CYBERLABS

WINTER OF CODE 5.0

MACHINE LEARNING ALGORITHM LIBRARY

Malakji Ozair Sarfraz

22JE0535

LINEAR REGRESSION

Linear Regression is a type of Supervised Learning Algorithm. In this we hypothesis a linear equation to fit the given data, To achieve the best rate we use the method of gradient descent.

In this regression I have implemented the gradient descent function for finding the best parameters.

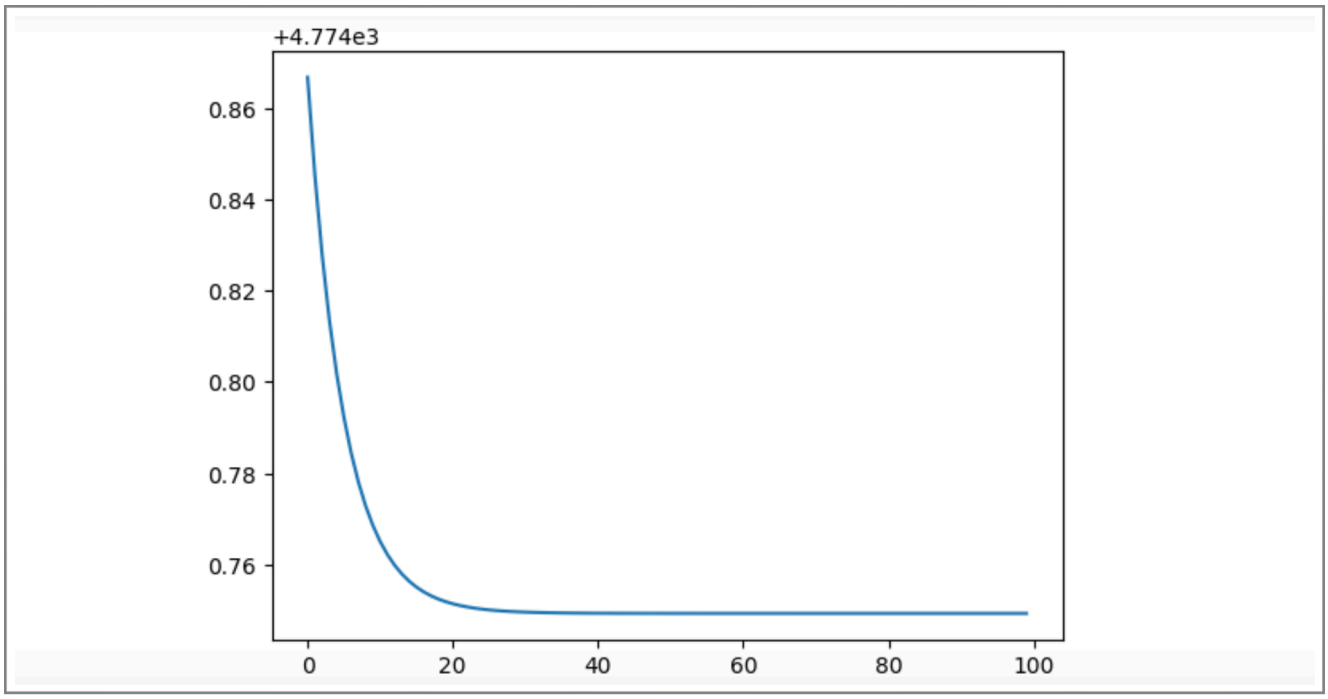
I have used the **R2 score** as metric to judge the performance of my model, and successfully achieved **0.8417828704601524** as its value.

```
In [20]: #learning rate=0.001
         #no of iterations=1000
         wnew3,bnew3=plottingcost(X,Y,w,b,.001,1000)
         y_predicted3=(np.matmul(wnew3,X_test.T)).T+bnew3
         R2score(y_predicted3,Y_test)

         4774.749201291785

Out[20]: 0.8417828704601524
```

The following is the graph plotted between **cost obtained vs number of iterations**.



POLYNOMIAL REGRESSION

Polynomial Regression is type of Supervised learning Regression. In this a polynomial is hypothesised to fit the training data to the best possible fit, here I have used a polynomial of degree 5 in an attempt to fit the data.

I have also applied feature scaling to reduce the variation of the training data set.Here also a similar Gradient descent function is used to get the best fitting polynomial.

