

# CS235 Winter'23 Project Final Report: Default Project

Sanjana Senthilkumar  
NetID: ssent013

Kishan Sivakumar  
NetID: ksiva011

Akash Sundaresan Mahalaxmi  
NetID: asund016

Deepak Urs Gagenahalli  
Veeraraje Urs  
NetID: dgage002

Shadhrush Swaroop  
NetID: sswar010

Puneet Singhania  
NetID: psing088

## ABSTRACT

Our project's scope was as defined in the default route where we are given the Breast Cancer Wisconsin (Diagnostic) Kaggle competition data set. The data set classifies data points into two categories, Malignant and Benign, denoting the nature of the growth. We have used each of the six provided methods to rightly classify the data points. Before applying the models, we had to do a number of pre-processing steps. The pre-processing for each of the six methods was different but the clustering techniques and classification techniques had some common steps among themselves. On applying the models we could see that the classification techniques did a better job than the clustering one's because of the nature of the problem which suits classification.

## KEYWORDS

data, mining, malignant, benign, data points, pre-processing, clustering, classification

### ACM Reference Format:

Sanjana Senthilkumar, Kishan Sivakumar, Akash Sundaresan Mahalaxmi, Deepak Urs Gagenahalli Veeraraje Urs, Shadhrush Swaroop, and Puneet Singhania. 2018. CS235 Winter'23 Project Final Report: Default Project. In *Proceedings of Data Mining Techniques (CS235 W23)*. ACM, New York, NY, USA, 15 pages. <https://doi.org/XXXXXXX.XXXXXXX>

## 1 INTRODUCTION

The default project option dictates that we use six different approaches to try predicting the target column of the Breast Cancer Wisconsin (Diagnostic) Kaggle competition data set. The approaches are:

- (1) Agglomerative Clustering (Shadhrush Swaroop)
- (2) Spectral Clustering (Sanjana Senthilkumar)
- (3) DBScan Clustering
- (4) Multi-Layer Perceptron
- (5) Random Forest Classifier (Deepak Urs G V)
- (6) KNN Classifier (Puneet Singhania)

Talking about the data set, each data point has 32 columns indicating the attributes of a patient and their tumor. The target column or

the column which needs to be predicted is the 'Diagnosis' column. The diagnosis of each patient/data point is either malignant or benign which indicates the nature of the tumor. There about 569 data points in the given data set. The motivation of each of our team member is to use their respective method to try and predict the diagnosis for a data point. To do this each of us had to first do pre-processing and used the respective methods from an off-the shelf source. The evaluation for the same was done as well. For the final part of the project each of us implemented the methods from scratch on our own and evaluated the same in two ways. One using the real-world data set and the other using the artificial data set to check implementation correctness.

## 2 PROPOSED METHODS

### 2.1 Agglomerative Clustering

Agglomerative clustering is a hierarchical clustering method which uses a bottom up approach to clustering. The main idea behind the algorithm is that at the beginning each point is a cluster on its own. On each iteration, we add to a cluster one or more points closest to the cluster based on the specified linkage. For the scope of this project I have used single linkage as directed.

- (1) **fitAggScratch()**: Here at each iteration, one point nearest to the cluster is added to it. This process continues until we reach the number of specified clusters. For the sake of the given data set, considering the types of diagnosis, 2 clusters would be the right choice.
- (2) **distanceMatrix()**: We need to initially calculate the distance of each point to every other point and store this in a distance matrix. This matrix needs to be updated every iteration when a point is added to a cluster. This updated values would be used for the next iteration.
- (3) **distanceFuntions()**: Here at each iteration, one point nearest to the cluster is added to it. This process continues until we reach the number of specified clusters. For the sake of the given data set, considering the types of diagnosis, 2 clusters would be the right choice. The distance function used to find the nearest data point is also something that we can specify out of 3 options. The options are Euclidean, Manhattan and Cosine metrics. This is the hyper parameter which I am trying to optimize for this problem. The above steps were coded as part of my from the scratch implementation. Before this, I am doing a few pre-processing steps which are:

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from [permissions@acm.org](https://doi.org/XXXXXXX.XXXXXXX).

CS235 W23, W 2023 quarter, Riverside, CA

© 2018 Association for Computing Machinery.

ACM ISBN 978-1-4503-XXXX-X/18/06...\$15.00

<https://doi.org/XXXXXXX.XXXXXXX>



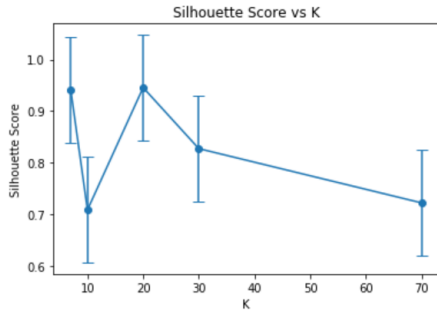


Figure 3: Silhouette Score for K-Nearest Neighbor Values

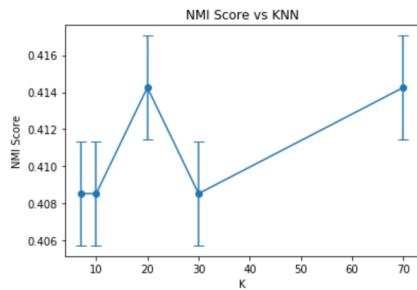


Figure 4: Normalised Mutual Index Score for K-Nearest Neighbor Values

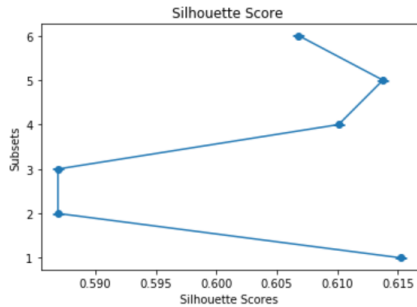


Figure 5: Silhouette Score for subsets of data

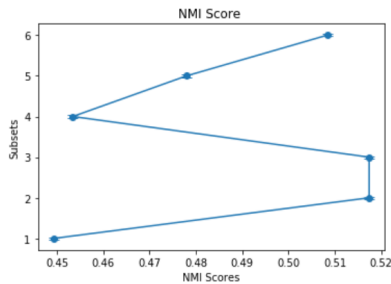


Figure 6: NMI Score for subsets of data

## 2.3 DBScan Clustering

DBSCAN is a clustering technique that uses the idea that clusters are dense regions separated by low-density regions. This algorithm groups data points that are closely located into one cluster without requiring the number of clusters to be predetermined, unlike K-means. Additionally, DBSCAN is resilient to outliers, making it ideal for analysing large spatial datasets. It only requires two input parameters: epsilon, which defines the radius around each data point to assess density, and min samples, which specifies the minimum number of data points needed to classify a point as a Core point.

The data is pre-processed using the following steps:

- (1) Removed null values from the dataset.
- (2) Dropped 'Unnamed: 32' and 'id' columns because they were irrelevant in determining the result.
- (3) Used LASSO method which performs both feature selection and regularization by shrinking the coefficients of less important features to zero.
- (4) Used t-SNE since it was particularly effective in dimensionality reduction of our dataset. This helps us visualize our model in 2-D space.

The implementation is divided into four parts:

**(1) Finding the best epsilon value:** The code finds the optimal value of the epsilon parameter for the DBSCAN algorithm. It does so by computing the distances between each point and its two nearest neighbors using the scikit-learn NearestNeighbors class. Then, it sorts the distances in ascending order and extracts the second column, which corresponds to the distances to the second nearest neighbor. It plots the sorted distances, and the optimal value of epsilon can be identified as the distance at which the rate of change in the plot is the highest, or the elbow of the plot. This optimal value represents the optimal radius to be used in the DBSCAN algorithm for clustering the dataset.

