COMP9417

Machine Learning Project Report

Project Title: Predicting Red Hat Business Value

Kaggle Link: https://www.kaggle.com/c/predicting-red-hat-business-value/overview

Group ID: 102

Submission Date: 1 August 2021

Group Members

zID	Name	Email
z5190639	Yao Lu	z5190639@ad.unsw.edu.au
z5219327	Xinyu Gu	z5219327@ad.unsw.edu.au
z5236652	Zeyu Kong	z5236652@ad.unsw.edu.au
z5027264	Yucheng Liu	z5027264@ad.unsw.edu.au
z5175938	Peixin Wang	z5175938@ad.unsw.edu.au

1. Abstract

This task mainly completes two contents, people data Extraction and business value analysis. The datasets contain a people.csv and activity.csv, so we joined these together to create a single dataset. Create various machine learning models that accurately identifies which customers have the most potential business value for Red Hat based on their characteristics and activities.

2. Introduction

With the development of Read Hat and the large number of user interaction data is collected, it is particularly important to analyze potential business value of customer according to these data. And this is an effective way to improve customer experience, develop customer value and assign resource priorities, which to generate more business and better serve their customers.

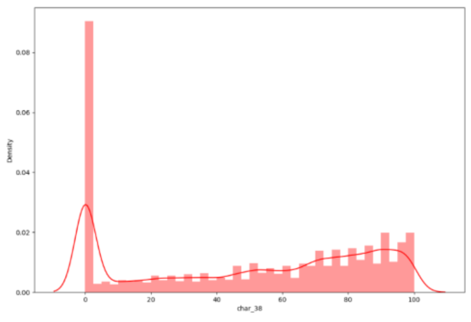
This challenge uses two separate data files are people dataset and activity dataset, use these data to predict the potential business value of a person who has performed a specific activity is the main work of this project. In order to improve the accuracy of prediction, we preprocessed this data, we combined the activity with the people dataset, and generated x_test, x_train, y_test and y_train by changing the values of binary classification features to 0 and 1, dealing with missing values and other preprocessing measures. After the completion of data processing, we used 8 models such as Decision Tree, Gradient Boost Decision Tree, K-Nearest Neighbour, Logistics Regression, Naive Bayes, Neural Network, Random Forests, Support Vector Machine and XGBoost to predict based on these data, then print and analyzed the confusion matrix, AUC, ROC and other quantitative indicators for each algorithm models in this report.

3. Implementation

3.1 Data preprocessing and feature engineering

3.1.1 Load datasets and basic exploration

Load datasets and view original data, We can clearly see that the char_38 is the only numeric feature in people.csv, so it is important to analysis this column, then we drew a distribution map about char_38, It can be seen from the image that he has good prediction ability.



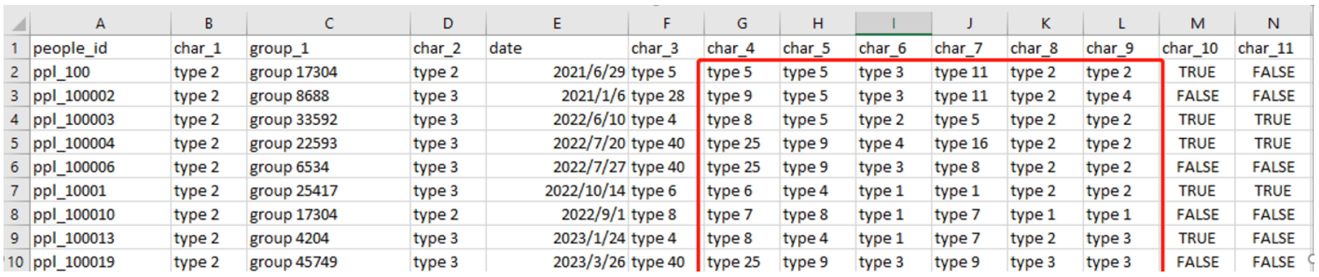
Through the analysis of the above data, we can temporarily draw conclusions about outcome and char_ 38, which is an important indicator of successful prediction.

3.1.2 Convert user id type

From the initial data, we can see that the user ID is of string type, which is not conducive to analysis and calculation. Therefore, we convert this column into an int self-increasing sequence number to ensure uniqueness.

3.1.3 Feature engineering

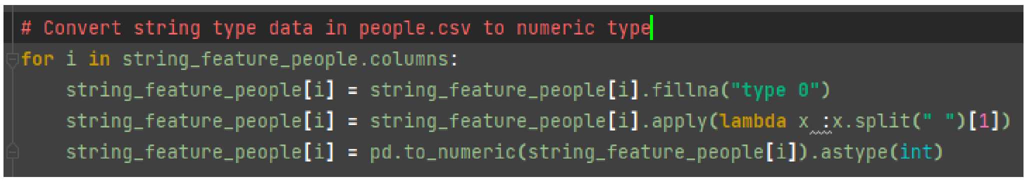
- From char_1 to char_9 we find out that they are string type, although it is a string type, it is still the most important influencing factor of digital information, so we convert these types into digital types.



	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	people_id	char_1	group_1	char_2	date	char_3	char_4	char_5	char_6	char_7	char_8	char_9	char_10	char_11
2	ppl_100	type 2	group 17304	type 2	2021/6/29	type 5	type 5	type 5	type 3	type 11	type 2	type 2	TRUE	FALSE
3	ppl_100002	type 2	group 8688	type 3	2021/1/6	type 28	type 9	type 5	type 3	type 11	type 2	type 4	FALSE	FALSE
4	ppl_100003	type 2	group 33592	type 3	2022/6/10	type 4	type 8	type 5	type 2	type 5	type 2	type 2	TRUE	TRUE
5	ppl_100004	type 2	group 22593	type 3	2022/7/20	type 40	type 25	type 9	type 4	type 16	type 2	type 2	TRUE	TRUE
6	ppl_100006	type 2	group 6534	type 3	2022/7/27	type 40	type 25	type 9	type 3	type 8	type 2	type 2	FALSE	FALSE
7	ppl_10001	type 2	group 25417	type 3	2022/10/14	type 6	type 6	type 4	type 1	type 1	type 2	type 2	TRUE	TRUE
8	ppl_100010	type 2	group 17304	type 2	2022/9/1	type 8	type 7	type 8	type 1	type 7	type 1	type 1	FALSE	FALSE
9	ppl_100013	type 2	group 4204	type 3	2023/1/24	type 4	type 8	type 4	type 1	type 7	type 2	type 3	TRUE	FALSE
10	ppl_100019	type 2	group 45749	type 3	2023/3/26	type 40	type 25	type 9	type 3	type 9	type 2	type 3	FALSE	FALSE

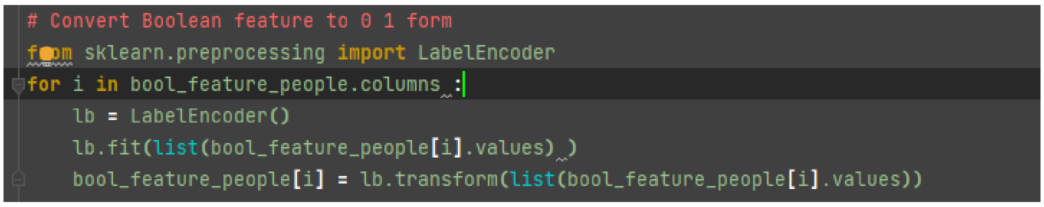
- Then for NAN data, we first delete the number of rows containing empty data. It is found that this leads to low data prediction accuracy, and a large number of data will be deleted. Therefore, we finally choose to set the null value to 0.
- From char_10 to char_37 we find out that they are Boolean type, So we use 0 and 1 instead.
- For date column, We thought this would interfere with the model, so we deleted the date column.

