Machine Learning Engineer Nanodegree

Capstone Project

Kalle Bylin
November 26th, 2017

I. Definition

Project Overview

This project is based on Porto Seguro's Safe Driver Prediction competition on Kaggle. Porto Seguro is one of the largest homeowner and auto insurance companies in Brazil.

Context

Most people consider life to be greatly unpredictable. In the middle of it all great effort is dedicated to maximizing utilities and reduce costs. Still, the unpredictability and uncertainty of most events often generate unexpected or unwanted outcomes. These outcomes can be caused by natural disasters, accidents, changing trends, premeditated harm, etc. Insurance is a means of protection against risks and was first used by Chinese and Babylonian traders several millennia ago [1] to spread risks from an individual to larger communities. In other words, the unfavorable event is not necessarily eliminated, but its effects are less critical to the victim because he is part of a community in which everyone is contributing expecting to receive help in case of an unfavorable outcome.

Today insurance companies are the entities focused on creating these communities with pooled resources. This pooling must consider the nature of the risk that is being hedged against. On one side, the cost of being insured has to be low enough for the insured to consider it beneficial. On the other side, the insurance fee must be high enough to cover the losses that may actually happen. Many insurance companies are now taking a data-centric approach to establish these fees. The overview of Kaggle's Porto Seguro Competition states that Porto Seguro has been using machine learning for the past 20 years [2].

Business problem

For the reasons above, Porto Seguro wants to keep improving its pricing strategy. This can be translated into a machine learning problem by creating a model that can predict if a driver insured by Porto Seguro is likely to file a claim or not in the next year. This way, clients with a low probability of filing a claim can receive lower fees, while a client with a high probability of filing a claim might be charged more.

This model will be chosen with empirical evidence by testing different machine learning algorithms and will require a certain degree of experimentation. The data provided by Porto

Seguro will be used to train different models which will be assessed with respect to a given evaluation metric. The model with the highest score on unseen data (the test set) will be presented as a tool for Porto Seguro to better understand its clients, predict claims and improve its insurance fees.

Data

The data for this project is available on https://www.kaggle.com/c/porto-seguro-safe-driver-prediction/data. It contains information for a set of their clients related to the driver, vehicle, region and more, collected during a 1-year period. For each client it also includes a variable stating if the client filed a claim or not during that time period.

Problem Statement

Being able to predict a claim, and the unwanted events, before they happen allows the insurance company to set accurate fees. This particular project focuses on claims made for automobile accidents. Porto Seguro, one of the largest automobile and homeowner insurance companies in Brazil, wants to be able to predict the probability that a driver will file an auto insurance claim in the next year. This prediction is used to set insurance fees.

This is a supervised binary classification problem. Each client is predicted to file a claim or not and by the end of each year, the company can compare the prediction for each client to what really happened and assess the accuracy of the model.

A quite simple but not very useful approach would be to each year calculate the ratio of people out of all the clients that initiated an insurance claim and use this as the probability of any driver initiating a claim. On top of this, machine learning techniques can be used to take into account the importance and value of each feature to generate a more accurate prediction for each individual driver.

We will be using the data provided by Porto Seguro consisting of a training set and a test set. This data was collected during the period of a year and each row is a customer. The columns are anonymized features related to each customer. The training data will be analyzed and studied to understand it better. With this information the data will be preprocessed to make it more suitable for the machine learning methods. This data contains information about the cars, drivers, region, etc. and will at this point be used as input to train different models. After training, each model will be used to generate predictions of validation data and then of the given test set by the Kaggle competition. The model with the highest score for the chosen evaluation metrics can then be used to predict future claims. The score provided by Kaggle will also be included in the analysis.

Metrics

The accuracy could be a first and relevant evaluation metric. This is a supervised classification problem, so it is relatively easy to calculate how many of the drivers were classified correctly. Still, in this context it is not the best metric because, given the nature of the industry, there should be significantly more drivers that don't file claims than drivers that do. A very good

accuracy could be achieved by just classifying every driver as "not filing a claim in the next year".

A better evaluation metric would be the F-beta score that takes into account precision and recall. In this case the F-beta is mentioned instead of simply the F1 score because it can be important for the insurance company to have better recall. In other words, it is important to evaluate out of the drivers that did file a claim how many were classified as such, to better adjust the insurance fees.

It is also important to mention that the competition evaluates submissions using the Normalized Gini Coefficient. The Gini coefficient is best known for and is usually used to assess inequality. But it is also used to, for example, evaluate the predictive power of credit scoring tools [3]. A good credit score is supposed to assign high scores to safe applicants and low scores to riskier applicants. The Gini Coefficient is used to evaluate how well it does that. It is a scale of predictive power from 0 to 1 in which a lower score means less predictive power and is equivalent to a random prediction, like using a coin toss. A Gini Coefficient of 1 indicates perfect predictive power. For credit scores it perfectly identifies who will repay and who will default.

There are several ways to calculate the Gini Coefficient, but a simple way it can be calculated using the ROC curve as the ratio of the area between the ROC curve and the diagonal line, and the area above the ROC curve. It can also be calculated with this formula:

Gini = 2 * AUC – 1

Being AUC the area under the ROC curve. The next image shows a more intuitive graphical representation.

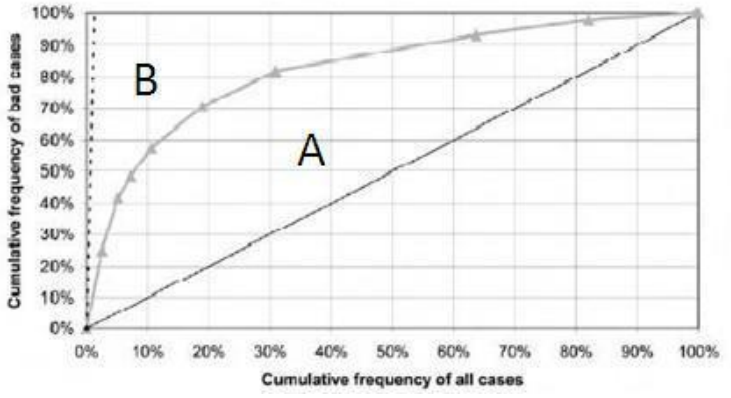

Figure 1. Example of a Lorenz curve showing the cumulative frequency of a class (bad cases) with respect to the whole sample (all cases). Source: https://thealphastrategist.wordpress.com/2014/08/11/an-insiders-guide-to-predictive-modeling-comparison/

Following the calculation stated above it is calculated in this image as:

Gini = A / (A + B)

The Normalized Gini Coefficient adjusts the score by the theoretical maximum so that the maximum score becomes 1.

II. Analysis

Data Exploration

The dataset for this competition has been provided as .csv files by Porto Seguro on Kaggle [4] and has been anonymized to avoid revealing personal information of its clients and to prevent competitors from taking advantage of their data. It has been collected by the company and has already been through a certain degree of cleaning and preprocessing.

```
In [80]: train.head()
```

Out[80]:

id	target	ps_ind_01	ps_ind_02_cat	ps_ind_03	ps_ind_04_cat	ps_ind_05_cat	ps_ind_06_bin	ps_ind_07_bin	ps_ind_08_bin	ps_ind_
7	0	2	2	5	1	0	0	1	0	
9	0	1	1	7	0	0	0	0	1	
13	0	5	4	9	1	0	0	0	1	
16	0	0	1	2	0	0	1	0	0	
17	0	0	2	0	1	0	1	0	0	

5 rows × 58 columns

Table 1. First five rows of the training set provided by Porto Seguros on Kaggle's competition. Due to the amount of features, only the first nine are visible. The dataset contains in total 58 columns and 595.212 rows.

