Group 4

Machine Learning Bootcamp

Project Report

Alok Raj

ADM. NO. 22JE0091

BRANCH: COMPUTER SCIENCE AND ENGINEERING

IIT ISM DHANBAD

Introduction:

In this project report, we will discuss the implementation and performance evaluation of five machine learning algorithms: Linear Regression, Polynomial Regression, Logistic Regression, K-Nearest Neighbours (K-NN), and Neural Networks. These algorithms will be applied to different datasets and their performance will be compared based on various metrics.

Tech Stack:

Google Colab Notebook, Jupiter Notebook, .csv files, .npy files

Libraries used:

NumPy, Pandas, Matplotlib.pyplot, cv2

Linear Regression

Linear regression is a regression algorithm used to model the relationship between a dependent variable and one or more independent variables. The goal is to find the linear relationship that best describes the data, which is achieved by fitting a straight line through the data points that minimizes the sum of the squared errors between the observed and predicted values.

In simple linear regression, there is only one independent variable, and the equation of the line can be represented as

$$y = mx + c$$

In multi-variate linear regression, there are multiple independent variables, and the equation of the line is represented as

$$y = m_0 + m_1 x_1 + \dots + m_n x_n$$

where y is the dependent variable or hypothesis, x_i are the independent variables, and m_i are the coefficients.

In matrix form:

$$Y = M^T . X$$

Y is the matrix of the batch of outputs, M is the matrix denoting the weights and biases, and X is the batch of all independent variables.

Standardization:

Standardization is a data pre-processing technique used to transform data in such a way that it has zero mean and unit variance. The goal of standardization is to make the data comparable.

Feature Standardization:

$$x^i := \frac{(x^i - \bar{x}^i)}{\sigma}$$

, where x^i is the i^{th} feature, $\bar{x}^i$ is the mean of all i^{th} features and σ is the standard deviation of all i^{th} features.

Cost Function:

Cost function used is the Mean Squared Error (MSE).

It calculates the average squared difference between the predicted and actual values.

The formula for MSE is:

$$J = (1/2b) \; * \; \sum_{i=1}^{b}(y^i \; - \; \hat{y}^i)^2$$

Where J is the loss, b is the number of data points in the dataset, y^i is the actual value, and $\hat{y}^i$ is the predicted value.

Batch Gradient Descent Algorithm:

Batch Gradient Descent (BGD) is an optimization algorithm used to minimize the cost or loss function in machine learning. It is a type of gradient descent that computes the gradient of the cost function over the entire training dataset to update the model's parameters

Weights are updated as:

$$m_j := m_j + \propto \frac{\partial J}{\partial m_j}$$

, where $\propto$ is the learning rate, m_j is the j^{th} weight

$$\frac{\partial J}{\partial m_j} = (1/b) \; * \sum_{i=1}^{b} \left(y^i \; - \; \hat{y}^i\right). x_j^i$$

In matrix form:

$$M := M \; + \frac{\propto}{b} \, (Y \; - \; \hat{Y}) \, . \; X$$

R2 Score:

The R-squared score, also known as the coefficient of determination, is a statistical measure that represents the proportion of the variance in the dependent variable that is predictable from the independent variable(s). In other words, it measures the goodness of fit of a regression model to the data.

Mathematically:

$$R^2 \, score \; = \; 1 \; - \; \frac{SS_{res}}{SS_{tot}}$$

$$SS_{res} \; = \; \sum_{i=1}^{b}(y^i \; - \; \hat{y}^i)^2$$

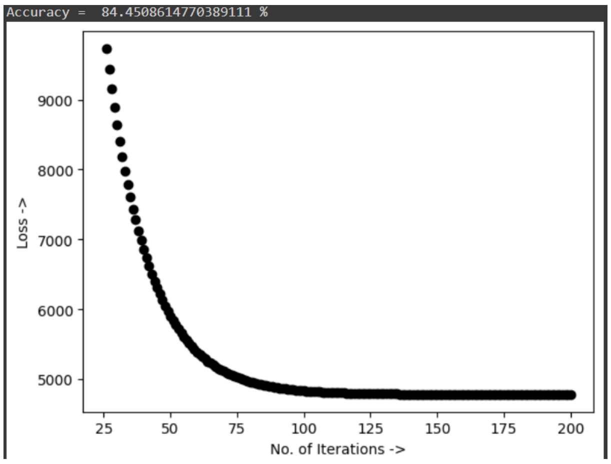
$$SS_{tot} \; = \; \sum_{i=1}^{b}(y^i \; - \; \bar{y})^2$$

, where y^i is the actual value, $\bar{y}$ is the mean of all y^i, and $\hat{y}^i$ is the predicted value.

Result:

After 500 iterations of the code, the R2 score came out to be 84.45% and the loss converged to approximately 4779.97 ,with a learning rate of 0.03.

Accuracy = 84.4508614770389111 %



The predicted output is stored in a csv file named “linear_regression_result.csv”

Polynomial Regression

Polynomial regression is an extension of Linear Regression that allows for modelling non-linear relationships between the outcome variable and predictor variables by modelling it as an nth-degree polynomial.

In univariate polynomial regression, there is only one independent variable, and the equation can be represented as

$$y = m_0 + m_1x_1 + m_2x_1^2 + \cdots + m_nx_1^n$$

In multivariate polynomial regression:

$$y = m_0 + m_1x_1 + m_2x_1^2 + \cdots + m_nx_1^n$$

$$+ m_{n+1}x_2 + m_{n+2}x_2^2 + \cdots + m_{2n}x_2^n + \ldots$$

, where $x_1, x_2, \ldots$ are the independent variables and m_i is the weight or coefficient.

In Matrix form:

$$Y = M^T.X$$

Y is the matrix of the batch of outputs, M is the matrix denoting the weights and bias, and X is the batch of all independent variables and their different combinations.

Standardization:

Standardization is a data pre-processing technique used to transform data in such a way that it has zero mean and unit variance. The goal of standardization is to make the data comparable.

Feature Standardization:

$$x^i := \frac{(x^i - \bar{x}^i)}{\sigma}$$

, where x^i is the i^{th} feature, $\bar{x}^i$ is the mean of all i^{th} features and σ is the standard deviation of all i^{th} features.

Label Standardization:

Reduces the no. of iterations required for convergence.

$$y := \frac{(y - \bar{y})}{\sigma}$$

, where y is the label, $\bar{y}$ is the mean of all labels and σ is the standard deviation of all labels.

Cost Function:

Cost function used is the Mean Squared Error (MSE).

It calculates the average squared difference between the predicted and actual values.

The formula for MSE is:

$$J = (1/2b) \ * \ \sum_{i=1}^{b}(y^i \ - \ \hat{y}^i)^2$$

Where J is the loss, b is the number of data points in the dataset, y^i is the actual value, and $\hat{y}^i$ is the predicted value.

Batch Gradient Descent Algorithm:

Batch Gradient Descent (BGD) is an optimization algorithm used to minimize the cost or loss function in machine learning. It is a type of gradient descent that computes the gradient of the cost function over the entire training dataset to update the model's parameters

Weights are updated as:

$$m_j := \ m_j + \propto \frac{\partial J}{\partial m_j}$$

, where $\propto$ is the learning rate, m_j is the j^{th} weight

$$\frac{\partial J}{\partial m_j} \ = \ \frac{1}{b}\sum_{i=1}^{b} \ (y^i \ - \ \hat{y}^i).x_j^i$$

In matrix form:

$$M \ := \ M \ + \frac{\propto}{b} \ (Y \ - \ \hat{Y}) \ . \ X$$

$$M \ := \ M \ + \frac{\propto}{b} \ (Y \ - \ \hat{Y}) \ . \ X$$

R2 Score:

The R-squared score, also known as the coefficient of determination, is a statistical measure that represents the proportion of the variance in the dependent variable that is predictable from the independent variable(s). In other words, it measures the goodness of fit of a regression model to the data.

Mathematically:

$$R^2 \, score \ = \ 1 \ - \ \frac{SS_{res}}{SS_{tot}}$$

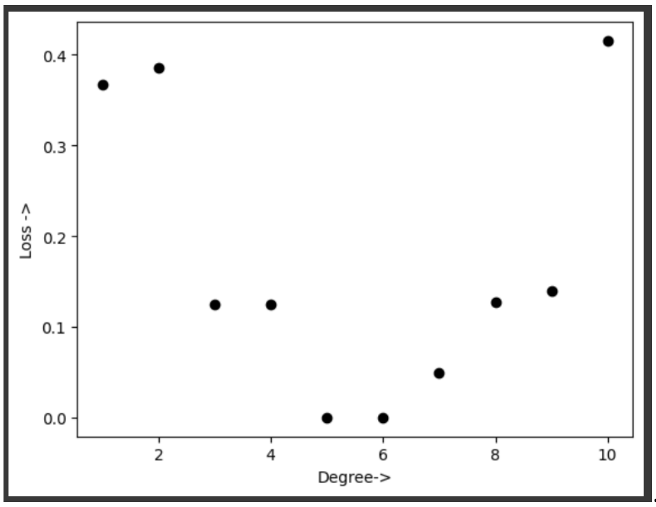
$$SS_{res} \ = \ \sum_{i=1}^{b}(y^i \ - \ \hat{y}^i)^2$$

$$SS_{tot} \ = \ \sum_{i=1}^{b}(y^i \ - \ \bar{y})^2$$

, where y^i is the actual value, $\bar{y}$ is the mean of all y^i, and $\hat{y}^i$ is the predicted value.

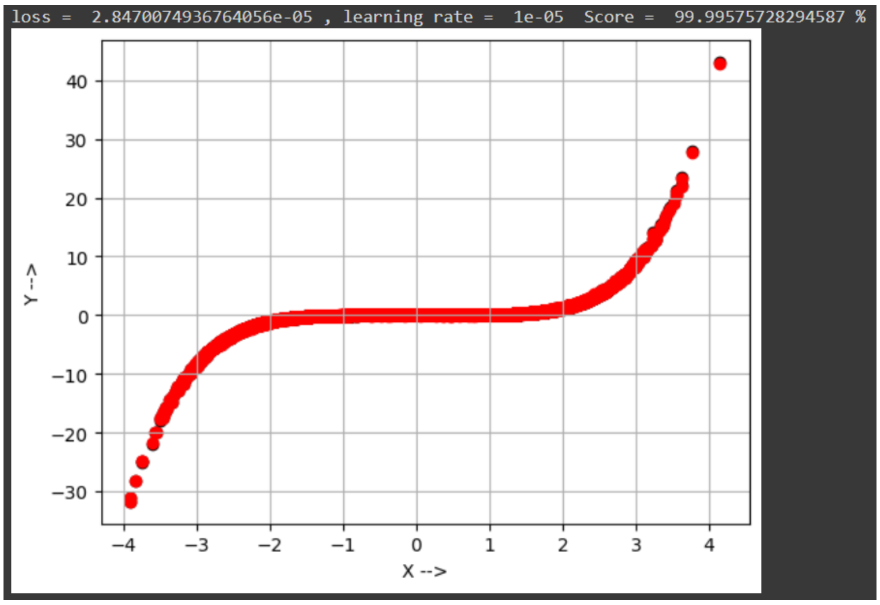
Result:

The most optimum degree was found out to be 5



The R2 score of the model exceeded 99.995% after 10000 iterations and reached almost 100% after 1000000 iterations with a learning rate of 1e-5.

The time taken to train model for 10000 iterations was 37 seconds.



```
loss =  2.8470074936764056e-05 , learning rate =  1e-05  Score =  99.99575728294587 %
```

The predicted labels are stored in a .csv file named "Polynomial_Regression_result.csv".

Logistic Regression

Logistic regression is an algorithm used for classification problems.

SoftMax Regression:

SoftMax Regression is an algorithm used for multiclass classification problems by using the softmax function to generate probability distributions of each class or label.

$$For\ \hat{y}^i = m^T.x^i$$

, where $\hat{y}^i$ denotes the predicted output of i^{th} input, m is the matrix of weights for each label and x^i is the matrix of input features.

$$SoftMax(y^i) = \frac{e^{y^i}}{\sum e^{y^i}}$$

The softmax function exponentiates the predictions, making any negative values positive, and then divides it by the sum of all exponentiated labels, thus generating a probability distribution over all labels.

Normalization:

Normalization is a process of scaling the data so that it falls within a specified range, typically between 0 and 1.

Upon analysing the labelled data, the noticeable things were that the 784 features were pixel values ranging from 0 to 255. Upon plotting the 784 pixels into a grid of 28 × 28 pixels by using the pixel values as grayscale values, different wearable items such as clothes, shoes, bags, etc.

To normalize the data, we can divide each feature by the maximum of all values i.e., 255

$$x := \frac{x}{255}$$

Cost Function:

The cost function used here is the Cross-Entropy Loss.

Cross entropy loss is a measure of the distance between the predicted probability distribution and the true probability distribution. The loss is calculated as the negative log likelihood of the true class labels, given the predicted class probabilities.

$$Cross\ Entropy\ Loss = J = -\sum y^i . \log(\hat{y}^i\ ) + (1 - y^i\ ). \log(1 - \hat{y}^i)$$

, where y^i is the One-Hot Encoded matrix form of the true label of the i^{th} sample and $\hat{y}^i$ is the matrix of probability distributions for each label of the i^{th} sample.

Mini-Batch Gradient Descent Algorithm:

Mini-Batch Gradient Descent is a variant of the Gradient Descent Algorithm in which instead of computing the gradient on the entire dataset, the dataset is divided into smaller subsets called mini-batches. The gradient is computed on each mini-batch separately, and the model parameters are updated based on the each gradient across all the batches. This process is repeated for a number of iterations, where each iteration consists of one pass through the entire dataset.

Weights are updated as:

$$m_j := m_j + \propto \frac{\partial J}{\partial m_j}$$

, where $\propto$ is the learning rate, m_j is the j^{th} weight

$$\hat{y}^i = m^T.x^i$$

$$\hat{y}^i := softmax(\hat{y}^i)$$

$$\frac{\partial J}{\partial m_j} = \frac{1}{b}\sum_{i=1}^{b} \left(y^i - \hat{y}^i\right).x_j^i$$

In matrix form:

$$M := M + \frac{\propto}{b}(Y - \hat{Y}) . X$$

Accuracy:

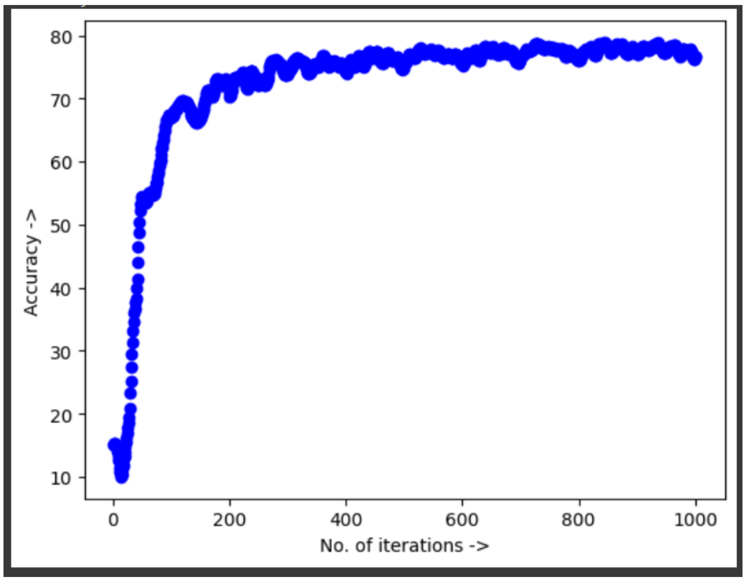
The accuracy is measured by direct comparison of predicted labels of the labelled test features and test labels.

$$Accuracy = \left(\frac{No. of\ matched\ cases}{Total\ cases} \times 100 \right)\%$$

Result:

The Accuracy of the model converged at around 82.98% after 1000 iterations with a learning rate of 1 and 100 mini-batches.

Time taken by the model to run up to 1000 iterations was around 5 mins.



The predicted labels are stored in a .csv file named "Logistic_Regression_Result.csv".

K-Nearest Neighbours

K-nearest neighbours (K-NN) is a supervised learning algorithm used in machine learning for classification and regression tasks.

The K-NN algorithm works by finding the K closest data points (i.e., nearest neighbours) to the new data point that needs to be classified or predicted. The value of K is a hyperparameter. Once the K nearest neighbours are identified, the algorithm takes a majority vote for classification tasks or a mean for regression tasks to predict the label or value of the new data point.

Normalization:

Normalization is a process of scaling the data so that it falls within a specified range, typically between 0 and 1.

Upon analysing the labelled data, the noticeable things were that the 784 features were pixel values ranging from 0 to 255. Upon plotting the 784 pixels into a grid of 28 × 28 pixels by using the pixel values as grayscale values, different wearable items such as clothes, shoes, bags, etc.

To normalize the data, we can divide each feature by the maximum of all values i.e., 255

$$x := \frac{x}{255}$$

Distance:

The distance used for the algorithm is the square of the Euclidean distance to reduce computational cost.

$$Distance = d = (x - a)^2$$

, where x and a are values of the same feature.

For all features:

$$d = \sum \left(x^i - \hat{x}^i\right)^2$$

, where x^i and $\hat{x}^i$ are values of the same feature.

Loss:

Cost function used is the Mean Squared Error (MSE).

It calculates the average squared difference between the predicted and actual values.

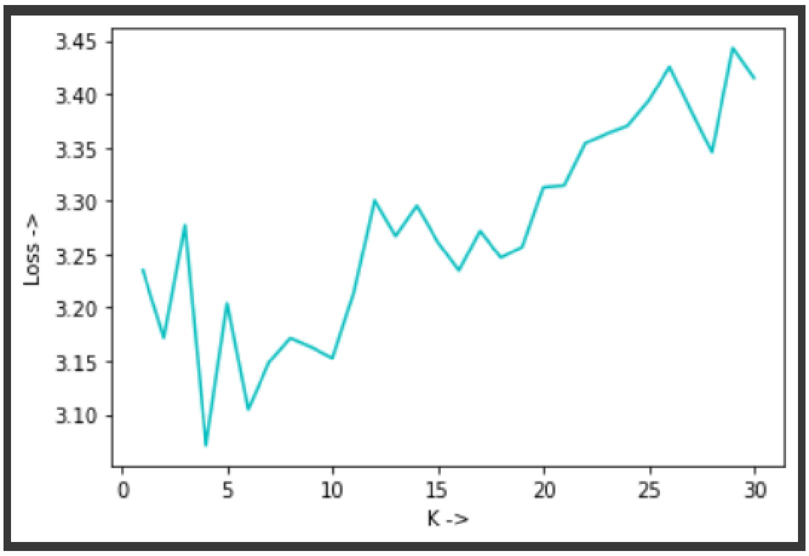
The formula for MSE is:

$$\text{J} = (1/2b) \, * \, \sum_{i=1}^{b} (y^i \, - \, \hat{y}^i)^2$$

Where J is the loss, b is the number of data points in the dataset, y^i is the actual value, and $\hat{y}^i$ is the predicted value.

Result:

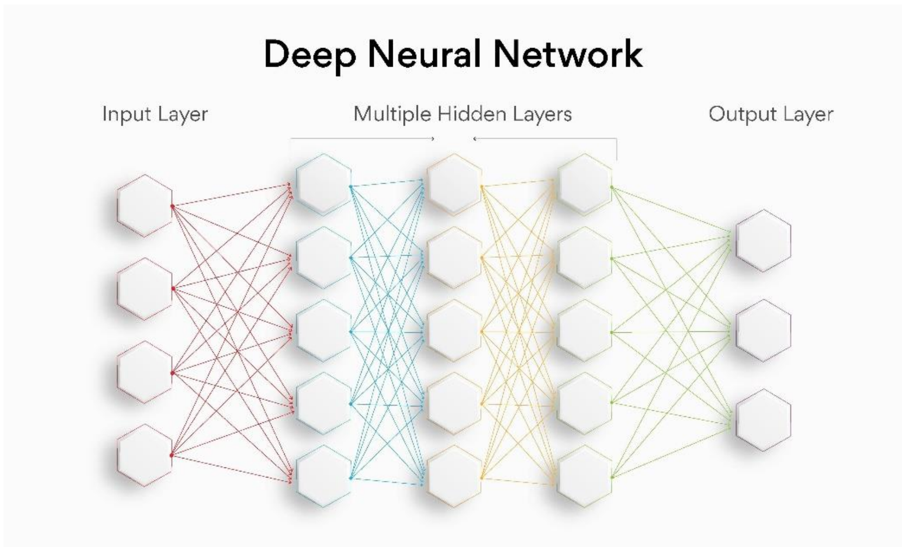
The optimum value of K was found to be 4 by plotting Loss vs K over 1000 examples from the training and testing labelled datasets.



The predicted values for the unlabelled dataset are stored in a .csv file named “KNN_Result.csv”.

Neural Networks

Neural Networks are a family of algorithms that are inspired by the structure and function of the human brain. They consist of layers of interconnected neurons that can learn to recognize patterns in data. They can be used for Regression and Classification.



Neurons:

A Neuron in a neural network functions as: It takes an input, multiplies it with weights, adds a bias to it and then applies an activation function on the output. The neuron sends this activated output forward to each neuron of the next layer.

$$z = x.w + b$$

$$a = Activation(z)$$

, where x is the input to the neuron, w is the weights, b is the bias, z is the output and a is the activated output.

Activation Function:

Activation function as a mathematical operation applied to the outputs of a neuron in order to remove its linearity and increase the accuracy of the model.

The Activation function used for Linear and Polynomial Regression is the Sigmoid Activation:

The Activation functions used for classification are Sigmoid Activation for each hidden layer neuron and SoftMax Activation for the output layer neurons.

Sigmoid Activation:

The sigmoid function maps all real numbers into a range from 0 to 1.

$$Sigmoid(x) = \frac{1}{1+e^{-x}}$$

$$\frac{\partial}{\partial x}\left(Sigmoid(x)\right) = Sigmoid(x).(1 - Sigmoid(x))$$

SoftMax Activation:

Converts Outputs into a probability distribution, which provides the probabilities for each class or label.

$$SoftMax\left(y^i\right) = \frac{e^{y^i}}{\sum e^{y^i}}$$

$$\frac{\partial}{\partial y^i}\left(SoftMax\left(y^i\right)\right) = SoftMax\left(y^i\right).\left(1 - SoftMax\left(y^i\right)\right)$$

Input Layer:

The input layer of the network is the layer in which the input features of the data are introduced into the neural network.

Hidden Layers:

In a neural network, a hidden layer is a layer of neurons that sits between the input layer and the output layer. The hidden layer is where the network performs most of its computation and learns to represent the input data in a way that is useful for making predictions.

The activation function applied to the neurons in the hidden layer is what introduces non-linearity into the output of the network.

Output Layer:

It is the last layer of the Neural Network which provides the final output for the Network. No activation is applied to this output in the case of Regression and a SoftMax activation is applied in this output in the case of Classification.

In Regression, there is only one neuron in the output layer and in Classification, there are as many neurons as there are classes in the output layer.

Forward Propagation:

In this process, the input features are passed into the network from the input layer, then they are propagated through all the hidden layers, updating the outputs of each neuron and then finally returning the final output.

$$z^{[1]} = x.w^{[1]} + b^{[1]}$$

$$a^{[1]} = Sigmoid(z^{[1]})$$

, x where is the input from the input layer, $z^{[1]}$ is the output, $w^{[1]}$ is the weights, $b^{[1]}$ is the bias and $a^{[1]}$ is the activated output for the first layer.

$$z^{[i]} = a^{[i-1]}.w^{[i]} + b^{[i]}$$

$$a^{[i]} = Sigmoid(z^{[i]})$$

, where $a^{[i-1]}$ is the activated output from the $(i-1)^{th}$ layer, $z^{[i]}$ is the output, $w^{[i]}$ is the weights, $b^{[i]}$ is the bias and $a^{[i]}$ is the activated output for the $(i)^{th}$ layer.

Backward Propagation:

Backpropagation is an algorithm used to train neural networks by adjusting the weights and biases at each layer in the network. It works by propagating the error from the output layer back through the network to adjust the weights and biases at each layer through mini-batch gradient descent. It uses the chain rule of differentiation for updating errors at each layer.

For n hidden layers:

$$Error\ at\ Output\ layer = e^{[n+1]} = y - a^{[n+1]}$$

$$Error\ at\ i^{th}\ hidden\ layer = e^{[i]} = (e^{[i+1]}\ .\ w^{[i+1]}) * Sigmoid\left(a^{[i+1]}\right).\left(1 - Sigmoid\left(a^{[i+1]}\right)\right)$$

$$Error\ at\ 1^{st}\ hidden\ layer = e^{[1]} = (e^{[2]}\ .\ w^{[2]}) * Sigmoid\left(a^{[2]}\right).\left(1 - Sigmoid\left(a^{[2]}\right)\right)$$

At i^{th} hidden layer and output layer:

$$\frac{\partial J}{\partial w_j} = a^{[i-1]} \, . \, e^{[i]}$$

$$\frac{\partial J}{\partial b_j} = \sum e^{[i]}{}_j$$

At 1^{st} hidden layer:

$$\frac{\partial J}{\partial w_j} = x \, . \, e^{[1]}$$

$$\frac{\partial J}{\partial b_j} = \sum e^{[1]}{}_j$$

Weights are updated as:

$$w_j := w_j + \propto \frac{\partial J}{\partial w_j}$$

Biases are updated as:

$$b_j := b_j + \propto \frac{\partial J}{\partial b_j}$$

Loss for Linear and Polynomial:

Cost function used is the Mean Squared Error (MSE).

It calculates the average squared difference between the predicted and actual values.

The formula for MSE is:

$$J = (1/2b) \; * \; \sum_{i=1}^{b} (y^i \; - \; \hat{y}^i)^2$$

Where J is the loss, b is the number of data points in the dataset, y^i is the actual value, and $\hat{y}^i$ is the predicted value.

Cost Function for Classification:

The cost function used here is the Cross-Entropy Loss.

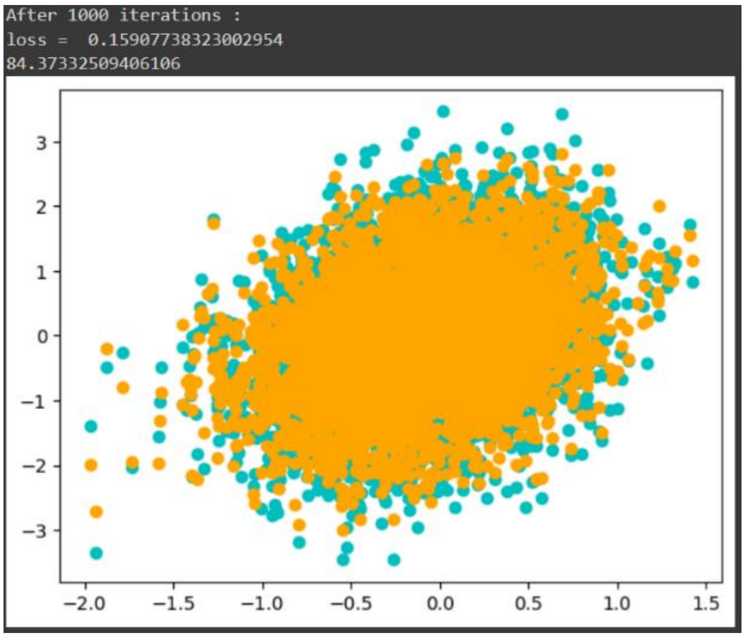
Cross entropy loss is a measure of the distance between the predicted probability distribution and the true probability distribution. The loss is calculated as the negative log likelihood of the true class labels, given the predicted class probabilities.

$$Cross\ Entropy\ Loss = J = -\sum y^i \, . \log (\hat{y}^i \ ) + (1 - y^i \ ). \log(1 - \hat{y}^i)$$

, where y^i is the One-Hot Encoded matrix form of the true label of the i^{th} sample and $\hat{y}^i$ is the matrix of probability distributions for each label of the i^{th} sample

Result:

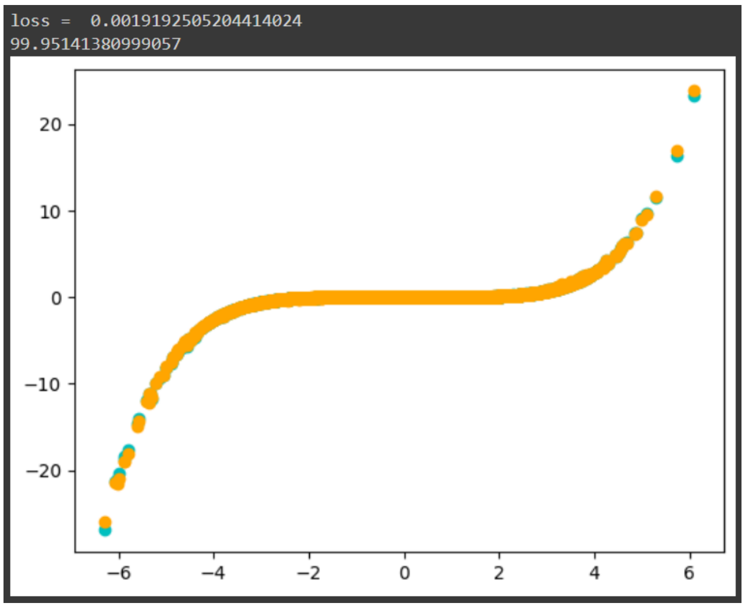
Linear Regression showed an R2 score of 84.37%, within 1000 iterations and learning rate of 0.01.



```
After 1000 iterations :
loss =  0.15907738323002954
84.37332509406106
```

The result for Linear Regression with Neural Networks is stored in a .csv file named “nnl_result.csv”.

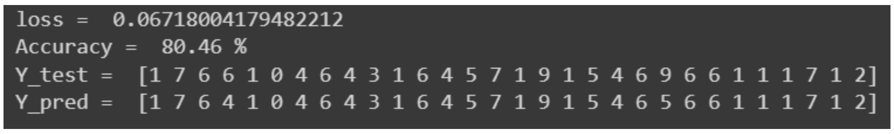
Polynomial Regression showed an R2 score of 99.95% within 10000 iterations and with a learning rate of 0.001



```
loss =  0.0019192505204414024
99.95141380999057
```

The result for Polynomial Regression with Neural Networks is stored in a .csv file named “nnp_result.csv”.

Multi-Class Classification showed an Accuracy of 80.46%, within 1000 iterations and learning rate of 1.



```
loss =  0.06718004179482212
Accuracy =  80.46 %
Y_test =  [1 7 6 6 1 0 4 6 4 3 1 6 4 5 7 1 9 1 5 4 6 9 6 6 1 1 1 7 1 2]
Y_pred =  [1 7 6 4 1 0 4 6 4 3 1 6 4 5 7 1 9 1 5 4 6 5 6 6 1 1 1 7 1 2]
```

The result for Multi-Class Classification with Neural Networks is stored in a .csv file named “nnc_result.csv”.

Conclusion

In conclusion, Implementation and evaluation of five machine learning algorithms: Linear Regression, Polynomial Regression, Logistic Regression, K-NN, and Neural Networks has been achieved. We have used three different datasets to test the performance of these algorithms. The hyperparameters of each algorithm were optimized to achieve the best possible performance.