CS235 Winter'23 Project Final Report

Mamadou Zerbo
NetID: mzerb001

Jason Sadler
NetID: jsadl003

Avinash Lakhmawad
NetID: alakh003

Austin Lee
NetID: alee235

Kanak Das
NetID: kdas006

Srinivasa Biradavolu
NetID: sbira001

ABSTRACT

Our work explores the Breast cancer Wisconsin Diagnostic dataset. The goal is to provide an insight into the various machine learning methods available for classification and clustering problems. We used a variety of supervised and unsupervised method. The project addressed two problem definitions: the first problem is the classification of the data points with the objective of making a determination if it is classified as malignant or Benign, and the second problem focuses on the clustering of the data points into coherent groups. The clustering into coherent groups is based on the similarities.We addressed the problems using different approaches. The approaches taken for the first problem are the Random Forest classifier, the Multi-Layer Perceptron classifier, and the K-Nearest Neighbors. The second problem was addressed using DBSCAN Clustering, Spectral Clustering, and the Agglomerative Clustering with Single Linkage. Our evaluation plan uses Accuracy, Precision, Recall, and F1 score for the classification method, and Silhouette coefficient and Normalized Mutual information measures for the clustering methods.

KEYWORDS

data, mining

ACM Reference Format:
Mamadou Zerbo, Jason Sadler, Avinash Lakhmawad, Austin Lee, Kanak Das, and Srinivasa Biradavolu. 2018. CS235 Winter'23 Project Final Report. In *Proceedings of Data Mining Techniques (CS235 F22).* ACM, New York, NY, USA, 12 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Breast cancer is a significant public health issue that affects millions of women worldwide. Early detection and accurate diagnosis are essential for effective treatment and improved patient outcomes. In this research paper, we explore two distinct problem definitions related to breast cancer diagnosis: classification and clustering.

In the first problem definition, we seek to classify a given breast mass as either malignant or benign based on numerical features extracted from the dataset. We propose three classification algorithms: Random Forest, Multi-Layer Perceptron (MLP), and K-nearest neighbors. These algorithms have been widely used in machine learning applications and have shown promising results in solving similar classification tasks.

In the second problem definition, we aim to cluster N data points into coherent clusters based on the similarity of their numerical features. For this task, we propose three clustering algorithms: DBSCAN clustering, Spectral clustering, and Agglomerative Clustering with Single Linkage. These algorithms have demonstrated their effectiveness in clustering tasks and have been successfully applied in various fields, including healthcare.

In this paper, we present a comparative study of the proposed algorithms for both problem definitions using a publicly available dataset on breast cancer diagnosis. Our goal is to evaluate the performance of each classification algorithm in terms of accuracy, precision, recall, and F1-score. And each clustering algorithm in terms of silhouette coefficient and NMI. The results of our study will provide valuable insights into the strengths and weaknesses of each algorithm.

The rest of the paper is organized as follows: In Section 2, we provide a brief overview of each proposed method. Section 3 describes the methodology used in our study, including the preprocessing steps, feature selection, and experimental design. In Section 4, we present the results of our experiments and discuss the performance of each algorithm. Finally, we conclude our study in Section 5, describing the extent to which each implemented system meets the intended requirements and specifications.

2 PROPOSED METHODS

2.1 MLPClassifier

The MLPClassifier is a popular neural network model for classification tasks. Here is the proposed method for using the MLPClassifier:

(1) Load the dataset: Load the dataset into memory. Ensure that the data is well-preprocessed and cleaned to improve the model's performance.
(2) Split the dataset: Split the dataset into training and testing datasets. This is important to evaluate the model's performance and ensure that it's not overfitting.
(3) Initialize the MLPClassifier: Import the MLPClassifier from the scikit-learn library and initialize the classifier with appropriate parameters. The most important parameters are the number of hidden layers, the number of neurons in each hidden layer, and the activation function.
(4) Train the MLPClassifier: Train the MLPClassifier on the training dataset using the fit() function. During training, the MLPClassifier adjusts its weights to minimize the loss function.
(5) Evaluate the MLPClassifier: After training, evaluate the MLPClassifier's performance on the testing dataset using the

CS235 F22, Fall 2022 quarter,, Riverside, CA
© 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/XXXXXXX.XXXXXXX

predict() function. This function returns the predicted class labels for the input data.

(6) Tune the hyperparameters: To improve the model's performance, experiment with different hyperparameters such as the number of layers, width of each layer, activation function per layer, optimizer, and learning rate. You can use techniques such as grid search or bayesian search to find the best hyperparameters.
(7) Save the MLPClassifier: Once you have found the optimal hyperparameters, save the trained MLPClassifier to disk using the joblib library. This will allow you to reuse the model on new data without having to retrain it from scratch.

Overall, the MLPClassifier is a powerful machine learning algorithm that can handle complex classification tasks. By following the above steps, the MLPClassifier can be used to build accurate and robust classification models.

2.2 RandomForestClassifier

The Random Forest Classifier is used as a method to address classification problems. It works based on multiple decision trees, and combines the predictions, in order to make a final prediction. Here is the proposed method to use the classifier.

(1) Data Preparation: Split the dataset into a training set, and a test set. For the training set, the purpose is to train the model. As for the test set, it is used to evaluate the performance.
(2) Sampling of Data: Take samples from the training set, and those are used to build decision trees. Use feature and data subsets to build the decision trees.
(3) Decision trees: For the training set, a decision tree algorithm is used to construct the tree. A subset of the input features is randomly selected at each node of the tree. Is also chosen, the feature that provides the best split.
(4) Making Predictions: After the decision tree is built, predictions are made across all the trees.The prediction on each tree is based on the subsets of the features that are contained in the input data. For the final decision, it is made based on the maximum number of the best trees.
(5) Evaluation: The performance is evaluated by using the test set. We make a comparison between the predicted values and the actual values in the test set to calculate the accuracy of the model.
(6) Tuning Parameters: There are few parameters in the random Forest classifier that can be tuned to improve the model's accuracy. Among those parameters are the number of trees to build, how many features need to be considered, the appropriate depth of each tree. Cross validation is used to tune those parameters.

Overall , for classification and regression problems, the random Forest Classifier is great tool to use.

2.3 KNN

KNN is a machine learning algorithm that can be used to target classification and regression tasks. It finds out k nearest neighbors in a training dataset for a test data point and based on the label or value of the neighbors, makes a decision about the label or value of the test data point. In this study, we used KNN to classify malignant and benign data points of the breast cancer dataset. To measure the distance between data points, we used two distance functions. Here is the methodology we propose for KNN classification,

(1) Data Preprocessing: First we identify irrelevant features from the dataset and remove them. By irrelevant we mean the features that are not real features that can be used in the training and classification phase where we need to calculate distances. SVD and MLP-based autoencoders are used to reduce the dimensionality of the features to analyze the most important latent features in the dataset.
(2) Distance Calculation: Two different distance functions, euclidean and manhattan distances, are used to measure distances between test and training data points. The goal is to find all distances from the test data point to the training data points.
(3) KNN Prediction: Once the distances are calculated, they are sorted in ascending order. The lowest k values are the k nearest neighbors. Then the most common label among the k neighbors is predicted as the label of the test data point.
(4) Evaluation: The dataset is split into 10 folds. Then in 10 iterations, for each iteration, one fold is considered as a test set and the rest nine are considered as training sets to evaluate accuracy, precision, recall, and f1 score. This procedure is repeated for the original dataset, two SVD-based representations, and two MLP AutoEncoder-based representations with both euclidean and manhattan distance functions.
(5) Tuning Parameters: The 10-fold cross-validation is performed for different values of k. Average accuracy, precision, recall, and f1 scores are recorded for each of these runs. The best-performing value of k, the distance function, and the used dataset is selected for reporting in this study.

2.4 DBSCAN

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) is a popular clustering algorithm used in machine learning and data mining. It is particularly effective in identifying clusters of varying shapes and sizes in data sets. It has two hyperparameters, $epslon$ and min_pts. Epslon is the radius of the objects in neighbourhood and min-pts is the minimum number of points required to be in the neighbourhood for it to be considered as a core point

(1) Data Preprocessing: Dropped the *id* column as they were numbers irrelevant to the other labels. Dropped the $Unnamed:$ 32 column as it was an empty column that will give unnecessary nan values upon creation of feature matrix. Dropped the *diagnosis* column as it is the independent variable and we only want to analyze the clusters and try to improve the quality of them to study the structure of the data. Used Standard Scalar normalization from *sklearn.preprocessing* library to create the feature matrix. We use the features
(2) Experimental Setup: For the off the shelf library implementation, I iterated epslon from $2 - 12$ with an interval of 1 and computed silhuoette and calinski scores and generated plots with them to find which values of epslon would provide better results. I know that the choice of points each time the algorithm is run is random, but I have not implemented the randomization in my implementation of the algorithm as I

was getting more worse silhouette and calinski scores. As a result I used the parameter values of the off the shelf library implementation and computed metrics with it

(3) Evaluation: The Silhouette coefficient is a measure of how well each data point fits within its assigned cluster, based on its proximity to other points in the same cluster compared to points in other clusters. The Calinski-Harabasz Index is another metric used to evaluate clustering quality, which measures the ratio of the between-cluster variance to the within-cluster variance.

2.5 Spectral Clustering

Spectral Clustering is a clustering method that uses a non-linear dimensionality reduction technique that maintains the closeness of data points to process the data into a form that is more suited to clustering methods. Here is the proposed method for using Spectral Clustering:

(1) Data Preprocessing: Ensure that the data is preprocessed and cleaned to improve the model's performance. This includes removing any irrelevant features.
(2) Graph Construction: If the data is not in the form of a graph, construct an unweighted undirected similarity graph. Two nodes or datapoints are connected with an edge if either of the datapoints has the other datapoint among the 2 nearest neighbors.
(3) Laplacian: Calculate the adjacency matrix based on the similarity graph. Calculate the degree matrix from the adjacency matrix by calculating the degree of each vertex and placing it across the diagonal of the matrix. Calculate the Laplacian matrix by inputting the degree matrix and adjacency matrix into the formula Laplacian $\bar{\text{D}}$egree Matrix - Adjacency Matrix. The normalized Laplacian matrix can be calculated with the formula $NormalizedLaplacian = DegreeMatrix^{(-1/2)} * LaplacianMatrix * DegreeMatrix^{(-1/2)}$.
(4) Spectral Embedding: Apply eigenvalue decomposition on the Normalized Laplacian to receive the eigenvalues and eigenvectors. Take the two eigenvectors that correspond to the k lowest non-zero eigenvalues, and create a matrix that uses these k eigenvectors as the columns. The spectral embeddings will be the rows of this matrix. K is a hyperparameter that can be chosen by the user by slightly modifying the code.
(5) K-means: Apply K-means with specified k on spectral embeddings in order to output the final centroids and assignments.
(6) Evaluation: Take the outputted centroids and assignments and use scikit-learn's silhouette_score and normalized_mutual_information_score functions to calculate the silhouette and NMI scores. These scores can be used to measure the performance of the model.
(7) Hyperparamter Tuning: Experiment with different numbers of clusters in order to maximize the performance of the model.

2.6 Agglomerative Clustering with Single Linkage

Single linkage Agglomerative clustering uses the minimum distance between two points to form a hierarchical clusters. Initially individual two points are treated as clusters.

(1) Data Pre-processing: This includes filling empty data points with appropriate/approximate data or removing the entry altogether.
(2) Scatter Plot: Visualizes the spread of the input data
(3) Distance Matrix: Calculates the minimum euclidean distance between all points.
(4) Iterative Cluster Forming: Using the above distance matrix, choose two points with minimum distance. This is a new cluster. Recalculate the distance matrix with distance of each point from the new cluster. The new distance is minimum distance between any point from the cluster and the corresponding point.
(5) Evaluation: The dataset is split in 10 subsets. The of-the-shelf silhoutte and NMI functions are used to calculate the individual scores of the subsets. Later an average and standard deviation is measures using scores of individual subsets.

3 EXPERIMENTAL EVALUATION

Dataset:
We used the Wisconsin Diagnostic Breast Cancer (WDBC) dataset, which contains 569 instances with 30 features each. The dataset is labeled, with 357 benign and 212 malignant cases.
Experimental Setup:
We conducted 10-fold cross-validation experiments and measured the performance of each method in terms of accuracy, precision, recall, and F1-score to evaluate the methods' ability to distinguish between benign and malignant cases.
Instead of conducting 10-fold cross-validation experiments, for spectral clustering, we ran 10 experiments with different random initial centroids and assignments and obtained the average Silhouette Coefficient and Normalized Mutual Information score to evaluate the performance.
Results:
Table 1 shows the average performance metrics of the three classifier methods.

Method	Accuracy	Precision	Recall	F1-Score
Random Forest	0.96	0.88	0.97	0.93
MLP	0.95	0.95	0.91	0.95
KNN	0.93	0.95	0.89	0.92

Table 1: Comparisons of classification methods

Table 2 shows the average performance metrics of the three clustering methods.

Method	Silhouette Coefficient	NMI
DBSCAN	0.384 ± 0.066	-
Spectral	0.7001 ± 0.1792	0.1285 ± 0.0660
Single Linkage	0.4701 ± 0.1366	0.154 ± 0.0

Table 2: Comparisons of clustering methods

4 DISCUSSION & CONCLUSIONS

4.1 Classification

For Random Forest, the precision was 0.88, the recall was 0.97, and the F1-score was 0.93. This indicates that the algorithm had a high recall, meaning it was able to identify most of the actual positive samples, but had a relatively lower precision, indicating that it also labeled some negative samples as positive.

For MLP, the precision was 0.95, the recall was 0.91, and the F1-score was 0.95. This indicates that the algorithm had a high precision and F1-score, indicating that it was able to identify most of the actual positive samples and had a low false positive rate.

For KNN, the precision was 0.95, the recall was 0.89, and the F1-score was 0.92. This indicates that the algorithm had a high precision but lower recall, indicating that it missed some of the actual positive samples.

In conclusion, all three algorithms performed well, with Random Forest and MLP performing slightly better in terms of accuracy and precision, while KNN had a higher false negative rate, leading to a lower recall and F1-score.

4.2 Clustering

For DBScan clustering, the off the shelf library implementation had a silhouette score of 0.346 and the calisnki score was 84.64 for the default dataset. When it comes to the from the scratch implementation, the calinski score was 45.45, sometimes tuning the hyperparameter such that getting a better silhouette score is preferred as we would then know if the data points are tightly grouped together which ensures we can rely on the analysis that we make.

For Spectral Clustering, the Silhouette Coefficient is 0.7001 with a standard deviation of 0.1792. The Normalized Mutual Information score is 0.1285 with a standard deviation of 0.0660. The relatively high Silhouette Coefficient indicates that my model does a good job in creating well-defined clusters, however, a low Normalized Mutual Information score may indicate that the dataset features are not closely related/linked with the diagnosis column.

For Single Linkage Agglomerative Clustering, the average Silhouette score is 0.47 with standard deviation of 0.13. Also, the average NMI score is 0.15 with no standard deviation. Silhouette score is progressing towards 1 indicating some overlapping clusters. Low standard deviation suggests the model performs consistantly. This is expected due to the deterministic nature of the single-linkage clustering.

In conclusion, Spectral Clustering does better job at forming more cohesive clusters. However, low NMI scores in Spectral and Single linkage indicate the high uncertainty in cluster formation.

5 IMPLEMENTATION CORRECTNESS REPORT

5.1 Random Forest Classifier

Random Forest classifier was trained on the artificial dataset. Cross validation was used to tune the hyperparameters of the Random Forest classifier. K-fold cross validation was performed on the training data for each combination of the hyperparameter. The training data is divided into a training set, and a validation set in each fold of the cross validation. On the training set, Random forest classifier is trained using the current hyperparameters.It is then used on the validation set to make prediction. The accuracy of the classifier used on the validation set is computed. Next, as a performance metric for the hyperparameters, the average accuracy across all the folds is taken.

5.1.1 Best hyperparameter.

To optimal hyperparameter is chosen among the hyperparameter that gives the highest average accuracy. The best hyperparameter is then used to train the ramdom foresst classifier on the training set. to evaluate the performance of the classifier, a confusion matrix is used along some performance metrics such as the accuracy, precision, recall and F1 score. Figure 1 displays the performance metrics of the classifier. Table 2 shows different values used to track the performance of the model.

Values	Training Accuracy	Validation Accuracy	Test Accuracy
0.001	0.80	0.78	0.76
0.01	0.85	0.81	0.79
0.1	0.90	0.82	0.81
1	0.95	0.75	0.77
10	0.97	0.77	0.72

Table 3: Hyperparameters Performance evaluation

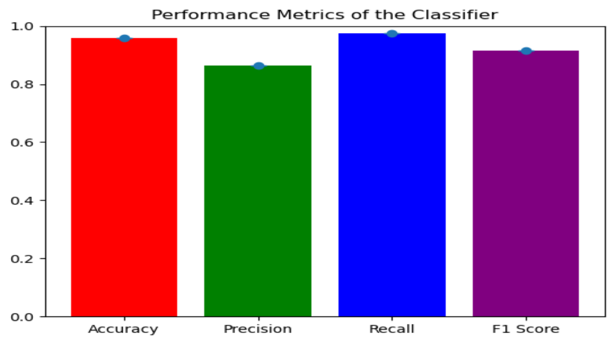

Figure 1: Random forest classifier performance metric

5.2 MLP

To obtain the best hyper-parameters two search strategies were implemented. A grid search provided by scikit-learn which we call MLP_Grid, and a Bayesian search provied by scikit-optimize which we call MLP_BO.

In performing the grid search, up to four layers are examined. The width of each layer ranged from 30 to 300 in multiples of 60. This provided a dimensionality change of zero upto ten times the original dimensinality of the data. For testing more than one layer, permutations of the range on the layers used. This allowed features to shrink and expand as it moved through the network. A total of 205 layer/width permutations were tested.

In performing the Bayesian search, the same range and number of layers were used as in the grid search. The layers widths are sampled randomly and no explicit widths are provided. Table 4 provides additional parameters tested for each layer permutation for both searches

activation	solver	learning_rate
logistic, relu, tanh	sgd, adam	constant, adaptive

Table 4: MLP Search parameters

5.2.1 *Dimensionality Reduction.*

The study explored two methods to examine the performance effects of dimensionality reduction on the data.

The first method involves using a neural network to gradually reduce the dimensions of the data. This approach was similar to the grid search method, where up to four layers were evaluated, and each layer was reduced in size from 29 to two, in increments of three. To ensure a funnel shape, the combinations were sorted instead of being permuted. A grid search was conducted using the parameters listed in table 3, and this approach was named MLP_Funnel.

The second method involved using Singular Value Decomposition (SVD) on the data. Figure 2 displays the output of SVD applied to the data using *numpy.linalg.SVD*. The figure indicates that approximately four features in the dataset contain the most significant information. To achieve a consistent comparison with the MLP_Funnel results, we selected the number of principal components identified by the MLP_Funnel search and used these features as input into the best MLP estimators obtained through grid search and Bayesian optimization stated earlier. We then compared this approach with the MLP_Funnel results.

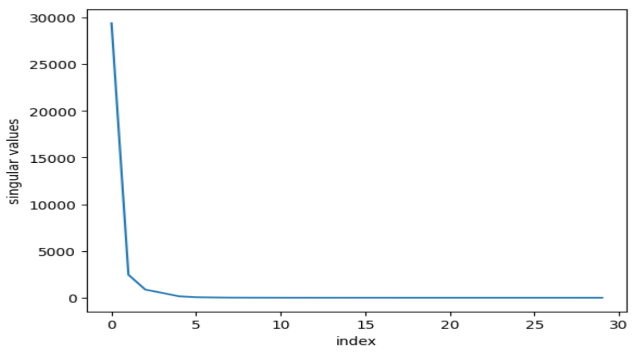

Figure 2: SVD

5.2.2 *MLP Correctness Results.*

The results of the hyper-parameter search are presented in Table 5. It can be observed that the Adam optimizer outperforms stochastic gradient descent. Additionally, it appears that the tanh activation function is the least useful for this dataset. Interestingly, there is no consensus on the optimal shape of the network.

Figure 3 shows the results of accuracy, precision, recall, and F1-score for the optimized MLP classifiers along with the results of dimensionality reduction. With the exception of MLP_Grid, all the classifiers have a higher precision than recall. This means that the classifiers are generally more conservative in its predictions, and tends to make fewer false positive errors at the cost of missing some true positives. In other words, the classifiers tend to be more selective in identifying positive instances, and is less likely to produce false positive results.

Performing dimensionality reduction on data has a significant impact on performance, as observed in this scenario. Linear transformation techniques such as SVD may result in only minor performance reduction, as seen with MLP_BO, while causing a complete collapse of classifier performance with MLP_Grid in the worst case. The use of a series of nonlinear transformations in crafting the architecture, such as with MLP_Funnel, leads to the best performance of all classifiers. This suggests the presence of non-linear relationships within the data that cannot be captured through linear methods like SVD.

Method	Layers	Width	Activation	Optimizer	Learning Rate
MLP_Grid	4	(150,270, 30, 210)	logistic	adam	adaptive
MLP_BO	1	300	relu	adam	adaptive
MLP_Funnel	2	(26, 2)	logistic	adam	constant

Table 5: MLP Optimal Hyper-parameters

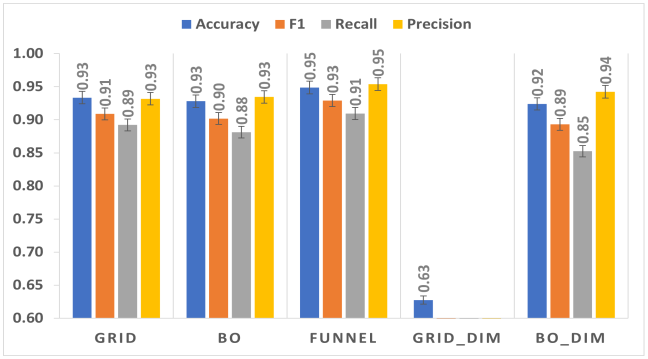

Figure 3: MLP Evaluations

5.3 KNN

5.3.1 *KNN Lower Rank Approximation.*

Two different techniques have been used to represent the features in a lower dimension. SVD has been performed using *scipy.linalg.svd* from SciPy. For MLP-based Autoencoder, *tensorflow.keras.layers* and *tensorflow.keras.models* have been used. Figure 4 plots the singular values of SVD decomposition. Figure 5 and figure 6 draw scatter-plot of the low-rank representations for the low and high-value ranks of 1 and 3 respectively.

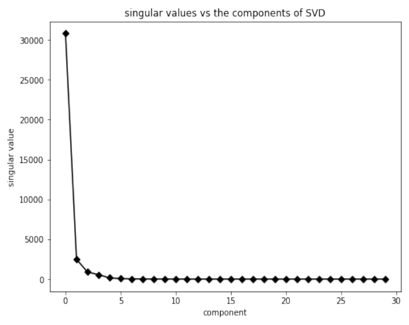

Figure 4: KNN: Semi-Logarithmic plot for Singular Values

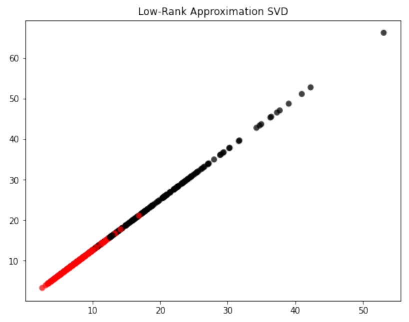

Figure 5: 2-d scatter plot for two principle components of approximated low-value rank of 1 using SVD

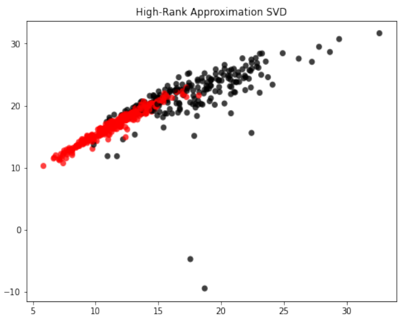

Figure 6: 2-d scatter plot for two principle components of approximated high-value rank of 3 using SVD

Figure 7 and figure 8 draw scatter-plot of the low-rank representations for 5% and 20% of the original features respectively.

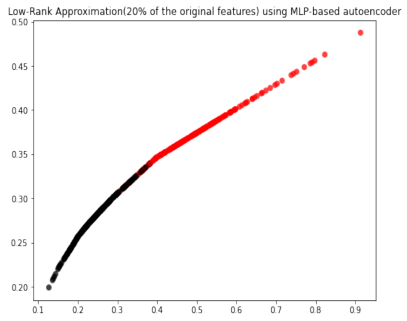

Figure 7: 2-d scatter plot for low-rank approximation (5% of original features) using MLP-based AutoEncoder

These representations are used for hyperparameter tuning, later on, to assess the best-performing distance function, k value, and dataset representation.

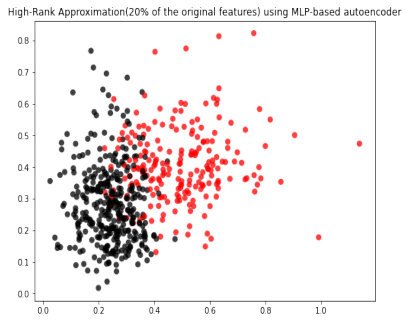

Figure 8: 2-d scatter plot for high-rank approximation (20% of original features) using MLP-based AutoEncoder

5.3.2 KNN Hyperparameter Tuning. Euclidean and Manhattan distance functions were evaluated with different k values ranging from 3 to 30 on the original dataset, two SVD-based representation datasets, and two AutoEncoder-based representation datasets. The average accuracy, precision, f1-score, and recall scores are presented in Table 6, 7, 8, 9, and 10. The highest value for each of the metrics is colored blue in each of the tables. This represents which distance function and k value worked best for each of the datasets we used.

Among all the datasets, the best average recall is obtained from MLP AutoEncoder-based representation that takes 5% of the original features with k=9. For this dataset, for each k value, euclidean and manhattan distance functions perform the same. However, we get the best average f1-score in the original dataset with k=12 and the manhattan distance function. This also performs second best in terms of average recall and average precision, and best in terms of average accuracy.

For the lower-rank representations of SVD and MLP-based AutoEncoder, the euclidean distance function and manhattan distance function performed the same in terms of all evaluation metrics and each k value which can be seen in table 7 and 9. This makes sense since the lower dimensional representations are linear for

Neighbors and Distance Func.	Avg. Accuracy	Avg. Precision	Avg. Recall	Avg. F1
3 Neighbor Euclidean Distance	0.9175549450549452	0.9106574014221074	0.8724459116595549	0.8871027631063783
3 Neighbor Manhattan Distance	0.923956043956044	0.9214492963177173	0.8748685464959829	0.8939461894635616
5 Neighbor Euclidean Distance	0.923956043956044	0.9169973544973544	0.8770907687182051	0.8936337202153408
5 Neighbor Manhattan Distance	0.9275274725274725	0.9297712336396546	0.8783168223580518	0.8997597593359551
7 Neighbor Euclidean Distance	0.9188461538461539	0.907954019664546	0.8813642729917094	0.8886186075828932
7 Neighbor Manhattan Distance	0.9237087912087913	0.9352706552706553	0.8732168595189584	0.8990304102246618
9 Neighbor Euclidean Distance	0.923956043956044	0.9209218559218559	0.888030939658376	0.8985621559699899
9 Neighbor Manhattan Distance	0.9308516483516482	0.9417587505087506	0.8892569932982226	0.9105898101747709
12 Neighbor Euclidean Distance	0.9277747252747254	0.9358112475759534	0.8823899140173503	0.9039054739423404
12 Neighbor Manhattan Distance	0.9326373626373627	0.9463531424715634	0.8935010251284614	0.9152827838677446
15 Neighbor Euclidean Distance	0.925989010989011	0.9243368886015946	0.889056580684017	0.90203527424956
15 Neighbor Manhattan Distance	0.9308516483516485	0.9416163003663003	0.8935010251284614	0.9129750915600523
20 Neighbor Euclidean Distance	0.9254945054945056	0.9400457875457876	0.8789121657120159	0.9049175667816428
20 Neighbor Manhattan Distance	0.9237087912087911	0.9398650472334683	0.8731130652622408	0.9017135493256037
30 Neighbor Euclidean Distance	0.9170604395604396	0.9439407814407815	0.848535354117813	0.890365010356264
30 Neighbor Manhattan Distance	0.9237087912087911	0.9564743589743591	0.858524205845795	0.9019732178669105

Table 6: KNN Hyperparameter Tuning for original dataset

Neighbors and Distance Func.	Avg. Accuracy	Avg. Precision	Avg. Recall	Avg. F1
3 Neighbor Euclidean Distance	0.871620879120879	0.8210937516589691	0.8608964652289239	0.8277076404211969
3 Neighbor Manhattan Distance	0.871620879120879	0.8210937516589691	0.8608964652289239	0.8277076404211969
5 Neighbor Euclidean Distance	0.8943406593406594	0.8682537779296429	0.8594024333986854	0.8556611513377721
5 Neighbor Manhattan Distance	0.8943406593406594	0.8682537779296429	0.8594024333986854	0.8556611513377721
7 Neighbor Euclidean Distance	0.9027747252747252	0.8752840566178162	0.8706468112097797	0.8657533215088519
7 Neighbor Manhattan Distance	0.9027747252747252	0.8752840566178162	0.8706468112097797	0.8657533215088519
9 Neighbor Euclidean Distance	0.900989010989011	0.8823619650641457	0.8637502594856418	0.8662471817588095
9 Neighbor Manhattan Distance	0.900989010989011	0.8823619650641457	0.8637502594856418	0.8662471817588095
12 Neighbor Euclidean Distance	0.9078846153846154	0.9131326222796812	0.8621225188687708	0.8792334603975045
12 Neighbor Manhattan Distance	0.9078846153846154	0.9131326222796812	0.8621225188687708	0.8792334603975045
15 Neighbor Euclidean Distance	0.904313186813187	0.8976663075486604	0.8664703449557273	0.8739022418240132
15 Neighbor Manhattan Distance	0.904313186813187	0.8976663075486604	0.8664703449557273	0.8739022418240132
20 Neighbor Euclidean Distance	0.9060989010989011	0.9218525102348633	0.8443152686477274	0.8755172351550551
20 Neighbor Manhattan Distance	0.9060989010989011	0.9218525102348633	0.8443152686477274	0.8755172351550551
30 Neighbor Euclidean Distance	0.9007417582417583	0.9175878043525101	0.8370619241661222	0.8670078936193978
30 Neighbor Manhattan Distance	0.9007417582417583	0.9175878043525101	0.8370619241661222	0.8670078936193978

Table 7: KNN Hyperparameter Tuning for SVD low-rank dataset

Neighbors and Distance Func.	Avg. Accuracy	Avg. Precision	Avg. Recall	Avg. F1
3 Neighbor Euclidean Distance	0.8775274725274723	0.8706140105164873	0.8232623367803278	0.8353813057060492
3 Neighbor Manhattan Distance	0.866813186813187	0.8594389776889777	0.8007281711708248	0.8151473982747734
5 Neighbor Euclidean Distance	0.8647802197802198	0.8704878867262769	0.7940543286049284	0.8138968174682459
5 Neighbor Manhattan Distance	0.8701373626373627	0.8784141516378359	0.7962978734991479	0.8221534885751064
7 Neighbor Euclidean Distance	0.8670604395604394	0.8937833561758384	0.7683543580773715	0.8137884615384616
7 Neighbor Manhattan Distance	0.8637362637362637	0.8788649263533165	0.7777171638539706	0.8096075496624924
9 Neighbor Euclidean Distance	0.8726648351648351	0.9020024412927226	0.7705765802995937	0.8186207571514184
9 Neighbor Manhattan Distance	0.8690934065934066	0.8930144009757013	0.783187249324056	0.8197489361358613
12 Neighbor Euclidean Distance	0.8657692307692308	0.9050190985485104	0.7446377580440549	0.804447985289573
12 Neighbor Manhattan Distance	0.867554945054945	0.9117424242424242	0.7525838971539872	0.8098402352993558
15 Neighbor Euclidean Distance	0.8619505494505495	0.8905531151854682	0.7525304783505682	0.80157975303529
15 Neighbor Manhattan Distance	0.8690934065934066	0.896423705013522	0.7742688879919015	0.816810589172448
20 Neighbor Euclidean Distance	0.8637362637362637	0.9036435786435787	0.7481826522636117	0.8030655168739951
20 Neighbor Manhattan Distance	0.8673076923076923	0.9036844283167813	0.7580971821781418	0.8090254409052504
30 Neighbor Euclidean Distance	0.8476648351648353	0.8781508932244225	0.7253754020425685	0.7784346745239324
30 Neighbor Manhattan Distance	0.8530219780219779	0.88339243960374	0.7415159855969451	0.7872275038644291

Table 8: KNN Hyperparameter Tuning for SVD high-rank dataset

Neighbors and Distance Func.	Avg. Accuracy	Avg. Precision	Avg. Recall	Avg. F1
3 Neighbor Euclidean Distance	0.9242032967032967	0.9020319582378408	0.8951659971296404	0.8963608434976766
3 Neighbor Manhattan Distance	0.9242032967032967	0.9020319582378408	0.8951659971296404	0.8963608434976766
5 Neighbor Euclidean Distance	0.9257417582417583	0.9189880952380953	0.885473343456477	0.8982880857414018
5 Neighbor Manhattan Distance	0.9257417582417583	0.9189880952380953	0.885473343456477	0.8982880857414018
7 Neighbor Euclidean Distance	0.9239560439560439	0.9127761188087276	0.8848211695434335	0.8955583885504108
7 Neighbor Manhattan Distance	0.9239560439560439	0.9127761188087276	0.8848211695434335	0.8955583885504108
9 Neighbor Euclidean Distance	0.9290659340659341	0.9116076206402294	0.8986142729917093	0.9034552123232604
9 Neighbor Manhattan Distance	0.9290659340659341	0.9116076206402294	0.8986142729917093	0.9034552123232604
12 Neighbor Euclidean Distance	0.9270329670329671	0.9251516685069318	0.8839437749074183	0.9021774239499927
12 Neighbor Manhattan Distance	0.9270329670329671	0.9251516685069318	0.8839437749074183	0.9021774239499927
15 Neighbor Euclidean Distance	0.9206318681318681	0.8946391332452714	0.8939437749074182	0.8914322227774487
15 Neighbor Manhattan Distance	0.9206318681318681	0.8946391332452714	0.8939437749074182	0.8914322227774487
20 Neighbor Euclidean Distance	0.9188461538461539	0.8978341266576562	0.8868220024603083	0.8891574705496834
20 Neighbor Manhattan Distance	0.9188461538461539	0.8978341266576562	0.8868220024603083	0.8891574705496834
30 Neighbor Euclidean Distance	0.9254945054945056	0.9190147352647353	0.8850442246825306	0.899455342926976
30 Neighbor Manhattan Distance	0.9254945054945056	0.9190147352647353	0.8850442246825306	0.899455342926976

Table 9: KNN Hyperparameter Tuning for MLP based AutoEncoder 5% feature dataset

Neighbors and Distance Func.	Avg. Accuracy	Avg. Precision	Avg. Recall	Avg. F1
3 Neighbor Euclidean Distance	0.8854120879120879	0.8664146745067797	0.85391181973116	0.8482956592305294
3 Neighbor Manhattan Distance	0.8912637362637362	0.8851315789473684	0.85658580325222	0.8572412785627073
5 Neighbor Euclidean Distance	0.8961263736263735	0.8898945279866333	0.8622874235958944	0.8642355454143381
5 Neighbor Manhattan Distance	0.896868131868132	0.89186004784689	0.8650652013736722	0.8657118661719186
7 Neighbor Euclidean Distance	0.8930494505494506	0.8910192601067888	0.8554913216468689	0.859707264957265
7 Neighbor Manhattan Distance	0.8981593406593407	0.901256810504522	0.8517990139545611	0.8645472547166095
9 Neighbor Euclidean Distance	0.8984065934065935	0.8992045454545453	0.8592652680070222	0.8658168601267524
9 Neighbor Manhattan Distance	0.8984065934065935	0.8944642857142856	0.8590212361767835	0.8648537443840014
12 Neighbor Euclidean Distance	0.8956318681318681	0.9156499954294072	0.8294362081779625	0.8608879265942535
12 Neighbor Manhattan Distance	0.900989010989011	0.9164324192095427	0.8406584304001846	0.8677834736185799
15 Neighbor Euclidean Distance	0.8961263736263737	0.9080801658781533	0.840576791732339	0.862585294030044
15 Neighbor Manhattan Distance	0.9032692307692308	0.9138333498922071	0.853025067594408	0.8726194028583457
20 Neighbor Euclidean Distance	0.9048076923076923	0.9287344599844601	0.8434511878676046	0.8754717288994653
20 Neighbor Manhattan Distance	0.9086263736263737	0.924489893211289	0.8525549821243225	0.8789447643778997
30 Neighbor Euclidean Distance	0.9137362637362637	0.9247196342305039	0.8707356994579634	0.88970284362072
30 Neighbor Manhattan Distance	0.9137362637362638	0.9241701836810533	0.8664916676277246	0.8879887494360377

Table 10: KNN Hyperparameter Tuning for MLP based AutoEncoder 20% feature dataset

both cases and calculated distances should be similar for both of the functions.

We use this setup with k=12, the manhattan distance function, and the original dataset for comparison with other methods used in the study.

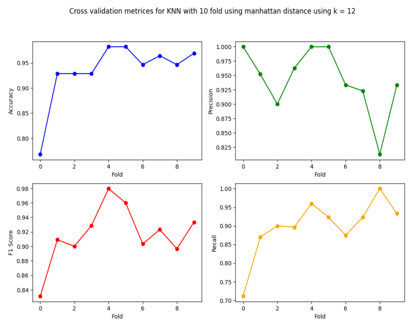

Figure 9: Performance of KNN for 10 fold cross validation with manhattan distance on the original dataset using k=12

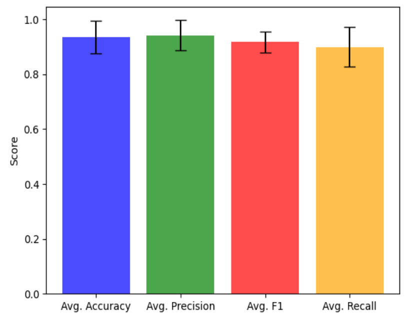

Figure 10: Average performance of KNN including error bars with manhattan distance on the original dataset using k=12

Evaluation of different metrics and standard errors using the mentioned hyperparameters are presented in figure 9 and figure 10.

5.3.3 KNN Correctness Results. KNN was trained on the provided artificial dataset with 17 points. The test data point used for implementation correctness measurement is $[1.4, 3]$. The predictions were performed using two distance functions - euclidean distance and manhattan distance. The hyperparameter k used was 3 as instructed.

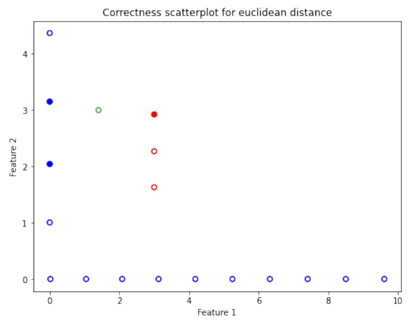

Figure 11: 2-d scatter plot for KNN using euclidean distance on artificial correctness dataset

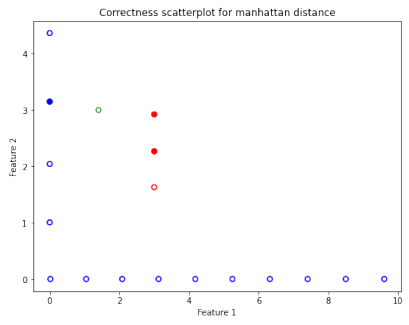

Figure 12: 2-d scatter plot for KNN using manhattan distance on artificial correctness dataset

In figure 11 and figure 12 the blue circles represent class 1 and the red circles represent class 2.

In figure 11, the nearest neighbors with filled circles for the test data points are two of class 1 and one of class 2 while using euclidean distance. Thus the predicted class is 1.

In figure 12, the nearest neighbors with filled circles for the test data points are two of class 2 and one of class 1 while using manhattan distance. Thus the predicted class is 2.

Additionally, we noticed that our implementation of KNN using the euclidean distance function gives out exact same average precision, recall, f1 score, and accuracy as the off-the-shelf implementation of KNN from *scikit-learn* for all the experimentations.

5.4 DBSCAN Clustering

5.4.1 Summary of Algorithm Implementation.

(1) We start off by choosing a point/object p and exploring its neighbourhood with respect to the *epslon* and min_pts parameters. We mark the object as visited and then begin to check all the objects and compute the euclidean distance between them.
(2) If the distance is less than *epslon* then that particular object is in the neighbourhood of p. Once we know all the objects in the neighbourhood of p, and if their count greater than or equal to the value of min_pts parameter, we create a new cluster and add the object list into it.
(3) Then we iterate through each object in the cluster and take its neighbourhood to see if any of those objects that are unvisited have satisfied the same criteria as above and if yes, add them to the cluster. If the radius and min_pts condition is not satisfied, then it is considered as noise.

5.4.2 Default Dataset on scikit.

(1) For the midterm report we wanted our silhouette coefficient to be as close as possible to 1 and our calinski scores as high as possible whose ideal value is considered as 100. But in order to maintain the quality of the cluster, we will select the ideal value where the silhouette coefficient is on the higher side and the calinski score has just begun to drop. We got $SilhouetteCoefficient = 0.365$ and $Calinski - Harabaszscore = 83.122$
(2) The dataset has 31 features out of which most of them are considered relevant. So I plotted scatterplots between Radius Mean and Texture Mean, Perimeter Mean vs Area Mean. We can with confidence say that the two clusters generated in the following two figures are consistent and relevant meaning that our analysis on them can produce better results by further tuning the hyperparameters

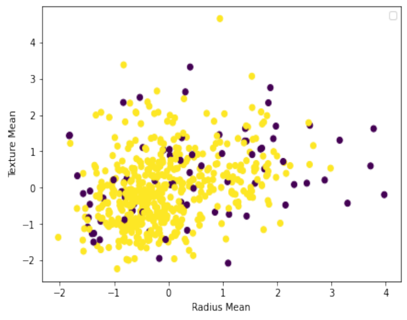

Figure 13: Scatterplots of Radius Mean vs Texture Mean

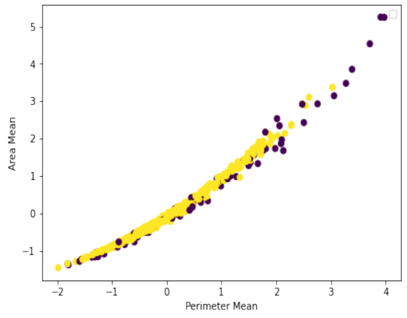

Figure 14: Scatterplots of Perimeter Mean vs Area Mean

5.4.3 *Artificial Dataset on scikit.*

(1) In order to compare the results with the from scratch implementation, I first ran the provided hyperparameter values on the scikit learn implementation of dbscan.Upon using the values of the hyperparameters $min_pts = 2$ and $e = 1.25$, there were 2 clusters that differentiated from each other
(2) The computed silhouette score was 0.482 and calinski score was 9.678. The silhouette score can be considered as acceptable but since we want higher calinski scores to say that the clusters are differentiated properly, I think that that the 2 clusters separated are not too far from each other (in terms of between the clusters separation)

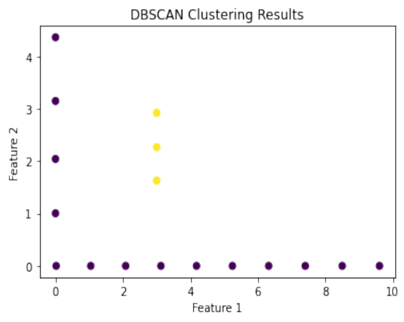

Figure 15: Scatterplot of Feature 1 vs Feature 2 WRT Labels

5.4.4 *Default Dataset on Scratch Implementation.*

(1) Upon comparing the assigning of labels, the off the shelf library performed better in forming clusters. Although there were more number of clusters formed in this approach which was 3 clusters compared to the 2 clusters of the off the shelf library, I think it is not much reliable because of the silhouette and calinski scores

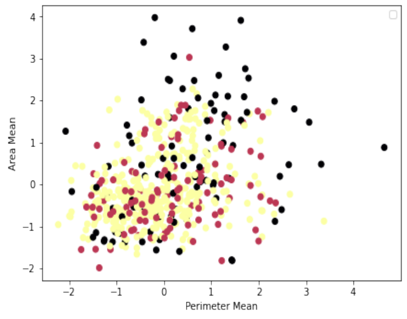

Figure 16: Scatterplot of Perimeter Mean vs Area Mean

5.4.5 *Artificial Dataset on Scratch Implementation.*

(1) The computed silhouette score was 0.16, meaning that the clusters are averagely well separated or the labels were not correctly assigned. But the calinski score is 1.93 which is less than that of the off the shelf library implementation. Calinski score is a measure how better the quality of the clusters is. So I think on the artificial data, it is somewhat low
(2) Generated a scatterplot of $Feature1$ vs $Feature2$. Comparing with the same scatterplot generated using scikit learn, assuming that scikit learn's implementation is more accurate because of the silhouette scores difference, we can say that the traversal of the neighbourhood expansion is not that accurate for the from the scratch implementation

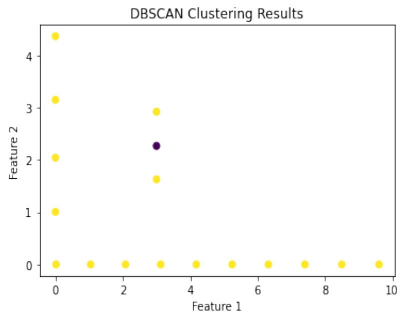

Figure 17: Scatterplot of Feature 1 vs Feature 2 WRT Labels

5.5 Spectral Clustering

In order to test the implementation correctness of Spectral Clustering, I took the dataset implementation_correctnes_dataset.csv and applied the following steps. 1. Constructed an unweighted undirected similarity graph. 2. Create from the similarity graph the adjacency matrix, degree matrix, Laplacian, and Normalized Laplacian. 3. Calculate the spectral embeddings of dimension 2 based on the Normalized Laplacian. 4. Run k-means for k = 2 on the spectral embeddings.

5.5.1 Spectral Embedding. Table 11 displays our spectral embeddings of dimension 2 based on the Normalized Laplacian. These embeddings provide the benefit of non-linear dimensionality reduction/augmentation that maintains the closeness between each datapoint. As a result, the embeddings corresponding to the points in the same cluster come closer together, while embeddings corresponding to points in different clusters tend to get further away from each other. This oftentimes creates more accurate results when applying clustering methods such as k-means to the spectral embeddings compared to performing clustering methods on the original data.

-0.258198890	-0.258198890
-0.122716465	-0.0415556789
-0.316227766	-0.402765823
0.389367867	0.158081088
3.16713902e-16	0.377964473
0.389367867	0.268648858
-0.316227766	0.284486447
-0.389367867	0.426729947
-0.316227766	0.118279375
-0.122716465	-0.309486753
-0.122716465	0.351042432
-0.365148372	0.182574186
0.0723036837	-0.0361518419
-0.0735609281	0.0367804640
0	0
0	0
0	0

Table 11: Spectral Embeddings of Dimension 2 based on the Normalized Laplacian

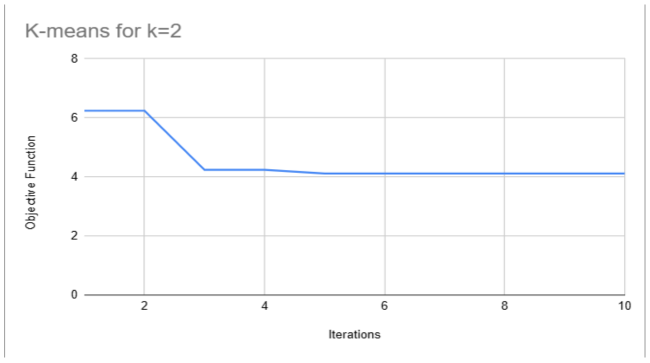

Figure 18: K-means iterations line graph

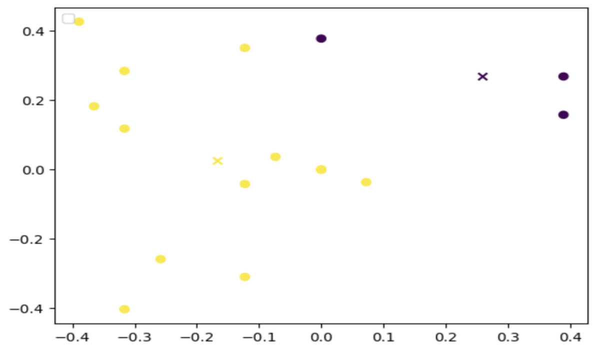

Figure 19: Scatterplot of Kmeans Clustering of Spectral Embeddings

5.5.2 K-means. I then ran k-means clustering on my spectral embeddings. Figure 18 shows a line graph where the horizontal axis corresponds to iterations of k-means and the vertical axis indicateds the objective function of k-means at that iteration. Each iteration denotes the updates of centroids and update of assignments as different iterations. As you can see in the figure, after 6 iterations my k-means stops changing the centroid locations as any further updates to the centroid location would result in either the same objective function or a higher objective function. Figure 19 displays a scatterplot showing the spectral embeddings (drawn as circles) and colored based on the cluster that k-means assigned them in, and the calculated centroids (drawn as x's) having the same color as the cluster they represent.

5.6 Agglomerative Clustering with Single Linkage

To test the implementation correctness of my code, I used the provided implementation_correctness_dataset.csv as well. My implementation will output the linkage matrix which is accepted by the library function dendrogram. To show the similarity, the submitted Jupyter notebook will produce the dendrogram using both my implementation of Agglomerative single linkage clustering as well as the of-the-shelf method.

5.6.1 Scatter Plot. The figure 20 shows the scatter plot of the input data.

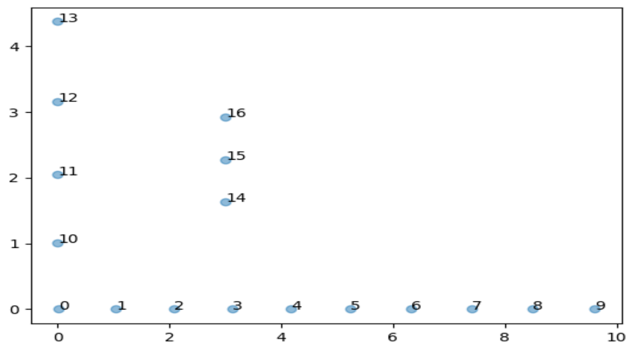

Figure 20: Scatterplot of the input dataset

5.6.2 Dendrogram. The figure 21 shows the dendrogram generated by my implementation of single linkage agglomerative clustering. The X-axis tick labels represent the unique integer assigned to each data-point from input dataset. Note this integer is same as show in scatterplot in figure 20

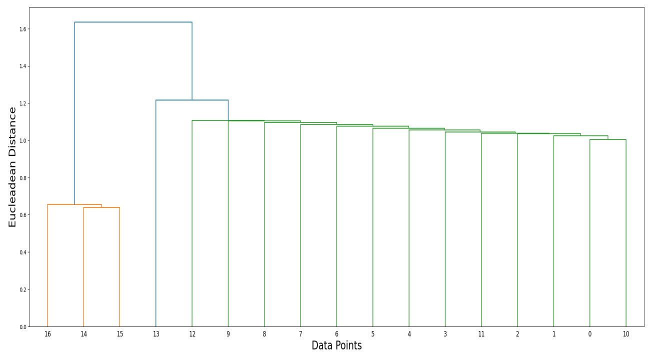

Figure 21: Dendrogram generated from my implementation of single linkage clustering

The figure 22 shows the dendrogram generated by the of-the-shelf library function from *scikit-learn*

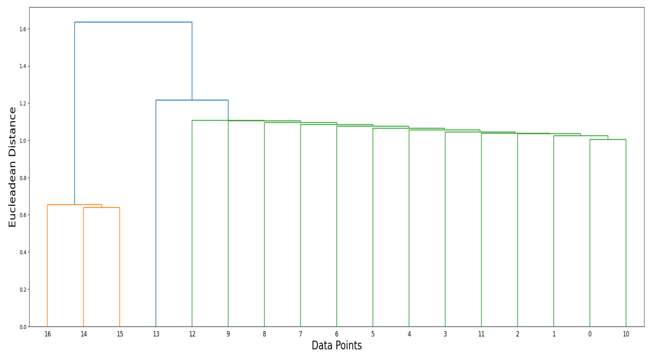

Figure 22: Dendrogram generated by scikit-learn linkage function

5.6.3 Determine the number of clusters. To find the clusters, MATHPLOTLIB library uses function 0.7∗(maximum distance between any of the cluster). This means, if a horizontal line is drawn at largest distance between any clusters (point on Y-axis), the number of vertical lines crossed by the horizontal line represents the optimal number of clusters. For the give dataset it is 3. Figure 23

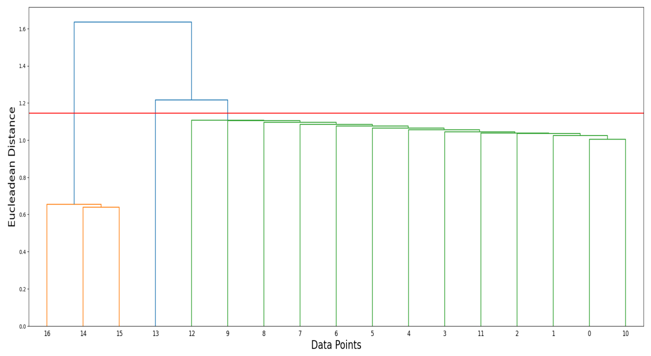

Figure 23: Dendrogram with horizontal line to split the data into optimal cluster.