hkvrhbiuu

January 23, 2025

[44]:
```python
import pandas as pd
from sklearn.model_selection import train_test_split
import xgboost as xgb
from sklearn.metrics import accuracy_score, classification_report,␣
↪confusion_matrix,mean_squared_error, mean_absolute_error, r2_score
import matplotlib.pyplot as plt
import seaborn as sns
import math
```

[45]:
```python
# Load Titanic dataset
data = pd.read_csv(r"C:\Users\91703\OneDrive\Desktop\TITANIC.csv")
```

[46]:
```python
data.head()
```

[46]:
```
   PassengerId  Survived  Pclass  \
0            1         0       3
1            2         1       1
2            3         1       3
3            4         1       1
4            5         0       3

                                                Name     Sex   Age  SibSp  \
0                            Braund, Mr. Owen Harris    male  22.0      1
1  Cumings, Mrs. John Bradley (Florence Briggs Th…  female  38.0      1
2                             Heikkinen, Miss. Laina  female  26.0      0
3       Futrelle, Mrs. Jacques Heath (Lily May Peel)  female  35.0      1
4                           Allen, Mr. William Henry    male  35.0      0

   Parch            Ticket     Fare Cabin Embarked
0      0         A/5 21171   7.2500   NaN        S
1      0          PC 17599  71.2833   C85        C
2      0  STON/O2. 3101282   7.9250   NaN        S
3      0            113803  53.1000  C123        S
4      0            373450   8.0500   NaN        S
```

[47]:
```python
data.describe()
```

[47]:
```
        PassengerId    Survived      Pclass         Age       SibSp  \
count    891.000000  891.000000  891.000000  714.000000  891.000000
mean     446.000000    0.383838    2.308642   29.699118    0.523008
std      257.353842    0.486592    0.836071   14.526497    1.102743
min        1.000000    0.000000    1.000000    0.420000    0.000000
25%      223.500000    0.000000    2.000000   20.125000    0.000000
50%      446.000000    0.000000    3.000000   28.000000    0.000000
75%      668.500000    1.000000    3.000000   38.000000    1.000000
max      891.000000    1.000000    3.000000   80.000000    8.000000

            Parch        Fare
count  891.000000  891.000000
mean     0.381594   32.204208
std      0.806057   49.693429
min      0.000000    0.000000
25%      0.000000    7.910400
50%      0.000000   14.454200
75%      0.000000   31.000000
max      6.000000  512.329200
```

[48]:
```python
data.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
```

[49]:
```python
# Select relevant columns and convert categorical variables
data = data[['Survived', 'Pclass', 'Sex', 'Age', 'SibSp', 'Parch', 'Fare']]
data['Sex'] = data['Sex'].map({'male': 0, 'female': 1})
```

[50]:
```python
data.head()
```

[50]:
```
   Survived  Pclass  Sex   Age  SibSp  Parch     Fare
0         0       3    0  22.0      1      0   7.2500
1         1       1    1  38.0      1      0  71.2833
2         1       3    1  26.0      0      0   7.9250
3         1       1    1  35.0      1      0  53.1000
4         0       3    0  35.0      0      0   8.0500
```

[51]:
```python
# Handle missing Age values
data['Age'] = pd.to_numeric(data['Age'], errors='coerce')
data['Age'] = data['Age'].fillna(data['Age'].mean())
```

[52]:
```python
# Define target variable (y) and feature variables (X)
y = data['Survived']
X = data[['Pclass', 'Sex', 'Age', 'SibSp', 'Parch', 'Fare']]
```

[53]:
```python
# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)
```

[54]:
```python
# Create and train XGBoost Classifier model
model = xgb.XGBClassifier(objective='binary:logistic', n_estimators=100,␣
 ↪random_state=42)
model.fit(X_train, y_train)
```

[54]:
```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=None, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=None, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              multi_strategy=None, n_estimators=100, n_jobs=None,
              num_parallel_tree=None, random_state=42, …)
```

