MACHINE LEARNING BOOTCAMP

Libraries used

- Numpy for mathematical calculations
- Pandas for reading data from csv files
- Matplotlib for plotting graphs
- Sklearn (Scikit Learn)

Data Initialization

MNIST train small and test data is used for this implementation. Train dataset has 20,000 samples, test dataset has 10,000 and they all have 1 outcome value and 784 features with each feature representing grayscale values of 28x28 pixel images of handwritten digits between 0 and 9.

First, I have imported both train data and test data using pandas and then split into X (2D array of samples and features) and y (Column array of corresponding outcomes).

This same data is used for all the algorithms below.

LINEAR REGRESSION

1. Algorithm

- Linear regression takes features and tries to find a straight line passing close to as many data points as possible and this line can be used to predict future values. Terms of higher powers can also be considered to get curves this increases accuracy but at the cost of increased training time.
- In this model Feature Scaling and Regularization is also implemented.
- Feature scaling is used to normalize the data so that all features contribute approximately proportionately and better accuracy can be achieved with fewer iterations
- Regularization is used to overcome the problem of overfitting (Good fitting, poor generalization) and underfitting (Poor fitting, poor generalization). Regularizations suppresses the weights of less important features.

2. Implementation on MNIST data.

Training the model

- First, I have applied feature scaling to each feature by standardizing the data.
- Then, experimented with different values for number of iterations, that is not too low (Resulting in less accuracy) and not too high (Resulting in more time).
- Then, I have calculated cost (Mean square error) and updated parameters by using gradient descent method with each iteration. And to check the proper working of gradient descent I have plotted Cost vs Number of iterations.
- After trying with different values of learning rate (Alpha), finally I have landed on a learning rate (0.05) such that it is not too low (Resulting in decreasing cost value but it was slow descent and requires more iterations) and not too high (Resulting in over shooting / increasing cost curve).
- In the same way, lambda value (0.01) is set in such a way that accuracy is high and cost value is low.

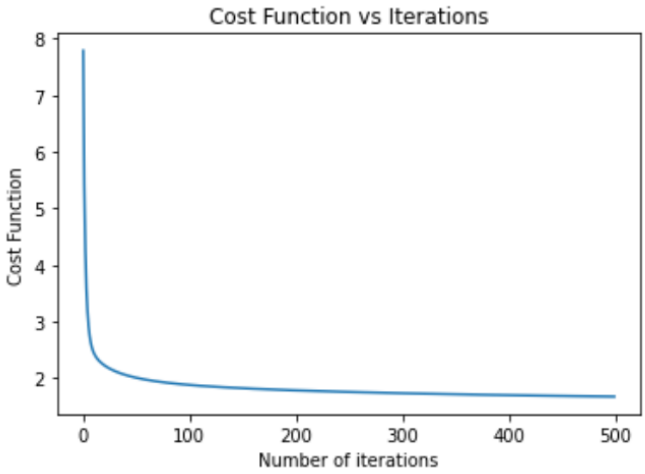
- Finally, using a loop I have tested different threshold values ranging from (-1 to 0.9 each with increment of 0.1) 0.9 gave me maximum accuracy.

Testing the model

- Then predictions are made for train data and test data by the trained model and the accuracy of the predictions is calculated.

3. Results

With **Learning Rate** (Alpha)=0.05, **lambda** (Regularization Constant) =0.001, and **number of iterations**=500



Time taken to train model: 12 sec

Accuracy of trained model on

Threshold value for maximum accuracy = 0.9

Train data: 24.64 %

Test data: 24.68 %

Accuracy compared with Scikit learn

Threshold value for maximum accuracy = 0.9

Train data: 24.16 %

Test data 24.14 %

4. My Learnings and Observations

- Vectorizing the calculation for finding cost and gradient made code run much faster compared to for-loops.
- Accuracy for linear regression on MNIST dataset is very low because linear function may not be sufficient to represent the data.

LOGISTIC REGRESSION

1. Algorithm

- Logistic regression is actually a classification algorithm and not regression algorithm.

- Logistic regression takes features and tries to find parameters such that these parameters separate one kind of data points from others data points (Binary Classification) and these parameters can be used to predict future values.
- This can be generalized for multiple class classification. Here we train separate models for each class.
- In Logistic Regression instead of using linear function directly (where output value can be any real number) we use sigmoid value of linear function (To restrict our output range from 0 to 1).
- Here mean square error cost function is not used, since we are using sigmoid function cost function may not be a smooth concave curve and can lead to local minima. So, logistic cost function is used.
- Here the output value represents the probability of our prediction.

  For binary classification

  So, our prediction is made as 1 if probability is greater then 0.5 and as 0 if probability is less than 0.5.

  For multiclass classification

  Prediction is made that it belongs to class which has highest probability.
- In this model Feature Scaling (Standardization) and Regularization is also implemented.

2. Implementation on MNIST data.

- Implementation is similar to logistic regression, only difference is we use sigmoid function and different cost function.

Training the model

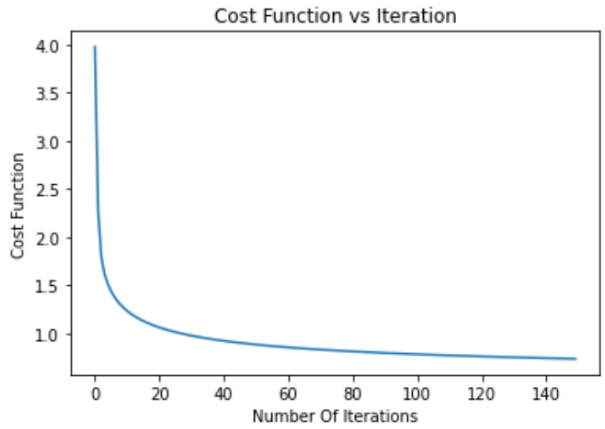
- Similar to linear regression feature scaling is applied to each feature by standardizing the data.
- Then, number of iterations is set such that is not too low (Resulting in less accuracy) and not too high (Resulting in more time).
- Then, I have calculated cost and updated parameters by using gradient descent method with each iteration. And to check the proper working of gradient descent I have plotted Cost vs Number of iterations.
- Similarly after trying with different values of learning rate (Alpha), finally I have landed on a learning rate (0.5) such that it is not too low (Resulting in decreasing cost value but it was slow descent and requires more iterations) and not too high (Resulting in over shooting / increasing cost curve).
- In the same way, lambda value (0.01) is set in such a way that accuracy is high and cost value is low.

Testing the model

- Then predictions are made for train data and test data by the trained model and the accuracy of the predictions is calculated.

3. Results

With **Learning Rate** (Alpha)=0.5, **lambda** (Regularization Constant) =0.01, and **number of iterations**=150



Time taken to train model: 35 sec

Accuracy of trained model on

Train data: 90.93 %

Test data: 91.08 %

Accuracy compared with Scikit learn

Train data: 97.18 %

Test data: 97.34 %

4. My Learnings and Observations

- Logistic regression is a binary classifier but we can use it for multiclass classification by training separate models for each class. Here we calculate the probability of each class and assign the value which has highest probability.
- Logistic regression has very high accuracy compared to linear regression for MNIST data set.

K-NEAREST NEIGHBORS (KNN)

1. Algorithm

- K-nearest neighbors is a classification algorithm. In this algorithm there no training of model, it directly predicts the outcome by comparing test data and finding similar data from train data.

2. Implementation on MNIST data.

- K-Nearest Neighbors algorithm takes a sample and finds k (Number of neighbors we want to considered) close samples from train data.
- First, I took a sample and calculated the distance (In this Euclidean Distance is used) of test sample from all samples of train data by broadcasting.
- Then I took k closest test samples and most repeated output of these k train samples is assigned it to the test sample.

