MIDDLE EAST TECHNICAL UNIVERSITY
NORTHERN CYPRUS CAMPUS

CNG 562
MACHINE LEARNING

Assignment-3

Report

Nisa Nur Odabaş
Kaan Taha Köken

June 2, 2020

Contents

List of Figures

1 Introduction

In this assignment, our aim is to combine methodologies learned earlier such as preprocessing and validation techniques with new classification methods, which are K-Means and DBScan. We are comparing silhouette score of two classifiers, and we will calculate the percentage of error that gave the best silhouette score among them.

1.1 Dataset

We are using Breast Cancer dataset for the take home exam assignment. It contains 2 classes, 569 instances with 30 features. We can understand by looking its shape. This is the shape of our data (569, 30).

In order to understand and get better perspective from our data. We project our data graph, and see how it looks.

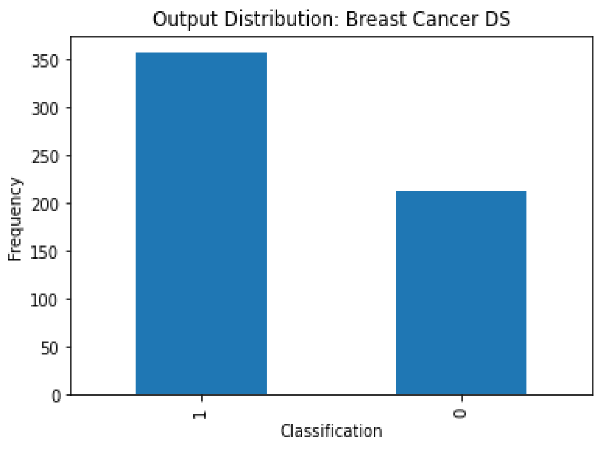
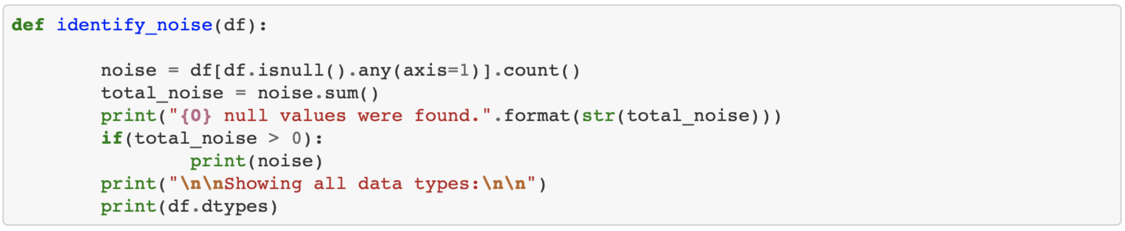


Figure 1: Data visualization

1.2 Data Cleaning

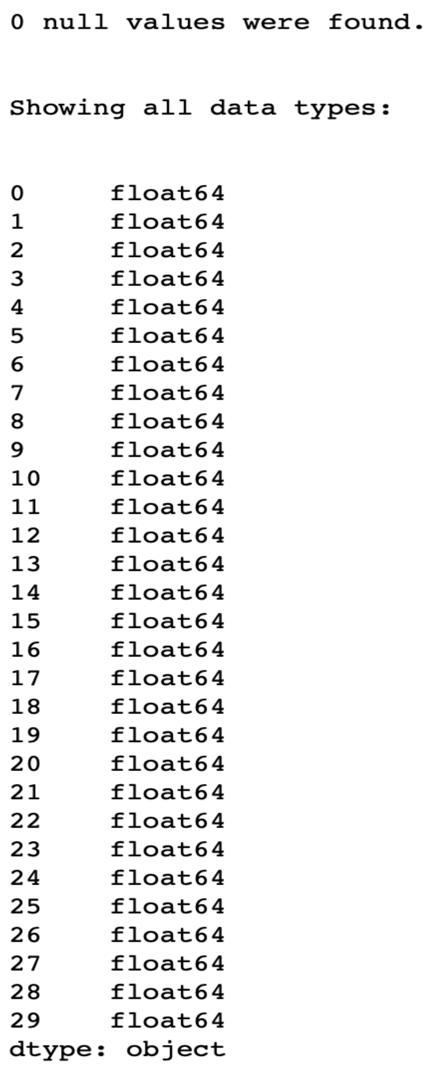
I conduct my experiments in two parts, one with raw data and one with cleaned data. In order to clean my data, I came up with some methods. First, I tried to identify the noisy data inside the data set.



```
def identify_noise(df):

        noise = df[df.isnull().any(axis=1)].count()
        total_noise = noise.sum()
        print("{0} null values were found.".format(str(total_noise)))
        if(total_noise > 0):
                print(noise)
        print("\n\nShowing all data types:\n\n")
        print(df.dtypes)
```

Figure 2: Identify Noise Method

As a result, as we can see result below, there is no noisy data, and we can also see the data types of features.



```
0 null values were found.


Showing all data types:


0      float64
1      float64
2      float64
3      float64
4      float64
5      float64
6      float64
7      float64
8      float64
9      float64
10     float64
11     float64
12     float64
13     float64
14     float64
15     float64
16     float64
17     float64
18     float64
19     float64
20     float64
21     float64
22     float64
23     float64
24     float64
25     float64
26     float64
27     float64
28     float64
29     float64
dtype: object
```

Figure 3: Identify Noise Method

To see correlation in the data, I created a method, and visualize the data as a heat map. As we can see in below, the feature pairs that have a white color are highly correlated. Therefore, correlation will cause a problem for our models. For example, feature pairs (0, 2), (0, 3), and (2, 3) are all highly correlated. If we do not removed them, models will be highly correlated.

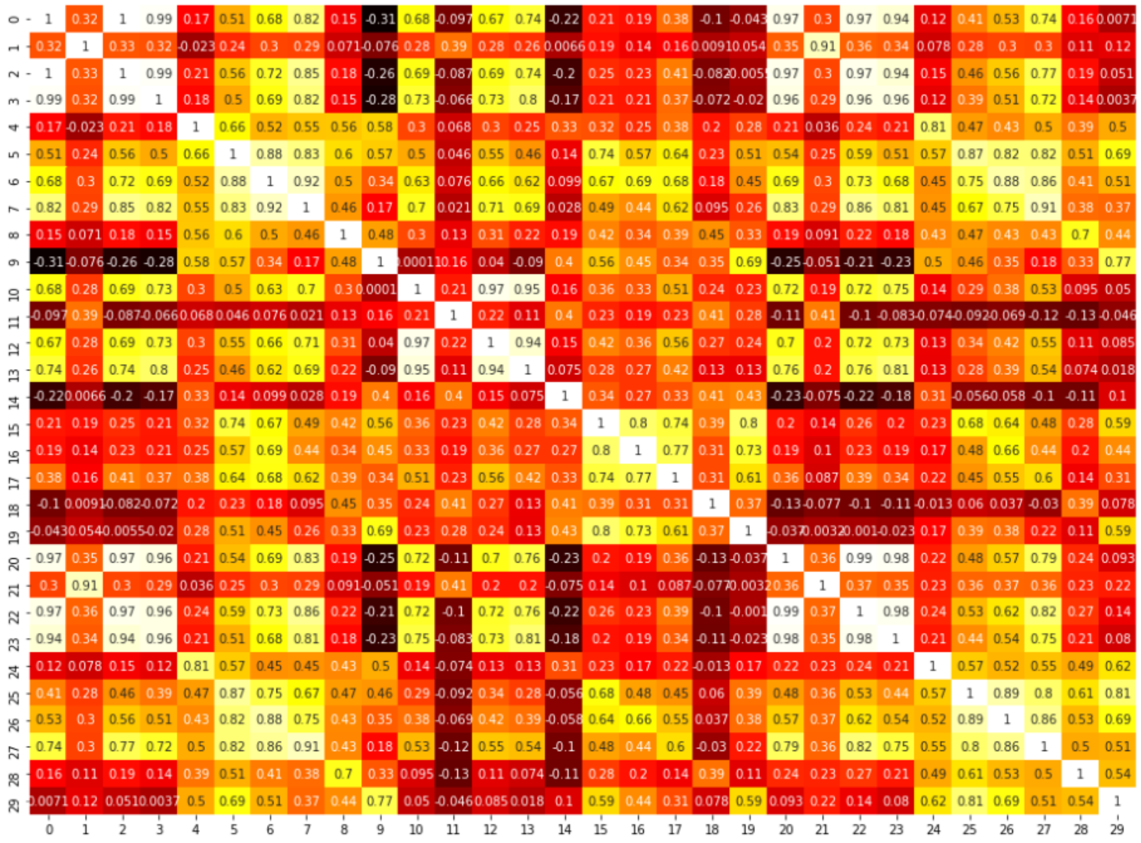


Figure 4: Heat map

```
def heat_map(df):
        fig, ax = plt.subplots()
        corr = df.corr()
        sns.heatmap(corr, annot=True, cmap='hot')
        plt.show()
```

```
def filter_features(data, bad_indices):
        # eliminate above column indices from the data and return new set
        filtered_data = np.delete(data, bad_indices, axis=1)

        return filtered_data
```

Figure 5: Heat Map and Filter Code

After done some cleaning using code above, again, we need to look graphs if there is a distinct difference between labels. For example, feature 7, looks like have distinct difference, so in the future, it will not be useful. Therefore, we need to remove it as well.

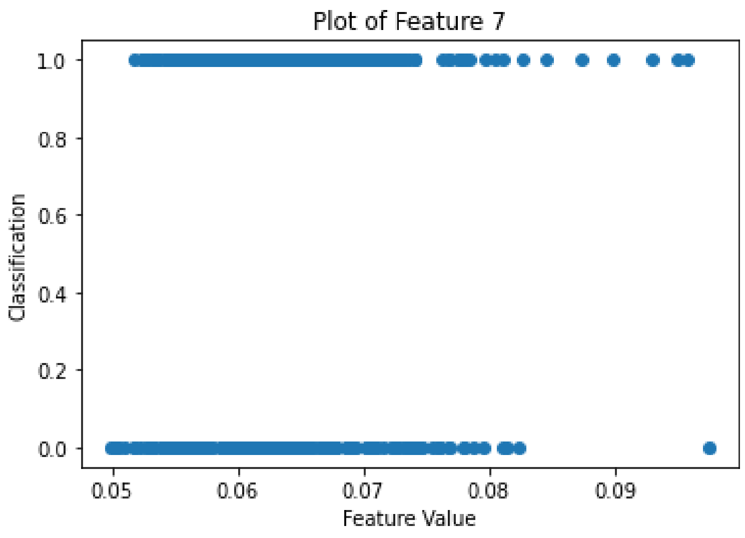


Figure 6: Feature 7

After the last cleaning, we have remained 14 features. Using the pairplot functionality, I display the data remain.