the code of convert string type data to numeric type and change NAN to 0 is:



```
# Convert string type data in people.csv to numeric type
for i in string_feature_people.columns:
    string_feature_people[i] = string_feature_people[i].fillna("type 0")
    string_feature_people[i] = string_feature_people[i].apply(lambda x :x.split(" ")[1])
    string_feature_people[i] = pd.to_numeric(string_feature_people[i]).astype(int)
```

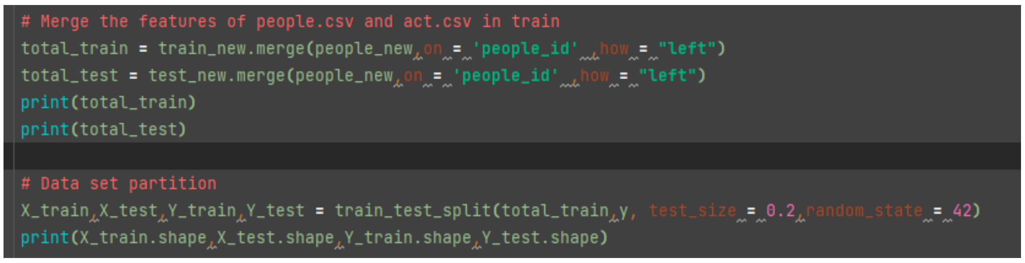
the code of Convert Boolean feature to 0 1 form is:



```
# Convert Boolean feature to 0 1 form
from sklearn.preprocessing import LabelEncoder
for i in bool_feature_people.columns :
    lb = LabelEncoder()
    lb.fit(list(bool_feature_people[i].values) )
    bool_feature_people[i] = lb.transform(list(bool_feature_people[i].values))
```

3.1.4 Merge the two datasets and data set partition

Merge the processed people.csv and act.csv together, and according to the set of test and training, we partition the standard output format, so that we can further put it into the model for calculation, the code and output is:



```
# Merge the features of people.csv and act.csv in train
total_train = train_new.merge(people_new,on = 'people_id' ,how = "left")
total_test = test_new.merge(people_new,on = 'people_id' ,how = "left")
print(total_train)
print(total_test)

# Data set partition
X_train,X_test,Y_train,Y_test = train_test_split(total_train,y, test_size = 0.2,random_state = 42)
print(X_train.shape,X_test.shape,Y_train.shape,Y_test.shape)
```

3.2 Model selection

- Decision Tree

Decision tree is a nonparametric supervised learning method. It can summarize decision rules from a series of characteristic and labeled data and present these rules with the structure of tree view to solve the problems of classification and regression. Decision tree algorithm is easy to understand, suitable for all kinds of data, and has good performance in solving all kinds of problems. Various integration algorithms with tree model as the core are widely used in various industries and fields.

- Gradient Boost Decision Tree

GBDT is an algorithm iterated by multiple decision trees, and the conclusions of all trees are accumulated to make the final answer. Its difference from the decision tree is that each node of the regression tree will get a predicted value, loop each threshold of each feature to find the best segmentation point, and the measurement standard is to minimize the square error.

- K-Nearest Neighbour

KNN is a commonly used model supervised learning method, which includes three parameters: distance measurement, K value and classification decision rules. If the classification task is heavy, the 'voting method' can be used, that is, select the category mark with the most appearance in K samples as the prediction result, and the 'average method' can be used in the regression task, that is, the real value of these K samples outputs the average value of the mark as the prediction result. It can also carry out weighted average or weighted voting based on the distance. The closer the distance, the greater the weight of the sample. Therefore, this is a good method, but it can be used in practice.

- Logistics Regression

Logistic regression is a classification model in machine learning. Because the algorithm is simple and efficient, it is widely used in practice. Because the data set is binary classification, linear regression is used. Because it is a posteriori probability distribution, the likelihood function is designed from the perspective of probability. Logistic model has better robustness to abnormal data than perceptron model.

- Naive Bayes

Bayesian method is based on Bayesian principle and uses the knowledge of probability and statistics to classify the sample data set. The advantage is that the misjudgment rate is very low. The characteristic of Bayesian method is to combine a priori probability and a posteriori probability, which not only avoids the subjective bias of using only a priori probability, but also avoids the over fitting phenomenon of using sample information alone. Bayesian classification algorithm shows high accuracy when the data set is large, and the algorithm itself is relatively simple.

- Neural Network

The principle of neural network is to select and determine the input and output, and then find one or more algorithms to get the output from the input. Then, find a set of data sets with known answers to train the model and estimate W and B. finally, once new data is generated and input into the model, the results can be obtained, and W and B can be corrected at the same time. The whole process requires massive computing, so it has high requirements for computer performance

- Random Forests

Random forest trains an optimal model for each group of resampled data sets, a total of K models. Then M features are randomly sampled from n features of the sample; Then, the optimal learning model is constructed for each sub data set. Finally, for the new input data, the final results are obtained according to K optimal learning models.

- XGBoost

Xgboost is widely used in kaggle competition and many other machine learning competitions. He can often get excellent results. It is similar to GBDT but has better accuracy. The advantage is that xgboost adds a regular term to control the model and reduce the over fitting. For the cost function, it uses a second-order Taylor expansion, adds a linear classifier, samples the data, and processes the missing values.

4. Experimentation

For the experimentation part, for the models listed above, we adjust the relevant parameters through GridSearch CV to obtain better performance parameters, AUC and accuracy were used as evaluation indexes to judge the quality of the prediction model, and a more appropriate model was selected by comparison.

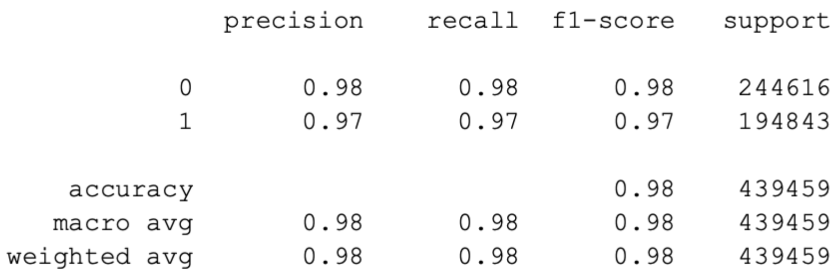
4.1. Model testing and optimization

4.1.1 Decision Tree

in this case, most of the features are discrete variables, so it is more suitable to use the decision tree model. Besides, the time is fast.

4.1.1.1 initial model

Firstly, we used default parameters of model which is "criterion = 'gini', max_depth = None, random_state = None" to train the data, the result we generated



```
              precision    recall  f1-score   support

           0       0.98      0.98      0.98    244616
           1       0.97      0.97      0.97    194843

    accuracy                           0.98    439459
   macro avg       0.98      0.98      0.98    439459
weighted avg       0.98      0.98      0.98    439459
```

The analysis shows that in the test set, the number of Class 0 samples is 244616 and the number of class 1 samples is 194843, so there are some data imbalances; However, in terms of specific quantitative indicators, the accuracy and recall rates of category 0 and category 1 are the same, which are 0.98 and 0.97 respectively, that is, the impact of unbalanced data received by the model is small; At the same time, the average accuracy of the decision tree model is 0.98, the average recall rate is 0.98, the average F1 score is 0.98, and the accuracy of the final model is also 0.98, that is, the overall performance of the model is good. the AUC of the model is 0.9756, close to 1, which is also considerable.

