Supervised ML: Classification

Titanic Survival Dataset from Kaggle

Overview

The data has been split into two groups:

```
training set (train.csv)
test set (test.csv)
```

The training set should be used to build your machine learning models. For the training set, we provide the outcome (also known as the "ground truth") for each passenger. Your model will be based on "features" like passengers' gender and class. You can also use feature engineering to create new features.

The test set should be used to see how well your model performs on unseen data. For the test set, we do not provide the ground truth for each passenger. It is your job to predict these outcomes. For each passenger in the test set, use the model you trained to predict whether or not they survived the sinking of the Titanic.

We also include gender_submission.csv, a set of predictions that assume all and only female passengers survive, as an example of what a submission file should look like.

- Variable Definition Key
- survival Survival 0 = No, 1 = Yes
- pclass Ticket class 1 = 1st, 2 = 2nd, 3 = 3rd
- sex Sex
- Age Age in years
- sibsp # of siblings / spouses aboard the Titanic
- parch # of parents / children aboard the Titanic
- ticket Ticket number
- fare Passenger fare
- cabin Cabin number
- embarked Port of Embarkation C = Cherbourg, Q = Queenstown, S = Southampton

Variable Notes

pclass: A proxy for socio-economic status (SES) 1st = Upper 2nd = Middle 3rd = Lower

age: Age is fractional if less than 1. If the age is estimated, is it in the form of xx.5

sibsp: The dataset defines family relations in this way... Sibling = brother, sister, stepbrother, stepsister Spouse = husband, wife (mistresses and fiancés were ignored)

parch: The dataset defines family relations in this way... Parent = mother, father Child = daughter, son, stepdaughter, stepson Some children travelled only with a nanny, therefore parch=0 for them.

In [1]:
```
# Import Necessary Libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```
titanic_train = pd.read_csv('titanic_train.csv')
titanic_test = pd.read_csv('titanic_test.csv')

display(titanic_train.head())
titanic_train.shape
```

	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

Out[2]:
```
(891, 12)
```

In [3]:
```
df = titanic_train.copy()
df.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
```

In [4]:
```
display(df.isnull().sum())
df = df.drop('Cabin', axis=1) # Dropped too many NaNs
df['Age'] = df['Age'].fillna(df.Age.median())
df['Embarked'].unique()
```

```
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age            177
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         2
dtype: int64
```

Out[4]:
```
array(['S', 'C', 'Q', nan], dtype=object)
```

In [5]:
```
df['Embarked'] = df.Embarked.fillna('S')
df.isnull().sum()
```

Out[5]:
```
PassengerId    0
Survived       0
Pclass         0
Name           0
Sex            0
Age            0
SibSp          0
Parch          0
Ticket         0
Fare           0
Embarked       0
dtype: int64
```

In [6]:
```
df.drop(['Name', 'Ticket'], axis=1, inplace=True)
```

In [7]:
```
y= df['Survived']
X_full = df.drop(['PassengerId', 'Survived'], axis =1)
X_full.head()
```

Out[7]:

	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
0	3	male	22.0	1	0	7.2500	S
1	1	female	38.0	1	0	71.2833	C
2	3	female	26.0	0	0	7.9250	S
3	1	female	35.0	1	0	53.1000	S
4	3	male	35.0	0	0	8.0500	S

In [8]:
```
from sklearn.preprocessing import OneHotEncoder

onehot = OneHotEncoder(handle_unknown='ignore')
conv_dict = {col: 'object' for col in X_full.columns if col not in ['Age', 'Fare']}
X_full = X_full.astype(conv_dict)
X_full['Age_cat'] = pd.cut(X_full.Age, 5, labels=['children', 'youth', 'middle_aged', 'jubileans', 'aged'])
X_full.dtypes
cat_cols = [col for col in X_full.columns if X_full[col].dtype != 'float']
X_full_cat = X_full[cat_cols]
X_cat_tran = onehot.fit_transform(X_full_cat).toarray()
X_cat = pd.DataFrame(X_cat_tran, columns=onehot.get_feature_names(cat_cols))
X_fare_scaled = ((X_full['Fare'] - X_full['Fare'].mean()) / X_full['Fare'].std()).to_frame()
X = pd.concat([X_cat, X_fare_scaled], axis=1)
X.head()
```

Out[8]:

	Pclass_1	Pclass_2	Pclass_3	Sex_female	Sex_male	SibSp_0	SibSp_1	SibSp_2	SibSp_3	SibSp_4	...	Parch_6	Embarked_C	Embarked_Q	Er
0	0.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0	0.0	0.0	...	0.0	0.0	0.0	
1	1.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	...	0.0	1.0	0.0	
2	0.0	0.0	1.0	1.0	0.0	1.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	
3	1.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	...	0.0	0.0	0.0	
4	0.0	0.0	1.0	0.0	1.0	1.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	

