Porto Seguro's Safe Driver Prediction Project Report

Hanlin He, Mingze Xu, Su Yang, Tao Wang
Department of Computer Science, The Erik Jonsson School of Engineering and Computer Science
The University of Texas at Dallas
Email: {hxh160630, mxx160530, sxy161730, txw162630}@utdallas.edu

***Abstract*—This report gave an overview of our approach toward Porto Seguro's Safe Driver Prediction competition hosted on Kaggle, include preprocess, training and result analysis. Multiple models were used in our experiment and their performances were compared.**

I. Introduction

Machine learning is emerging in the insurance industry and is being applied across multiple areas including the interpretation of data, business operations and driver safety. One key application is claim prediction. Inaccuracies in car insurance company's claim predictions raise the cost of insurance for good drivers and reduce the price for bad ones. A more accurate prediction will allow insurers to further tailor their prices, and hopefully make auto insurance coverage more accessible to more drivers.

In this report, we based on Kaggle's Featured Prediction Competition *Porto Seguro's Safe Driver Prediction* [3], conducted several experiments, compared different approaches' effectiveness to tackle the claim prediction problem, including *logistic regression*, *tensorflow estimator (neural network)* and *random forest*.

The organization of the following report is as follows. In section II, we will formally define the problem to solve and discuss the theoretical principle of the algorithm we used. Then we will analyze the data feature and our method of feature engineering in section III. After that, the experimental results are shown and analyzed in section IV. Finally, we will discuss related works and conclude the report.

II. Problem Definition and Algorithm

A. Task Definition

A machine learning problem is defined as to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E [29]. The claim prediction problem can be defined as follow:

- E previous year's policy holders' information and whether or not a claim was filed for that policy holder.
- T predicting the possibility that an auto insurance policy holder will file an insurance claim next year.
- P the accuracies and Gini Coefficient was used to measure the effectiveness of models.

B. Algorithm Definition

We used logistic regression, ensemble methods like random forests and gradient boost in our model building. In this section, we will discuss the theoretical foundation of these algorithms.

1) Logistic Regression: Logistic Regression is an approach to learning functions of the form $f : X \rightarrow Y$ [28] or in our case $P(Y|X)$ where Y is discrete-valued, and $X = \langle X_1 ... X_n \rangle$ is any vector containing discrete and continuous variables. The parametric model assumed by Logistic Regression in the case where Y is Boolean is:

$$P(Y = 1|X) = \frac{1}{1 + \exp(w_0 + \sum_{i=1}^{n} w_i X_i)}$$

$$P(Y = 0|X) = \frac{\exp(w_0 + \sum_{i=1}^{n} w_i X_i)}{1 + \exp(w_0 + \sum_{i=1}^{n} w_i X_i)}$$

One reasonable approach to training Logistic Regression is to choose parameter values that maximize the conditional data likelihood. We also used regularization to reduce the overfitting problem. The penalized log likelihood function is as followed:

$$W \leftarrow \arg\max_{W} \sum_{l} \ln P(Y^l|X^l, W) - \frac{\lambda}{2}||W||^2$$

where the last term is a penalty proportional to the squared magnitude of W.

In general, the algorithm used gradient ascent to repeatedly update the weights in the direction of the gradient, on each iteration changing every weight w_i, beginning with initial weights of zero, according to:

$$w_i \leftarrow w_i + \eta \sum_{l} X_i^l (Y^l - \hat{P}(Y^l = 1|X^l, W)) - \eta \lambda w_i$$

where η is a small constant which determines the step size.

The actual implementation of Scikit-learn library includes multiple solvers, such as Stochastic Average Gradient (`SAG`) descent, `SAGA` and Broyden-Fletcher-Goldfarb-Shanno (`LBFGS`).

2) Ensemble Methods: An ensemble of classifiers is a set of classifiers whose individual decisions are combined in some way typically by weighted or unweighted voting to classify new examples [18].

Random forests [27] is an ensemble learning method for classification, regression and other tasks, 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. Random decision forests correct for decision trees' habit of overfitting to their training set.

The algorithm for random forests is shown in algorithm 1.

Algorithm 1 Algorithm for Random Forests

Input: Dataset D containing N instances and M attributes
Output: A Classifier
Training Phase:
for $b = 1$ to B **do**
Create a bootstrap sample of size N from original data
Grow tree T_b using the bootstrap sample as follows:
Create a random sample of m attributes ($m < M$).
Use these attributes to construct a decision tree as usual.
end for
Testing Phase:
for Each a new data point X **do**
Take the aggregated prediction of models:
For classification: $aggregate = majority$
For regression: $aggregate = average$
end for

Gradient boosting [22] is 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. It builds the model in a stage-wise fashion like other boosting methods do, and it generalizes them by allowing optimization of an arbitrary differentiable loss function. The algorithm for random forests is shown in algorithm 2.

