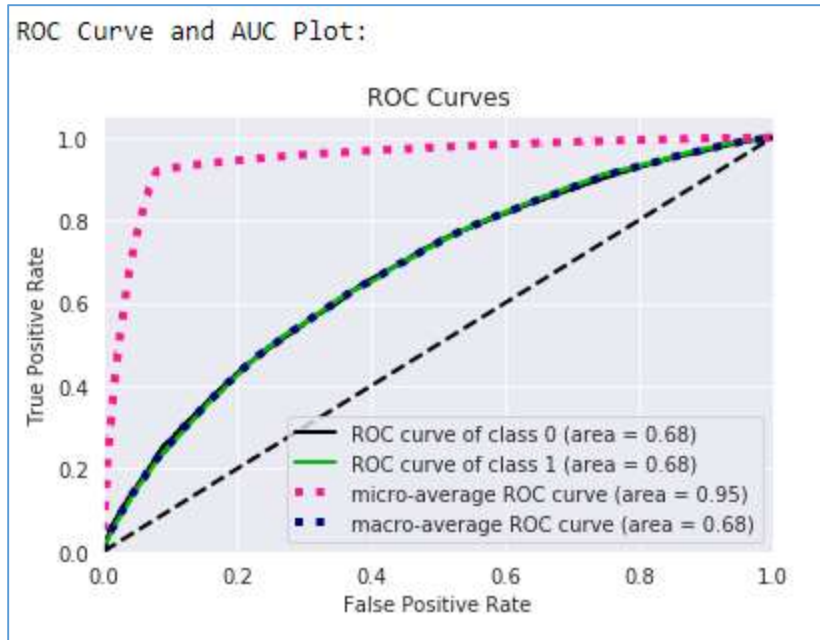


Figure 10 ROC/AUC for Logistic

Next we decided to take into account the class imbalance and use the class weighting and retry Logistic Regression to see if it would fare any better. The following code was used to compute the class weights and store them in a dictionary. This dictionary object was then provided as an argument to the **class\_weight** parameter of the model, and also used with subsequent model fittings.

```
from sklearn.utils import class_weight
classes=np.unique(YTrain) # Get the classes
class_weights = class_weight.compute_class_weight('balanced', classes, YTrain) # compute the class weights
class_weight_dict = dict(zip(classes, class_weights)) #turn into a dictionary for call to LogisticRegression
class_weight_dict

{0: 0.543947782710983, 1: 6.188569128597304}
```

Figure 11 Code to obtain class weights

The subsequent execution of Logistic Regression yielded a lower Accuracy score of 65%

Fit_Report(weighted_model, XTrain, YTrain, XTest, YTest)				
	precision	recall	f1-score	support
0	0.96	0.65	0.77	56554
1	0.14	0.67	0.24	4949
avg / total	0.89	0.65	0.73	61503
Predictor: [Accuracy score: 0.6509]				

Figure 12 Logistic classification report after applying class weights

However, the number of 1's increased dramatically! We also got a 3% increase in our UAC metric. Next as part of our methodology and implementation we will try several other models and another approach (SMOTE) to handling imbalanced classification, and use our **Logistic Regression with class weights 71% AUC as our baseline** for comparing other model results.

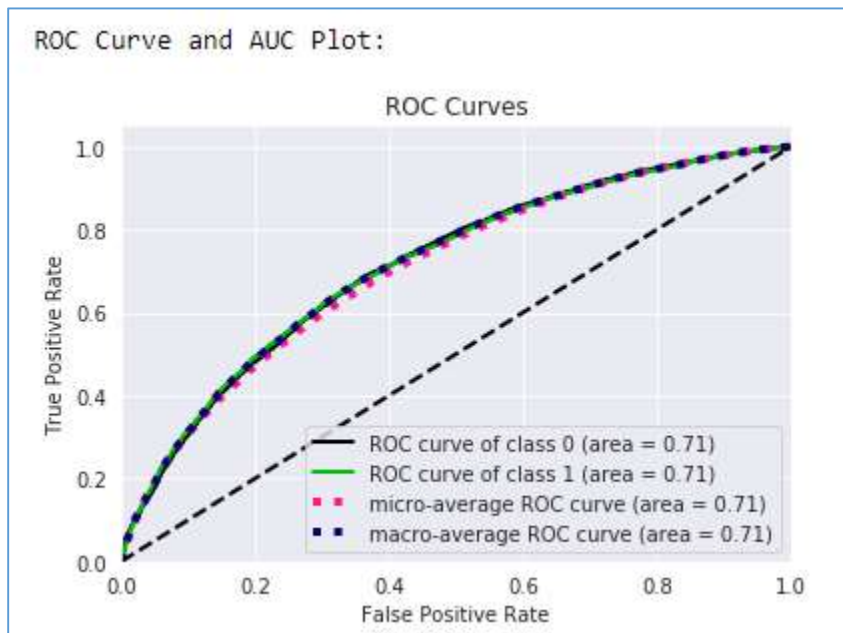


Figure 13 AUC after applying class weights

### 3 METHODOLOGY

#### 3.1 Data Preprocessing

**NOTE:** In an effort to avoid duplicating the entire notebook here I will focus on the key items. Please note that the notebook file contains additional details and is heavily commented.

One of the anomalies noticed in the application\_{train|test} file was that make value of DAYS\_EMPLOYED, its large but also positive while the rest of the days worked as well as age are stored as negative. The value of 365243 appeared to be some sort of outlier that affected about 8.66% of the loans and about 5.4% of defaults. This value was replaced with null and a new column created to indicate the anomalous rows[4].

```
app_train['DAYS_EMPLOYED'].describe()

count    307511.000000
mean      63815.045904
std     141275.766519
min     -17912.000000
25%     -2760.000000
50%     -1213.000000
75%     -289.000000
max     365243.000000
Name: DAYS_EMPLOYED, dtype: float64
```

Figure 14 Anomaly is days employed

After the correction the resulting histogram was well formed

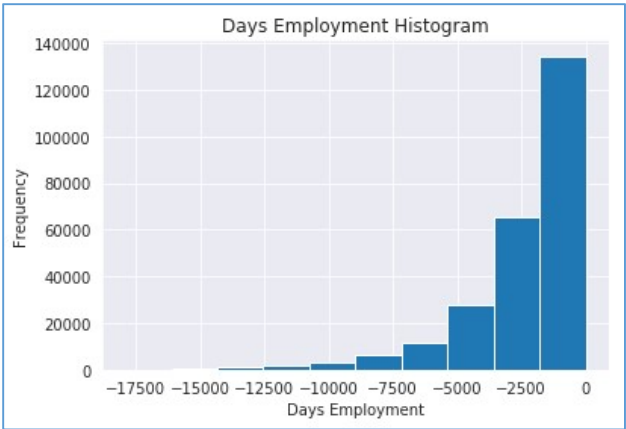


Figure 15 Distribution of days employed after fix

Another area of concern was with the AMT\_INCOME\_TOTAL, the range of values was heavily skewed.

