Capstone II: Loan Default Classifier

Springboard

Data Science Career Track

Dec. 2020

Student: Joseph Tran

Program Mentor: Misael Manjarres

Contents

1 Project Proposal

The business case I would like to investigate for the second capstone project is loan default classification. I will be using the Imperial College London's dataset for the problem available from Kaggle. I chose this business case and data set because it posed several challenges I had not yet encountered. First, the feature names have been masked with names not indicative of what the column represents. Second, there is a large column space of over 750 columns. Third, the target class is heavily imbalanced, with only a 10 distribution of the positive target class.

Classification tasks have enormous value for any business. The ability to target a set of individuals or groups based on specific features holds potential value in any domain. There are many use cases of classification tasks in the finance industry, however this project focuses on classifying whether or not a client, or potential client will default on a loan. Creating a reliable model that predicts loan default would increase revenue by reducing losses. A financial institution could use this model to assist in determining which potential clients could be offered a loan by approving loan applicants who show less risk of default. Similarly, the same type of model could be used for existing clients to help identify which accounts are at risk of default. In turn, a outreach campaign could be conducted to provide some type of intervention prior to default.

In this project I will explore different algorithms, in addition to feature selection and dimension reduction techniques. To reduce the size of the column space two methods of feature selection will be used separately, filter and ensemble methods. In addition I will apply principal component analysis on the entire data set I select an optimum level of features. These 3 separate datasets, encompassing different feature selection techniques will be benchmarked against each other to see which feature selection method provides the best results for this use case. Different algorithms will also be examined: logistic regression, adaboost, random forest, and xgboost will all be applied to see which shows the best performance. Lastly, to tackle the class imbalance issue, up-sampling, or SMOTE will be used to balance the target class and the above procedure will be used to examine the up-sampled data.

2 Data Wrangling

Anonymized features names and the large feature space associated with this data set introduces a few challenges. A large feature space means an increased amount of computational resources, in addition to longer training times. Anonymized features means that there is no way to apply domain knowledge for business decisions and engineering meaningful features. In a sense we a left shooting in the dark. The goal of the data wrangling steps in the project life-cycle is to reduce the features, keeping only what is likely to me most important for the model. This will be achieved through feature selection methods. Two data sets will be created through filter and ensemble feature selection methods. A third data set will be created via dimension reduction with principle component analysis. These data sets will be used in the modeling process to see which feature selection method is most successful for our use case.

2.1 Data Importation

The data set is provided in csv format, so we use pandas read csv function to load in the data. Checking the basic attributes shows that we have a data frame in the shape of $(105471, 771)$. Upon loading the data pandas throws an error regarding four columns with mixed data types, and these columns are dropped. Next, we look at the target variable 'loss' and see that there are several values indicating default. A lambda function is applied to column to make it a binary variable by setting all values greater than zero to indicate default. As a sanity check, we ensure that the number of unique values in the 'id' column is equal to the length of the data frame.

2.2 Examining Data Types

The first data type to be examined are the object columns. These columns are filtered from the data frame by using the 'select dtypes' function in pandas. We see that 15 of the features are categorical in nature. Looking at these values we see nothing meaningful as the values are a series of numbers encoded as strings. With no feature names, it is difficult to say whether this data could be meaningful to the model. To understand whether the information contained in the object columns has value we construct a data frame which shows the cardinality of each feature. Each of the fifteen categorical columns show high cardinality, with the lowest and highest values being 8663 and 104754, respectively. With the large number of values each feature can take on we can assume that there is no predictive power contained in these variables. Furthermore, attempting to create dummy variables for these features would drastically increase the column space, taking us in the opposite direction in which we want to go. As such, these columns are dropped from the data set leaning us with only integer and float data types remaining. It is here that we begin the feature selection process.

