Semester Project

Daozhen Lu, Siyi Xian, Shihan Zhang, Yifan Yin[1]*

Abstract
use different technique(k-means, naive bayes, xgboost and knn) to predict class in a huge data set

Keywords
Enormous data set — Preprocessing — Algorithm

[1] *Computer Science, School of Informatics , Computing and Engineering, Indiana University, Bloomington, IN, USA*

Contents

1. Problem and Data Description

Porto Seguro is a auto and homeowner insurance company, wants to improve its model to predicts the probability that a driver will initiate an auto insurance claim next year. We are given a train and a test file and we are asked to build a new model based on the training data and output the target probability for the test data.
The ”target” in training data is the target we want to predict in testing data. Those data from column 3 to 58, (ps_ind_01 to 18, ps_reg_01 to 03, ps_car_01 to 15, ps_calc_01 to 20) are features.

2. Data Preprocessing & Exploratory Data Analysis

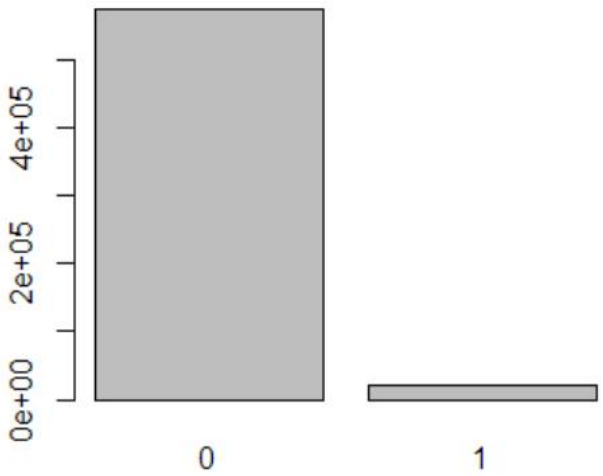

Figure 1. target distribution

From the figure above, we can realize that there are about 3% of data of target is 1, and 97% of data of target is 0.

2.1 Handling Missing Values

There is no text missing value need to be replace in this data matrix, the missing value is -1 in data set. We can get the mean of each column to replace the missing value of that column.

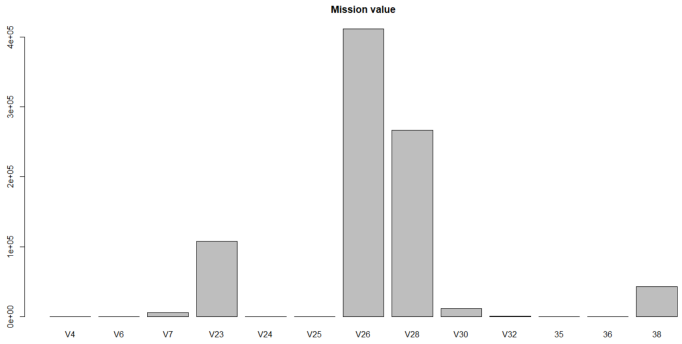

Figure 2. missing value distribution

The figure above shows that the number of missing value in each feature.

2.2 Correlation to reduce dimension

Use Correlation between all columns to find out relation of target and the other variable. Such that we can delete those columns that not so contribute to our goal.

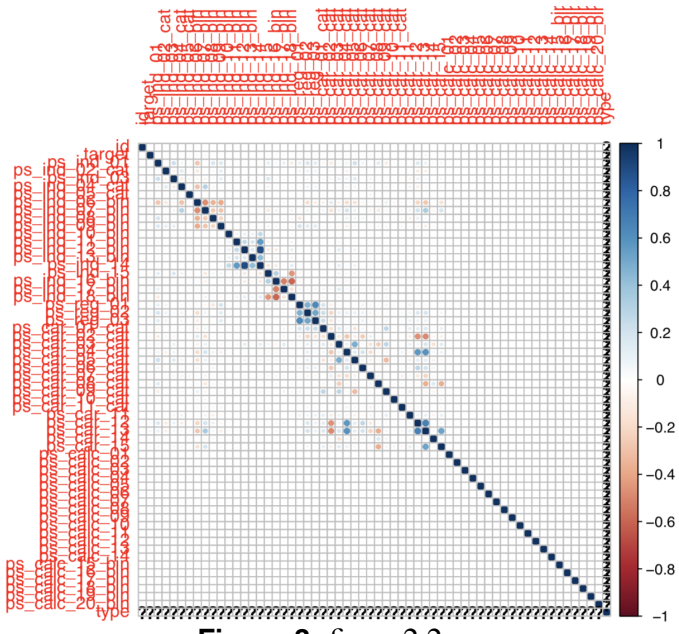

Figure 3. figure2.2

From the figure above, we can find out that ps_calc_01 to 20 is just correlated to itself. So, when any algorithm we run based on this data set can ignore those variable and just consider column 3-38 to predict target.

