2024

WINTER OF CODE 6.0

ML BOOTCAMP PROJECT REPORT

PRATYUSH TRIPATHI | 23JE0739

CYBER LABS | IIT ISM DHANBAD

Project Report:

Implementation of linear regression, polynomial regression, logistic regression, KNN for classification and N – layer Neural Network with the help of numpy, pandas and matplotlib libraries only.

Linear Regression

A linear regression model was employed for prediction, seeking to establish a relationship between independent and dependent variables. The dataset was imported from a CSV file having 20 features and 50000 training examples using the numpy library. After that I normalised the data to make each features in particular range by **Z-score normalisation** which also increased fast cost calculation and hurried convergence.

```
max in original :993.6949441473038, min in original  :-1031.0753603448304
max in normalized :4.914432208552592, min in normalized  :-5.113882511205697
```

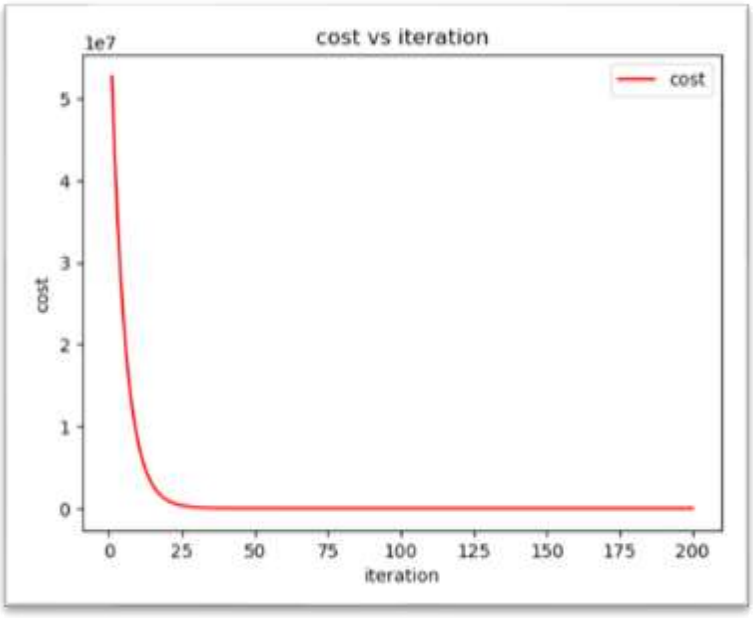
The dataset was split into training having **70%** of total training data and cross-validation comprises **30%** i.e., around 1600 sets of training example to train the model and assess its generalization performance. The cost at various iteration values was calculated, and the results were printed at specific intervals. Different values of the hyperparameter alpha and no. of iteration were tested to enhance the model's accuracy.

Like at alpha equals 0.001 and 0.01 the cost was increasing instead of decreasing and after trying alpha equals **0.1** cost start decreasing and finally converges at some value.

```
alpha: 0.1 No. of iteration: 200

Iteration         0, Cost: 5.26797e+07
Iteration        20, Cost: 8.08246e+05
Iteration        40, Cost: 1.26878e+04
Iteration        60, Cost: 2.03175e+02
Iteration        80, Cost: 3.31447e+00
Iteration       100, Cost: 5.97646e-02
Iteration       120, Cost: 5.97713e-03
Iteration       140, Cost: 5.07691e-03
Iteration       160, Cost: 5.06168e-03
Iteration       180, Cost: 5.06142e-03
```

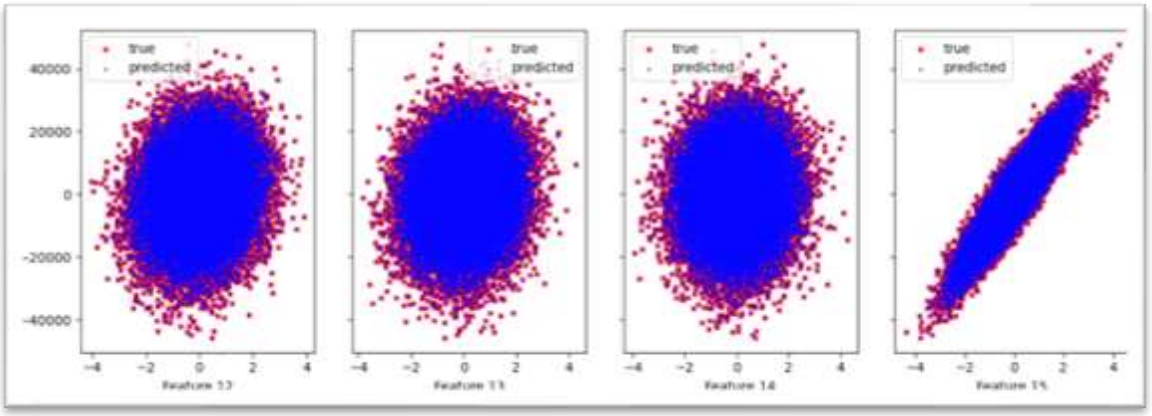
A graph was plotted, illustrating the relationship between the cost (calculated using the final chosen alpha) and the number of iterations.



Finally, The cost function was calculated on the cross-validation data which allowed us to monitor how well the model generalizes to unseen data and detect signs of overfitting. The accuracy checked by computing R-squared (R2) score on the cross-validation data, providing measure of the model's goodness of fit.

```
cost by cross valiting the data: 0.005050128218935684
r2score: 0.9999999999232125
```

I plotted graphs between predicted and true value at first 4 features below.



The optimized model was then applied to test data to predict target values. The linear regression model, optimized through hyperparameter tuning and cross-validation, demonstrated promising predictive performance.

Polynomial Regression with Feature Engineering

In this analysis, polynomial regression with feature engineering was employed using 3 existing features. Feature engineering was applied to create additional polynomial features, with degrees 1 to 6 by taking combination of each 3 features resulting in a varying number of new features. The cost at various iteration values was calculated, and results were monitored to identify the most suitable polynomial degree

Data is normalised as before by Z score normalisation.

```
max in original :252135504217851.8, min in original  :-45432446512019.4
max in normalized :105.0673195586038, min in normalized  :-110.57972542835242
```

Again, the dataset was split into training and cross-validation in ratio **70:30**. At different value of degree different shaped featured array created. At different degree from 1 to 5 the accuracy checked by R2-score is found low as expected but in **6-degree** polynomial my accuracy becomes **1.0.** Like in 6-degree polynomial.

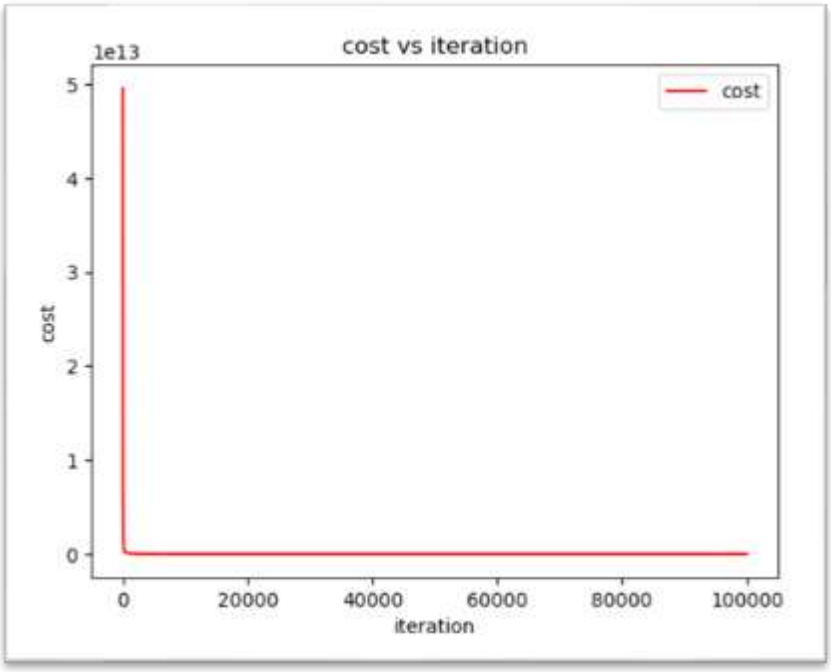
```
shape of engineered data : (50000, 86)
```

