CS235 Winter'23 Project Final Report: Default Project

Kunal Mittal
SID: 862131428

Mohit Asudani
SID: 862393012

Swastyak Gaur
SID: 862063974

Kailin Zhuo
SID: 862150180

Ankith Vijay
SID: 862394125

Saul Gonzalez
SID: 862064016

ABSTRACT

We present the findings of our CS235 Winter'23 Default project, a paper where we perform and evaluate classification and clustering models primarily on the Breast Cancer Wisconsin Diagnostic Dataset. We wish to determine which methods are best for classifying malignant and benign tumors, and attempt to cluster each type of data into groups of similar features. This paper uses Random Forest, Multi-Layer Perceptron, and K-Nearest Neighbor models for the classification tasks and DBSCAN, Spectral Clustering and Agglomerative Clustering with Single Linkage for the clustering methods. Our findings show what classification and clustering models could have advantages on datasets similar in nature to the cancer dataset we used in our research.

KEYWORDS

data mining, binary classification, clustering, Random Forest, Multi-Layer Perceptron (MLP), K-Nearest Neighbors, DBSCAN, Spectral Clustering, Agglomerative Clustering, dimensionality reduction, supervised learning, unsupervised learning

ACM Reference Format:
Kunal Mittal, Mohit Asudani, Swastyak Gaur, Kailin Zhuo, Ankith Vijay, and Saul Gonzalez. 2018. CS235 Winter'23 Project Final Report: Default Project. In *Proceedings of Data Mining Techniques (CS235 F22).* ACM, New York, NY, USA, 14 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

In this paper we compare different approaches to determine the performances of different classifications methods. Classification is a widely used machine learning technique that is used to assign labels to a given input based on prior knowledge[1]. Features and labels from existing information is used with various algorithms to best label unclassified data, while ensuring that there is a reasonable amount of confidence to say that the classification is correct. We compare three of the most popular methods here: Random Forest, which classifies each point based upon information gain on a tree-like structure; Multi-Layer Perceptron, which feeds the features into a neural network and outputs weights corresponding to the probability that it's a certain class; and K-Nearest Neighbors, which uses a distance function to see how far a point is from K-neighbors to classify it.

The performance of each of these clustering methods will be evaluated with the Breast Cancer Wisconsin Diagnostic Dataset. We will then compare the results found from each of the three clustering methods to determine which method performed the best given this spread of data.

We also evaluate how different clustering methods are able to accurately divide data into smaller clusters with related items. Clustering is a machine learning technique that breaks a given dataset into groups, with the goal of having items in the same groups be close to each other[2]. We analyze the following three clustering methods: Spectral Clustering, which is a density based machine learning model that uses eigenvalues to determine the smallest split of two clusters; DBSCAN, which is also a density based model that can create non-uniform clusters using a propagation-like technique; and Agglomerative Clustering with Single Linkage, which creates clusters by continuously merging points into clusters using the single linkage criteria.

To demonstrate the correctness of our implementations, we also ran our algorithms on a much smaller dataset. The correctness dataset consists of 17 objects in total, with 2 features each, and we verify the distinguishable inputs using the given class labels.

2 PROPOSED METHODS

2.1 Multi-layer Perceptron

2.1.1 Overview. Multi-layer Perceptron or MLP is a supervised learning algorithm that feeds feature values to a feed-forward neural network to give out all the hyperparameters/weights corresponding to the probability that it belongs to some class (in case of classification, the different objective for regression problems). It consists of multiple layers of interconnected neurons that process input data and produce output predictions. These hyperparameters are then iteratively updated in the training process to find the best convergence point to find the near-optimum solution.

2.1.2 Classification. In MLP classification, the aim is to predict the input data's class labels on the basis of a set of features. The input layer of the MLP receives the input data, processes it through one or more hidden layers, and then generates output predictions in the output layer.

The network can recognize intricate patterns in the data by applying non-linear changes to the input data using neurons in the hidden layers. The neurons in the output layer reflect several potential class labels.

MLP training involves modifying the weights of the connections between neurons to minimize the difference between the expected

CS235 F22, Fall 2022 quarter,, Riverside, CA

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/XXXXXXX.XXXXXXX

output and the true labels of the training data. An optimization approach like stochastic gradient descent is used for this.

2.1.3 Grid Search. In order to identify the best architecture and hyperparameters for our neural network, We did a grid search over the space of hyperparameters such as

- Number of hidden layers - 2
- Width of each layer - (50,100) and (50,50) (but 50,100 and 100,200 works better in general when trained exclusively)
- Activation function per layer - Relu
- Optimizer - ADAM (Though SGD worked just as well as ADAM but not always)
- Learning rate - (1e-3, 1e-2)

2.1.4 Bayesian Optimization. Bayesian optimization is a method for enhancing difficult-to-evaluate complex black-box functions. It is especially helpful in scenarios when the objective function is non-convex, non-linear, and where the ideal set of hyperparameters is uncertain.

The fundamental principle underlying Bayesian optimization is to first create a probabilistic model of the objective function based on the data at hand, and then use this model to decide which hyperparameters to assess next based on the model's predictions. The model can be modified as new data becomes available, allowing the optimization process to focus on promising portions of the search space.

The model typically is in the form of a probabilistic Gaussian process, which defines a distribution over functions. In order to direct the search toward promising areas, the model can be used to estimate the mean and variance of the objective function at any location in the search space. The experimentation section discusses more on the hyperparameters that were obtained using this method.

2.1.5 Cross Fold Validation. Cross-validation is a method for assessing a machine learning model's performance by segmenting the given data into numerous subsets, or "folds." This process is done several times, each time utilizing a different subset of the data for evaluation. The model is trained on a subset of the data and then evaluated on the remaining data. To provide a more accurate indication of the model's performance, the results are then averaged across the various folds.

Stratified k-fold cross-validation is used, in which the data is separated into k equally sized folds with almost identical data distributions. The model is then trained on k-1 folds, followed by an assessment of the final fold. This process is repeated k times, with each evaluation using a different fold. The model's performance is then estimated by averaging the results across the k-folds.

When a model is trained on a little quantity of data, there is a chance of overfitting, which can be reduced with the use of cross-validation. Cross-validation gives a more reliable assessment of the model's performance and aids in locating potential sources of bias or variance in the data by repeatedly training and evaluating the model on various subsets of the data. The following averages are calculated for the 10 iterations of KFolds using 10 splits: Precision, recall, and F1-score for both classes. The average cross-validation accuracy calculated for 10 folds is 0.953125. The test accuracy for learning rate 1e-3 is calculated to be 0.985074.

2.1.6 Inputs. The above functions (Classification, Grid Search, Bayesian optimization, and Cross Fold Validation) were all used together on a varying set of initial inputs. We tested the performance of the models using the original dataset which is normalized and the column "Unnamed:32" is removed. Divided the dataset into training and test at 10% and remaining training data to train and validation set at 20%. Lastly converted to tensors for the data loader. When the data was further visualized, there was no class disparity either.

2.2 K-Nearest Neighbors (KNN)

2.2.1 Overview. K-Nearest Neighbors is an algorithm that, for a given input, tries to assign the input a label based on the labels of the neighbors around it. The amount of neighbors "k" is a hyperparameter, and the distance can be computed in a variety of ways. The two methods that will be looked in this paper will be the Manhattan distance and the Euclidean distance.

2.2.2 Classification. For each point in an input of unclassified objects, the distance from each object to every other object that's in the known dataset is calculated. Once the distances have been calculated, they are sorted in ascending order (where the closest objects have lower distance values). Then for each point, a label is assigned based on the majority of the labels that the unknown point is closest to.

2.2.3 GridSearch. Grid search techniques were utilized to try to determine the best amount of neighbors to use. For this paper, k values from 1 to 30 were utilized.