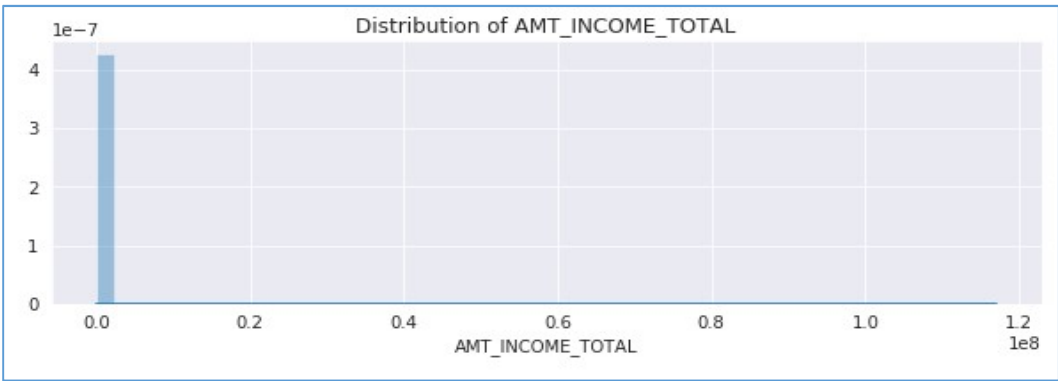


Figure 16 Skewed Amount\_Income\_Total

I performed a logarithmic transformation on this feature prior to scaling, which resulting in a well-shaped bellcurve.

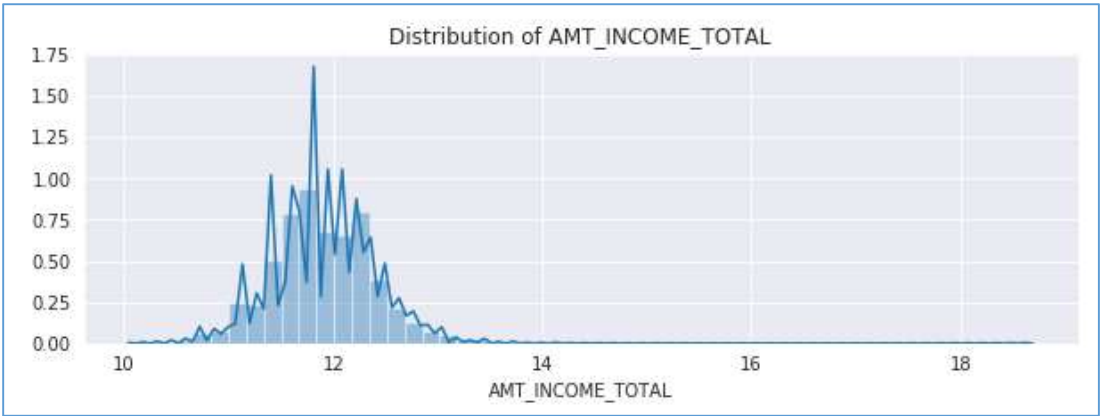


Figure 17 After Logarithmic transformation

We added some additional custom features [4]

```
# Aggregate the supplemental datasets so that they have a single row per customer

# add a few new features
app_train['LOAN_INCOME_RATIO'] = app_train['AMT_CREDIT'] / app_train['AMT_INCOME_TOTAL']
app_train['ANNUITY_INCOME_RATIO'] = app_train['AMT_ANNUITY'] / app_train['AMT_INCOME_TOTAL']
app_train['ANNUITY_LENGTH'] = app_train['AMT_CREDIT'] / app_train['AMT_ANNUITY']
app_train['DAYS_EMPLOYED_PERCENT'] = app_train['DAYS_EMPLOYED'] / app_train['DAYS_BIRTH']

print('Combined train & test shape before merging = {}'.format(app_train.shape))

Combined train & test shape before merging = (307511, 127)
```

Figure 18 We added a few custom features

We also removed columns that had greater than 25% missing values

```
Columns dropped for being over 25.0% null:
['OWN_CAR_AGE', 'OCCUPATION_TYPE', 'EXT_SOURCE_1', 'APARTMENTS_AVG', 'BASEMENTAREA_AVG', 'YEARS_BEGINEXPLUATATION_AVG', 'YEARS_BUILD_AVG', 'COMMONAREA_AVG', 'ELEVATORS_AVG', 'ENTRANCES_AVG', 'FLOORSMAX_AVG', 'FLOORSMIN_AVG', 'LANDAREA_AVG', 'LIVINGAPARTMENTS_AVG', 'LIVINGAREA_AVG', 'NONLIVINGAPARTMENTS_AVG', 'NONLIVINGAREA_AVG', 'APARTMENTS_MODE', 'BASEMENTAREA_MODE', 'YEARS_BEGINEXPLUATATION_MODE', 'YEARS_BUILD_MODE', 'COMMONAREA_MODE', 'ELEVATORS_MODE', 'ENTRANCES_MODE', 'FLOORSMAX_MODE', 'FLOORSMIN_MODE', 'LANDAREA_MODE', 'LIVINGAPARTMENTS_MODE', 'LIVINGAREA_MODE', 'NONLIVINGAPARTMENTS_MODE', 'NONLIVINGAREA_MODE', 'APARTMENTS_MODE', 'BASEMENTAREA_MODE', 'YEARS_BEGINEXPLUATATION_MODE', 'YEARS_BUILD_MODE', 'COMMONAREA_MODE', 'ELEVATORS_MODE', 'ENTRANCES_MODE', 'FLOORSMAX_MODE', 'FLOORSMIN_MODE', 'LANDAREA_MODE', 'LIVINGAPARTMENTS_MODE', 'LIVINGAREA_MODE', 'NONLIVINGAPARTMENTS_MODE', 'NONLIVINGAREA_MODE', 'APARTMENTS_MODE', 'BASEMENTAREA_MODE', 'YEARS_BEGINEXPLUATATION_MODE', 'YEARS_BUILD_MODE', 'COMMONAREA_MODE', 'ELEVATORS_MODE', 'ENTRANCES_MODE', 'FLOORSMAX_MODE', 'FLOORSMIN_MODE', 'LANDAREA_MODE', 'LIVINGAPARTMENTS_MODE', 'LIVINGAREA_MODE', 'NONLIVINGAPARTMENTS_MODE', 'NONLIVINGAREA_MODE', 'FONDKAPREMONT_MODE', 'HOUSETYPE_MODE', 'TOTALAREA_MODE', 'WALLSMATERIAL_MODE', 'EMERGENCYSTATE_MODE']
```

Figure 19 Columns with lots of nulls

And finally we imputed and scaled the data prior to executing our models

```
imputed_features = Imputer(strategy = 'median').fit_transform(train.values)
imputed_features_df = pd.DataFrame(imputed_features, index=train.index, columns=train.columns)
scaled_features = MinMaxScaler(feature_range = (0, 1)).fit_transform(imputed_features_df.values)
train = pd.DataFrame(scaled_features, index=imputed_features_df.index, columns=imputed_features_df.columns)
```

Figure 20 Imputation and Scaling

Feature selection was performed using the XBoost Model, and described in the implementation section that follows.

### 3.2 Implementation

Solution Architecture: Programming Language and Libraries

- Python 3.
- scikit-learn. Open source machine learning library for Python.
- Keras – was used to create the Artificial Neural Network
- sklearn\_evaluation – was used to provide additional plotting capabilities
- Matplotlib and Seaborn were used for plotting
- time and datetime – were used to time some functions and display timing results
- gc – Garbage collection was used in an attempt to help manage memory resources associated with large dataframes
- operator – was used to provide the use of itemgetter
- scipy.stats – was used to support the generation of parameters ranges for random search
- imblearn.over\_sampling – was used to import SMOTE
- tensorflow, functools – was used to support the creation of a AUC metric for the Keras ANN

Project Design: The general sequence of steps are as follows-

- a. Data Visualization: Visual representation of data to find the degree of correlations between predictors and target variable and find out correlated predictors. Additionally, we can see ranges and visible patterns of the predictors and target variable. -
- b. Data Preprocessing: Scaling and Normalization operations on data and splitting the data in training, validation and testing sets.
- c. Feature Engineering: Finding relevant features, engineer new features using methods like PCA if feasible.
- d. Model Selection: Experiment with various algorithms to find out the best algorithm for this use case.
- e. Model Tuning: Fine tune the selected algorithm to increase performance without overfitting.
- f. Testing: Test the model on testing dataset.