**(2) The Scratch Implementation of DBscan Algorithm:** The code implements the DBSCAN clustering algorithm from scratch, without using any built-in functions. It contains three functions:

- (1) **isInsideCircle():** This function calculates the Euclidean distance between two points and returns True if the distance is less than or equal to a given radius, indicating that the second point lies inside the circle centered at the first point with the given radius.
- (2) **corePoints():** This function identifies the core points and non-core points in the dataset. It loops through each data point and checks how many other points lie within a given radius of it. If the number of such points is greater than or equal to a specified minimum number of samples, then the point is a core point, and it is added to the corePointsList. Otherwise, the point is a non-core point, and it is added to the nonCorePointsList.
- (3) **dbscanfromscratch():** This function performs the actual DBSCAN clustering. It first calls the corePoints function to identify the core and non-core points. Then, it initializes an empty dictionary d, which will store the clusters. The

function then randomly selects a core point as the center of a new cluster, and adds it to `d` as a new cluster. It then loops through all the other core points that lie within the given radius of the center point and adds them to the same cluster. It repeats this process until no more core points can be added to the cluster. It then adds all the non-core points that lie within the given radius of the center point to the same cluster. Once all the points have been assigned to clusters, the function returns a list named `dbspread` indicating the cluster number (starting from 1) of each data point in the input dataset `X`.

**(3) Performing cross validation:** Performs cross-validation on the DBSCAN algorithm to find the best value for the min samples parameter. The cross-validation is performed by iterating over a list of min samples values and for each value, running the `dbscanfromscratch()` function 10 times on the dataset `X` with the given `eps` and min samples values. For each run, the code computes the silhouette score and the normalized mutual information score (NMI) between the predicted clusters and the true labels `y`. The average silhouette score and NMI score are computed over the 10 runs and stored in their lists respectively. The standard deviations of the silhouette and NMI scores are also computed and stored in the lists respectively. Finally, the code finds the min samples value that resulted in the highest NMI score and prints it as the best min sample value after cross validation. It also prints the average silhouette score and NMI score over all the min samples values tested.

```
Best min sample value after cross-validation is: 2
Average Silhouette score 0.5069422
Average NMI score 0.510294135723836
```

Figure 7: Results after cross-validation

**(4) Cluster formation using Scatter Plots:** Creates a figure with two subplots (`ax1` and `ax2`) to compare the clustering results of the DBSCAN algorithm implementation (in `ax1`) with the actual clusters (in `ax2`). For both subplots, scatter plots are created with `X[:,0]` as the x-axis values and `X[:,1]` as the y-axis values. The color of the scatter points is determined by the cluster label, either from the DBSCAN algorithm or the actual labels (`y`). The `cmap` parameter is used to specify the color map to use, and `edgecolor` and `alpha` parameters are used to adjust the appearance of the scatter points. Finally, each subplot is given a title, "My DBScan clustering plot" for `ax1` and "Actual clusters" for `ax2`.

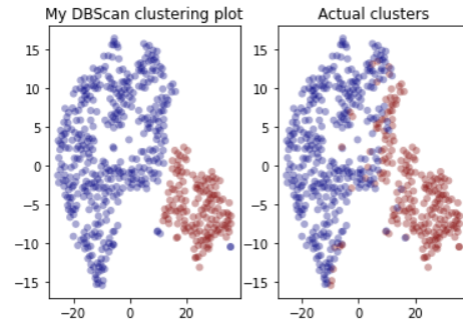


Figure 8: Cluster formation using Scatter Plots

Error bars are created for the silhouette score and NMI score against the min sample parameter of the DBSCAN algorithm. The error bars represent the standard deviation of the scores, and the dots represent the mean values.

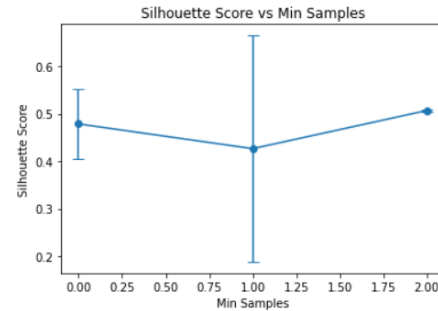


Figure 9: Silhouette Score vs Min Samples

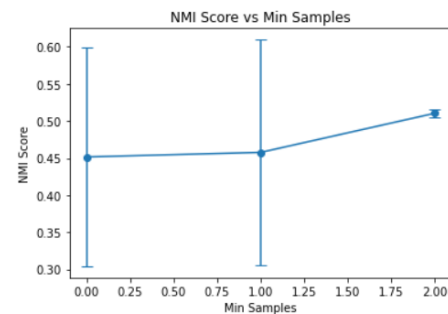


Figure 10: Normalised Mutual Index Score vs Min Samples

## 2.4 Multi-Layer Perceptron

A Multi-Layer Perceptron (MLP) is an artificial neural network that processes input data in a feedforward manner, consisting of multiple layers of perceptrons or neurons. During training, the weights are adjusted using backpropagation to minimize the error between the network's predictions and the actual output. MLPs

are useful for classification, regression, and pattern recognition tasks and have various applications. However, training MLPs can be computationally expensive, and overfitting is a common problem. Regularization techniques like dropout and L2 regularization can help mitigate overfitting.

The implementation is divided into five parts:

**(1) MLP: MLP using the Standard Version from sklearn outputs** Uses the StandardScaler to scale the training and testing data. Then, the default MLPClassifier from sklearn is used to fit the training data, predict the output for the testing data, and output the accuracy, ROC AUC score, and classification report. The training set score and testing set score are also printed. This code is used to establish a baseline for comparison with other MLP models that may be created.

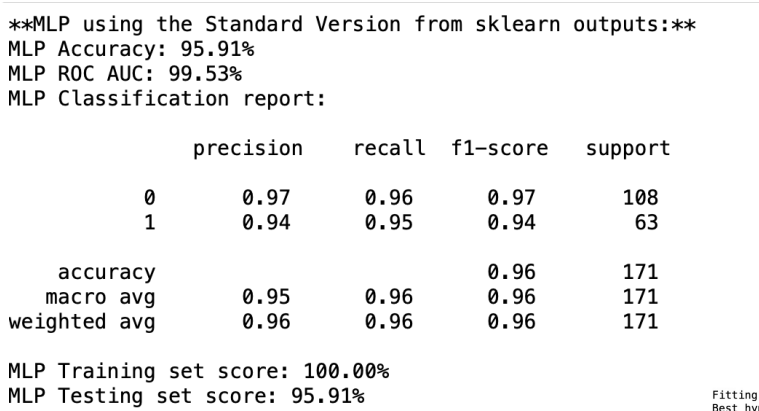


Figure 11: MLP using the Standard Version from sklearn outputs

**(2) MLP Grid: MLP using GridSearchCV parameters outputs** The code fits an MLP classifier with grid search CV parameters to the training data after scaling the features using the StandardScaler. The predictions and prediction probabilities on the test set are computed and output along with the accuracy, ROC AUC, classification report, and training and testing set scores. The purpose is to compare these results with the results obtained from the default MLP classifier from sklearn, used as a baseline in a previous code snippet.

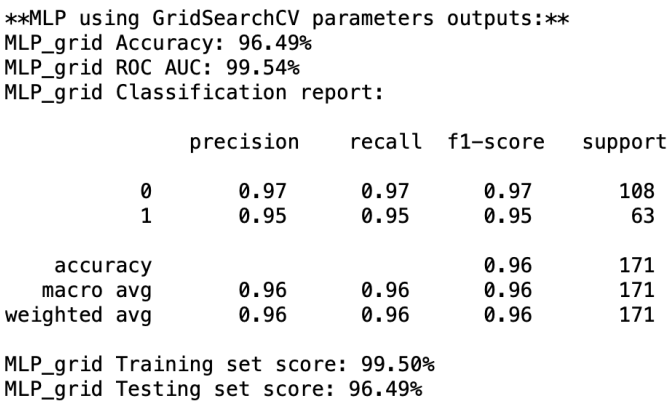


Figure 12: MLP using GridSearchCV parameters outputs

**(1) Grid Search CV:** The code creates a pipeline to preprocess data using different scalers and classify using an MLP classifier. It then sets up a parameter grid for hyperparameter tuning using GridSearchCV, with a range of preprocessors, activation functions, solvers, regularization parameters, learning rate, number of iterations, and random state. The GridSearchCV is performed with stratified k-fold cross-validation, using F1 score as the evaluation metric. Finally, the best hyperparameters and F1 score are calculated.