2.2.4 Cross Fold Validation. With each number of neighbors from 1 to 30, K-Folds cross validation was used to determine the performance of our models. The *scikit*'s K-Folds cross validator was used to split the data into training and test splits[3]. KFolds was performed with 10 splits, where for the 10 iterations, the following averages were computed: F1-Score (macro); F1-Score, precision, and recall for malignant classifications; and F1-Score, precision, and recall for benign classifications. These will be used later to produce metric vs. hyperparameter figures, along with comparing our model's performance against other models.

2.2.5 Inputs. The above functions (Classification, GridSearch, and Cross Fold Validation) were all used together on a varying set of initial inputs. We tested the performance of the models using the original dataset, the dataset reduced with low rank approximation using the SVD, and the dataset reduced to the bottleneck layer of an MLP-based AutoEncoder. The approximation ranks for the SVD-feature reps were determined by finding a "low" value and a "high" value. The low value was the number of features before the singular values dropped, and the high value is the number of features after. Latent layers that were 20% of the original input and 5% of the original input were used for the MLP-based portion.

All models were tested with both the Manhattan and the Euclidean distance metrics.

2.3 Random Forest Tree

2.3.1 Overview. Random forest tree is uses multiple decisions tree and classified data through majority voting(prediction made by

individual tree). Decision tree splits input data into subset and the split is based on information gain for this project. The node spliting will stop once

2.3.2 Maximize information gain. The entropy is used to evaluate how to split the node. In random forest, we want to minimize the entropy, in terms of maximize information gain.

2.3.3 Basic steps to construct random forest tree. 1. bootstrapping - randomly select data from original dataset to create a new dataset
2. feature selection - randomly select features from bootstrapping dataset
3. construct decision tree - repeat step 2 for every node and split the feature into 2 group
4. split the node - use information gain to split
5. produce prediction - based on number of appearances of labels produce by individual tree.

2.3.4 Cross Fold Validation. 10-folds cross validation was used to determine the performance of the random forest tree, the average of precision and recall are computed.

2.4 DBSCAN

2.4.1 Overview. DBSCAN is a density based algorithm that uses two given inputs: ϵ and *min_pts* as hyper-parameters. Using these hyper-parameters we group points by using ϵ to determine which points in the vicinity of our "core point" are considered part of the "neighborhood" and we use *min_pts* to determine if we need to continue propagating our algorithm. More details on what core points are and how we continue and stop our algorithm are listed in the following sections.[3]

2.4.2 In Depth Explanation. Here is pseudocode detailing how DBSCAN works(based off our implementation) in Algorithm 1 [4]

2.4.3 GridSearch. As the from scratch implementation of DBSCAN is not optimized, the run time is rather long. Hence we do a small grid search to look for optimal hyper-parameters. The ranges for these hyper-parameters in the grid search are inspired by the grid search done on the fast and optimized DBSCAN library. We utilize the range from 10 to 20 for ϵ and the range 40 to 50 for *min_pts*. We maximize the silhouette score in order to get the best hyper-parameters. This however, may cause a very low normalized mutual information score. The grid search yielded hyper-parameters of of 12 for ϵ and 43 for *min_pts* with a maximum silhouette score of 0.6606668813897673.

2.4.4 Cross Fold Validation. We do K-Fold cross validation with a K value of 10 with hyper-parameters of 12 for ϵ and 43 for *min_pts*. The silhouette score and the normalized mutual information are shown in figures 18 and 19 later. The y-axis depicts the respective score and the x-axis show which fold we are on. For example: Fold 1 means we are on the first iteration and are ignoring the first tenth of the data; Fold 2 means we are on the second iteration and are only ignoring the second tenth of the data etc.

2.4.5 Inputs/Dataset. We did normalize data using the python *StandardScaler*() library to ensure euclidean distance between points isn't dominated by a particular feature. We also removed unnecessary columns such as the id column, the label column, and an "unamed32" column.

Algorithm 1 DBSCAN Detailed Pseudocode

```
procedure DBSCAN(dataframe, ε, min_pts)
    labelcolumn ← −1
    visitedcolumn ← False
    for row in dataframe do
        if row has not been assigned a cluster then
            candidateSet ← []
            for row2 in dataframe ≠ row1 do
                if row2 is within ε distance of row then
                    candidateSet ← candidateSet + row2
                end if
            end for
            if len(candiateSet) ≥ min_pts then   ▷ Core point
                row is now officially part of a NEW cluster
            end if
            for candidateRow in candidateSet do
                if candidateRow is not visited or labelled then
                    candidateRow is now visited      ▷ Update col
                    for r3 in dataframe ≠ candidateRow do
                        ccSet ← []
                        if r3 within ε of candidateRow then
                            ccSet ← ccSet + r3
                        end if
                    end for
                    if len(ccSet) ≥ min_pts then    ▷ Core point
                        candidateSet ← candidateSet + ccSet
                    end if
                    if candidateRow has no label then
                        candidateRow has same cluster as row
                    end if
                end if
            end for
        end if
    end for
end procedure
```

2.5 Spectral Clustering

2.5.1 Overview. Spectral clustering is a powerful technique used in unsupervised machine learning to cluster data points into groups. The approach is based on analyzing the eigenvalues and eigenvectors of a similarity matrix, which is typically derived from the data itself. It uses *n_clusters* its hyper-parameter. Using these hyper-parameters we group points by using *n_clusters* to determine the number of eigenvectors to use for the spectral embedding. Additional detail regarding how the clustering is implemented, is highlighted in the next section

2.5.2 Clustering. The general idea of spectral clustering is to represent the data points as nodes in a graph, where the edges between nodes represent the similarity between the data points. The similarity measure could be any metric, such as distance, correlation, or kernel function. The similarity matrix is then constructed by computing the pairwise similarities between all data points.

Once the similarity matrix is obtained, the next step is to compute its eigenvectors and eigenvalues. The eigenvectors are then used to transform the data into a new space where clustering is performed. The spectral embedding is obtained by taking the top k eigenvectors corresponding to the k largest eigenvalues, where k is the number of clusters.

The clustering is then performed on the spectral embedding, typically using a standard clustering algorithm like k-means. The resulting clusters are then mapped back to the original data space, allowing the original data points to be assigned to their respective clusters.

2.5.3 Cross Fold Validation. Cross-validation is necessary in spectral clustering because it allows us to evaluate the performance of the algorithm on a given dataset and choose the best hyper parameters, such as the number of clusters (*n_clusters*). The process of cross-validation in spectral clustering with k=10 and hyper parameters *n_clusters* involves splitting the dataset into k equal-sized parts or "folds." For each fold, we build a spectral clustering model on the other k-1 folds and evaluate the performance of the model on the testing set. We then repeat this process for each time, recording the clustering performance for each value of *n_clusters* that we want to consider. Finally, we calculate the average clustering performance across all k folds for each value of *n_clusters* and choose the value that gives the best average performance.

2.5.4 Inputs/Dataset. The data is normalized using the python *StandardScaler*() library to ensure euclidean distance between points isn't dominated by a particular feature. We also removed unnecessary columns such as the id column, the diagnosis column, and an "unamed32" column.

2.6 Agglomerative Clustering

2.6.1 Overview. Agglomerative Clustering is a hierarchical unsupervised clustering method that uses the distance between points as the clustering criterion. This particular method is a bottom-up approach that starts with n amount of clusters, where n is the number of points in the data set, and continues to merge clusters using a distance metric until a termination condition is met.

2.6.2 Termination Condition. The termination condition for the method can be a number of clusters, a distance threshold, etc. There is no "ideal" termination condition that fits all problems. For our implementation of this method, we use the number of clusters as the termination criteria. After performing a search for the ideal number of clusters for the termination condition, 2 clusters worked the best for our given dataset.

