Machine Learning Assignment 2

Goktug Ekinci

2380343

Part 1

In this part, I coded the KNN algorithm to classify the dataset. I used cross validation method to eliminate the randomness of selection of the testing and training data, with 10 fold. All the model configuartions are tested 5 times regarding the cross validation. In other words, all the models have tested 50 times in total. I took the average accuracy of the folds of cross validation.

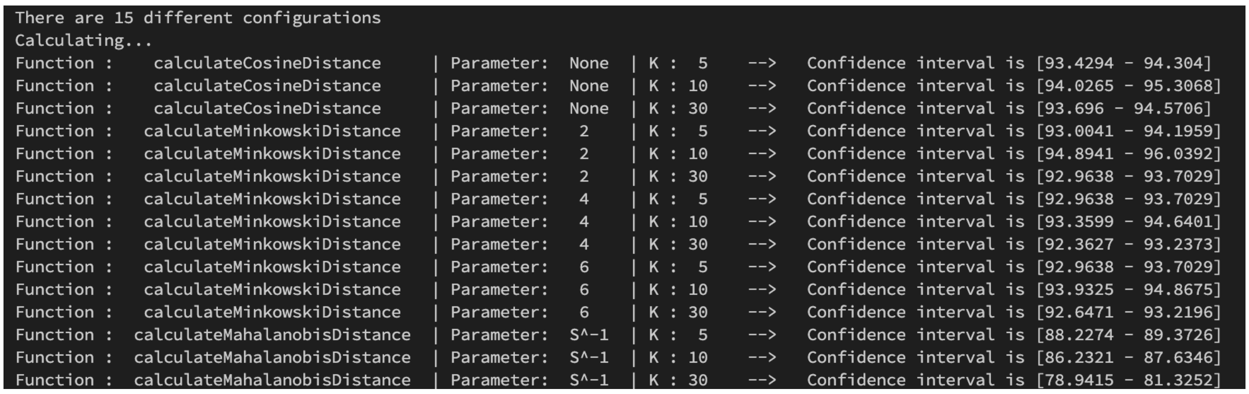
Configurations

I experimented with 15 different configurations using regarding 3 different hyperparameters, which are K, Distance Function and the p parameter for Minkowski Distance.

$$\textbf{Distances} = \{\textbf{Cosine, Minkowski, Mahalanobis}\}$$
$$\textbf{K} = \{\textbf{5,10,30}\}$$
$$\textbf{p for Minkowski} = \{\textbf{2,4,6}\}$$

Results

Figure 1 shows the performances and the model configurations of the experiment. The best performance model is the Minkowski Distance with p=2 and K=10.



```
There are 15 different configurations
Calculating...
Function :    calculateCosineDistance      | Parameter:  None  | K :  5    -->   Confidence interval is [93.4294 - 94.304]
Function :    calculateCosineDistance      | Parameter:  None  | K : 10    -->   Confidence interval is [94.0265 - 95.3068]
Function :    calculateCosineDistance      | Parameter:  None  | K : 30    -->   Confidence interval is [93.696 - 94.5706]
Function :   calculateMinkowskiDistance    | Parameter:   2    | K :  5    -->   Confidence interval is [93.0041 - 94.1959]
Function :   calculateMinkowskiDistance    | Parameter:   2    | K : 10    -->   Confidence interval is [94.8941 - 96.0392]
Function :   calculateMinkowskiDistance    | Parameter:   2    | K : 30    -->   Confidence interval is [92.9638 - 93.7029]
Function :   calculateMinkowskiDistance    | Parameter:   4    | K :  5    -->   Confidence interval is [92.9638 - 93.7029]
Function :   calculateMinkowskiDistance    | Parameter:   4    | K : 10    -->   Confidence interval is [93.3599 - 94.6401]
Function :   calculateMinkowskiDistance    | Parameter:   4    | K : 30    -->   Confidence interval is [92.3627 - 93.2373]
Function :   calculateMinkowskiDistance    | Parameter:   6    | K :  5    -->   Confidence interval is [92.9638 - 93.7029]
Function :   calculateMinkowskiDistance    | Parameter:   6    | K : 10    -->   Confidence interval is [93.9325 - 94.8675]
Function :   calculateMinkowskiDistance    | Parameter:   6    | K : 30    -->   Confidence interval is [92.6471 - 93.2196]
Function :  calculateMahalanobisDistance   | Parameter:  S^-1  | K :  5    -->   Confidence interval is [88.2274 - 89.3726]
Function :  calculateMahalanobisDistance   | Parameter:  S^-1  | K : 10    -->   Confidence interval is [86.2321 - 87.6346]
Function :  calculateMahalanobisDistance   | Parameter:  S^-1  | K : 30    -->   Confidence interval is [78.9415 - 81.3252]
```

Figure 1: KNN Experiment configurations and results of them

Part 2

In this part, I coded both Kmeans and Kmeans++ algorithms. Difference between these algorithms are only in the initialization. The rest of the models, mean moving and loss calculations are all the same.
I trained the methods 100 times in total for each K, and for each algorithm, with batches of 10s. I took the minimum value of each batch, and used them as αs, and used these values to calculate the confidence intervals of the model cluster numbers. These two algorithms gave two different elbow method graphs for each data from the K 2 to 10.

Configurations

I trained the models for K from 2 to 10. This means I had 9 different K's for both Kmeans and KMeans++ models.

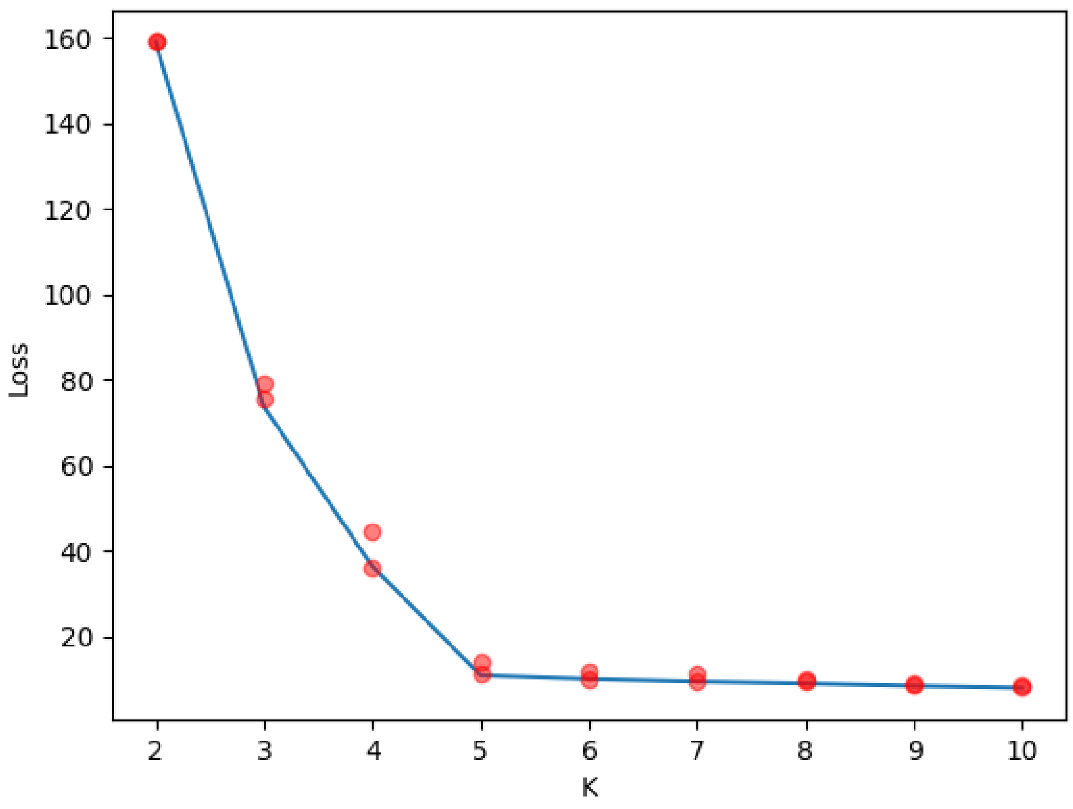


Figure 2: Kmeans Model Losses vs K's for Dataset 1

K versus Loss Graphs

Figure 2 is the elbow method graph of the vanilla kmeans methods. Red dots are the confidence intervals regarding K. Since it is randomly initialized, the intervals are large, but taking the minimum does the job.

Elbow Method: From this graph we can observe that the king is the K =5 since it is the elbow point on the graph. Optimum K for this dataset is 5.