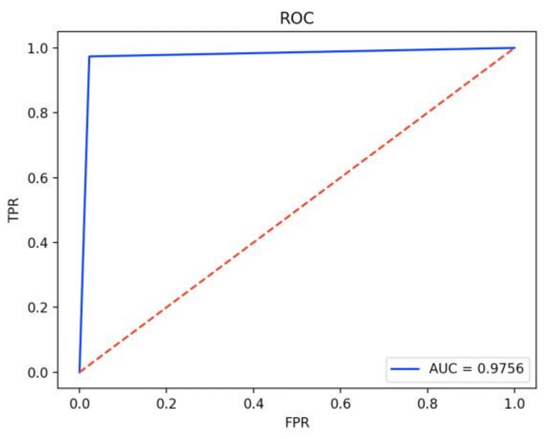
This graph illustrates that the accuracy is around 98% which is a quite good score. This may be because in this case, most of the features are discrete variables, so it is more suitable to use the decision tree model. For the preprocessed data, the decision tree algorithm is used for modeling. Although we do not set max depth, it still does not have over fitting.

the confusion matrix of the model in the test set is as follows:

```
[[239101   5515]
 [  5154 189689]]
```

That is, in 244616 samples of Class 0, the number of samples correctly classified by the model is 239101 and the number of samples incorrectly classified is 5515; Among the 194843 samples in class 1, the number of samples correctly classified by the model is 189689 and the number of samples incorrectly classified is 5154.

The ROC plot we generated:



4.1.1.2 Optimizing model

Given that default parameters often do not provide the best performance, we decide to use GridSearchCV to find the parameters that can generate the best performance of the model. We set cv fold to 10 and set criterion list to ['gini', 'entropy']. Then, we set max depth list to range(1,50). After CV, the parameters shown below

```
The best classifier: {'criterion': 'gini', 'max_depth': 20}
```

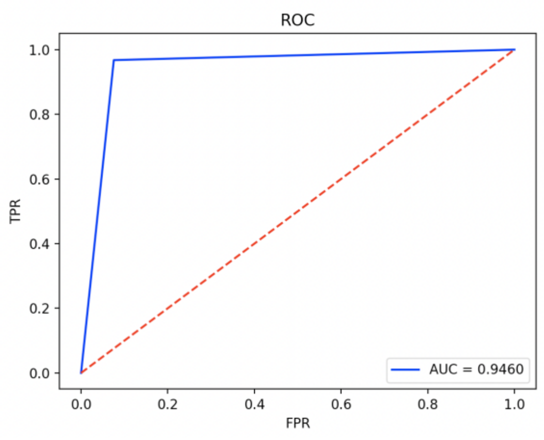
So we use the parameter "criterion=**'gini'**, max_depth=20", the result we generated

```
              precision    recall  f1-score   support

           0       0.97      0.92      0.95    244616
           1       0.91      0.97      0.94    194843

    accuracy                           0.94    439459
   macro avg       0.94      0.95      0.94    439459
weighted avg       0.95      0.94      0.94    439459
```

the confusion matrix of the model in the test set is as follows and the ROC plot, we generated

:

```
[[226136  18480]
 [  6313 188530]]
```



Compare with initial model and optimal model, we find that the accuracy of initial model is higher. In our opinion, the reason is that the phenomenon of overfitting in our data set is not obvious.

4.1.2 Naive Bayes

The time of Naive Bayes classifier is fast. However, the accuracy of Gaussian Naive Bayes, Bounoulli Naive Bayes and multinomial Naive Bayes are all not good. In my opinion, the reason is that Naive Bayes model assumes that attributes are independent of each other, which is often not true in practical application. When the number of attributes is large or the correlation between attributes is large, the classification effect is not good.

4.1.2.1 Gaussian Naive Bayes

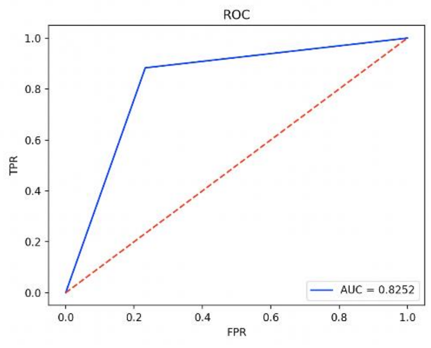
Because Gaussian Naive Bayes classifier does not have parameter, so we use Gaussian Naive Bayes classifier to train the data directly, the result we generated:

```
              precision    recall  f1-score   support

           0       0.89      0.77      0.82    244616
           1       0.75      0.88      0.81    194843

    accuracy                           0.82    439459
   macro avg       0.82      0.83      0.82    439459
weighted avg       0.83      0.82      0.82    439459
```

```
[[192091  52525]
 [ 23103 171740]]
```

From the result, we can find that the accuracy of Gaussian Naive Bayes classifier is not good. As we mentioned before, Gaussian naive Bayes estimates the conditional probabilities for each category under each feature by assuming a Gaussian distribution (i.e., a normal distribution). Therefore, If the feature is numerical, preferably a normally distributed numerical, then use Gaussian Naive Bayes.

The ROC plot we generated



4.1.2.2 Bernoulli Naive Bayes

4.1.2.2.1 Initial model

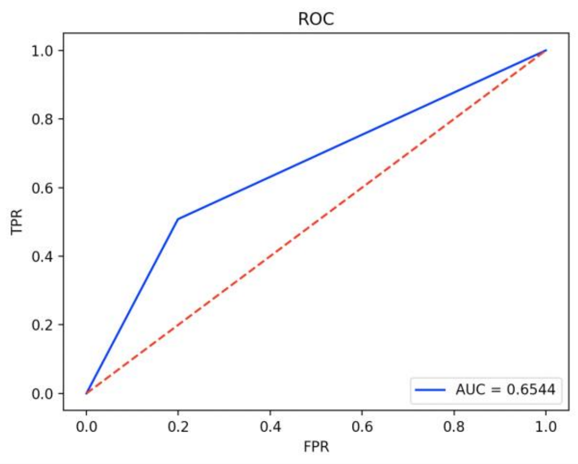
Then, we also use the Bernoulli Naive Bayes classifier with the default parameter (alpha=1) to train the data, the result we generated:

```
              precision    recall  f1-score   support

           0       0.64      0.57      0.60    244616
           1       0.52      0.59      0.55    194843

    accuracy                           0.58    439459
   macro avg       0.58      0.58      0.58    439459
weighted avg       0.59      0.58      0.58    439459
```

```
[[139632 104984]
 [ 79804 115039]]
```

From the result, we can find that the accuracy of Bernoulli Naive Bayes classifier is lower than that of Gaussian Naive Bayes classifier. As we mentioned before, If the feature is binary, then use Bernoulli Naive Bayes.

