Space X Falcon 9 First Stage Landing Prediction

Assignment: Machine Learning Prediction

Estimated time needed: **60** minutes

Space X advertises Falcon 9 rocket launches on its website with a cost of 62 million dollars; other providers cost upward of 165 million dollars each, much of the savings is because Space X can reuse the first stage. Therefore if we can determine if the first stage will land, we can determine the cost of a launch. This information can be used if an alternate company wants to bid against space X for a rocket launch. In this lab, you will create a machine learning pipeline to predict if the first stage will land given the data from the preceding labs.

Several examples of an unsuccessful landing are shown here:

Most unsuccessful landings are planed. Space X; performs a controlled landing in the oceans.

Objectives

Perform exploratory Data Analysis and determine Training Labels

- create a column for the class
- Standardize the data
- Split into training data and test data

-Find best Hyperparameter for SVM, Classification Trees and Logistic Regression

- Find the method performs best using test data

Import Libraries and Define Auxiliary Functions

We will import the following libraries for the lab

In [1]:
```
# Pandas is a software library written for the Python programming language for
data manipulation and analysis.
import pandas as pd
# NumPy is a library for the Python programming language, adding support for l
arge, multi-dimensional arrays and matrices, along with a large collection of
 high-level mathematical functions to operate on these arrays
import numpy as np
# Matplotlib is a plotting library for python and pyplot gives us a MatLab lik
e plotting framework. We will use this in our plotter function to plot data.
import matplotlib.pyplot as plt
#Seaborn is a Python data visualization library based on matplotlib. It provid
es a high-level interface for drawing attractive and informative statistical g
raphics
import seaborn as sns
# Preprocessing allows us to standarsize our data
from sklearn import preprocessing
# Allows us to split our data into training and testing data
from sklearn.model_selection import train_test_split
# Allows us to test parameters of classification algorithms and find the best
 one
from sklearn.model_selection import GridSearchCV
# Logistic Regression classification algorithm
from sklearn.linear_model import LogisticRegression
# Support Vector Machine classification algorithm
from sklearn.svm import SVC
# Decision Tree classification algorithm
from sklearn.tree import DecisionTreeClassifier
# K Nearest Neighbors classification algorithm
from sklearn.neighbors import KNeighborsClassifier
```

This function is to plot the confusion matrix.

In [2]:
```
def plot_confusion_matrix(y,y_predict):
    "this function plots the confusion matrix"
    from sklearn.metrics import confusion_matrix

    cm = confusion_matrix(y, y_predict)
    ax= plt.subplot()
    sns.heatmap(cm, annot=True, ax = ax); #annot=True to annotate cells
    ax.set_xlabel('Predicted labels')
    ax.set_ylabel('True labels')
    ax.set_title('Confusion Matrix');
    ax.xaxis.set_ticklabels(['did not land', 'land']); ax.yaxis.set_ticklabels
(['did not land', 'landed'])
```

Load the dataframe

Load the data

In [3]:
```
#data = pd.read_csv("https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DS0321EN-SkillsNetwork/datasets/dataset_part_2.csv")

# If you were unable to complete the previous lab correctly you can uncomment and load this csv

data = pd.read_csv('https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperSkillsNetwork-DS0701EN-SkillsNetwork/api/dataset_part_2.csv')

data.head()
```

Out[3]:

	FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Flights	Gric
0	1	2010-06-04	Falcon 9	6104.959412	LEO	CCAFS SLC 40	None None	1	
1	2	2012-05-22	Falcon 9	525.000000	LEO	CCAFS SLC 40	None None	1	
2	3	2013-03-01	Falcon 9	677.000000	ISS	CCAFS SLC 40	None None	1	
3	4	2013-09-29	Falcon 9	500.000000	PO	VAFB SLC 4E	False Ocean	1	
4	5	2013-12-03	Falcon 9	3170.000000	GTO	CCAFS SLC 40	None None	1	

In [11]:
```
#X = pd.read_csv('https://cf-courses-data.s3.us.cloud-object-storage.appdomai
n.cloud/IBM-DS0321EN-SkillsNetwork/datasets/dataset_part_3.csv')

# If you were unable to complete the previous lab correctly you can uncomment
 and load this csv

X = pd.read_csv('https://cf-courses-data.s3.us.cloud-object-storage.appdomain.
cloud/IBMDeveloperSkillsNetwork-DS0701EN-SkillsNetwork/api/dataset_part_3.csv'
)

X.head(100)
```