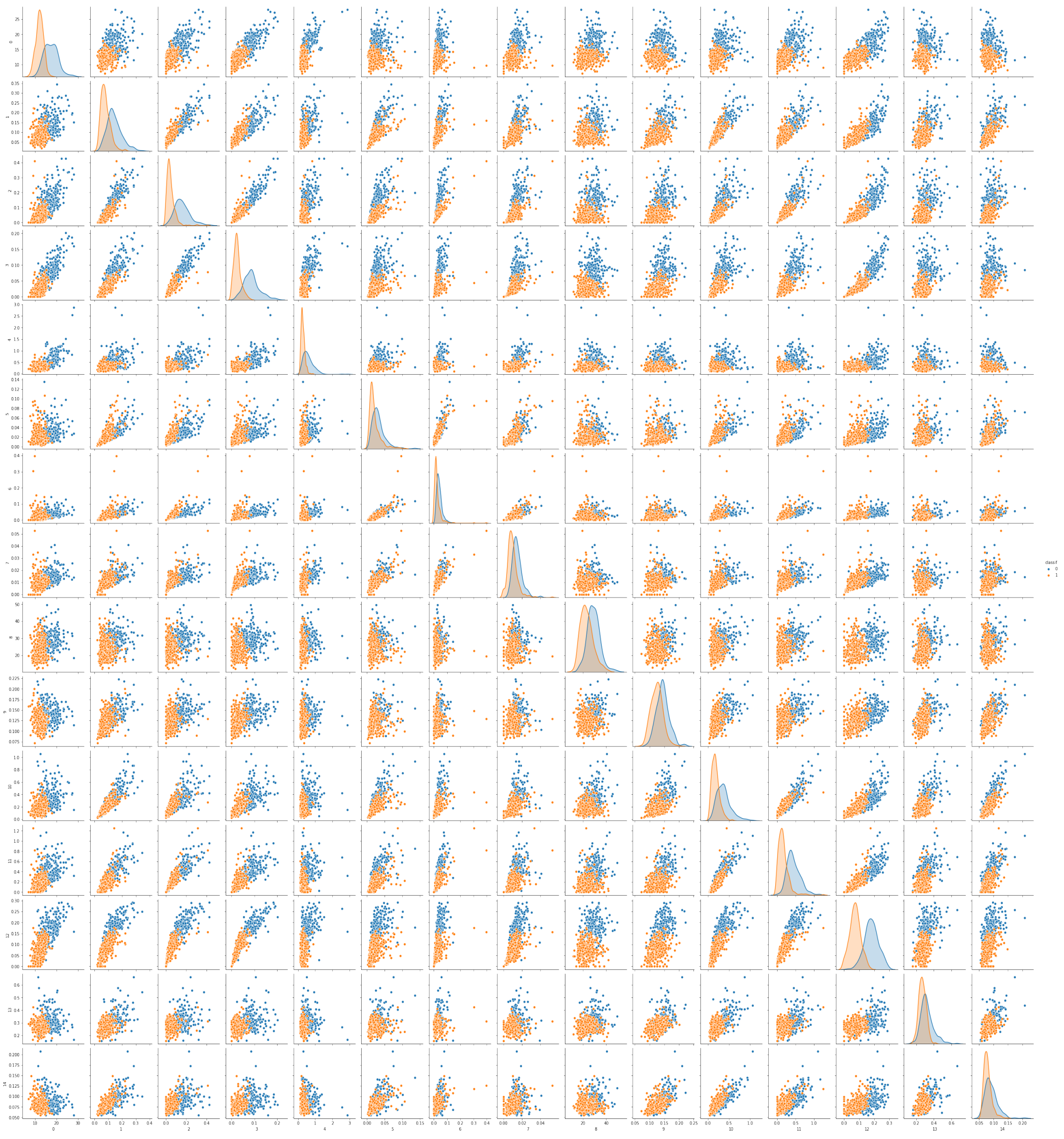


Figure 7: Pair plot

1.3 Silhouette Score

Silhouette analysis can be used to study the separation distance between the resulting clusters. The silhouette plot displays a measure of how close each point in one cluster is to points in the neighboring clusters and thus provides a way to assess parameters like number of clusters visually.

2 K-Means

To process the learning data, the K-means algorithm starts with a first group of randomly selected centroids, which are used as the beginning points for every cluster, and then performs iterative (repetitive) calculations to optimize the positions of the centroids.It halts creating and optimizing clusters when the centroids have stabilized - there is no change in their values because the clustering has been successful or the defined number of iterations has been achieved.

2.1 Experiment

In this experiment, first of all, we divided the dataset into **20% test set** and **80% train set**. Then, we scale our data using **MinMaxScaler** in scikit-learn.F Even though we know that there are two clusters (malignant, benign), we wanted to see optimum K value using elbow method. As you can see in figure 8, K value is 2 as we expected.

