Machine Learning - Final Project Report

Arin Ghosh aring@sfu.ca
Garrett Kryt gkryt@sfu.ca
Gustavo Fehlberg gfelhber@sfu.ca

1 Introduction

The objective of this report is to present the problem, approach and results for the final project as part of the coursework for CMPT 726 - Machine Learning administered in Fall 2017 at Simon Fraser University. The topic to be covered in this project were free to choose by the participants, and we therefore decided to tackle a problem that was already well defined with the data provided. The chosen problem for this project was a Kaggle competition hosted by Porto Seguro, one of Brazil's largest insurance companies. The goal of this competition was to predict whether or not auto insurance holders were likely to make an insurance claim based on the insurance company's data on that driver. This problem presented a unique "real-world" machine learning problem, complete with a huge unbalanced data set, missing data, and a test data set with no target values.

2 Approach

We wanted to follow an approach that we wouldn't have to dedicate too much time to extracting the data to focus mostly on the data preprocessing and construction of the models. Therefore we decided to join an active Kaggle competition since it provides a structured dataset with clear objectives. An active competition added the additional project components of facing deadlines and restrictions in the number of daily submissions.

Another decision we took was related to the programming language we were going to adopt. Since each one of us has a broader experience in Python or R or Matlab, we decided that we would implement some algorithms in the language we feel more comfortable and compare and ensemble the results.

2.1 Kaggle Competition: Porto Seguros Safe Driver Prediction

The competition can be accessed at `https://www.kaggle.com/c/porto-seguro-safe-driver-prediction`. Its objective is to build a model that predicts the probability that a driver will initiate an auto insurance claim in the next year. The competition officially ended in November 22, 2017.

3 Experiments

3.1 Data Overview

The data was pre-split by Porto Seguro into training and test sets. Each data point corresponded to an auto insurance policy holder with a corresponding set of variables. The training test set included a binary target variable which was used to train the model. The test data did not include a target variable with which to calculate the test set accuracy. Instead, the competition required that we run the test data through our model to predict the probability that an insurance holder files a claim. The problem was therefore a binary classification problem for training the model, but required the final output to be a probability corresponding to each policy holder. The test data was scored in the kaggle competition by computing the Gini coefficient for the submitted results.

The actual predictor variables included names such as ”ind” or ”car” that grouped them in terms of similarity. In addition, the predictors included postfixes such as ”bin” or ”cat” that indicated the type of predictor. Missing data values were coded with a -1 in both data sets. No real description of the different predictors was given, which theoretically could help eliminate personal bias in predictor selection. However, the lack of information could also mean that low-importance features were unavoidably added to the model.

3.2 Preprocessing

Initial attempts at preprocessing included the elmination of variables with a large number of missing values, as well as standardization of continuous variables and the creation of dummy variables for the categorical variables. Initial results with this processed dataset were very poor, with the massive class imbalance in the training set creating a highly imbalanced model. The raw training set had 595213 samples, with 573519 of class 0 and 21694 of class 1. Further preprocessing steps were needed to create a a more robust model that accommodated for this weighting

The resulting preprocessing steps used by the team were derived from an analysis posted in the kaggle competition discussion by Bert Carremans `https://www.kaggle.com/bertcarremans/data-preparation-exploration`. The posted python code provided insight into areas of further investigation by the team into preprocessing the data.

The data preprocessing began with an initial assessment of the data in it’s raw format. The data set included continuous variables with decimals and only integers, categorical variables, and a binary target variable. The first step in data preprocessing was to assess the predictor variables and eliminate variables with too many missing values. The correlation was assessed for all continuous variables using a heat map and the result is shown in Figure 1. There is good correlation between six of the continuous variables, but in this case we decided not to act on processing the data based on this correlation and instead left all the continuous variables in the model.

