Landis Data Science Report

Ahmed Shoukr

Introduction

In this report, I will discuss the approach I took to create a baseline model and a second iteration on that model to predict the scores of new people, given their reports. My goal here was three-fold. First, to get a better understanding of the data and the underlying dynamics and relationships between different features. Second, to build a baseline model that can predict the score of a given report based on selected / manufactured attributes. Third, to deploy this model in production and allow a customer to upload their report in xml file and receive a predicted score.

Approach

The first thing I had to do with the data is convert it from a relational format to a more structured dataframe set. This required repeating each node for as many times as it had children nodes. All of the code for parsing the train and validation sets is included in parser.py file.

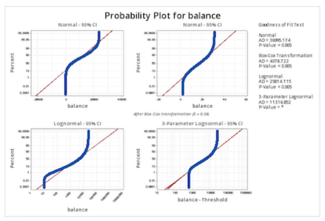
Once I had the data in a structured form, I proceeded by doing very quick exploration of the dataset looking at the following:

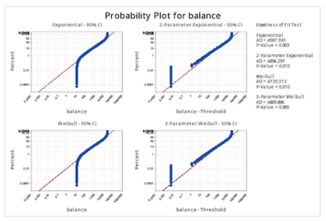
- Distributions of numerical columns
- Outliers
- Imbalance in the data
- Value Counts for categorical variables and what they are
- Highly correlated features (especially to score)
- What features are a must?
- What additional engineered features can add more contextual information and variance between different reports?
- If there is any additional information I can scrape online.

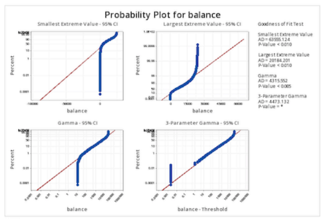
EDA

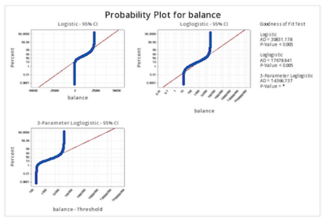
I first started by running the data through PandasProfile package, which allowed me to get a quick intuition about the data: the value counts of each categorical variable such as type, active, status and status of each payment. This would allow me to understand the attributes that are important for the score as well as how sparse the final training matrix may be without doing anything else. I also saw some other quick insights from this.

Further, to find the distributions of the numerical columns: score, high_credit, balance and overdue, I used a statistical software called MiniTab to run hypothesis testing on 14 different probability distributions and 2 transformations at the same time. The software uses the Anderson-Darling statistic (AD) to calculate the p-value. In this case, I was looking for a high p-value to reject the alternative hypothesis for any test. In addition, it also gave probability plots for each distribution. Here is the an example for the balance column:

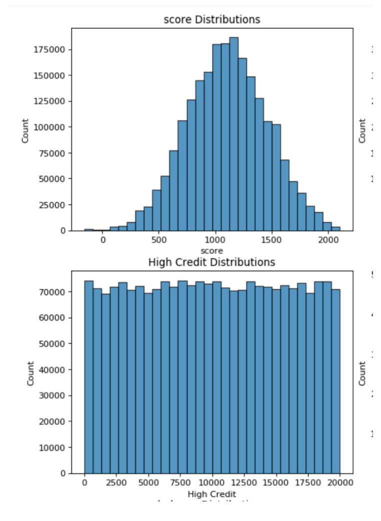


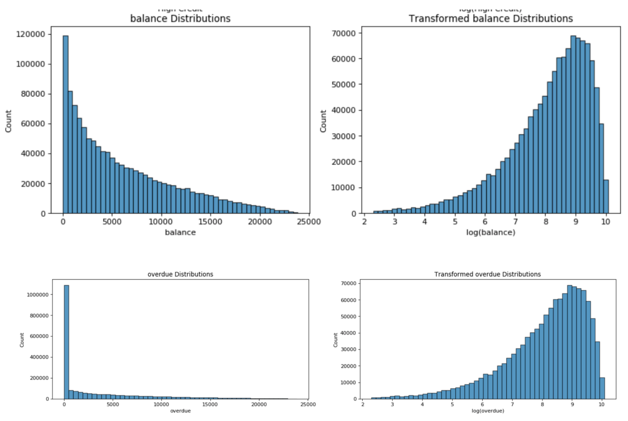






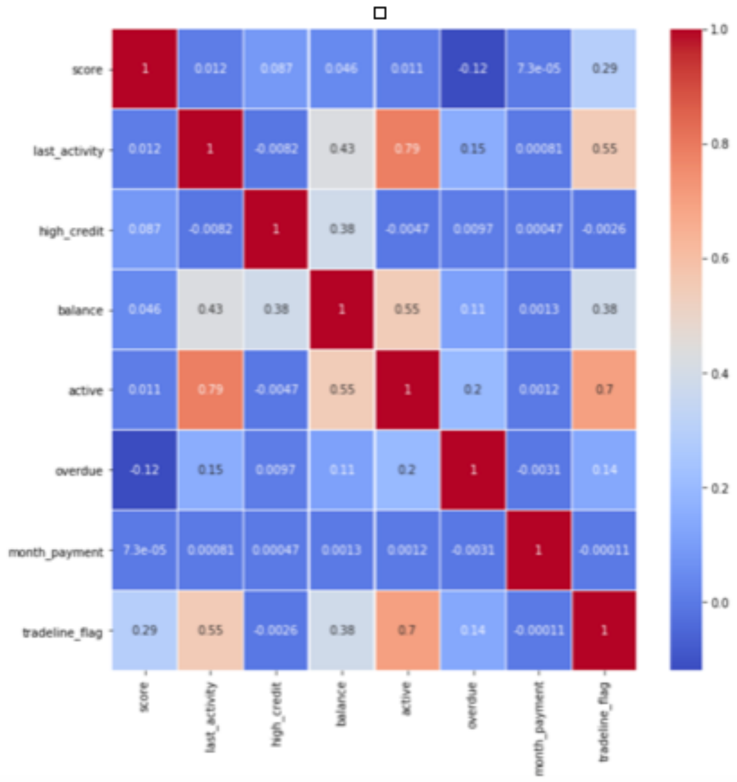
However, this approach was a little bit difficult because of the amount of training data and the software often crashed. I then generated some histogram plots to view the frequencies of each numerical column again.





The score and high credit columns were normally and uniformly distributed. However, because a large number of the balance data (45%) was 0 and most of the overdue data was also 0, it was difficult for me to tell the distribution. I ended up using a normal distribution for the sake of time, although there are better alternatives to this.

Next, I looked for highly correlated features. For this, I have a heat map here that shows the correlation between each of the features.



This heat map used Pearson's r coefficient to calculate the correlation on a scale of -1 to +1, with a more positive value indicating a higher positive correlation, and a more negative value indicating an inverse relationship.

Evidently, the features with the highest relationship to the score were:

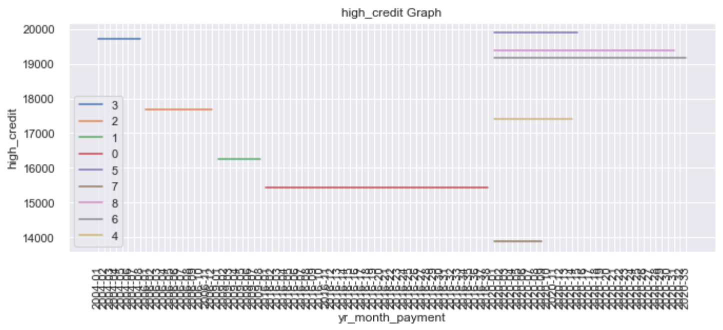
- tradeline_flag (additional feature I created while parsing to view in time-series manner —> each tradeline in a report received a unique tradeline_flag number that was incremented)
  - last_activity and active are also part of this (they show a similar thing and are highly correlated to each other as well with 0.43 coefficient).
  - This also showed me that more engineered features are required here to give more context and break down these features further.
- Overdue is negatively correlated with score with -0.12 coefficient.
- The high_balance and score features will be further explored.

Method 1:

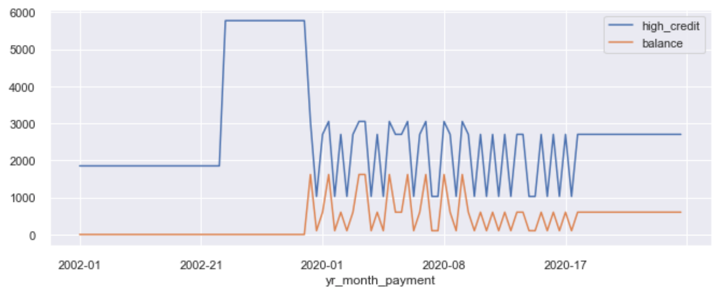
Next, I created a new feature: last_activity_month that turned the data into a time-series-like format with the rows going in order: 2002-1, 2002-2, …, 2020-7, etc. I selected one report with a high average high_credit, another with a medium average high_credit, and a third with a low average high_credit. I explored each of these reports separately to get a further understanding of the relationships between the different features and what features I may need to add.

