WOC 6.0
PROJECT REPORT
ML BOOTCAMP

S V SRIYUKTHIKA

23JE0967

COMPUTER SCIENCE AND ENGINEERING

Implementation of Linear Regression Algorithm

Linear Regression algorithm is applied in machine learning, particularly for predicting continuous outcomes. This implementation aims to develop a robust model that accurately predicts values for unknown samples by fitting the data into a linear function.

OBJECTIVE:

To discover the most suitable linear function that minimizes the cost for a given set of training data. we have to determine optimal values for parameters- weights(w) and biases(b)

IMPLEMENTATION DETAILS:

Dataset: The dataset consists of 50,000 examples and 20 features.

Preprocessing: Dataset has been randomly shuffled and the data is split into a training set (80%) and a validation set (20%) for assessing model performance. All features have undergone Z-score normalization to ensure consistency in scale. This transformation centers the values around a mean of 0 with a standard deviation of 1.

ALGORITHM:

Gradient Descent Algorithm is employed to iteratively update all parameters simultaneously to reach minimum cost , it does so by moving in the direction opposite to the gradient of the cost function. The cost function in linear regression is convex, meaning it has a single global minimum. GDA is particularly effective in such scenarios because it is guaranteed to converge to the global minimum for convex functions. The step size of the update is influenced by learning rate(alpha).

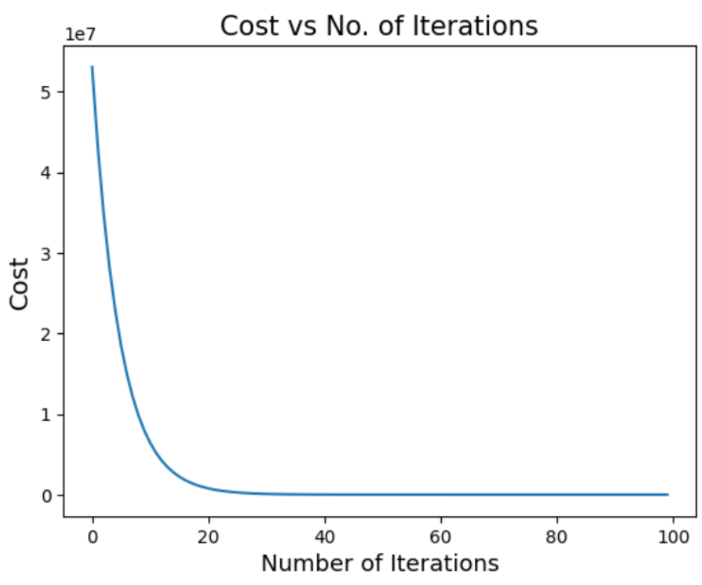
HYPERPARAMETER TUNING:

Learning rate(alpha) - Experimented with several values of alpha, for all the values of alpha less than 0.1 , I have observed slow convergence, for values greater than 0.1 , the cost was overshooting, hence i have decided 0.1 as optimal learning rate for my model.

Initialized w and b with zeros of respective size and performed GDA for 1000 iterations.

After finding optimal w and b ,predicted the output for each sample, model is also evaluated over validation dataset . metrics like R2 score and mse are used for evaluating model performance.

Learning curve: The cost was decreasing in each iteration and converged finally.



RESULTS:

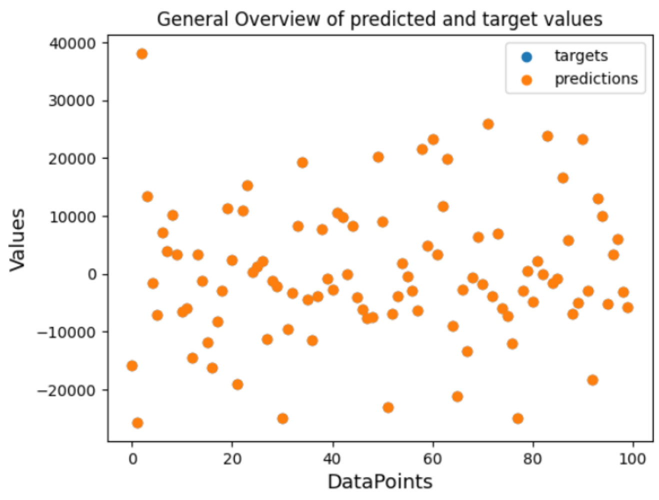
Training performance:

R2 Score - 0.9999999999231622

MSE - 0.005050773937537865

The high R2 score and very low MSE observed in the training data affirm the excellent fit of our Linear Regression model to the training dataset.

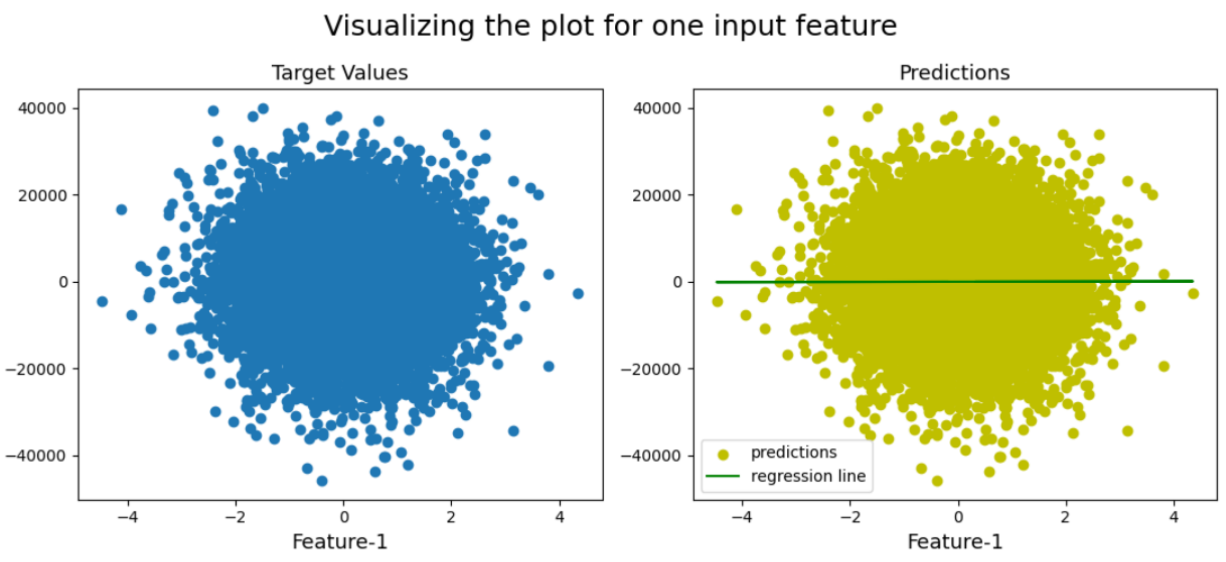
Validation Performance:



We observe that predicted and target values almost overlap with each other indicating accurate prediction.

R2 Score - 0.9999999999218246

MSE - 0.005054569222780917



The **Green line** above is The Regression line with respect to the Feature-1 of the dataset.

The high R2 score and very low mse value for model over Validation data , coupled with insights from the graph highlights the model's strong performance on new, unseen data. This indicates a robust generalization ability beyond the training dataset.

CHALLENGES:

- Initially, my code was inefficient due to extensive loop usage. The iterative nature of these loops resulted in considerable time consumption, leading to slow convergence. After some research, I got to know about the concept of vectorization using NumPy arrays. I modified my entire code, opting for a vectorized approach which led to a significant enhancement in both speed and efficiency.
- initially , while performing Z-score normalization over validation set, i have used mean and standard deviation of validation dataset itself and i realized that my model was inefficient and predictions of validation set weren't as accurate as predictions of training set , my model performance was quiet bad over new unseen data, then i got to know that while normalizing validation or test set , mean and standard deviation of training data should be used. This ensures that model applies the same normalization it learned during training, keeping its understanding of the data distribution consistent.
  After implementing these changes, there has been a substantial enhancement in accuracy and an improved generalization ability of my model.

Implementation of Polynomial Regression Algorithm

The principal behind the Polynomial Regression is same as linear regression. This implementation aims to develop a robust model that accurately predicts values for unknown samples by fitting the data into a polynomial curve.

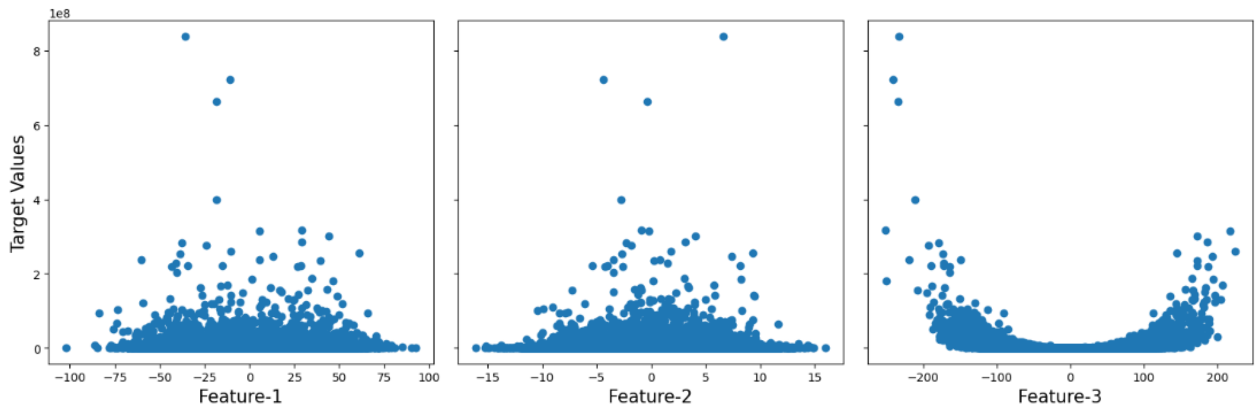
OBJECTIVE:

To discover the most suitable curve that minimizes the cost for a given set of training data. we have to determine optimal values for parameters- weights(w) and biases(b).

IMPLEMENTATION DETAILS:

Dataset: The dataset consists of 50,000 examples and 3 features.

Preprocessing: Dataset has been randomly shuffled and the data is split into a training set (80%) and a validation set (20%) for assessing model performance.



Above is the visualization of Training set, each subplot represents Target values with respect to each Feature.

Defined functions for creating higher degree polynomial , experimented with different degrees, employed GDA for 50000 iterations by initializing weights and biases with zeros of respective size and following is observed.

for degrees 1,2,3,4,5 - the saturated cost is very high

for degree 6 - the saturated cost is very low and time taken for convergence is also good.

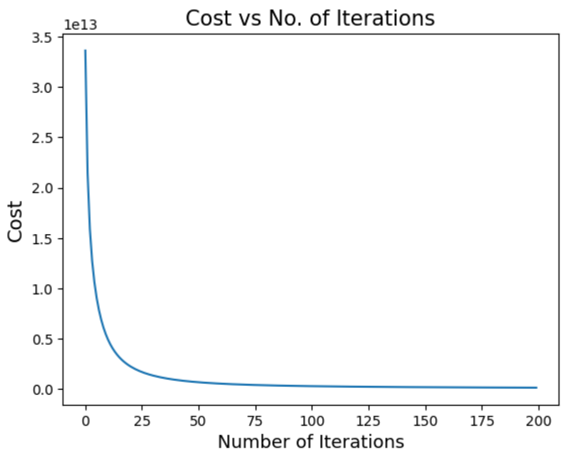
for degrees greater than 6 - the saturated cost increased quiet a bit and also i observed very slow convergence.

Hence, I chose 6 as optimal degree for my model, now the number of features got increased to 83.Now, applied Z-score normalization to all the features to ensure consistency in scale.

HYPERPARAMETER TUNING:

Learning rate(alpha) - Experimented with several values of alpha, for all the values of alpha less than 0.1 , I have observed slow convergence, for values greater than 0.1 , the cost was overshooting, hence i have decided 0.1 as optimal learning rate for my model.

Learning curve:



RESULTS:

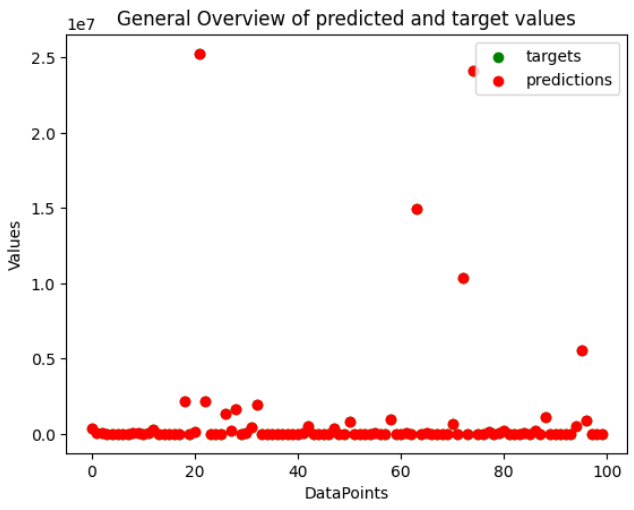
Training performance:

R2 Score - 1.0

MSE - 1.0853223639147256e-15

The high R2 score and very low MSE observed in the training data affirm the excellent fit of our Linear Regression model to the training dataset.

Validation Performance:



We observe that predicted and target values almost overlap with each other indicating accurate prediction.

R2 Score - 1.0

MSE - 7.922074452059576e-16

The high R2 score and very low mse value for model over Validation data , coupled with insights from the graph highlights the model's strong performance on new, unseen data. This indicates a robust generalization ability beyond the training dataset.

CHALLENGES:

The challenge in developing this model was generation of higher degree polynomial features. I spent quiet a lot time to develop the logic and code it up. First , I made a function to count total number of features that will be generated for a particular degree. I have then created an array X_new of zeros of size (m , count) where the count is obtained from the previous function. updated the array X_new with polynomial features of all degrees less than equal to the specified degree using loops.

Implementation of Logistic Regression Algorithm

Logistic Regression in typically used for dealing with classification tasks.

OBJECTIVE:

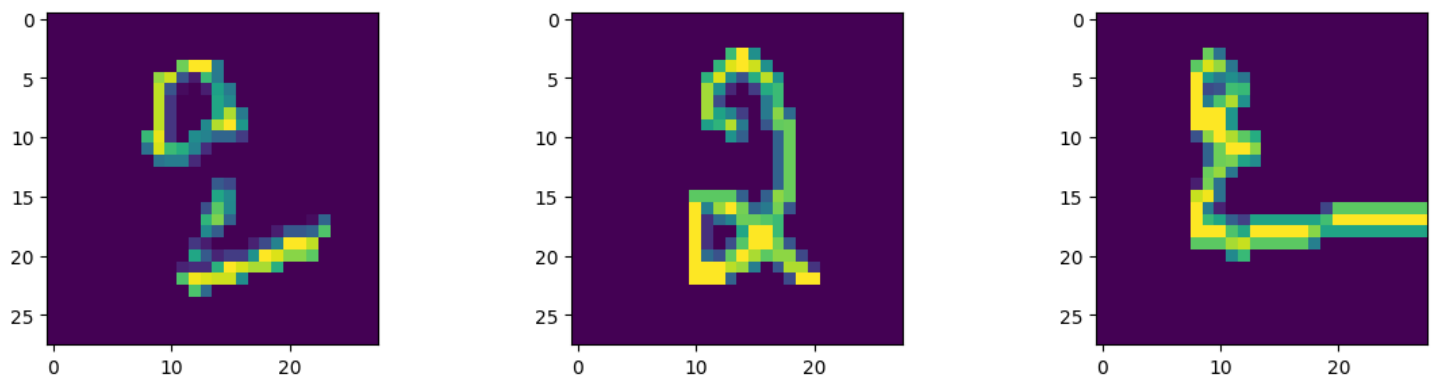
This implementation aims to develop a model that can predict categories of unknown samples.

IMPLEMENTATION DETAILS:

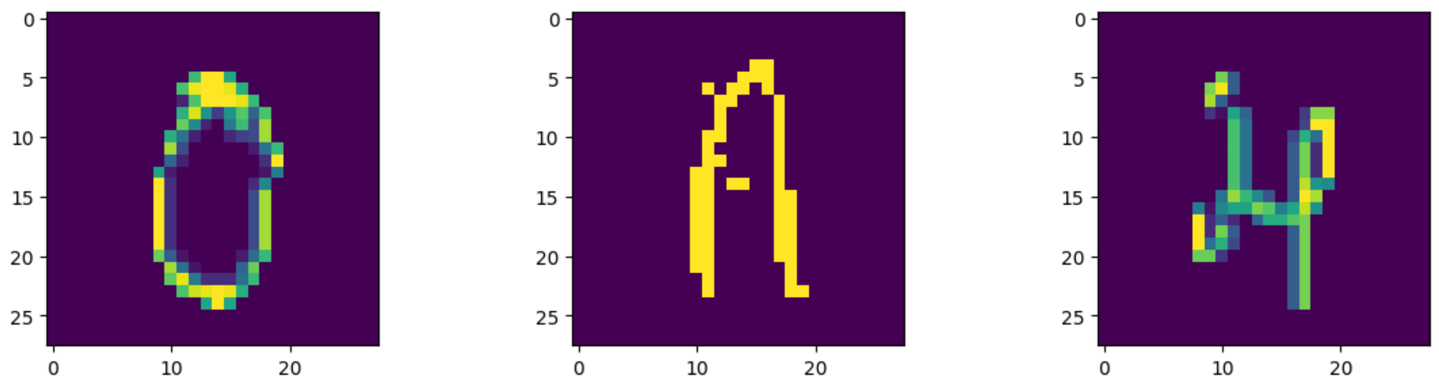
Dataset: The dataset consists of 30,000 examples and 784 features. The data basically had pixel values of images.

Preprocessing: Dataset has been randomly shuffled and the data is split into a training set (20,000 samples) and a validation set (10,000 samples) for assessing model performance.

First 3 samples from Training set:



First 3 samples from Validation set:



These images appear to represent numbers 1 to 9 in the Kannada number system.

Therefore we have 10 classes and we need to classify unknown samples into Kannada numbers.

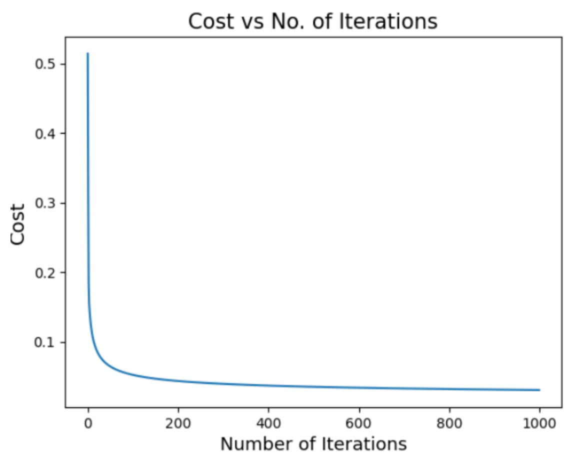
All pixels have undergone Layer Normalization, where each pixel is normalized based on the values of all pixels in the image. Layer norm addresses Spatial dependencies and spatial invariance which are crucial for image recognition. Spatial Dependence in images means that neighboring pixels tend to have some kind of correlation with each other ,and Layer Norm considers the entire pixel set, aiding the model in capturing those spatial relationships. Spatial invariance ensures the model recognizes patterns regardless of their location. By normalizing each pixel independently, Layer Norm maintains consistent scaling and mean values across different parts of the image.

Dealing with multi-class classification ,we use one vs all approach i.e. we extend the concept of binary classification to multi-class classification by considering the correct class as one category and all the other classes put together as another category. Hence we do something called One hot encoding, which labels correct class as 1 and all other classes as 0.

Here for predictions , we use non-linear sigmoid function. The sigmoid function transforms the raw output of the linear combination of input features into a probability value between 0 and 1. Output prediction will be the class with highest probability. We employ GDA for finding optimal parameters which minimizes the Cross entropy or log loss function. we use cross entropy instead of mean squared error because the mean squared error cost function is not convex in the case of logistic regression because of non linear sigmoid function, so GDA might not be effective in leading towards minimum cost. Log loss function on the other hand is convex in nature. Rest is performed same as earlier algorithms.

Here , I found alpha = 1 as optimal learning rate and performed GDA for 10000 iterations.

Learning Curve:



RESULTS:

Training performance:

Accuracy: 97.795

Validation Performance:

Accuracy: 96.73

CHALLENGES:

initially , i have performed batch normalization, later i got to know about the significance of Layer norm in case of image data, I have then modified my code accordingly.

initially , i have performed one hot encoding using loops and indexes ,later i got know that it can be done using numpy function. i have modified my code with numpy.eye , now it runs relatively faster.

Implementation of N-layer Neural Network Algorithm.

The goal is to labels the unknown samples with their categories.

The dataset provided and preprocessing is same as done for logistic regression.

I have developed a 2 layer Neural Network - one hidden layer and one output layer. i have arbitrarily chosen 32 as hidden layer size, i.e. 32 neurons in hidden layer and output layer has no. of neurons equal to no. of classes i.e. 10.

During forward propagation in a neural network, the training dataset, post-normalization, serves as the input layer. This input traverses through the hidden layer and undergoes activation through the Rectified Linear Unit (ReLU) activation function. The activated outputs then progress through the output layer, followed by the application of the softmax activation function. The final prediction is determined by identifying the class with the highest probability, as yielded by the softmax function.

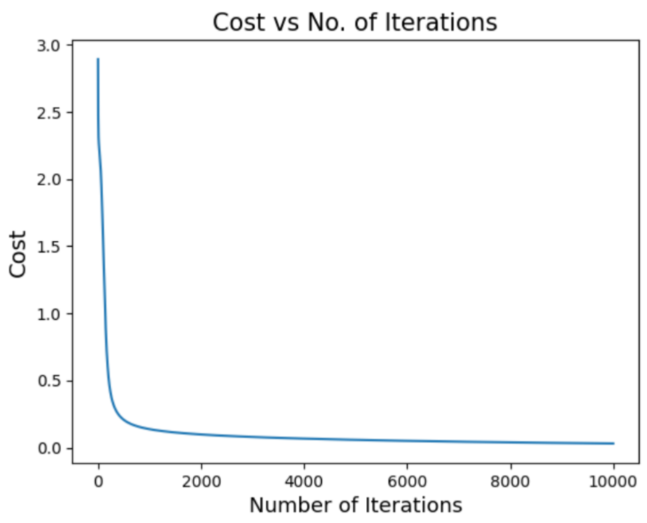
ReLU introduces non-linearity to the network, allowing it to learn and approximate complex, non-linear relationships within the data. ReLU makes all negative values as zeros and leaves positives values unchanged .The simplicity of ReLU accelerates both the forward and backward passes during training.

The softmax function transforms the raw output scores into a probability distribution by exponentiating each raw score and dividing by the sum of all exponentiated scores. It ensures that the values produced by the output layer sum to 1, making them interpretable as probabilities. Each value in the resulting distribution represents the likelihood of the input belonging to a particular class.

Here we use convex error function- Cross entropy as cost function

we have to find optimal parameters w1 , w2 , b1 , b2 using GDA , for which we have to compute gradient of cost function with respect to these parameters. Back propagation helps us to do so by using chain rule.

We randomly initialize weights and biases for both layers and perform GDA for 10000 iterations . I found alpha = 0.1 as optimal learning rate.



After finding optimal parameters , calculated predictions using Forward propagation function.

i have clipped the predicted values to be in between 1e-7 and 1-1e-7, since log is undefined at 0.

RESULTS:

Training performance:

Accuracy: 99.42

Validation Performance:

Accuracy: 97.38

Implementation of K-Nearest Neighbors Algorithm

The objective , dataset , preprocessing is same as earlier.

KNN is a simple algorithm, where the classification of an unknown data point is determined by the majority class among its k-nearest neighbors.

We find Euclidian distance between each datapoint from validation set to each sample in training set, then arrange the distances in ascending order and consider the first k labels. Final prediction will be the class with the majority.

To find the optimal k, i have chosen range of values of k and performed KNN , I measured accuracy for each k value and chose the one that gave the best accuracy.

RESULTS:

for k = 3 , i got 97.74 accuracy.

CHALLENGES:

initially i have used loop and calculated Euclidian distance , it took me almost 5 hours for the final result to show up(including finding optimal k), i have then came across the concept of using (a - b)^2 = a^2 + b^2 - 2ab for computing Euclidian distance, this is done by performing one matrix multiplication and 2 broadcast matrix additions.

After vectorizing the code this way, the computation time significantly reduced.

Implementation of K-Means Clustering Algorithm

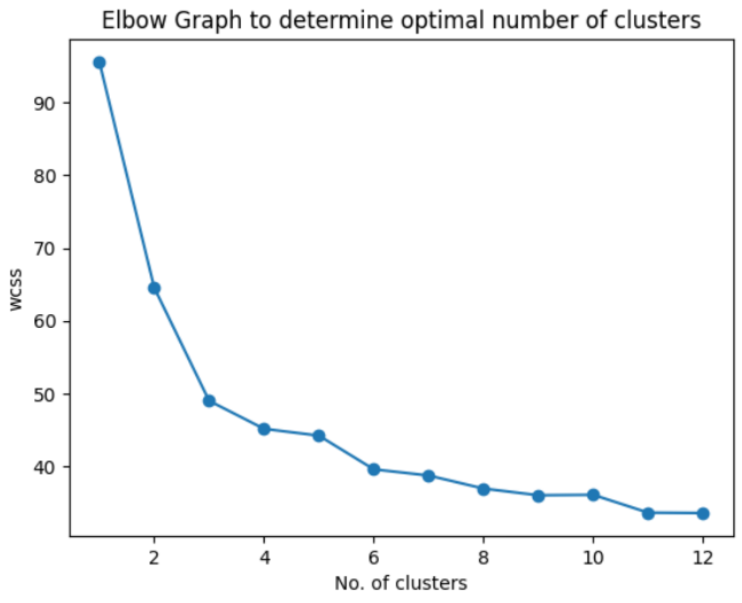
K-means Clustering is an Unsupervised Learning algorithm. The aim of the implementation is to form k clusters minimizing WCSS.

The dataset had 178 examples and 13 features. I have performed mean normalization for all the features.

I have randomly chosen k points as centroids from the given datapoints. Then assigned labels to all datapoints of dataset to their closest centroid. now , calculated new centroids(mean of the clusters formed) and reassigned the labels to datapoints to their new closest centroid. The algorithm iteratively assigns data points to clusters and adjusts the cluster centroids to minimize the WCSS. This process continues until convergence. i have defined a tolerance of 1e-4, i.e the process stops when difference between new centroids and previous centroids is 1e-4.

Also computed wcss for each iteration. wcss stands for Within-Cluster Sum of Squares.

WCSS measures how closely related data points within the same cluster are to each other. It represents the sum of squared distances between each data point in a cluster and the centroid of that cluster. A lower WCSS value indicates tighter and more compact clusters. WCSS is used in the "elbow method" to determine the optimal number of clusters (K). By calculating the WCSS for different values of K and plotting the results, we look for an "elbow" point in the graph where the rate of decrease in WCSS slows down.It is the point after which adding more clusters doesnot significantly decrease wcss .



```
[95.59953778 64.53766702 48.98884443 45.13845603 44.19938061 39.56895258
 38.71654797 36.92215855 36.00352261 36.0616277  33.60857583 33.55840877]
```

The values below graph are WCSS values corresponding to each K value. we notice wcss is not changing much beyond k=6 , hence i choose 6 as optimal number of clusters.