Here also the **R2 metric** is used to measure the accuracy of the model, and obtained an R2 score of **0.9999996850326017**

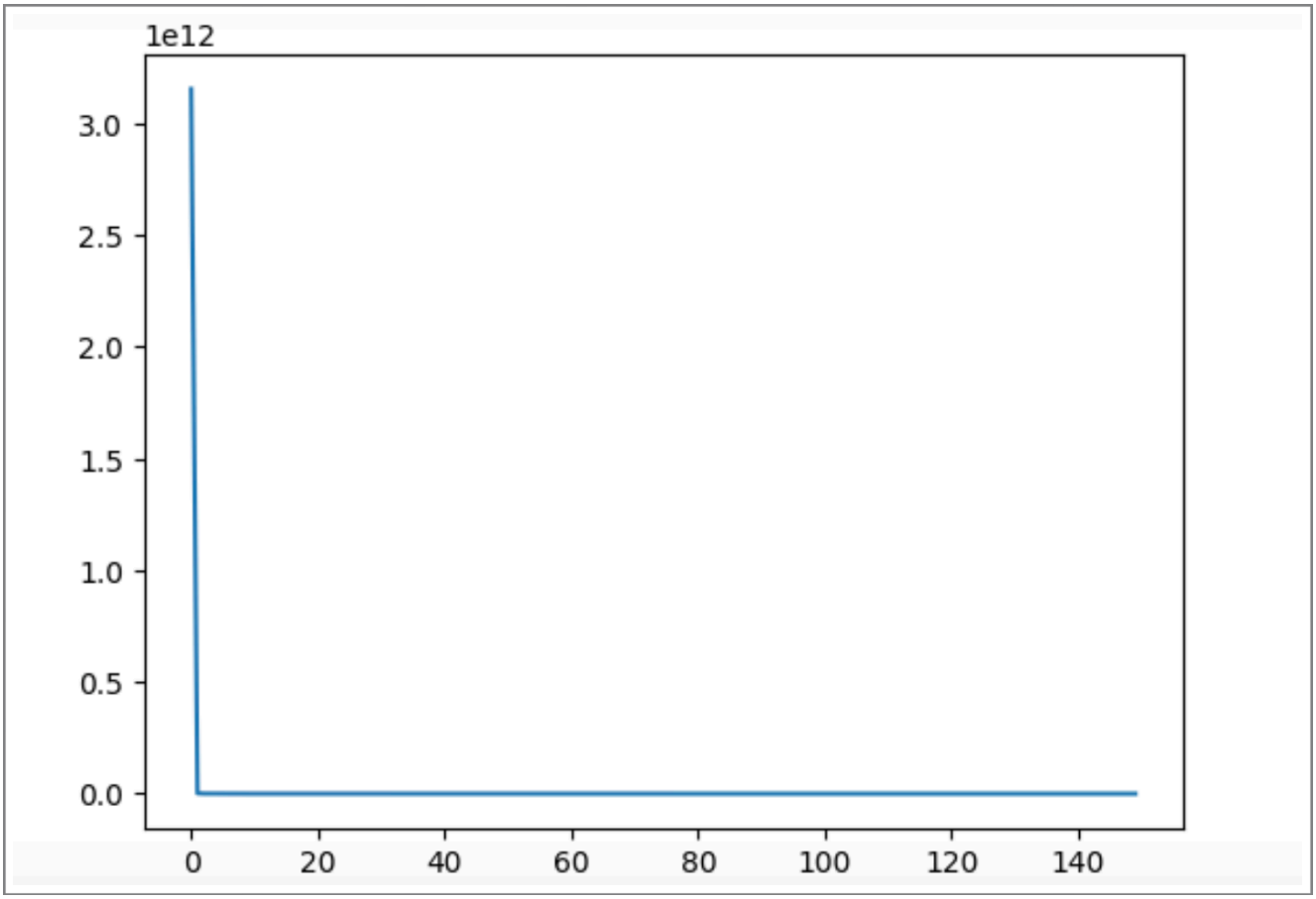
In [12]:
```
#testing the optimised parameters using test data onthe basis of R2score
x_test=(x_test-mean)/std
x_test=adding_features(x_test,5)
y_predicted=(np.matmul(wnw,x_test.T)).T+bnw
print(R2score(y_predicted,y_test))
```

```
0.9999996850326017
```

I have also an additional feature where we can initialise a polynomial of n degrees

```
In [4]: def adding_features(x,n):
            x_a=x[:,0]
            x_b=x[:,1]
            x_c=x[:,2]
            for k in range (n-1):
                for i in range (k+3):
                    for j in range (k+3):
                        if i+j>k+2:
                            continue
                        x=np.append((x_a.reshape(-1,1)**i)*(x_b.reshape(-1,1)**j)*(x_c.reshape(-1,1)*
            return x
        x=adding_features(x,5)
```

I have also plotted the graph between the cost and number of iterations similar to that in the Linear Regression.