Out[11]:

	FlightNumber	PayloadMass	Flights	Block	ReusedCount	Orbit_ES-L1	Orbit_GEO	Orbit_GTO
0	1.0	6104.959412	1.0	1.0	0.0	0.0	0.0	0.0
1	2.0	525.000000	1.0	1.0	0.0	0.0	0.0	0.0
2	3.0	677.000000	1.0	1.0	0.0	0.0	0.0	0.0
3	4.0	500.000000	1.0	1.0	0.0	0.0	0.0	0.0
4	5.0	3170.000000	1.0	1.0	0.0	0.0	0.0	1.0
...	...	...	...	...	...	...	...	...
85	86.0	15400.000000	2.0	5.0	2.0	0.0	0.0	0.0
86	87.0	15400.000000	3.0	5.0	2.0	0.0	0.0	0.0
87	88.0	15400.000000	6.0	5.0	5.0	0.0	0.0	0.0
88	89.0	15400.000000	3.0	5.0	2.0	0.0	0.0	0.0
89	90.0	3681.000000	1.0	5.0	0.0	0.0	0.0	0.0

90 rows × 83 columns

TASK 1

Create a NumPy array from the column `Class` in `data`, by applying the method `to_numpy()` then assign it to the variable `Y`,make sure the output is a Pandas series (only one bracket df['name of column']).

In [7]:
```
Y = data['Class'].to_numpy()

Y
```

Out[7]:
```
array([0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1,
       1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1,
       1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1])
```

TASK 2

Standardize the data in `X` then reassign it to the variable `X` using the transform provided below.

In [12]:
```
# students get this
transform = preprocessing.StandardScaler()
```

In [13]:
```
X = transform.fit(X).transform(X)
X[0:5]
```

Out[13]:
```
array([[-1.71291154e+00, -1.94814463e-16, -6.53912840e-01,
        -1.57589457e+00, -9.73440458e-01, -1.05999788e-01,
        -1.05999788e-01, -6.54653671e-01, -1.05999788e-01,
        -5.51677284e-01,  3.44342023e+00, -1.85695338e-01,
        -3.33333333e-01, -1.05999788e-01, -2.42535625e-01,
        -4.29197538e-01,  7.97724035e-01, -5.68796459e-01,
        -4.10890702e-01, -4.10890702e-01, -1.50755672e-01,
        -7.97724035e-01, -1.50755672e-01, -3.92232270e-01,
         9.43398113e+00, -1.05999788e-01, -1.05999788e-01,
        -1.05999788e-01, -1.05999788e-01, -1.05999788e-01,
        -1.05999788e-01, -1.05999788e-01, -1.05999788e-01,
        -1.05999788e-01, -1.05999788e-01, -1.05999788e-01,
        -1.05999788e-01, -1.05999788e-01, -1.05999788e-01,
        -1.05999788e-01, -1.05999788e-01, -1.05999788e-01,
        -1.05999788e-01, -1.50755672e-01, -1.05999788e-01,
        -1.05999788e-01, -1.05999788e-01, -1.05999788e-01,
        -1.05999788e-01, -1.50755672e-01, -1.05999788e-01,
        -1.50755672e-01, -1.50755672e-01, -1.05999788e-01,
        -1.50755672e-01, -1.50755672e-01, -1.05999788e-01,
        -1.05999788e-01, -1.50755672e-01, -1.50755672e-01,
        -1.50755672e-01, -1.05999788e-01, -1.05999788e-01,
        -1.05999788e-01, -1.50755672e-01, -2.15665546e-01,
        -1.85695338e-01, -2.15665546e-01, -2.67261242e-01,
        -1.05999788e-01, -2.42535625e-01, -1.05999788e-01,
        -2.15665546e-01, -1.85695338e-01, -2.15665546e-01,
        -1.85695338e-01, -1.05999788e-01,  1.87082869e+00,
        -1.87082869e+00,  8.35531692e-01, -8.35531692e-01,
         1.93309133e+00, -1.93309133e+00],
       [-1.67441914e+00, -1.19523159e+00, -6.53912840e-01,
        -1.57589457e+00, -9.73440458e-01, -1.05999788e-01,
        -1.05999788e-01, -6.54653671e-01, -1.05999788e-01,
        -5.51677284e-01,  3.44342023e+00, -1.85695338e-01,
        -3.33333333e-01, -1.05999788e-01, -2.42535625e-01,
        -4.29197538e-01,  7.97724035e-01, -5.68796459e-01,
        -4.10890702e-01, -4.10890702e-01, -1.50755672e-01,
        -7.97724035e-01, -1.50755672e-01, -3.92232270e-01,
        -1.05999788e-01,  9.43398113e+00, -1.05999788e-01,
        -1.05999788e-01, -1.05999788e-01, -1.05999788e-01,
        -1.05999788e-01, -1.05999788e-01, -1.05999788e-01,
        -1.05999788e-01, -1.05999788e-01, -1.05999788e-01,
        -1.05999788e-01, -1.05999788e-01, -1.05999788e-01,
        -1.05999788e-01, -1.05999788e-01, -1.05999788e-01,
        -1.05999788e-01, -1.50755672e-01, -1.05999788e-01,
        -1.05999788e-01, -1.05999788e-01, -1.05999788e-01,
        -1.05999788e-01, -1.50755672e-01, -1.05999788e-01,
        -1.50755672e-01, -1.50755672e-01, -1.05999788e-01,
        -1.50755672e-01, -1.50755672e-01, -1.05999788e-01,
        -1.05999788e-01, -1.50755672e-01, -1.50755672e-01,
        -1.50755672e-01, -1.05999788e-01, -1.05999788e-01,
        -1.05999788e-01, -1.50755672e-01, -2.15665546e-01,
        -1.85695338e-01, -2.15665546e-01, -2.67261242e-01,
        -1.05999788e-01, -2.42535625e-01, -1.05999788e-01,
        -2.15665546e-01, -1.85695338e-01, -2.15665546e-01,
        -1.85695338e-01, -1.05999788e-01,  1.87082869e+00,
        -1.87082869e+00,  8.35531692e-01, -8.35531692e-01,
         1.93309133e+00, -1.93309133e+00],
       [-1.63592675e+00, -1.16267307e+00, -6.53912840e-01,
```

```
 -1.57589457e+00, -9.73440458e-01, -1.05999788e-01,
 -1.05999788e-01, -6.54653671e-01, -1.05999788e-01,
  1.81265393e+00, -2.90408935e-01, -1.85695338e-01,
 -3.33333333e-01, -1.05999788e-01, -2.42535625e-01,
 -4.29197538e-01,  7.97724035e-01, -5.68796459e-01,
 -4.10890702e-01, -4.10890702e-01, -1.50755672e-01,
 -7.97724035e-01, -1.50755672e-01, -3.92232270e-01,
 -1.05999788e-01, -1.05999788e-01,  9.43398113e+00,
 -1.05999788e-01, -1.05999788e-01, -1.05999788e-01,
