Machine Learning Engineer Nanodegree

Capstone Project: Loan Default Prediction

Ricardo Szczerbacki
October 27, 2019

I. Definition

Introduction

In the credit loan business, excessive uncertainties about credit applications default risk are dangerous to credit companies and borrowers alike.

Greater uncertainties lead to more restrictions to credit applications and too many restrictions can impede potential capable creditors' access to the credit they need and reduce the income of credit companies, while insufficient restrictions can impose unpayable debts to clients and losses to loan companies.

Looking forward to reduce this uncertainty and broaden the financial inclusion for the unbanked population, the competition “Home Credit Default Risk” was created on **Kaggle ®,** by the company **Home Credit ®,** focused on the search of algorithms to better predict the default credit applications based on data available of the applicants, their assets and their previous credit applications and payments history. The competition can be accessed in https://www.kaggle.com/c/home-credit-default-risk.

The main idea is that, provided that we have enough previous loans information, including the asset data and credit history of the applicants, and the “outcome” of the loan, i.e., if the clients were indeed able to pay the loan, a Machine Learning algorithm could be used to create a model that, to a certain extent, predicts if an applicant will have problems paying his loan.

We can see this as a binary classification problem, where we want to predict, with a certain probability, if a borrower will have payment difficulties or not.

About the Data

The dataset used for this project is the same one originally provided for the competition and contains a train set with 307,511 previous loan applications, with information about the application and the applicant, including data on previous credits (and payments) from the credit company and from other financial institutions and repayment history for the previous disbursed credits.

The data available includes relevant information about the applicant’s car, house, income, education, occupation, marital status, children, documents presented and other information that should, in different degrees, correlate to the capacity of the applicants to honor their debts.

Also, the data on previous loans and payments can, at some extent, help to identify behavior patterns that could have influence on the probability of a credit default loan.

Evaluation Metrics

For the evaluation metric I will use, as proposed by the competition, the area under the ROC (Receiver Operating Characteristic) curve, between the predicted probability of an application being default and the observed target.

To understand the AUC – ROC (area under the curve for the ROC curve), lets start explaining the how the ROC is generated.

The ROC curve is a graphical plot of the true positive rate against the false positive rate at various discrimination threshold settings.

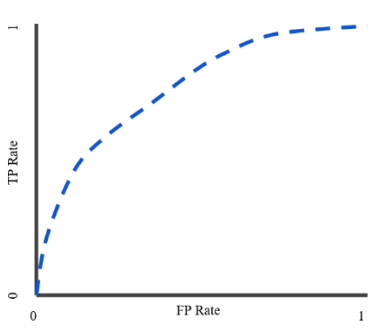
The true positive rate, also known as recall, can say how good our model is identifying the positive samples, and is calculated by the following formula:

$$True\ Positive\ Rate = \frac{True\ Positives}{True\ Positives + False\ Negatives}$$

The false positive rate, that can be also seen as the probability of false alarms, can be calculated as follows:

$$False\ Positive\ Rate = \frac{False\ Positives}{False\ Positives + True\ Negatives}$$

An example of a ROC curve can be seen in the figure 1a. The area under the curve (AUC), that extracts one value from the ROC curve, to be used as a metric, can be seen in the figure 1b.



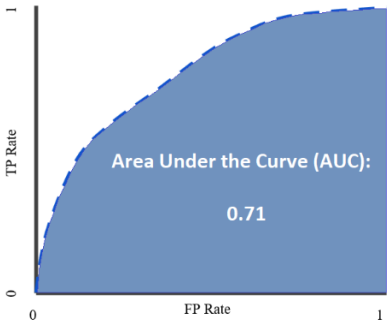


Figure 1a (left): an example of an ROC curve.

Figure 1b (right): the same ROC curve with the corresponding AUC value.

The area under the curve (AUC) relates to how well the model is capable of distinguishing between classes and is often used for model comparison in Machine Learning [4].

II. Analysis

Data Exploration

From the dataset used the following six files, originally in a CSV text format, were used in this project.