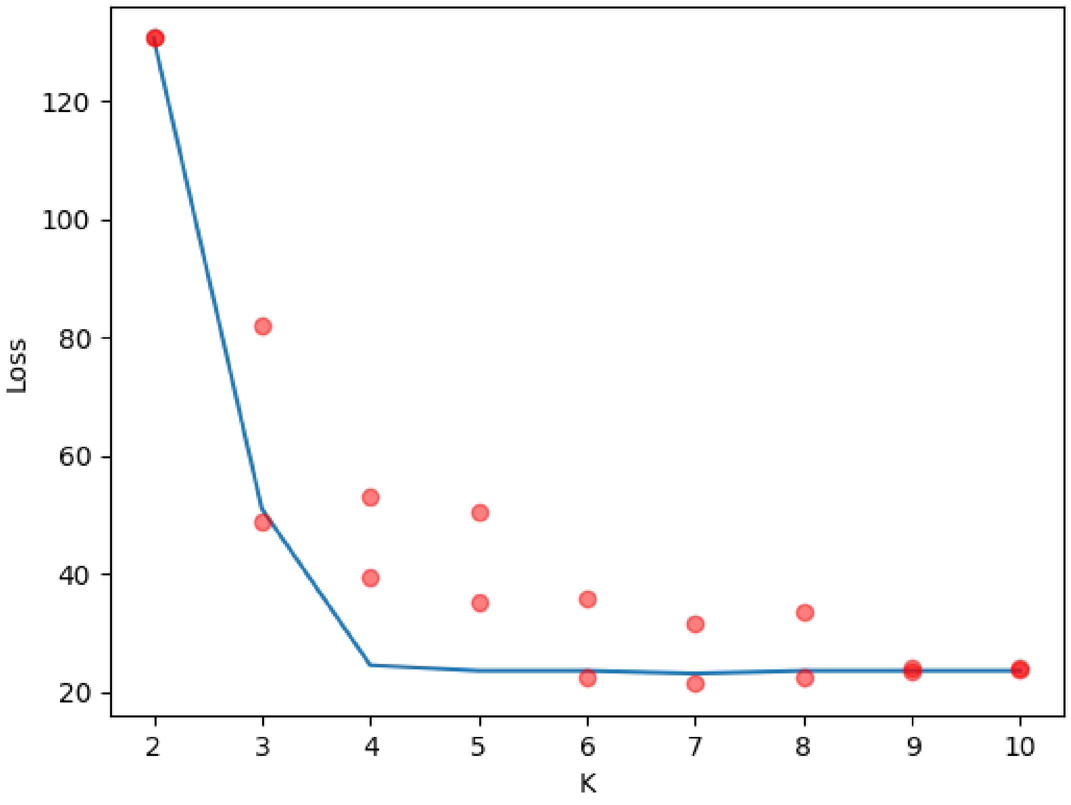


Figure 3: Kmeans Model Losses vs K's for Dataset 2

Figure 3 is the elbow method graph of the vanilla kmeans methods. Red dots are the confidence intervals regarding K. Since it is randomly initialized, the intervals are large, even larger from the dataset2, but taking the minimum does the job. This graph shows that second dataset is more vulnerable to the random initialization.

Elbow Method: From this graph we can observe that the king is the K = 4 since it is the elbow point on the graph. Optimum K for this dataset is 4.

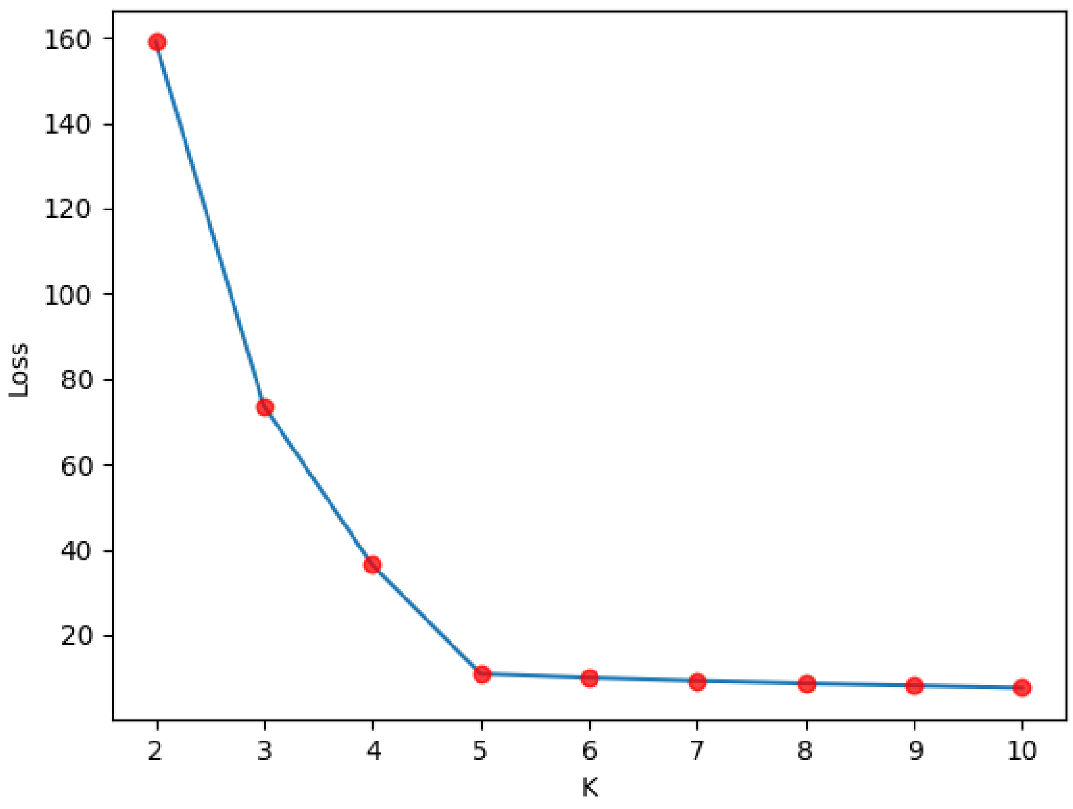


Figure 4: Kmeans++ Model Losses vs K's for Dataset 1

Figure 4 is the elbow method graph of the kmeans++ method. Red dots are the confidence intervals regarding K. Since it is randomly initialized, the intervals are large, even larger from the dataset1, but taking the minimum does the job. This graph's confidence intervals are so different than the vanilla Kmeans method. This is because Kmeans++ initialize the means better. Randomization is not that a big deal for this method.
Elbow Method: From this graph we can observe that the king is the K = 5 since it is the elbow point on the graph. Optimum K for this dataset is 5.

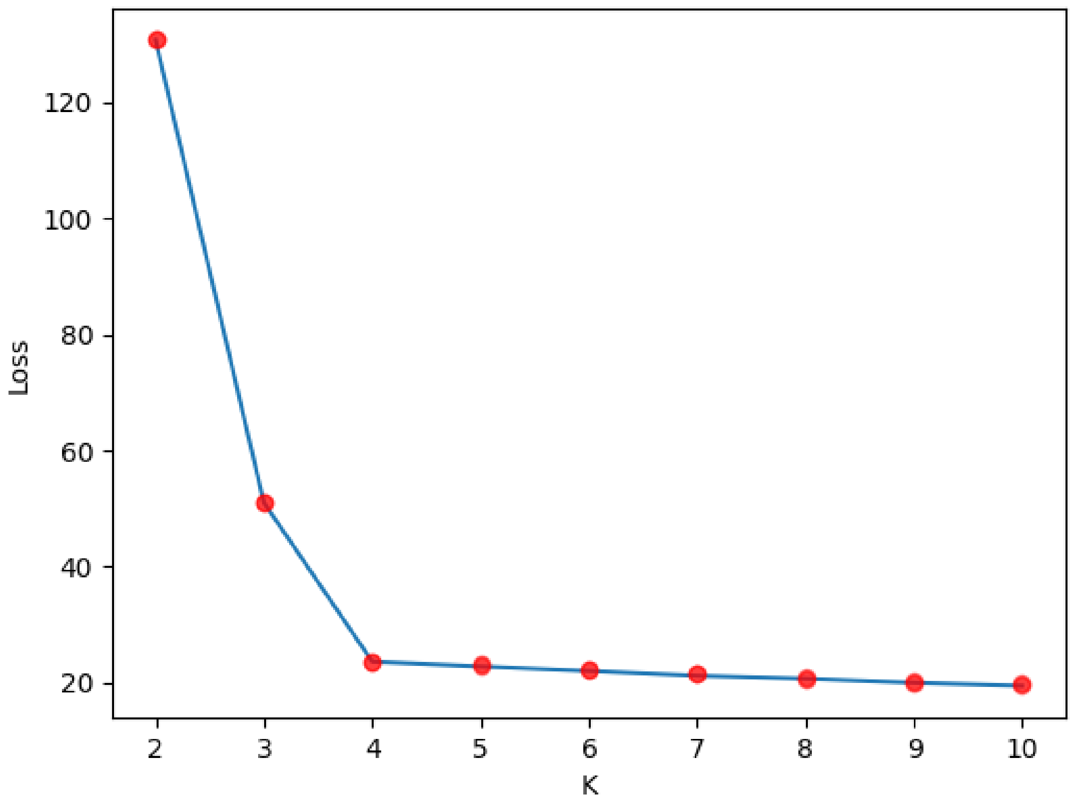


Figure 5: Kmeans++ Model Losses vs K's for Dataset 2

Figure 5 is the elbow method graph of the kmeans++ method. Red dots are the confidence intervals regarding K. Since it is randomly initialized, the intervals are large, even larger from the dataset2, but taking the minimum does the job. This graph's confidence intervals are so different than the vanilla Kmeans method. This is because Kmeans++ initialize the means better. Randomization is not that a big deal for this method.
Elbow Method: From this graph we can observe that the king is the K = 4 since it is the elbow point on the graph. Optimum K for this dataset is 4.

Time Complexity Analysis

The worst time analysis of the Kmeans method is:

$$O(N * K * N)$$

Where N is the sample count, and the K cluster count. For each sample(N) we compute the distance to the clusters(K), and in each iteration of N, we have to move the means to the next point. So we

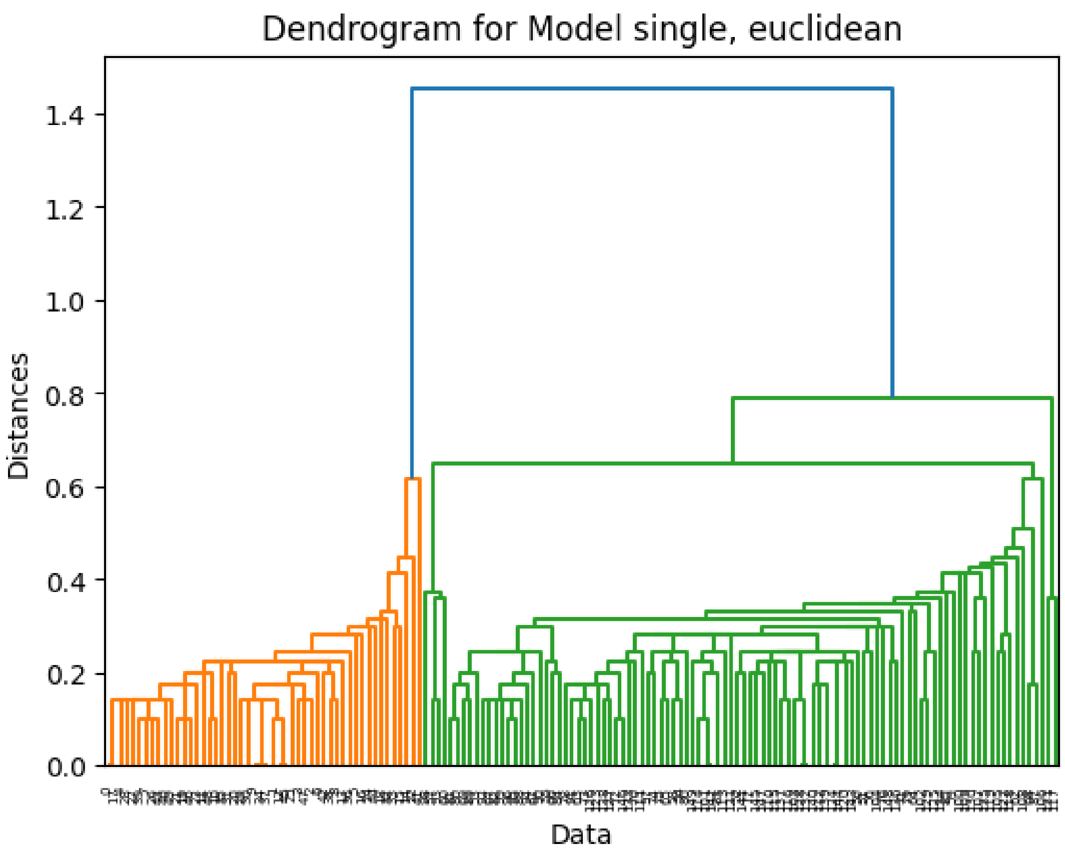


Figure 6: Dendrogram for single euclidian

have to take the average of current presented samples, which is averagely N/2. I assume that all the distance calculations are done in O(1) regardless of the dimensions of the points due to the parallel programming, GPUs.

Part 3

In this part, I trained a HAC model and created dendrograms using them. Here are the dendrograms:

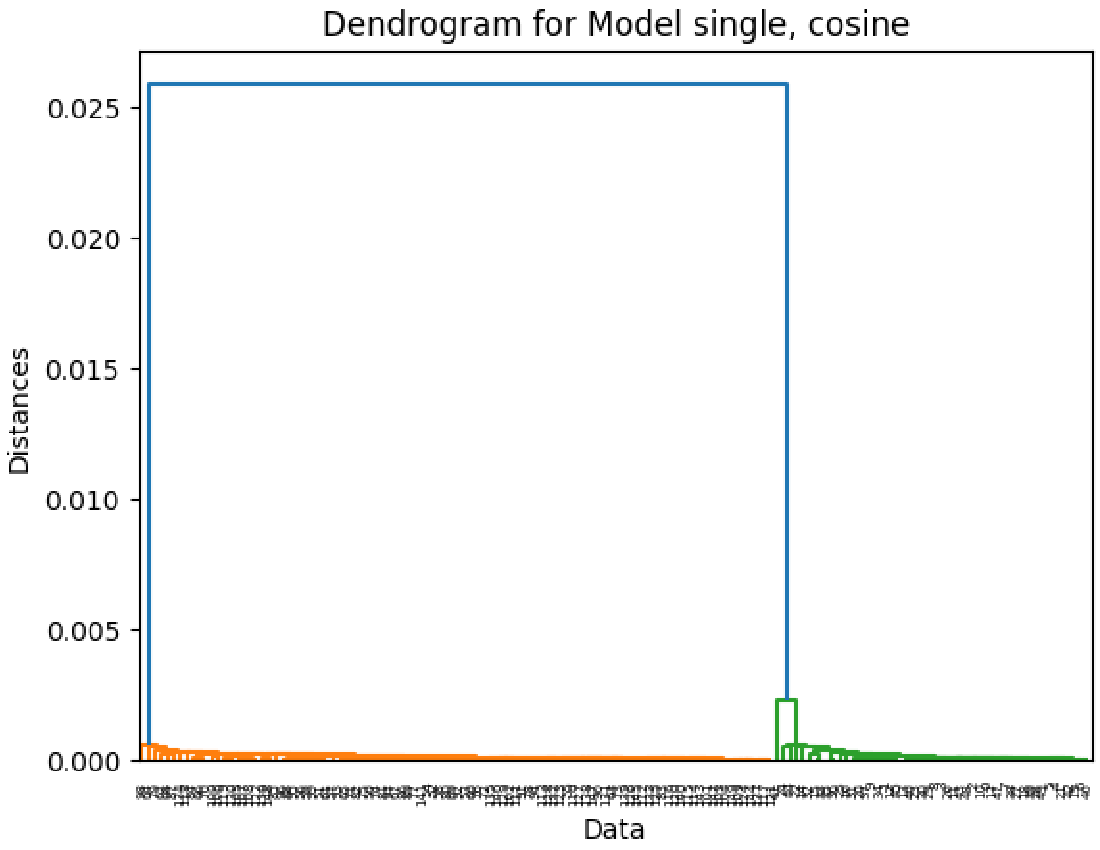


Figure 7: Dendrogram for single cosine

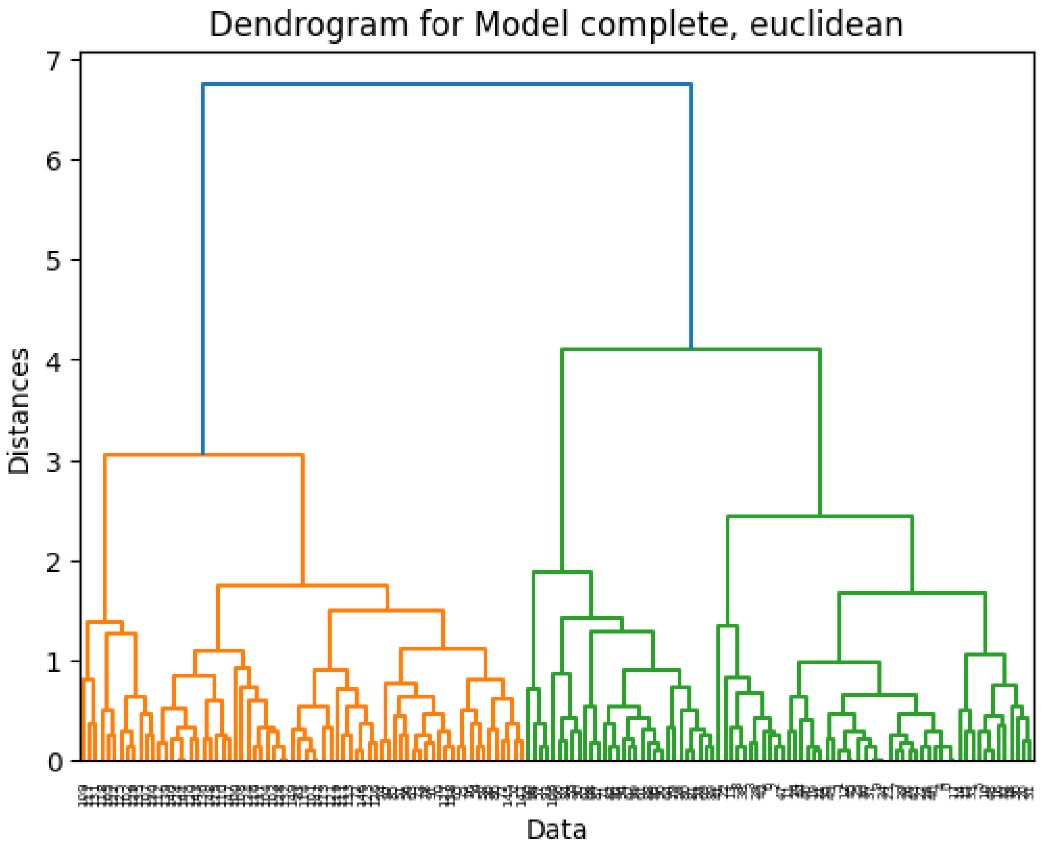


Figure 8: Dendrogram for complete euclidian

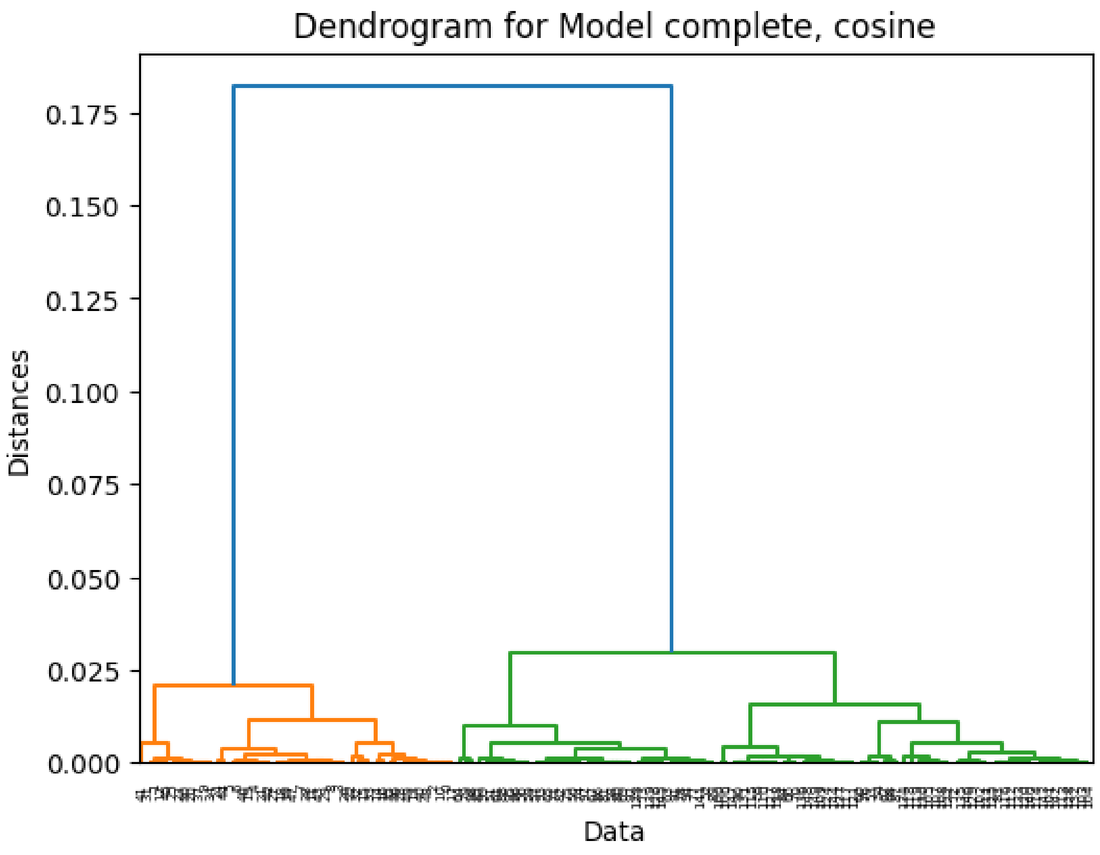


Figure 9: dendrogram for complete cosine

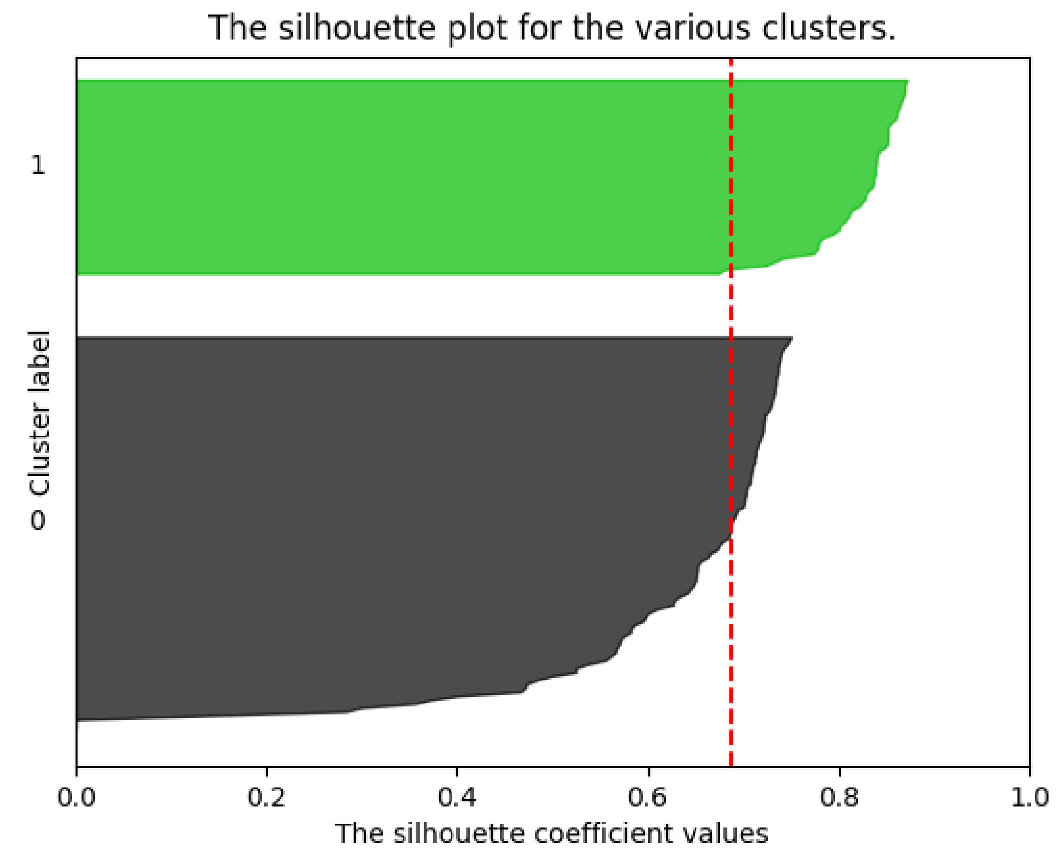


Figure 10: For single, Euclidian K =2

Silhouette Scores

All the below graphs and one above graph shows the silouette scores of the configurations.

Conclusion

Best average score of silhouette is for the configuration complete, euclidian.

Complexity Analyisis

The complexity of the hac algorithm for given case is:

$$N^2 * K * D$$

Where N is the sample count, K is the cluster count, and the D is the dimension of the samples. I assume that the distance calculation is O(1) due to the parallel programing, GPUs. Method