The data has already been split into a training and a test set. The training set has a total of 595.212 rows and 58 columns. Out of those, one column is for the targets indicating if each client filed a claim or not. The target variable for the test set is stored on Kaggle's servers and cannot be used locally. The features have been grouped according to similarity and this grouping can be seen in the names of each feature. "Car" means that the feature is related to the car, "ind" that it is related to the driver or individual, "reg" that it is related to the region and "calc" that it is a calculated feature.

```
In [4]: test.columns
Out[4]: Index(['ps_ind_01', 'ps_ind_02_cat', 'ps_ind_03', 'ps_ind_04_cat',
               'ps_ind_05_cat', 'ps_ind_06_bin', 'ps_ind_07_bin', 'ps_ind_08_bin',
               'ps_ind_09_bin', 'ps_ind_10_bin', 'ps_ind_11_bin', 'ps_ind_12_bin',
               'ps_ind_13_bin', 'ps_ind_14', 'ps_ind_15', 'ps_ind_16_bin',
               'ps_ind_17_bin', 'ps_ind_18_bin', 'ps_reg_01', 'ps_reg_02', 'ps_reg_0
       3',
               'ps_car_01_cat', 'ps_car_02_cat', 'ps_car_03_cat', 'ps_car_04_cat',
               'ps_car_05_cat', 'ps_car_06_cat', 'ps_car_07_cat', 'ps_car_08_cat',
               'ps_car_09_cat', 'ps_car_10_cat', 'ps_car_11_cat', 'ps_car_11',
               'ps_car_12', 'ps_car_13', 'ps_car_14', 'ps_car_15', 'ps_calc_01',
               'ps_calc_02', 'ps_calc_03', 'ps_calc_04', 'ps_calc_05', 'ps_calc_06',
               'ps_calc_07', 'ps_calc_08', 'ps_calc_09', 'ps_calc_10', 'ps_calc_11',
               'ps_calc_12', 'ps_calc_13', 'ps_calc_14', 'ps_calc_15_bin',
               'ps_calc_16_bin', 'ps_calc_17_bin', 'ps_calc_18_bin', 'ps_calc_19_bi
       n',
               'ps_calc_20_bin'],
              dtype='object')
```

Figure 2. All of the names of the columns/features in Porto Seguros training set.

Additionally, each feature name also indicates if it is binary ("bin"), categorical ("cat"), or continuous (the rest). Not knowing exactly what each feature represents is an added challenge. Still, it is relevant and appropriate for this problem because we do know that they describe different aspects of the car, the driver, the population and the geographical region. The

calculated features are also assumed to have been created to add important information. The rules of the competition prohibit using external data. It could be that the calculated fields also already contain data collected from other logical sources.

The 57 features (excluding the target variable from 58 columns) are distributed in the following way:

Type/Group	# Features
Binary	17
Continuous	26
Categorical	14
Individual	18
Car	16
Region	3
Calculated	20

Table 2. Without the target variable, the dataset contains 57 features of different types and groups. This table shows how many belong to each group (ind, car, reg and calc) and how many there are of each type (binary, continuous and categorical)

The dataset was also checked for missing values:

```
In [5]: train.isnull().any().any()
Out[5]: False
```

Figure 3. The Python code used to evaluate if there are any missing values in the dataset.

The check shows that there are no missing values, but the data description offered by Kaggle states that for the whole dataset values of -1 indicate that a feature was missing from the observation.

All the binary variables are either 0 or 1. All the other features vary in their ranges and distributions (which will be shown in the next section). They are all positive with exception from the missing values that are stored as values of -1. Here is an example of the descriptive statistics for the continuous features related to the driver:

```
In [81]: train[feature_dict['ind_cont']].describe()
```

Out[81]:

	ps_ind_01	ps_ind_03	ps_ind_14	ps_ind_15
count	595212.000000	595212.000000	595212.000000	595212.000000
mean	1.900378	4.423318	0.012451	7.299922
std	1.983789	2.699902	0.127545	3.546042
min	0.000000	0.000000	0.000000	0.000000
25%	0.000000	2.000000	0.000000	5.000000
50%	1.000000	4.000000	0.000000	7.000000
75%	3.000000	6.000000	0.000000	10.000000
max	7.000000	11.000000	4.000000	13.000000

Table 3. Descriptive statistics like count, mean, standard deviation, maximum, minimum and percentiles for each of the continuous features in the group related to the driver

Categorical values usually must be processed to be used by the algorithm. In this case, the categorical data has already been processed into numerical values. Providing the features in this format is also a way for the company to keep the data anonymized.

Exploratory Visualization

Visualizations are a powerful tool to gain insights about the data and to understand it better. We can start by visualizing the target variable because it is the variable we want to predict:

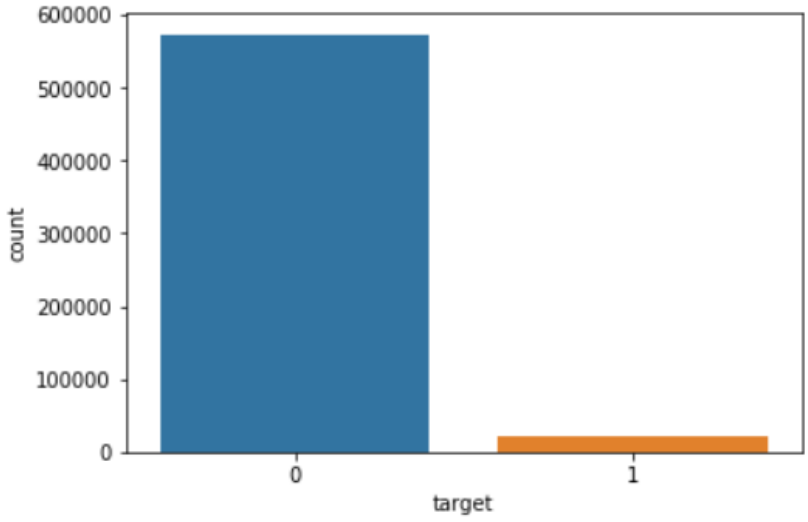

Figure 4. Barplot comparing the count of clients in each of the classes (filed a claim vs didn't file a claim). The dataset is heavily imbalanced with only 3,64% having filed a claim.

There are 21.694 clients who have filed claims, which is equal to 3,64 % of the dataset, and 573.518 clients who have not filed claims, equal to 96,36 % of the dataset. We also notice that the target variable is binary. A value of 0 means that they didn't file a claim during the period the data was collected and a value of 1 means that a claim was filed by a client.

It is also noteworthy that the target data is highly imbalanced, with just under 4% of the clients filing claims. Imbalanced data is a recurring problem, often in classification problems where classes are not represented equally. There are different methods, like sampling, which can be used to work with imbalanced data and will be discussed in more depth in a future section.

Not all the features will be visualized in this report, because of the number of features. A scatter matrix of different features is a good way to have a better understanding of the data and we will start by looking at the features labeled as continuous related to the car (the first column and row is the target variable).

