Titanic

February 20, 2024

1 Titanic data challenge

The goal is to predict whether or not a passenger survived based on attributes such as their age, sex, passenger class, where they embarked and so on.

First, login to Kaggle competition to download train.csv and test.csv. Save them to the titanic directory.

[3]:
```
import pandas as pd
import numpy as np
```

[4]:
```
titanic = pd.read_csv('/Users/apple/Downloads/train.csv', header = 0,␣
↪dtype={'Age': np.float64})
titanic.tail()
```

[4]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
886	887	0	2	Montvila, Rev. Juozas	male	27.0	0	0	211536	13.00	NaN	S
887	888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	112053	30.00	B42	S
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W./C. 6607	23.45	NaN	S
889	890	1	1	Behr, Mr. Karl Howell	male	26.0	0	0	111369	30.00	C148	C
890	891	0	3	Dooley, Mr. Patrick	male	32.0	0	0	370376	7.75	NaN	Q

2 Performing Data Cleaning and Analysis

1. Understanding meaning of each column: Data Dictionary: Variable Description

Survived - Survived (1) or died (0) Pclass - Passenger's class (1 = 1st, 2 = 2nd, 3 = 3rd) Name - Passenger's name Sex - Passenger's sex Age - Passenger's age SibSp - Number of siblings/spouses aboard Parch - Number of parents/children aboard (Some children travelled only with a nanny, therefore parch=0 for them.) Ticket - Ticket number Fare - Fare Cabin - Cabin Embarked - Port of embarkation (C = Cherbourg, Q = Queenstown, S = Southampton)

2. Analysing which columns are completely useless in predicting the survival and deleting them Note - Don't just delete the columns because you are not finding it useful. Or focus is not on deleting the columns. Our focus is on analysing how each column is affecting the result or the prediction and in accordance with that deciding whether to keep the column or to delete the column or fill the null values of the column by some values and if yes, then what values.

[5]:
```
titanic.describe()
```

[5]:
```
       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
```

[6]:
```
#Name column can never decide survival of a person, hence we can safely delete␣
 ↪it
del titanic["Name"]
titanic.head()
```

[6]:
```
   PassengerId  Survived  Pclass     Sex   Age  SibSp  Parch  \
0            1         0       3    male  22.0      1      0
1            2         1       1  female  38.0      1      0
2            3         1       3  female  26.0      0      0
3            4         1       1  female  35.0      1      0
4            5         0       3    male  35.0      0      0

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

[7]:
```
del titanic["Ticket"]
titanic.head()
```

[7]:

	PassengerId	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Cabin	Embarked
0	1	0	3	male	22.0	1	0	7.2500	NaN	S
1	2	1	1	female	38.0	1	0	71.2833	C85	C
2	3	1	3	female	26.0	0	0	7.9250	NaN	S
3	4	1	1	female	35.0	1	0	53.1000	C123	S
4	5	0	3	male	35.0	0	0	8.0500	NaN	S

[8]:
```
del titanic["Fare"]
titanic.head()
```

[8]:

	PassengerId	Survived	Pclass	Sex	Age	SibSp	Parch	Cabin	Embarked
0	1	0	3	male	22.0	1	0	NaN	S
1	2	1	1	female	38.0	1	0	C85	C
2	3	1	3	female	26.0	0	0	NaN	S
3	4	1	1	female	35.0	1	0	C123	S
4	5	0	3	male	35.0	0	0	NaN	S

[9]:
```
del titanic['Cabin']
titanic.head()
```

[9]:

	PassengerId	Survived	Pclass	Sex	Age	SibSp	Parch	Embarked
0	1	0	3	male	22.0	1	0	S
1	2	1	1	female	38.0	1	0	C
2	3	1	3	female	26.0	0	0	S
3	4	1	1	female	35.0	1	0	S
4	5	0	3	male	35.0	0	0	S

[10]:
```
# Changing Value for "Male, Female" string values to numeric values , male=1
 ↪and female=2
def getNumber(str):
    if str=="male":
        return 1
    else:
        return 2
titanic["Gender"]=titanic["Sex"].apply(getNumber)
#We have created a new column called "Gender" and
#filling it with values 1,2 based on the values of sex column
```

