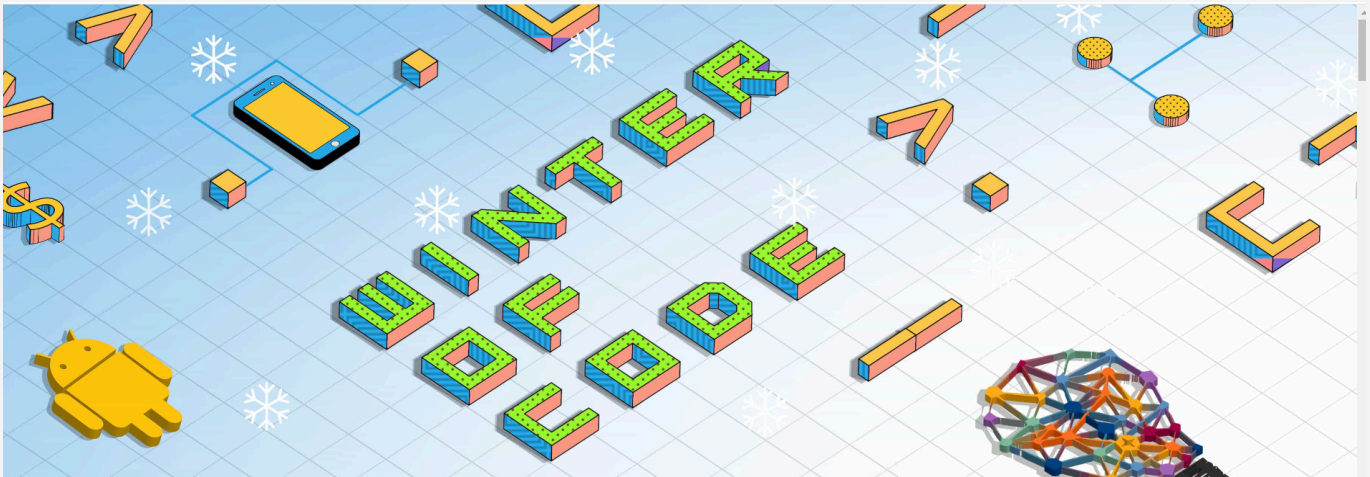


ML Bootcamp

Project Report by Daksh Mor



Project :

This year's Winter of Code project wants us to make 6 different ML algos from scratch in python using limited python libraries and then implement them in jupyter notebook.

Linear Regression

Brief meaning :

- It is a supervised Machine Learning algorithm
- Used when the data is linear and a straight line can fit the model well

Formulas used :

Cost :

$$J(w, b) = \frac{1}{2m} \sum_{i=1}^m (f_{w,b}(x^{(i)}) - y^{(i)})^2$$

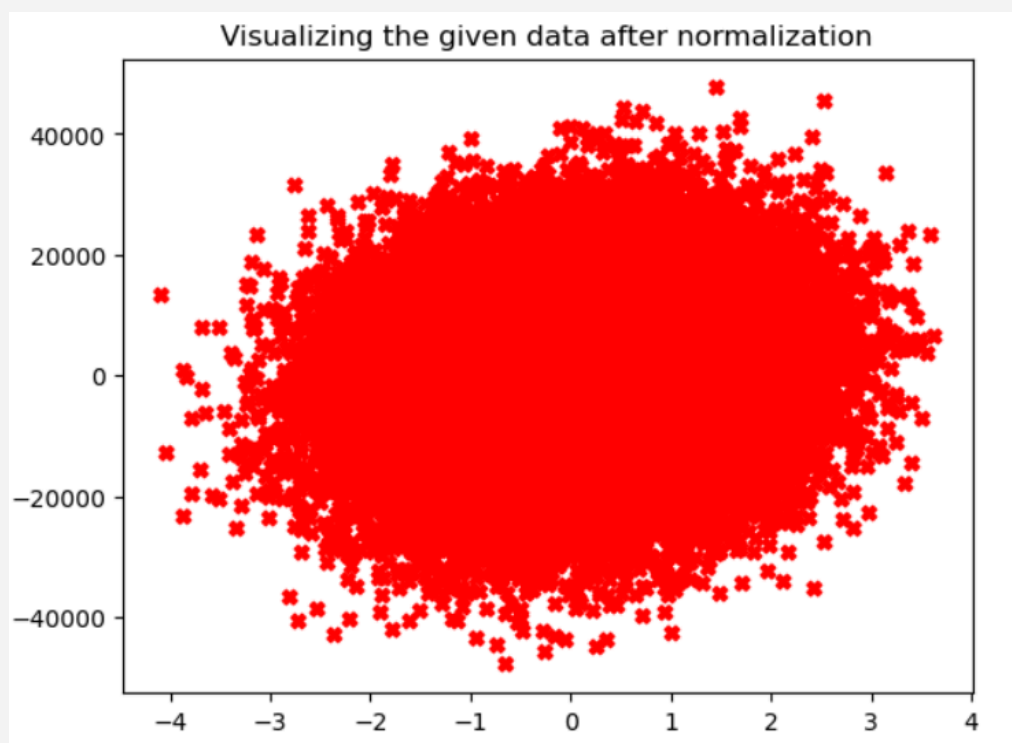
Gradient :

