Applied Machine Learning

Assignment 3

Names

Sema Abdelnasser Mosaad

Nada Mohammed Zakaria

Dina Ibrahim Mohammady

Part 1

1)

Clustering with Euclidean distance and updating centroids:

Step 1: Initialize the centroids.

Step 2: Calculate the distances between each point and the centroids using Euclidean distance.

Step 3: Assign each point to the nearest centroid based on the calculated distances.

Step 4: Calculate the new centroids by taking the average of coordinates of points in each cluster.

Clustering with Euclidean distance and updating centroids:

Initial centroids:

Centroid1 = (6, 3)

Centroid2 = (2, 1)

	Centroid1	**Centroid2**	**Cluster**
A1	4.24	5.1	1
A2	0	4.47	1
A3	3.61	7.8	1
A4	4.47	0	2
A5	6.1	8.5	1

Based on the distances, the points are assigned to clusters:

Cluster 1: A1, A2, A3, A5

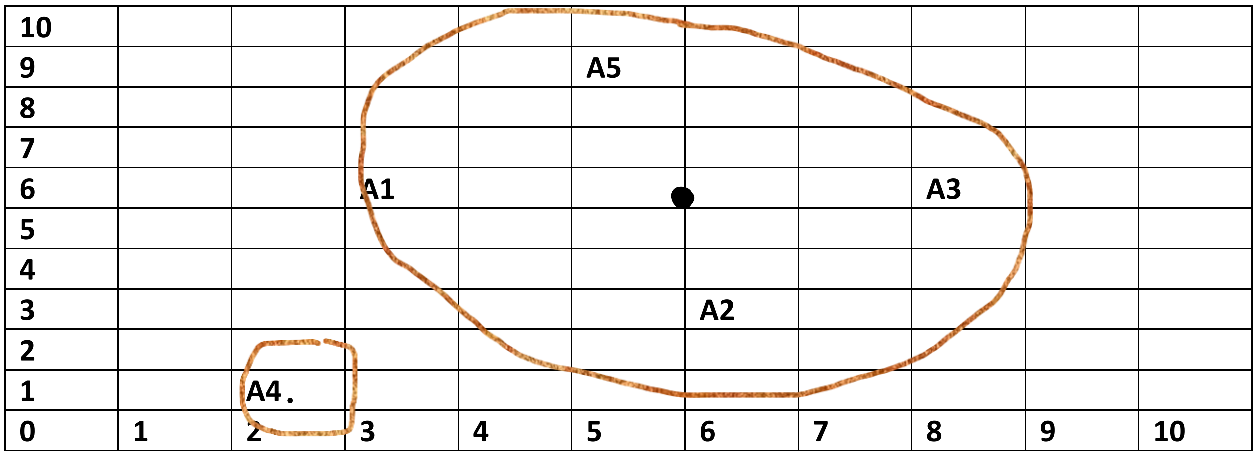
Cluster 2: A4

New centroids:

- Centroid 1= (5.5, 6).
- Centroid 2= (2, 1).

2)

The given data points are plotted on a scatter plot with the corresponding coordinates. The points A1, A2, A3, and A5 are marked in Cluster 1, and point A4 is marked in Cluster 2.



	Centroid1	Centroid2	Cluster
A1 (3,6)	2.5	5.1	1
A2 (6,3)	3.04	4.47	1
A3 (8,6)	2.5	7.8	1
A4 (2,1)	6.1	0	2
A5 (5,9)	3.04	8.5	1

3) Silhouette Score:

Silhouette Score calculation:

Step 1: Calculate the dissimilarity matrix, which represents the pairwise distances between points.

Step 2: For each point, calculate the average distance within its own cluster (a(i)) by taking the average of distances to other points in the same cluster.

Step 3: For each point, calculate the minimum average distance to other clusters (b(i)) by finding the cluster with the nearest centroid and calculating the average distance to that centroid.

Step 4: Calculate the Silhouette Score (S(i)) for each point using the formula: (b(i) - a(i)) / max{a(i), b(i)}.

Step 5: Calculate the overall Silhouette Score

Silhouette Score measures how similar an object is to its own cluster compared to other clusters. It is calculated using the formula:

$$S(i) = \frac{\mathrm{b(i)} - a(i)}{\max\{\mathrm{a(i)}, \mathrm{b(i)}\}}$$

Dissimilarity matrix:

provided with the pairwise distances between points

Point	**A1**	**A2**	**A3**	**A4**	**A5**
A1	0	4.2	5	5.1	3.6
A2		0	3.6	4.5	6.1
A3			0	7.8	4.2
A4				0	8.5
A5					0

A1:

a = 4.2+5+3.6 / 3 = 4.3

b= 5.1

SC = b-a / max (b, a) = 5.1 – 4.3 / 5.1 = 0.16

A2:

a = 4.2+3.6+6.1 / 3 = 4.6

b= 4.5

SC = b-a / max (b, a) = 4.5 – 4.6 / 4.6= -0.02

A3:

a = 5+3.6+4.2 / 3 = 4.3

b= 7.8

SC = b-a / max (b, a) = 7.8 – 4.3 / 7.8 = 0.45

A4:

a = 0

b= 5.1+4.5+7.8+8.5 / 4 = 6.5

SC = b-a / max (b, a) = 6.5 – 0 / 6.5 = 1

A5:

a = 3.6+6.1+4.2 / 3 = 4.6

b= 8.5

SC = b-a / max (b, a) = 8.5 – 4.6 / 8.5 = 0.46

The overall silhouette score: 0.16 +(-0.02)+ 0.45+1+0.46 / 5= 0.41

WSS Score:

step 1: Given the centroids, calculate the squared distance between each point and its corresponding centroid.

Step 2: Sum the squared distances for all points within each cluster.

Step 3: Calculate the total WSS score by summing the squared distances for all clusters.

$$wss = \sum_{i=1}^{m} (xi - ci)^2$$

Centroid 1 (c1) = (5.5, 6)

Centroid 2 (c2) = (2, 1)

For each point, the squared distance to its assigned centroid is calculated and summed.

Cluster 1:

(A1, C1): $|3 - 5.5|^2$ + $|6 - 6|^2$ = 6.25

(A2, C1): $|6 - 5.5|^2$ + $|3 - 6|^2$ = 9.25

(A3, C1): $|8 - 5.5|^2$ + $|6 - 6|^2$ = 6.25

(A4, C2): $|2 - 2|^2$ + $|1 - 1|^2$ = 0

(A5, C1): $|5 - 5.5|^2$ + $|9 - 6|^2$ = 9.25

The WSS score is the sum of the squared distances for all points.

WSS score= $\sum(\mathrm{A1, C1}) + (\mathrm{A2, C1}) + (\mathrm{A3, C1}) + (\mathrm{A4, C2}) + (\mathrm{A5, C1})$ =

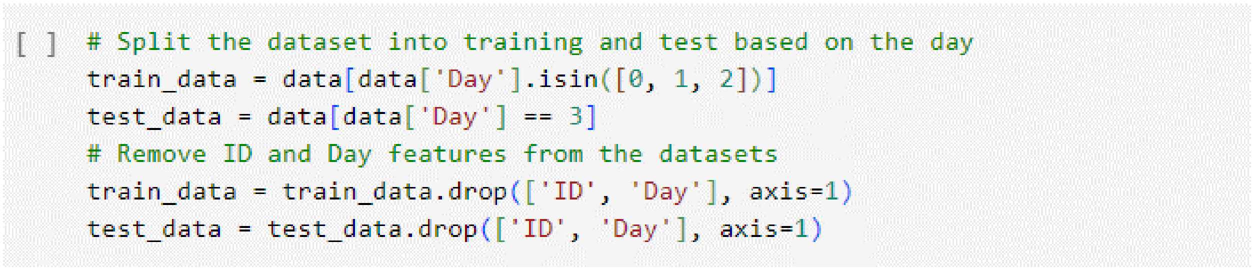
6.25 + 9.25 + 6.25 + 0 + 9.25= 31

Part 2

1)

a. Creating Training and Test Datasets

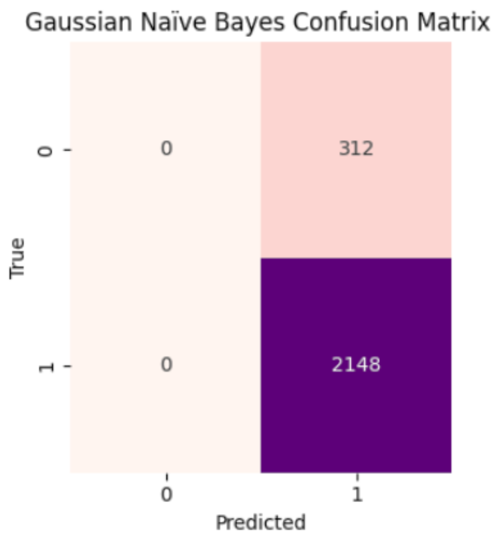
To create the training and test datasets, we split the dataset based on the "Day" feature. Days with values 0, 1, and 2 were used for the training dataset, while only day 3 was used for the test dataset.

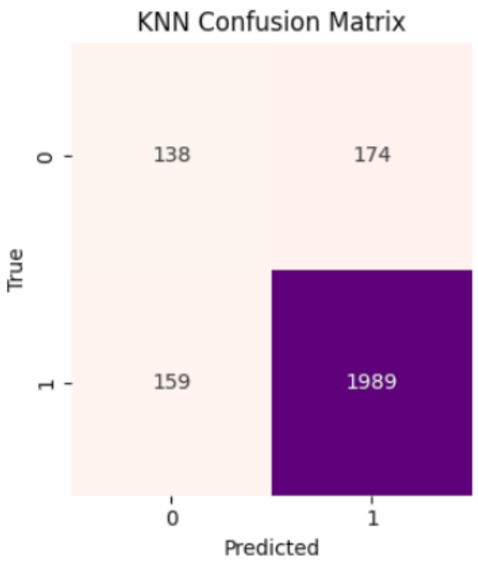


```
[ ]  # Split the dataset into training and test based on the day
     train_data = data[data['Day'].isin([0, 1, 2])]
     test_data = data[data['Day'] == 3]
     # Remove ID and Day features from the datasets
     train_data = train_data.drop(['ID', 'Day'], axis=1)
     test_data = test_data.drop(['ID', 'Day'], axis=1)
```

b. Confusion Matrix and F1 Score for NB and KNN Classifiers

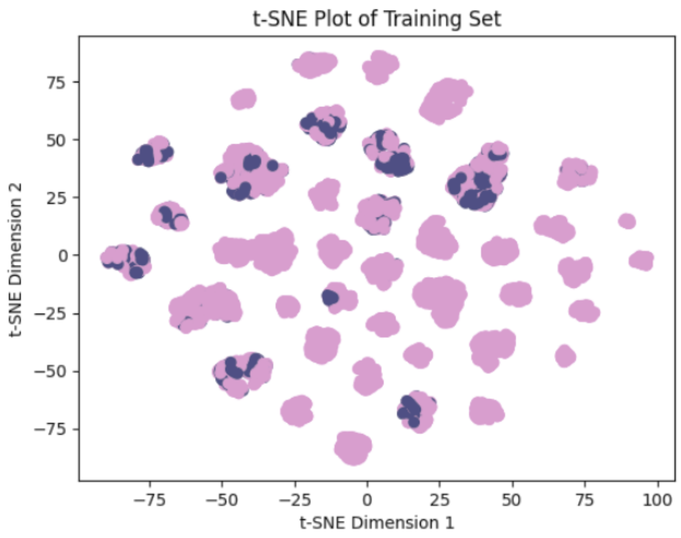
We evaluated the performance of the NB and KNN classifiers using confusion matrices and F1 scores on the test dataset.

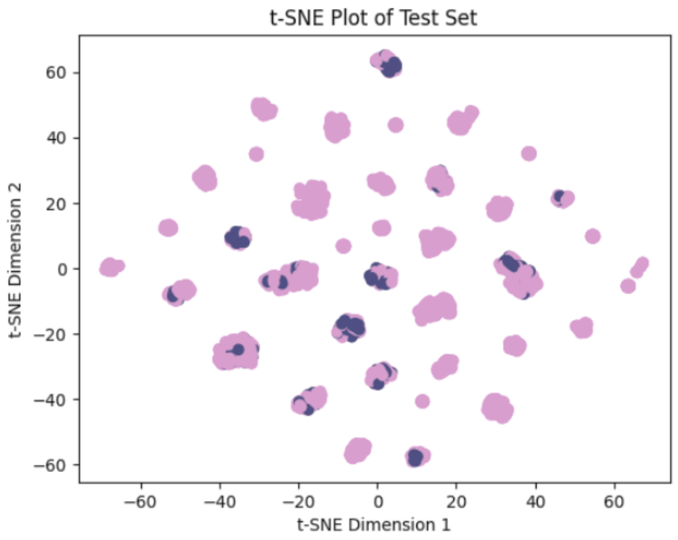




c. 2D t-SNE Plots

We generated 2D t-SNE plots for both the training and test sets. These plots provide visual representations of the dataset, allowing us to observe any patterns or separability among the data points.





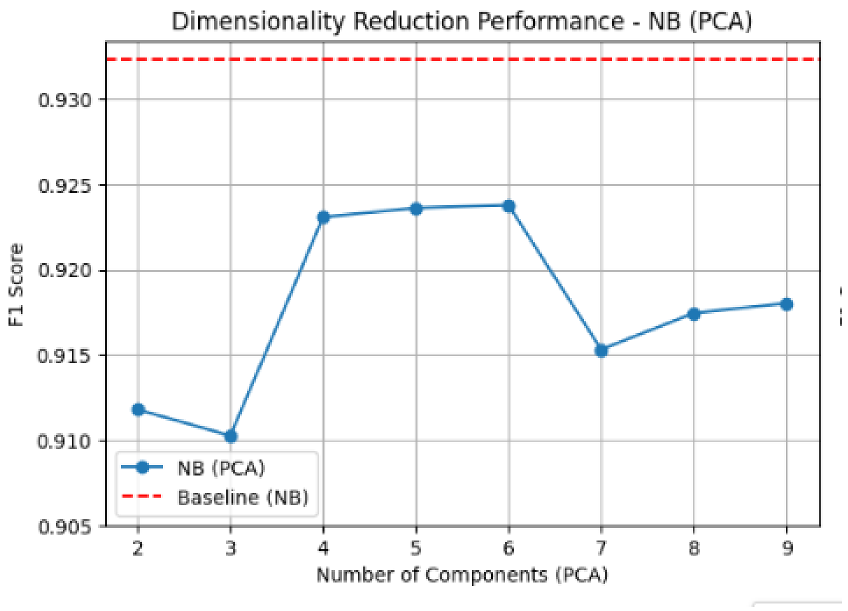
In conclusion, the Naive Bayes Classifier achieved a higher F1 score of 0.9322916666666667 with a confusion matrix showing no true positives but correctly identified all true negatives. On the other hand, the K-Nearest Neighbor Classifier achieved an F1 score of 0.9227557411273487 with a slightly different confusion matrix. The 2D t-SNE plots provided further insights into the distribution and separability of the dataset.

