Rocks vs Mine Prediction ML Projects

July 30, 2023

0.1 Importing Liberies

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

1 Data Collection and Data Processing

[2]:
```
# Load the Dataset
```

[4]:
```
df = pd.read_csv("D:\DataSets\sonar-data.csv",header = None)
```

[5]:
```
df.head()
```

[5]:

	0	1	2	3	4	5	6	7	8	\
0	0.0200	0.0371	0.0428	0.0207	0.0954	0.0986	0.1539	0.1601	0.3109	
1	0.0453	0.0523	0.0843	0.0689	0.1183	0.2583	0.2156	0.3481	0.3337	
2	0.0262	0.0582	0.1099	0.1083	0.0974	0.2280	0.2431	0.3771	0.5598	
3	0.0100	0.0171	0.0623	0.0205	0.0205	0.0368	0.1098	0.1276	0.0598	
4	0.0762	0.0666	0.0481	0.0394	0.0590	0.0649	0.1209	0.2467	0.3564	

	9	…	51	52	53	54	55	56	57	\
0	0.2111	…	0.0027	0.0065	0.0159	0.0072	0.0167	0.0180	0.0084	
1	0.2872	…	0.0084	0.0089	0.0048	0.0094	0.0191	0.0140	0.0049	
2	0.6194	…	0.0232	0.0166	0.0095	0.0180	0.0244	0.0316	0.0164	
3	0.1264	…	0.0121	0.0036	0.0150	0.0085	0.0073	0.0050	0.0044	
4	0.4459	…	0.0031	0.0054	0.0105	0.0110	0.0015	0.0072	0.0048	

	58	59	60
0	0.0090	0.0032	R
1	0.0052	0.0044	R
2	0.0095	0.0078	R
3	0.0040	0.0117	R
4	0.0107	0.0094	R

[5 rows x 61 columns]

[6]:
```
df.shape
```

[6]:
```
(208, 61)
```

[10]:
```
df.columns
```

[10]:
```
Int64Index([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
            17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33,
            34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50,
            51, 52, 53, 54, 55, 56, 57, 58, 59, 60],
           dtype='int64')
```

[12]:
```
df.describe()
```

[12]:

	0	1	2	3	4	5	\
count	208.000000	208.000000	208.000000	208.000000	208.000000	208.000000	
mean	0.029164	0.038437	0.043832	0.053892	0.075202	0.104570	
std	0.022991	0.032960	0.038428	0.046528	0.055552	0.059105	
min	0.001500	0.000600	0.001500	0.005800	0.006700	0.010200	
25%	0.013350	0.016450	0.018950	0.024375	0.038050	0.067025	
50%	0.022800	0.030800	0.034300	0.044050	0.062500	0.092150	
75%	0.035550	0.047950	0.057950	0.064500	0.100275	0.134125	
max	0.137100	0.233900	0.305900	0.426400	0.401000	0.382300	

	6	7	8	9	…	50	\
count	208.000000	208.000000	208.000000	208.000000	…	208.000000	
mean	0.121747	0.134799	0.178003	0.208259	…	0.016069	
std	0.061788	0.085152	0.118387	0.134416	…	0.012008	
min	0.003300	0.005500	0.007500	0.011300	…	0.000000	
25%	0.080900	0.080425	0.097025	0.111275	…	0.008425	
50%	0.106950	0.112100	0.152250	0.182400	…	0.013900	
75%	0.154000	0.169600	0.233425	0.268700	…	0.020825	
max	0.372900	0.459000	0.682800	0.710600	…	0.100400	

	51	52	53	54	55	56	\
count	208.000000	208.000000	208.000000	208.000000	208.000000	208.000000	
mean	0.013420	0.010709	0.010941	0.009290	0.008222	0.007820	
std	0.009634	0.007060	0.007301	0.007088	0.005736	0.005785	
min	0.000800	0.000500	0.001000	0.000600	0.000400	0.000300	
25%	0.007275	0.005075	0.005375	0.004150	0.004400	0.003700	
50%	0.011400	0.009550	0.009300	0.007500	0.006850	0.005950	
75%	0.016725	0.014900	0.014500	0.012100	0.010575	0.010425	
max	0.070900	0.039000	0.035200	0.044700	0.039400	0.035500	

	57	58	59
count	208.000000	208.000000	208.000000
mean	0.007949	0.007941	0.006507

```
std      0.006470    0.006181    0.005031
min      0.000300    0.000100    0.000600
25%      0.003600    0.003675    0.003100
50%      0.005800    0.006400    0.005300
75%      0.010350    0.010325    0.008525
max      0.044000    0.036400    0.043900

[8 rows x 60 columns]
```