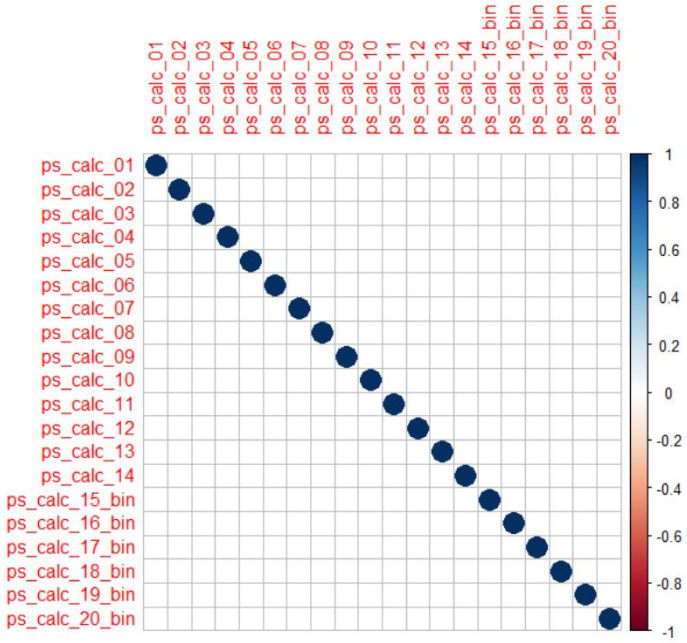

Figure 4. figure39-58

This figure means that the ps_calc_01 to 20 is just related to itself.

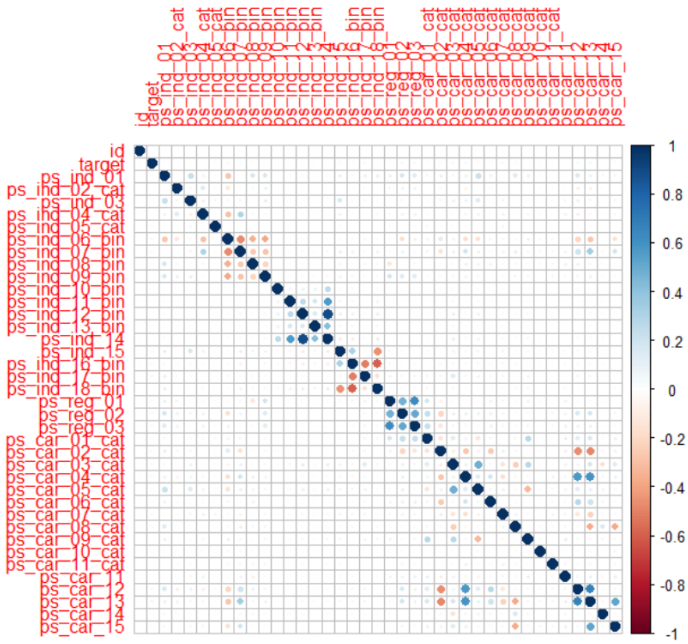

Figure 5. figure1-38

This figure means that column 3 to 38 is some

how related to each other Such that we use they to do our prediction of target.

2.3 K fold cross validation

Use K fold cross validation to randomly build up some train and test with training data. Use

$$error = \frac{\Sigma(predict\,probability[i] - real value[i])}{total number of test}.$$

(Here i is one column of test). Compare each error to find out the best model to do the test prediction.

2.4 Data normalizing

This part is needed for k means.
Since some data distributed from 0 to 1, (for example ps_ind_04) while some data distributed from 1 to more than 10(for example ps_ind_03). If we run k means on they, the distance from the larger distributed data is far more important than those data distributed only in 0 to 1. Such that, we can normalize all of them distributed between 0 to 1. We can just divided each data by the largest value in this column to normalize it.