The implementation in Scikit-learn library of these two ensemble methods are `RandomForestClassifier` and `GradientBoostingClassifier` respectively.

III. Feature Analysis and Engineering

The data comes in the traditional Kaggle form of one training and test file each: `train.csv` and `test.csv`. Each row corresponds to a specific policy holder and the columns describe their features. The target variable is named `target` here and it indicates whether this policy holder made an insurance claim in the past.

A. Data Overview

In total, there are 595212 training data instances and 892816 testing data instances. Since it's a competition, testing data had no `target` label. We cannot test our model accuracy based on testing data. Instead, we would use k-fold and cross validation on the training data to evaluate our model, which is discussed in more detail in section IV.

In the train and test data, features that belong to similar groupings are tagged as such in the feature names (e.g., `ind`, `reg`, `car`, `calc`). In addition, feature names include the postfix `bin` to indicate binary features and `cat` to indicate

Algorithm 2 Algorithm for Gradient Boosting [11]

Input: training set $\{(x_i, y_i)\}_{i=1}^n$, a differentiable loss function $L(y, F(x))$ number of iterations M.
Algorithm:
Initialize model with a constant value:

$$F_0(x) = \arg\min_{\gamma} \sum_{i=1}^{n} L(y_i, \gamma).$$

for $m = 1$ to M **do**
1. Compute so-called pseudo-residuals:

$$r_{im} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)} \quad \text{for } i = 1, \ldots, n.$$

2. Fit a base learner (e.g. tree) $h_m(x)$ to pseudo-residuals, i.e. train it using the training set $\{(x_i, r_{im})\}_{i=1}^n$.
3. Compute multiplier γ_m by solving the following one-dimensional optimization [12] problem:

$$\gamma_m = \arg\min_{\gamma} \sum_{i=1}^{n} L\left(y_i, F_{m-1}(x_i) + \gamma h_m(x_i)\right)$$

4. Update the model:

$$F_m(x) = F_{m-1}(x) + \gamma_m h_m(x)$$

end for
Output $F_M(x)$.

categorical features. Features without these designations are either continuous or ordinal. Feature count in each category and type are shown in table I.

TABLE I
Feature Counts in Each Category and Type

Type	Binary	Categorical	Numeric	Total
`ind`	11	3	4	18
`reg`	0	0	3	3
`car`	0	11	5	16
`calc`	6	0	14	20

Although feature's categories are provided, the meaning of each feature remains unknown. Some participants have guessed the meaning of several features, for example the *binary* variables `ps_ind_06-10` are *one-hot encoded*, and `ps_car_13` might be car's mileage. Our experiment shows that some of the assumptions are plausible, however, we did not rely on this information to build our model.

B. Distribution Analysis

Features' distribution for `ind`, `reg`, `car`, `calc` are shown as histograms in fig. 1, fig. 2, fig. 3 and fig. 4 respectfully.

From histograms, we may conclude the following:

- Some binary features are dominated by `True` (`False`), for example `ps_ind_06-10`.

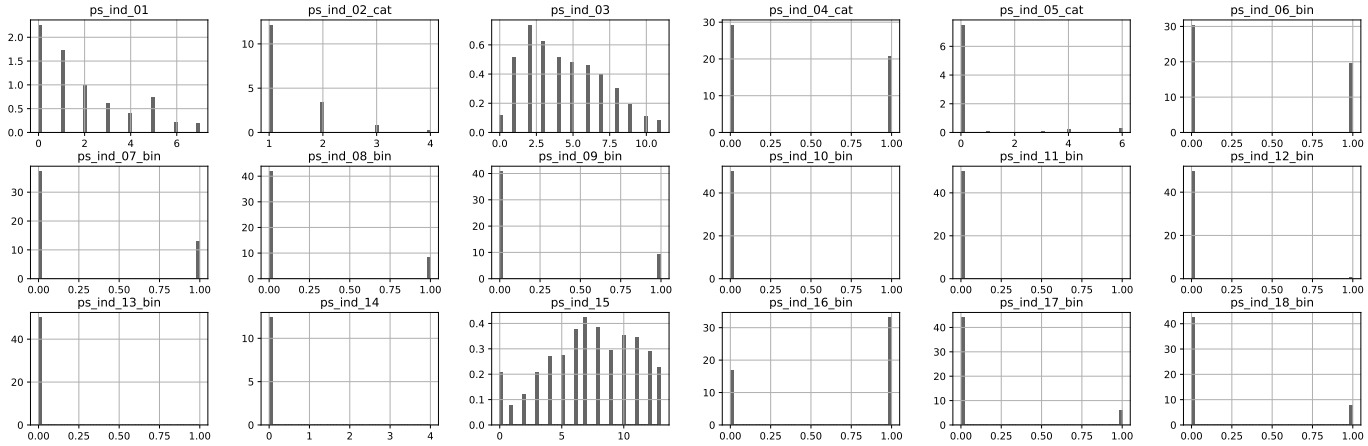

Fig. 1. Histogram for the `ind` Attributes.