Following I will show some relevant information about each one of these files and the data contained.

application_train.csv

This file contains information about the past applications for loans in the credit company, like the amount of credit requested and the hour of the application, and also information about the applicant, like income, age, current car and current house (figure 2).

SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT
100002	1	Cash loans	M	N	Y	0	202500.0	406597.5
100003	0	Cash loans	F	N	N	0	270000.0	1293502.5
100004	0	Revolving loans	M	Y	Y	0	67500.0	135000.0
100006	0	Cash loans	F	N	Y	0	135000.0	312682.5
100007	0	Cash loans	M	N	Y	0	121500.0	513000.0

Figure 2: few rows and columns from the table (file) **application_train.csv**.

This table have 307,511 rows (data samples) and 122 columns.

As the first column **SK_ID_CURR** is a number the uniquely identifies each application (primary key) and correlates each application to data present in the other files, it will be used to join tables and not as a feature in our models.

The remaining 121 columns can be seen as features that could be used to train our models.

For each feature I checked for missing values. Some features didn't have missing values, some only few and others had much more.

There are numerical and categorical features in the file. And the categorical features, in some cases are just binary, like "Y" or "N" (ex: FLAG_OWN_CAR), and in other cases have multiples values (ex: NAME_INCOME_TYPE: "Working", "Pensioner", etc).

The numerical features have different ranges (max and min values), different distribution of data, with some more normal like (figure 3a) and others more skewed (figure 3b) distributions.

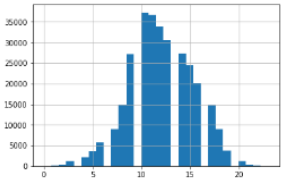


Figure 3a (left): Feature HOUR_APPR_PROCESS_START, with normal like distribution (with gaps)

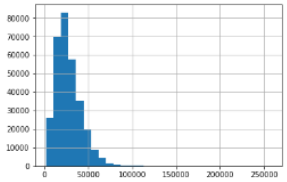


Figure 3b (right): Feature AMT_ANNUITY, with a skewed data distribution

Each of the 121 features were analyzed for the items described above (missing values, difference of ranges, skewness, categorical nature) and treated accordingly to get most of the prediction models, as will be shown in further in this report.

bureau.csv and bureau_balance.csv

The file *bureau.csv* have 27.299.925 rows and 3 columns, while the file *bureau_balance.csv* have 1.716.428 rows and 17 columns.

These files contain information from the Credit Bureau about previous applications (in ther credit companies) related to the same applicant from some application in the file *application_train.csv*. The contents of the files include also monthly balance from previous credits with payment status and will be consolidated as features for the applications later.

SK_ID_CURR	SK_ID_BUREAU	CREDIT_ACTIVE	CREDIT_CURRENCY	DAYS_CREDIT	CREDIT_DAY_OVERDUE	DAYS_CREDIT_ENDDATE	DAYS_ENDDATE_FACT
215354	5714462	Closed	currency 1	-497	0	-153.0	-153.0
215354	5714463	Active	currency 1	-208	0	1075.0	NaN
215354	5714464	Active	currency 1	-203	0	528.0	NaN
215354	5714465	Active	currency 1	-203	0	NaN	NaN
215354	5714466	Active	currency 1	-629	0	1197.0	NaN

Figure 4: few rows and columns from the table (file) **bureau.csv**.

	SK_ID_BUREAU	MONTHS_BALANCE	STATUS
0	5715448	0	C
1	5715448	-1	C
2	5715448	-2	C
3	5715448	-3	C
4	5715448	-4	C

Figure 5: few rows from the table (file) **bureau_balance.csv**.

previous_applications.csv

This file has 1.670.214 rows and 37 columns.

This table contains information about previous applications in the same credit company, related to some application in *application_train.csv*.

The information includes the amount and payment status of each previous application.

The contents of this file will be consolidated as new features for the applications later.