2.5 PCA reduce dimension

Although PCA cannot help us decreasing dimension of the whole dataset, it can help us partially decided how many clusters should we use in k-means. First of all, the average probability of each cluster should be calculate as the comparators factors. Because, we want all data to be as separate as much. Then we calculate the variance of each set of clusters. Overall, the variance is increasing, but there are still some laws we can find. According to result, we can find that is it likely a curve. Then the turning point in the plot is at around 40 clusters. The reason why we do not choose as much clusters as we want is that the more clusters will let k-means run slowly. Thus, we will choose 40 as our clusters amount. Then we put the original data into k-means in order to avoid the leave out data point in PCA.

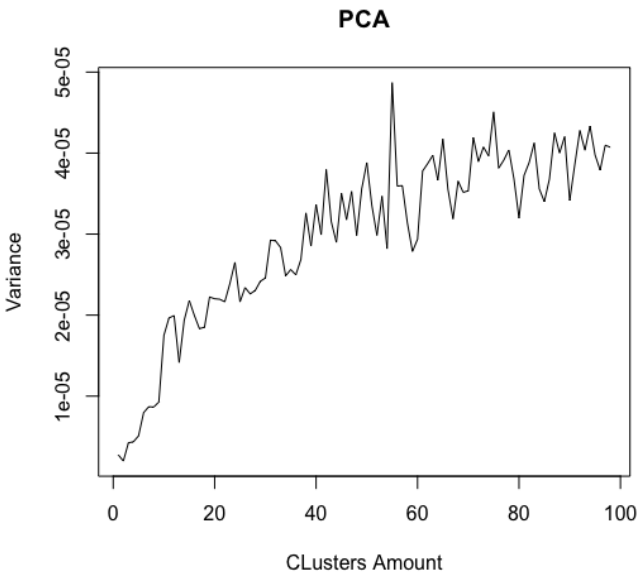

Figure 6. pca determine number of cluster

2.6 Down sampling

Since the percentage of target 0 is too large, we can randomly drop some data with target 0 in the training set, such that the percentage of 1s is higher.

3. Algorithm and Methodology

3.1 K means algorithm with cluster probability

First we can use K means algorithm to cluster each data in training set into k group. In our algorithm, we use k = 100.

Then we focus on the target label of each data. For each cluster, we find out the number of target label 1 and divided it by the total number of data in this cluster. Such that, we can get a probability of 1 in this cluster.

Finally, we can input the test data. Based on training data clusters, we find the cluster of each test data and output the probability of 1 in that cluster as our prediction of the probability of target as 1 for the test data.

3.2 Naive Bayes algorithm

Use Naive Bayes algorithm to output the probability of each test data as 1. Naive Bayes

$$P(B|A) = \frac{P(B) * P(A|B)}{P(A)}$$

this is the basic function we would use in this algorithm.
advantage: Naive bayes is a popular algorithm used in many fields, especially in medical science and cancer detection. It is a very simple yet so powerful algorithm for very large datasets.
disadvantage: Naive bayes works with the assumption of predictor independence, so it can not figure out the relationship between predictors. And it works only with categorical predictors.

3.3 xgboost algorithm

Use xgboost algorithm to output the probability of each test data as 1.
xgboost is more efficient and accurate on classification and regression predictive modeling problems compared to other implementations of gradient xgboost. The eventual model for an xgboost is a forest formed by a set of decision trees by using the gradient tree xging algorithm. In each signal tress inside, we use an Optimization function combined with tree model to determine whether a tree split by a feather or not.

Because our model is based on a forest, so our prediction can be represent as following:

fun.jpg

$$\hat{y}_i = \sum_{k=1}^{K} f_k(x_i), \quad f_k \in \mathcal{F}$$

Figure 7. xgboost function

We use iteration to increase our forest, which means in each iteration we add one more tree to our forest.

$$\begin{aligned}
\hat{y}_i^{(0)} &= 0 \\
\hat{y}_i^{(1)} &= f_1(x_i) = \hat{y}_i^{(0)} + f_1(x_i) \\
\hat{y}_i^{(2)} &= f_1(x_i) + f_2(x_i) = \hat{y}_i^{(1)} + f_2(x_i) \\
&\cdots \\
\hat{y}_i^{(t)} &= \sum_{k=1}^{t} f_k(x_i) = \hat{y}_i^{(t-1)} + f_t(x_i)
\end{aligned}$$

