ASSINGMENT 3

Applied Machine Learning

Prepared by:

Ahmed Ahmed

Hadir Hassan

Donia Abdelreheem

Supervised by:

DR/Murat

PART 1

A) SHOW STEP-BY-STEP THE PERFORMED CALCULATIONS TO CLUSTER THE 5 POINTS.

1. Initialize the centroids

We are given that the initial centroids are A2=(6,3) and A4=(2,1).

2. Assign each point to the nearest centroid

We calculate the distance between each point and the two centroids using the Euclidean distance formula. The table below shows the calculations for one iteration and we assign each point to the nearest centroid, with cluster 1 being assigned to A2 and cluster 2 being assigned to A4.

Euclidean Distance Formula

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Data point	Euclidean distance to A2 (6,3) (Claster 1 (C1))	Euclidean distance to A4 (2,1) (Claster 2 (C2))	Cluster
A1 = (3,6)	4.242640687119285	5.291502622129189	Cluster 1
A2 = (6,3)	0	3.1622776601683795	Cluster 1
A3 = (8,6)	1.4142135623730951	3.1622776601683795	Cluster 1
A4 = (2,1)	5.291502622129189	0	Cluster 2
A5 =(5,9)	2.23606797749979	3.1622776601683795	Cluster 1

3. Calculate the new centroids

We calculate the mean of each cluster to get the new centroids.

Cluster 1 (C1): A1, A2, A3, A5

New centroid: ((3+6+8+5)/4, (3+6+6+9)/4) = (5.5, 6)

Cluster 2 (C2): A4

New centroid: (2, 1)

Cluster	Data Points	New Centroid
1	A1, A2, A3, A5	(5.5, 6)
2	A4	(2, 1)

.

B) DRAW A 10 BY 10 SPACE WITH ALL THE CLUSTERED 5 POINTS AND THE COORDINATES OF THE NEW CENTROIDS

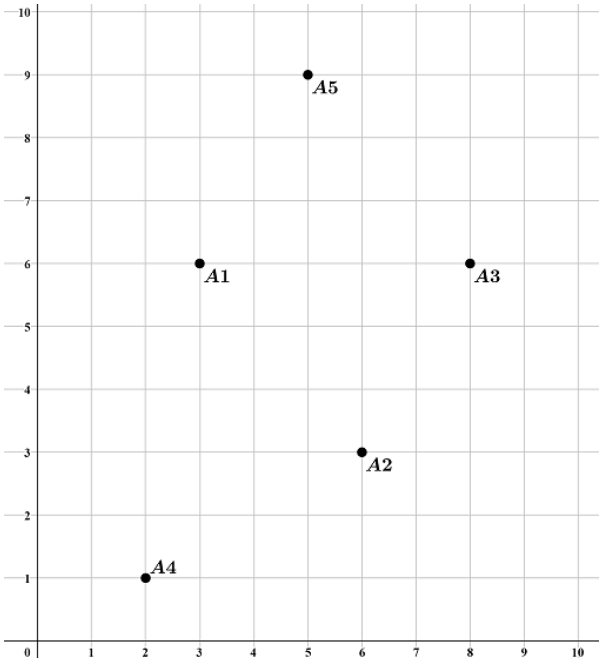


Figure 1: Data points

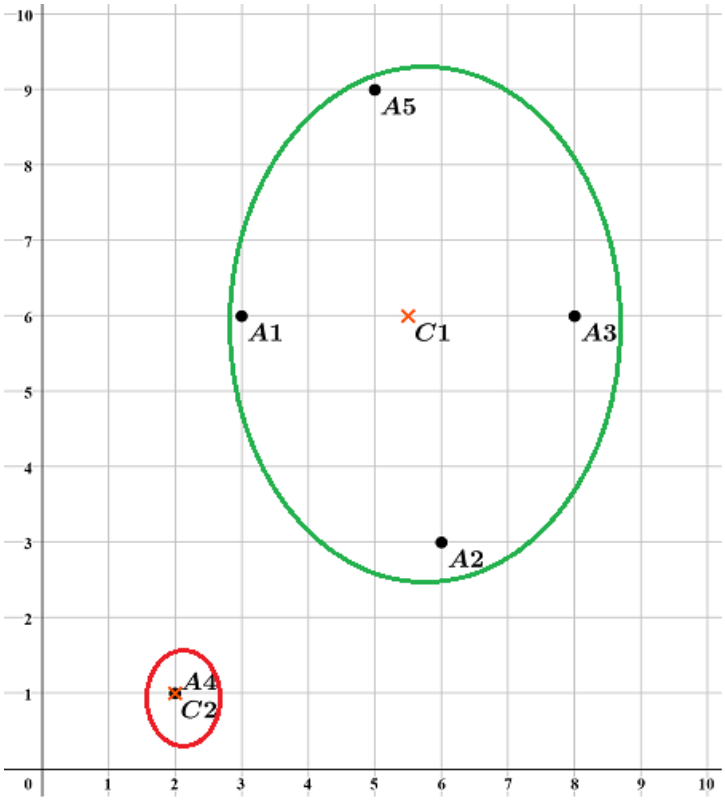


Figure 2: Clustered data points

C) CALCULATE THE SILHOUETTE SCORE AND WSS SCORE.

$$\textbf{Silhouette Score} = \frac{\mathbf{b(i)-a(i)}}{\mathbf{max[a(i),b(i)]}}$$

a(i)= the average distance between each point within a cluster.

b(i)= the average distance between all clusters.

Calculate Euclidean distance :

	A1=(3,6)	
A1=(3,6)	0	
A2=(6,3)	$\sqrt{(3-6)^2+(6-3)^2} = 4.24$	a(i)

A3=(8,6)	$\sqrt{(3-8)^2+(6-6)^2}=5$	a(i)
A4=(2,1)	$\sqrt{(3-2)^2+(6-1)^2}=5.099$	b(i)
A5=(5,9)	$\sqrt{(3-5)^2+(6-9)^2}=3.605$	a(i)

	A2=(6,3)	
A1=(3,6)	$\sqrt{(6-3)^2+(3-6)^2}=4.24$	a(i)
A2=(6,3)	0	
A3=(8,6)	$\sqrt{(6-8)^2+(3-6)^2}=3.605$	a(i)
A4=(2,1)	$\sqrt{(6-2)^2+(3-1)^2}=4.47$	b(i)
A5=(5,9)	$\sqrt{(6-5)^2+(3-9)^2}=6.08$	a(i)

	A3=(8,6)	
A1=(3,6)	$\sqrt{(8-3)^2+(6-6)^2}=5$	a(i)
A2=(6,3)	$\sqrt{(8-6)^2+(6-3)^2}=3.605$	a(i)
A3=(8,6)	0	
A4=(2,1)	$\sqrt{(8-2)^2+(6-1)^2}=7.746$	b(i)
A5=(5,9)	$\sqrt{(8-5)^2+(6-9)^2}=4.24$	a(i)

	A4=(2,1)	
A1=(3,6)	$\sqrt{(2-3)^2+(1-6)^2}=5.099$	b(i)
A2=(6,3)	$\sqrt{(2-6)^2+(1-3)^2}=4.47$	b(i)
A3=(8,6)	$\sqrt{(2-8)^2+(1-6)^2}=7.8$	b(i)

A4=(2,1)	0	
A5=(5,9)	$\sqrt{(2-5)^2+(1-9)^2}=8.5$	b(i)

	A5=(5,9)	
A1=(3,6)	$\sqrt{(5-3)^2+(9-6)^2}=3.6$	a(i)
A2=(6,3)	$\sqrt{(5-6)^2+(9-3)^2}=6.08$	a(i)
A3=(8,6)	$\sqrt{(5-8)^2+(9-6)^2}=4.24$	a(i)
A4=(2,1)	$\sqrt{(5-2)^2+(9-1)^2}=8.5$	b(i)
A5=(5,9)	0	

silhouette score for point A1:

$a(A1) = \frac{(4.24+5+3.605)}{3} = 4.281$

b(A1) = 5.099

$S(A1) = \frac{(5.099-4.28)}{5.099} = 0.143$

silhouette score for point A2:

$a(A2) = \frac{(4.24+3.605+6.08)}{3} = 4.641$

b(A2) = 4.47

$S(A2) = \frac{(4.47-4.641)}{4.641} = -0.037$

silhouette score for point A3:

$a(A3) = \frac{(5+3.605+4.24)}{3} = 4.281$

$b(A3) = 7.746$

$S(A3) = \frac{(7.746-4.281)}{7.746} = 0.447$

silhouette score for point A4:

$a(A4) = 0$

$b(A4) = \frac{(5.099+4.47+7.8+8.5)}{4} = 6.467$

$S(A4) = \frac{(6.467-0)}{6.467} = 1$

silhouette score for point A5:

$a(A5) = \frac{(3.6+6.08+4.24)}{3} = 4.64$

$b(A5) = 8.5$

$S(A5) = \frac{(8.5-4.64)}{8.5} = 0.454$

So the silhouette scores for each point are :

$S(A1) \approx 0.143$

$S(A2) \approx -0.037$

$S(A3) \approx 0.447$

$S(A4) \approx 1$

$S(A5) \approx 0.454$

Silhouette score = $\frac{0.143-0.037+0.447+1+0.454}{5} = 0.4$

WSS score

$$WSS = \sum_{i=1}^{m} (x_i - c_i)^2$$

	C1(5.5, 6)	C2(2, 1)
A1	$(3 - 5.5)^2 + (6 - 6)^2 = 6.25$	
A2	$(6 - 5.5)^2 + (3 - 6)^2$=9.25	
A3	$(8 - 5.5)^2 + (6 - 6)^2 = 6.25$	
A4		$(2 - 2)^2 + (1 - 1)^2 = 0$
A5	$(5 - 5.5)^2 + (9 - 6)^2 = 9.25$	
∑	31	

WSS = 31

1

```
df=pd.read_csv('MCSDatasetNEXTCONLab.csv');
df.head()
```

	ID	Latitude	Longitude	Day	Hour	Minute	Duration	RemainingTime	Resources	Coverage	OnPeakHours	GridNumber	Ligitimacy
0	1	45.442142	-75.303369	1	4	13	40	40	9	91	0	131380	1
1	1	45.442154	-75.304366	1	4	23	40	30	9	91	0	131380	1
2	1	45.442104	-75.303963	1	4	33	40	20	9	91	0	121996	1
3	1	45.441868	-75.303577	1	4	43	40	10	9	91	0	121996	1
4	2	45.447727	-75.147722	2	15	49	30	30	5	47	0	140784	1

Figure 1 the Dataset shape

A)

Here we split the features and labels for training and testing set assuming that the values 0, 1 and 2 in column day should be used for training dataset and only the value 3 in column day should be used for test dataset. ID indicates the index, and it cannot be used as the feature. Feature day is different for training and test datasets so it cannot be used as feature in datasets.

```
train_data = df[df['Day'].isin([0, 1, 2])]
test_data = df[df['Day'] == 3]
```

```
x_train=train_data.drop(["ID",'Day','Ligitimacy'],axis=1)
y_train=train_data['Ligitimacy']
x_test=test_data.drop(["ID",'Day','Ligitimacy'],axis=1)
y_test=test_data['Ligitimacy']
```

Figure 2 splitting dataset.

B) PROVIDE CONFUSION MATRIXES AND F1 SCORES OF NB AND KNN CLASSIFIER AS BAELINE PERFORMANCE

```
GNB=GaussianNB()
GNB.fit(x_train,y_train)
GNB_pred=GNB.predict(x_test)
GNB_Confusion_matrix=metrics.confusion_matrix(y_test,GNB_pred)
GNB_cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix = GNB_Confusion_matrix, display_labels = [False, True])
GNB_cm_display.plot()
plt.title("GaussianNB")
plt.show()

GNB_f1_score=f1_score(y_test,GNB_pred)
print("F1 Score: ",GNB_f1_score)
```

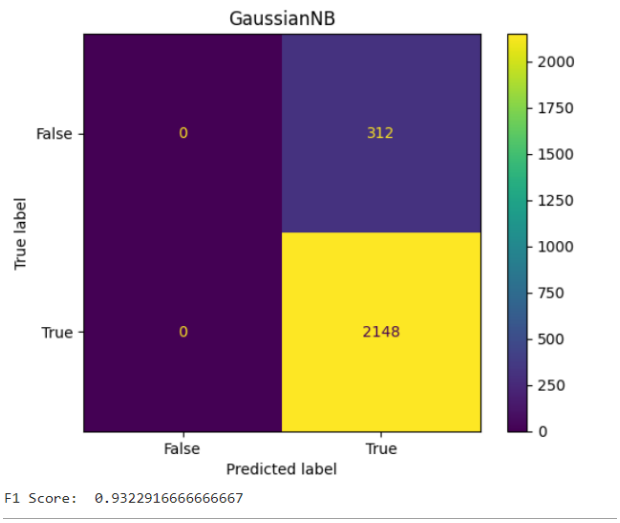
Figure 3 GNB implementation

Here we applied gaussian bayes classifier, calculated the f1_score and plotted the confusion matrix
We got the f1_score is as F1 Score: 0.9322916666666667 and in figure (####)

```
KNN=KNeighborsClassifier()
KNN.fit(x_train,y_train)
KNN_pred=KNN.predict(x_test)
KNN_Confusion_matrix=metrics.confusion_matrix(y_test,KNN_pred)
KNN_cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix = KNN_Confusion_matrix, display_labels = [False, True])
KNN_cm_display.plot()
plt.title("KNeighborsClassifier")
plt.show()
KNN_f1_score=f1_score(y_test,KNN_pred,)
print("F1 Score: ",KNN_f1_score)
```

Figure 4 KNN implementation

Here we applied Nearest Neighbor classifier, calculated the f1_score and plotted the confusion matrix
We got the f1_score is as F1 Score: 0.9227557411273487



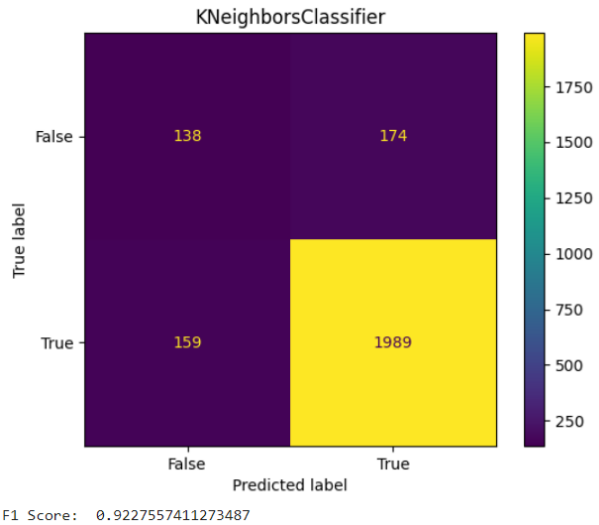


Figure 6 confusion matrix for test set model Gaussian Naive Bayes

Figure 6 confusion matrix for test set model **KNN**

After applying the GNB we found out that the model correctly predicted a positive class when the actual class was also positive as there are 2148 true positive and the cases where the model predicted positive class, but the actual class was negative are 312.