5 rows × 28 columns

In [9]:
```
# Mute the sklearn warning about regularization
import warnings
warnings.filterwarnings('ignore', module='sklearn')
warnings.filterwarnings('ignore', module='xgboost')
from sklearn.model_selection import train_test_split, GridSearchCV, cross_val_score
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier as KNN
from sklearn.ensemble import VotingClassifier, BaggingClassifier, RandomForestClassifier
from sklearn.metrics import accuracy_score, roc_auc_score
import xgboost as xgb

SEED=42
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=.25, random_state=SEED)

# Instantiate Classifiers
logreg = LogisticRegression(random_state=SEED)
knn = KNN()
dt = DecisionTreeClassifier(max_depth=6, random_state=SEED)
bc = BaggingClassifier(base_estimator=dt, n_estimators=100, n_jobs=-1)
rf = RandomForestClassifier(max_depth=6, n_estimators=200, random_state=SEED)
xg_clf = xgb.XGBClassifier(objective='binary:logistic',seed=SEED)

classifiers = [('logreg', logreg),
               ('KNN', knn),
               ('DTree', dt),
               ('Bag', bc),
               ('Forest', rf),
               ('XGB', xg_clf)]

vc = VotingClassifier(estimators = classifiers)
all_class = classifiers + [('voter', vc)]

for clf_name, clf in all_class:
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)
    score = accuracy_score(y_test, y_pred)
    print(f'Estimators {clf_name}; Accuracy_score: {score}')
```

```
Estimators logreg; Accuracy_score: 0.7847533632286996
Estimators KNN; Accuracy_score: 0.7937219730941704
Estimators DTree; Accuracy_score: 0.820627802690583
Estimators Bag; Accuracy_score: 0.8071748878923767
Estimators Forest; Accuracy_score: 0.820627802690583
[01:06:20] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/learner.cc:1095: Starting
in XGBoost 1.3.0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'err
or' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
Estimators XGB; Accuracy_score: 0.8116591928251121
[01:06:20] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/learner.cc:1095: Starting
in XGBoost 1.3.0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'err
or' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
Estimators voter; Accuracy_score: 0.8251121076233184
```

In [16]:
```
display(titanic_test.head())

df2 = titanic_test[titanic_test.Parch != 9].copy()
df2 = df2.drop('Cabin', axis=1) # Dropped too many NaNs
df2['Age'] = df2['Age'].fillna(df2.Age.median())
df2['Embarked'] = df2.Embarked.fillna('S')
df2.drop(['Name', 'Ticket'], axis=1, inplace=True)
X_full2 = df2.drop(['PassengerId'], axis =1)

X_full2 = X_full2.astype(conv_dict)
X_full2['Age_cat'] = pd.cut(X_full2.Age, 5, labels=['children', 'youth', 'middle_aged', 'jubileans', 'aged'])
X_full2.dtypes
# cat_cols = [col for col in X_full.columns if X_full[col].dtype != 'float']
X_full_cat2 = X_full2[cat_cols]
X_cat_tran2 = onehot.fit_transform(X_full_cat2).toarray()
X_cat2 = pd.DataFrame(X_cat_tran2, columns=onehot.get_feature_names(cat_cols))
X_full2['Fare'] = X_full2.Fare.fillna(X_full2.Fare.median())
X_fare_scaled2 = ((X_full2['Fare'] - X_full2['Fare'].mean()) / X_full2['Fare'].std()).to_frame()
X_test_real = pd.concat([X_cat2, X_fare_scaled2], axis=1)
X_test_real.fillna(0, inplace=True)
X_test_real.head()

rf2 = RandomForestClassifier(max_depth=6, n_estimators=300, random_state=SEED)
rf2.fit(X, y)
print(rf.feature_importances_)
y_pred = rf2.predict(X_test_real)
```

	PassengerId	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	892	3	Kelly, Mr. James	male	34.5	0	0	330911	7.8292	NaN	Q
1	893	3	Wilkes, Mrs. James (Ellen Needs)	female	47.0	1	0	363272	7.0000	NaN	S
2	894	2	Myles, Mr. Thomas Francis	male	62.0	0	0	240276	9.6875	NaN	Q
3	895	3	Wirz, Mr. Albert	male	27.0	0	0	315154	8.6625	NaN	S
4	896	3	Hirvonen, Mrs. Alexander (Helga E Lindqvist)	female	22.0	1	1	3101298	12.2875	NaN	S

```
[0.03550107 0.02685047 0.08493127 0.25111512 0.23161811 0.01346777
 0.02029614 0.00514019 0.00590724 0.00589075 0.00199003 0.00573771
 0.02287622 0.01405381 0.01040122 0.00229678 0.00205551 0.00197462
 0.00084001 0.02152555 0.00813463 0.01679824 0.00151884 0.03367482
 0.00569199 0.01094644 0.01317376 0.14559169]
```

In [18]:
```
pd.DataFrame(list(zip(X.columns, rf.feature_importances_)), columns = ['features', 'importances'])
```

Out[18]:

	features	importances
0	Pclass_1	0.035501
1	Pclass_2	0.026850
2	Pclass_3	0.084931
3	Sex_female	0.251115
4	Sex_male	0.231618
5	SibSp_0	0.013468
6	SibSp_1	0.020296
7	SibSp_2	0.005140
8	SibSp_3	0.005907
9	SibSp_4	0.005891
10	SibSp_5	0.001990
11	SibSp_8	0.005738
12	Parch_0	0.022876
13	Parch_1	0.014054
14	Parch_2	0.010401
15	Parch_3	0.002297
16	Parch_4	0.002056
17	Parch_5	0.001975
18	Parch_6	0.000840
19	Embarked_C	0.021526
20	Embarked_Q	0.008135
21	Embarked_S	0.016798
22	Age_cat_aged	0.001519
23	Age_cat_children	0.033675
24	Age_cat_jubileans	0.005692
25	Age_cat_middle_aged	0.010946
26	Age_cat_youth	0.013174
27	Fare	0.145592

Best Estimator is RandomForestClassifier with an accuracy of 0.821

- Possible revisit to the data is to tune hyperparameters and use other classifiers such as Neural Networks Classifiers
- Also, more data would be needed to properly get the best from the data