[55]:
```python
# Make predictions on test data
y_pred = model.predict(X_test)
print(y_pred)
```

```
[0 0 1 1 1 1 1 0 0 1 1 0 0 0 0 1 1 1 0 0 0 1 0 0 0 0 0 0 0 1 0 1 1 1 0 0 0
 1 1 0 0 0 0 0 0 0 0 0 0 1 1 1 0 1 0 1 0 1 1 0 0 1 1 0 0 1 0 1 0 1 1 1 1 1
 0 0 1 1 1 1 0 1 1 0 1 0 1 1 0 0 1 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1
 0 1 1 0 0 0 0 1 0 0 1 1 1 0 1 1 0 0 0 1 0 0 1 1 0 1 1 0 0 0 1 0 0 0 1 0 0
 1 0 0 0 0 0 0 0 0 1 1 1 0 0 0 1 0 0 0 1 0 0 0 1 1 1 0 0 0 1 1]
```

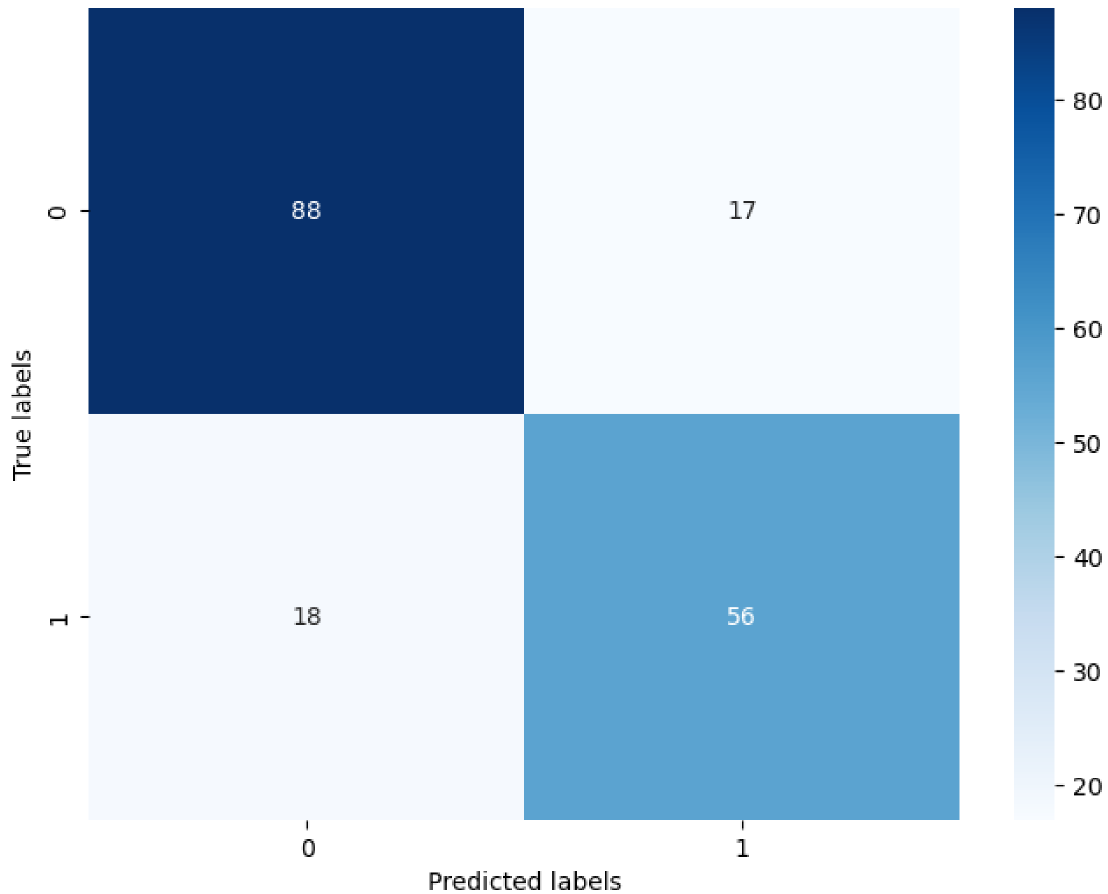
[56]:

```python
# Evaluate model performance
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
print("Classification Report:", classification_report(y_test, y_pred))
print("Confusion Matrix:", confusion_matrix(y_test, y_pred))
```

