CS534 Final Project Report: Porto Seguro's Safe Driver Prediction

Chuan Tian and Shangjia Dong

***Abstract*— This final project is based on a Kaggle competition. The goal of the project is to predict the probability that if a driver will file an insurance claim next year. Methods include logistic regression, deep neural network, multilayer perceptron are used in this project. Various feature engineering techniques are used to improve the model performance. Finally, we are able to achieve the gini score of 0.26764, and ranked 3763 out of 5798 teams (64.90%)on Kaggle private leaderboard which uses 70% of the test data for evaluation.**

I. INTRODUCTION

Twenty states and the District of Columbia have mandatory requirements for uninsured or underinsured motorist coverage. According to a 2014 study by the Insurance Research Council (IRC), 12.6 percent of motorists, or about one in eight drivers, was uninsured in 2012 [1]. The high insurance bill is a painful to pay when you know you are a good driver. It is not fair that you have to pay so much if have been cautious on the road for years. Here we seek for a better solution using provided policy data.

Various machine learning methods have been developed over the years to conduct classifications and predict probabilities. Andrew Ng and Michael Jordan [2] conducted an comparison between Logistic Regression and Naive Bayes. As a member of discriminative learning algorithms, the logistic regression is more asymptotically efficient than its generative learning counterpart Naive Bayes. In this project, we decide to use logistic regression as one of the models to classfy the data. In addition, deep neural network is widely adopted for different applications, we also used it based on its general good performance. To build upon the methods we have learned through the course, a multi-layer perceptron method is also employed in this project.

*This work is prepared for the CS534 final project

Chuan Tian is a Ph.D. student in Statistics Department, Oregon State University, M115 Kidder Hall, Corvallis, OR, 97331. tianc@oregonstate.edu

Shangjia Dong is a Ph.D. student in School of Civil and Construction Engineering, Oregon State University, 211 Kearney Hall, Corvallis, OR, 97331. dongs@oregonstate.edu

There are 595,212 examples in training set, 57 features (excluding the *id* and *target* column), and 892,816 examples in test set, 57 features. There are Four groups of data: ind (18), reg (3), car (16), and calc (20). Two types of feature value: categorical (binary is considered as categorical), and numerical. The dataset is also extremely imbalanced: there are only 3.65% 1's (the people who file claim) and 96.35% 0's (people who didn't file claim).

TABLE I
DESCRIPTIVE STATISTICS

count	595212
mean	0.036448
std	0.187401
25%	0.000000
50%	0.000000
75%	0.000000
max	1.000000

The link to the Kaggle competition is here.

This report is organized as follows. Section II reviews existing approaches on this problem, mostly from Kaggle.com discussion board. Section III introduces the traditional accuracy evaluation and the performance measurement metric kaggle used to judge the model performance. Following by that, section IV described the experiments we have conducted throughout the projects, and section V presents the procedures we adopted to achieve the best results. Section VI concludes the project with major findings. Finally, section VII presents a discussion regards to challenges and achievements we accomplished in this project, and future directions to improve the model performance.

II. EXISTING APPROACHES

The competition was active till Wednesday afternoon (11/29/2017). After the competition, we reviewed other participant's approaches. We found that most of them use ensemble techniques to combine the results from sophisticated training methods such as lightgbm, xgboost and Neural Network. Old-school method like logistic regressions were also used, and have comparable results as the mentioned-above newer

methods. Some used single model and obtained decent result, but in general the ensembled models work better. Cross-validation is also widely used for tuning parameters/improving accuracy.

As of feature engineering, most people only used one-hot encoding on the categorical features. It is noteworthy that the 1st place winner "Michael Jahrel" (**the link to his post is here**) stopped here and let his models to do their magic for him (1 lightgbm and 5 Neural Network). He did a very good job on parameter tuning. Most of the others did more feature engineering than that. Some of them did Exploratory Data Analysis and deleted irrelevant features. For missing values, various ways of imputations could be applied. In addition, many people selected 2-way, 3-way and 4-way interactions and added to the model. The 3rd place winner in the public leader board "MSChuan" (**the link to their solution is here**) actually used greedy algorithm to search through all the possible interactions and added them to the model recursively, instead of using any feature importance score. The 2nd place winner (**the link to their post is here**) also mentioned a way of synthesizing features: they put similar features in the same group (ind, car, reg) and then take turns to use one group as response, and add XGBoost prediction of that group as features.