$$\frac{\partial J}{\partial w} = \frac{1}{m} \sum_{i=1}^m (f_{w,b}(x^{(i)}) - y^{(i)}) \cdot x^{(i)}$$

$$\frac{\partial J}{\partial b} = \frac{1}{m} \sum_{i=1}^m (f_{w,b}(x^{(i)}) - y^{(i)})$$

Graph and hyperparameter values :

Training visualisations

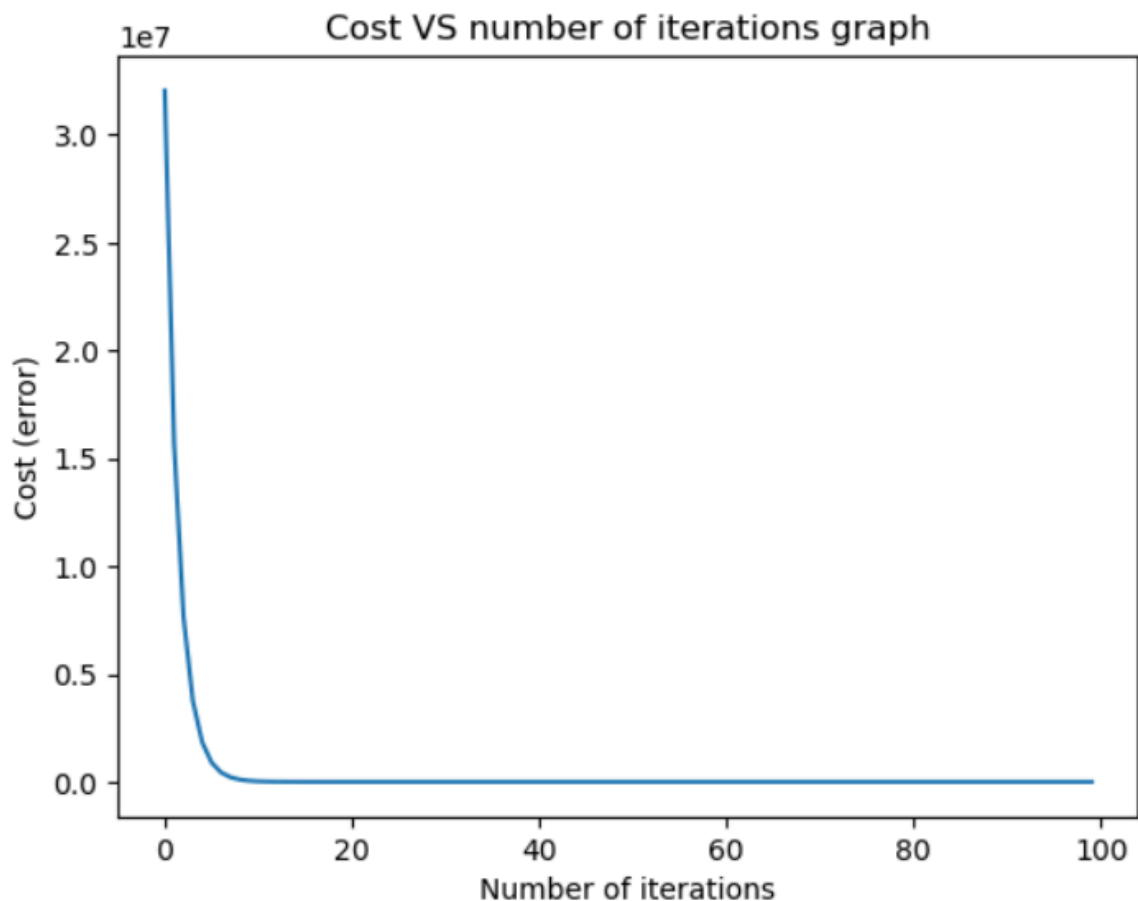


Hyperparameters

```
# Optimal alpha  
alpha = 0.3
```

Training logs

```
Cost for 0th iteration = starting  
Cost for 100th iteration = 0.005053987768831164  
Cost for 200th iteration = 0.005053987768831513  
Cost for 300th iteration = 0.005053987768831513  
Cost for 400th iteration = 0.005053987768831513  
Cost for 500th iteration = 0.005053987768831513  
Cost for 600th iteration = 0.005053987768831513  
Cost for 700th iteration = 0.005053987768831513  
Cost for 800th iteration = 0.005053987768831513  
Cost for 900th iteration = 0.005053987768831513
```



^^^^^^^^^^^^^^^^For alpha = 0.3^^^^^^^^^^^^^^^^^^^^

R2 Score: 0.999999999229305

R2 Score: 0.999999999226438

NOTE : above R2 score for training and below for cross validation

More info :

```
def compute_gradient(x,y,w,b):  
    variab = (np.dot(x,w.T)+b-y) #where variable = w.x+b-y  
    m,n=x.shape #where m = number of observations and n = number of features  
    dj_db = np.mean(variab,axis=0)  
    dj_dw = (np.dot(variab.T,x))/x.shape[0]  
    return dj_dw,dj_db
```

Before I used a loop for dj_dw but then as I made all the algorithms I got to know more about vectorization and then applied it here also for dj_dw. Time taken = 20secs for 1000 iterations

Polynomial Regression

Brief meaning :

- It is a supervised Machine Learning algorithm
- Used when the data is non linear and a polynomial function can fit the model well

Formulas used :

Cost :

$$J(w, b) = \frac{1}{2m} \sum_{i=1}^m (f_{w,b}(x^{(i)}) - y^{(i)})^2$$

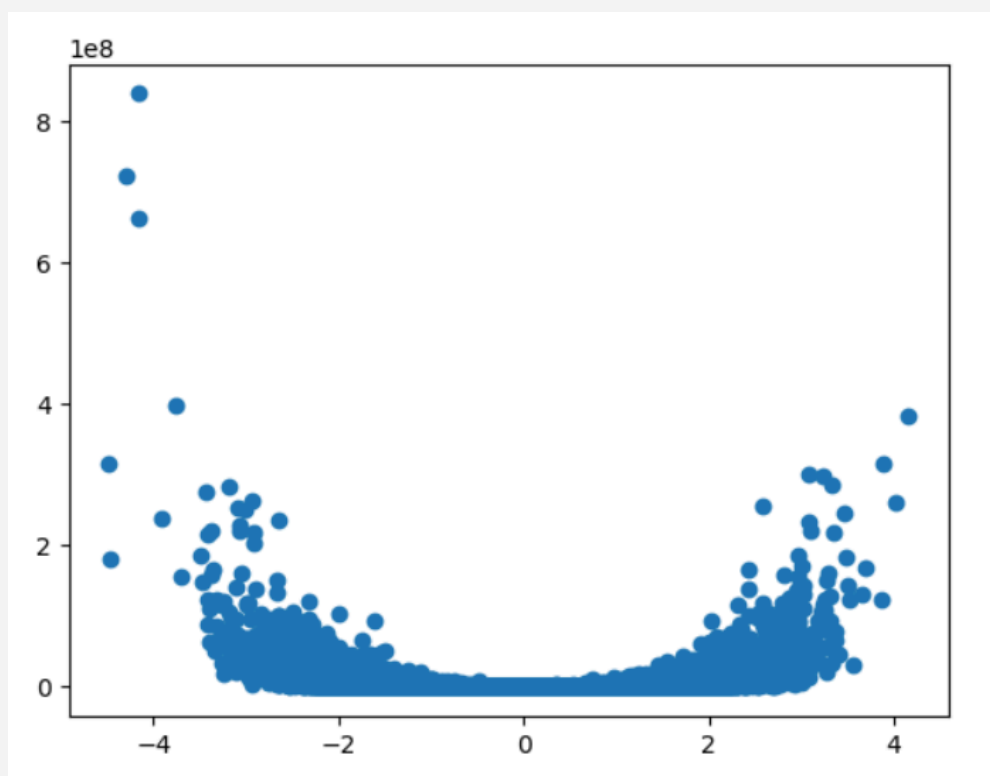
Gradient :

$$\frac{\partial J}{\partial w} = \frac{1}{m} \sum_{i=1}^m (f_{w,b}(x^{(i)}) - y^{(i)}) \cdot x^{(i)}$$

$$\frac{\partial J}{\partial b} = \frac{1}{m} \sum_{i=1}^m (f_{w,b}(x^{(i)}) - y^{(i)})$$

Graph and hyperparameter values:

Training visualisations

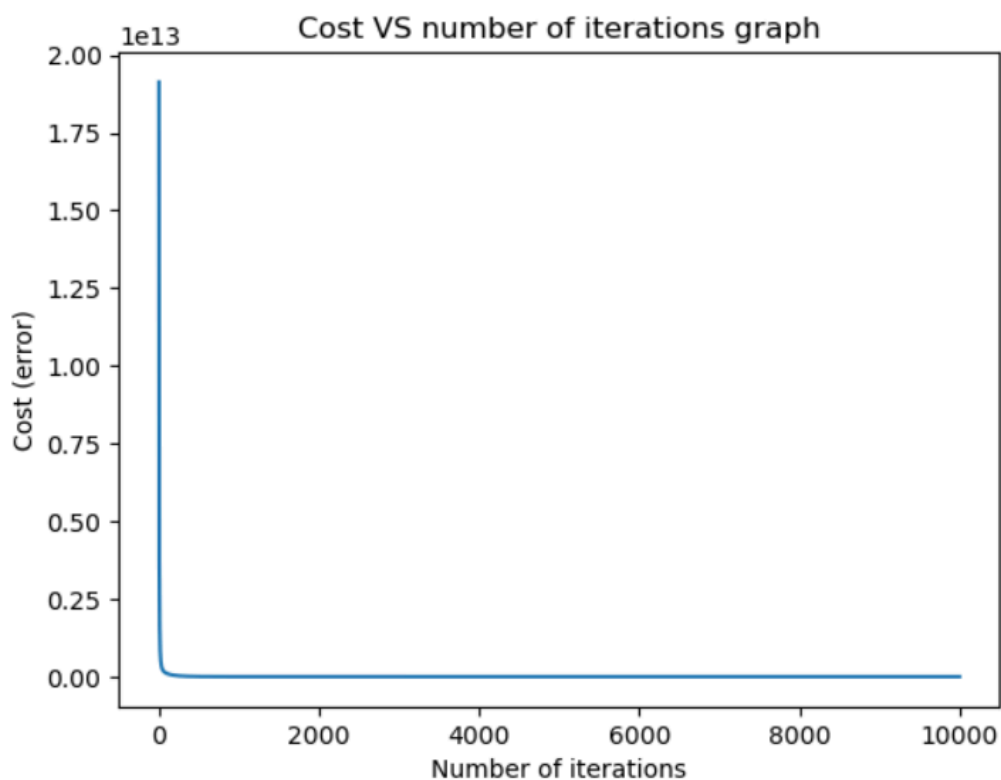


Hyperparameters

- $\alpha = 0.27$
- 6 Degree Polynomial

Training logs

```
Cost for 0th iteration = starting
Cost for 1000th iteration = 204680770.73082796
Cost for 2000th iteration = 429490.69843555934
Cost for 3000th iteration = 1484.8856107829818
Cost for 4000th iteration = 7.847061589557382
Cost for 5000th iteration = 0.0492442939848225
Cost for 6000th iteration = 0.00032387789360139744
Cost for 7000th iteration = 2.1552806751595126e-06
Cost for 8000th iteration = 1.4389436042309152e-08
Cost for 9000th iteration = 9.618731609331944e-11
```



Minimum cost = $6.437621713661311e-13$

```
Training set R2 score : 1.0
Cross validation set R2 score : 1.0
```

More info :

I tried different degree even polynomial function as it is mostly an even degree polynomial because of the graph plotted for given data

Degree = 6 suited well with alpha = 0.27

And also I made a generalised function to convert 3 features into any degree polynomial by observing the pattern .

Polynomial Regression = Linear Regression if you convert the features into more features of required degree . Time taken = 10-15 min , 10k iters

Logistic Regression

Brief meaning :

- It is a supervised Machine Learning algorithm
- Used for classification using logistic(sigmoid) function
- For multiclass classification one vs rest method can be used

Formulas used :

Gradient : Same as Linear Regression, just the function f is sigmoid func.

$$\frac{\partial J}{\partial w} = \frac{1}{m} \sum_{i=1}^m (f_{w,b}(x^{(i)}) - y^{(i)}) \cdot x^{(i)}$$

$$\frac{\partial J}{\partial b} = \frac{1}{m} \sum_{i=1}^m (f_{w,b}(x^{(i)}) - y^{(i)})$$

Graph and hyperparameter values :

Training visualisations



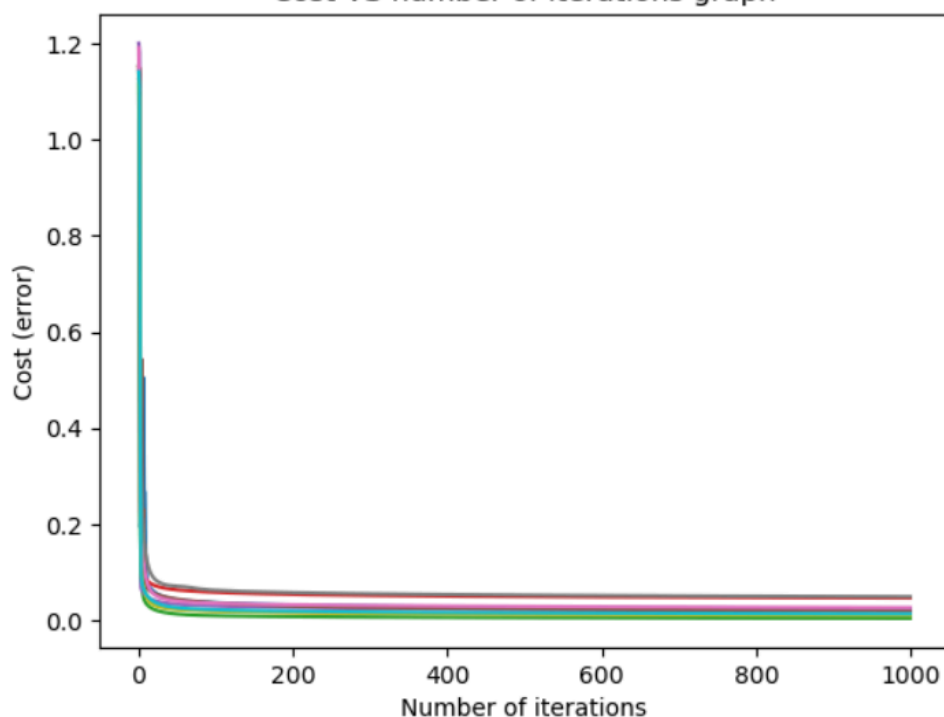
Hyperparameters

alpha = 5.7

Training logs

```
0th number of iteration with cost : [1.1531244 1.15261107 1.11511026 1.1438672 1.20094777 1.15082269
1.19170072 1.12591627 1.13652999 1.14155725]
100th number of iteration with cost : [0.03649547 0.02037985 0.00953861 0.05826292 0.02971098 0.03616095
0.03529388 0.06231098 0.0189537 0.02198275]
200th number of iteration with cost : [0.03193071 0.01720975 0.00761615 0.05392646 0.02661375 0.02966914
0.03243253 0.05766365 0.01603981 0.01900737]
300th number of iteration with cost : [0.02957718 0.01553998 0.00664518 0.05171182 0.02496033 0.02653514
0.03087497 0.05525431 0.01443968 0.01746907]
400th number of iteration with cost : [0.02806017 0.01440864 0.00597556 0.05030062 0.02386244 0.02458231
0.02979115 0.05364521 0.0133353 0.01643925]
500th number of iteration with cost : [0.02697447 0.0135573 0.00547233 0.04929185 0.02305305 0.02320194
0.02896246 0.05246147 0.01250392 0.01567445]
600th number of iteration with cost : [0.02614607 0.01287748 0.00508086 0.04851693 0.02241794 0.02215048
0.02829356 0.05153808 0.01184562 0.01507074]
700th number of iteration with cost : [0.02548572 0.0123135 0.00476699 0.04789204 0.02189863 0.02130921
0.02773414 0.05078863 0.01130576 0.01457454]
800th number of iteration with cost : [0.02494231 0.01183308 0.00450804 0.0473706 0.02146147 0.02061238
0.02725436 0.05016256 0.01085133 0.01415485]
900th number of iteration with cost : [0.02448421 0.01141583 0.00428936 0.04692445 0.02108542 0.02002019
0.02683504 0.04962793 0.01046101 0.01379223]
```

Cost VS number of iterations graph



NOTE : there are 10 cost because of 10 classes as I have used one vs all strategy

Accuracy for training set = 97.504

Accuracy for Cross validation set = 96.74

Left side actual , Right side predicted values



More Info :

In logistic Regression I only know how to solve binary classification and in the process I get to know how to solve multi-class classification using one vs all strategy. Time Taken = 5-6 min

K nearest neighbours (KNN)

Brief meaning :

- It is a supervised Machine Learning algorithm
- Used to classify the data using the euclidean distance formula
- It can also be used for regression

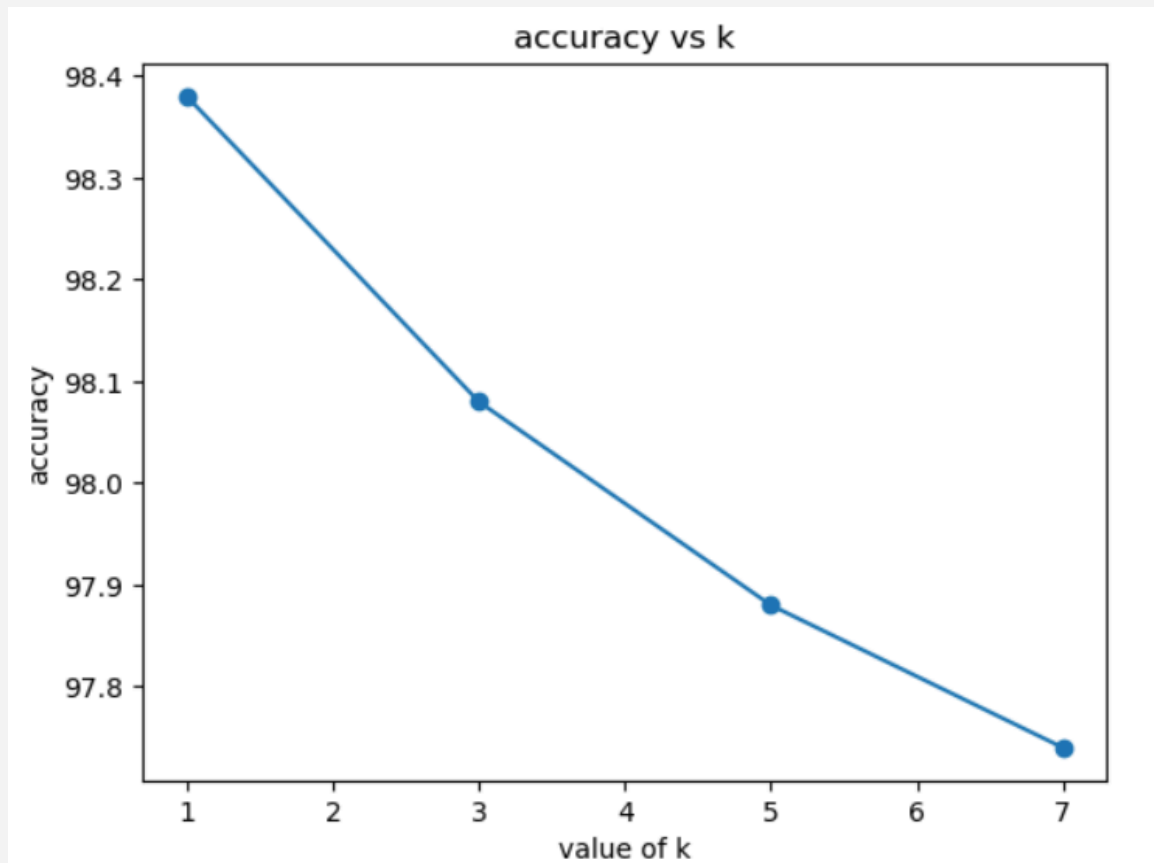
Graph and hyperparameter values :

Training visualisations



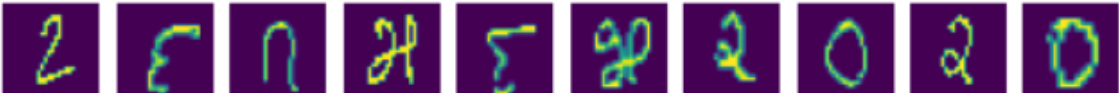
Hyperparameters

K = 1 has the best accuracy



Left side actual value , Right Side predicted value

[1],[1][4],[4][2],[2][6],[6][3],[3][2],[2][6],[6][5],[5][7],[7][6],[6]



More Info :

learned new way to calculate the distance of the test data wrt training

data without the loop using $(a - b)^2 = a^2 + b^2 - 2a \cdot b$ which makes the code much much faster due to the no loop for calculating distances

Time Taken : 2.3 mins for k =1,3,5,7

K-Means Clustering

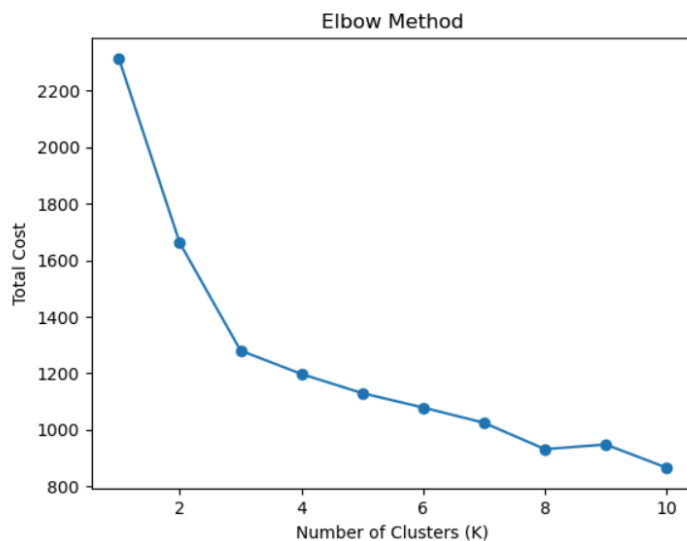
Brief meaning :

- It is a unsupervised Machine Learning algorithm
- Used to divide a dataset into K clusters

Graph and hyperparameter values :

Hyperparameter : K=3 using the elbow method

[2314.0, 1661.2542233273616, 1279.7311231046358, 1196.8520574117506, 1129.0029194716558, 1078.1623431433077, 1024.1273518025187, 930.6520171082942, 947.258156682988, 865.1217557324187]



More Info :

Here also the knn no loop method helped to calculate the distance

Time Taken : 3.5 seconds for 500 iterations and 10 k (1 to 10)

n-layer Neural Network

Brief meaning :

- Inspired by biological neural networks in the human brain
- Fundamental component of deep learning
- Deep Learning comes under Machine Learning

Graph and hyperparameter values :

Training visualisations

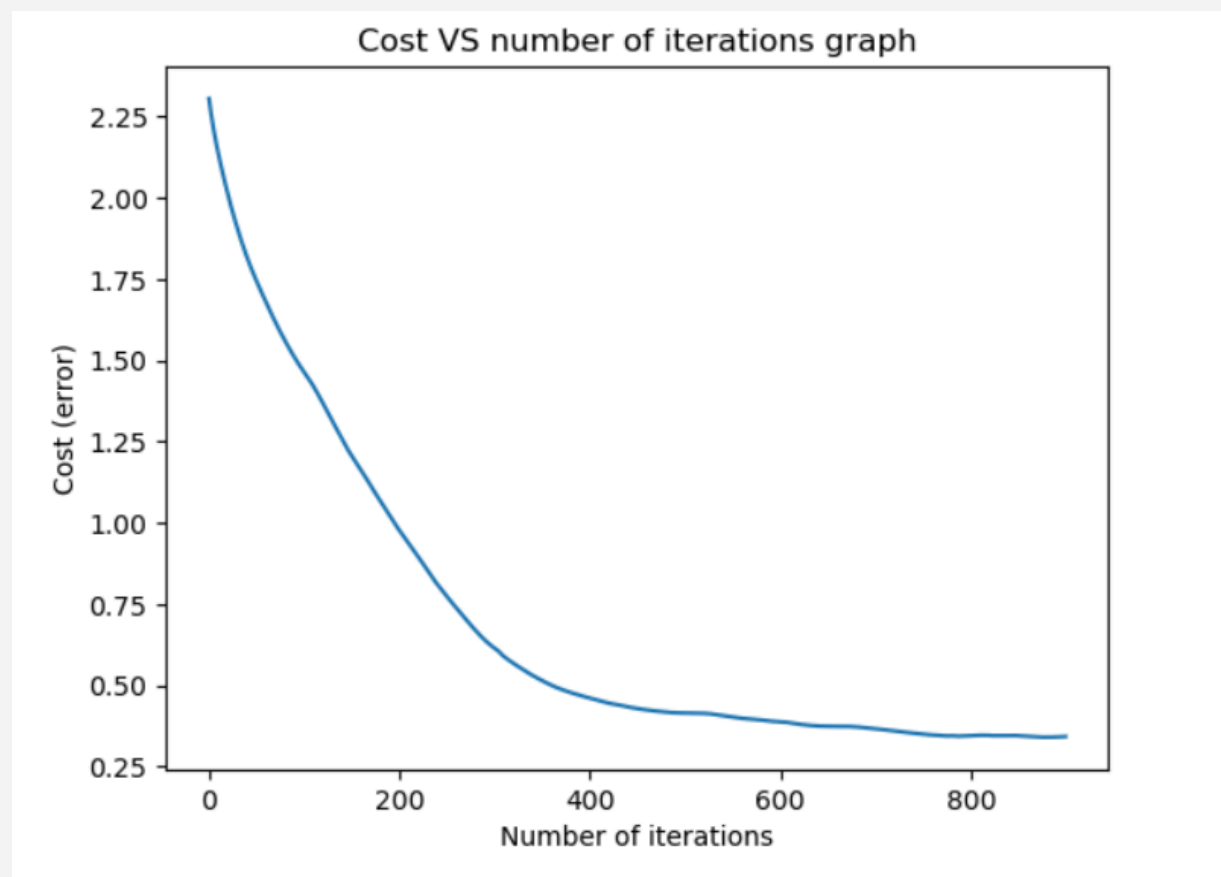


Hyperparameters

- Number of layers = 3
- With neurons sizes = 512,256,10
- 10 neurons in the output layer so that softmax can be used easily to predict the classes
- ReLu is used as a activation function in hidden layers and softmax in the output layer
- Alpha = 0.00008

Training logs

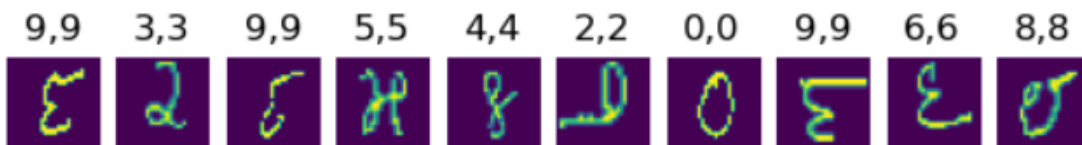
| No of iter | Cost |
|------------|---------------------|
| 0 | 2.30461983355323 |
| 100 | 1.4626123625550538 |
| 200 | 0.9761876095301921 |
| 300 | 0.6138494105843144 |
| 400 | 0.46061198121516195 |
| 500 | 0.4152537377827896 |
| 600 | 0.38784658389362464 |
| 700 | 0.3665548319834625 |
| 800 | 0.34574687363473494 |
| 900 | 0.34321634583050187 |



Accuracy of training set : 89.18

Accuracy of cross validation set : 88.96

Left side actual value , Right Side predicted value



More Info :

I learned what `np.random.seed()` means , why initialising the `w` and `b` parameters with random numbers is not good instead and using He normal in ReLu activation function is much better.

Overall Learnings

Learned why we split data into three parts:

- *Training set* - used to train the model
- *Cross validation set* - used to evaluate whether the model is overfitting or not.
- *Test set* - used to give a fair estimate of your chosen model's performance against new examples.

I First made a separate file named `all_in_one_functions` then integrated the useable function in model itself instead of importing the file because it is taking time to edit the code if found some mistake

Vectorization is must it makes the code so much faster and learned about how the broadcasting in numpy worked

Accuracy of different models

- ☐ Linear Regression : 99.99%
- ☐ Polynomial Regression : 100%
- ☐ Logistic Regression : 96.74%
- ☐ KNN : 98.36%
- ☐ Neural Network : 88.96%

Personal Details

- Name : Daksh Mor
- Branch : B-tech in Computer Science and Engineering
- Admission No : 23JE0286
- Phone No. : 9350114220
- D.O.B : 04/05/2005
- Github profile : <https://github.com/daksh-mor>

Why should I be selected ?

I have attended all the cyberlab division's workshops and finally concluded that I am interested in Machine Learning and want to pursue it as a career path. Being part of Cyberlabs will allow me to explore this field deeper.

I believe that my interest in coding and a growing curiosity about AI makes me a suitable candidate for this project.