

Assignment 3 Data Mining in action

Name: Songwen Hua

Stduent ID: 13419460



I use Pandas, Numpy, Scikit-Learn in Jupyter through Python.

Data Mining Problem

How to solve the problem

Data preprocessing and transformations

- 1 Data loading**
- 2 Examine missing data**
- 3 Data Shape Problem**
- 4 Processing data that is not closely related to quote_flag**
- 5 Dummy variable handling (Change string data to numeric data)**

Classification techniques used and summary of the results and parameter settings

- 1 Decision Tree**
- 2 Random Forest Classifier**
- 3 K Nearest Neighbor Classifier**
- 4 Support Vector Machine Classifier**
- 5 GradientBoostingClassifier**
- 6 Use Grid Search to optimize our random forest**
- 7 XGBoost and Problem After Kaggle Submission**

Final Model contrast

Final Best classifier that I selected

Kaggle Submission

Summary

Data Mining Problem

Our task is to predict the decision of customers' purchase insurance which is quote_flag. We could collect to help insurance companies better specify targeted insurance and improve sales performance based on insurance coverage, personal property information, geographical information, etc.

In the process of processing data, we will encounter many different problems. There are 8 main problems.

1 There is several data that is not relevant to the Quote_Flag

For example, quote_id has nothing to do with quote_flag. This may affect the accuracy of the model.

2 Dataset include many missing values

Data includes a lot of missing values. If missing values are not preprocessed, the classifier cannot be used directly.

3 The overall data contains several properties with too massive amounts.

For instance, the Sales_info5 numbers are enormous, many of them over 10,000

4 Some of the data is strings that are not appropriate for the algorithm model

Much of the Attributes of the given information are of type String. However, several Classifiers require numeric data. Data type mismatch

5 The accuracy of the decision tree model is not high enough

After the completion of preprocessing, the model's accuracy is sometimes not high, and higher accuracy is needed.

6 For data preprocessing purposes, Quote_flag is string data.

7 Some values such as 'Property_info3_P' and 'Personal_info3_ZV' are present in the training data but not in the unknown data.

Problems may be encountered when predicting data.

8 Process unbalanced data

In quote_flag, the values of 0 and 1 are not even, and there are many more 1s than 0s

How to solving the problem

1 There are several data that is not relevant to the Quote_Flag

1.1 Find out which attributes are independent of quote_flag by looking at the data.

1.2 The corr function is used to determine the relationship between other attributes and quote_flag, removing any attributes that are not relevant

2 Dataset include many missing values

2.1 Delete empty columns if there are only dozens of data.

2.2 Missing values, such as personal_info, can be as high as 40%, filling the data with a median.

3 The overall data contains several properties with too massive amounts.

Try to standardize specific data. Z-score is a good option.

4 Some of the data is strings that are not appropriate for the algorithm model

Change all string values to numeric representations. Use the get_dummies function in Pandas.

5 The accuracy of the decision tree model is not high enough

When facing the decision tree, try to use the random function and reduce the branches of the tree to reduce the fitting degree

6 For data preprocessing purposes, Quote_flag is string data.

Converts the data to Date type, and then to numeric type year, month, and day

7 Some values such as 'Property_info3_P' and 'Personal_info3_ZV' are present in the training data but not in the unknown data.

Find the redundant data between the training data and the test data to ensure that the characteristics of the data are the same and help the model make smooth predictions.

8 Process unbalanced data

Use SMOTE function to deal with this issue.

Data preprocessing and transformations

1 Data loading

Firstly, I opened Jupyter through Anaconda and loaded the CSV file.

```
In [5]: # load csv data
Data = pd.read_csv('/Users/huasongwen/Downloads/Assignment 3/Dataset/Assignment3-TrainingData.csv')
Data.head()
```

Out[5]:

	Quote_Id	Quote_Date	Quote_Flag	Field_info1	Field_info2	Field_info3	Field_info4	Coverage_info1	Coverage_info2	Coverage_info3	...	Property_info1	Proj
0	2	14/5/14	0	B	0.9153	935	N	5	2	D	...	N	
1	3	19/6/13	0	J	0.9691	1,165	N	5	22	F	...	N	
2	6	6/2/15	0	B	0.9153	935	N	6	22	D	...	N	
3	9	15/10/14	0	J	0.8793	1,113	N	1	22	F	...	N	
4	10	30/3/15	1	F	1.0101	548	N	13	22	E	...	N	

5 rows × 30 columns

2 Examine missing data

Through the isnull function, we checked the missing values and found that there were many missing values in Personal Info5 and a small number of missing values in Personal Info1 and Property Info1.

