

Capstone Project

Machine Learning Nanodegree

Rodney Sales Nogueira Jr

November 2017

1 Definition

1.1 Project Overview

In this project, I created machine learning models to analyse credit scoring based on the Kaggle Give Me Some Credit Competition [4]. This study is focused on our company ongoing *AI project* to analyse potential credit frauds, and credit scores.

”The need for credit analysis was born in the beginning of commerce in conjunction with the borrowing and lending of money, and the purchasing authorization to pay any debt in future. However, the modern concepts and ideas of credit scoring analysis emerged about 70 years ago with Durand” [2].

A credit score is a numerical expression based on a level analysis of a person’s credit files, to represent the creditworthiness of an individual. A credit score is primarily based on a credit report information typically sourced from credit bureaus [1]

So the main idea of credit scoring models is to identify the features that influence the payment or the non-payment behavior of the customer as well as his default risk, occurring as the classification into two distinct groups characterized by the decision on the acceptance or rejection of the credit application [2].

Some of the problems are to solve a common supervised learning approach to credit scoring, as the scores can be a numerical expression in a certain range, such as a FICO Score: 300-850. Our particular goal is to comprehend how the machine learning design process works for applications in this domain. So credit scoring is a good way to start.

The goal of this project is to understand, and learn the challenges of building a machine learning model to analyse credit scoring, based on publicly available data. The idea is to have a overview and a start in the solutions for kind of problems.

1.2 Problem Statement

As stated by [2] sometimes the decision is binary based on an acceptance/rejection rate, meaning that this problem can be treated as a classification problem. Meaning that the datasets provided might have common features such as: age, credit status, expenses, income, assets, marital status, etc ...

Our problem was focused on the Kaggle Give Me Some Credit Competition to study the data, and focus on selecting a good set of features, and creating a Model to a late submission on the competition.

1.3 Metrics

The metrics in the competition is the AUC [4]. The competition is finished, and there's a lot of submissions, and discussions on the forum around the solutions, benchmarks, and the data. It will be easy to know if the models are fitting the data well. The best models for the competition are around > 0.8 , so we have a good criteria for comparison of our own results.

2 Analysis

2.1 Data Exploration

The Give Me Some Credit dataset contains both a training set, and a testing set. The training set contains 150000 entries, and the test set contains 101503 entries, but the test set does not contains the *ground truth* variable.

The features of the dataset are:

- SeriousDlqin2yrs, yes or no, and it's the class value
- RevolvingUtilizationOfUnsecuredLines
- Age
- NumberOfTime30-59DaysPastDueNotWorse
- DebtRatio
- MonthlyIncome
- NumberOfOpenCreditLinesAndLoans
- NumberOfTimes90DaysLate
- NumberRealEstateLoansOrLines
- NumberOfTime60-89DaysPastDueNotWorse
- NumberOfDependents

There are two classes, so it's a binary classification problem. The features includes two percentage values (RevolvingUtilizationOfUnsecuredLines, DebtRatio), a real value (MonthlyIncome), and the rest are integers values.

The exploration phase also included searching the competition forum, looking for the competitors thoughts on the data. As stated there is a problem with the class imbalance of the data.

2.2 Data Visualization

The training set contains 150.000 entries, the frequency is 93.3% for the first class, and 6.7% for the next class, as shown in the Figure 1. This shows us a great problem and a challenge, there is a big class imbalance in the data, and metrics such as accuracy might not be a good way to validate this kind of data.

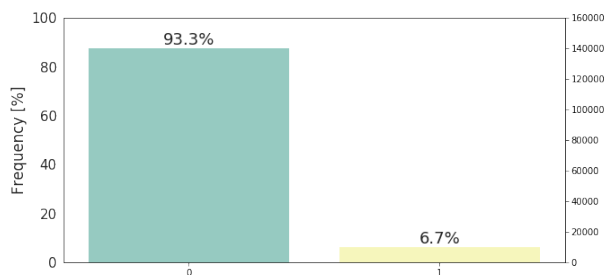


Figure 1: Dataset Class Frequency

More analysis of the data ¹ shows some interesting results, such as the percentage of the missing (NaN) values, which in the column of the MonthlyIncome there is a 19% of missing values.