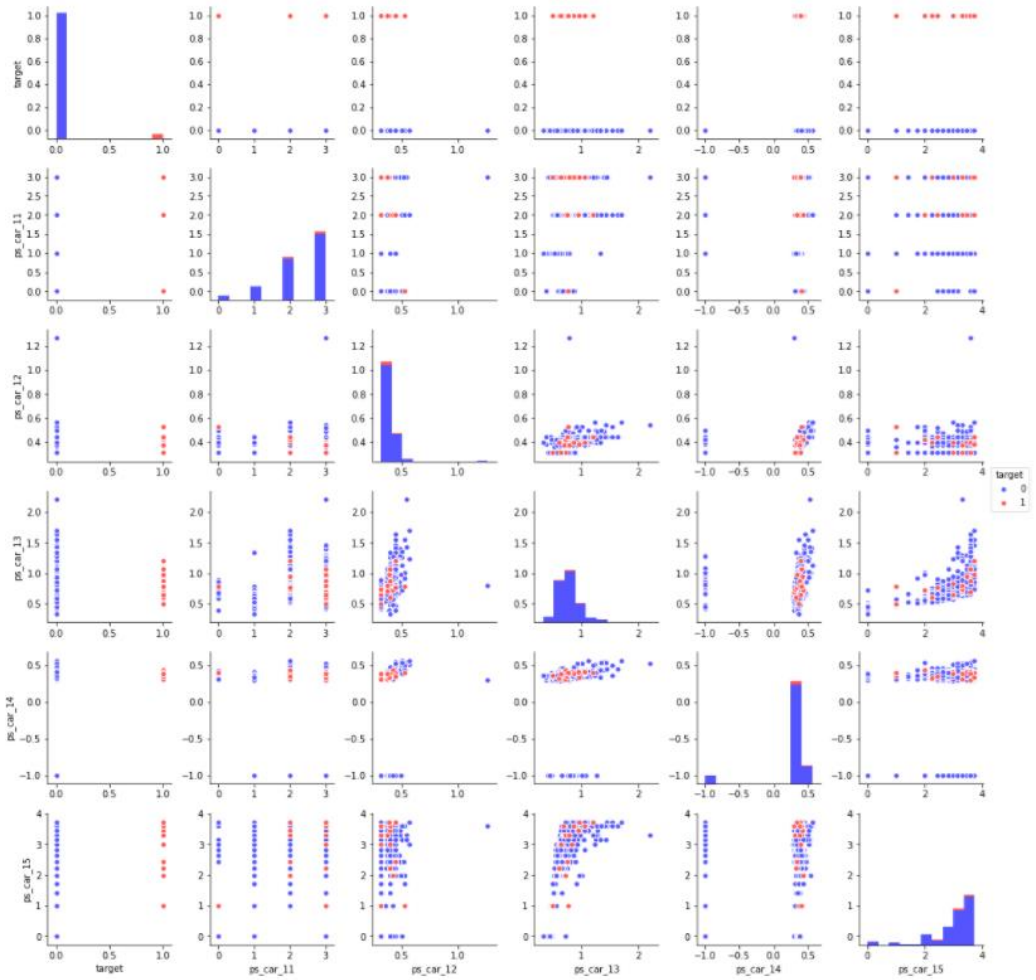

Figure 5. Pairplot showing distributions and scatterplots of the continuous features related to the vehicle.

First, we see that none of the features seem to be normally distributed, most are skewed. Looking at the relationships between the variables, these plots offer some initial information that would still have to be validated. Most of the features seem to be correlated to some degree and, at first glance, drivers that filed a claim seem to cluster around certain values in some plots. For example, higher values of the 'ps_car_15' feature seems visually to have more drivers that file claims than lower values of that feature.

Still, as mentioned this is just something to look into and could simply be caused by the sparsity of the target=1 values. All drivers seem to have higher 'ps_car_15' values and a larger sample

of no-claim drivers would logically also produce more outliers from that class. This would make the graph visually misleading. Here is a close-up of the plot between the 'ps_car_14' and 'ps_car_15' features:

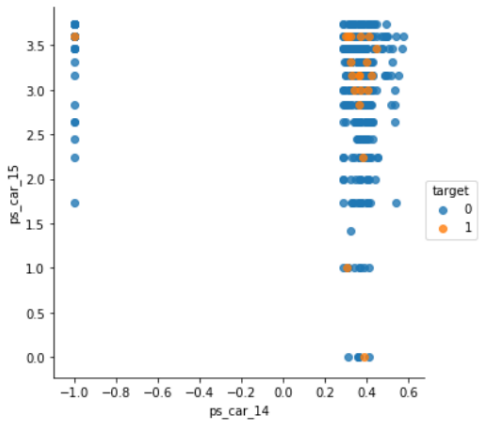

Figure 6. Scatterplot showing relationship between the features 'ps_car_14' and 'ps_car_15'

We can also take a look at the continuous variables related to the driver. Here we see much less correlation between variables and visually it is even more difficult than with the previous plots to see some underlying trend related to the target variable. This is more like a preview to the complexity of predicting an insurance claim.

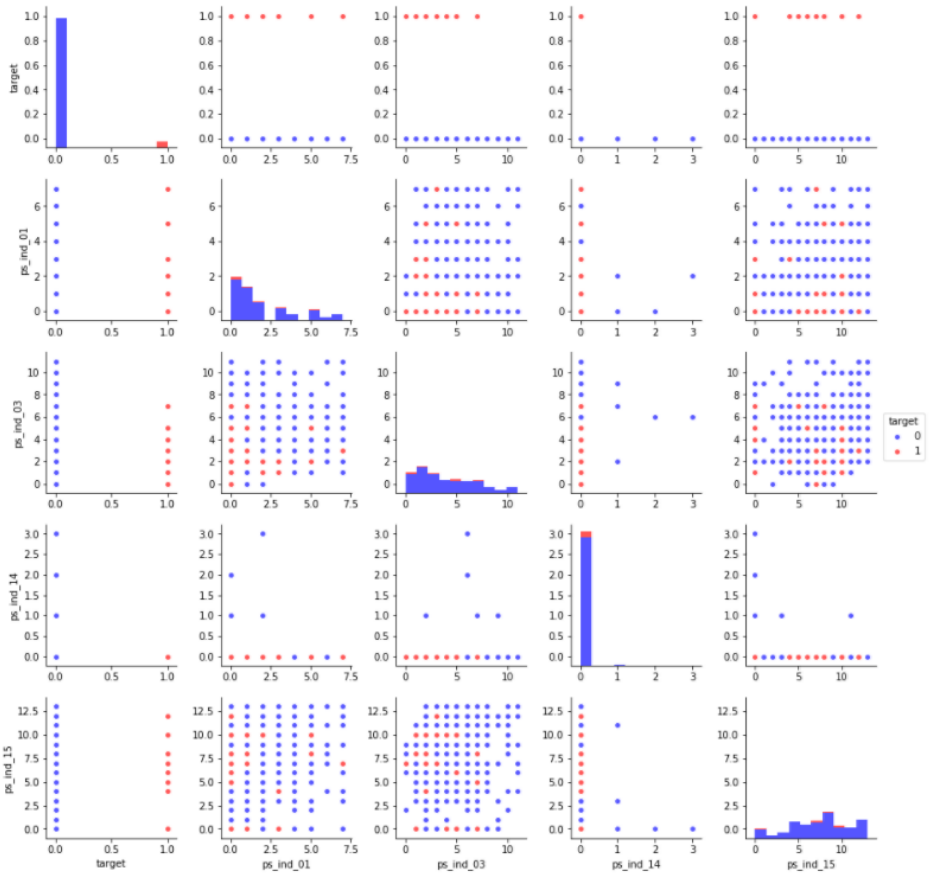

Figure 7. Pairplot showing distributions and scatterplots of the continuous features related to the driver.

