Rock Vs Mine Prediction

August 5, 2024

[1]:
```
#importing the necessary libraries
import pandas as pd #for loading data into tables
import numpy as np #for arrays
from sklearn.model_selection import train_test_split #to split our data into␣
 ↪training and test data
from sklearn.linear_model import LogisticRegression #supervised learning␣
 ↪algorithm use to predict a dependent categorical target variable
from sklearn.metrics import accuracy_score #it calculates the accuracy of a␣
 ↪classification model
```

Data Collection and Data Processing

[2]:
```
#loading the data to to pandas DataFrame

sonar = pd.read_csv('sonar_data.csv', header = None)
```

[3]:
```
#first five rows

sonar.head()
```

[3]:

	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]
```

[4]:
```
#checking random file samples from your dataset.
sonar.sample(5)
```

[4]:
```
         0       1       2       3       4       5       6       7       8   \
100  0.0629  0.1065  0.1526  0.1229  0.1437  0.1190  0.0884  0.0907  0.2107
176  0.0635  0.0709  0.0453  0.0333  0.0185  0.1260  0.1015  0.1918  0.3362
54   0.0132  0.0080  0.0188  0.0141  0.0436  0.0668  0.0609  0.0131  0.0899
169  0.0130  0.0120  0.0436  0.0624  0.0428  0.0349  0.0384  0.0446  0.1318
34   0.0311  0.0491  0.0692  0.0831  0.0079  0.0200  0.0981  0.1016  0.2025

         9   …      51      52      53      54      55      56      57  \
100  0.3597  …  0.0089  0.0262  0.0108  0.0138  0.0187  0.0230  0.0057
176  0.3900  …  0.0048  0.0025  0.0087  0.0072  0.0095  0.0086  0.0085
54   0.0922  …  0.0044  0.0028  0.0021  0.0022  0.0048  0.0138  0.0140
169  0.1375  …  0.0084  0.0100  0.0018  0.0035  0.0058  0.0011  0.0009
34   0.0767  …  0.0087  0.0032  0.0130  0.0188  0.0101  0.0229  0.0182

         58      59  60
100  0.0113  0.0131   M
176  0.0040  0.0051   M
54   0.0028  0.0064   R
169  0.0033  0.0026   M
34   0.0046  0.0038   R

[5 rows x 61 columns]
```

[5]:
```
#number of rows and column
sonar.shape
```

[5]:
```
(208, 61)
```

[6]:
```
#describing the statistical numbers of thedata
sonar.describe()
```

[6]:
```
                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]
```

[7]:
```
#checking for the total count of Rock(s) Vs Mines(s) using our target column to␣
 ↪know if our data is fit for our ML model.
# M == Mine
# R == Rocks
#N/B column 60 is the target column
sonar[60].value_counts()
```

[7]:
```
60
M    111
R     97
Name: count, dtype: int64
```

[8]:
```
#grouping the dataset base on M and R

sonar.groupby(60).mean()
```

[8]:
```
          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]
```

[9]:
```
#seperating the data from the labels
x = sonar.drop(columns=60, axis = 1)
y = sonar[60]
```

[10]:
```
print(x)
print(y)
```

```
        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
..      …       …       …       …       …       …       …       …       …
203  0.0187  0.0346  0.0168  0.0177  0.0393  0.1630  0.2028  0.1694  0.2328
204  0.0323  0.0101  0.0298  0.0564  0.0760  0.0958  0.0990  0.1018  0.1030
205  0.0522  0.0437  0.0180  0.0292  0.0351  0.1171  0.1257  0.1178  0.1258
206  0.0303  0.0353  0.0490  0.0608  0.0167  0.1354  0.1465  0.1123  0.1945
207  0.0260  0.0363  0.0136  0.0272  0.0214  0.0338  0.0655  0.1400  0.1843

          9   …      50      51      52      53      54      55      56  \
0    0.2111  …  0.0232  0.0027  0.0065  0.0159  0.0072  0.0167  0.0180
1    0.2872  …  0.0125  0.0084  0.0089  0.0048  0.0094  0.0191  0.0140
2    0.6194  …  0.0033  0.0232  0.0166  0.0095  0.0180  0.0244  0.0316
3    0.1264  …  0.0241  0.0121  0.0036  0.0150  0.0085  0.0073  0.0050
4    0.4459  …  0.0156  0.0031  0.0054  0.0105  0.0110  0.0015  0.0072
..      …  …      …       …       …       …       …       …       …
```