After applying the KNN we found out that the model correctly predicted a positive class when the actual class was also positive as there are 1989 true positive and the cases where the model predicted positive class but the actual class was negative are 174

C) PROVIDE 2D TSNE PLOTS, ONE FOR THE TRAINING SET AND ONE FOR THE TEST SET.

```
import plotly.express as px
fig = px.scatter(x=tsne_train[:, 0], y=tsne_train[:, 1], color=y_train)
fig.update_layout(
    title="Training Set t-SNE Plot",
    xaxis_title="First t-SNE",
    yaxis_title="Second t-SNE",
    width=800, height=600
)
fig.show()
```

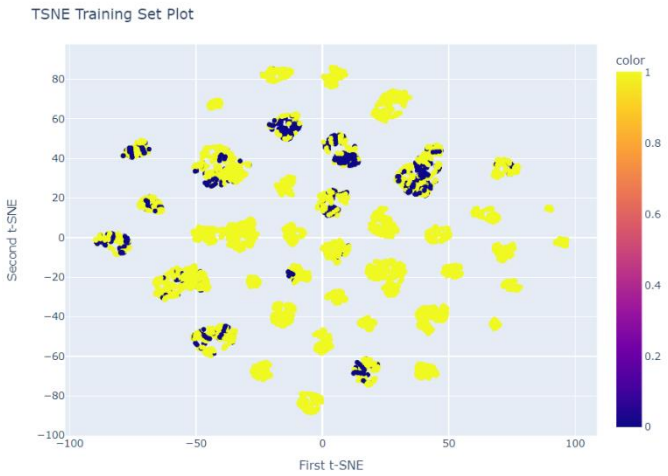


Figure 7 TSNE for training data

```
import plotly.express as px
tsne_test = tsne.fit_transform(x_test)
fig = px.scatter(x=tsne_test[:, 0], y=tsne_test[:, 1], color=y_test)
fig.update_layout(
    title="t-SNE visualization of testing data",
    xaxis_title="First t-SNE",
    yaxis_title="Second t-SNE",
    width=800, height=600
)
fig.show()
```

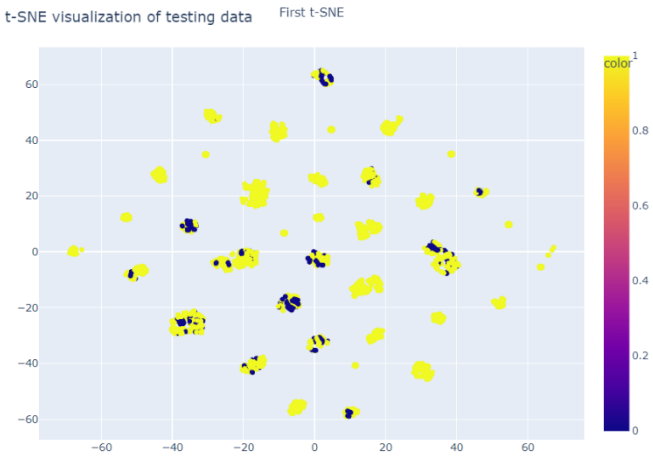


Figure 8 TSNE for testing data

CONCLUSION

W After applying a Gaussian Bayes classifier and k-nearest classifier to the data we got f1_score of 0,93 and correctly predicted 2148 positive cases while in correctly predicting 312 negative cases for the Gaussian Bayes classifier

And for the KNN classifier we got f1score of ,92 and correctly predicted 1989positive cases while incorrectly predicting 174 negative cases.

.

2)) APPLYING PCA AND AE ON THE TRAINING DATA AND APPLYING THE GNB AND KNN

A) THE BEST REDUCED DIMENSIONS OF PCA AND AE BASED ON F1 SCORE

```
# Define the range of dimensions/components for PCA and AE
dimensions = list(range(2, 11))

# Initialize lists to store F1 scores for each dimensionality reduction method
pca_f1_scores_nb = []
pca_f1_scores_knn = []
# Perform dimensionality reduction using PCA and AE for different dimensions
for n in dimensions:
    # PCA
    pca = PCA(n_components=n, random_state=0)
    pca_pipeline = Pipeline([
        ('scaler', MinMaxScaler()),
        ('pca', pca),
    ])
    X_train_pca = pca_pipeline.fit_transform(x_train)
    X_test_pca = pca_pipeline.transform(x_test)

    # Apply classifiers on the reduced data (after pca)
    GNB.fit(X_train_pca, y_train)
    GNB_pred_pca = GNB.predict(X_test_pca)
    GNB_f1_score_pca = f1_score(y_test, GNB_pred_pca)
    pca_f1_scores_nb.append(GNB_f1_score_pca)

    KNN.fit(X_train_pca, y_train)
    KNN_pred_pca = KNN.predict(X_test_pca)
    KNN_f1_score_pca = f1_score(y_test, KNN_pred_pca)
    pca_f1_scores_knn.append(KNN_f1_score_pca)
```

Figure 9 PCA implementation

Here we applied PCA to the training data and after that applied the GNB and KNN classifier to the reduced data and calculated the f1_score for each classifier.

We applied that for different dimension to store the best f1_score for each classifier in a list to get the best dimensionality reduction performance using one of the NB and KNN classifiers.

```
plt.plot(dimensions, pca_f1_scores_nb , marker='o', linestyle='-', color='b',label='(NB)')
plt.axhline(GNB_f1_score, linestyle='--', color='r', label='Baseline (NB)')
plt.plot(dimensions, pca_f1_scores_knn, marker='o', linestyle='-', color='g',label='(KNN)')
plt.axhline(KNN_f1_score, linestyle='--', color='y', label='Baseline (KNN)')
plt.title('PCA - F1 Score vs. Number of Components (NB)')
plt.xlabel('Number of Components')
plt.ylabel('F1 Score')
plt.legend()
plt.show()
```

