Machine Learning Engineer Nanodegree

Capstone Proposal

Rafael Paulino
January 25th, 2017

Proposal: Porto Seguro's Safe Driver Prediction

Domain Background

"Nothing ruins the thrill of buying a brand new car more quickly than seeing your new insurance bill. The sting's even more painful when you know you're a good driver. It doesn't seem fair that you have to pay so much if you've been cautious on the road for years.

Porto Seguro, one of Brazil's largest auto and homeowner insurance companies, completely agrees. Inaccuracies in car insurance company's claim predictions raise the cost of insurance for good drivers and reduce the price for bad ones."

We cannot say that the the use of machine learning to solve this kind of problem (insurance claim prediction problem) is a new one. For example, Chapados et al (2001) has already studied the problem but, instead of trying to predict if one will or not claim the insurance, it tried to estimate insurance premia (a regression problem). There is also news articles, for instance "How AI And Machine Learning Are Used To Transform The Insurance Industry" published on Forbes on 24th October 2017, showing how the industry is changing with the use of Machine Learning.

Problem Statement

The problem I'll try to solve is simple to explain: to predict the probability that a driver will initiate an auto insurance claim in the next year. It is pretty straight forward to see that we can easily collect data that can help us on this problem: past information about insurance companies customers (personal information, car, place, historical information, etc.) and if this customer has already started or not an auto insurance claim (and when). It is pretty easy, with that information, to create some probabilities like:

- Probability of a woman (or man) to claim auto-insurance;
- Probability of certain age interval to claim it;
- Etc.

Datasets and Inputs

The dataset that will be use can be found at a Kaggle Competition named "Porto Seguro's Safe Driver Prediction". For privacy reasons, Porto Seguro did not shared the meaning of each input feature, but it seems that they are related to the individual, registration, car or calculated information. The "target" feature is given (in the train dataset) and it is simple as 1 when that customer filed claim request and 0 otherwise.

The dataset consists of 57 input fields and 1 target and 595.212 samples. There are some missing values indicated with -1. Below is the list of features:

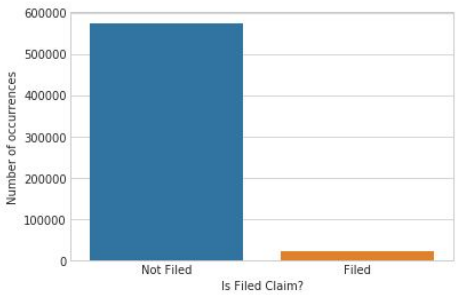
Feature Name	Type	Group
ps_ind_01	continuous / ordinal	ind (individual?)
ps_ind_02_cat	categorical	ind (individual?)
ps_ind_03	continuous / ordinal	ind (individual?)
ps_ind_04_cat	categorical	ind (individual?)
ps_ind_05_cat	categorical	ind (individual?)
ps_ind_06_bin	binary (0 or 1)	ind (individual?)
ps_ind_07_bin	binary (0 or 1)	ind (individual?)
ps_ind_08_bin	binary (0 or 1)	ind (individual?)
ps_ind_09_bin	binary (0 or 1)	ind (individual?)
ps_ind_10_bin	binary (0 or 1)	ind (individual?)
ps_ind_11_bin	binary (0 or 1)	ind (individual?)
ps_ind_12_bin	binary (0 or 1)	ind (individual?)
ps_ind_13_bin	binary (0 or 1)	ind (individual?)
ps_ind_14	continuous / ordinal	ind (individual?)
ps_ind_15	continuous / ordinal	ind (individual?)
ps_ind_16_bin	binary (0 or 1)	ind (individual?)
ps_ind_17_bin	binary (0 or 1)	ind (individual?)
ps_ind_18_bin	binary (0 or 1)	ind (individual?)

ps_reg_01	continuous / ordinal	reg (registration?)
ps_reg_02	continuous / ordinal	reg (registration?)
ps_reg_03	continuous / ordinal	reg (registration?)
ps_car_01_cat	categorical	car
ps_car_02_cat	categorical	car
ps_car_03_cat	categorical	car
ps_car_04_cat	categorical	car
ps_car_05_cat	categorical	car
ps_car_06_cat	categorical	car
ps_car_07_cat	categorical	car
ps_car_08_cat	categorical	car
ps_car_09_cat	categorical	car
ps_car_10_cat	categorical	car
ps_car_11_cat	categorical	car
ps_car_11	continuous / ordinal	car
ps_car_12	continuous / ordinal	car
ps_car_13	continuous / ordinal	car
ps_car_14	continuous / ordinal	car
ps_car_15	continuous / ordinal	car
ps_calc_01	continuous / ordinal	calculated
ps_calc_02	continuous / ordinal	calculated
ps_calc_03	continuous / ordinal	calculated
ps_calc_04	continuous / ordinal	calculated
ps_calc_05	continuous / ordinal	calculated
ps_calc_06	continuous / ordinal	calculated
ps_calc_07	continuous / ordinal	calculated