Figure 8. xgboost function

Now the problem is how to build an optimal tree in each iteration when we add it to our forest model. At here, we will accord to an Optimization function to get optimal tree including the weight of each leaf and the tree structure. The Optimization function is like following, which contains two part: an error rate function and a function for Regularization.

$$Obj = \sum_{i=1}^{n} l(y_i, \hat{y}_i) + \sum_{k=1}^{K} \Omega(f_k)$$

Figure 9. xgboost function

Based on our forest model prediction, we can represent the Optimization function to following, because we add a new tree to our forest, so our prediction for a certain datapoint equals to the prediction based on the current forest before this iteration and plus the prediction on the new tree added to our forest.

$$\begin{aligned}
Obj^{(t)} &= \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t)}) + \sum_{i=1}^{t} \Omega(f_i) \\
&= \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \Omega(f_t) + constant
\end{aligned}$$

Figure 10. xgboost function

We can get it's proximate value through apply Tylor formula.

$$Obj^{(t)} \approx \sum_{i=1}^{n}[l(y_i, \hat{y}_i^{(t-1)}) + g_i f_t(x_i) + \frac{1}{2}h_i f_t^2(x_i)] + \Omega(f_t) + constant$$
$$= \sum_{i=1}^{n}[g_i f_t(x_i) + \frac{1}{2}h_i f_t^2(x_i)] + \Omega(f_t) + [\sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t-1)}) + constant]$$

Figure 11. xgboost function

$$g_i = \partial_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)}),$$

Figure 12. xgboost function

$$g_i = \partial_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)}),$$

Figure 13. xgboost function

$$h_i = \partial^2_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)});$$

Figure 14. xgboost function

Then we can ignore the constant part in this formular, then we can get:

$$Obj^{(t)} \approx \sum_{i=1}^{n}[g_i f_t(x_i) + \frac{1}{2}h_i f_t^2(x_i)] + \Omega(f_t)$$

Figure 15. xgboost function

Now we are ready to move the tree expression and tree Regularization to our Optimization formula. Then we can change our expression based on each tree leaf like the j=1 and up to T (means the total leaves in a tree).

$$Obj^{(t)} \approx \sum_{i=1}^{n}[g_i f_t(x_i) + \frac{1}{2}h_i f_t^2(x_i)] + \Omega(f_t)$$
$$= \sum_{i=1}^{n}[g_i w_{q(x_i)} + \frac{1}{2}h_i w_{q(x)}^2] + \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2$$
$$= \sum_{j=1}^{T}[(\sum_{i \in I_j} g_i)w_j + \frac{1}{2}(\sum_{i \in I_j} h_i + \lambda)w_j^2] + \gamma T$$

Figure 16. xgboost function

The next step is to get the optimal solution for our Optimization function which is a quadratic function.

$$W_j^* = -\frac{G_j}{H_j + \lambda}$$
$$Obj = -\frac{1}{2}\sum_{j=1}^{T}\frac{G_j^2}{H_j + \lambda} + \gamma T$$

Figure 17. xgboost function

The last step is that we can use the optimal solution in the above to make decision on splitting or not. One way for doing that is we go through all the attributes and find optimal split point on each attribute then we rank all of them to get our eventual optimal split point. Then using following concept to determine can we do this split.

$$Obj_{split} = -\frac{1}{2}[\frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda}] + \gamma T_{split}$$
$$Obj_{noSplit} = -\frac{1}{2}\frac{(G_L + G_R)^2}{H_L + H_R + \lambda} + \gamma T_{noSplit}$$
$$Gain = Obj_{noSplit} - Obj_{split}$$
$$= \frac{1}{2}[\frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda} - \frac{(G_L + G_R)^2}{H_L + H_R + \lambda}] - \gamma(T_{split} - T_{nosplit})$$

Figure 18. xgboost function

3.4 K nearest neighbor algorithm

First we input test data, and find k nearest neighbor from training data.

From the k training data, we find out the number of target 1 namely m, and get the probability of getting 1 as m /k.

4. Experiments and Results

4.1 Kmeans algorithm

Using Kmeans algorithm.

Use kmeans package in R package

Here we use

$$error = \frac{\Sigma(predict\,probability[i] - real value[i])}{total number of test}.$$

(Here i is one column of test)
We plot the error rate of 5 folds validation data set, and use the model of the lowest error rate to predict the test case.