	SK_ID_PREV	SK_ID_CURR	NAME_CONTRACT_TYPE	AMT_ANNUITY	AMT_APPLICATION	AMT_CREDIT	AMT_DOWN_PAYMENT	AMT_GOODS_PRICE
0	2030495	271877	Consumer loans	1730.430	17145.0	17145.0	0.0	17145.0
1	2802425	108129	Cash loans	25188.615	607500.0	679671.0	NaN	607500.0
2	2523466	122040	Cash loans	15060.735	112500.0	136444.5	NaN	112500.0
3	2819243	176158	Cash loans	47041.335	450000.0	470790.0	NaN	450000.0
4	1784265	202054	Cash loans	31924.395	337500.0	404055.0	NaN	337500.0

Figure 6: few rows and columns from the table (file) **previous_applications.csv**.

credit_card_balance.csv

This file has 3.840.312 rows and 23 columns.

This table contains information about balance and payment status for the credit cards from the credit company from the applicant of some application in the file *application_train.csv*.

The contents of this file will be consolidated as new features for the applications later.

	SK_ID_PREV	SK_ID_CURR	MONTHS_BALANCE	AMT_BALANCE	AMT_CREDIT_LIMIT_ACTUAL	AMT_DRAWINGS_ATM_CURRENT	AMT_DRAWINGS_CURRENT
0	2562384	378907	-6	56.970	135000	0.0	877.5
1	2582071	363914	-1	63975.555	45000	2250.0	2250.0
2	1740877	371185	-7	31815.225	450000	0.0	0.0
3	1389973	337855	-4	236572.110	225000	2250.0	2250.0
4	1891521	126868	-1	453919.455	450000	0.0	11547.0

Figure 7: few rows and columns from the table (file) **credit_card_balance.csv**.

POS_CASH_balance.csv

This file has 10.001.358 rows and 8 columns.

This table contains information about previous point of sales loans (loans used directly while clients are buying goods) from the same applicant of some application in the file *application_train.csv*.

The contents of this file will be consolidated as new features for the applications later.

	SK_ID_PREV	SK_ID_CURR	MONTHS_BALANCE	CNT_INSTALMENT	CNT_INSTALMENT_FUTURE	NAME_CONTRACT_STATUS	SK_DPD	SK_DPD_DEF
0	1803195	182943	-31	48.0	45.0	Active	0	0
1	1715348	367990	-33	36.0	35.0	Active	0	0
2	1784872	397406	-32	12.0	9.0	Active	0	0
3	1903291	269225	-35	48.0	42.0	Active	0	0
4	2341044	334279	-35	36.0	35.0	Active	0	0

Figure 8: few rows from the table (file) **POS_CASH_balance.csv**.

installments_payments.csv

This file has 13.605.401 rows and 8 columns.

This table contains information about previous installments from the applicant of some application in the file *application_train.csv*.

The information includes the amount of the installment and amount of payment.

The contents of this file will be consolidated as new features for the applications later.

PREV	SK_ID_CURR	NUM_INSTALMENT_VERSION	NUM_INSTALMENT_NUMBER	DAYS_INSTALMENT	DAYS_ENTRY_PAYMENT	AMT_INSTALMENT	AMT_PAYMENT
54186	161674	1.0	6	-1180.0	-1187.0	6948.360	6948.360
30831	151639	0.0	34	-2156.0	-2156.0	1716.525	1716.525
85231	193053	2.0	1	-63.0	-63.0	25425.000	25425.000
52527	199697	1.0	3	-2418.0	-2426.0	24350.130	24350.130
14724	167756	1.0	2	-1383.0	-1366.0	2165.040	2160.585

Figure 9: few rows and columns from the table (file) **installments_payments.csv**.

Algorithms and Techniques

As stated before in this report, the problem we need to solve can be seen as a binary classification problem, where we want to predict, with a certain probability, if a borrower will have payment difficulties or not.

There are several algorithms that can be used to solve this kind of problem and for this project it was chosen three. The algorithms chosen and the reasons for their selection I will describe now.

