PRACTICAL MACHINE LEARNING – Assignment 1

CONTENTS:

REPORT ON PRACTICAL MACHINE LEARNING ASSIGNMENT – 1

AUTHOR: SHANKAR PENDSE

STUDENT ID: R00195877

PRACTICAL MACHINE LEARNING – Assignment 1

PART 1(A): Development of Distance Weighted KNN for Regression

Introduction: K Nearest Neighbour regression considers "K" number of nearest neighbours for the query instance and assigns the average/mean value as the prediction to the query instance.

In Distance Weighted KNN for regression, we will multiply the inverse of distances of "K" nearest neighbours of our query instance, with their respective target value, sum it up and divide by the sum of inverse of distances, assign the resulting value as regression value for our query instance.

Generally, we call this assignment of inverse distances as giving weight to the neighbours, and by assigning this weight, "*we are getting rid of one specific drawback of KNN where it treats each of the K nearest neighbours with equal importance*". This weight assignment ensures that, the nearest neighbour is given more weightage and the relatively far neighbour is given less weightage in deciding the regression value for our query instance.

Below 3 variations are implemented as part of Distance Weighted KNN for Regression:

Code inclusion goes out of alignment, so I am just providing the link to the text file which contains all the three functions. Please **Click Here** to access the file

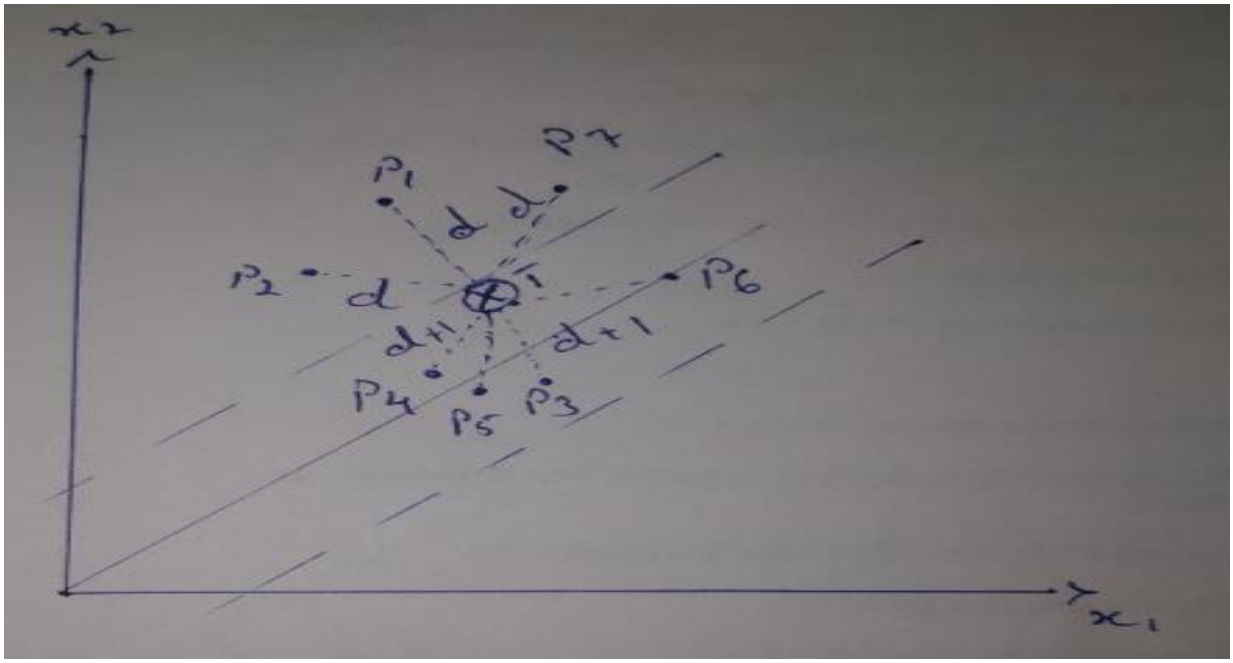
Below is the R2 value obtained from the implementation:

SL NO	TYPE OF KNN	R2 Score	Run time
1	Simple KNN	0.81659	1.73
2	Distance Weighted	0.81857	1.73
3	Squared Distance Weighted	0.81971	1.21

PART 1(B): Parameters/Techniques Impact Model Performance

K-NN when using Euclidean distance as a method to calculate the distance between data points, it considers each feature with equal importance and just returns the distance as square root of squared sum of their differences without considering the distribution of the data. Let us consider below figure to have a better and easy understanding as what the above statement means.