```
titanic.head()
```

[10]:

	PassengerId	Survived	Pclass	Sex	Age	SibSp	Parch	Embarked	Gender
0	1	0	3	male	22.0	1	0	S	1
1	2	1	1	female	38.0	1	0	C	2
2	3	1	3	female	26.0	0	0	S	2
3	4	1	1	female	35.0	1	0	S	2
4	5	0	3	male	35.0	0	0	S	1

[11]:
```
#Deleting Sex column, since no use of it now
del titanic["Sex"]
titanic.head()
```

[11]:

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Embarked	Gender
0	1	0	3	22.0	1	0	S	1
1	2	1	1	38.0	1	0	C	2
2	3	1	3	26.0	0	0	S	2
3	4	1	1	35.0	1	0	S	2
4	5	0	3	35.0	0	0	S	1

[12]:
```
titanic.isnull().sum()
```

[12]:
```
PassengerId      0
Survived         0
Pclass           0
Age            177
SibSp            0
Parch            0
Embarked         2
Gender           0
dtype: int64
```

2.0.1 Fill the null values of the Age column. Fill mean Survived age(mean age of the survived people) in the column where the person has survived and mean not Survived age (mean age of the people who have not survived) in the column where person has not survived

[13]:
```
meanS= titanic[titanic.Survived==1].Age.mean()
meanS
```

[13]:
```
28.343689655172415
```

2.0.2 Creating a new "Age" column , filling values in it with a condition if goes True then given values (here meanS) is put in place of last values else nothing happens, simply the values are copied from the "Age" column of the dataset

[14]:
```
titanic["age"]=np.where(pd.isnull(titanic.Age) & titanic["Survived"]==1 ␣
 ↪,meanS, titanic["Age"])
titanic.head()
```

[14]:

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Embarked	Gender	age
0	1	0	3	22.0	1	0	S	1	22.0
1	2	1	1	38.0	1	0	C	2	38.0
2	3	1	3	26.0	0	0	S	2	26.0
3	4	1	1	35.0	1	0	S	2	35.0
4	5	0	3	35.0	0	0	S	1	35.0

[15]:
```
titanic.isnull().sum()
```

[15]:
```
PassengerId      0
Survived         0
Pclass           0
Age            177
SibSp            0
Parch            0
Embarked         2
Gender           0
age            125
dtype: int64
```

[16]:
```
# Finding the mean age of "Not Survived" people
meanNS=titanic[titanic.Survived==0].Age.mean()
meanNS
```

[16]:
```
30.62617924528302
```

[17]:
```
titanic.age.fillna(meanNS,inplace=True)
titanic.head()
```

[17]:

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Embarked	Gender	age
0	1	0	3	22.0	1	0	S	1	22.0
1	2	1	1	38.0	1	0	C	2	38.0
2	3	1	3	26.0	0	0	S	2	26.0
3	4	1	1	35.0	1	0	S	2	35.0
4	5	0	3	35.0	0	0	S	1	35.0

[18]:
```
titanic.isnull().sum()
```

[18]:
```
PassengerId      0
Survived         0
```

```
Pclass           0
Age            177
SibSp            0
Parch            0
Embarked         2
Gender           0
age              0
dtype: int64
```

[19]:
```
del titanic['Age']
titanic.head()
```

[19]:

	PassengerId	Survived	Pclass	SibSp	Parch	Embarked	Gender	age
0	1	0	3	1	0	S	1	22.0
1	2	1	1	1	0	C	2	38.0
2	3	1	3	0	0	S	2	26.0
3	4	1	1	1	0	S	2	35.0
4	5	0	3	0	0	S	1	35.0

2.0.3 We want to check if "Embarked" column is important for analysis or not, that is whether survival of the person depends on the Embarked column value or not

[20]:
```
# Finding the number of people who have survived
# given that they have embarked or boarded from a particular port

survivedQ = titanic[titanic.Embarked == 'Q'][titanic.Survived == 1].shape[0]
survivedC = titanic[titanic.Embarked == 'C'][titanic.Survived == 1].shape[0]
survivedS = titanic[titanic.Embarked == 'S'][titanic.Survived == 1].shape[0]
print(survivedQ)
print(survivedC)
print(survivedS)
```

```
30
93
217
```

```
/var/folders/j9/ryqgqp6n03schy65y0b0g4sm0000gn/T/ipykernel_22873/3300902897.py:4
: UserWarning: Boolean Series key will be reindexed to match DataFrame index.
  survivedQ = titanic[titanic.Embarked == 'Q'][titanic.Survived == 1].shape[0]
/var/folders/j9/ryqgqp6n03schy65y0b0g4sm0000gn/T/ipykernel_22873/3300902897.py:5
: UserWarning: Boolean Series key will be reindexed to match DataFrame index.
  survivedC = titanic[titanic.Embarked == 'C'][titanic.Survived == 1].shape[0]
/var/folders/j9/ryqgqp6n03schy65y0b0g4sm0000gn/T/ipykernel_22873/3300902897.py:6
: UserWarning: Boolean Series key will be reindexed to match DataFrame index.
  survivedS = titanic[titanic.Embarked == 'S'][titanic.Survived == 1].shape[0]
```

[21]:
```
survivedQ = titanic[titanic.Embarked == 'Q'][titanic.Survived == 0].shape[0]
survivedC = titanic[titanic.Embarked == 'C'][titanic.Survived == 0].shape[0]
survivedS = titanic[titanic.Embarked == 'S'][titanic.Survived == 0].shape[0]
print(survivedQ)
print(survivedC)
print(survivedS)
```

```
47
75
427
```

```
/var/folders/j9/ryqgqp6n03schy65y0b0g4sm0000gn/T/ipykernel_22873/3240960939.py:1
: UserWarning: Boolean Series key will be reindexed to match DataFrame index.
  survivedQ = titanic[titanic.Embarked == 'Q'][titanic.Survived == 0].shape[0]
/var/folders/j9/ryqgqp6n03schy65y0b0g4sm0000gn/T/ipykernel_22873/3240960939.py:2
: UserWarning: Boolean Series key will be reindexed to match DataFrame index.
  survivedC = titanic[titanic.Embarked == 'C'][titanic.Survived == 0].shape[0]
/var/folders/j9/ryqgqp6n03schy65y0b0g4sm0000gn/T/ipykernel_22873/3240960939.py:3
: UserWarning: Boolean Series key will be reindexed to match DataFrame index.
  survivedS = titanic[titanic.Embarked == 'S'][titanic.Survived == 0].shape[0]
```

[22]:
```
30/(30 + 47), 93/(93+75), 217/(427+217)
```

[22]:
```
(0.38961038961038963, 0.5535714285714286, 0.33695652173913043)
```

As there are significant changes in the survival rate based on which port the passengers aboard the ship. We cannot delete the whole embarked column(It is useful). Now the Embarked column has some null values in it and hence we can safely say that deleting some rows from total rows will not affect the result. So rather than trying to fill those null values with some vales. We can simply remove them.

[23]:
```
titanic.dropna(inplace=True)
titanic.head()
```

[23]:

	PassengerId	Survived	Pclass	SibSp	Parch	Embarked	Gender	age
0	1	0	3	1	0	S	1	22.0
1	2	1	1	1	0	C	2	38.0
2	3	1	3	0	0	S	2	26.0
3	4	1	1	1	0	S	2	35.0
4	5	0	3	0	0	S	1	35.0

[24]:
```
titanic.isnull().sum()
```

[24]:
```
PassengerId    0
Survived       0
Pclass         0
SibSp          0
Parch          0
Embarked       0
```

```
Gender          0
age             0
dtype: int64
```

[25]:
```
#Renaming "age" and "gender" columns
titanic.rename(columns={'age':'Age'}, inplace=True)
titanic.head()
```

[25]:

	PassengerId	Survived	Pclass	SibSp	Parch	Embarked	Gender	Age
0	1	0	3	1	0	S	1	22.0
1	2	1	1	1	0	C	2	38.0
2	3	1	3	0	0	S	2	26.0
3	4	1	1	1	0	S	2	35.0
4	5	0	3	0	0	S	1	35.0

[26]:
```
titanic.rename(columns={'Gender':'Sex'}, inplace=True)
titanic.head()
```

[26]:

	PassengerId	Survived	Pclass	SibSp	Parch	Embarked	Sex	Age
0	1	0	3	1	0	S	1	22.0
1	2	1	1	1	0	C	2	38.0
2	3	1	3	0	0	S	2	26.0
3	4	1	1	1	0	S	2	35.0
4	5	0	3	0	0	S	1	35.0

[27]:
```
def getS(str):
    if str=="S":
        return 1
    else:
        return 0
titanic["S"]=titanic["Embarked"].apply(getS)
def getQ(str):
    if str=="Q":
        return 1
    else:
        return 0
titanic["Q"]=titanic["Embarked"].apply(getQ)
def getC(str):
    if str=="C":
        return 1
    else:
        return 0
titanic["C"]=titanic["Embarked"].apply(getC)
titanic.head()
```

[27]:

	PassengerId	Survived	Pclass	SibSp	Parch	Embarked	Sex	Age	S	Q	C
0	1	0	3	1	0	S	1	22.0	1	0	0

1	2	1	1	1	0	C	2	38.0	0	0	1
2	3	1	3	0	0	S	2	26.0	1	0	0
3	4	1	1	1	0	S	2	35.0	1	0	0
4	5	0	3	0	0	S	1	35.0	1	0	0

[28]:
```
del titanic['Embarked']
titanic.head()
```

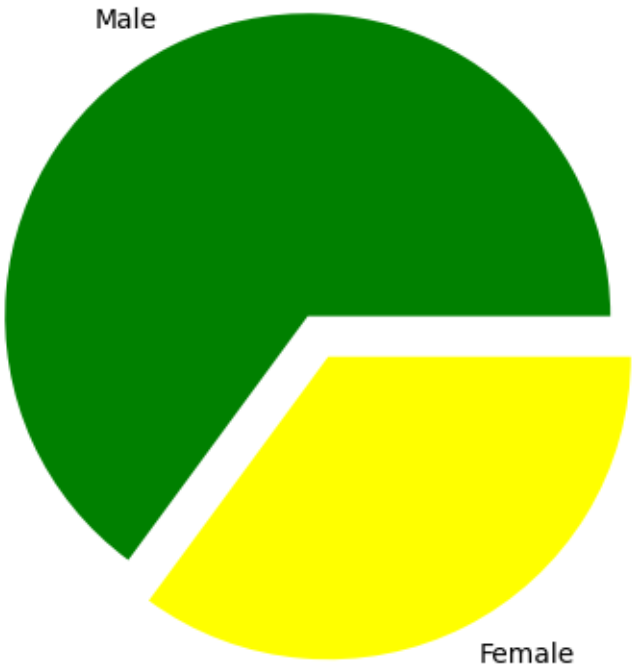
[28]:

	PassengerId	Survived	Pclass	SibSp	Parch	Sex	Age	S	Q	C
0	1	0	3	1	0	1	22.0	1	0	0
1	2	1	1	1	0	2	38.0	0	0	1
2	3	1	3	0	0	2	26.0	1	0	0
3	4	1	1	1	0	2	35.0	1	0	0
4	5	0	3	0	0	1	35.0	1	0	0

[29]:
```
#Drawing a pie chart for number of males and females aboard
import matplotlib.pyplot as plt
from matplotlib import style

males = (titanic['Sex'] == 1).sum()
#Summing up all the values of column gender with a
#condition for male and similary for females
females = (titanic['Sex'] == 2).sum()
print(males)
print(females)
p = [males, females]
plt.pie(p,    #giving array
       labels = ['Male', 'Female'], #Correspndingly giving labels
       colors = ['green', 'yellow'],   # Corresponding colors
       explode = (0.15, 0),    #How much the gap should me there between the␣
 ↪pies
       startangle = 0)  #what start angle should be given
plt.axis('equal')
plt.show()
```