When alpha equal **0.03** at different value of no. of iteration the cost was not converging finally at 100000 cost converges and accuracy increased.

```
alpha: 0.03 No. of iteration: 100000

Iteration         0, Cost: 4.96138e+13
Iteration     10000, Cost: 3.80040e+08
Iteration     20000, Cost: 2.06201e+06
Iteration     30000, Cost: 1.42934e+04
Iteration     40000, Cost: 1.27881e+02
Iteration     50000, Cost: 1.39291e+00
Iteration     60000, Cost: 1.69157e-02
Iteration     70000, Cost: 2.15825e-04
Iteration     80000, Cost: 2.81156e-06
Iteration     90000, Cost: 3.69491e-08
Iteration    100000, Cost: 4.87662e-10
```

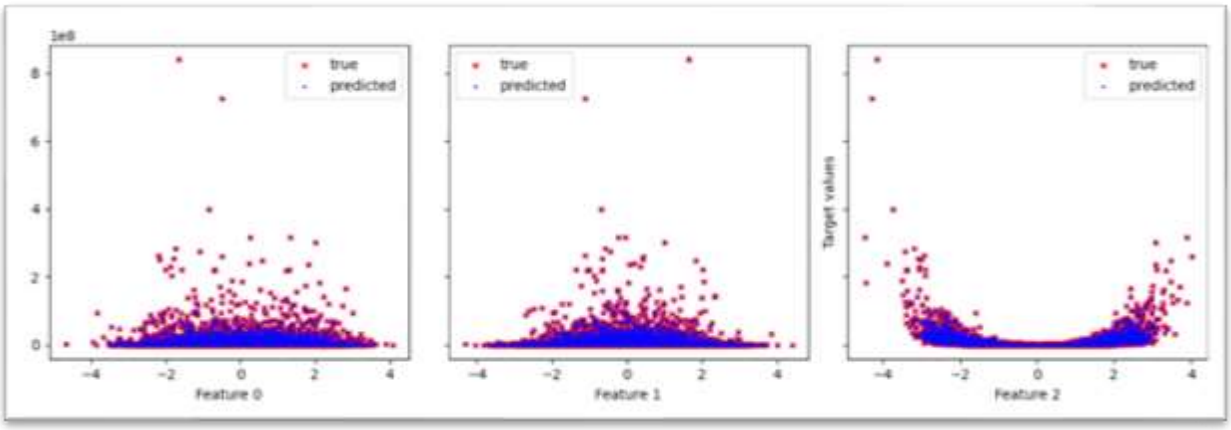
Graphs were plotted to illustrate the relationship between the cost (calculated using the final chosen polynomial degree) and the number of iterations.



The cost function was calculated on the cross-validation data. The R-squared (R2) score was computed on the cross-validation data.

```
cost by cross valiting the data: 1.6025016908342113e-09
r2score: 1.0
```

I plotted graphs between predicted and true value at original 3 features below.



The analysis highlighted the impact of feature engineering and polynomialization on the model's performance.

Logistic Regression.

Prior to model training, the dataset underwent normalization, wherein each feature was divided by its maximum value **(255)**. This standardization ensures consistent scale across features that is 0 to 1, helpful in the convergence of the algorithm.

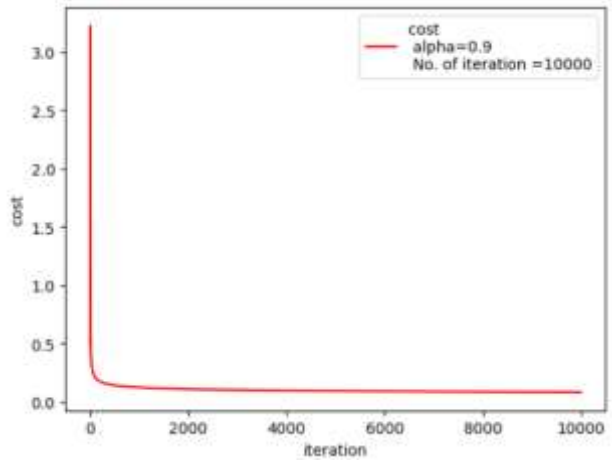
```
max in original :255, min in original :0
max in normalized :1.0, min in original :0.0
```

The dataset was then split into training and cross-validation sets. The true values were **one-hot encoded** to create an array of 10 columns, corresponding to the 10 possible classes. This encoding facilitated the classification task, transforming the target values into a binary matrix. Look below some examples are shown.

1-hot-encoding done

	0	1	2	3	4	5	6	7	8	9
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0
2	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0
4	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0
5	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0
6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0
7	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0
8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0
9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0

Gradient descent was employed as the optimization algorithm to minimize the cost function. The linear function, parameterized by the weights and biases, was transformed using the sigmoid activation function to yield the predicted probabilities.The impact of varying alpha and iteration values on the cost function was evaluated to identify the most suitable combination. Graphs were plotted to visualize the cost function's behaviour at different iteration values for various alpha values. This graphical representation helps in understanding the convergence of cost using suitable hyperparameters.



In the end, the cost is calculated from cross validation data and checked accuracy and calculated f1- score with help of confusion matrix.

```
cost after cross validating: 0.05123099734246858
Train Accuracy: 96.744083
```

```
f1score: 0.967310105751835
```

confusion_matrix

	0	1	2	3	4	5	6	7	8	9
0	853	22	0	3	0	0	1	0	3	2
1	4	909	0	2	0	0	0	0	0	1
2	5	0	891	4	0	0	0	0	0	0
3	15	0	0	836	6	12	9	17	0	0
4	2	0	1	2	902	15	0	2	0	3
5	0	0	0	12	8	908	2	0	0	0
6	0	0	0	6	0	0	877	27	0	6
7	1	0	1	22	1	1	32	800	1	1
8	6	1	1	1	0	0	0	0	851	4
9	0	0	0	1	11	2	7	4	3	879

And a test array as input predicts target values and written in a test CSV file named “**Logistic_test_prediction.csv”**.

K- Nearest Neighbour

The data is form of CSV file which is converted in numpy array using numpy module. Data had been normalized by dividing each value by the maximum value of the training array, which is set to 255. This normalization scales the data to a range between 0 and 1, improving the stability and convergence of the KNN algorithm.

```
max in original :255.0, min in original :0.0
max in normalized :1.0, min in original :0.0
```

The dataset had been divided into two subsets: a training set, comprising 83% of the data, and a cross-validation set, comprising the remaining 17%.

The KNN algorithm is applied to the normalized training data to predict values on the cross-validation set. The predictions are then compared to the true values using a confusion matrix and calculated F1score of cross validation array of data.

```
Train Accuracy: 98.352941
f1score: 0.9834858111319988
```

confusion_matrix

	0	1	2	3	4	5	6	7	8	9
0	477	14	0	0	0	0	0	0	0	0
1	4	504	0	0	0	0	0	0	0	0
2	2	0	511	1	1	0	0	0	0	0
3	4	0	0	508	1	0	0	4	0	0
4	1	0	0	1	512	1	1	1	0	2
5	0	0	0	4	2	521	1	0	0	1
6	0	0	2	1	0	0	505	5	0	4
7	0	0	0	7	1	0	8	488	0	1
8	6	0	0	0	0	0	0	0	474	1
9	0	0	0	0	0	0	2	0	0	516

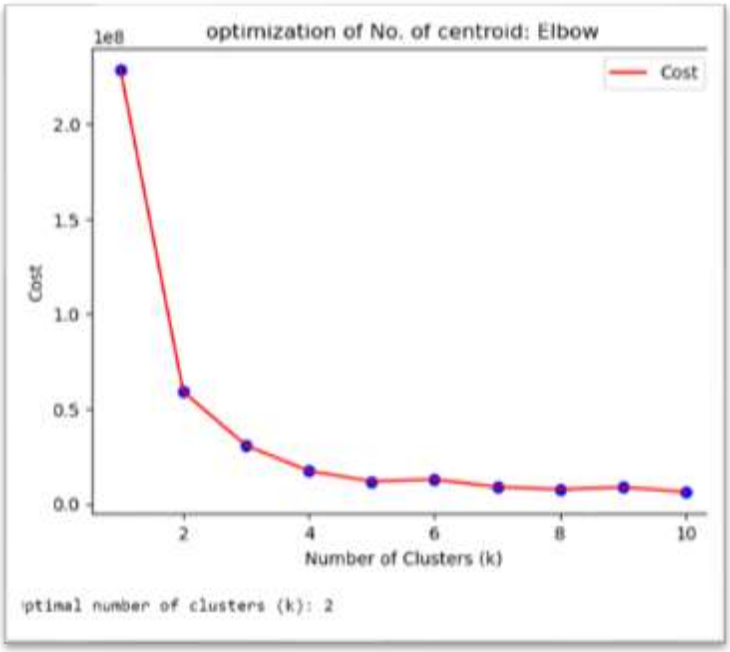
Finally, the test data is loaded in numpy array and predicted classes and written in CSV file name **KNN_test_prediction.csv**.

K- Means

Implementation of the K-Means algorithm applied to training. The training data is loaded from a CSV file, and the K-Means algorithm is initiated with random centroids chosen from the training set. This approach allows the algorithm to converge efficiently and produce meaningful clusters. The K-Means algorithm is executed for various values of K. For each run, the number of iterations required for convergence is recorded. Additionally, the final centroids are obtained, and the distance from these centroids to all training examples is calculated.

```
K-Means cluster: 1 ---- No. of iteration: 1 ---- final distance: 228699852.9856099
K-Means cluster: 2 ---- No. of iteration: 9 ---- final distance: 59094701.797291875
K-Means cluster: 3 ---- No. of iteration: 7 ---- final distance: 30818965.928178564
K-Means cluster: 4 ---- No. of iteration: 5 ---- final distance: 17386586.55222388
K-Means cluster: 5 ---- No. of iteration: 9 ---- final distance: 11952485.278751403
K-Means cluster: 6 ---- No. of iteration: 9 ---- final distance: 12933249.739228837
K-Means cluster: 7 ---- No. of iteration: 9 ---- final distance: 8997952.209806636
K-Means cluster: 8 ---- No. of iteration: 9 ---- final distance: 7745461.022386361
K-Means cluster: 9 ---- No. of iteration: 9 ---- final distance: 8826005.75569609
K-Means cluster: 10 ---- No. of iteration: 7 ---- final distance: 6417804.395036268
```

The Elbow Method is employed to determine the optimal number of centroids (K). The sum of squared distances (inertia) between data points and their assigned centroids is plotted against different values of K. The 'elbow' in the graph, where the inertia starts to decrease at a slower rate, indicates the optimal number of centroids.



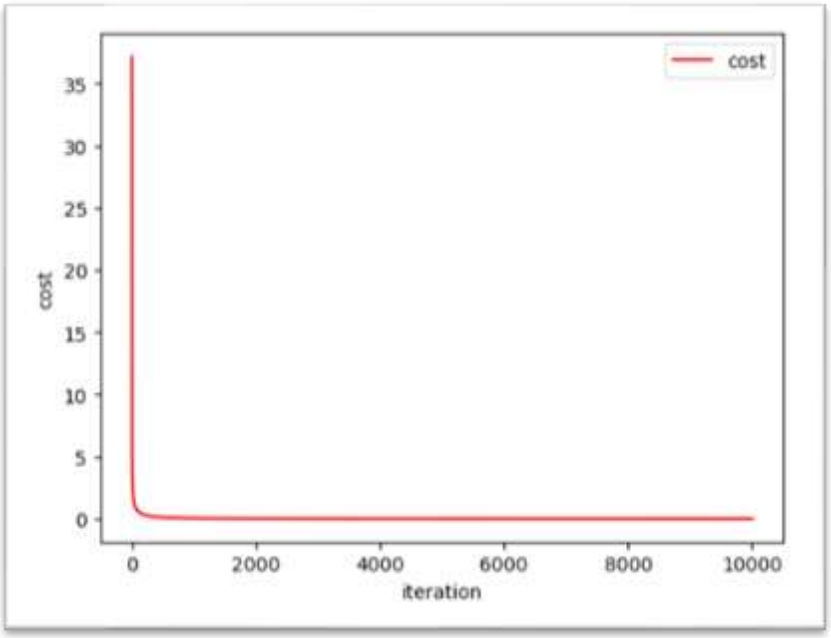
A CSV file named **'kmeans_centroid_prediction.csv'** is generated by the algorithm. This file contains information about the final centroids and their assignments for each training example.

N Layered neural network

The dataset is loaded from a CSV file containing features and target values and converted into NumPy arrays and scaled numerical features by dividing with max value that is 255. The dataset is split into training and cross-validation sets in an 83:17 ratio. The neural network are tested, including different numbers of layers, neurons per layer, and alpha values. The model is trained with different sets of hyperparameters, and the cost function is monitored over iterations. The goal is to identify suitable hyperparameters at which the cost converges, indicating optimal model training.

```
units = [128, 10] alpha = 1 layers = 2 steps = 10000
Iteration         0, Cost: 37.140011465594846
Iteration      1000, Cost: 0.052788596638572674
Iteration      2000, Cost: 0.018830931161895624
Iteration      3000, Cost: 0.008525655817449329
Iteration      4000, Cost: 0.0044266216276174246
Iteration      5000, Cost: 0.0027200824036308054
Iteration      6000, Cost: 0.0018855351912992117
Iteration      7000, Cost: 0.001409588122764899
Iteration      8000, Cost: 0.0011151847090566786
Iteration      9000, Cost: 0.0009155935061359194
Iteration     10000, Cost: 0.0007726137471864329
```

Finally, at chosen hyperparameter values graph is plotted between cost vs no. of iteration.



The neural network's performance is evaluated using accuracy and F1-score on the cross-validation set. The confusion matrix is shown below.

Accuracy: 95.9403805%, F1 Score: 0.96

confusion_matrix

	0	1	2	3	4	5	6	7	8	9
0	468	13	4	1	0	0	0	3	2	0
1	13	488	2	0	0	0	2	1	1	1
2	1	0	511	1	0	1	0	0	1	0
3	2	0	2	494	4	2	1	10	0	2
4	1	0	0	4	498	5	1	5	0	5
5	0	1	4	4	4	509	0	1	6	0
6	0	2	0	2	0	0	489	17	0	7
7	0	0	0	11	0	0	18	473	0	3
8	4	1	1	3	1	2	1	0	467	1
9	2	3	0	1	8	1	2	3	2	495

These metrics provide insights into the model's ability to correctly classify instances from the unseen data.

The final model, trained with the optimal hyperparameters, is used to predict the target values for test data and written in a CSV file named "**neural_test_prediction.csv".**