Finance 6500 : Predicting loan default

Willem Peter van der Schans

MSBA & MSF Program

University of Utah

4/19/2020

Contents

Introduction

This documentation describes the process that was followed to fulfill all the points as outlined in the project one assignment documentation on canvas. The project's goal is to predict loan default by using an algorithm, after which the trained algorithms have to be accessed through an app a loan officer could use to predict any potential loans on inputs. Python is used as the coding language in both Spyder and Jupyter notebook. The sections of this documentation go, in chronological order (as outlined in the Jupyter notebook) over each important step taken to fulfill towards the end goals. The documentation is not just a technical description of what happened but will also describe considerations and points of improvement where needed. For reference, all files are uploaded and accessible at https://github.com/Kydoimos97/FINAN6500DATA

Exploratory Analysis and Data Cleaning

This section continuously references the codebook found in appendix A. It will go over train data and the transformations made to remedy any discrepancies between the given data and the codebook. After this, I quickly outline transformations made on the test data to get it in line with the train data; however, since the considerations are identical, they will not be touched upon again.

Overview

The data provided consists of two data sets; "Train" and "Test". Both train and test consist of 26 variables, as outlined in Appendix A. The length of the train and test data sets are respectively 10000 and 1000.

Identifiers

There are several identifying variables present in the data set which should be removed. The identifying variables have to be removed is because they are too specific and don't generalize to a population. A name, for example, is specific to one business and does not generalize further to a population, such a variable does not provide any information to algorithms that try to predict default and if anything only increases run-time. Based on the notion that identifying variables do not provide information to algorithms, they are all removed. The removed identifying variable are: "LoanNr_ChkDgt", "Name" and "NAICS"

Highly Specific Variables

Highly specific variables are variables that are so specific that its potential generalization is limited. Highly specific variables like city touch upon one city which limits the sample size for the population in that city. When taking the central limit theorem in account which states that the sampling distribution takes on a normal distribution if the sample size is sufficiently large (Boston School of Public Health, 2016), I can decide if city should be accounted for or not. Generally speaking, a sample size should be at least 30 to be generalizable, when looking at the value counts of city however the outline shows that for this variable this is often not the case. For only 21 entries the sample size for the specific city is over 30, the total entries these 21 cities account for is 1340. Since you are basically creating subsamples in the data set by including a variable like this, I chose to remove the City variable since most entries are not generalizable to the population of the cities. The same conclusion is drawn for Zip

Datetime objects

Date variables are saved as Datetime objects in Python with which algorithms don't work with. One-Hot encoding Datetime objects also is not straight forward. Due to the implication Datetime objects bring to running the algorithms effectively, they are removed. The affected variables are ApprovalDate and DisbursementDate

Missing Data

The missing data in the provided data set can be seen in table 1. Potential target variables are ChgOffDate and MIS_Status; they will be touched upon in section 4, and therefore ignored in this section. The variables Bank and BankState have their rows with NA's removed since these are impossible to attribute. Initially there have been efforts to look for patterns in data to determine the which gave out the loans however those efforts were in vain resulting in removal of the rows. The loss of data is minimal with 6 on 10000 making this decision easier.

Variable	Sum of Missing
ChgOffDate	8053
MIS_Status	21
Bank	6
BankState	6
RevLineCr	1

Table 1: Missing values in the train data set

RevLineCr

RevlineCr is a variable which is described in the codebook (appendix A) as having two possible states, Y and N. In the supplied data, however, there are a total of 7 entries present; Y, N, 0, T, `, "," and NA. Since it is hard to determine what should be done with these variables entries tests were conducted. The test consisted of running the algorithms with different

imputation and removal of the variable. The final result is that mode imputing the faulty entries in the variable with N netted the best result which is the option that I decided upon in the end.

BalanceGross

While initially it seems that BalanceGross has no missing entries after inspecting this variable it can be seen that all the entries are 0. Having just 0's as entries means that BalanceGross carries no information and should, therefore, be removed from the dataset.