Logistic Regression is a statistical classification model where the goal is to predict the probability of occurrence a specific classification or event. The Logistic regression uses a logistic function to model the relationship between the dependent variable X and the categorical outcome Y [5].

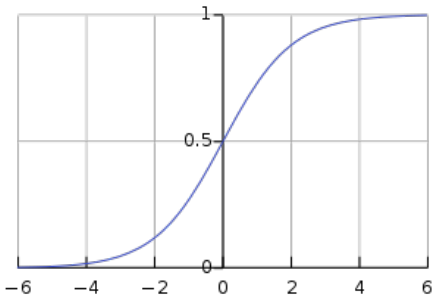


Figure 10: Standard logistic function [6].

When dealing with a binary classification problem, the use of a Logistic Regression classifier is one of the most popular and cited approach. And that is why I selected the Logistic Regression as the first algorithm to be evaluated.

The Random Forest Classifier is an ensemble method that use decorrelated decision trees as “weak learners” to form “strong learners” through a bootstrap aggregating strategy (bagging), where each “weak learner” vote with equal weight [5,7].

This approach outperformed other Machine Learning methods in many previous studies on Loan Default Prediction [1,2] and this is why I chose the Random Forest Classifier as the second algorithm to be evaluated.

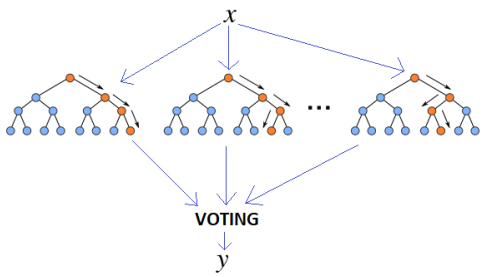


Figure 11: Random Forest method simplified.

Finally, we have the eXtreme Gradient Boosting (XGBoost), an implementation of the Gradient Boosted Tree algorithm, that tries to use the most of the computational resources available to achieve the best computational speed and model performance, using parallelization, out-of-core computing and cache optimization [3,8].

The gradient boosting is also an ensemble method where we combine “weak learners” to build “strong learners”, but here we apply successively the models using gradient descent to minimize the loss function and gradually improve our model.

Because it is one of the most popular classifiers in Machine Learning for hackathons and competitions nowadays, I chose the XGBoost as the third algorithm to be evaluated for our problem.

For the Random Forest and the XGBoost algorithms there is great number of possibilities for their hyperparameters that we should evaluate in search for the best models. So, for these two methods I will perform hyperparameter tuning.

Two common approaches for hyperparameter tuning are Grid Search and Randomized Search methods.

While the Grid Search tries exhaustively all combinations of the preset list of values for the hyperparameters, the Randomized Search uses a fewer random combinations of that list, demands a fraction of the time it would take to Grid Search and often also picks the best result.

Because Grid Search is much more time consuming and the results of the two methods are usually similar, in many known cases, I chose to use the Randomized Search to find the best hyperparameters to our models.

To evaluate the generalization ability of our models I used a 10-fold cross validation approach, where the training dataset where partitioned into 10 mutually exclusive subsets of equal sizes (folds) and the models were trained 10 times, using each time one of the folds for testing the model trained with the remaining k-1 folds [5].

A nested cross validation strategy was considered, using k-fold cross validation for the hyperparameter tuning and another k-fold cross validation for each initial fold to evaluate the models, but it was too computationally expensive. So, I decided to split the initial data in two. Using 10% of the data to the hyperparameter tuning and in the remaining 90% I used a 10-fold cross validation strategy to evaluate the models.

At the end it was also evaluated average ensembles models combining all the final three models and combinations of each two to check if any of these combinations can perform reasonably better than each model alone.

Benchmark

Although there are previous studies for this domain, as the data chosen as predictors vary widely in these studies, the direct comparison of the results obtained is not adequate.

Therefore, I chose to benchmark the evaluated models with two models: a random prediction algorithm, as the baseline model to be surpassed, and the winning contest solution, as the state-of-the-art classifier for this specific data, but that is too much complex to be adopted as a general and practical solution.