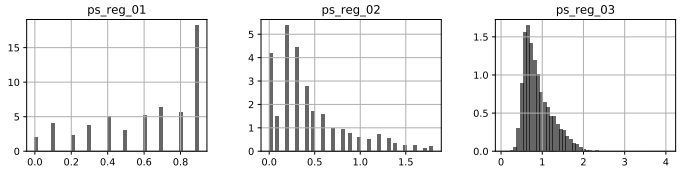

Fig. 2. Histogram for the `reg` Attributes.

- Some features' distribution is fairly even, for example `ps_calc_01`, `ps_calc_02` and `ps_calc_03`.

It might seem plausible to discard these features in training phase. But the deletion requires more careful consideration. As mentioned above, `ps_ind_06-10` might be *one-hot encoded*, thus, all these four features would be naturally exclusive of each other.

C. Correlation Analysis

Features' Correlation within each category, i.e., `ind`, `reg`, `car`, `calc` are shown as histograms in appendix fig. 6, fig. 7, fig. 8 and fig. 9 respectfully.

From correlation plot, we can conclude that strong correlation did not exist. It is not possible to conduct any dimension reduction based on correlations between features.

D. Missing Value Mechanism and Data Imputation

Values of -1 indicate that the feature was missing from the observation. Missing values count in each feature is shown in table II:

There are several approaches to handle missing values [30]:

- Deletion methods: cases with missing values are discarded, the analyses are restricted to cases that have complete data.
- Single imputation methods: imputes (i.e., *fills in*) the missing data with seemingly suitable replacement values. Mean, median and mode are the most popular averaging techniques to use [25].

TABLE II
MISSING VALUE COUNTS IN EACH FEATURE

Feature Name	Count	Percentage
`ps_ind_02_cat`	216	0.0363%
`ps_ind_04_cat`	83	0.0139%
`ps_ind_05_cat`	5809	0.9760%
`ps_reg_03`	107772	18.1065%
`ps_car_01_cat`	107	0.0180%
`ps_car_02_cat`	5	0.0008%
`ps_car_03_cat`	411231	69.0898%
`ps_car_05_cat`	266551	44.7825%
`ps_car_07_cat`	11489	1.9302%
`ps_car_09_cat`	569	0.0956%
`ps_car_11`	5	0.0008%
`ps_car_12`	1	0.0002%
`ps_car_14`	42620	7.1605%

- Multiple imputation methods: creates several copies of the data set, each containing different imputed values. These methods have greater performance but are harder to implemented.

From the table we can see that, most features have no missing value (only 13 features out of 51 have). Some features only have a small number of instances with missing value (`ps_car_02_cat` and `ps_car_11` have 5 missing value and `ps_car_12` have only 1).

Based on these observation, we handled the missing values as follow:

- For features with a few missing values, delete those data instances.
- For categorical features, we treat the missing values a new category.
- For numeric features, we use the mean of the population as missing value.

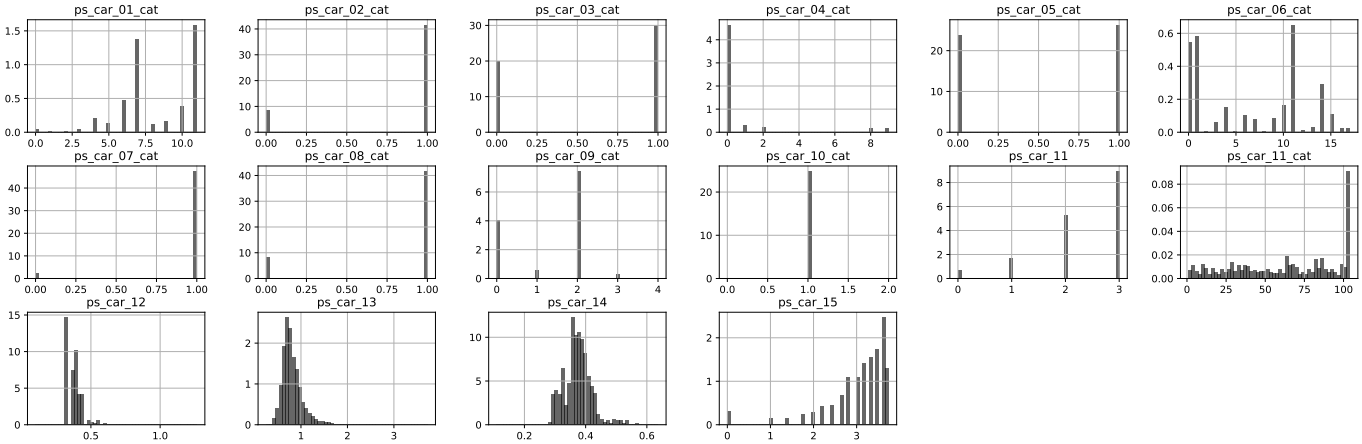

Fig. 3. Histogram for the `car` Attributes.