```
577
312
```



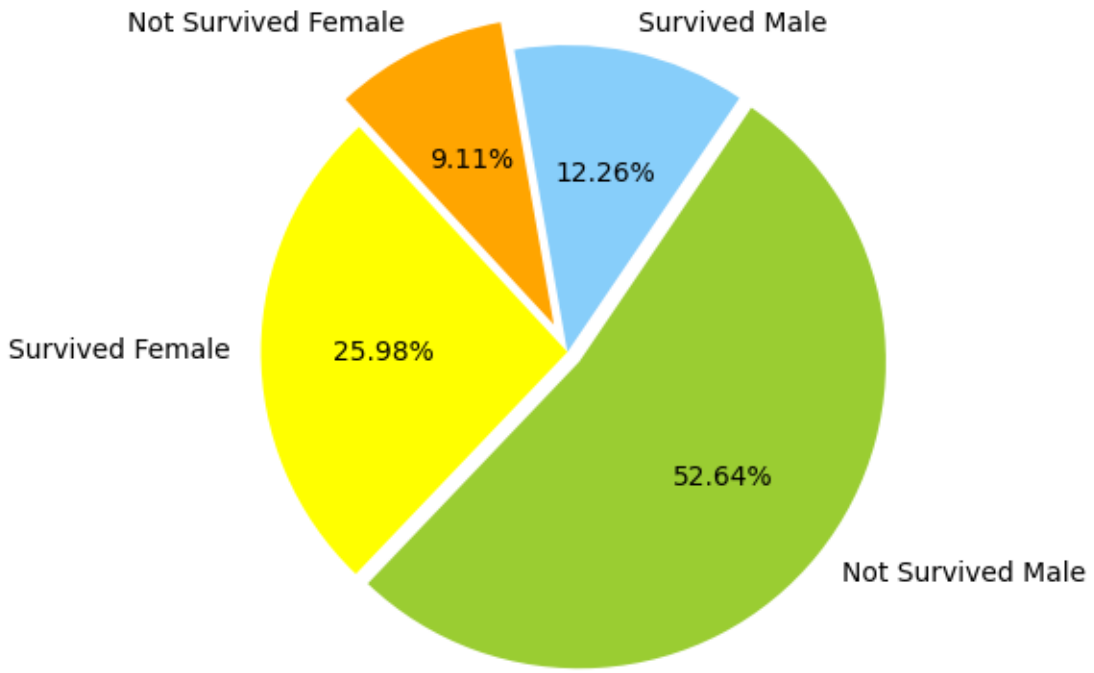
[30]:
```
# More Precise Pie Chart
MaleS=titanic[titanic.Sex==1][titanic.Survived==1].shape[0]
print(MaleS)
MaleN=titanic[titanic.Sex==1][titanic.Survived==0].shape[0]
print(MaleN)
FemaleS=titanic[titanic.Sex==2][titanic.Survived==1].shape[0]
print(FemaleS)
FemaleN=titanic[titanic.Sex==2][titanic.Survived==0].shape[0]
print(FemaleN)
```

```
109
468
231
81
```

```
/var/folders/j9/ryqgqp6n03schy65y0b0g4sm0000gn/T/ipykernel_22873/3105620411.py:2
: UserWarning: Boolean Series key will be reindexed to match DataFrame index.
  MaleS=titanic[titanic.Sex==1][titanic.Survived==1].shape[0]
/var/folders/j9/ryqgqp6n03schy65y0b0g4sm0000gn/T/ipykernel_22873/3105620411.py:4
: UserWarning: Boolean Series key will be reindexed to match DataFrame index.
  MaleN=titanic[titanic.Sex==1][titanic.Survived==0].shape[0]
/var/folders/j9/ryqgqp6n03schy65y0b0g4sm0000gn/T/ipykernel_22873/3105620411.py:6
: UserWarning: Boolean Series key will be reindexed to match DataFrame index.
  FemaleS=titanic[titanic.Sex==2][titanic.Survived==1].shape[0]
```