The winning contest solution achieved a score of 0.81724, as can be seen in the competition leaderboard in:

https://www.kaggle.com/c/home-credit-default-risk/leaderboard

Nevertheless, it is important to say that the scores generated in my report are based on a cross validation over 90% of the data made available for the competition, while the official score in the leaderboard is based on a different set of data. So, while we could have some similar results, if the generalization of the model is good enough, we can only use this value as an approximation and compare the result with this information always in mind.

III. Methodology

Data Processing

For the main table (application_train.csv file), that contains information about previous applications and their applicants, I analyzed and processed the data following the steps bellow:

1. **Missing values:** for each feature I checked how many samples (rows) had missing values. Each case where evaluated individually and when the missing value had an implicit meaning like number 0 or categorical value ‘Other’, it was replaced. In the other cases for few (less then 1%) samples with missing values, they were removed from the dataset. The remaining missing values were replaced by the mean of the feature value, for numeric features and ‘None’ for the categorical ones.

If the feature was categorical:

2. **For simple binary categories:** for such cases (like “Y” or “N”) I just replaced the values by the numbers one and zero, using always the number one for the positive (“Y” or “True”) meaning.

3. **For the features with more values**: I used the One Hot Encoding technique, creating one column for each possible categorical value, with numbers one and zero indicating if this value is or not valid for the sample.

If the feature was numerical:

4. **Checked data distribution:** if the histogram showed that data was skewed, I tried to use a log transformation in the feature and if the result seemed to near normal, I applied it to the feature.
5. **Feature scaling:** I applied a min-max scaler to the feature, so I my numerical features have the range, from zero to one. Normalization can help the Logist Regression to converge faster.

For one feature, the OWN_CAR_AGE, a special treatment was made. This feature contains the age of applicants the car, if he/she has one.

If you see the feature histogram (figure 12), you can see that this feature has or seems to have a bimodal distribution.

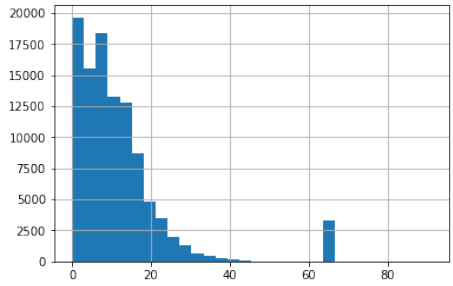


Figure 12: Distribution (histogram) for the feature OWN_CAR_AGE.

If you think about the meaning of the feature you come to realize that cars over a certain age can be in two situations. Can be only old cars or can be considered classic, vintage or antiques. In this case this feature can have a totally different influence over the prediction. So, it is like we have two different features combined in one (age of your regular car and age or your antique).

Because it would be difficult to know if a very old car is an antique, I decided to remove the samples with cars over 40 years old from our training dataset.

For the remaining tables, that contains series values that are associated to some previous applications from our main table, I decided to choose relevant columns from which statistics, like maximum and mean, could create meaningful features for the related previous applications. For example, from the Credit Bureau data, I used the maximum and mean values for amount of the debt and amount overdue, among others, as new features for the related previous applications.

I also create a new derived feature, using only the Credit Bureau information, that tries is an approximated debt to income ratio, that relates but not corresponds to the debt-to-income ratio (DTI) used in the mortgage industry. The total active debts divided by the applicant's income could be good estimator of his capacity to pay his debts.

After all the mentioned data processing our dataset had 307 features and 302,054 samples, that could be used for tuning and evaluating our models.

Implementation

The first step of my implementation was to split my dataset in two parts. One part, with 10% of the samples, to be used in the hyperparameters tuning for the Random Forest and the XGBoost models and the other with 90% of the samples, to be used for the evaluation of the models.

The first approach that I tried was to evaluate the models using all the 307 features. But the computational power and time needed was too high.

The alternative I chose was to make a ranking of the features and try to increase gradually the number of features trying to use always the most important ones, and check when the increase doesn't bring improvement to our models.