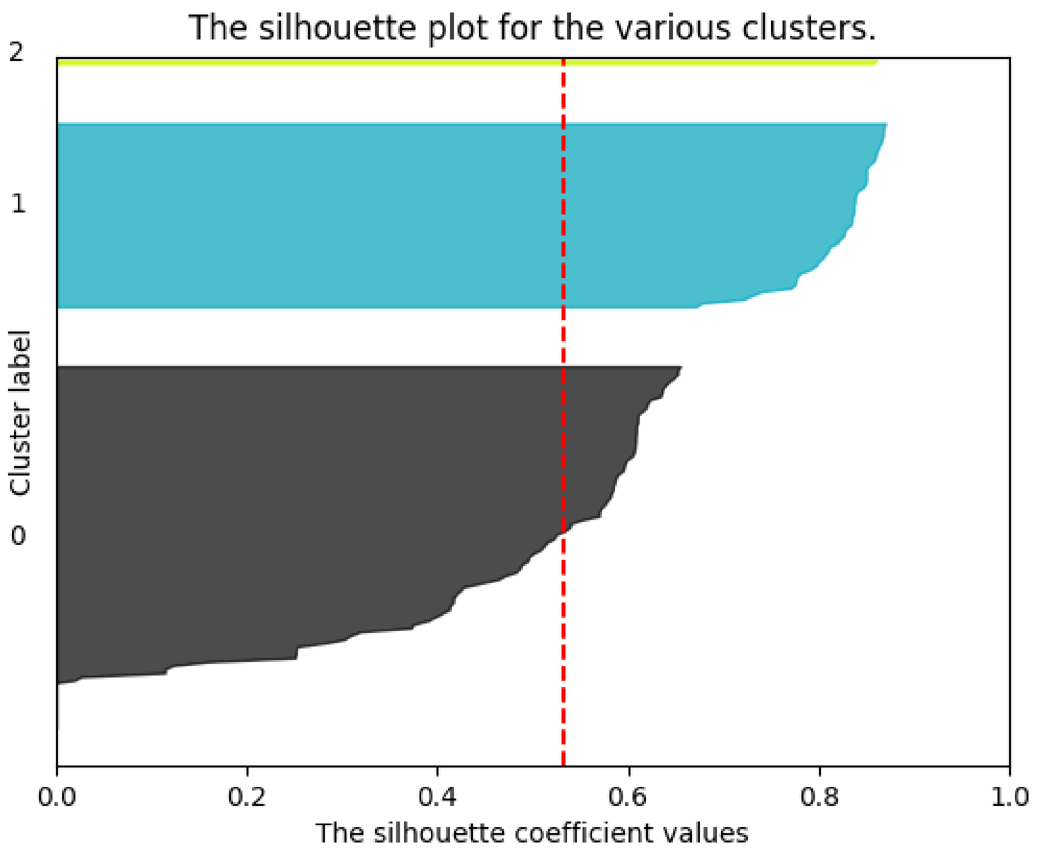


Figure 11: For single, Euclidian K =3

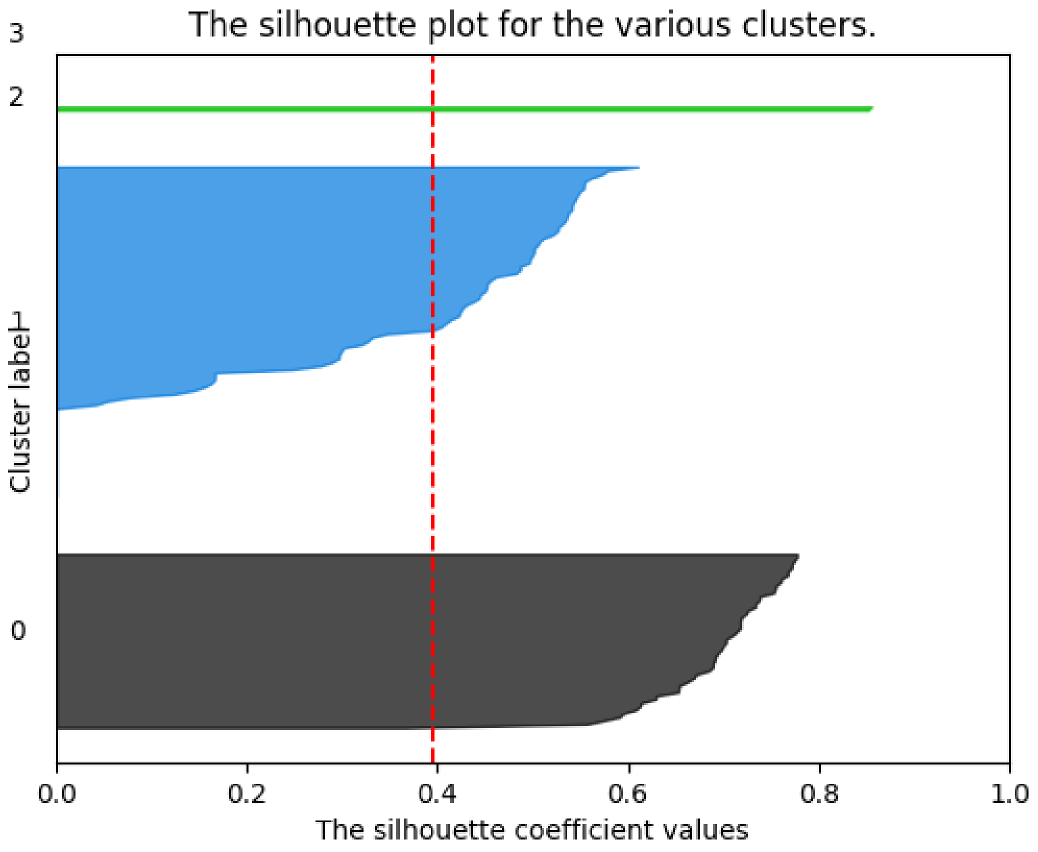


Figure 12: For single, Euclidian K =4

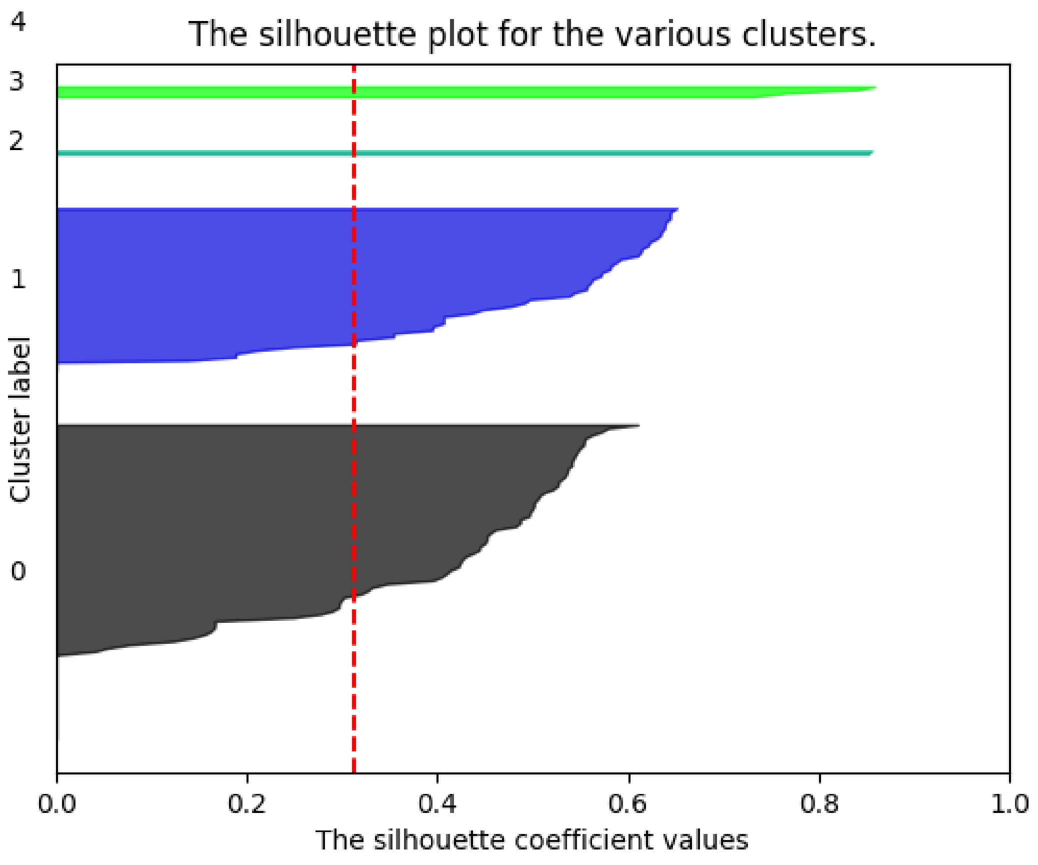


Figure 13: For single, Euclidian K =5

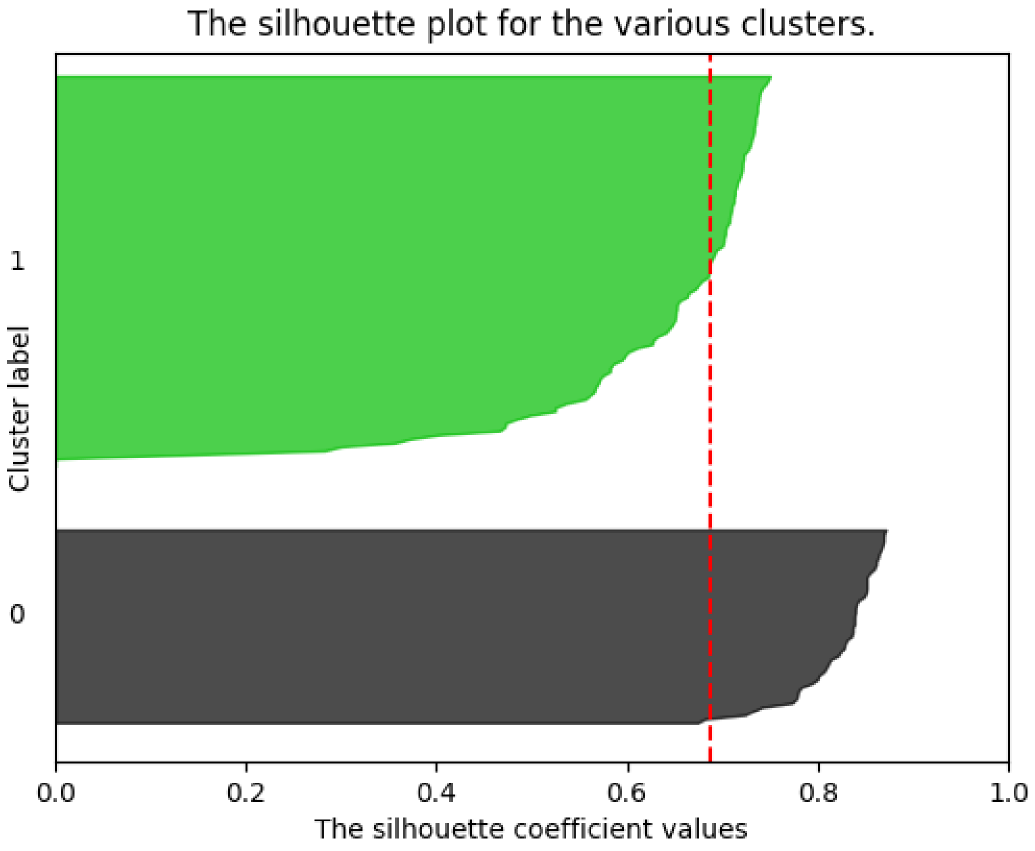


Figure 14: For single, cosine K =2

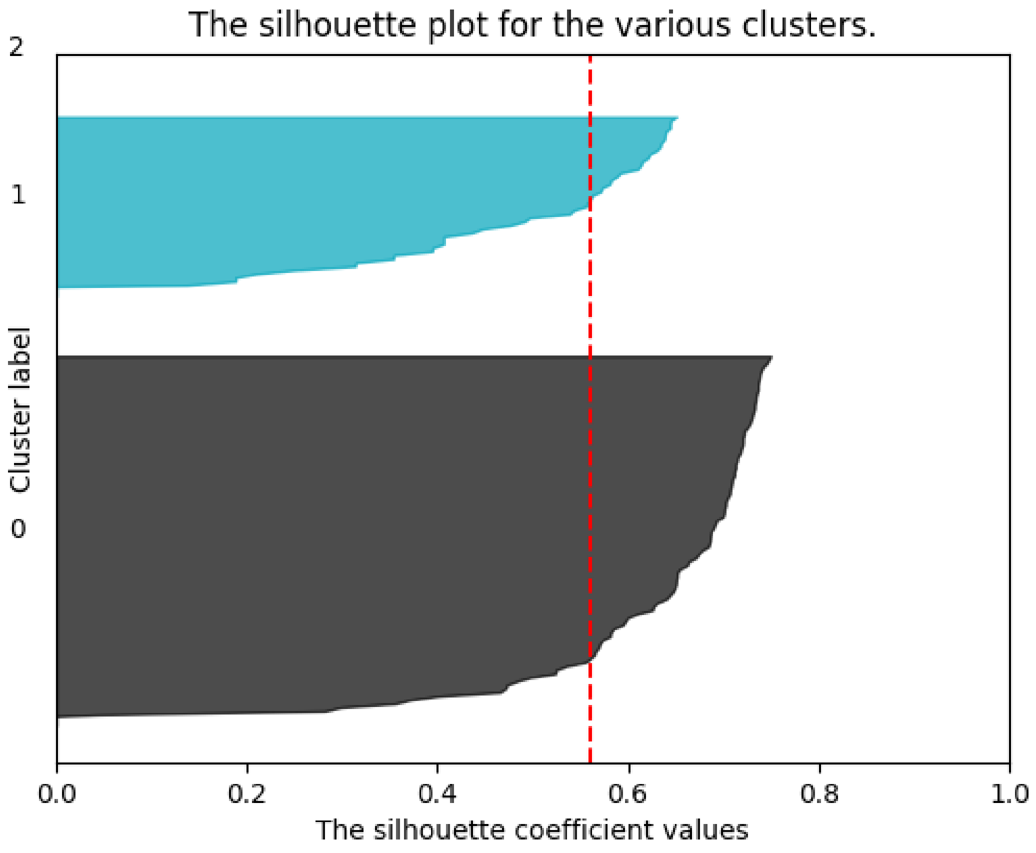


Figure 15: For single, cosine K =3

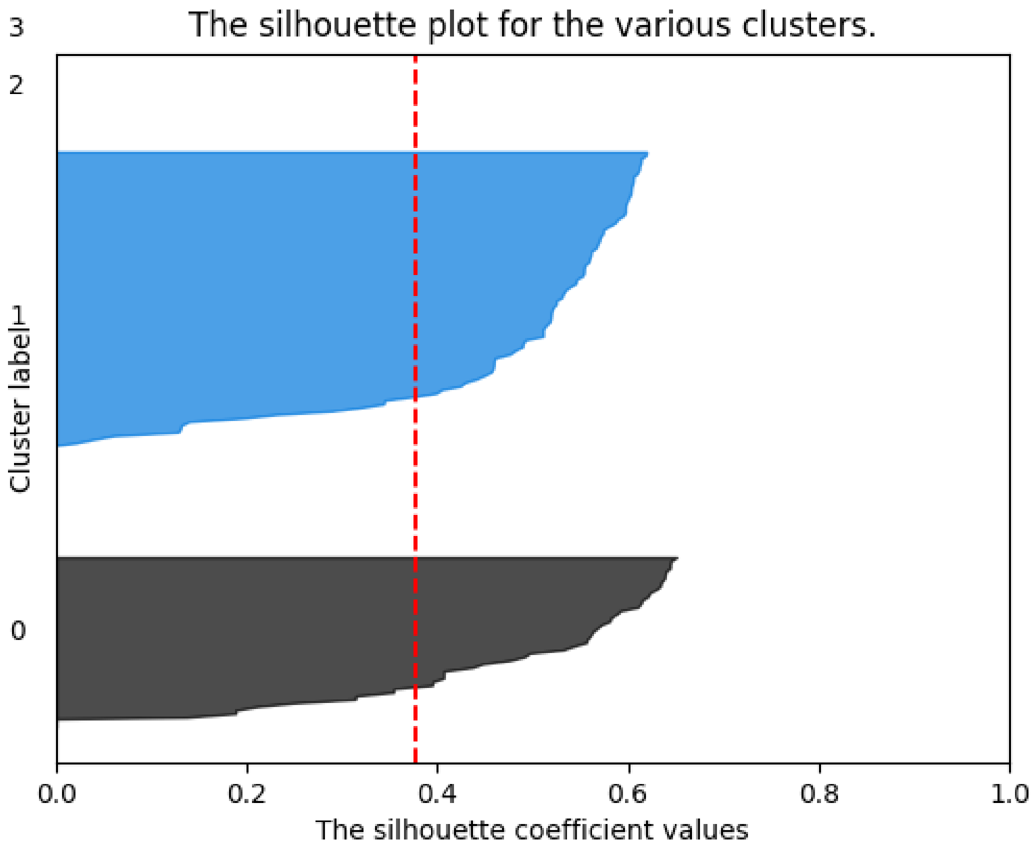


Figure 16: For single, cosine K =4

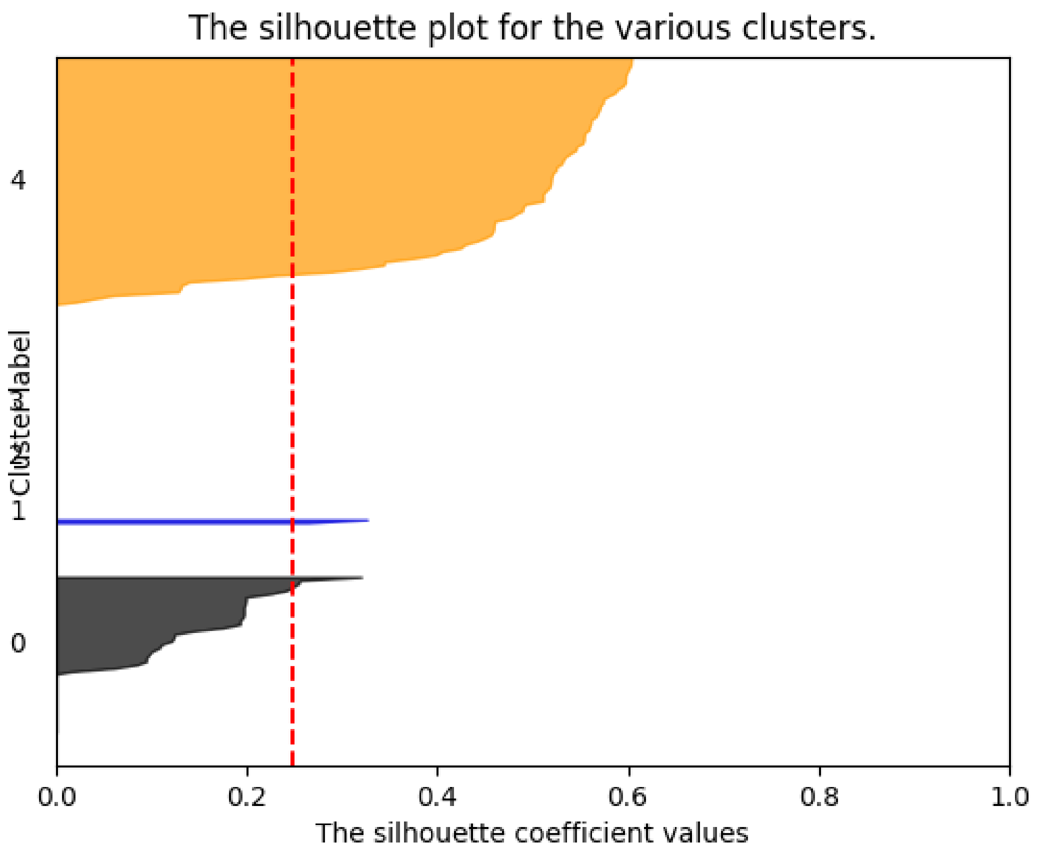


Figure 17: For single, cosine K =5

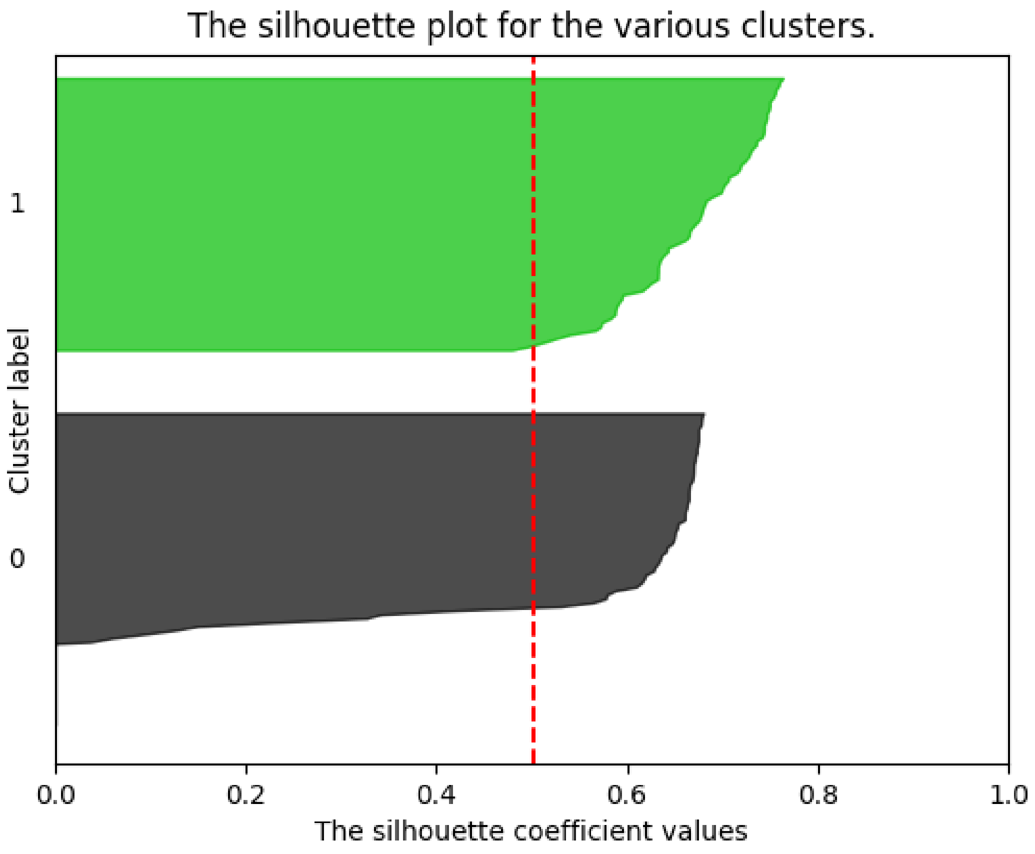


Figure 18: For complete, euclidian K=2

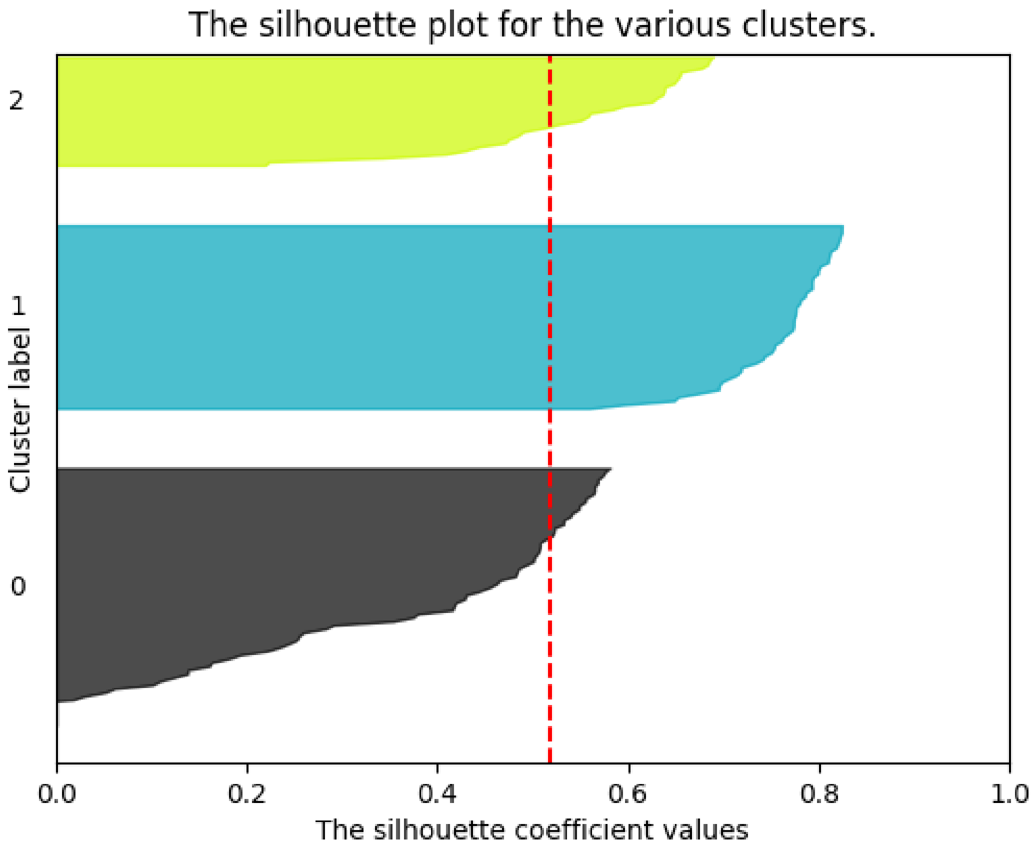