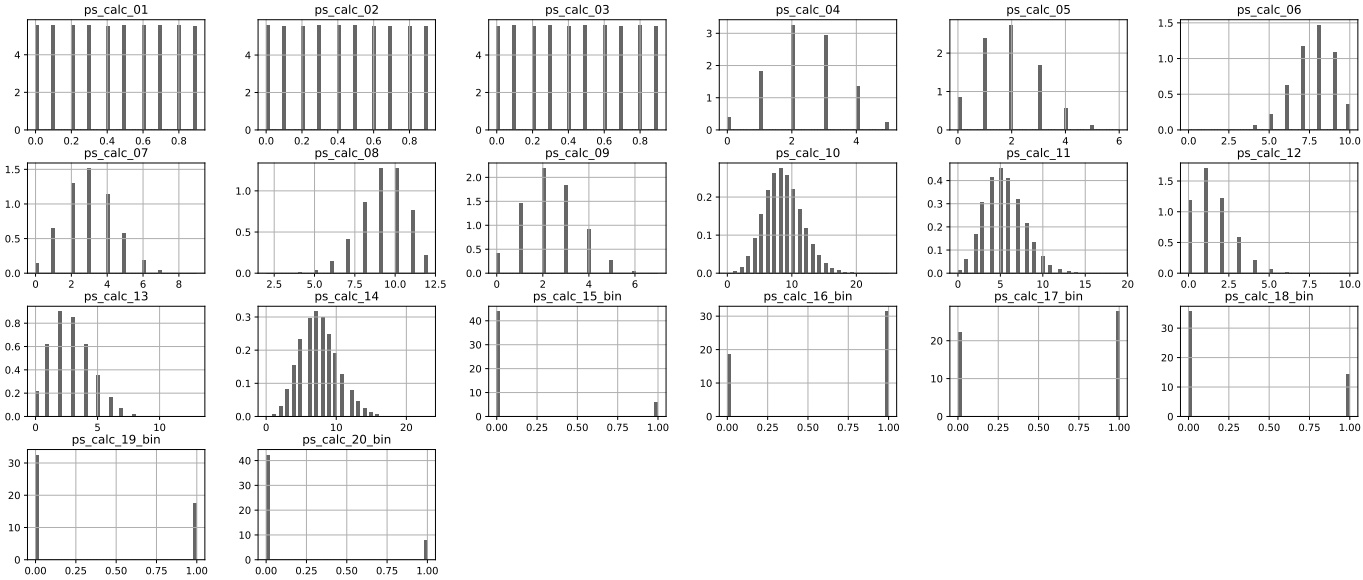

Fig. 4. Histogram for the `calc` Attributes.

IV. Experimental Evaluation

A. Hyperparameter Tuning

Hyper-parameters are parameters that are not directly learnt within estimators. Hyperparameter settings could have a big impact on the prediction accuracy of the trained model. Optimal hyperparameter settings often differ for different datasets [36].

- Grid Search, which is commonly accepted as naive because the number of required samples grows exponentially in dimension, assuming the number of points per axis is held fixed. Though most expensive in terms of total computation time, if run in parallel, it is fast in terms of wall clock time. [26] has provided some insights into the relationship between classical grid search and probabilistic roadmaps.
- Random Search, which is a slightly variation on grid search. In stead of searching over the entire grid, random search only evaluates a random sample of points on the grid. [15] has shown that random search is more efficient than grid search for hyper-parameter optimization in the case of several learning algorithms on several data sets.

Scikit-learn provides both Exhaustive Grid Search [7] and Randomized Parameter Optimization [8] class. To gain more insight about the hyperparameter tuning process (though might be useless), we used `ParameterGrid` in our implementations. Test results of all parameter sets are included in log files.

B. Evaluation Metrics

We used per-class accuracy, AUC and Normalized Gini Coefficient to evaluate out model.