And personal_info5 has a missing value of 29788, accounting for 47.599% of the total.

```
In [12]: # Missing number per column
np.sum(Data_a3.isnull(),axis = 0)
```

Out[12]:

Quote_Id	0
Quote_Date	0
Quote_Flag	0
Field_info1	0
Field_info2	0
Field_info3	0
Field_info4	0
Coverage_info1	0
Coverage_info2	0
Coverage_info3	0
Sales_info1	0
Sales_info2	0
Sales_info3	0
Sales_info4	0
Sales_info5	0
Personal_info1	23
Personal_info2	0
Personal_info3	0
Personal_info4	0
Personal_info5	29788
Property_info1	16
Property_info2	0
Property_info3	0
Property_info4	0
Property_info5	0
Geographic_info1	0
Geographic_info2	0
Geographic_info3	0
Geographic_info4	1
Geographic_info5	0

dtype: int64

```
In [11]: # Missing rate of each column
Data_a3.apply(lambda x:sum(x.isnull())/len(x),axis=0)
```

Out[11]:

Quote_Id	0.000000
Quote_Date	0.000000
Quote_Flag	0.000000
Field_info1	0.000000
Field_info2	0.000000
Field_info3	0.000000
Field_info4	0.000000
Coverage_info1	0.000000
Coverage_info2	0.000000
Coverage_info3	0.000000
Sales_info1	0.000000
Sales_info2	0.000000
Sales_info3	0.000000
Sales_info4	0.000000
Sales_info5	0.000000
Personal_info1	0.000368
Personal_info2	0.000000
Personal_info3	0.000000
Personal_info4	0.000000
Personal_info5	0.475999
Property_info1	0.000256
Property_info2	0.000000
Property_info3	0.000000
Property_info4	0.000000
Property_info5	0.000000
Geographic_info1	0.000000
Geographic_info2	0.000000
Geographic_info3	0.000000
Geographic_info4	0.000016
Geographic_info5	0.000000

dtype: float64

Deletes data that entire is empty by dropna function.

```
# Looking for missing values
Data_a3.info

# Delete all empty lines, not one empty line
Data_a3.dropna(how="all")
```

	Quote_Id	Quote_Date	Quote_Flag	Field_info1	Field_info2	Field_info3	Field_info4	Coverage_info1	Coverage_info2	Coverage_info3	...	Property_info1
0	2	14/5/14	0	B	0.9153	935	N	5	2	D	...	N
1	3	19/6/13	0	J	0.9691	1,165	N	5	22	F	...	N
2	6	6/2/15	0	B	0.9153	935	N	6	22	D	...	N
3	9	15/10/14	0	J	0.8793	1,113	N	1	22	F	...	N
4	10	30/3/15	1	F	1.0101	548	N	13	22	E	...	N
...
62575	104296	28/6/13	0	F	0.9919	564	N	4	25	J	...	N
62576	104297	25/1/14	0	F	0.9838	548	N	13	2	E	...	N
62577	104299	24/1/14	0	B	0.9403	935	N	7	22	E	...	N
62578	104300	7/12/14	0	J	0.8870	1,113	N	15	25	G	...	N
62579	104301	8/4/14	0	F	1.0006	548	N	16	22	E	...	Y

62580 rows x 30 columns

Sales_info5
0.82040859
0.0814795
0.88821883
0.10332351
0.91044998
0.86615146
0.81221895
0.82730278
0.7841657
0.44958158
0.14226153
0.78024956
0.13070669
0.8156586
0.85945085
0.75806308
0.32774949
0.0588463
0.92821406
0.13517377
0.85940618

I found that the numbers in Sales_info5 were large, so I chose to standardize the data using the Z-score.

3 Data Shape Problem

First, I imported the unknown data into Jupyter. Then, I preprocessed the unknown data again as above.

```
# load csv data
Data_a3_un = pd.read_csv('/Users/huasongwen/Downloads/Assignment 3/Dataset/Assignment3-UnknownData.csv')

time_value = pd.to_datetime(Data_a3_un['Quote_Date'])
time_value.head()

0    2014-01-25
1    2013-06-18
2    2013-09-24
3    2014-09-15
4    2013-06-18
Name: Quote_Date, dtype: datetime64[ns]

Date = pd.DatetimeIndex(time_value)

Data_a3_un['Quote_Year'] = Date.year
Data_a3_un['Quote_Month'] = Date.month
Data_a3_un['Quote_Day'] = Date.day

# Data Preprocessing
# Data with little relevance
# Looking for missing values

# Delete all empty lines, not one empty line
Data_a3_un.dropna(how="all")

Data_a3_un = Data_a3_un.drop(['Quote_Id', 'Quote_Date'], axis = 1)

# 'Geographic_info3', 'Personal_info5', 'Coverage_info1',
# 'Geographic_info1', 'Sales_info2', 'Property_info2', 'Property_info3'
# 'Personal_info3'

# Field_info4 Y N -> 1 0
Data_a3_un['Field_info4'] = Data_a3_un['Field_info4'].replace('Y', 1)
Data_a3_un['Field_info4'] = Data_a3_un['Field_info4'].replace('N', 0)
```

But I meet the problem of the shape of the data. This problem occurs because the columns in the prediction set and the columns in the training set cannot be precisely the same. By directly printing two copies of data, I find that the total number of columns is different. So, I use the drop function to delete the extra columns. Then I compare again and find that the total number of feature columns is the same.