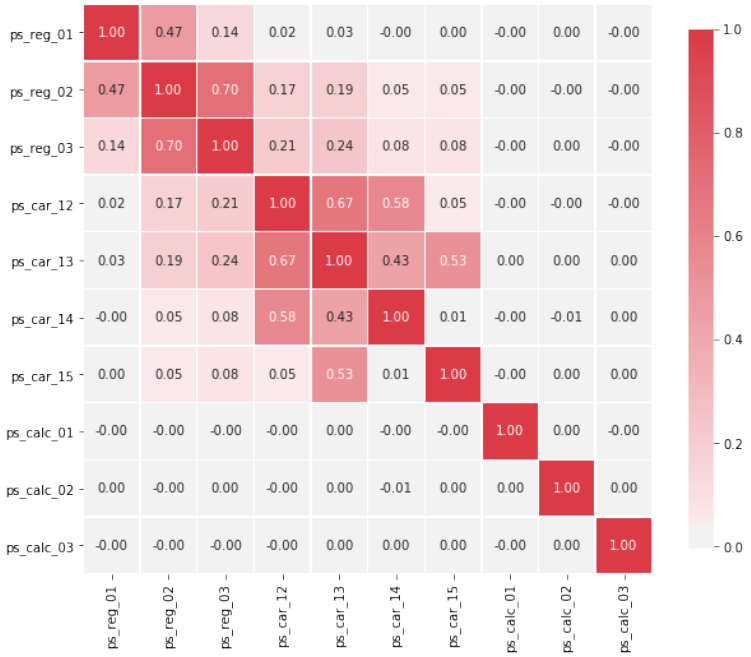

Figure 1: Correlation Plot for Continuous Data [1]

Due to the large class imbalance present in the data, undersampling was employed to reduce the number of targets with class 0 in the training set. The ratio of class 1’s to 0’s in the raw training data was 0.0364. An undersampling rate was calculated based on adjusting this ratio to 0.1. Training sampled with target 0 were randomly selected based on this undersampling rate and resulted in reducing the number of targets of class 0 to 195246.

The remaining continuous data was then normalized using the python function StandardScaler(). Continuous variables with only a few missing values were imputed with the mean/mode replacing the missing values. Interaction terms were then created for all variables. Finally, a variance threshold of 0.01 was set to eliminate variables with less of a potential impact and improve computation time. the resulting data set from this preprocessing included 135 predictor variables, mainly made up of interaction terms. All algorithms were tested on this data set to maintain consistency among results.

3.3 XGBoost

Gradient boosting model is on the rise in competitive data science competitions. The internet is full of examples where the top entries used nothing but boosting algorithms. The advantages with XGB is that it uses a regularized model and therefore yields better results especially for structured data such as the one we are using here. In general XGB is fast, reliable and efficient algorithm which works almost perfectly out of the box.

We used R's 'xgboost' library along with 'data.table' for dataframes. The table below presents the hyperparameters we utilized to get the best model.

Hyperparameter	Default Value	Parameter Value used in the model
nrounds	2	630
objective	reg:linear	binary:logistic
eta	1	1
gamma	1	1
max_depth	6	7
subsample	0.5	0.8
colsample_bytree	0.1	0.8
min_child_weight	5	1
base_score	0.5	median(train_data)

In order to interpret how well the model worked, we can take a look into the xgb.importance functions. In our model, the model importance can be shown using the following commands:

$importance < -xgb.importance(\texttt{feature\_names} = colnames(dtrain), model = \texttt{gb\_dt})$

$head(importance)$

```
> head(importance)
                Feature       Gain      Cover   Frequency
1:           ps_car_13 0.07690467 0.06999898 0.044136879
2: ps_reg_02 ps_car_13 0.03094357 0.04512123 0.032230820
3:           ps_ind_03 0.03010975 0.04508349 0.027626417
4:     ps_ind_05_cat_0 0.02728141 0.01303455 0.005231037
5:           ps_reg_03 0.02628067 0.02256930 0.024275283
6: ps_reg_01 ps_car_13 0.02598662 0.03634769 0.028852441
```

Gain is the improvement in accuracy brought by a feature to the branches it is on. The idea is that before adding a new split on a feature X to the branch there was some wrongly classified elements, after adding the split on this feature, there are two new branches, and each of these branch is more accurate (one branch saying if your observation is on this branch then it should be classied as 1, and the other branch saying the exact opposite).

From the above, we can see that as we progressed through features, the gain has came down.

We can also plot the top 10 important features in our model:

$> xgb.plot.importance(\texttt{importance\_matrix} = importance[1:10,])$

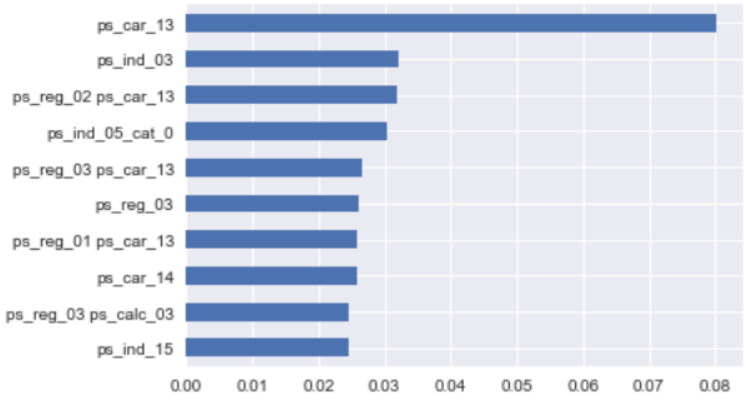

Figure 2: Top 10 important features

3.4 Linear Discriminant Analysis and Matlab Classification Learner App

The use of linear discriminant analysis for this problem came from using the classification learner app in Matlab. The classification learner app allows for evaluation of multiple machine learning classification algorithms on the same data set using parallel computing. The performance metric for evaluating the results of the algorithms included the confusion matrix and the ROC curve. Algorithms tested included Logistic Regression (LR), linear support vector machines, decision trees and linear discriminant analysis (LDA). Many of the higher order algorithms were not able to produce a result because of extremely long run times. The best performing algorithm as judged by the ROC curve and confusion matrix was the LDA algorithm. Figure 3 shows the ROC curves for the LR and LDA results. The results are similar but the LDA algorithm won out for both the area under the curve and the confusion matrix results for the percent of positive predictive values. Analysis of the confusion matrices indicates that the LR model appeared to be overpredicting the number of people that would file a claim when their target was actually not having filed a claim.

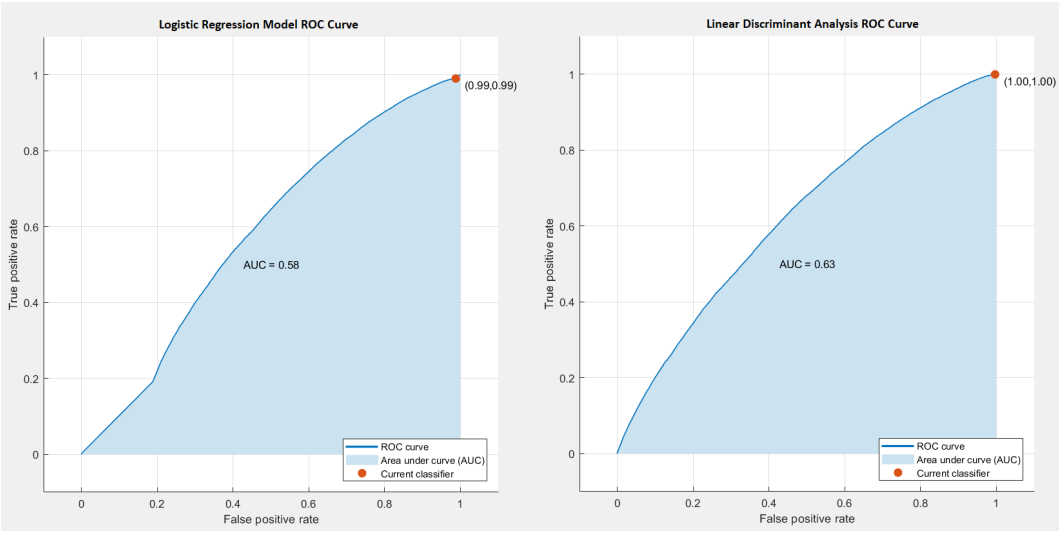

Figure 3: Comparison of ROC Curves

Linear Discriminant Analysis is a classification method that can be selected as an alternative to logistic regression. LDA makes use of Bayes' Law, but models the class conditional distribution of

the data as a multivariate Gaussian distribution [3]. This leads to several assumptions that must be met for LDA, including that the independent predictor variables must be normally distributed with equal covariance matrices.