```
/var/folders/j9/ryqgqp6n03schy65y0b0g4sm0000gn/T/ipykernel_22873/3105620411.py:8
: UserWarning: Boolean Series key will be reindexed to match DataFrame index.
  FemaleN=titanic[titanic.Sex==2][titanic.Survived==0].shape[0]
```

[31]:
```
chart=[MaleS,MaleN,FemaleS,FemaleN]
colors=['lightskyblue','yellowgreen','Yellow','Orange']
labels=["Survived Male","Not Survived Male","Survived Female","Not Survived␣
 ↪Female"]
explode=[0,0.05,0,0.1]
plt.
 ↪pie(chart,labels=labels,colors=colors,explode=explode,startangle=100,counterclock=False,aut
 ↪2f%%")
plt.axis("equal")
plt.show()
```



[32]:
```
y_Train = titanic["Survived"]
del titanic['Survived']
X_Train = titanic
X_Train.head()
```

[32]:

	PassengerId	Pclass	SibSp	Parch	Sex	Age	S	Q	C
0	1	3	1	0	1	22.0	1	0	0
1	2	1	1	0	2	38.0	0	0	1

```
2            3       3       0       0    2  26.0  1  0  0
3            4       1       1       0    2  35.0  1  0  0
4            5       3       0       0    1  35.0  1  0  0
```

The data is already split into a training set and a test set. We need to preprocess the test data in the same step as we have done for the training data.

2.1 Homework: Training classifiers

We are now ready to train classifiers.

We first split the training data into training and validation sets. You can use cross validation as well.

Train the following three classifers:

Logistic regression

SVM

KNN classifier

Report the features of each model, the confusion matrix, classification summary, and AUC.

Select the best model and use it to predict the test set. Submit your predictions in the test set.

[33]:
```
from sklearn.metrics import confusion_matrix, classification_report,␣
 ↪roc_auc_score, roc_curve, auc
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler
```

[34]:
```
# Standardize features

scaler = StandardScaler()

X_scaled = scaler.fit_transform(X_Train)

# Split dataset into train set and validation set

X_train, X_valid, y_train, y_valid = train_test_split(X_scaled, y_Train,␣
 ↪test_size=0.2, random_state=42)
```

[35]:
```
from sklearn.model_selection import GridSearchCV

# Parameter grid for logistic regression

param_grid_lr = {
```

```
'C': [0.01, 0.1, 1, 10, 100],
'penalty': ['l1', 'l2'],
'solver': ['liblinear'] # A good choice for l1 regularization
}

# Initialize grid search
grid_search_lr = GridSearchCV(LogisticRegression(max_iter=1000), param_grid_lr,␣
 ↪cv=5, scoring='accuracy')
grid_search_lr.fit(X_train, y_train)

# Print best parameters and best accuracy
print("Logistic Regression Best Parameters:", grid_search_lr.best_params_)
print("Best Accuracy:", grid_search_lr.best_score_)
```

```
Logistic Regression Best Parameters: {'C': 1, 'penalty': 'l1', 'solver':
'liblinear'}
Best Accuracy: 0.801674381956072
```

[36]:
```
# Parameter grid for SVM

param_grid_svm = {
'C': [0.1, 1, 10, 100],
'gamma': [1, 0.1, 0.01, 0.001],
'kernel': ['rbf', 'linear'] # Commonly used kernel functions
}

# Initialize grid search
grid_search_svm = GridSearchCV(SVC(probability=True), param_grid_svm, cv=5,␣
 ↪scoring='accuracy')
grid_search_svm.fit(X_train, y_train)

# Print best parameters and best accuracy
print("SVM Best Parameters:", grid_search_svm.best_params_)
print("Best Accuracy:", grid_search_svm.best_score_)
```

```
SVM Best Parameters: {'C': 10, 'gamma': 0.1, 'kernel': 'rbf'}
Best Accuracy: 0.8255687973997834
```

[37]:
```
# Parameter grid for KNN

param_grid_knn = {
'n_neighbors': [3, 5, 7, 9, 11],
'weights': ['uniform', 'distance'],
'metric': ['euclidean', 'manhattan']
}

