cognitiveclass.ai logo

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 [53]:
```
# Pandas is a software library written for the Pyth
import pandas as pd
# NumPy is a library for the Python programming lan
import numpy as np
# Matplotlib is a plotting library for python and p
import matplotlib.pyplot as plt
#Seaborn is a Python data visualization library bas
import seaborn as sns
# Preprocessing allows us to standarsize our data
from sklearn import preprocessing
# Allows us to split our data into training and tes
from sklearn.model_selection import train_test_spli
# Allows us to test parameters of classification al
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
import warnings
warnings.filterwarnings('ignore')
```

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=Tr
    ax.set_xlabel('Predicted labels')
    ax.set_ylabel('True labels')
    ax.set_title('Confusion Matrix');
    ax.xaxis.set_ticklabels(['did not land', 'land
```

Load the dataframe

Load the data

In [3]:

```
data = pd.read_csv("https://cf-courses-data.s3.us.c

# If you were unable to complete the previous lab c

# data = pd.read_csv('https://cf-courses-data.s3.us

data.head()
```

Out[3]:

	FlightNumber	Date	BoosterVersion	PayloadMass	Orbi
0	1	2010-06-04	Falcon 9	6104.959412	LE(
1	2	2012-05-22	Falcon 9	525.000000	LE(
2	3	2013-03-01	Falcon 9	677.000000	IS
3	4	2013-09-29	Falcon 9	500.000000	P(
4	5	2013-12-03	Falcon 9	3170.000000	GT(

In [4]:

```
X = pd.read_csv('https://cf-courses-data.s3.us.clou

# If you were unable to complete the previous lab c
```

```
# X = pd.read_csv('https://cf-courses-data.s3.us.cl

X.head(100)
```

Out[4]:

	FlightNumber	PayloadMass	Flights	Block	ReusedCou
0	1.0	6104.959412	1.0	1.0	0
1	2.0	525.000000	1.0	1.0	0
2	3.0	677.000000	1.0	1.0	0
3	4.0	500.000000	1.0	1.0	0
4	5.0	3170.000000	1.0	1.0	0
...	...	...	...	...	
85	86.0	15400.000000	2.0	5.0	2
86	87.0	15400.000000	3.0	5.0	2
87	88.0	15400.000000	6.0	5.0	5
88	89.0	15400.000000	3.0	5.0	2
89	90.0	3681.000000	1.0	5.0	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 [6]:
```
Y = data['Class'].to_numpy()
Y
```

Out[6]:
```
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 [7]:
```
# students get this
transform = preprocessing.StandardScaler()
```

In [10]:
```
X = transform.fit(X).transform(X)
```

```
X[0:5]
```

Out[10]:
```
array([[-1.71291154e+00, -5.29526321e-17, -6.53912
840e-01,
         -1.57589457e+00, -9.73440458e-01, -1.05999
788e-01,
         -1.05999788e-01, -6.54653671e-01, -1.05999
788e-01,
         -5.51677284e-01,  3.44342023e+00, -1.85695
338e-01,
         -3.33333333e-01, -1.05999788e-01, -2.42535
625e-01,
         -4.29197538e-01,  7.97724035e-01, -5.68796
459e-01,
         -4.10890702e-01, -4.10890702e-01, -1.50755
672e-01,
         -7.97724035e-01, -1.50755672e-01, -3.92232
270e-01,
          9.43398113e+00, -1.05999788e-01, -1.05999
788e-01,
         -1.05999788e-01, -1.05999788e-01, -1.05999
788e-01,
         -1.05999788e-01, -1.05999788e-01, -1.05999
788e-01,
```

```
           -1.05999788e-01, -1.05999788e-01, -1.05999
788e-01,
           -1.05999788e-01, -1.05999788e-01, -1.05999
788e-01,
           -1.05999788e-01, -1.05999788e-01, -1.05999
788e-01,
           -1.05999788e-01, -1.50755672e-01, -1.05999
788e-01,
           -1.05999788e-01, -1.05999788e-01, -1.05999
788e-01,
           -1.05999788e-01, -1.50755672e-01, -1.05999
788e-01,
           -1.50755672e-01, -1.50755672e-01, -1.05999
788e-01,
           -1.50755672e-01, -1.50755672e-01, -1.05999
788e-01,
           -1.05999788e-01, -1.50755672e-01, -1.50755
672e-01,
           -1.50755672e-01, -1.05999788e-01, -1.05999
788e-01,
           -1.05999788e-01, -1.50755672e-01, -2.15665
546e-01,
```

```
         -1.85695338e-01, -2.15665546e-01, -2.67261
242e-01,
         -1.05999788e-01, -2.42535625e-01, -1.05999
788e-01,
         -2.15665546e-01, -1.85695338e-01, -2.15665
546e-01,
         -1.85695338e-01, -1.05999788e-01,  1.87082
869e+00,
         -1.87082869e+00,  8.35531692e-01, -8.35531
692e-01,
          1.93309133e+00, -1.93309133e+00],
        [-1.67441914e+00, -1.19523159e+00, -6.53912
840e-01,
         -1.57589457e+00, -9.73440458e-01, -1.05999
788e-01,
         -1.05999788e-01, -6.54653671e-01, -1.05999
788e-01,
         -5.51677284e-01,  3.44342023e+00, -1.85695
338e-01,
         -3.33333333e-01, -1.05999788e-01, -2.42535
625e-01,
         -4.29197538e-01,  7.97724035e-01, -5.68796
```

```
459e-01,
        -4.10890702e-01, -4.10890702e-01, -1.50755
672e-01,
        -7.97724035e-01, -1.50755672e-01, -3.92232
270e-01,
        -1.05999788e-01,  9.43398113e+00, -1.05999
788e-01,
        -1.05999788e-01, -1.05999788e-01, -1.05999
788e-01,
        -1.05999788e-01, -1.05999788e-01, -1.05999
788e-01,
        -1.05999788e-01, -1.05999788e-01, -1.05999
788e-01,
        -1.05999788e-01, -1.05999788e-01, -1.05999
788e-01,
        -1.05999788e-01, -1.05999788e-01, -1.05999
788e-01,
        -1.05999788e-01, -1.50755672e-01, -1.05999
788e-01,
        -1.05999788e-01, -1.05999788e-01, -1.05999
788e-01,
        -1.05999788e-01, -1.50755672e-01, -1.05999
```

```
788e-01,
        -1.50755672e-01, -1.50755672e-01, -1.05999
788e-01,
        -1.50755672e-01, -1.50755672e-01, -1.05999
788e-01,
        -1.05999788e-01, -1.50755672e-01, -1.50755
672e-01,
        -1.50755672e-01, -1.05999788e-01, -1.05999
788e-01,
        -1.05999788e-01, -1.50755672e-01, -2.15665
546e-01,
        -1.85695338e-01, -2.15665546e-01, -2.67261
242e-01,
        -1.05999788e-01, -2.42535625e-01, -1.05999
788e-01,
        -2.15665546e-01, -1.85695338e-01, -2.15665
546e-01,
        -1.85695338e-01, -1.05999788e-01,  1.87082
869e+00,
        -1.87082869e+00,  8.35531692e-01, -8.35531
692e-01,
         1.93309133e+00, -1.93309133e+00],
```

```
        [-1.63592675e+00, -1.16267307e+00, -6.53912
840e-01,
         -1.57589457e+00, -9.73440458e-01, -1.05999
788e-01,
         -1.05999788e-01, -6.54653671e-01, -1.05999
788e-01,
          1.81265393e+00, -2.90408935e-01, -1.85695
338e-01,
         -3.33333333e-01, -1.05999788e-01, -2.42535
625e-01,
         -4.29197538e-01,  7.97724035e-01, -5.68796
459e-01,
         -4.10890702e-01, -4.10890702e-01, -1.50755
672e-01,
         -7.97724035e-01, -1.50755672e-01, -3.92232
270e-01,
         -1.05999788e-01, -1.05999788e-01,  9.43398
113e+00,
         -1.05999788e-01, -1.05999788e-01, -1.05999
788e-01,
         -1.05999788e-01, -1.05999788e-01, -1.05999
788e-01,
```

```
          -1.05999788e-01, -1.05999788e-01, -1.05999
788e-01,
          -1.05999788e-01, -1.05999788e-01, -1.05999
788e-01,
          -1.05999788e-01, -1.05999788e-01, -1.05999
788e-01,
          -1.05999788e-01, -1.50755672e-01, -1.05999
788e-01,
          -1.05999788e-01, -1.05999788e-01, -1.05999
788e-01,
          -1.05999788e-01, -1.50755672e-01, -1.05999
788e-01,
          -1.50755672e-01, -1.50755672e-01, -1.05999
788e-01,
          -1.50755672e-01, -1.50755672e-01, -1.05999
788e-01,
          -1.05999788e-01, -1.50755672e-01, -1.50755
672e-01,
          -1.50755672e-01, -1.05999788e-01, -1.05999
788e-01,
          -1.05999788e-01, -1.50755672e-01, -2.15665
546e-01,
```

```
        -1.85695338e-01, -2.15665546e-01, -2.67261
242e-01,
        -1.05999788e-01, -2.42535625e-01, -1.05999
788e-01,
        -2.15665546e-01, -1.85695338e-01, -2.15665
546e-01,
        -1.85695338e-01, -1.05999788e-01,  1.87082
869e+00,
        -1.87082869e+00,  8.35531692e-01, -8.35531
692e-01,
         1.93309133e+00, -1.93309133e+00],
       [-1.59743435e+00, -1.20058661e+00, -6.53912
840e-01,
        -1.57589457e+00, -9.73440458e-01, -1.05999
788e-01,
        -1.05999788e-01, -6.54653671e-01, -1.05999
788e-01,
        -5.51677284e-01, -2.90408935e-01, -1.85695
338e-01,
         3.00000000e+00, -1.05999788e-01, -2.42535
625e-01,
        -4.29197538e-01, -1.25356634e+00, -5.68796
```

```
459e-01,
         2.43373723e+00, -4.10890702e-01, -1.50755
672e-01,
        -7.97724035e-01, -1.50755672e-01, -3.92232
270e-01,
        -1.05999788e-01, -1.05999788e-01, -1.05999
788e-01,
         9.43398113e+00, -1.05999788e-01, -1.05999
788e-01,
        -1.05999788e-01, -1.05999788e-01, -1.05999