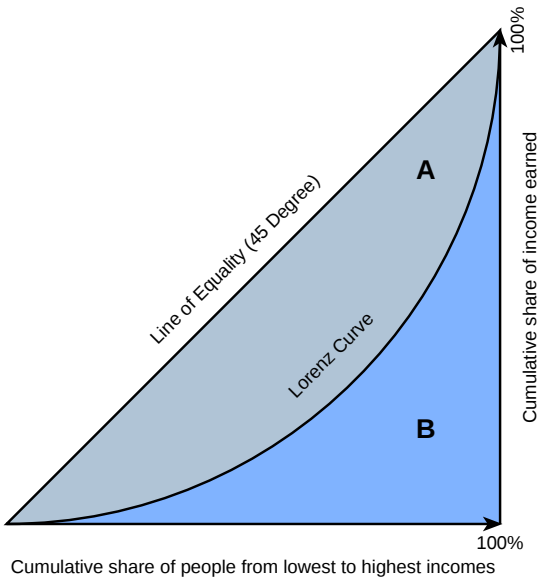

Fig. 5. Graphical Representation of the Gini Coefficient

1) Per-Class Accuracy: Accuracy simply measures how often the classifier makes the correct prediction. It's the ratio between the number of correct predictions and the total number of predictions (the number of data points in the test set) [36]:

$$accuracy = \frac{\#\text{ correct predictions}}{\#\text{ total data points}}$$

Per-class accuracy is the average of the accuracy for each class. By using per-class accuracy, we can have a better understanding of the model if the target class was dominated by one label.

2) Normalized Gini Coefficient and AUC: The Normalized Gini coefficient, (named for the similar Gini coefficient/index used in Economics, which originally developed by Italian statistician and sociologist Corrado Gini [23]), measures the inequality among values of a frequency distribution (for example, levels of income) [10]. It is most commonly defined as twice the area between the ROC curve and the diagonal (with this area being taken as negative in the rare event that the curve lies below the diagonal) [20].

Figure 5 shows that the Gini coefficient is equal to the area marked A divided by the sum of the areas marked A and B, that is, $Gini = A/(A + B)$. It is also equal to $2A$ and to $1 - 2B$ due to the fact that $A + B = 0.5$ (since the axes scale from 0 to 1).

The normalized Gini coefficient and AUC are closely related. AUC stands for area under the curve. When using normalized units, the AUC is equal to the probability that a classifier will rank a randomly chosen positive instance higher than a randomly chosen negative one (assuming 'positive' ranks higher than 'negative') [21]. Elementary geometry shows that

$$Gini = 2 \times \text{AUC} - 1. \quad (1)$$

The competition used normalized Gini coefficient to measure participants' submission performance. In our experiment, we worked in terms of both Gini coefficient and AUC.

C. Evaluation Mechanisms

We use k-fold cross-validation as our validation technique.

In k-fold cross-validation, we first divide the training dataset into k folds. For a given hyperparameter setting, each of the k folds takes turns being the hold-out validation set; a model is trained on the rest of the $k-1$ folds and measured on the held-out fold. The overall performance is taken to be the average of the performance on all k folds. Repeat this procedure for all of the hyperparameter settings that need to be evaluated, then pick the hyperparameters that resulted in the highest k-fold average.

Scikit-learn provides multiple variations of k-fold [32], for example,

- `KFold`, which divides all the samples in k groups of samples, called folds of equal sizes (if possible). The prediction function is learned using $k - 1$ folds, and the fold left out is used for test.
- `LeaveOneOut`, which is basically special simple case when $k = n$ in `KFold`.
- `StratifiedKFold`, which returns *stratified* folds, i.e., each set contains approximately the *same* percentage of samples of each target class as the complete set.

We chose `StratifiedKFold` in out implementations during training phase.

D. Results

The best parameters of each model in our experiment is shown in table III.

The result in public dataset performs best in *Gradient Boosting*, then it is almost same for *Logistic Regression* and *Random Forest*. It is the same on private dataset.

Weak learners have high bias and low variance. For a decision tree, the weak learners are shallow trees and sometimes just stumps. Boosting mainly reduces bias to reduce error. On the other hand, *Random Forest* tackles error by reducing variance in the opposite way as *Gradient Boosting* does. As a result, it cannot deal with bias. *Logistic Regression* is some way like Gradient Boosting. It depends on the data set and type of features to know which model performs better.

For getting the best result, we adjusted the max depth of *Gradient Boosting* to 4 after times of attempts, and since this model works well for overfitting we set the estimator to higher value as 200. We also used the 'deviance' value for the loss parameter to optimize the lost function just like the way *Logistic Regression*. This may get the conclusion that the comparison of *Gradient Boosting* and *Logistic Regression* depends on the dataset and features.