# Initialize grid search
```

```
grid_search_knn = GridSearchCV(KNeighborsClassifier(), param_grid_knn, cv=5,␣
 ↪scoring='accuracy')
grid_search_knn.fit(X_train, y_train)

# Print best parameters and best accuracy
print("KNN Best Parameters:", grid_search_knn.best_params_)
print("Best Accuracy:", grid_search_knn.best_score_)
```

```
KNN Best Parameters: {'metric': 'manhattan', 'n_neighbors': 11, 'weights':
'uniform'}
Best Accuracy: 0.8255983453166552
```

[38]:
```
models = {
    "Logistic Regression": LogisticRegression(C=1, penalty='l1',␣
 ↪solver='liblinear', max_iter=1000, random_state=24066039),
    "SVM": SVC(C=10, gamma=0.1, kernel='rbf', probability=True,␣
 ↪random_state=24066039),
    "KNN": KNeighborsClassifier(metric='manhattan', n_neighbors=11,␣
 ↪weights='uniform')
}

# Train models and calculate metrics
results_updated = {}

print("Model Evaluation Results:\n")

for name, model in models.items():
    # Train model
    model.fit(X_train, y_train)
    # Predict
    y_pred = model.predict(X_valid)
    y_prob = model.predict_proba(X_valid)[:, 1] if hasattr(model,␣
 ↪"predict_proba") else model.decision_function(X_valid)
    # Calculate metrics
    cm = confusion_matrix(y_valid, y_pred)
    cr = classification_report(y_valid, y_pred)
    auc_score = roc_auc_score(y_valid, y_prob)
    # Store results
    results_updated[name] = {"Confusion Matrix": cm, "Classification Report":␣
 ↪cr, "AUC": auc_score}

    # Print results
    print(f"--- {name} ---")
    print("Confusion Matrix:\n", cm)
    print("\nClassification Report:\n", cr)
    print(f"AUC: {auc_score:.4f}\n")
```

```python
    # Feature importances (Logistic regression only)
    if name == "Logistic Regression":
        feature_importance = model.coef_[0]
        print("Feature Importances:")
        for i, col in enumerate(X_Train.columns):
            print(f"{col}: {feature_importance[i]:.4f}")
        print("\n")
```

```
Model Evaluation Results:

--- Logistic Regression ---
Confusion Matrix:
 [[84 25]
 [15 54]]

Classification Report:
              precision    recall  f1-score   support

           0       0.85      0.77      0.81       109
           1       0.68      0.78      0.73        69

    accuracy                           0.78       178
   macro avg       0.77      0.78      0.77       178
weighted avg       0.78      0.78      0.78       178

AUC: 0.8529

Feature Importances:
PassengerId: 0.0289
Pclass: -1.0059
SibSp: -0.4122
Parch: -0.0374
Sex: 1.3067
Age: -0.6113
S: -0.1978
Q: 0.0000
C: 0.0336


--- SVM ---
Confusion Matrix:
 [[93 16]
 [20 49]]

Classification Report:
              precision    recall  f1-score   support

           0       0.82      0.85      0.84       109
```

```
           1       0.75      0.71      0.73        69

    accuracy                           0.80       178
   macro avg       0.79      0.78      0.78       178
weighted avg       0.80      0.80      0.80       178

AUC: 0.7894

--- KNN ---
Confusion Matrix:
 [[91 18]
 [20 49]]

Classification Report:
              precision    recall  f1-score   support

           0       0.82      0.83      0.83       109
           1       0.73      0.71      0.72        69

    accuracy                           0.79       178
   macro avg       0.78      0.77      0.77       178
weighted avg       0.79      0.79      0.79       178

AUC: 0.8415
```

Among these three models, logistic regression performed the best in terms of AUC value, indicating that it was the most excellent at distinguishing between surviving and non-surviving passengers. SVM slightly outperformed in accuracy but was slightly inferior in AUC value, while KNN struck a balance between the two in performance.

We then choose logistic regression to predict the result on test set.

Now we preprocess test set data

[39]:
```
test_data = pd.read_csv("/Users/apple/Downloads/test.csv")
test_data.head()
```

[39]:

	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							
4	896	3	Hirvonen, Mrs. Alexander (Helga E Lindqvist)	female							

```
3  27.0      0      0   315154   8.6625   NaN        S
4  22.0      1      1  3101298  12.2875   NaN        S
```

[40]:
```
test_data.describe()
```

[40]:
```
       PassengerId      Pclass         Age       SibSp       Parch        Fare
count   418.000000  418.000000  332.000000  418.000000  418.000000  417.000000
mean   1100.500000    2.265550   30.272590    0.447368    0.392344   35.627188
std     120.810458    0.841838   14.181209    0.896760    0.981429   55.907576
min     892.000000    1.000000    0.170000    0.000000    0.000000    0.000000
25%     996.250000    1.000000   21.000000    0.000000    0.000000    7.895800
50%    1100.500000    3.000000   27.000000    0.000000    0.000000   14.454200
75%    1204.750000    3.000000   39.000000    1.000000    0.000000   31.500000
max    1309.000000    3.000000   76.000000    8.000000    9.000000  512.329200
```

[41]:
```
#deal with the test data
del test_data["Name"]
del test_data["Ticket"]
del test_data["Fare"]
del test_data['Cabin']
```

[42]:
```
# Changing Value for "Male, Female" string values to numeric values , male=1␣
 ↪and female=2
def getNumber(str):
    if str=="male":
        return 1
    else:
        return 2
test_data["Gender"]=test_data["Sex"].apply(getNumber)
del test_data["Sex"]
```

[43]:
```
mean= test_data.Age.mean()
mean
test_data["age"]=np.where(pd.isnull(test_data.Age) ,mean, test_data["Age"])
del test_data['Age']
```

[44]:
```
test_data.dropna(inplace=True)
test_data.head()
test_data.isnull().sum()
```

[44]:
```
PassengerId    0
Pclass         0
SibSp          0
Parch          0
Embarked       0
Gender         0
age            0
```

```
dtype: int64
```

[45]:
```
#Renaming "age" and "gender" columns
test_data.rename(columns={'age':'Age'}, inplace=True)
test_data.rename(columns={'Gender':'Sex'}, inplace=True)
test_data.head()
```

[45]:

	PassengerId	Pclass	SibSp	Parch	Embarked	Sex	Age
0	892	3	0	0	Q	1	34.5
1	893	3	1	0	S	2	47.0
2	894	2	0	0	Q	1	62.0
3	895	3	0	0	S	1	27.0
4	896	3	1	1	S	2	22.0

[46]:
```
def getS(str):
    if str=="S":
        return 1
    else:
        return 0
test_data["S"]=test_data["Embarked"].apply(getS)
def getQ(str):
    if str=="Q":
        return 1
    else:
        return 0
test_data["Q"]=test_data["Embarked"].apply(getQ)
def getC(str):
    if str=="C":
        return 1
    else:
        return 0
test_data["C"]=test_data["Embarked"].apply(getC)
del test_data['Embarked']
test_data.head()
```

[46]:

	PassengerId	Pclass	SibSp	Parch	Sex	Age	S	Q	C
0	892	3	0	0	1	34.5	0	1	0
1	893	3	1	0	2	47.0	1	0	0
2	894	2	0	0	1	62.0	0	1	0
3	895	3	0	0	1	27.0	1	0	0
4	896	3	1	1	2	22.0	1	0	0

[47]:
```
# Standardize the test data
test_data_scaled = scaler.transform(test_data)

logistic_regression_model = models["Logistic Regression"]
```

```
# Make predictions on the test set using logistic regression model
y_test_pred = logistic_regression_model.predict(test_data_scaled)

test_data['pred'] = y_test_pred

# Create submission DataFrame
test_pred_result = test_data

# Save submission DataFrame to CSV file without row indexes
test_pred_result.to_csv('/Users/apple/Downloads/test_pred_result.csv',␣
 ↪index=False)

# Print file path for downloading
print('test_pred_result file saved to /Users/apple/Downloads/test_pred_result.
 ↪csv')
```

```
test_pred_result file saved to /Users/apple/Downloads/test_pred_result.csv
```

[ ]:

[ ]: