Space X Falcon 9 First Stage Landing Prediction

Nick Belgau

Objective

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

In [2]:
```
import pandas as pd #data manipulation & analysis
import numpy as np #matrix operations and high level mathematical functions
import matplotlib.pyplot as plt #plotting framework
import seaborn as sns #data visualization based on matplotlib

from sklearn import preprocessing #stadnardize the data
from sklearn.model_selection import train_test_split #split the data
from sklearn.model_selection import GridSearchCV #find optimized test paramete
rs

from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
```

This function is to plot the confusion matrix.

In [3]:
```
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

In [41]:
```
data = pd.read_csv("https://cf-courses-data.s3.us.cloud-object-storage.appdoma
in.cloud/IBM-DS0321EN-SkillsNetwork/datasets/dataset_part_2.csv")
data.head()
```

Out[41]:

	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 [42]:
```
X = pd.read_csv('https://cf-courses-data.s3.us.cloud-object-storage.appdomain.
cloud/IBM-DS0321EN-SkillsNetwork/datasets/dataset_part_3.csv')
X.head()
```

Out[42]:

	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

5 rows × 83 columns

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()
```

Standardize the data in `X` then reassign it to the variable `X`.

In [7]:
```
transform = preprocessing.StandardScaler()
X = transform.fit_transform(X)
```

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`.

Split data into train and test data

`X_train, X_test, Y_train, Y_test`

In [8]:
```
X_train, X_test, Y_train, Y_test = train_test_split(X, y, test_size=0.2, rando
m_state=2)
```

we can see we only have 18 test samples.

In [9]:
```
Y_test.shape
```

Out[9]:
```
(18,)
```

Creating a logistic regression object using then create a GridSearchCV object logreg_cv with cv = 10.

Fit the object to find the best parameters from the dictionary parameters

In [12]:
```
parameters ={"C":[0.01,0.1,1],'penalty':['l2'], 'solver':['lbfgs']} # l1 lasso
l2 ridge
lr=LogisticRegression()
gscv = GridSearchCV(lr,parameters,scoring='accuracy',cv=10)
logreg_cv = gscv.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 [43]:
```
print("tuned hyperparameters :(best parameters) ",logreg_cv.best_params_)
print("accuracy :",logreg_cv.best_score_)
```

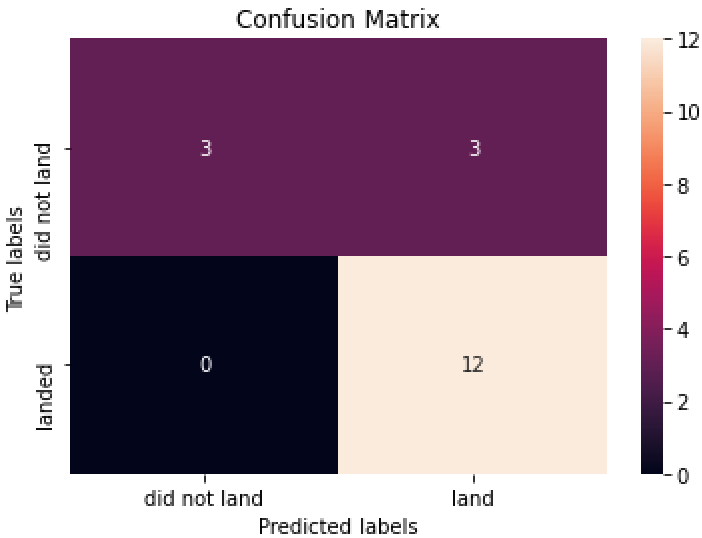
```
tuned hyperparameters :(best parameters)  {'C': 0.01, 'penalty': 'l2', 'solve
r': 'lbfgs'}
accuracy : 0.8464285714285713
```

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

In [15]:
```
print('Accuracy=  ',logreg_cv.score(X_test,Y_test))
```

```
Accuracy=   0.8333333333333334
```

In [44]:
```
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.

Creating SVM object then creating a `GridSearchCV` object `svm_cv` with cv - 10.

Fit the object to find the best parameters from the dictionary `parameters` .

In [21]:
```
parameters = {'kernel':('linear', 'rbf','poly','rbf', 'sigmoid'),
              'C': np.logspace(-3, 3, 5),
              'gamma':np.logspace(-3, 3, 5)}
svm = SVC()
gscv = GridSearchCV(svm,parameters,scoring='accuracy',cv=10)
svm_cv = gscv.fit(X_train,Y_train)
```

In [45]:
```
print("tuned hyperparameters :(best parameters) ",svm_cv.best_params_)
print("accuracy :",svm_cv.best_score_)
```

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

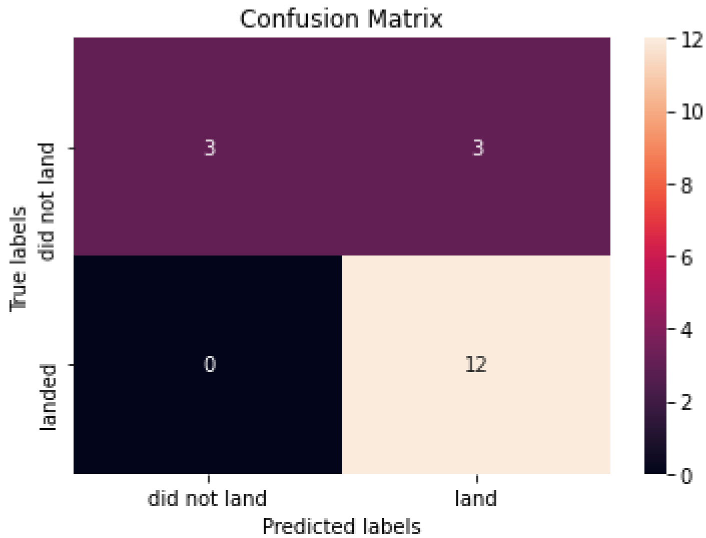
Calculate the accuracy on the test data using the method `score` :

In [23]:
```
print("accuracy: ",svm_cv.score(X_test,Y_test))
```

```
accuracy:  0.8333333333333334
```

We can plot the confusion matrix

In [24]:
```
yhat=svm_cv.predict(X_test)
plot_confusion_matrix(Y_test,yhat)
```



Creating a decision tree classifier object then creating a `GridSearchCV` object `tree_cv` with cv = 10.

Fit the object to find the best parameters from the dictionary `parameters`.

In [25]:
```
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 [26]:
```
gscv = GridSearchCV(tree,parameters,scoring='accuracy',cv=10)
tree_cv = gscv.fit(X_train,Y_train)
```

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

```
tuned hyperparameters :(best parameters)  {'criterion': 'gini', 'max_depth': 10, 'max_features': 'auto', 'min_samples_leaf': 2, 'min_samples_split': 2, 'splitter': 'best'}
accuracy : 0.8892857142857142
```

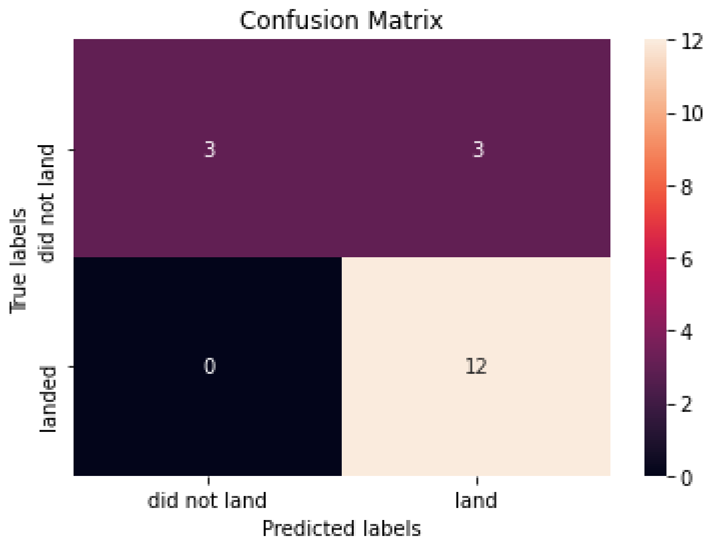
Calculating the accuracy of tree_cv on the test data using the method `score`:

In [28]:
```
print("accuracy: ",  tree_cv.score(X_test,Y_test))
```

```
accuracy:  0.7777777777777778
```

We can plot the confusion matrix

In [29]:
```
yhat = svm_cv.predict(X_test)
plot_confusion_matrix(Y_test,yhat)
```



Creating a KNN object then creating a `GridSearchCV` object `knn_cv` with cv = 10.

Fit the object to find the best parameters from the dictionary `parameters` .

In [30]:
```
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 [31]:
```
gscv = GridSearchCV(KNN,parameters,scoring='accuracy',cv=10)
knn_cv = gscv.fit(X_train,Y_train)
```

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

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

In [ ]:
```
print("accuracy: ",knn_cv.score(X_test,Y_test))
```

We can plot the confusion matrix

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

Finding the method performs best:

In [40]:
```
algorithms = {'KNN':knn_cv.best_score_,'Decision Tree':tree_cv.best_score_,'LogisticRegression':logreg_cv.best_score_}
bestalgorithm = max(algorithms, key=algorithms.get)
print('Best Algorithm:',bestalgorithm,'with a score of',algorithms[bestalgorithm])
```

```
Best Algorithm: Decision Tree with a score of 0.8892857142857142
```

In [39]:
```
if bestalgorithm == 'Decision Tree':
    print('Best Parameters:',tree_cv.best_params_)
if bestalgorithm == 'KNN':
    print('Best Parameters:',knn_cv.best_params_)
if bestalgorithm == 'LogisticRegression':
    print('Best Parameters:',logreg_cv.best_params_)
```

```
Best Parameters: {'criterion': 'gini', 'max_depth': 10, 'max_features': 'auto', 'min_samples_leaf': 2, 'min_samples_split': 2, 'splitter': 'best'}
```