Figure 11 code plot PCA

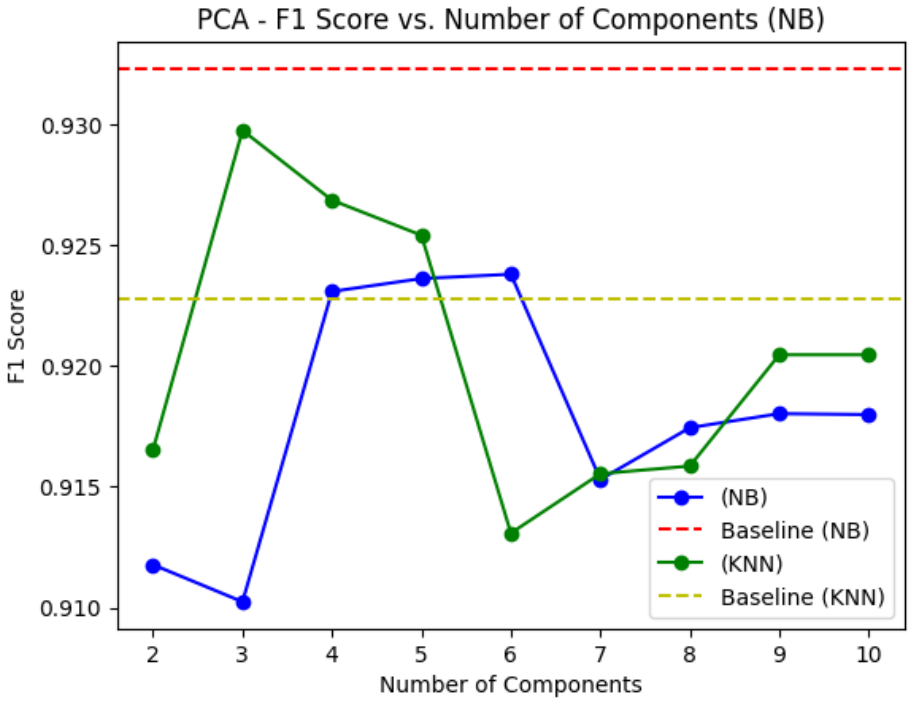


Figure 10 PCA F1 Score

.

```
plt.plot( dimension,ae_f1_scores_nb, marker='o', linestyle='-', color='b', label='AE_NB')
plt.axhline(GNB_f1_score, linestyle='--', color='r', label='Baseline (NB)')

plt.plot( dimension,ae_f1_scores_knn, marker='o', linestyle='-', color='g', label='AE_KNN')
plt.axhline(KNN_f1_score, linestyle='--', color='y', label='Baseline (KNN)')
plt.title('Autoencoder - F1 Score vs. Number of Components (NB)')
plt.xlabel('Number of Components')
plt.ylabel('F1 Score')
plt.legend()
plt.show()
```

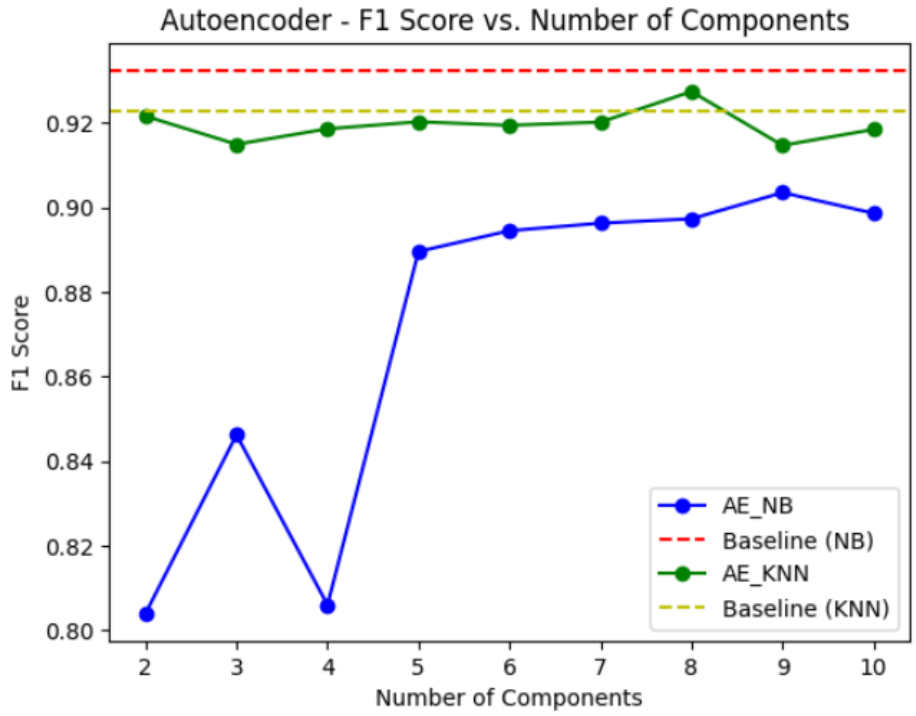


Figure 12 Figure 6 Auto Encoder-F1 Score VS number of component

B) PROVIDE 2D TSNE PLOTS FOR THE BEST PERFORMANCE

Perform dimensionality reduction using PCA and AE with the best dimensions

```
# Perform dimensionality reduction using PCA and AE with the best dimensions
best_pca_nb = PCA(n_components=best_pca_dimension_nb, random_state=0)
best_pca_knn = PCA(n_components=best_pca_dimension_knn, random_state=0)


best_pca_pipeline_nb = Pipeline([
    ('scaler', MinMaxScaler()),
    ('pca', best_pca_nb),
])

best_pca_pipeline_knn = Pipeline([
    ('scaler', MinMaxScaler()),
    ('pca', best_pca_knn),
])
X_train_best_pca_nb = best_pca_pipeline_nb.fit_transform(x_train)
X_test_best_pca_nb = best_pca_pipeline_nb.transform(x_test)

X_train_best_pca_knn = best_pca_pipeline_knn.fit_transform(x_train)
X_test_best_pca_knn = best_pca_pipeline_knn.transform(x_test)
```

Generate TSNE Plots For the best performance in the previous part in PCA

```
from sklearn.manifold import TSNE
tsne_train_best_pca_nb = TSNE(n_components=2, random_state=0).fit_transform(X_train_best_pca_nb)
tsne_test_best_pca_nb = TSNE(n_components=2, random_state=0).fit_transform(X_test_best_pca_nb)
tsne_train_best_pca_knn = TSNE(n_components=2, random_state=0).fit_transform(X_train_best_pca_knn)
tsne_test_best_pca_knn = TSNE(n_components=2, random_state=0).fit_transform(X_test_best_pca_knn)
```

```
dispaly_TSNE(data_tsne_after_fit=tsne_train_best_pca_nb,y_for_color=y_train,title="TSNE - Best PCA NB (Train)")
dispaly_TSNE(data_tsne_after_fit=tsne_test_best_pca_nb,y_for_color=y_test,title="TSNE - Best PCA NB (Test)")

######TSNE - Best PCA KNN

dispaly_TSNE(data_tsne_after_fit=tsne_train_best_pca_knn,y_for_color=y_train,title="TSNE - Best PCA KNN (Train)")
dispaly_TSNE(data_tsne_after_fit=tsne_test_best_pca_knn,y_for_color=y_test,title="TSNE - Best PCA KNN (Test)")
```

TSNE Visualizations For the best performance in the previous part in PCA for training and testing data for both classifier GNB and KNN

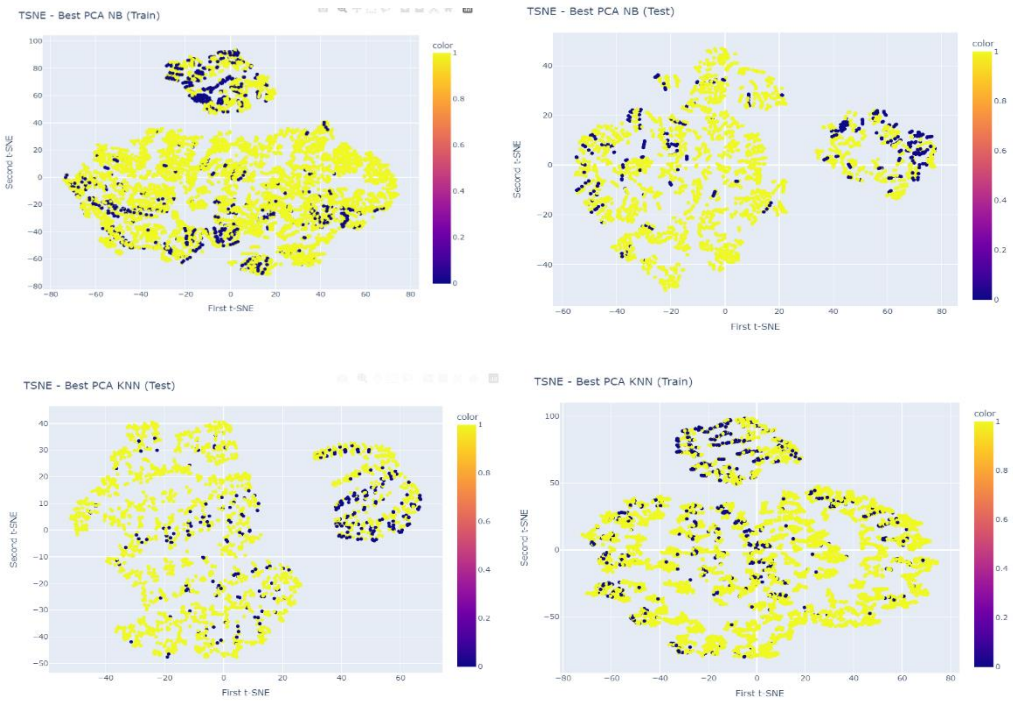


Figure 13 TSNE Visualizations For the best performance in the previous part in PCA

get the best dimensions and Perform dimensionality reduction using AE with the best dimensions

```
best_ae_nb = MLPRegressor(hidden_layer_sizes=[best_ae_dimension_nb], activation='relu', solver='adam',random_state=0)
best_ae_knn = MLPRegressor(hidden_layer_sizes=[best_ae_dimension_knn],activation='relu', solver='adam',random_state=0)
scaler= MinMaxScaler()
X_train = scaler.fit_transform(x_train,y_train)
X_test = scaler.transform(x_test)
# Reduce dimensionality with the trained autoencoder
best_ae_nb.fit(X_train,X_train)
X_train_best_ae_nb = best_ae_nb.predict(X_train)
X_test_best_ae_nb = best_ae_nb.predict(X_test)
# X_test_best_ae_nb = best_ae_pipeline_nb['ae'].predict(x_test)
# X_train_best_ae_knn = best_ae_pipeline_knn['ae'].fit(x_train,y_train)
# X_test_best_ae_knn = best_ae_pipeline_knn['ae'].predict(x_test)
best_ae_knn.fit(X_train,X_train)
X_train_best_ae_knn = best_ae_knn.predict(X_train)
X_test_best_ae_knn = best_ae_knn.predict(X_test)
```

```
tsne_train_best_ae_nb = TSNE(n_components=2, random_state=0).fit_transform(X_train_best_ae_nb)
tsne_test_best_ae_nb = TSNE(n_components=2, random_state=0).fit_transform(X_test_best_ae_nb)
tsne_train_best_ae_knn = TSNE(n_components=2, random_state=0).fit_transform(X_train_best_ae_knn)
tsne_test_best_ae_knn = TSNE(n_components=2, random_state=0).fit_transform(X_test_best_ae_knn)
```

```
##TSNE plot NB
dispaly_TSNE(data_tsne_after_fit=tsne_train_best_ae_nb,y_for_color=y_train,title="TSNE - Best AE NB (Train)")
dispaly_TSNE(data_tsne_after_fit=tsne_test_best_ae_nb,y_for_color=y_test,title="TSNE - Best AE NB (Test)")


##TSNE plot KNN
dispaly_TSNE(data_tsne_after_fit=tsne_train_best_ae_knn,y_for_color=y_train,title="TSNE - Best AE KNN (Train)")
dispaly_TSNE(data_tsne_after_fit=tsne_test_best_ae_knn,y_for_color=y_test,title="TSNE - Best AE KNN (Test)")
```

TSNE Visualizations For the best performance in the previous part in AE for training and testing data for both classifier GNB and KNN

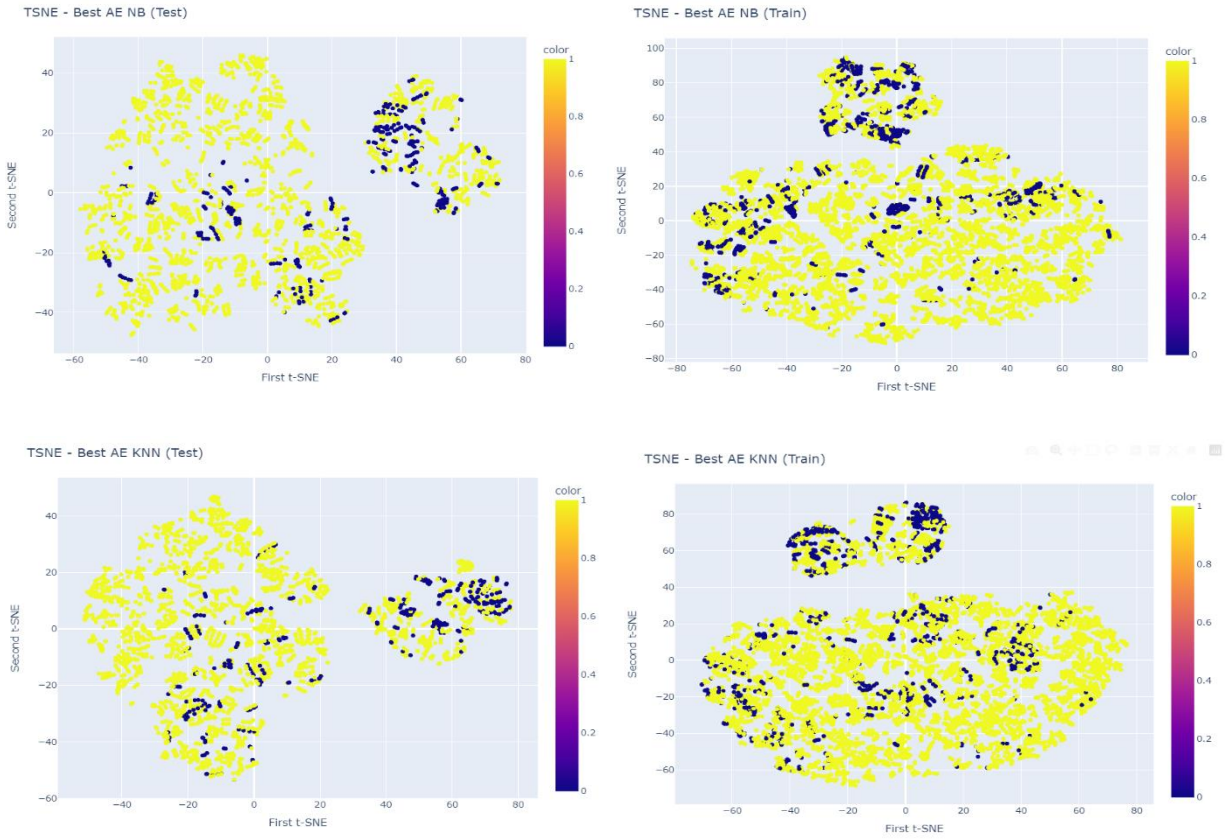


Figure 14 TSNE Visualizations For the best performance in the previous part in AE

CONCLUSION

We tried different dimensions and stored F1 score for each classifier in a list to get the best dimensionality reduction performance using one of the NB and KNN classifiers we found the best dimensions for PCA after applying the GNB classifiers were 6 and for KNN classifier were 3 dimensions and for the autoencoder, the best dimensions after applying GNB classifier were 9 dimensions and for KNN classifier were 8 dimensions

3) USE THE FOLLOWING FEATURE SELECTION METHODS (ONE FOR EACH METHOD).

