Survivor
PREDICTION

3-survivors-transported-prediction

June 8, 2025

[1]:
```python
import numpy
import pandas
from sklearn.preprocessing import OneHotEncoder
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import GridSearchCV
from xgboost import XGBClassifier
import matplotlib.pyplot as mat
import seaborn as sns
from sklearn.svm import SVC

# To remove the warnings for deperacations of the functions
pandas.set_option('future.no_silent_downcasting', True)
```

[2]:
```python
# Bringing Data
main_train_data = pandas.read_csv(r"/kaggle/input/spaceship-titanic/train.csv")
main_test_data = pandas.read_csv(r"/kaggle/input/spaceship-titanic/test.csv")

main_test_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4277 entries, 0 to 4276
Data columns (total 13 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   PassengerId   4277 non-null   object 
 1   HomePlanet    4190 non-null   object 
 2   CryoSleep     4184 non-null   object 
 3   Cabin         4177 non-null   object 
 4   Destination   4185 non-null   object 
 5   Age           4186 non-null   float64
 6   VIP           4184 non-null   object 
 7   RoomService   4195 non-null   float64
 8   FoodCourt     4171 non-null   float64
 9   ShoppingMall  4179 non-null   float64
 10  Spa           4176 non-null   float64
```

```
 11  VRDeck        4197 non-null   float64
 12  Name          4183 non-null   object
dtypes: float64(6), object(7)
memory usage: 434.5+ KB
```

[3]:
```python
main_train_data.describe()
```

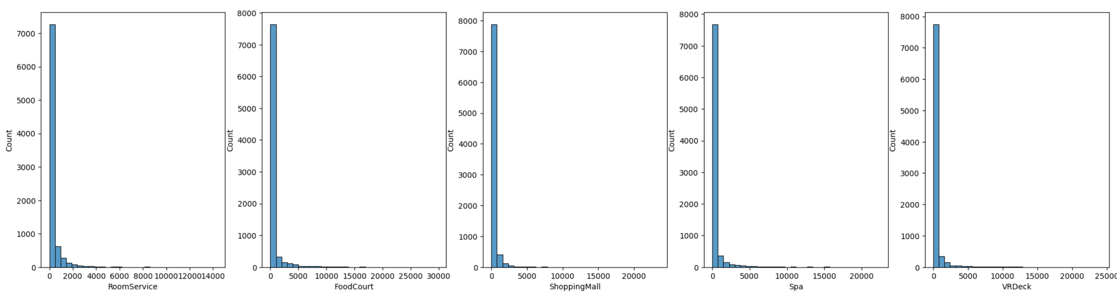
[3]:
```
                Age   RoomService     FoodCourt  ShoppingMall           Spa  \
count   8514.000000   8512.000000   8510.000000   8485.000000   8510.000000
mean      28.827930    224.687617    458.077203    173.729169    311.138778
std       14.489021    666.717663   1611.489240    604.696458   1136.705535
min        0.000000      0.000000      0.000000      0.000000      0.000000
25%       19.000000      0.000000      0.000000      0.000000      0.000000
50%       27.000000      0.000000      0.000000      0.000000      0.000000
75%       38.000000     47.000000     76.000000     27.000000     59.000000
max       79.000000  14327.000000  29813.000000  23492.000000  22408.000000

             VRDeck
count   8505.000000
mean     304.854791
std     1145.717189
min        0.000000
25%        0.000000
50%        0.000000
75%       46.000000
max    24133.000000
```

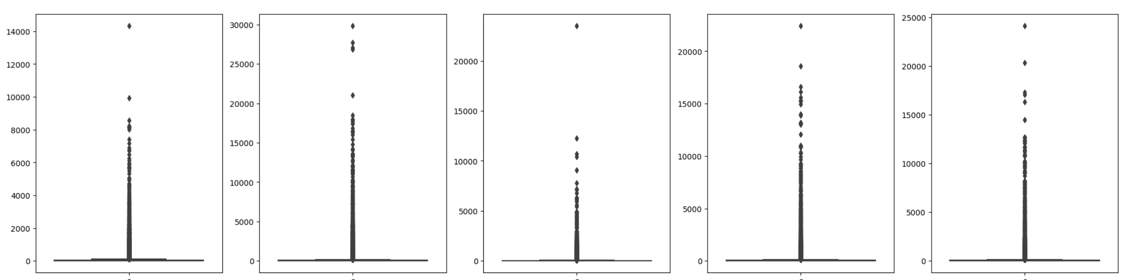
[4]:
```python
fig, ax = mat.subplots(1, 5, figsize=(25, 6))

sns.histplot(main_train_data['RoomService'], bins=30, ax=ax[0])
sns.histplot(main_train_data['FoodCourt'], bins=30, ax=ax[1])
sns.histplot(main_train_data['ShoppingMall'], bins=30, ax=ax[2])
sns.histplot(main_train_data['Spa'], bins=30, ax=ax[3])
sns.histplot(main_train_data['VRDeck'], bins=30, ax=ax[4])

fig, axis = mat.subplots(1, 5, figsize=(25, 6))

sns.boxplot(main_train_data['RoomService'], ax=axis[0])
sns.boxplot(main_train_data['FoodCourt'], ax=axis[1])
sns.boxplot(main_train_data['ShoppingMall'], ax=axis[2])
sns.boxplot(main_train_data['Spa'], ax=axis[3])
sns.boxplot(main_train_data['VRDeck'], ax=axis[4])
```

```
/usr/local/lib/python3.10/dist-packages/seaborn/_oldcore.py:1119: FutureWarning:
use_inf_as_na option is deprecated and will be removed in a future version.
Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
/usr/local/lib/python3.10/dist-packages/seaborn/_oldcore.py:1119: FutureWarning:
```

```
use_inf_as_na option is deprecated and will be removed in a future version.
Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
/usr/local/lib/python3.10/dist-packages/seaborn/_oldcore.py:1119: FutureWarning:
use_inf_as_na option is deprecated and will be removed in a future version.
Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
/usr/local/lib/python3.10/dist-packages/seaborn/_oldcore.py:1119: FutureWarning:
use_inf_as_na option is deprecated and will be removed in a future version.
Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
/usr/local/lib/python3.10/dist-packages/seaborn/_oldcore.py:1119: FutureWarning:
use_inf_as_na option is deprecated and will be removed in a future version.
Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```

[4]: <Axes: >





- We have a little bot of outliers and thus we cannot remove them now.
- Identical pattern can be seen

[5]:
```
# Feature Engineering