These assumptions presented a confounding result of using the classification learner app in Matlab. On the one hand, many algorithms were easily tested and compared on the data set using the app. On the other hand, results were being generated without effectively considering the underlying requirements of the algorithm. After further reading about LDA and LR, it became apparent that LDA should underperform LR when categorical variables are included in the model, as these categorical variables theoretically violate the assumption of normality [5]. While preprocessing had changed all categorical variables to binary dummy variables, it has been shown that binary explanatory variables should achieve a better result using LR [4].

In summary, based on the literature it was strange that LDA outperformed LR and many other tested algorithms, based on the violation of the assumption of normality for categorical variables. It does make sense however that LDA could give a good result in this instance because one would expect a large sample of people's driving habits (e.g. the predictor variables) to follow a normal distribution. An answer to LDA's better performance in comparison to LR was not definitively found for this project as the group moved on to working with XGBoost when it was seen how well that algorithm performed. One possible explanation could be that the binary variables are less significant, as it has been shown that binarized insignificant variables have only a minor effect on LDA's performance in comparison to LR [4].

3.5 Random Forest

Random Forests is an ensemble learning method for classification and regression that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees. The main reason we chose this algorithm is that it tends to correct the overfitting sometimes caused by decision trees to their training set. To enhance our predictions, we also used a tuning method to find the best hyperparameters based on some previous options provided.

To implement this algorithm we used Python with Scikit-learn. We used the libraries RandomForestRegressor and GridSearchCV. This last one is responsible for simulating all the possible combinations of the hyperparameters we provided and chooses the best of them to build the model and calculate the predictions.

The table below shows the hyperparameters tested and the ones chosen by GridSearchCV as the best:

Hyperparameters	Set to Evaluation	Best Parameters
`n_estimators`	500, 1000, 2000	1000
`max_features`	0.5, 1.0, 'auto', 'sqrt', 'log2', None	0.5
`min_samples_leaf`	1, 5, 10	5
`min_samples_split`	2, 4	4

As a result, running this algorithm with our test set and submitting to Kaggle we got the score 0.20254.

3.6 AdaBoost

An AdaBoost regressor is a meta-estimator that begins by fitting a regressor on the original dataset and then fits additional copies of the regressor on the same dataset but where the weights of instances are adjusted according to the error of the current prediction. As such, subsequent regressors focus more on difficult cases. This algorithm was presented by Yoav Freund and Robert Schapire in [2] and they won the 2003 Gödel Prize for their work.

To implement this algorithm we used Python with Scikit-learn. We used the libraries AdaBoostRegressor and GridSearchCV to test the combinations of hyperparameters and define the model using the best. As base estimator for AdaBoost we utilized the DecisionTreeRegressor provided also by Scikit-learn.

The table below shows the hyperparameters tested and the ones chosen by GridSearchCV as the best:

Hyperparameters	Set to Evaluation	Best Parameters
`n_estimators`	100, 300, 500, 800	500
`learning_rate`	0.5, 0.1, 0.05, 0.01	0.05
`loss`	'linear', 'square', 'exponential'	'linear'

As a result, running this algorithm with our test set and submitting to Kaggle we got the score 0.20270.

4 Results

The results are based on the submission to the Kaggle competition and the resulting score. Our first submission was sent in November, 25 and the deadline for submissions was in November 29. At the end, we submitted 12 files and our final position on the leaderboard was 2985.

It is worthy to mention that the winner of this competition reached the score 0.29698 after 83 submissions and the second place reached the score 0.29413 after 231 submissions.

