Logistic Regression

Debatreya Das
Roll No. 12212070
ML Lab CS A4 (20th Aug)

Importing the required libraries

In [1]:
```
import pandas as pd
import numpy as np
```

Loading the data frames

In [2]:
```
df = pd.read_csv('./bank/bank.csv', sep=';')
df
```

Out[2]:

	age	job	marital	education	default	balance	housing	loan	contact	day	month	duration	campaign	pdays	previous	poutcome	y
0	30	unemployed	married	primary	no	1787	no	no	cellular	19	oct	79	1	-1	0	unknown	no
1	33	services	married	secondary	no	4789	yes	yes	cellular	11	may	220	1	339	4	failure	no
2	35	management	single	tertiary	no	1350	yes	no	cellular	16	apr	185	1	330	1	failure	no
3	30	management	married	tertiary	no	1476	yes	yes	unknown	3	jun	199	4	-1	0	unknown	no
4	59	blue-collar	married	secondary	no	0	yes	no	unknown	5	may	226	1	-1	0	unknown	no
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
4516	33	services	married	secondary	no	-333	yes	no	cellular	30	jul	329	5	-1	0	unknown	no
4517	57	self-employed	married	tertiary	yes	-3313	yes	yes	unknown	9	may	153	1	-1	0	unknown	no
4518	57	technician	married	secondary	no	295	no	no	cellular	19	aug	151	11	-1	0	unknown	no
4519	28	blue-collar	married	secondary	no	1137	no	no	cellular	6	feb	129	4	211	3	other	no
4520	44	entrepreneur	single	tertiary	no	1136	yes	yes	cellular	3	apr	345	2	249	7	other	no

4521 rows × 17 columns

One-Hot Encoding (of categorical data)

One-Hot Encoding is a method of representing characters or words by a vector where only one element is set to one and all others are zero, based on their position in the vocabulary. This results in a sparse, semantically independent vector with a high dimension.

In [3]:
```
data = pd.get_dummies(df, drop_first=True)
data
```

Out[3]:

	age	balance	day	duration	campaign	pdays	previous	job_blue-collar	job_entrepreneur	job_housemaid	...	month_jun	month_mar	month_may	month_nov	month_oct	month_sep	poutcome_other	poutcome_success	poutcome_unknown	y_yes
0	30	1787	19	79	1	-1	0	False	False	False	...	False	False	False	False	True	False	False	False	True	False
1	33	4789	11	220	1	339	4	False	False	False	...	False	False	True	False	False	False	False	False	False	False
2	35	1350	16	185	1	330	1	False	False	False	...	False	False	False	False	False	False	False	False	False	False
3	30	1476	3	199	4	-1	0	False	False	False	...	True	False	False	False	False	False	False	False	True	False
4	59	0	5	226	1	-1	0	True	False	False	...	False	False	True	False	False	False	False	False	True	False
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
4516	33	-333	30	329	5	-1	0	False	False	False	...	False	False	False	False	False	False	False	False	True	False
4517	57	-3313	9	153	1	-1	0	False	False	False	...	False	False	True	False	False	False	False	False	True	False
4518	57	295	19	151	11	-1	0	False	False	False	...	False	False	False	False	False	False	False	False	True	False
4519	28	1137	6	129	4	211	3	True	False	False	...	False	False	False	False	False	False	True	False	False	False
4520	44	1136	3	345	2	249	7	False	True	False	...	False	False	False	False	False	False	True	False	False	False

4521 rows × 43 columns

Separate Dependent Varables (Y) and Independent Variables (X)

In [4]:
```
X = data.iloc[:, :-1].values  # 'y_yes' is the encoded target column after one-hot encoding
Y = data.iloc[:, -1].values
X = np.array(X,dtype=int)
```

```
print(X)
print(Y)
```

```
[[  30 1787   19 ...    0    0    1]
 [  33 4789   11 ...    0    0    0]
 [  35 1350   16 ...    0    0    0]
 ...
 [  57  295   19 ...    0    0    1]
 [  28 1137    6 ...    1    0    0]
 [  44 1136    3 ...    1    0    0]]
[False False False ... False False False]
```

Normalize / Standardise features (calculatind z-score of normal distribution)

$$z = (X - \mu)/\sigma$$

In [5]:
```
X_mean = np.mean(X, axis=0)
X_std = np.std(X, axis=0)

X_std[X_std == 0] = 1

x = (X - X_mean) / X_std

x
```

Out[5]:
```
array([[-1.05626965,  0.12107186,  0.37405206, ..., -0.21344711,
        -0.1713814 ,  0.46930045],
       [-0.77258281,  1.1186443 , -0.59602646, ..., -0.21344711,
        -0.1713814 , -2.1308311 ],
       [-0.58345826, -0.02414438,  0.01027262, ..., -0.21344711,
        -0.1713814 , -2.1308311 ],
       ...,
       [ 1.49691189, -0.37472364,  0.37405206, ..., -0.21344711,
        -0.1713814 ,  0.46930045],
       [-1.24539421, -0.09492484, -1.20232553, ...,  4.68500145,
        -0.1713814 , -2.1308311 ],
       [ 0.26760226, -0.09525714, -1.56610497, ...,  4.68500145,
        -0.1713814 , -2.1308311 ]])
```

Logistc Regression Model

In [21]:
```
class LogisticRegressionManual:
    def __init__(self, learning_rate=0.001, iterations=1000):
        self.learning_rate = learning_rate
        self.iterations = iterations

    def sigmoid(self, z):
        z = np.clip(z, -500, 500)  # Clip to avoid overflow
        return 1 / (1 + np.exp(-z))

    def fit(self, X, y):
        self.m, self.n = X.shape
        self.W = np.zeros(self.n)
        self.b = 0
        self.X = X
        self.y = y

        for i in range(self.iterations):
            # Linear model
            z = np.dot(self.X, self.W) + self.b
            # Sigmoid function
            y_pred = self.sigmoid(z)

            # Gradient descent
            dW = (1/self.m) * np.dot(self.X.T, (y_pred - self.y))
            db = (1/self.m) * np.sum(y_pred - self.y)

            # Update weights
            self.W -= self.learning_rate * dW
            self.b -= self.learning_rate * db


    def predict(self, X):
        z = np.dot(X, self.W) + self.b
        y_pred = self.sigmoid(z)
        y_pred_class = [1 if i > 0.5 else 0 for i in y_pred]
        return y_pred_class
```

Splitting the Dataset and Training Model

In [23]:
```
split_ratio = 0.7
split_index = int(split_ratio * len(X))

X_train, X_test = X[:split_index], X[split_index:]
```

```
y_train, y_test = Y[:split_index], Y[split_index:]

model = LogisticRegressionManual(learning_rate=0.001, iterations=1000)
model.fit(X_train, y_train)
```

Make Predictions

In [24]:
```
y_pred = model.predict(X_test)
```

Printing some predictions

In [25]:
```
print("\nPredicted vs Actual:")
print("Predicted:", y_pred[:10])
print("Actual:   ", y_test[:10])
```

```
Predicted vs Actual:
Predicted: [0, 0, 0, 0, 0, 1, 0, 0, 0, 0]
Actual:    [False False False False False False False False False False]
```

Check accuracy (Evaluate Model)

In [26]:
```
accuracy = np.sum(y_pred == y_test) / len(y_test)
print(f"\nModel accuracy: {accuracy * 100:.2f}%")
```

```
Model accuracy: 84.60%
```