2.6.3 Single Linkage. The crux of the model is the distance metric or "linkage", which is used to determine which clusters are being merged together at the given iteration of the method. There are many linkage metrics that can be used for this method such as complete linkage, average linkage, ward linkage, etc. however the focus of our method is single linkage. Single linkage merges clusters based on the shortest distance over all possible pairs between the clusters. The distance calculation portion of the criteria is using LP norm in order to generate the distance between the points that are being compared. [5]

$$\text{DIST-SINGLELINK}(\{x_n\}_{n=1}^{N}, \{y_m\}_{m=1}^{M}) = \min_{n,m} ||x_n - y_m||,$$

Figure 1: Single Linkage Criteria Formula

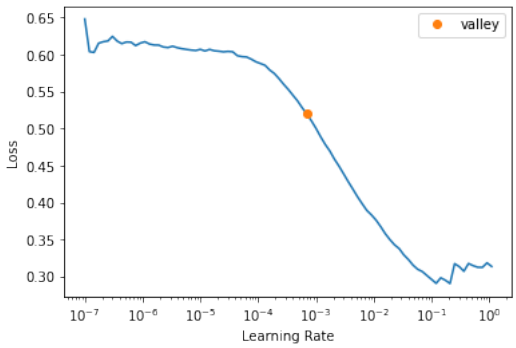


Figure 2: Convex function for lrFind

2.6.4 Cross Fold Validation. In order to validate the performance of the method, we performed a 10-fold cross-validation with the number of clusters set to 2. The distributions of the silhouette scores and the normalized mutual information scores over the 10 folds are shown in the figures below in our experimental evaluation section. The y-axis depicts the respective score and the x-axis shows which fold produced the score.

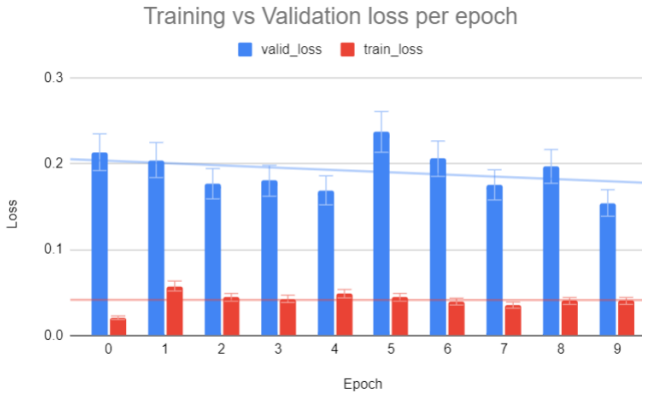
2.6.5 Inputs. Before using data from the breast cancer data set for agglomerative clustering, we removed unnecessary columns such as the id column, the malignant/benign column, and an "unnamed32" column.

3 EXPERIMENTAL EVALUATION

3.1 Multi-Layer Perceptron

3.1.1 Base Implementation. Sklearn was not used for MLP implementation. Rather Pytorch and Fastai are used. Simple neural network architecture is designed with the first layer as a batch norm layer with input 30 (As we have 30 features in our Dataset) for a batch size of 64 giving 200 output size. 2 Hidden Layers having 200 and 100 neurons respectively along with a Relu and a batch normalization layer. Finally, 1 Output layer classifying into 2 categories. By building networks with various architectures, the values for the number of layers and neurons per layer were determined. The following parameters and inputs have been used for the classifier:

- Hidden Layer Size = (200,100)
- Batch size = 64
- Activation Function = relu
- Iterations = 10
- Kfold = 10
- Total parameters = 26,862
- Optimizer = Adam
- Loss function = Flattened Cross Entropy Loss



Grid Search	Constraints	Result
#hidden layers	(1,3)	2
Layer width	(50,200)	(50,50)
Activation fn	Relu, Sigmoid	Relu
Optimizer	Adam, SGD	Adam
Learning Rate	(1e-3, 0.1)	2e-03
Max_iters	(100, 500)	100

	precision	recall	f1-score	support
B	0.99	0.96	0.97	71
M	0.93	0.98	0.95	43
accuracy			0.96	114
macro avg	0.96	0.97	0.96	114
weighted avg	0.97	0.96	0.97	114

Figure 3: Classification report: MLP_grid

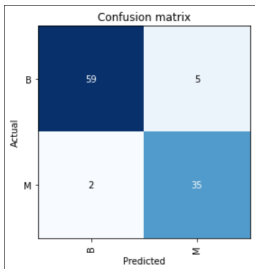


Figure 4: Confusion matrix: MLP_grid

3.1.2 Grid Search: MLP_grid. Experimented with various libraries as there is no grid search library for PyTorch. Skorch and Sklearn were used for this implementation. Grid search, in general, can give useful insights but it doesn't cover a range of parameters but rather a definite value of those parameters resulting in multiple grid searches. The results shown in the table are the ones having better performance when trained exclusively.

3.1.3 Bayesian optimization: MLP_BO. The *bayesianOptimization* function is imported from the bayes_opt library to perform this method. BOTORCH was also experimented with but the documentation of that library is not that good to understand clearly. The parameter constraint ranges and the result is shown in the table. After selecting the parameters obtained, the test accuracy is coming out to be 98.1818%.

Bayesian Optimize	Constraints	Result
#hidden layers	(1,3)	1.185
Layer 1	(10,200)	158.048
Layer 2	(100,1000)	100.102
Layer 3	(200,2000)	744.198
Dropout	(0.01, 0.5)	0.214
Optimizer	Adam, SGD	Adam
Learning Rate	(1e-3, 0.1)	0.0155
Weight Decay	(4e-4, 0.4)	0.0748

	precision	recall	f1-score	support
B	0.97	1.00	0.99	71
M	1.00	0.94	0.97	32
accuracy			0.98	103
macro avg	0.99	0.97	0.98	103
weighted avg	0.98	0.98	0.98	103

Figure 5: Classification report: MLP_BO

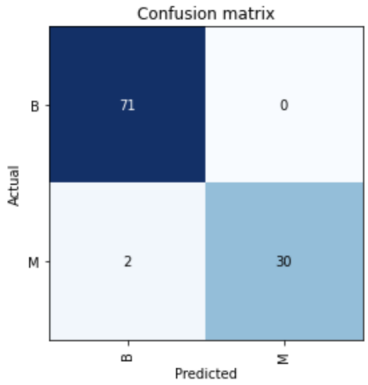


Figure 6: Confusion matrix: MLP_BO

3.1.4 Cross Fold Validation. The following averages are calculated for the 10 iterations of KFolds using 10 splits: Precision, recall, and F1-score for both classes. The average cross-validation accuracy calculated for 10 folds is 0.953125. The test accuracy for learning rate 1e-3 is calculated to be 0.985074.

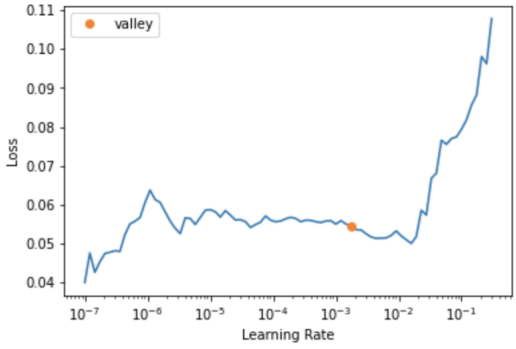


Figure 7: Best Local optimum

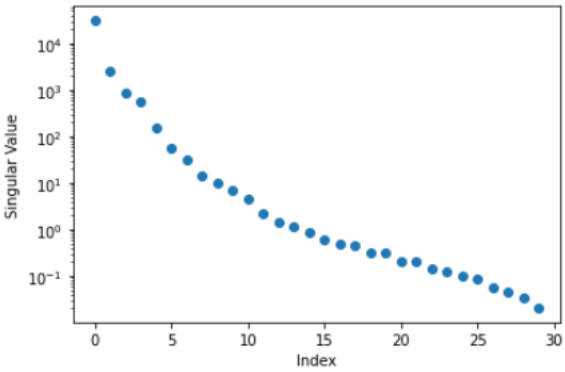
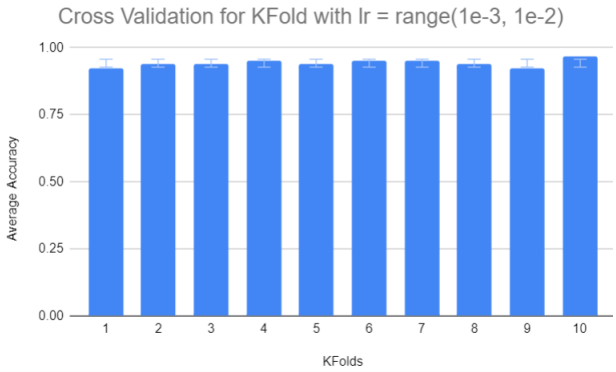


Figure 8: Relevant drop on graph between 4 and 5 features



3.1.5 Emulating the Kernel Trick with an MLP: SVM. SVM (Support Vector Machine) works by finding a hyperplane that separates the data into different classes, with the goal of maximizing the margin between the hyperplane and the closest data points.