For each algorithm we tried different hyperparameters and we could improve our scores in some cases. The following table presents the results of our best submissions per algorithm.

Algorithm	Score on Kaggle
XGBoost	0.27411
Ensemble of XGBoost and LDA	0.26036
Ensemble of all	0.24958
Linear Discriminant Analysis	0.24801
Logistic Regression	0.23207
AdaBoost	0.20270
Random Forest	0.20254

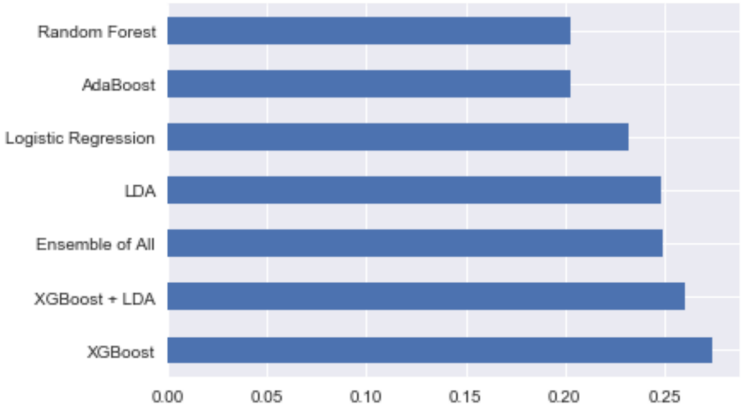

Figure 4: Kaggle Scores per Algorithm

5 Contributions of each group member

We considered that the contributions and the workload for each of the participants were well-balanced.

Since each one of participants have a more profound knowledge in one of the three programming languages we utilized, we decided to let each one decide what algorithm to implement.

Arin utilized R and chose the algorithms XGBoost, and Logistic Regression. The results from Logistic Regression didn't improve the default submission score. Finally he used the R's xgboost library to implement a basic XGBoost algorithm which improved the model.

Garrett utilized Matlab and chose the algorithms Logistic Regression, Neural Networks, Linear Support Vector Machine and Linear Discriminant Analysis. From those, the best results were obtained using LDA. A good result was not achieved using neural networks, and this was attributed to primarily to the large class imbalance.

Gustavo utilized Python and chose the algorithms Random Forest, AdaBoost and Neural Networks. From those, the best results were obtained using Random Forest. As the Matlab version of the Neural Networks, the results were not good and not considered as a submission to the competition.

The report was written collaboratively.

6 Conclusions

This project brought to us the opportunity to test different algorithms for predicting the probability of an insurance claim filing. Our best result in the Kaggle competition was achieved using the XGBoost algorithm. Allowing each participant to program in their preferred programming language was a good way of this doing this project as it allowed each group member to utilize their specific knowledge. In addition, it allowed for a broad cross-platform comparison of many classification algorithms. This was a good approach for starting a machine learning problem such as a Kaggle competition, because it allowed us to narrow down many algorithms and choose which ones to focus on.

A particularly exciting aspect of this project was participating in a Kaggle competition, competing with people all around the world to get the best score. This was our first experience in such kind of competition. Unfortunately we didn't have the time to do more testing and simulations as our window of submission was basically of one week, while a significant portion of the other competitors started submitting their results two months ahead. However, we are satisfied with our first attempt, and we feel challenged to keep participating in more competitions.

References

[1] Bert Carremans. Data preparation and exploration. `https://www.kaggle.com/bertcarremans/data-preparation-exploration`, 2017.

[2] Yoav Freund and Robert Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, page 55, 1997.

[3] The Mathworks Inc. Creating discriminant analysis model. `https://www.mathworks.com/help/stats/creating-discriminant-analysis-model.html`, 2017.

[4] Maja Pohar, Mateja Blas, and Sandra Turk. Comparison of logistic regression and linear discriminant analysis: a simulation study. *Metodoloski zvezki*, 1(1):143, 2004.

[5] S. James Press and Sandra Wilson. Choosing between logistic regression and discriminant analysis. *Journal of the American Statistical Association*, 73(364):699–705, 1978.