SONAR Rock vs Mine Prediction using Logistic Regression

The enemy countries have planted some mines (which become explosives when an object comes into contact with them) in the ocean. However, there will be some rocks present in the ocean. Finally, the task is to build a system to predict rock or mine using SONAR data.

Steps Followed

1. Collect sonar data
2. Data Preprocessing
3. Train Test Split
4. Logistic Regression Model (Binary Classification)
5. Predictions on old data (train and test)
6. Model Performance Results
7. Creating predictive system

In [1]:

```
import numpy as np
import pandas as pd

from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
```

1. Collect/Load Sonar Data

In [2]:

```
sonar_df = pd.read_csv("C:/Users/Mohankumar MC/Desktop/ML Projects/SONAR Rock  vs Mine P
                       header = None)
```

2. Data Preprocessing

In [3]:

```
sonar_df.head()
```

Out[3]:

	0	1	2	3	4	5	6	7	8	9	...	51
0	0.0200	0.0371	0.0428	0.0207	0.0954	0.0986	0.1539	0.1601	0.3109	0.2111	...	0.0027
1	0.0453	0.0523	0.0843	0.0689	0.1183	0.2583	0.2156	0.3481	0.3337	0.2872	...	0.0084
2	0.0262	0.0582	0.1099	0.1083	0.0974	0.2280	0.2431	0.3771	0.5598	0.6194	...	0.0232
3	0.0100	0.0171	0.0623	0.0205	0.0205	0.0368	0.1098	0.1276	0.0598	0.1264	...	0.0121
4	0.0762	0.0666	0.0481	0.0394	0.0590	0.0649	0.1209	0.2467	0.3564	0.4459	...	0.0031

5 rows × 61 columns

In [4]:

```
sonar_df.shape
```

Out[4]:

```
(208, 61)
```

In [5]:

```
sonar_df.describe()
```

Out[5]:

	0	1	2	3	4	5	6
count	208.000000	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	0.121747
std	0.022991	0.032960	0.038428	0.046528	0.055552	0.059105	0.061788
min	0.001500	0.000600	0.001500	0.005800	0.006700	0.010200	0.003300
25%	0.013350	0.016450	0.018950	0.024375	0.038050	0.067025	0.080900
50%	0.022800	0.030800	0.034300	0.044050	0.062500	0.092150	0.106950
75%	0.035550	0.047950	0.057950	0.064500	0.100275	0.134125	0.154000
max	0.137100	0.233900	0.305900	0.426400	0.401000	0.382300	0.372900

8 rows × 60 columns

In [6]:

```
sonar_df[60].value_counts()
```

Out[6]:

```
M    111
R     97
Name: 60, dtype: int64
```

In [7]:

```
sonar_df.groupby(60).mean()
```

Out[7]:

	0	1	2	3	4	5	6	7	8
60									
M	0.034989	0.045544	0.050720	0.064768	0.086715	0.111864	0.128359	0.149832	0.21349
R	0.022498	0.030303	0.035951	0.041447	0.062028	0.096224	0.114180	0.117596	0.13739

2 rows × 60 columns

3. Train Test Split

In [8]:

```
# Importing the required class
from sklearn.model_selection import train_test_split

# Specifying the columns as predictor and target variable
predictors = sonar_df.drop(columns=60,axis=1)
target = sonar_df[60]

# Spliting the data in training and test set in 90:10 ratio
X_train, X_test, Y_train, Y_test = train_test_split(predictors, target, test_size=0.1,
                                                    stratify=target,random_state=1)
```

In [9]:

```
print("Shape of X_train:", X_train.shape)
print("Shape of y_train:", Y_train.shape)
print("Shape of X_test:", X_test.shape)
print("Shape of y_test:", Y_test.shape)
```

```
Shape of X_train: (187, 60)
Shape of y_train: (187,)
Shape of X_test: (21, 60)
Shape of y_test: (21,)
```

4. Logistic Regression Model (Binary Classification)

In [10]:

```
# Importing the required class
from sklearn.linear_model import LogisticRegression

# Creating the object of the class LogisticRegression
model = LogisticRegression()

# Fitting the model to the training data
model.fit(X_train,Y_train)

# Getting the intercept and the coefficients of the model
print("Intercept:",model.intercept_,"\nCoefficients:", model.coef_)
```

```
Intercept: [2.94224506]
Coefficients: [[-0.21882473 -0.22934564 -0.19024714 -0.40843118 -0.320930
88 -0.12470474
   0.08731375 -0.01502358 -0.87357141 -1.07822789 -1.52394522 -1.44966936
  -0.73267048 -0.06497675  0.15220066  0.48883377  0.40835514  0.42737259
  -0.32304323 -0.65776606 -0.68501787 -0.50021218 -0.43324867 -0.3644048
   0.33952982  0.0533055  -0.13067547 -0.3489243  -0.35394463 -0.35069647
   0.85100622 -0.22878875 -0.11076488  0.14769486  0.54231878  1.28389545
   0.78820916 -0.23346495 -0.39136517  0.51960252  0.01620924 -0.66494894
  -0.87707093 -0.99885056 -1.61009839 -1.34113535 -0.77586003 -0.78351105
  -0.52640426 -0.03060355 -0.12948876 -0.10267112 -0.03449633 -0.06872122
  -0.01584387 -0.03740321 -0.00581692 -0.0652067  -0.05774097 -0.0240060
3]]
```

5. Predictions on old data (train and test)

In [11]:

```
X_train_prediction = model.predict(X_train)
X_train_prediction
```

Out[11]:

```
array(['M', 'R', 'M', 'M', 'R', 'M', 'R', 'R', 'R', 'R', 'M', 'M', 'R',
       'R', 'M', 'R', 'M', 'M', 'R', 'R', 'M', 'R', 'R', 'R', 'M', 'M',
       'R', 'R', 'R', 'M', 'M', 'R', 'M', 'R', 'M', 'R', 'R', 'M', 'M',
       'M', 'R', 'M', 'M', 'R', 'M', 'M', 'M', 'M', 'M', 'R', 'M', 'M',
       'M', 'R', 'M', 'M', 'R', 'M', 'R', 'M', 'R', 'R', 'M', 'R', 'M',
       'M', 'R', 'M', 'R', 'R', 'R', 'M', 'M', 'M', 'M', 'R', 'R', 'R',
       'R', 'M', 'M', 'R', 'R', 'M', 'R', 'M', 'M', 'M', 'M', 'M', 'M',
       'M', 'M', 'M', 'M', 'R', 'R', 'R', 'M', 'M', 'R', 'M', 'R', 'M',
       'M', 'R', 'R', 'R', 'M', 'M', 'R', 'M', 'R', 'R', 'M', 'M', 'R',
       'M', 'R', 'M', 'M', 'M', 'R', 'R', 'R', 'M', 'R', 'R', 'R', 'M',
       'M', 'M', 'M', 'M', 'M', 'M', 'R', 'R', 'M', 'M', 'M', 'M', 'R',
       'M', 'R', 'M', 'R', 'R', 'M', 'M', 'M', 'M', 'M', 'R', 'M', 'R',
       'R', 'M', 'R', 'M', 'M', 'R', 'M', 'R', 'R', 'M', 'R', 'M', 'R',
       'M', 'R', 'R', 'R', 'M', 'R', 'R', 'R', 'M', 'M', 'R', 'M', 'R',
       'M', 'R', 'M', 'M', 'M'], dtype=object)
```

In [12]:

```
X_test_prediction = model.predict(X_test)
X_test_prediction
```

Out[12]:

```
array(['M', 'R', 'R', 'M', 'M', 'M', 'M', 'M', 'R', 'M', 'R', 'M', 'R',
       'M', 'R', 'M', 'M', 'M', 'R', 'R', 'R'], dtype=object)
```

6. Model Performance Results

In [13]:

```
# Training Data Accuracy
training_data_accuracy = accuracy_score(X_train_prediction,Y_train)
training_data_accuracy
```

Out[13]:

```
0.8342245989304813
```

In [14]:

```
# Testing Data Accuracy
training_data_accuracy = accuracy_score(X_test_prediction,Y_test)
training_data_accuracy
```

Out[14]:

```
0.7619047619047619
```

7. Creating predictive system

In [15]:

```
# Creating predictive system

input_data = (0.0453,0.0523,0.0843,0.0689,0.1183,0.2583,0.2156,0.3481,0.3337,0.2872,0.49

# convert an list to an array
input_data_as_numpy_array = np.asarray(input_data)

# reshape the numpy array as we are predicting only one instance
input_data_reshape = input_data_as_numpy_array.reshape(1,-1)

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

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

```
['R']
Object is a Rock
```

In [16]:

```
input_data = (0.0522,0.0437,0.0180,0.0292,0.0351,0.1171,0.1257,0.1178,0.1258,0.2529,0.27
input_data_as_numpy_array = np.asarray(input_data)

# reshape the numpy array as we are predicting only one instance
input_data_reshape = input_data_as_numpy_array.reshape(1,-1)

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

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

```
['M']
Object is a Mine
```