"Unknown_data" doesn't have "quote_flag"

Data_a3_un						
	Field_info2	Field_info4	Coverage_info1	Coverage_info2	Sales_info1	
0	0.9472	0	8	22	1	
1	0.9919	0	11	22	0	
2	0.9403	0	4	22	0	
3	0.9153	0	12	22	1	
4	0.9485	0	5	22	1	
...	
41716	0.9368	0	2	22	1	
41717	0.9153	0	4	22	1	
41718	0.9219	0	11	22	1	
41719	0.9403	0	12	22	0	
41720	0.9403	0	2	22	1	
41721 rows × 118 columns						

Data_a3				
	Quote_Flag	Field_info2	Field_info4	Covera
0	0	0.9153	0	
1	0	0.9691	0	
2	0	0.9153	0	
3	0	0.8793	0	
4	1	1.0101	0	
...	
62575	0	0.9919	0	
62576	0	0.9838	0	
62577	0	0.9403	0	
62578	0	0.8870	0	
62579	0	1.0006	0	
62580 rows × 119 columns				

4 Processing data that is not closely related to quote_flag

Use corr function to view data and reduce the dimensions and complexity of the data. Some data is unrelated to quote_flag, such as quote_id. Then I delete the data. A negative number means a negative correlation, and a positive number means a positive correlation.

The correlation coefficient is negative.

```
In [35]: Data_a3_corr_matrix = Data_a3.corr ()
Data_a3_corr_matrix["Quote_Flag"].sort_values(ascending =
```



```
Out[35]: Quote_Flag          1.000000
Field_info2          0.138347
Sales_info3          0.114128
Property_info4       0.108637
Sales_info1          0.105319
Property_info5       0.076558
Geographic_info2     0.062983
Coverage_info2       0.036405
Personal_info4       0.029890
Personal_info2       0.025298
Sales_info5         -0.000231
Quote_Id            -0.005891
Geographic_info3    -0.008622
Personal_info5      -0.046120
Coverage_info1     -0.077228
Geographic_info1    -0.186625
Sales_info2        -0.355711
Property_info2              NaN
Name: Quote_Flag, dtype: float64
```

```
# Data with little relevance
Data_a3 = Data_a3.drop(['Quote_Id', 'Quote_Date'], axis = 1)
```

I'm going to delete the middle-hand side number because whether the correlation is positive or negative affects the model, but if a value like quote_flag has nothing to do with quote_id, then I'm going to delete it.

I use the drop function to delete attributes.

5 Dummy variable handling (Change string data to numeric data)

Another significant step in data preprocessing is to change YES to 1 and NO to 0. I choose to use replace function.


```

: # Field_info4 Y N -> 1 0
Data_a3['Field_info4'] = Data_a3['Field_info4'].replace('Y', 1)
Data_a3['Field_info4'] = Data_a3['Field_info4'].replace('N', 0)
Data_a3['Field_info4']

: 0      0
  1      0
  2      0
  3      0
  4      0
  ..
62575    0
62576    0
62577    0
62578    0
62579    0
Name: Field_info4, Length: 62580, dtype: int64

```

```

In [98]: # get_dummies string to number
Data_a3 = pd.get_dummies(Data_a3)
Data_a3.head()

```

```

Out[98]:

```

nfo4_Y	Geographic_info5_CA	Geographic_info5_IL	Geographic_info5_NJ	Geographic_info5_TX
0	1	0	0	0
0	0	0	0	1
0	1	0	0	0
0	0	0	0	1
0	0	0	1	0

get_dummies can turn all strings into numeric data. We can see the result by looking directly at the return value.

```

In [33]: # Looking for missing values
Data_a3.info

# Delete all empty lines, not one empty line
Data_a3.dropna(how="all")

# Fill in missing values (median)
Data_a3.fillna(Data_a3.median(), inplace=True)

```

There are still many missing values in all the data, so we directly use the median to fill in the missing values. I chose the median because there might be some values of 0, 1, or fixed values like 1,2,3,4,5,6... 25. If I use the average, it will be like 13.46, which is not part of 1 to 25, so I use the median to fill in the data.

```
In [5]: time_value = pd.to_datetime(Data_a3['Quote_Date'])
        time_value.head()
```

```
Out[5]: 0    2014-05-14
        1    2013-06-19
        2    2015-06-02
        3    2014-10-15
        4    2015-03-30
        Name: Quote_Date, dtype: datetime64[ns]
```

```
In [6]: Date = pd.DatetimeIndex(time_value)
```

```
In [7]: Data_a3['Quote_Year'] = Date.year
        Data_a3['Quote_Month'] = Date.month
        Data_a3['Quote_Day'] = Date.day
```

Quote_Date value is also a string type data. I use `pd.to_datetime` to change the string to Date Time type. And I could use `Date.year` to get the year of numeric type.

Note:

Unbalanced data handling is not mentioned here because I didn't find out about it until after I uploaded Kaggle, so I decided to write the unstable data handling section in a later section.

