ML Bootcamp

Shashank Kumar
IIT(ISM) Dhanbad

REPORT

Linear Regression: -

Idea: -

I have stored the features (From A-T) and Target variable (Label) from Linear_train.csv in x (shape= (50000, 20)) and y (shape= (50000,) NumPy array I have normalised the features $\vec{x}$ using formula $\frac{X_i - \mu_i}{max(x_i) - min(x_{i)}}$ in every i[th] column. Where μ_i is mean of i[th] column

Then minimised the cost function(J)

$$J(\mathbf{w}, b) = \frac{1}{2m} \sum_{i=0}^{m-1} (f_{\mathbf{w},b}(\mathbf{x}^{(i)}) - y^{(i)})^2$$

$$f_{\mathbf{w},b}(\mathbf{x}^{(i)}) = \mathbf{w} \cdot \mathbf{x}^{(i)} + b$$

using Gradient descent algorithm

repeat until convergence: {

$$w_j = w_j - \alpha \frac{\partial J(\mathbf{w}, b)}{\partial w_j} \qquad \text{for j = 0..n-1}$$

$$b = b - \alpha \frac{\partial J(\mathbf{w}, b)}{\partial b}$$

}

Where

$$\frac{\partial J(\mathbf{w}, b)}{\partial w_j} = \frac{1}{m} \sum_{i=0}^{m-1} (f_{\mathbf{w},b}(\mathbf{x}^{(i)}) - y^{(i)}) x_j^{(i)}$$

$$\frac{\partial J(\mathbf{w}, b)}{\partial b} = \frac{1}{m} \sum_{i=0}^{m-1} (f_{\mathbf{w},b}(\mathbf{x}^{(i)}) - y^{(i)})$$

Training Log: -

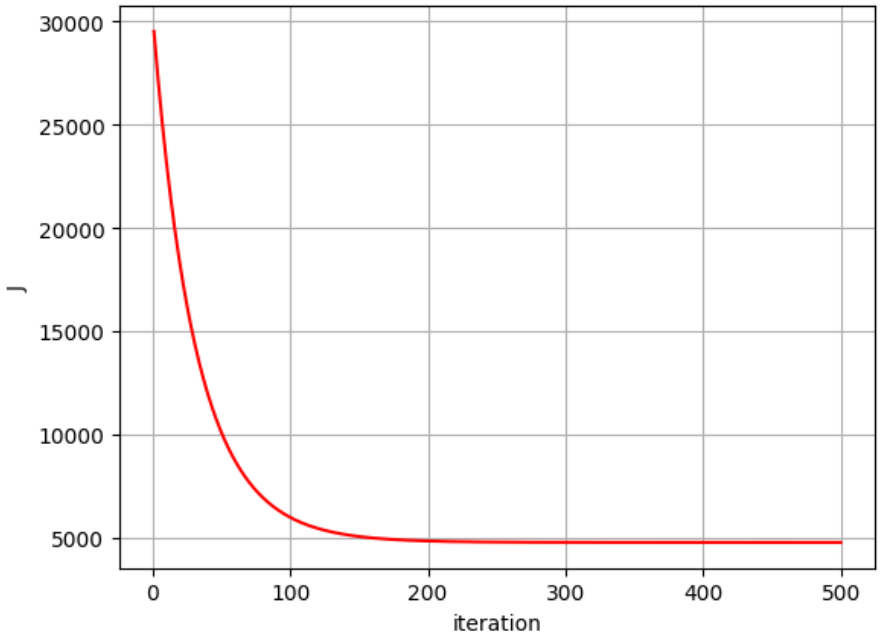
```
for iterations=1 cost function=29526.52873486254
for iterations=51 cost function=9940.74218702937
for iterations=101 cost function=5948.623555061025
for iterations=151 cost function=5048.051190251162
for iterations=201 cost function=4836.760166237751
for iterations=251 cost function=4786.1381302888485
for iterations=301 cost function=4773.820505172735
for iterations=351 cost function=4770.78321943122
for iterations=401 cost function=4770.025352798265
for iterations=451 cost function=4769.834229244515
```

