Network

Space X Falcon 9 First Stage Landing Prediction

Hands on Lab: Complete the Machine Learning Prediction lab

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:

odide) | Unknown Mode: Command Ln 14, Col 49 SpaceX_Machine Learning Prediction_Part_5.ipynb 2

Launcher | SpaceX_Machine Learning Pr

Code | Python (Pyodide)

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

[*]:
```
import piplite
await piplite.install(['numpy'])
await piplite.install(['pandas'])
await piplite.install(['seaborn'])
```

We will import the following libraries for the lab

[ ]:
```
# 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-lev
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
```

yodide) | Unknown   Mode: Command   Ln 14, Col 49   SpaceX_Machine Learning Prediction_Part_5.ipynb   2

```
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.

[ ]:
```
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

[ ]:
```
from js import fetch
```

[ ]:

```python
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]:

```python
data.head()
```

[5]:

	FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Flights	GridFins	Reused	Legs	LandingPad	Block	ReusedCount	Serial	Longitude	Latitu
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	28.5618
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	28.5618
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	28.5618
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	34.6320
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	28.5618

[6]:

```python
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]:

```python
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	G

[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	G
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	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	
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	
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	
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	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
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	
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	
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	
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	
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	

90 rows × 83 columns

TASK 1

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.

[11]:
```
# 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`

[13]:
```
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.

[14]:
```
Y_test.shape
```

[14]: (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`.

[15]:
```
parameters ={'C':[0.01,0.1,1],
             'penalty':['l2'],
             'solver':['lbfgs']}
```

[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)
logreg_cv.fit(X_train,Y_train)
```

[20]:
```
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_`.

[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', 'solver': 'lbfgs'}
accuracy : 0.8464285714285713
```

TASK 5

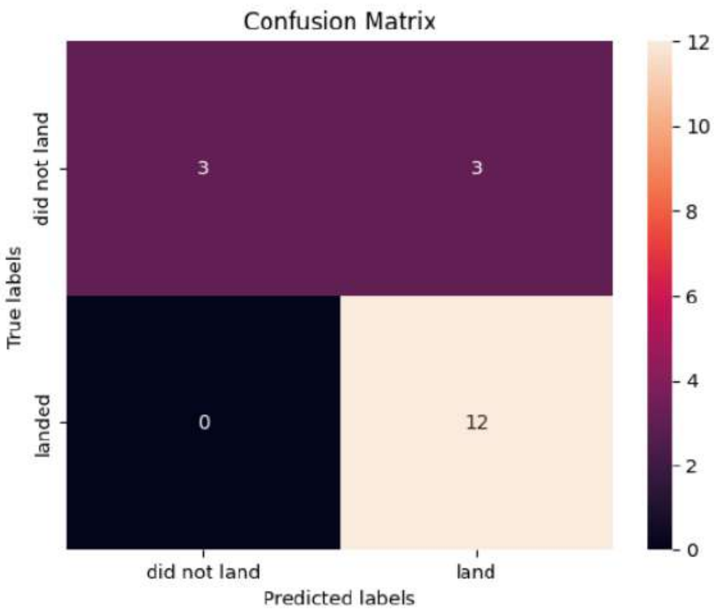
Calculate the accuracy on the test data using the method `score` :

[22]:
```
test_accuracy=logreg_cv.score(X_test,Y_test)
print("Test set accuracy:",test_accuracy)
```

```
Test set accuracy: 0.8333333333333334
```

Lets look at the confusion matrix:

[23]:
```
yhat=logreg_cv.predict(X_test)
plot_confusion_matrix(Y_test,yhat)
```



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`.

[24]:
```
parameters = {'kernel':('linear', 'rbf','poly','rbf', 'sigmoid'),
              'C': np.logspace(-3, 3, 5),
              'gamma':np.logspace(-3, 3, 5)}
svm = SVC()
```

[25]:
```
svm_cv=GridSearchCV(svm,parameters,cv=10)
svm_cv.fit(X_train,Y_train)
```

[25]:
```
GridSearchCV
  estimator: SVC
    SVC
```

[26]:
```
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`:

[28]:
```
test_accuracy=svm_cv.score(X_test,Y_test)
print("Test accuracy: ",test_accuracy)
```

```
Test accuracy:  0.8333333333333334
```

We can plot the confusion matrix

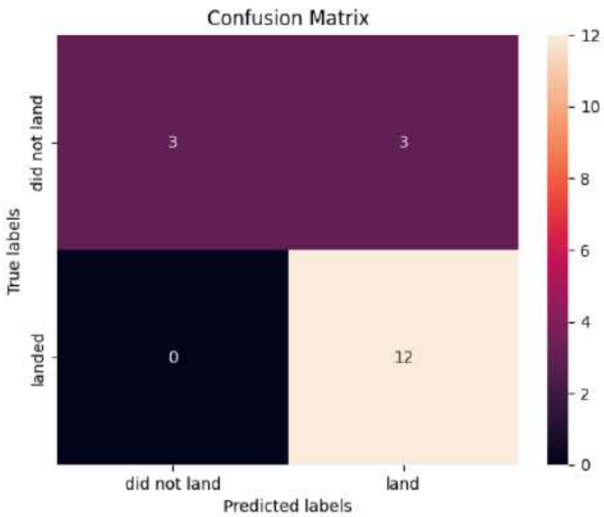
Launcher | SpaceX_Machine Learning Pr

Python (Pyodide)

```
Test accuracy:  0.8333333333333334
```

We can plot the confusion matrix

```
[29]: 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`.

n (Pyodide) | Unknown   Mode: Command   Ln 14, Col 49   SpaceX_Machine Learning Prediction_Part_5.ipynb   2

ENG   22:01

Launcher | SpaceX_Machine Learning Pr

Code | Python (Pyodide)

```
[30]: 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()
```

```
[32]: tree_cv=GridSearchCV(tree,parameters,cv=10)
tree_cv.fit(X_train,Y_train)
```

```
/lib/python3.12/site-packages/sklearn/model_selection/_validation.py:547: 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:
--------------------------------------------------------------------------------
3240 fits failed with the following error:
Traceback (most recent call last):
  File "/lib/python3.12/site-packages/sklearn/model_selection/_validation.py", line 895, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "/lib/python3.12/site-packages/sklearn/base.py", line 1467, in wrapper
    estimator._validate_params()
  File "/lib/python3.12/site-packages/sklearn/base.py", line 666, in _validate_params
    validate_parameter_constraints(
  File "/lib/python3.12/site-packages/sklearn/utils/_param_validation.py", line 95, in validate_parameter_constraints
    raise InvalidParameterError(
sklearn.utils._param_validation.InvalidParameterError: The 'max_features' parameter of DecisionTreeClassifier must be an int in the range [1, inf), a float in the r
ange (0.0, 1.0], a str among {'sqrt', 'log2'} or None. Got 'auto' instead.

  warnings.warn(some_fits_failed_message, FitFailedWarning)
/lib/python3.12/site-packages/sklearn/model_selection/_search.py:1051: UserWarning: One or more of the test scores are non-finite: [       nan        nan        nan
nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
 0.67142857 0.76071429 0.80535714 0.79285714 0.79107143 0.7625
 0.74642857 0.71071429 0.75357143 0.83392857 0.725      0.7625
 0.79464286 0.76071429 0.83214286 0.76607143 0.72321429 0.78214286
        nan        nan        nan        nan        nan        nan
```

Pyodide) | Unknown   Mode: Command   Ln 14, Col 49   SpaceX_Machine Learning Prediction_Part_5.ipynb   2

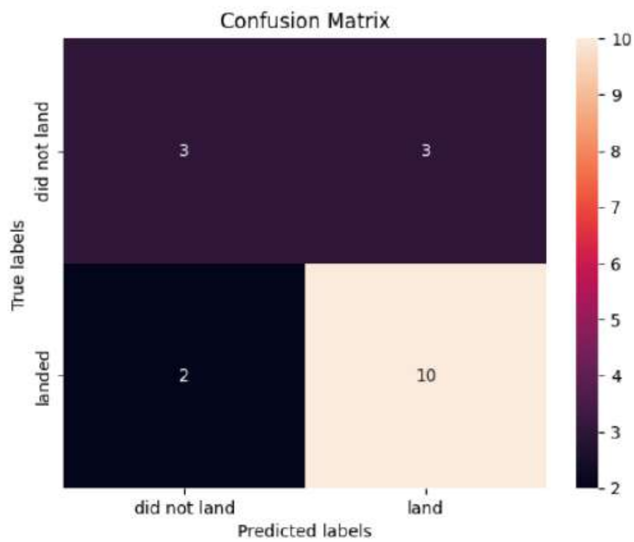
Code Python (Pyodide)

```
print("Test  accuracy:",tree_cv_accuracy)
```

```
Test  accuracy: 0.7222222222222222
```

We can plot the confusion matrix

```
[36]: 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`.

Code Python (Pyodide)

did not land land
Predicted labels

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`.

[37]:
```
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()
```

[38]:
```
knn_cv=GridSearchCV(KNN,parameters,cv=10)
knn_cv.fit(X_train,Y_train)
```

[38]:
```
GridSearchCV
estimator: KNeighborsClassifier
KNeighborsClassifier
```

[39]:
```
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`:

[40]:
```
knn_test_accuracy = knn_cv.score(X_test, Y_test)
print("Test set accuracy:", test_accuracy)
```

TASK 11

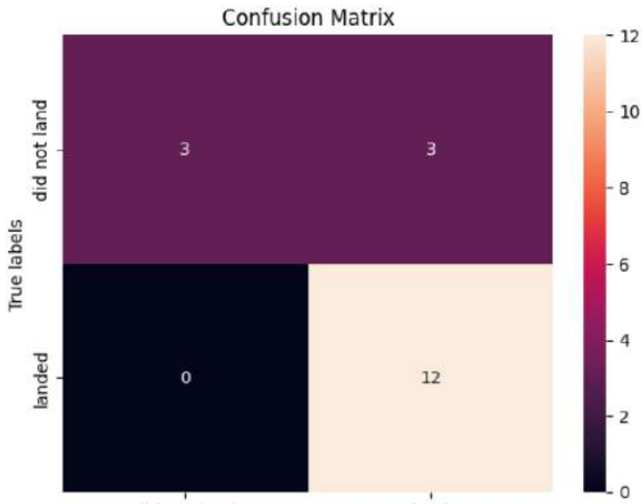
Calculate the accuracy of knn_cv on the test data using the method `score` :

[40]:
```
knn_test_accuracy = knn_cv.score(X_test, Y_test)
print("Test set accuracy:", test_accuracy)
```

```
Test set accuracy: 0.8333333333333334
```

We can plot the confusion matrix

[41]:
```
yhat = knn_cv.predict(X_test)
plot_confusion_matrix(Y_test,yhat)
```



Code

TASK 12

Find the method performs best:

[ ]:
```
accuracy_results = {}
for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    acc = accuracy_score(y_test, y_pred)
    accuracy_results[name] = acc

# Plot bar chart
plt.figure(figsize=(8, 5))
plt.bar(accuracy_results.keys(), accuracy_results.values(), color='skyblue')
plt.title('Model Accuracy Comparison')
plt.xlabel('Model')
plt.ylabel('Accuracy')
plt.ylim(0, 1)
plt.grid(axis='y')
plt.show()
```

Authors