The hyperplane is determined by a set of weights or coefficients, which are learned during the training process. The SVM algorithm seeks to find the hyperplane that maximizes the margin between the closest data points of different classes, which reduces the risk of overfitting and improves the generalization performance of the model.

SVMs can also handle non-linearly separable data by mapping the input data to a higher-dimensional space using a non-linear function called a kernel function. In the higher-dimensional space, the SVM can find a hyperplane that separates the data into different classes.

SVMs are particularly useful when dealing with high-dimensional data, where the number of features is larger than the number of samples. Experiments were done on the following kernel functions and their hyperparameters were obtained using grid search. For MLP_Kernel, the number of layers was selected with bayesian optimization as the hyperparameters selected are performing the best and the architecture is already like a funnel.

- Linear kernel (not converging for this problem)
- RBF kernel (radial-basis function)
- Polynomial kernel

Both RBF kernel and Polynomial kernel converge resulting in RBF output very close to the MLP_kernel. Both were performing poorly in recall for the malign class as compared to the best MLP model chosen.

3.2 K-Nearest Neighbors (KNN)

3.2.1 Base Implementation. When running the KNN method on the entire dataset, we see that our best performing values were with 9 neighbors with Manhattan distance, and 12 neighbors with the Euclidean distance. The F1-macro scores of the two were roughly 0.93864 and 0.93650, respectively. The F1-macro score was also the highest in the entire KNN method, somewhat beating the methods with dimensionality reduction using SVD and the AutoEncoder.

3.2.2 SVD Reduction. SVD Reduction began with finding suitable "low" and "high" values to create our feature reps. numpy's *linalg.svd* allows us to break the features up into three vectors; the left singular and right singular vectors, and the array of singular values[6]. Plotting the array of singular values will let us see where the singular values drop significantly, as shown in Figure 8. After reducing the dataset to include 4 and 5 features respectively, we evaluated the performance of the models, and found that they were slightly weaker than the implementations on the entire dataset. The F1-macro scores were 0.88636 with Manhattan distance on the low rank feature rep, and 0.88216 with Euclidean distance on the high rank feature rep.

3.2.3 MLP-based AutoEncoder Reduction. The other dimensionality reduction technique used to evaluate the KNN models was the reduction with scikit's *MLPRegressor*, which we can specify the hidden layer's activation function. We set up our AutoEncoder to have two encoder and two decoder layers, which utilize ReLu activations. We evaluated the models using an input that has 20% of the orignal features and 5% of the original features. As we are only interested in the latent layer before the decoding levels, we exracted the latent layer as a numpy array and used it for model evaluation.

The performance using this technique of reduction resulted in slightly better performance than that of SVD Reduction, but still faltered to the evaluations on the entire dataset by just a little. When using the 5% feature size latent layer, we observed that the F1-macro

scores using the Manhattan distance and Eucidean distance metrics were 0.91789 and 0.91531, respectively.

The following performance data was collected on each of the models: F1-macro score; F1-Score, Precision, and Recall for Malignant diagnoses; and F1-Score, Precision, and Recall for Benign diagnoses. The performance of the F1-macro scores, Precision and Recall on the Malignant diagnoses are shown in Figures 9 through 13.

The performance of the base implementation and the AutoEncoder implementations are close in performance, while the SVD implementation faltered further behind, not being able to hit a a 90% or above F1-macro score.

3.3 Random Forest Tree

Dimensional reduction was not performed, bootstrapping was performed on the entirety of the dataset. Grid search was used to find the best hyper parameters, max depth and number of trees in the random forest. Then cross validation was applied to determine the performance metrics(precision and recall), of the random forest tree. As shown in figure 14 and figure 15, due to computation time and to prevent over-fitting, constrain the max depth of tree to be 12 and number of trees to be 70 appear to be reasonable. As shown in figure 16 and figure 17, the precision and recall indicate we have a well performed random forest.

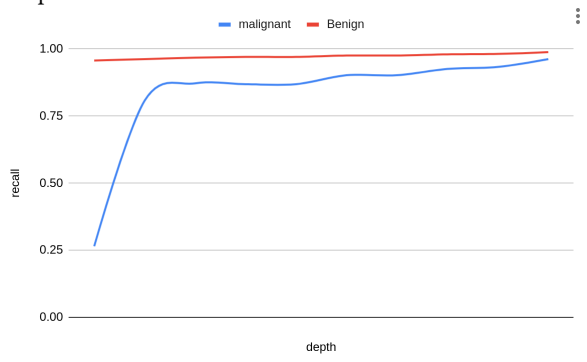


Figure 14: Hyper-parameter vs recall

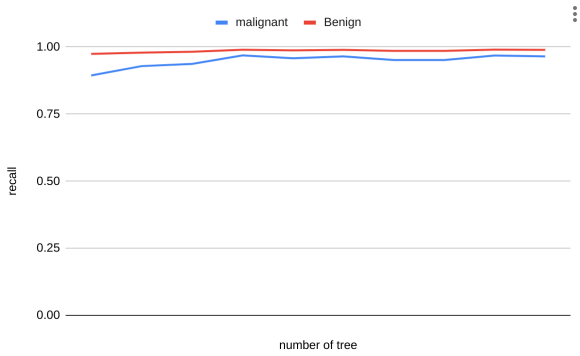


Figure 15: Hyper-parameter vs recall

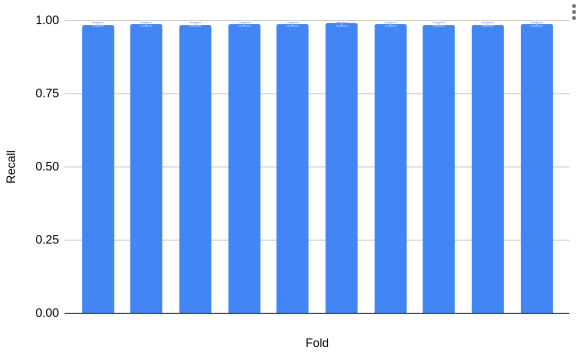


Figure 16: recall vs fold

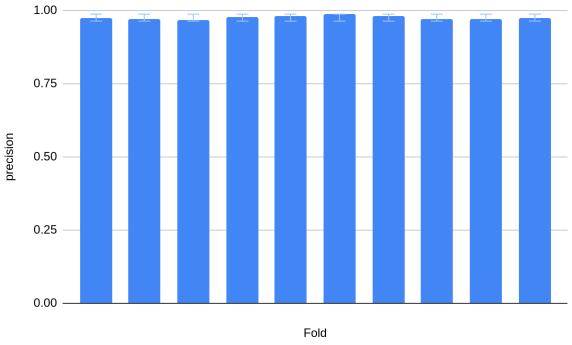


Figure 17: precision vs fold

3.4 DBSCAN

3.4.1 Base Implementation. No Dimensional reductionality takes place, all tumor-relevant features are kept and normalized. We use grid search to find the best hyper-parameters so we run 10-fold cross validation with an ϵ of 12 and a *min_pts* of 43. The graphs show that silhouette score does not vary and error is reasonable. Normalized Mutual Information on the other hand, varies and have rather large error bars in proportion to the Normalized Mutual Information scores, but the average score still appears fine. Average silhouette score across the 10 folds/iterations is 0.65882 and average normalized mutual information score across the 10 folds/iterations is 0.00334. Figure 18 and 19 shows a breakdown of the scores per fold

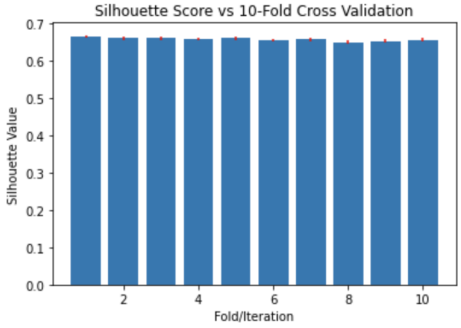


Figure 18: Silhouette Scores

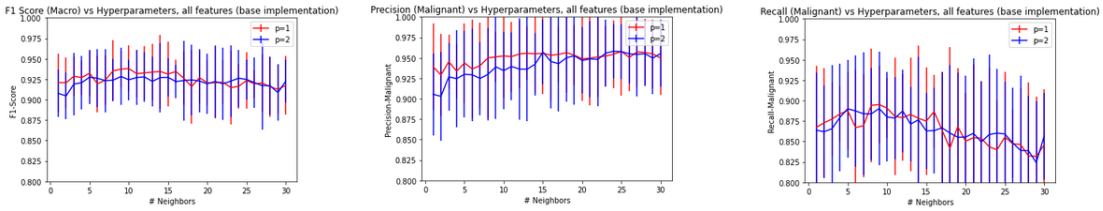


Figure 9: KNN model performance with the base implementation (entire dataset)

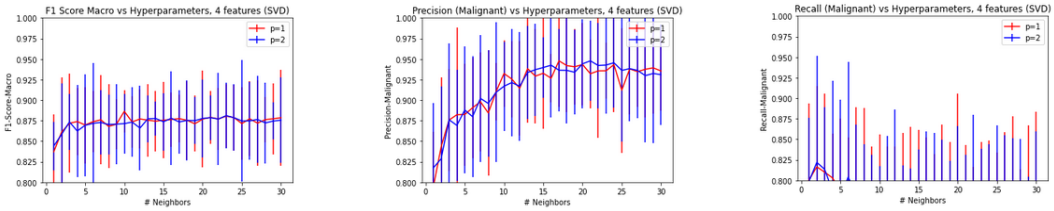


Figure 10: KNN model performance with SVD-based feature rep (low rank)

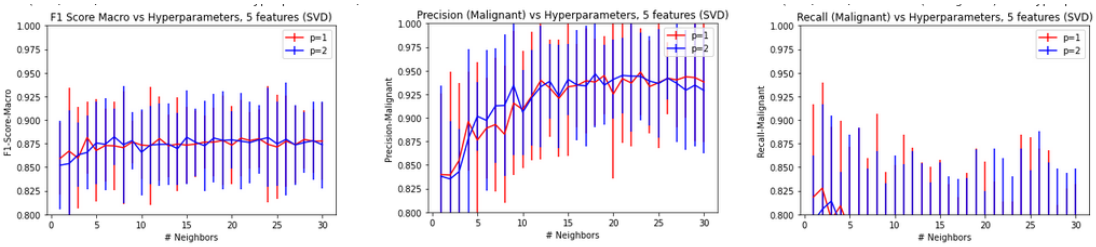


Figure 11: KNN model performance with SVD-based feature rep (high rank)

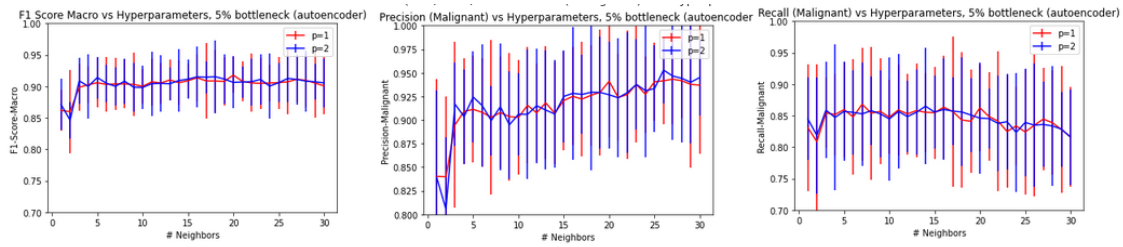


Figure 12: KNN model performance with latent layer 20% of original features

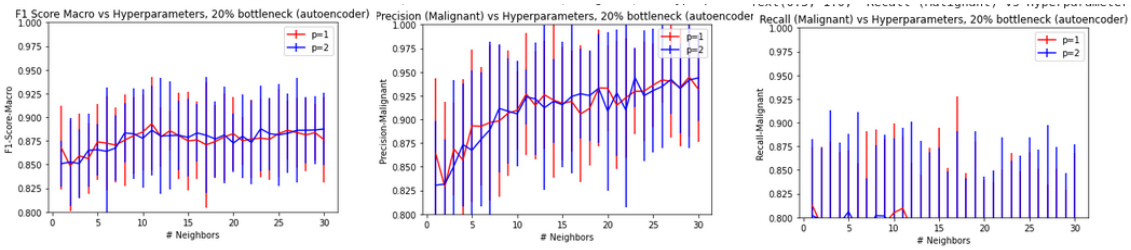


Figure 13: KNN model performance with latent layer 5% of original features

Depicted in figure 18 are the Silhouette Scores with error bars. Silhouette Scores are relatively stable accross each iteration and have very little variance. The overall average indicates that we have strong clusters at hand.

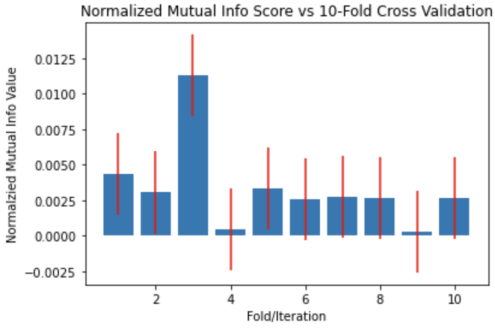


Figure 19: Normalized Mutual Information Scores

Depicted in figure 19 are Normalized Mutual Information Scores with error bars. Normalized Mutual Information fluctuates and has error bars that are rather large in proportion to . Overall, the averages of each score are normal

3.5 Spectral Clustering

3.5.1 Base Implementation. There has been no dimensional reduction, all the previous relevant features are kept and normalized. The cross validation shows the value of the hyperparameter *n_clusters* that yield the highest mean Silhouette score and NMI score over all 10 folds of the data. The Silhouette score measures how well the clusters are separated and how dense they are, while the NMI score measures the agreement between the true labels and the predicted labels. In this case, the graphs depict that neither the silhoutte score nor the the Normalized Mutual Information score vary too much from one another.

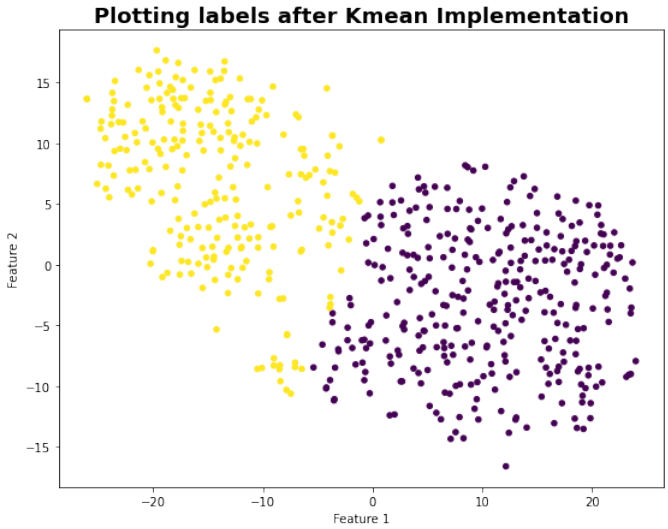


Figure 20: Plotting labels after KMeans Implementation

Depicted above is the plot of the labels after KMeans Clustering has been implemented from scratch.