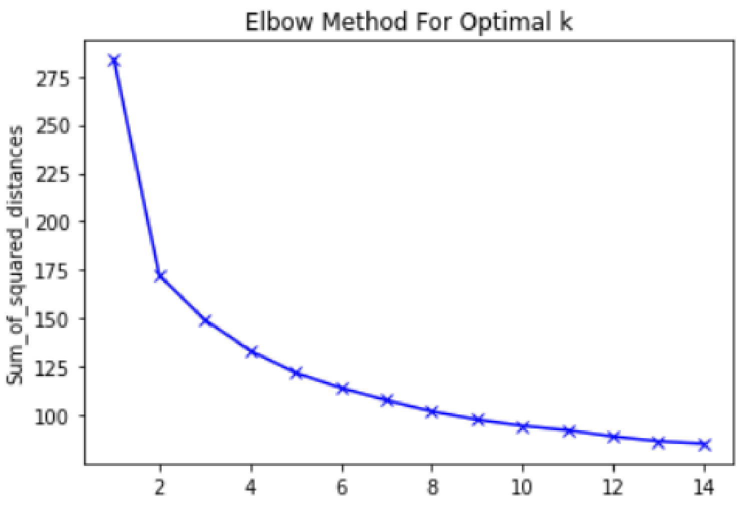
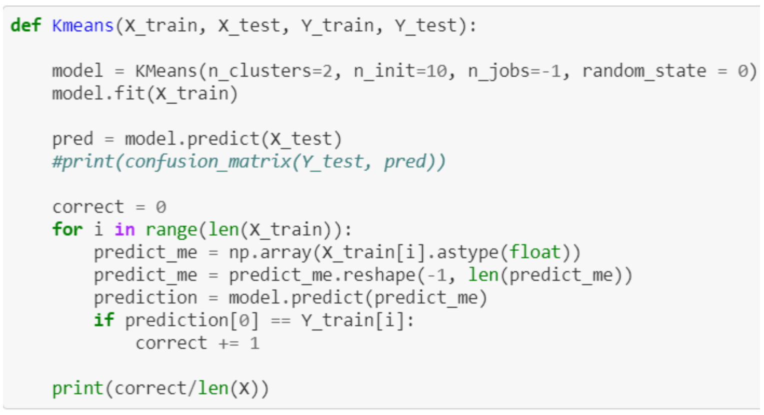


Figure 8: Elbow method for K value

We created our model using the function below.



```python
def Kmeans(X_train, X_test, Y_train, Y_test):

    model = KMeans(n_clusters=2, n_init=10, n_jobs=-1, random_state = 0)
    model.fit(X_train)

    pred = model.predict(X_test)
    #print(confusion_matrix(Y_test, pred))

    correct = 0
    for i in range(len(X_train)):
        predict_me = np.array(X_train[i].astype(float))
        predict_me = predict_me.reshape(-1, len(predict_me))
        prediction = model.predict(predict_me)
        if prediction[0] == Y_train[i]:
            correct += 1

    print(correct/len(X))
```

Figure 9: KMeans model

In order to check the cluster cohesion, we used silhouette score. We implemented a function which evaluates silhouette score and visualizes cluster cohesion. As a result we found **0.3807%**.

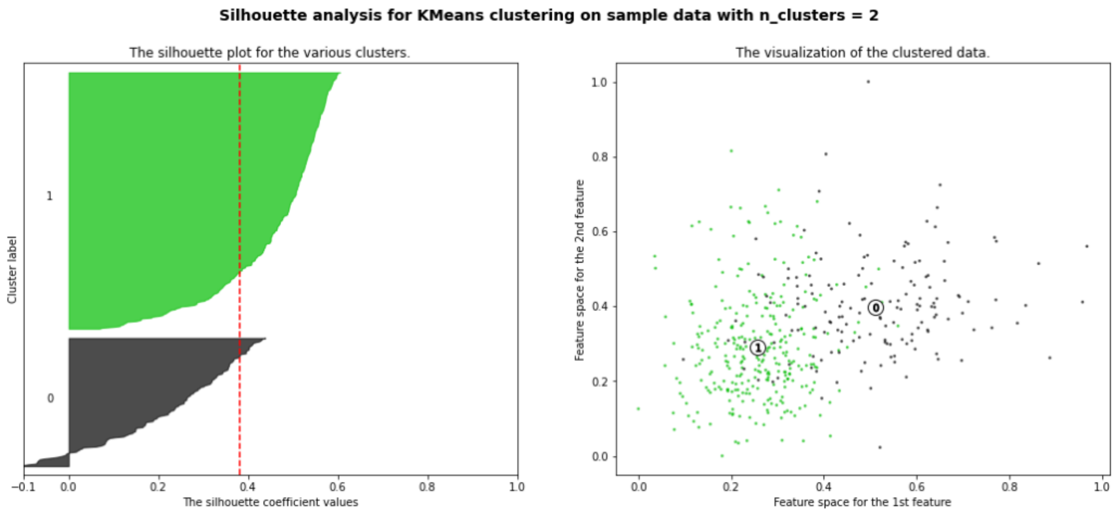


Figure 10: Silhouette analysis for KMeans clustering

Then, we test the model and achieve **71.53%** accuracy. In order to improve model, we tried different random_state numbers. Moreover, we tried to use all data set without filtering the correlated features. Finally, using nonfiltered dataset and random state number as 21, we achieved **74.87%** accuracy.

3 DBScan

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) is a popular unsupervised learning method utilized in model building and machine learning algorithms. The main concept of DBSCAN algorithm is to locate regions of high density that are separated from one another by regions of low density.

3.1 Distance Metrics

In KNN predictor, we have a lot distance metrics to use in. Through over our experiment, we will use different metrics and we will choose best result provider.

3.1.1 Minkownski Distance

Minkowski distance is a metric in Normed vector space. A Normed vector space is a vector space on which a norm is defined.

$$\left(\sum_{i=1}^{n} |x_i - y_i|^p\right)^{1/p}$$

Figure 11: Formula of Minkowski Distance

Minkowski distance is the generalized distance metric. We can manipulate the above formula to calculate the distance between two data points in different ways. As mentioned above, we can manipulate the value of $\boldsymbol{p}$ and calculate the distance in three different ways:

- p = 1, Manhattan Distance
- p = 2, Euclidean Distance
- p = ∞, Chebychev Distance.

3.1.2 Manhattan Distance

In Manhattan distance, if we need to calculate the distance between two data points in a grid like path. As mentioned above, we use ***Minkowski*** distance formula to find Manhattan distance by setting ***p***'s value as ***1***. Distance d will be calculated using an ***absolute sum of difference*** between its cartesian co-ordinates as below:

$$d = \sum_{i=1}^{n} |x_i - y_i|$$

Figure 12: Formula of Manhattan Distance

3.1.3 Euclidean Distance

Euclidean distance is one of the most used distance metric. It is calculated using Minkowski Distance formula by setting ***p***'s value to ***2***.

$$d(x, y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

Figure 13: Formula of Euclidean Distance

3.1.4 Mahalonobis Distance

Mahalonobis distance is the distance between a point and a distribution. And not between two distinct points. It is effectively a multivariate equivalent of the Euclidean distance.

$$D^2 = (x - m)^T \cdot C^{-1} \cdot (x - m)$$

Figure 14: Formula of Mahalonobis Distance

3.1.5 Chebyshev Distance

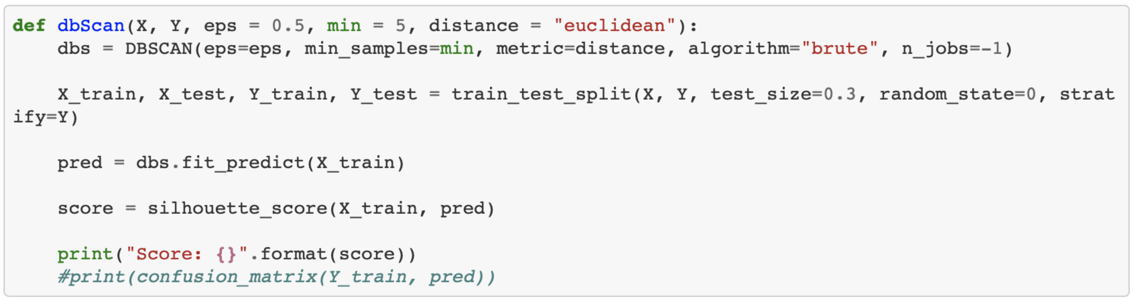
Chebyshev distance is also called Maximum value distance. It examines the absolute magnitude of the differences between coordinates of a pair of objects.

$$\max(|x_1 - x_2|, |y_1 - y_2|)$$

Figure 15: Formula of Chebyshev Distance

3.2 Experiment

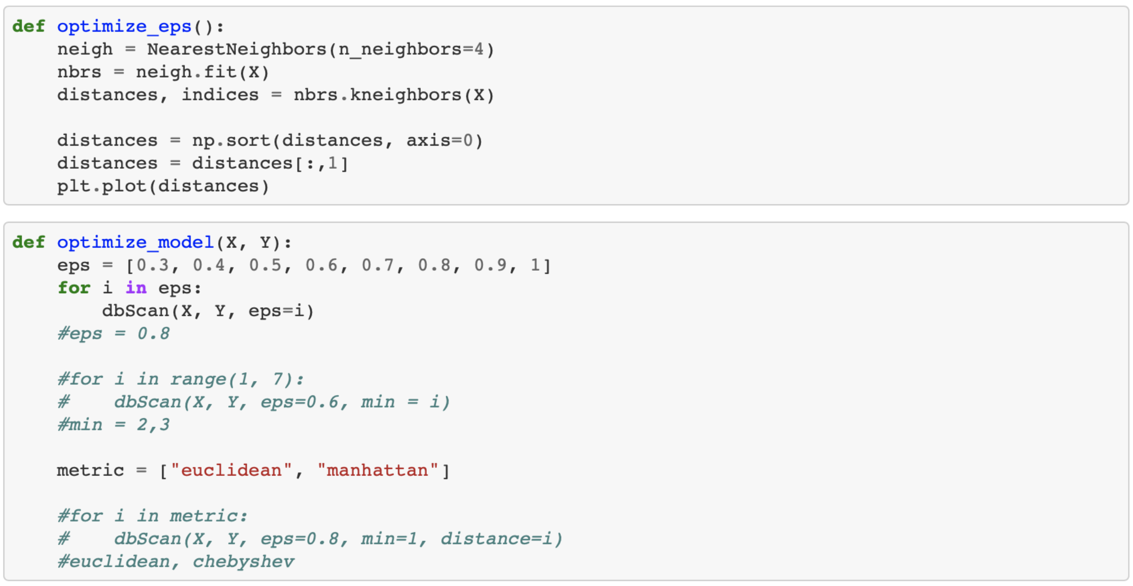
After we cleaned our data, we built the DBScan model, and we started to play with the parameters of the DBScan. Also, we applied to **MinMaxScaler** to our data to do better clustering. In order to train the model, we split our data as **%70 train** and **%30** test.



```
def dbScan(X, Y, eps = 0.5, min = 5, distance = "euclidean"):
    dbs = DBSCAN(eps=eps, min_samples=min, metric=distance, algorithm="brute", n_jobs=-1)

    X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.3, random_state=0, strat
ify=Y)

    pred = dbs.fit_predict(X_train)

    score = silhouette_score(X_train, pred)

    print("Score: {}".format(score))
    #print(confusion_matrix(Y_train, pred))
```

Figure 16: DBScan model

In order to decide the parameters, we have created two different methods. First one helps to find Epsilon value using the K-Nearest Neighbour classifier, and second method is iterating parameters using loops.



```
def optimize_eps():
    neigh = NearestNeighbors(n_neighbors=4)
    nbrs = neigh.fit(X)
    distances, indices = nbrs.kneighbors(X)

    distances = np.sort(distances, axis=0)
    distances = distances[:,1]
    plt.plot(distances)
```

```
def optimize_model(X, Y):
    eps = [0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1]
    for i in eps:
        dbScan(X, Y, eps=i)
    #eps = 0.8

    #for i in range(1, 7):
    #    dbScan(X, Y, eps=0.6, min = i)
    #min = 2,3

    metric = ["euclidean", "manhattan"]

    #for i in metric:
    #    dbScan(X, Y, eps=0.8, min=1, distance=i)
    #euclidean, chebyshev
```

Figure 17: 2 optimization method

During out experiment, we changed ***Epsilon***, ***Distance Metrics*** and ***minimum sample size***, and we got the best silhouette score which is **0.5659** with when **epsilon** = 0.8. However, when we played with the other parameters such as **minimum sample size** and **distance metrics**, but it did not effect the result of silhouette score.

4 Result

We created two individual method for calculating accuracy/error of the two model. In those methods, we compared predicted labels with actual labels. We got the best result silhouette score with DBScan;however, the DBScan model gave the worst accuracy results. We searched the reason why we got such a low Accuracy, and we displayed the data.

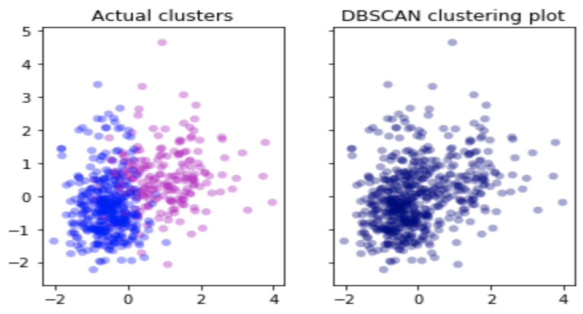


Figure 18: DBscan comparision

As we can see the clustering plot, DBscan could not seperate the cluster from each other, and this is why we got such a low accuracy which is **0.374**. After we got a such a bad result, we tried to get accuracy/error of the KMeans, and we got approximately **0.75**. In order to be sure, we also visualized the results.