 -1.05999788e-01, -1.05999788e-01, -1.05999788e-01,
 -1.05999788e-01, -1.05999788e-01, -1.05999788e-01,
 -1.05999788e-01, -1.05999788e-01, -1.05999788e-01,
 -1.05999788e-01, -1.05999788e-01, -1.05999788e-01,
 -1.05999788e-01, -1.50755672e-01, -1.05999788e-01,
 -1.05999788e-01, -1.05999788e-01, -1.05999788e-01,
 -1.05999788e-01, -1.50755672e-01, -1.05999788e-01,
 -1.50755672e-01, -1.50755672e-01, -1.05999788e-01,
 -1.50755672e-01, -1.50755672e-01, -1.05999788e-01,
 -1.05999788e-01, -1.50755672e-01, -1.50755672e-01,
 -1.50755672e-01, -1.05999788e-01, -1.05999788e-01,
 -1.05999788e-01, -1.50755672e-01, -2.15665546e-01,
 -1.85695338e-01, -2.15665546e-01, -2.67261242e-01,
 -1.05999788e-01, -2.42535625e-01, -1.05999788e-01,
 -2.15665546e-01, -1.85695338e-01, -2.15665546e-01,
 -1.85695338e-01, -1.05999788e-01,  1.87082869e+00,
 -1.87082869e+00,  8.35531692e-01, -8.35531692e-01,
  1.93309133e+00, -1.93309133e+00],
[-1.59743435e+00, -1.20058661e+00, -6.53912840e-01,
 -1.57589457e+00, -9.73440458e-01, -1.05999788e-01,
 -1.05999788e-01, -6.54653671e-01, -1.05999788e-01,
 -5.51677284e-01, -2.90408935e-01, -1.85695338e-01,
  3.00000000e+00, -1.05999788e-01, -2.42535625e-01,
 -4.29197538e-01, -1.25356634e+00, -5.68796459e-01,
  2.43373723e+00, -4.10890702e-01, -1.50755672e-01,
 -7.97724035e-01, -1.50755672e-01, -3.92232270e-01,
 -1.05999788e-01, -1.05999788e-01, -1.05999788e-01,
  9.43398113e+00, -1.05999788e-01, -1.05999788e-01,
 -1.05999788e-01, -1.05999788e-01, -1.05999788e-01,
 -1.05999788e-01, -1.05999788e-01, -1.05999788e-01,
 -1.05999788e-01, -1.05999788e-01, -1.05999788e-01,
 -1.05999788e-01, -1.05999788e-01, -1.05999788e-01,
 -1.05999788e-01, -1.50755672e-01, -1.05999788e-01,
 -1.05999788e-01, -1.05999788e-01, -1.05999788e-01,
 -1.05999788e-01, -1.50755672e-01, -1.05999788e-01,
 -1.50755672e-01, -1.50755672e-01, -1.05999788e-01,
 -1.50755672e-01, -1.50755672e-01, -1.05999788e-01,
 -1.05999788e-01, -1.50755672e-01, -1.50755672e-01,
 -1.50755672e-01, -1.05999788e-01, -1.05999788e-01,
 -1.05999788e-01, -1.50755672e-01, -2.15665546e-01,
 -1.85695338e-01, -2.15665546e-01, -2.67261242e-01,
 -1.05999788e-01, -2.42535625e-01, -1.05999788e-01,
 -2.15665546e-01, -1.85695338e-01, -2.15665546e-01,
 -1.85695338e-01, -1.05999788e-01,  1.87082869e+00,
 -1.87082869e+00,  8.35531692e-01, -8.35531692e-01,
  1.93309133e+00, -1.93309133e+00],
[-1.55894196e+00, -6.28670558e-01, -6.53912840e-01,
 -1.57589457e+00, -9.73440458e-01, -1.05999788e-01,
```

```
       -1.05999788e-01,  1.52752523e+00, -1.05999788e-01,
       -5.51677284e-01, -2.90408935e-01, -1.85695338e-01,
       -3.33333333e-01, -1.05999788e-01, -2.42535625e-01,
       -4.29197538e-01,  7.97724035e-01, -5.68796459e-01,
       -4.10890702e-01, -4.10890702e-01, -1.50755672e-01,
       -7.97724035e-01, -1.50755672e-01, -3.92232270e-01,
       -1.05999788e-01, -1.05999788e-01, -1.05999788e-01,
       -1.05999788e-01,  9.43398113e+00, -1.05999788e-01,
       -1.05999788e-01, -1.05999788e-01, -1.05999788e-01,
       -1.05999788e-01, -1.05999788e-01, -1.05999788e-01,
       -1.05999788e-01, -1.05999788e-01, -1.05999788e-01,
       -1.05999788e-01, -1.05999788e-01, -1.05999788e-01,
       -1.05999788e-01, -1.50755672e-01, -1.05999788e-01,
       -1.05999788e-01, -1.05999788e-01, -1.05999788e-01,
       -1.05999788e-01, -1.50755672e-01, -1.05999788e-01,
       -1.50755672e-01, -1.50755672e-01, -1.05999788e-01,
       -1.50755672e-01, -1.50755672e-01, -1.05999788e-01,
       -1.05999788e-01, -1.50755672e-01, -1.50755672e-01,
       -1.50755672e-01, -1.05999788e-01, -1.05999788e-01,
       -1.05999788e-01, -1.50755672e-01, -2.15665546e-01,
       -1.85695338e-01, -2.15665546e-01, -2.67261242e-01,
       -1.05999788e-01, -2.42535625e-01, -1.05999788e-01,
       -2.15665546e-01, -1.85695338e-01, -2.15665546e-01,
       -1.85695338e-01, -1.05999788e-01,  1.87082869e+00,
       -1.87082869e+00,  8.35531692e-01, -8.35531692e-01,
        1.93309133e+00, -1.93309133e+00]])
```

We split the data into training and testing data using the function `train_test_split`. The training data is divided into validation data, a second set used for training data; then the models are trained and hyperparameters are selected using the function `GridSearchCV`.

TASK 3

Use the function train_test_split to split the data X and Y into training and test data. Set the parameter test_size to 0.2 and random_state to 2. The training data and test data should be assigned to the following labels.

`X_train, X_test, Y_train, Y_test`

In [17]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, rando
m_state=2)
print ('Train set:', X_train.shape, Y_train.shape)
print ('Test set:', X_test.shape, Y_test.shape)
```

```
Train set: (72, 83) (72,)
Test set: (18, 83) (18,)
```

we can see we only have 18 test samples.

In [18]:
```
Y_test.shape
```
Out[18]:
```
(18,)
```

TASK 4

Create a logistic regression object then create a GridSearchCV object `logreg_cv` with cv = 10. Fit the object to find the best parameters from the dictionary `parameters` .

In [19]:
```
parameters ={'C':[0.01,0.1,1],
             'penalty':['l2'],
             'solver':['lbfgs']}
```

In [20]:
```
parameters ={"C":[0.01,0.1,1],'penalty':['l2'], 'solver':['lbfgs']}# l1 lasso
 l2 ridge
lr=LogisticRegression()

logreg_cv =GridSearchCV(lr, parameters, cv = 10,)

# fitting the model for grid search
logreg_cv.fit(X_train, Y_train)
```
Out[20]:
```
GridSearchCV(cv=10, estimator=LogisticRegression(),
             param_grid={'C': [0.01, 0.1, 1], 'penalty': ['l2'],
                         'solver': ['lbfgs']})
```

We output the `GridSearchCV` object for logistic regression. We display the best parameters using the data attribute `best_params\_` and the accuracy on the validation data using the data attribute `best_score\_` .

In [21]:
```
print("tuned hpyerparameters :(best parameters) ",logreg_cv.best_params_)
print("accuracy :",logreg_cv.best_score_)
```
```
tuned hpyerparameters :(best parameters)  {'C': 0.01, 'penalty': 'l2', 'solve
r': 'lbfgs'}
accuracy : 0.8464285714285713
```

TASK 5

Calculate the accuracy on the test data using the method `score` :

In [29]:
```
from sklearn.metrics import jaccard_score
from sklearn.metrics import f1_score
from sklearn.metrics import log_loss

yhatLR = logreg_cv.predict(X_test)
yhat_probLR = logreg_cv.predict_proba(X_test)
print ('Logistic Regression prediction prob',yhat_probLR)
print ('Logistic Regression prediction',yhatLR)
print ('Logistic regression F1 score',f1_score(Y_test, yhatLR, average='weight
ed'))
print ('Logistic Regression Jaccard score', jaccard_score(Y_test, yhatLR,pos_l
abel=1))
print ('Logistic regression Log loss', log_loss(Y_test, yhat_probLR))

# Use score method to get accuracy of model
score = logreg_cv.score(X_test, Y_test)
print('Score Method:',score)
```

```
Logistic Regression prediction prob [[0.31575125 0.68424875]
 [0.16763308 0.83236692]
 [0.23666893 0.76333107]
 [0.17174248 0.82825752]
 [0.25380985 0.74619015]
 [0.17798224 0.82201776]
 [0.1951042  0.8048958 ]
 [0.63389092 0.36610908]
 [0.18214528 0.81785472]
 [0.58445662 0.41554338]
 [0.59904046 0.40095954]
 [0.34220165 0.65779835]
 [0.31851887 0.68148113]
 [0.16522618 0.83477382]
 [0.24745211 0.75254789]
 [0.25429632 0.74570368]
 [0.34246043 0.65753957]
 [0.326117   0.673883  ]]
Logistic Regression prediction [1 1 1 1 1 1 1 0 1 0 0 1 1 1 1 1 1 1]
Logistic regression F1 score 0.8148148148148149
Logistic Regression Jaccard score 0.8
Logistic regression Log loss 0.4786666968559154
Score Method: 0.8333333333333334
```

Lets look at the confusion matrix:

In [30]:
```
yhat=logreg_cv.predict(X_test)
plot_confusion_matrix(Y_test,yhat)
```

Examining the confusion matrix, we see that logistic regression can distinguish between the different classes. We see that the major problem is false positives.

TASK 6

Create a support vector machine object then create a `GridSearchCV` object `svm_cv` with cv - 10. Fit the object to find the best parameters from the dictionary `parameters` .

In [31]:
```
parameters = {'kernel':('linear', 'rbf','poly','rbf', 'sigmoid'),
              'C': np.logspace(-3, 3, 5),
              'gamma':np.logspace(-3, 3, 5)}
svm = SVC()
```

In [33]:
```
svm_cv =GridSearchCV(svm, parameters, cv = 10,)