ps_calc_08	continuous / ordinal	calculated
ps_calc_09	continuous / ordinal	calculated
ps_calc_10	continuous / ordinal	calculated
ps_calc_11	continuous / ordinal	calculated
ps_calc_12	continuous / ordinal	calculated
ps_calc_13	continuous / ordinal	calculated
ps_calc_14	continuous / ordinal	calculated
ps_calc_15_bin	binary (0 or 1)	calculated
ps_calc_16_bin	binary (0 or 1)	calculated
ps_calc_17_bin	binary (0 or 1)	calculated
ps_calc_18_bin	binary (0 or 1)	calculated
ps_calc_19_bin	binary (0 or 1)	calculated
ps_calc_20_bin	binary (0 or 1)	calculated

Another important information about the dataset: it's **unbalanced**. Check out the distribution below:

```
In [7]: ### Target variable exploration
        sns.countplot(train_data.target_name);
        plt.xlabel('Is Filed Claim?');
        plt.ylabel('Number of occurrences');
        plt.show()
```



Source: https://www.kaggle.com/neviadomski/data-exploration-porto-seguro-s-safe-driver (thanks to Sergei Neviadomski)

Or in numbers:

Total rows:	595.212
Claims (target = 1):	21.694
Proportion:	**3.64%**

What does that tell us? The dataset is very very unbalanced. That means that if I create a model that ALWAYS predicts that there will be no insurance claim, I would be 96.36% of time right! And I haven't learned anything about the dataset (except that it is unbalanced). I'll explain how I will deal with that in the last section of this proposal.

Solution Statement

To solve the problem we need to use the information from the dataset and train a machine learning probabilistic classifier. We take out the "target" feature, calling it Y, and the other feature from the dataset could be called X. The model is then trained to find the best coefficients so that for a given X it should predict Y. The way this training is done vary considerably depending on the model/algorithm selected. It is also important to not only rely on one model, but compare to other models and check which get us better performance.

Benchmark Model

To benchmark my model I'll compare it with a simple Naive Bayes classifier. If you want, you can see the code on https://github.com/rtpaulino/mlnd/tree/master/capstone.

To create that model:

- Dropped columns with high percentage of missing values;
- Replaced missing values by the mean (real values) or mode (integer values);
- Encoded categorical data with One Hot Encoding
- Scaled the values to be between 0 and 1 using Min-Max Scaler
- Splitted the dataset between train and test with 0.8 / 0.2 proportion respecting target distribution
- Trained a Gaussian Naive Bayes classifier with train dataset
- Got ROC AUC score using test dataset

This classifier scored 0.58. It is still considered a poor classifier, but it is slightly better than answering always 0, which would lead to a score of 0.5.

Evaluation Metrics

We definitely will not use Accuracy, because it does not work well with unbalanced datasets. Instead I will use "Compute Area Under the Receiver Operating Characteristic Curve" a.k.a. **ROC AUC** to evaluate my model. From a tip from a udacity's reviewer, it is very close to Gini Coefficient, which is used in the kaggle competition. Later I may give Gini Coefficient a try to compare my result with the competition leaderboard.

Project Design

There are some approaches to handle unbalanced data as we can see on the quora thread "In classification, how do you handle an unbalanced training set?" or the post by *Jason Browniee* "8 Tactics to Combat Imbalanced Classes in Your Machine Learning Dataset". There are even books about the topic, for instance "Imbalanced Learning" by Haibo He and Yunqian Ma. But the best post I found was "How to handle Imbalanced Classification Problems in machine learning?" by Upasana.

To solve this problem I'll give **XGBoost** a try. It seems to be a good option because:

1. It handles well imbalanced data;
2. It is 10 times faster than normal Gradient Boosting;
3. One of the most used winner algorithm on Kaggle for structured data problems;

I will now write down the steps to successfully solve the problem:

1. Data Analysis and Pre-processing:
    - Transform categorical data (One Hot Encoding)
    - Normalize the data
    - Treat properly missing data (valued with -1) (*XGBoost can handle it internally*)
2. Data splitting:
    - First of all, I shall split my dataset in training and testing datasets keeping the same unbalance ratio of the target. (*Tip: scikit learn's train_test_split stratify option*) Then I'll work with the training dataset only and leave the testing dataset for evaluation only.
3. Model training:
    - Train model with XGBoost;
4. Model evaluation:
    - Evaluate the model performance over the test dataset and compare with benchmark and previous runs;
5. Use Grid Search to fine tune XGBoost hyperparameters and re-evaluate;

It shall be important to notice if:

- Is there any signs of underfitting? (is my model too simple?)
- Is there any signs of overfitting? (is my model too complex and is not good for new data?)

After reasoning about all that, I may have found a good model to predict insurance claim.