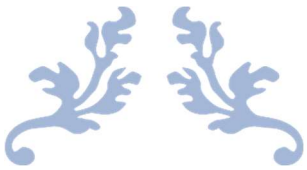
uOttawa



ELG 5255:
Applied Machine Learning
Assignment 3

Dr. Murat Simsek

GROUP 14

Table of Contents

Table of Figures

Part 1:

Question (1):

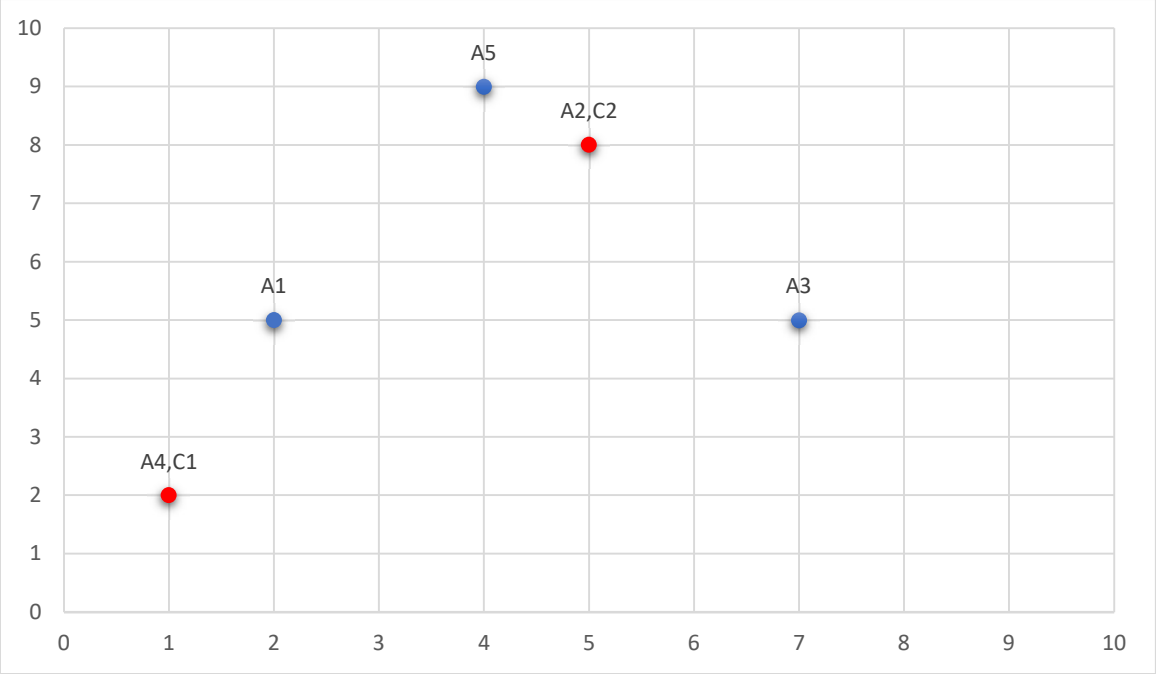


Figure 1 The data before clustering

a)

1. Calculating the distance of all data points to the centroids and assigning data points to the closest cluster.

$$Euclidean\ distance(A1, A2) = \sqrt{(2-5)^2 + (5-8)^2} = 4.24$$

$$Euclidean\ distance(A1, A4) = \sqrt{(2-1)^2 + (5-2)^2} = 3.16$$

$$\rightarrow \boldsymbol{A1\ will\ be\ in\ the\ same\ cluster\ with\ A4(given\ centroid)}$$

$$Euclidean\ distance(A2, A2) = \sqrt{(5-5)^2 + (8-8)^2} = 0$$

$$\rightarrow \boldsymbol{A2\ will\ be\ in\ the\ same\ cluster\ with\ A2(given\ centroid)}$$

$$Euclidean\ distance(A2, A4) = \sqrt{(5-1)^2 + (8-2)^2} = 7.21$$

$$Euclidean\ distance(A3, A2) = \sqrt{(7-5)^2 + (5-8)^2} = 3.61$$

$$\rightarrow \boldsymbol{A3\ will\ be\ in\ the\ same\ cluster\ with\ A2(given\ centroid)}$$

$$Euclidean\ distance(A3, A4) = \sqrt{(7-1)^2 + (5-2)^2} = 6.71$$

$$Euclidean\ distance(A4, A2) = 7.21$$

$$Euclidean\ distance(A4, A4) = \sqrt{(1-1)^2 + (2-2)^2} = 0$$

$$\rightarrow \boldsymbol{A4\ will\ be\ in\ the\ same\ cluster\ with\ A4(given\ centroid)}$$

$$Euclidean\ distance(A5, A2) = \sqrt{(4-5)^2 + (9-8)^2} = 1.41$$

$$\rightarrow \boldsymbol{A5\ will\ be\ in\ the\ same\ cluster\ with\ A2(given\ centroid)}$$

$$Euclidean\ distance(A5, A4) = \sqrt{(4-1)^2 + (9-2)^2} = 7.62$$

2. Finding the new centroids.

$$C_1 = \frac{(2+1\ ,5+2)}{2} = (1.5\ ,3.5)$$

$$C_2 = \frac{(5+7+4\ ,8+5+9)}{3} = (5.33\ ,7.33)$$

B)

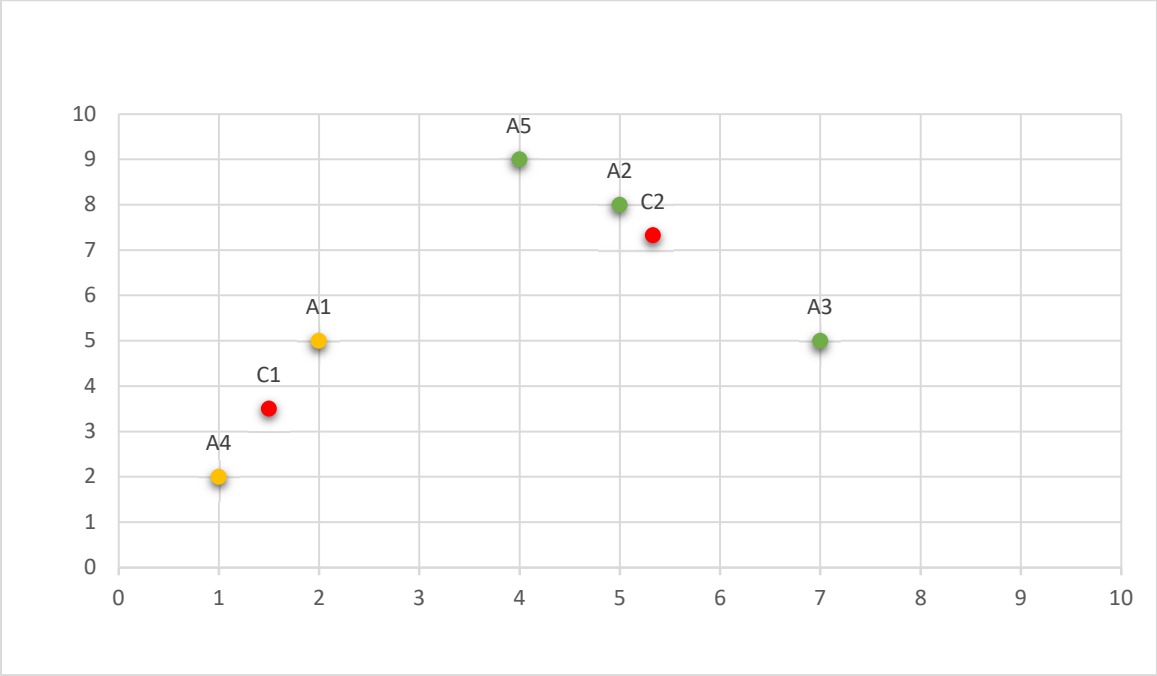


Figure 2 The Clustered data with their new centroids

C)