Classification techniques used and summary of the results and parameter settings

1 Decision Tree

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn import tree
from sklearn import metrics

# Features: such as buy or not buy searched in addition to Quote_Flag
x = Data_a3.loc[:,Data_a3.columns!='Quote_Flag']
# Labels: such as "salary field or geography information" search Quote_Flag
y = Data_a3.loc[:,Data_a3.columns=='Quote_Flag']
# Data_a3['Quote_Flag']
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.2)
```

```
# Instantiation, recommended to evaluate model objects
clf = DecisionTreeClassifier(splitter="random",random_state=30, max_depth=10, min_samples_leaf=9)
# The maximum depth is set to 10 -----handle Overfitting
# If you pick them all at random, the tree will be deeper, the fit to the training set will be lower,
# Do nothing 80%, random 81%, limit maximum depth 82.9%
# My data is bound to overfit because I have a lot of features

# The model is trained through the model interface
clf = clf.fit(x_train,y_train)

# Extract the needed information through the model interface
result = clf.score(x_test, y_test)
```

result

0.829658037711729

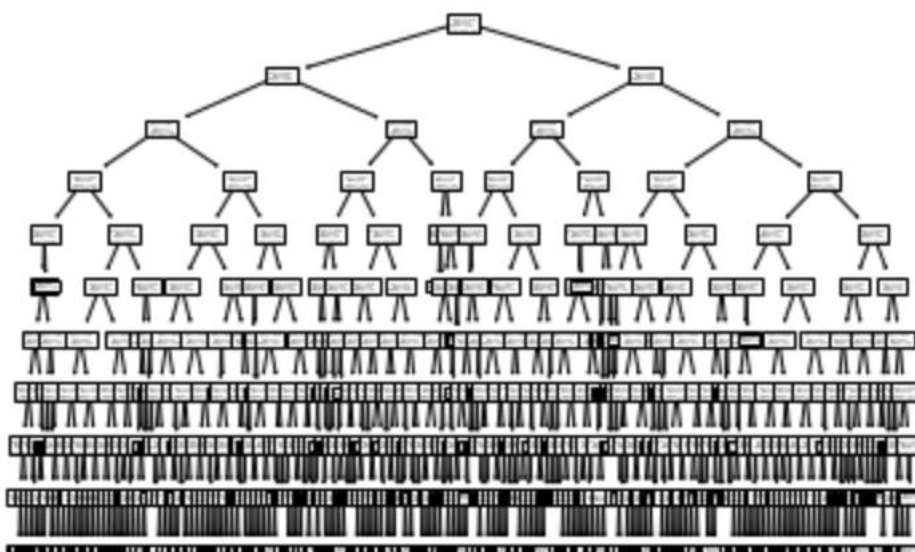
We need to determine the features and labels of the data. The features are all of the data in addition to quote_flag, which is quote_flag. So, we use the loc function's search function to find two values and pass the values to x and y.

Then we created the generated decision tree through DecisionTreeClassifier. After testing, we needed to prune the decision tree to make the decision tree more generalizable. Without fitting, our decision trees tend to perform well on the training set but poorly on the test set. Because the sample data we collected is impossible to be entirely consistent with the overall situation, when a decision tree has the too excellent interpretation of the training set, the rules found must contain the noise of the training sample. And the degree of fit to the unknown data is insufficient.

In addition, I also modified the random option with Splitter and used Random, which is all random regardless of the importance. Because I already know that my data has a lot of characteristics, I just added these.

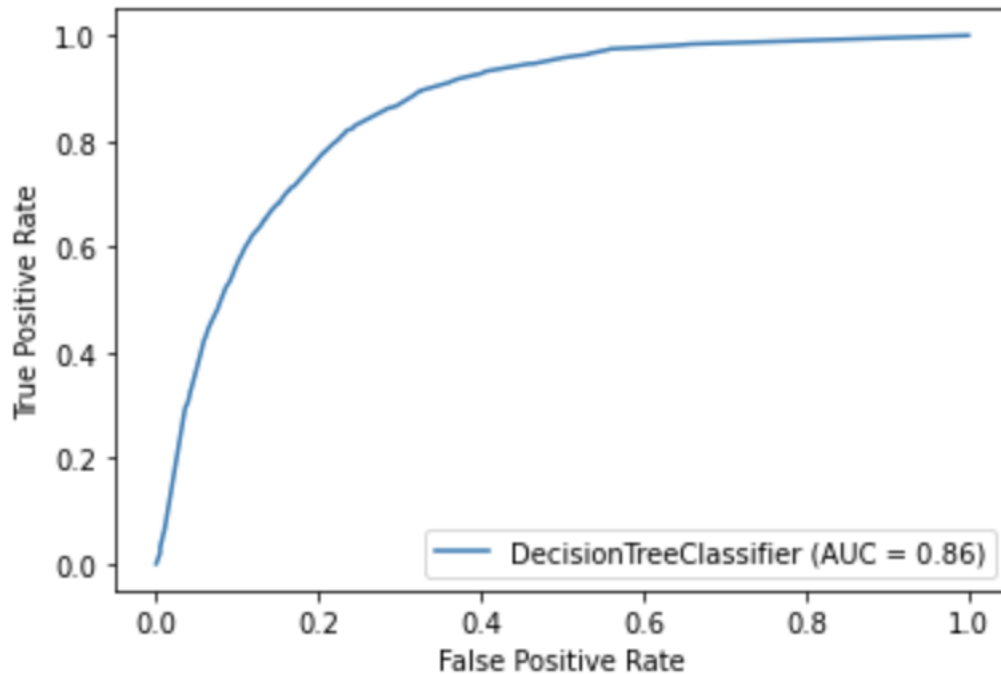
We can see that the final accuracy is 0.829658037711729

```
tree.plot_tree(clf);
```



Then we can actually draw a tree using `plot_tree`.

Finally, we drew ROC curve through `plot_roc_curve` and found that the accuracy was 0.84.



2 Random Forest Classifier

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
x_train = sc.fit_transform(x_train)
x_test = sc.fit_transform(x_test)
```

```
# Random forest

from sklearn.ensemble import RandomForestClassifier # Module to import

#classifier = RandomForestClassifier(n_estimators=20, random_state=0)
#classifier.fit(x_train, y_train.values.ravel())
#y_pred = classifier.predict(x_test)

rfc = RandomForestClassifier(n_estimators=20, random_state=0) # Generating random forest
rfc = rfc.fit(x_train, y_train.values.ravel()) #Training model
score = rfc.score(x_test, y_test) #View accuracy
score

0.8375679130712688
```

First, create a random forest with the RandomForestClassifier. Set `n_estimators=1000`, which means 1000 decision trees in random forest and `random_state=0` to set a random forest.

At the same time, I also set `max_depth=50`, `max_leaf_nodes=2000`, `min_samples_leaf=2`, `min_samples_split=3` to ensure that the data is avoided as far as possible to train a better model

Then the model is trained by the fit func, and the test data set is obtained by SCORE. We can see that the accuracy of the random forest is 0.8495365931607542.

```
In [49]: # Random forest

from sklearn.ensemble import RandomForestClassifier # Module to import

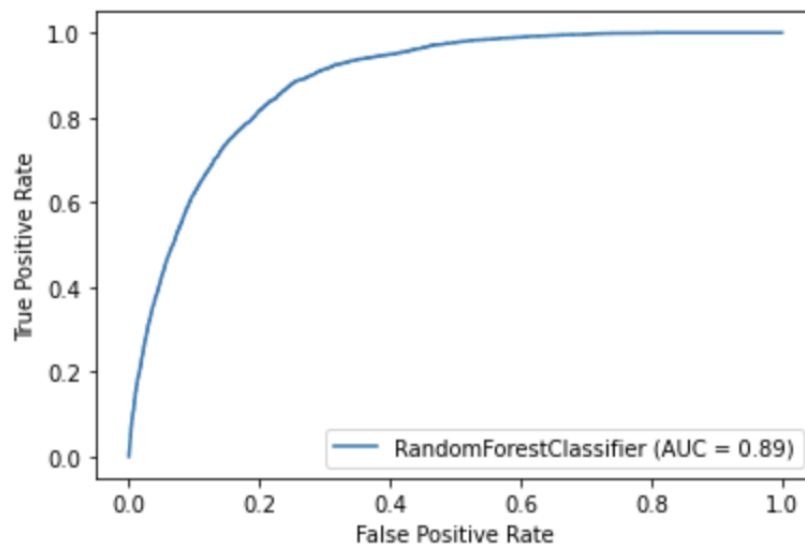
#classifier = RandomForestClassifier(n_estimators=20, random_state=0)
#classifier.fit(x_train, y_train.values.ravel())
#y_pred = classifier.predict(x_test)

# Generating random forest
rfc = RandomForestClassifier(n_estimators=1000, random_state=0, max_depth=50,
                           max_leaf_nodes=2000, min_samples_leaf=2,
                           min_samples_split=3)

rfc = rfc.fit(x_train, y_train.values.ravel()) #Training model
score = rfc.score(x_test, y_test) #View accuracy
score
```

```
Out[49]: 0.8495365931607542
```

```
: display = metrics.plot_roc_curve(rfc, x_test, y_test)
```



Random forest is the optimal solution of a decision tree. We can compare decision trees and random forests to see which algorithm is better. As shown in the figure below, in the case of 10 runs, the accuracy of the random forest

```
: # Decision tree vs Random forest

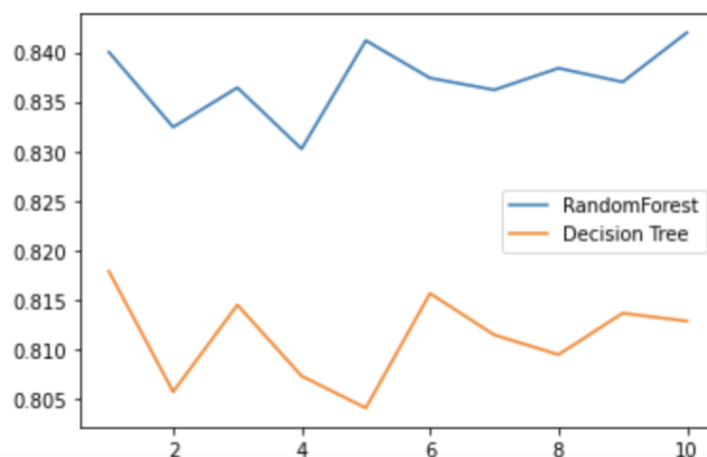
: from sklearn.model_selection import cross_val_score
: import matplotlib.pyplot as plt

rfc = RandomForestClassifier(n_estimators=25)
rfc_s = cross_val_score(rfc, x_train, y_train.values.ravel(), cv=10)

clf = DecisionTreeClassifier()
clf_s = cross_val_score(clf, x_train, y_train, cv=10)

plt.plot(range(1, 11), rfc_s, label = "RandomForest")
plt.plot(range(1, 11), clf_s, label = "Decision Tree")

plt.legend()
plt.show()
```



is all above the decision tree, so the random forest is a relatively better choice.

3 K Nearest Neighbor Classifier

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=10)
knn.fit(x_train, y_train.values.ravel())

score = knn.score(x_test, y_test) #View accuracy
score
```

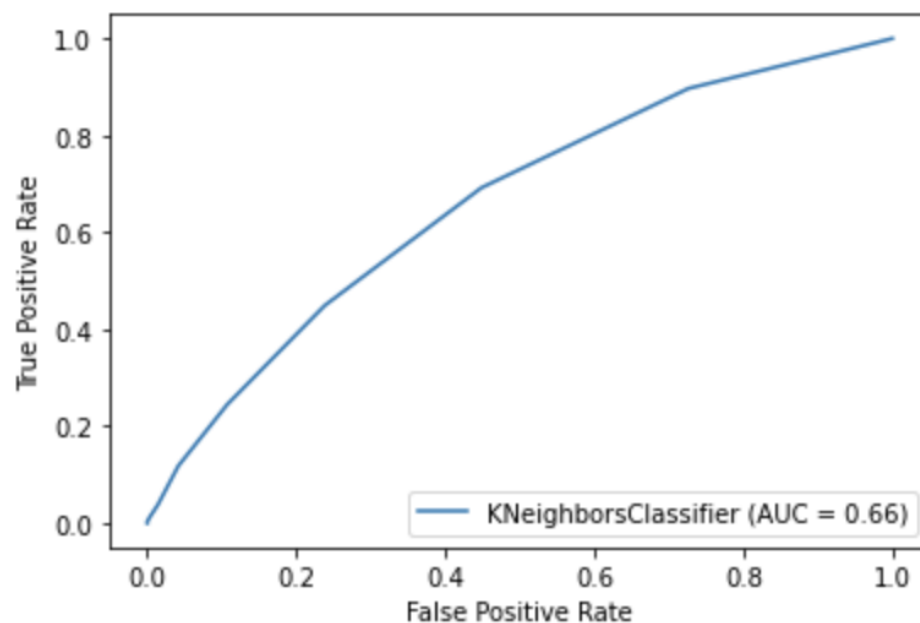
0.8080057526366251

I used KNeighborsClassifier to create the KNN algorithm, set the neighbor to 10, and then trained the model through FIT. And then we get the result by score. The product is 0.80800575. However, the performance in the ROC curve is not very good, only 0.66, so we try not to consider this model.

```
score
```

```
0.8080057526366251
```

```
display = metrics.plot_roc_curve(knn, x_test, y_test)
```



4 Support Vector Machine Classifier

```
x_train, x_test, y_train, y_test = train_test_split(x, y, random_state = 20000)
```

```
from sklearn.svm import LinearSVC
clf = LinearSVC(C=200, loss="hinge", random_state=42)
clf.fit(x_train, y_train.values.ravel())
score = clf.score(x_test, y_test)
score
```

```
0.8069670821348673
```

I also use a LinearSVC call to create a support vector machine; I set $c=200$, $random_state=42$. When the data is noisy, $loss="hinge"$ often works better than the default setting. So, I used $loss="hinge"$.

5 GradientBoostingClassifier

```
import pandas as pd
from sklearn import metrics
from sklearn.ensemble import GradientBoostingClassifier

gbm0 = GradientBoostingClassifier(random_state=1000, subsample=0.7)
gbm0.fit(x_train, y_train.values.ravel())
score = gbm0.score(x_test, y_test)
score

0.8476829658037712
```

Gradient boosting trees are also made up of decision trees, but they are different from random forests in many ways. These trees are regression trees. That means that the model has some scoring function, not just classification. Each tree of the gradient lift tree builds on the previous three.

Subsample represents the subsample, and its value is between [0,1]. Choosing less than one can reduce the variance, that is, prevent overfitting, but it will increase the deviation of sample fitting, so the value cannot be too low, so it is between [0.5, 0.8]

6 Use Grid Search to optimize our random forest

So far, the performance of random forest is the best, so we use random forest for further parameter tuning, automatic parameter tuning we can use grid search.

```
# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 1000, stop = 2000, num = 100)]
n_random_state = [0, 10, 20, 30, 40]
#max_depth=50, max_leaf_nodes=2000, min_samples_leaf=2, min_samples_split=3
```

```
# Create the random grid
param_grid = {'n_estimators': n_estimators,
              'random_state': n_random_state}
print (param_grid)
```

```
{'n_estimators': [10, 20], 'random_state': [0, 10, 20, 30, 40]}
```

```
rf_Model = RandomForestClassifier()
```

```
rf_Model = RandomForestClassifier()
```

```
from sklearn.model_selection import GridSearchCV
rf_Grid = GridSearchCV(estimator = rf_Model, param_grid = param_grid, cv = 3, verbose = 2, n_jobs = 4)
```

```
rf_Grid.fit(x_train, y_train)
```

```
Fitting 3 folds for each of 10 candidates, totalling 30 fits
```



```
[Parallel(n_jobs=4)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=4)]: Done 30 out of 30 | elapsed: 7.5s finished
/Users/huasongwen/opt/anaconda3/lib/python3.8/site-packages/sklearn/model_selection/_search.py:765: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples,), for example using ravel().
  self.best_estimator_.fit(X, y, **fit_params)