The ROC plot we generated



4.1.2.2.2 Optimizing model

Given that default parameters often do not provide the best performance, we decide to use GridSearchCV to find the parameters that can generate the best performance of the model. We set cv fold to 5 and set alpha list to [0.001,0.01,0.1,1.5,3,5]. After CV, the parameters shown below

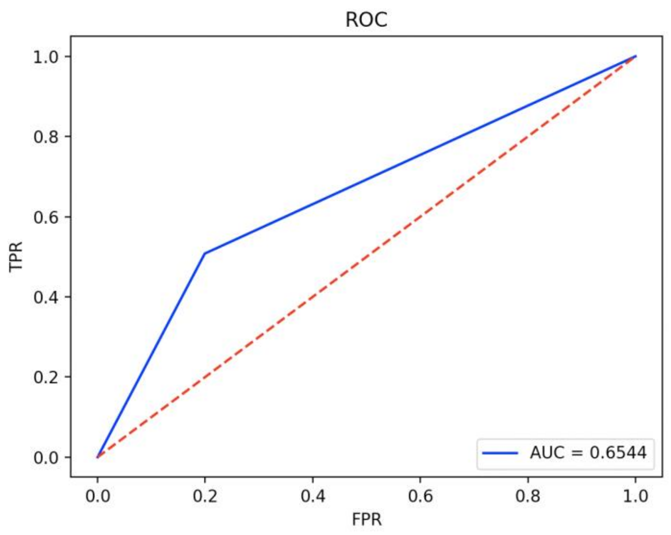
```
The best classifier{'alpha': 0.001}
```

the result we generated

```
              precision    recall  f1-score   support

           0       0.67      0.80      0.73    244616
           1       0.67      0.51      0.58    194843

    accuracy                           0.67    439459
   macro avg       0.67      0.65      0.65    439459
weighted avg       0.67      0.67      0.66    439459
```

```
[[195888  48728]
 [ 95880  98963]]
```

The ROC plot we generated



From these results, we find that the accuracy of new model is higher than initial model. Therefore, we choose new model (alpha = 0.01) as our Bernoulli Naive Bayes classifier.

4.1.2.3 Multinomial Naive Bayes

4.1.2.3.1 Initial model

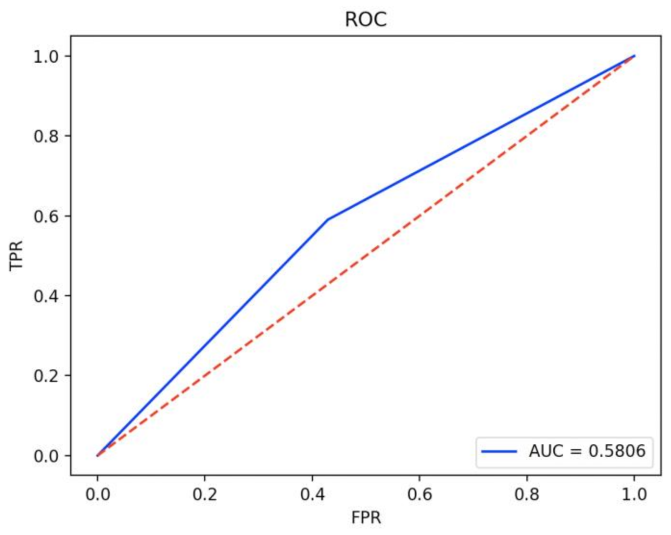
We use the Multinomail Naive Bayes classifier with the default parameter (alpha=1) to train the data, the result we generated

```
              precision    recall  f1-score   support

           0       0.64      0.57      0.60    244616
           1       0.52      0.59      0.55    194843

    accuracy                           0.58    439459
   macro avg       0.58      0.58      0.58    439459
weighted avg       0.59      0.58      0.58    439459
```

```
[[139632 104984]
 [ 79804 115039]]
```

From the result, we can find that the accuracy of Bernolli Naive Bayes classifier is lower than that of Gaussian Naive Bayes classifier and larger than that of Bernoulli Naive Bayes. As we mentioned before, If the feature is categorical, then use Multinomail Naive Bayes.

The ROC plot we generated



4.1.2.3.2 Optimizing model

Given that default parameters often do not provide the best performance, we decide to use GridSearchCV to find the parameters that can generate the best performance of the model. We set cv fold to 5 and set alpha list to [0.001,0.002,0.005,0.01,0.1,1.5,3,5]. After CV, the parameters shown below

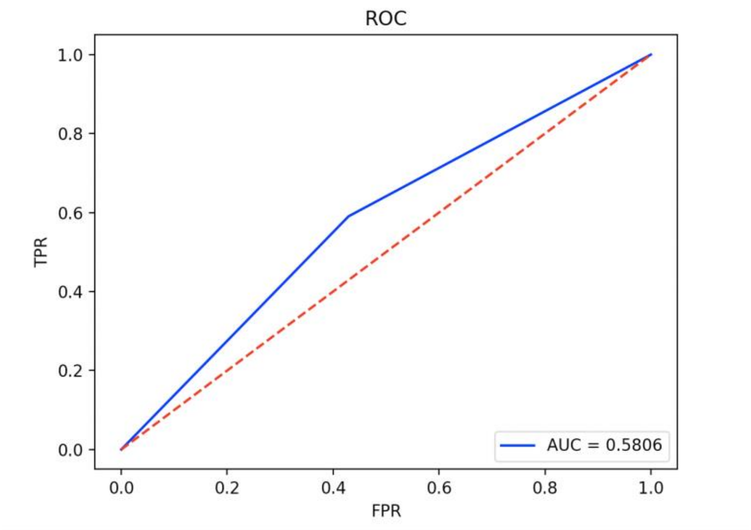
```
The best classifier{'alpha': 1.5}
```

the result we generated

```
              precision    recall  f1-score   support

           0       0.64      0.57      0.60    244616
           1       0.52      0.59      0.55    194843

    accuracy                           0.58    439459
   macro avg       0.58      0.58      0.58    439459
weighted avg       0.59      0.58      0.58    439459
```

```
[[139632 104984]
 [ 79804 115039]]
```

The ROC plot we generated



From these results, we find that the accuracy of new model is the same as initial model. Therefore, we choose optimal model (alpha = 1.5) as our Multinomail Naive Bayes classifier.

In the Naive Bayes, we could find that the accuracy of Gaussian Naive Bayes model is the highest. Therefore, we choose Gaussian Naive Bayes model as our Naive Bayes classifier.

4.1.3 Logistic Regression

4.1.3.1 intial model

```
              precision    recall  f1-score   support

           0       0.87      0.79      0.83    244616
           1       0.77      0.85      0.80    194843

    accuracy                           0.82    439459
   macro avg       0.82      0.82      0.82    439459
weighted avg       0.82      0.82      0.82    439459
```

At the begining, we built the model with default parameter that is 'penalty = l2, solver = liblinear', from the picture above, we can easily find that the accuracy of model is about 0.82, which means the correct predict data account for

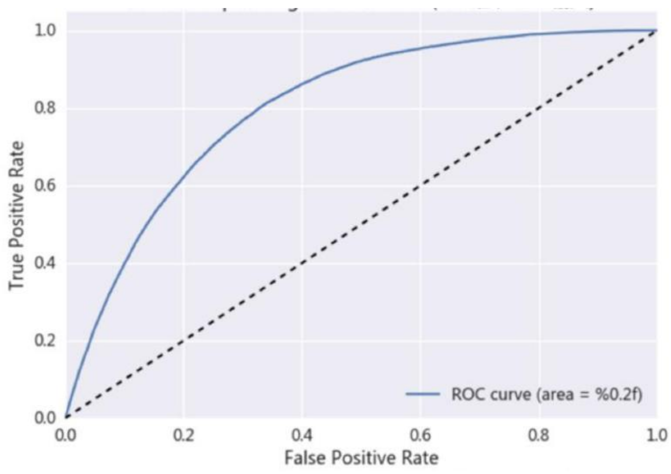
82%. Picture 3.1.3.1.1 clearly show that the precision and f1-score of class 1 is lower than class 0, which may caused by using default parameters and unbalacd preprocessing data.

the analysis shows that in the test set, the number of Class 0 samples is 244616 and the number of class 1 samples is 194843, so there are some data imbalances; However, in terms of specific quantitative indicators, the accuracy and recall rates of category 0 and category 1 are the same, which are 0.79 and 0.85 respectively, that is, the impact of unbalanced data received by the model is bigger; At the same time, the average accuracy of this model is 0.82, the average recall rate is 0.82, the average F1 score is 0.82, and the accuracy of the final model is also 0.82, that is, the overall performance of the model is good. the AUC of the model is 0.82, close to 1, which is also considerable, but it could be better.

the confusion matrix of the model in the test set is as follows:

```
[[194074  50542]
 [ 30140 164703]]
```

That is, in 244616 samples of Class 0, the number of samples correctly classified by the model is 194074 and the number of samples incorrectly classified is 50542; Among the 194843 samples in class 1, the number of samples correctly classified by the model is 164703 and the number of samples incorrectly classified is 30140.