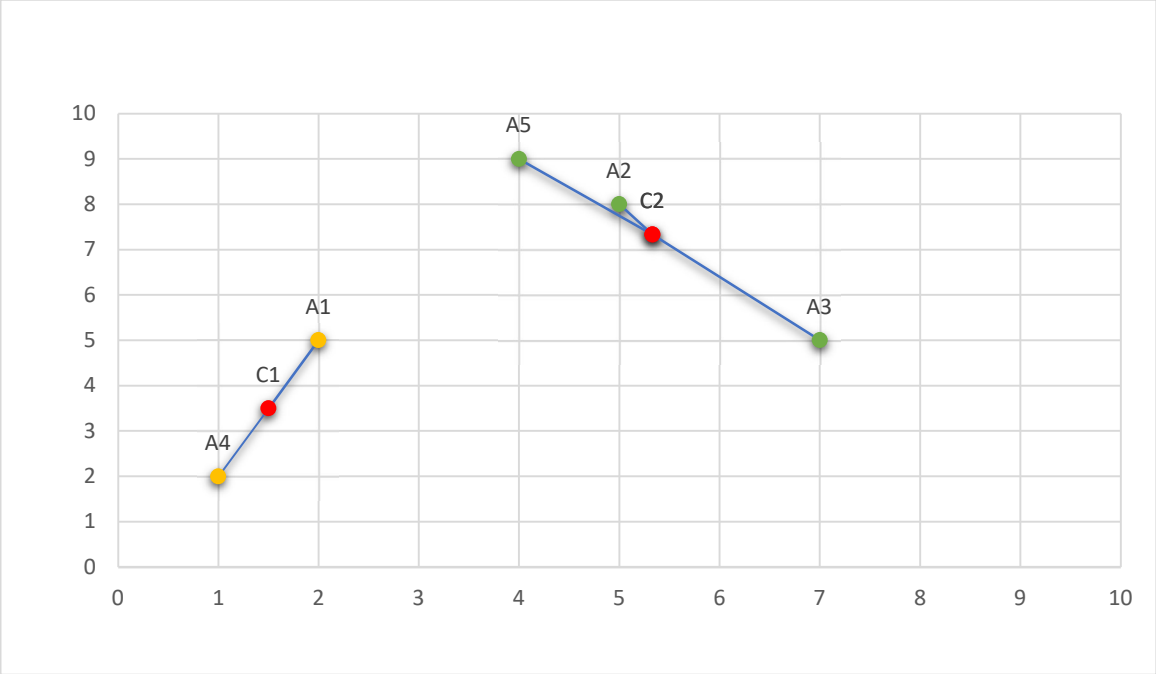


Figure 3 Calculating WSS

$$WSS = \sum_{i=1}^{5} (Ai - C_i)^2 = (A1 - C_1)^2 + (A2 - C_2)^2 + (A3 - C_2)^2 + (A4 - C_1)^2 + (A5 - C_2)^2$$

$$= ((2 - 1.5)^2 + (5 - 3.5)^2) + ((5 - 5.33)^2 + (8 - 7.33)^2)$$
$$+ ((7 - 5.33)^2 + (5 - 7.33)^2) + ((1 - 1.5)^2 + (2 - 3.5)^2)$$
$$+ ((4 - 5.33)^2 + (9 - 7.33)^2) = \mathbf{18.3334}$$

Silhouette:

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

$$b(1) = \frac{Euclidean\ distance(A1, A2) + Euclidean\ distance(A1, A3) + Euclidean\ distance(A1, A5)}{3}$$

$$= \frac{4.24 + \sqrt{(2-7)^2 + (5-5)^2} + \sqrt{(2-4)^2 + (5-9)^2}}{3} = \frac{4.24 + 5 + 4.47}{3} = \mathbf{4.57}$$

$$a(1) = Euclidean\ distance(A1, A4) = \mathbf{3.16}$$

$$S(1) = \frac{b(1) - a(1)}{\max\{a(1), b(1)\}} = \frac{b(1) - a(1)}{b(1)} = \mathbf{0.309}$$

$$b(2) = \frac{Euclidean\ distance(A2, A1) + Euclidean\ distance(A2, A4)}{2} = \frac{4.24 + 7.21}{2} = \mathbf{5.725}$$

$$a(2) = \frac{Euclidean\ distance(A2, A3) + Euclidean\ distance(A2, A5)}{2} = \frac{3.61 + 1.41}{2} = \mathbf{2.51}$$

$$S(2) = \frac{b(2) - a(2)}{\max\{a(2), b(2)\}} = \frac{b(2) - a(2)}{b(2)} = \mathbf{0.562}$$

$$b(3) = \frac{Euclidean\ distance(A3, A1) + Euclidean\ distance(A3, A4)}{2} = \frac{5 + 6.71}{2} = \mathbf{5.855}$$

$$a(3) = \frac{Euclidean\ distance(A3, A2) + Euclidean\ distance(A3, A5)}{2} = \frac{3.61 + \sqrt{(7-4)^2 + (5-9)^2}}{2}$$

$$= \frac{3.61 + 5}{2} = \mathbf{4.305}$$

$$S(3) = \frac{b(3) - a(3)}{\max\{a(3), b(3)\}} = \frac{b(3) - a(3)}{b(3)} = \mathbf{0.265}$$

$$b(4) = \frac{Euclidean\ distance(A4, A2) + Euclidean\ distance(A4, A3) + Euclidean\ distance(A4, A5)}{3}$$

$$= \frac{7.21 + 6.71 + 7.62}{3} = \mathbf{7.18}$$

$$a(4) = Euclidean\ distance(A4, A1) = \mathbf{3.16}$$

$$S(4) = \frac{b(4) - a(4)}{\max\{a(4), b(4)\}} = \frac{b(4) - a(4)}{b(4)} = \mathbf{0.56}$$

$$b(5) = \frac{Euclidean\ distance(A5, A1) + Euclidean\ distance(A5, A4)}{2} = \frac{4.47 + 7.62}{2} = \mathbf{6.045}$$

$$a(5) = \frac{Euclidean\ distance(A5, A2) + Euclidean\ distance(A5, A3)}{2} = \frac{1.41 + 5}{2} = \mathbf{3.205}$$

$$S(5) = \frac{b(5) - a(5)}{\max\{a(5), b(5)\}} = \frac{b(5) - a(5)}{b(5)} = \mathbf{0.47}$$

$$Silhouette\ Score = AverageSilhouette = \frac{S(1) + S(2) + S(3) + S(4) + S(5)}{5} = \mathbf{0.4332}$$

Part 2:

1.

```
X=df.iloc[:,:-1]
y=df.iloc[:,-1]
```

```
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.25, shuffle=False )
```

```
print(x_train.shape)
print(y_train.shape)
print(x_test.shape)
print(y_test.shape)
```

```
(576, 8)
(576,)
(192, 8)
(192,)
```

Figure 4 Splitting data output

```
def eval(model,x_test,y_test):
  y_pred=model.predict(x_test)
  acc=accuracy_score(y_test,y_pred)
  print("_____________________________________________________")
  print("The Test acceracy :",acc)
  print("_____________________________________________________")
  print(classification_report(y_test,y_pred))
  print("_____________________________________________________")
  ConfusionMatrixDisplay.from_estimator(
      model, x_test, y_test)
```

```
def tsne(X,y,k,title):
  tsne = TSNE(n_components=2, verbose=1, random_state=0)
  z = tsne.fit_transform(X)
  df = pd.DataFrame()
  df["y"] = y
  df["comp-1"] = z[:,0]
  df["comp-2"] = z[:,1]
  sns.scatterplot(x="comp-1", y="comp-2", hue=df.y.tolist(),
                palette=sns.color_palette("hls", k),
                data=df).set(title=title)
  plt.show()
```

A)

```
eval(LR,x_test,y_test)
```

```
_____________________________________________
The Test acceracy : 0.75
_____________________________________________
              precision    recall  f1-score   support

           0       0.79      0.84      0.82       127
           1       0.65      0.57      0.61        65

    accuracy                           0.75       192
   macro avg       0.72      0.71      0.71       192
weighted avg       0.74      0.75      0.75       192
```

Figure 6 Logistic Regression accuracy and classification report

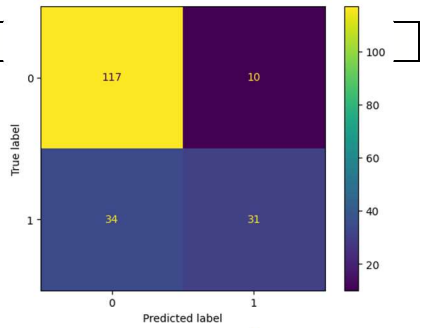


Figure 5 Logistic Regression Confusion matrix

```
eval(KNN,x_test,y_test)
```

```
_____________________________________________
The Test acceracy : 0.7708333333333334
_____________________________________________
              precision    recall  f1-score   support

           0       0.77      0.92      0.84       127
           1       0.76      0.48      0.58        65

    accuracy                           0.77       192
   macro avg       0.77      0.70      0.71       192
weighted avg       0.77      0.77      0.75       192
```

Figure 8 KNN accuracy and classification report

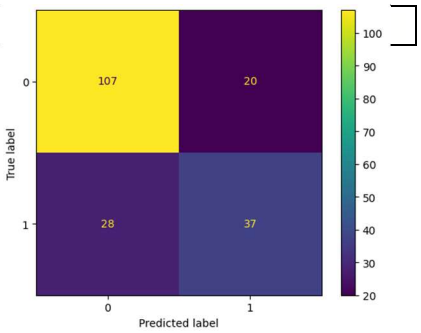


Figure 7 KNN Confusion matrix

B)

```
y_train_pred=LR.predict(x_train)
tsne(x_train,y_train_pred,2,'train dataset')
```