GridSearchCV(cv=3, estimator=RandomForestClassifier(), n_jobs=4,
             param_grid={'n_estimators': [10, 20],
                          'random_state': [0, 10, 20, 30, 40]},
             verbose=2)

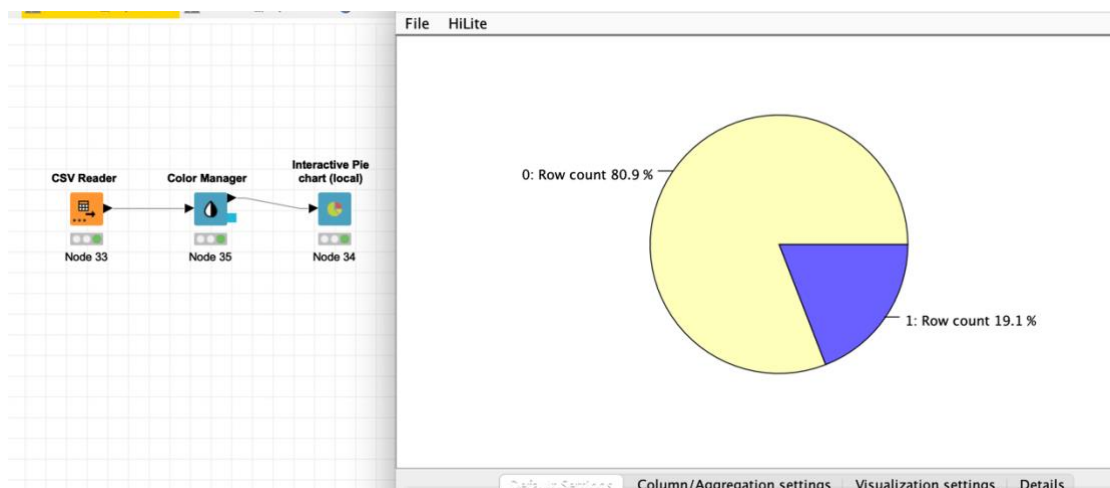
score = rf_Grid.score(x_test, y_test)
score
0.8441594758708852
```

The idea of grid search is that I can customize the values of the parameters, and the machine can automatically adjust the parameters to get the best combination. However, grid search's automatic parameter adjustment function also has a drawback: the training time is too long. In many cases, I need to train for several hours.

7 XGBoost and Problem After Kaggle Submission

When I passed the above best-performing random forests into Kaggle, my prediction set performed poorly. So, I thought it might be because of the imbalanced data problem.

```
0    50625
1     11955
Name: Quote_Flag, dtype: int64
```



Knime shows that 19.1% of the quote_flag data is 1, and 80.9% of the quote_flag information is 0.

With classification_report, I can see how 1 and 0 perform in the test set.

It may be that there are more data in unknown data. 1. So, I decided to conduct unbalanced processing on the data.

```
data_1 = Data_a3[Data_a3["Quote_Flag"]==1]
data_0 = Data_a3[Data_a3["Quote_Flag"]==0]
```

First, I split the quote_flag data into a 0 and a 1.

```
In [25]: from sklearn.metrics import classification_report
print(classification_report(clf.predict(x_test),y_test))
```

	precision	recall	f1-score	support
0	0.90	0.89	0.90	20457
1	0.55	0.57	0.56	4575
accuracy			0.83	25032
macro avg	0.72	0.73	0.73	25032
weighted avg	0.84	0.83	0.84	25032

```
In [30]: test = pd.concat([data_1[:2000],data_0[:2000]])# Take 2000 ones and 2000 zeros
test_data = test.iloc[:,1:] # features
test_labels = test.iloc[:,0] # labels
```

```
In [64]: train = pd.concat([data_1[2000:],data_0[2000:]])
train_data = train.iloc[:,1:]
train_labels = train.iloc[:,0]
```

I chose to take 2000 of them as my test set and the rest of them as my training set.

```
from imblearn.over_sampling import SMOTE
# Due to the uneven distribution of data, oversampling is carried out
train_data, train_labels = SMOTE().fit_resample(train_data,train_labels)
```

Then inbalance the data through the SMOTE function.

```

from xgboost import XGBClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import train_test_split, KFold
from sklearn import metrics
import warnings
warnings.filterwarnings("ignore")
XGB = XGBClassifier()
learning_rate = [0.01, 0.05, 0.1] # Learning rate
n_estimators = [300, 500, 700]
max_depth = [4, 6, 10]
# Convert to dictionary format
param_grid = dict(learning_rate = learning_rate, n_estimators = n_estimators, max_depth = max_depth)

# Grid search
clfcv = GridSearchCV(estimator=XGB, param_grid=param_grid,
                    scoring='r2', cv=10)

clfcv.fit(train_data, train_labels)

# Save optimal parameters
best_parameters = clfcv.best_estimator_.get_params()
print(best_parameters)

```

Then I try another model ----XGBoost. XGBoost is a promotion func, and the basic classification model is the decision tree.

First, I Create XGBClassifier func. Then I tried to modify learning_rate, n_estimators, and max_depth. And set different values so that grid_search automatically adjusts the parameters. Then save the best parameters. Once again, import the best parameters into XGBClassifier. After that, predict the accuracy of test values. Finally, the evaluation indexes are obtained through Classification_Report again.

```

In [*]: end = pd.read_csv("/Users/huasongwen/Downloads/Assignment 3/Dataset/Assignment3-UnknownData.csv")
end["Quote_Flag"] = xgb_model.predict(data_pred)

In [*]: end[["Quote_Id", "Quote_Flag"]].to_csv("/Users/huasongwen/Downloads/Assignment 3/Dataset/t1.csv", index=None)

```

Finally, the data is exported and uploaded to Kaggle again to see if it improves.

```

In [137]: score = xgb_model.score(test_data, test_labels)
score

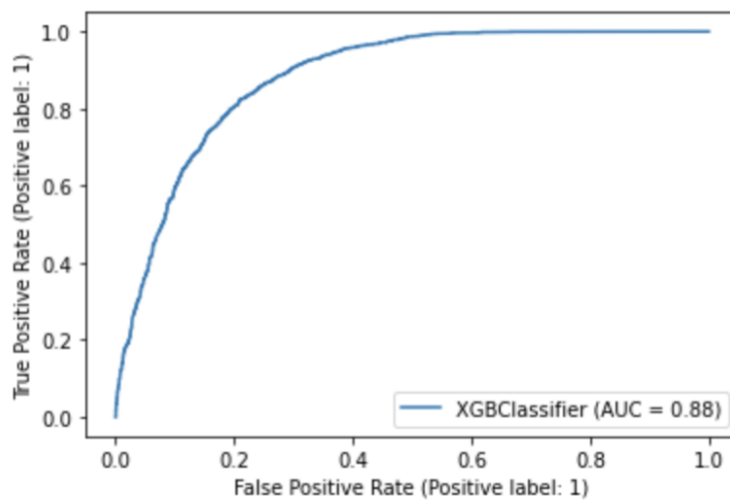
```

```

Out[137]: 0.79425

```

```
display = metrics.plot_roc_curve(xgb_model, test_data, test_labels)
```



```
: import seaborn as sns
from sklearn.metrics import classification_report
# Various evaluation indexes
print(classification_report(test_labels,y_pre))
```

	precision	recall	f1-score	support
0	0.78	0.82	0.80	2000
1	0.81	0.77	0.79	2000
accuracy			0.79	4000
macro avg	0.80	0.79	0.79	4000
weighted avg	0.80	0.79	0.79	4000

Although the local score and ROC curves are not as good as those of the random forest, the XGBoost model is much more accurate than the previous random forest model without data imbalance processing, with an overall accuracy of 0.79.

Final Model contrast

Accurate rate

Decision Tree: 0.829

After Inbalance Processing: 0.75

```
from sklearn.metrics import classification_report
print(classification_report(clf.predict(test_data),test_labels))
```

	precision	recall	f1-score	support
0	0.82	0.71	0.76	2297
1	0.67	0.79	0.73	1703
accuracy			0.75	4000
macro avg	0.75	0.75	0.74	4000
weighted avg	0.76	0.75	0.75	4000

Random Forest: 0.849

Random Forest After Grid Search: 0.85

After Inbalance Processing: 0.75

```
print(classification_report(clf.predict(test_data),test_labels))
```

	precision	recall	f1-score	support
0	0.82	0.71	0.76	2297
1	0.67	0.79	0.73	1703
accuracy			0.75	4000
macro avg	0.75	0.75	0.74	4000
weighted avg	0.76	0.75	0.75	4000

SVM: 0.806

XGBoost: 0.79425

```
import seaborn as sns
from sklearn.metrics import classification_report
# Various evaluation indexes
print(classification_report(test_labels,y_pre))
```


	precision	recall	f1-score	support
0	0.78	0.82	0.80	2000
1	0.81	0.77	0.79	2000
accuracy			0.79	4000
macro avg	0.80	0.79	0.79	4000
weighted avg	0.80	0.79	0.79	4000


I chose XGBoost with the data imbalance because the model was relatively accurate in predicting 0 and 1.

Final Best classifier that I selected


In the end, I chose XGBoost as the best modeler based on the above contrast. The accuracy rate of XGBoost is the highest. Therefore, I will use the prediction func of XGBoost to classify unknown data.

Kanggle Submission

19	Sw_H6666		0.80066	9	~10s
----	----------	---	---------	---	------

Your Best Entry 

Your submission scored 0.80066, which is an improvement of your previous score of 0.68087. Great job!

 Tweet this!

My final model also got not bad results on Kaggle.

Summary

In Assignment3, I learned several Python libraries for data analysis, such as Numpy and Pandas. I learned how to preprocess data in Python. In the meantime, I also got to know the Decision tree, random forest, support vector machine, K-nearest-neighbor, XGBoost, and Gradient Boosting Decision Tree in Sklearn. And these algorithms are preliminarily tried and used. Through additional tuning parameters, I understand the core idea of these algorithms. In the future study, I will have a deeper understanding and use of these algorithms and better deal with overfitting problems.