PRACTICAL MACHINE LEARNING – Assignment 1



In the above figure, the dashed lines represent the data distribution (the desired), point marked as **T** is our query instance, p1 to p7 are our training instance data. Out of those p1 to p7 training data instances, p1, p2 and p7 are equally distanced from our query instance and remaining points are bit far when compared to p1 p2 and p7.

Interesting thing to note here is that, the data points p1, p2 and p7 are all outside the desired distribution of the data and others are within the distribution. Euclidean distance just returns the nearest neighbours as p1, p2 and p7 (because it does not care about the distribution of the data) we end up predicting the wrong value for our query instance.

Below is another issue in connection with Euclidean distance.

Let's say we have two features in dataset namely "**Electricity units consumed**" and "**Electricity Bill**", these two features are obviously correlated, which means, as the electricity units consumed increases, the Electricity bill also increases, but when using the Euclidean distance, it just gives equal importance to both the features and in this case, it's like utilizing the impact of same feature twice!. So, what we are saying here is Euclidean distance does not consider the collinearity between the features while calculating the distances, it simply considers each feature as equally important and calculate the distance.

To tackle the above issues, we can consider few things as listed below:

1. Assign individual weights to the features based on their importance (a lengthy procedure which could be considered as optimization problem) (reference: **Lecture video**)
2. Use different distance metric:
    a. Manhattan (block wise distance also called as taxicab distance)
    b. Mahalanobis
    c. Minkowski (Generalized version of Manhattan and Euclidean, but it could have different effect depending on the value we set)

   d. Hamming distance
   e. Jaccard distance
3. Identify the strong and weak predictors (correlation) and remove the weaker ones

We will look into what Mahalanobis distance metric is and use it on our problem

Unlike Euclidean distance, Mahalanobis distance calculates the distance between a point and the distribution. It first standardizes the data by subtracting the distance of all feature data points with its mean and then divide by the covariance matrix of feature data points. Mathematically it is represented as :

(featureData – Mean of each column) / (covariance matrix of featureData)

Please note that, featureData is excluding the target column (variable). Basically, what we are doing here is getting all the datapoints in the feature data closer to its mean value(distribution will be dense around the mean value for the variables which are correlated). If the features (variables) are uncorrelated, the distance remains larger (does not get reduced by much)

Thus, while selecting the nearest neighbours now, this makes sure that the datapoints outside the distribution are not selected (because of larger distance of the non-correlated features). In other words, the datapoints which are really (mathematically) close to are selected.

Below is the table with R2 scores:

SL NO	DISTANCE	Neighbours	Scaling	R2 Score
1	Inverse Squared distance Euclidean	4	Min Max	0.8437
2	Mahalanobis	4	NA	0.8408

There is not much difference in our R2 Score when using Mahalanobis and Euclidean distance, if we look at it carefully, Mahalanobis because of its nature, should have overcome the issue associated with Euclidean distance (treating all features with equal importance) either this or the data to belonging to different clusters are very close to each other

There seems to be something else which might be dragging the performance of KNN towards lower side

Other techniques to improve the performance of KNN:

1. Apart from checking on different distance metrics, we know that there is something called strong and weak predictors (features). The features which have more influence on the value associated with the target variable, are called Strong predictors and the features which have negligible influence on the value associated with the target variable are called weak predictors. And if this is the case, as Euclidean distance treats all features with equal importance, the weak predictors could be influencing the regressor prediction value in a negative way (could be the

case where these features are contributing more in deciding the value of our target variable). We can also try different values for K (no of nearest neighbours)

Let us look at the correlation of all the features with our target feature. We will be using pandas for better visualization of correlation table, which is shown in below correlation matrix:

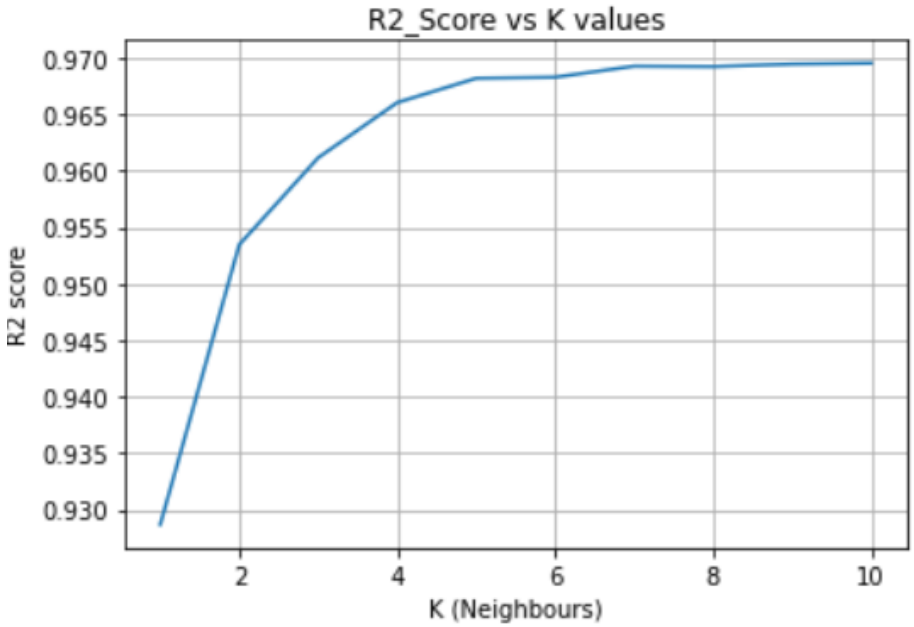
```
In [10]: #Let us check the correlation
         df_train.corr()
```

Out[10]:

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	1.000000	-0.004362	0.007072	0.000705	0.008373	-0.011065	0.022645	-0.000628	0.014265	0.003355	-0.004684	-0.017824	0.004609
1	-0.004362	1.000000	-0.020603	-0.001333	-0.005931	0.008843	0.008947	-0.004167	-0.021844	0.004767	0.004464	-0.005987	-0.001219
2	0.007072	-0.020603	1.000000	0.000111	-0.014672	0.014021	-0.010896	-0.008521	0.000466	-0.013469	0.002297	-0.013291	-0.022593
3	0.000705	-0.001333	0.000111	1.000000	-0.000592	0.020930	0.003464	0.018884	-0.009665	-0.013561	0.014955	0.000111	0.006303
4	0.008373	-0.005931	-0.014672	-0.000592	1.000000	0.018053	0.000553	0.006604	-0.018079	0.001130	-0.007775	-0.002404	0.000415
5	-0.011065	0.008843	0.014021	0.020930	0.018053	1.000000	0.010928	0.009479	0.009403	-0.014863	0.013834	0.009100	0.009425
6	0.022645	0.008947	-0.010896	0.003464	0.000553	0.010928	1.000000	0.000829	0.004241	-0.012111	-0.014598	0.001990	0.500095
7	-0.000628	-0.004167	-0.008521	0.018884	0.006604	0.009479	0.000829	1.000000	-0.015647	-0.008439	0.006654	-0.010313	0.484448
8	0.014265	-0.021844	0.000466	-0.009665	-0.018079	0.009403	0.004241	-0.015647	1.000000	0.016315	-0.005209	-0.027966	0.152783
9	0.003355	0.004767	-0.013469	-0.013561	0.001130	-0.014863	-0.012111	-0.008439	0.016315	1.000000	0.013414	0.000968	0.466152
10	-0.004684	0.004464	0.002297	0.014955	-0.007775	0.013834	-0.014598	0.006654	-0.005209	0.013414	1.000000	-0.004695	-0.000782
11	-0.017824	-0.005987	-0.013291	0.000111	-0.002404	0.009100	0.001990	-0.010313	-0.027966	0.000968	-0.004695	1.000000	0.504330
12	0.004609	-0.001219	-0.022593	0.006303	0.000415	0.009425	0.500095	0.484448	0.152783	0.466152	-0.000782	0.504330	1.000000

Column and row numbered 12 is our target variable, and we can see that the features numbered 0,1,2,3,4,5 and 10 are least correlated with our target variable.

