Report on Machine Learning Algorithms

Libraries Used:

- NumPy – For numerical Calculations
- Matplotlib – For plotting the graphs
- Pandas – For reading data from CSV files
- Scikit-learn – To compare the results

Data Initialization

The datasets used are mnist_train_small.csv for training the Model and mnist_test.csv for testing the model. The training dataset consists of 20,000 samples whereas the testing dataset consists of 10,000 samples. Each sample has 784 features representing the grayscale values of 28 × 28 pixel images of handwritten digits ranging from 0 to 9; hence the outcome of each sample is a single-digit non-negative integer. First, I imported training and testing data using pandas and then split the data into **X** (2D array of samples and features), and **y** (Column array of corresponding outcomes).

Feature Scaling

Feature Scaling is implemented using **Standardization Technique**. It is used to normalize the features present in the data in a fixed range so that every feature will be on a comparable scale.

$$x_{scale} = \frac{x_i - x_{mean}}{x_{std}}$$

Few Parameters and their representation

$\mathbf{m}$ = Number of samples; $\mathbf{n}$ = Number of features; $\mathbf{c}$ = Number of different classes

$\boldsymbol{\lambda}$ = Regularization constant; $\boldsymbol{\alpha}$ = Learning Rate; $\boldsymbol{\beta}$ = L1 Ratio; $\mathbf{h}$ = Hidden layer size

|| Linear Regression

- It is a supervised Machine Learning algorithm that computes an appropriate linear relationship between the features and the outcome from the given data.

$$X = \{1, x_1, x_2, \dots x_n\} : \ x_i \ is \ the \ i^{th} feature \ value \ for \ i \ \in (1, n)$$

$$\hat{y} = \ \theta_0 + \sum_{i=1}^{n} \theta_i x_i = X.\theta = Hypothesis \ Equation$$

$$\hat{y} : Predicted \ Outcome$$

- Regularization is implemented using the **Elastic Net Technique**. It prevents overfitting by adding penalty terms to the Cost Function(J), discouraging the model from assigning too much importance to a particular feature. Regularizations suppress the weights of less important features.

$$Cost\ Function(J) = \frac{1}{2m}\sum_{j=1}^{m}\left|\hat{y}_j - y_j\right|^2 + \lambda\beta\sum_{i=1}^{n}|\theta_i| + \frac{\lambda(1-\beta)}{2}\sum_{i=1}^{n}{\theta_i}^2$$

$$Gradient\ Descent\ Function(d\theta_i) = \frac{\alpha}{m}\left(X^T(\hat{y} - y)\right)_i + \alpha\beta\lambda + \alpha\lambda(1-\beta)\theta_i$$

Implementation of Model on MNIST Dataset

Training the Model

1. Applied feature scaling to each feature present in the data by standardizing the data.

2. Conducted experiments to obtain the maximum accuracy. To check the proper working of the Gradient Descent Algorithm, I have plotted a graph between the Cost Function and the Number of iterations and it is observed that the drop in the cost value is very slow.

3. Finally, using a loop, I tested different threshold values ranging from (-1 to 0.9 each with an increment of 0.1).

Testing the Model

1. Predictions are made for training and testing data by the trained model and the accuracies are calculated. These accuracies are compared with the Scikit-learn model to check the performance.

Results

Time taken to train the model : ~20s.

Trained model:

Threshold Value for Maximum Accuracy = 0.8

Accuracy on Training Data = 18.69%

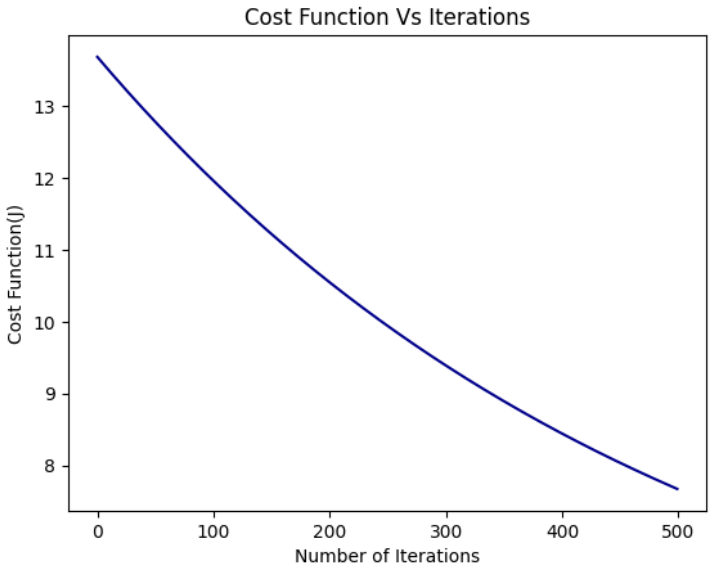
Accuracy on Testing Data = 18.40%

Scikit-learn model:

Threshold Value for Maximum Accuracy = 0.9

Accuracy on Training Data = 24.03%

Accuracy on Testing Data = 19.34%



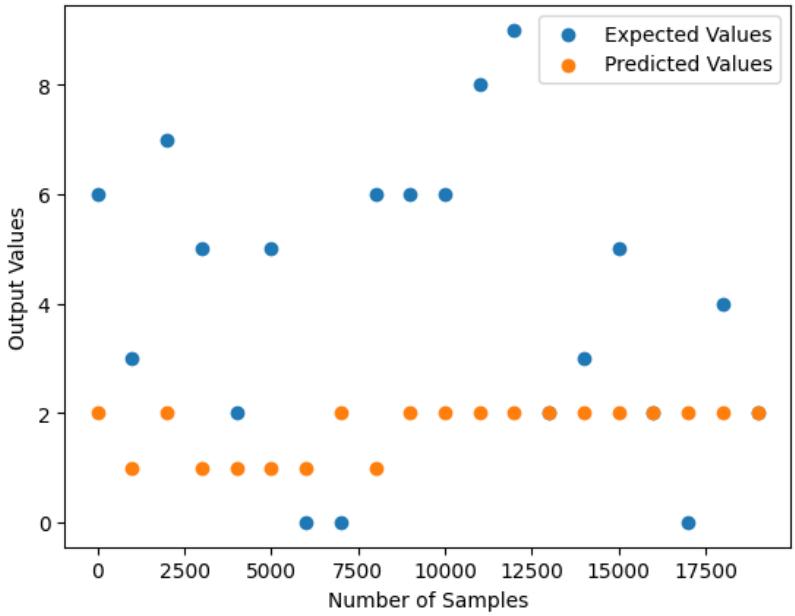
Observations

1. Vectorized calculations significantly improved the code's performance compared to using for loops.

2. During the experiments, I varied one parameter while keeping all other parameters of the model constant. Ultimately, this led to achieving the highest accuracy for the given parameter values. α = 0.001, λ = 0.001, β = 0.5, Number of iterations = 500. Achieved almost the same accuracy for different values of β = 0.1,0.2,0.4,0.6,0.8.

3. The accuracy is low because the linear function couldn't represent the data. It is a classification problem and, hence fails with the regression model.

4. Plotted a graph between the expected and predicted outcomes to visualize why the model's accuracy is low. I took one sample from every 1000 samples from the training dataset.



|| Logistic Regression

• It is a supervised machine learning algorithm that is used for the classification. Here the training is done for each of the different classes in the target variable.

$$X = \{1, x_1, x_2, \dots x_n\} : \; x_i \; is \; the \; i^{th} feature \; value \; for \; i \; \in (1, n)$$

$$w = \begin{bmatrix} w_{10} & \cdots & w_{1n} \\ \vdots & \ddots & \vdots \\ w_{c0} & \cdots & w_{cn} \end{bmatrix} \; ; \; Hypothesis \; Equation(h) = \frac{1}{1 + e^{-X.w^T}}$$

In the above '*w*' equation, each row represents the corresponding weights for a particular class in the target variable.

$yclass_{m \times c}$ is a matrix, where each column represents the expected outcome of a particular class.

$\widehat{yclass}_{m \times c}$ is a matrix, where each column represents the predicted outcome of a particular class.

• In logistic regression, instead of using linear function directly (where the output value can be any real number), we use the sigmoid value of linear function (to restrict the output range from 0 to 1).

• The cost function for logistic regression of a particular class (*i*) including regularization is given below.

$$Cost(J) = -\frac{1}{m}\left(\sum_{j=1}^{m}\left(yclass_i \log\left(\widehat{yclass_i}\right) + (1 - yclass_i)\log\left(1 - \widehat{yclass_i}\right)\right)\right) + \frac{\lambda}{2m}\sum_{k=1}^{n} {w_{ik}}^2$$

- Here the output value represents the probability of our prediction.

For Binary Classification:

Prediction is made as 1 if the probability is greater than 0.5, and 0 if it is less than 0.5.

For Multiclass Classification:

Here, the probability for all classes is calculated, and the class with more probability will be considered as the output for that particular sample.

Implementation of Model on MNIST Dataset

Training the Model

1. Applied feature scaling to each feature present in the data by standardizing the data.

2. The number of iterations is set such that the accuracy should be comparatively more and it should consume less time.

3. Conducted experiments to obtain the maximum accuracy. To check the proper working of the Gradient Descent Algorithm, I have plotted a graph between the Cost Function and the Number of iterations and it is observed that the drop in the cost value is very fast.

Testing the Model

1. Predictions are made for training and testing data by the trained model and the accuracies are calculated. These accuracies are compared with the Scikit-learn model to check the performance.

Results

Time taken to train the model : ~50s.

Trained model:

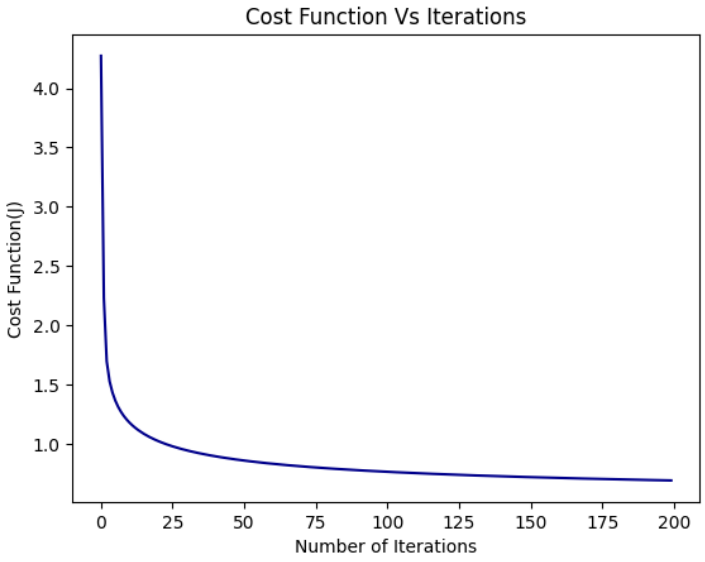
Accuracy on Training Data = 91.545%

Accuracy on Testing Data = 90.05%

Scikit-learn model:

Accuracy on Training Data = 97.175%

Accuracy on Testing Data = 88.3%

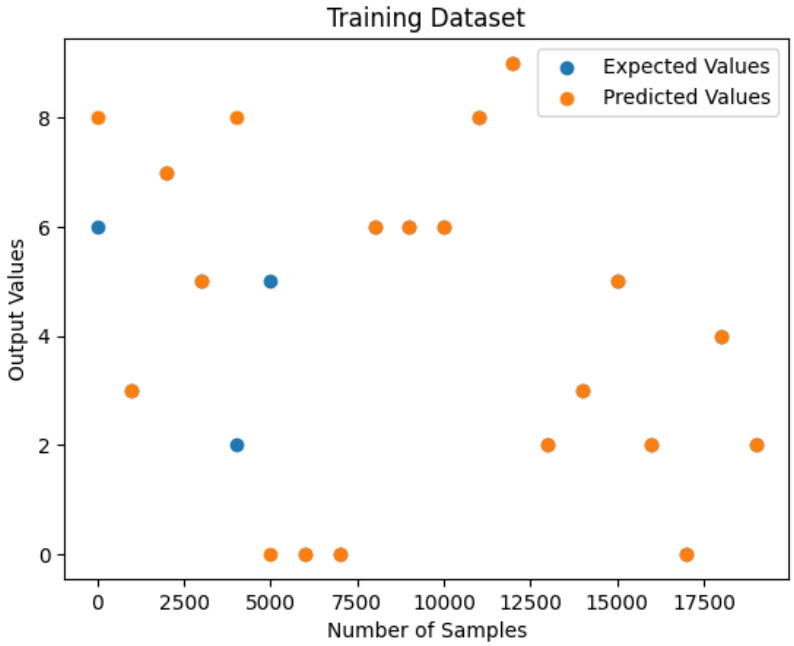


Obtained model parameters: α = 0.6, λ = 0.01, Number of iterations = 200

Observations

1. Logistic Regression is a binary classifier. For multiclass classification, it is used for all different classes separately, and binary classification is applied for each class. The class with a higher probability for a particular sample will be considered as the output.

2. Logistic Regression has more accuracy than Linear Regression; this can be visualized by plotting a scatter plot for one sample out of 1000 samples in the dataset. The plot below shows that most of the points have the same expected and predicted values.



|| K-Nearest Neighbours (KNN)

• It is a supervised machine learning algorithm employed to tackle classification and regression problems. There is no training of the model, it directly predicts the outcome by comparing test data and finding similar data points from the training data.

• Let's say the test point is x_t, the model predicts the outcome by taking the '*k*' nearest points from the training data to the test data point, and the result will be the most occurred value among those '*k*' points.

• The distance metric used in this model is Euclidean Distance.

$$Distance(x_1, x_2) = \sqrt{\sum_{i=1}^{n} (x_{1i} - x_{2i})^2} \; ; \; x_{1i} \; is \; the \; i^{th} feature \; value \; of \; x_1$$

• The value of the 'k' is chosen such that, if input data has more outliers, the higher value of 'k' is recommended. Choosing the odd 'k' is also recommended to avoid ties for classification.

Implementation of Model on MNIST Dataset

For choosing ‘k’, I’ve tried different values of k = 5,11,101,501. Finally, k = 5 is chosen.

Results

My model:

Time taken to predict the outcome for 10,000 samples : ~20m.

Time taken to predict the outcome of one sample : ~0.12s.

Accuracy on Test Data with Train Data = 95.97%

Scikit-learn Model:

Time taken to predict the outcome for 10,000 samples : ~10m.

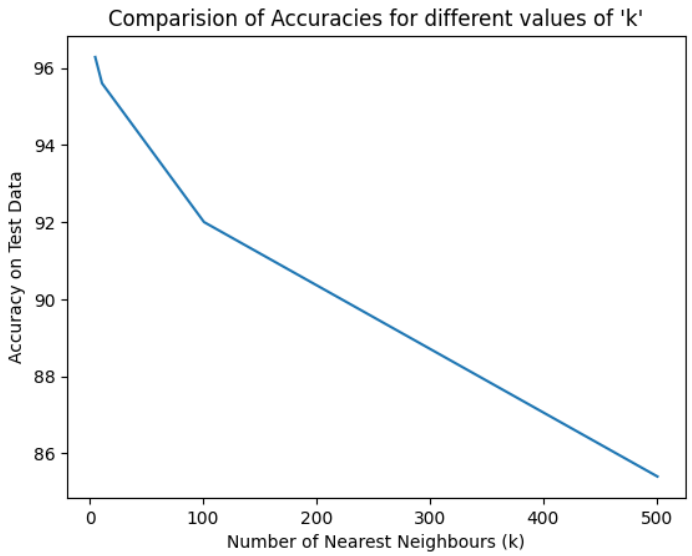
Time taken to predict the outcome of one sample : ~0.06s.

Accuracy on Test Data with Train Data = 95.91%

Observations

1. The time taken to predict the outcome is directly proportional to the number of samples. The time taken to execute this algorithm is relatively very high as compared to other models.

2. The ‘k’ value largely affects the accuracy. The given plot shows how the accuracy is reduced, as the value of ‘k’ increases.



3. If some samples are more frequent, then the prediction is biased towards those outcomes and accuracy decreases.

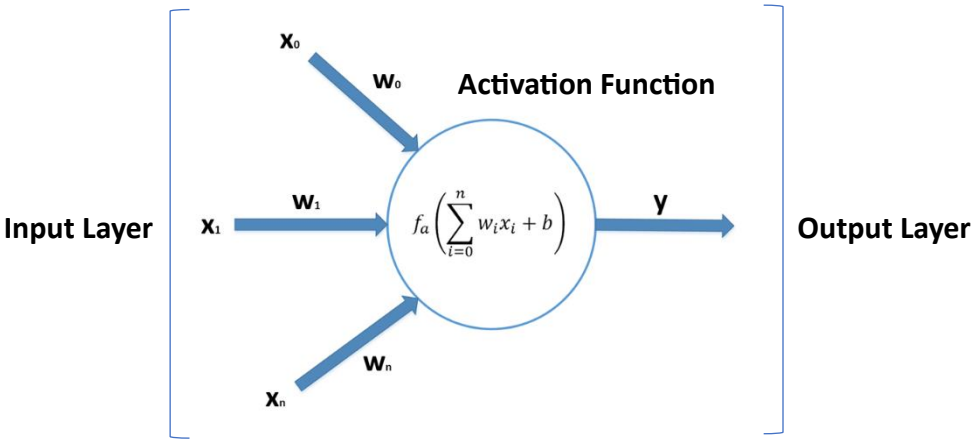
4. In this model, feature scaling is not implemented. Feature scaling can be used when we want data such that all features contribute approximately proportionately.

5. If the dataset contains a lot of features, then it may take very long, to overcome this, we can decrease the number of features with PCA (Principal Component Analysis).

|| Neural Networks

• An Artificial Neural Network is a directed structure that connects an input layer with the output layer through a few hidden layers. Inputs activate the nodes present in the hidden layers, and these activated nodes process the output.

• A pictorial view of the Neural Network is shown below.



• In this model, three types of activation functions such as Sigmoid, Tanh, and ReLU were used.

• Xavier Initialization of weights is applied for the Tanh activation function, H-et-al Initialization for other activation functions.

>> Two-Layered Neural Network

• In this implementation, hidden layer of size 40(i.e., 40 nodes) is considered.

• It is implemented in two ways.

– Batch Gradient Descent: Entire data is considered for iteration.

– Mini Batch Gradient: Entire data is split into 40(i.e., $\frac{m}{500}$) batches, containing 500 samples.

Forward Propagation

$$X = \{1, x_1, x_2, \dots x_n\} : \ x_i \ is \ the \ i^{th} feature \ value \ for \ i \ \in (1, n)$$

$yclass_{m\times c}$ is a matrix, where each column represents the expected outcome of a particular class.

