**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**

[1]:

**import** piplite

**await** piplite.install(['numpy'])

**await** piplite.install(['pandas'])

**await** piplite.install(['seaborn'])

We will import the following libraries for the lab

[2]:

*# 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

This function is to plot the confusion matrix.

[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'])

plt.show()

**Load the dataframe**

Load the data

[4]:

**from** js **import** fetch

**import** io

URL1 **=** "https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DS0321EN-SkillsNetwork/datasets/dataset\_part\_2.csv"

resp1 **=** **await** fetch(URL1)

text1 **=** io.BytesIO((**await** resp1.arrayBuffer()).to\_py())

data **=** pd.read\_csv(text1)

[5]:

data.head()

[5]:

|  | **FlightNumber** | **Date** | **BoosterVersion** | **PayloadMass** | **Orbit** | **LaunchSite** | **Outcome** | **Flights** | **GridFins** | **Reused** | **Legs** | **LandingPad** | **Block** | **ReusedCount** | **Serial** | **Longitude** | **Latitude** | **Class** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 1 | 2010-06-04 | Falcon 9 | 6104.959412 | LEO | CCAFS SLC 40 | None None | 1 | False | False | False | NaN | 1.0 | 0 | B0003 | -80.577366 | 28.561857 | 0 |
| **1** | 2 | 2012-05-22 | Falcon 9 | 525.000000 | LEO | CCAFS SLC 40 | None None | 1 | False | False | False | NaN | 1.0 | 0 | B0005 | -80.577366 | 28.561857 | 0 |
| **2** | 3 | 2013-03-01 | Falcon 9 | 677.000000 | ISS | CCAFS SLC 40 | None None | 1 | False | False | False | NaN | 1.0 | 0 | B0007 | -80.577366 | 28.561857 | 0 |
| **3** | 4 | 2013-09-29 | Falcon 9 | 500.000000 | PO | VAFB SLC 4E | False Ocean | 1 | False | False | False | NaN | 1.0 | 0 | B1003 | -120.610829 | 34.632093 | 0 |
| **4** | 5 | 2013-12-03 | Falcon 9 | 3170.000000 | GTO | CCAFS SLC 40 | None None | 1 | False | False | False | NaN | 1.0 | 0 | B1004 | -80.577366 | 28.561857 | 0 |

[6]:

URL2 **=** 'https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DS0321EN-SkillsNetwork/datasets/dataset\_part\_3.csv'

resp2 **=** **await** fetch(URL2)

text2 **=** io.BytesIO((**await** resp2.arrayBuffer()).to\_py())

X **=** pd.read\_csv(text2)

[7]:

X.head(100)

[7]:

|  | **FlightNumber** | **PayloadMass** | **Flights** | **Block** | **ReusedCount** | **Orbit\_ES-L1** | **Orbit\_GEO** | **Orbit\_GTO** | **Orbit\_HEO** | **Orbit\_ISS** | **...** | **Serial\_B1058** | **Serial\_B1059** | **Serial\_B1060** | **Serial\_B1062** | **GridFins\_False** | **GridFins\_True** | **Reused\_False** | **Reused\_True** | **Legs\_False** | **Legs\_True** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 1.0 | 6104.959412 | 1.0 | 1.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 1.0 | 0.0 | 1.0 | 0.0 | 1.0 | 0.0 |
| **1** | 2.0 | 525.000000 | 1.0 | 1.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 1.0 | 0.0 | 1.0 | 0.0 | 1.0 | 0.0 |
| **2** | 3.0 | 677.000000 | 1.0 | 1.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 1.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 1.0 | 0.0 | 1.0 | 0.0 | 1.0 | 0.0 |
| **3** | 4.0 | 500.000000 | 1.0 | 1.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 1.0 | 0.0 | 1.0 | 0.0 | 1.0 | 0.0 |
| **4** | 5.0 | 3170.000000 | 1.0 | 1.0 | 0.0 | 0.0 | 0.0 | 1.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 1.0 | 0.0 | 1.0 | 0.0 | 1.0 | 0.0 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **85** | 86.0 | 15400.000000 | 2.0 | 5.0 | 2.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 1.0 | 0.0 | 0.0 | 1.0 | 0.0 | 1.0 | 0.0 | 1.0 |
| **86** | 87.0 | 15400.000000 | 3.0 | 5.0 | 2.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 1.0 | 0.0 | 0.0 | 0.0 | 0.0 | 1.0 | 0.0 | 1.0 | 0.0 | 1.0 |
| **87** | 88.0 | 15400.000000 | 6.0 | 5.0 | 5.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 1.0 | 0.0 | 1.0 | 0.0 | 1.0 |
| **88** | 89.0 | 15400.000000 | 3.0 | 5.0 | 2.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 1.0 | 0.0 | 0.0 | 1.0 | 0.0 | 1.0 | 0.0 | 1.0 |
| **89** | 90.0 | 3681.000000 | 1.0 | 5.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 1.0 | 0.0 | 1.0 | 1.0 | 0.0 | 0.0 | 1.0 |

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']).

[8]:

Y **=** data['Class'].to\_numpy()

**TASK 2**

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

[14]:

*# students get this*

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.

**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

[15]:

**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, random\_state**=**2)

we can see we only have 18 test samples.

[16]:

Y\_test.shape

[16]:

(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.

[17]:

parameters **=**{'C':[0.01,0.1,1],

'penalty':['l2'],

'solver':['lbfgs']}

[21]:

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\_train, Y\_train)

[21]:

GridSearchCV

estimator: LogisticRegression

LogisticRegression

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\_.

[22]:

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 : 0.8464285714285713

**TASK 5**

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

[25]:

logreg\_cv.score(X\_test, Y\_test)

[25]:

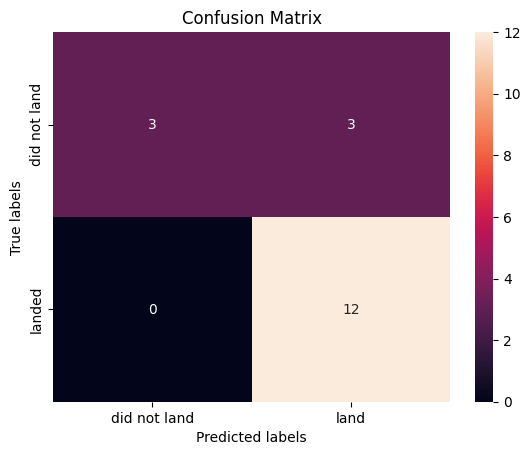
0.8333333333333334

Lets look at the confusion matrix:

[26]:

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.

[28]:

parameters **=** {'kernel':('linear', 'rbf','poly','rbf', 'sigmoid'),

'C': np.logspace(**-**3, 3, 5),

'gamma':np.logspace(**-**3, 3, 5)}

svm **=** SVC()

[29]:

svm\_cv **=** GridSearchCV(svm, parameters, cv **=** 10)

svm\_cv.fit(X\_train, Y\_train)

[29]:

GridSearchCV

estimator: SVC

SVC

[30]:

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 : 0.8482142857142856

**TASK 7**

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

[31]:

svm\_cv.score(X\_test, Y\_test)

[31]:

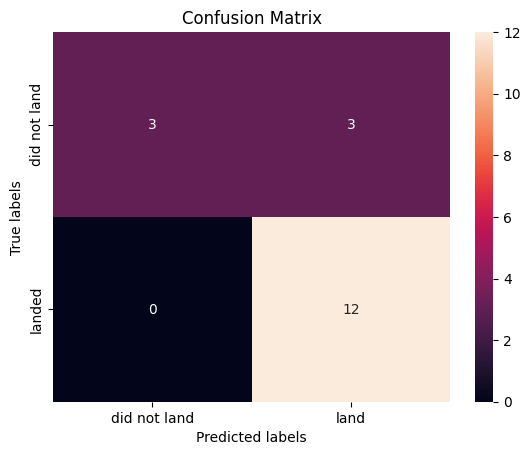
0.8333333333333334

We can plot the confusion matrix

[32]:

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.

[33]:

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()

[35]:

tree\_cv **=** GridSearchCV(tree, parameters, cv **=** 10)

tree\_cv.fit(X\_train, Y\_train)

/lib/python3.11/site-packages/sklearn/model\_selection/\_validation.py:425: FitFailedWarning:

3240 fits failed out of a total of 6480.

The score on these train-test partitions for these parameters will be set to nan.

If these failures are not expected, you can try to debug them by setting error\_score='raise'.

Below are more details about the failures:

[35]:

GridSearchCV

estimator: DecisionTreeClassifier

DecisionTreeClassifier

[36]:

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': 5, 'splitter': 'best'}

accuracy : 0.8732142857142857

**TASK 9**

Calculate the accuracy of tree\_cv on the test data using the method score:

[37]:

tree\_cv.score(X\_test, Y\_test)

[37]:

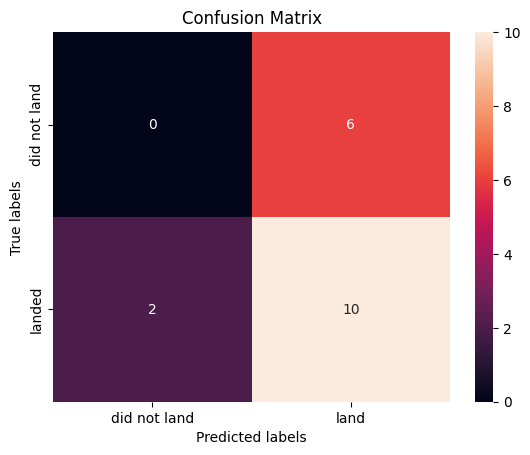
0.5555555555555556

We can plot the confusion matrix

[38]:

yhat **=** tree\_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.

[43]:

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()

[44]:

knn\_cv **=** GridSearchCV(knn, parameters, cv **=** 10)

knn\_cv.fit(X\_train, Y\_train)

[44]:

GridSearchCV

estimator: KNeighborsClassifier

KNeighborsClassifier

[45]:

print("tuned hpyerparameters :(best parameters) ",knn\_cv.best\_params\_)

print("accuracy :",knn\_cv.best\_score\_)

tuned hpyerparameters :(best parameters) {'algorithm': 'auto', 'n\_neighbors': 10, 'p': 1}

accuracy : 0.8482142857142858

**TASK 11**

Calculate the accuracy of knn\_cv on the test data using the method score:

[46]:

knn\_cv.score(X\_test, Y\_test)

[46]:

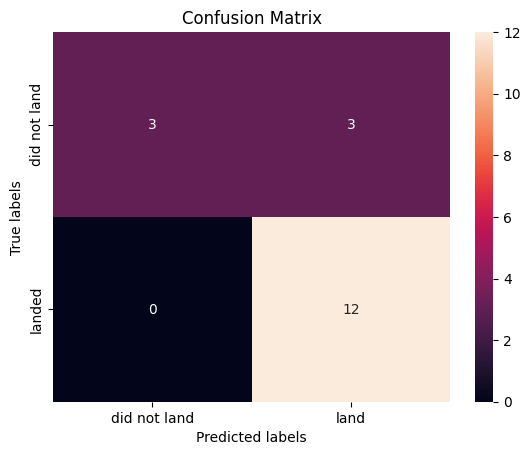
0.8333333333333334

We can plot the confusion matrix

[47]:

yhat **=** knn\_cv.predict(X\_test)

plot\_confusion\_matrix(Y\_test,yhat)



**TASK 12**

Find the method performs best: