Predict survival on the Titanic

Qing Zhao

1.Introduction

The sinking of the Titanic is one of the most infamous shipwrecks in history. Titanic sank after colliding with an iceberg. Unfortunately, there weren't enough lifeboats for everyone onboard, resulting in the death of 1502 out of 2224 passengers and crew. While there was some element of luck involved in surviving, it seems some groups of people were more likely to survive than others.

In this assignment, I will use passengers' data to build a predictive model to figure out what sorts of people were more likely to survive. Since the target label in train set is available, the learning process is supervised and since the values of target are "Survival or not", it's a classification task.

I will try five types of models (naive bays, decision tree, neural_network, logistic regression, random forest), using metircs like accuracy, auc, and precision to evaluate them, hoping to find the optimal one. After that, I will try to adjust hyper-parameters both manually and with the grid search tools to fine tune the optimal model in order to improve its performance in prediction.

2. Data Exploration

2.1 data structure description

The data has been split into two groups, one is the train set containing 891 records and 11 features plus the target label "Survived", the other is the test set containing 418 records which is used to validate the performance of models.

Firstly, I Use methods such as shape, head(), info(), describe(), and value_counts() in Pandas for data structure description.

```
train.shape
```

```
(891, 12)
```

```
train.head()
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

```
# Age, Cabin, and Embarked have missing values
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   PassengerId  891 non-null    int64  
 1   Survived     891 non-null    int64  
 2   Pclass       891 non-null    int64  
 3   Name         891 non-null    object 
 4   Sex          891 non-null    object 
 5   Age          714 non-null    float64
 6   SibSp        891 non-null    int64  
 7   Parch        891 non-null    int64  
 8   Ticket       891 non-null    object 
 9   Fare         891 non-null    float64
 10  Cabin        204 non-null    object 
 11  Embarked     889 non-null    object 
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
```

Since the three variables of "passengerid" ,"name" and "Ticket" serve as unique identifiers for passenger identities but not universally applicable, they are unsuitable for model training. They will be dropped latter. The remaining variables are categorized into two groups based on their data types: numerical and categorical . Different processing will be applied to these groups during data preparation:

```
features_unique=["Name","PassengerId","Ticket"]
features_cat=['Sex','Cabin','Embarked']
features_num=['Pclass', 'Age', 'SibSp', 'Parch', 'Fare']
```

;

```
%matplotlib inline
import matplotlib.pyplot as plt
train.hist(bins=50, figsize=(20,15))
#save_fig("attribute_histogram_plots")
plt.show()
```

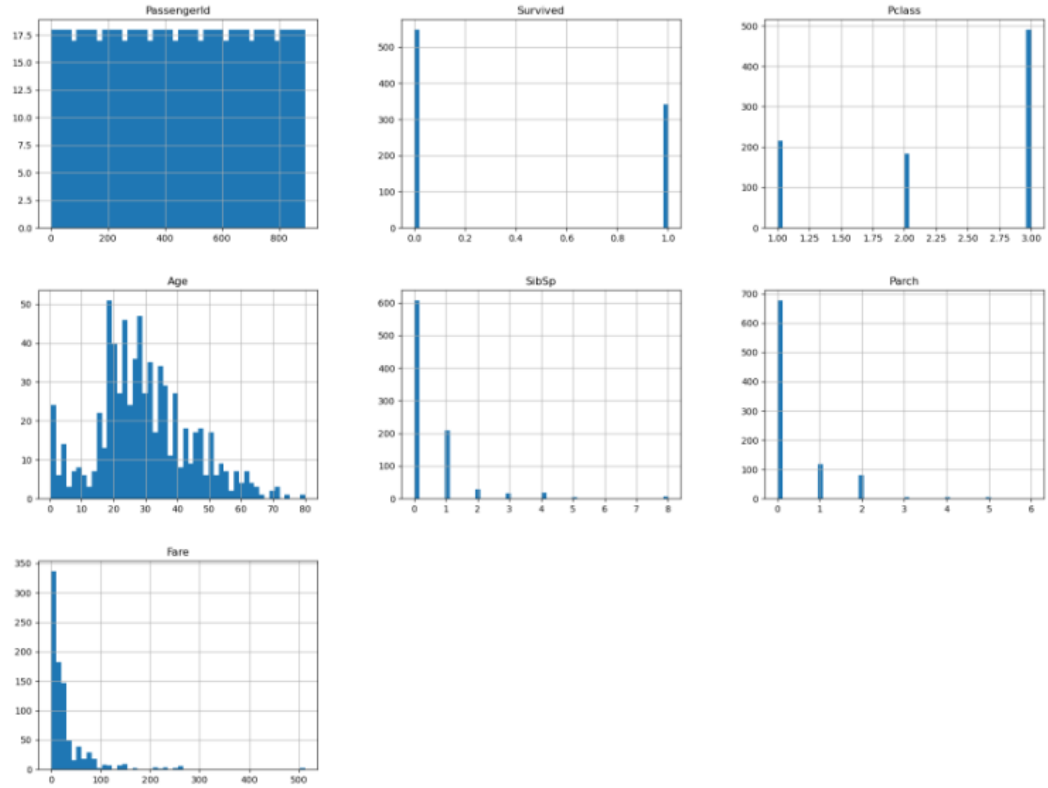


Figure 1 Distribution plot of numerical features