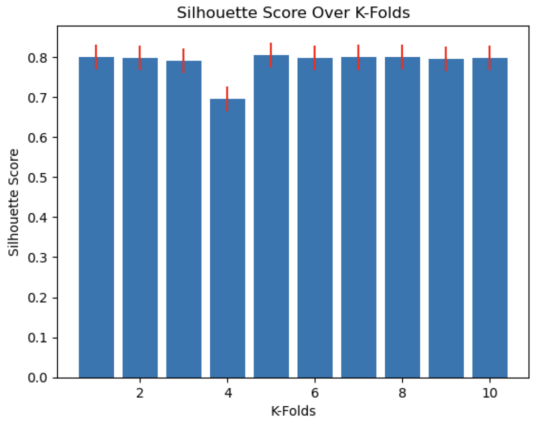


Figure 22: Silhouette Score Distribution

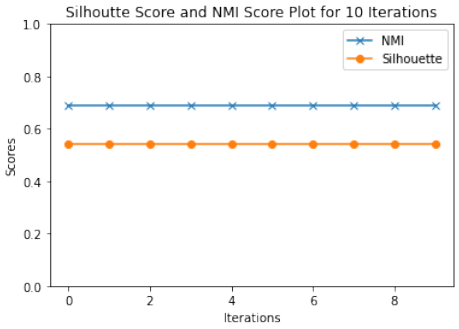


Figure 21: Silhoutte Score and NMI Scores for Spectral Clustering

Depicted above is the Silhoutte score and NMI scores for the Spectral Clustering Implementation from scratch.

3.6 Agglomerative Clustering

3.6.1 Base Implementation. The model starts at *n* amount of clusters, or 569 in our specific case, and beings to generate a distance matrix for every cluster on the graph. The distance matrix is generated using the single linkage criteria. The distance calculation for the linkage was done using numpy's LP Norm function. [7] After constructing the distance matrix, the two closest clusters were selected and merged together. This newly constructed cluster is then added back into the cluster list. This process continues until we have our desired amount of clusters, in our case it is 2. Over the 10-folds, our model had an average silhouette score of 0.78806 and an average normalized mutual information score of 0.00576266.
As we can see from the graphs, the distributions for the normalized mutual information scores don't change much and the silhouette score only dropped a bit in the fifth fold. The relatively high average silhouette score informs us that our model produced strong clusters. However, the low average normalized mutual information score informs us that our model did not find groupings that are closely related to the malignant/benign labels from the dataset.

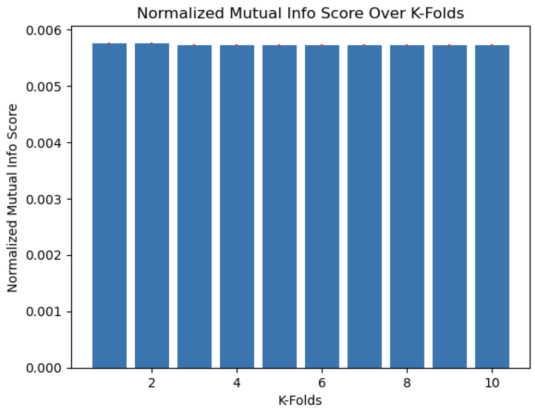


Figure 23: Normalized Mutual Information Score Distribution

4 DISCUSSION & CONCLUSIONS

4.1 Malignant Classification Scores

Method	F1	Precision	Recall
Random Forest	0.96	0.97	0.95
MLP	0.97	1	0.95
KNN	0.920084	0.95231	0.89082

4.2 Benign Classification Scores

Method	F1	Precision	Recall
Random Forest	0.98	0.97	0.98
MLP	0.99	0.97	1
KNN	0.95601	0.93751	0.97571

4.3 Classification Score Graph

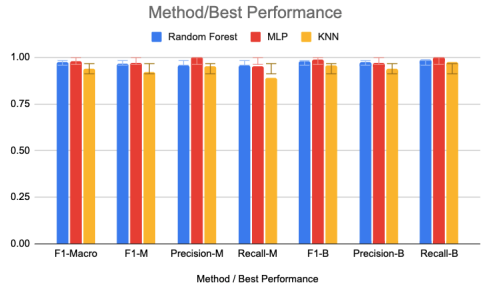


Figure 24: Figure above shows the classification methods and their corresponding scores.

4.4 Classification Explanation

For each of the methods (Random Forest, MLP and KNN), the scores of the best performances were taken for us to analyze. The data show that, on average, the best model from the MLP implementation performed the best, while the KNN method ranked third. As mentioned earlier in the paper, the hidden layers have neurons that apply non-linear transformations to the data, which potentially allows for more learning on complex relationships between features. This potentially explains why MLP performed the best. For classification problems similar to the one we're trying to answer on the Wisconsin Breast Cancer dataset, it may be advantageous to use a MLP model to classify unknown cases, based on our results. Random Forest performed similarly to MLP, and would also be a valid choice for classification. If a less computationally expensive, lazy learning model is desired, KNN still offers good performance, but the recall scores have desire a little more. For high stake predictions like cancer clarification, models with high recall are likely more preferred.

4.5 Clustering Methods and Scores

Model	Silhouette	NMI
DBSCAN	0.65882	0.0033374
Spectral	0.689513	0.0054
Agglomerative	0.78806	0.0057266

4.6 Clustering Score Graph

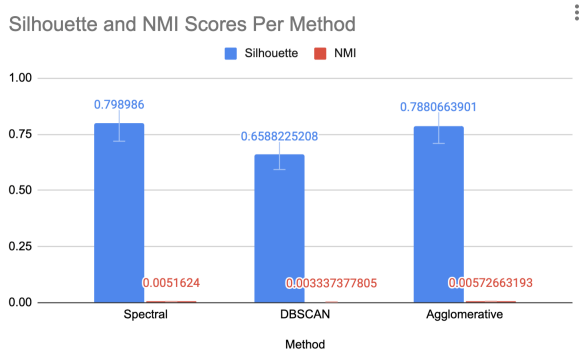


Figure 25: Figure above shows the clustering methods and their corresponding scores. Normalized mutual information scores may be difficult to see, but they exist nonetheless

4.7 Clustering Explanation

Each clustering method has their silhouette score and their normalized mutual information scores recorded to determine which method would act as the best clustering method for our breast cancer data set. Our hyper-parameters were the intention of maximizing silhouette score which could potentially explain why the normalized mutual information is so low for all methods. The reason for this is because the higher the silhouette score, the better clusters our method has made. Whereas the higher our normalized mutual information score is, the closer our cluster labels relate to the group truth labels given to us in our breast cancer dataset. Looking at the table, we can see that when it comes to silhouette scores, Spectral clustering is at the the top. This means it creates the clusters where points are very nicely grouped and related to the correct cluster, rather than other incorrect clusters. Agglomerative comes by as a close second, and finally DBSCAN is at the bottom in both silhouette and normalized mutual information score. Our theory behind these scores are due to a multitude of factors. For one, it could simply be how each model handles splitting the data.

DBSCAN and Spectral Clustering are density based clustering algorithms but they way they handles splitting clusters is very different. DBSCAN and agglomerative clustering both do clustering differently, but how they merge points into one cluster seems similar. From the data, it appears that maybe the way DBSCAN and agglomerative clustering handle making clusters is worse than spectral clustering on this dataset. This answers why spectral clustering is doing the best, but why is agglomerative performing better than DBSCAN? We believe the reason behind this is that the clusters in the dataset are more uniform and hence agglomerative performs better. Spectral Clustering is unaffected by this trait just because of the nature of the spectral clustering model.

5 IMPLEMENTATION CORRECTNESS REPORT

5.1 Random Forest Tree

The goal of decision trees used by random forest algorithm is to reduce impurity, meaning it is trying to create a subset contains only one label. The order is purely random, but it is making a subset homogeneous. Once the random forest tree picks a feature, it uses information gain, a measurement of homogeneous, to split that feature. As the algorithm is trying to reduce impurity, the higher the information gain, the better. Furthermore, the random forest was performed on implementation_correctness_dataset.csv, and test its classification via 20 random forest trees on the data point [4,4]. 7 out of 20 random forest trees consider the data point [4,4] to be class As shown in figure 4, the data point [4,4] is determined to be class 1. The final classification does not appear to be correct from the graphical view. However, consider the the class imbalance in the training dataset, this is a expected result.

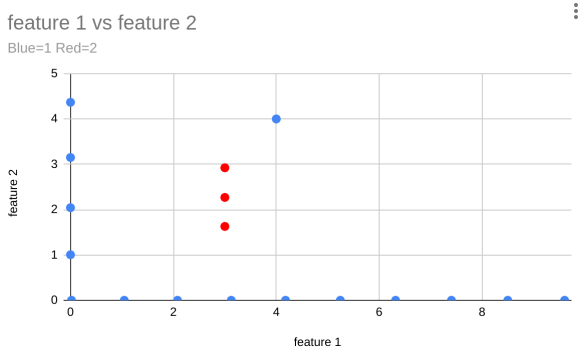


Figure 26: data point [4,4] is determined to be class 2

5.2 Multi-layer Perceptron

The correctness of the MLP_grid, MLP_BO, and MLP_kernel is determined using cross-validation and emulating the implementation of the SVM kernel with MLP by dimensionality "blow-up" of the feature space. MLP_kernel has a "funnel" shape, i.e., the width of the hidden layers progressively reduces the dimensionality of the data. RBF and Polynomial kernel functions are used for SVM implementation and their hyperparameters are chosen using grid search. MLP kernel was further optimized by visualizing the convex function, using sliced learning rates (by /10) per epoch, and by setting the momentum for converging. Even though the test accuracy for MLP_kernel is 0.985 which is a little better than SVM, the actual

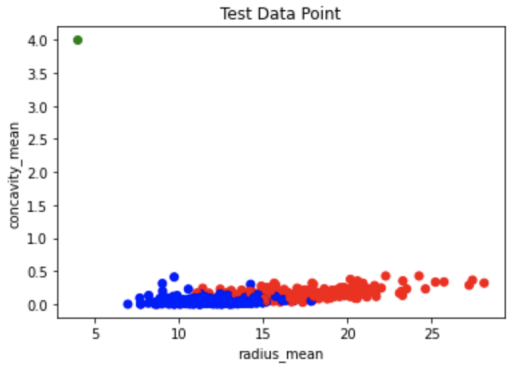


Figure 27: Scatter plot of 2 feature data with test point

Score	MLP_grid	BO	kernel	SVM_RBF	Poly
Cross-val	0.97	0.98	0.98	0.97	0.98
Test set	0.93	0.95	0.985	NA	NA
F1 score M	0.95	0.97	0.98	0.98	0.98
Precision M	0.93	1	1	1	1
Recall M	0.99	0.94	0.95	0.95	0.95
F1 score B	0.97	0.98	0.99	0.99	0.99
Precision B	0.98	0.97	0.97	0.97	0.97
Recall B	0.96	1	1	1	1

classification report is nearly the same. All the models and their performance is compared in the table above.

5.2.1 RBF kernel. With grid search, the best C parameter is "1" and the best gamma is "scale". The gamma parameters are the inverse of the radius of influence of samples chosen as support vectors by the model. The C parameter compromises between correctly classifying training samples and maximizing the margin of the decision function. It performed pretty well with the best cross-validation score of 0.97.

```
              precision    recall  f1-score   support

           B       0.97      1.00      0.99        71
           M       1.00      0.95      0.98        43

    accuracy                           0.98       114
   macro avg       0.99      0.98      0.98       114
weighted avg       0.98      0.98      0.98       114
```

5.2.2 Polynomial kernel. With grid search, the best C parameter is "0.1", the best coef0 is "2", the best degree is "2" (quadratic) and the best gamma value is "0.1". coef0 adjusts the independent term in the kernel function. It performed better than the RBF kernel with the best cross-validation score of 0.98. The classification report for both kernels is the same.

5.3 K-Nearest Neighbors

The correctness of the KNN algorithm was determined using the *implementation_correctness_dataset.csv* file provided. KNN was performed using a test datapoint of [1.4, 3]. The dataset, which had two classes, was plotted on a scatterplot with different colors for each class. Then, the test datapoint was plotted in red. Using

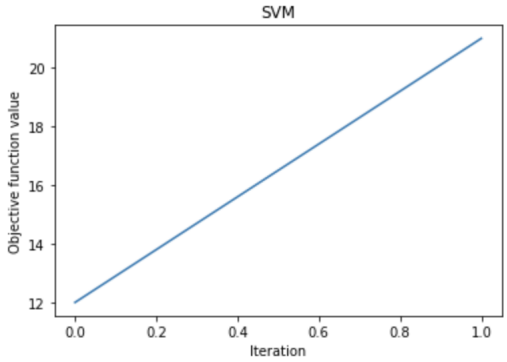


Figure 28: Linear SVM kernel not converging

the Euclidean heuristic, the label of the datapoint was determined.

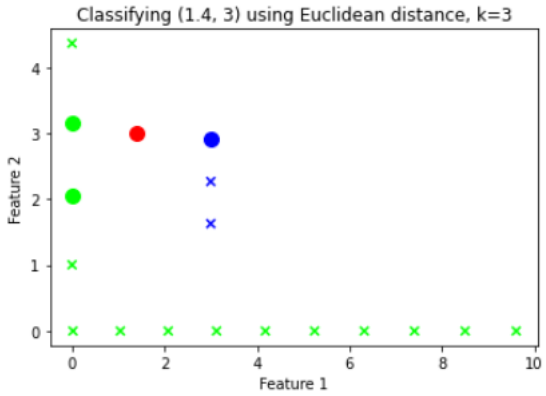


Figure 29: (1.4, 3) being identified as Class 1

As shown in Figure 29, when using the Euclidean distance metric and 3 neighbors, the test datapoint is identified as Class 1 as it is close to 2 points from Class 1 whereas it's only close to 1 point in Class 2.

When we run KNN on the test datapoint using the Manhattan distance function instead, the results are different.

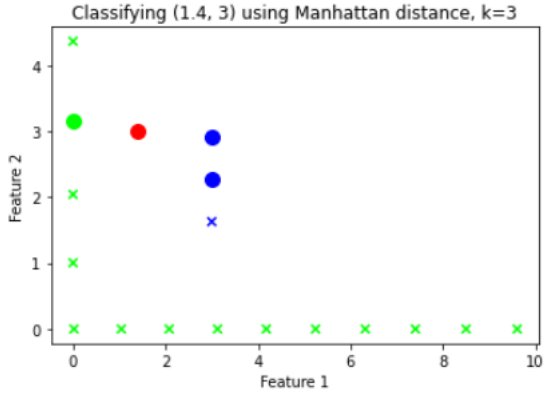


Figure 30: (1.4, 3) being identified as Class 2

As shown in Figure 30, when using Manhattan distance calculations, the test datapoint is identified as Class 2. This is because in the Manhattan space it is closer to two of the points in Class 2, as opposed to 1 point in Class 1.