We should try to use the least number of the most important features that produce better models.

So, the next step was to create a feature ranking of all our 307 features.

To do the feature ranking I used an Extra Trees Classifier. A plot of how the importance change over the 307 features can be seen in the figure 13.

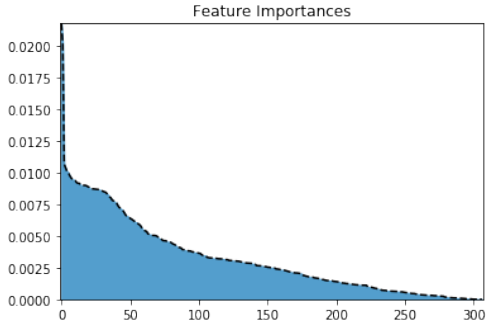


Figure 13: Feature importances of the features ordered by their importance

Following, I used a Randomized Search, using all the 307 features, to find the right hyperparameters to my Random Forest model.

The grid of values of hyperparameters to be evaluated was:

- **Maximum depth of the tree**
  10, 50, 100, None
- **Minimum number of samples required to be at a leaf node**
  1, 2, **4**
- **Minimum number of samples required to split an internal node**
  2, 5, **10**
- **Number of trees in the forest**
  20, 100, 200, **500**

The values in red where the hyperparameters of the best estimator and were used for the model evaluation.

Then, I used also the Randomized Search to find the right hyperparameters to my XGBoost model.

The grid of values of hyperparameters to be evaluated was:

- **Maximum depth of the tree**
  6, **10**, 20
- **Learning rate**
  0.001, 0.01, **0.1**, 0.2, 0,3
- **Subsample ratio of the training instances**
  0.5, 0.8, **1.0**
- **Subsample ratio of columns when constructing each tree**
  0.4, 0.7, **1.0**
- **Subsample ratio of columns for each level**
  0.4, **0.7**, 1.0
- **Minimum sum of instance weight (hessian) needed in a child**
  0.5, 3.0, 5.0, **10.0**
- **Minimum loss reduction required to make a further partition (gamma)**
  0, **0.25**, 0.5, 1.0
- **L2 regularization term on weights (lambda)**
  0.1, 1.0, 10.0, 50.0, **100.0**
- **Number of trees**
  20, 50, **100**

The values in red where the hyperparameters of the best estimator and were used for the model evaluation.

To a first evaluation I considered only half of the features (153) and then split the features in 10 groups with (almost) equal sizes.

The idea was to evaluate the models starting with only the first group of the most important features and repeat the process, adding the always the group of features to the current considered features.

If the plot of the result shows that we reached stable predictions in this half of the features we can stop here. Otherwise we should increase the number of feature and continue the evaluation.

The 153 features initially considered were enough to achieve our goals as we will see later in the refinement section.

Following the steps specified above, the models were evaluated 10 times, respectively using 16, 31, 46, 62, 77, 92, 108, 123, 138 and 153 features.

For each step, with different number of features, I used a 10-fold cross validation to compute the scores for the baseline, Logistic Regression, Random Forest and XGBoost models.

The baseline model was implemented with a simple random prediction classifier, that generates predictions uniformly at random.

The Random Forest and the XGBoost were implemented using the hyperparameters selected in the hyperparameter tuning step.

After analyzing the results, the best models for the Logistic Regression, Random Forest and XGBoost were derived.

With these three models I used ensemble averaging to create combined models using all the three models and combinations with each two models.

At the end, the final results of the Logistic Regression, Random Forest, XGBoost and the derived ensembled models were compared to select one of these models as the final solution to our problem.

Refinement

Because of the use of the Randomized Search, I didn't try different configurations of hyperparameters with the models.

But the multiple evaluations of the baseline and the candidate models, using the number of features of 16, 31, 46, 62, 77, 92, 108, 123, 138 and 153, allowed me to see how the models behaved with an increasing number of features and how well they generalized across the different folds of the data.

