Assignment No 2

The goal of the assignment is to select, implement and evaluate the machine learning algorithm from scratch and compare it against a reference implementation from an open-source machine learning package. To accomplish the above goal, I have selected and implemented a Logistic Regression algorithm. Compared it with reference implementation from an open-source machine learning package and have observed the difference between the results.

Logistic Regression :

Logistic regression is a predictive analysis that is used to describe data and to explain the relationship between one dependent binary variable and one or more independent variables. It is another statistical model of a machine learning algorithm that uses the "logistic function" to model a dependant variable. It produces results in a binary format which is used to predict the outcome of a categorical dependent variable. So, outcome should be discrete/ categorical such as yes/ no, true/false, 0/1 , high/low.

Reason for using Logistic Regression over the Linear Regression :

The **first** problem with linear Regression is that if we add an outlier in our dataset, the best fit line in linear regression completely gets deviated and shifts to fit that point. It produces a high error rate. Hence, we can say that linear regression is prone to outliers.

The **Second** problem the predicted values may be out of range. To overcome these problems we use Logistic Regression, which converts this straight best fit line in linear regression to an S-curve using the sigmoid function, which will always give values between 0 and 1.

To understand the how logistic Regression model produces the output between 0 and 1, we must go through with little maths and formula.

- **Logistic Function** :

1. The equation of the best fit line is **y = mx + c,** we can denote this as y = b0 + b1x

$$y = \beta_0 + \beta_1 x$$

2. Here, we will calculate the probabilities hence considering y as P

$$P = \beta_0 + \beta_1 x$$

3. The value of probabilities will exceed 1 or will go down below 0 but to produce logistic regression model output, we have restrictions to the probabilities values those values should be in the range of 0 to 1 . To achieve this, taking odds of P is the easiest way hence we take **odds of P.**
4. Odds are nothing but the ratio of the probability of success and probability of failure. The range of odds can always be positive (0,+ ∞).

$$\frac{P}{1-P} = \beta_0 + \beta_1 x$$

5. By restricting the range, we are decreasing the number of data points and if we decrease our data points eventually our correlation will decrease. To prevent this, we take the **logs of odds** which provide the range from (-∞,+∞).

$$\log\left(\frac{P}{1-P}\right) = \beta_0 + \beta_1 x$$

6. To predict the probability P, we must multiply by exponent on both sides to produce a Sigmoid Function that is Logistic Function.

$$\exp[\log(\frac{p}{1-p})] = \exp(\beta_0 + \beta_1 x)$$

7. **Sigmoid Function**: we can define it as below as well:

$$p = \frac{1}{1+e^{-(\beta_0+\beta_1 x)}}$$

$$\hat{Y} = \frac{1}{1+e^{-Z}} \qquad Z = w.X + b$$

Y_hat --> predicted value
X --> Input Variable
w --> weight
b --> bias

8. **Decision Boundary:** The sigmoid function will produce the probability. To predict the class of class a data belongs a threshold can be set. Based upon this threshold, the obtained estimated probability is classified into classes. It decides the threshold value.
   if predicted value ≥ 0.5, then set it as 1
   predicted values ⩽ 0.5 then set it as 0
9. **Cost Function:** The cost function represents optimization objective i.e. We create a cost function and minimize it so that we can develop an accurate model with less error. In logistic Regression, it would end up being a **non-convex** function with many local minimums, in which it would be very **difficult** to **minimize the cost value** and find the global minimum. Hence, to reduce the cost in Logistics I have used Gradient Descent.
10. **Gradient Descent**: It is an optimization algorithm often used in machine learning applications to find the model parameters that correspond to the best fit between predicted and actual outputs. The main goal of Gradient descent is to **minimize the cost value**.

Data Preprocessing and Data Splitting :

1. Imported all libraries of pandas, sklearn, numpy and read the dataset csv file using pandas read()
2. Loaded the dataset into dataset variable. The dataset has 204 rows and total 10 columns(feature)
3. The co relation of month, day and year column is very less with fire predications hence dropped these 3 columns from the dataset.
4. Split the data into 2/3 for training and 1/3 for testing. Train_data has 136 rows 7 columns, test_data 68 rows, 7 columns. From test and train data copied the label column into the train and test label and dropped from the same column from train _data and test_data. The Length of train_label and test_label is 136 rows ,1 column and 68 rows 1 column respectively.

Build a Logistic regression model using Sklearn's predefined alogorithm:

1. Trained the model by sklearn's inbuild logistic regression classifier. Predicted the probability by passing test_data and calculated the accuracy that is 85.29.
2. Repeated the same steps 10 times in for loop with different random divisions and got an accuracy average is 78.97%

Implementation of Logistic Regression Algorithm from scratch in Python (Design Decisions) :

1. Created a class named Logistic Regression and initiated two hyperparameters for this logistic Regression in init method.The two parameters are learning rate and number of iterations.

   *Learning rate = It is the tuning parameter in an optimization algorithm. While moving toward minimum loss of a function it determines the step size at each iteration.

   *Number of iterations = If we pass no_of_iterations = 1000 then each time the model will try to change its parameters. Parameters are weight(w) and bias (b) of the model.

```
class Logistic_Regression():
    '''Declaring Hyperparameters which is learning rate and number of iterations'''
    def __init__(self, learning_rate,no_of_iterations):
        self.learning_rate = learning_rate
        self.no_of_iterations = no_of_iterations
```

2. To train the model, I have created a fit function with 2 arguments X and Y. The X value will be features (train/ test data) and Y value will be target column(train/ test labels). By using the .shape function stored the rows and columns of the dataset in m and n variables respectively.
3. Initiated weight and bias value of the model with 0.

```
 def fit(self,X,Y):
     '''Fit Function to train the model with dataset'''
     self.m, self.n = X.shape

     '''Initaiting weight and bias value'''
     self.w = np.zeros(self.n)
     self.b = 0
     self.X = X
     self.Y = Y

     '''Gradient Descent to find best fit between predicted and actual outputs'''
     for i in range(self.no_of_iterations):
         self.update_weights()
```

4. Implemented Gradient descent algorithm is used to compute the minimum cost. For this, I have created a for loop and passing argument as a number of iterations. Based on the number of iterations model will go through again and again for that specified number and will try to update the weights and bias to check which weight and bias value is most optimum one and inside the for loop called the update weights method.

```
def update_weights(self):
    Y_cap= 1/(1+np.exp(-(self.X.dot(self.w)+self.b)))

    '''Derivatives'''
    dw = (1/self.m)*np.dot(self.X.T,(Y_cap - self.Y))
    db = (1/self.m)*np.sum(Y_cap - self.Y)

    '''Using Gradient Descent updating the weight and bias values'''
    self.w = self.w - self.learning_rate * dw
    self.b = self.b - self.learning_rate * db
```

5. To update the parameter of the model created a function update weights. Defined derivatives inside the function.

   Y_cap= 1/(1+np.exp(-(self.X.dot(self.w)+self.b)))

6. In order to implment Gradient decent I have derived below formula of derivatives dw and db dw = 1/m * (Y_cap – Y).X [X = train/test data, Y = train/test label]

   db = 1/m * (Y_cap – Y)
   w = w - α*dw
   b = b - α*db

```
def predict(self,X):
    '''Sigmoid equation (Y_cap) and setting a threshold'''
    Y_pred = 1/ (1+np.exp(- (X.dot(self.w) + self.b)))
    Y_pred = np.where( Y_pred > 0.5,1,0)
    self.y_pred = Y_pred
    return Y_pred
```

7. The for loop will run with all these values and will find the optimum value for weight and bias.
8. Created the predict function to predict the fire based on feature values like temp, humidity, rainfall, and 3 others. Here, to find the probability, Y values(train/test label) should be equal to one or zero for that I have defined the sigmoid function. This function will produce a Y_cap value that will lie between 0 and 1 like the normal probability values lie between 0 and 1. In the predict function, it is finding the Y_cap value. If the Y_cap value is greater than 0.5 then it will consider it 1 and if the Y_cap value is below 0 it will consider as 0. These are labels for the predictions.
9. Fit the model by calling the developed logistic classifier. Passed 1000 as no of iterations and learning rate as 0.01
10. Repeated the same steps 10 times in for loop with different random divisions and got an accuracy average is 83.38%

Comparison of Sklearn's model and developed scratch model:

1. Developed and tested Logistic Regression model in both ways, by sklearns classifier, and by own scratch function.
2. Evaluated different accuracies. As in my own developed model, I have explicitly defined the gradient descent, sigmoid function and have tested locally with one type of dataset.
3. The own model took a long time around 2min 30 sec to train the model whereas the sklearn model was executed within 15 seconds.

Observations:

1. Observed the execution time difference between both models.
2. For better results, have to try to run the same model with different types of datasets.

Conclusion:

In this report, I have represented the implementation of Logistic Regression using sklearns classifier and using developed own classifier. Based on the evaluated accuracy, average accuracy, execution time and comparison can be defined as own developed model requires improvisation to achieve better results.

Machine Learning Assignment 2

The goal of the assignment is to select, implement and evaluate the machine learning algorithm from scratch and compare it against a reference implementation from an open-source machine learning package. To accomplish the above goal I have selected and implemented a Logistic Regression algorithm. Compared it with reference implementation from an open-source machine learning package and have observed the difference between the results.

Logistic Regression :

Logistic regression is a predictive analysis that is used to describe data and to explain the relationship between one dependent binary variable and one or more independent variables. It is another statistical model of a machine learning algorithm that uses the "logistic function" to model a dependant variable. It produces results in a binary format which is used to predict the outcome of a categorical dependent variable. So outcome should be discrete/ categorical such as yes/ no, true/false, 0/1 , high/low. Logistic Regression is used in various fields of machine learning, social science, and healthcare field.

In [1]:

```
# Importing necessary libraries
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.metrics import accuracy_score, confusion_matrix, f1_score
```

Data Collection and Analysis

Read the wildfire csv files using pandas pd.read function and storing entire file into dataset variable for future preprocessing and analysing.

In [2]:

```
dataset = pd.read_csv('WildFires.csv',sep = '\t',header=0)
dataset = dataset.drop(["year", "month", "day"], axis=1)
dataset
```

Out[2]:

	yes	temp	humidity	rainfall	drought_code	buildup_index	wind_speed
0	no	28	59	0.0	8.06	3.47	19
1	no	30	61	1.3	8.17	4.03	13
2	no	26	83	13.1	8.08	3.59	22
3	no	25	87	2.5	7.18	2.42	15
4	no	28	77	0.0	14.98	4.63	18
...	...	...	...	...	...	...	...
199	yes	31	67	0.0	45.15	17.89	15
200	no	29	89	4.4	8.74	6.52	15
201	no	27	88	0.5	8.87	3.71	30
202	no	25	56	0.1	15.54	6.10	20
203	no	24	62	0.2	16.72	5.75	17

204 rows × 7 columns

Data Analysis steps:

1. To understand the trainDataset in terms of column and rows
2. Getting the statistical measure of data(mean,SD,min,max,count)
3. Understand the how many values are there in the target variable.

In [3]:

```
dataset.shape
```

Out[3]:

```
(204, 7)
```

In [4]:

```
1 dataset.describe()
```

Out[4]:

	temp	humidity	rainfall	drought_code	buildup_index	wind_speed
count	204.000000	204.000000	204.000000	204.000000	204.000000	204.000000
mean	31.906863	62.279412	0.823529	48.537647	16.542304	16.446078
std	3.814175	15.209388	2.117959	49.133366	14.634994	3.098074
min	22.000000	21.000000	0.000000	7.180000	1.320000	6.000000
25%	29.000000	52.750000	0.000000	10.627500	6.067500	14.750000
50%	32.000000	63.000000	0.000000	30.550000	11.535000	16.000000
75%	35.000000	74.250000	0.600000	62.367500	22.665000	18.000000
max	43.000000	92.000000	16.800000	221.350000	68.270000	30.000000

In [5]:

```
1 dataset['yes'].value_counts()
```

Out[5]:

```
yes        101
no          92
yes          4
yes          2
no           2
no           1
no           1
no           1
Name: yes, dtype: int64
```

Data Preprocessing :

1. Dataset has 7 columns out of which the very first column "yes" is the label column.
2. Stored 1st column of the dataset in train_label variable as it is a target variable of the dataset.
3. The column 'yes'(fire) has 7 unique values such as ['no ', 'yes ', 'yes', 'yes ', 'no', 'no ', 'no '] , to remove extra white space from stat and end of each variable, performed .strip function.
4. The column 'yes' has string values and the machine learning algorithm cannot process the string values hence to manipulate the data used get_dummies function. It converts the categorical data into indicator variables.
5. Split the data into train and test data with sizes of 136 and 68 respectively.
6. To standardize the data use the scalar function.

In [6]:

```
def preprocess(dataset, labelName):
    dataset[labelName] = dataset[labelName].str.strip()
    dataset[labelName] = pd.get_dummies(dataset[labelName])[labelName]
    return dataset

def read_prepare_data(file_name):
    dataset = pd.read_csv(file_name,sep = '\t',header=0)
    dataset = dataset.drop(["year", "month", "day"], axis=1)
    train_data, test_data =  train_test_split(dataset, test_size=0.33)
    train_data.shape, test_data.shape

    train_data = preprocess(train_data, "yes")
    test_data  = preprocess(test_data, "yes")

    train_label = train_data["yes"]
    train_data  = train_data.drop("yes", axis=1)

    test_label = test_data["yes"]
    test_data  = test_data.drop("yes", axis=1)


    #Data Standardization
    from sklearn.preprocessing import StandardScaler

    scalar = StandardScaler()

    scalar.fit(train_data)
    train_data = scalar.transform(train_data)
    test_data  = scalar.transform(test_data)

    return train_data, test_data, train_label, test_label
```

Evaluation metrics for both models

Created an acc_list to store the accuracies and to calculate the average of accuracies. The evaluation metrics method simply calculate and will print the accuracy, classification report, and confusion matrix

In [7]:

```
acc_list = []
def evaluation_metrics(predicted_values, actual_values):
    acc = accuracy_score(predicted_values, actual_values)
    acc_list.append(acc)
    print("Accuracy of the Model :" , round(acc*100,2))
    print('')
    print("Classification Report of Logistic Regression Model :")
    print(classification_report(predicted_values, actual_values))
    print("Confusion Matrix :")
    print(confusion_matrix(predicted_values, actual_values))
    print("\n\n")
    return (acc, f1_score(predicted_values, actual_values))
```

Predefined SKLearn Model

Training the model and predicting the accuracy of the model using the existing sklearn Logistic Regression algorithm. Evaluate average accuracy of the model with 10 different divisions using for loop.

In [8]:

```
import time
start = time.time()

for _ in range(10):
    train_data, test_data, train_label, test_label = read_prepare_data('WildFires.csv')
    logModel = LogisticRegression()
    logModel.fit(train_data, train_label)
    predefined_predictions = logModel.predict(test_data)
    evaluation_metrics(predefined_predictions, test_label)

end = time.time()
print("Total Execution Time : ",end - start, "seconds")
```

```
Accuracy of the Model : 91.18

Classification Report of Logistic Regression Model :
              precision    recall  f1-score   support

           0       0.94      0.88      0.91        34
           1       0.89      0.94      0.91        34

    accuracy                           0.91        68
   macro avg       0.91      0.91      0.91        68
weighted avg       0.91      0.91      0.91        68

Confusion Matrix :
[[30  4]
 [ 2 32]]



Accuracy of the Model : 83.82
```

Implementation of Logistic Regression from scratch

Logistic Regression :

Sigmoid Funtion defined as :

$$\hat{Y} = \frac{1}{1 + e^{-Z}} \qquad Z = w.X + b$$

Y_cap --> predicted value

X --> Input Variable

w --> weight

b --> bias

Gradient Descent : It is used for updating the parameters of the learning model.