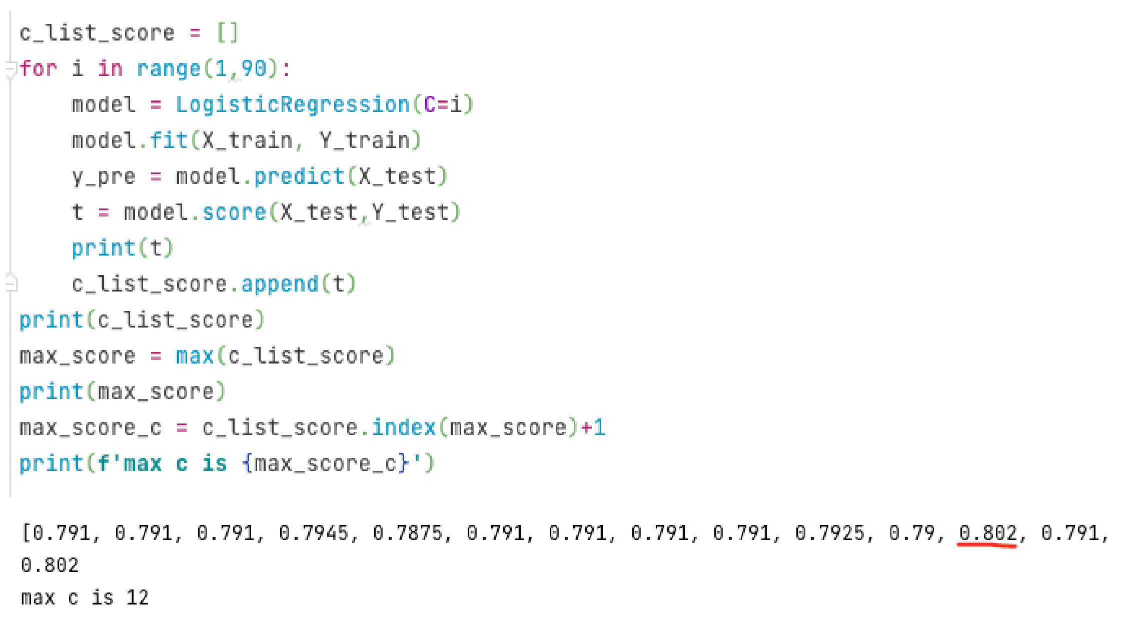


The picture 3.1.3.1.2 above is the plot of ROC, y axis represents True Positive Rate and x axis represents False Positive Rate, which means that it could be better, the AUC index is 0.82. In our opinion, the performance of accuracy could be better if we change some parameters, such as c_value.

4.1.3.2 Optimizing model

Since using default parameter didn't have a great performance of predict, a good way to improve the performance of model is changing parameters,

Firstly we cut some data in order to spent some time. and we set C from 1 to 90 in order to find the best value of 'C'

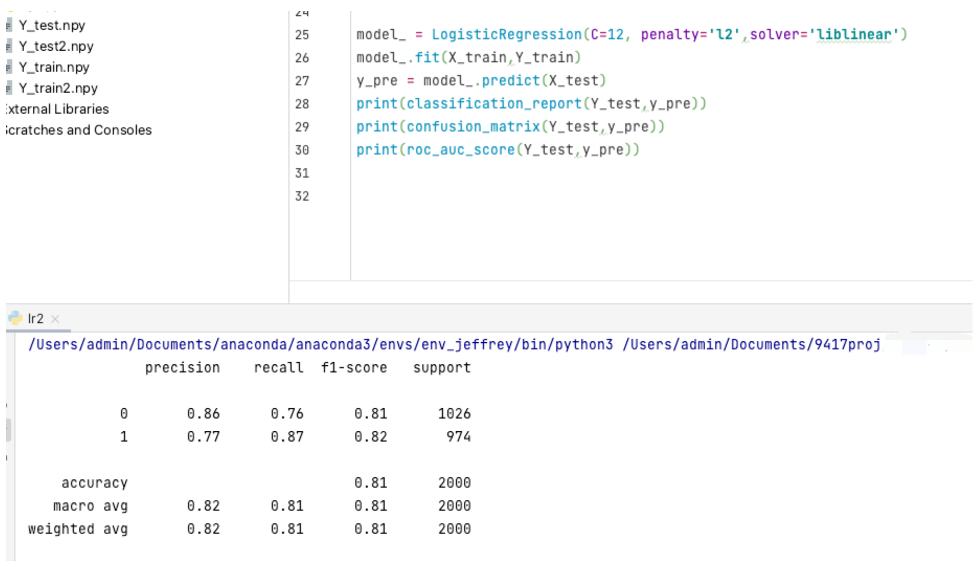


```
c_list_score = []
for i in range(1,90):
    model = LogisticRegression(C=i)
    model.fit(X_train, Y_train)
    y_pre = model.predict(X_test)
    t = model.score(X_test,Y_test)
    print(t)
    c_list_score.append(t)
print(c_list_score)
max_score = max(c_list_score)
print(max_score)
max_score_c = c_list_score.index(max_score)+1
print(f'max c is {max_score_c}')
```

```
[0.791, 0.791, 0.791, 0.7945, 0.7875, 0.791, 0.791, 0.791, 0.791, 0.7925, 0.79, 0.802, 0.791,
0.802
max c is 12
```

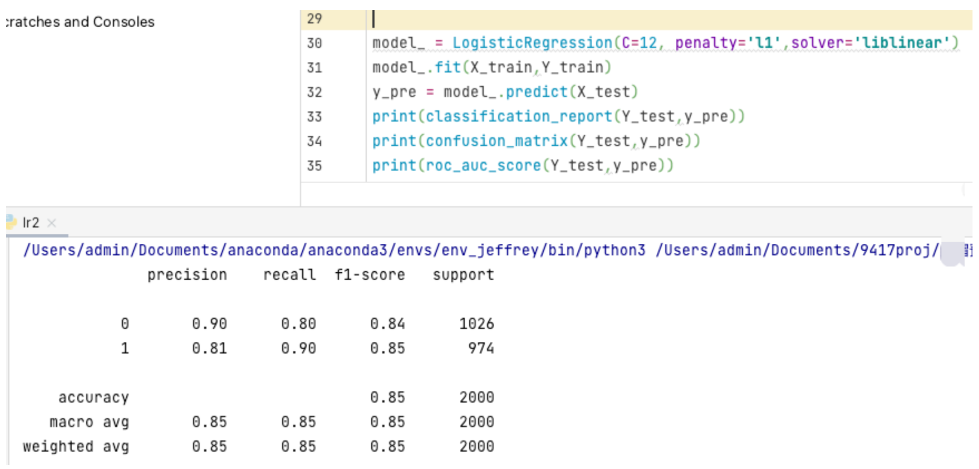
The best C is 12.