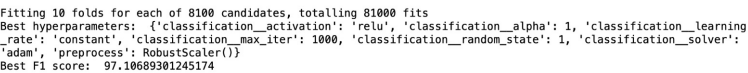


Figure 13: Grid Search CV

**(3)MLP BO: MLP using BayesSearchCV parameters outputs** Uses the MLPClassifier model from scikit learn library to fit and predict the target variable on the test data using the parameters obtained from Bayesian optimization search. It first creates an MLPClassifier object with the specified parameters and fits the model on the scaled training data. Then, it predicts the target variable using the predict and predict\_proba methods on the scaled test data. Finally, it computes various performance metrics like accuracy, ROC AUC, and classification report using scikit learn’s metrics module, and compares the results with the baseline model output.

**\*\*MLP using BayesSearchCV parameters outputs:\*\***

MLP\_BO Accuracy: 96.49%

MLP\_BO ROC AUC: 99.56%

MLP\_BO Classification report:

	precision	recall	f1-score	support
0	0.97	0.97	0.97	108
1	0.95	0.95	0.95	63
accuracy			0.96	171
macro avg	0.96	0.96	0.96	171
weighted avg	0.96	0.96	0.96	171

MLP\_BO Training set score: 98.99%

MLP\_BO Testing set score: 96.49%

**Figure 14: MLP using BayesSearchCV parameters outputs**

- (1) **Bayes Search CV:** the code performs hyperparameter tuning using Bayesian optimization with cross-validation. It uses a pipeline with a preprocessor and an MLPClassifier as the classifier. The search space for the hyperparameters is defined using a dictionary. A stratified k-fold cross-validation object is created with 10 splits. A BayesSearchCV object is created with the pipeline, search space, cross-validation object, and F1 score as input parameters. The fit() method is called with the input data and target labels. Finally, the best hyperparameters and the best F1 score are calculated.

```
Best hyperparameters: OrderedDict([('classification_activation', 'logistic'), ('classification_alpha', 0.0001), ('classification_learning_rate', 'constant'), ('classification_max_iter', 2000), ('classification_random_state', 2), ('classification_solver', 'adam'), ('preprocess', StandardScaler())])
Best F1 score: 97.11610800493388
```

**Figure 15: MLP using BayesSearchCV parameters outputs**

- (4) **MLP Funnel:** MLP Funnel using GridSearchCV parameters outputs In this implementation, we use scikit learn's MLPClassifier class to define the MLP Funnel model with a funnel shape architecture. We specify the hidden layer sizes(128, 64, 32)parameter to be a tuple of decreasing size, which will create the funnel shape. We also specify the activation function to be ReLU and the solver to be 'sgd'.

**\*\*MLP\_Funnel using GridSearchCV parameters outputs:\*\***

MLP\_Funnel Accuracy: 97.08%

MLP\_Funnel ROC AUC: 99.40%

MLP\_Funnel Classification report:

	precision	recall	f1-score	support
0	0.97	0.98	0.98	108
1	0.97	0.95	0.96	63
accuracy			0.97	171
macro avg	0.97	0.97	0.97	171
weighted avg	0.97	0.97	0.97	171

MLP\_Funnel Training set score: 99.50%

MLP\_Funnel Testing set score: 96.49%

**Figure 16: MLP Funnel using GridSearchCV parameters outputs**

- (5) **MLP SVD** In this implementation, we use scikit learn's TruncatedSVD class to perform low rank SVD on the input data, reducing the dimensionality of the data. And then splitting the preprocessed data into training and test sets and obtained the results using the pre trained MLP grid and MLP BO with the preprocessed input data.

**\*\*MLP\_grid using SVD outputs:\*\***

MLP\_grid\_svd Accuracy: 96.49%

MLP\_grid\_svd ROC AUC: 99.57%

MLP\_grid\_svd Classification report:

	precision	recall	f1-score	support
0	0.97	0.97	0.97	108
1	0.95	0.95	0.95	63
accuracy			0.96	171
macro avg	0.96	0.96	0.96	171
weighted avg	0.96	0.96	0.96	171

MLP\_grid\_svd Training set score: 98.99%

MLP\_grid\_svd Testing set score: 96.49%

**\*\*MLP\_BO using SVD outputs:\*\***

MLP\_BO\_svd Accuracy: 96.49%

MLP\_BO\_svd ROC AUC: 99.56%

MLP\_BO\_svd Classification report:

	precision	recall	f1-score	support
0	0.97	0.97	0.97	108
1	0.95	0.95	0.95	63
accuracy			0.96	171
macro avg	0.96	0.96	0.96	171
weighted avg	0.96	0.96	0.96	171

MLP\_BO\_svd Training set score: 98.99%

MLP\_BO\_svd Testing set score: 96.49%

**Figure 17: MLP SVD**

## 2.5 Random Forest Classifier

Often employed in classification and regression applications, Random Forest is a well-liked machine learning technique. During training, numerous decision trees are built, and the mean prediction for regression or the mode of the classes for classification are output from each tree. The main principle of Random Forest is to decrease overfitting and enhance generalization performance by adding randomization to the decision tree model. During each decision tree split, data and feature samples are taken at random to do this. The algorithm also favors variety among the trees by only taking into account a portion of attributes for each tree. Because of its reliability, high accuracy, and capacity for handling big datasets with high-dimensional features, the Random Forest approach is widely employed. The proposed method of implementing the Random Forest Classification method is as follows -

### (1) Basic Node definition:

- (a) **Data:** We know that a node is the basic unit of a decision tree. Decision trees are the basic units of Random Forests. Thus, a Node class is defined with the necessary data variables in it such as the information of the leaf node values, left and right sub trees, thresholds and the features in it. We have a method which helps in checking whether a node is leaf-node or not. It is very handy while performing a tree traversal later in the model

### (2) Decision Tree Class definition:

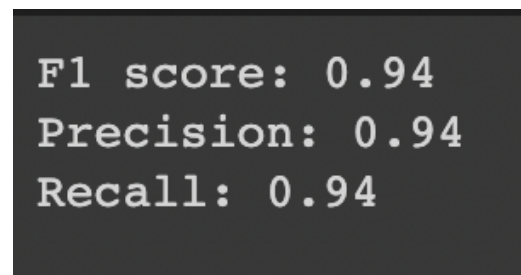
- (a) **fitting():** This method helps in training the decision tree model. Initially, we determine the number of features for understanding the data. Next, we use the next function mainly that helps in building the tree.
  - (i) **build decision tree():** This is one of the most important and comprehensive methods of the model which has many helper functions associated with it. It is a recursive function as well. Thus, we define the stopping criteria in the beginning of the function with the check being done on values like the 'number of labels' or 'number of samples' or based on the 'depth'. Next, we randomly choose a feature index and calculate the best feature and the best threshold value accordingly. This helps us in determining the right and left subtrees of the decision tree. The notable helper methods are defined below
  - (ii) **calculate split best():** The function helps in the determining the best possible split at a given node. The information gain value is a major metric in determining the split.
  - (iii) **info gain():** The function determines the information based on calculating the entropy of the children nodes. The compute-entropy() does the mathematical calculation to find the entropy of a data node and then the parent entropy will also be utilized. The difference of the parent and child entropy helps in determining the information gain
- (b) **prediction classification():** Here, the function takes the test input and helps in building the decision tree and provide a classification on the same.

### (3) Random Forest Class definition:

- (a) **fitting():** We create a number of decision trees and have the classifications obtained from them. But, we take a random samples to take an aggregate by bootstrap-sampling concept.
- (b) **prediction calculation():** The average of the all the tree prediction results are stored to make the final informed classification by the random forest. We get a final classification of whether the result belongs to the class-1 or class-2 of the proposed data

Based on the above definitions, we could see the following results

- (1) **Data Preparation :** Upon the import of the UCI Breast cancer data, we have the test and training data split being done with a 75percent and 25percent of data being done respectively.
- (2) **Random Forest model fit:** We start experimenting the Random forest model with '20' trees initially. Next, we have considered scenarios of other number of trees such as 5,40,50 as well. Next, the classification of the data is done to get the predictions.
- (3) **Correctness:** Here, based on the test values and the predictions that we received, we calculated the f1 score, precision and recall scores with a 'average weight' concept.

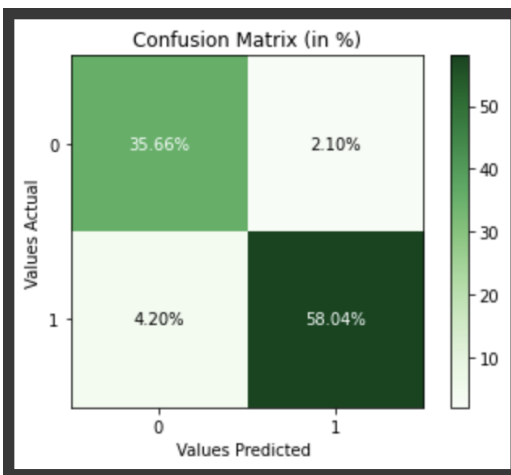


```
F1 score: 0.94
Precision: 0.94
Recall: 0.94
```

Figure 18: Random Forest correctness scores seen

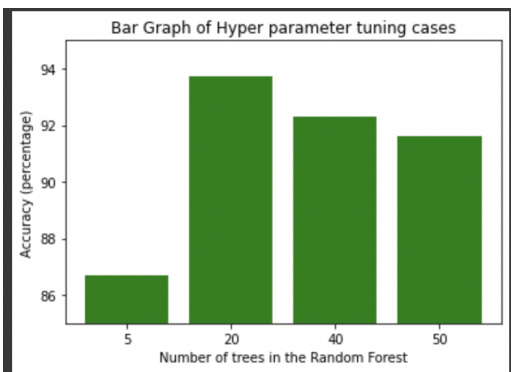
Next, we compute the confusion matrix with help of matplotlib and then show the 2\*2 matrix of the values predicted versus the actual values.





### Figure 19: Confusion Matrix of Random Forest

Finally, we have the comparison of the hyperparameters tuning by comparing the model performances with different number of trees being fed into it. A bar graph of the accuracies seen is plotted after marking it on an instance. We should note that there is some variation in model based on the bootstrap sampling received



**Figure 20: Bar graph of accuracies with respect different number of decision trees**

## 2.6 KNN Classifier

K-nearest neighbors (KNN) is a supervised machine learning algorithm utilized for different purposes like classification. It helps in predicting the class on the basis of the majority or average of the K neighbors. The main concept that enables this is by finding the K data points in the training set that are closest to a new data point in the test set. K denotes the number of neighbors. This hyperparameter is tuned for getting optimum performance. The distance metric is very essential here for determining how similar the data points are. Euclidean distance is the most commonly utilized metric. For our project, we have taken both Manhattan and Euclidean distances into consideration and obtained the performance metrics like F1, Precision, and Recall based on them.

In this project, we are performing multiple steps in the KNN approach to analyze and finally classify the data. We performed KNN classification on a dataset of breast cancer diagnosis utilizing the scikit-learn toolkit and then did a custom implementation from scratch. We determined the best value of K (number of neighbors) and evaluated the performance of the model. Multiple steps were performed like reading the data from the provided dataset, pre-processing the data, splitting the data into training and testing sets, utilizing 10-fold cross-validation for obtaining the performance metrics, plotting the outcomes separately using matplotlib library with the error bars representing the standard deviation of the metrics over the 10 cross-validation folds, reducing the dimensionality of the training data by applying SVD (Singular Value Decomposition), obtaining the optimal values of k using by applying k-NN on the low-rank and high-rank approximations of the training data using TruncatedSVD from scikit-learn, executing a classification task by utilizing two autoencoders and K-Nearest Neighbors (KNN) classifiers. We utilized the `tf.keras.Model` class. This comprises an encoder and a decoder which is defined to include two Autoencoder subclasses. This maps the input data to a latent space with lower dimensions and subsequently reconstructs the input data. Additionally, the fit approach is used to train the autoencoders, and a specific optimizer (Adam optimizer) and loss function (MeanSquaredError) are used. Following the training process, the autoencoders forecast the training data's encoded representation. Afterward, the encoded data generated by the Autoencoders is classified using KNN classifiers. We did a thorough hyperparameter tuning using cross-validation.

The data is pre-processed using the following steps:

- (1) Dropping the rows with missing values.
- (2) Utilizing the `corr()` method to determine the correlation between the columns in the remaining data
- (3) The predictor variables are chosen from a subset of the columns.
- (4) The 'diagnosis' column, which holds each patient's diagnosis as either "B" for benign or "M" for malignant, is the target variable. Modifying the variables "B" and "M" to 0 and 1, respectively.

The major methods that are developed for scratch implementation are basically the methods for fitting, predicting, and finding the nearest neighbors of a test point.

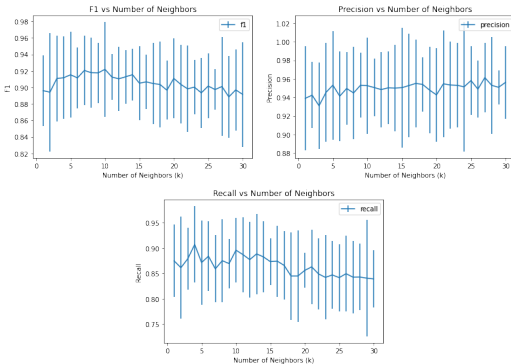
A feature matrix `X` and a target vector `y` are basically the inputs required by the `fit` method. `Self.X train` and `Self.y train` are the instance variables used by this method to store them.

The **predict** method requires an input feature matrix X. As a part of this, utilizing the selected distance metric(Manhattan or Euclidean), the distances between each sample in X and each sample in self are determined. The k closest neighbors and the class labels that correspond to them are picked in this. In the end, the most common class label among the k nearest neighbors is the predicted class label for each sample in X.



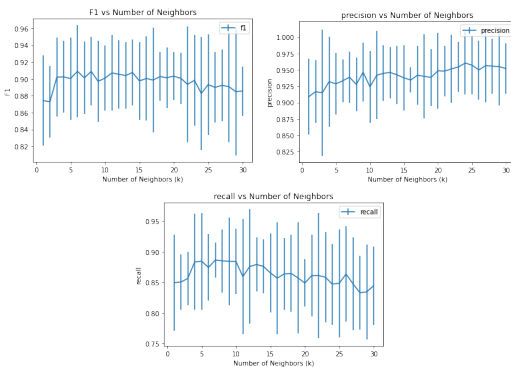
The method `getNearestNeighbors` takes test point X as an input. This assists in finding the test point's k closest neighbors. This method determines the distance between the test point and the remaining data points, and then picks the k shortest distances, and finally returns the matching data points.

The performance metrics obtained for all the given requirements are as follows.



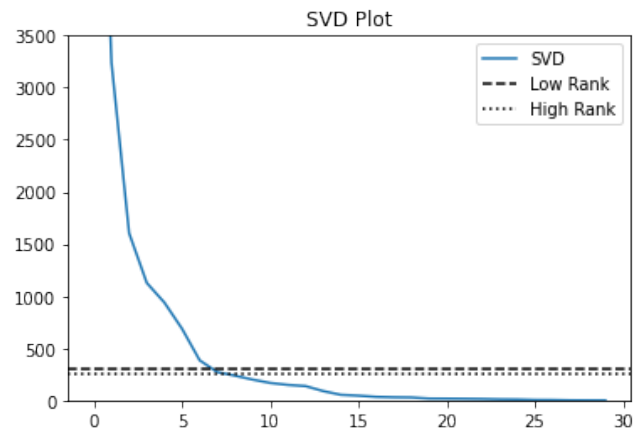
**Figure 21: Manhattan Distance based F1, Precision, Recall**

As shown in Fig: 21, the F1 score is approx 0.92, the precision score is 0.96 and the recall score is 0.90 if Manhattan distance is used. This is for scratch implementation. As far as the off-the-shelf sklearn implementation is concerned, the F1 score is approx 0.91, the precision score is 0.97 and the recall score is 0.88.



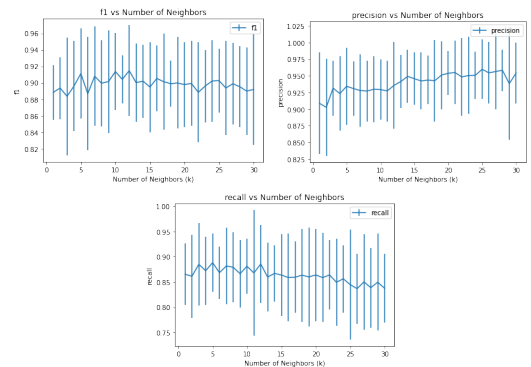
**Figure 22: Euclidean Distance based F1, Precision, Recall**

As shown in Fig: 22, the F1 score is approx 0.90, the precision score is 0.96 and the recall score is 0.88 if Euclidean distance is used. This is for scratch implementation. As far as the off-the-shelf sklearn implementation is concerned, the F1 score is approx 0.91, the precision score is 0.96 and the recall score is 0.88. Based on this, the value of n-components in SVD low-value and high-value rank approximation is considered. The optimal value is approx 6.



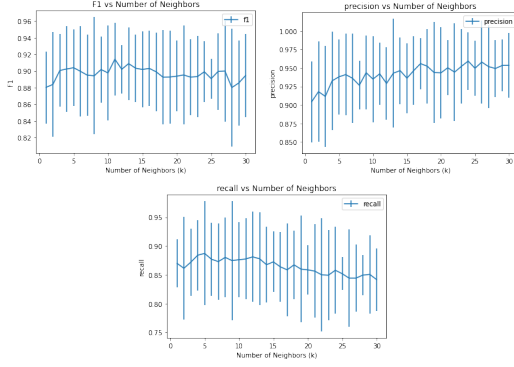
**Figure 23: SVD Plot**

As shown in Fig: 23, we have provided an SVD plot for scratch KNN implementation. Singular values produced from the SVD are being plotted in this graph. X-axis comprises the index of the singular values and Y-axis comprises singular values.



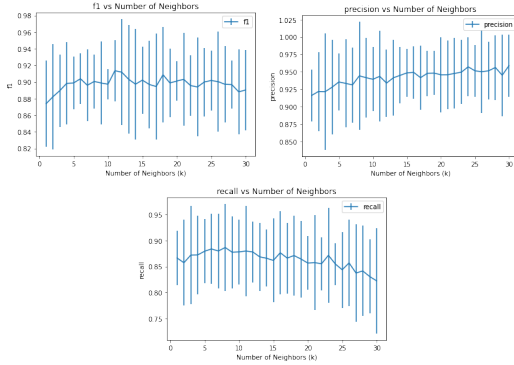
**Figure 24: SVD Low Approximation based F1, Precision, Recall**

As shown in Fig: 24, the F1 score is approx 0.91, the precision score is 0.95 and the recall score is 0.88 if SVD Low Approximation is used. This is for scratch implementation. As far as the off-the-shelf sklearn implementation is concerned, the F1 score is approx 0.89, the precision score is 0.95 and the recall score is 0.87.



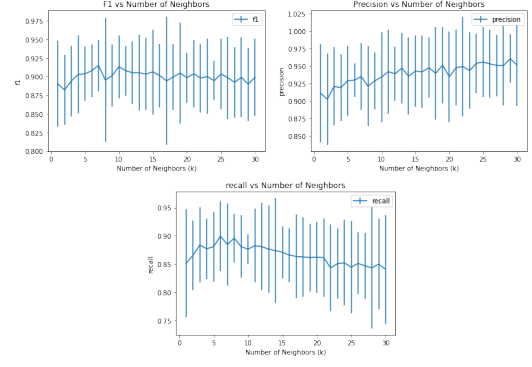
**Figure 25: SVD High Approximation based F1, Precision, Recall**

As shown in Fig: 25, the F1 score is approx 0.91, the precision score is 0.95 and the recall score is 0.88 if SVD High Approximation is used. This is for scratch implementation. As far as the off-the-shelf sklearn implementation is concerned, the F1 score is approx 0.90, the precision score is 0.95 and the recall score is 0.88.



**Figure 26: Auto Encoder Bottleneck 5% based F1, Precision, Recall**

As shown in Fig: 26, the F1 score is approx 0.91, the precision score is 0.95 and the recall score is 0.88 if Auto Encoder Bottleneck 5% is used. This is for scratch implementation. As far as the off-the-shelf sklearn implementation is concerned, the F1 score is approx 0.89, the precision score is 0.91 and the recall score is 0.89.



**Figure 27: Auto Encoder Bottleneck 20% F1, Precision, Recall**

As shown in Fig: 27, the F1 score is approx 0.91, the precision score is 0.96 and the recall score is 0.89 if Auto Encoder Bottleneck 20% is used. This is for scratch implementation. As far as the off-the-shelf sklearn implementation is concerned, the F1 score is approx 0.94, the precision score is 0.98 and the recall score is 0.93.

F1 score	Precision	Recall
0.9218980545704684	0.9613404228621618	0.9070448586501219
0.9090611311245738	0.9603491436100132	0.8864892392571757
0.9147009269209099	0.9599008898911888	0.8881496792681004
0.9142097882274489	0.9593178007698467	0.8871630446585075
0.9135869096198075	0.9589150319127437	0.886536986031871
0.915053279956757	0.9602475391376993	0.8993752689805321

We have this table displaying F1, Precision, and Recall scores for Manhattan, Euclidean, SVD Low Approximation, SVD High Approximation, Auto Encoder Bottleneck 5%, and Auto Encoder Bottleneck 20% based approaches respectively.

### 3 EXPERIMENTAL EVALUATION

#### 3.1 Clustering

(1) Agglomerative Clustering:

Silhouette Score	NMI
0.61398584	0.39292912110588435

(2) Spectral Clustering:

Silhouette Score	NMI
0.67633227	0.40853876069480044

(3) DBScan Clustering:

Silhouette Score	NMI
0.5069422	0.510294135723836034

(4) Overall:

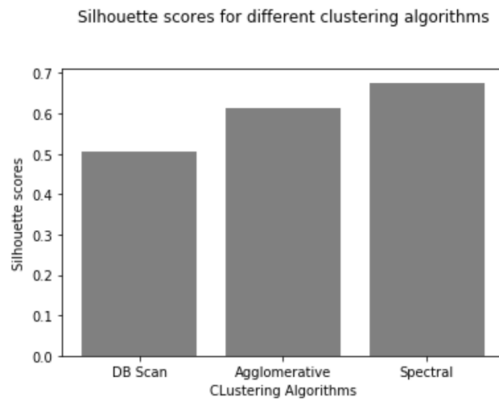


Figure 28: Overall Silhouette Score comparison

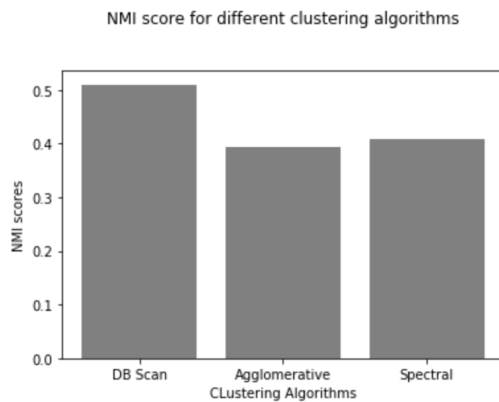


Figure 29: Overall NMI Score comparison

## 3.2 Classification

## 3.3 Overall

We can see that Classification methods outperform clustering methods for this data set.