Also, we can see that the number of occurrences for the people ages grows on the dataset, there are more older people on the data. And we also cross checked that older people tend to have more financial problems.

The same exploration was done with the DebtRatio and the Monthly income, and we can analyse that lower MonthlyIncome tend to have a higher DebtRatio, which makes sense.

2.3 Algorithms and Techniques

According to the forums [4, 7, 8] the algorithms which achieved good results were:

1. Random Forest
2. Gradient Boosting (XGB)
3. QDA Ensemble

¹<https://github.com/rodsnrj/credit-scoring/blob/master/Data.ipynb>

4. Neural Networks

As planned in the capstone I selected simpler models to analyse the data, and followed with the models used in the forums. The simple models selected were: gaussian naive bayes; logistic regression; and decision trees.

After the initial analysis I narrowed my choices to two approaches: neural networks, which is a trending nowadays with the deep neural networks; and the gradient boosting classifier which shown good results in the competition leaderboards, and my initial tests.

An artificial neural network is a biologically inspired algorithm, that perform certain tasks, like clustering, classification, or regression. They can be viewed as weighted directed graphs, where neurons are nodes, and directed edges with weights are the connections between the neuron outputs and inputs, as shown in Figure 2. The input layer is feed with the features of our data, the hidden layers are the networks core, where it learns to classify our data, and the output layer give us the final result, depending on the network architecture. For a binary classification network, the output layer would give us a percentual value based on the current classes, closer to 0, or 1 [3].

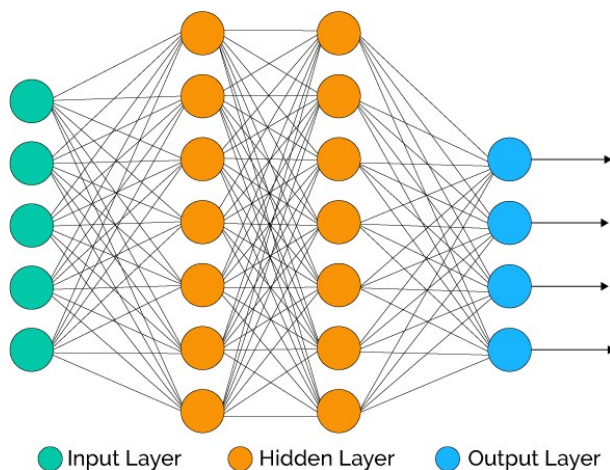


Figure 2: Artificial neural network overview

The Figure 3 shows an overview of the operations of the network neurons, or layers. The layer operation is a matrix multiplication followed by a activation function of the neuron, and optionally the addition of a bias, denoted by: $f(X * W) + b$.

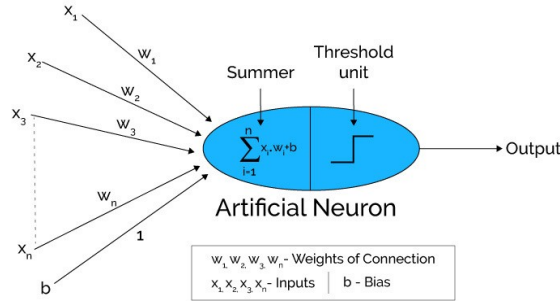


Figure 3: Artificial neural network neuron

There are several neural network architectures, as shown in Figure 4. For our study we use a simple fully connected multi layer perceptron model. The training this neural networks is based on the gradient descent, and back propagation algorithm. In back propagation the error (different between the desired output, and current output), is propagated backward from the output layer to the input layer updating the weights of the hidden layers. This process is done for several iterations on the training data [3].

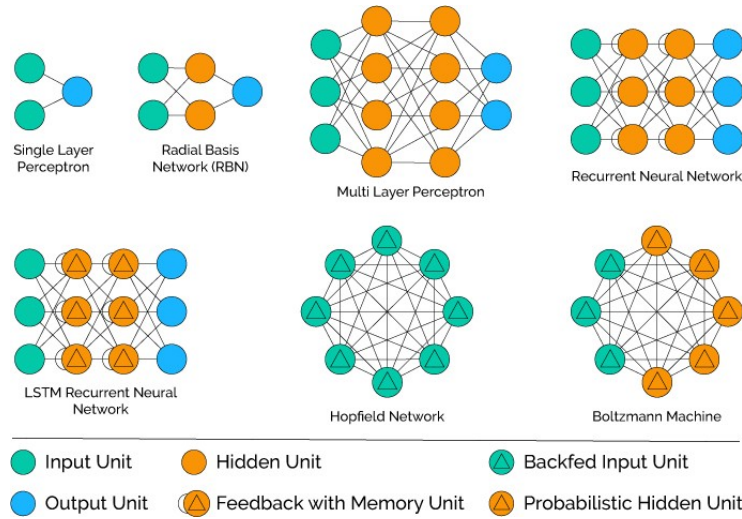


Figure 4: Popular neural network architectures

And a Gradient boosting algorithm is an ensembling technique, which means that prediction is done by an ensemble of simpler estimators. While this theoretical framework makes it possible to create an ensemble of various estimators, in practice we almost always use GBDT — gradient boosting over decision trees. This is the case I cover in the demo, but in principle any other estimator could be used instead of a decision tree [5]. The aim of gradient boosting is to create

(or "train") an ensemble of trees, given that we know how to train a single decision tree. This technique is called boosting because we expect an ensemble to work much better than a single estimator [5].

2.4 Benchmark

To evaluate my testings, and benchmark my models, I used the hiper-parameters and configurations exposed in the Competition Forums, with the models used by them, and evaluated it on a test set made with 20 to 30% of the data in the training set.

The mean AUC for a 10 fold cross-validation from all of the following were:

- GaussianNB: 0.71
- DecisionTreeClassifier: 0.61
- LogisticRegression: 0.70
- MLPClassifier: 0.83
- AdaBoostClassifier: 0.86
- RandomForestClassifier: 0.78
- GradientBoostingClassifier: 0.86
- BaggingClassifier: 0.78
- XGBClassifier: 0.86

And for the mean AUC of a 10 fold cross-validation, for the experiments done in the 3rd competitor's feature set:

- GaussianNB: 0.83
- DecisionTreeClassifier: 0.61
- LogisticRegression: 0.86
- MLPClassifier: 0.58
- AdaBoostClassifier: 0.86
- RandomForestClassifier: 0.78
- GradientBoostingClassifier: 0.86
- BaggingClassifier: 0.78
- XGBClassifier: 0.86

Then I started to develop my own tests with my configurations on the models, and compared with those results.

And lastly I used the *late submission* of the competition to benchmark the final result of each one of my selected models. I aimed for score of at least 0.86, but my goal was to get close to the sample entry benchmark (0.864248) as shown in Figure 5.


356	▼ 62	AdjustedRSquared		0.864250	12	6y
		 Sample Entry Benchmark		0.864248		
357	▼ 60	Anthony Goldbloom		0.864248	2	6y
358	▼ 60	bamboochen		0.864248	1	6y

Figure 5: Kaggle Competition Sample Entry Benchmark

3 Methodology

3.1 Data Preprocessing

To start the simple evaluation of the classifier I've preprocessed the training feature set with the Scikit-Learn Standard Scaler ². And uploaded a submission with the GradientBoosting, which achieved 0.77 AUC in the Cross-Validation.

Your most recent submission				
Name	Submitted	Wait time	Execution time	Score
first_submission_gb.csv	a few seconds ago	8 seconds	1 seconds	0.843094
Complete				
Jump to your position on the leaderboard				

Figure 6: Submission on the Simple Scaled Feature Set

For the data pre processing I based myself in the code shared by the 3rd place on the competition [7]. The feature set consisted in 80 features varying from the Log values of the main features, and some Unknown values for the same features. The implementation is on the Data Transform Notebook ³.