788e-01,
        -1.05999788e-01, -1.05999788e-01, -1.05999
788e-01,
        -1.05999788e-01, -1.05999788e-01, -1.05999
788e-01,
        -1.05999788e-01, -1.05999788e-01, -1.05999
788e-01,
        -1.05999788e-01, -1.50755672e-01, -1.05999
788e-01,
        -1.05999788e-01, -1.05999788e-01, -1.05999
788e-01,
        -1.05999788e-01, -1.50755672e-01, -1.05999
```

```
788e-01,
          -1.50755672e-01, -1.50755672e-01, -1.05999
788e-01,
          -1.50755672e-01, -1.50755672e-01, -1.05999
788e-01,
          -1.05999788e-01, -1.50755672e-01, -1.50755
672e-01,
          -1.50755672e-01, -1.05999788e-01, -1.05999
788e-01,
          -1.05999788e-01, -1.50755672e-01, -2.15665
546e-01,
          -1.85695338e-01, -2.15665546e-01, -2.67261
242e-01,
          -1.05999788e-01, -2.42535625e-01, -1.05999
788e-01,
          -2.15665546e-01, -1.85695338e-01, -2.15665
546e-01,
          -1.85695338e-01, -1.05999788e-01,  1.87082
869e+00,
          -1.87082869e+00,  8.35531692e-01, -8.35531
692e-01,
           1.93309133e+00, -1.93309133e+00],
```

```
          [-1.55894196e+00, -6.28670558e-01, -6.53912
840e-01,
           -1.57589457e+00, -9.73440458e-01, -1.05999
788e-01,
           -1.05999788e-01,  1.52752523e+00, -1.05999
788e-01,
           -5.51677284e-01, -2.90408935e-01, -1.85695
338e-01,
           -3.33333333e-01, -1.05999788e-01, -2.42535
625e-01,
           -4.29197538e-01,  7.97724035e-01, -5.68796
459e-01,
           -4.10890702e-01, -4.10890702e-01, -1.50755
672e-01,
           -7.97724035e-01, -1.50755672e-01, -3.92232
270e-01,
           -1.05999788e-01, -1.05999788e-01, -1.05999
788e-01,
           -1.05999788e-01,  9.43398113e+00, -1.05999
788e-01,
           -1.05999788e-01, -1.05999788e-01, -1.05999
788e-01,
```

```
          -1.05999788e-01, -1.05999788e-01, -1.05999
788e-01,
          -1.05999788e-01, -1.05999788e-01, -1.05999
788e-01,
          -1.05999788e-01, -1.05999788e-01, -1.05999
788e-01,
          -1.05999788e-01, -1.50755672e-01, -1.05999
788e-01,
          -1.05999788e-01, -1.05999788e-01, -1.05999
788e-01,
          -1.05999788e-01, -1.50755672e-01, -1.05999
788e-01,
          -1.50755672e-01, -1.50755672e-01, -1.05999
788e-01,
          -1.50755672e-01, -1.50755672e-01, -1.05999
788e-01,
          -1.05999788e-01, -1.50755672e-01, -1.50755
672e-01,
          -1.50755672e-01, -1.05999788e-01, -1.05999
788e-01,
          -1.05999788e-01, -1.50755672e-01, -2.15665
546e-01,
```

```
        -1.85695338e-01, -2.15665546e-01, -2.67261
242e-01,
        -1.05999788e-01, -2.42535625e-01, -1.05999
788e-01,
        -2.15665546e-01, -1.85695338e-01, -2.15665
546e-01,
        -1.85695338e-01, -1.05999788e-01,  1.87082
869e+00,
        -1.87082869e+00,  8.35531692e-01, -8.35531
692e-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 [11]:
```
X_train, X_test, Y_train, Y_test = train_test_split
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 [12]:
```
Y_test.shape
```

Out[12]:
```
(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 [54]:
```
parameters ={'C':[0.01,0.1,1],
             'penalty':['l2'],
             'solver':['lbfgs']}
lr=LogisticRegression()
grid_search = GridSearchCV(lr, parameters, cv=10)
logreg_cv = grid_search.fit(X_train, Y_train)
```

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 [19]:
```
print("tuned hpyerparameters :(best parameters) ",l
print("accuracy :",logreg_cv.best_score_)
```

```
tuned hpyerparameters :(best parameters)  {'C': 0.
01, 'penalty': 'l2', 'solver': 'lbfgs'}
accuracy : 0.8472222222222222
```

TASK 5

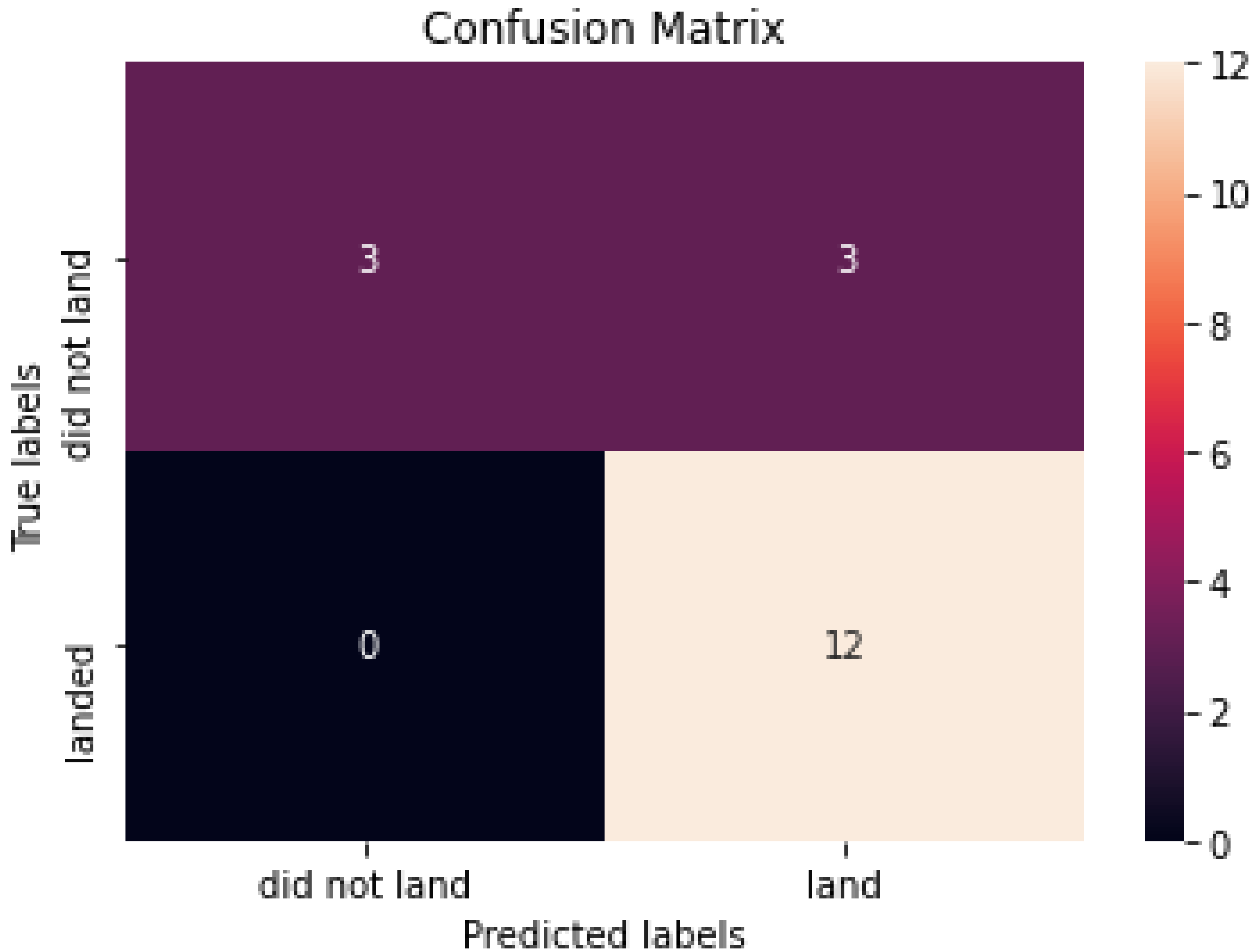
Calculate the accuracy on the test data using the method `score` :

In [55]:
```
logreg_cv.score(X_test,Y_test)
```

Out[55]:
```
0.8333333333333334
```

Lets look at the confusion matrix:

In [56]:
```
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 [57]:
```
parameters = {'kernel':('linear', 'rbf','poly','rbf
              'C': np.logspace(-3, 3, 5),
              'gamma':np.logspace(-3, 3, 5)}
svm = SVC()
```

In [58]:
```
GridSearch = GridSearchCV(svm, parameters, cv=10)
svm_cv = GridSearch.fit(X_train,Y_train)
```

In [30]:
```
print("tuned hpyerparameters :(best parameters) ",s
print("accuracy :",svm_cv.best_score_)
```

```
tuned hpyerparameters :(best parameters)  {'C': 1.
0, 'gamma': 0.03162277660168379, 'kernel': 'sigmoi
d'}
accuracy : 0.8472222222222222
```

TASK 7

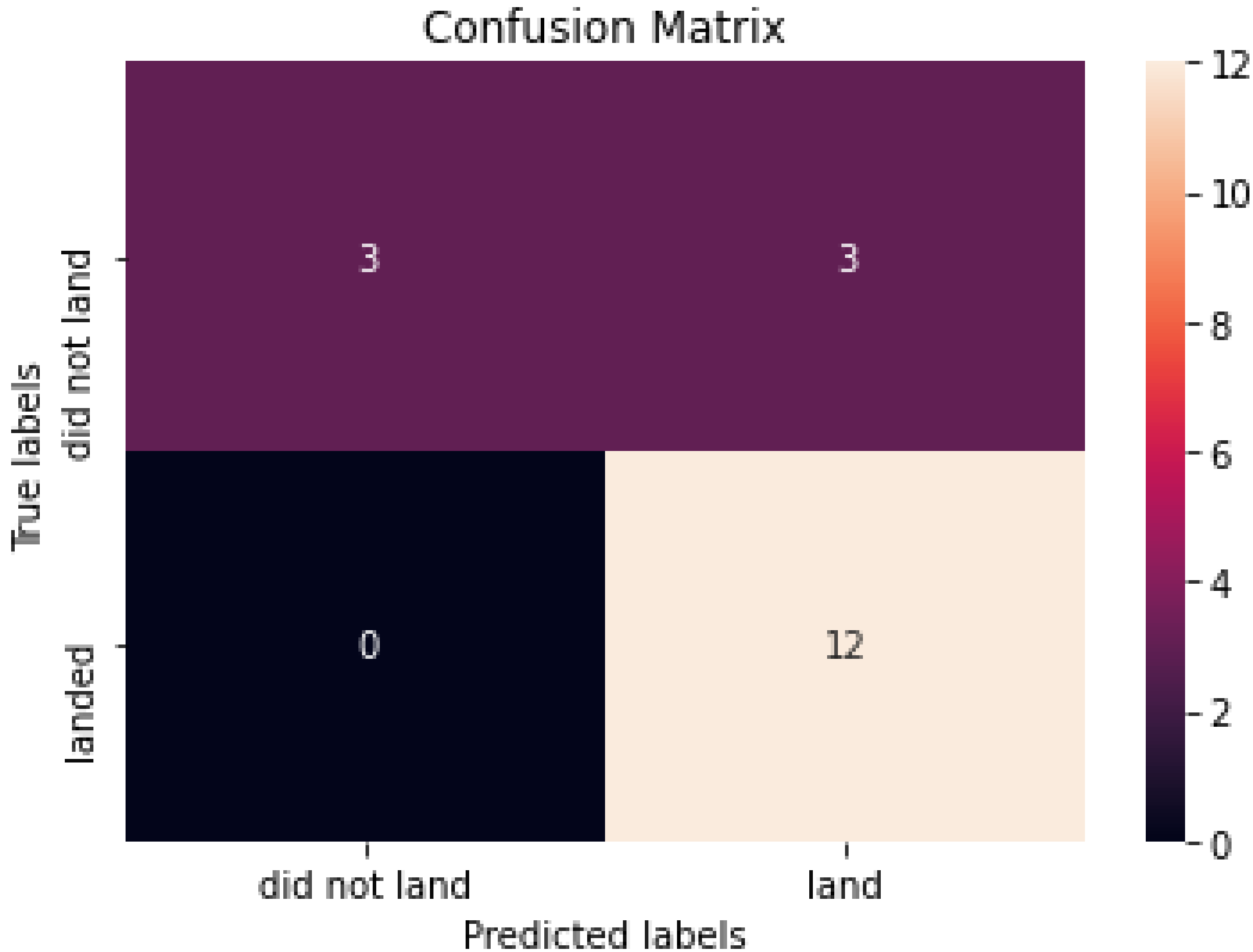
Calculate the accuracy on the test data using the method `score` :

In [32]:
```
svm_cv.score(X_test, Y_test)
```

Out[32]:
```
0.8333333333333334
```

We can plot the confusion matrix

In [33]:
```
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 [59]:
```
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 [60]:
```
grid_search = GridSearchCV(tree, parameters, cv=10)
tree_cv = grid_search.fit(X_train, Y_train)
```

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

```
tuned hpyerparameters :(best parameters)  {'criter
ion': 'gini', 'max_depth': 12, 'max_features': 'sq
rt', 'min_samples_leaf': 1, 'min_samples_split':
5, 'splitter': 'random'}
accuracy : 0.875
```

TASK 9

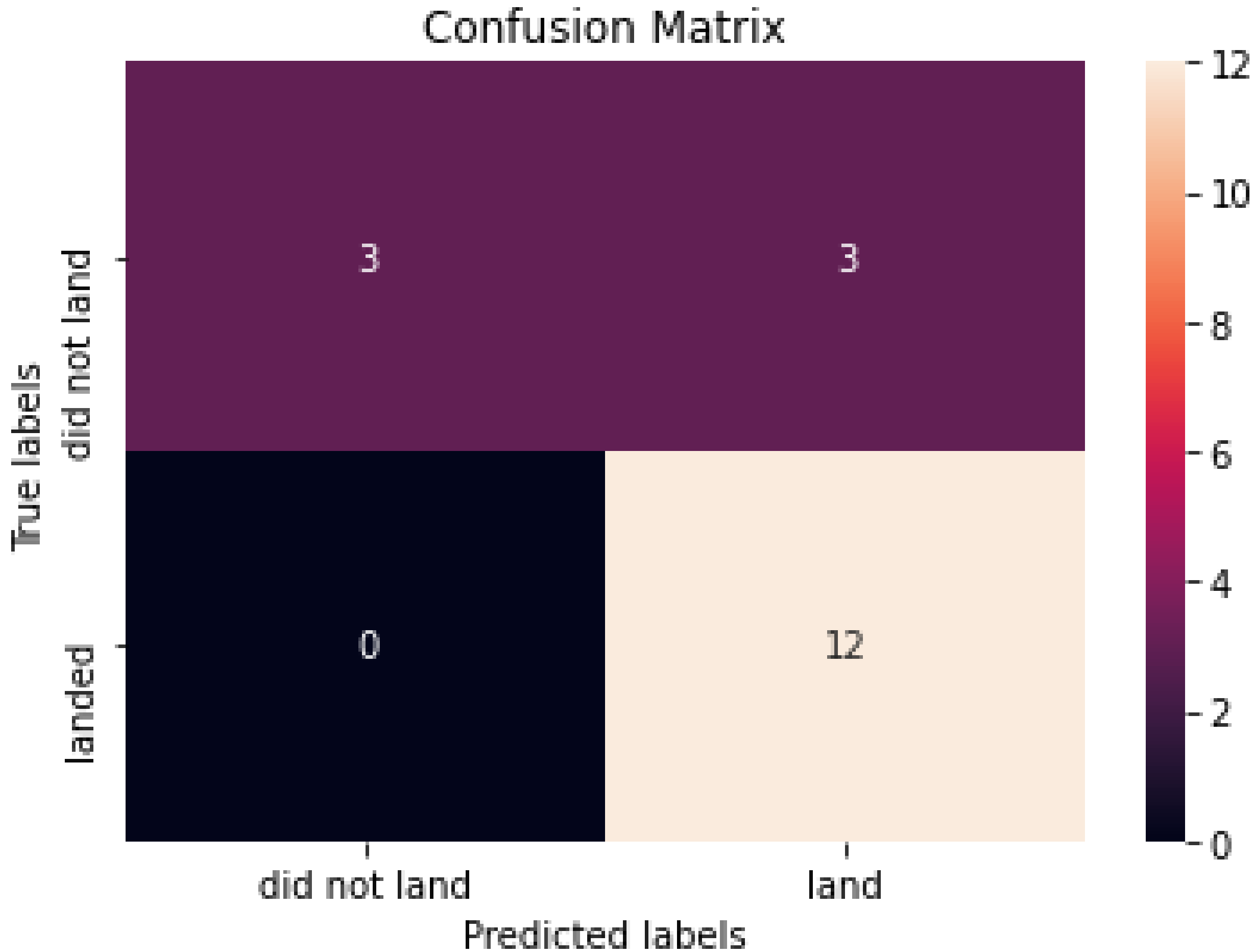
Calculate the accuracy of tree_cv on the test data using the method `score` :

In [61]:
```
tree_cv.score(X_test, Y_test)
```

Out[61]:
```
0.6666666666666666
```

We can plot the confusion matrix

In [62]:
```
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 [64]:
```
parameters = {'n_neighbors': [1, 2, 3, 4, 5, 6, 7,
              'algorithm': ['auto', 'ball_tree', 'k
              'p': [1,2]}

KNN = KNeighborsClassifier()
```

In [66]:
```
GridSearch = GridSearchCV(KNN, parameters, cv=10)
knn_cv = GridSearch.fit(X_train, Y_train)
```

In [67]:
```
print("tuned hpyerparameters :(best parameters) ",k
print("accuracy :",knn_cv.best_score_)
```

```
tuned hpyerparameters :(best parameters)  {'algori
thm': 'auto', 'n_neighbors': 9, 'p': 1}
accuracy : 0.8472222222222222
```

TASK 11

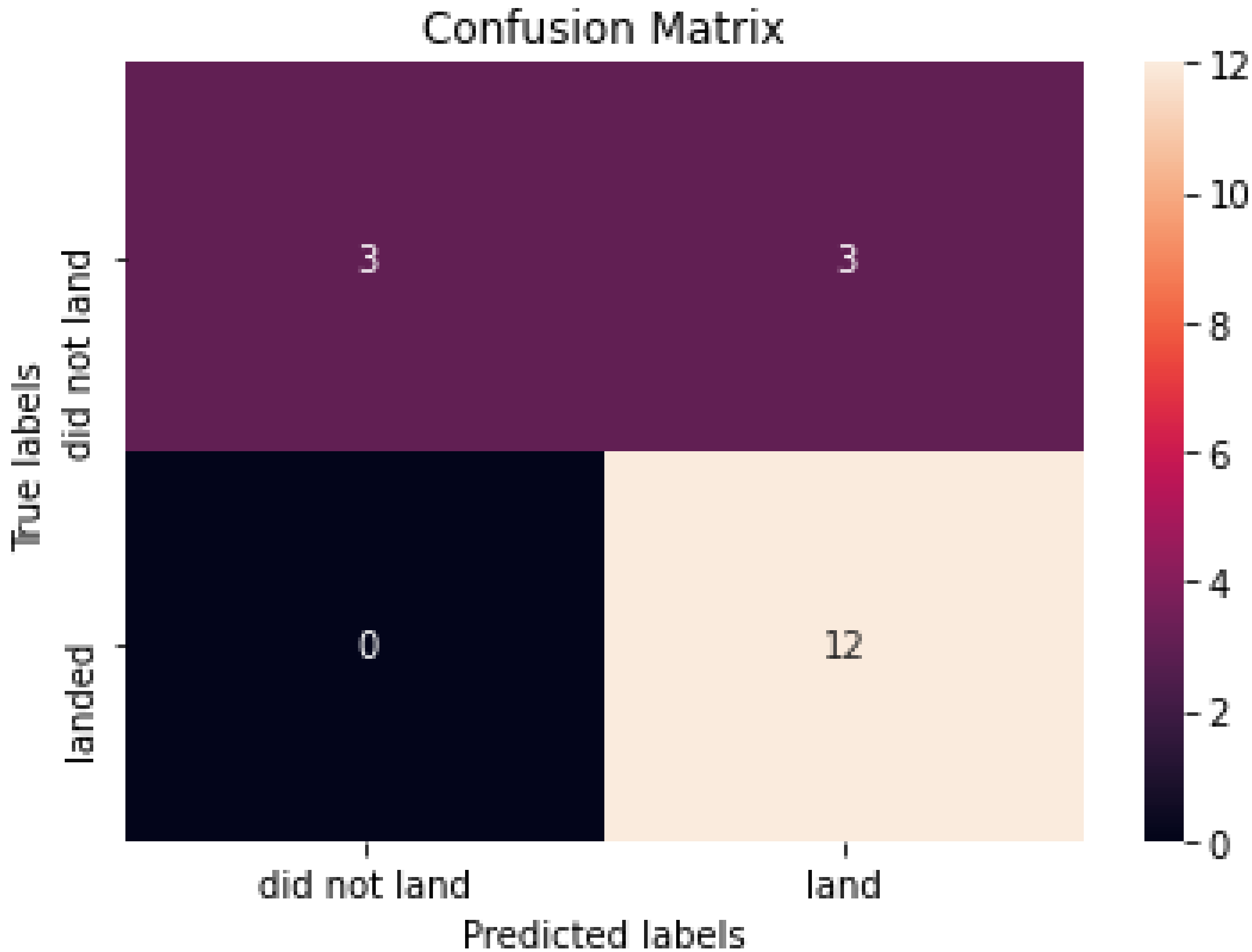
Calculate the accuracy of tree_cv on the test data using the method `score` :

In [69]:
```
knn_cv.score(X_test,Y_test)
```

Out[69]:
```
0.8333333333333334
```

We can plot the confusion matrix

In [70]:
```
yhat = knn_cv.predict(X_test)
plot_confusion_matrix(Y_test,yhat)
```


Confusion Matrix
True labels
did not land
landed
3
3
0
12
did not land
land
Predicted labels
12
10
8
6
4
2
0

TASK 12

Find the method performs best:

In [77]:
```
print('Accuracy for Logistics Regression method is
print( 'Accuracy for Support Vector Machine method
# print('Accuracy for Decision tree method:', tree_
print('Accuracy for K nearsdt neighbors method is:
```

```
Accuracy for Logistics Regression method is: 0.833
3333333333334
Accuracy for Support Vector Machine method is: 0.8
333333333333334
Accuracy for K nearsdt neighbors method is: 0.8333
333333333334
```

Authors

Joseph Santarcangelo 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