In *Logistic Regression*, we set the C parameter to only 0.1 to inverse the regularization strength to prevent overfitting. The

TABLE III
PERFORMANCE OF EACH MODELS

Model	Optimal Parameters	Public Dataset	Private Dataset
Gradient Boost	`'loss': 'deviance', 'max_depth': 4, 'n_estimators': 200`	0.283	0.277
Logistic Regression	`'C': 0.1, 'fit_intercept': False, 'penalty': 'l1' 'random_state': 0, 'solver': 'liblinear'`	0.266	0.257
Random Forest	`'bootstrap': False, 'criterion': 'entropy', 'max_depth': 10 'n_estimators': 40, 'warm_start': False`	0.263	0.258

random state is set to 0 to control the random for every attempt to get the same and best result, compared to not setting random state or other state values, when dividing training set and testing set, avoid mess when in the next attempt. The parameter of `solver` is set to `liblinear` to work with the `l1` penalty, at the same time, this problem is not a multiclass problem so there is no need to use other values of this parameter.

For the model of *Random Forest*, the maximum depth is set to a number, not infinite by default to save running time however may be the reason why *Random Forest* is a little worse than *Gradient Boosting*. Other parameter is the best after many attempts to the best classification result.

V. RELATED WORK

Machine learning techniques have been widely used in the field of car insurance to help the insurance companies in predicting the customers' choices in order to provide more competitive services. Related works concerned following topics.

1) Insurance Purchase Prediction: Asma S Alshamsi [14] built a classification model based on random forest that could be applied in predicting which of the insurance policies would likely to be chosen by the customers. Saba Arslan Shah and Mehreen Saeed [33] gave an overview of the methodology developed for predicting the purchased policy for a customer (in a Kaggle hosted competition [2]), using logistic regression, naive Bayes, SVM and random forest.

2) Drive Style Modeling: [19], [31] analyzed 'good' and 'bad' drivers' driving style using unsupervised learning and deep learning with vehicle sensor data, e.g., GPS. [34] used Convolutional Neural Networks (CNNs) to perform deep learning on an image dataset of drivers in their cars performing specific actions, trying to identify the *unsafe* actions of a driver.

3) Claim Prediction: Dal Pozzolo et al. [17] surveyed many popular machine learning models' performances on claim prediction. The technique and measure metric (Gini index) used in their paper were very similar to ours, and greatly inspired our progress. [24] gave a full report about yet another Kaggle hosted competition Allstate Claim Prediction Challenge [1].

VI. FUTURE WORK

Though multiple models were compared in our experiments, the overall performance was not very good. Our final score received on Kaggle was 0.277 in Gini score, using the Gradient Boosting model of Scikit-learn. The gold medal submission was 0.291. There was a great gap in between.

We believe our work can be further optimized in following aspects.

1) Dimension Reduction: In our preprocessing part, we calculated the correlation matrix to see the possible relations between features. As a result, we conclude that we cannot ignore any feature just by looking at the correlation matrix, since none of them are closely related. This decision can be made automatically using library.

Scikit-learn provide a *Principal component analysis* (`PCA`) library, performing linear dimensionality reduction using *Singular Value Decomposition* of the data to project it to a lower dimensional space. We tried this library with a few experiments. Though the our result showed that there is still no clear separation between points, it is still a possible way to improve the final result.

Another method with great potential is t-Distributed Stochastic Neighbor Embedding (t-SNE) [35], a (prize-winning) technique for dimensionality reduction that is particularly well suited for the visualization of high-dimensional datasets [4]. Unlike PCA, t-SNE preserves the local structure of data points. The problem with Scikit-learn implementation of t-SNE is its lack of memory optimization and running time is unbearably high. There are some optimized implementations in other language like Julia [5] and Java [6]. Further works can try on these library for Dimension Reduction.

2) More Powerful Library, Models and Machines: The goodness of the implementation of library is vital in model's performance. It's meaningless to use the 'best' model if the training phase lasts forever. Also, different libraries have implementation of same model with greatly differed efficiency.

In our experiment, besides the model listed above, we also tried the neural network model implemented in Scikit-learn, i.e., `MLPClassifier` [9] class. No matter what parameters we tried, the results were either the training phase was unbearably long, or the Gini scores were not better than just random guess. However, as the competition ended, it turned out that the second place model was built on deep learning neural network using a different library (Keras [16]) based on tensorflow [13].

Hence, a notable future work direction would be trying to build the model using more powerful library and models.

VII. CONCLUSION

In this report, we first formally defined the *Porto Seguro's Safe Driver Prediction* problems we were trying to tackle with.

Then we gave an overview of our approach's methodology toward the problem, include algorithm description, evaluation metrics, preprocessing phase, training phase and result analysis. Multiple models were used in our experiment and their performances were compared.