[19]:
```
df[60].value_counts()
```

[19]:
```
M    111
R     97
Name: 60, dtype: int64
```

[21]:
```
df.groupby(60).mean()
```

[21]:
```
           0         1         2         3         4         5         6   \
60
M   0.034989  0.045544  0.050720  0.064768  0.086715  0.111864  0.128359
R   0.022498  0.030303  0.035951  0.041447  0.062028  0.096224  0.114180

           7         8         9   …        50        51        52        53  \
60                                   …
M   0.149832  0.213492  0.251022  …  0.019352  0.016014  0.011643  0.012185
R   0.117596  0.137392  0.159325  …  0.012311  0.010453  0.009640  0.009518

           54        55        56        57        58        59
60
M   0.009923  0.008914  0.007825  0.009060  0.008695  0.006930
R   0.008567  0.007430  0.007814  0.006677  0.007078  0.006024

[2 rows x 60 columns]
```

[22]:
```
# Seprating the Data and Label
```

[24]:
```
X = df.drop(columns = 60,axis=1)
Y = df[60]
```

[27]:
```
Y.reset_index()
```

[27]:
```
     index 60
0        0  R
1        1  R
2        2  R
3        3  R
4        4  R
..     … ..
```

```
203    203  M
204    204  M
205    205  M
206    206  M
207    207  M

[208 rows x 2 columns]
```

1.1 Splitting the Dataset (Train and Test)

[28]:
```
from sklearn.model_selection import train_test_split
X_train,X_test,Y_train,Y_test = train_test_split(X,Y,test_size=0.1,stratify =Y␣
 ↪,random_state =1)
```

[31]:
```
X.shape,X_train.shape,X_test.shape
```

[31]:
```
((208, 60), (187, 60), (21, 60))
```

[36]:
```
Y_test.shape
```

[36]:
```
(21,)
```

1.2 Model Training –> Logistics Regression

[38]:
```
from sklearn.linear_model import LogisticRegression
```

[39]:
```
lr = LogisticRegression()
```

[40]:
```
# Traing the logistics Regression  model with traing data
```

[41]:
```
lr.fit(X_train,Y_train)
```

[41]:
```
LogisticRegression()
```

[44]:
```
# Check the accuracy
```

[43]:
```
from sklearn.metrics import accuracy_score
```

[48]:
```
# Accuracy of train data
X_train_prediction = lr.predict(X_train)
training_data_accuracy = accuracy_score(X_train_prediction,Y_train)
```

[47]:
```
print('Accuracy on training data :' ,training_data_accuracy)
```

```
Accuracy on training data : 0.8342245989304813
```

[50]:
```
# Accuracy of test data
X_test_prediction = lr.predict(X_test)
```

```
test_data_accuracy = accuracy_score(X_test_prediction,Y_test)
```

[51]:
```
print('Accuracy on Test data :' ,test_data_accuracy)
```

```
Accuracy on Test data : 0.7619047619047619
```

1.3 Making the prediction

[65]:
```
input_data=(0.0443,0.0446,0.0235,0.1008,0.2252,0.2611,0.2061,0.1668,0.1801,0.
↪3083,0.3794,0.5364,0.6173,0.7842,0.8392,0.9016,1.0000,0.8911,0.8753,0.7886,0.
↪7156,0.7581,0.6372,0.3210,0.2076,0.2279,0.3309,0.2847,0.1949,0.1671,0.1025,0.
↪1362,0.2212,0.1124,0.1677,0.1039,0.2562,0.2624,0.2236,0.1180,0.1103,0.2831,0.
↪2385,0.0255,0.1967,0.1483,0.0434,0.0627,0.0513,0.0473,0.0248,0.0274,0.0205,0.
↪0141,0.0185,0.0055,0.0045,0.0115,0.0152,0.0100)

# changing the input_data to a numpy array
input_data_as_numpy_array = np.asarray(input_data)

# reshaped the np array as we are prediction for one instance
input_data_reshaped = input_data_as_numpy_array.reshape(1,-1)

prediction =  lr.predict(input_data_reshaped)
print(prediction)

if(prediction[0]== 'R'):
    print('The Object is a Rock ')
else:
    print('The object is a Mine')
```

```
['M']
The object is a Mine
```

[66]:
```
import joblib

# Download the trained Logistic Regression model
joblib.dump(lr, 'trained_model.joblib')
```

[66]:
```
['trained_model.joblib']
```

[ ]:
```

```