Other Cleaning

LowDoc

According to the codebook (appendix A), LowDoc has the possible values of Y and N; however, in the provided data, there also is a C present. This value is mode imputed with N.

FranchiseCode

FranchiseCode has a lot of variables showing if a company is a franchise or not. FranchiseCode is, first and foremost, an identifier variable; this has to be remedied as previously discussed. To do this, I set all the 0 and 1 entries to 0 and the other entries to 1. The transformation executed transforms the variable to a categorical variable showing the algorithms if a business is a franchise (1) or not (0)

Web scraping

  For the web scraping, part of the assignment data was obtained from the St. Louis federal reserve (Federal Reserve Bank of St. Louis, n.d.). The data obtained is quarterly recession data from 1967 until 2019 (Federal Reserve Bank of St. Louis, 2020). The extracted data shows in what quarter the US has been in a recession.

  After downloading the aforementioned data, I created a dictionary with a Boolean variable indicating if the US was in a recession (0=NO, 1=Yes) by using mode generalization of quarterly data to years. This means if 1969 had three quarters in which there was a recession, then the dictionary included a 1 for 1969. After creating the dictionary, the Approval FY column was duplicated copied and then renamed to the created dictionary keys with ".replace." This created a variable that showed per row if the loan was approved in a recession or not.

  After adding the recession variable, the final score of the algorithms increased slightly, with 0.0001%.

  To improve on this process, a quarterly dictionary should be created, and the variables imputed based on ApprovalDate resulting in higher precision regarding if the loan was approved in a recession.

Target Variable

The datasets included three potential target variables; MIS_Status, ChgOffDate, and ChgOffPrinGr. I will go over all variables and discuss why the variables were picked or not picked as the final target variable.

ChgOffPrinGr

Initially, I looked at ChgOffPrinGr as my target variable; however, due to the zero-inflated probability distribution (figure 1) of the variable, it was hard to work with. The reason a zero-inflated probability distribution is hard to work with is because it cannot be easily transformed to be normal. After transforming the variable with the natural logarithm of one plus the input array, the distribution still was not normalized (figure 2). Zero-inflated probability distributions can be worked with when using specific algorithms, e.g., Decision tree; this would restrict me heavily in what algorithms I could use; therefore, I opted not to chose ChgOffPrinGr as my target variable.

Figure 1: Distribution of ChgOffPringGr

MIS_Status

I chose not to try to predict MIS-Status since there were 21 missing entries (table 1). While 21 on 10000 is not a lot of data to lose, if the option exists to keep this data, I rather choose for that option and include these 21 entries in training the algorithms.

Figure 2: Distribution of ChgOffPringGr after transformation

ChgOffDate

Looking at table 1, it seems that ChgOffDate has more missing values then MIS_Status does. In the case of ChgOffDate, however, missing values are not a problem since the implication of them is clear: "A missing value means no default." Knowing this, I changed the variable to a binominal categorical variable with 0 = no default and 1 = default.

The final frequencies of 0 and 1 can be seen in figure 3. 0 has a frequency of 8028, and 1 has a frequency of 1937, this gives us the majority class predictor accuracy = 80.56%. This 80.56% accuracy is the benchmark score to beat by the machine learning algorithms deployed.
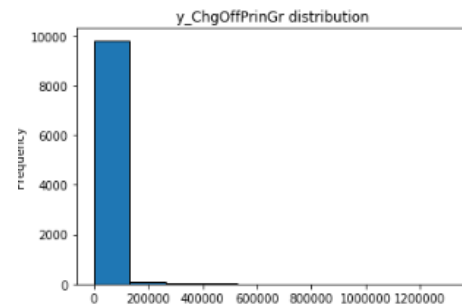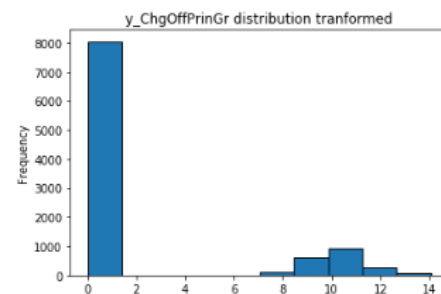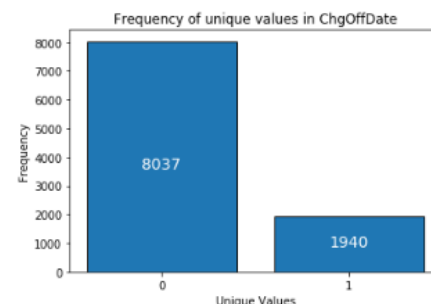