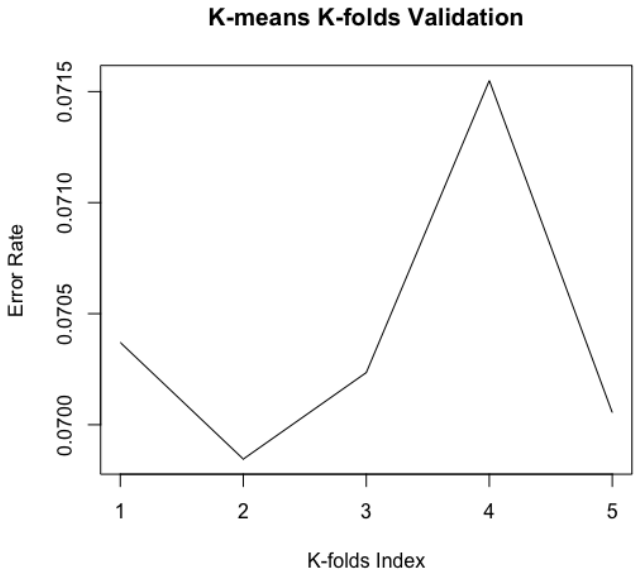

Figure 20. kmeans result head

id	target
0	0.0316524
1	0.0314041
2	0.02905078
3	0.01561659
4	0.02905078
5	0.04087962
6	0
8	0.02221338
10	0.07184092
11	0.02303053
12	0.02828976
14	0.02828976
15	0.05093743
18	0.03974241
21	0.02252664
23	0.04159613
24	0.04357088
25	0.06697055
27	0.02156304
29	0.04201825
30	0.04801623
31	0.04195426
32	0.05054696
33	0.02321933
37	0.04174455

result.jpg
Figure 19. kmeans result head

Figure 21. kmeans kaggle score

Since the target 0 is too much compared with target 1. We use down sampleing idea and randomly dropped half of the data with target 0. However, the kaggle score does not improve by this idea, so we didn't use this algorithm on the other method.

id	target
0	0.16249176
1	0.08084272
2	0.16249176
3	0.13019001
4	0.08084272
5	0.13019001
6	0.16249176
8	0.13019001
10	0.09690122
11	0.11753614
12	0.0218233
14	0.0218233
15	0
18	0.16249176
21	0.13019001
23	0.16249176
24	0.16249176
25	0
27	0.13019001
29	0.16249176
30	0.07550408

Figure 22. kmeans with down sampleing head

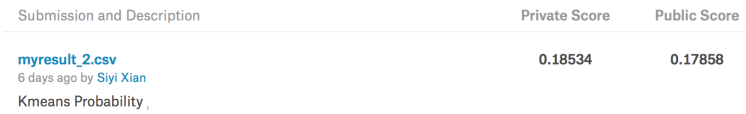

Figure 23. kmeans down sampleing kaggle score

4.2 Naive Bayes

Using naive Bayes algorithm.

Use Naive bayes in R package

Here we use

$$error = \frac{\Sigma(predict\,probability[i] - realvalue[i])}{total number of test}.$$

(Here i is one column of test)
We plot the error rate of 5 folds validation data set, and use the model of the lowest error rate to predict the test case.

id	target
0	2.419588e-03
1	2.089728e-03
2	2.846216e-03
3	2.849143e-04
4	2.838364e-03
5	1.420913e-02
6	5.318435e-03
8	1.673139e-03
10	9.991136e-01
11	4.373157e-02
12	9.753553e-03
14	3.047137e-03
15	2.911046e-01
18	1.461426e-02
21	1.730831e-01
23	1.245190e-03

Figure 24. Naive Bayes result head

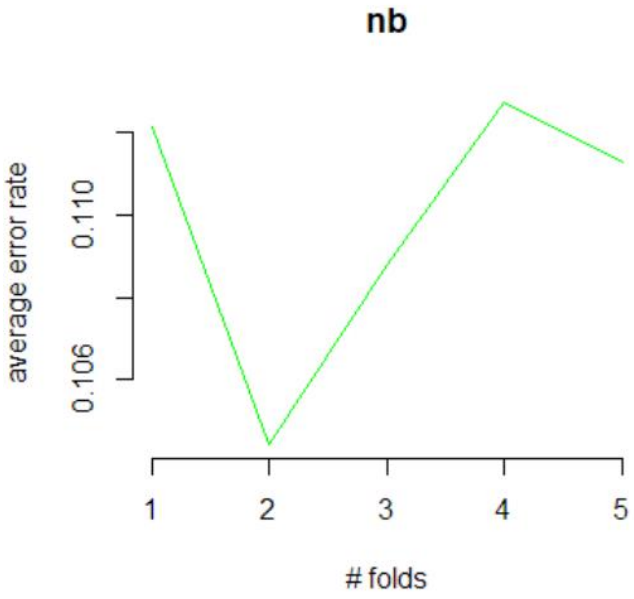

Figure 25. Naive Bayes error rate

myresult.csv 0.23337 0.2257(
6 days ago by yinyifan

Figure 26. Naive Bayes kaggle score

4.3 xgboost

Using xgboost algorithm.

In machine learning, underfitting is unusual and it is normal to see model is overfitting. xgboost has many parameters that control the degree of fitting. In this xgboost experiments, we adjusted the parameters "hundreds" of times to approach best normalization gini score.

Here we use

$$error = \frac{\Sigma(predict\,probability[i] - real value[i])}{total number of test}.$$

(Here i is one column of test)
We plot the error rate of 5 folds validation data set, and use the model of the lowest error rate to predict the test case.

id	target
0	0.04208300
1	0.04671931
2	0.04582400
3	0.02758905
4	0.05761336
5	0.05981418
6	0.03868280
8	0.04806903
10	0.09351771
11	0.08857085
12	0.04832660
14	0.04137227
15	0.06649072
18	0.06680978
21	0.07116555
23	0.04004427
24	0.04064060

Figure 27. xgboost result head

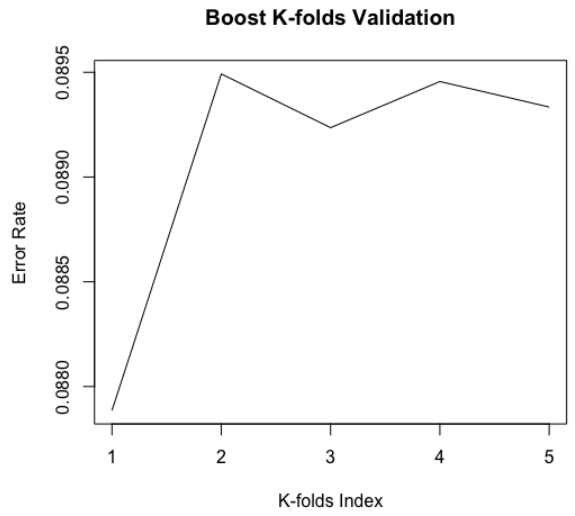

Figure 28. xgboost error for 5 fold

Figure 29. xgboost kaggle score

4.4 KNN

We try to use K nearest neighbor algorithm in this project. We find k neighbor and use the 1 rate of target labels as our prediction.

However, when using KNN we need to calculate the distance from each test points and training points. Thus, the running time is too long and it is hard for us to find a appropriate k value.

5. Summary and Conclusions

When using KNN we need to calculate the distance from each test points and training points.Thus, the running time is too long and it is hard forus to find a appropriate k value. Such that KNN algorithm is not useful here.
By apply down sampleing on k means algorithm, we found that the score we get is smaller that the normal version of k means. So we decided to not apply down sampleing on other algorithm anymore. The kaggle score of k means is 0.19, on the naive Bayes is 0.22, on xgboost is 0.28.

By comparing the score in kaggle, the optimal model for us is xgboost.

By using 5-fold cross-validation with our own error funciton. Error rate of k means algorithm is about 0.07, error rate of naive bayes is about 0.11, the error rate of xgboost is about 0.089.

k means algorithm performed better. Our evaluation function did not work well, because the score from kaggle doesn't match with our evaluation result well.
So finally, we think xgboost algorithm is the best among all algorithm we tried.

boost.jpg

```
    Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
0.008546 0.036063 0.049958 0.056525 0.068602 0.443707
```

Figure 30. summary of xgboost result

References

zhihu user . "Xgboost theorem." zhihu, 23 Apr. 2017.
www.zhihu.com/question/58883125/answer/206813653.

Porto SafeDriveR II - Final — Kaggle.
www.kaggle.com/nonserial/porto-safedriver-ii-final/code.