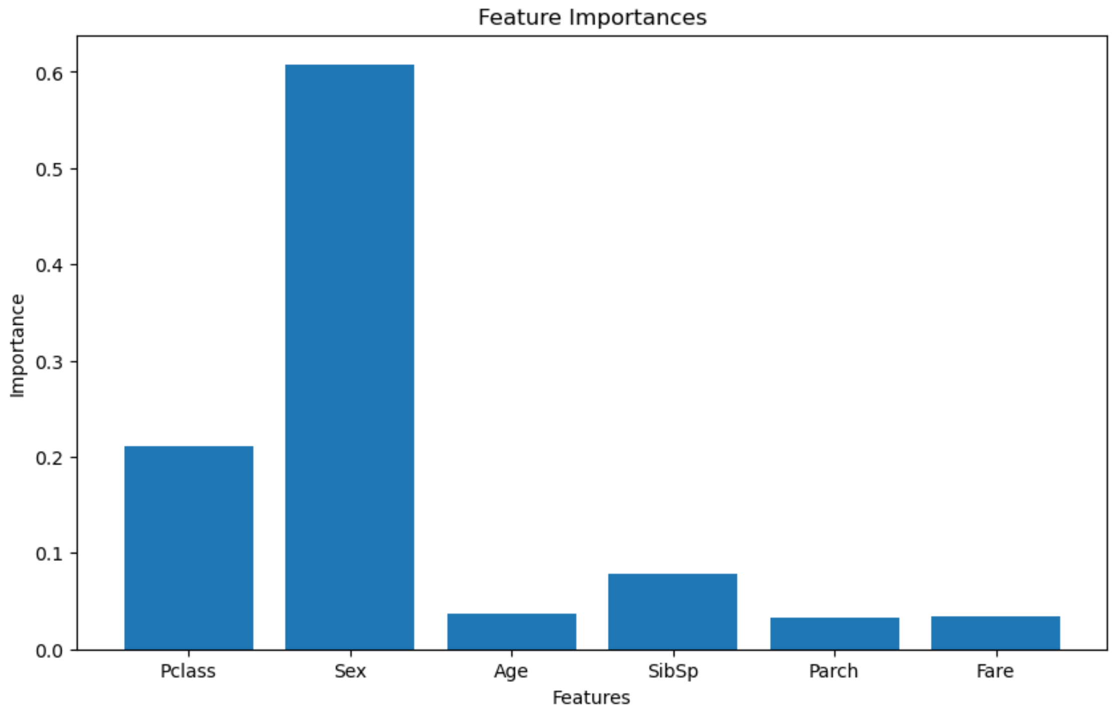
```
Accuracy: 0.8044692737430168
Classification Report:               precision    recall  f1-score   support

           0       0.83      0.84      0.83       105
           1       0.77      0.76      0.76        74

    accuracy                           0.80       179
   macro avg       0.80      0.80      0.80       179
weighted avg       0.80      0.80      0.80       179

Confusion Matrix: [[88 17]
 [18 56]]
```

[57]:

```python
# Plot Confusion Matrix
plt.figure(figsize=(8, 6))
sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, cmap='Blues')
plt.xlabel("Predicted labels")
plt.ylabel("True labels")
plt.show()
```



[58]:
```
#Plot Feature Importances
feature_importances = model.feature_importances_
plt.figure(figsize=(10, 6))
plt.bar(X.columns, feature_importances)
plt.xlabel("Features")
plt.ylabel("Importance")
plt.title("Feature Importances")
plt.show()
```