# fitting the model for grid search
svm_cv.fit(X_train, Y_train)
```

Out[33]:
```
GridSearchCV(cv=10, estimator=SVC(),
             param_grid={'C': array([1.00000000e-03, 3.16227766e-02, 1.000000
00e+00, 3.16227766e+01,
       1.00000000e+03]),
                         'gamma': array([1.00000000e-03, 3.16227766e-02, 1.00
000000e+00, 3.16227766e+01,
       1.00000000e+03]),
                         'kernel': ('linear', 'rbf', 'poly', 'rbf', 'sigmoi
d')})
```

In [34]:
```
print("tuned hpyerparameters :(best parameters) ",svm_cv.best_params_)
print("accuracy :",svm_cv.best_score_)
```

```
tuned hpyerparameters :(best parameters)  {'C': 1.0, 'gamma': 0.0316227766016
8379, 'kernel': 'sigmoid'}
accuracy : 0.8482142857142856
```

TASK 7

Calculate the accuracy on the test data using the method `score` :

In [35]:
```
yhatSVM = svm_cv.predict(X_test)
print ('SVM predicion', yhatSVM [0:5])
print ('SVM F1 score',f1_score(Y_test, yhatSVM, average='weighted'))
print ('SVM Jaccard score', jaccard_score(Y_test, yhatSVM,pos_label=1))

# Use score method to get accuracy of model
score = svm_cv.score(X_test, Y_test)
print('Score Method:',score)
```

```
SVM predicion [1 1 1 1 1]
SVM F1 score 0.8148148148148149
SVM Jaccard score 0.8
Score Method: 0.8333333333333334
```

We can plot the confusion matrix

In [36]:
```
yhat=svm_cv.predict(X_test)
plot_confusion_matrix(Y_test,yhat)
```

TASK 8

Create a decision tree classifier object then create a `GridSearchCV` object `tree_cv` with cv = 10. Fit the object to find the best parameters from the dictionary `parameters` .

In [38]:
```
parameters = {'criterion': ['gini', 'entropy'],
     'splitter': ['best', 'random'],
     'max_depth': [2*n for n in range(1,10)],
     'max_features': ['auto', 'sqrt'],
     'min_samples_leaf': [1, 2, 4],
     'min_samples_split': [2, 5, 10]}

tree = DecisionTreeClassifier()
```

In [39]:
```
tree_cv =GridSearchCV(tree, parameters, cv = 10,)

# fitting the model for grid search
tree_cv.fit(X_train, Y_train)
```

Out[39]:
```
GridSearchCV(cv=10, estimator=DecisionTreeClassifier(),
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_depth': [2, 4, 6, 8, 10, 12, 14, 16, 18],
                         'max_features': ['auto', 'sqrt'],
                         'min_samples_leaf': [1, 2, 4],
                         'min_samples_split': [2, 5, 10],
                         'splitter': ['best', 'random']})
```

In [40]:
```
print("tuned hpyerparameters :(best parameters) ",tree_cv.best_params_)
print("accuracy :",tree_cv.best_score_)
```

```
tuned hpyerparameters :(best parameters)  {'criterion': 'gini', 'max_depth':
14, 'max_features': 'sqrt', 'min_samples_leaf': 1, 'min_samples_split': 5, 's
plitter': 'best'}
accuracy : 0.8875
```

TASK 9

Calculate the accuracy of tree_cv on the test data using the method `score` :

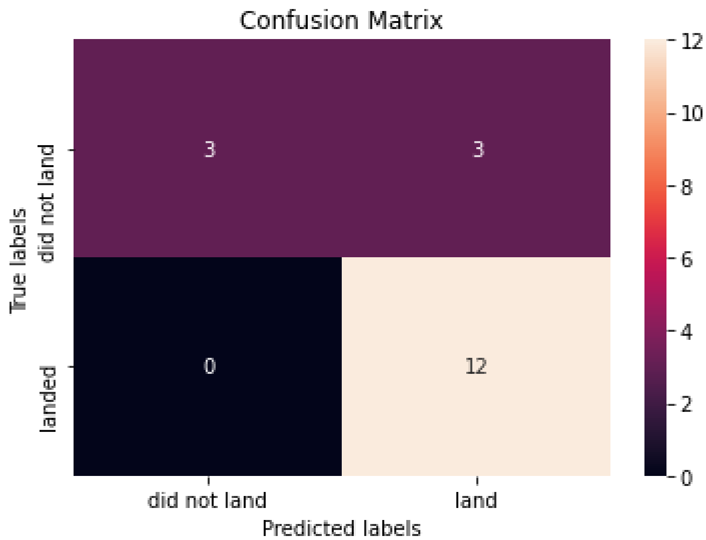
In [41]:
```
yhatTree = tree_cv.predict(X_test)
print ('Tree predicion', yhatTree [0:5])
print ('Tree F1 score',f1_score(Y_test, yhatTree, average='weighted'))
print ('Tree Jaccard score', jaccard_score(Y_test, yhatTree,pos_label=1))