Secondly, from the experience of homework2, we know that different penalty would cause different performance, and add solver = 'liblinear'.



```
model_ = LogisticRegression(C=12, penalty='l2',solver='liblinear')
model_.fit(X_train,Y_train)
y_pre = model_.predict(X_test)
print(classification_report(Y_test,y_pre))
print(confusion_matrix(Y_test,y_pre))
print(roc_auc_score(Y_test,y_pre))
```

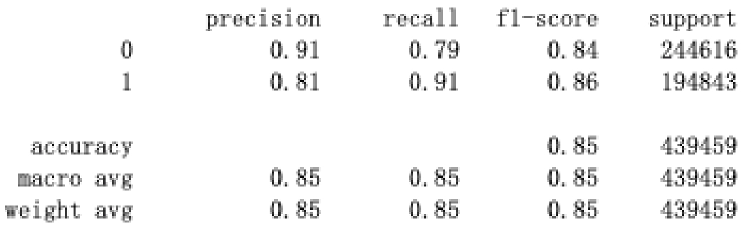
```
/Users/admin/Documents/anaconda/anaconda3/envs/env_jeffrey/bin/python3 /Users/admin/Documents/9417proj
              precision    recall  f1-score   support

           0       0.86      0.76      0.81      1026
           1       0.77      0.87      0.82       974

    accuracy                           0.81      2000
   macro avg       0.82      0.81      0.81      2000
weighted avg       0.82      0.81      0.81      2000
```



```
model_ = LogisticRegression(C=12, penalty='l1',solver='liblinear')
model_.fit(X_train,Y_train)
y_pre = model_.predict(X_test)
print(classification_report(Y_test,y_pre))
print(confusion_matrix(Y_test,y_pre))
print(roc_auc_score(Y_test,y_pre))
```

```
/Users/admin/Documents/anaconda/anaconda3/envs/env_jeffrey/bin/python3 /Users/admin/Documents/9417proj/
              precision    recall  f1-score   support

           0       0.90      0.80      0.84      1026
           1       0.81      0.90      0.85       974

    accuracy                           0.85      2000
   macro avg       0.85      0.85      0.85      2000
weighted avg       0.85      0.85      0.85      2000
```

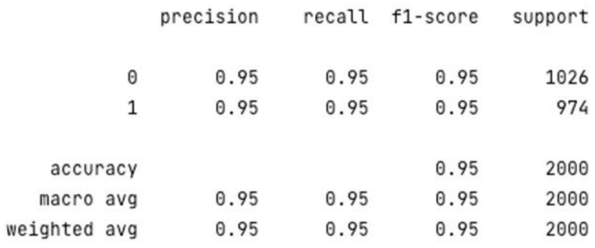
we can clearly find that penalty 'l1' have better performance. Then substitute the best values we found: c, penalty, solver into the complete data set and get the result:



```
              precision    recall  f1-score   support
           0       0.91      0.79      0.84    244616
           1       0.81      0.91      0.86    194843

    accuracy                           0.85    439459
   macro avg       0.85      0.85      0.85    439459
weight avg         0.85      0.85      0.85    439459
```

4.1.4 K-Nearest neighbor classification

4.1.4.1 intial model



```
              precision    recall  f1-score   support

           0       0.95      0.95      0.95      1026
           1       0.95      0.95      0.95       974

    accuracy                           0.95      2000
   macro avg       0.95      0.95      0.95      2000
weighted avg       0.95      0.95      0.95      2000
```

the analysis shows that in the test set, the number of Class 0 samples is 244616 and the number of class 1 samples is 194843, so there are some data imbalances; However, in terms of specific quantitative indicators, the accuracy and recall rates of category 0 and category 1 are the same, which are 0.96 and 0.96 respectively, that is, the impact of unbalanced data received by the model is small; At the same time, the average accuracy of this model is 0.96, the average recall rate is 0.96, the average F1 score is 0.96, and the accuracy of the final model is also 0.96, that is, the overall performance of the model is good. the AUC of the model is 0.958, close to 1, which is considerable and better than Logistic Regression.

the confusion matrix of the model in the test set is as follows:

That is, in 244616 samples of Class 0, the number of samples correctly classified by the model is 235058 and the number of samples incorrectly classified is 9588. Among the 194843 samples in class 1, the number of samples correctly classified by the model is 186244 and the number of samples incorrectly classified is 8579.

Since KNN time complexity is O(N), if the sample is too large, it will lead to a particularly long time. In this project, our data is too large, runing it need to spend lots of time.

4.1.3.2 Optimizing model

Since runing it need to spend at least 30 mins, we decide to cut some training and test data. Firstly, we find the best number for the parameter of n_neigbors.

```
X_train = np.load("X_train2.npy", )
X_test = np.load("X_test2.npy", )
Y_train = np.load("Y_train2.npy", )
Y_test = np.load("Y_test2.npy", )
best_score = []
best_k = -1
# choose best value for n_neighbors

model_ori = KNeighborsClassifier()
model_ori.fit(X_train, Y_train)
y_pre_ori = model_ori.predict(X_test)
print(classification_report(Y_test, y_pre_ori))

for k in range(1,11):
    model = KNeighborsClassifier(n_neighbors=k)
    model.fit(X_train, Y_train)
    y_pre = model.predict(X_test)
    t = model.score(X_test, Y_test)
    best_score.append(t)

best_k = best_score.index(max(best_score))+1
print(best_score)
```

the result is shown below.

```
[0.9655, 0.95, 0.955, 0.948, 0.9495, 0.944, 0.942, 0.9395, 0.932, 0.9315]
```

It's clear that when n_neighbors = 1, there are best score for knn model. So we fit n_neighbors to 1.

Secondly, in order to find the best method of weight, I test different method that is 'uniform' and 'distance'.

```
best_method = []
weight = ['uniform','distance']
for w in weight:
    model = KNeighborsClassifier(n_neighbors=1, weights=w)
    model.fit(X_train, Y_train)
    y_pre = model.predict(X_test)
    t = model.score(X_test, Y_test)
    best_method.append(t)

print(best_method)
```

```
best_method is [0.9655, 0.9655]
```

We found that both 'distance' and 'uniform' have good performance, but if we want to add new parameter 'p', we chose using 'distance' method as weights.

Finally， we set different value of 'p' to find best number of 'p'.

```
p_scores_list = []
for i in range(1,6):
    knn = KNeighborsClassifier(n_neighbors=1, weights='distance',p=i)
    knn.fit(X_train, Y_train)
    y_pre = knn.predict(X_test)
    t = knn.score(X_test, Y_test)
    p_scores_list.append(t)
print(p_scores_list)
```

```
[0.966, 0.9655, 0.9655, 0.964, 0.964]
```

Therefore, the best parameters for knn is:

```
best_score is 0.9655
best n_neighbors is 1
best_method is distance
best p is 1
```

```
              precision    recall  f1-score   support

           0       0.97      0.96      0.97      1026
           1       0.96      0.97      0.97       974

    accuracy                           0.97      2000
   macro avg       0.97      0.97      0.97      2000
weighted avg       0.97      0.97      0.97      2000
```

4.1.5 Random Forest

4.1.5.1 initial model

When many decision trees are randomly generated, then at this time, a single test sample can classify each tree and then select the most probable classification, which is more reliable than the classification ability of a single tree. In addition, one of the advantages of random forest is that it can solve over-fitting to avoid sampling errors as much as possible. The result is shown in the figure below.

```
              precision    recall  f1-score   support

           0       0.98      0.98      0.98    244616
           1       0.97      0.98      0.97    194843

    accuracy                           0.98    439459
   macro avg       0.98      0.98      0.98    439459
weighted avg       0.98      0.98      0.98    439459

 0.9768558596908549
```

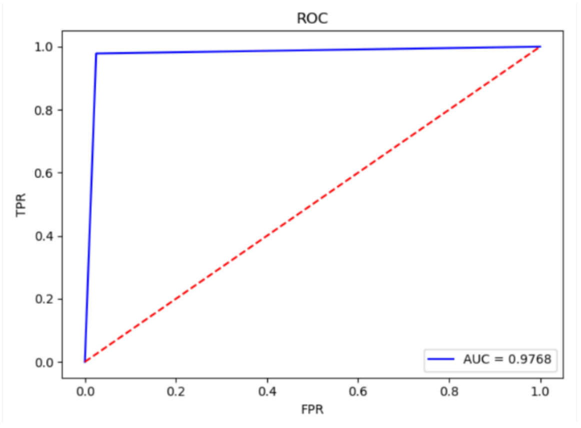
The analysis shows that in the test set, the number of samples in category 0 is 244,616, and the number of samples in category 1 is 194843, so there is a certain data imbalance; but from the specific quantitative indicators, the accuracy and recall of category 0 and category 1 The rates are almost the same, respectively 0.98 and 0.97, that is, the impact of the unbalanced data received by the model is small; at the same time, the average accuracy rate of the decision tree model is 0.98, the average recall rate is 0.98, and the average F1 score is 0.98. The accuracy rate is also 0.98, which means that the overall performance of the model is very good. The AUC of the model is 0.9768, which is close to 1, which is also very impressive.

The confusion matrix of the random forest model is as follows:

```
[[238556   6060]
 [  4192 190651]]
```

That is, among the 244,616 samples of class 0, the number of samples correctly classified by the model is 238556, and the number of samples incorrectly classified is 6060; among the 194843 samples of class 1, the number of samples correctly classified by the model is 190651, which is incorrectly classified The number of samples is 4192.

ROC graph and auc value as below:



4.1.5.2 Optimizing model

When tuning the random forest model, it is usually necessary to change the entropy, n estimator and max features. For entropy, random is generally used when there is a lot of data. The n estimator is the number of decision trees. Generally speaking, the more the number, the better the effect, but when it is increased to a certain value, the result will not be significantly improved, which will only produce more unnecessary complexity and Consumption. And max features is the maximum number of features allowed by the model for a single decision tree.I tried to adjust n estimator but the result cannot be better. It can be seen from the results that the default parameters provide very superior performance, so there is no need to adjust the parameters of the model.

4.1.6 XGB Classifier

4.1.6.1 Initial model

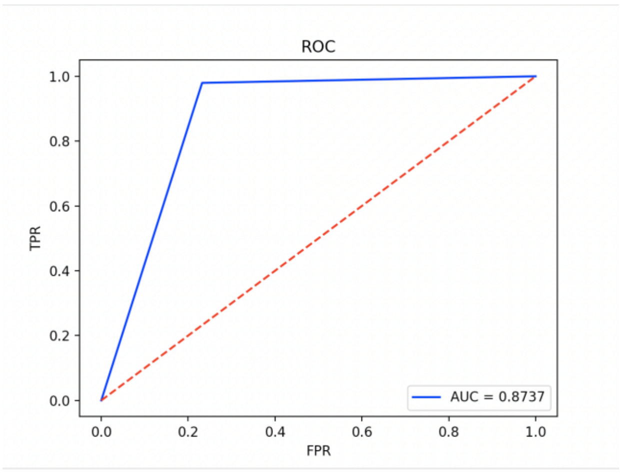
At first attempt, I tried to use the Xgbclassifier model to fit data. Xgbclassifier is a model based on trees. I set n_estimators = 3, learning_rate = 0.1. The classification report shown as below.

```
              precision    recall  f1-score   support

           0       0.98      0.77      0.86    244616
           1       0.77      0.98      0.86    194843

    accuracy                           0.86    439459
   macro avg       0.87      0.87      0.86    439459
weighted avg       0.89      0.86      0.86    439459
```

And confusion matrix as

```
[[187839  56777]
 [  3979 190864]]
```

ROC graph and auc value as below



We can see the model has high precision on category 0 but recall is not very high, on the contract, model has high recall on category 1 but not very high precision. But I think it is ok for this task. Because the task is to predict who is possible to buy the product. So High recall on the person who buys this product is good.

4.1.6.2 Optimizing model

In the preview initialization stage, I just use some default parameters, so the model may have room to improve. For optimization, I try to use the GridsearchCV() function. Set n_estimators = [3, 100], learning_rate = [0.1, 1.2], use Log Loss to minimize error then got new confusion matrix as below

```
              precision    recall  f1-score   support

           0       0.97      0.87      0.92    244616
           1       0.86      0.97      0.91    194843

    accuracy                           0.92    439459
   macro avg       0.92      0.92      0.92    439459
weighted avg       0.92      0.92      0.92    439459
```

AUC as below

```
0.921980397903635
```

After the optimization, the precision of category 1 changed from 0.77 to 0.86 as well as the recall of category 0 changed from 0.87. The model has made a huge improvement. The auc of the model improved from 0.87 to 0.92.

4.1.7 GradientBoostingClassifier

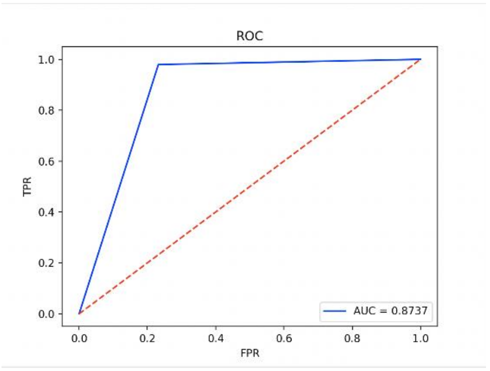
4.1.7.1 initial model

First time I approach this model by setting every parameter to default and get a baseline as in the picture shown below. loss = deviance, learning rate = 0.1, n_estimators = 100.

```
              precision    recall  f1-score   support

           0       0.96      0.79      0.87    244616
           1       0.79      0.96      0.86    194843

    accuracy                           0.87    439459
   macro avg       0.87      0.87      0.87    439459
weighted avg       0.88      0.87      0.87    439459
```

confusion matrix shown and chart as below.

```
[[193737  50879]
 [  8228 186615]]
```



By observing the data set, we can see that data in category 0 is 244616 and in category 1 is 194843. To some extent, the data set is a little imbalanced. But that's ok because recall of 0 is not very low which means the model doesn't category test data to category 0 arbitrarily, so we can consider that the dataset has minimal impact on the model.

4.1.7.2 Optimizing model

Then I try to optimize mode by using GridserchCV function. because train all data takes very long time. So, I use 10000 train and test data to apply gridsearchCV function to find optimal parameters. After training, when parameters list {'learning_rate': 0.3, 'min_samples_split': 2, 'n_estimators': 300} the model get best result. The result shown below.

```
              precision    recall  f1-score   support

           0       0.96      0.85      0.90    244616
           1       0.84      0.96      0.89    194843

    accuracy                           0.90    439459
   macro avg       0.90      0.91      0.90    439459
weighted avg       0.91      0.90      0.90    439459
```

```
[[208381  36235]
 [  8033 186810]]
roc result = 0.905
```

After tuning, model get perfect result comparing default one. For big dataset, we should make n_estimators large. And for learning, we should take it big as small learning rate tend to make model to be overfitting.

4.1.8. Neural Network

4.1.8.1 initial model

I first tested the neural network classifier with default parameters, also called MLP Classifier. The default activation is relu and the solver is adam.