A different and useful visualization is a heatmap of correlation between features. Here we can see the correlation between all the continuous variables:

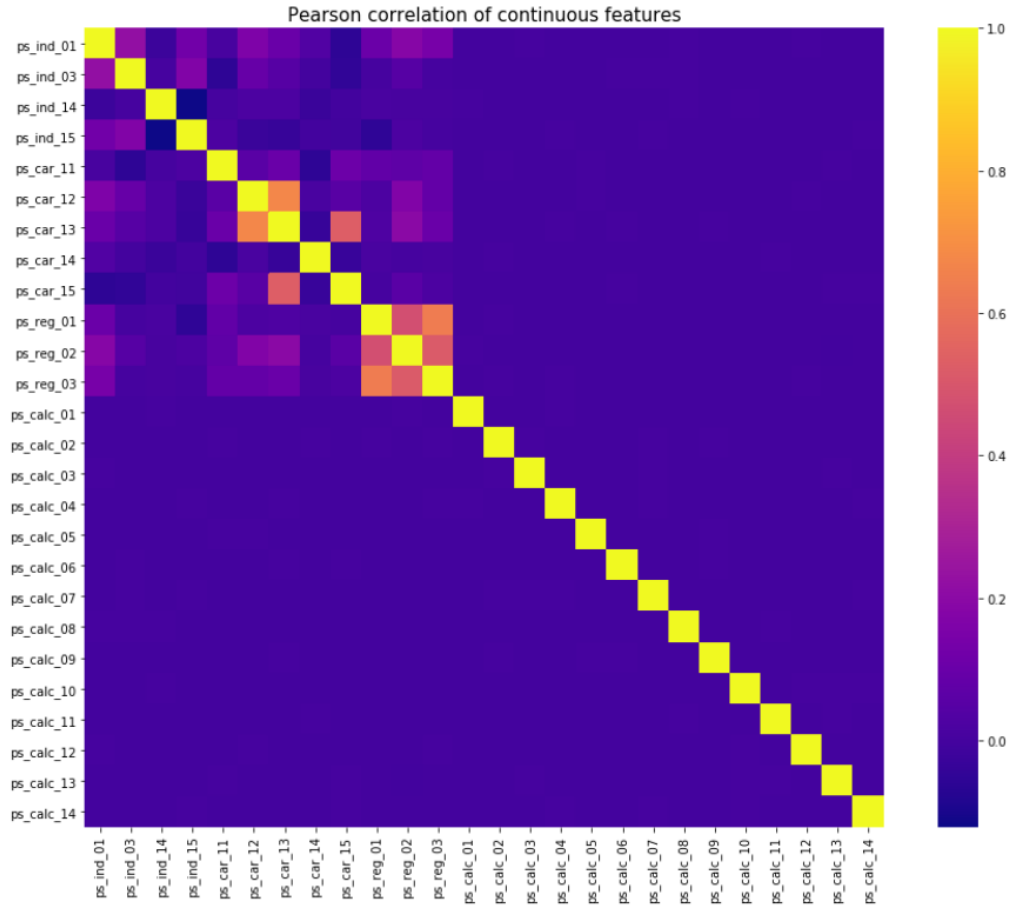

Figure 8. Heatmap showing correlation of the continuous features in the dataset.

Most of the features have very low to no correlation, especially the calculated features. Some features within certain groups do show some degree of correlation with other features in the same group. For example, within the 'car' group the features ps_car_12 and ps_car_13 have a correlation of around 0.7 and the features ps_car_13 and ps_car_15 have a correlation of approximately 0.5. Also, all three continuous features related to region show correlation values between 0.45 to 0.65.

Intuitively it seems normal for similar features to have a certain degree of correlation. For classification purposes a low correlation between features can be interesting because they are all offering different information which can be used for the classifier.

Before moving on, we can also take a look at the heatmap showing correlation between all the binary variables:

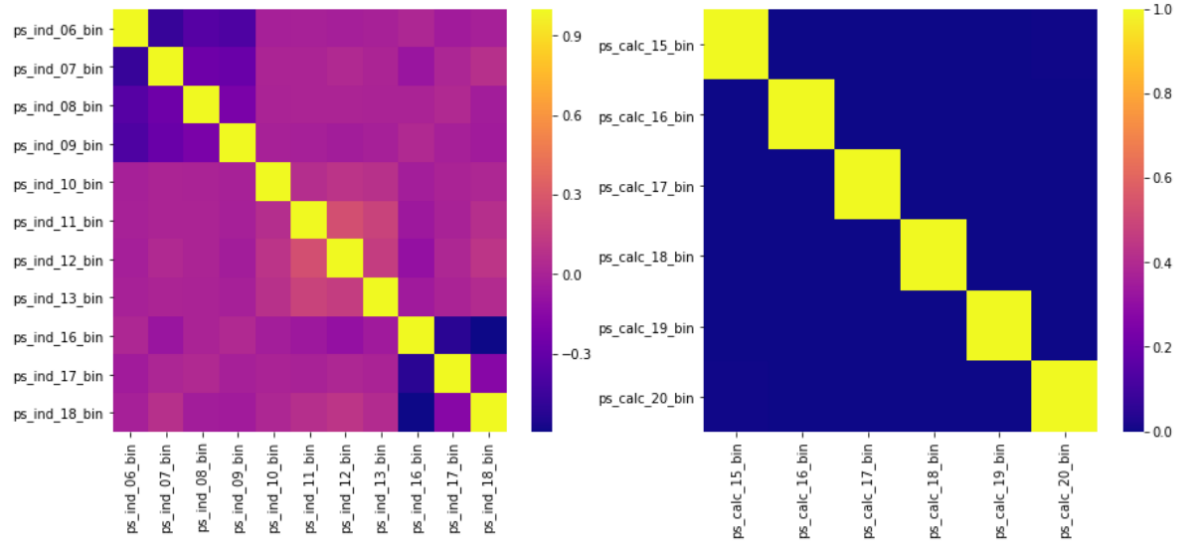

Figure 9. Two heatmaps showing correlation of the binary features in the dataset.

These heatmaps on binary features separated by groups (the 'car' and 'reg' don't have binary features) show the same trend as the heatmap above. The calculated features have practically no correlation. The features related to the driver have a certain degree of correlation but the highest seems to be around 0.5.

Algorithms and Techniques

The two main algorithms that will be used for this project are logistic regression and XGBoost. Each algorithm will be explained and justified below.

Logistic regression

- What is it?

Logistic regression is a classification algorithm often used for predictive analysis and tries to explain the relationship between one dependent binary variable and one or more independent variables. A short explanation posted on Analytics Vidhya is that "it predicts the probability of ocurrence of an event by fitting data to a logit function"[5].

A real-world example for logistic regression could be as a predictor for wether a patient has a disease, like diabetes, or not, based on certain characteristics of the patient like age, body mass index, sex, results tests, etc.

- Strengths of the model

Logistic regression does not make some of the key assumptions of linear regression and general linear models in terms of, for example, linearity, normality, homoscedasticity, and measurement level. It works with relationships that are not linear because it applies a non-linear log transformation to the predicted odds ratio. It is also popular because its results are

relatively easy to interpret. The estimated regression coefficients can be back-transformed off of a log scale to interpret the conditional effects of each feature.

- Weaknesses of the model

Still, as every model, it has its trade-offs and weaknesses. Logistic regression is a classification model for discrete, binary problems. In order to predecit multiple classes a scheme like one-vs-rest (OvR) has to be used. High correlation variables and outliers can make the model perform poorly. It sometimes needs large datasets because maximum likelihood estimates are less powerful than the ordinary least squares used for linear regression.

- Fit for this problem

Logistic regression works well when there is a single decision boundary, which can be beneficial in a problem with binary labels like this one. With a little over 46.694 records in the training example it seems like the dataset is large enough for the model. Some statisticians recommend at least 10-30 cases for each parameter to be estimated

XGBoost

- What is it?

XGBoost stands for "Extreme Gradient Boosting". The term "Gradient Boosting" was proposed by Jerome Friedman in his paper "Greedy Function Approximation: A Gradient Boositng Machine"[6].

Boosting in itself is a sequential technique which works by combining a set of weak learners (models that individually are not strong enough to successfully classify the data) through the principle of ensemble learning. The prediction of each weak learner is combined to create a strong learner.

Gradient Boosted is defined on Wikipedia as "a machine learning technique for regression and classification problems, which produces a prediction model in the form of an ensemble of weak prediction models, typically decision trees."

XGBoost uses the Gradient Boosting concept through a tree ensemble model, which sums the prediction of multiple trees together, and according to the author of the algorithm Tianqi Chen, it "uses a more regularized model formalization to control overfitting, which gives it better performance". The regularization term controls the complexity of the model, which helps to avoid overfitting. The XGBoost documentation states that the regularization is one part most tree packages treat less carefully, or simply ignore.

- Strengths of the model

The XGBoost library is focused on speed and model performance. Szilard Pafka performed a benchmark comparing XGBoost to other implementations of gradient boosting and bagged decision trees in 2015 [7]. His results showed that XGBoost was almost always faster than the

other benchmarked implementations from Python, R, Spark And H20. One of the reasons behind XGBoost's speed is that it implements parallel processing.

In terms of performance, XGBoost has proven itself over and over again in different domains. There is even a list of people that have won or managed a top three position on Kaggle using XGBoost here: https://github.com/dmlc/xgboost/tree/master/demo#machine-learning-challenge-winning-solutions

It is also quite flexible, it allows users to define custom optimization objectives and evaluation criteria.

- Weaknesses of the model

Usually, individual decision tree models can tend to overfit. XGBoost takes care of this first by ensemble, but also with regularization to reduce overfitting.

Given large amounts of data, even XGBoost can take a long time to train. This has led to the development of algorithms like Light GBM which can be several times faster than XGBoost in some cases and can be a better approach when dealing with large datasets.

Other algortishms performa better in other domains. Tianqi, the author of the algorithm, has explained that XGBoost is good for tabular data with a small number of variables. For example, neural nets based deep learning would probably perfrom better for images or data with a large number of variables.

- Fit for this problem

We are working on a structured tabular dataset which is where XGBoost usually shines and the size of the dataset with 595.212 rows (46.694 rows if we use sample_train) and 58 columns also seems reasonable for this algorithm.

Benchmark

The company behind the competition does not want to share their benchmark model or score as they consider this information part of their competitive edge. For that reason, this project will be starting off with a Naïve model as a benchmark in which all the company's clients are treated the same. This would be the scenario in which the company doesn't have a way to evaluate the probability of a client filing a claim. Even after a few years of operation, the company should be able to use historic data in its predictions, but that information is not available either.

This model will be evaluated with the same metric described in the next section which will also be used to evaluate the final model. In other words, predictions will be made using the Naïve model which will be compared to the target variable and evaluated according to the chosen metric. For example, if the metric is accuracy then the Naïve model is going to evaluated according to how many of the predictions are correct with respect to the target variable.

The Naïve model was implemented by assigning the value of 1 (files a claim) to all the clients. It was done this way because assigning a 0 to all values (don't file a claim) would be like assuming that no accidents will happen. The insurance company has to be able to cover the expenses of the unexpected events. So, in a Naïve case in this context it makes more sense to err by the side of caution.

```
TP = float(np.sum(train['target'])) # True positives - Sum of positive values in naive case
TN = 0 # True negatives - No predicted negatives in the naive case
FN = 0 # False Negatives - No predicted negatives in the naive case
FP = train['target'].count() - TP # False positives - Specific to this naive case (Total count - True P
accuracy = TP / train['target'].count()
recall = TP / (TP + FN)
precision = TP / (TP + FP)

# Here we calculate the F-Beta score with a beta of 1.5 to focus on recall, and the Normalized Gini
beta = 1.5
fscore = (1 + beta**2) * (precision * recall) / ((beta**2 * precision) + recall)
n_gini = gini_normalized(train['target'], train['target']*0+1)

# Print the results
print("Naive Predictor: Accuracy score= {:.4f}, F-score= {:.4f}, Normalized Gini= {:.4f}".format(accura
```

```
Naive Predictor: Accuracy score= 0.0364, F-score= 0.1095, Normalized Gini= 0.0006
```

Figure 10. Python code used to calculate the normalized Gini and F-beta score of the Naïve model (all the clients classified as probably filing a claim). This model is used as the benchmark of the project.

As explained above, the Gini Coefficient can be understood as a measure of predictive power, being a Gini Coefficient of 0 the least predictive power equal to random guessing. The Naïve model scores 0.0006 is practically equal to a random guess. This is the benchmark that will be used to compare with the created models.

III. Methodology

Data Preprocessing

As explained earlier, the data has already been through a certain degree of preprocessing to make it easier to use with machine learning algorithms and to keep the data anonymized. For that reason, not much preprocessing was needed for this project. In this section the preprocessing steps will be explained with an overview of the preprocessing that the provider of the dataset probably did and why.

Starting with the preprocessing done by the provider, an important step that has to be mentioned is the anonymization of the features. The columns of the dataset were organized according to the type and group of the data and the feature names reflect this information. We do not know what each feature actually means.

We saw above that 14 of the features are categorical. These could reflect questions like "which is the use of the vehicle?" with "commercial", "personal", "industrial" and more like possible answers. There are also 17 binary features which could be questions like "Are you married?" with a "yes" or "no" answer. To be able to use these features as inputs for machine learning algorithms preprocessing is needed to convert the data into a format that the algorithm understands. The binary features are translated into zeros and ones, while categorical answers are mapped to different numerical values. Dummy variables could also be created but it seems to be that the first approach was chosen for this dataset.

The groups of data present in this dataset are "ind" for features related to the driver, "car" related to the vehicle, "reg" related to the geographical region and "calc" for calculated. We saw above that the calculated features have practically no correlation with other variables. On one hand, these calculated features probably contain information related to groups different to the driver, car or region and that are considered to be important for the task. On the other hand, many of these features might have been calculated from other features. They could be transformations of other features belonging to the existing groups of features ('ind', 'reg', 'car') but were dropped in favor of the transformed versions.

Earlier it was mentioned that the dataset was imbalanced. Imbalanced data is a recurring problem, often in classification problems where classes are not represented equally. Jason Brownlee offers some tips to work with imbalanced data on this link: https://machinelearningmastery.com/tactics-to-combat-imbalanced-classes-in-your-machine-learning-dataset/

It is important to deal with imbalanced data because it might cause misleading results. For example, with less than 4% filing a claim, a very good accuracy could be achieved by just classifying every driver as "not filing a claim in the next year". This is called the *accuracy paradox*.

One of the first tips is to change the **performance metric**. The F-beta Score and Normalized Gini metric were chosen for this project. Another tip is **sampling** the dataset to make the classes more balanced. Two techniques are over-sampling by adding copies of instances from the under-represented class, and under-sampling by deleting instances from the over-represented class.

By under-sampling we created a more balanced subset with 46.46% of the clients filing a claim and 53.54% not filing a claim.