2.3 Filter Methods

To begin the process of feature selection with filter methods, we isolate the integer data type columns and look at the correlation of each feature in relationship to the target variable. Construction of a data frame showing each integer column, and its correlation to the target show minimum correlation with feature 'f25' showing the highest correlation with a value of 0.101. In order to keep features most associated with the target we filter the data frame to find the features with a correlation less than 0.01. We then remove these features from the original data frame 'df.' Dropping the features with a correlation value of less than 0.01 reduces the amount of integer columns from 99 to 35.

The same process is then repeated for the float data types. The float columns are isolated, and correlated against the target variable. In this case the feature with the highest correlation to the target is 'f322' with a correlation of 0.1238. The data frame is filtered to remove the features with a correlation to the target of less than 0.04. This process removes 409 features from the data set bringing the amount of float data type columns to 244 from 653.

To further apply filter methods to reduce the data set we remove constant and quasi-constant features from the data set. To do this a list comprehesion is created that looks at all features left in the data and returns the features with a standard deviation of less than or equal to 0.01. We find that there are 11 features in the remaining data that are constant or quasi-constant, and these features are removed.

Lastly, we want to remove any independent features that are correlated with each other. To so this we create a function that uses the data frame and threshold values as arguments and calls the correlation function on the remaining independent features and look for pairs of features that with a correlation value greater than the threshold, which is set to 0.8. Calling the function returns a list of 214 features with correlations greater than 0.8, these features are then removed from the remaining data. In total, by using combination of filter methods we reduced the columns space from 767 to 54. This data is the first of 3 separate data sets to be created from the original data.

2.4 Ensemble Methods

The next feature selection method to be used is an ensemble method using a random forest classifier. Random forest classifier and 'the select from model' classes are imported from scikit-learn. The independent features are stored in a variable 'X' and the target column in 'y.' A 'select from model' object is instantiated with a random forest classifier with default parameters, and n estimators set to 10. This object is saved to the variable 'select.' The list 'selected features' is created to store the features selected by the 'select from model' object. Relevant variables are obtained by calling the 'get support' function on the 'select' object. We are left with a list containing 333 features that were determined by the model to have predictive power in relationship to the target.

The features remaining from the filter and ensemble methods are then persisted in a dictionary and saved as a json file for easy access in future notebooks.

2.5 Dimension Reduction

The last data set to be created for use in the model is generated from performing principle component analysis on the data. PCA and StandardScaler are imported to perform the operation. In order to properly use PCA the features must be scaled, so prior to calling PCA the independent features are scaled usng the StandardScaler. A plot of the PCA is created using the first two components, with the target class encoded by color. What we hope to see is some separation between the positive and negative class. However, no distinct pattern can be discerned. This does not mean the some structure does not exist, only that none is observable in a two dimensional plot. What is observable in the plot is what looks like a 90 degree angle created from the data. This is somewhat expected as we know the principle components are orthogonal to each other.

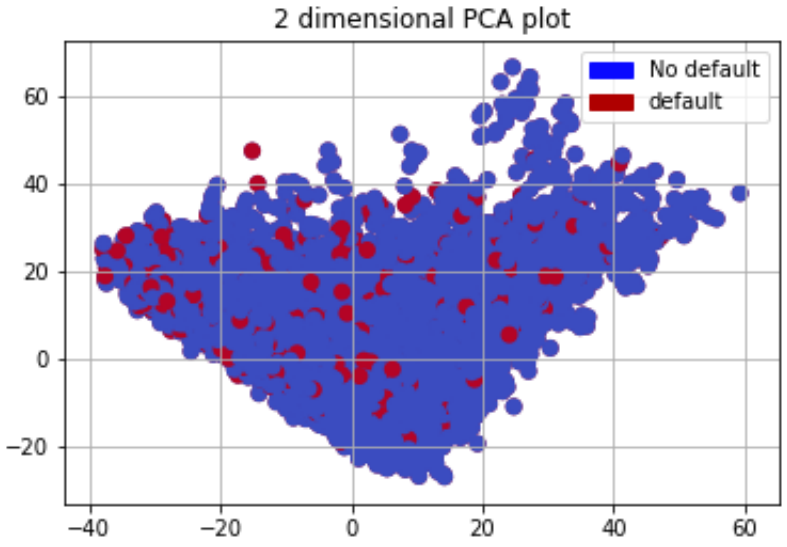


Figure 1: 2 Dimensional PCA Plot

Lastly, in order to determine the number of components to keep for the model we make a plot of the cumulative explained variance to visually inspect the number of components needed to explain greater than 90 percent of the explained variance.