```
train.Sex.value_counts(normalize=True,dropna=False)
```

```
male      0.647587
female    0.352413
Name: Sex, dtype: float64
```

```
train.Cabin.value_counts(normalize=True,dropna=False)
```

```
NaN            0.771044
C23 C25 C27    0.004489
G6             0.004489
B96 B98        0.004489
C22 C26        0.003367
                 ...
E34            0.001122
C7             0.001122
C54            0.001122
E36            0.001122
C148           0.001122
Name: Cabin, Length: 148, dtype: float64
```

```
train.Embarked.value_counts(normalize=True,dropna=False)
```

```
S      0.722783
C      0.188552
Q      0.086420
NaN    0.002245
Name: Embarked, dtype: float64
```

Figure2 :Statistical summary of categorical data distribution

Upon observation, it is noticed that there are missing values in the numerical feature “Age” and the categorical features “Cabin” and “Embarked”, requiring necessary handling in the subsequent data preparation process.

2.2 visualizing the data to discover more insights

After data visualizing, some scatter plots provide me with some insights, including:(1) Passengers from the upper class are more likely to purchase tickets costing over 100;(2) The cumulative quantity of Sibsp for passengers from the lower class exceeds 3;(3) With increasing age, the quantity of passengers from S tends to decrease;(4) There are few survivors aged 65 and above.

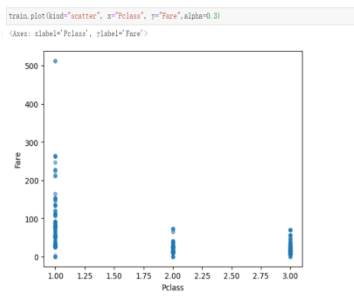


Figure 3: Scatter plot of PClass&Fare

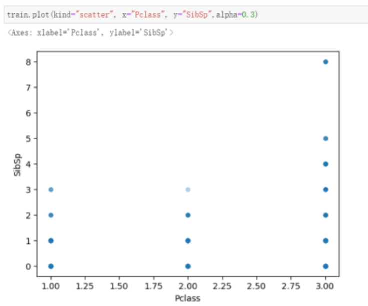


Figure 4:Scatter plot of PClass&Sibsp

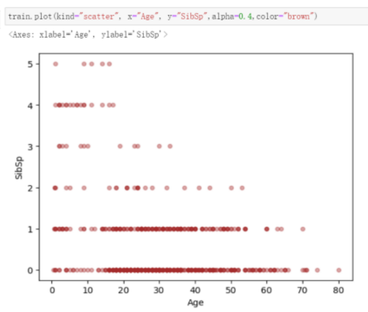


Figure 5:Scatter plot of Age&Sibsp

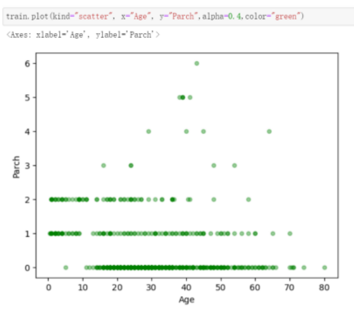


Figure 6: Scatter plot of Age&Parch

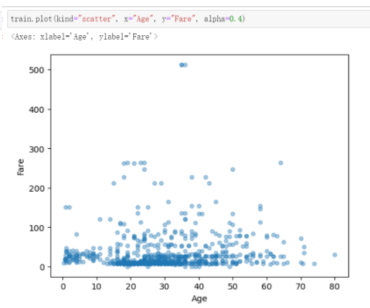


Figure 7:Scatter plot of Age&Fare

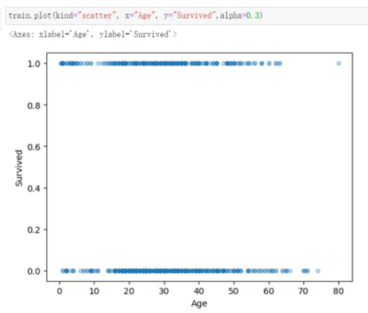


Figure 8:Scatter plot of Age&Survived

2.3 Feature correlation analysis

Then, I Utilize corr() in pandas to investigate the correlation between each numerical feature, the coefficient matrix reveals that none of these features exhibit a strong correlation. This suggests they can be effectively used together in model fitting. However, there is a degree of negative correlation between Fare and Pclass; with higher P levels, the Fare may increase. Similarly, this trend holds for Age and Pclass, where an increase in Pclass level corresponds to an older Age.

```
#corelation coefficient matrix
corr = train[features_num].corr()
corr
```

	Pclass	Age	SibSp	Parch	Fare
Pclass	1.000000	-0.369226	0.083081	0.018443	-0.549500
Age	-0.369226	1.000000	-0.308247	-0.189119	0.096067
SibSp	0.083081	-0.308247	1.000000	0.414838	0.159651
Parch	0.018443	-0.189119	0.414838	1.000000	0.216225
Fare	-0.549500	0.096067	0.159651	0.216225	1.000000

Figure 9: correlation coefficient matirx

3.Data Preparation

During the data preparation stage, the dataset undergoes cleaning procedures, including removing irrelevant features, filling missing values, standardizing numerical data, and performing one-hot encoding for categorical data. After this stage, the dataset will be transformed into a standardized collection suitable for model training. Firstly, passengers identity variable "Name", "PassengerId" and "Ticket" are dropped from the dataset ; then do the same to “Cabin”, which has high levels of missing values. Now, the rest of 7 feature ['Pclass', 'Sex', 'Age', 'SibSp', 'Parch', 'Fare', 'Embarked'] can be used to fit a model.

```
Y=train["Survived"].copy()
X=train.drop("Survived",axis=1)
X.drop(features_unique,axis=1,inplace=True)
X.drop("Cabin", axis=1,inplace=True)
```

```
X.columns
```

```
Index(['Pclass', 'Sex', 'Age', 'SibSp', 'Parch', 'Fare', 'Embarked'], dtype='object')
```

Further, I divide the features into numerical and categorical feature groups. To numerical features, the processing method is filling missing values with the median and standardizing, and to categorical ones, filling missing values with the mode and encoding. In the process,I use ColumnTransformer() to integrate these two sets of data processing methods for comprehensive processing of X dataset. Generate the prepared training X_prepared.

```python
X_num = X.drop(['Embarked','Sex'], axis=1)
X_cat = X.loc[:,['Embarked','Sex']]

from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import OneHotEncoder
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer

num_pipeline = Pipeline([
        ('imputer_median',SimpleImputer(strategy="median")),
        ('std_scaler', StandardScaler()),
    ])

cat_pineline=Pipeline([
        ('imputer_mode',SimpleImputer(strategy="most_frequent")),
        ('OneHotEncoder',OneHotEncoder())

    ])

num_attribs = list(X_num)
cat_attribs = list(X_cat)

full_pipeline = ColumnTransformer([
        ("num", num_pipeline, num_attribs),
        ("cat",cat_pineline, cat_attribs),
    ])

X_prepared = full_pipeline.fit_transform(X)
```

```python
X_prepared
```

```
array([[ 0.82737724, -0.56573646,  0.43279337, ...,  1.        ,
         0.        ,  1.        ],
       [-1.56610693,  0.66386103,  0.43279337, ...,  0.        ,
         1.        ,  0.        ],
       [ 0.82737724, -0.25833709, -0.4745452 , ...,  1.        ,
         1.        ,  0.        ],
       ...,
       [ 0.82737724, -0.1046374 ,  0.43279337, ...,  1.        ,
         1.        ,  0.        ],
       [-1.56610693, -0.25833709, -0.4745452 , ...,  0.        ,
         0.        ,  1.        ],
       [ 0.82737724,  0.20276197, -0.4745452 , ...,  0.        ,
         0.        ,  1.        ]])
```

Finally, Performing the same operations on the test set, resulting in the prepared test dataset X_test_prepared.

```python
test=pd.read_csv("D:/python_study/Assignment1/test.csv")
test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 418 entries, 0 to 417
Data columns (total 11 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   PassengerId  418 non-null    int64  
 1   Pclass       418 non-null    int64  
 2   Name         418 non-null    object 
 3   Sex          418 non-null    object 
 4   Age          332 non-null    float64
 5   SibSp        418 non-null    int64  
 6   Parch        418 non-null    int64  
 7   Ticket       418 non-null    object 
 8   Fare         417 non-null    float64
 9   Cabin        91 non-null     object 
 10  Embarked     418 non-null    object 
dtypes: float64(2), int64(4), object(5)
memory usage: 36.0+ KB
```

```python
X_test=test.drop(["Cabin","Name","PassengerId","Ticket"],axis=1)
X_test.head()
```

	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
0	3	male	34.5	0	0	7.8292	Q
1	3	female	47.0	1	0	7.0000	S
2	2	male	62.0	0	0	9.6875	Q
3	3	male	27.0	0	0	8.6625	S
4	3	female	22.0	1	1	12.2875	S

```python
X_test_prepared = full_pipeline.transform(X_test)
```

```
X_test_prepared
```

```
array([[ 0.82737724,  0.39488658, -0.4745452 , ...,  0.        ,
         0.        ,  1.        ],
       [ 0.82737724,  1.35550962,  0.43279337, ...,  1.        ,
         1.        ,  0.        ],
       [-0.36936484,  2.50825727, -0.4745452 , ...,  0.        ,
         0.        ,  1.        ],
       ...,
       [ 0.82737724,  0.70228595, -0.4745452 , ...,  1.        ,
         0.        ,  1.        ],
       [ 0.82737724, -0.1046374 , -0.4745452 , ...,  1.        ,
         0.        ,  1.        ],
       [ 0.82737724, -0.1046374 ,  0.43279337, ...,  0.        ,
         0.        ,  1.        ]])
```

4.Fitting models and evaluating the performance

4.1 Fitting models

In this assignment, I use GaussianNB, DecisionTreeClassifier, MLPClassifier, LogisticRegression, and RandomForestClassifier for modeling, with accuracy, auc, and precision for model evaluation.

Table 1: The matrix of the model training

NO.	model	accuracy	auc	precision
1	GaussianNB()	0.79	0.78	0.72
2	DecisionTreeClassifier(criterion="gini", max_depth=5, min_samples_leaf=10, random_state=28)	0.83	0.80	0.87
3	MLPClassifier(solver='adam', alpha=1, max_iter=1000, random_state=42)	**0.85**	**0.82**	0.86
4	LogisticRegression()	0.80	0.78	0.86
5	RandomForestClassifier(n_estimators=10, max_depth=4, max_features=3, random_state=42)	0.84	0.81	0.85

The table displayed above shows that MLPClassifier has the best performance with accuracy 0.85 and auc 0.82, followd by RandomForestClassifier (accuracy 0.84 and auc 0.81).

```
# neural_network Perceptron
from sklearn.neural_network import MLPClassifier

nnk_clf = MLPClassifier(solver='adam',
                        alpha=1,
                        max_iter=1000,
                        random_state=42)

Classifier=nnk_clf.fit(X_prepared, Y)
Y_predictions =Classifier.predict(X_prepared)
display_metrics(Y,Y_predictions,Classifier)

np_predictions=pickup_model(Classifier,X_test_prepared)
test_prediction= pd.concat([test["PassengerId"],pd.DataFrame(np_predictions,columns=["Survived"])],axis= 1)
test_prediction.to_csv("D:/python_study/Assignment1/submission_NetWorkPerception_adam.csv",index=False)
```

```
Accuracy: 0.85  AUC: 0.82   Precision: 0.86
```

```
# RandomForestClassifier

from sklearn.ensemble import RandomForestClassifier

forest_clf = RandomForestClassifier(n_estimators=10,
                                    max_depth=4,
                                    max_features=3,
                                    #min_samples_split=10,
                                    #min_samples_leaf=5,
                                    random_state=42)

Classifier=forest_clf.fit(X_prepared, Y)
Y_predictions =Classifier.predict(X_prepared)
display_metrics(Y,Y_predictions,Classifier)

np_predictions=pickup_model(Classifier,X_test_prepared)
test_prediction= pd.concat([test["PassengerId"],pd.DataFrame(np_predictions,columns=["Survived"])],axis= 1)
test_prediction.to_csv("D:/python_study/Assignment1/submission_RandomForest.csv",index=False)
```

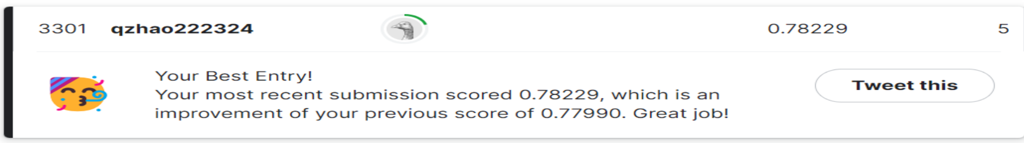
```
Accuracy: 0.84  AUC: 0.81   Precision: 0.85
```

4.2 Predicting the test set

According the feedback from Kaggle,the model that performed the best on the test set remains MLPClassifier, with the highest accuracy 0.78229.

Table 2: The accuracy of applying the above five models to the test set

NO.	model	test_accuracy
1	GaussianNB()	0.75358
2	DecisionTreeClassifier(criterion="gini",max_depth=5,min_samples_leaf=10,random_state=28)	0.77990
3	MLPClassifier(solver='adam',alpha=1,max_iter=1000,random_state=42)	**0.78229**
4	LogisticRegression()	0.76555
5	RandomForestClassifier(n_estimators=10,max_depth=4, max_features=3,random_state=42)	0.77033



5.Fine tune

According the validation results on the test set, model of MLPClassifier exhibits the best performance. Moving forward, I aim to further enhance the model's learning effectiveness through fine-tuning hyperparameters. After modifying hyperparameters such as 'solver' and 'alpha' and 'earning_rate_init', the performance on the training set improved. However, when validated on the test set, there was no improvement in performance. This indicates that these adjustments were unsuccessful.

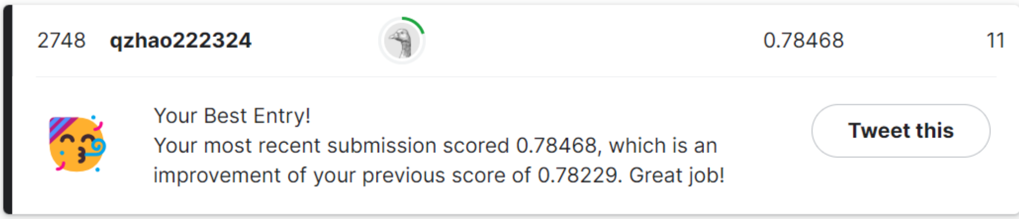
Table 3: The accuracy of fine tune models

	model	train_accuracy	train_auc	train_precision	test_accuracy
origanal	MLPClassifier(solver='adam',alpha=1,max_iter=1000,random_state=42)	0.85	0.82	0.86	**0.78229**
Fine-tune1	MLPClassifier(solver='lbfgs',alpha=1,max_iter=1000,random_state=42)	0.87	0.85	0.9	0.76315
Fine-tune2	MLPClassifier(solver='sgd',learning_rate_init=0.05,alpha=1,max_iter=1000,random_state=42)	0.85	0.82	0.88	0.7799
Fine-tune3	MLPClassifier(solver='adam',alpha=0.002,max_iter=1000,random_state=42)	0.87	0.84	0.89	0.76794

Finally, I try to use gribsearchCV() for automatic hyperparameter tuning resulted in MLPClassifier(solver='adam',alpha=0.03,learning_rate_init=0.01， max_iter=1000,random_state=42) as the optimal combination under given conditions. After retraining and testing the model, its performance on the test set improved from 0.78229 to 0.78468.

Table 4: The metrics of fine tune models

	model	train_ accuracy	train_auc	train_ precision	test_ accuracy
Fine-tune4	MLPClassifier(solver='adam',alpha=0.03,learning_rate_init=0.01, max_iter=1000,random_state=42)	0.85	0.82	0.89	0.78468



6.Summary

In this Assignment, I tried five types of models (naive bays, decision tree, neural_network, logistic regression, random forest). After validated by test set and the score feedbacked by Kaggle, neural_network MPLClassifier stands out. Finally, with the grid search tools, I found the optimal combination of hyper-parameters under given conditions, which improving the accuracy by 0.00239. As of now, I am ranked 2748 on the leaderboard.