I started exploring these reports by creating a plot for high_credit in a time series with each tradeline given a different color on the legend.

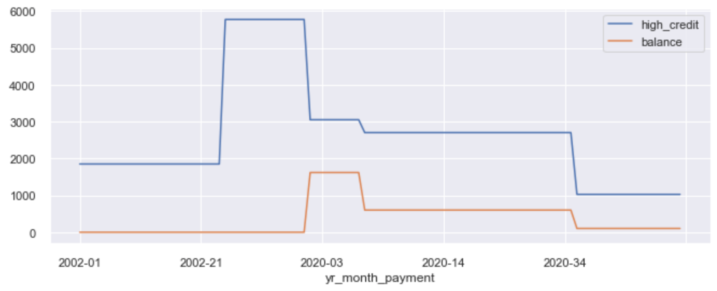
Here is an example of high average high_credit over time.



More information can be plotted here such as the type of tradeline. Additionally, it’s clear that there is an aggregate statistic of the high_score that is feeding the score feature at any given point in time, since there is a large overlap. This is shown below when I plot the given report for high_credit vs. balance without separating the tradelines, so the graph ends up oscillating as different tradelines can have widely varying balance / high_score for a given time period.



Therefore, when the data is sorted by the time-series label last_activity_month_payment for each given tradeline, the plot is much smoother. It is apparent here that there is also a windowing / time delay issue where



the rise in balance does not result in a drop in high_score until after a given period of time.

Method 2:

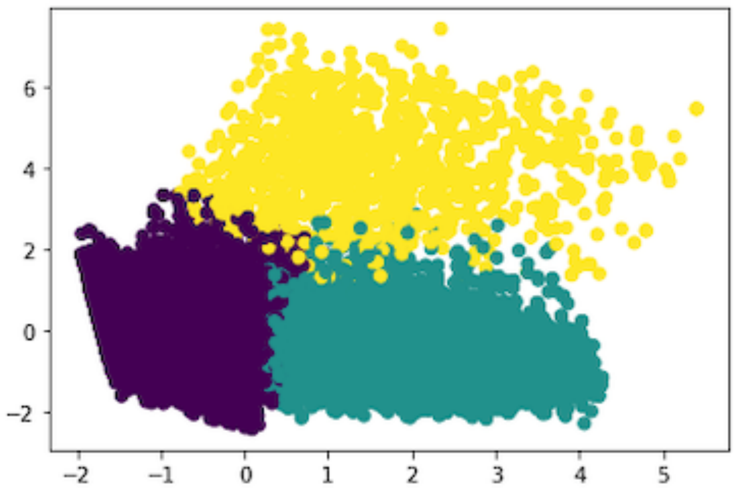
Although this was not part of the baseline model, I tried to run the train data through an unsupervised segmentation model to see if I am able visualize any meaningful clusters from the data. I did this in 2 steps. First, I decomposed the data using Principle Value Decomposition, PCA, while maintaining 95% of the variance. Then I ran it through a kmeans algorithm to try to see if I can get useful segments.

KMeans is an algorithm that starts with a certain number of random points and continuously adds points to the closest cluster centroid to them (which therefore changes the centroid each time).

However, it makes the assumption that the data is spherical with each cluster being of similar size, which may take more effort to accomplish here.

Although I did not use the segments in the baseline or second model, it is something worth exploring further in the future.

Here are some samples of the results in 2& 3 dimensions:



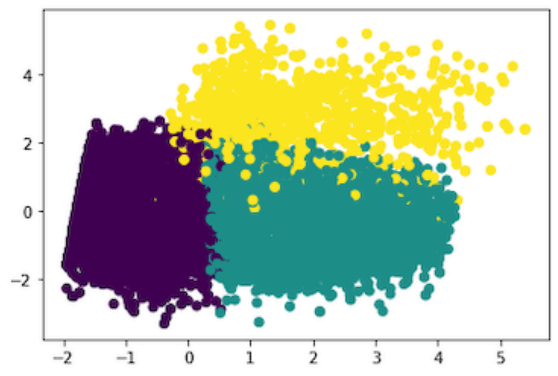
Baseline Model

For the baseline model, I used a XGBoost Algorithm.

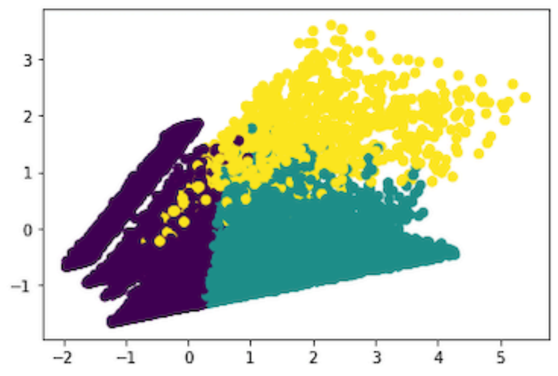
Assumptions:

Since an XGBoost model is a variant of a random forest ensemble technique, it does not make any assumptions about the data, other than that each subsample is representative of the population.

```
0 2
```



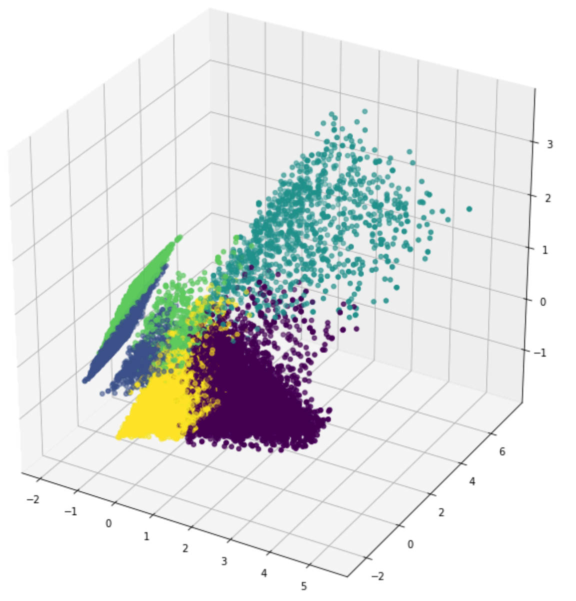
```
0 3
```



Train / Validation Split

Although I did not use this in the final model (as there I trained on the entire dataset), in this case, I split the train data into 2 sets, with the validation set in order to optimize and get a sense of the hyperparameters and hyperparameter ranges that work for this problem.

Hyperparameters:
These are the hyperparameters used in training the baseline model. I also included values for the best trained model after running 20 different training models with different hyperparameters, using bayesian optimization to search for the best model.



Eta: this is effectively the learning rate of the algorithm. - 0.05

Gamma: minimum loss reduction required to make further partition on a leaf node of the tree. This in effect adds some regularization effect to the model. — 7.46

max_depth: maximum depth of the tree. The deeper the tree, the more likely to overfit - 3

min_child_weight: minimum instance weight of a child. if the weight of a leaf is less than this value, the partition will not occur - 6

num_round: the number of rounds for boosting - 200 subsample: percent of data that is sampled in each iteration - 0.52

Deployment

I deployed the XGBoost baseline model using AWS Sagemaker. I deployed it on a server and exposed it through an API endpoint. This endpoint is called by a serverless function in AWS that preprocesses the data it receives and makes an inference on the model, which it then returns to the frontend. I used API Gateway to communicate through the frontend and the serverless function and deployed the frontend index.html file with a php app on heroku. **Here is the link: https://landis-app.herokuapp.com.**

You can paste any report from the validation set here and it will return the predicted score from the model in just a few seconds.

Engineered Features

In order to add more contextual information that can differentiate the reports more and add more variance in the training data, I added the following features to the train set:

Number of
- Late tradelines
- Collections tradelines

- Edu tradelines
- Auto tradelines
- Revolving tradelines

- Late edu tradelines
- Collections edu tradelines

- Late auto tradelines
- Collections auto tradelines

- Late revolving tradelines
- Collections revolving tradelines

- Current non-active tradelines

- Tradelines
- Overdue tradelines

Note: In order to add more context in a time-series sense, I appended these features in increasing order as each tradeline was read.

In addition, I parsed a dictionary from online that allowed me to map each state from the address column to a region and a division in the US. The Regions include: west, south, northeast, midwest and the divisions include mid Atlantic, mountain, new england, etc. (9 divisions total) - this is to help gain more information from the address column.

Final Model

The Final Model used XGBoost as well. I did a feature reduction down to 7 features while maintaining most of the variance. The model was trained on the entire train set.

Next Steps
Some of the next steps to improve this model are:
- Manually check where model breaks down by making an inference on multiple reports with all static features except for 1 (within a range) and see if it makes a difference.
- Augmenting the data: the way that I represented the relational data is only one way of doing this, there may be a way where each report can be represented as a single record.