A) FILTER METHODS

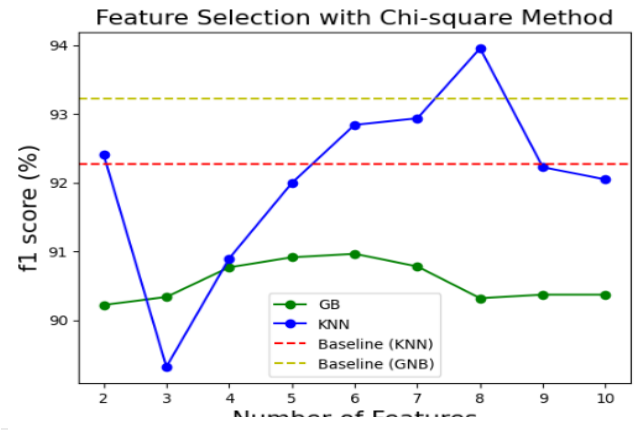


Figure 15 feature selection (Filter Method) using the chi-square method.

The results show that the KNN classifier achieved a higher maximum f1 score of 93.96% with 8 features selected, compared to the GB classifier which achieved a maximum f1 score of 90.96% with 6 features selected. These results suggest that the Chi-square method with KNN classifier can potentially improve the performance of machine learning models on the data.

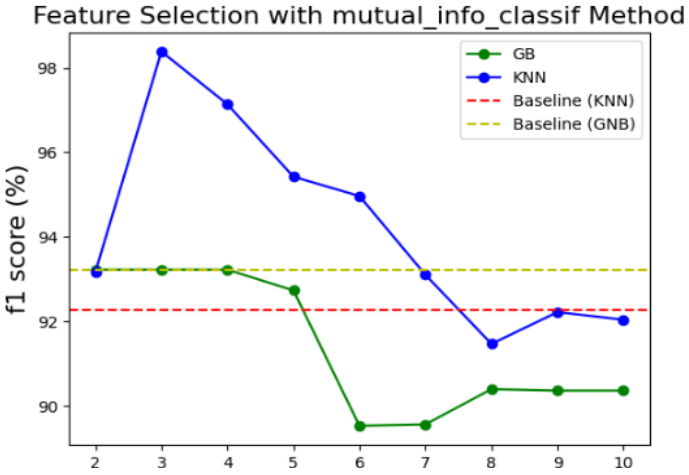
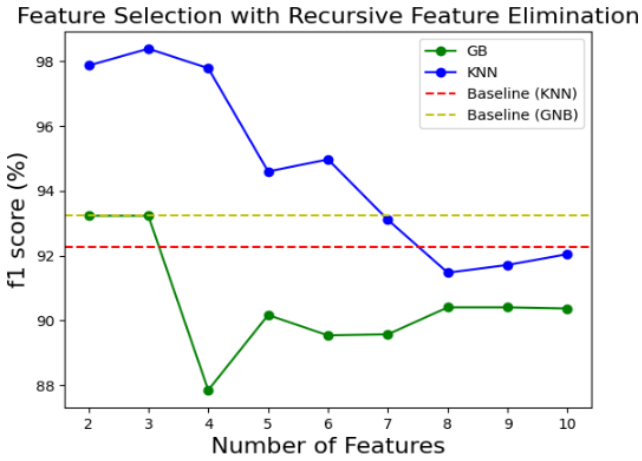


Figure 16 : Feature selection (Filter Method) using Mutual information measures.

The results show that the KNN classifier achieved a higher maximum f1 score of 98.38% with 3 features selected, compared to the GB classifier which achieved a maximum f1 score of 93.23% with 2 features selected. These results suggest that the mutual information method with KNN classifier can potentially improve the performance of machine learning models on the data.

B) WRAPPER METHODS



The results show that the KNN classifier achieved a higher maximum f1 score of 98.38% with 3 features selected, compared to the GB classifier which achieved a maximum f1 score of 93.23% with 2 features selected. These results suggest that the RFE method with KNN classifier can potentially improve the performance of machine learning models on the data.

C) 2D TSNE PLOTS

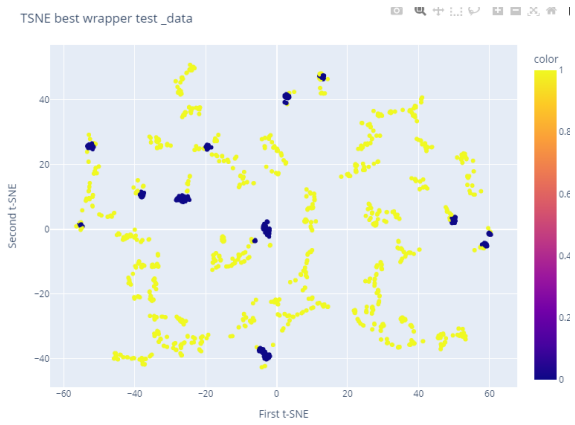


Figure 18 2D TSNE plot of the selected features using the RFE wrapper method on the test sets.

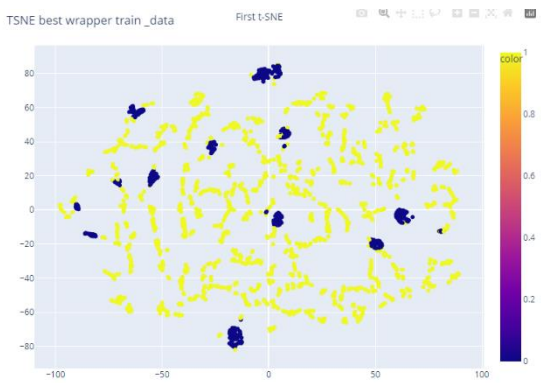


Figure 18 2D TSNE plot of the selected features using the RFE wrapper method on the training set.