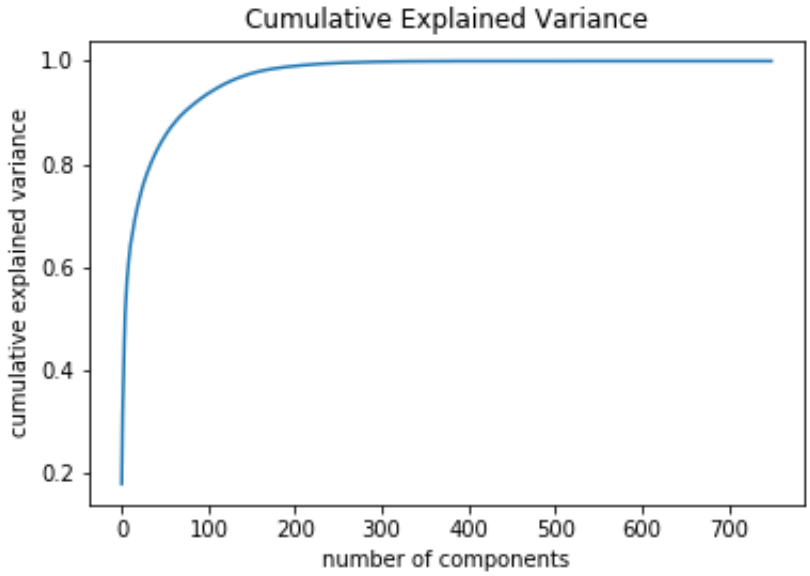


Figure 2: Cumulative Explained Variance

3 Statistical Analysis

Given the large feature space of the original and derived data sets we want to try and look at the variables that have the most importance. For our use, we define most important as the intersection of the 57 features selected via filter methods, and the 333 features obtained from the ensemble method. This results in 30 features that are used for statistical analysis.

Knowing that that balance of the target class can impact the efficacy of the model, the first thing we do is look at the distribution of the positive and negative classes. Using seaborn's countplot function we plot the target column and see that the distribution of the target class suffers from a severe class imbalance. The plot shows and ratio of about 90/10, which is expected in the case of default data. We know that a small percentage of samples will actually default in comparison to non-defaulting clients.

With the target class suffering from a large imbalance we will need to try and mitigate this in some way. For this project we will use the popular up-sampling technique, SMOTE. A version of the algorithm is available with the imbalance package. SMOTE works by sampling from the minority class until the distribution is equal. Other sampling techniques, such as adsysn, are also in the imbalance library. In addition to up-sampling techniques, the xgboost classifier comes with the parameter, 'scale pos weight.' This parameter can be used to train a class-weighted version of an xgboost classifier.