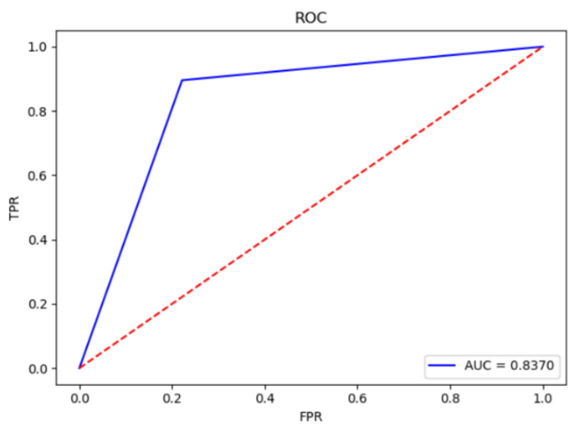
```
              precision    recall  f1-score   support

           0       0.90      0.78      0.84    244616
           1       0.76      0.90      0.82    194843

    accuracy                           0.83    439459
   macro avg       0.83      0.84      0.83    439459
weighted avg       0.84      0.83      0.83    439459
```

```
0.8369569686373306
```

Analysis shows that in the test set, the number of samples in category 0 is 244,616, and the number of samples in category 1 is 194843, so there is a certain data imbalance; but from the specific quantitative indicators, the accuracy and recall of category 0 and category 1 The rates are almost the same, respectively 0.84 and 0.82, that is, the impact of the unbalanced data received by the model is small; meanwhile, the average accuracy of the decision tree model is 0.83, the average recall rate is 0.83, and the average F1 score is 0.83. The final model is accurate The rate is also 0.83, which means that the overall performance of the model is very good. The AUC of the model is 0.8370, which is close to 1, which is also very impressive.

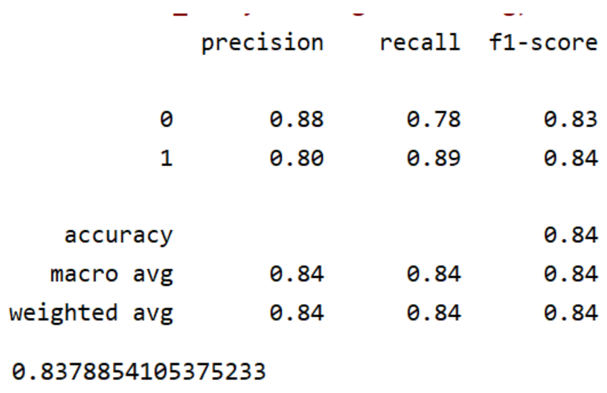
The confusion matrix of the random forest model is as follows and ROC graph and auc value as below:

```
[[190381  54235]
 [ 20336 174507]]
```

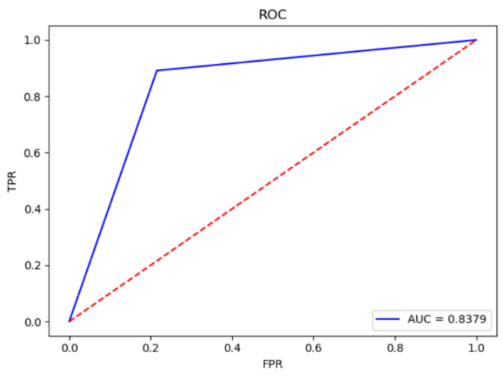


4.1.8.2 Optimizing model

I tried to modify activation, solver and max_iter. When activation is the default relu, the effect is best. When it is logistic or tanh, the effect is very unsatisfactory. Solver is similar to activation. When it is the default adam, the result is the best. Ideally, this is because adam works well in terms of training time and validation scores for relatively large data sets (contains thousands of training samples or more), but for example, lbfgs is more effective for small data sets. The default value of max_iter is 200. I tried it from 20 to 300. When it is 300, the effect is best, ROC graph and auc value as below:



```
              precision    recall  f1-score

           0       0.88      0.78      0.83
           1       0.80      0.89      0.84

    accuracy                           0.84
   macro avg       0.84      0.84      0.84
weighted avg       0.84      0.84      0.84

 0.8378854105375233
```



4.2 Model Evaluation

According to the following table, we should concentrate on the accuracy and AUC value. AUC and accuracy could reflect the quality of the model. It is easy for us to find for the two evaluation indicators, random forest model shows wonderful results, and the running time is not too long. Besides, the result of decision tree is also good. As we all know, the running time of decision tree is also short, and random forest is a classifier that integrating multiple decision trees. By referring to the results of multiple decision trees, the influence of outliers is effectively reduced and over fitting is prevented. In addition, another advantage of random forest is that it could inherite fast prediction of decision tree, when the scale is not large.

Compared with these two models, we can see that K-Nearest Neighbour model also provide a good result. However, the scores of K-Nearest Neighbour model are slight lower than random forest and decision tree.

In the other hand, the abscissa of the ROC curve is false positive rate (FPR) and the ordinate is true positive rate (TPR), so The closer the ROC curve is to the upper left corner, the better the performance of the classifier. Therefore, we choose the random forest model as the best model for this case.

Models		**accuracy**	**precision**	**recall**	**F1-score**	**AUC**
Decision Tree		0.98	0.98	0.98	0.98	0.9756
Naive Bayes	**GNB**	0.82	0.82	0.83	0.82	0.8252
	BNB	0.67	0.67	0.65	0.65	0.6544
	MNB	0.58	0.58	0.58	0.58	0.5806
Gradient Boost Decision Tree		0.92	0.86	0.97	0.92	0.92
K-Nearest Neighbour		0.97	0.96	0.97	0.97	0.9702
Logistics Regression		0.82	0.87	0.79	0.83	0.8273
Neural Network		0.84	0.84	0.84	0.84	0.8379
Random Forests		0.98	0.98	0.98	0.98	0.9768

XGBoost	0.92	0.97	0.87	0.92	0.92

4.2.1 Model evaluation

5. Conclusion

For this assignment, we test several learning models, which include linear model, tree model, k-nearest neighbor classification, Bayes based classifier and neural network. Tree models can get very high accuracy and also excellent runtime benefits. especially Random Forest model, which get highest accuracy at 0.98 and auc 0.977. Beside tree models, knn also have nice accuracy and recall on this task. KNN has nice math explicable properties. Logistic regression, however accuracy and auc perform not as well as tree model and knn, the training time is shortest. For this task neural networks perform slightly better than logistic regression, but running time is much longer than it. So neural network is not suitable for this task. Bayes' base model performance is lowest among all models. From this task we know data processing is also an important part in the training model. For example we need to fill nan data. This is also important.

By doing this project, we can dig further into every model's pros and cons by comparing different models. Moreover we learnt some techniques on tuning data. For example, if dataset is too large and takes long time to train, we can use part of the dataset to apply the gridsearchCV function and find optimal parameters and then use those parameters to train the entire dataset.

6. References

- Breiman, L. 1996. Bagging predictors.. *Machine Learning*, 26: pp. 123–140
- Briem, GJ, Benediktsson, JA and Sveinsson, JR. 2002. Multiple classifiers applied to multisource remote sensing data.. *IEEE Transactions on Geoscience and Remote Sensing*, 40: pp. 2291–2299
- A. W. Moore, D. Zuev, "Internet Traffic Classification Using Bayesian Analysis Techniques", in Proceedings of ACM SIGMETRICS, Banff, Canada, June 2005
- https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
- https://blog.csdn.net/liudongdong19/article/details/107420520
- https://blog.csdn.net/lizz2276/article/details/105931731