The models we implemented in the following sequence and manner

Phase 1 – Using only application\_{train|test}

1. Logistic Regression – no optimization other than C
2. Logic Regression – no optimization other than C, plus class\_weights
3. Decision Tree – Optimized using GridSearchCV
4. Random Forest – Optimized using RandomSearchCV, with class\_weights
5. Random Forest – Optimized using RandomSearchCV, with SMOTE
6. Artificial Neural Network – with class-weights
7. XGBoost – with class\_weights
8. XGBoost – with feature importance and feature selection

Phase 2 – Supplementing the only application\_{train|test} data with bureau and previous loan data from the other provided files

1. XGBoost – with class\_weights
2. XGBoost – with feature importance and feature selection

**NOTE:** *In an effort to avoid duplicating the entire notebook here I will re-insert the numbered lists from Phase 1 and Phase 2 above and insert and focus on the key items, learnings and outcomes of each step. Please note that the notebook file contains additional details and is heavily commented.*

Phase 1 – Using only application\_{train|test}

1. Logistic Regression – no optimization other than C : **(Covered above in the baseline section)**
2. Logic Regression – no optimization other than C, plus class\_weights: **(Covered above in the baseline section)**
3. Decision Tree – Optimized using GridSearchCV:

Top parameters of gridsearch

```
{'criterion': 'entropy',
 'max_depth': 5,
 'min_samples_leaf': 5,
 'min_samples_split': 10}
```

Figure 21 Decision Tree grid search results



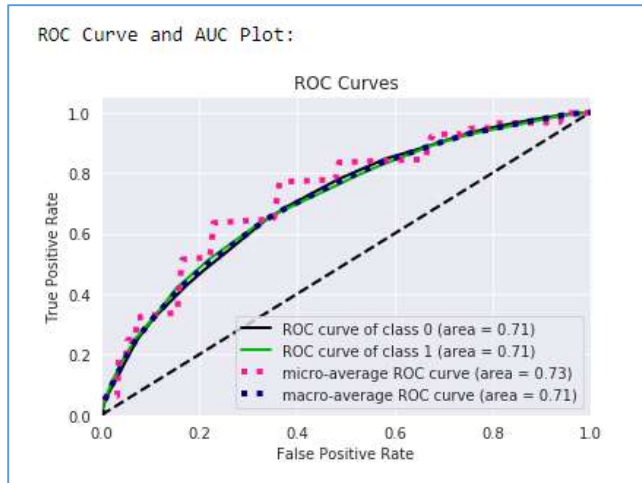


Figure 22 Decision Tree AUC (.71)

#### 4. Random Forest – Optimized using RandomSearchCV, with class\_weights

```
RandomizedSearchCV took 982.40 seconds for 10 candidates parameter settings.
Model with rank: 1
Mean validation score: 0.743 (std: 0.002)
Parameters: {'n_estimators': 150, 'min_samples_split': 2, 'min_samples_leaf': 50, 'max_features': 'auto', 'max_depth': None, 'criterion': 'gini', 'bootstrap': False}
```

Figure 23 RandomSearchCV Random Forest top result

**Using class weights** - Weighted random forest is one approach to make random forest more suitable for learning from extremely imbalanced data follows the idea of cost sensitive learning. Since the RF classifier tends to be biased towards the majority class, we shall place a heavier penalty on misclassifying the minority class. We assign a weight to each class, with the minority class given larger weight (i.e., higher misclassification cost). The class weights are incorporated into the RF algorithm in two places. In the tree induction procedure, class weights are used to weight the Gini criterion for finding splits. In the terminal nodes of each tree, class weights are again taken into consideration. The class prediction of each terminal node is determined by “weighted majority vote”; i.e., the weighted vote of a class is the weight for that class times the number of cases for that class at the terminal node. The final class prediction for RF is then determined by aggregating the weighted vote from each individual tree, where the weights are average weights in the terminal nodes. Class weights are an essential tuning parameter to achieve desired performance [15].

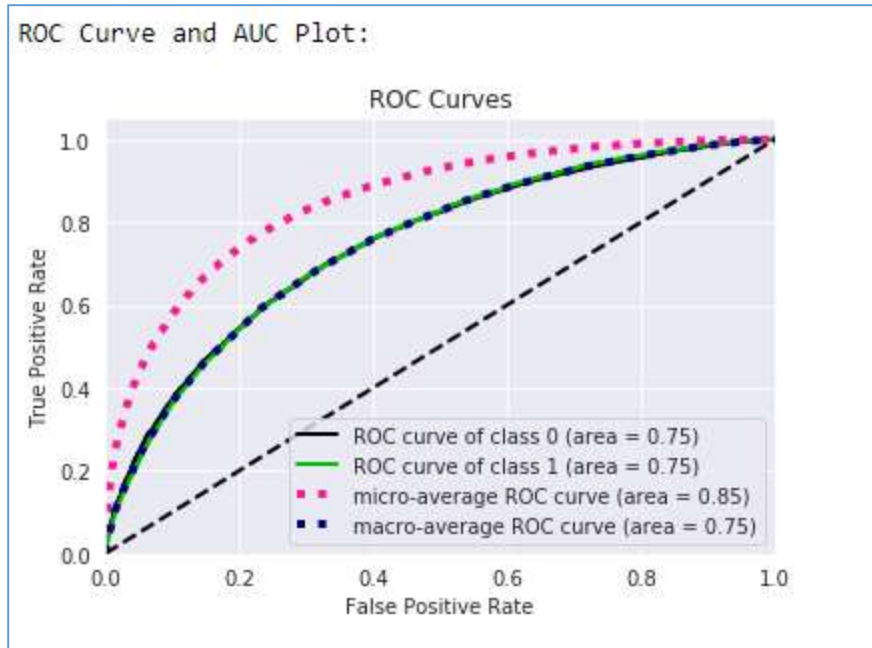


Figure 24 Random Forest UAC with class weights

#### 5. Random Forest – Optimized using RandomSearchCV, with SMOTE

Using SMOTE – Synthetic Minority Over-sampling Technique (SMOTE) is another approach to handling the imbalanced classification problem. At a high level, SMOTE creates synthetic observations of the minority class (bad loans) by finding the k-nearest-neighbors for minority class observations (finding similar observations) and randomly choosing one of the k-nearest-neighbors and using it to create a similar, but randomly tweaked, new observation. After up-sampling to a class ratio of 1.0, we should have a balanced dataset. There's no need (and often it's not smart) to balance the classes, but it magnifies the issue caused by incorrectly timed oversampling. The right way to oversample is to apply it to the training data only.

```
x_train, x_val, y_train, y_val = train_test_split(XTrain, YTrain,
                                                test_size = .2,
                                                random_state=RANDOM_SEED)
```

By oversampling only on the training data, none of the information in the validation data is being used to create synthetic observations. So these results should be generalizable. Let's see if that's true.

```
sm = SMOTE(random_state=RANDOM_SEED, ratio = 1.0)
x_train_res, y_train_res = sm.fit_sample(x_train, y_train)
```

Figure 25 Setup of SMOTE

Here we check to see if we are going in the right direction with our SMOTE implementation, before completing the model

Our SMOTE Random Forest results were a bit less than our class weight approach with Random Forest



```

from sklearn.metrics import recall_score
print ('Validation Results')
print (clsRF.score(x_val, y_val))
print (recall_score(y_val, clsRF.predict(x_val)))
print ('\nTest Results')
print (clsRF.score(XTest, YTest))
print (recall_score(YTest, clsRF.predict(XTest)))

```

Validation Results  
 0.9030323970570302  
 0.08213755566551212

Test Results  
 0.90395590459002  
 0.08648211759951506

The validation results closely match the unseen test data results, which is exactly what I would want to see after putting a model into production.

Figure 26 SMOTE setup sanity check

This is the AUC .71 after conducting the SMOTE operation with Random Forest, it is .4 less that the Random Forest approach using class weights, but it is equal to the AUC of the Logistic Regression approach using class weights.

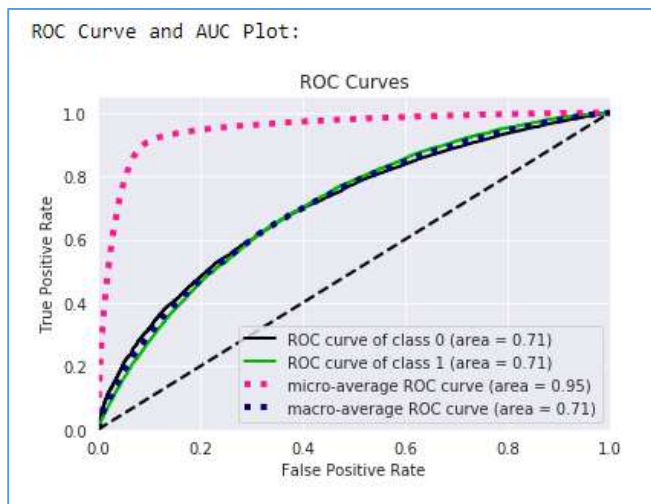


Figure 27 Random Forest UAC with SMOTE

- Artificial Neural Network – with class-weights
- Use Tensorflow to create an AUC method which KERAS natively doesn't have

```

def as_keras_metric(method):
    import functools
    from keras import backend as K
    import tensorflow as tf
    @functools.wraps(method)
    def wrapper(self, args, **kwargs):
        """ Wrapper for turning tensorflow metrics into keras metrics """
        value, update_op = method(self, args, **kwargs)
        K.get_session().run(tf.local_variables_initializer())
        with tf.control_dependencies([update_op]):
            value = tf.identity(value)
        return value
    return wrapper

auc_roc = as_keras_metric(tf.metrics.auc)
recall = as_keras_metric(tf.metrics.recall)

```

Figure 28 KERAS AUC Metric Setup

Set up our network architecture, we used a smaller hidden layer as it tends to force the model to quickly identify the key features.

```
from sklearn.utils import class_weight
class_weights = class_weight.compute_class_weight('balanced',
                                                  np.unique(YTrain),
                                                  YTrain)

model = Sequential()
model.add(Dense(32, input_dim=165, activation='relu', kernel_initializer='normal'))
model.add(Dense(16, activation='relu', kernel_initializer='normal'))
model.add(Dense(units=1, activation='sigmoid', kernel_initializer='normal'))

model.compile(loss='binary_crossentropy',
              optimizer=optimizers.Adam(lr=0.001), metrics=[auc_roc])

model.fit(XTrain, YTrain, epochs=150, batch_size=512, class_weight=class_weight_dict)
```

Figure 29 Neural Network architecture

One issue we encountered with the ANN was that it would not converge. Many hours were spent trying to optimize the model and experiment with different parameters and values. Eventually it was determined the SK\_ID\_CURR still included in the training data was preventing the model from training. We were able to remove the SK\_ID\_CURR and the model would train, with an AUC of .74, simply adding the single SK\_ID\_CURR back into the data would cause the model to no longer converge. While it is true that SK\_ID\_CURR is a key in the data rather than a feature associated with an attribute of the customer one might have expected a degraded prediction as opposed to a complete failure to train.

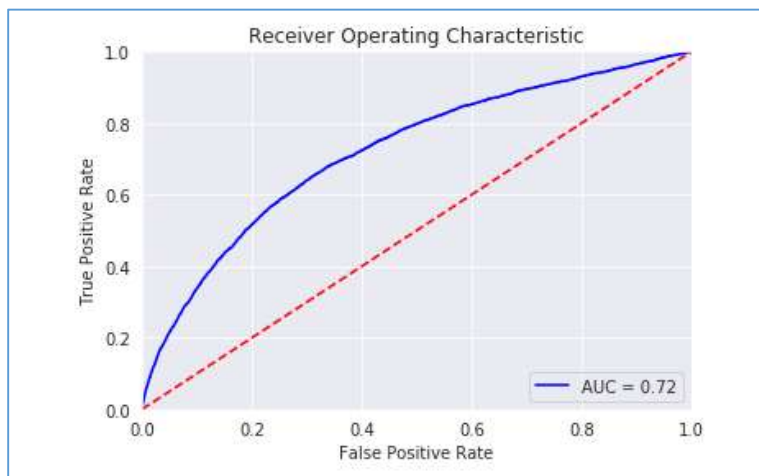


Figure 30 ANN all features (164)

## 8. XGBoost – with class\_weights

```
RandomizedSearchCV took 305.22 seconds for 10 candidates parameter settings.
Model with rank: 1
Mean validation score: 0.919 (std: 0.000)
Parameters: {'colsample_bytree': 0.9644472728421691, 'gamma': 6.912264934416453, 'learning_rate': 0.5, 'max_depth': 3, 'min_chi
            id_weight': 45.905603608711495, 'n_estimators': 30, 'reg_alpha': 10.168338716685366, 'subsample': 0.9144011957370972}
```

Figure 31 XGBoost RandomizedSearchCV top parameters

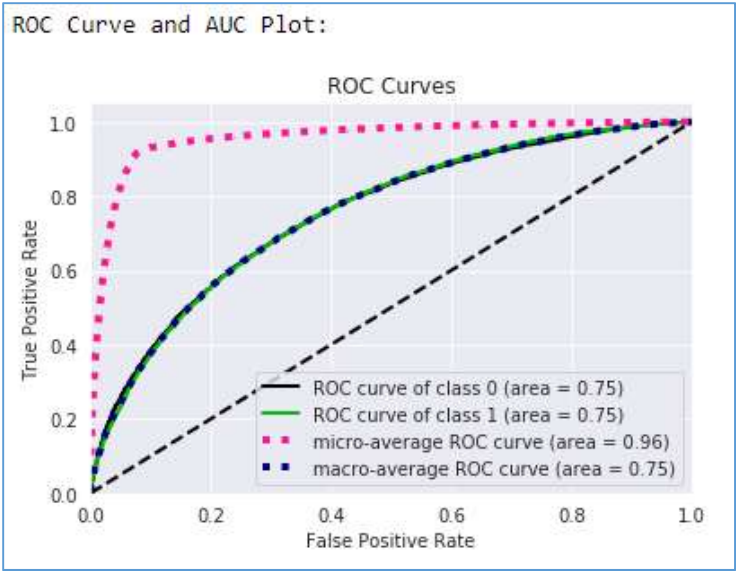


Figure 32 XGBoost AUC (.75) with class weights

9. XGBoost – with feature importance and feature selection

Ranking	importance	cumulative	feature
0	1	0.15	ANNUITY_LENGTH
1	2	0.25	EXT_SOURCE_3
2	3	0.34	EXT_SOURCE_2
3	4	0.39	AMT_GOODS_PRICE
4	5	0.44	DAYS_BIRTH
5	6	0.48	AMT_ANNUITY
6	7	0.52	DAYS_ID_PUBLISH
7	8	0.56	ANNUITY_INCOME_RATIO
8	9	0.59	AMT_CREDIT
9	10	0.62	DAYS_EMPLOYED
10	11	0.65	DEF_30_CNT_SOCIAL_CIRCLE
11	12	0.67	FLAG_OWN_CAR
12	13	0.69	DAYS_REGISTRATION
13	14	0.71	REGION_RATING_CLIENT_W_CITY
14	15	0.73	LOAN_INCOME_RATIO
15	16	0.75	DAYS_EMPLOYED_PERCENT

Figure 33 XGBoost AUC featue importance

This plot shows the cumulative impotence and features required to account for 90%

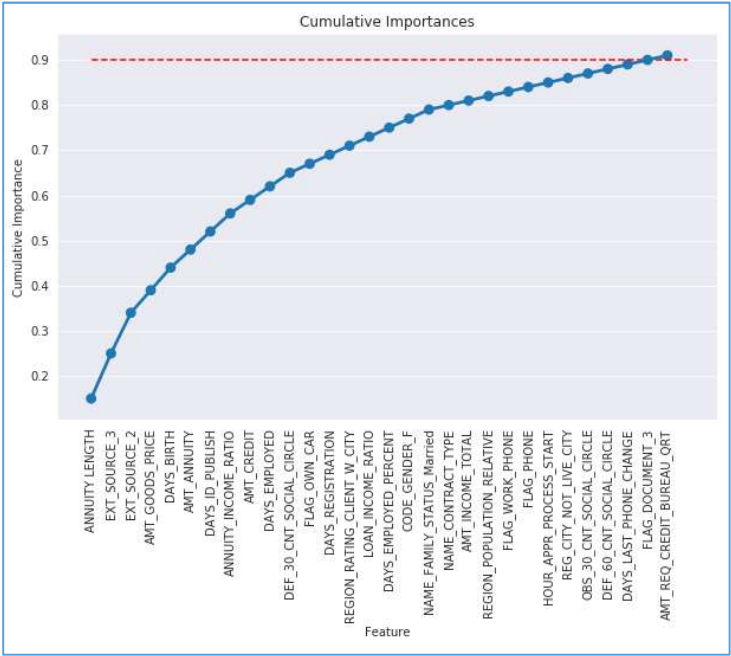


Figure 34 Cumulative feature importance

How does a model perform if we only use a subset of all the available features in the data? With less features required to train, the expectation is that training and prediction time is much lower — at the cost of performance metrics. From the visualization above, we see that the top five most important features contribute more than half of the importance of all features present in the data. This hints that we can attempt to reduce the feature space and simplify the information required for the model to learn. The code cell below will use the same optimized model you found earlier, and train it on the same training set with only the top most important features.

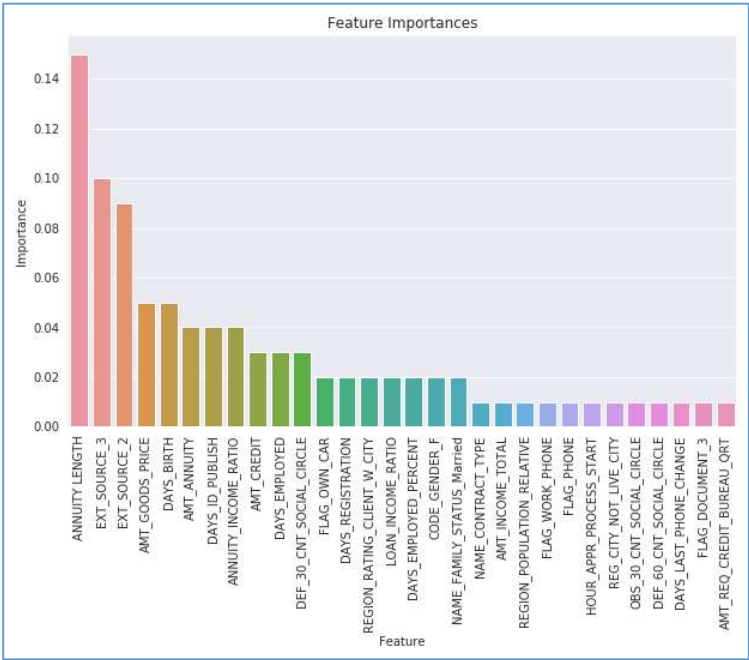


Figure 35 Feature importance ranking

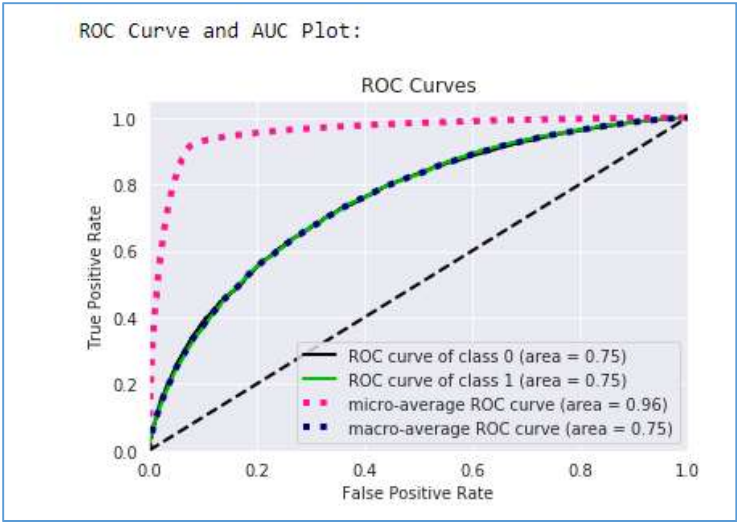
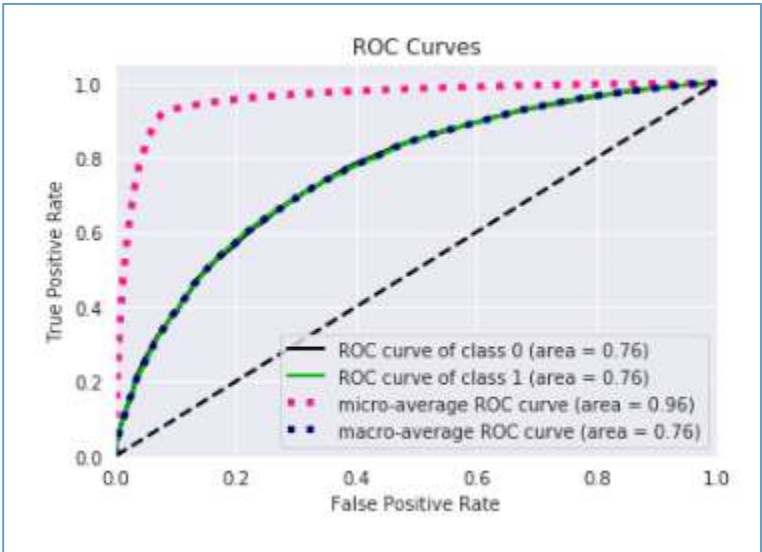


Figure 36 XGBoost using important features

Phase 2 – In the previous phase we used only application\_{train|test} data. However, in phase 2 we supplement the data set with that contained in bureau and previous loan and application data from the other provided files. The process is discussed below in the refinement section.

1. XGBoost – with class\_weights after merging the data



## 2. XGBoost – with feature importance and feature selection

	Ranking	importance	cumulative	feature
0	1	0.13	0.13	EXT_SOURCE_2
1	2	0.11	0.24	EXT_SOURCE_3
2	3	0.10	0.34	SK_DPD_DEF_CAVG
3	4	0.05	0.39	CNT_INSTALMENT_FUTURE
4	5	0.03	0.42	DAYS_EMPLOYED
5	6	0.03	0.45	ANNUITY_LENGTH
6	7	0.03	0.48	CNT_PAYMENT
7	8	0.03	0.51	DAYS_FIRST_DRAWING
8	9	0.03	0.54	DAYS_CREDIT
9	10	0.03	0.57	AMT_PAYMENT
10	11	0.03	0.60	NAME_CONTRACT_STATUS_2
11	12	0.02	0.62	DAYS_BIRTH
12	13	0.02	0.64	DAYS_ID_PUBLISH
13	14	0.02	0.66	DEF_30_CNT_SOCIAL_CIRCLE
14	15	0.02	0.68	AMT_CREDIT_SUM_OVERDUE

Figure 37 Important features after merging additional data sets

We can see that a few of the original top features which were contain in the application\_{train|test} data have been replaced by features contained in the supporting data files.

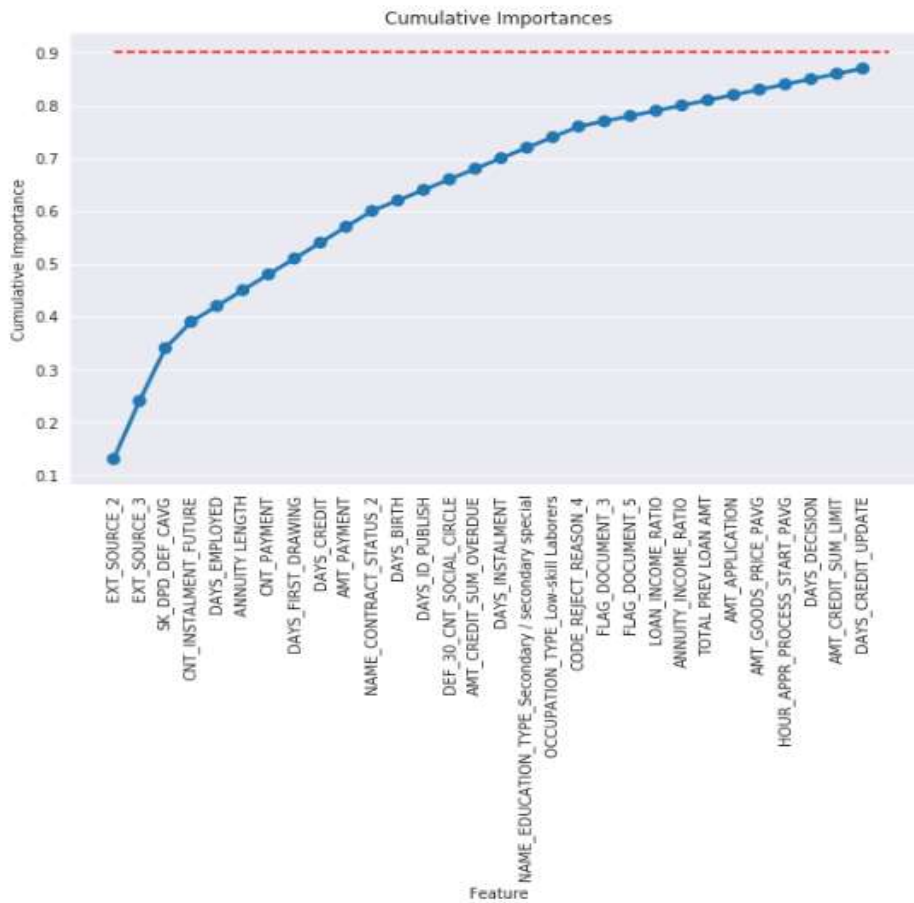


Figure 38 Cumulative features after merging additional data sets



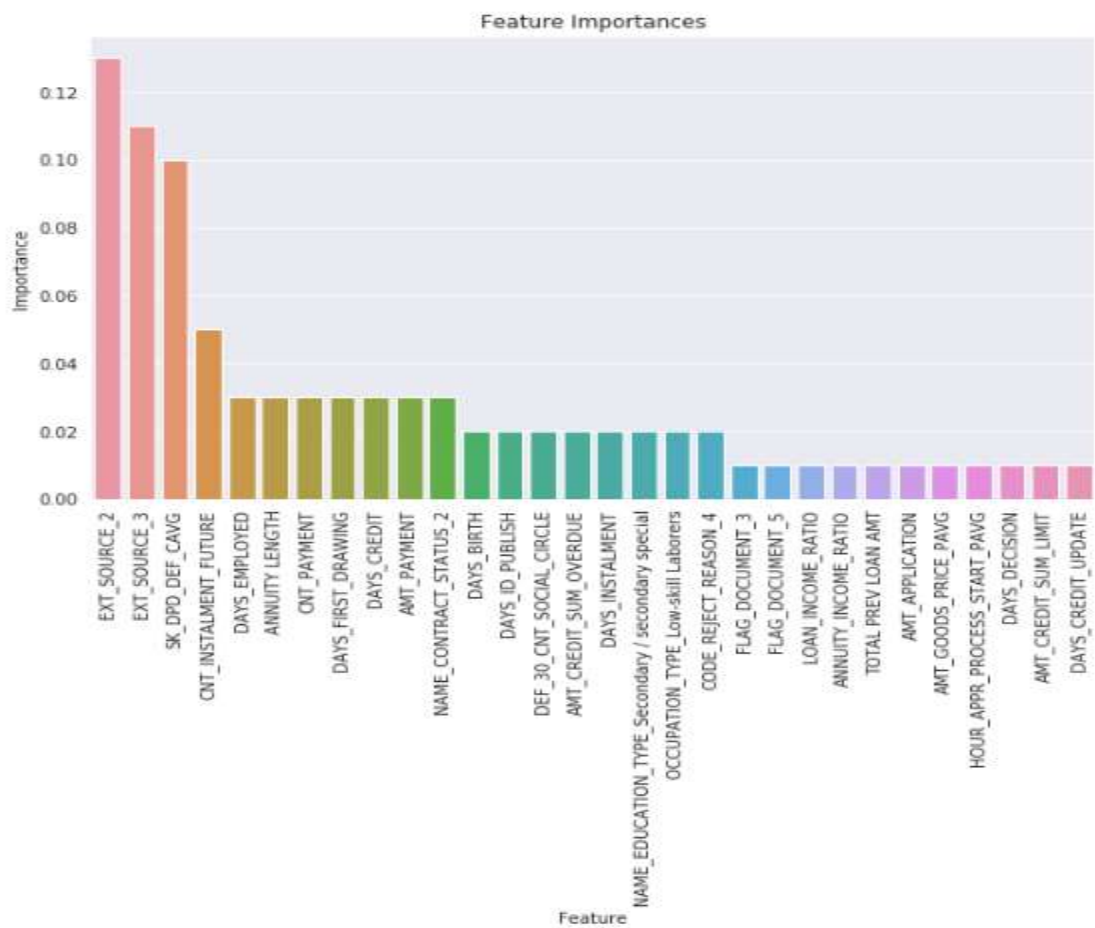


Figure 39 Feature ranking after merging additional data sets

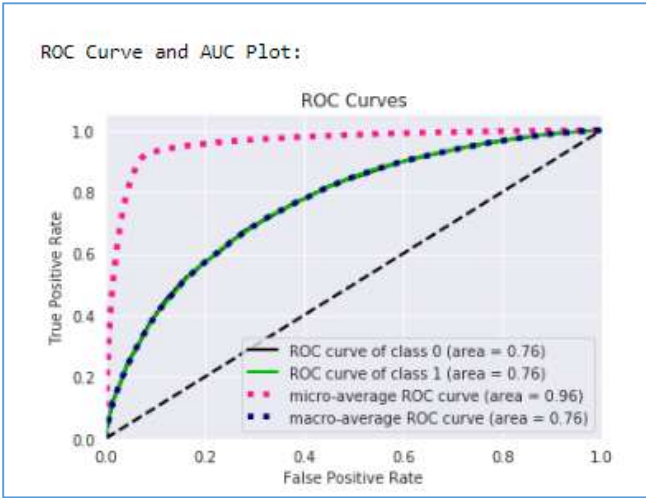


Figure 40 XGBoost Final UAC .76

### 3.3 Refinement

My baseline model was Logistic Regression which yielded a baseline AUC of .71 the final model XGBoost with and without feature importance yielded and AUC of .75, and the final model which was XGBoost using feature importance and selection which yielded a local AUC of .76 and a Kaggle score of .758. There were really only three actions taken to refine the score

1. The use of RandomizedSearchCV with cross-validation to help identify the best hyperparameters
2. The use of feature importance and a reduction of features to those identified as the top features requires to reach 90% variation in the model (which turned out to be about 23 features)
3. The aggregation and merging of the addition datasets into a single dataset. The merging yielded over 200 features, after encoding, one-hotting, and deleting those with greater than 25% null values.

This is a snippet of the code used to aggregate the previous loan application file details, the other files were aggregated and merged in a similar manner.

```
# Previous applications
agg_funs = {'SK_ID_CURR': 'count', 'AMT_CREDIT': 'sum'}
prev_apps = prev_app_df.groupby('SK_ID_CURR').agg(agg_funs)
prev_apps.columns = ['PREV APP COUNT', 'TOTAL PREV LOAN AMT']
merged_df = app_data.merge(prev_apps, left_on='SK_ID_CURR', right_index=True, how='left')
```

Figure 41 Aggregate and merge example

Prior to encoding and aggregating I took the approach of concatenating the training and test data , by stacking them one over the other, this way all the changes made to the training data would not have to be made again to the test data, this allowed all changes to be made at once. After the process was complete I split the data back into two data sets based on the know length of each file.

```
# Merge the datasets into a single one for processing columns
# we'll split the set after processing back into train & test
len_train = len(app_train) #use this variable later to know where to split
app_data = pd.concat([app_train, app_test])
```

Figure 42 Stacking the data sets

Code to split the file after the merge process

```
# Re-separate into labelled and unlabelled
#train_df = pd.concat([key_train, scaled_features_df[:len_train]], axis=1)
train_df = scaled_features_df[:len_train]

train_df.shape
(307511, 406)

#predict_df = pd.concat([key_test, scaled_features_df[len_train:]], axis=1)
predict_df = scaled_features_df[len_train:]

predict_df.shape
(48744, 406)
```

Figure 43 Separating after merge



4 RESULTS

4.1 Model Evaluation and Validation

The final model chosen is the XGBoost model using feature importance and the merged data set. Other flavors of XGBoost evaluated have numbers that are close as does Random Forest with class weights. However, this model is slightly better in AUC, perform father that Random Forest, and does better than random forest and XGBoost with class weight with fewer features required.

	precision	recall	f1-score	support
0	0.92	1.00	0.96	56554
1	0.58	0.02	0.05	4949
avg / total	0.89	0.92	0.88	61503
Predictor: [Accuracy score: 0.9196]				

Figure 44 XGBoost selected model classification report

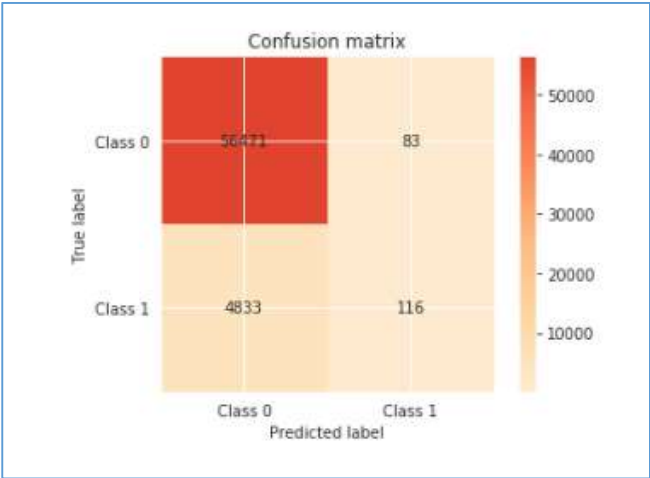


Figure 45 XGBoost selected model confusion matrix

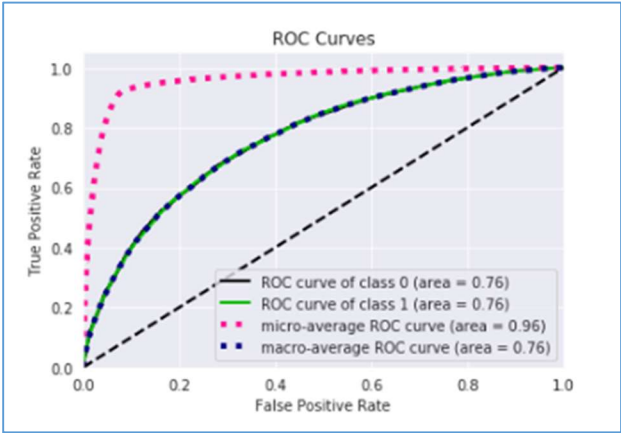


Figure 46 XGBoost selected model AUC (.76)

## 4.2 Justification

My baseline model was Logistic Regression which yielded a baseline AUC of .71 the final model XGBoost with and without feature importance yielded an AUC of .75. As stated in my proposal my goal was not to win a Kaggle contest. My AUC of .75 yielded a Kaggle score of .724. This was an improvement on the initial Logistic Regression Kaggle score of .591. At the time of this writing the leading score on the public leader board .812. Most of the top scores appear to have been submitted by teams, each submitting hundreds of submissions. This seems to be a difficult completion with hundreds of submissions separated by one-tenth of a percentage point. Given that and my current level of experience with machine learning I would consider my solution significant enough to meet my goals with the project and the rubric requires while not getting "in the money" on the competition.

## 5 CONCLUSION

### 5.1 Free-Form Visualization

I think the most relevant visualizations from this project would be the confusion matrices and their evolution as we progress through various models and techniques. They highlight the imbalanced classification problem and they summarize the results of improvement in the ability of a model to identify the minority class and in the ratio of misclassifications.



Figure 49 Logistic Baseline, AUC .68, no 1's

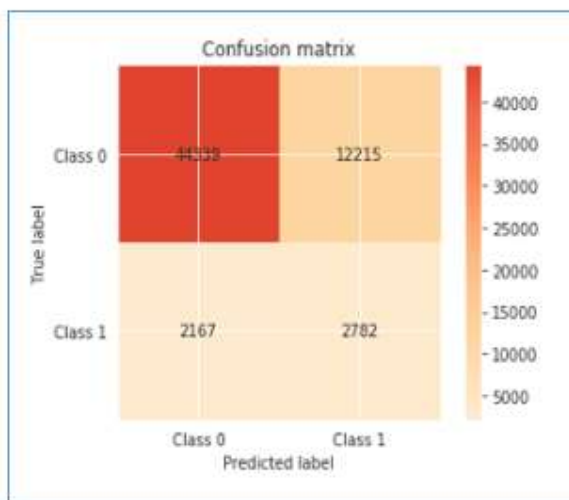


Figure 48 Random Forest, AUC 74, overclassifying 1's

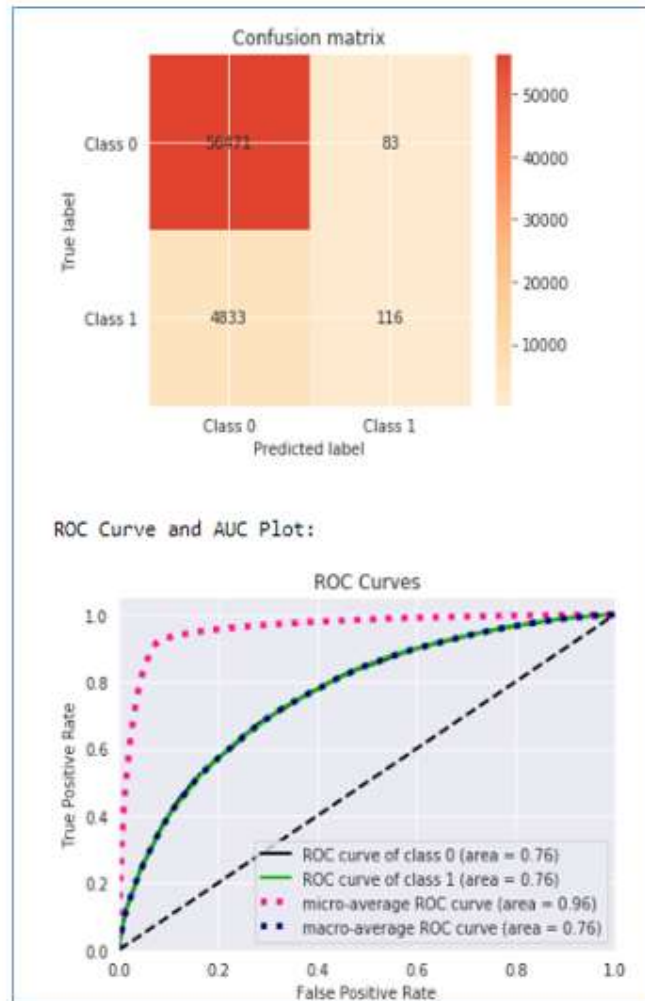


Figure 47 XGBoost, AUC .76, improvement

## 5.2 Reflection

The final model meets with my expectations for the project, my current score is .758 and I have successfully implemented several model showing improvement over the baseline. The most interesting, difficult and time consuming part of the project was the work to aggregate and merge the additional data sets with the main application\_{train|test} file.

The end-to-end summary of this project has really been provided in the preceding sections. In Phase 1 of the project, the data was obtained and then analyzed and inspected for anomalies. A number of analogies such as skewed data were transformed or corrected. A handful of new features were extracted, label encoding was used on small categories and one-hot encoding on the remaining categorical features. Numerical features were scaled. The imbalanced dataset was addressed using class weights and with an attempt to use SMOTE. Several classification algorithms were selected, implemented, and optimized using grid/random search. The selected model was then subjected to feature selection and re-executed in search of an improvement. Finally, in Phase 2 the additional datasets were aggregated and merged and a subset of the steps were performed again on the new consolidated dataset and then the selected models were re-applied using feature importance/selection. The selected classifiers with optimal parameters were fit to the processed data and corresponding labels. Predictions were made using the unlabeled test data, and probability of class membership was generated as a file output and submitted to Kaggle. The Kaggle calculated AUC was returned for the file submitted. This process was repeated several times in search of the best possible score.




While I have enjoyed learning more about Machine Learning the experience has led me to believe that it is probably not a field I will pursue. I have worked in information technology for over 20 years, including nearly 10 years as a software developer. I considered Machine Learning as a possible area to which to transition to during the later stage of my career. However, I find the work extremely fragile, tedious, and frustrating. Even though I was using AWS with a p2.xlarge server the training seemed to take forever, leading to a frustratingly slow cycle time to debug issues. Aggregation of the additional data files and merging it into a single data set took 52 minutes to run. I spent at least 20 hours trying to figure out why my ANN would not learn or converge, only to later determine that it was due to ONE FIELD (SK\_ID\_CURR) not being removed. The sheer size of the data being used in Machine Learning makes it difficult to test and debug issues which is frustrating for a former programmer used to having a variety of debug tools in their arsenal.

## 5.3 Improvement

My second submission after the Logistic model baseline was the un-optimized XGBoost submission, it yield a .724

Your most recent submission				
Name	Submitted	Wait time	Execution time	Score
XGBoost_baseline.csv	4 minutes ago	2 seconds	0 seconds	0.724
Complete				
<a href="#">Jump to your position on the leaderboard</a>				

Subsequently, XGBoost using the additional data set combined with feature selection yielded a score of .758. An increase of .34 advanced me up the leaderboard 1,036 places.

4420	new	Randy Jackson		0.758	3	now
<b>Your Best Entry</b> 						
You advanced 1,036 places on the leaderboard!						
Your submission scored 0.758, which is an improvement of your previous score of 0.724. Great job!						 Tweet this!

There are several things that could be done to potentially improve these classifications.

1. Continue to optimize the hyperparameters. I attempted to optimize the parameters identified as those most important based on my research using 3-fold cross validation with grid and random search. The process was extremely time-consuming and tedious. I attempted 5 and 10-fold cross-validation but the time required was unbearable. I completed this project using AWS on p2.xlarge server and it was still painfully slow. However, it is likely that additional hyperparameters and greater k-folding could lead to improved results.
2. Additional feature-engineering appears to be the key to hitting high AUC's however it requires not only technical knowledge but domain knowledge as well. I found the field descriptions provided with the contest unclear and would have had great difficulty attempting to understand or derive additional features.
3. Additional cleanup and feature engineering with the additional data files. Prior to using the additional data files I removed null columns, encoded, and scaled using straightforward steps and functions but I did not apply any effort to transform skewed features or looking for anomalies in those data files.
4. Investigate other models or combinations of model, for example using Principal Component Analysis (PCA) to provide input into another classification model.

## 6 REFERENCES

- [1] <https://www.kaggle.com/c/home-credit-default-risk>
- [2] <https://medium.com/henry-jia/bank-loan-default-prediction-with-machine-learning-e9336d19dffa>
- [3] <https://towardsdatascience.com/predicting-loan-repayment-5df4e0023e92>
- [4] <https://www.kaggle.com/willkoehrsen/start-here-a-gentle-introduction/>
- [5] <https://stats.stackexchange.com/questions/132777/what-does-auc-stand-for-and-what-is-it>
- [6] <http://www.chioka.in/class-imbalance-problem/>
- [7] <https://towardsdatascience.com/what-metrics-should-we-use-on-imbalanced-data-set-precision-recall-roc-e2e79252aeba>
- [8] <https://datascience.stackexchange.com/questions/13490/how-to-set-class-weights-for-imbalanced-classes-in-keras>
- [9] <https://stackoverflow.com/questions/41032551/how-to-compute-receiving-operating-characteristic-roc-and-auc-in-keras>
- [10] <https://hackernoon.com/simple-guide-on-how-to-generate-roc-plot-for-keras-classifier-2ecc6c73115a>
- [11] <https://machinelearningmastery.com/train-final-machine-learning-model/>
- [12] <https://machinelearningmastery.com/tactics-to-combat-imbalanced-classes-in-your-machine-learning-dataset/>
- [13] <https://beckernick.github.io/oversampling-modeling/>
- [14] <https://discuss.analyticsvidhya.com/t/imbalance-class-classification-using-random-forest/37311/2>
- [15] Chao Chen, Andy Liaw and Leo Breiman, July 2004, Using Random Forest to Learn Imbalanced Data
- [16] <https://jessesw.com/XG-Boost/> (A Guide to Gradient Boosted Trees with XGBoost in Python)
- [17] <https://machinelearningmastery.com/feature-importance-and-feature-selection-with-xgboost-in-python/>
- [18] <https://datawhatnow.com/feature-importance/>
- [19] <https://blancas.io/sklearn-evaluation/api/plot.html>
- [20] <http://amateurdatascientist.blogspot.com/2012/01/random-forest-algorithm.html>
- [21] <https://www.deeplearningtrack.com/single-post/2017/07/09/Introduction-to-NEURAL-NETWORKS-Advantages-and-Applications>