$w = w - \alpha * dw$

$b = b - \alpha * db$

Derivatives:

$dw = 1/m * (Y\_cap - Y).X$

$db = 1/m * (Y\_cap - Y)$

Setting a threshold value is 0.5 if sigmoid function value produces results more than 0.5 then it will set it as 1 and if it is less than 0.5 it will set it as 0

In [9]:

```
class Logistic_Regression_FromScratch():
    '''Declaring Hyperparameters which is learning rate and number of iterations'''
    def __init__(self, learning_rate,no_of_iterations):
        self.learning_rate = learning_rate
        self.no_of_iterations = no_of_iterations

    def fit(self,X,Y):
        '''Fit Function to train the model with dataset'''
        self.m, self.n = X.shape

        '''Initaiting weight and bias value'''
        self.w = np.zeros(self.n)
        self.b = 0
        self.X = X
        self.Y = Y

        '''Gradient Descent to find best fit between predicted and actual outputs'''
        for i in range(self.no_of_iterations):
            self.update_weights()

    def update_weights(self):
        Y_cap= 1/(1+np.exp(-(self.X.dot(self.w)+self.b)))

        '''Derivatives'''
        dw = (1/self.m)*np.dot(self.X.T,(Y_cap - self.Y))
        db = (1/self.m)*np.sum(Y_cap - self.Y)

        '''Using Gradient Descent updating the weight and bias values'''
        self.w = self.w - self.learning_rate * dw
        self.b = self.b - self.learning_rate * db

    def predict(self,X):
        '''Sigmoid equation (Y_cap) and setting a threshold'''
        Y_pred = 1/ (1+np.exp(- (X.dot(self.w) + self.b)))
        Y_pred = np.where( Y_pred > 0.5,1,0)
        self.y_pred = Y_pred
        return Y_pred

import time
start = time.time()

for _ in range(10):
    train_data, test_data, train_label, test_label = read_prepare_data('WildFires.csv')
    classifier = Logistic_Regression_FromScratch(learning_rate= 0.01, no_of_iterations
    classifier.fit(train_data, train_label)
    B_predictions = classifier.predict(test_data)
    evaluation_metrics(B_predictions, test_label)

end = time.time()
print("Total Execution Time : ",end - start, "seconds")
```

```
Accuracy of the Model : 86.76

Classification Report of Logistic Regression Model :
              precision    recall  f1-score   support

           0       0.97      0.80      0.88        40
           1       0.77      0.96      0.86        28
```

```
    accuracy                           0.87        68
   macro avg       0.87      0.88      0.87        68
weighted avg       0.89      0.87      0.87        68

Confusion Matrix :
[[32  8]
 [ 1 27]]


Accuracy of the Model : 82.35
```

Average accuracy

In [10]:

```
1 Avg_Acc_sklearn_classifier=round((sum(acc_list[0:9])/10)*100,2)
2 print("Avg_Acc_of_sklearn_classifier is : ", Avg_Acc_sklearn_classifier)
3 Avg_Acc_logistic_classifierfromscratch=round((sum(acc_list[10:20])/10)*100,2)
4 print("Avg_Acc_if_logistic_classifierfromscratch is : ", Avg_Acc_logistic_classifierfr
```

```
Avg_Acc_of_sklearn_classifier is :  78.97
Avg_Acc_if_logistic_classifierfromscratch is :  83.38
```