CONCLUSION

We chose wrapper was the best but not for the higher F1 score ,we guess this would overfitting but we chose it because had the acceptable F1 score at the half of all feature in KNN and GNB

4 LATITUDE AND LONGITUDE FEATURES SHOULD BE CONSIDERED FOR CLUSTERING BASED METHODS

```
[129] train_data_x=train_data.drop(["ID",'Day'],axis=1)
      x_train=train_data.drop(["ID",'Day','Ligitimacy','Hour','Minute','RemainingTime','Resources','Coverage','OnPeakHours','GridNumber','Duration'],axis=1)

      y_train=train_data['Ligitimacy']
      x_test=test_data.drop(["ID",'Day','Ligitimacy','Hour','Minute','RemainingTime','Resources','Coverage','OnPeakHours','GridNumber','Duration'],axis=1)
      x_test_x=test_data.drop(["ID",'Day'],axis=1)

      y_test=test_data['Ligitimacy']
```

Figure 21 prepare data set.

```
def calculter(model_after_fit,x_test,data_test):
   culter={}
   for index, row in x_test.iterrows():
      elemat=pd.DataFrame(np.array(row).reshape(1, -1),columns=['Latitude','Longitude'])
      number_culster=model_after_fit.predict(elemat)

      Ligitimacy=((data_test.loc[(data_test['Longitude'] == row['Longitude']  ) & (data_test['Latitude'] == row['Latitude'])])["Ligitimacy"]).astype(int)

      number_culster=int(number_culster)
      try:

       x=int(Ligitimacy)

      except:
        # to  remove duplicate and get on of them
         x=list(Ligitimacy)[0]


      try:
        cout=culter[number_culster]
        if(x==1 and cout!=-1):
           # for calculate the number ligamtate in each class
          culter[number_culster]=cout+1
        else:
          # for  fake classes not included
          culter[number_culster]=-1
      except:

       culter[number_culster]=x
   return culter
```

Figure 20 function to calculate the number of only legitimate.

```
def culterdict(dim,data_dict):
    cout=0
    for i in range(dim):
      if(data_dict[i]!=-1):
        cout=cout+data_dict[i]
    return cout
```

Figure 19 function to calculate to sum all only legitimate.

Here we made some function to help us in part 4 and we will used them.

A) APPLY K-MEANS ALGORITHM TO PLOT THE NUMBER OF CLUSTERS (8,12,16,20 AND 32)

```
from numpy.core.multiarray import result_type
from sklearn.cluster import KMeans
import numpy as np
result={}
for i  in [8,12,16,20,32]:
    kmeans = KMeans(n_clusters=i, random_state=0, n_init="auto").fit(x_train)
    #print("centroid",kmeans.cluster_centers_)
    x=calculter(kmeans,x_train,train_data)
    #print(x)
    score=culterdict(i,x)
    result[i]=score
```

```
[133] result

    {8: 1520, 12: 2717, 16: 3291, 20: 4158, 32: 4744}
```

Figure 22 K-means implementation

```
dict
```

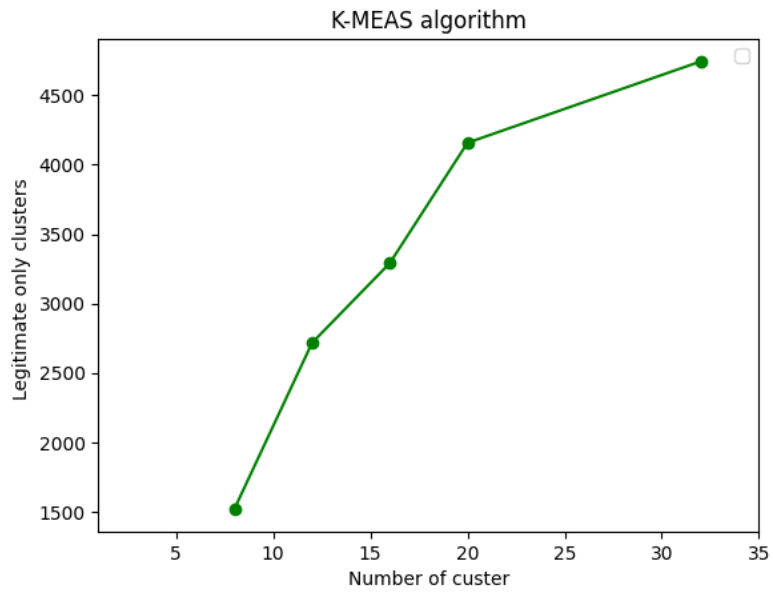


Figure 23 number of legitimate only vs number of clusters

The output is a plot showing the number of legitimate only clusters for different numbers of clusters (8, 12, 16, 20, and 32) generated by the K-Means algorithm. The number of legitimate only clusters is represented on the y-axis, and the number of clusters is represented on the x-axis. The plot shows that the number of legitimate only clusters increases as the number of clusters increases, with the highest number of legitimate only clusters (4744) generated by 32 clusters. The output also includes a dictionary containing the number of legitimate only clusters generated by each number of clusters (8, 12, 16, 20, and 32).

B) APPLY SOFM ALGORITHM TO PLOT THE NUMBER OF CLUSTERS (8,12,16,20 AND 32)

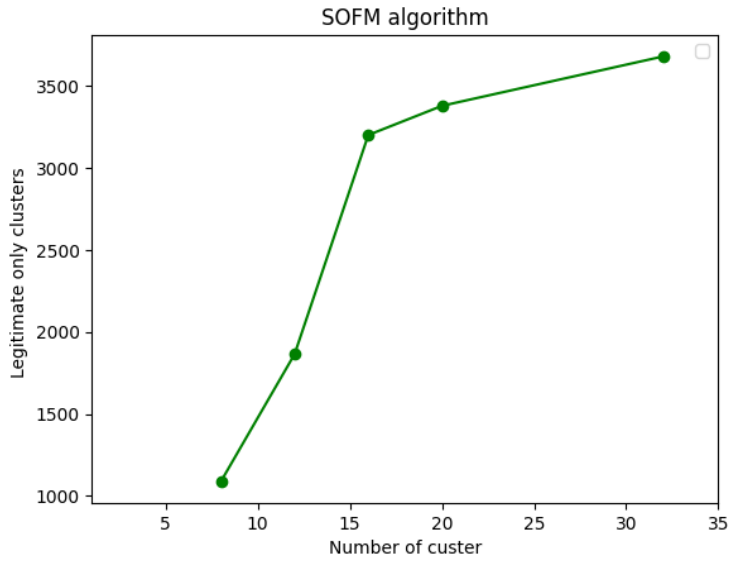
```
from sklearn_som.som import SOM
result={}
data_test=train_data
for i  in [8,12,16,20,32]:
    sofm=SOM(m=int(i/2), n=2, dim=2,random_state=0)
    sofm.fit(np.array(x_train))
    culter={}

    for index, row in x_train.iterrows():
        #elemat=pd.DataFrame(np.array(row).reshape(1, -1),columns=['Latitude','Longitude'])
        number_culster=sofm.predict(np.array(row).reshape(1, -1))
        #number_culster=model_after_fit.fit_predict(elemat)
        Ligitimacy=((data_test.loc[(data_test['Longitude'] == row[1]  ) & (data_test['Latitude'] == row[0])])["Ligitimacy"]).astype(int)
        number_culster=int(number_culster)
        try:
         x=int(Ligitimacy)
        except:
          #for remove duplacates return form data set
          x=list(Ligitimacy)[0]
        try:
          cout=culter[number_culster]
          if(x==1 and cout!=-1):
            culter[number_culster]=cout+1
          else:
            culter[number_culster]=-1
        except:
          #for intiztion culster in dict
         culter[number_culster]=x
    cout=0
    for K in culter.keys():
      if(culter[K]!=-1):
        cout=cout+culter[K]

    result[i]=cout
```

Figure 24 SOFM implementation

```
dict_items([(8, 1089), (12, 1865), (16, 3200), (20, 3378), (32, 3679)])
```



The output shows the number of legitimate-only clusters obtained by running the Self-Organizing Feature Map (SOFM) algorithm on a dataset with different numbers of clusters, using the code provided. The output shows that increasing the number of clusters generally leads to an increase in the number of legitimate-only clusters, as expected. Specifically, the algorithm obtained 1089 legitimate-only clusters with 8 clusters, 1865 with 12 clusters, 3200 with 16 clusters, 3378 with 20 clusters, and 3679 with 32 clusters.