## 4 DISCUSSION & CONCLUSIONS

We worked on the default Breast Cancer Wisconsin (Diagnostic) Kaggle competition data set. The 6 team members used different data mining techniques falling under clustering and classification categories. We could see that the classification models did better than the clustering models. The performance of all the clustering models were close to each other with silhouette scores between 0.5 - 0.6. The F1 scores of the classification models are between 0.88 to 0.98.

## 5 IMPLEMENTATION CORRECTNESS REPORT

### 5.1 Agglomerative Clustering

- (1) Scatterplot of the data set with a unique integer number on top of each point:

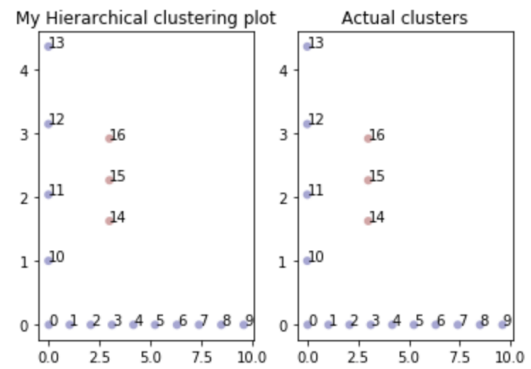


Figure 30: Scatter plot showing points with numbers.

- (2) The corresponding Dendrogram where on each leaf the same integer number as the corresponding point of the scatterplot:

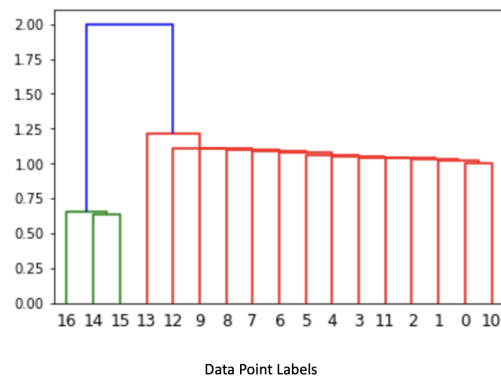


Figure 31: Dendrogram showing order which the clusters were formed.

Silhouette Score	NMI
-0.029728807189886405	1

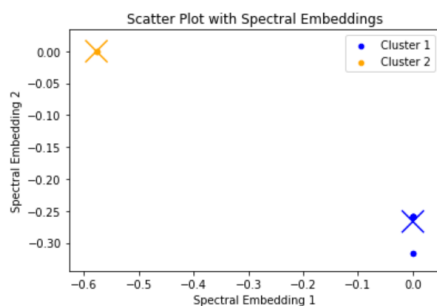
- (3) The artificial data set contains two clusters as per its design. I applied my from scratch agglomerative method on the data. Following the directives as given, I received two clusters as expected which correctly parted the data points as expected and visually obvious. The clusters were formed using the euclidean distance metric and adding the the two closest clusters into one. Continuing to do the same until the specified 2 clusters are left. The order in which the clusters are formed is clearly explained in the dendrogram. The height of each line represents the order and the neighbour represents the point added to the cluster. From the graph it is clear that the points 14, 15 and 16 are apart from the rest and these three are the closest. Hence they are added together first. Then the points 0 and 10 are the next closest. Building on top of that 1, 2, 11, 3, 4, 5, 6, 7, 8, 9, 12, 13 are added to the second cluster. As the points are added to a cluster, the distance of every other point from the newly formed cluster is recalculated and using the new distances, the next point is selected.

## 5.2 Spectral Clustering

- (1) Spectral Embeddings The use of Spectral Embeddings for clustering does have benefit over clustering the original points directly. Spectral embeddings represent the data in a lower dimensions and can reveal hidden patterns in the data. They also help give more importance to relevant features over irrelevant ones.
- (2) Spectral Embeddings of dimension 2:

Spectral Embedding 1	Spectral Embedding 2
0.	-0.25819889
0.	-0.25819889
0.	-0.25819889
0.	-0.25819889
0.	-0.25819889
0.	-0.25819889
0.	-0.25819889
0.	-0.31622777
0.	-0.25819889
0.	-0.25819889
0.	-0.25819889
0.	-0.31622777
0.	-0.25819889
0.	-0.25819889
-0.57735027	0.
-0.57735027	0.
-0.57735027	0.

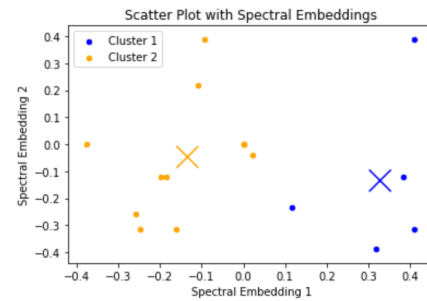
- (3) Scatter plot that shows the spectral embeddings (drawn as circles) the calculated centroids (drawn as x's):



**Figure 32: Scatter plot of Clustered Spectral Embeddings and their Centroids**

Silhouette Score	NMI
0.9804629238540817	1

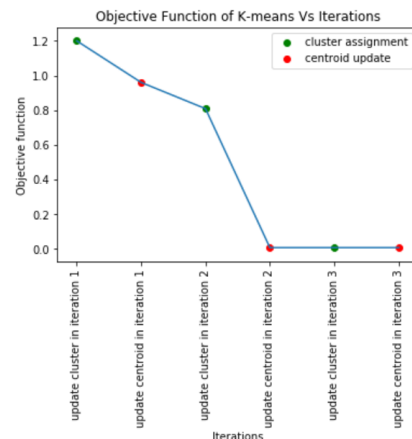
- (4) As the scatter plot reveals that most of the points are zero or the same and as a result the representation in the scatter plot has few points, I assumed the eigen vectors obtained in the code of the wrong orientation. So I took a transpose of these values and generated the scatter plot below. This saw a reduction in Silhouette score and NMI score.
- (5) Scatter plot that shows the spectral embeddings (drawn as circles) the calculated centroids (drawn as x's):



**Figure 33: Scatter plot of Clustered Spectral Embeddings and their Centroids**

Silhouette Score	NMI
0.4146230742616825	0.12237803483120178

- (6) The objective function is calculated as the sum of squares between the centroid and the data points. The cluster and centroid changes happen in the same iteration but are represented as alternating ones in the plot. Iteration  $i$  corresponds to objective function during cluster assignment and  $i+1$  corresponds to centroid update.
- (7) Iteration Vs Objective function for Cluster and Centroid updates:



**Figure 34: Plot that denotes the updates of the centroids and the updates of the assignments as different iterations.**

### 5.3 DBScan Clustering

- (1) Scatterplot of the dataset with an integer on top of each point showing the order in which it was visited by the algorithm.

**Explanation on why this order is correct:** The implemented code provides DBSCAN clustering algorithm and generates a scatter plot of the dataset with integers representing the order in which points were visited by the algorithm.

The order of points visited by the algorithm depends on the order in which the input points are provided and the distance metric used to calculate the distance between points. It also provides insights into how the algorithm works and how it groups points based on the distance metric and minimum number of points required to form a cluster. Therefore, the order of point visits follows the DBSCAN algorithm's rules for expanding clusters and is correct.