After 500 iterations the value of R^2 we got is R^2 =0.8428148765274308

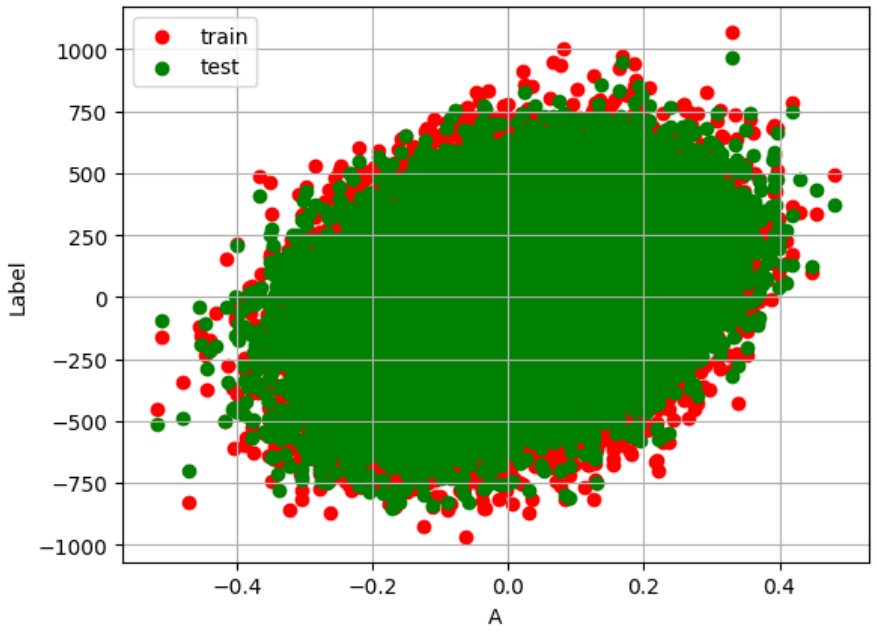
Hyperparameters: -

Learning rate=1
Epochs=500
Optimization Algorithm=Gradient Descent
Cost Function=Root mean square cost function
Activation function=linear

Training Visualisations: -



This represents how the cost function is decreasing with number of iterations in the gradient descent algorithm



This represents the relation between predicted label vs features A and train label and feature A

Polynomial Regression: -

Ideas and findings: -

For polynomial regression same method as linear regression is followed except we have create a polynomial of desired out of the give features A,B and C

For this I have used the concepts few of permutations and combination and multinomial expansion

Required concept: -

1)Numbers of terms of n degree polynomial with 3 features

$$= \sum_{r=1}^{n} \left(\frac{n+r-1}{r-1} \right) = \sum_{r=1}^{n} \left(\frac{n+2}{2} \right) \text{ where r=3(A,B,C)}$$

2)Total degree of each term $(A+B+C)^n = n$

The function we will get for degree 2

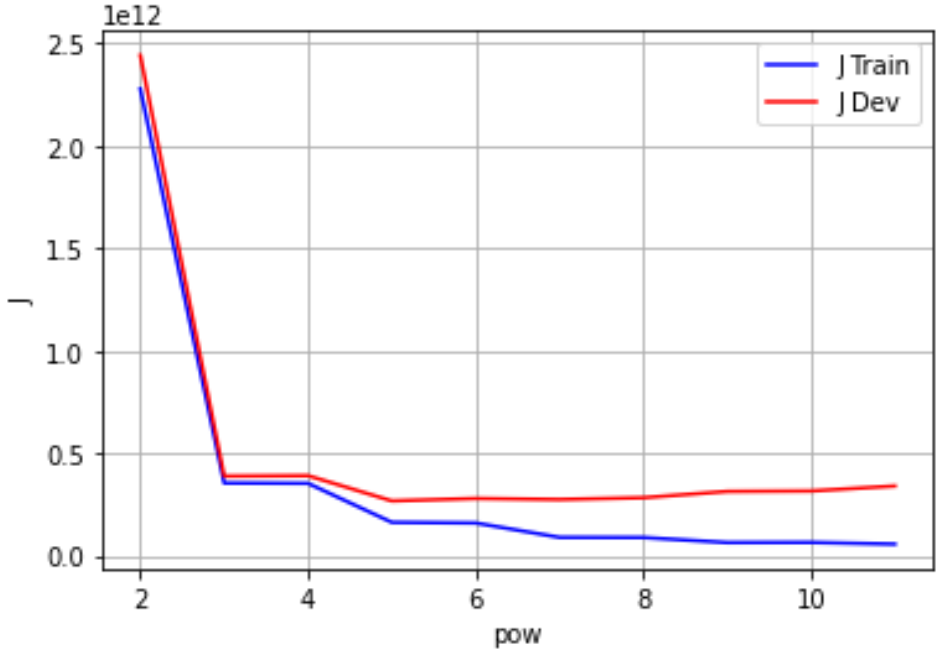
$$f_{w,b}(x) = w_1 A + w_2 B + w_3 C + w_4 A^2 + w_5 B^2 + w_6 C^2 + w_7 AB + w_8 BC + w_9 CA + b$$