REFERENCES

[1] Allstate Claim Prediction Challenge, 2011. [Online; accessed December 2, 2017].

[2] Allstate Purchase Prediction Challenge, 2014. [Online; accessed December 2, 2017].

[3] Porto Seguros Safe Driver Prediction, 2017. [Online; accessed December 2, 2017].

[4] t-SNE, 2017. [Online; accessed December 2, 2017].

[5] t-SNE in Julia, 2017. [Online; accessed December 2, 2017].

[6] T-SNE-Java, 2017. [Online; accessed December 2, 2017].

[7] Scikit-learn api. 3.2.1. Exhaustive Grid Search, 2017. [Online; accessed December 2, 2017].

[8] Scikit-learn api. 3.2.2. Randomized Parameter Optimization, 2017. [Online; accessed December 2, 2017].

[9] Scikit-learn api. sklearn.neural_network.MLPClassifie, 2017. [Online; accessed December 2, 2017].

[10] Wikipedia: Gini coefficient. https://en.wikipedia.org/wiki/Gini_coefficient, 2017. [Online; accessed December 2, 2017].

[11] Wikipedia: Gradient boosting. https://en.wikipedia.org/wiki/Gradient_boosting, 2017. [Online; accessed December 2, 2017].

[12] Wikipedia: Line search. https://en.wikipedia.org/wiki/Line_search, 2017. [Online; accessed December 2, 2017].

[13] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.

[14] Asma S Alshamsi. Predicting car insurance policies using random forest. In *Innovations in Information Technology (INNOVATIONS), 2014 10th International Conference on*, pages 128–132. IEEE, 2014.

[15] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(Feb):281–305, 2012.

[16] François Chollet. Keras (2015). *URL http://keras. io*, 2017.

[17] Andrea Dal Pozzolo, Gianluca Moro, Gianluca Bontempi, and Dott Yann Aël Le Borgne. *Comparison of Data Mining Techniques for Insurance Claim Prediction*. PhD thesis, PhD thesis, University of Bologna, 2010.

[18] Thomas G Dietterich et al. Ensemble methods in machine learning. *Multiple classifier systems*, 1857:1–15, 2000.

[19] Weishan Dong, Jian Li, Renjie Yao, Changsheng Li, Ting Yuan, and Lanjun Wang. Characterizing driving styles with deep learning. *arXiv preprint arXiv:1607.03611*, 2016.

[20] Robert Dorfman. A formula for the gini coefficient. *The Review of Economics and Statistics*, 61(1):146–149, 1979.

[21] Tom Fawcett. An introduction to roc analysis. *Pattern Recogn. Lett.*, 27(8):861–874, June 2006.

[22] Jerome H Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232, 2001.

[23] Corrado Gini. *Variabilitá e Mutuabilitá.* 1912. Contributo allo Studio delle Distribuzioni e delle Relazioni Statistiche. C. Cuppini, Bologna.

[24] Dan Huangfu. *Data Mining for Car Insurance Claims Prediction*. PhD thesis, WORCESTER POLYTECHNIC INSTITUTE, 2015.

[25] Jacob Joseph. How to treat missing values in your data : Part I, 2016. [Online; accessed December 2, 2017].

[26] Steven M LaValle, Michael S Branicky, and Stephen R Lindemann. On the relationship between classical grid search and probabilistic roadmaps. *The International Journal of Robotics Research*, 23(7-8):673–692, 2004.

[27] Andy Liaw, Matthew Wiener, et al. Classification and regression by randomforest. *R news*, 2(3):18–22, 2002.

[28] Thomas M. Mitchell. *Generative and Discriminative Classifiers: Naive Bayes and Logistic Regression*. draft of February, 2016.

[29] Thomas M. Mitchell. *Machine Learning*. McGraw-Hill, Inc., New York, NY, USA, 1 edition, 1997.

[30] Amanda N Baraldi and Craig K Enders. An introduction to modern missing data analyses. 48:5–37, 02 2010.

[31] Vladimir Nikulin. Driving style identification with unsupervised learning. In *Machine Learning and Data Mining in Pattern Recognition*, pages 155–169. Springer, 2016.

[32] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[33] Saba Arslan Shah and Mehreen Saeed. Predicting purchased policy for customers in allstate purchase prediction challenge on kaggle.

[34] Diveesh Singh. Using convolutional neural networks to perform classification on state farm insurance driver images.

[35] Laurens Van Der Maaten. Accelerating t-sne using tree-based algorithms. *Journal of machine learning research*, 15(1):3221–3245, 2014.

[36] Alice Zheng. *Evaluating Machine Learning Models*. OReilly Media, Inc., 1 edition, 2015.

