SpaceX_Machine Learning Prediction_Part_5

June 22, 2022

1 Space X Falcon 9 First Stage Landing Prediction

1.1 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.

1.2 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

1.3 Import Libraries and Define Auxiliary Functions

We will import the following libraries for the lab

[61]:
```
import warnings
warnings.filterwarnings('ignore')
```

[62]:
```
# 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␣
↪large, 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 like␣
↪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␣
↪provides a high-level interface for drawing attractive and informative␣
↪statistical graphics
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
from sklearn import metrics
```

This function is to plot the confusion matrix.

[63]:
```
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'])
```

1.4 Load the dataframe

Load the data

[64]:
```
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()
```

[64]:
```
   FlightNumber        Date BoosterVersion  PayloadMass Orbit    LaunchSite  \
0             1  2010-06-04       Falcon 9  6104.959412   LEO  CCAFS SLC 40
1             2  2012-05-22       Falcon 9   525.000000   LEO  CCAFS SLC 40
2             3  2013-03-01       Falcon 9   677.000000   ISS  CCAFS SLC 40
3             4  2013-09-29       Falcon 9   500.000000    PO   VAFB SLC 4E
4             5  2013-12-03       Falcon 9  3170.000000   GTO  CCAFS SLC 40

       Outcome  Flights  GridFins  Reused   Legs LandingPad  Block  \
0    None None        1     False   False  False        NaN    1.0
1    None None        1     False   False  False        NaN    1.0
2    None None        1     False   False  False        NaN    1.0
3  False Ocean        1     False   False  False        NaN    1.0
4    None None        1     False   False  False        NaN    1.0

   ReusedCount Serial   Longitude   Latitude  Class
0            0  B0003  -80.577366  28.561857      0
1            0  B0005  -80.577366  28.561857      0
2            0  B0007  -80.577366  28.561857      0
3            0  B1003 -120.610829  34.632093      0
4            0  B1004  -80.577366  28.561857      0
```

[65]:
```
X = pd.read_csv('https://cf-courses-data.s3.us.cloud-object-storage.appdomain.
↪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)
```

[65]:

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

	Orbit_GEO	Orbit_GTO	Orbit_HEO	Orbit_ISS	…	Serial_B1058	\
0	0.0	0.0	0.0	0.0	…	0.0	
1	0.0	0.0	0.0	0.0	…	0.0	
2	0.0	0.0	0.0	1.0	…	0.0	
3	0.0	0.0	0.0	0.0	…	0.0	
4	0.0	1.0	0.0	0.0	…	0.0	
..	…	…	…	…	…	…	
85	0.0	0.0	0.0	0.0	…	0.0	
86	0.0	0.0	0.0	0.0	…	1.0	
87	0.0	0.0	0.0	0.0	…	0.0	
88	0.0	0.0	0.0	0.0	…	0.0	
89	0.0	0.0	0.0	0.0	…	0.0	

	Serial_B1059	Serial_B1060	Serial_B1062	GridFins_False	GridFins_True	\
0	0.0	0.0	0.0	1.0	0.0	
1	0.0	0.0	0.0	1.0	0.0	
2	0.0	0.0	0.0	1.0	0.0	
3	0.0	0.0	0.0	1.0	0.0	
4	0.0	0.0	0.0	1.0	0.0	
..	…	…	…	…	…	
85	0.0	1.0	0.0	0.0	1.0	
86	0.0	0.0	0.0	0.0	1.0	
87	0.0	0.0	0.0	0.0	1.0	
88	0.0	1.0	0.0	0.0	1.0	
89	0.0	0.0	1.0	0.0	1.0	

	Reused_False	Reused_True	Legs_False	Legs_True
0	1.0	0.0	1.0	0.0
1	1.0	0.0	1.0	0.0
2	1.0	0.0	1.0	0.0
3	1.0	0.0	1.0	0.0
4	1.0	0.0	1.0	0.0
..	…	…	…	…
85	0.0	1.0	0.0	1.0

```
86            0.0          1.0          0.0          1.0
87            0.0          1.0          0.0          1.0
88            0.0          1.0          0.0          1.0
89            1.0          0.0          0.0          1.0

[90 rows x 83 columns]
```

1.5 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’]).

[66]:
```
Y = data['Class'].to_numpy()
Y
```

[66]:
```
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])
```

1.6 TASK 2

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

[67]:
```
# students get this
transform = preprocessing.StandardScaler()
```

[68]:
```
X = transform.fit(X).transform(X.astype(float))
X[0:5]
```

[68]:
```
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.

1.7 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

[69]:
```
X_train, X_test, Y_train, Y_test = train_test_split( X, Y, test_size=0.2,␣
 ↪random_state=2)
```

we can see we only have 18 test samples.

[70]:
```
Y_test.shape
```

[70]:
```
(18,)
```

1.8 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.

[71]:
```
parameters ={"C":[0.01,0.1,1],'penalty':['l2'], 'solver':['lbfgs']}# l1 lasso␣
 ↪l2 ridge
lr=LogisticRegression()
logreg_cv = GridSearchCV(lr, parameters, cv=10)
logreg_cv.fit(X, Y)
```

[71]:
```
GridSearchCV(cv=10, error_score='raise-deprecating',
       estimator=LogisticRegression(C=1.0, class_weight=None, dual=False,
fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l2', random_state=None, solver='warn',
          tol=0.0001, verbose=0, warm_start=False),
       fit_params=None, iid='warn', n_jobs=None,
       param_grid={'C': [0.01, 0.1, 1], 'penalty': ['l2'], 'solver': ['lbfgs']},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring=None, verbose=0)
```

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_.

[72]:
```
print("tuned hpyerparameters :(best parameters) ",logreg_cv.best_params_)
print("accuracy :",logreg_cv.best_score_)
```

```
tuned hpyerparameters :(best parameters)  {'C': 0.01, 'penalty': 'l2', 'solver':
'lbfgs'}
accuracy : 0.8222222222222222
```

1.9 TASK 5

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

[73]:
```
metrics.accuracy_score(Y_test, logreg_cv.predict(X_test))
```

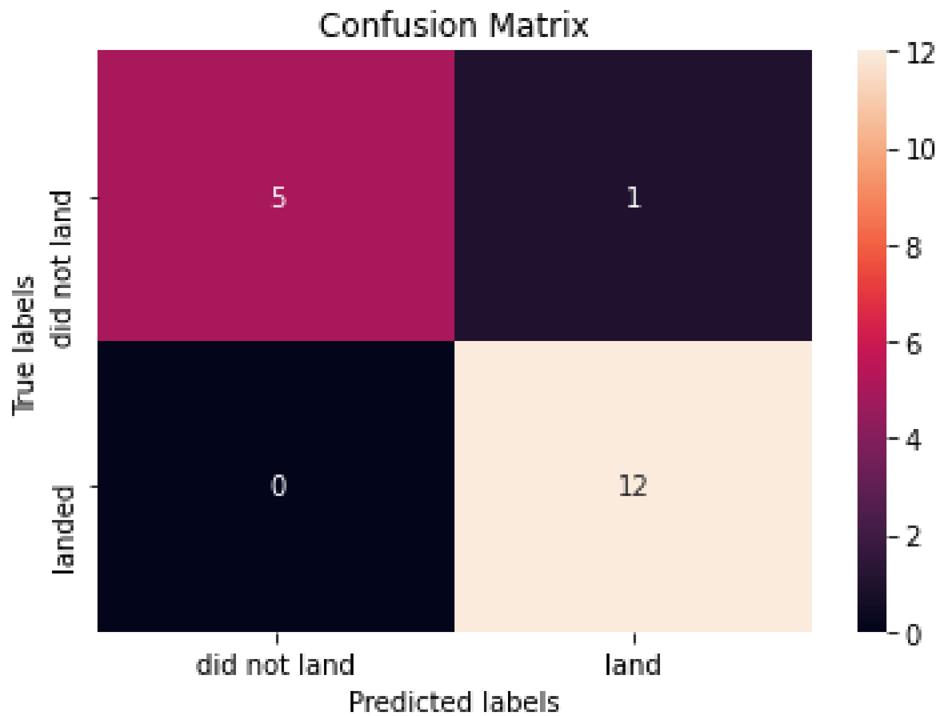
[73]:
```
0.9444444444444444
```

[74]:
```
logreg_cv.score(X_test, Y_test)
```

[74]:
```
0.9444444444444444
```

Lets look at the confusion matrix:

[75]:
```
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.

1.10 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.

[76]:
```
parameters = {'kernel':('linear', 'rbf','poly','rbf', 'sigmoid'),
              'C': np.logspace(-3, 3, 5),
              'gamma':np.logspace(-3, 3, 5)}
svm = SVC()
```

[77]:
```
svm_cv = GridSearchCV(svm, parameters, cv=10)
svm_cv.fit(X, Y)
```

[77]:
```
GridSearchCV(cv=10, error_score='raise-deprecating',
       estimator=SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto_deprecated',
  kernel='rbf', max_iter=-1, probability=False, random_state=None,
```

```
  shrinking=True, tol=0.001, verbose=False),
       fit_params=None, iid='warn', n_jobs=None,
       param_grid={'kernel': ('linear', 'rbf', 'poly', 'rbf', 'sigmoid'), 'C':
array([1.00000e-03, 3.16228e-02, 1.00000e+00, 3.16228e+01, 1.00000e+03]),
'gamma': array([1.00000e-03, 3.16228e-02, 1.00000e+00, 3.16228e+01,
1.00000e+03])},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring=None, verbose=0)
```

[78]:
```python
print("tuned hpyerparameters :(best parameters) ",svm_cv.best_params_)
print("accuracy :",svm_cv.best_score_)
```

```
tuned hpyerparameters :(best parameters)  {'C': 1.0, 'gamma':
0.03162277660168379, 'kernel': 'sigmoid'}
accuracy : 0.8222222222222222
```

1.11 TASK 7

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

[79]:
```python
metrics.accuracy_score(Y_test, svm_cv.predict(X_test))
```

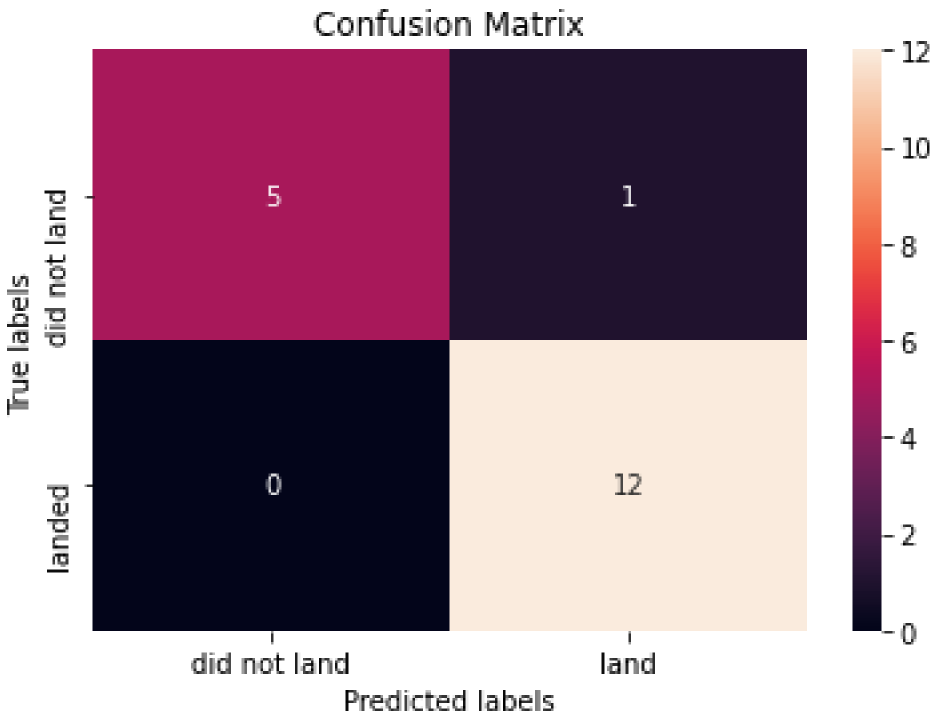
[79]:
```
0.9444444444444444
```

[80]:
```python
svm_cv.score(X_test, Y_test)
```

[80]:
```
0.9444444444444444
```

We can plot the confusion matrix

[81]:
```python
yhat=svm_cv.predict(X_test)
plot_confusion_matrix(Y_test,yhat)
```



1.12 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.

[82]:
```
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()
```

[83]:
```
tree_cv = GridSearchCV(tree, parameters, cv=10)
tree_cv.fit(X, Y)
```

[83]:
```
GridSearchCV(cv=10, error_score='raise-deprecating',
       estimator=DecisionTreeClassifier(class_weight=None, criterion='gini',
max_depth=None,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=None,
```

```
            splitter='best'),
       fit_params=None, iid='warn', n_jobs=None,
       param_grid={'criterion': ['gini', 'entropy'], 'splitter': ['best',
'random'], '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]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring=None, verbose=0)
```

[84]:
```
print("tuned hpyerparameters :(best parameters) ",tree_cv.best_params_)
print("accuracy :",tree_cv.best_score_)
```

```
tuned hpyerparameters :(best parameters)  {'criterion': 'gini', 'max_depth': 4,
'max_features': 'sqrt', 'min_samples_leaf': 1, 'min_samples_split': 2,
'splitter': 'best'}
accuracy : 0.8888888888888888
```

1.13 TASK 9

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

[85]:
```
metrics.accuracy_score(Y_test, tree_cv.predict(X_test))
```

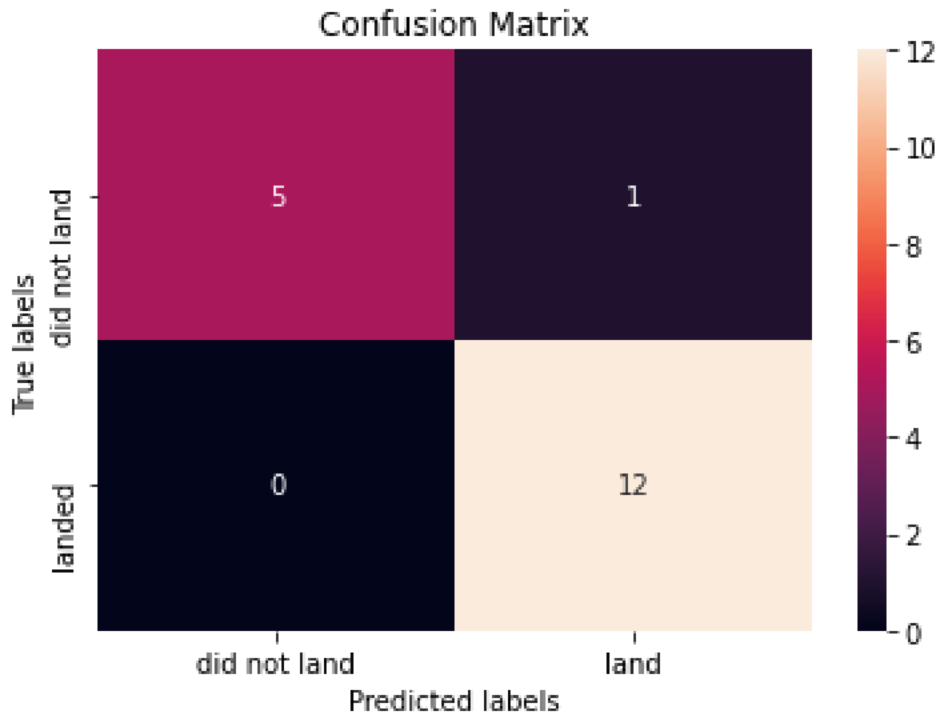
[85]: 0.9444444444444444

[86]:
```
tree_cv.score(X_test, Y_test)
```

[86]: 0.9444444444444444

We can plot the confusion matrix

[87]:
```
yhat = tree_cv.predict(X_test)
plot_confusion_matrix(Y_test,yhat)
```



1.14 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.

[88]:
```
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()
```

[89]:
```
knn_cv = GridSearchCV(KNN, parameters, cv=10)
knn_cv.fit(X, Y)
```

[89]:
```
GridSearchCV(cv=10, error_score='raise-deprecating',
       estimator=KNeighborsClassifier(algorithm='auto', leaf_size=30,
metric='minkowski',
           metric_params=None, n_jobs=None, n_neighbors=5, p=2,
           weights='uniform'),
       fit_params=None, iid='warn', n_jobs=None,
       param_grid={'n_neighbors': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10], 'algorithm':
['auto', 'ball_tree', 'kd_tree', 'brute'], 'p': [1, 2]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring=None, verbose=0)
```

[90]:
```
print("tuned hpyerparameters :(best parameters) ",knn_cv.best_params_)
print("accuracy :",knn_cv.best_score_)
```

```
tuned hpyerparameters :(best parameters)  {'algorithm': 'auto', 'n_neighbors':
5, 'p': 1}
accuracy : 0.8444444444444444
```

1.15 TASK 11

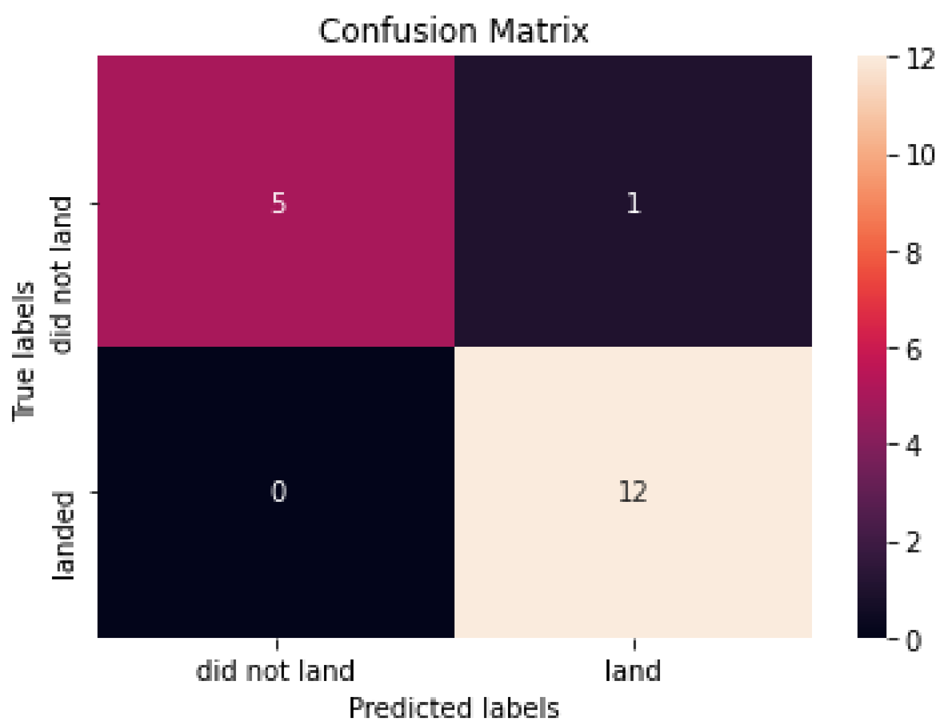
Calculate the accuracy of tree_cv on the test data using the method score:

[91]:
```
knn_cv.score(X_test, Y_test)
```

[91]: 0.9444444444444444

We can plot the confusion matrix

[92]:
```
yhat = knn_cv.predict(X_test)
plot_confusion_matrix(Y_test,yhat)
```



1.16 TASK 12

Find the method performs best:

[94]:
```
# Compare the best_scores of these tests:
# The decision tree had the highest best accuracy of 0.8888!
```

The test with the highest accuracy was the decision tree test!

1.17 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.

1.18 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