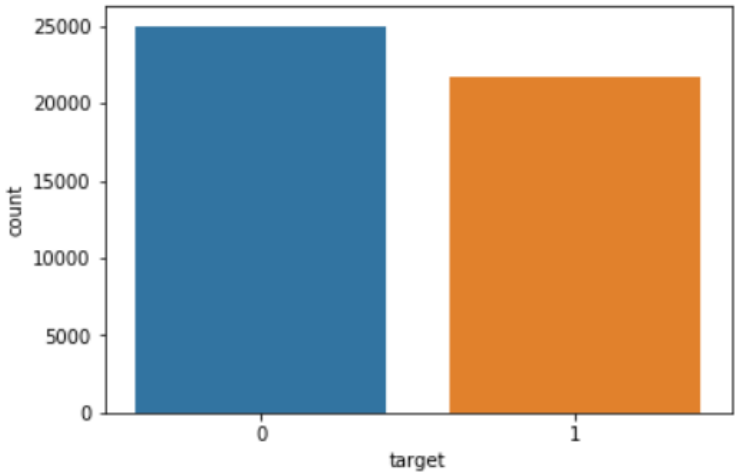

Figure 11. Barplot comparing the count of clients in each of the classes (filed a claim vs didn't file a claim) after sampling. The dataset is much more balanced with 46,46% having filed a claim.

In
the visual exploration of the data we also noticed that the features are not normally distributed. In these cases, it can help to apply non-linear scaling. The Box-Cox test is a way to do this that calculates the best power transformation of the data to reduce skewness.

$$x(\lambda) = \begin{cases} \frac{x^{\lambda}-1}{\lambda} & \lambda \neq 0 \\ log(x) & \lambda = 0 \end{cases}$$

Figure 12. Mathematical representation of the Boxcox transformation

However, the Boxcox transformation above has a log term which will probably cause problems with the "-1" label used for missing values. To apply this method all the values have to be shifted. For categorical data we can use sci-kit learn's LabelEncoder which can be used to normalize labels. Sklearn's MinMaxScaler was also used to scale the features into values between 0 and 1. Scaling is used to keep input values approximately in the same range. This is to avoid attributes in greater numeric ranges from dominating those in smaller numeric ranges.

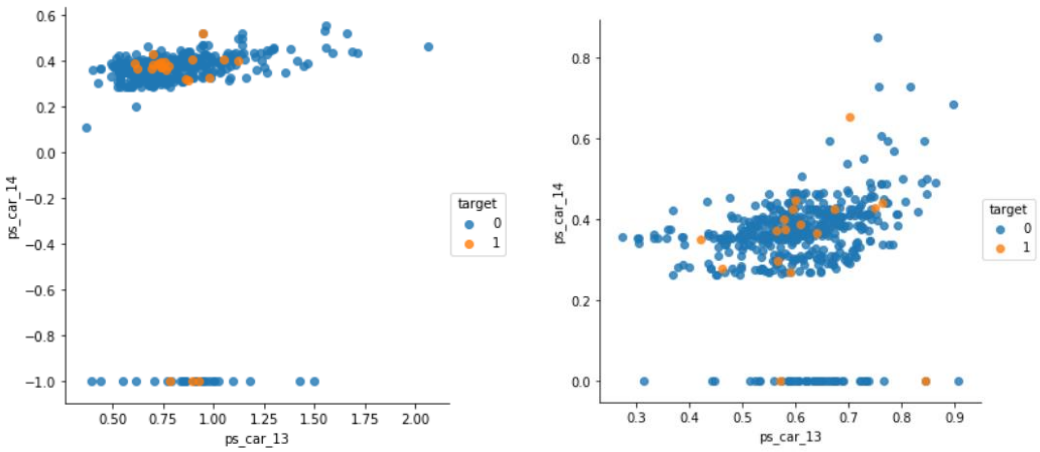

Figure 13. Scatterplots between the features 'ps_car_14' and 'ps_car_15' before and after being through the Boxcox transformation and Minmax scaling. The range is now between 0 and 1 on both axis making it easier to see the relationship between the features (slightly positive).

Implementation

The data for this project is available on https://www.kaggle.com/c/porto-seguro-safe-driver-prediction/data. The first step was to download the data and set up a directory for the project. A virtual environment was created specifically for this project to install the necessary dependencies and libraries. With this set up, the data was loaded into the working environment with the Pandas library. Some initial data exploration was performed to understand it better. We studied the size, shape and content of the data.

After a general look at the data, the target variable was analyzed in more detail to know exactly what it is we want to predict and how to do it. As mentioned above, due to the nature of the industry, accidents are not exceptional events and thus we would expect to see many more drivers that don't file claims than those that do. Imbalanced data can often make it more difficult for models to accurately learn, so at this point we applied a technique like sampling to make the classes more balanced.

In the next step, the features were explored in more depth. Tools like a heatmap or pairplots showing correlation between the features and the distribution of each feature were used. Given that most of the data was skewed and not normally distributed we decided to apply non-linear scaling like the Box-Cox test to transform the data into a form which is more suitable for training. We also scaled the data into values between 0 and 1 to keep input values approximately in the same range.

After preprocessing the data, we started building models that can learn to predict the desired variable. First, we constructed a Naïve model to use as a benchmark. Given that this is a binary classification, we then tried a logistic regression and then a more advanced model like XGBoost was used to compare if there was any improvement in the predictive power of the model. To train the XGBoost model a variation of the evaluation metric was created working as a loss function minimizing the negative of the Normalized Gini function.

To improve the XGBoost model, we also tried applying techniques like grid search with different parameters and cross-validation. This is useful to save time instead of manually trying out different parameters for the model. When tuning the parameters, a lot of code was repeated. To reduce this and simplify the process, functions were created which could be used to reuse the repetitive code, especially for training and tuning. When doing this, one simple problem but that took time to solve was a variable that was not assigned correctly inside the function. This caused the model to output the exact same results every time, even with drastically different parameters.

Once this was finished, the results of the different models were analyzed and the one with the best performance given the chosen evaluation metrics is the model that will be presented to be used to predict future claims by the insurance company. The results will be explained in a future section.

Refinement

The logistic regression model scored on average a normalized gini of 0.249 during cross-validation and 0.242 on the public test set on Kaggle. XGBoost did considerably better with an average of 0.275 locally during cross-validation and 0.266 on Kaggle's test set. For this reason, parameter tuning was focused on the XGBoost model.

This tuning was done following Aarshay Jain's recommendations in his article "Complete Guide to Parameter Tuning in XGBoost" [8]. His recommended approach is to start with a high learning rate, move on to tuning tree-specific parameters (like max_depth, min_child_depth, gamma, subsample and colsample_bytree) and then tune the regularization parameters (like lambda and alpha) which are used to reduce the model's complexity and enhance performance. After this is done we try reducing the learning rate and add more trees to the model.

Tuning was done using grid search iteratively. An estimator model, a parameter dictionary and a normalized gini scorer function (created with sklearn's "make_scorer") were used as input for sklearn's GridSearchCV. Iterations were done satarting with broader ranges and intervals for each parameter and repeating with narrower ranges around the chosen parameter from the broader iterations.

Parameter	Range	Typical values	Chosen value
'max_depth'	1-10	3-10	4
'min_child_weight'	1-10	1	5
'gamma'	0-10	0	0.4
'subsample'	0-1	0,5-1	0.65
'colsample_bytree'	0-1	0,5-1	0.65

'reg_alpha'	0-15	0	10
'reg_lambda'	0-15	1	10

Table 4. The first column shows the different parameters that were fine-tuned to improve the model. The second column indicates which ranges were tried for each parameter and the third column shows typical values that are normally used for each parameter as reference. The last column shows the chosen parameters for the final model.

With these parameters the model did even better moving from an average of 0.275 locally during cross-validation and 0.266 on Kaggle's test set, to 0.2778 locally on average with cross-validation and 0.270 on Kaggle.

IV. Results

Model Evaluation and Validation

The final chosen model is the tuned XGBoost model. The Naïve model had a normalized Gini score of almost 0 and a F-beta score of 0,1095 (beta=1,5 for more focus on recall). The final model did better than logistic regression and XGBoost model without tuning. It managed a normalized Gini of 0,2778 and a F-beta score of 0,6938.

Model	Normalized Gini	F-beta Score
Naïve	0,0006	0,1095
Logistic	0,2495	0,4729
XGBoost	0,2752	0,5033
Tuned XGB	0,2778	0,6938

Table 5. The first column indicates the name of the model that was trained. The second and third columns show the normalized Gini and F-beta scores obtained during cross-validation.

The XGBoost algorithm was chosen because we are working on a structured tabular dataset which is where XGBoost usually shines. Additionally, it is an algorithm that has proven itself in terms of speed and performance, as we saw earlier. However, after a first round of training right out of the box, the algorithm was tuned and several combinations of parameters were tried. The final model has the following parameters:

```
tuned_model = XGBClassifier(n_estimators=1000, max_depth=4,
                        objective="binary:logistic", learning_rate=0.05,
                        gamma=0.4, scale_pos_weight=1.5,
                        min_child_weight=5, colsample_bytree=0.65,
                        subsample=0.65, reg_alpha=10, reg_lambda=10)
```

Figure 14. Screenshot of the chosen parameters for the tuned XGBoost model.

The local average score was already reported to be 0,2778 during cross-validation. In order to test if the model generalizes well to unseen data and different inputs, predictions were generated for the test set stored on Kaggle's servers. The normalized Gini score on that data set was 0,270.

Figure 15. Screenshot of the score obtained by the tuned XGBoost model on Kaggle.

This score is close to the solution expectations. At a first glance, it might seem as a low score given that the maximum normalized Gini score is 1. Still, this has to be understood in the context of the problem and other proposed solutions for the problem. At this moment the top three scores for the competition range from 0,289 to 0,291. This means that there are ways to improve the model, but it is still relatively close to some of the best models.

#	△1w	Team Name	Score
1	—	**Michael Jahrer**	0.291
2	—	**Not So Dumb Any More**	0.290
3	—	**MSChuan-ironbar**	0.289

Figure 16. Screenshot of the top three scores in the Porto Seguros competition on Kaggle at the time of writing.

Justification

As can be seen in the following graph, the chosen model obtain a better score on both the normalized Gini and the F-beta evaluation metrics. Additionally, the results are significantly better than the benchmark model (Naïve).

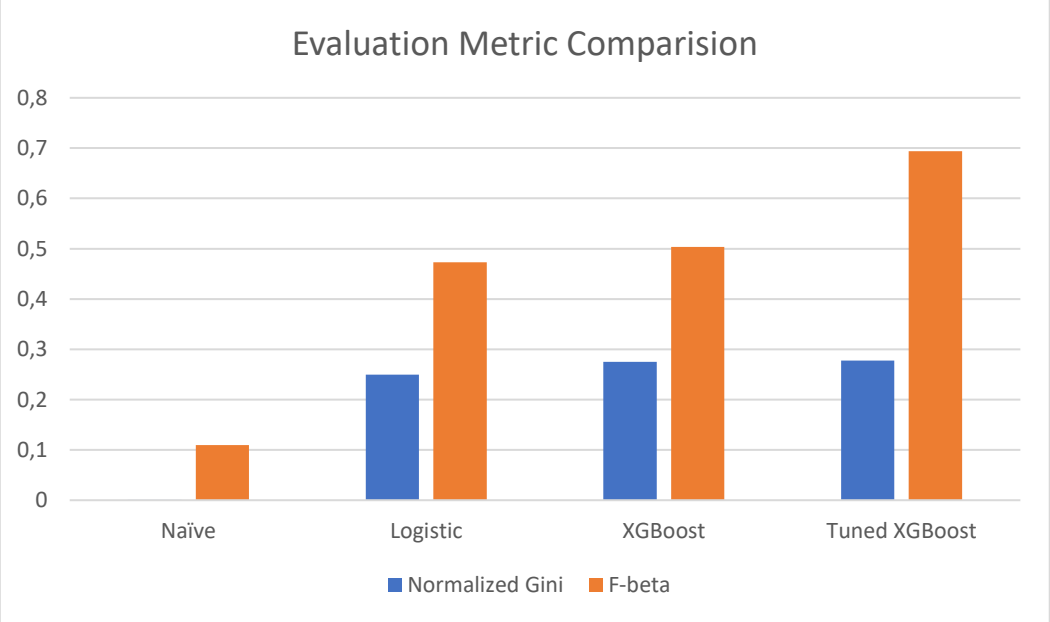

Figure 17. Barplot comparing the normalized Gini and F-beta scores obtained by each of the models used for this project.

Even though we have used the normalized Gini as our principal evaluation metric for this project. The F-beta score played a very important role when choosing the final model. As seen in the graph, the logistic regression algorithm already obtained a great improvement with

respect to the normalized Gini of the benchmark model. From there on, improvements on the normalized Gini were not very large. During parameter tuning there was even a model that performed slightly better on the normalized Gini than the chosen model.

It is worth remembering that the F-beta score takes into account both precision and recall. Precision would help us evaluate of how many of the samples classified as filing a claim actually were filed a claim, while recall calculates how many of those that actually filed a claim were classified as such. In this case the F-beta is used with a beta of 1.5 because it can be important for the insurance company to have better recall. In other words, it is important to evaluate out of the drivers that did file a claim how many were correctly classified as such to better adjust the insurance fees.

Coming back to the chosen model, most of the variations were showing very small improvements in terms of normalized Gini. The chosen model did improve with respect to the other models, but at the same time it achieved a F-beta score of almost 0,7 and was therefore chosen. This difference allowed the model to make a jump from 0.266 to 0.270 on Kaggle's public test set.

A last though on the chosen model and even the top models on Kaggle at this model is that they are all very far from having perfect predictive power. This shows the complexity of the problem at hand. Car accidents are very exceptional events for most people and even though there are factors that make it more probable for someone to end up filing a claim (like a careless driver), it is still very dependent on chance. Because of this, we are still a long way from actually being to be able to predict if someone is going to file a claim in the coming year. But this model does serve the principal objective of this project which is to adjust insurance fees to better reflect the probability of a client filing a claim. It is far from perfect, but it is a tool the company can use to better adjust its pricing strategies.

V. Conclusion

Free-Form Visualization

With the trained model it is often insightful to plot the importance of the weights for the model.