LOGISTIC REGRESSION

In this we a regression technique to classify images. It uses the Sigmoid function for the classification. Output of the sigmoid corresponds to the probability, Logistic Regression finds a **linear decision boundary** to classify the data into two classes.

Here we used this binary Classification technique for **multiple classification** using **One Hot Encoding.**

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
#loading data
df=pd.read_csv('classification_train.csv')
array=df.to_numpy()
x=array[:24000,2:]
y=array[:24000,1]
x_test=array[24000:,2:]
y_test=array[24000:,1]
y_logistic=np.zeros((24000,10))
for i in range (24000):
    y_logistic[i,y[i]]=1
```

Here the metric is a **self defined** accuracy function and obtained an accuracy of **77.18333333333334%**.

In [15]:
```
Hits(y_final,y_test)
```

```
6000
```

Out[15]:
```
(77.18333333333334, 4631)
```

K-NEAREST NEIGHBOUR CLASSIFICATION

It is a **Supervised Classification Model,** which predicts the class of a given data point based on the classes of it's K nearest neighbour's classes.

I have used the **Euclidian distance** to figure the distances of the nearest neighbours.

```
In [2]: #no training is reguired
        def distance_function(x1,x2):
            distance=np.sum(np.square(x2-x1))
            return distance
```

In this I have used a **self defined** function to obtain the **accuracy** and have obtained an accuracy of **79.46%**.

```
In [56]: def Hits(y_predicted,y):
             shape=y.shape
             m=shape[0]
             print(m)
             count=0
             for i in range (m):
                 if y_predicted[i]==y[i]:
                     count+=1
             accuracy=(count/m)*100
             return accuracy,count
```

```
In [57]: Hits(y_predicted,y_test)

         5000

Out[57]: (79.46, 3973)
```

N-NEURAL NETWORK

It is a model that tries to mimic human brain by having **multiple layers of neurons** which are interconnected with one another to provide the best decision boundary

It overcomes the limitation of Logistic Regression that it can only form linear decision boundary. Data requiring any kind of decision boundary can be classified using the Neural Network

Here I have used the activation functions **ReLU(Rectified Linear Equation)** or **tanh** for initial layers. For the end layer I have used the **Softmax function** as the Activation Function. In neural Network we use **back Propagation** to train it.

In [6]:
```
def RelU_(z):
    relu=np.maximum(0,z)
    return relu
RelU=np.vectorize(RelU_)
```

In [7]:
```
def Tanh_function(z):
    return np.tanh(z)
def Tanh_function_d(z):
    return 1-(np.tanh(z))**2
```

In [14]:
```
def back_prop(x,y,w,b,activation):
    m=x.shape[0]
    l=len(w)
    a=forward_prop(x,w,b,activation)
    dw={}
    db={}
    dc_z=(1/m)*(a[l]-y)
    for i in range (l):
        dw[l-i]=np.matmul(dc_z.T,a[l-i-1])
        db[l-i]=np.sum(dc_z,axis=0)
        if l-i-2<0:
            break
        if activation=="tanh":
            dc_z=np.matmul(dc_z,w[l-i])*Tanh_function_d(z_function(a[l-i-2],w[l-i-1],b[l-i-1]
        elif activation=="relu":
            dc_z=np.matmul(dc_z,w[l-i])*RelU_d(z_function(a[l-i-2],w[l-i-1],b[l-i-1]))


    return dw,db
```

I have used this model for the classification of the image data and achieved an accuracy of **75.625%** by a self defined accuracy function.

In [20]:
```
w_new1,b_new1=update_parameters(x,y_logistic,w_new,b_new,0.01,500,"relu")
```

```
accuracy after 0 iterations is (74.35416666666667, 17845)
accuracy after 30 iterations is (74.425, 17862)
accuracy after 60 iterations is (74.49583333333332, 17879)
accuracy after 90 iterations is (74.58749999999999, 17901)
accuracy after 120 iterations is (74.72083333333333, 17933)
accuracy after 150 iterations is (74.8, 17952)
accuracy after 180 iterations is (74.91666666666667, 17980)
accuracy after 210 iterations is (74.95833333333334, 17990)
accuracy after 240 iterations is (75.00833333333333, 18002)
accuracy after 270 iterations is (75.09583333333333, 18023)
accuracy after 300 iterations is (75.17083333333333, 18041)
accuracy after 330 iterations is (75.24583333333334, 18059)
accuracy after 360 iterations is (75.32083333333334, 18077)
accuracy after 390 iterations is (75.36666666666667, 18088)
accuracy after 420 iterations is (75.47083333333333, 18113)
accuracy after 450 iterations is (75.575, 18138)
accuracy after 480 iterations is (75.625, 18150)
```