Safe Driver Prediction Base on Different Machine Learning Models

Xianzhi Liu, Qingquan Song

***Abstract*—Safe Driver prediction can be very useful for government to require more tests for those with higher potential of causing the traffic accident or ask drivers for regular courses based on the probability. Usually speaking, the number of drivers who have ever caused the traffic accident is often very few, which results in the learning samples to be very unbalanced. In this paper, we implement and test multiple machine learning models, Logistic Regression, Neural Network, SVM (support vector machine) and XGBoost (a scalable machine learning system for tree boosting). Base on Normalized Gini Index Metrics, It turns out that XGBoost performs the best for these highly unbalance data, and simple Logistic Regression outperforms NN and SVM because of the limited computation resource and the highly overlapped data (very hard for linear separation).**

Keywords: traffic safety, ranking, Logistic Regression, Neural Network, SVM, XGBoost

I. Introduction

Driving safety has always been a focus of concern. For the government, it will be great if we can reduce the traffic accident, and one good way is to educate those drivers with a higher probability of causing traffic problems. On the other hand, insurance is a necessity for each driver to reduce the loss as much as possible in case that traffic accidents happen. Statistically speaking, better drivers have less chance of involving into the traffic accidents thus result in less amount of payment from the insurance companies on average. However, nowadays, drivers pay almost the same amount of money for insurance, which seems unreasonable due to their varying driving behaviors. It does not seem fair that a good driver being cautious on the road for years has to pay so much. As described in AA, the inaccuracies in car insurance companys claim predictions are one of the most important reasons that raise the cost of insurance for good drivers and reduce the price for bad ones. It motivates us to build up more accurate models which could potentially reduce the traffic accident, allow the good drivers to tailor their prices and potentially increase the coverage and accessibility of the auto insurances.

II. Previous Work

Generally speaking, this problem is a classification problem, which aims at predicting whether a driver will perform safe driving in the following amount of time (such as a year) based on the previous data. Typical classification algorithms like KNN, Logistic Regression, SVM are the most often used ones. KNN makes use of the nonlinear boundary (either in the original feature space or transformed feature space) of different classes. For the data that is hard to separate in the feature space clearly, KNN depends more on the density of the samples distribution, which doesnt have a meaningful correlation between the labels. Logistic Regression is another traditional classification based on predicting the probability and the logistic function. It is fast and very powerful for those linearly separable dataset. However, it may not be suitable for those problems with nonlinear data. SVM is also a powerful classification problem, and it can work well for some nonlinear problems with a proper nonlinear kernel. However, finding a good kernel is usually very hard. If a suitable kernel cannot be found, SVM may work badly for nonlinear problems.

In recent years, Neural Networks (NN) especially Deep neural networks (DNN) have shown powerful ability in solving those problems which have massive features (input data) and very nonlinear. For these problems, it is usually hard to find proper combinations of features or feature transformations by simply adopting traditional machine learning algorithms. One problem of DNNs is that deep networks require massive computation resource, which may not be accomplished by a personal desktop.

Besides NN, Tree boosting has been shown to give state-of-the-art results on many standard classification benchmarks [2]. XGBoost is one kind of tree boosting algorithms. XGBoost has good scalability in all scenarios. The system runs more than ten times faster than existing popular solutions on a single machine and very easy to scale. The scalability of XGBoost is due to several important systems and algorithmic optimizations, including a novel tree learning algorithm is for handling sparse data; a theoretically justified weighted quantile sketch procedure enables handling instance weights in approximate tree learning. Parallel and distributed computing make learning faster, which enables quicker model exploration. More importantly, XGBoost exploits out-of-core computation and enables data scientists to process hundred millions of examples on a desktop [2]. In this paper, we implement several different classification algorithms including Logistic Regression, SVM, NN, and XGBoost. We compare their results based on the Normalized Gini Index and experimental prove that XGBoost is the most suitable algorithm among all the baselines for the safe driving classification task (especially involved with ranking), and enables fast convergence.