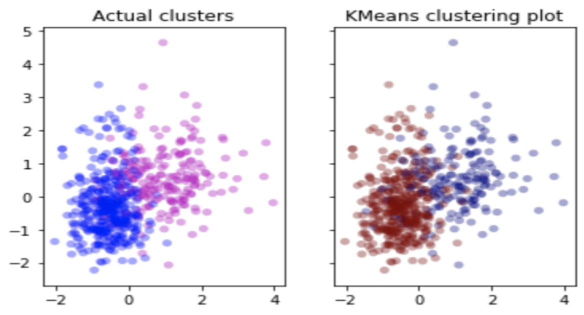


Figure 19: KMeans comparision

5 Appendix

5.1 Project Link

https://github.com/nisanuro/CNG562-Assignment-3

5.2 Code

```
# -*- coding: utf-8 -*-
"""CNG562_Assignment3.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/github/nisanuro/CNG562-
                                      Assignment-3/blob/master/
                                      CNG562_Assignment3.ipynb
"""

# Commented out IPython magic to ensure Python compatibility.
import seaborn as sns
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split, KFold,
                                      StratifiedKFold, cross_val_score
from sklearn import metrics, datasets, preprocessing
from sklearn.datasets import load_breast_cancer
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans, DBSCAN
from sklearn.neighbors import NearestNeighbors
from sklearn.metrics import silhouette_score
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import confusion_matrix
# %matplotlib inline

def correlation_map(df):
    plt.figure(figsize=(20,12))

    corr = df.corr()
    sns.heatmap(corr, annot=True, cmap='hot')
    plt.show()

def filter_features(data, feature_indexes):
    # eliminate above column indices from the data and return new
                                      set
    filtered_data = np.delete(data, feature_indexes, axis=1)

    return filtered_data

def fourError(X, Y, model, r, future_scaling):
    X_train, X_test, Y_train, Y_test = train_test_split(X, Y,
                                    test_size=0.3, random_state = r,
                                    stratify=Y)

    if(future_scaling):
        sc = StandardScaler()
        X_train = sc.fit_transform(X_train)
        X_test = sc.transform(X_test)

    Train_x, TrainDev_x, Train_y, TrainDev_y = train_test_split(
                                    X_train, Y_train, test_size=0.2,
                                    random_state=0, stratify=Y_train)
    Dev_x, Test_x, Dev_y, Test_y = train_test_split(X_test, Y_test,
                                     test_size=0.5, random_state=0,
                                    stratify=Y_test)

    model.fit(Train_x, Train_y)

    y_true, trainDev_pred = TrainDev_y, model.predict(TrainDev_x)

    print("\nTrain-Train Dev,   e1:", metrics.mean_squared_error(
                                    TrainDev_y, trainDev_pred))
    print("Accuracy: ", 1 - metrics.mean_squared_error(TrainDev_y,
                                    trainDev_pred),"\n")

    y_true, dev_pred = Dev_y, model.predict(Dev_x)
    print("Train-Dev,   e2", metrics.mean_squared_error(Dev_y,
                                    dev_pred))
    print("Accuracy: ", 1 - metrics.mean_squared_error(Dev_y,
                                    dev_pred),"\n")

    y_true, test_pred = Test_y, model.predict(Test_x)
    print("Train-Test,   e3: ", metrics.mean_squared_error(Test_y,
                                    test_pred))
    print("Accuracy: ", 1 - metrics.mean_squared_error(Test_y,
                                    test_pred),"\n")

    y_true, devTest_pred = Y_test, model.predict(X_test)
    print("Train-(Dev+Test),   e4: ", metrics.mean_squared_error(
                                    Y_test, devTest_pred))
    print("Accuracy: ", 1 - metrics.mean_squared_error(Y_test,
                                    devTest_pred),"\n")

def vis_all_feat(data, class_):
    for col_ind in range(data.shape[1]):
        print("Viewing Feature #{0}".format(str(col_ind)))
        vis_single_feat(data, class_, col_ind)
```

```python
def vis_single_feat(data, class_, ind):
    # create graph of classification and feature values
    plt.figure(100) # display two plots on separate figures
    df = pd.DataFrame(data)
    feat_vals = df.iloc[:, ind]
    plt.scatter(feat_vals, class_)
    plt.title("Plot of Feature {0}".format(str(ind)))
    plt.xlabel("Feature Value")
    plt.ylabel("Classification")

    # create bar graph of mean feature values for each classification
    plt.figure(200)
    plt.title("Mean Values of Feature {0}".format(str(ind)))
    plt.xlabel("Classification")
    plt.ylabel("Mean Feature Value")
    mean_df = pd.concat([df.iloc[:, ind], pd.Series(class_)], axis=1)
    mean_df.columns = ["values", "classif"]
    mean_df.groupby("classif", as_index=False)["values"].mean().loc[:,"values"].plot(kind='bar')

    plt.show()

def dbScan(X, Y, eps = 0.5, min = 5, distance = "euclidean"):
    dbs = DBSCAN(eps=eps, min_samples=min, metric=distance, algorithm="brute", n_jobs=-1)

    X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.3, random_state=0, stratify=Y)

    pred = dbs.fit_predict(X_train)

    score = silhouette_score(X_train, pred)

    print("Score: {}".format(score))
    #print(confusion_matrix(Y_train, pred))

def optimize_eps():
    neigh = NearestNeighbors(n_neighbors=4)
    nbrs = neigh.fit(X)
    distances, indices = nbrs.kneighbors(X)

    distances = np.sort(distances, axis=0)
    distances = distances[:,1]
    plt.plot(distances)

def optimize_model(X, Y):
    eps = [0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1]
    for i in eps:
        dbScan(X, Y, eps=i)
    #eps = 0.8

    #for i in range(1, 7):
    #    dbScan(X, Y, eps=0.6, min = i)
    #min = 2,3

    metric = ["euclidean", "manhattan"]

    #for i in metric:
    #    dbScan(X, Y, eps=0.8, min=1, distance=i)
    #euclidean, chebyshev

def acc(X_train, X_test, Y_train, Y_test):
    model = DBSCAN(eps=0.8, min_samples=2, metric="euclidean",
                                algorithm="brute", n_jobs=-1)

    #pred = model.predict(X_test)
    #print(confusion_matrix(Y_test, pred))
    prediction = model.fit_predict(X_train)

    correct = 0
