Capstone Proposal

Stas Sajin

May 28, 2017

Disclaimer

Note that this project should not be used for making investment decisions. This is purely a toy project undertaken for personal development.

Domain Background

The proposed project will focus on using machine learning to optimize the underwriting workflow for a firm specializing in consumer lending. The last 5-7 years have seen a rise in the number of financial start-ups that want to reform the consumer lending industry (see reports for more information here, here, and here). Specifically, the premise of companies like Lending Club, Prosper, SoFi, and LendIT is that they can use technology, data, and machine learning in a more optimized and scalable way, thus leading to a reduction in interest costs to the borrower and increases in return to the investor. The challenge faced by these companies if that they generally have to figure out a way to make the underwriting process less manual and more accurate. For many traditional banks, underwriters (UW) are usually credit officers who review an application. They generally end up making two decisions: first, they decide if a user should get a loan or not. Second, once they decide that the user should get a loan, they have to figure out an appropriate rate. Generally, the underwriting process is slow, inconsistent, and error-prone. Hence, this project will focus on creating a product driven by a machine learning algorithm that helps a hypothetical startup scale its underwriting operations.

Problem Statement

We'll assume that we got hired by a financial startup that deals with Peer-to-Peer Lending. Product managers have specified that they need a product that optimizes the underwriting workflow. Specifically, they want a product centered around a machine learning algorithm that helps underwriters be faster and more accurate at reviewing personal loan applications. They have specified the following problems:

Problem 1: Underwriters often complain that they have too review too many applications that should've been declined automatically due to very bad credit. Hence, they have requested that there be a process put in place that filters users who will get declined for a loan. A solution to this problem is to create an algorithm that automatically classifies the user as an automatic decline (`auto_declined`) or let the user proceed to UW (`proceed_to_uw`). The performance of this algorithm will be assessed on a cross-validated time-series using AUC as the main metric.

Problem 2: Product managers have also requested for a product feature that allows them to input the user application data into an UI and obtain a risk score and an interest rate associated with a particular loan. In other words, rather than calculating a risk score for a user manually or using subjective criteria, they have requested an application interface that does everything for them as long as they input all the user data into the UI. A solution to this problem is to create a Flask application that embeds a model that has the goal of predicting the likely rate associated with a loan and the likely risk bucket score (e.g., 29% interest rate with a risk score of HR, which is the highest risk score).

Problem 3: Legal has requested visibility into what inputs are selected by the models above, how the model performance is being tracked and how models are being updated. The issue is that the model essentially

affects the the actual UW outcome, since the human decision is taken out of the loop. This means that once the models are being built, we can't assess the model performance because the model effectively guides the outcome. For instance, if a model auto-declines an application, we will never know if a user should've been declined or not by a human. Hence, legal and product have requested that the model be implemented in a pipeline that allows for A/B testing models and that allows a portion of auto-declined applications to go through the application process. The solution to this problem is to create a modeling pipeline that tracks all the inputs and outputs, model versions, and also allows for a small portion of the users to go through to under-writing as a way to continuously monitor model performance.

Datasets and Inputs

The the purpose of this project, I will be using Prosper Market Place listing data. This dataset contains all the credit report data, income data and employment data associated with a loan listing. You will need to create an account with Prosper in order to be able to download the dataset. I will be using loan data that is available up to March 31, 2017. I generally, will categorize the inputs into the following:

- Inputs that underwriters should not modify: this consists of data from credit report. This data is pulled from a credit rating agency such as Experian or Transunion, and UW don't have a reason to change that information. There are almost 150 input features associated with credit report data.
- Inputs that underwriters have the ability to modify based on the evidence that is presented by the user. Data from employment income will be one of those inputs.

Unfortunately, there are no rich public datasets that contain data on declined credit applications. This is mainly because availability of this data to the public would threaten the intellectual property and competitive advantage of a company. For instance, having declines data would allow one to reverse engineer the underwriting process of a company. Because we don't have decline data, I will make several modifications to the dataset:

All loans that have a risk score of HR, D, or E will be coded as loan that got declined by underwriters. Loans with risk scores above E will be classified as loans that were approved the underwriters. This modification will effectively create an outcome variable that will be used in the model for automatically declining applications. These models will use all the data that is available in the listing for a loan (over 200 fields).

To summarize, we end up with the following: * Input data such as modifiable and modifiable inputs. * Output data: whether the user got declined or not (for the model that automatically declines users). For the risk score model, the outcome will be actual interest rate associated with a loan.

Solution Statement

The solution will comprise of the following: - Generate a model that auto-automatically declines users who do not qualify for credit. The model needs to achieve the best possible AUC score on the test data. The users who do not qualify for credit will not be shown to underwriters (except for a small portion that will be used for future model training). - A model that identifies the interest rate associated with a loan. The purpose of this model is to come as close as possible to the actual interest rate assigned by the UW. The model performance will be assessed using RMSE. - The model that identifies the interest rate will be embedded into a Flask application that allows the UW to modify or input data into mutable fields. Once they input or verify the data, they can call the model for a interest rate.
- The flask application needs to log model version data, model outcome data, provide easy monitoring through dashboards, and visibility into the main drivers behind a model prediction (if possible). This data will be stored in a local database within the application.

Benchmark Model

About 51% of listings fall into the grades D, E, and HR, which for our purposes will represent users who should be declined. This means that a properly performing model should at least beat this benchmark when it comes to automatically declining users.

The risk score model will be optimized for RMSE. Nonetheless, as part of model evaluation, the model needs to be better than an R^2 of 0, which will represent the model that predicts an average interest rate for all the users.

Evaluation Metrics

- The model for automatically declining lenders involves classification. The model will be evaluated using Receiver Operating Characteristic Curve (ROC curve). The training on the other hand will be done on optimizing for logarithm loss. The purpose of the logarithmic loss function is to make sure that we are penalizing the model when it is too confident in a prediction that ends up not fitting the outcome.
- The model for identifying the interest rate for a loan is in nature a regression task, hence the model will minimize the root mean square error.

Project Design

The project will follow the following steps:

- Create a data pipeline that will load all the listing csv files from the Prosper site into a local database that can be queried by an UW. The db will be documented with a schema indicating the field name and definitions.
- Explore the data for patterns and insights. This will be provided as a separate markdown document that will justify feature selection.
- Create a data preprocessing and transformation pipeline for each of the two models.
- Serialize the models.
- Build a flask API that makes a prediction whenever a UW queries for an application id. In case the application id gets rejected, the UW sees a message indicating that the application was rejected. When the application need review, the UW sees the full application info. The models will be invoked in the following flow:
- When a UW submits a query, the decline model is invoked. Based on the outcome of the decline model, the UW sees the full application or the rejection message.
- When the UW selects GET RATE button for an application, the risk score model is invoked with an interest rate associated with the application.
- The model score, outcome, version, time of prediction is logged into a local db within the application.
- For model training, my first attempts will be to build a pipeline in sklearn and pipe it though a gradient boosted tree implemented in Xgboost or lightGBM. I'm leaning on boosted trees because many features are sparse in nature, have missing data, or data with high skew/kurtosis. Boosted trees handle this type of data pretty well.
- Also, because there are so many features in the credit report data, I plan to use an auto-encoder to perform some feature extraction. The features generated by the auto-encoder will then be provided into the other two models.

- The application should provide a response to the user in 200ms since the moment the model is invoked. Hence, preference will be given to models that perform fairly well when making predictions.