III. Classification Models

A. Logistic Regression

Logistic regression measures the relationship between the categorical dependent variable and one or more independent variables by estimating probabilities using a logistic function, which is the cumulative logistic distribution. It treats the same

set of problems as probit regression using similar techniques, while the latter one using a cumulative normal distribution curve instead. Equivalently, in the latent variable interpretations of these two methods, logistic regression assumes a standard logistic distribution of errors but probit regression assumes a standard normal distribution of errors [3]. The original logistic function is formally defined as:

$$f(x) = \frac{1}{1 + e^{-x}} \tag{1}$$

The value of this function range from 0 to 1, which indicates the probability. Based on this function, we haves the following classification model:

$$f(x) = \frac{1}{1 + e^{-\mathbf{w}^T \mathbf{x}}} \tag{2}$$

Here **w** is the parameter, **x** is the input features.

B. SVM

Support Vector Machine constructs a hyperplane or set of hyperplanes in a high- or infinite-dimensional space, which can be used for classification, regression, or other tasks like outliers detection. Intuitively, a good separation is achieved by the hyperplane that has the largest distance to the nearest training-data point of any class (so-called functional margin), since in general the larger the margin the lower the generalization error of the classifier [4]. After formulated in math, SVM becomes finding the minimum error with the margin as the restriction, which can be solved by Lagrange method. After solving the optimization problem, those vectors with zero margin become the support vectors, used as the classification helper. SVM is shown in Fig. 1

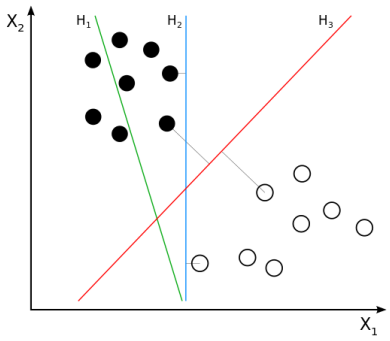

Fig. 1. H_3 separate the two classes with the maximum margin

Support Vector Machine constructs a hyperplane or set of hyperplanes in a high- or infinite-dimensional space, which can be used for classification, regression, or other tasks like outliers detection. Intuitively, a good separation is achieved by the hyperplane that has the largest distance to the nearest training data point of any class (so-called functional margin), since in general the larger the margin, the lower the generalization error of the classifier [4]. Mathematically, SVM aims to maximize the margin among classes based on sample constraints, which can be solved by Lagrange method. Bt solving the optimization problem, vectors with zero margin become the support vectors, which specify the boundaries for classification purpose. The illustration of SVM is shown in Fig. 1

C. Neural Networks

Neural Networks simulate the neural system of animals. Experiments show that cats visual neural cells will react differently for different objects. Thus, people are inspired to design networks with multiple layers, which take the original features as inputs and perform (non)linear transformations through those layers for classification or regression purposes. The outline figure of a basic DNN is shown as follows:

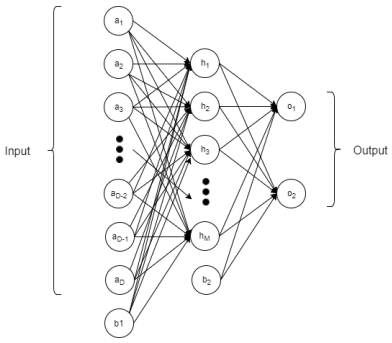

Fig. 2. example of Neural Network

D. XGBoost

XGBoost is trying to build a forest with several trees, each tree is similar to a decision tree but has scores for each leaf. For a given dataset with n examples and m features, it could be mathematically formulated as:

$$D = (\mathbf{x}_i, y_i) \tag{3}$$

A tree ensemble model (shown in Fig. 3) uses K additive functions to predict the output:

$$y_i = \phi(\mathbf{x}_i) = \sum_{k=1}^{K} f_k(\mathbf{x}_i), \; f_k \in F \tag{4}$$

Here, $F = f(x) = w_{q(x)}(q : R^m \rightarrow T, w \in R^T)$ is the space of regression trees (also known as CART). Here q represents the structure of each tree that maps an example to the corresponding leaf index. T is the number of leaves in the tree. Each f_k corresponds to an independent tree structure q and leaf weights w. [2]

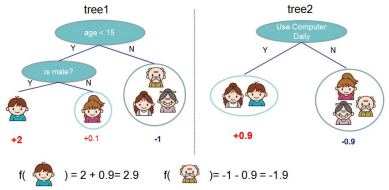

Fig. 3. Tree Ensemble Model. The final prediction for a given example is the sum of predictions from each tree

To learn the set of functions used in the model, we only need to minimize the following regularized objective:

$$L(\phi) = \sum_i l(y_i^*, y_i) + \sum_k \Omega(f_k), \tag{5}$$

where $\Omega(f) = \gamma T + \frac{1}{2}\lambda||\omega||^2$. The first term is a differentiable convex loss function that measures the difference between the prediction and the target. The second term penalizes the complexity of the model.

IV. Experiments

To understand the data better, we plot the data in the original feature space in Fig. 4.

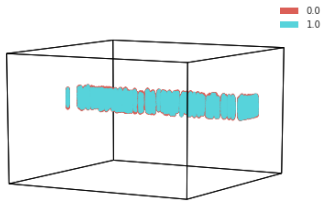

Fig. 4. Data distribution, 0 for class 1, 1 for class 2

As we can see, positive and negative samples overlap heavily with each other. Thus, we did a PCA transformation, and the transformed feature space is shown in Fig. 5

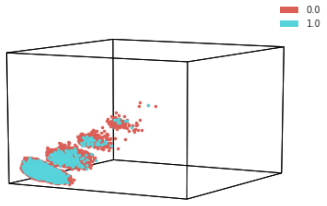

Fig. 5. Data distribution after PCA transformation

The overlap is mitigated, but still serious. If we plot the first three PCA components, we get Fig. 6:

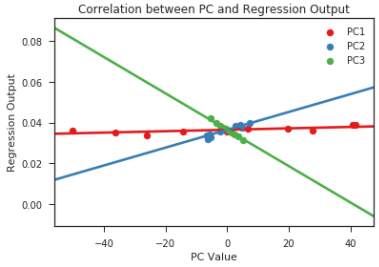

Fig. 6. First three components of PCA

As we can see, there is no obvious correlation between the PCA components and the probability; all the y-values gather round a small value.

For Neural Network, we define such two networks:

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(54, 30)
        self.fc2 = nn.Linear(30, 20)
        self.fc3 = nn.Linear(20, 10)
        self.fc4 = nn.Linear(10, 2)

    def forward(self, x):
        x = x.view(-1, 54)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = F.relu(self.fc3(x))
        x = self.fc4(x)
        return x
```

Fig. 7. Net1

Fig.

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.dp=torch.nn.Dropout(p=0.5)
        self.fc1 = nn.Linear(54, 30)
        self.fc2 = nn.Linear(30, 20)
        self.fc3 = nn.Linear(20, 10)
        self.fc4 = nn.Linear(10, 2)
        self.lru = nn.LeakyReLU(0.1)

    def forward(self, x):
        x = x.view(-1, 54)
        x = F.relu(self.dp(self.fc1(x)))
        x = self.lru(self.fc2(x))
        x = F.relu(self.dp(self.fc3(x)))
        x = self.fc4(x)
        return x
```

Fig. 8. Net2

Fig.

The final results:

Logistic	NN1	NN2	10-depXgboost	SVM
0.24124	-0.00340	-0.00974	0.24388	Too slow

Fig. 9. Results for each model

Here, we use Normalize Gini index as the evaluation metrics.

V. Conclusion

From the results, we find that XGBoost performs the best as we expect. However, complex models such as NNs work very badly, the probable reason for this is that the network is not deep enough and our limited computation resource cannot provide us with an optimal solution. Also, for the heavily overlapped problem, the result shows that SVM may not work well as we cannot easily find a proper kernel.

References

[1] https://www.kaggle.com/c/porto-seguro-safe-driver-prediction/data
[2] T. Chen and C. Guestrin, Xgboost: A scalable tree boosting system, in SIGKDD, 2016.
[3] https://en.wikipedia.org/wiki/Logistic_regression
[4] https://en.wikipedia.org/wiki/Support_vector_machine