After removing those least correlated features, our **R2 score jumped to 0.966** with k=4 nearest neighbours. Below is the plot of R2 Score vs different values of K:



Now, it is safe to conclude that our data did not have issue with the data points distribution or their scale (how much the values vary) . The problem with this dataset was, there were almost 50% of weak predictors which were affecting the prediction value of our target variable in a bad way. We have to be careful in weeding out too

many predictors from our dataset, as we might be losing some important information out of those.

2. We can assign weights to each feature and use those weights while calculating the normal Euclidean distance, so that the weak features are given less importance automatically. But the problem here is to find the weights for each feature. For quick analysis, if this works, we can consider any of the ensemble techniques, to get the feature importance. Plug in that feature importance while calculating the Euclidean distances, then perform the regular KNN Regression steps. The results should basically match (between the KNN regression and any of the ensemble techniques)

Please **Click Here** to take a look at the code

PART 2(A): Development of KMeans clustering Algorithm

Introduction: KMeans clustering comes under the category of unsupervised learning, where there is no training step, we directly use the available data to get to our goal.

KMeans clustering is a technique where we group the given data points into K number of clusters. We initially would not know this number (K) and we follow below steps to determine the same

1. Select K number of clusters
2. Randomly select K datapoints as initial centroids (cluster points)
3. Then calculate the distance between centroids and each data point and assign the data point to that centroid which is nearer. Repeat this step till all data points are assigned to the centroids (we call this step as assign centroids)
4. From the data points assigned to each centroid in step 3, calculate their mean value and return these mean values as new K centroids (we call this step as move centroids)
5. Calculate the distortion cost function, after selection of new K centroids from step 4
6. Repeat the steps 3, 4 and 5 till there is no change in the centroids value or in other words, the cost does not change (changes are negligible) / repeat for some iterations
7. Repeat the steps from 1 till 6 for different values of K, and plot the graphs representing values of K on X axis, and value of distortion cost on Y axis
8. As K value increases initially, we can generally see the reduction in cost in acceptable terms, but once K reaches some value, the cost does not change much (the change is negligible) which forms a kind of elbow shaped curve. The point where we see the elbow, is considered as the appropriate number of clusters (centroids).
9. We use the value of K where the curve takes shape of elbow, as appropriate number of centroids because of the fact that, the change in distortion cost value after this point is very small that it is not worth waiting for so much time.
   Let's have a look at below plot of cost vs different K (number of cluster) values:

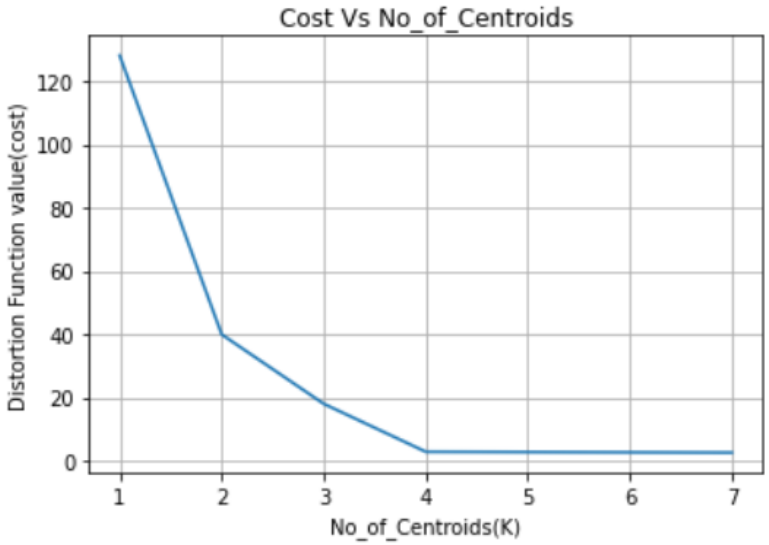


Fig 1.1: Before Normalizing the data

As we can clearly see from the above graph, at the point where K = 4, we can see that our curve started to take kind of an elbow shape, after which for all values of K 5,6,7 the rate of change of cost is negligible.

We can see that the cost is quite high, let us see if we get any improvement after normalizing the data (min max normalization) which basically brings all the values in the range of 0 to 1

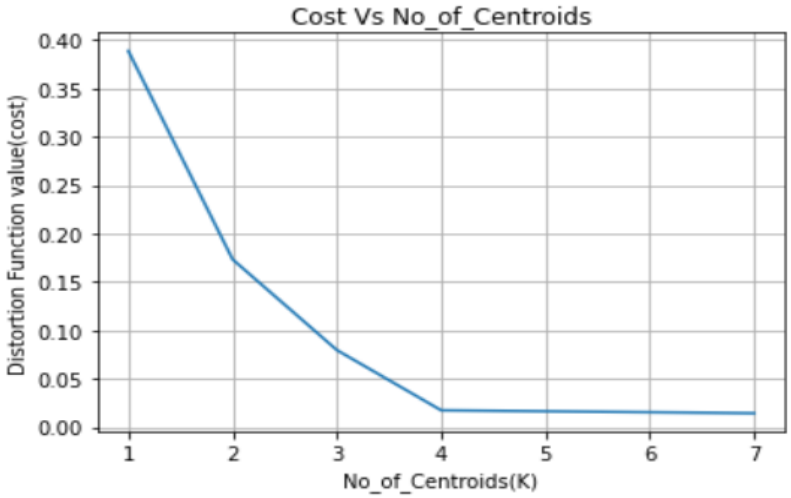


Fig 1.2 After normalizing the data

As we can clearly see from the above two plots, the distortion cost has improved significantly after normalizing the data. This is because earlier, K Means clustering using Euclidean distance as distance metric, used to give more weight on the features which vary on large scale, thus minimizing the impact of other features.

By normalizing, we basically removed the above said impact, and thus our K Means clustering saw major improvements with distortion cost function value at 0..017 when K = 4. To have a look at implementation of each function, please **click here**

PRACTICAL MACHINE LEARNING – Assignment 1

PART 2(B): Researching alternatives to KMeans

Mini Batch KMeans: KMeans clustering gets quite slow as the data increases, as for each iteration distance whole data set needs to be in memory and distance needs to be calculated again and again between each cluster centroids and the datapoints. This is both time consuming and space consuming

Let's see how this works in below steps:

We will use **data** to denote the overall data points, **data_batch** to denote the batch samples out of **data**, **centroids** to denote the data points selected as centroids

1. Select the initial specified number of centroids at random
2. Decide on the **batch size** (number of points to be considered for each batch)
3. For specified number of iterations, repeat the below steps:
    a. Randomly select the data points of **batch size,** we call it as **data_batch**
    b. For each data point in **data_batch** perform assign them to the cluster as below:
        i. Assign each of the data points in **data_batch** to the nearest **centroid** from **centroids**
    c. For each data point in **data_batch** update the respective centroid as below:
        i. Average of all the previous data points assigned to the respective centroid
        ii. (i) + moving average of the Centroid with respect the selected sample (**data from batch_data**) (1/no_samples_assigned_to_cluster)

        Basically, all we are doing is updating each centroid value in **centroids** by the moving average with respect to the value of the data point assigned to that particular centroid

Mini Batch KMeans converges faster due to two reasons:

1. We are calculating the distance between centroids and subset of whole **data** which we call it as **data_batch**
2. The move centroid step (to update the centroid values) we do not compute average considering all data points in **data_batch**, but instead it is update for each data in the **data_batch** using the moving average (Which we do by storing the previous average and calculating only the differential)

Due to the above said modifications, lot of calculation time is saved and also the memory is utilized efficiently. But thing to note here in the formula is 1/no_samples_assigned_to_cluster, which makes the cluster movement slow as the number of data points assigned to the cluster increases.

But as we are selecting the **data_batch** at random, there may be chance that few data points are not considered for cluster assignment and movements. This will result in slightly higher distortion cost value, but it will almost give the similar result as of KMeans.

One good way to measure the suitable value of K (no of clusters) is to check the silhouette score, which tells us that how dense are the clusters with for each K value. It is best to choose the K which gives high silhouette score.

REFERENCES:

1. PART1B) https://www.machinelearningplus.com/statistics/mahalanobis-distance/
2. PART2B):
   https://scikitlearn.org/stable/auto_examples/cluster/plot_mini_batch_kmeans.html
   https://towardsdatascience.com/k-means-clustering-fa4df5990fff