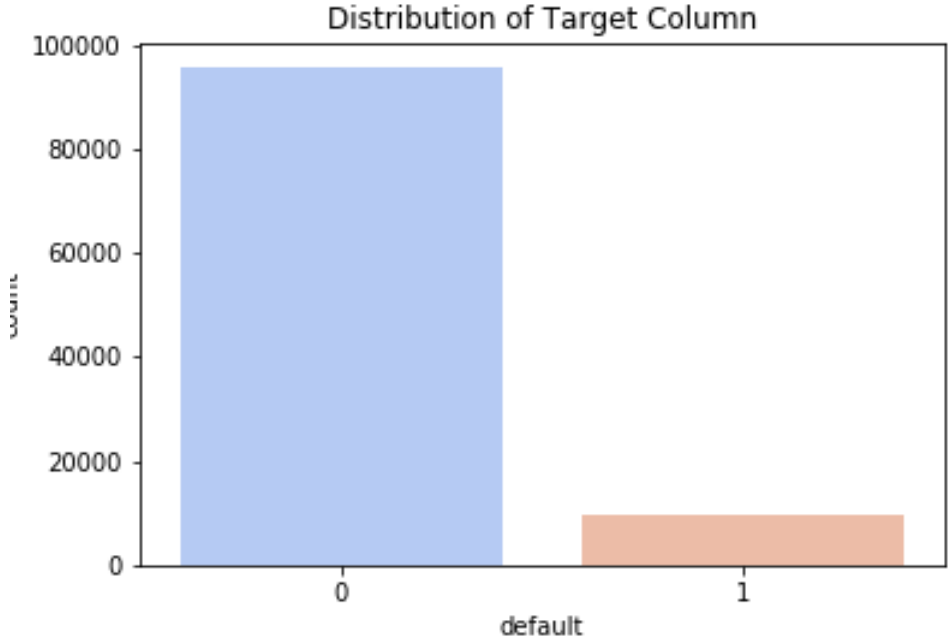
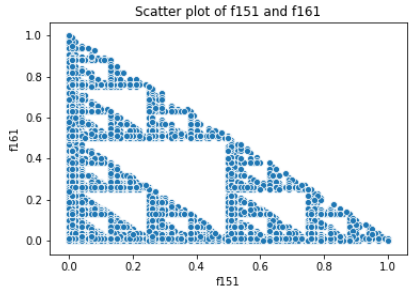


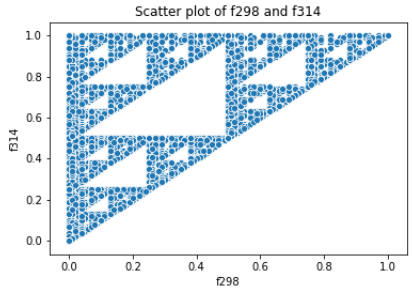
Figure 3: Target Class Distribution

3.1 Descriptive Statistics

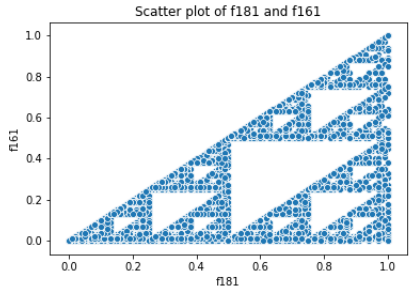
The first thing we do when conducting descriptive statistics is create a pairplot to look at scatter plots of the features. It appears that there is a lot of noise in the data, with some plots appearing as though they are derived from synthetically created data. Eight of the features shows abnormal scatter plots when in pairs. An example of the scatter plot from the variables 'f151' and 'f161' and others are shown below. These features a recorded to be removed from the data set as they seem to exhibit unnatural behavior.



:
'f151"f161'

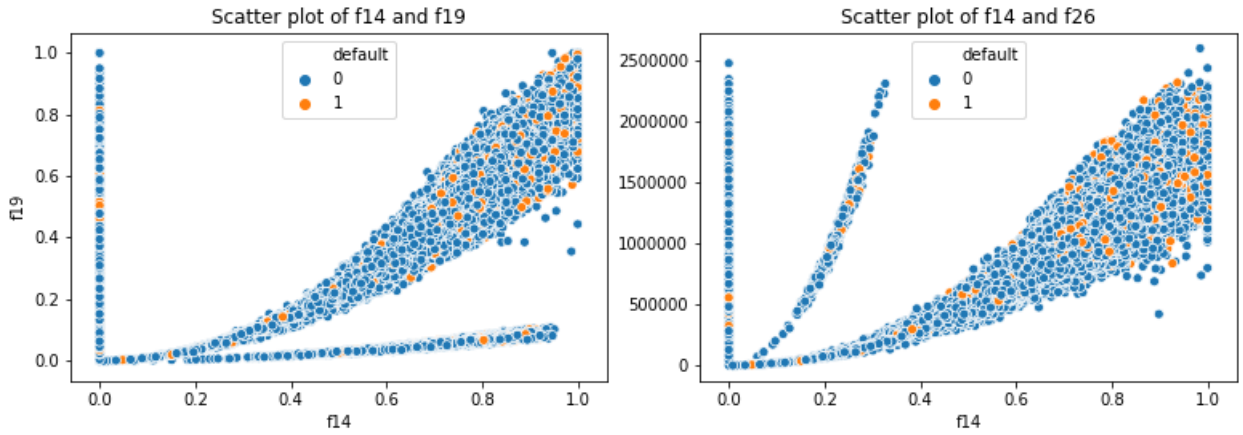


:
'f298"f314'



:
'f181"f161'

Most of the variable interactions seem to look like noise. However, aside from the odd scatter plots seen above, there are a few variables that show some structure. These plots are shown below.



: ’f14’ ’f19

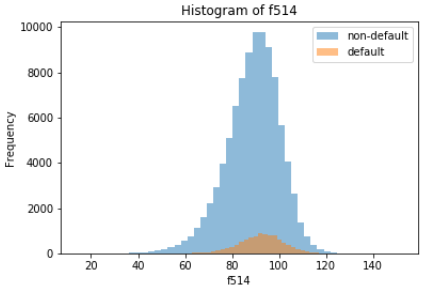
: ’f14’ ’f26’

Most of the interactions in the pairplot look as though it is mainly noise. In addition, there does not appear to be any separation between the classes in the scatter plots. The variable ’f14’ is interesting because there appears to be some type of exponential relationship between ’f14’ and ’f19’ and ’f26.’ In each plot there is some data the branches of from the rest of the data as if they are two groups.

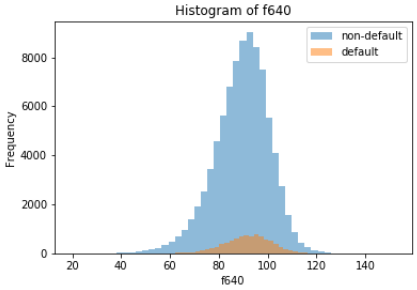
The next step in the process is to look for, and remove outliers. This is accomplished by defining the the inter-quartile range, and the 25th and 75th percentiles. One we have defined these values, a filtering statement is used to grab all the data not above/below the range of 1.5 +- the IQR. Once we perform this operation, we see that over 75 percent of the data falls in a range considered to be an outlier. This poses a problem as we want to keep as much data as possible. Being a little apprehensive about how to proceed a read a few articles online that stated sometimes outliers are an inherent part of some data, and could possibly be an artifact of how the data was generated. Being that so much data is lost, I am deciding to include all of the data, including what appear to be outliers.

3.2 Inferential Statistics

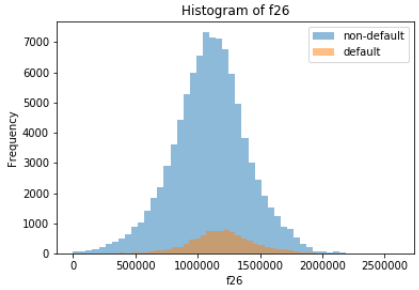
Inferential statistics are used when trying to compare two separate samples. The two samples we will look at are the sample from the negative and positive classes. We would like these two samples to a different to a statistically significant degree. By proving that the samples from the negative and positive class originate from two different distributions, and can clue us in on whether there is a intrinsic structure within the data that the model can find to make its classifications. We plot histograms of the various features, encoding the target classes by color to easily inspect how the distributions compare over one another. To test whether the two samples originate from the same distribution two way t-tests are conducted in the variables that appear to show a normal distribution.



:
’f514

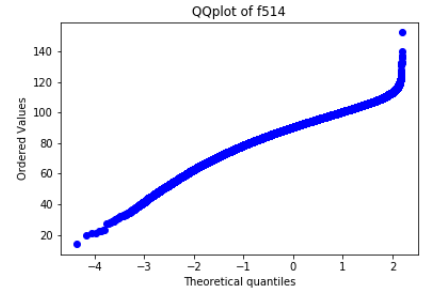


:
’f640

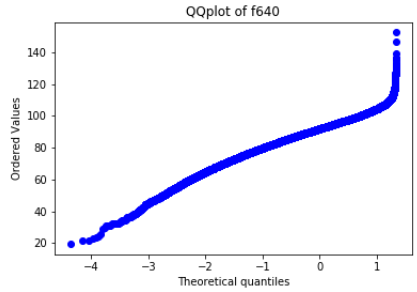


:
’f26’

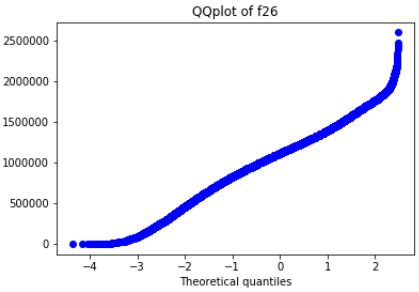
We can see from the above histograms that the classes do not appear to be separable at all. To verify whether the two classes are sampled from the same population we perform t-tests.



:
’f514



:
’f640



:
’f26’

The above histograms of the two classes show what appear to be normal distributions. This is confirmed by the QQ-plots below the histograms. There is a little deviation at the tails, but we assume normality for the t-tests. We conduct t-test for the variables ’f514’ ’f640’ and ’f26.’ The t-tests performed are paired t-tests with unequal variances. This is verified by comparing the variances of each class, for each variable. The results for each t-test reveal t values in the negatives, and p values that are extremely low. This leads us to accept the null hypothesis that the two class samples are drawn from the same distribution. It would have been nice if the opposite was true, because that would provide some separation for the classifier to perform better.

4 Data Modeling

Various feature selection methods have been applied to create three datasets that will be used for modeling: filtered features (105471,53), random forest selected features (105471,335) and PCA features (105471,175). Each dataset is passed through a pipeline that imputes any missing values with the mean of the column, and then calls one of 3 classifiers: logistic regression, random forest, adaboost. The classifiers are chosen to represent one of each model type: standard, boosting, and bagging. A parameter grid is created with each classifier and its respective hyper-paramters to search over. The pipeline and parameter grid are then fed to a randomized search cv object for training and evaluation. Inside the randomized search, 5-fold cross validation is performed over 25 iterations and roc auc chosen as the evaluation metric. The pipeline for the PCA features includes extra steps for feature scaling and extracting the PCA features for the classifier.

In addition to the three algorithms above, the xgboost algorithm is fit to the three datasets. Since xgboost can accept null values no pipeline is needed and the model and parameter grid are created. However, for the PCA features, a pipeline is used to impute for feature scaling prior to extracting the PCA features, which are then fed to the classifier.

For each dataset the xgboost algorithm performed the best, which was expected. The main reason most likely being that it has the capability to handle an imbalanced target class with the 'scale pos weight' hyper-parameter.

4.1 Filtered Features

The following tables how the results of the model's performance on the train and testing set, along with the model hyper-parameters. The logistic regression model does not appear due to it not ranking withing the 25 best models.

Table 1: Hyper-parameter abbreviations

ne	n estimators
lr	leanring rate
md	max depth
csbt	col sample by tree
γ	γ
spw	scale pos weight
msl	min samples leaf
mln	min leaf nodes

Table 2: Model Results

model	ne	lr	md	csbt	γ	spw	msl	mln	**auc train**	**auc test**
xgboost	150	0.1	4	0.2	0.1	9			0.73	0.67
adaboost	200	.56							0.697	0.668
random forest	10		none				2	10	n/a	0.637

Using auc score as an evaluation metric, the performance of the xgboost model slightly outperforms the adaboost. However, both models show some over-fitting to the training data. Overall, the models did not perform well, only performing about 20 percent better than random guessing. This is most likely the result of a signal/noise ration imbalance in the data. While xgboost and adaboost appear to perform relatively the same based on auc score, it does not tell the entire story. Recall, or the proportion of positives correctly identified is much higher in the xgboost model (0.600), compared the the adaboost model (0.003). This is evident when comparing the confusion matrices which show. In the business context of predicting loan default the stakeholders would most likely seek to minimize false negatives as this would lead to a loss in revenue. However, a balance needs to be struck as too many false positives may lead the business to miss out on opportunity.

4.2 Random Forest Features

Table 3: Model Results

model	ne	lr	md	csbt	γ	spw	msl	mln	**auc train**	**auc test**
xgboost	150	0.1	4	0.2	0.1	9			0.801	0.706
adaboost	200	.34							0.732	0.697
random forest	10		30				5	10	n/a	0.670

The results from the random forest selected features are similar to that of the filtered features. The algorithms rank in the same order with similar results for the mean test auc scores, which also show some overfitting to the training data. Most notably the in the xgboost model where there difference between train and test scores is 0.10. This leads one to believe that datasets created from the filtered features and random forest features have about the same predictive power, while the filtered features column space is much smaller, 52 compared to 332 features.

4.3 PCA n=175 Features

Table 4: Model Results

model	ne	lr	md	csbt	γ	spw	msl	mln	penalty	C	**auc train**	**auc test**
logistic regression									L2	10	0.718	0.699
adaboost	200	.24									n/a	0.690
xgboost	150	0.1	4	0.2	0.1	9					0.73	0.676
random forest	1000		none				2	10			n/a	0.637

The results from the PCA selected features show some variation between the other two datasets. The logistic regression model performed the best, with minimal over-fitting to the training data with train and test auc scores of 0.718 and 0.699, respectively. However, this is just taking into considered the auc score as the evaluation metric. As mentioned above, we need another metric to better evaluate the models. Recall for the positive class is 0.005 0.626 for logistic regression and xgboost, respectively. Even though the logistic model produces a better auc score, recall on the positive class is very low due to a high number of false negatives. In the business context, the aim is to reduce the number of false negatives, as they cost the business money, while also striking a balance with the amount of false positives, which could lead to loss of potential revenue.

5 Final Model

The final model we select for further tuning will be an xgboost model on the random forest selected features. An xgboost model is selected because recall on the positive class is much higher than when using other algorithms. In addition, this model and data show the largest discrepancy between the train and test scores, while also having the highest auc scores of all the models. The thinking as that regularization can be added to reduce model complexity to help mitigate over-fitting to the training data. The final parameters are shown in the image below.

```
objective='binary:logistic',
base_score=0.5, booster='gbtree', colsample_bylevel=.75,
colsample_bynode=.3, colsample_bytree=0.2, gamma=0.1, gpu_id=-1,
importance_type='gain',
learning_rate=0.001, max_delta_step=0, max_depth=4,
min_child_weight=15, missing=np.nan,
n_estimators=200, n_jobs=0, num_parallel_tree=1, random_state=0,
reg_alpha=0, reg_lambda=20, scale_pos_weight=9, subsample=.75,
tree_method='exact', validate_parameters=1, verbosity=None,
```

Figure 4: Final xgboost parameters

The main hyper-parameters used to control the complexity of the model were reg lambda, min child weight, max depth, colsample by level/node, learning rate and subsample. While it was possible to bring the train and test scored to within a 0.004 difference, a severe loss in auc scored occurred with train and test auc being 0.630 and 0.626. Even though the auc score is too low for the train and test sets to implement the model in a business setting, it is reasonable to assume that the model will generalize well to unseen data. This is preferable to having higher auc scores with a large discrepancy. A high test auc score may lead one to believe the model is performing well, but that says nothing about how it will generalize to unseen data if the training auc is larger.

The most reasonable guess as the why the models had low auc scores is because there was too much noise in the data. Typically, the recommendation to improve a model is to collect more data. However, in this case more data of the same variety may not help the performance. If it is possible to get the feature names, feature engineering with the help of domain knowledge may help to create more informative features. Since at this time those feature names are not available we will try to improve the performance by balancing the target class with the help of over-sampling the minority or under-sampling the majority class with SMOTE.

6 Advanced Method: SMOTE

The imblearn library in python offers many varieties of over/under sampling. The documentation for the library states that under sampling generally performs better. Here, we apply the basic methods for over sampling in addition to to under sampling to see which fit the needs of the model best. XGBoost was the best model in the previous runs, so we will use this model test the sampling techniques. As before, a parameter grid is used inside a random search to look for a best model. An imblearn pipeline class is used to create a pipeline for the model. Unlike before when we did not need to impute values for the XGBoost model, here we need to include an imputation step prior to calling the sampling technique. Both mean, and median imputation methods were tried with median imputation performing better, so that is what is used in the pipeline. The data from the random forest features are then fit to the randomized search object.

6.1 Under Sampling with SMOTE

In terms of roc-auc score, the XGBoost model with under-sampling performs the worst of all the models with train and test scores of .597 and .577 percent. However, when trying to minimize false negatives and increase false positives, it performs the best with a sensitivity score of 0.965. Sensitivity quantifies the ability to avoid false negatives. In the business context of identifying loan defaults, we want to avoid false negatives as they represent a loss in revenue. When compared to the best XGBoost model without SMOTE techniques, We see that the false negative rate is reduced from 758, to 67, while the false positive rate is increased from 6876 to 15501. Furthermore, the true positive rate increases from 1199 to 1890. The result is that we 'trade' true negative classifications for false positives.

6.2 Over Sampling with SMOTE

Over-sampling produced results similar to the previous models with a roc scores of .677 and .635 percent for train and test, respectively. In addition, over-sampling yielded a confusion matrix better suited for the business application. When compared with the XGBoost model without SMOTE techniques, over-sampling was able to increase true and false positives and decrease true and false negatives. The sensitivity and specificity of the model were also more balanced than the under-sampling methods with values of .720 and .551, respectively.

7 Further Development

The biggest issue with the data used was the lack of names for each feature. Knowing the feature names would go far in providing a more robust model. Without the feature names, the model is reliant on feature selection methods to find features that are useful in predicting the target. This information would be useful when trying to find edge cases that are close to the decision boundary, and provide information as to what features are driving the prediction of a loan default.

From a data ethics standpoint, having access to to the feature names would help to ensure that we are not introducing some type of negative bias that affects disadvantaged communities. I recall a metric in the Boston Housing data set that accounts for the amount of black people in a neighborhood. If features like this exist in the data that was used for our model we would have no way of knowing as the features names are not provided.

Lastly, it must be noted that the model provided would not serve well in the context of identifying *who* the business gives a loan to. The low roc scores do not offer enough confidence to make a yes/no decision on who should receive a loan. Additionally, the large amount of false positives make for a highly risk averse decisions. This could affect revenue by not offering loans to clients who are loan worthy, but are incorrectly classified as a person who would default on the loan. The model provided would only serve well in a risk intervention scenario where the business is trying to identify who is currently at risk of defaulting.

8 Conclusion

Of all the models and data sets used, XGBoost proved to be the best algorithm for the classification task. In addition, the SMOTE techniques provided confusion matrices that better capture how we want the model to perform. The false negative rate decreased in each instance of over and under-sampling. This had the affect of increasing the false positive rate in addition to the true positive rate. Since the target variable was imbalanced, every model performed well when classifying true negatives. However, this is not the information we are after. We are looking for an increase in true positives and false positives, while decreasing the false negative rate. The false negative rate represents a loss in revenue for the business, so minimizing the false negative rate is of utmost importance. If the model is used internally, subsequent to a loan being issued to identify at risk clients, then a considerable amount of false positives will not directly affect revenue. In this case, the business could institute a method of intervention prior to the loan defaulting. False positives would most likely represent people who are at risk of defaulting and applying a method of intervention to these clients could save the company money, so a amount false positives would not be critical to the bottom line.