Figure 3:Frequency of unique values in ChgOffDate

Feature Engineering

For the data to be usable with the machine learning algorithms, it is necessary to do feature engineering. In this section, I will go over what methods I used and why I used them.

Label Encoding

Label encoding is the transformation of text data to numeric data; such a transformation is needed since most algorithms cannot work with text data. The variable that I label encoded was Bank. I chose to do this to optimize performance since one-hot encoding Bank would result in a data set with 1100 columns instead of 120 columns, which drastically increases run-time from around 120 seconds to around 750 seconds while lowering the final performance of the algorithms.

A point of notice would be that label encoding brings with it the danger that an algorithm unintentionally handles the data as continuous data where for example, Male= 0 and Female = 1 female is seen as better than male by the algorithm. I decided that based on the final scores of the algorithms, I would revisit the decision so label-encode Bank; however, since the final scores were satisfactory, I decided the decision was acceptable. It should be noted that run-time is important for the app used by the loan officers, as having an extremely long run-time could severely hinder workflow and, therefore, should be kept to a minimum, which contributed to the decision to label encode bank.

Box-Cox Transformation

To combat excessive skewness in feature data, I decided to run a box cox transformation on variables that had an absolute skewness of more than 0.75, the skewness of variables can be seen in appendix B. The Box-Cox transformation I conduct has a lambda value of .15 and resulted in skewness being eliminated where applied.

One-Hot Encoding

I one-hot encoded any resulting variables at this point in the process that had the object data type in Python. The object data type points to data consisting of strings. Strings are not suitable for algorithms to work with and should, therefore, be either label-encoded or one-hot encoded. Since the remaining variables do not have a large number of entries (max 50), label encoding would not create a data set that was too big to hinder performance. The one-hot encoded variables are; State, BankState, RevLineCr, and, LowDoc. The resulting data frame has 120 columns over 17 columns before one-hot encoding.

Data Scaling

I looked into data scaling to combat my matrix being ill-conditioned, however running a data scaling transformation on my variables increased run-time by 10 seconds and lowered the final performance of my algorithms. I, therefore, decide it is better not to use data scaling since an ill-conditioned matrix seems to be no issue when no kernel tricks are used.

Machine Learning

In this section, I go over the validation methods and algorithms used and the final scores of the algorithms.

Validation Methods

To have better insight into the performance of my algorithms, I opted to use both cross-validation and hold-out evaluation. I wanted better insight because I kept running into having an in-sample score that was perfect with two algorithms. While this initially seemed like overfitting, it turned out that the algorithm just ran well since the out-of-sample scores were also high. I cross-validated with five folds and did my hold-out evaluation with 10% of the train data set.

Algorithms

I decided to pick algorithms based on research conducted by other parties. I found a website that clearly described classification algorithms and gave final accuracy scores per algorithm (Analytics India Magazine, 2018). I decided to use the four highest-ranking algorithms, which are: Logistic Regression, Random Forest Classifier, Decision Trees Classifier, Decision trees, and Support Vector Classifier. Next to the four aforementioned algorithms, I also decided to include KNN and Boost. I included KNN since we went over it in class so extensively and XGBoost due to past results being good.

Hyperparameters

First, all the algorithms run with conservative hyperparameters, the results of which can be seen in Table 2 and Appendix D, I do this to find the best algorithms which I can then further subject to a grid search to find the most optimal hyperparameters (Analytics Vidhya, 2016). The algorithms that are subjected to a grid search are XGBoost Classification, Random Forest Classification, and Decision Trees Classifier. I wanted to make sure to pick the already best algorithms for the data set since the processing power needed for a grid search is fast. I tried to use my GPU to do a grid search but could, in the end, not figure it out. I am certain that using CUDA cores instead of CPU computing will improve run times, so I will be doing further research on this topic soon. The final hyperparameters are listed in Appendix C, note that the cross-validation score is leading as this score is based on five-folds instead of one-hold out evaluation. If I could improve on the grid search process, I would make a dictionary of the best hyperparameters found by the functions which I would then automatically apply to the final model run.

When evaluating the final code, I recommend turning off the grid search since the run time went from around 2 minutes to 26 minutes.

Model	CV_Score	HO_Out-Of-Sample_Score	HO_In-Sample_Score
XGBoost Classification	0.9537	0.9519	0.9880
Random forest Classification	0.9385	0.9405	1.0000
Decision Trees Classifier	0.9282	0.9452	0.9442
K-Neighbors Classification	0.8803	0.9018	1.0000
Logistic Regression	0.8727	0.8811	0.8841
Support Vector Classification	0.8056	0.8090	0.8050

Table 2: Accuracy Scores before Grid Searching

Final Scores

The final accuracy scores can be seen in Appendix E and table 3. As you can see, there is a slight improvement in scores before and after grid searching. The final XGB score of 95.58% accuracy is a clear improvement over the majority-class prediction score of 80.56%. I tried to find ways to improve on this score, but it has been hard; it seems like I am pushing the limit, and therefore I think this a good stopping point for this project. Most of the improvements stated in this document will result in small changes in this score; however, I expect no large improvements.

Model	CV_Score	HO_Out-Of-Sample_Score	HO_In-Sample_Score
XGBoost Classification	0.9558	0.9479	0.9854
Random forest Classification	0.9386	0.9419	1.0000
Decision Trees Classifier	0.9373	0.9452	0.9467

Table 3: Accuracy Scores after Grid searching

To make the final predictions, I will use the two best scoring algorithms, which are XGBoost and Random Forest Classification. In Figures 4 and 5, the final prediction value frequencies can be seen. As can be observed, RFC is slightly more forgiving then XGB regarding loan default predictions. With both scores being high, I opted to use both algorithms in the loan officer app to give a 'second opinion' with RFC.
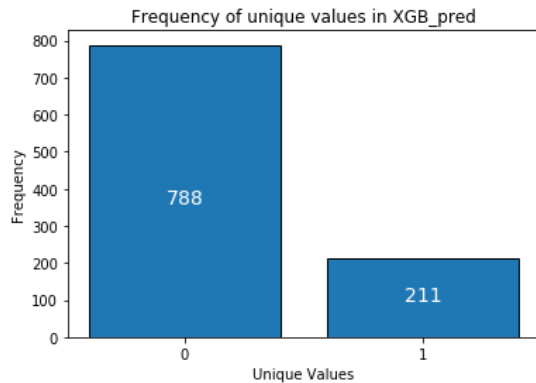
Figure 4: Frequency of unique values in XGB Prediction
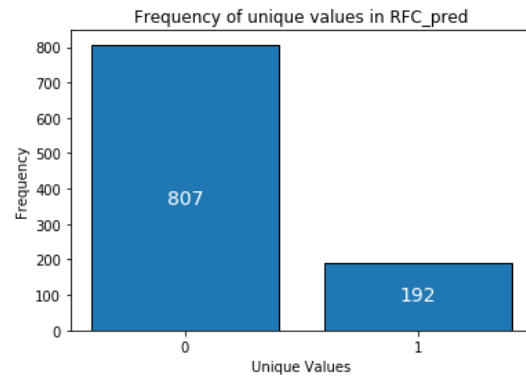
Figure 5: Frequency of unique values in RFC Prediction

App

Objective

      The objective of the app is to allow loan officers to qualify a business for a potential loan. I set an additional objective for myself, namely that the app has to be standalone from the machine learning code. This made the app code longer (+- 700 lines); however, now it can be used on any computer without having to run through the Machine learning code first to train models. I also uploaded all data to GitHub to be able to pull data directly from the internet, further allowing the app to be standalone from any environment. The app also incorporates the web scraping column Recession. The UI can be seen in appendix F.

Use

      The app is a tool and therefore has an advisory role in the process it; therefore, outputs only objective results and no implication, the means possible output is as follows: e.g., "No Default predicted" and "Only Random Forest predicts Default."

Functions

      What the app does run all of the relevant machine learning and web scraping code in a large function when the submit button has been clicked. The app downloads, cleans, feature engineers, and submits both train and test data sets to both XGBoost and RFC algorithms with the hyperparameters found in the grid search and which then individually predict the outcome of the test data set created from user input. There are two features the app incorporates namely;

1. A data dump to CSV of the last input made by the user so the data can be stored.
2. Free-submission check buttons. Small functions that check if the data entered is an integer and has the right format.

References

Analytics India Magazine. (2018, 01 19). *7 Types Of Classification Algorithms*. Retrieved from Analytics India Magazine: https://analyticsindiamag.com/7-types-classification-algorithms/

Analytics Vidhya. (2016, 03 01). *Complete Guide to Parameter Tuning in XGBoost with codes in Python*. Retrieved from https://www.analyticsvidhya.com/: https://www.analyticsvidhya.com/blog/2016/03/complete-guide-parameter-tuning-xgboost-with-codes-python/

Boston School of Public Health. (2016). *Central Limit Theorem*. Retrieved from Boston School of Public Health: http://sphweb.bumc.bu.edu/otlt/MPH-Modules/BS/BS704_Probability/BS704_Probability12.html

Federal Reserve Bank of St. Louis. (2020, 01 31). *Dates of U.S. recessions as inferred by GDP-based recession indicator*. Retrieved from Federal Reserve Bank of St. Louis: https://fred.stlouisfed.org/series/JHDUSRGDPBR

Federal Reserve Bank of St. Louis. (n.d.). *About*. Retrieved from Federal Reserve Bank of St. Louis: https://research.stlouisfed.org/about.html

Appendix

Appendix A: CodeBook

	Variable name	Data type	Description of variable
0	LoanNr_ChkDgt	Text	Identifier – Primary key
1	Name	Text	Borrower name
2	City	Text	Borrower city
3	State	Text	Borrower state
4	Zip	Text	Borrower zip code
5	Bank	Text	Bank name
6	BankState	Text	Bank state
7	NAICS	Text	North American industry classification system …
8	ApprovalDate	Date/Time	Date SBA commitment issued
9	ApprovalFY	Text	Fiscal year of commitment
10	Term	Number	Loan term in months
11	NoEmp	Number	Number of business employees
12	NewExist	Text	1 = Existing business, 2 = New business
13	CreateJob	Number	Number of jobs created
14	RetainedJob	Number	Number of jobs retained
15	FranchiseCode	Text	Franchise code, (00000 or 00001) = No franchise
16	UrbanRural	Text	1 = Urban, 2 = rural, 0 = undefined
17	RevLineCr	Text	Revolving line of credit: Y = Yes, N = No
18	LowDoc	Text	LowDoc Loan Program: Y = Yes, N = No
19	ChgOffDate	Date/Time	The date when a loan is declared to be in default
20	DisbursementDate	Date/Time	Disbursement date
21	DisbursementGross	Currency	Amount disbursed
22	BalanceGross	Currency	Gross amount outstanding
23	MIS_Status	Text	Loan status charged off = CHGOFF, Paid in full…
24	ChgOffPrinGr	Currency	Charged-off amount
25	GrAppv	Currency	Gross amount of loan approved by Bank
26	SBA_Appv	Currency	SBA's guaranteed amount of approved loan

Appendix B: Skewness

Variable	Skew
NoEmp	70.913397
RetainedJob	16.645331
CreateJob	14.712354
Recession	8.883851
SBA_Appv	5.030664
DisbursementGross	4.700242
GrAppv	4.611379
FranchiseCode	4.49458
Term	1.786785
NewExist	0.7227
Bank	0.416096
UrbanRural	0.261759
ApprovalFY	-2.004951

Appendix C: Optimal Hyperparameters found in Grid Search

<u>XGBoost</u>

HyperParameter	Value
Max_Depth	8
Min_Child_Weight	5
Gamma	0.4
Colsample_Bytree	0.6
SubSample	0.8
Reg_Alpha	0
Learning_Rate	0.08
N_Estimators	230
Reg_Lambda	0.41

<u>Random Forest</u>

HyperParameter	Value
N_Estimators	290
Max_Features	Auto
Max_Depth	55
Criterion	Gini

<u>Decision Trees</u>

HyperParameter	Value
Max_Depth	10
Criterion	Gini
Min_Samples_Leaf	30

Appendix D: Scores before Grid searching
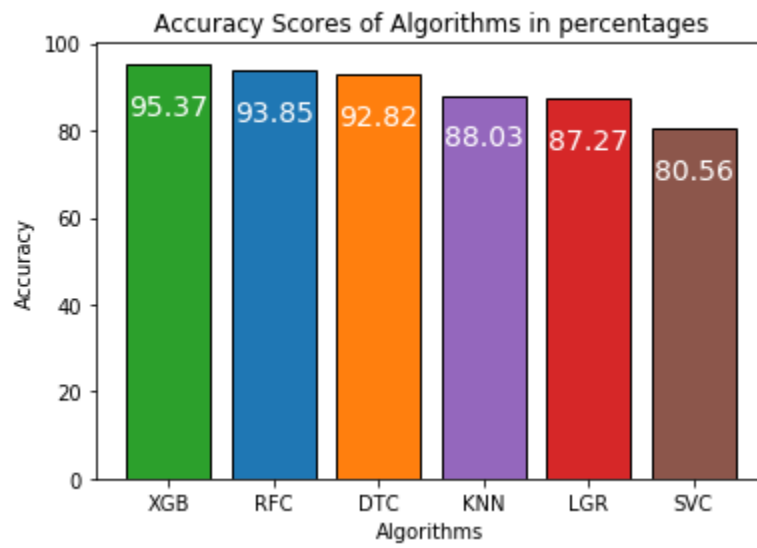
Figure 4: Accuracy Scores before Grid Searching

Appendix E: Scores after Grid Searching

Accuracy Scores after Grid searching

Accuracy Scores of Algorithms in percentages
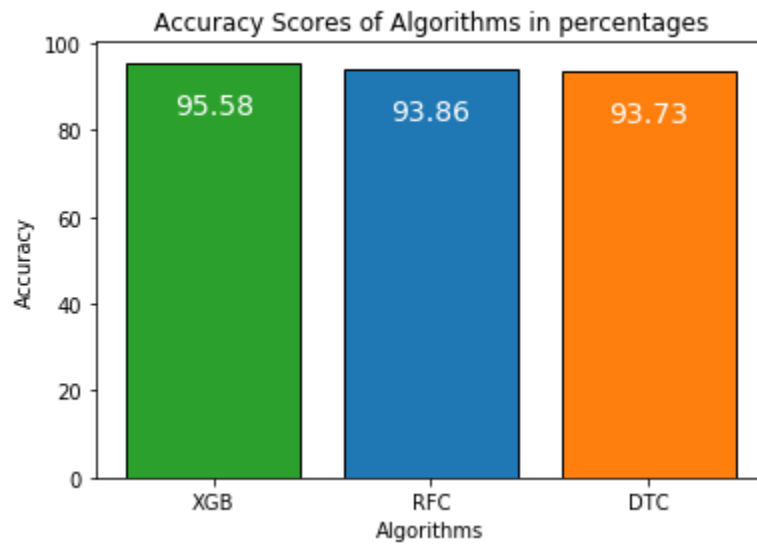
Accuracy Scores after Grid searching

Appendix F: App UI