III. Evaluation Metric

A. Normalized Gini's Coefficient

The competition used normalized Gini's coefficient as the evaluation metric. After collecting the predicted probabilities of filing a claim on the test set, the evaluation algorithm first rearrange the actual label values (0's and 1's) based on the rank of the predictions' magnitude. Then, running sum of actual values as a function of the number of observations was computed, and this function is called "Lorenz curve". The area between Lorenz curve and the 45° line, which represents a random guess, is the Gini's coefficient. The algorithm then normalize this coefficient with the maximum of possible Gini's coefficients, which results from the case when the predicted probabilities are "perfect", as they are the actual values themselves. As a result, the normalized score is between 0 and 1, with 0 being the score of a random guess, 1 being the score of perfect predictions.

The figure from Wikipedia (**link is here**) illustrates the calculation of Gini's coefficient:

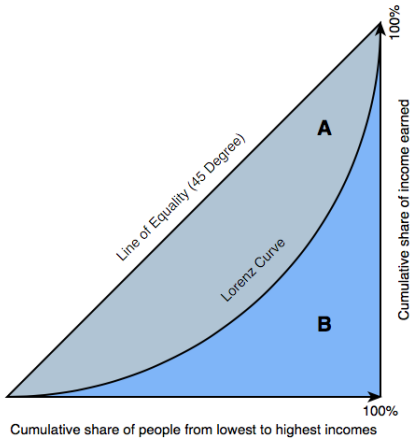

Fig. 1. Gini's coefficient illustration

B. Accuracy

Due to the extreme imbalance of the data classes, the prediction accuracy is usually dominated by the majority class, meaning that the standard learning algorithm would eventually learned an "naive model" and predict every example to be in the majority class and treat the negative examples as outliers. Because of that, it is very hard to improve the prediction accuracy between reasonable models (our baseline model already achieved the best accuracy, as we will point out below). However, we found that if the accuracy of a model is bad, its Normalized Gini's coefficient would also be quite off. So though accuracy might not be the best criterion for model improvement in this particular problem, it is still a good metric for double check.

IV. Preliminary Experiment

We have tried different feature engineering methods and training models to predict the probabilities. Following is a list of experiments we have conducted.

A. Baseline

To have an initial attempt, we build up the training framework with raw data as the input. The training accuracy is as follows:

TABLE II
Baseline Results

	Logistic Regression	DNN	MLP
Baseline	96.27%	96.27%	96.27%

As the results suggest, without feature engineering, model performance does not differ much. The best Gini

score achieved is 0.225. To improve the performance, we conducted following feature engineering.

B. Binarize Data

First, we tried to binarize the categorical data. There are 175 unique values in categorical data. After binarization, the number of features increased to 318. We used the binarized data as input, the training performance is as follows.

TABLE III
BINARIZED CATEGORICAL RESULTS

	Logistic Regression	DNN	MLP
Binarized Categorical	96.27%	96.27%	96.27%

We further tried to standardized the numerical features, the prediction accuracy does not change, and Gini score got worse. In addition, we binarized some of the numerical features, the performance is very similar.

C. Adding Feature Interaction

To added the model complexity, we attempted to adding polynomial interaction between features. However, this procedure failed due to the extremely large data size.

Above attempts made us realize that prediction accuracy is not the best evaluation metric for this problem. The data is extremely imbalanced. Current prediction accuracy is resulted from the bias. Therefore, feature selection is very necessary. With selected important features, we will be able to add feature interactions.

D. Techniques for Imbalanced Data

Due to the dominant effect of negative examples, we tried several techniques to improve the prediction. We first tried to increase the "class_weight" for the minority class in logistic regression, but that doesn't change the results much. We then tried Adaboosting which trains a series of classifiers sequentially and automatically increase the weight for incorrectly classified cases while decreasing the weights for the correctly classified cases. That took a long time, but the results remains similar as before. We then consider both undersampling and oversampling. Considering the huge dataset at hand, oversampling the minority class and synthesize even more examples doesn't seem a good idea. Using "imblearn" package in Python, we repeated undersampled the majority class and trained several claffiers, and then ensemble them via bagging. However, that actually make both accuracy and Gini's normalized coefficients worse, in the most cases.

V. RESULTS

After different experiments, we obtained our best model at time. The procedures are documented in the following subsections.