5.4 DBSCAN

DBSCAN correctness is determined using the *implementation_correctness_dataset.csv* file provided. DBSCAN was proven correct and as shown by Figure 21, exhibits proper behavior. The red points are properly denoted as cluster 1 and the green points are properly denoted by cluster 2. The correctness is determined by the given(actual) labels of the clusters, but the graph is representative of our "from scratch" implementation.

In addition, the filled circles that represent the core points(which are the chosen points whose neighborhood exceeds *min_pts*) appear to be correct too. Each of the core points has at least 2 people within an ϵ distance of 1.5. This matches the definition of a core point in DBSCAN which is a point whose neighborhood exceeds *min_pts*

On top of this, the order in which we traverse the graph follows the DBSCAN algorithm. The DBSCAN algorithm stays within one cluster and slowly expands until it is unable to do so anymore. Then it moves on to another cluster at a random point that is unvisited. That is when we see that the green points(cluster 2) are visited after we finish the edge points of cluster 1. Note that the core point of cluster 2 is what is visited first as that is the first point that we can label as part of cluster 2. We are unable to use the other two points as code points as they don't satisfy the criterion.

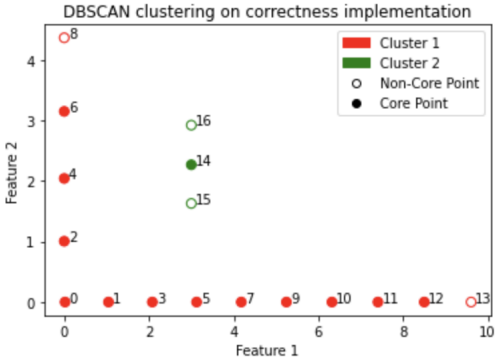


Figure 31: DBSCAN Performance

As shown in figure 31, DBSCAN accurately clusters the two groups. The figure also shows that the from scratch DBSCAN algorithm correctly identifies core points and can correctly traverse points in a reasonable order

5.5 Spectral Clustering

Spectral Clustering correctness is determined using the *implementation_correctness_dataset.csv* file provided. Spectral Clustering was proven correct and as shown by the graph below, exhibits proper behavior. The red points are properly denoted as cluster 1 and the blue points are properly denoted by cluster 2.

The algorithm accurately performs on the dataset by representing the similarity graph as a matrix and then finds the eigenvectors and eigenvalues of that matrix. The eigenvectors are then used

to project the data points into a lower dimensional space, where it is clustered using K-means. The k-means algorithm accurately clusters Cluster 1 and Clusters 2 and hence has proven to be correct

```
array([[0.02       , 0.        ],
       [1.045      , 0.        ],
       [2.08       , 0.        ],
       [3.125      , 0.        ],
       [4.18       , 0.        ],
       [5.245      , 0.        ],
       [6.32       , 0.        ],
       [7.405      , 0.        ],
       [8.5        , 0.        ],
       [9.605      , 0.        ],
       [0.         , 1.005     ],
       [0.         , 2.04287094],
       [0.         , 3.15067663],
       [0.         , 4.36758347],
       [3.         , 1.62946271],
       [3.         , 2.26894388],
       [3.         , 2.92310397]])
```

Figure 32: Spectral Embedding

As shown in the figure above, it is spectral embeddings of dimension 2 based on the Normalized Laplacian

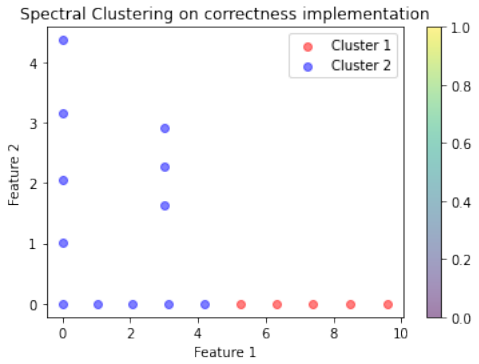


Figure 33: Spectral Clustering Performance

As shown in the figure above, Spectral Clustering accurately clusters the two groups. The figure also shows that the from scratch Spectral Clustering algorithm correctly identifies clusters.

5.6 Agglomerative Clustering

Our model's implementation correctness is validated through the use of the implementation correctness data set that was provided. As shown in the scatter plot below, our model correctly labeled each point as its own cluster at the start of the clustering algorithm. Eventually, each point is either clustered into one of two clusters that remain at the end of the algorithm.

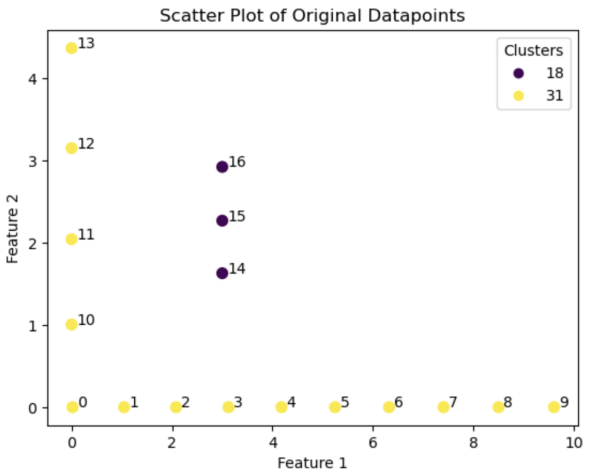


Figure 34: Cluster Scatter Plot

In addition, the dendrogram in the figure below displays how the individual clusters were grouped and merged over the iterations of the algorithm. If we compare these clusters to the ground truth clusters, we can see that they are nearly identical in grouping. This conclusion is supported by the dendrogram clustering points 14, 15, and 16 together and all the other points into another cluster exactly how the ground truth labels are in the dataset.

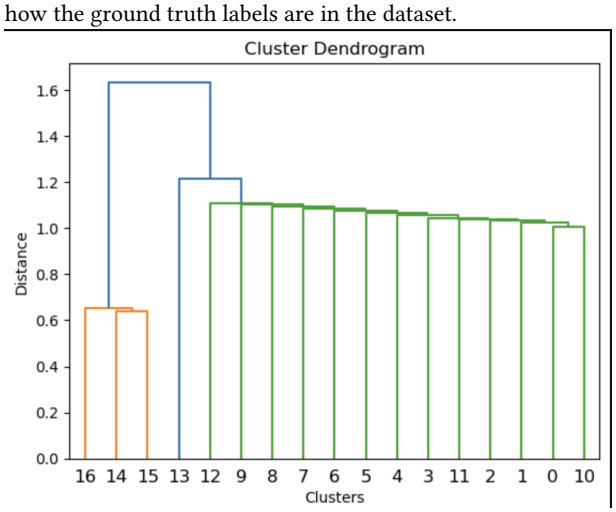


Figure 35: Clustering Dendrogram

BIBLIOGRAPHY

[1] Emeritus. Artificial intelligence and machine learning: Classification in machine learning. https://emeritus.org/blog/artificial-intelligence-and-machine-learning-classification-in-machine-learning/, February 2 2022.
[2] Stanford University. Clustering. https://web.stanford.edu/class/cs102/lectureslides/ClusteringSlides.pdf.
[3] Harshit Saini. Dbscan clustering algorithm implementation from scratch in python. https://becominghuman.ai/dbscan-clustering-algorithm-implementation-from-scratch-python-9950af5eed97, January 4 2019.
[4] Evangelos Papalexakis. $07b_u nsupervised, Februar y2023$.
[5] Ivan Kostic, Niharika Deo, and Bill Wang. Hierarchical clustering. https://www.cs.princeton.edu/courses/archive/spring19/cos324/files/hierarchical\protect\discretionary{\char\hyphenchar\font}{}{}clustering.pdf, April 4 2019.
[6] NumPy. numpy.linalg.svd. https://numpy.org/doc/stable/reference/generated/numpy.linalg.svd.html.

[7] NumPy. numpy.linalg.norm. https://numpy.org/doc/stable/reference/generated/numpy.linalg.norm.html.