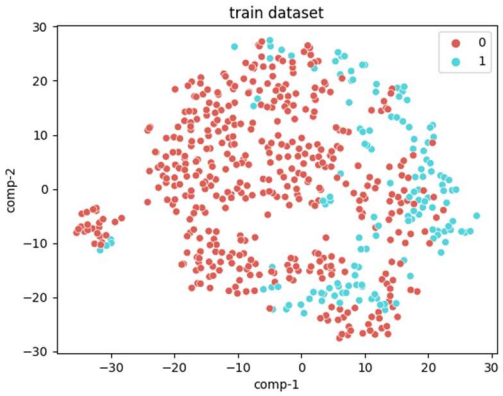


Figure 9 T-SNE for training dataset for LR

```
y_test_pred=LR.predict(x_test)
tsne(x_test,y_test_pred,2,'test dataset')
```

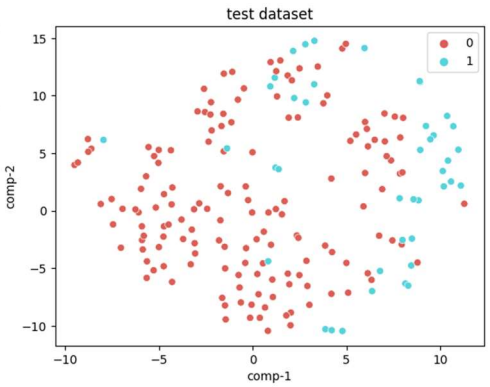


Figure 10 T-SNE for Test dataset for LR

```
y_train_pred=KNN.predict(x_train)
tsne(x_train,y_train_pred,2,'train dataset')
```

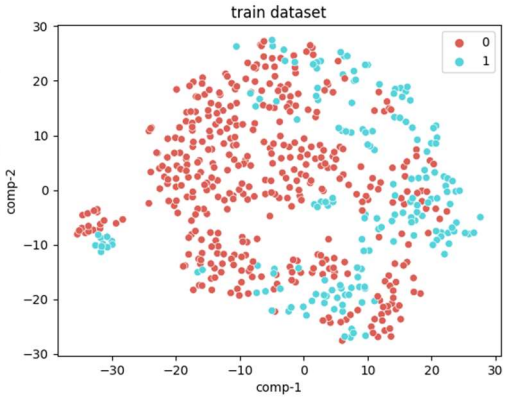


Figure 11 T-SNE for training dataset for KNN

```
y_test_pred=KNN.predict(x_test)
tsne(x_test,y_test_pred,2,'test dataset')
```

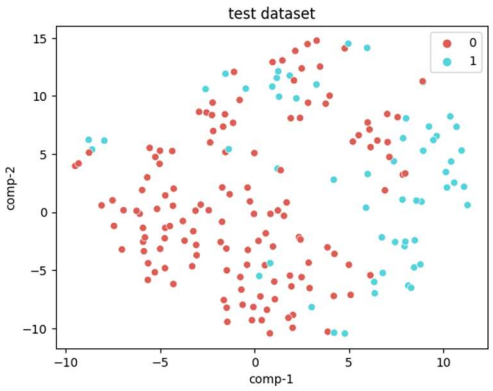


Figure 12 T-SNE for Test dataset for KNN

2.

A)

```
from yellowbrick.cluster import KElbowVisualizer
fig = plt.figure(figsize=(15, 10))
Elbow_M = KElbowVisualizer(KMeans(), k=20,metric='silhouette',timings=
False)
Elbow_M.fit(X)
Elbow_M.show()
```

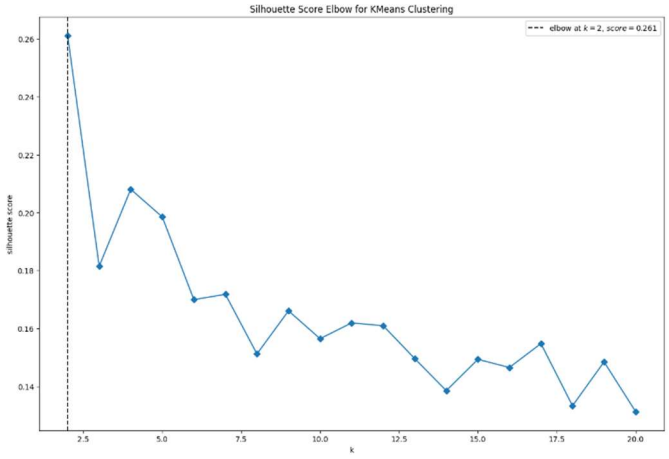


Figure 13 silhouette score elbow method for K-means

B)

shown in above figure the best value of k is 2 because it has the highest silhouette score

C)

```
model = KMeans(n_clusters=2)
model.fit(X)

tsne(X,model.labels_,2,'data')
```

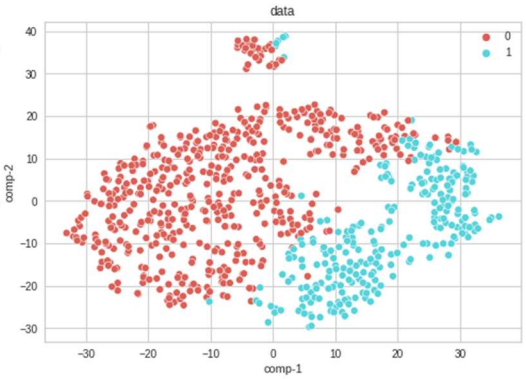


Figure 14 Data with optimal number of clusters

3.

```
def PCA_models(model,x_train,y_train,X_test,y_test,label):
  models = []
  values= []
  model.fit(x_train, y_train)
  y_pred=model.predict(X_test)
  acc = accuracy_score(y_test,y_pred)
  models.append('baseline')
  values.append(acc*100)
  max=0
  for i in range(1,9):
    pca = PCA(n_components=i,random_state=0)
    pca.fit(x_train)
    # apply transform to dataset
    transformed = pca.transform(x_train)
    model.fit(transformed, y_train)
    test_transformed = pca.transform(X_test)
    y_pred=model.predict(test_transformed)
    acc = accuracy_score(y_test,y_pred)
    models.append(i)
    values.append(acc*100)
    if max<acc*100:
      max=acc*100
      maxid=i
      best_x_train = transformed
      best_x_test=test_transformed
      best_y_pred=y_pred
      best_y_train=model.predict(transformed)

  fig = plt.figure(figsize=(15, 10))
  fig.suptitle(label)
  ax=sns.barplot(x=models, y=values,color='b')
  plt.xlabel("Number of Components")
```

```
  plt.ylabel("Test Accuracy %")
  ax.set(ylim=(60, 85))
  plt.show()

  print("the best number of components in PCA with the highest accuracy
{} with accuracy {} % ".format(maxid,max))

  tsne(best_x_train,best_y_train,2,'train dataset')
  tsne(best_x_test,best_y_pred,2,'test dataset')
  return maxid,max
```

A & B &C)

```
LR = LogisticRegression()
best_PCA_ind_LR,best_PCA_LR=PCA_models(LR,x_train,y_train,x_test,y_test,
'PCA With Logistic Regression ')
```

Figure 15 Best value for N-components; LR

```
the best number of components in PCA with the highest accuracy 7 with accuracy 78.64583333333334 %
```

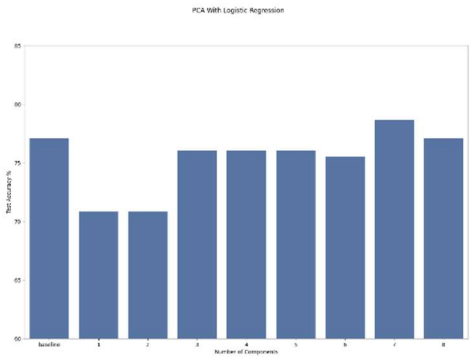


Figure 18 Bar chart for LR

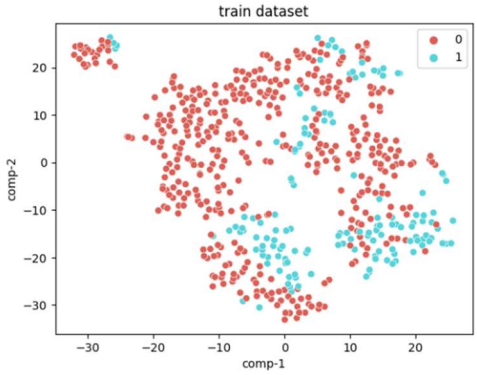


Figure 17 T-SNE; train; LR

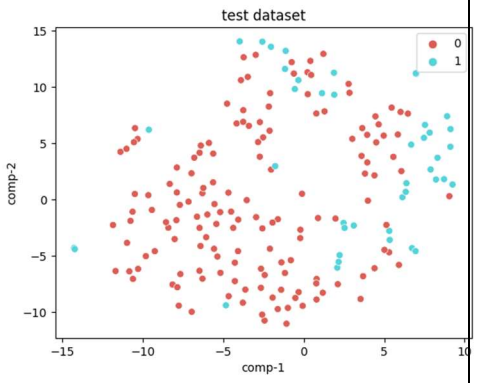


Figure 16 T-SNE; Test; LR

```
KNN=KNeighborsClassifier()
best_PCA_ind_KNN,best_PCA_KNN=PCA_models(KNN,x_train,y_train,x_test,y_te
st,'PCA with KNN')
```