- Then I have tried different **k** values like 5,10,500,2000… and finally chose 5. (Larger value of k reduces noise but if there is a more frequent sample then it may dominate and can bias the result)

3. Results

With **K Value** = 5

By this algorithm

Time taken to predict one sample = 0.06015 sec

Time taken to predict 20,000 samples = 20 mins 3 sec

Accuracy on test data with train data = 97.75 %

By Scikit learn algorithm

Time taken to predict one sample = 0.03685 sec

Time taken to predict 20,000 samples = 12 min 17 sec

Accuracy on test data with train data = 97.51 %

4. My Learnings and Observations

- As there is no training involved time taken by the algorithm to predict the outcomes is directly proportional to sample size of both train and test data. So, with lesser test samples it predicts faster than other algorithms and with large data it takes very long compared to other algorithms.
- K value largely affects the accuracy. Choosing proper k value is very important.

  First, I thought as there are 20,000 samples, I need larger k value like 100, 500 or maybe 1000 but the accuracy was low. Then I have changed to lower values like 5 and 10.
- If some samples are more frequent then the prediction is biased towards those outcomes and accuracy decreases.
- In this model Feature Scaling generally is not implemented. Feature scaling can be used when we want data so that all features contribute approximately proportionately.
- If the dataset contains a lot of features, then it may take very long, to overcome this we can decrease number of features with PCA (Principal Component Analysis).

NEURAL NETWORK

1. Algorithm

Artificial Neural Network is a classifier machine learning algorithm. In this model along with input and output layers it also contains some hidden layers in between them. Inputs activate nodes in hidden layers and these activated nodes gives output.

2. Implementation on MNIST data.

I had implemented Neural Network algorithm of two-layers and then modified it to multilayered.

- Here I have used sigmoid function for activation.

- First, I have applied feature scaling to the train and test dataset, converted y to One vs All matrix and initialized some random weights with standard deviation of 0.01
- And then with each iteration, performed forward propagation, calculated the cost value, and the average error associated with each node. Then with these error values, performed back propagation by gradient descent method and updated weights. I have used vectorized implementation for the entire FP, cost and BP processes.
- And finally tried different values for parameters (learning rate, lambda, number of iterations) similar to logistic regression.

Two Layered

- In this implementation I took a hidden layer (of size = 40 nodes).
- Implemented in two ways.
- 1) Batch gradient descent, in this I took entire data set for each iteration.
- 2) Mini-batch gradient descent, in this I took train data in small batches (After many trials, finally I took **500** samples for each iteration).

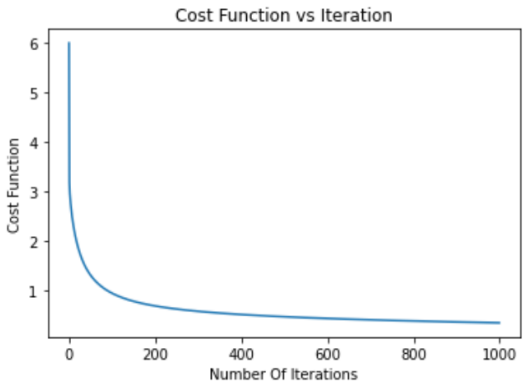
Multi Layered

- Multi layered neural network has more nodes and connections and it can separate more patterns. So, accuracy increases but requires more computational power.
- In this implementation I took four hidden layers (of sizes = 16,16,16,16 nodes)
- Implemented as mini-batch gradient descent.