# Total Bill Generated during the journey

def engineeringfeature(data):
```

```
    # First Feature
    data['Total Bill Paid'] = (data['RoomService'] +
        data['FoodCourt'] +
        data['ShoppingMall'] +
        data['Spa'] +
        data['VRDeck']
        )

    # Second Feature
    data['NoExpense'] = data.apply(lambda row: 0 if (row[['RoomService',␣
 ↪'FoodCourt', 'ShoppingMall', 'Spa']] == 0).sum() == 4 else 1, axis=1)

    return 'Done'


# For Training Model
print(engineeringfeature(data=main_train_data))
# For Testing Model
print(engineeringfeature(data=main_test_data))
```

```
Done
Done
```

0.1 - Handling the missing values for train dataset

[6]:

```
transported_train = main_train_data['Transported']
```

[7]:

```
# Function that handles both of the sets
encoder = OneHotEncoder(sparse_output=False)

def missing_value_handling(data, set):

    # Dividing the set into two parts
    categorical_fet = data.select_dtypes(include='object')
    continous_fet = data.select_dtypes(include=['float', 'int'])

    # Using interpolate function for numerical features
    interpolated_num = continous_fet.interpolate()

    # Using mode for categorical fetures to fill the missing values
    data['Cabin'] = data['Cabin'].fillna(data['Cabin'].mode()[0])
    listings = categorical_fet.columns

    for feature_name in listings:
        categorical_fet[feature_name] = categorical_fet[feature_name].
 ↪fillna(categorical_fet[feature_name].mode()[0])
```

```
    # One Hot Encoder
    cat_encoded = encoder.fit_transform(categorical_fet[['HomePlanet',␣
↪'CryoSleep', 'Destination', 'VIP']])
    categorical_fet = pandas.DataFrame(
        cat_encoded,
        columns = encoder.get_feature_names_out(['HomePlanet', 'CryoSleep',␣
↪'Destination', 'VIP'])
    )

    # Condition for knowing the dataset
    if set == 'train':
        data = pandas.concat(
            [categorical_fet, interpolated_num, transported_train,␣
↪data[['PassengerId', 'Cabin']]], axis=1
        )

    else:
        data = pandas.concat(
            [categorical_fet, interpolated_num, data[['PassengerId',␣
↪'Cabin']]], axis=1
        )

    return data

train_data = missing_value_handling(data=main_train_data, set='train')
test_data = missing_value_handling(data=main_test_data, set='test')
```

[8]:
```
# Separation of passenger ids
def passenger_id(main_data, data):
    first = []
    second = []

    for ids in main_data['PassengerId']:
        splited_numbers = ids.split('_')

        first.append(int(splited_numbers[0]))
        second.append(int(splited_numbers[1]))

    data['passenger_id'] = first
    data['passenger_seq'] = second

    return 'Done'

print(passenger_id(main_data=main_train_data, data=train_data)) # For the␣
↪Training Set
```

```
print(passenger_id(main_data=main_test_data, data=test_data)) # For the␣
↪Training Set
```

```
Done
Done
```

[9]:
```
scaler = StandardScaler()

def transformer(data):

    splitted_data = []
    for all in data['Cabin']:
        splitted_data.append(all.split('/'))

    first = []
    second = []
    third = []

    for all in splitted_data:
        first.append(all[0])
        second.append(all[1])
        third.append(all[2])

    data['cabin_num'] = second
    return 'Done'

print(transformer(data=train_data))
print(transformer(data=test_data))
```

```
Done
Done
```

[10]:
```
train_data = train_data.drop(columns='Cabin')
test_data = test_data.drop(columns='Cabin')
```

0.1.1 Training Set Model Training and Model Prediction

[11]:
```
# Random Forest Initialization
randomforest = RandomForestClassifier(
    n_estimators=100,
    max_depth=2,
    bootstrap=True,
    oob_score=True
)