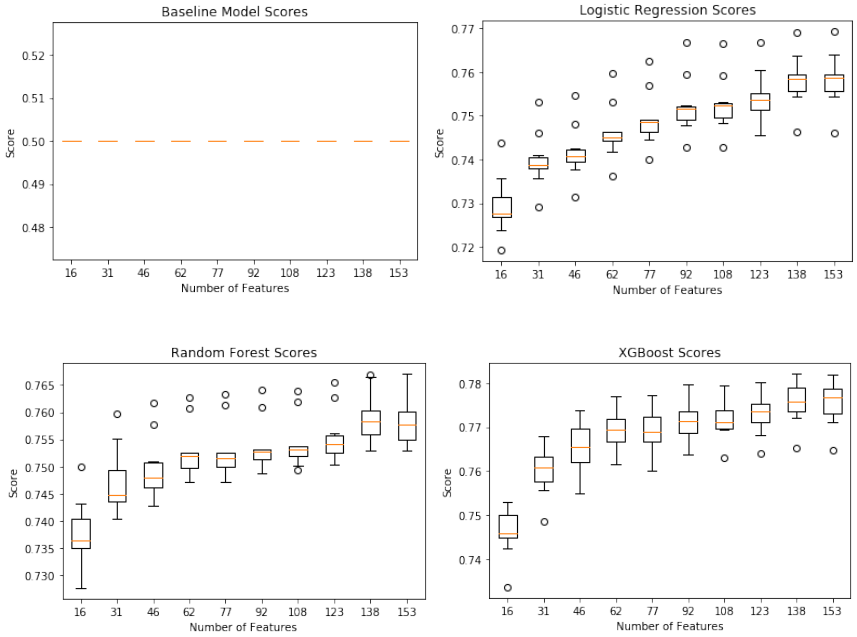


Figure 14: Scores for all of the folds with different number of features, for the baseline and each candidate model.

As we can see in the figure 14, the baseline model always achieved the same score of 0.5 for all folds, with all number of features.

For the Logistic Regression the scores of the folds seems to concentrate near the median (good generalization, but with some outliers). The score values are clearly increasing as

we increase the number of features and seem to be stabilizing near the end (with 153 features considered). The last model, using 153 features, will be used as the final model for the Logistic Regression.

With the Random Forest the generalization is a little worse than the Logistic Regression and the score values also increases and seems to stabilize near 77 features, but we can see a little improvement and new stabilization in the end. So, I will also use the last model, using 153 features, as the final model for the Random Forest.

For the XGBoost the generalization is similar to the Random Forest algorithm (worst in the middle number of features, but similar in the best models, with 153 features) and the score values seems to increase faster than the other models, as we increase the number of features. But also, the number of 153 seems to be a reasonable choice for the final model of the XGBoost.

IV. Results

Model Evaluation and Validation

As we can see in the plot below (figure 15), when we compare the scores of all folds, for all method, including the averaging ensembles of the combinations of the final models of the three methods, the XGBoost have clearly the best results with a reasonable generalization and higher score values than the other candidate models, including the averaging ensembles.

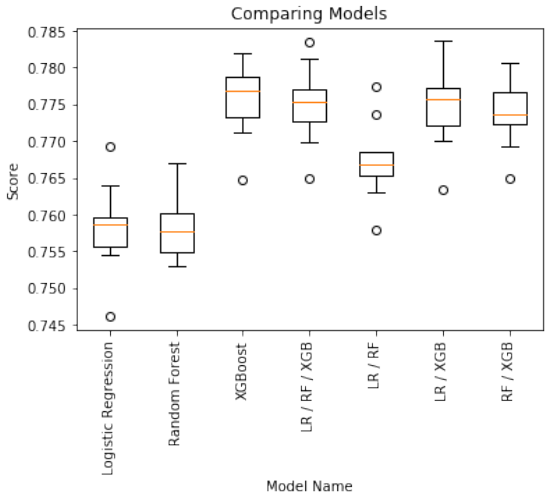


Figure 15: Final scores for all candidate models, including the ensemble combinations.