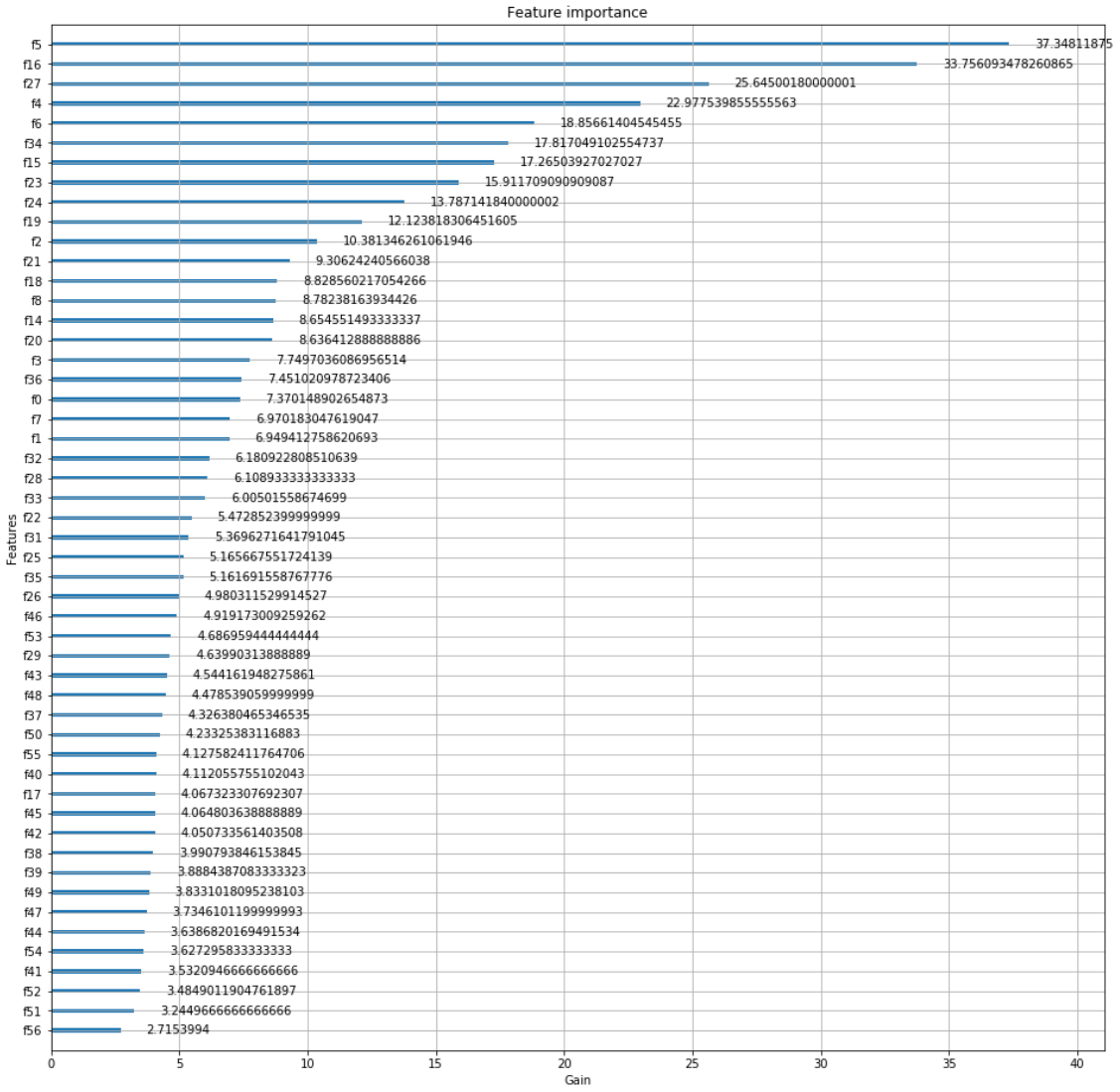

Figure 18. Plot showing the relative importance of each feature for making predictions by the tuned XGBoost model

The features themselves are anonymized but we do know what groups they belong to. The vertical axis of the plot doesn't show the original feature names. When training the XGBoost algorithm, they were automatically renamed according to their index in the input array.

Manually mapping these indices back to their corresponding feature names shows us that the top five in terms of importance are:

1. 'ps_ind_06_bin'
2. 'ps_ind_17_bin'

3. 'ps_car_07_cat'
4. 'ps_ind_05_cat'
5. 'ps_ind_07_bin'

Curiously, out of the 5 most important features for our trained model, 4 are features that are related to the client as an individual. These could be questions like "have you been in a car accident before?", the gender of the person, the age, etc. We don't know exactly, but even though weather of a region or car model are are probably related to accidents, it does make sense for certain individual characteristics to be highly related to the probability of the client filing a claim or not.

Reflection

The whole project started by identifying the problem to be solved, which in this case is adjustment of insurance fees according to the probability of a client filing a claim or not. With this at hand it had to be translated into a machine learning problem. The company behind the challenge has available data on its clients labeled if they filed a claim or not. For this reason, the problem was translated into a binary supervised learning problem.

Having the machine learning problem, we need the data to train on. The data for this project was given by the company Porto Seguros and is available on https://www.kaggle.com/c/porto-seguro-safe-driver-prediction/data. After downloading the data and setting up a directory for the project, a virtual environment was created specifically for this project to install the necessary dependencies and libraries. The data was loaded into the working environment with the Pandas library and before jumping into the training we did some exploratory data analysis. We not only studied important things like the size, shape and content of the data. We also had a look at the target variable and we analyzed in more detail the different available features in the dataset.

One of the most challenging moments of the project was exploring and analyzing anonymized data. Datasets often have the features labeled according to what they contain. This information makes it easier to study the data, make deductions and transform the data into new features.

The categorical and binary features were already preprocessed, but during this initial exploration, the data was found to be imbalanced data, and the features could be transformed and scaled into a format that facilitates learning. A technique like sampling was used to make the data more balanced. Given that most of the data was skewed and not normally distributed we also decided to apply non-linear scaling like the Box-Cox test to transform the data and scaled the continuous features into values between 0 and 1 to keep input values approximately in the same range.

After preprocessing the data, different models were chosen and trained to predict the desired variable. A Naïve model was used as a benchmark and for training logistic regression and XGBoost algorithms were used for training. It was interesting to study the different types of available algorithms for binary supervised classification problems. This was one of the most insightful moments of the project.

Achieving significant improvements on the models was also challenging. Grid search and cross-validation were very useful and helpful tools to save time instead of manually trying out different parameters for the model.

Finally, the results of the different models were analyzed and the one with the best performance in terms of the normalized Gini and F-beta score combined was chosen as the final model. As mentioned above, the model performed well compared to the expectations. It is important to mention though that this particular model should not be used in a general setting. It was constructed specifically on Porto Seguros' data. Other companies might have similar data on their clients, but variations and different ways of representing the data will probably be used. Additionally, other companies might be based in other countries or might focus on other types of insurance. This would also have to be taken into account. This means that most part of the workflow can probably be used to train similar models, but not this final model and solution.

Improvement

There are different ways to improve the models. First of all, more time and resources spent on parameter tuning could have helped to find a better configuration of the parameters. Because of the computational resources available, parameter tuning was done in several iterations using one or two parameters at a time. Tuning more parameters at a time implies many more candidates and longer tuning times. With better resources the model can be tuned with more parameters at a time, which could lead to a better combination.

Other algorithms and models might also be used which could possibly perform better on this particular problem and dataset. On top of this, the ensemble principle can be taken a step further. XGBoost uses several decision trees to make its prediction. For example, stacking is a model ensembling technique used to combine information from multiple predictive models to generate a new model. The stacked model often outperforms each of the individual models. Instead of choosing one of several XGBoost models we can stack them together for them to work together.

Finally, it would also be useful to be able to understand the data better. In this case it would probably require to either be an employee of the company or to sign a confidentiality agreement. This way it would be possible to explore the data without it being anonymized. Industry-specific knowledge and external data could be used to improve the model.

Sources:

1. https://en.wikipedia.org/wiki/History_of_insurance
2. https://www.kaggle.com/c/porto-seguro-safe-driver-prediction
3. https://www.eflglobal.com/every-lender-has-a-gini-coefficient-so-what-exactly-is-it/
4. https://www.kaggle.com/c/porto-seguro-safe-driver-prediction/data
5. https://www.analyticsvidhya.com/blog/2015/11/beginners-guide-on-logistic-regression-in-r/
6. https://statweb.stanford.edu/~jhf/ftp/trebst.pdf
7. http://datascience.la/benchmarking-random-forest-implementations/
8. https://www.analyticsvidhya.com/blog/2016/03/complete-guide-parameter-tuning-xgboost-with-codes-python/