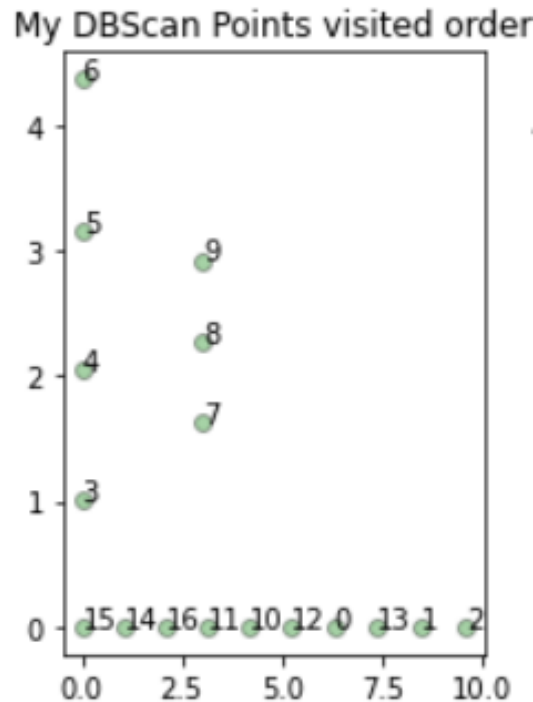


Figure 35: Scatter plot showing the order visited by the algorithm with integers.

- (2) Denoting the filled circles for the core points and with empty circles for the non core points.

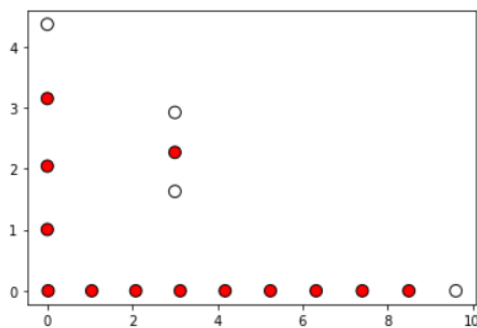


Figure 36: Circles denoting the core and non core points.

- (3) Using different colors for points belonging to different clusters as obtained by DBSCAN.

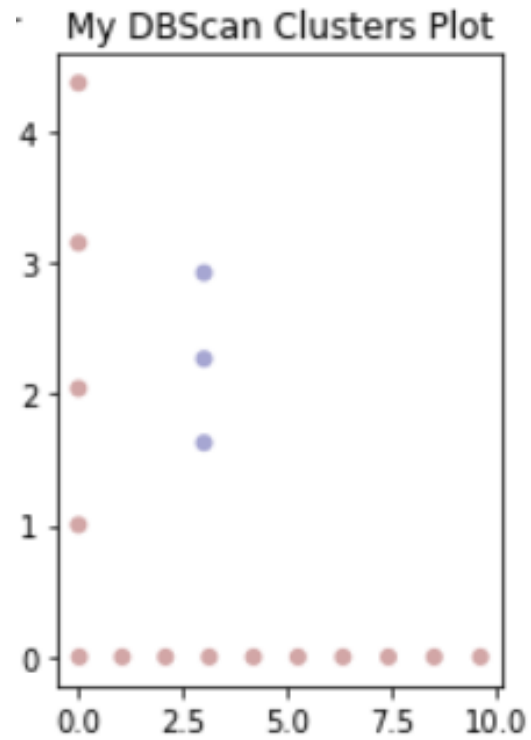


Figure 37: Colors denoting the different clusters as obtained by DBSCAN.

- (4) Results of the Implementation Correctness Report

Original Value: [1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2]  
 Predicted Values: [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2]  
 Silhouette Score: -0.029728807189886384  
 NMI Score: 1.0

Figure 38: Results of the Implementation Correctness Report

Silhouette Score	NMI
-0.029728807189886384	1

## 5.4 Multi-Layer Perceptron

## 5.5 Random Forest Classifier

A Random Forest was done from the base as shown in the proposed section. In order to establish the implementation correctness, we had to follow the next steps and the results were as follows - First, we imported the artificial dataset given by the evaluation team. A copy of the test data set was made as well. This helped in easier parsing of the data. The data was prepared by dropping the headers and taking the feature1 and feature2 and the similarly the 'class' column. The representative 'X' and 'y' columns were treated by a MinMaxScaler which helps in predicting the values not in the data in an effective manner. After the previous steps, the usual steps of

training and testing of the model with the artificial data is done with the final results as seen in the notebook. For the [4,4] data point, we get the result as 'Class [2]'

```
Class predictions on artificial dataset - Class: [2]
```

**Figure 39: Random forest model output on the artificial data**

Now, the addressing to the questions asked in the evaluation. In case of the decision trees, the order of 20 trees selected when logged in console was seen as follows which gives the respective result

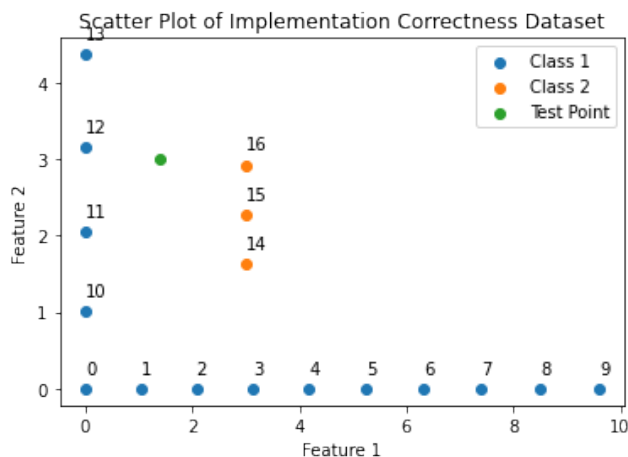
```
RF- most_common_label called [1 2 2 1 1 2 2 2 1 1 2 1 2 1 1 2 1 1 2 2]
nearest_feature_CIP: (11, 10, 12, 10)
```

**Figure 40: The 20 decision tree results**

Each value split is based on the information gained. It is dependent on the above

## 5.6 KNN Classifier

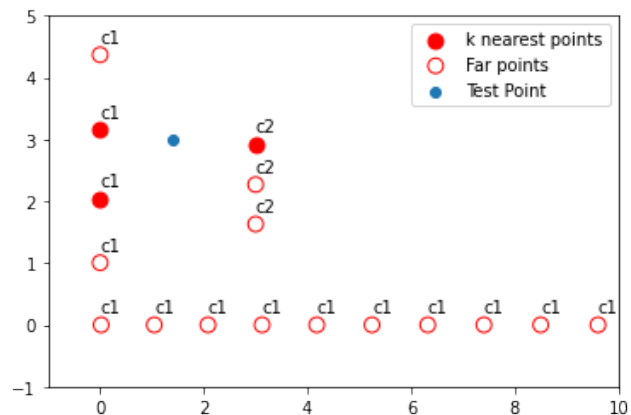
We built a K-Nearest Neighbors classifier from scratch, using the Manhattan and Euclidean distance metrics. We used the artificial dataset provided for this purpose. We create a scatter plot using the implementation correctness dataset, two classes (class 1 and class 2), and one test point (as shown in Fig: 41). We are utilizing the distance from the training data to determine which neighbors are closest to the test data point. Afterward, utilizing the Euclidean and Manhattan distance metrics, we output the predicted class labels for the given test data point.



**Figure 41: Scatter plot of implementation correctness dataset**

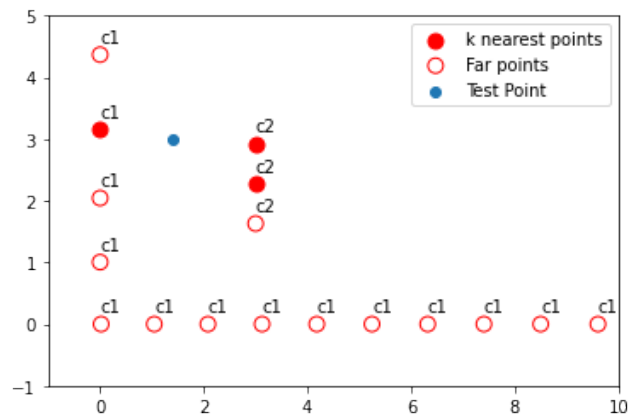
For the Euclidean distance metric, following the provided instructions, we can see from the scatter plot (as shown in Fig: 42), 2 class1 points are near the test point and 1 class2 point is near

the test point. Based on this, we can say, the test point belongs to class1.



**Figure 42: Scatter plot based on Euclidean distance**

For the Manhattan distance metric, following the provided instructions, we can see from the scatter plot (as shown in Fig: 43), 2 class2 points are near the test point and 1 class1 point is near the test point. Based on this, we can say, the test point belongs to class2.



**Figure 43: Scatter plot based on Manhattan distance**

## 5.7 References

- (1) Agglomerative Clustering (Shadrush Swaroop)
  - (a) <https://www.kaggle.com/code/vishwaparekh>
  - (b) <https://towardsdatascience.com>
  - (c) <https://www.youtube.com/watch?v=RdT7bhm1M3E>
  - (d) <https://stackoverflow.com/questions/9838861/scipy-linkage-format>
  - (e) <https://opensourceoptions.com/blog/10-ways-to-initialize-a-numpy-array-how-to-create-numpy-arrays/>



- (f) <https://www.w3schools.com/python/python-syntax.asp>
- (g) Referred sources for syntax support with respect to python including StackOverflow and GeeksForGeeks
- (2) Spectral Clustering (Sanjana Senthilkumar)
- (a) <https://www.kaggle.com/code/vishwaparekh>
- (b) <https://towardsdatascience.com/spectral-clustering-82d3cff3d3b7>
- (c) <https://juanitorduz.github.io/spectral-clustering/>
- (d) <https://arxiv.org/abs/0711.0189>
- (e) <https://analyticssteps.com/blogs/what-spectral-clustering>  
<https://arxiv.org/abs/0711.0189>
- (f) <https://analyticssteps.com/blogs/what-spectral-clustering>
- (g) <https://www.youtube.com/watch?v=YHz0PHcuJnk>
- (h) <https://stackoverflow.com/questions/4128755/how-to-convert-a-directed-graph-to-an-undirected-graph>
- (i) <https://stackoverflow.com/questions/16369714/transform-a-simple-directed-graph-to-a-simple-undirected-graph>
- (j) <https://machinelearninggeek.com/spectral-clustering/>
- (k) <https://cs.stackexchange.com/questions/56195/how-to-convert-a-directed-graph-to-an-undirected-graph-adjacency-matrix>
- (l) <https://www.geeksforgeeks.org/graph-and-its-representations/>
- (m) <https://www.geeksforgeeks.org/java-program-to-find-laplacian-matrix-of-an-undirected-graph/>
- (n) <https://www.programiz.com/python-programming/matrix>
- (o) <https://opensourceoptions.com/blog/10-ways-to-initialize-a-numpy-array-how-to-create-numpy-arrays/>
- (p) <https://numpy.org/doc/stable/reference/generated/numpy.subtract.html>
- (q) <https://sh-tsang.medium.com/tutorial-normalized-graph-laplacian-f74593feace7>
- (r) <https://towardsdatascience.com/k-means-clustering-explain-it-to-me-like-im-10-e0badf10734a>
- (s) <https://numpy.org/doc/stable/reference/random/generated/numpy.random.choice.html>
- (t) <https://www.w3schools.com/python/ref-math-dist.asp>
- (u) <https://www.geeksforgeeks.org/calculate-the-euclidean-distance-using-numpy/>
- (v) <https://medium.com/@tomernahshon/spectral-clustering-from-scratch-38c68968eae0>
- (w) <https://towardsdatascience.com/spectral-clustering-aba2640c0d5b>
- (x) [https://scikit-learn.org/stable/modules/generated/sklearn.metrics.pairwise\\_distances.html](https://scikit-learn.org/stable/modules/generated/sklearn.metrics.pairwise_distances.html)
- (y) <https://www.reneshbedre.com/blog/kmeans-clustering-python.html>
- (z) <https://stackoverflow.com/questions/6422700/how-to-get-indices-of-a-sorted-array-in-python>
- (1) <https://numpy.org/doc/stable/reference/generated/numpy.fill-diagonal.html>
- (3) DBScan Clustering (Akash Sundaresan Mahalaxmi)
- (a) <https://scikit-learn.org/stable/modules/generated/sklearn.cluster.DBSCAN.html>
- (b) <https://towardsdatascience.com/dbscan-clustering-explained-97556a2ad556>
- (c) <https://www.geeksforgeeks.org/dbscan-clustering-in-ml-density-based-clustering/>
- (d) <https://www.youtube.com/watch?v=RDZUdRSDOok>
- (e) <https://www.kaggle.com/code/josephstalinpeter/clustering-algorithms-breast-cancer-wiscosin>
- (f) <https://www.kaggle.com/code/raskoshik/tutorial-finding-outliers>
- (g) <https://towardsdatascience.com/an-introduction-to-t-sne-with-python-example-5a3a293108d1>
- (h) <https://scikit-learn.org/stable/modules/generated/sklearn.manifold.TSNE>
- (4) Multi-Layer Perceptron (Kishan Sivakumar)
- (a) [https://scikit-learn.org/stable/modules/generated/sklearn.neural\\_network.MLPClassifier.html](https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html)
- (b) [https://scikit-learn.org/stable/modules/generated/sklearn.model\\_selection.GridSearchCV.html](https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html)
- (c) <https://scikit-learn.org/stable/modules/generated/sklearn.pipeline.Pipeline>
- (d) <https://scikit-learn.org/stable/modules/generated/sklearn.pipeline.Pipeline>
- (e) <https://towardsdatascience.com/hyperparameter-optimization-with-scikit-learn-scikit-opt-and-keras-f13367f3e796>
- (f) <https://scikit-optimize.github.io/stable/modules/generated/skopt.BayesS>
- (g) <https://analyticsindiamag.com/beginners-guide-to-truncated-svd-for-dimensionality-reduction/>
- (h) <https://www.kaggle.com/code/parthsuresh/binary-classifier-using-keras-97-98-accuracy>
- (i) <https://www.kaggle.com/code/ptynecki/breast-cancer-prediction-with-mlp-99-5>
- (5) Random Forest Classifier (Deepak Urs G V)
- (a) <https://towardsdatascience.com/an-implementation-and-explanation-of-the-random-forest-in-python-77bf308a9b76>
- (b) <https://link.springer.com/article/10.1007/s41133-020-00032-0>
- (c) <https://www.youtube.com/watch?v=kFwe2ZZU7yw>
- (d) <https://www.youtube.com/watch?v=NxEHSAfFK8>
- (e) <https://www.youtube.com/watch?v=v6VJ2RO66Ag>
- (6) KNN Classifier (Puneet Singhania)
- (a) Official KNN Sklearn website
- (b) Official TruncatedSVD Sklearn website
- (c) learn.g2.com
- (d) Wikipedia
- (7) K-Nearest Neighbors Classifier (Tomernahshon)
- (1) <https://www.kaggle.com/code/vishwaparekh/cluster-analysis-of-breast-cancer-dataset>
- (2) <https://www.kaggle.com/code/dreamslab/eda-classification-and-clustering-diagnosticFeature-engineering>
- (3) <https://towardsdatascience.com/multilayer-perceptron-explained-with-a-real-life-example-and-python-code-sentiment-analysis-4108e4d9441>
- (4) <https://www.simplilearn.com/tutorials/deep-learning-tutorial/multilayer-perceptron>
- (5) <https://www.youtube.com/watch?v=7YaqzpBXw>
- (6) <https://www.geeksforgeeks.org/multi-layer-perceptron-learning-in-tensorflow/>
- (7) <https://wiki.pathmind.com/multilayer-perceptron>
- (8) [https://scikit-learn.org/stable/modules/generated/sklearn.neural\\_network.MLPClassifier.html](https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html)
- (9) <https://towardsdatascience.com/machine-learning-algorithms-part-12-hierarchical-agglomerative-clustering-example-in-python-1e18e0075019>
- (10) <https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html>
- (11) <https://www.youtube.com/watch?v=D8repXHkKdk>
- (12) <https://scikit-learn.org/stable/modules/generated/sklearn.cluster.DBSCAN.html>
- (13) <https://towardsdatascience.com/dbscan-clustering-explained-97556a2ad556>
- (14) <https://www.geeksforgeeks.org/dbscan-clustering-in-ml-density-based-clustering/>