APPENDIX

Features' Correlation within each category, i.e., `ind`, `reg`, `car`, `calc` are shown as histograms in fig. 6, fig. 7, fig. 8 and fig. 9 respectfully.

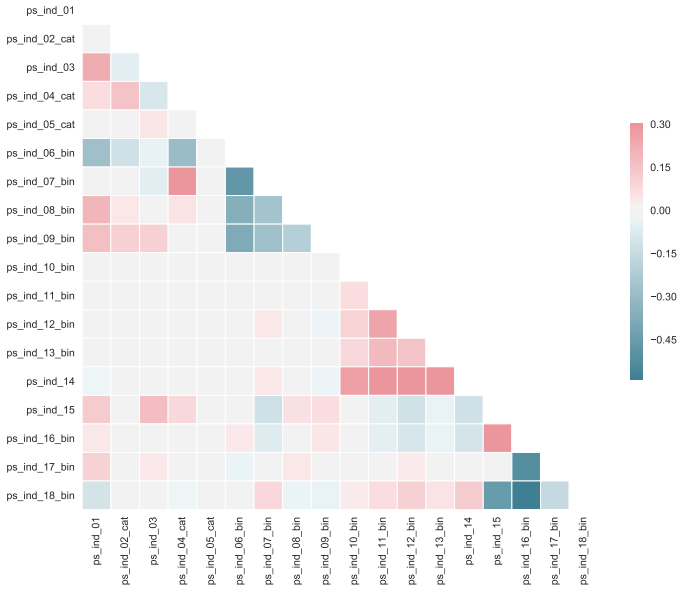

Fig. 6. Correlation for the `ind` Attributes.

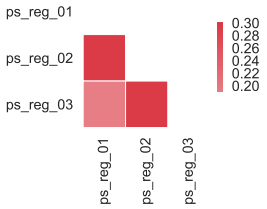

Fig. 7. Histogram for the `reg` Attributes.

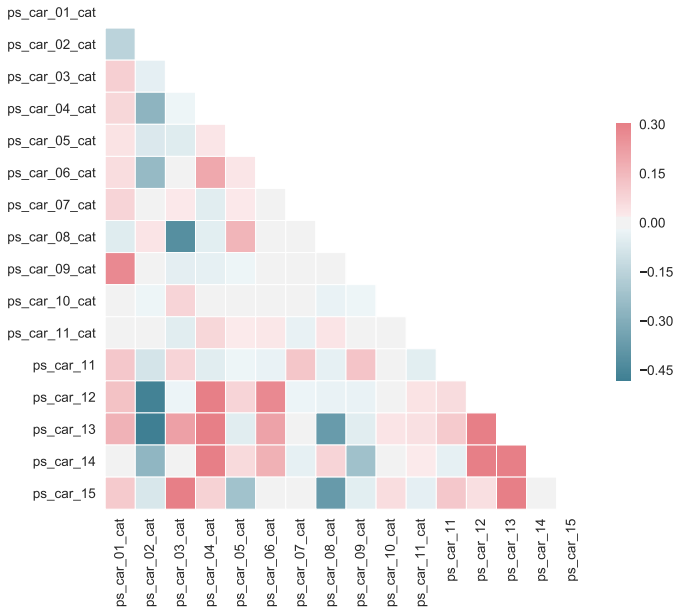

Fig. 8. Histogram for the `car` Attributes.

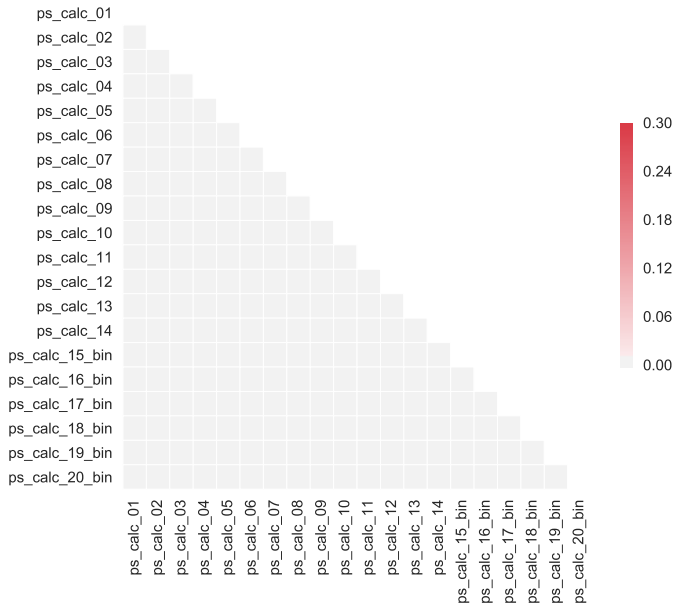

Fig. 9. Histogram for the `calc` Attributes.