Figure 19: For complete, euclidian K=3

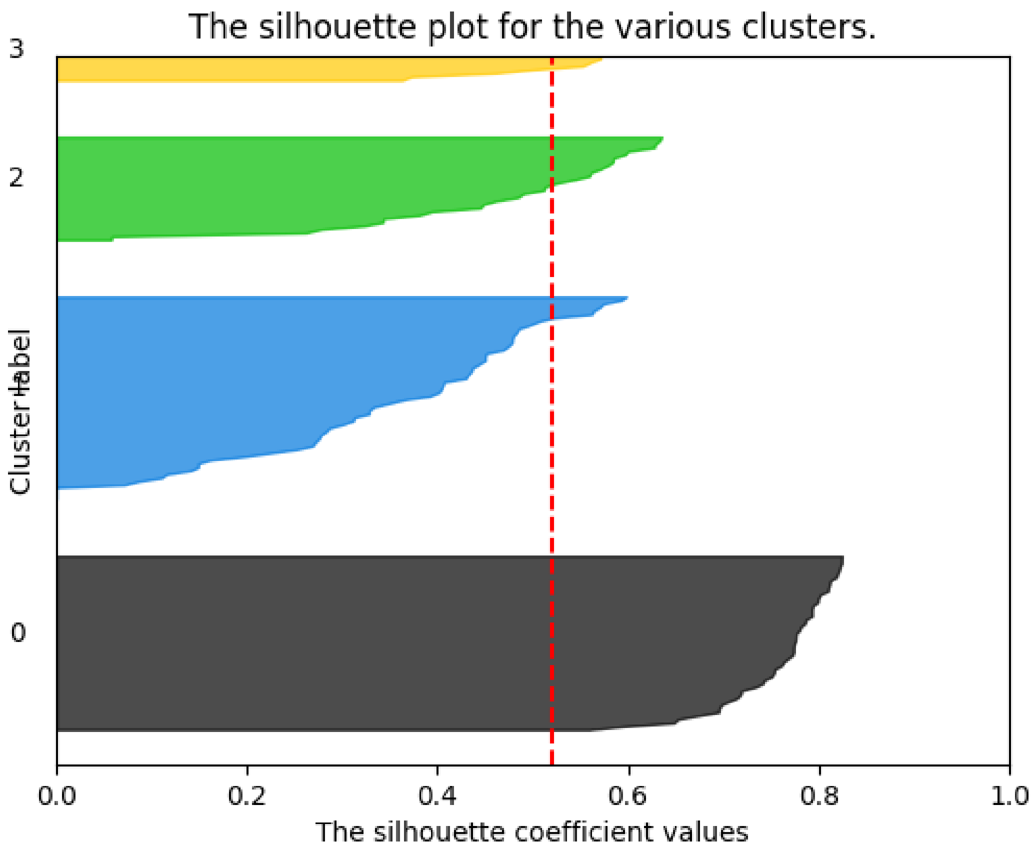


Figure 20: For complete, euclidian K=4

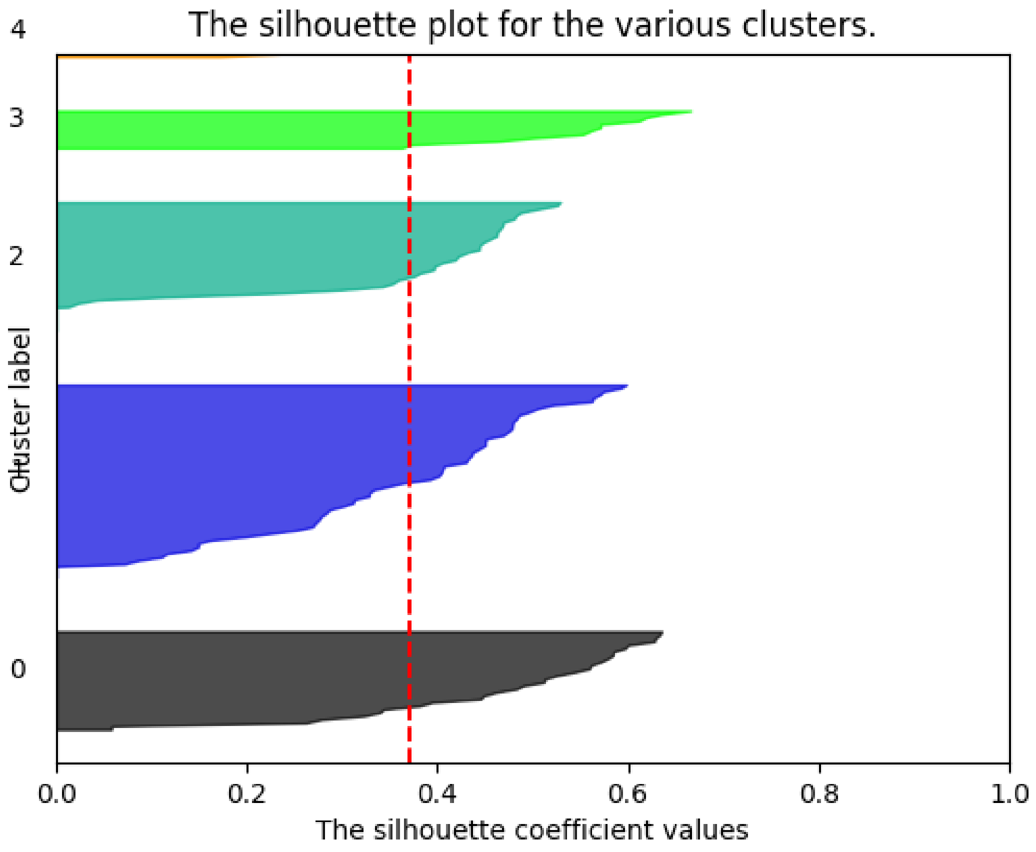


Figure 21: For complete, euclidian K=5

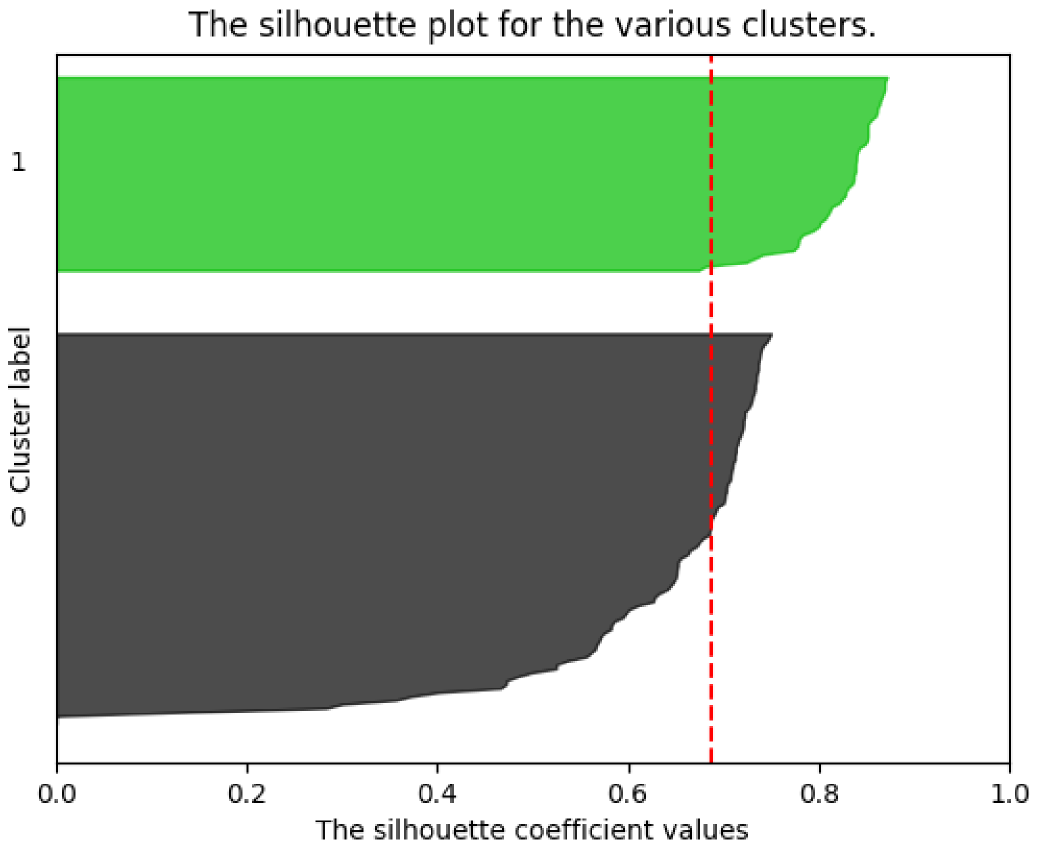


Figure 22: For complete, cosine K=2

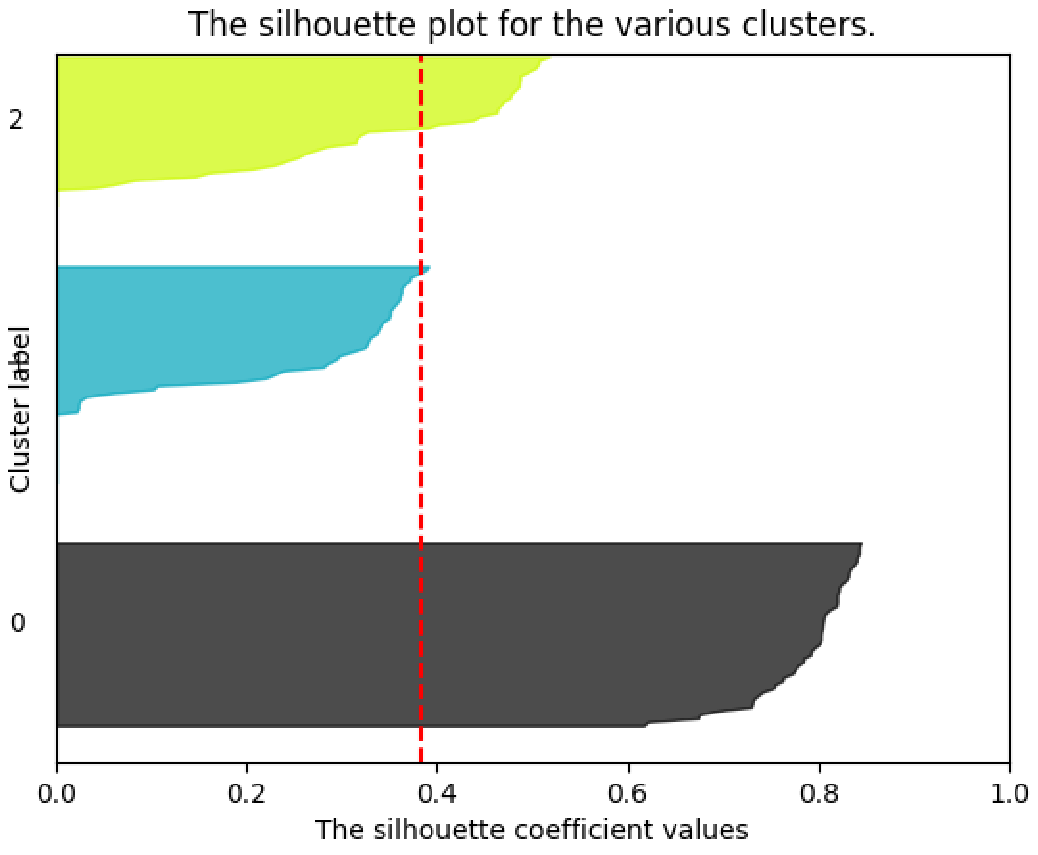


Figure 23: For complete, cosine K=3

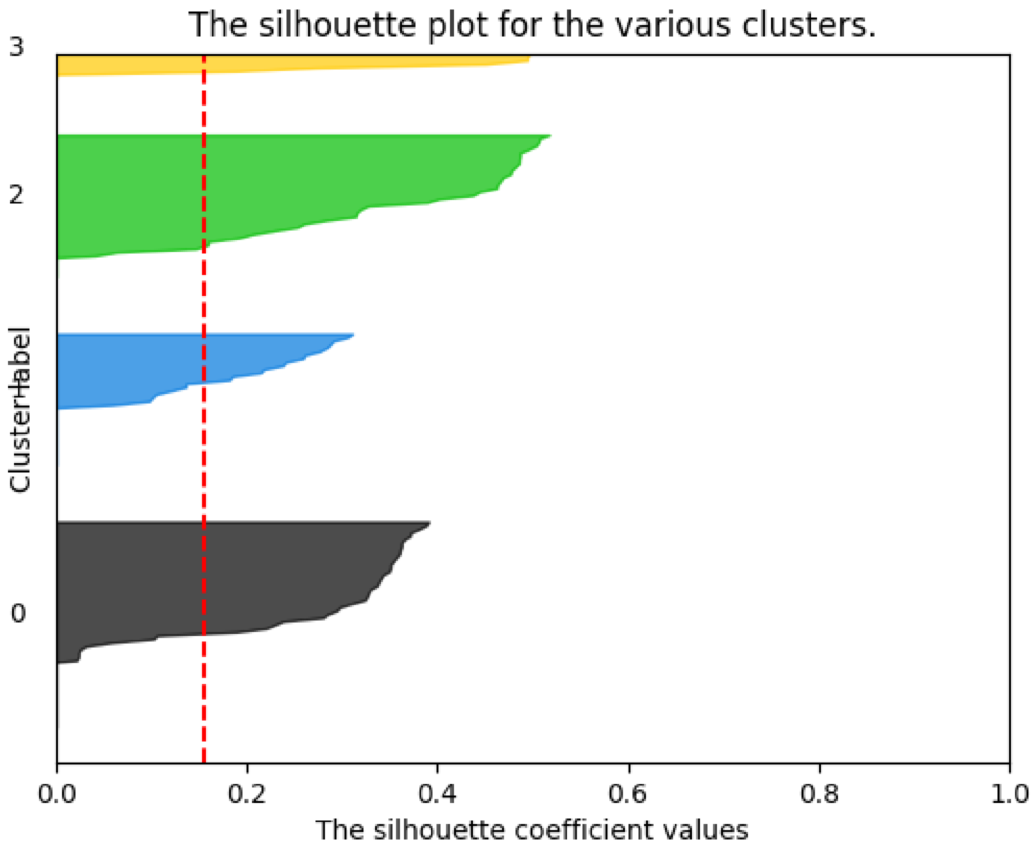


Figure 24: For complete, cosine K=4

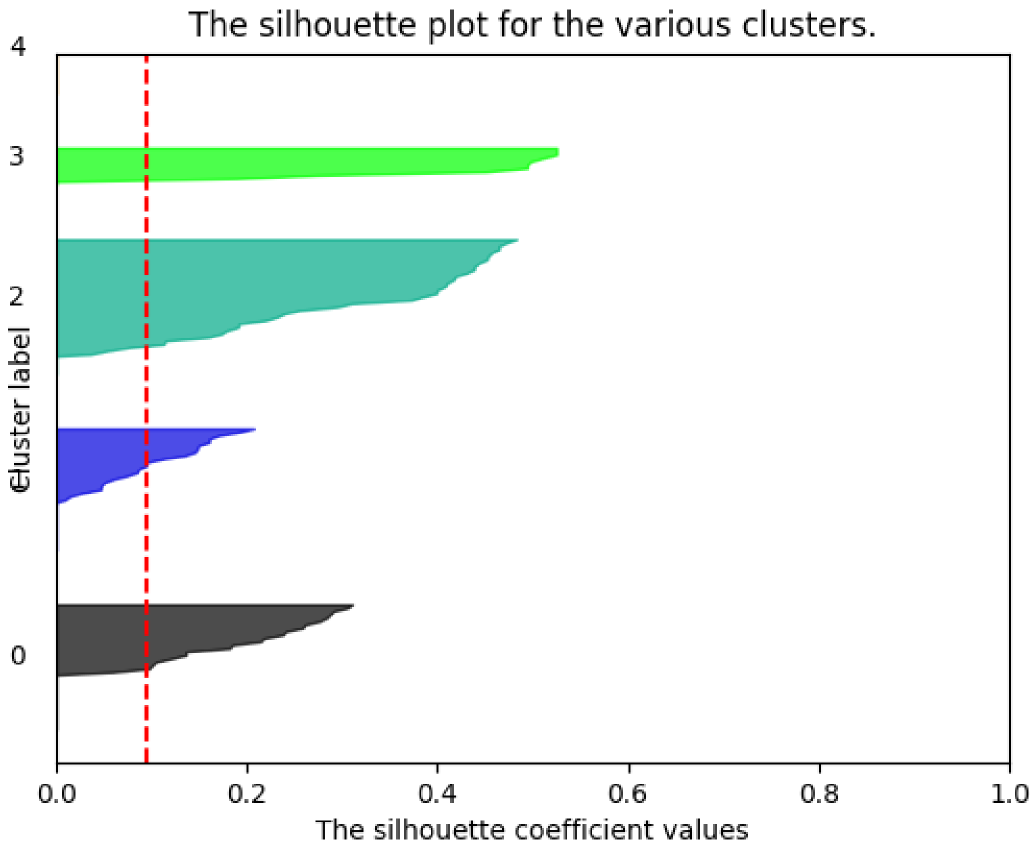


Figure 25: For complete, cosine K=5

selection can be done according to the need of the task. If you want a hierarchical clustering, you would choose hac, otherwise Kmeans.