Figure 19 Best value for N-components; KNN

```
the best number of components in PCA with the highest accuracy 4 with accuracy 76.04166666666666 %
```

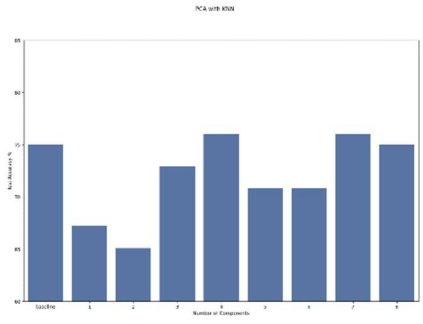


Figure 22 Bar chart for KNN

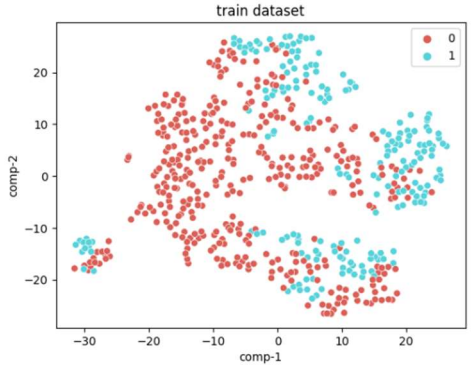


Figure 21 T-SNE; train; KNN

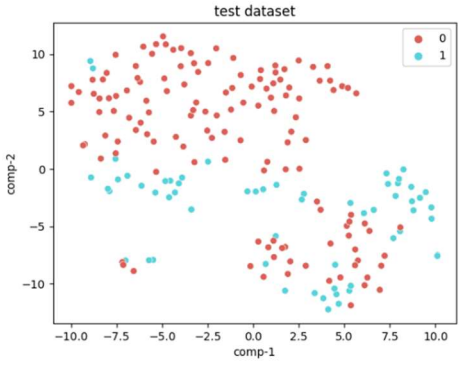


Figure 20 T-SNE; Test; KNN

4.

A&B)

```
sns.set()
def filter_methods(model,x_train,y_train,X_test,y_test,typ,old_acc,visul
aization=False):
  if typ=='IG':
    models_acc={}
    feature_imp=mutual_info_classif(x_train,y_train,random_state=0)
    df=pd.Series(feature_imp)
    df.plot(kind='bar',color='teal')
    plt.show()
    feature_sorted=(-feature_imp).argsort()
    best_features=0
    m=0
    for i in range(1,9):
      best_ind=feature_sorted[:i]
      model.fit(x_train.iloc[:,best_ind], y_train)
      y_pred=model.predict(x_test.iloc[:,best_ind])
      acc = accuracy_score(y_test,y_pred)
      models_acc[i]=acc*100
      if  models_acc[i]>m:
        best_features=best_ind
        m=models_acc[i]
    best=max(models_acc,key=models_acc.get)
    models_acc['baseLine']=old_acc
    fig = plt.figure(figsize=(15, 10))
    fig.suptitle(str(model)+' with Information Gain features selection '
)
    ax=sns.barplot(x=list(models_acc.keys()), y=[models_acc[i] for i in
models_acc.keys()],color='b')
    plt.xlabel("Number of features")
    plt.ylabel("Test Accuracy %")
    ax.set(ylim=(70, 80))
    plt.show()
    print("the best number of features is {} and there are {} it achive
{} % ".format(best,best_features,models_acc[best]))
    return best_features,models_acc[best]

  elif typ=='VT':
    models_acc={}
    varience=x_train.var()

    varience=varience.sort_values(ascending=False)
```

```
    for i in range(1,9):
      if i != 8:
        VT=VarianceThreshold(threshold=varience[i])
      else:
        VT=VarianceThreshold(0)

      x_ranformed=VT.fit_transform(x_train)

      model.fit(x_ranformed, y_train)
      x_test_tranformed=VT.transform(x_test)
      y_pred=model.predict(x_test_tranformed)
      acc = accuracy_score(y_test,y_pred)
      models_acc[i]=acc*100
    best=max(models_acc,key=models_acc.get)
    models_acc['baseLine']=old_acc
    fig = plt.figure(figsize=(15, 10))
    fig.suptitle(str(model)+' with Variance Threshold')
    ax=sns.barplot(x=list(models_acc.keys()), y=[models_acc[i] for i in
models_acc.keys()],color='b')
    plt.xlabel("Number of features")
    plt.ylabel("Test Accuracy %")
    ax.set(ylim=(50, 80))
    plt.show()



    if best != 8:
      VT=VarianceThreshold(threshold=varience[best])
    else:
      VT=VarianceThreshold(0)
    x_tranformed=VT.fit_transform(x_train)


    print("the best number of features is {} and there are {} it achive
{} % ".format(best,VT.get_feature_names_out(),models_acc[best]))

    return best,models_acc[best]
```

```
def Wrapper_methods(model,x_train,y_train,X_test,y_test,typ,old_acc):
  if typ=='FE':
    models_acc={}
    for i in range(1,8):
      ffs=SFS(model,n_features_to_select=i,direction ='forward')
      ffs.fit(x_train, y_train)

      model.fit(x_train.iloc[:,ffs.get_support()],y_train)
      y_pred=model.predict(x_test.iloc[:,ffs.get_support()])
      acc = accuracy_score(y_test,y_pred)
      models_acc[i]=acc*100
    best=max(models_acc,key=models_acc.get)
    models_acc['baseLine']=old_acc
    fig = plt.figure(figsize=(15, 10))
    fig.suptitle(str(model)+' with Forward Feature Elimination')
    ax=sns.barplot(x=list(models_acc.keys()), y=[models_acc[i] for i in
models_acc.keys()],color='b')
    plt.xlabel("Number of features")
    plt.ylabel("Test Accuracy %")
    ax.set(ylim=(60, 85))
    plt.show()


    ffs=SFS(model,n_features_to_select=best,direction ='forward')
    ffs.fit(x_train, y_train)
    print("the best number of features is {} and there are {} it achive
{} % ".format(best,x_train.columns[ffs.get_support()],models_acc[best]))

    return ffs.get_support(),models_acc[best]
  elif typ=='BE':
    models_acc={}
    for i in range(1,8):
      ffs=SFS(model,n_features_to_select=i,direction ='backward')
      ffs.fit(x_train, y_train)

      model.fit(x_train.iloc[:,ffs.get_support()],y_train)
      y_pred=model.predict(x_test.iloc[:,ffs.get_support()])
      acc = accuracy_score(y_test,y_pred)
      models_acc[i]=acc*100
    best=max(models_acc,key=models_acc.get)
    models_acc['baseLine']=old_acc
    fig = plt.figure(figsize=(15, 10))
    fig.suptitle(str(model)+' with Backword Feature Elimination')
    ax=sns.barplot(x=list(models_acc.keys()), y=[models_acc[i] for i in
models_acc.keys()],color='b')
```

```
    plt.xlabel("Number of features")
    plt.ylabel("Test Accuracy %")
    ax.set(ylim=(60, 85))
    plt.show()


    ffs=SFS(model,n_features_to_select=best,direction ='backward')
    ffs.fit(x_train, y_train)

    print("the best number of features is {} and there are {} it achive
{} % ".format(best,x_train.columns[ffs.get_support()],models_acc[best]))

    return ffs.get_support(),models_acc[best]
  elif typ=='RFE':

    models_acc={}
    for i in range(1,9):
      rfe=RFE(model,n_features_to_select=i)
      rfe.fit(x_train, y_train)
      model.fit(x_train.iloc[:,rfe.support_],y_train)
      y_pred=model.predict(x_test.iloc[:,rfe.support_])
      acc = accuracy_score(y_test,y_pred)

      models_acc[i]=acc*100
    best=max(models_acc,key=models_acc.get)
    models_acc['baseLine']=old_acc
    fig = plt.figure(figsize=(15, 10))
    fig.suptitle(str(model)+' with , Recursive Feature Elimination')
    ax=sns.barplot(x=list(models_acc.keys()), y=[models_acc[i] for i in
models_acc.keys()],color='b')
    plt.xlabel("Number of features")
    plt.ylabel("Test Accuracy %")
    ax.set(ylim=(60, 85))
    plt.show()


    rfe=RFE(model,n_features_to_select=best)
    rfe.fit(x_train, y_train)

    print("the best number of features is {} and there are {} it achive
{} % ".format(best,np.array(x_train.columns)[rfe.support_],models_acc[be
st]))

    return rfe.support_,models_acc[best]
```

Logistic Regression:

Filter methods:

Information gain:

```
LR = LogisticRegression()
best_LR_IG_ind,best_IG_LR=filter_methods(LR,x_train,y_train,x_test,y_test,'IG',best_PCA_LR)
```

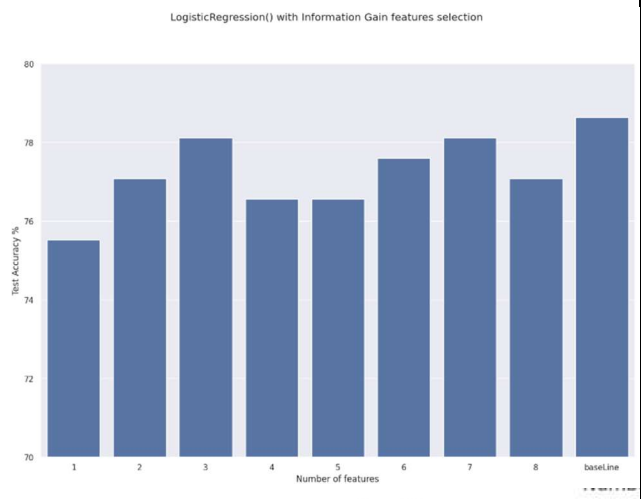


Figure 24 Number of features versus accuracy graph of Information Gain - LR

Figure 23 Best number of features and accuracy of Information Gain – LR – (Less than baseline)

```
the best number of features is 3 and there are [1 5 7] it achive 78.125 %
```

Variance Threshold:

```
LR = LogisticRegression()
best_VT_LR_ind,best_VT_LR=filter_methods(LR,x_train,y_train,x_test,y_test,'VT',best_
PCA_LR)
```

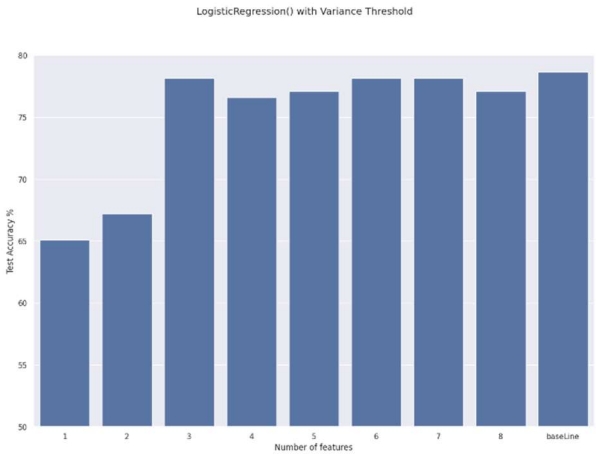


Figure 26 Number of features versus accuracy graph of Variance Threshold - LR

Figure 25 Best number of features and accuracy of Variance Threshold – LR - (Less than baseline)

```
the best number of features is 3 and there are Index(['Glucose', 'BMI', 'Age'], dtype='object') it achive 78.125 %
```

Wrapper methods:

Forward feature elimination:

```
LR = LogisticRegression()
best_RFE_LR_ind,best_RFE_LR=Wrapper_methods(LR,x_train,y_train,x_test,y
_test,'FE',best_PCA_LR)
```

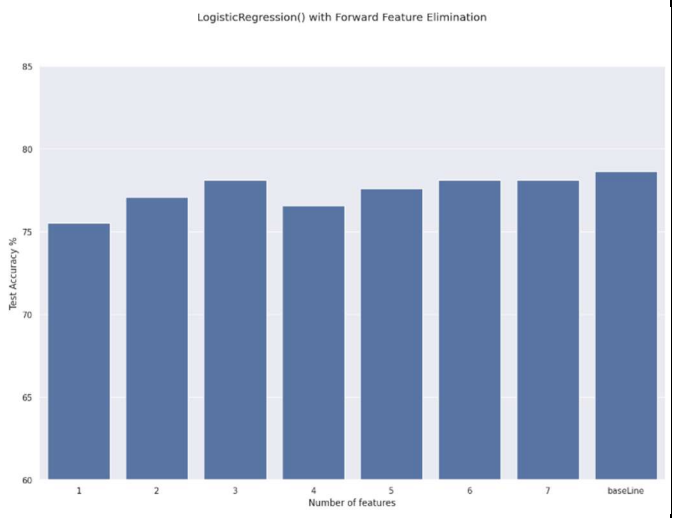


Figure 27 Number of features versus accuracy graph of Forward Elimination - LR

Figure 28Number of features versus accuracy graph of Forward Elimination – LR - (Less than baseline)

```
the best number of features is 3 and there are Index(['Glucose', 'BMI', 'Age'], dtype='object') it achive 78.125 %
```

Backward Feature Elimination:

```
LR = LogisticRegression()
best_RFE_LR_ind,best_RFE_LR=Wrapper_methods(LR,x_train,y_train,x_test,y
_test,'BE',best_PCA_LR)
```

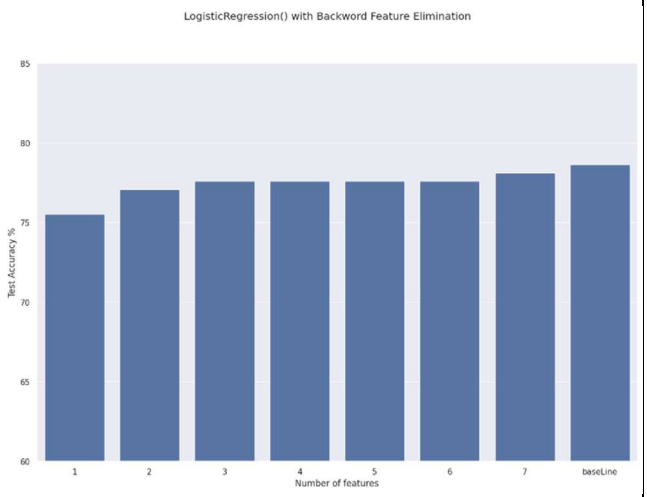


Figure 30 Number of features versus accuracy graph of Backward Elimination - LR

Figure 29 Number of features versus accuracy graph of Backward Elimination – LR - (Less than baseline)

```
the best number of features is 7 and there are Index(['Pregnancies', 'Glucose', 'BloodPressure', 'Insulin', 'BMI',
       'DiabetesPedigreeFunction', 'Age'],
      dtype='object') it achive 78.125 %
```

Recursive Feature Elimination

```
LR = LogisticRegression()
best_RFE_LR_ind,best_RFE_LR=Wrapper_methods(LR,x_train,y_train,x_test,y
_test,'RFE',be
st_PCA_LR)
```

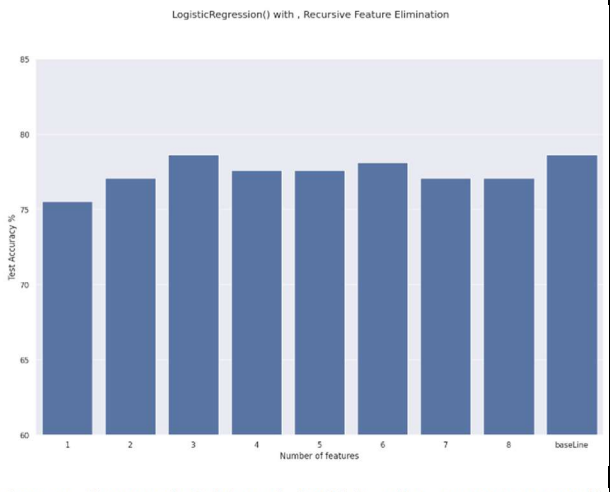


Figure 32 Number of features versus accuracy graph of Recursive Feature Elimination - LR

Figure 31 Best number of features and accuracy of Recursive Feature Elimination – LR - (Equals to baseline; has the best accuracy)

```
the best number of features is 3 and there are ['Pregnancies' 'Glucose' 'BMI'] it achive 78.64583333333334 %
```

K-Nearest Neighbors:

Filter Methods:

Information Gain:

```
KNN=KNeighborsClassifier()
best_ind_IG_KNN,best_IG_KNN=filter_methods(KNN,x_train,y_train,x_test,y
_test,'IG',best_PCA_KNN)
```

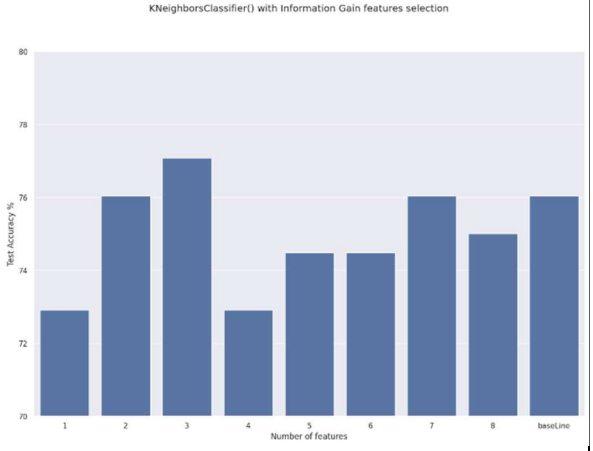


Figure 34 Number of features versus accuracy graph of Information Gain - KNN

Figure 33 Best number of features and accuracy of Information Gain – KNN - (Higher than baseline)

```
the best number of features is 3 and there are [1 5 7] it achive 77.08333333333334 %
```

Variance Threshold:

```
KNN=KNeighborsClassifier()
best_VT_KNN_ind,best_VT_KNN=filter_methods(KNN,x_train,y_train,x_test,y
_test,'VT',best_
PCA_KNN)
```

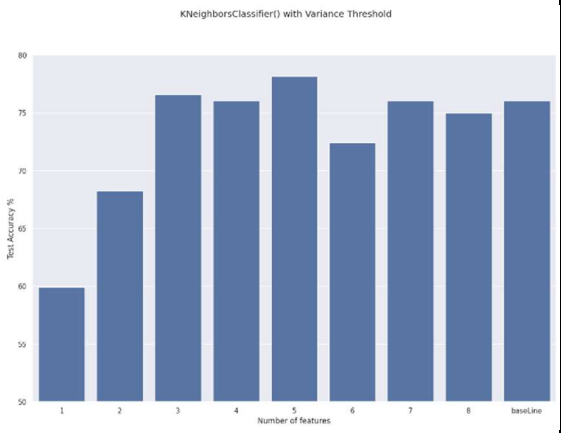


Figure 36 Number of features versus accuracy graph of Variance Threshold - KNN

```
the best number of features is 5 and there are ['Pregnancies' 'Glucose' 'BloodPressure' 'SkinThickness' 'Age'] it achive 78.125 %
```

Figure 35 Best number of features and accuracy of Variance Threshold – KNN - (Higher than baseline)

Wrapper Methods:

Forward Feature Elimination:

```
KNN = KNeighborsClassifier()
best_FE_kNN_ind,best_FE_kNN=Wrapper_methods(KNN,x_train,y_train,x_test,
y_test,'FE',best_PCA_KNN)
```

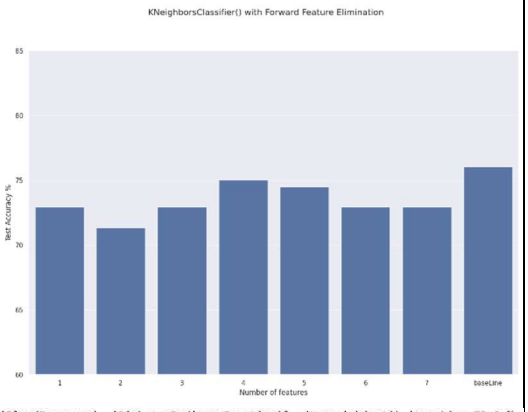


Figure 38 Number of features versus accuracy graph of Forward Elimination - KNN

```
the best number of features is 4 and there are Index(['Pregnancies', 'Glucose', 'BloodPressure', 'DiabetesPedigreeFunction'], dtype='object') it achive 75.0 %
```

Figure 37 Best number of features and accuracy of Forward Elimination – KNN - (Less than baseline)

Backward Feature Elimination:

```
KNN = KNeighborsClassifier()
best_BE_kNN_ind,best_BE_kNN=Wrapper_methods(KNN,x_train,y_train,x_test
,y_test,'BE',best
_PCA_KNN)
```

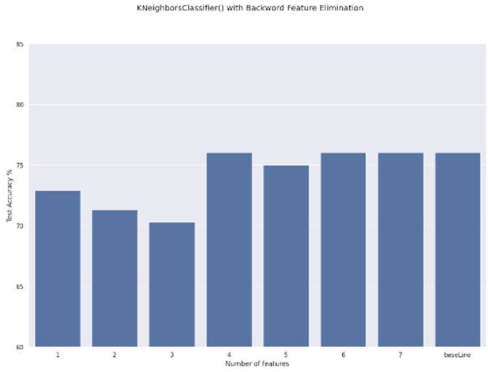


Figure 40 Number of features versus accuracy graph of Backward Elimination - KNN

Figure 39 Best number of features and accuracy of Backward Elimination - KNN - (Equals to baseline)

N.B: it isn't possible to use k-nearest neighbors algorithm with recursive feature elimination because k-nearest neighbors algorithm does not provide information about feature importance (e.g. coef_, feature_importances_).

C)

In this part we chose the best methods for each classifier; Recursive feature elimination (3 features) and variance threshold (5 features) for logistic regression and K-nearest neighbors respectively as they have achieved the highest test accuracies for each classifier.

Logistic Regression (Recursive Feature Elimination):

```
LR.fit(x_train.iloc[:,best_RFE_LR_ind],y_train)
y_pred_train=LR.predict(x_train.iloc[:,best_RFE_LR_ind])
y_pred=LR.predict(x_test.iloc[:,best_RFE_LR_ind])
tsne(x_train.iloc[:,best_RFE_LR_ind],y_pred_train,2,'train dataset')
tsne(x_test.iloc[:,best_RFE_LR_ind],y_pred,2,'test dataset')
```

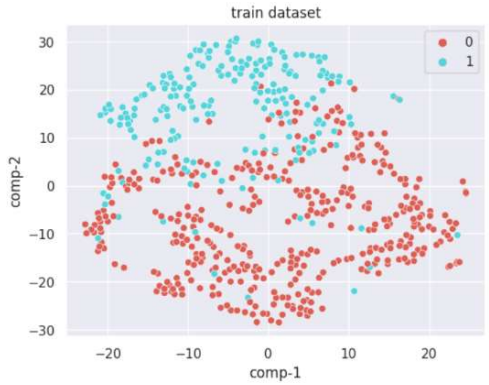


Figure 41 Recursive Feature Elimination of LR - Train Dataset

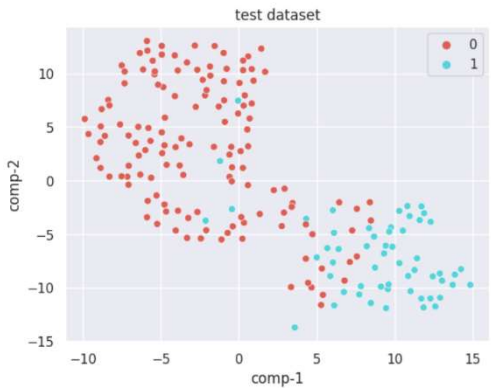


Figure 42 Recursive Feature Elimination of LR - Test Dataset

K-Nearest Neighbor (Variance Threshold):

```
varience=x_train.var()
varience=varience.sort_values(ascending=False)
if best_VT_KNN_ind != 8:
  VT=VarianceThreshold(threshold=varience[best_VT_KNN_ind])
else:
  VT=VarianceThreshold(0)
x_tranformed=VT.fit_transform(x_train)
best_features=VT.get_support()
model.fit(x_train.iloc[:,best_features], y_train)
y_pred=model.predict(x_test.iloc[:,best_features])
y_pred_train=model.predict(x_train.iloc[:,best_features])
tsne(x_train.iloc[:,best_features],y_pred_train,2,'train dataset')
tsne(x_test.iloc[:,best_features],y_pred,2,'test dataset')
```

Figure 43 Variance Threshold of KNN - Train Dataset

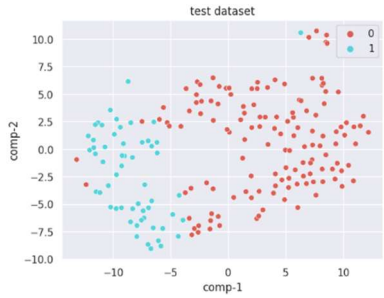


Figure 44 Variance Threshold of KNN - Test Dataset

5.

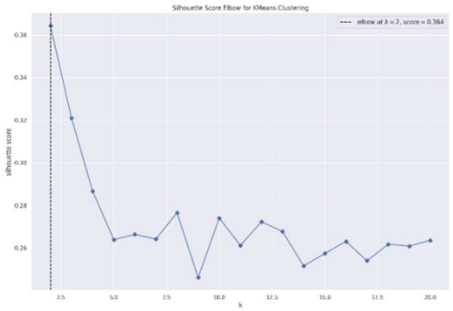
we chose Logistic regression with Recursive feature elimination instead of PCA while both had the same highest-test (78.65%) accuracy as it has less complexity than PCA because it only uses 3 features while PCA uses 7 features.

A)

```
df_RFE=X.iloc[:,best_RFE_LR_ind]
transformed=np.array(df_RFE)
```

```
from yellowbrick.cluster import KElbowVisualizer
fig = plt.figure(figsize=(15, 10))
Elbow_M = KElbowVisualizer(KMeans(), k=20,metric='silhouette',timi
ngs=False)
Elbow_M.fit(transformed)
Elbow_M.show()
```

Figure 45 silhouette score vs the number of clusters of K-means



B)

Optimal number of clusters (K) Equals (2)

6.

```
!pip install minisom
from minisom import MiniSom
```

A)

```
s=[]
for i in range(2,31):
  som = MiniSom(1,i,3 , sigma=0.3, learning_rate=0.5,random_seed=0) # in
itialization of 6x6 SOM
  som.train_batch(transformed, 1500)# trains the SOM with 1500 iteration
s
  # each neuron represents a cluster
  winner_coordinates = np.array([som.winner(x) for x in transformed]).T

  cluster_index = np.ravel_multi_index(winner_coordinates, (1,i))

  score = silhouette_score(transformed, cluster_index, random_state=0)
  s.append(score)
print(s)
plt.figure(figsize=(25, 10))
g=sns.lineplot(range(2,31),s)
g.set_xlabel('Number of Neurons’)
g.set_ylabel('silhouette')
plt.show()
plt.show()

  # with np.ravel_multi_index we convert the bidimensional
  # coordinates to a monodimensional index

  #print(som.get_weights())
```

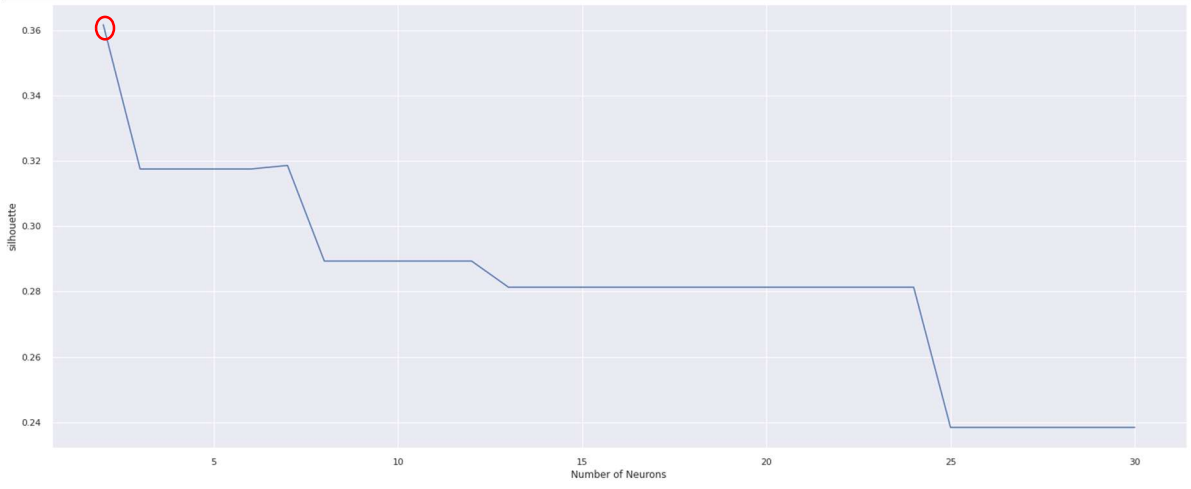


Figure 46 Silhouette score vs the number of neurons

B)

In the previous silhouette score plot, the number of neurons that showed the highest value was “2”.

C)

```
som = MiniSom(1,2,3 , sigma=0.3, learning_rate=0.5,random_seed=0) # init
ialization of 6x6 SOM
plt.figure(figsize=(15, 10))
winner_coordinates = np.array([som.winner(x) for x in transformed]).T

cluster_index = np.ravel_multi_index(winner_coordinates, (1,2))
# plotting the clusters using the first 2 dimentions of the data
# Creating figure
plt.figure(figsize=(25, 10))

# plotting the clusters using the first 2 dimentions of the data
for c in np.unique(cluster_index):
    plt.scatter(transformed[cluster_index == c, 0],
                transformed[cluster_index == c, 1], label='cluster='+str
(c), alpha=.7)

# plotting centroids
for centroid in som.get_weights():
    plt.scatter(centroid[:, 0], centroid[:, 1], marker='x',
                s=80, linewidths=35, color='k', label='centroid')
plt.legend();
```

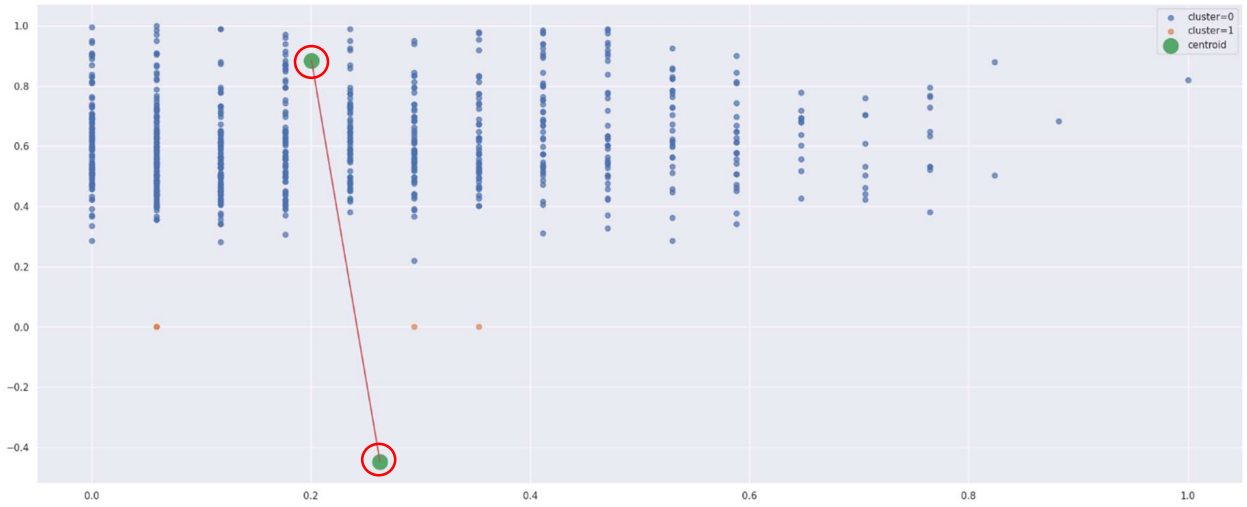


Figure 47 Initial Neurons' positions with 2 dimensions only in 2D

N.B: Neurons’ positions are marked with the red circles.

```
som = MiniSom(1,2,3 , sigma=0.3, learning_rate=0.5,random_seed=0) # init
ialization of 6x6 SOM
plt.figure(figsize=(15, 10))
winner_coordinates = np.array([som.winner(x) for x in transformed]).T

cluster_index = np.ravel_multi_index(winner_coordinates, (1,2))
# plotting the clusters using the first 2 dimentions of the data
# Creating figure
fig = plt.figure(figsize = (10, 7))
ax = plt.axes(projection ="3d")
for c in np.unique(cluster_index):
   ax.scatter3D(transformed[cluster_index == c, 0],
                transformed[cluster_index == c, 1],transformed[cluster_i
ndex == c, 2], label='cluster='+str(c), alpha=.7)
print(som.get_weights())
# plotting centroids
for centroid in som.get_weights():
    ax.scatter3D(centroid[:, 0], centroid[:, 1],centroid[:, 2], marker='
x',
                 s=300, linewidths=35,  label='centroid')
    ax.plot3D(centroid[:, 0], centroid[:, 1],centroid[:, 2],color='r')
plt.legend();
```

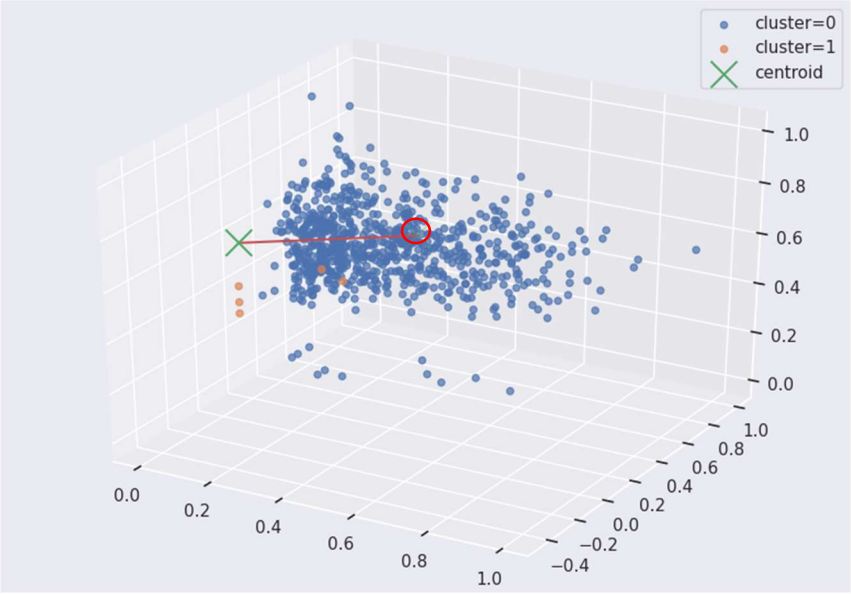


Figure 48 Initial Neurons' positions with 2 dimensions only in 3D

```
som.train_batch(transformed, 1500) # trains the SOM with 100 iterations
# each neuron represents a cluster
winner_coordinates = np.array([som.winner(x) for x in transformed]).T
cluster_index = np.ravel_multi_index(winner_coordinates, (1,2))
score = silhouette_score(z, cluster_index, random_state=0)
```

```
plt.figure(figsize=(15, 10))
winner_coordinates = np.array([som.winner(x) for x in transformed]).T

cluster_index = np.ravel_multi_index(winner_coordinates, (1,2))
# plotting the clusters using the first 2 dimentions of the data
# Creating figure
plt.figure(figsize=(25, 10))

# plotting the clusters using the first 2 dimentions of the data
for c in np.unique(cluster_index):
    plt.scatter(transformed[cluster_index == c, 0],
                transformed[cluster_index == c, 1], label='cluster='+str
(c), alpha=.7)

# plotting centroids
for centroid in som.get_weights():
    plt.scatter(centroid[:, 0], centroid[:, 1],
                s=300, label='centroid')
plt.plot(centroid[:, 0], centroid[:, 1],color='r')
plt.legend();
```

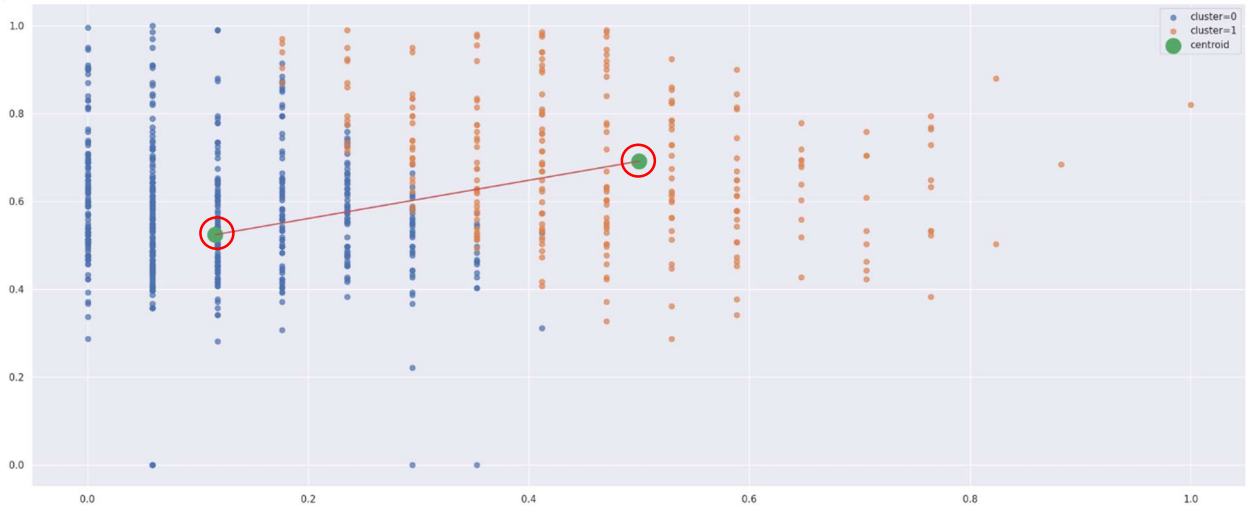


Figure 49 Final Neurons' positions with 2 dimensions only in 2D

```
fig = plt.figure(figsize = (10, 7))
ax = plt.axes(projection ="3d")
for c in np.unique(cluster_index):

   ax.scatter3D(transformed[cluster_index == c, 0],
                transformed[cluster_index == c, 1],transformed[cluster_i
ndex == c, 2], label='cluster='+str(c), alpha=.7)
print(som.get_weights())
# plotting centroids
for centroid in som.get_weights():
    ax.scatter3D(centroid[:, 0], centroid[:, 1],centroid[:, 2],
                s=300,  label='centroid')
ax.plot3D(centroid[:, 0], centroid[:, 1],centroid[:, 2],color='r')
```

Figure 50 Final Neurons' positions with 2 dimensions only in 3D

7.

```
epsList, msList, accList, clusterList = list(), list(), list(), list()

for eps in tqdm(np.arange(0.3, 0.75, 0.05)):
  for ms in range(2, 16):
    db = DBSCAN(eps=eps, min_samples=ms)
    predLabels = db.fit_predict(X)
    n_clusters = len(set(db.labels_)) - (1 if -1 in db.labels_ else 0)
    if n_clusters > 1:
      score = silhouette_score(X, predLabels, random_state=0)
    else:
      score = 'None'
    epsList.append(eps)
    msList.append(ms)
    accList.append(score)
    clusterList.append(n_clusters)

db_data = {'epsilon': epsList, 'minPoints': msList, 'Silhouette': accLis
t,  'Number Of Clusters': clusterList}

df_db = pd.DataFrame(db_data)
df_db_2c = df_db[df_db['Number Of Clusters'] == 2]

# Getting 10 combinations with the highest Silhouette score
for i in range (df_db_2c.shape[0], 10, -1):
  df_db_2c = df_db_2c[df_db_2c.Silhouette != df_db_2c.Silhouette.min()]

# Printing the ten combinations
print(df_db_2c)
# Plotting epsilon versus number of clusters
plt.figure(figsize=(10, 5))
plt.plot(list(range(1,11)), df_db_2c['Number Of Clusters'], marker='o')
plt.xlabel('Epsilon', fontsize=14)
plt.ylabel('Number of clusters', fontsize=14)
plt.title('Epsilon vs Number of clusters', fontsize=14)
plt.xticks(list(range(1,11)),labels=df_db_2c.epsilon.apply(lambda x:str(
round(x,2))))
plt.yticks(df_db_2c['Number Of Clusters']);

# Plotting minpoints versus number of clusters
plt.figure(figsize=(10, 5))
plt.plot(list(range(1,11)), df_db_2c['Number Of Clusters'], marker='o')
plt.xlabel('minPoints', fontsize=14)
plt.ylabel('Number of clusters', fontsize=14)
plt.title('MinPoints vs Number of clusters', fontsize=14)
```

```
plt.xticks(list(range(1,11)),labels=df_db_2c.minPoints)
plt.yticks(df_db_2c['Number Of Clusters']);
```

```
100%|██████████| 9/9 [00:06<00:00,  1.35it/s]
    epsilon  minPoints Silhouette  Number Of Clusters
5      0.30          7    0.25569                   2
6      0.30          8   0.255465                   2
7      0.30          9   0.241373                   2
23     0.35         11   0.273477                   2
24     0.35         12   0.266164                   2
25     0.35         13   0.256723                   2
26     0.35         14   0.253778                   2
27     0.35         15   0.251914                   2
28     0.40          2   0.286938                   2
42     0.45          2   0.427027                   2
```

Figure 51 The ten combinations of epsilon and minpoints that made two clusters with highest silhouette score

A)

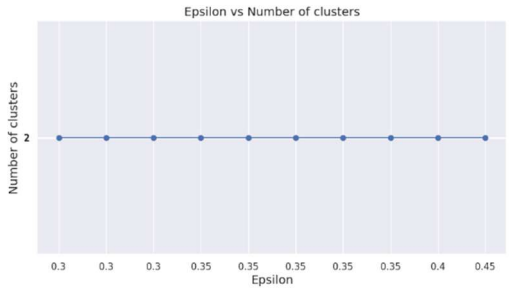


Figure 52 Epsilon vs number of clusters

B)

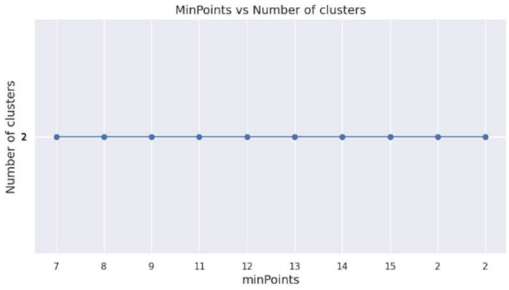


Figure 53 Minpoints vs number of clusters

In the two previous graphs each epsilon in (A) combines with a minimum number of points in (B) both forming the number of clusters which is 2. i.e. an epsilon = 0.3 with min points of 7 or 8 or 9 will result a number of 2 clusters.

8. Conclusion

A)

In Question 2 the silhouette score was 0.261 while it was 0.364 in Question 5, so there is a significant increase in the chosen model in Question 5 (logistic regression with Recursive Feature Elimination). The optimal value of clusters in both questions was K=2.

B)

Regarding the LR model, in Q1 and Q3, the two classes (both in training and testing datasets) are overlapping while in Q4 the two classes are not overlapping (can be separated). We can conclude the reason for that because in Q4 the data dimensions are 3 while in Q1 and Q3 the data dimensions are 8 and 7 respectively.

Regarding KNN model, in Q1 the two classes (both in training and testing datasets) are overlapping while in Q3 the overlapping between classes is decreased and in Q4 the two classes are not overlapping (can be separated). We can conclude the reason of that because in Q1 the data dimensions are 8 while in Q3 and Q4 the data dimensions are 4 and 5 respectively.

As a result, we can conclude that, in general using feature selection decreases the overlapping between classes significantly and independent on the number of used features (using 3 or 5 features will reduce overlapping equally) as in Q4, while using PCA decreases slightly the overlapping but depends on how many features are used (using less features will make less overlapping) as in Q3.