I have split the data into three parts train test and cross validation set (also known as developers set)

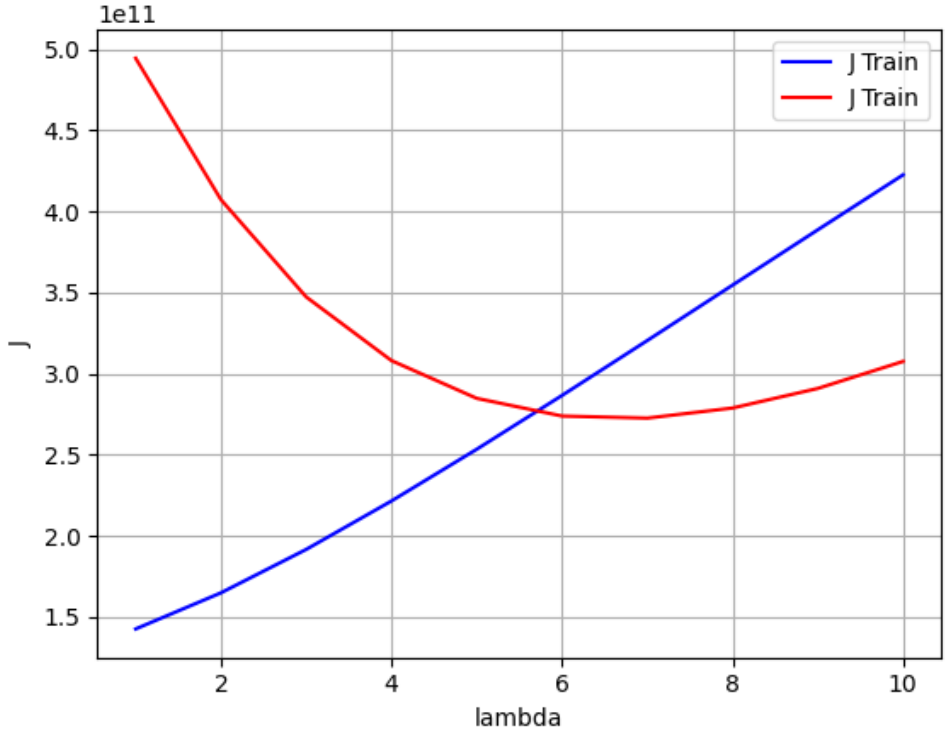
And plot the graph between J_{train} VS degree and J_{Dev} VS degree

And using the concept of trade off between Bias and variance

I have found that the best accuracy will be obtain at degree=5



Similarly, I have found best value for regularisation constant $\lambda = 5$
This is giving 92% accuracy at train set while 93% at dev set



Training Log: -

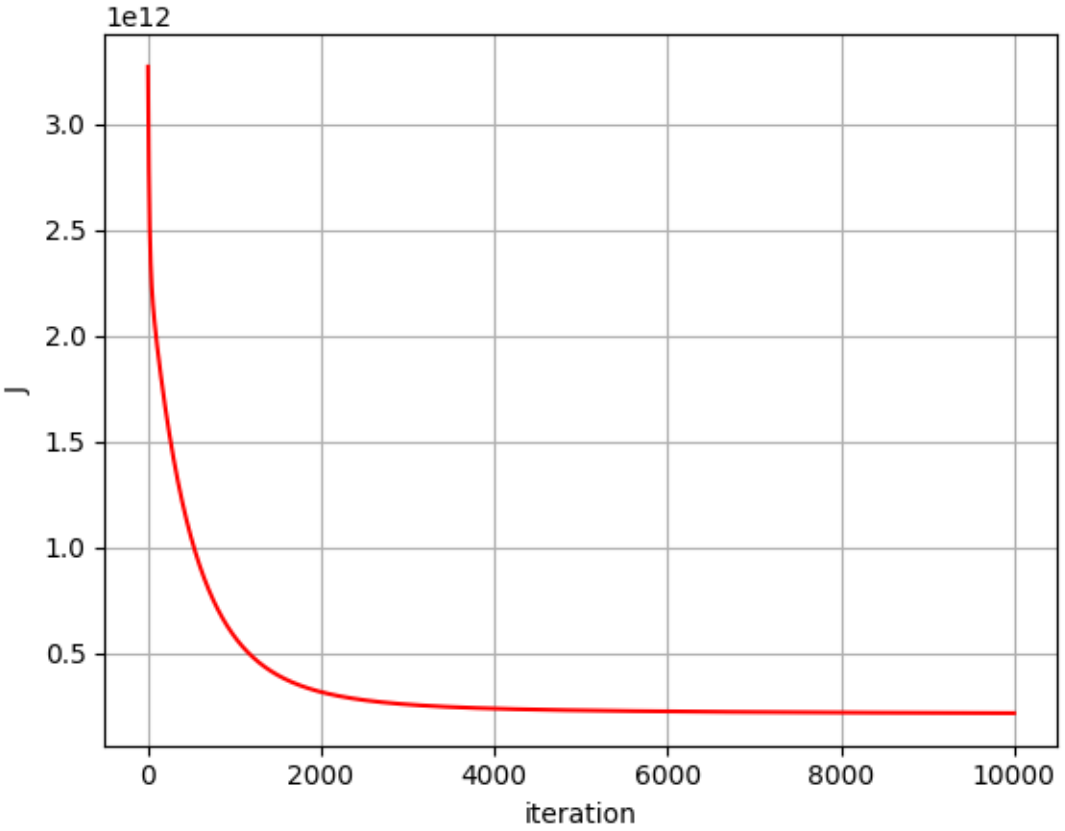
```
for iteration=0 value of J=3270134231514.8423
for iteration=500 value of J=1042401764829.5642
for iteration=1000 value of J=576969581590.1357
for iteration=1500 value of J=394960422276.3132
for iteration=2000 value of J=315670249117.3948
for iteration=2500 value of J=277248648962.26245
for iteration=3000 value of J=256748348906.66797
for iteration=3500 value of J=244852171291.56577
for iteration=4000 value of J=237417524154.06815
for iteration=4500 value of J=232447770193.28683
for iteration=5000 value of J=228916241254.83096
for iteration=5500 value of J=226268162382.61243
for iteration=6000 value of J=224191668089.96017
for iteration=6500 value of J=222505153331.83517
for iteration=7000 value of J=221099000266.80936
for iteration=7500 value of J=219904359705.00565
for iteration=8000 value of J=218876014838.71207
for iteration=8500 value of J=217982802653.32977
for iteration=9000 value of J=217202175289.7938
for iteration=9500 value of J=216517062044.53928
```

After 10000 iterations the value of R^2 we got is R^2= 0.9352435

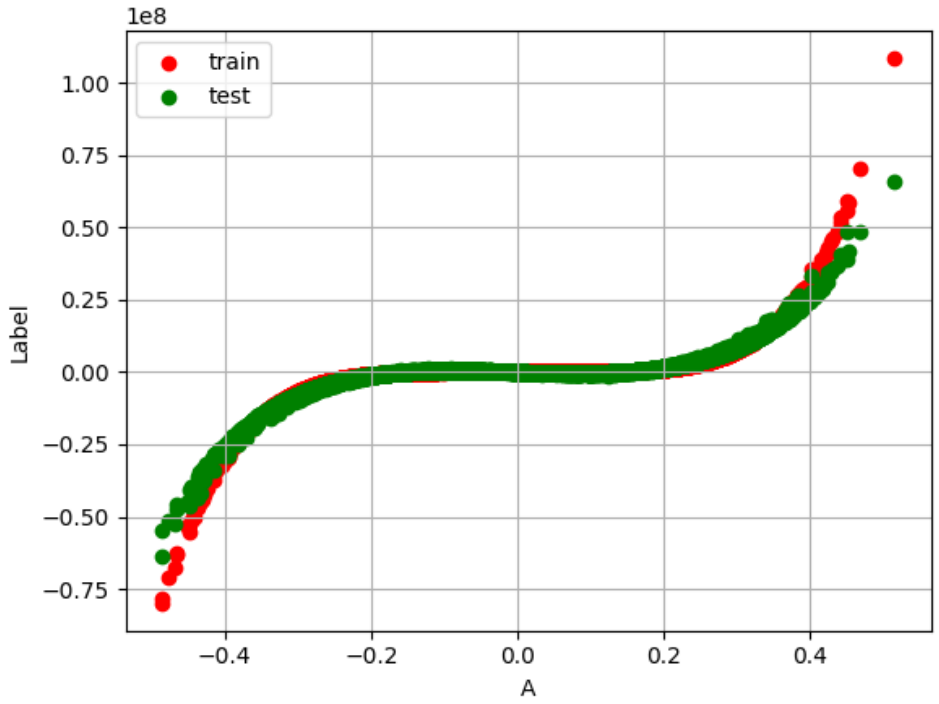
Hyperparameters: -

Train-cross validation-Test split fraction=80/20
Learning Rate=2
Optimization Algorithm=Gradient Descent
Degree of Polynomial=5
Cost function=Root mean square
Epochs=40000
Activation function=Polynomial
Regularisation constant=5

Training Visualisations: -



This represents how the cost function is decreasing with number of iterations in the gradient descent algorithm



This represents the relation between predicted label vs features A and train label and feature A

Logistics Regression: -

Idea and findings: -

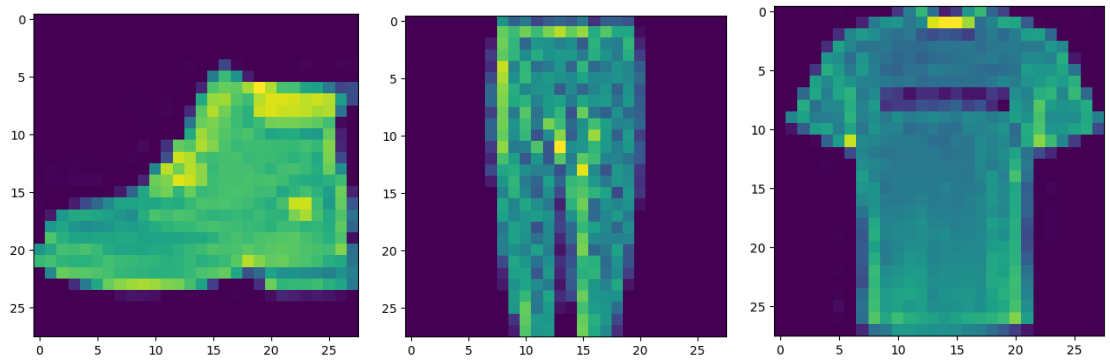
1)In the case of logistic regression, have used one vs many approach the one-vs-many method involves training a separate logistic regression model for each class, where the target variable is a binary variable indicating whether an observation belongs to the current class or not.

For example, let's say we have a classification problem with three classes: A, B, and C. We would create three separate logistic regression models:

1. Model 1: Class A vs (Class B and Class C)
2. Model 2: Class B vs (Class A and Class C)
3. Model 3: Class C vs (Class A and Class B)

Each model would be trained on the same set of features and their corresponding target variable, and would output a probability of an observation belonging to the positive class (i.e., Class A, Class B, or Class C). The class with the highest probability would be chosen as the predicted class for that observation.

2)secondly, I have used One hot encoding: -One hot encoding is a technique used to convert categorical variables, such as class labels, into a numerical format that can be processed by machine learning algorithms. In the context of multilabel classification, one hot encoding is used to represent each possible label as a binary vector. To perform one hot encoding, first, we create a list of all possible labels in the dataset. Then, for each instance or observation, we create a binary vector of the same length as the list of possible labels. In this vector, we set a 1 in the index corresponding to the label that applies to the instance and 0s in all other indices. For example, suppose we have a multilabel classification problem with three possible labels: "cat," "dog," and "bird." Suppose we have an observation with the labels "cat" and "bird." The one hot encoding for this observation would be [1, 0, 1], where the first and third indices correspond to "cat" and "bird," respectively, and the second index (corresponding to "dog") is 0. Once we have encoded all the labels in this way, we can use these binary vectors as target variables in our machine learning model to train it to predict the appropriate labels for new instances.



The above and many more data is given in the form of 28*28 Pixels

Training Log: -

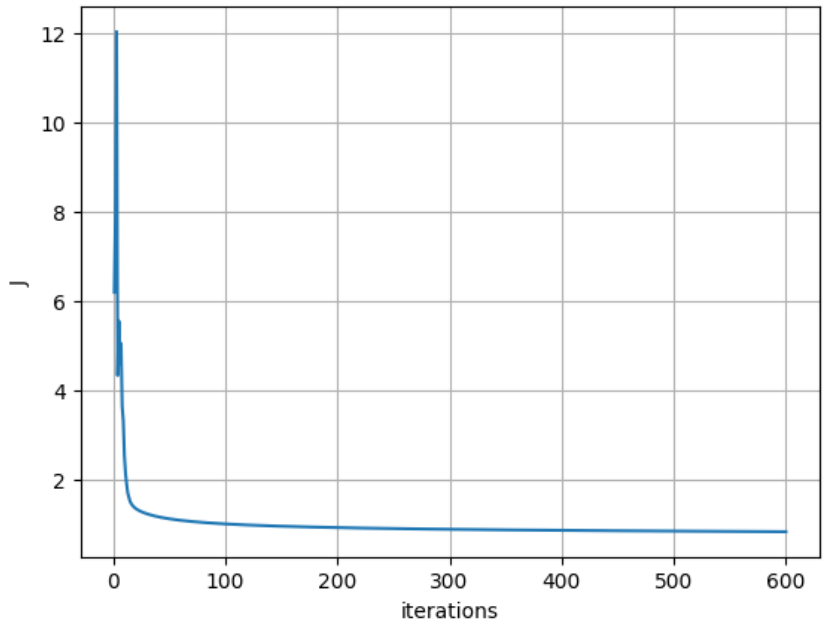
```
for iteration=1 value of J=6.191947977224742
for iteration=51 value of J=1.1221736043158403
for iteration=101 value of J=1.0102254991715292
for iteration=151 value of J=0.95946188512948
for iteration=201 value of J=0.9286066421021094
for iteration=251 value of J=0.9070979126346544
for iteration=301 value of J=0.8908639587531618
for iteration=351 value of J=0.8779602390951018
for iteration=401 value of J=0.8673248520942746
for iteration=451 value of J=0.8583217430897926
for iteration=501 value of J=0.8505431711704629
for iteration=551 value of J=0.8437137617159943
```

It is giving 84% accuracy at 20% test data set after training on 80% train data set

Hyperparameters: -

Train-test split ratio=80/20
Learning rate =1
Optimization algorithm=gradient descent
Activation function=sigmoid
Loss function=Binary Cross Entropy/Log Loss function
Epochs=600

Training Visualisations: -



This represents how the cost function is decreasing with number of iterations in the gradient descent algorithm

K-Nearest Neighbour: -

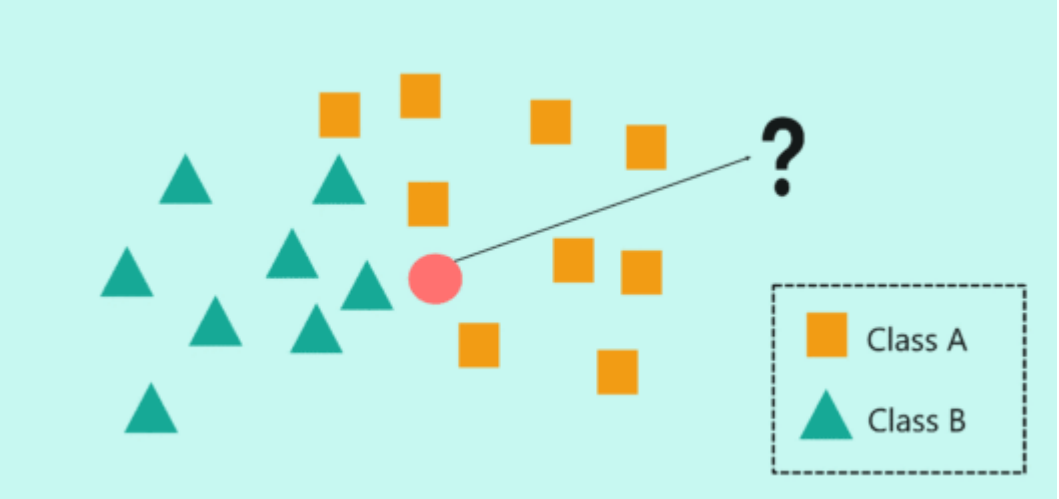
Idea: -

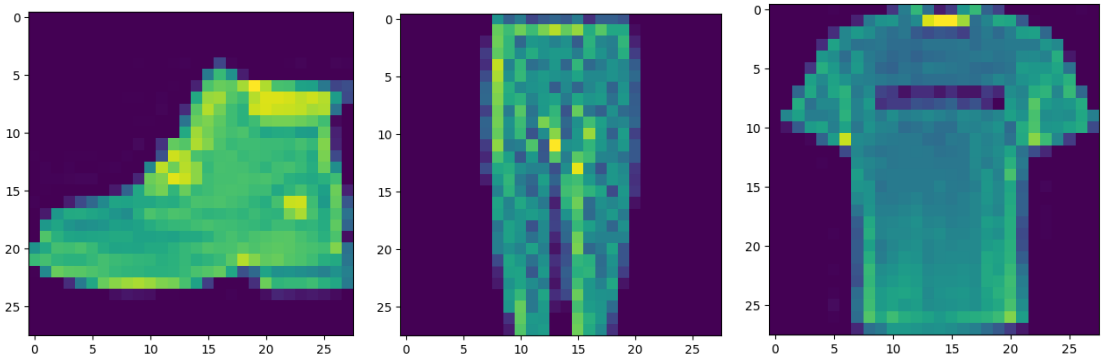
Normalised the data

To classify images using the k-nearest neighbour algorithm, I treated the pixels as points in space and calculated the Euclidean distance between a given pixel set and the rest. Then, I identified the K nearest neighbours and assigned the label of the most frequent class among them to the given image.

In general, for points given by Cartesian coordinates in n-dimensional Euclidean space, the distance is

$$d(p, q) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + \cdots + (p_n - q_n)^2}.$$





The above and many more data is given in the form of 28*28 Pixels

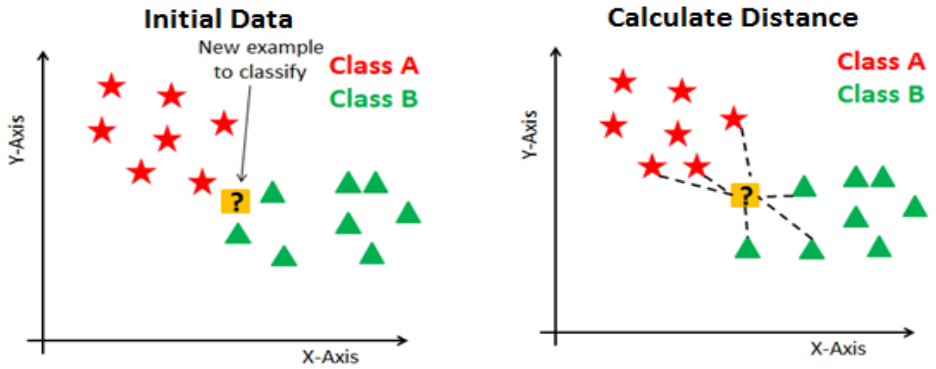
Training Log: -

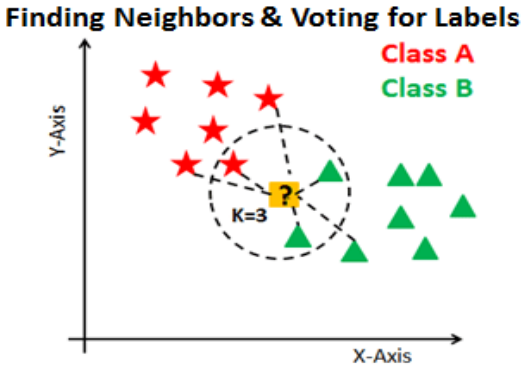
Accuracy on 20% test set is 81% after training on 80% train set

Hyperparameters: -

Test split ratio=80/20
#LazyLearner
K=$int(\sqrt{total\ Number\ of\ train\ example})$
If k is even:
K=k+1

Training Visualisations: -

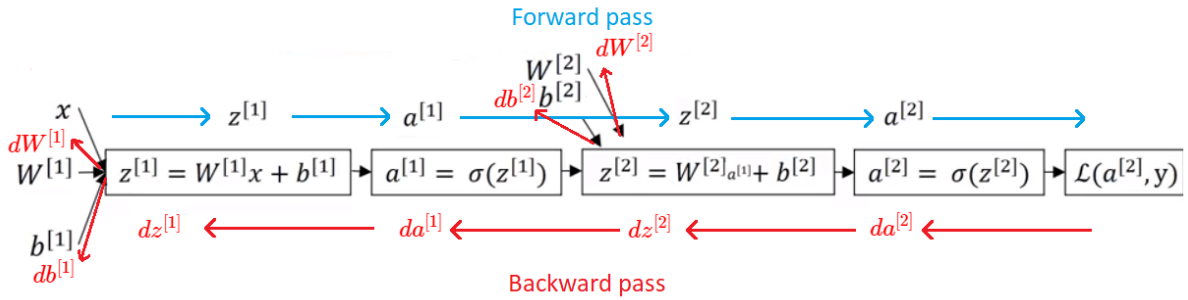


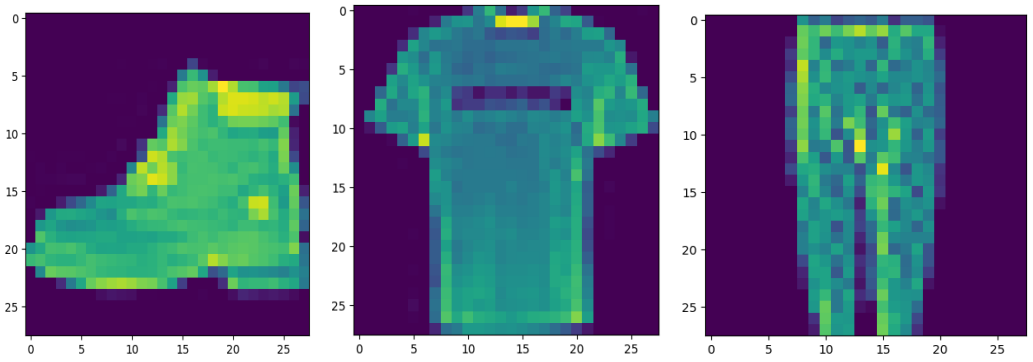


N Layer Neural Network: -

Idea: -

- Normally initialised the weights and used ReLu in the Hidden layers for preventing vanishing gradient problem
- used the forward propagation algorithm for obtaining output
- used backward propagation algorithm for updating weights and bias





The above and many more data is given in the form of 28*28 Pixels

Training Log: -

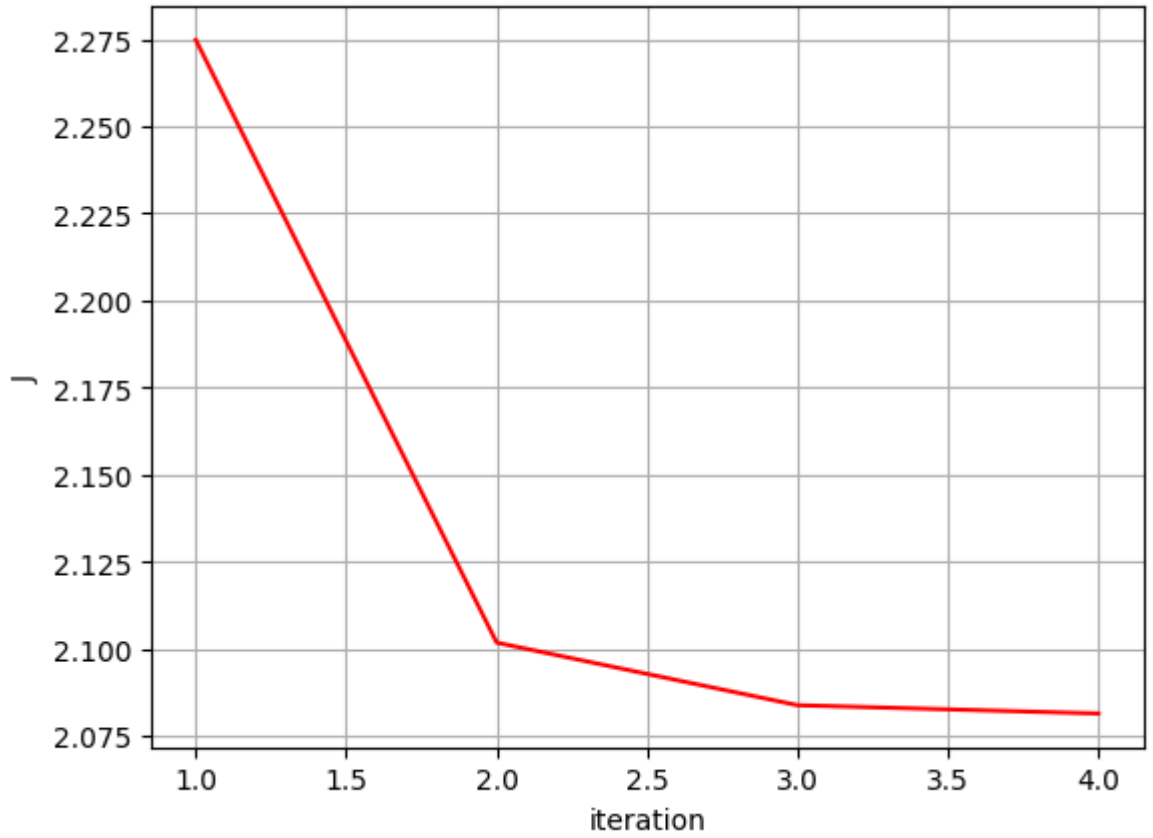
It is giving 84% accuracy at 20% test data set after training on 80% train data set

Hyperparameters: -

Train-test split ratio=80/20

Learning rate=0.01
optimization algorithm=gradient descent
activation function=ReLu in hidden layers and SoftMax in output layers
cost function=sparse cross entropy function
Number of hidden layers=3
With 1st hidden layer with 30 neural units
With 2nd hidden layer with 20 neural units
With 3rd hidden layer with 15 neural units
Output layer with 10 classifier
Epochs=7000

Training Visualisations: -



obtained by one hidden ReLu layer of 20 neurons

My model neural network model is not working properly will improve to model in the future