X = train_data.drop(columns='Transported')
y = train_data['Transported']
```

```
# Training and predicting using training dataset
X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    random_state=25,
    test_size=0.2
)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.fit_transform(X_test)

# Training the model
randomforest.fit(X_train_scaled, y_train)

# Predictions
predict = randomforest.predict(X_test_scaled)

# Classification Report
classification_repo = classification_report(y_test, predict, output_dict=True)

cls_repo = pandas.DataFrame(classification_repo)
```

[12]:
```
print('Classification Report :')
cls_repo
```

```
Classification Report :
```

[12]:

	False	True	accuracy	macro avg	weighted avg
precision	0.675771	0.802980	0.719954	0.739376	0.738169
recall	0.865688	0.568581	0.719954	0.717135	0.719954
f1-score	0.759030	0.665752	0.719954	0.712391	0.713276
support	886.000000	853.000000	0.719954	1739.000000	1739.000000

[13]:
```
print('Out-Of-Bag Score ')
print(randomforest.oob_score_)
```

```
Out-Of-Bag Score
0.7361230946218004
```

XGBoost

[14]:
```
# XGBOOST - X_training

parameters = {
    'estimators':[100, 101, 103, 104],
    'learning_rate':[0.23, 0.24, 0.25, 0.26],
    'max_depth':[1, 2, 1, 3],
}
```

```
listi = ['estimators', 'learning_rate', 'max_depth']

reports = []

itr = 0

while itr < 1:
    for index in range(len(parameters['estimators'])):

        xg = XGBClassifier(
            random_state=42,
            colsample_bylevel=1,
            colsample_bytree=1,
            n_estimators=parameters[listi[itr]][index],
            max_depth=parameters[listi[itr + 2]][index],
            learning_rate=parameters[listi[itr + 1]][index]
        )

        xg.fit(X_train_scaled, y_train) # Training the model

        predict = xg.predict(X_test_scaled)

        classificationreport = classification_report(y_test, predict,␣
 ↪output_dict=True)
        cls_reports = reports.append(classificationreport)
        print(f'Done {cls_reports}')

    break
```

```
Done None
Done None
Done None
Done None
```

[15]:
```
def score_checker(
    report_list, # A list of report
    lookup_score, # What score we want
    type, # both True and false or True or False
    report=None # A simple report also can be given
):

    if report_list:
        if isinstance(report_list, list):
            try:
                if lookup_score:
                    if lookup_score != 'accuracy':
```

```
                        for report in report_list:
                            true_val = report['True']
                            false_val = report['False']

                            true_sorted = sorted(true_val.items(), key=lambda␣
 ↪item: item[1])
                            false_sorted = sorted(false_val.items(), key=lambda␣
 ↪item: item[1])


                            if type == 'both':
                                return {
                                        'True Value':true_sorted,
                                        'False Value':false_sorted
                                    }
                            elif type == 'true':
                                return true_sorted

                            else:
                                return false_sorted
                    else:
                        for report in report_list:
                            accuracy = report[lookup_score]
                            return accuracy

            except (TypeError):
                print('There should be a string given like\nprecision, recall␣
 ↪and so on')
        else:
            pass
    elif report != None:
        pass
    else:
        pass

accuracy = score_checker(report_list=reports, type='true',␣
 ↪lookup_score='accuracy')
print('Accuracy -', round(accuracy, 3))
```

```
Accuracy - 0.75
```

1 Original Predictions

1.1 Random Forest

[16]:
```
# Random Forest
X = train_data.drop(columns=['Transported', 'PassengerId'])
y = train_data['Transported']

X_scaled = scaler.fit_transform(X)

randomforest.fit(X_scaled, y)

# Predicting
predict = randomforest.predict(scaler.fit_transform(test_data.
 ↪drop(columns='PassengerId')))

print('Out-of-box score :')
print(randomforest.oob_score_)
```

```
Out-of-box score :
0.735649373058783
```

1.2 XG Boost Classifier

[17]:
```
xgboost = XGBClassifier(
    n_estimators=100,
    max_depth=1,
    learning_rate=0.01,
    random_state=41
)

xgboost.fit(X_scaled, y)

predict_values = xgboost.predict(scaler.fit_transform(test_data.
 ↪drop(columns='PassengerId')))

predict_values
```

[17]:
```
array([1, 0, 1, …, 1, 0, 1])
```

1.3 SVM (Support Vector Machine)

[18]:
```
svm = SVC(
    kernel='rbf',
    C=1.0,
    gamma='scale',
    class_weight='balanced',
```

```
    probability=True
)

svm.fit(X_scaled, y)

predict = svm.predict(scaler.fit_transform(test_data.
 ↪drop(columns='PassengerId')))

predict
```

[18]: array([True, False, True, …, True, True, True])

[19]:
```
dataframe = pandas.DataFrame({'PassengerID':
 ↪main_test_data['PassengerId'],'Transported':predict}).
 ↪to_csv('second_competition_v2.csv', index=False)
```