

# Rock Vs Mine Prediction

April 16, 2025

## 1 Rock Vs Mine prediction using the sonar data of a submarine

**Situation :** Suppose there is a war going on and the enemy country has placed mines under the sea and also there are rocks also and we are moving with a submarine. And you are given with the data taken in the lab containing the data of testing for rock and mine using different parameters

**Goal :** Submarine needs to find out the mines and our job is to predict whether it is a mine or a rock using a sonar

```
[1]: # Importing Libraries and required dependencies
import pandas as pd
import numpy as np
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
```

### Data Collection and Data Processing

```
[2]: # Importing/loading sonar data into pandas dataframe
sonar_data = pd.read_csv('dataset/sonar data.csv', header = None)
```

```
[3]: # Printing the first 5 rows
sonar_data.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]

Now the column no 60 shows the target i.e is it a rock or a mine Here : R represents Rock M represents Mine

```

[4]: # Number of rows and cols
sonar_data.shape

```

[4]: (208, 61)

```

[5]: sonar_data.describe() # describing some statistical measures of the data

```

```

[5]:
      count      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

      count      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

      count      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]

```
[6]: # Checking total no of Rocks and Mine examples
sonar_data[60].value_counts()
```

```
[6]: 60
M    111
R     97
Name: count, dtype: int64
```

The data is not so much imbaanced therefore i didnt cut or reduced the data

```
[7]: # Mean value for all the cols w.r.t Rock and Mine
sonar_data.groupby(60).mean()
```

```
[7]:
```

	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]

From this we can see that there is a difference between tne values of rock and mine and using this we can try to predict the object is either a mine or a rock

```
[8]: # Checking for any null values
sonar_data.isnull().sum()
```

	..
56	0
57	0
58	0
59	0
60	0

```
# Seprating the data based on feature and label
X = sonar_data.drop(columns=60,axis=1)
y = sonar_data[60]
```

```
# Checking if the data is seprated or not
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

```

[11]: # Splitting the data into training and test data
      # Stratify is splitting the data based on the number of R or M so the no of R
      ↪or M remains same in both data
      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.shape,X_train.shape,X_test.shape)

```

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

```

[13]: 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
```

### Training the Machine Learning Model ( Logistic Regression )

```
[14]: model = LogisticRegression()
```

```
[15]: # training the Logistic Regression Model With training data
      model.fit(X_train,y_train)
```

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

### Model Evaluation

```
[16]: # Accuracy on training data
      X_train_prediction = model.predict(X_train)
      training_data_accuracy = accuracy_score(X_train_prediction,y_train)
```

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

Accuracy on training data : 0.8342245989304813

```
[18]: # Accuracy on test data
      X_test_prediction = model.predict(X_test)
      test_data_accuracy = accuracy_score(X_test_prediction,y_test)
```

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

Accuracy on test data : 0.7619047619047619

### Making a Predicting System

```
[20]: input_data = (0.1313,0.2339,0.3059,0.4264,0.4010,0.1791,0.1853,0.0055,0.1929,0.
      ↪2231,0.2907,0.2259,0.3136,0.3302,0.3660,0.3956,0.4386,0.4670,0.5255,0.3735,0.
      ↪2243,0.1973,0.4337,0.6532,0.5070,0.2796,0.4163,0.5950,0.5242,0.4178,0.3714,0.
      ↪2375,0.0863,0.1437,0.2896,0.4577,0.3725,0.3372,0.3803,0.4181,0.3603,0.2711,0.
      ↪1653,0.1951,0.2811,0.2246,0.1921,0.1500,0.0665,0.0193,0.0156,0.0362,0.0210,0.
      ↪0154,0.0180,0.0013,0.0106,0.0127,0.0178,0.0231)
```

```
# changing the datatype to np arr
```

```
input_data_as_numpy_array = np.asarray(input_data)
```

```
# Reshaping the numpy arr as we are predicting for one instance
```

```
input_data_resaped = input_data_as_numpy_array.reshape(1,-1) # 1,-1 represent_
      ↪there is 1 instance
```

```
prediction = model.predict(input_data_resaped)
```

```
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