So, from now on, I will call the final XGBoost classifier, with the hyperparameters selected in the hyperparameters tuning step and using the best 153 ranked features, the **solution model**.

The solution model has the following statistics for the score values across the 10 folds:

Minimum	0.76479
Maximum	0.78191
Mean	0.77569
Standard Deviation	0.00490

Even the minimum value is close to or higher than most of the results of the other candidates, except for the ensembles.

Even if the results of ensembles are similar, the solution model have better results and is a simpler model.

The minimum score is close to the maximum, for the XGBoost, across 10 folds of independent data, so we can say that the solution model seems to generalize well and to be robust to changes in the data.

Justification

Benchmark:

Model	Score
Baseline	0.50000
XGBoost (solution model)	0.77569
Winner of the contest *	0.81724

(*) is important to be said that even if the winner of the contest solution used the same dataset for the training, different data were used to the test and scoring, so the numbers are not directly comparable.

The scores from the solution model shows that the solution generalizes well, is robust, performs far better than the baseline solution and close to the winner of the contest (WoC) model, noting that the WoC model if far more complex and used a different test set, what should lead to some difference in the scores.

V. Conclusion

Reflection

All the Data Analysis and pre-processing steps took a great amount of time. From 60 to 70% of the time was spent preparing the data for the algorithms.

And the new feature that was created using domain knowledge (that correlates to the debt-to-income ratio, from the mortgage industry), was not as good as expected, with only a 34th position, among the 307 features.

The feature ranking and the multiple evaluations with the increasing number of features were critic to the feature selection, that was necessary due to the number of features.

To problem was very interesting to investigate, because the it is a very familiar one to most of the people. A great number of people get loans or have applications denied, so understanding loan default chances can help not only credit companies, but also all of us, potential clients.

Improvement

Many improvements could be made to the process of finding the solution model.

First, with more computational power, a grid search could be used, with more values in the grids, to find better hyperparameters to our models.

Artificial Neural Networks could be also evaluated, alone and combined with the other models.

Also, as the training data was unbalanced (with only about 10% of samples in the possible default class) a technique such as SMOTE could be used to generate more samples artificially.

Finally, better results can almost always be achieved, through deep understanding of the data and with enough time and computational power to try and evaluate different models and approaches.

References

[1] Alomari, Zakaria. (2017). Loan Default Prediction and Identification of Interesting Relations between Attributes of Peer-to-Peer Loan Applications. New Zealand Journal of Computer-Human Interaction.

[2] Lifeng Zhou, Hong Wang. (2012). Loan Default Prediction on Large Imbalanced Data Using Random Forests. TELKOMNIKA (e-ISSN: 2087-278X), Vol.10, No.6, October 2012, pp. 1519~1525.

[3] Chen, T., & Guestrin, C. (2016). XGBoost: A Scalable Tree Boosting System. KDD.

[4] Wikipedia contributors. (2019). Receiver operating characteristic. In Wikipedia, The Free Encyclopedia. Retrieved October 5, 2019, from: https://en.wikipedia.org/w/index.php?title=Receiver_operating_characteristic&oldid=917442289

[5] Maritte Awad, Rahul Khanna. (2015). Efficient Learning Machines: Theories, Concepts and Applications for Engineers and System Designers. ISBN 978-1-4302-5989-3.

[6] Wikipedia contributors. (2019, October 24). Logistic function. In Wikipedia, The Free Encyclopedia. Retrieved October 27, 2019, from: https://en.wikipedia.org/w/index.php?title=Logistic_function&oldid=922872267

[7] Wikipedia contributors. (2019, October 15). Random forest. In Wikipedia, The Free Encyclopedia. Retrieved October 27, 2019, from: https://en.wikipedia.org/w/index.php?title=Random_forest&oldid=921334071

[8] Wikipedia contributors. (2019, October 21). Gradient boosting. In Wikipedia, The Free Encyclopedia. Retrieved October 27, 2019, from: https://en.wikipedia.org/w/index.php?title=Gradient_boosting&oldid=922411214