A. Feature Engineering

1) Missing Value: First we checked if there are missing value in the data, figure 2 shows there is a large amount of data missing, therefore, we treated missing values by its mode of the column.

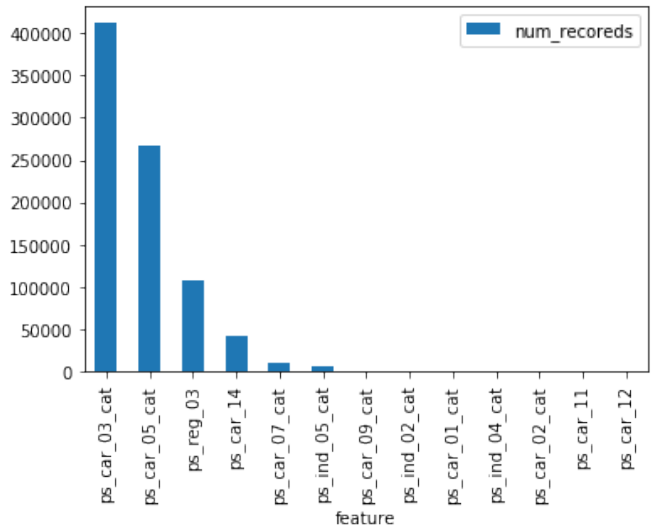

Fig. 2. Missing value in Data Set

2) Feature Correlation: Second, we decided to look into the correlation between the features. Figure 3 displays the correlation between the features. As we can observe, *calc* group does not have a strong relationship with other data. Therefore, we decided to eliminate them from modeling training process.

3) Binarize Categorical Feature: From original dataset, we have 14 features are categorical data, in total there are 175 unique values. Since the size is relatively small. we decide to binarize all of the categorical feature.

4) Scale the Numerical Feature: There are 12 numerical features. The dispersion of data would potential lead to the learning performance drop. Thus, we scaled the data into zero mean and unit variance.

5) Feature Selection And Increase Model Complexity: Noticing that the techniques for imbalanced data didn't work, we suspect that the data is inseparable in the linear feature space. If we had more computing resources, feature selection wouldn't be necessary for

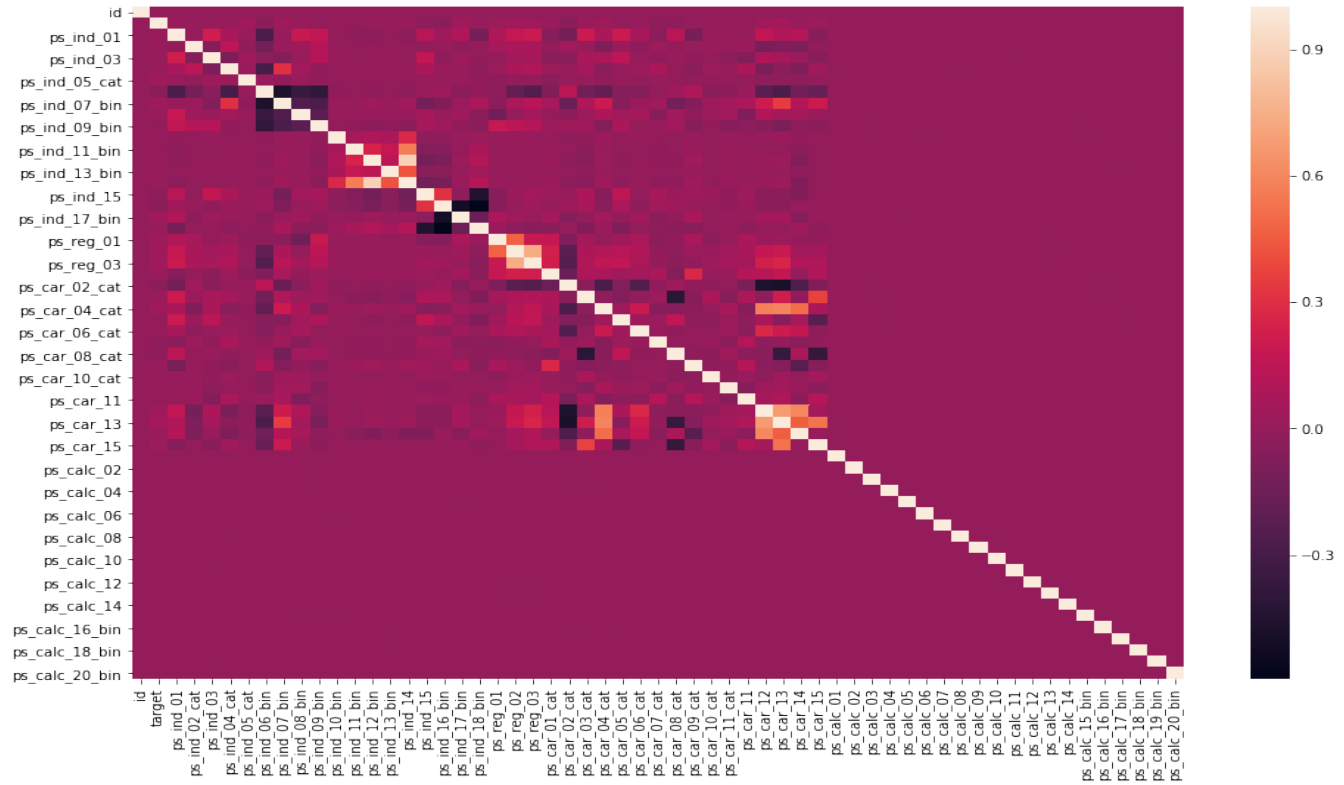

Fig. 3. Correlation between features

the purpose of prediction. However, we quickly ran out of memory in previous attempts to simply square all the features. Thus, we used XGBoost to select the important linear features, and then square all of those important features. Then we reached our best model.

B. Model

1) Logistic Regression: Unlike the traditional linear regression model which models the response directly, logistic regression models the probability of getting a positive response (get a "1") instead. Being a discriminative model which models the posterior distribution, logistic regression is believed to have lower asymptotic error rate than its generative counterpart (Naive Bayes, for example), and have good probability prediction accuracy, since it models probability directly and needs no further calibration. Considering our huge training size, and that we have a competition goal of predicting probabilities instead of making classifications, we think logistic regression is a good choice. And in the end, it gave us the best results.

2) Deep Neural Network: Distinguished from the common neural networks which have single-hidden-layer neural networks, deep neural networks have more depth by encapsulating multiple hidden layers. In deep learning network, each layer of nodes trains on a distinct set of features based on the previous layer's output. The further you advance into the neural net, the more complex the features are [3]. Also, deep learning networks are able to automatically extract features without human intervention. In this project, we used *Keras* to build up the model pipeline. We used three layers. The detailed parameter is displayed in the figure below. The input_dim depends on the input data shape. As you may notice, we dropped 50% the feature between the layers to prevent the over-fitting occurring.

3) Multi-layer Perceptron: A multilayer perceptron (MLP) is a supervised learning algorithm. Different from logistic regression, between input and output layer, there can be one or more non-linear layers. MLP is good at on-line learning and learning non-linear models. However, it is sensitive to feature scaling and requires parameter tuning. In this project, I used sklearn.neural_neural.MLPClassifer to build up the model. The model is consist of input layer, a hidden layer (60 neurons), and a output layer. I used adam

Layer (type)	Output Shape	Param #	Connected to
dense_1 (Dense)	(None, 794)	631230	dense_input_1[0][0]
dropout_1 (Dropout)	(None, 794)	0	dense_1[0][0]
dense_2 (Dense)	(None, 300)	238500	dropout_1[0][0]
dropout_2 (Dropout)	(None, 300)	0	dense_2[0][0]
dense_3 (Dense)	(None, 100)	30100	dropout_2[0][0]
dropout_3 (Dropout)	(None, 100)	0	dense_3[0][0]
dense_4 (Dense)	(None, 1)	101	dropout_3[0][0]

Total params: 899,931
Trainable params: 899,931
Non-trainable params: 0

Fig. 4. Deep Neural Network Structure

optimizer, and adaptive learning rate for training.

C. Best Model Result And Kaggle Ranking

After the data preprocessing mentioned about, we got our best Normalized Gini's coefficient to be 0.26764 from a single logistic regression model with hyper parameter C = 1e10. The competition was closed a couple of days ago so our ranking couldn't update, but by comparing it manually with other people's results, we could rank 3763 out of 5798 teams on the private leader board (which use 70% of the test data for evaluation, and counts as the final competition result. The public leader board only use the other 30% of the test data and is there to give people an estimate of their final ranking).

VI. Conclusion

In this project, we used missing feature elimination, feature selection, binarization, and feature interaction to engineer the data. After that, we fed the processed data to the training model and obtained the best performance. Among the methods we tried, logistic regression gives us the best result, which achieves a Gini score of 0.26764, and rank 3763/5798. Although with other recommended methods in the forum, we were able to achieve the results of 0.285, we decide to not report it as our major result since we did not have enough time and computing power to tune the parameters by ourselves.

One major lesson we have learned from this project is how to deal with the imbalanced dataset. Feature engineering is the most important part, it will boost the performance regardless of the model selection. However, if a more accurate result is desired, advanced techniques such as lightbgm, or xgboost should be considered.

VII. Discussion

With current feature selection and feature engineering, we are able to achieve the results of 0.26764. However, this result is still far from the best result achieved on leaderboard. Therefore, we decided to check the training models that other participants used. Lightgbm is one of the most frequent used methods on the leadeboard and gives a very good performance. Therefore, we also adopted the lightgbm and explored its classification performance on this dataset. By using K-fold cross validation, and trained lighgbm parameter from **this kernel**, we were able to achieve the Gini score of 0.285, which would make us rank around 35%. We believe with more parameter tuning, we can achieve better results with lightbgm. However, we were not able to fully understand this method and implemented it on our own. So we decide to report the results we obtained from our original work.

It seems that the mastery of the most cutting-edge techniques such as Lightgbm, XGBoost and Neural Network is essential in this Kaggle competition. Like we mentioned in "Existing Approches", the top 1 winner saved lots of efforts in feature engineering by combining Lightgbm and Neural Network together. Although we also used Neural Network, our tuning skills still have a lot of space for improvement. People also reported that they have good results from single

model XGBoost, whereas we only used it for feature selection. Learning the mechanism behind those methods and how to use them is definitely in our study plan for machine learning. In addition, comparing several methods and select the best one seems out-of-date comparing with aggregating several best-performing ones. In addition to simply averaging the predictions from different models, one could also combine the models via ensemble techniques like bagging and boosting.

It also worth noticing that the techniques customized for fighting imbalanced data didn't help. We suspect the culprit is that by balancing the data, we changed its class distributions before training and that is not good for a learning algorithm sometimes, depending on the specific distributions. We tried combining those balancing techniques with squaring the features, but they only made the results worse. And we learned by another example that no approach is guaranteed to work in every case.

Further, it seems that adding more interactions led to our key improvement from the baseline. In fact, we don't think selecting on the linear feature space and then adding all 2-way interactions is the best approach of increasing model complexity. Being "unimportant" as a linear feature doesn't mean that it is also unimportant when it is a part of a 2-way interaction, for example. It would make more sense if we could search through all the 2-way, 3-way and 4-way interactions and select the most important subsets of features based on some carefully-chosen criterion. However, our laptops ran out of memory after we have about 900 features in total, so we had to stop at the current model. As a matter of fact, most top participants on leader board reveal that they took advantage of the modern computation resources, like huge memory and multicore CPU and GPU's. The 1st prize winner claimed that most of his models could be trained on a 32GB memory machine, however, that is still beyond our current laptop configuration (Chuan's mid-2012 Macbook Pro only has 8GB memory, for instance).

Time was also a limit on our final model performance. We entered the competition in the last a couple of weeks before its closure, and many teams in the competition have already been working on it for two or three months, and have more time to attempt various approaches. Sufficient time for training is an important factor to winning as well. The top winner mentioned that the training of his model took a lot of time, and so did the 3rd team on the public leader board who use greedy algorithm to search interaction features. The silver lining in this situation is that we were able to see the solutions from the winners in the last two days before the final report is due, and actually got a lot of inspirations. However, we were unable to follow most of the solutions even if we have much more powerful workstations, since they wouldn't finish training in time.

REFERENCES

[1] *Facts + Statistics: Uninsured motorists*. https://www.iii.org/fact-statistic/facts-statistics-uninsured-motorists.

[2] Andrew Y Ng and Michael I Jordan. On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes. In *Advances in neural information processing systems*, pages 841–848, 2002.

[3] deeplearning4j. *Introduction to Deep Learning*. https://deeplearning4j.org/.