3. Results

Two layered (Batch gradient descent)

With **Learning Rate** (Alpha)=0.5, **lambda** (Regularization Constant) =0.15, **number of iterations**=1000 and **hidden layer size** = 40



Time taken to train model: 3 min 56 sec

Accuracy of trained model on

Train data: 95.22 %

Test data: 95.28 %

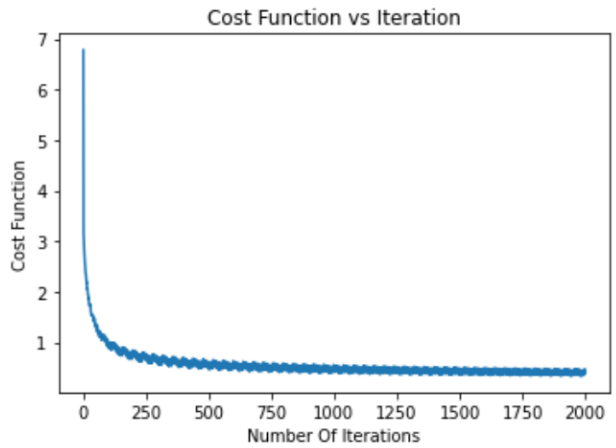
Accuracy compared with Scikit learn

Train data: 97.18 %

Test data: 97.34 %

Two layered (Mini Batch gradient descent)

With **Learning Rate** (Alpha)=0.5, **lambda** (Regularization Constant) =0.15, **number of iterations**=50, **mini batch size** = 500 and **hidden layer size** = 40



Time taken to train model: 13 sec

Accuracy of trained model on

Train data: 96.58 %

Test data: 96.69 %

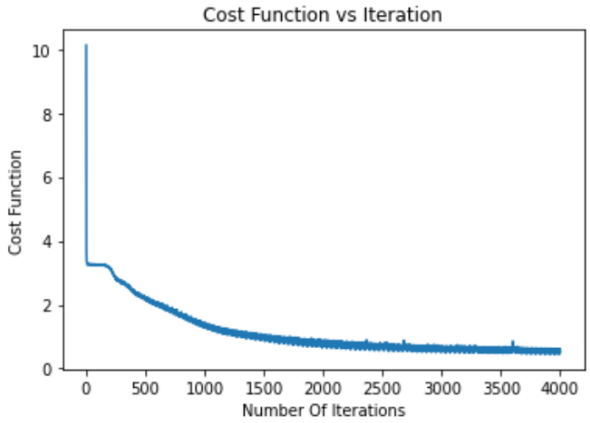
Accuracy compared with Scikit learn

Train data: 97.53 %

Test data: 97.79 %

Multi Layered

With **Learning Rate** (Alpha)=0.5, **lambda** (Regularization Constant) =0.15, **number of iterations**=100, **mini batch size** = 500 and **hidden layer size** = [16,16,16,16]



Time taken to train model: 25 sec

Accuracy of trained model on

Train data: 97.36 %

Test data: 97.75 %

Accuracy compared with Scikit learn

Train data: 97.53 %

Test data: 97.79 %

4. My learnings and Observations

- For activation, instead of sigmoid function, relu function can also be used and that gives much better results.
- **Mini batch gradient** descent has many advantages over batch gradient descent.
- 1) As we are taking small sets it requires less computational power.
- 2) It requires a smaller number of iterations. So, it is way faster, as we can see above it only took 13 sec whereas batch gradient method took 236 secs for almost similar accuracy.
- Disadvantages
- 1) We need to find the proper batch size that should not be too large (computationally expensive), and too small (less number of samples, cost function oscillates a lot and takes large number of iterations).
- 2) We may see some sharp spikes in the plot, these occur due to some batches in which one kind of sample dominates other kind of samples. Though it is temporary and doesn't have huge effect on training but still, we can minimize this to some extent by randomizing the train data.
- **Multi-Layer Neural Network**
- As there are more layers more complex relations can be formed and more patterns can be detected for better prediction.
- One use that sparked in mind is in case of logistic regression where higher order terms (To achieve complex polynomial which better fits train data) need to considered and there are already large number of features. We can use multi-layer neural network.
- Selecting proper architecture (number of layers and number of nodes in each layer) is important it largely effects accuracy and time for training.

Errors I encountered

During this implementation, I faced an error, where cost value decreased for few iterations and increased rapidly. First, I thought it was due to higher alpha, but there was no change even with very small alpha (0.00001). Then later I understood it was due to wrong back propagation.