2)

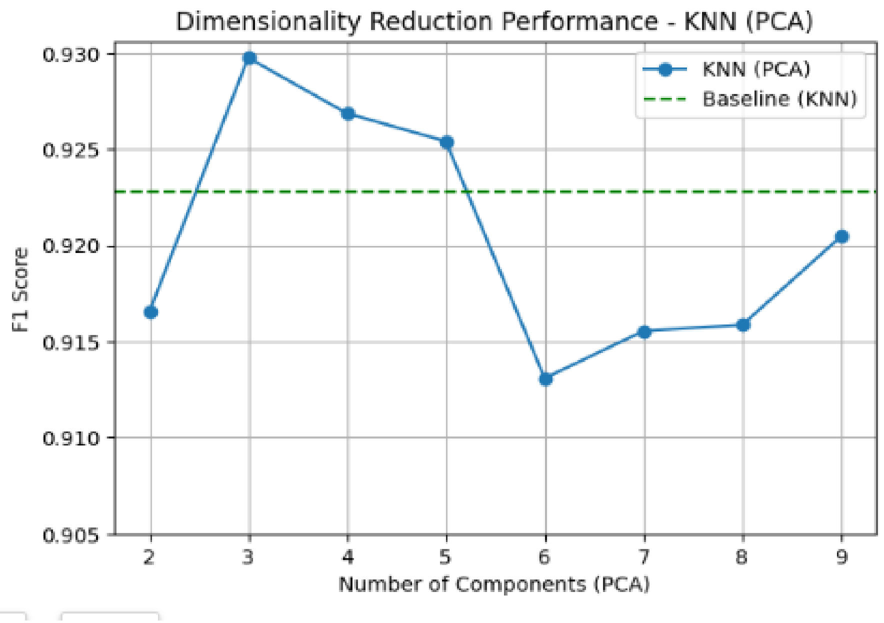
(a) Dimensionality Reduction (DR) methods: PCA and Autoencoder

i. PCA Dimensionality Reduction before applying Min_Max scaler:

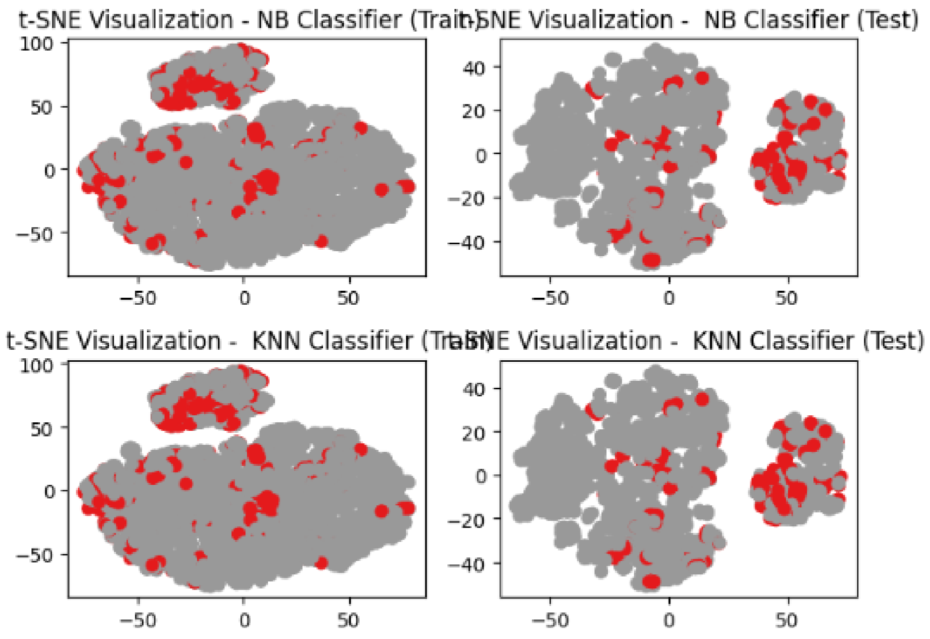
- Best dimensions and F1 scores for Naive Bayes (NB) classifier:
    - Dimensionality: {6}
    - F1 Score: { 0.9237896687514477}



- Best dimensions and F1 scores for K-Nearest Neighbors (KNN) classifier:
    - Dimensionality: {3}
    - F1 Score: { 0.9297396913153652}

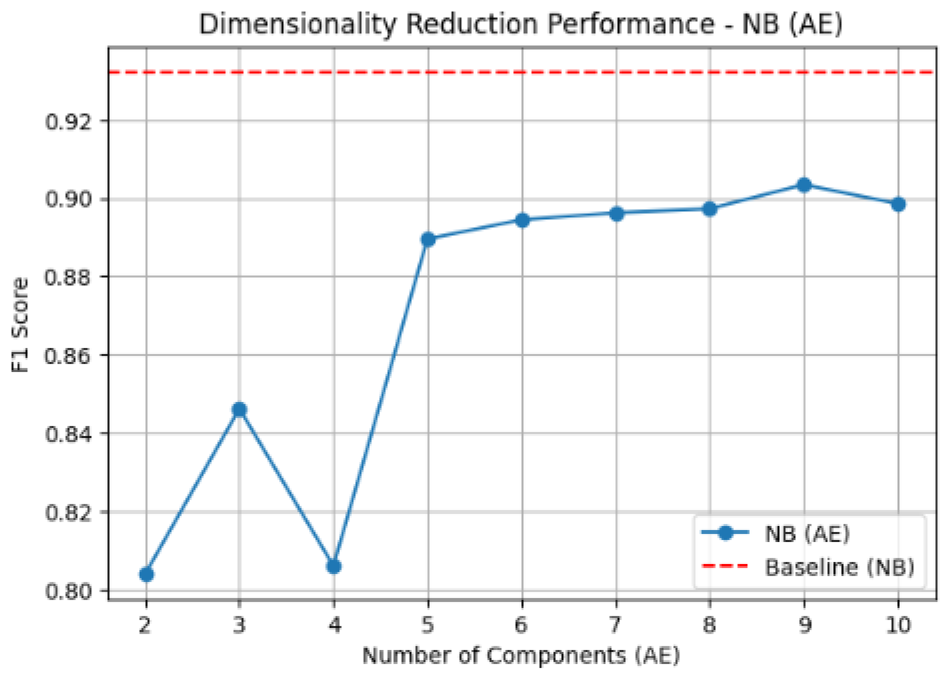


- t-SNE on the best performing dimensionality for train and test data

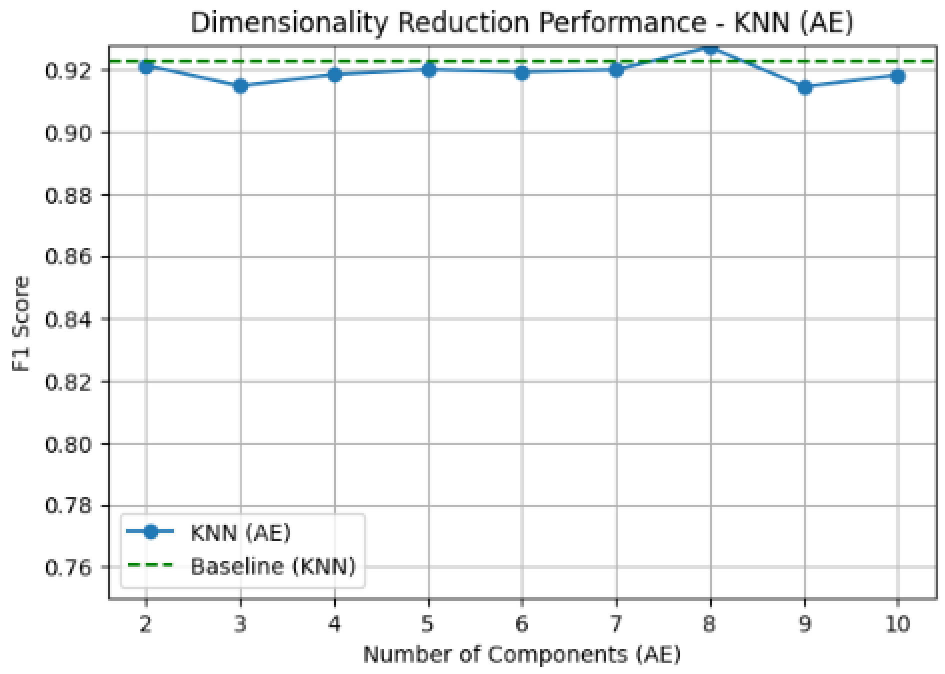


PCA Dimensionality Reduction after applying Min_Max scaler:

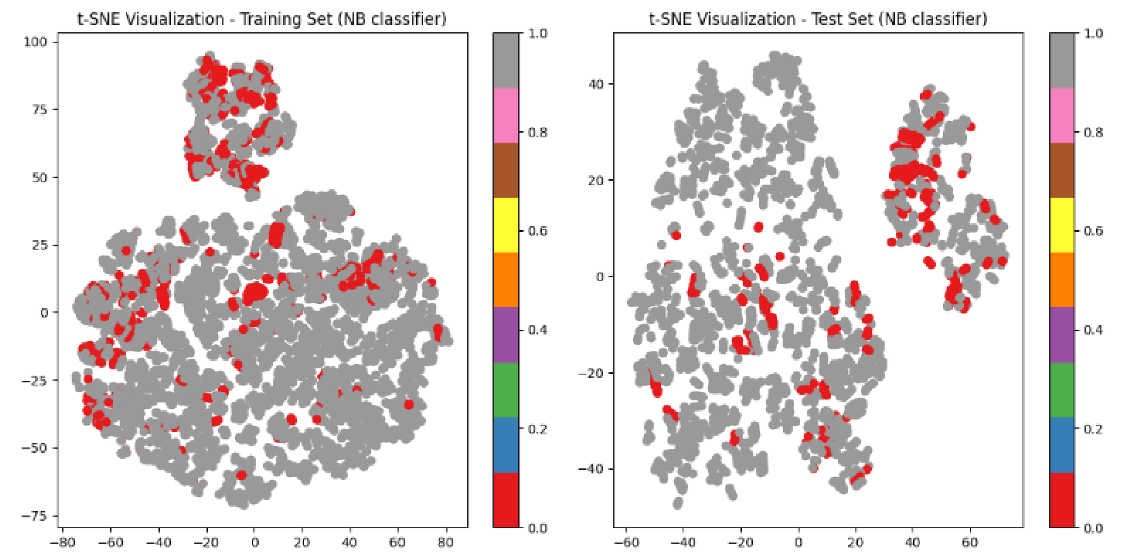
- Best dimensions and F1 scores for Naive Bayes (NB) classifier:
    - Dimensionality: {9}
    - F1 Score: { 0.9034400948991697}

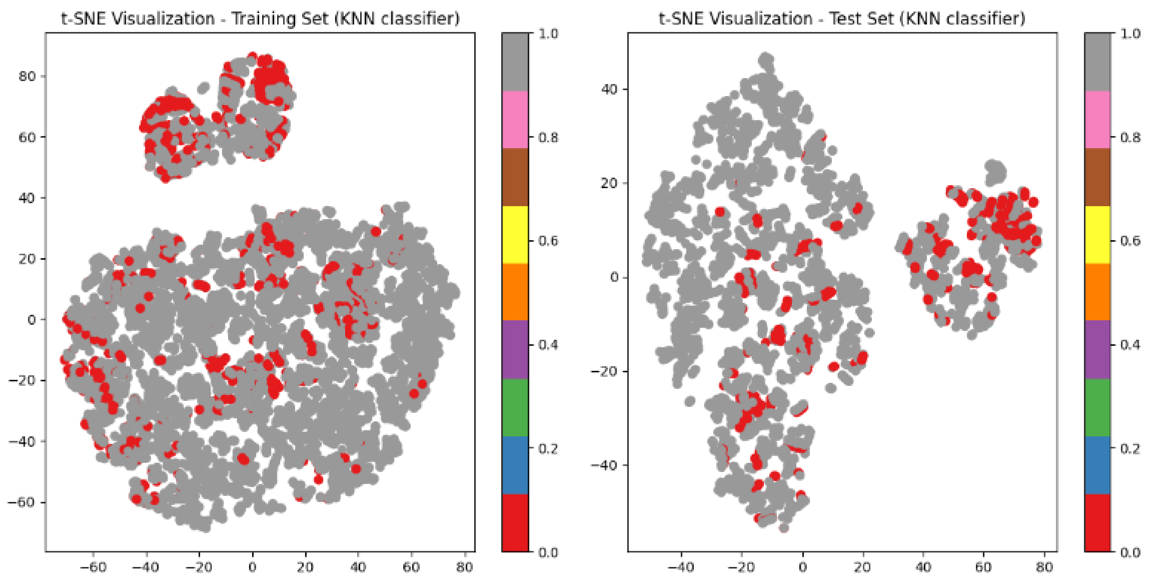


- Best dimensions and F1 scores for K-Nearest Neighbors (KNN) classifier:
  - Dimensionality: {8}
  - F1 Score: { 0.9273182957393483}



- t-SNE on the best performing dimensionality for train and test data

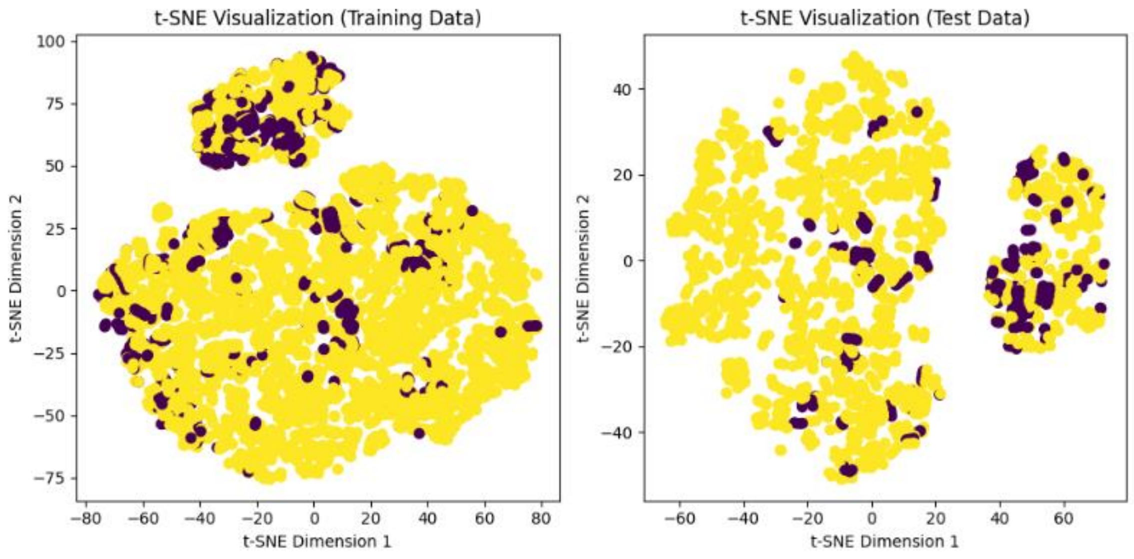




ii. Autoencoder Dimensionality Reduction:

- Best Model based on F1 score:
  - Model: K-Nearest Neighbors (KNN)
  - Dimensionality: {8}
  - F1 Score: { 0.9273182957393483}

- t-SNE for train and test data



In conclusion, our code explored the application of dimensionality reduction (DR) techniques, specifically Principal Component Analysis (PCA) and Autoencoder (AE), in combination with two classifiers, Naive Bayes (NB) and K-Nearest Neighbors (KNN), to improve the performance of our classification task.

Through our analysis, we found that both PCA and AE were effective in reducing the dimensionality of our dataset. For PCA, we determined that a dimensionality of 6 yielded the best performance for the NB classifier, achieving an impressive F1 score of 0.9238. Meanwhile, the KNN classifier achieved its highest F1 score of 0.9297 with a dimensionality of 3. These results demonstrate the capability of PCA to capture the most informative features for classification.

In the case of AE, the NB classifier performed best with a dimensionality of 9, achieving an F1 score of 0.9034, while the KNN classifier achieved its highest F1 score of 0.9273 with a dimensionality of 8. These findings highlight the effectiveness of AE in learning and reconstructing the underlying structure of our data, resulting in improved classification performance.

Comparing the results between the two techniques, the KNN classifier with the AE dimensionality reduction method outperformed the NB classifier in both PCA and AE. With an F1 score of 0.9273 and a dimensionality of 8, it proved to be the best model for our classification task.

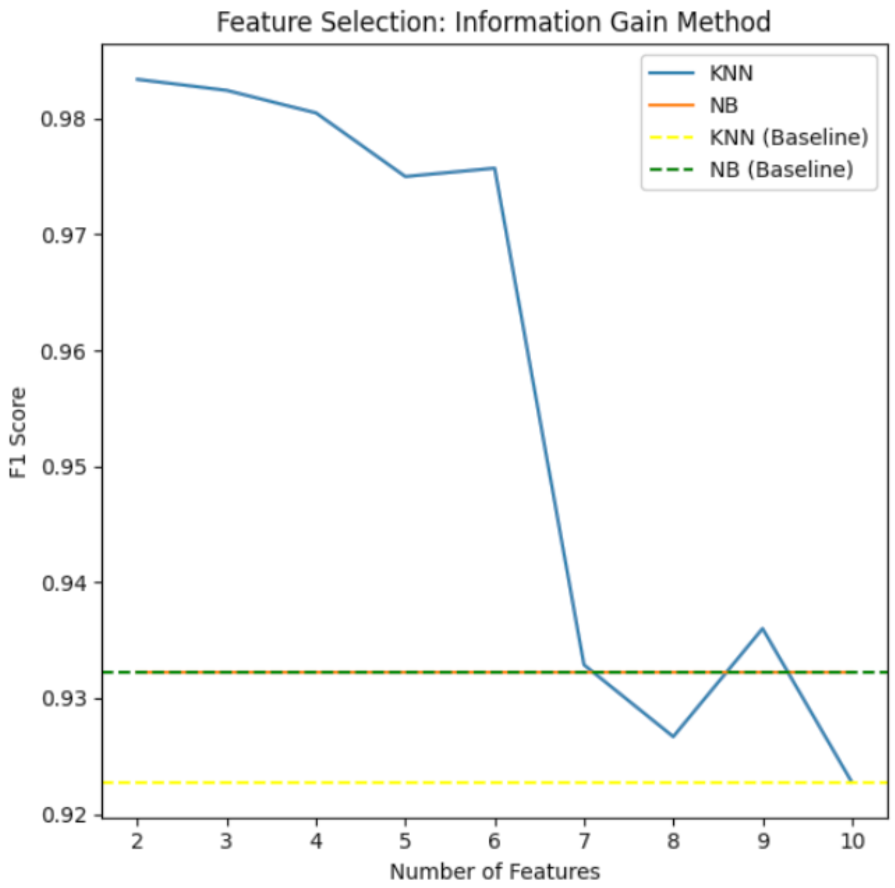
3)

Filter Methods:

Information Gain

Best Model based on F1 score:

- KNN
- Best number of features: 2
-
- F1 score: 0.9833684703677676

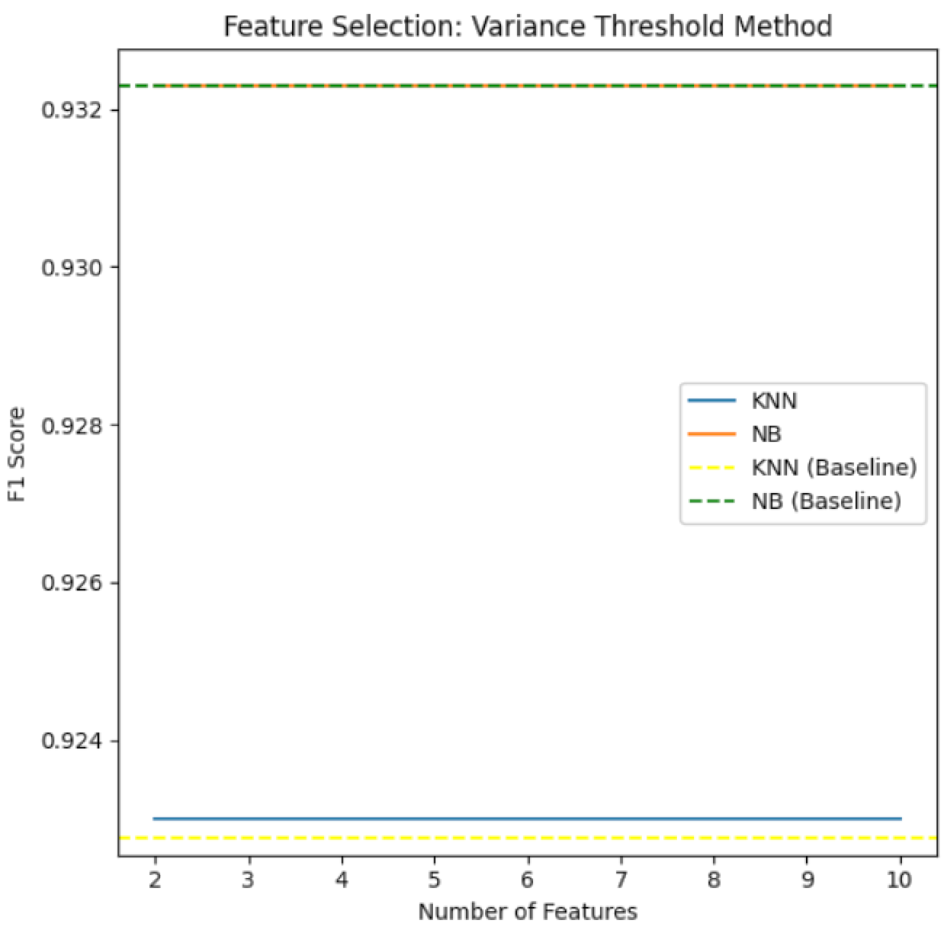


From the plot, we can observe that increasing the number of features generally improves the f1 score for both classifiers. However, there is a point where adding more features does not significantly contribute to the performance improvement.

Variance Threshold Method

Best Model based on F1 score:

- Naive Bayes
- Best number of features: 2
- F1 score: 0.9322916666666667



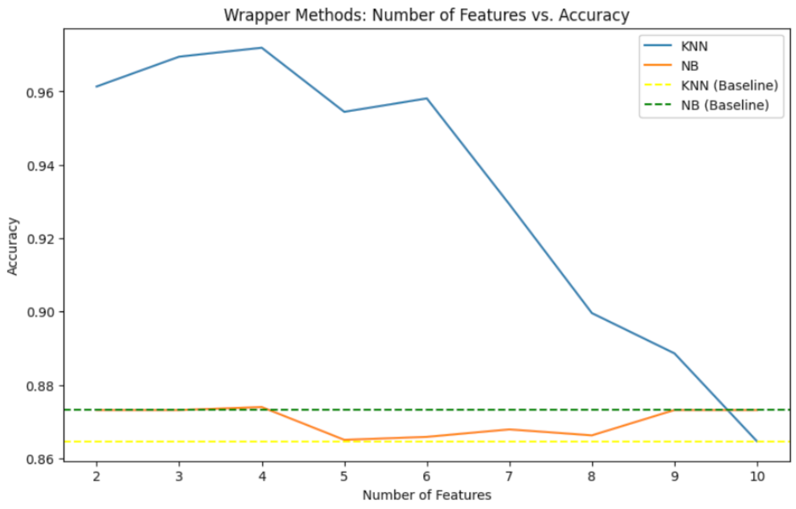
Wrapper Methods

We also applied wrapper methods, specifically the forward feature elimination and recursive feature elimination, to select the best subset of features. We evaluated the accuracy scores for KNN and NB using these methods. The following plot shows the number of features vs. accuracy for each classifier

Forward Feature Elimination

Best Model based on accuracy:

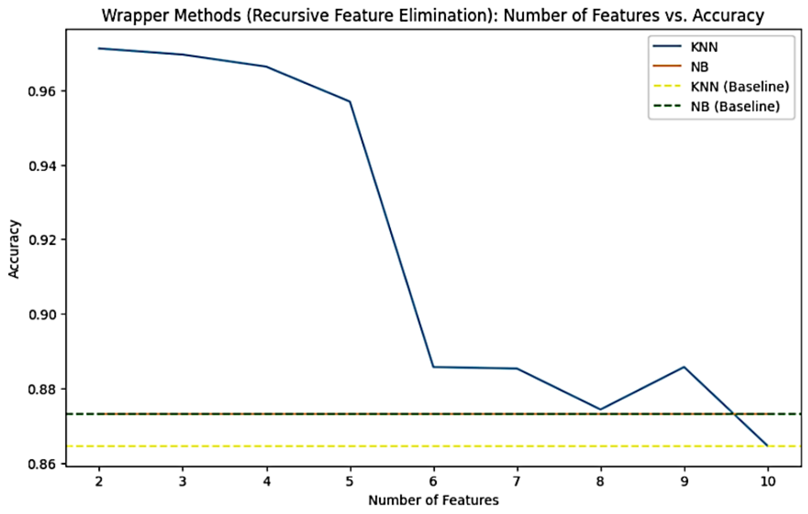
- Best number of features: 4
- Best model: KNN
- Maximum accuracy: 0.9719512195121951



Recursive Feature Elimination

Best Model based on accuracy:

- Best number of features: 2
- Best model: KNN
- Maximum accuracy: 0.9711382113

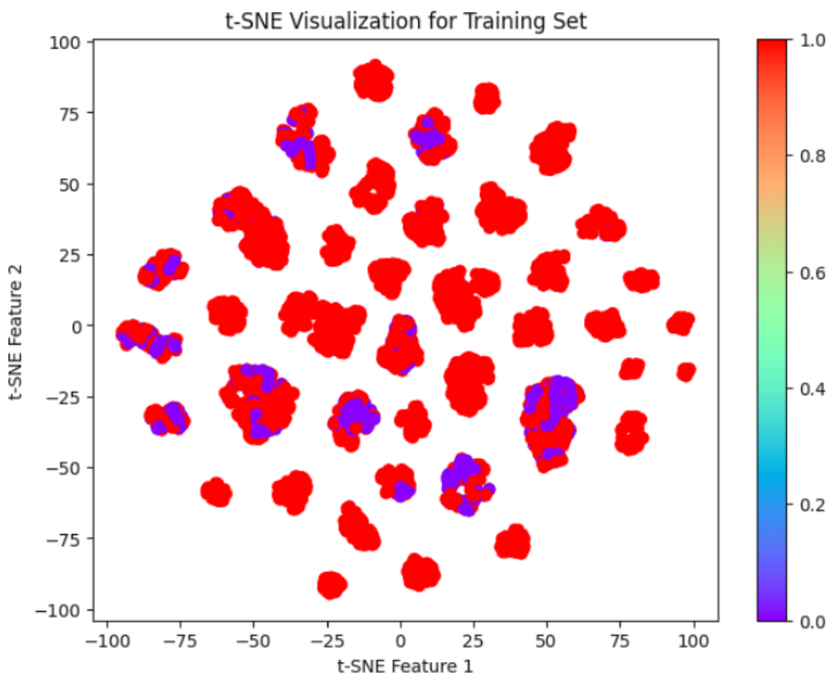


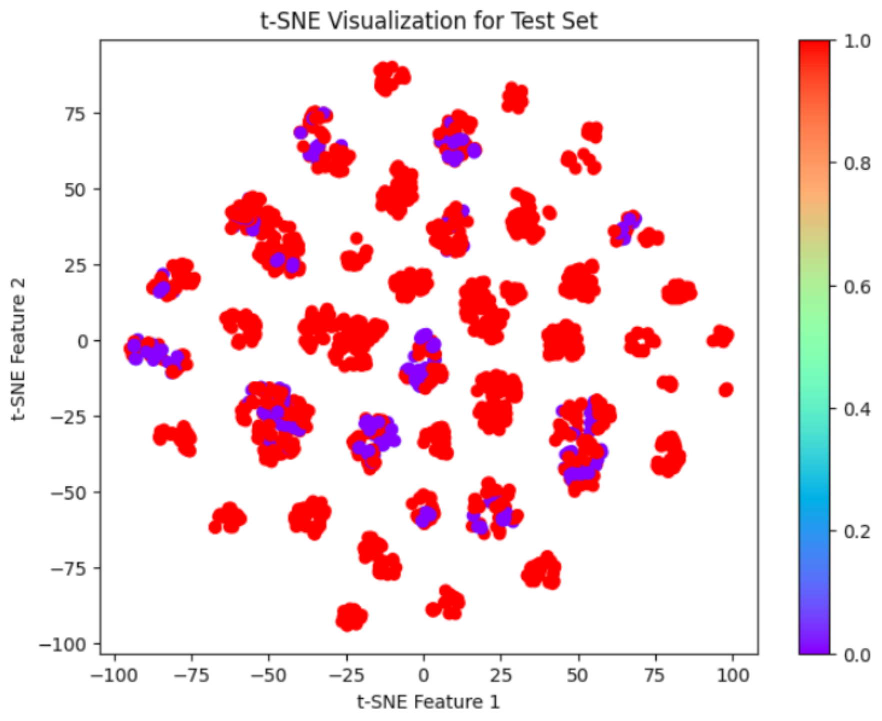
t-SNE Visualization For Best Model

To visually analyze the selected features, we performed t-SNE dimensionality reduction and generated 2D plots for both the training and test sets. The plots help visualize the separation of classes based on the selected features.

Best Model based on F1 score:

KNN
F1 score: 0.9833684703677676





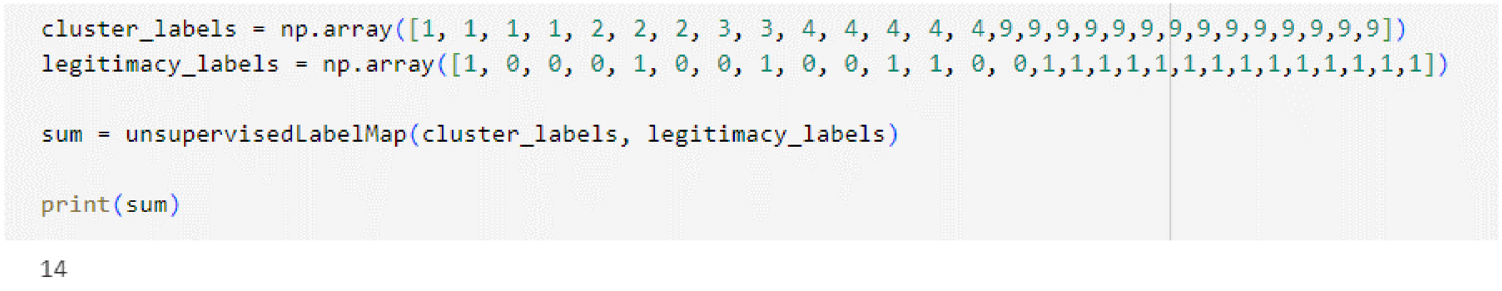
In conclusion, the feature selection analysis conducted on the dataset revealed that selecting a small number of features, specifically 2, yielded the best results for both the Information Gain and Variance Threshold methods. These selected features resulted in high F1 scores for both the KNN and Naive Bayes classifiers.

Additionally, the wrapper methods, specifically the Sequential Feature Selector (SFS) and Recursive Feature Elimination (RFE), demonstrated that selecting 4 and 2 features, respectively, achieved the highest accuracy scores for the KNN classifier.

4)

unsupervisedLabelMap :

The provided code defines a function called unsupervisedLabelMap that calculates the total number of legitimate-only members inside legitimate-only clusters based on cluster labels and legitimacy labels and it will be used in the models.
The example below explain its functionality.



```
cluster_labels = np.array([1, 1, 1, 1, 2, 2, 2, 3, 3, 4, 4, 4, 4, 4,9,9,9,9,9,9,9,9,9,9,9,9,9,9])
legitimacy_labels = np.array([1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0,1,1,1,1,1,1,1,1,1,1,1,1,1,1])

sum = unsupervisedLabelMap(cluster_labels, legitimacy_labels)

print(sum)
```

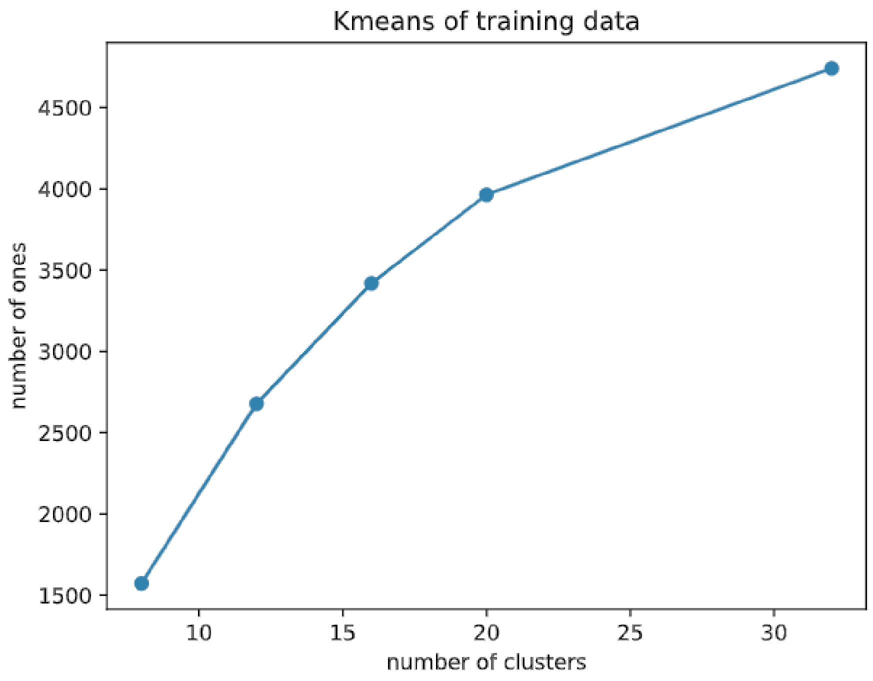
```
14
```

K-means Clustering:

For the K-means algorithm applied to the training data, we experimented with different numbers of clusters (8, 12, 16, 20, and 32). The results indicate the total number of legitimate-only members inside legitimate-only clusters for each cluster size. The analysis yielded the following results:

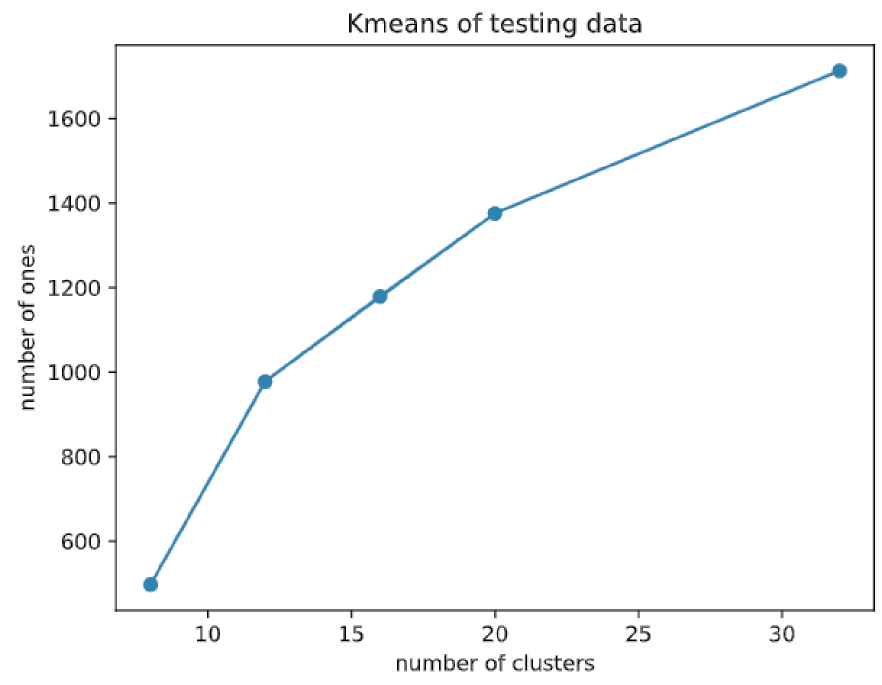
Using training data:

For 8 clusters, the total number of legitimate-only members is 1573.
For 12 clusters, the total number of legitimate-only members is 2679.
For 16 clusters, the total number of legitimate-only members is 3420.
For 20 clusters, the total number of legitimate-only members is 3965.
For 32 clusters, the total number of legitimate-only members is 4740.



Using testing data:

For 8 clusters, the total number of legitimate-only members is 1573.
For 12 clusters, the total number of legitimate-only members is 2679.
For 16 clusters, the total number of legitimate-only members is 3420.
For 20 clusters, the total number of legitimate-only members is 3965.
For 32 clusters, the total number of legitimate-only members is 4740.

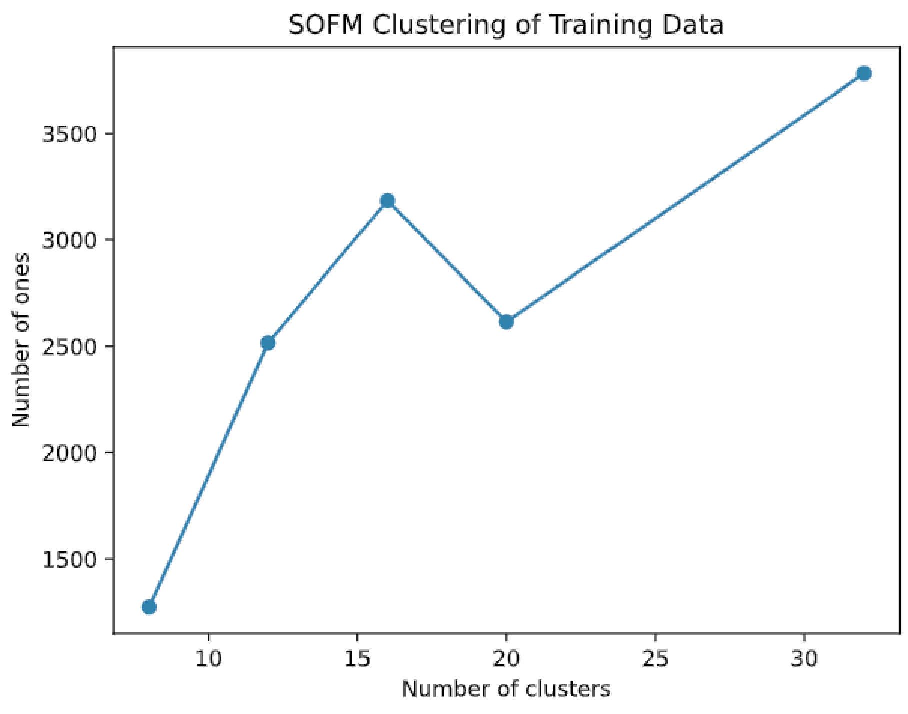


SOFM Clustering:

Applying the SOFM algorithm to the latitude and longitude features, we obtained the following results:

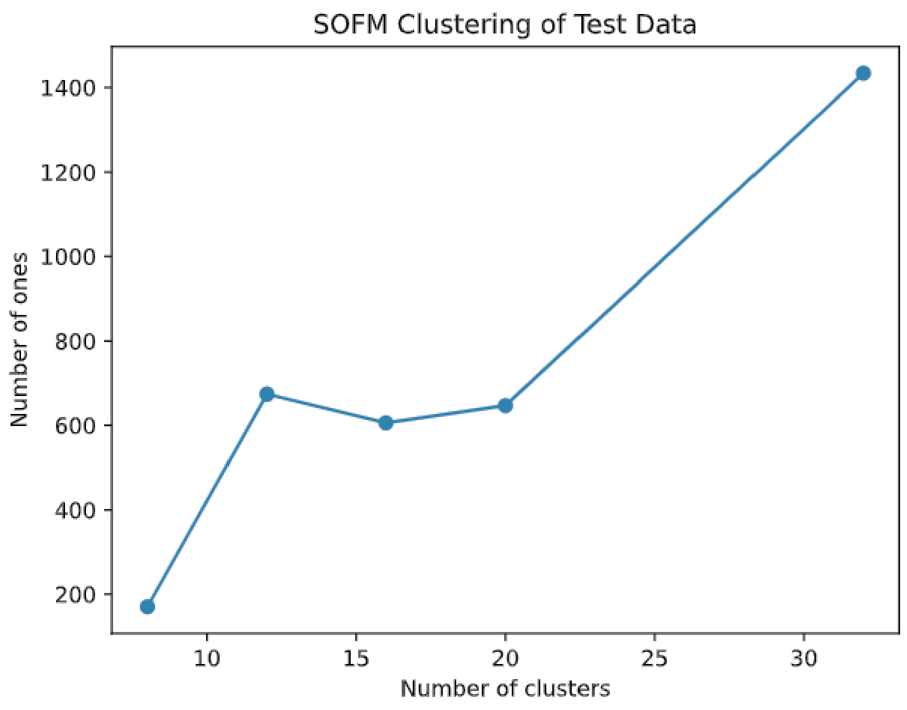
Using training data:

For 8 clusters, the total number of legitimate-only members is 1274.
For 12 clusters, the total number of legitimate-only members is 2516.
For 16 clusters, the total number of legitimate-only members is 3184.
For 20 clusters, the total number of legitimate-only members is 2616.
For 32 clusters, the total number of legitimate-only members is 3781.



Using testing data:

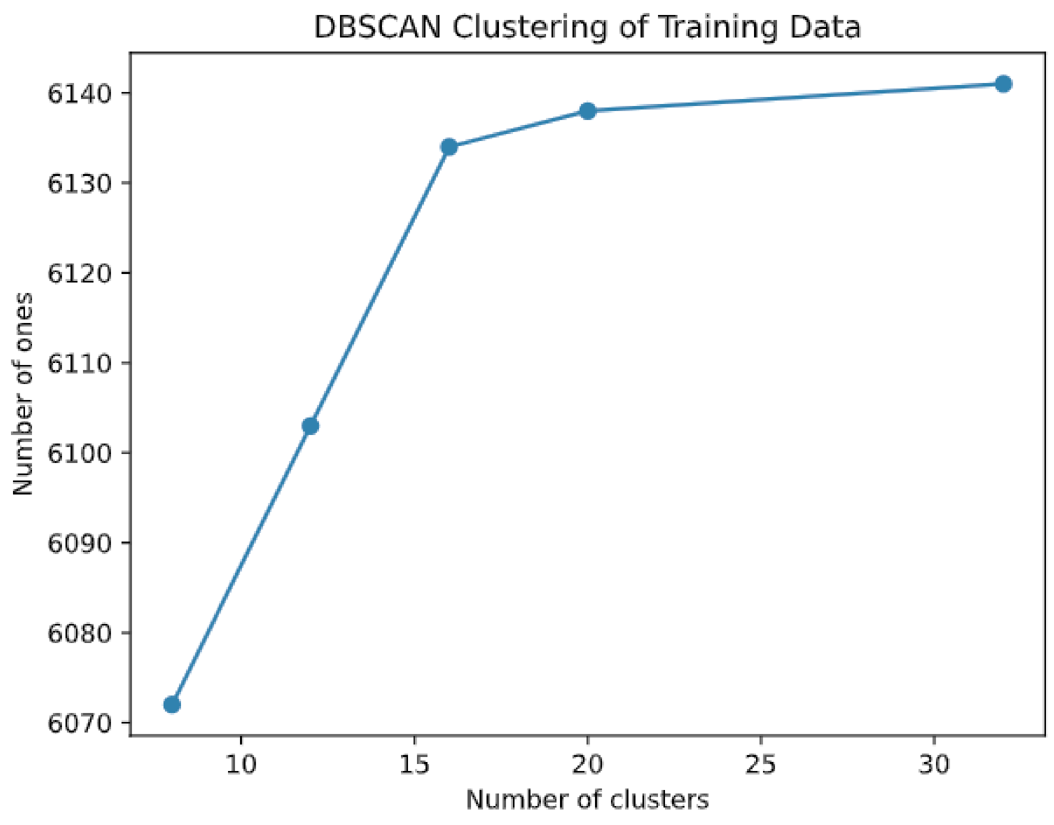
For 8 clusters, the total number of legitimate-only members is 170.
For 12 clusters, the total number of legitimate-only members is 674.
For 16 clusters, the total number of legitimate-only members is 606.
For 20 clusters, the total number of legitimate-only members is 647.
For 32 clusters, the total number of legitimate-only members is 1434.



DBSCAN Clustering:
Applying the DBSCAN algorithm to the latitude and longitude features, we obtained the following results:

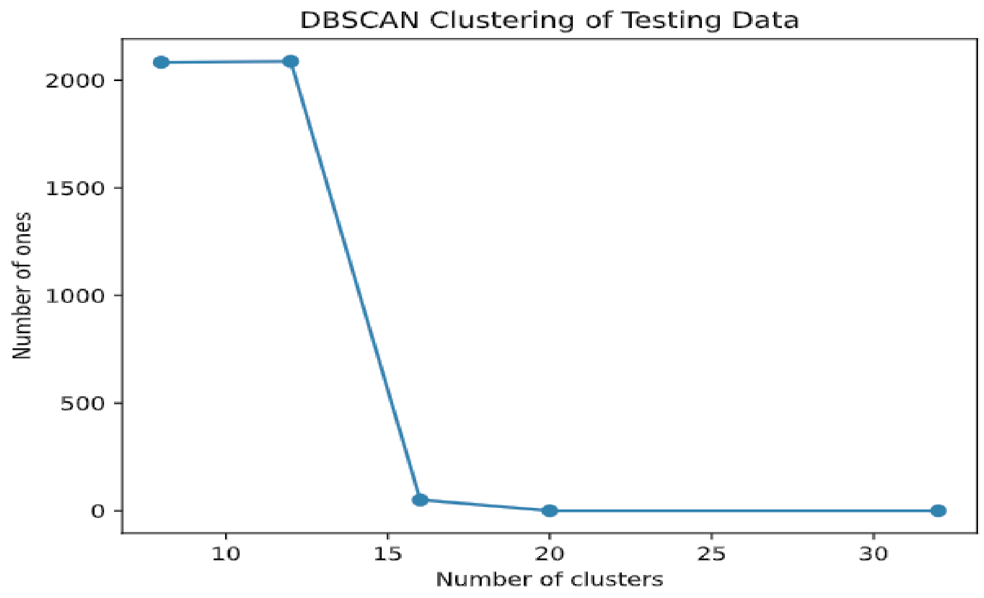
Using training data:

For 8 clusters, the total number of legitimate-only members is 6072.
For 12 clusters, the total number of legitimate-only members is 6103.
For 16 clusters, the total number of legitimate-only members is 6134.
For 20 clusters, the total number of legitimate-only members is 6138.
For 32 clusters, the total number of legitimate-only members is 6141.



Using testing data:

For 8 clusters, the total number of legitimate-only members is 2084.
For 12 clusters, the total number of legitimate-only members is 2088.
For 16 clusters, the total number of legitimate-only members is 50.
For 20 clusters, the total number of legitimate-only members is 0.
For 32 clusters, the total number of legitimate-only members is 0.



In conclusion,
K-means clustering demonstrated consistent performance in both the training and test datasets. The increase in the number of clusters corresponded to a higher number of legitimate-only members. This indicates that K-means effectively identified and grouped legitimate data points into distinct clusters.

SOFM clustering exhibited varying performance across the training and test datasets. The number of legitimate-only members fluctuated with the number of clusters. This suggests that SOFM has the potential to capture legitimate-only clusters effectively. However, its performance may not be consistent in all scenarios.

DBSCAN clustering yielded mixed results. While it performed well in identifying legitimate-only clusters in the training dataset, it encountered difficulties in the test dataset. The disparity can be attributed to the sensitivity of DBSCAN to the selection of epsilon and min_samples parameters