3.2 Implementation

The implementation is split in three main stages:

- Implementation of the data handling, pre processing, and visualizations

²<http://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html>

³<https://github.com/rodsnrj/credit-scoring/blob/master/Data-Transform.ipynb>

- The implementation of the classifiers, including training, splitting and testing
- The implementation and refinements of the algorithms, and of the submissions generation functions

For the first stage I started with the exploration of the discussion threads from the Kaggle Competition, and based some of my implementations according to the community shared studies, and codes.

The data handling was made with Pandas, Numpy, and Scikit-Learn toolkits, using both Pandas Queries, and and Numpy utility functions. Scikit-Learn was also used to pre-process and select my features. And the visualizations were made using Matplotlib plots and the built-in plots in Pandas.

The final data pre-processing was solely based on the 3rd place shared *R* script. Where I extracted the same 80 features to compare with my own previous feature selections.

The implementations of the classifiers then followed using Scikit-Learn, in addition with Keras to build my Neural Network models. I've created a few helper functions to generate multiple results from multiple classifier trainings and scorings.

I've built simple functions to: train and test, multiple classifiers, with or without cross-validation; split the dataset into training and testing; and I've also built functions to generate the submission file from both Keras models, or Scikit-Learn classifiers;

Since the implementations were used to the Kaggle competition, which is solely based on finding a good classifier, and submitting its results. I did not focused on having a organized or performatic code.

3.3 Refinement

As stated for the final part of the project I've chosen two models: a artificial neural network; and a gradient boosting classifier.

To refine the models the training dataset was split in two parts: a training set, and a testing set with 20% of the data.

The artificial neural network model refinement included:

- Adding more hidden layers, and changing their input/output size
- Changing the activation functions
- Chaning the loss function, and the optimizers.

Then followed by training with 10 to 50 epochs, using a batch size of around 2048/4096. After the training is finished the model is evaluated in the test set. The model was considered good when the AUC score was above 0.76, which was the benchmark given from our previous stages, then I would submit the model to the challenge's test set, and document the results.

For the refinement of the Gradient Boosting classifier I've used the Grid Search 10-fold cross validation with a set of combinations from the following hiper-parameters of the model:

- number of estimators, between 100, and 500
- max depth, between 3 and 8
- criterion, all of the available
- min samples split, between 2 and 6
- max features, all of the available choices
- min samples leaf, from 0.6 to 2

The same was done to evaluate the model performance, first I would test it on my test set, then I would submit to the challenge's test set, and document the results. If no satisfactory results were achieved, a new set of hiper-parameters would be chosen.

I did this because of hardware limitations, both trainings would take a really long time with my hardware. The possible results would be better with a longer training time, and a bigger set of possible hiper parameters.

3.4 Model Evaluation and Validation

For the evaluation of the neural network I've used a training and a testing set. After the model achieved a somewhat plausible result I've tested the model classifying the data from the competition's available test set, generated the submission file, submitted and if the result was satisfactory I would then document it and follow with the model refinement if needed.

The best neural network model I've come up with is shown in Figure 7.