[59]:
```
# Define target variable (y) and feature variables (X)
y = data['Age']
X = data[['Pclass', 'Sex', 'SibSp', 'Parch', 'Fare']]
```

[60]:
```
# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)
```

[61]:
```
# Create and train XGBoost Regressor model
model = xgb.XGBRegressor(objective='reg:squarederror', n_estimators=100,␣
↪random_state=42)
model.fit(X_train, y_train)
```

[61]:
```
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, device=None, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, feature_types=None,
             gamma=None, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=None, max_bin=None,
             max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=None, max_leaves=None,
             min_child_weight=None, missing=nan, monotone_constraints=None,
             multi_strategy=None, n_estimators=100, n_jobs=None,
             num_parallel_tree=None, random_state=42, …)
```

[62]:
```python
# Make predictions on test data
y_pred = model.predict(X_test)
print(y_pred)
```

```
[34.456184   32.783123   31.337349   27.583454   12.894008   31.473656
 29.464012   19.475527   29.464012   28.124666   24.621515   29.840971
 29.431765   31.094555   31.998625   47.26982    35.157482   21.611671
 31.998625   32.23708    29.390882   50.091675   19.80008    19.897299
 25.972647    9.876298   30.707668   31.998625    9.876298   21.01747
 29.390882   29.464012   39.301514   23.321783   29.840971   26.733416
 42.585773   29.464012   25.319326   29.390882   27.743006   24.906473
 29.840971   33.80651    17.985836   48.399864   28.493532   19.897299
 24.202341   24.283619    0.80724084 43.92668    48.18969    35.89958
 33.80651    29.847437   32.783123   31.010103   30.077312   28.611385
 30.548288   40.837154   28.281397   46.406418   33.80651    35.398415
 18.429457   28.295893   14.822876   51.58079     7.578495   21.083878
 44.3008     49.961662   28.295893   30.643974   29.464012   27.291847
 31.505043   23.677633    5.9831247  35.398415   49.754482   28.022041
 39.301514   28.525219   22.62831    28.47828    34.623486   32.047707
 24.633507    5.954818   39.014965   28.621103   29.390882   28.295893
 35.83145    29.840971   31.505043   29.390882   33.403744   31.512476
 30.90827    24.202341   29.840971   25.321844   51.772957   33.717537
 26.043966   32.23393    35.828175   32.047707   39.704784   39.02451
 31.463264   20.308886   42.5742     30.98962    26.874895   48.461018
  0.69820094 34.785942   50.203407   17.815742   32.047707   50.091675
 47.82472    43.08586    21.743963   29.840971   28.611385   34.456184
 29.83419    29.948875   39.568726   22.571812   24.792702   30.548288
 29.179548   30.314743   33.66125    28.281397   31.998625   36.125763
 34.11547    22.282433   31.998625   32.783123   49.59976    29.840971
 29.9259     31.956175   24.571924   28.51809    29.840971   24.202341
 29.59447    22.045774   28.281397   29.808167   28.21134    50.091675
 31.998625   25.184313   33.66125    50.157597   32.783123   51.962532
 29.390882   24.202341   50.157597   18.922506   39.02451    29.808167
 31.512476   31.094555   40.300625   35.398415   26.42828   ]
```

[63]:
```python
# Evaluate model performance
mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
rmse = math.sqrt(mse)
r2 = r2_score(y_test, y_pred)
print("Mean Squared Error:", mse)
print("Mean Absolute Error:", mae)
print("Root Mean Squared Error:", rmse)
print("R-squared:", r2)
```

```
Mean Squared Error: 154.5745228332585
Mean Absolute Error: 9.080691027556679
Root Mean Squared Error: 12.432800281242296
```

R-squared: 0.08703754490610083

[64]:
```
# Plot predicted vs actual values
plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred)
plt.xlabel("Actual Age")
plt.ylabel("Predicted Age")
plt.show()
```