$\widehat{yclass}_{m\times c}$ is a matrix, where each column represents a particular class's predicted outcome (output of the last layer).

$$\vartheta_1 = \begin{pmatrix} \theta_{10} & \cdots & \theta_{1n} \\ \vdots & \ddots & \vdots \\ \theta_{h0} & \cdots & \theta_{hn} \end{pmatrix}$$

ϑ_1 is the weights matrix corresponding to the first layer, $X.{\vartheta_1}^T$ is the input to the 1st Hidden layer.

$$\vartheta_2 = \mathrm{T}\begin{pmatrix} \theta_{11} & \cdots & \theta_{1(h+1)} \\ \vdots & \ddots & \vdots \\ \theta_{c1} & \cdots & \theta_{c(h+1)} \end{pmatrix} : \ \mathrm{T} \text{ is initialization value.}$$

$\mathrm{T} = \sqrt{\frac{1}{h}}$: Xavier Initialization, $\mathrm{T} = \sqrt{\frac{2}{h}}$: H-et-al Initialization

ϑ_2 is the weights matrix corresponding to the second layer. After applying activation to the output of the first hidden layer and biasing that output gives the input for the output layer, its matrix multiplication with ${\vartheta_2}^{\mathrm{T}}$ and applying activation gives $\widehat{yclass}_{m\times c}$.

• The cost function of an i^{th} class is similar to the logistic regression cost function, here a new penalty corresponding to the second layer will be added, 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 FP and BP processes.

Implementation of Model on MNIST Dataset

Results

Model Parameters obtained: α = 0.5, λ = 0.15

Activation Function	Batch Gradient (500)		Mini Batch Gradient (50)		Scikit-learn	
	Training Accuracy	Testing Accuracy	Training Accuracy	Testing Accuracy	Training Accuracy	Testing Accuracy
Sigmoid	93.29%	91.4%	96.64%	92.54%	99.99%	92.38%
Tanh	87.78%	86.75%	91.54%	89.05%	100%	92.16%
ReLU	95.53%	91.93%	97.67%	92.55%	100%	94.23%

Cost Function vs Iterations plot is shown in the Notebook.

The number of iterations taken is shown in the brackets.

>> Multi-Layered Neural Network

• Multi-layered neural network has more nodes and connections. Therefore, it can separate more patterns. So, accuracy increases but requires more computational power.

• In this implementation, four hidden layers of size 16 (i.e., each hidden layer has 16 nodes) are considered. It is implemented using mini-batch gradient descent.

Implementation of Model on MNIST Dataset

Results

Model Parameters obtained: α = 0.5, λ = 0.15, Number of iterations = 50

Activation Function	Mini Batch Gradient		Scikit-learn	
	Training Accuracy	Testing Accuracy	Training Accuracy	Testing Accuracy
Sigmoid	95.81%	89.98%	99.01%	84.15%
Tanh	97.85%	90.33%	99.83%	88.65%
ReLU	9.81%	9.8%	100%	94.31%

Cost Function vs Iterations plot is shown in the Notebook.

Observations

1. Two-Layered Neural Network

- Mini-batch gradient descent offers several advantages over batch gradient descent, primarily because it processes smaller subsets of data at a time. This results in reduced computational requirements. The results indicate that mini-batch gradient descent executes significantly faster than batch gradient descent.
- Mini batch gradient descent has some disadvantages as well. We need to find the proper batch size that should not be too large (computationally expensive) and not too small (cost function oscillates a lot and takes a large number of iterations).
- We might observe occasional sharp spikes in the plot, which happen when certain batches are dominated by one type of sample. Although these spikes are temporary and don't significantly impact the overall training process, we can mitigate them somewhat by shuffling the training data.

2. Multi-Layered Neural Network

- As there are more layers, more complex relations can be formed and more patterns can be detected for better prediction.
- Selecting the proper architecture (number of layers and number of nodes in each layer) is important it largely affects accuracy and time for training.
- The ReLU activation function produced very poor results, and the cost was so high, it could not able to find the optimal weights.