Figure 25 number of legitimate only vs number of clusters

The output shows the number of legitimate-only clusters obtained by running the Self-Organizing Feature Map (SOFM) algorithm on a dataset with different numbers of clusters, using the code provided. The output shows that increasing the number of clusters generally leads to an increase in the number of legitimate-only clusters, as expected. Specifically, the algorithm obtained 1089 legitimate-only clusters with 8 clusters, 1865 with 12 clusters, 3200 with 16 clusters, 3378 with 20 clusters, and 3679 with 32 clusters.

C) APPLY DBSCAN ALGORITHM TO PLOT THE NUMBER OF CLUSTERS (8,12,16,20 AND 32)

```
def calculter_DBSCAN(culters_number,x_test,data_test):
    culter={}
    counter=0
    for index, row in x_test.iterrows():

        number_culster=culters_number[counter]
        counter=counter+1

        Ligitimacy=((data_test.loc[(data_test['Longitude'] == row['Longitude']  ) & (data_test['Latitude'] == row['Latitude'])])["Ligitimacy"]).astype(int)

        number_culster=int(number_culster)
        try:

         x=int(Ligitimacy)

        except:
           #print(list(Ligitimacy))
           x=list(Ligitimacy)[0]

        try:
          cout=culter[number_culster]
          if(x==1 and cout!=-1):

            culter[number_culster]=cout+1
          else:
            culter[number_culster]=-1
        except:
          #for intiztion culster in dict
          culter[number_culster]=x
    return culter
```

Figure 26 enhanced function to DBSCAN

This function like the first function but with some additions feature to be useable for DBSCAN, we sent the predilected classes as parameter and after this we start calculate the only legitimate in each class and save it in dictionary and return this dictionary .

```
from sklearn.cluster import DBSCAN
result={}
minPts = 5
#[0.01, .0001, .5, .00310, .0020]
for index ,i  in enumerate([0.001, .005, .002, .0001, .00632]):
    #kmeans = KMeans(n_clusters=i, random_state=0, n_init="auto").fit(x_train)
    clustering = DBSCAN(eps=i, min_samples=7)
    #clustering.fit(np.array(x_train))
    number_culster=clustering.fit_predict(x_train)
    print("centroid",clustering.labels_)
    culter=calculter_DBSCAN(number_culster,x_train,train_data)
    print(x)
    cout=0
    print(culter)
    print(len(culter))
    for K in culter.keys():
      if(culter[K]!=-1):
        cout=cout+culter[K]

    result[i]=cout
```

```
centroid [ 0  0  0 ... -1 -1 -1]
1
{0: 11, -1: -1, 92: 7, 1: 7, 2: -1, 3: -1, 4: 11, 86: 6, 5: -1, 127: 7, 6: -1, 7: 8,
132
centroid [0 0 0 ... 0 0 0]
1
{0: -1, 1: 198, 2: 84, -1: 328, 3: 122, 4: -1, 5: 35, 6: -1, 7: -1, 8: 17, 9: 73, 10:
71
centroid [ 0  0  0 ... 84 84 84]
1
{0: 32, -1: 3045, 121: 11, 1: 8, 2: 11, 3: 22, 4: 16, 5: 22, 6: -1, 7: -1, 8: -1, 9:
258
centroid [-1 -1 -1 ... -1 -1 -1]
1
{-1: -1}
1
centroid [0 0 0 ... 0 0 0]
1
{0: -1, -1: 95, 1: 59, 2: 21, 3: 47, 4: -1, 5: 16, 6: 16, 7: 11, 8: 13, 9: 15, 10: 14
31
```

In this figure we tried a lot of epsilon and we chose this numbers
(.001 ,.005, .002,.0001,.00632)
this gave us a variation of clusters with each epsilon number.
first with .001 we got 132 clusters
, with .005 we got 71 clusters
, with .002 we got 258 clusters
,with .0001 we got 1 cluster and all data outlier
,and the last with .0632 we got 31 clusters

Figure 27 DBSCAN implementation

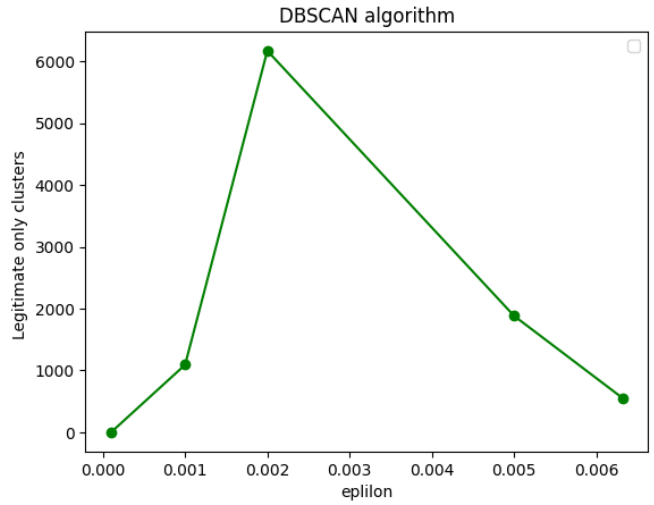
```
result
```

```
{0.001: 1092, 0.005: 1884, 0.002: 6173, 0.0001: 0, 0.00632: 553}
```

Figure 28 results from each epsilon.

This results for showing us the number of only legitimate at each epsilon.

```
dict_items([(0.001, 1092), (0.005, 1884), (0.002, 6173), (0.0001, 0), (0.00632, 553)])
```



this gave us a variation of clusters with each epsilon number.
first with .0001 replace 1 cluster and all data outlier, with .001 replace 132 clusters,
with .002 replace 258 clusters
, with .005 replace 71 clusters
,and the last with .0632 we got 31 clusters

Figure 29 number of legitimate only vs epsilon

The output result shows the number of legitimate only clusters found by the DBSCAN algorithm for different values of the hyperparameter epsilon. The hyperparameter epsilon controls the radius of a neighborhood around a point, and the algorithm considers points that are within this radius to be part of the same cluster.

The first four rows of the output show the results for epsilon values of 0.001, 0.005, 0.002, and 0.0001. The number of legitimate only clusters found for these values are 1092, 1884, 6173, and 0, respectively. This suggests that as the value of epsilon increases, the number of legitimate only clusters found decreases. This is because a larger value of epsilon means that more points are part of the same cluster, which can lead to the merging of legitimate and illegitimate clusters.

CONCLUSION

With the result form above the winner is DBSCAN at epsilon .002 and we get 258 clusters and this overcome the K-means with 32 number of clusters got 4744 and SOFM with 32 cluster got 3679.
we have one comment when the number of cluster increased the number of fake legitimate decreased and we get high number at higher clusters.