    for i in range(len(Y_train)):
        if prediction[i] == Y_train[i]:
            correct += 1

    print(correct/len(X_train))

def ElbowMethod(data):
    Sum_of_squared_distances = []
    K = range(1,15)
    for k in K:
        km = KMeans(n_clusters=k, n_init=10, n_jobs=-1,
                                random_state = 0)
        km = km.fit(data)
        Sum_of_squared_distances.append(km.inertia_)

    plt.plot(K, Sum_of_squared_distances, 'bx-')
    plt.xlabel('k')
    plt.ylabel('Sum_of_squared_distances')
    plt.title('Elbow Method For Optimal k')
    plt.show()

def SilhouetteAnalysis(X, Y):
    range_n_clusters = [2, 3, 4]

    for n_clusters in range_n_clusters:

        # Create a subplot with 1 row and 2 columns
        fig, (ax1, ax2) = plt.subplots(1, 2)
        fig.set_size_inches(18, 7)

        # The 1st subplot is the silhouette plot
        # The silhouette coefficient can range from -1, 1 but in this example all
        # lie within [-0.1, 1]
        ax1.set_xlim([-0.1, 1])
        # The (n_clusters+1)*10 is for inserting blank space between silhouette
        # plots of individual clusters, to demarcate them clearly.
        ax1.set_ylim([0, len(X) + (n_clusters + 1) * 10])

        # Initialize the clusterer with n_clusters value and a random generator
        # seed of 10 for reproducibility.
        clusterer = KMeans(n_clusters=n_clusters, n_init=10, n_jobs=-1,random_state = 0)
        cluster_labels = clusterer.fit_predict(X)

        # The silhouette_score gives the average value for all the samples.
        # This gives a perspective into the density and separation of the formed
        # clusters
        silhouette_avg = silhouette_score(X, cluster_labels)

        print("For n_clusters =", n_clusters,
            "The average silhouette_score is :", silhouette_avg)



        # Compute the silhouette scores for each sample
        sample_silhouette_values = silhouette_samples(X, cluster_labels)

        y_lower = 10
        for i in range(n_clusters):
            # Aggregate the silhouette scores for samples belonging to
            # cluster i, and sort them
            ith_cluster_silhouette_values = \
                sample_silhouette_values[cluster_labels == i]

            ith_cluster_silhouette_values.sort()

            size_cluster_i = ith_cluster_silhouette_values.shape[0]
            y_upper = y_lower + size_cluster_i

            color = cm.nipy_spectral(float(i) / n_clusters)
            ax1.fill_betweenx(np.arange(y_lower, y_upper),
                            0, ith_cluster_silhouette_values,
                            facecolor=color, edgecolor=color, alpha=0.7)

            # Label the silhouette plots with their cluster numbers at the middle
            ax1.text(-0.05, y_lower + 0.5 * size_cluster_i, str(i))

            # Compute the new y_lower for next plot
            y_lower = y_upper + 10  # 10 for the 0 samples

        ax1.set_title("The silhouette plot for the various clusters.")
        ax1.set_xlabel("The silhouette coefficient values")
        ax1.set_ylabel("Cluster label")

        # The vertical line for average silhouette score of all the values
        ax1.axvline(x=silhouette_avg, color="red", linestyle="--")

        ax1.set_yticks([])  # Clear the yaxis labels / ticks
        ax1.set_xticks([-0.1, 0, 0.2, 0.4, 0.6, 0.8, 1])

        # 2nd Plot showing the actual clusters formed
        colors = cm.nipy_spectral(cluster_labels.astype(float) / n_clusters)
        ax2.scatter(X[:, 0], X[:, 1], marker='.', s=30, lw=0, alpha=0.7,
                    c=colors, edgecolor='k')

        # Labeling the clusters
        centers = clusterer.cluster_centers_
        # Draw white circles at cluster centers
        ax2.scatter(centers[:, 0], centers[:, 1], marker='o',
                    c="white", alpha=1, s=200, edgecolor='k')

        for i, c in enumerate(centers):
            ax2.scatter(c[0], c[1], marker='$%d$' % i, alpha=1,
                        s=50, edgecolor='k')

        ax2.set_title("The visualization of the clustered data.")
        ax2.set_xlabel("Feature space for the 1st feature")
        ax2.set_ylabel("Feature space for the 2nd feature")

        plt.suptitle(("Silhouette analysis for KMeans clustering on sample data "
                    "with n_clusters = %d" % n_clusters),
                    fontsize=14, fontweight='bold')

    plt.show()

def Kmeans(X_train, X_test, Y_train, Y_test):

    model = KMeans(n_clusters=2, n_init=10, n_jobs=-1, random_state
                                   = 0)
    model.fit(X_train)

    pred = model.predict(X_test)
    #print(confusion_matrix(Y_test, pred))

    correct = 0
    for i in range(len(X_train)):
        predict_me = np.array(X_train[i].astype(float))
        predict_me = predict_me.reshape(-1, len(predict_me))
        prediction = model.predict(predict_me)
        if prediction[0] == Y_train[i]:
            correct += 1

    print(correct/len(X))

if __name__ == '__main__':

    breast_cancer = datasets.load_breast_cancer()
    X = breast_cancer.data
    Y = breast_cancer.target

    X = filter_features(X, [2, 3, 20, 22, 23, 12, 13])

    vis_all_feat(X, Y)
    X = filter_features(X, [1, 2, 6, 7, 9, 10, 14, 15])
    scaler = MinMaxScaler()
    scaler.fit(X)
    X = scaler.transform(X)
    optimize_eps()
    X_train, X_test, Y_train, Y_test = train_test_split(X, Y,
                                        test_size=0.2, random_state=0,
                                        stratify=Y)
    optimize_model(X, Y)
    acc(X_train, X_test, Y_train, Y_test)

ElbowMethod(X_train)
    SilhouetteAnalysis(X_train, Y_train)

    Kmeans(X_train, X_test, Y_train, Y_test)
```