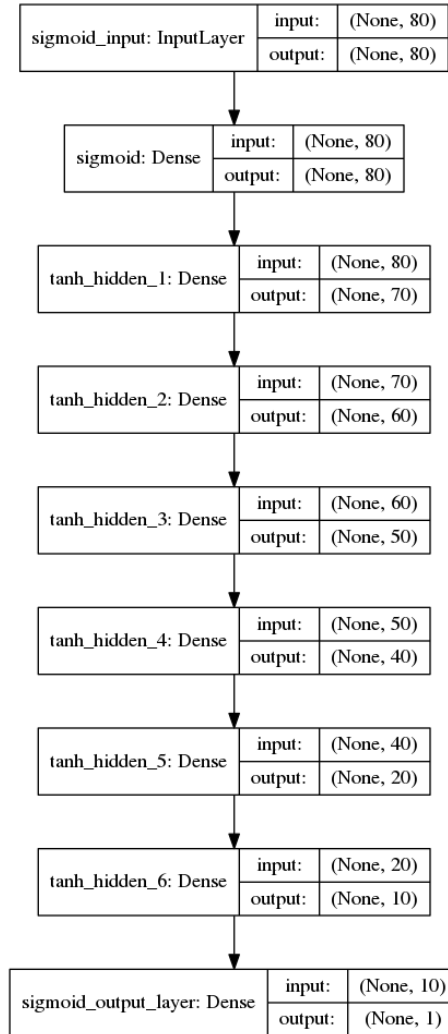


Figure 7: Multi Layer Neural Network Architecture for Credit Scoring

The results of the model are shown in Figure 8, they are not a competitive result for the Credit Scoring Challenge. So I discarded the neural network model, and decided to try the next model on my list (Gradient Boosting).

Your most recent submission				
Name	Submitted	Wait time	Execution time	Score
submission_last.csv	just now	2 seconds	1 seconds	0.837229
Complete				
Jump to your position on the leaderboard				

Figure 8: Multi Layer Neural Network Benchmark

For the Gradient Boosting, I've started submitting a fitted model with the default parameters from the Scikit-Learn, and got the following results shown in the Figure 9.

Your most recent submission				
Name	Submitted	Wait time	Execution time	Score
submission_gb.csv	just now	5 seconds	1 seconds	0.860077
Complete				
Jump to your position on the leaderboard				

Figure 9: Default Gradient Boosting Results

I've followed some of the recommendation hiper-parameters from the competitions forum, and tried a few of my own, and after three days of hiper-parameter tuning, the best classifier achieved as shown in Figure 10, achieved a satisfactory result, but still not so competitive, and bellow the sample submission results.

Your most recent submission				
Name	Submitted	Wait time	Execution time	Score
fs_submission_gb.csv	just now	0 seconds	1 seconds	0.861453
Complete				
Jump to your position on the leaderboard				

Figure 10: Optimized Gradient Boosting Results

4 Conclusion

The current study was about understanding how machine learning works for enterprise credit scoring solutions. The dataset selected was a public Kaggle's challenge credit scoring dataset, with about 300.000 data instances. The complete study compared two different model approaches, one based on Gradient Boosting, and another on Neural networks.

The results obtained were satisfactory for the study purpose but not competitive enough to have a good position on the competition's leaderboard. The Kaggle challenges are great to improve the users machine learning skills, by a competitive, and collaborative way.

The selection of a good feature set, and model determines the best results. There is also some competitors that used both unsupervised learning, and supervised learning techniques, and had good competitive results.

Also, fitting a good model on a big dataset can be time consuming, it took me three days to fit a regular model on the dataset with 80 features, and 150.000 data instances. From the two models I've evaluated, the best model was the Gradient Boosting Classifier.

References

- [1] Wikipedia. Credit Score, https://en.wikipedia.org/wiki/Credit_score
- [2] Louzada Francisco, Ara Anderson, Fernandes B. Guilherme. Classification methods applied to credit scoring: Systematic review and overall comparison. Surveys in Operations Research and Management Science.
- [3] Overview and Applications of Artificial Neural Networks XenonStack <https://medium.com/@xenonstack/overview-of-artificial-neural-networks-and-its-applications-2525c1addff7>
- [4] Kaggle Competition, Give me Some Credit <https://www.kaggle.com/c/GiveMeSomeCredit>
- [5] Alex Rogozhnikov, Gradient Boosting explained http://arogozhnikov.github.io/2016/06/24/gradient_boosting_explained.html
- [6] Sean Owen, Machine Learning: What is an intuitive explanation of AUC? <https://www.quora.com/Machine-Learning-What-is-an-intuitive-explanation-of-AUC>
- [7] jmalicki (username), 3rd in the Give Me Some Credit Competition source code: <https://kaggle2.blob.core.windows.net/forum-message-attachments/1583/model.R>
- [8] GaryMulder (username), 0.870112 on Private Leaderboard, Kaggle Discussion, <https://www.kaggle.com/c/GiveMeSomeCredit/discussion/31514>.