# Use score method to get accuracy of model
score = tree_cv.score(X_test, Y_test)
print('Score Method:',score)
```

```
Tree predicion [1 1 1 0 1]
Tree F1 score 0.8361204013377926
Tree Jaccard score 0.7692307692307693
Score Method: 0.8333333333333334
```

We can plot the confusion matrix

In [42]:
```
yhat = svm_cv.predict(X_test)
plot_confusion_matrix(Y_test,yhat)
```



TASK 10

Create a k nearest neighbors object then create a `GridSearchCV` object `knn_cv` with cv = 10. Fit the object to find the best parameters from the dictionary `parameters` .

In [44]:
```
parameters = {'n_neighbors': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
              'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute'],
              'p': [1,2]}

KNN = KNeighborsClassifier()
```

In [45]:
```
knn_cv =GridSearchCV(KNN, parameters, cv = 10,)

# fitting the model for grid search
knn_cv.fit(X_train, Y_train)
```

Out[45]:
```
GridSearchCV(cv=10, estimator=KNeighborsClassifier(),
             param_grid={'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brut
e'],
                         'n_neighbors': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
                         'p': [1, 2]})
```

In [46]:
```
print("tuned hpyerparameters :(best parameters) ",knn_cv.best_params_)
print("accuracy :",knn_cv.best_score_)
```

```
tuned hpyerparameters :(best parameters)  {'algorithm': 'auto', 'n_neighbor
s': 10, 'p': 1}
accuracy : 0.8482142857142858
```

TASK 11

Calculate the accuracy of tree_cv on the test data using the method `score` :

In [47]:
```
yhatKNN = knn_cv.predict(X_test)
print ('KNN predicion', yhatKNN [0:5])
print ('KNN F1 score',f1_score(Y_test, yhatKNN, average='weighted'))
print ('KNN Jaccard score', jaccard_score(Y_test, yhatKNN,pos_label=1))

# Use score method to get accuracy of model
score = knn_cv.score(X_test, Y_test)
print('Score Method:',score)
```

```
KNN predicion [1 1 1 1 1]
KNN F1 score 0.8148148148148149
KNN Jaccard score 0.8
Score Method: 0.8333333333333334
```

We can plot the confusion matrix

In [48]:
```
yhat = knn_cv.predict(X_test)
plot_confusion_matrix(Y_test,yhat)
```

TASK 12

Find the method performs best:

In [54]:
```
print("Logistic regression accuracy :",logreg_cv.best_score_)
print("SVM accuracy :",svm_cv.best_score_)
print("Decision Tree accuracy :",tree_cv.best_score_)
print("KNN accuracy :",knn_cv.best_score_)

print("Highest is Decision Tree accuracy :",tree_cv.best_score_)
```

```
Logistic regression accuracy : 0.8464285714285713
SVM accuracy : 0.8482142857142856
Decision Tree accuracy : 0.8875
KNN accuracy : 0.8482142857142858
Highest is Decision Tree accuracy : 0.8875
```

Authors

Joseph Santarcangelo (https://www.linkedin.com/in/joseph-s-50398b136/?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=10006555&utm_id=NA-SkillsNetwork-Channel-SkillsNetworkCoursesIBMDS0321ENSkillsNetwork26802033-2021-01-01) has a PhD in Electrical Engineering, his research focused on using machine learning, signal processing, and computer vision to determine how videos impact human cognition. Joseph has been working for IBM since he completed his PhD.

Change Log

Date (YYYY-MM-DD)	Version	Changed By	Change Description
2021-08-31	1.1	Lakshmi Holla	Modified markdown
2020-09-20	1.0	Joseph	Modified Multiple Areas