```
203  0.2684  …  0.0203  0.0116  0.0098  0.0199  0.0033  0.0101  0.0065
204  0.2154  …  0.0051  0.0061  0.0093  0.0135  0.0063  0.0063  0.0034
205  0.2529  …  0.0155  0.0160  0.0029  0.0051  0.0062  0.0089  0.0140
206  0.2354  …  0.0042  0.0086  0.0046  0.0126  0.0036  0.0035  0.0034
207  0.2354  …  0.0181  0.0146  0.0129  0.0047  0.0039  0.0061  0.0040

         57      58      59
0    0.0084  0.0090  0.0032
1    0.0049  0.0052  0.0044
2    0.0164  0.0095  0.0078
3    0.0044  0.0040  0.0117
4    0.0048  0.0107  0.0094
..      …       …       …
203  0.0115  0.0193  0.0157
204  0.0032  0.0062  0.0067
205  0.0138  0.0077  0.0031
206  0.0079  0.0036  0.0048
207  0.0036  0.0061  0.0115

[208 rows x 60 columns]
0      R
1      R
2      R
3      R
4      R
      ..
203    M
204    M
205    M
206    M
207    M
Name: 60, Length: 208, dtype: object
```

Train and Test Data

[11]:
```
#spliting the data into training and test data
# test_size 0.2 = 20% data,
#y = stratify, it ensures that the distrribution of y is maintained in both␣
 ↪y_train and x_train
#random_state = the same data split will be generated everytime I run the code

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.1,␣
 ↪stratify = y, random_state=1)
```

[12]:
```
print(x_train)
print(y_train)
```

```
         0       1       2       3       4       5       6       7       8   \
115  0.0414  0.0436  0.0447  0.0844  0.0419  0.1215  0.2002  0.1516  0.0818
```

```
38   0.0123  0.0022  0.0196  0.0206  0.0180  0.0492  0.0033  0.0398  0.0791
56   0.0152  0.0102  0.0113  0.0263  0.0097  0.0391  0.0857  0.0915  0.0949
123  0.0270  0.0163  0.0341  0.0247  0.0822  0.1256  0.1323  0.1584  0.2017
18   0.0270  0.0092  0.0145  0.0278  0.0412  0.0757  0.1026  0.1138  0.0794
..      …       …       …       …       …       …       …       …
140  0.0412  0.1135  0.0518  0.0232  0.0646  0.1124  0.1787  0.2407  0.2682
5    0.0286  0.0453  0.0277  0.0174  0.0384  0.0990  0.1201  0.1833  0.2105
154  0.0117  0.0069  0.0279  0.0583  0.0915  0.1267  0.1577  0.1927  0.2361
131  0.1150  0.1163  0.0866  0.0358  0.0232  0.1267  0.2417  0.2661  0.4346
203  0.0187  0.0346  0.0168  0.0177  0.0393  0.1630  0.2028  0.1694  0.2328

          9   …      50      51      52      53      54      55      56  \
115  0.1975   …  0.0222  0.0045  0.0136  0.0113  0.0053  0.0165  0.0141
38   0.0475   …  0.0149  0.0125  0.0134  0.0026  0.0038  0.0018  0.0113
56   0.1504   …  0.0048  0.0049  0.0041  0.0036  0.0013  0.0046  0.0037
123  0.2122   …  0.0197  0.0189  0.0204  0.0085  0.0043  0.0092  0.0138
18   0.1520   …  0.0045  0.0084  0.0010  0.0018  0.0068  0.0039  0.0120
..      …   …       …       …       …       …       …       …       …
140  0.2058   …  0.0798  0.0376  0.0143  0.0272  0.0127  0.0166  0.0095
5    0.3039   …  0.0104  0.0045  0.0014  0.0038  0.0013  0.0089  0.0057
154  0.2169   …  0.0039  0.0053  0.0029  0.0020  0.0013  0.0029  0.0020
131  0.5378   …  0.0228  0.0099  0.0065  0.0085  0.0166  0.0110  0.0190
203  0.2684   …  0.0203  0.0116  0.0098  0.0199  0.0033  0.0101  0.0065

         57      58      59
115  0.0077  0.0246  0.0198
38   0.0058  0.0047  0.0071
56   0.0011  0.0034  0.0033
123  0.0094  0.0105  0.0093
18   0.0132  0.0070  0.0088
..      …       …       …
140  0.0225  0.0098  0.0085
5    0.0027  0.0051  0.0062
154  0.0062  0.0026  0.0052
131  0.0141  0.0068  0.0086
203  0.0115  0.0193  0.0157

[187 rows x 60 columns]
115    M
38     R
56     R
123    M
18     R
      ..
140    M
5      R
154    M
131    M
```

```
203    M
Name: 60, Length: 187, dtype: object
```

[13]:
```
#checking the shape of the data
#x_train.shap = 80%
#x_test.shape = 20%

print(x.shape, x_train.shape, x_test.shape)
```

```
(208, 60) (187, 60) (21, 60)
```

Model Training using Logistic Model

[14]:
```
#loading the logistic regerssion function into the variable model
model = LogisticRegression()
```

[15]:
```
#Training the logistic Regression Model with training data
# x_train = training data and y_train is the label

model.fit(x_train, y_train)
```

[15]:
```
LogisticRegression()
```

Model Evaluation

[16]:
```
#accuracy score of our model

x_train_prediction = model.predict(x_train)
training_data_accuracy = accuracy_score(x_train_prediction, y_train)
```

[17]:
```
print('Accuracy score on training data: ', training_data_accuracy)
```

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

[18]:
```
#accuracy score of our model

x_test_prediction = model.predict(x_test)
test_data_accuracy = accuracy_score(x_test_prediction, y_test)
```

[19]:
```
print('Accuracy score on test data: ', test_data_accuracy)
```

```
Accuracy score on test data:  0.7619047619047619
```

Making a Predictive System

[22]:
```
input_data = (0.0200,0.0371,0.0428,0.0207,0.0954,0.0986,0.1539,0.1601,0.3109,0.
↪2111,0.1609,0.1582,0.2238,0.0645,0.0660,0.2273,0.3100,0.2999,0.5078,0.4797,0.
↪5783,0.5071,0.4328,0.5550,0.6711,0.6415,0.7104,0.8080,0.6791,0.3857,0.1307,0.
↪2604,0.5121,0.7547,0.8537,0.8507,0.6692,0.6097,0.4943,0.2744,0.0510,0.2834,0.
↪2825,0.4256,0.2641,0.1386,0.1051,0.1343,0.0383,0.0324,0.0232,0.0027,0.0065,0.
↪0159,0.0072,0.0167,0.0180,0.0084,0.0090,0.0032)
```

```
#changing the input data to numpy arrray since the processing is more efficient
input_data_to_numpy_array = np.asarray(input_data)

#reshape the np array as we are predicting for one instance
input_data_reshaped = input_data_to_numpy_array.reshape(1, -1)
#1, -1 == one instanecand we are going to predict it

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


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

```
['R']
The object is a Rock
```

[23]:
```
input_data = (0.0163,0.0198,0.0202,0.0386,0.0752,0.1444,0.1487,0.1484,0.2442,0.
 ↪2822,0.3691,0.3750,0.3927,0.3308,0.1085,0.1139,0.3446,0.5441,0.6470,0.7276,0.
 ↪7894,0.8264,0.8697,0.7836,0.7140,0.5698,0.2908,0.4636,0.6409,0.7405,0.8069,0.
 ↪8420,1.0000,0.9536,0.6755,0.3905,0.1249,0.3629,0.6356,0.8116,0.7664,0.5417,0.
 ↪2614,0.1723,0.2814,0.2764,0.1985,0.1502,0.1219,0.0493,0.0027,0.0077,0.0026,0.
 ↪0031,0.0083,0.0020,0.0084,0.0108,0.0083,0.0033
)
#converting the input_data to a numpy array
input_data_to_numpy_array = np.asarray(input_data)

#reshape the np array as we are predicting for one instance
input_data_reshaped = input_data_to_numpy_array.reshape(1, -1)
#1, -1 == one instanecand we are going to predict it

prediction = model.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.
```

[ ]: