MACHINE LEARNING

REPORT

TOPICS

- MULTIPLE LINEAR REGRESSION
- POLYNOMIAL REGRESSION
- LOGISTIC REGRESSION
- K NEAREST NEIGHBOR
- N NEURAL NETWORK

Admission Number: 22je0533

Name: Madhav Kalra

Highlights

- **Completed** all the **5 algorithms** that were required for the finishing of the project.
- Achieved a **high r2 score/accuracy** in all the algorithms.
- Plotted all the necessary graphical data for the user to understand it quickly.
- Reduced running time by the use of vectorization.

Challenges

- With no prior experience of python or machine learning, I have learnt 5 major algorithms and python libraries like numpy and pandas in a short period of 1 month.
- To get the in-depth knowledge about machine learning, I had to search for more material online along with the course given by the seniors.

Machine Learning

Machine learning enables computer systems to learn and improve automatically based on experience without being explicitly programmed. It involves the development of algorithms and statistical models that allow computer systems to identify patterns in data and make predictions or decisions based on that data.

There are mainly two types of machine learning algorithms: Supervised and Unsupervised machine learning. All the algorithms that I have solved in this project are a part of Supervised machine learning. In supervised learning, we give datasets of input x and output y on test dataset x.

ALGORITHMS

MULTIPLE LINEAR REGRESSION

Multiple Linear Regression is a part of Supervised Machine learning in which we have to find the best value of weights and bias which predict the best outcome to get the maximum r2 score.

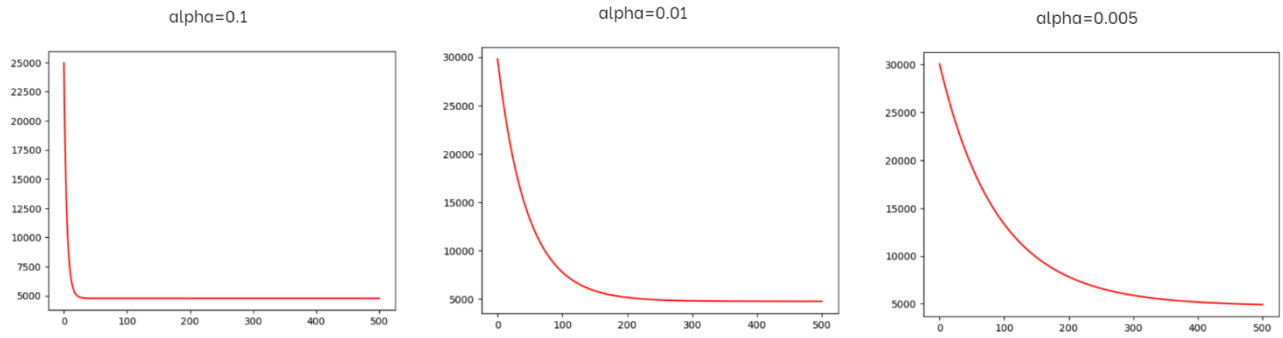
Linear regression being the first algorithm to be implemented made me aware of the various terms used in machine learning, we got to know about:

- Cost Function
- Vectorization of code
- Gradient Descent
- Running the model for different values of alpha

Highlights :

- The time for running the code is "**0 seconds**"
- The r2 score for the test data step is **0.8447**
- The code is well vectorized
- The code id divided into **different functions** for the user to understand the code easily and quickly

The graph for **cost** is represented by the following graph :



The **r2 score** is represented in the cell below:

```
X_t=(X_t-mu)/sigma

y1=(np.matmul(X_t,w))+b
mean=np.sum(y_t)/5000
e1=np.sum(np.square(y1-y_t))
e2=np.sum(np.square(y_t-mean))
acc=1-(e1/e2)
print(acc)
```

```
0.8447322343956226
```

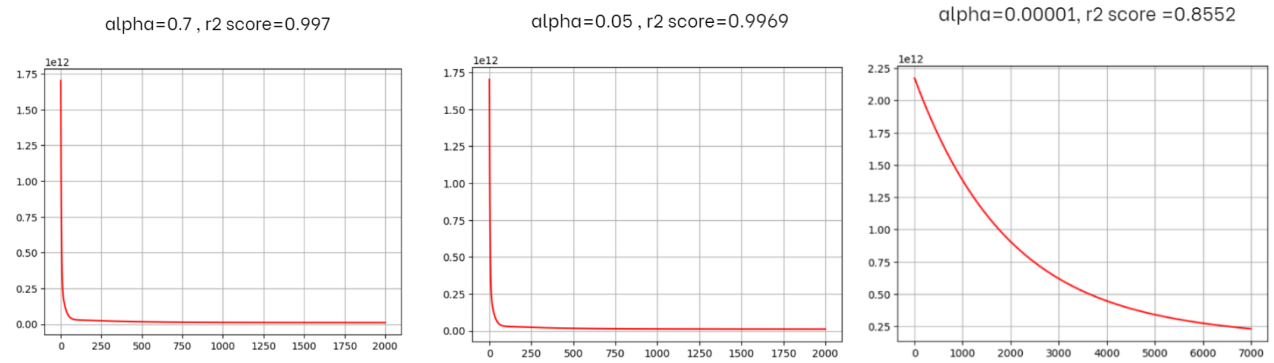
POLYNOMIAL REGRESSION

Polynomial Regression is very similar to linear regression, it has the same functionality but it is more useful in some cases as the linear curve cannot satisfy all the points in a data set where we can use polynomial regression.

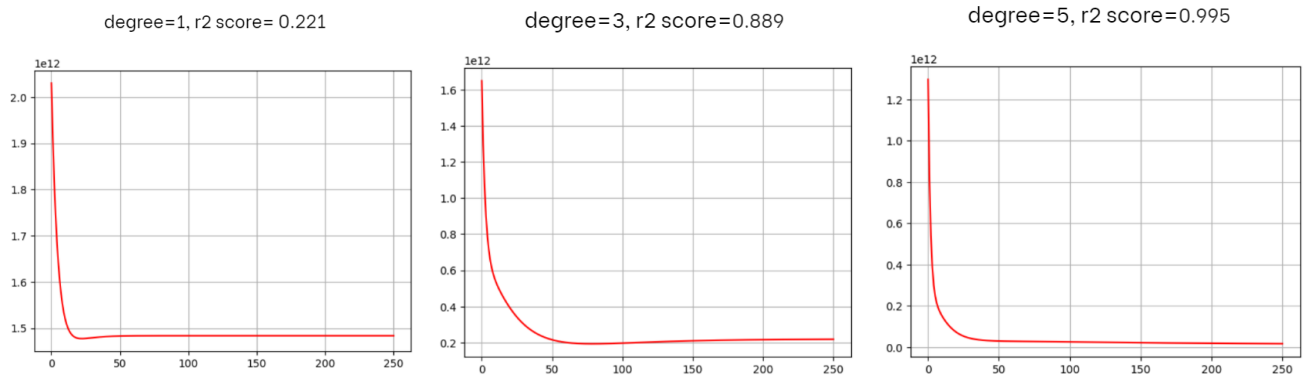
In polynomial regression we have to effectively choose the value of degree so that we get the best r2 and the best fitting of data points with the curve.

While performing polynomial regression we can get some errors like overfitting and underfitting

Comparison of r2 score with alpha



Comparison of r2 score with degree of polynomial



Therefore, in polynomial regression we learn that the best degree to use is **5th degree** and the best value of **alpha to be used is 0.05.**

LOGISTIC REGRESSION

Cost for all the labels in logistic regression

```
********** for  1  label **********
cost after  0 iteration is :  0.6931471805599452
cost after  200 iteration is :  0.3405634907207739
cost after  400 iteration is :  0.32646346611935
cost after  600 iteration is :  0.3248925744326519
********** for  2  label **********
cost after  0 iteration is :  0.6931471805599452
cost after  200 iteration is :  0.33942144605799573
cost after  400 iteration is :  0.32520885211112505
cost after  600 iteration is :  0.323614279864094
********** for  3  label **********
cost after  0 iteration is :  0.6931471805599452
cost after  200 iteration is :  0.33982959406920654
cost after  400 iteration is :  0.32565730476705673
cost after  600 iteration is :  0.32407123032376384
********** for  4  label **********
cost after  0 iteration is :  0.6931471805599452
cost after  200 iteration is :  0.3411336155249196
cost after  400 iteration is :  0.32708954560307774
cost after  600 iteration is :  0.32553036274559144
********** for  5  label **********
cost after  0 iteration is :  0.6931471805599452
cost after  200 iteration is :  0.34105220573075534
cost after  400 iteration is :  0.3270001557043254
cost after  600 iteration is :  0.3254393054856478
********** for  6  label **********
cost after  0 iteration is :  0.6931471805599452
cost after  200 iteration is :  0.3424345141631985
cost after  400 iteration is :  0.3285175199868148
cost after  600 iteration is :  0.32698477523479313
********** for  7  label **********
cost after  0 iteration is :  0.6931471805599452
cost after  200 iteration is :  0.33901299236626936
cost after  400 iteration is :  0.3247599808291247
cost after  600 iteration is :  0.3231568655289932
********** for  8  label **********
cost after  0 iteration is :  0.6931471805599452
cost after  200 iteration is :  0.3404004880675174
cost after  400 iteration is :  0.3262844360940159
cost after  600 iteration is :  0.32471018299760057
********** for  9  label **********
cost after  0 iteration is :  0.6931471805599452
cost after  200 iteration is :  0.34235329938135567
cost after  400 iteration is :  0.32842839629060355
cost after  600 iteration is :  0.32689401230200316
********** for  10  label **********
cost after  0 iteration is :  0.6931471805599452
cost after  200 iteration is :  0.3434891987441874
cost after  400 iteration is :  0.3296746182664025
cost after  600 iteration is :  0.328163025675398
```

Logistic regression is a statistical method that is used for building machine learning models where the dependent variable is binary.
Logistic regression is used to describe data and the relationship between one dependent variable and one or more independent variables.
The dataset given to us contained 784 pixels (features) and 30000 examples, which contained values which were not binary, it contained values till 255. Therefore, we had to do one hot encoding i.e. changing all the values into binary by making an array of 10x30000.

```
#to make an array of Y values in terms of 0 and 1 only

Y=np.zeros((30000,10))
for j in range(10):
  for i in range(30000):
    if (p[i]==j):
      Y[i,j]=1
#now, Y is the target values for logistic regression

#splitting the target values
y=Y[0:24000,:]
y_t=Y[24000:30000,:]
```

HIGHLIGHTS:

- I have achieved a very high accuracy of 89.96 % in the logistic regression

```
the accuracy for 1  label is  89.91666666666667
the accuracy for 2  label is  89.68333333333334
the accuracy for 3  label is  89.76666666666667
the accuracy for 4  label is  90.03333333333333
the accuracy for 5  label is  90.01666666666667
the accuracy for 6  label is  90.3
the accuracy for 7  label is  89.60000000000001
the accuracy for 8  label is  89.88333333333334
the accuracy for 9  label is  90.28333333333333
the accuracy for 10  label is  90.18333333333334
==================================================
average accuracy is:  89.96666666666667
```

- I have split the code into various functions to make it easier for the user to understand the code
- The time for running the code is very less i.e. around 1.5 minutes
- The code is highly vectorized and written neatly

K NEARENST NEIGHBORS (KNN)

The k-nearest neighbors algorithm, also known as KNN or k-NN, is a non-parametric, supervised learning classifier, which uses proximity to make classifications or predictions about the grouping of an individual data point.

In KNN we find the distance of a given test point from the other training points and the first k values of the distance determine the accuracy of the classification algorithm.

```
def dis(X,X_t):
  w=np.sqrt(np.sum((X-X_t)**2,axis=1))
  return w
```

The value of k greatly effects the value of accuracy. For my code, when the value of **k is 11** the function gives maximum accuracy which is **84.8%.**

```
t=[0]*1000
count=0
for i in range(1000):
  t[i]=val(X_t[i],y_t[i],173)
  if(t[i]==y_t[i]):
    count+=1
acc=count/10
print(acc)
```

```
84.8
```

N-Neural Network

A neural network is a method in artificial intelligence that teaches computers to process data in a way that is inspired by the human brain. It is a type of machine learning process, called deep learning, that uses interconnected nodes or neurons in a layered structure that resembles the human brain.

Making a neural network, consists of some steps that I kept in mind while performing:

1. Parameters initialization
2. Forward Propagation
3. Cost Calculation
4. Back Propagation
5. Parameters Update
6. Model prediction
7. Test Accuracy

One hidden layer network:

For the complex learning of n layer neural network, I first made a model to perform single hidden layer neural network.
In my code I have given the user to choose among different activation functions like tanh and relu.
For my code, I derived all the backward propagation equations which are as follows:

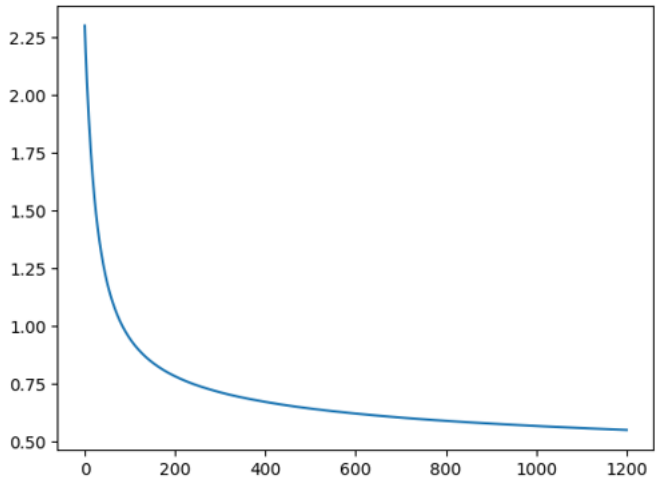
```
def backward_prop(x, y, parameters, forward_func):
    w1 = parameters['w1']
    b1 = parameters['b1']
    w2 = parameters['w2']
    b2 = parameters['b2']

    a1 = forward_func['a1']
    a2 = forward_func['a2']

    m = x.shape[1]

    dz2 = (a2 - y)
    dw2 = (1/m)*np.dot(dz2, a1.T)
    db2 = (1/m)*np.sum(dz2, axis = 1, keepdims = True)

    dz1 = (1/m)*np.dot(w2.T, dz2)*derivative_tanh(a1)
    dw1 = (1/m)*np.dot(dz1, x.T)
    db1 = (1/m)*np.sum(dz1, axis = 1, keepdims = True)

    grads={
        "dw1" : dw1,
        "dw2" : dw2,
        "db1" : db1,
        "db2" : db2,
        "dz1" : dz1,
        "dz2" : dz2
    }
```

In the neural network, I have made the use of dictionaries, to store the various values of parameters and gradients which make the calling of the variables very easy and for the user to understand the code easily.

The following snippet is the model which calls all the various functions used in the code.

```
def model(x, y, n, learning_rate, iterations):

    p = x.shape[0]
    q = y.shape[0]

    cost_list = []

    parameters = parameter(p, n, q)

    for i in range(iterations):

        forward_func = forward_prop(x, parameters)
        cost_cal = cost(forward_func['a2'], y)
        cost_list.append(cost_cal)
        grads = backward_prop(x, y, parameters, forward_func)
        parameters=update_parameters(parameters, grads, learning_rate)
        #if(i%10 == 0):
        print("Cost after", i, "iterations is :", cost_cal)

    return parameters, cost_list
```

The following graph displays the cost vs iteration graph for single layer neural network



In the following single layer network, I have used 784 neurons which trains my model to a good extent. For the network, I have achieved an accuracy of 81.55%

```
w1 = Parameters['w1']
b1 = Parameters['b1']
w2 = Parameters['w2']
b2 = Parameters['b2']

z1 = np.dot(w1, X_t) + b1
a1 = relu(z1)

z2 = np.dot(w2, a1) + b2
a2 = softmax(z2)

acc=0
a=np.random.rand(6000)
for i in range(6000):
  a[i]=np.argmax(a2[:,i])
a=a.reshape(6000,1)
for i in range(6000):
  if(label[i]==a[i]):
    acc+=1
print(round((acc/60),3))
```

```
81.55
```

N Layer Neural Network

Using the knowledge of one layer neural network, I trained my n layer network using generalization of one layer network. I used the similar concept of dictionaries to make it easier to call the various gradients and parameters and easily initialize all the dimensions of w's and b's. For example:

```
def parameters_initialization(dimensions):
  parameters={}
  N=len(dimensions)   #N is the number of layers
  for i in range(1, N):
        parameters['W' + str(i)] = np.random.randn(dimensions[i], dimensions[i-1])*0.01
        parameters['b' + str(i)] = np.zeros((dimensions[i], 1))

  return parameters
```

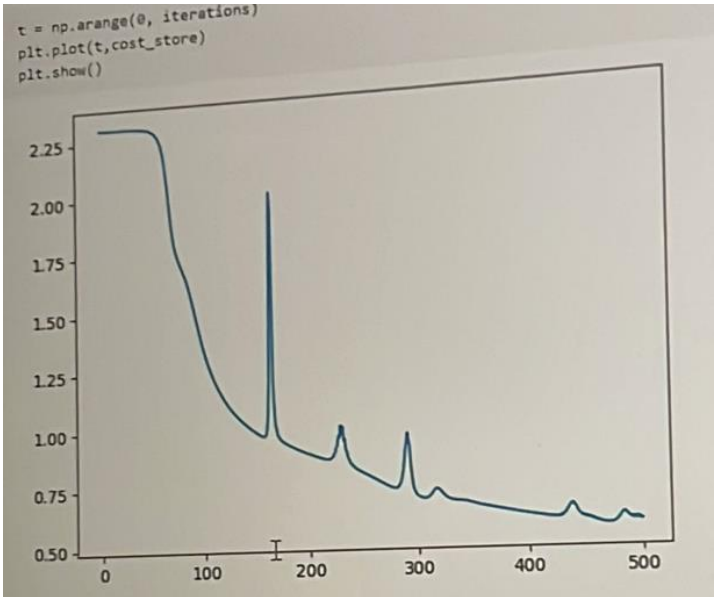
For the calculation of cost, I have used cross entropy cost function

```
def cost(AN,Y):
  m=y.shape[1]
  cost = -(1/m)* np.sum(Y * np.log(AN))
  cost=np.squeeze(cost)
  return cost
```

For the backward propagation, I have generalized the code written for single layer neural network. Here are my back propagation values of derivatives using dictionaries

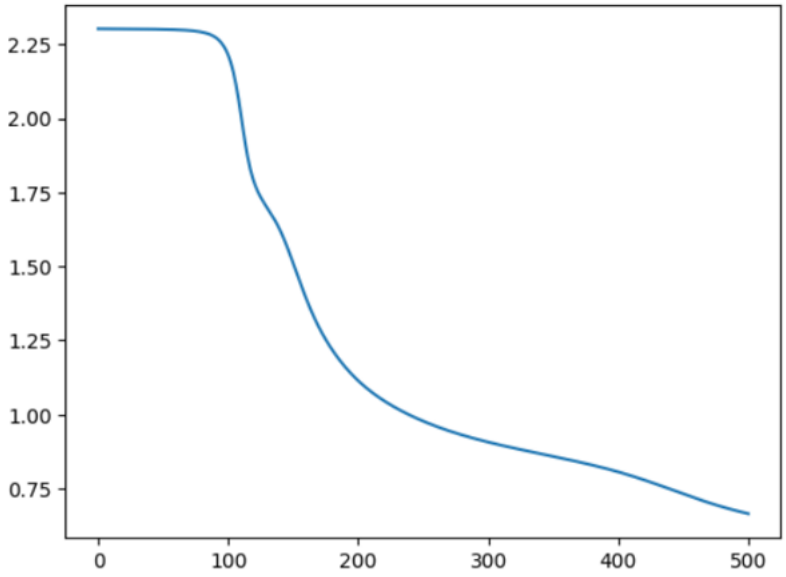
```
def back_propagation(AN,y,parameters,forward_func,activation):
  gradient={}
  N = len(parameters) // 2
  m = AN.shape[1]

  gradient["dZ" + str(N)] = AN - y
  gradient["dW" + str(N)] =  (1/m)*np.matmul(gradient["dZ" + str(N)],forward_func['A' + str(N-1)].T)
  gradient["db" + str(N)] =  (1/m)*np.sum(gradient["dZ" + str(N)], axis = 1, keepdims = True)

  for i in reversed(range(1, N)):
      if activation == 'tanh':
            gradient["dZ" + str(i)] = np.matmul(parameters['W' + str(i+1)].T,gradient["dZ" + str(i+1)])*d_tanh(forward_func['A' + str(i)])
      else:
            gradient["dZ" + str(i)] = np.matmul(parameters['W' + str(i+1)].T,gradient["dZ" + str(i+1)])*d_relu(forward_func['A' + str(i)])

      gradient["dW" + str(i)] =  (1/m)*np.matmul(gradient["dZ" + str(i)],forward_func['A' + str(i-1)].T)
      gradient["db" + str(i)] =  (1/m)*np.sum(gradient["dZ" + str(i)], axis = 1, keepdims = True)


      parameters["W" + str(i+1)] = parameters["W" + str(i+1)] - alpha * gradient["dW" + str(i+1)]
      parameters["b" + str(i+1)] = parameters["b" + str(i+1)] - alpha * gradient["db" + str(i+1)]

  return parameters
```

After running my model, for 3 layers with 784, 196 and 49 neurons respectively, I got an accuracy of 75.733%. In the following snippet I have shown a cost vs iterations graph before optimizing the values of alpha and neurons which gives a lot of jitter in the cost curve.



After optimizing the values of alpha and neurons, I have achieved a smooth graph of cost vs iterations which gives a accuracy of nearly 78-80% for a range of values.

```
t = np.arange(0, iterations)
plt.plot(t,cost_store)
plt.show()
```



The following snippet gives the accuracy of the n layer neural network after applying forward propagation on the output value of w's and b's.

```
forward_func_t = {}
N = len(parameters) // 2

forward_func_t['A0']=X_t

for i in range(1,N):
    forward_func_t['Z'+str(i)] = np.matmul(parameters['W'+str(i)],forward_func_t['A'+str(i-1)]) + parameters['b'+str(i)]

    if activation == 'tanh':
        forward_func_t['A' + str(i)] = tanh(forward_func_t['Z' + str(i)])
    else:
        forward_func_t['A' + str(i)] = relu(forward_func_t['Z' + str(i)])

forward_func_t['Z'+str(N)] = np.matmul(parameters['W'+str(N)],forward_func_t['A'+str(N-1)]) + parameters['b'+str(N)]
forward_func_t['A'+str(N)] = softmax(forward_func_t['Z'+str(N)])

acc=0
a=np.random.rand(6000)
for i in range(6000):
  a[i]=np.argmax((forward_func_t['A'+str(N)])[:,i])
a=a.reshape(6000,1)
for i in range(6000):
  if(label[i]==a[i]):
    acc+=1
print(round((acc/60),3))
```

```
75.733
```

Conclusion

The support of the mentors for entire period of winter of code made it easy for me to complete the project. The course provided by the seniors was quite fulfilling and contained vast information about machine learning. I am very thankful to all the seniors for accepting my proposal which allowed me to participate in this project and it makes me proud to say that **I have completed all the algorithms** with the **maximum possible r2 score/accuracy** that I could achieve. THANK YOU

About Me:

Name: Madhav Kalra
Admission Number: 22je0533
Email Address: 22je0533@iitism.ac.in