**Programming Homework 2**

**201935023 KIM SU HYEON**

**201735841 JANG KEON**

**201635840 LEE JUNG MYUNG**

**1. Dataset inspection & preprocessing**

We used the "California-housing-prices" dataset. This data set consists of 10 columns and has a total of 20640 data. Each column has longitude, latitude, number of rooms, income, and population.

First, a total\_bedroom (20433 element) having a lot of null data is dropped. After that, for measuring purity later, media\_house\_value use is separated and divided into 10 categories.

Now, preprocessing such as scaling and encoding must be performed. Ocean\_proximity is Category data, and all other data are numerical data. So, We created “**Preprocessing”** function that proceeds with five scaling ("standard", "minmax", "robust", "maxabs", and "normalize".) and two encoding (“ordinal”, “one-hot”)

Also, we are planning to test by creating a combination of various features. A total of four feature combination cases were created.

**Case 1 : All features except median\_house\_value.**

**Case 2 : [median\_income, housing\_median\_age, longtitude, latitude]**

**Case 3 : [median\_income, housing\_median\_age, longtitude, latitude, total\_rooms]**

**Case 4 : [median\_income, housing\_median\_age, longtitude, latitude, total\_rooms, population, ocean\_proximity]**

**2. Model Training**

Now, I need to make cluster models: K-means, EM, clarans, DBSCAN, Meanshift

To this end, I created “**FindBestCombination”** function that learns four models using the Dataset.

Experiments were conducted on the following combinations.

n\_cluster = list(range(2, 13))

DBSCAN\_list = {'eps':[0.1, 0.2, 0.5, 5, 10, 100, 1000],'min\_sample': [10, 20]}

MeanShift\_list = [None, 1.0, 2.0]

The param and score of each model were stored and used to find the best combination for each model after all training was completed. In the case of Clarans, model training took too much time (up to 12 hours), so only the results for original scaling were inevitably obtained.

**3. Result & Discussion**

We analyzed the results from various perspectives.

|  |  |
| --- | --- |
| Feature case | Best K in  k-mean |
| 1 | 4 |
| 2 | 4 |
| 3 | 4 |
| 4 | 4 |

First, it’s a task of finding the best K in k-means for each feature case. For all cases, when k is 4, an appropriate sum of distance is shown.

But, can k=4 be said to be the best K in k-means at all cases? So we examined other indicators too. Let’s look at the silhouette score of k-means first. In the case of ‘Feature case 1’, when k was 4, the highest score was recorded. However, the rest of the cases recorded the best silhouette score when k=2. Accordingly, we learned that k obtained by the elbow method may not be the best K in Silhouette Score.

<k-means silhouette score>

|  |  |  |  |
| --- | --- | --- | --- |
| Case 1 | Case 2 | Case 3 | Case 4 |
|  |  |  |  |

We also found unique point about Purity Score. Each time the number of clusers grew, in most cases the purity score also increased proportionally. High purity is easy to achieve when the number of clusters is large - in particular, purity is 1 if each document gets its own cluster.

<purity score>

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Case 1 | Case 2 | Case 3 | Case 4 |
| k-mean |  |  |  |  |
| EM |  |  |  |  |
| clarans |  |  |  |  |

In addition, when looking at the linearity of purity, it can be seen that ‘Case2’ and ‘Case3’ are good feature selections.

Below tables are an analysis that simply finds the best numerical results.

<Best Silhouette>

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Feature case | k-mean | EM | Clarans | DBSCAN | MeanShift |
| 1 | 0.8640 | 0.3149 | 0.57241 | 0.9192 | 0.8886 |
| 2 | 0.5606 | 0.5602 | 0.34325 | 0.5985 | 0.5571 |
| 3 | 0.9171 | 0.8114 | 0.53089 | 0.9357 | 0.7888 |
| 4 | 0.7188 | 0.5542 | 0.42186 | 0.9329 | 0.7783 |

<Best Purity>

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Feature case | k-mean | EM | Clarans | DBSCAN | MeanShift |
| 1 | 0.8640 | 0.31225 | 0.20847 | 0.29123 | 1.0 |
| 2 | 0.3088 | 0.3209 | 0.227422 | 0.2849 | 0.4130 |
| 3 | 0.2914 | 0.3058 | 0.20208 | 0.2488 | 0.9942 |
| 4 | 0.2069 | 0.3021 | 0.21036 | 0.2665 | 1.0 |

When looking at the Silhouette Score, ‘Case 3’ seems to be an appropriate features for clustering.

When looking at the Purity, ‘Case 2’ seems to be an appropriate feature for clustering. Meanshift has a case that the bandwidth is too small. This case, too many clusters have been created, resulting in 1.0 Purity Score. So it is safe to say that is an outlier.

Below tables are the scaler&encoder combination for the best numerical results.

<Best Silhouette>

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Feature case | k-mean | EM | Clarans | DBSCAN | MeanShift |
| 1 | original  +  onehot | original  +  onehot |  | robust  +  onehot | robust  +  onehot |
| 2 | normalize | normalize |  | robust | maxabs |
| 3 | normalize | normalize |  | original | robust |
| 4 | original  +  onehot | maxabs  +  ordinal |  | maxabs  +  ordinal | ordinal  +  onehot |

<Best Purity>

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Feature case | k-mean | EM | Clarans | DBSCAN | MeanShift |
| 1 | original  +  ordinal | original  +  onehot |  | minmax  +  onehot | original  +  onehot |
| 2 | robust | standard |  | original | original |
| 3 | standard | robust |  | robust | original |
| 4 | maxabs  +  ordinal | robust  +  ordinal |  | ordinal  + onehot | maxabs  + ordinal |

Clarans takes too much time to train the model, so there is no data because the experiment is conducted only on the original scaling.

In the case of Best Silhouette, most of them used ordinal and normalize scaling. And in the case of encoder, it appeared different for each feature case. ‘Case 2’ and ‘Case 3’ do not use categorical feature, so there is no encoder. However, in ‘Case 1’, all of the encoders that produce the best score are one-hot encoder. And in ‘Case 4’, the ordinal was most effective.

In the case of Purity, best Purity appeared in various scalers and encoders.

**As a result**, it is confirmed that clustering was performed well in the feature sets of **‘Case 2’** and **‘Case 3’**. In addition, in the case of Silhouette Score, it was found that original scaling and normalize scaling are effective.

* Code

*import* pandas *as* pd

*import* numpy *as* np

*import* pprint

*import* matplotlib.pylab *as* plt

*from* sklearn *import* preprocessing, metrics

*from* sklearn.cluster *import* KMeans, MeanShift, DBSCAN

*from* sklearn.mixture *import* GaussianMixture

*from* sklearn.metrics *import* silhouette\_score

*from* pyclustering.cluster.clarans *import* clarans

*from* pyclustering.utils *import* timedcall;

*import* time

*#sum of distance for elbow method*

kmeans\_sumofDistance = {}

*#silhouette*

kmeans\_silhouette = {}

gmm\_silhouette = {}

clarans\_silhouette = {}

meanshift\_silhouette = {}

dbscan\_silhouette = {}

*#purity*

kmeans\_purity = {}

gmm\_purity = {}

clarans\_purity = {}

meanshift\_purity = {}

dbscan\_purity = {}

def main():

*#hyperparameter*

    n\_cluster = list(range(2, 13))

    DBSCAN\_list = {'eps': [0.1, 0.2, 0.5, 5, 10, 100, 1000], 'min\_sample': [10, 20]}

    MeanShift\_list = [None, 1.0, 2.0, 10, 100]

    MeanShift\_list\_plot = [0, 1.0, 2.0, 10, 100]

    print("=== 1. Data Load & Missing Data check")

    dataset = pd.read\_csv('Lab2\_PHW2\housing.csv') *#load dataset*

*# print(dataset.describe())*

*# print(dataset.isna().sum())*

*# print(dataset.info())*

    print("=== 2. split median\_house\_value & labeling")

    median\_house\_value = pd.DataFrame(dataset["median\_house\_value"])

    dataset = dataset.drop(*columns*=["median\_house\_value"])

    bins = list(range(14998, 500002, 48500))

    median\_house\_value['label'] = pd.cut(median\_house\_value["median\_house\_value"],

                                    bins,

*labels*=[1, 2, 3, 4, 5, 6, 7, 8, 9, 10])

*# print(median\_house\_value.groupby('label')['median\_house\_value'].apply(my\_summary).unstack())*

    print("=== 3. drop not use data (total\_bedrooms)")

    dataset = dataset.drop(*columns*=["total\_bedrooms"])

*# print(dataset.info())*

    print("=== 4. Preprocessing")

    pre\_feature = Preprocessing(dataset, ["ocean\_proximity"], ["longitude", "latitude", "housing\_median\_age", "total\_rooms", "population", "households", "median\_income"])

*# pprint.pprint(pre\_feature)*

    print("=== 5. make clustering")

*for* key, value *in* pre\_feature.items():

        FindBestCombination(key, value, n\_cluster, DBSCAN\_list, MeanShift\_list, median\_house\_value['label'])

    print("=== 6. Result")

*#check sum of distance for elbow method*

    makeplot("KMeans\_distance", kmeans\_sumofDistance, n\_cluster)

*# #silhouette*

    makeplot("KMeans\_silhouette", kmeans\_silhouette, n\_cluster)

    makeplot("EM\_silhouette", gmm\_silhouette, n\_cluster)

    makeplot("DBSCAN\_silhouette", dbscan\_silhouette, DBSCAN\_list['eps'])

    makeplot("MeanShift\_distance", meanshift\_silhouette, MeanShift\_list\_plot)

    makeplot("Clarans\_distance", clarans\_silhouette, n\_cluster)

    key, value = fineMaxValueKey(kmeans\_silhouette)

    print("k-means best silhouette : ", value, key)

    key, value = fineMaxValueKey(gmm\_silhouette)

    print("EM best silhouette : ", value, key)

    key, value = fineMaxValueKey(dbscan\_silhouette)

    print("DBSCAN best silhouette : ", value, key)

    key, value = fineMaxValueKey(meanshift\_silhouette)

    print("MeanShift best silhouette : ", value, key)

    key, value = fineMaxValueKey(clarans\_silhouette)

    print("Clarans best silhouette : ", value, key)

*#purity*

    makeplot("KMeans\_purity", kmeans\_purity, n\_cluster)

    makeplot("EM\_purity", gmm\_purity, n\_cluster)

    makeplot("DBSCAN\_purity", dbscan\_purity, DBSCAN\_list['eps'])

    makeplot("MeanShift\_distance", meanshift\_purity, MeanShift\_list\_plot)

    makeplot("Clarans\_distance", clarans\_purity, n\_cluster)

    key, value = fineMaxValueKey(kmeans\_purity)

    print("k-means best purity : ", value, key)

    key, value = fineMaxValueKey(gmm\_purity)

    print("k-means best purity : ", value, key)

    key, value = fineMaxValueKey(dbscan\_purity)

    print("DBSCAN best purity : ", value, key)

    key, value = fineMaxValueKey(meanshift\_purity)

    print("MeanShift best purity : ", value, key)

    key, value = fineMaxValueKey(clarans\_purity)

    print("Clarans best purity : ", value, key)

*#for one-hot-encoding*

def dummy\_data(*data*, *columns*):

*for* column *in* columns:

        data = pd.concat([data, pd.get\_dummies(data[column], *prefix* = column)], *axis*=1)

        data = data.drop(column, *axis*=1)

*return* data

def Preprocessing(*feature*, *encode\_list*, *scale\_list*):

*# feature : dataframe of feature*

*#scaler*

    scaler\_stndard = preprocessing.StandardScaler()

    scaler\_MM = preprocessing.MinMaxScaler()

    scaler\_robust = preprocessing.RobustScaler()

    scaler\_maxabs = preprocessing.MaxAbsScaler()

    scaler\_normalize = preprocessing.Normalizer()

    scalers = [None, scaler\_stndard, scaler\_MM, scaler\_robust, scaler\_maxabs, scaler\_normalize]

    scalers\_name = ["original", "standard", "minmax", "robust", "maxabs", "normalize"]

*# encoder*

    encoder\_ordinal = preprocessing.OrdinalEncoder()

*#one hot encoding => using pd.get\_dummies() (not used preprocessing.OneHotEncoder())*

    encoders\_name = ["ordinal", "onehot"]

*# result box*

    result\_dictionary = {}

    i = 0

*if* encode\_list == []:

*for* scaler *in* scalers:

*if* i == 0: *#not scaling*

                result\_dictionary[scalers\_name[i]] = feature.copy()

*else*:

*#===== scalers*

                result\_dictionary[scalers\_name[i]] = feature.copy()

                result\_dictionary[scalers\_name[i]][scale\_list] = scaler.fit\_transform(feature[scale\_list]) *#scaling*

            i = i + 1

*return* result\_dictionary

*for* scaler *in* scalers:

*if* i == 0: *#not scaling*

            result\_dictionary[scalers\_name[i] + "\_ordinal"] = feature.copy()

            result\_dictionary[scalers\_name[i] + "\_ordinal"][encode\_list] = encoder\_ordinal.fit\_transform(feature[encode\_list])

            result\_dictionary[scalers\_name[i] + "\_onehot"] = feature.copy()

            result\_dictionary[scalers\_name[i] + "\_onehot"]

= dummy\_data(result\_dictionary[scalers\_name[i] + "\_onehot"], encode\_list)

*else*:

*#===== scalers + ordinal encoding*

            result\_dictionary[scalers\_name[i] + "\_ordinal"] = feature.copy()

            result\_dictionary[scalers\_name[i] + "\_ordinal"][scale\_list] = scaler.fit\_transform(feature[scale\_list])

            result\_dictionary[scalers\_name[i] + "\_ordinal"][encode\_list] = encoder\_ordinal.fit\_transform(feature[encode\_list])

*#===== scalers + OneHot encoding*

            result\_dictionary[scalers\_name[i] + "\_onehot"] = feature.copy()

            result\_dictionary[scalers\_name[i] + "\_onehot"][scale\_list] = scaler.fit\_transform(feature[scale\_list])

            result\_dictionary[scalers\_name[i] + "\_onehot"]

= dummy\_data(result\_dictionary[scalers\_name[i] + "\_onehot"], encode\_list)

        i = i + 1

*return* result\_dictionary

def FindBestCombination(*preprocessing\_name*, *feature*, *n\_cluster*, *DBSCAN\_list*, *MeanShift\_list*, *purity\_GT*):

    print(preprocessing\_name)

*# n\_cluster : list number of cluster (use in Kmeans, GMM, clarans)*

*# clarans\_list : list of clarans parameters (numlocal, maxneighbor)*

*# DBSCAN\_list : list of DBSCAN parameters (eps, min\_sample)*

*# MeanShift\_list : list of MeanShift parameters (bandwidth)*

*# KMeans*

    print("Kmeans")

    start\_time = time.time()

    kmean\_sum\_of\_squared\_distances = []

    kmean\_silhouette\_sub = []

    kmeans\_purity\_sub = []

*for* k *in* n\_cluster:

        kmeans = KMeans(*n\_clusters*=k).fit(feature)

*# sum of distance for elbow methods*

        kmean\_sum\_of\_squared\_distances.append(kmeans.inertia\_)

*# silhouette (range -1~1)*

        kmean\_silhouette\_sub.append(silhouette\_score(feature, kmeans.labels\_, *metric*='euclidean'))

*# purity*

        kmeans\_purity\_sub.append(purity\_score(purity\_GT, kmeans.labels\_))

    kmeans\_sumofDistance[preprocessing\_name] = kmean\_sum\_of\_squared\_distances

    kmeans\_silhouette[preprocessing\_name] = kmean\_silhouette\_sub

    kmeans\_purity[preprocessing\_name] = kmeans\_purity\_sub

    print(time.time() - start\_time)

*# GaussianMixture (EM, GMM)*

    print("EM")

    start\_time = time.time()

    gmm\_silhouette\_sub = []

    gmm\_purity\_sub = []

*for* k *in* n\_cluster:

        gmm = GaussianMixture(*n\_components*=k)

        labels = gmm.fit\_predict(feature)

        gmm\_silhouette\_sub.append(silhouette\_score(feature, labels, *metric*='euclidean'))

        gmm\_purity\_sub.append(purity\_score(purity\_GT, labels))

    gmm\_silhouette[preprocessing\_name] = gmm\_silhouette\_sub

    gmm\_purity[preprocessing\_name] = gmm\_purity\_sub

    print(time.time() - start\_time)

*# clarans*

    print("clarans")

    data = np.array(feature)

    data = data.tolist()

    clarans\_silhouette\_sub = []

    clarans\_purity\_sub = []

    max\_sum = 0

*for* k *in* n\_cluster:

        clarans\_instance = clarans(data, k, 6, 4)

        (ticks, result) = timedcall(clarans\_instance.process)

        clusters = clarans\_instance.get\_clusters()

        label = clusterToIdx(clusters)

        clarans\_silhouette\_sub.append(silhouette\_score(feature, label, *metric*='euclidean'))

        clarans\_purity\_sub.append(purity\_score(purity\_GT, label))

    clarans\_silhouette[preprocessing\_name] = clarans\_silhouette\_sub

    clarans\_purity[preprocessing\_name] = clarans\_purity\_sub

*# DBSCAN*

    print("dbscan")

    start\_time = time.time()

    dbscan\_silhouette\_sub = []

    dbscan\_purity\_sub = []

*for* eps *in* DBSCAN\_list["eps"]:

        max\_silhouette = -2

        max\_purity = -2

*for* sample *in* DBSCAN\_list["min\_sample"]:

            dbscan = DBSCAN(*eps*=eps, *min\_samples*=sample)

            label = dbscan.fit\_predict(feature)

*try*:

                current\_silhouette = silhouette\_score(feature, label, *metric*='euclidean')

*except*:

                current\_silhouette = -5

*if* max\_silhouette < current\_silhouette:

                max\_silhouette = current\_silhouette

            current\_purity = purity\_score(purity\_GT, label)

*if* max\_purity < current\_purity:

                max\_purity = current\_purity

        dbscan\_silhouette\_sub.append(max\_silhouette)

        dbscan\_purity\_sub.append(max\_purity)

    dbscan\_silhouette[preprocessing\_name] = dbscan\_silhouette\_sub

    dbscan\_purity[preprocessing\_name] = dbscan\_purity\_sub

    print(time.time() - start\_time)

*# meanShift*

    print("meanshift")

    start\_time = time.time()

    meanshift\_silhouette\_sub = []

    meanshift\_purity\_sub = []

*for* bw *in* MeanShift\_list:

        meanShift = MeanShift(*bandwidth*=bw)

        label = meanShift.fit\_predict(feature)

        print(label)

        print(time.time() - start\_time)

*try*:

            current\_silhouette = silhouette\_score(feature, label, *metric*='euclidean')

*except*:

            current\_silhouette = -1

        meanshift\_silhouette\_sub.append(current\_silhouette)

        meanshift\_purity\_sub.append(purity\_score(purity\_GT, label))

    meanshift\_silhouette[preprocessing\_name] = meanshift\_silhouette\_sub

    meanshift\_purity[preprocessing\_name] = meanshift\_purity\_sub

    print(time.time() - start\_time)

*#Test purity*

def purity\_score(*y\_true*, *y\_pred*):

*# compute contingency matrix (also called confusion matrix)*

    contingency\_matrix = metrics.cluster.contingency\_matrix(y\_true, y\_pred)

*# return purity*

*return* np.sum(np.amax(contingency\_matrix, *axis*=0)) / np.sum(contingency\_matrix)

def makeplot(*title*, *dict*, *x\_list*):

*for* key, value *in* dict.items():

        plt.plot(x\_list, value, *label*=key)

    plt.title(title)

    plt.legend(*bbox\_to\_anchor*=(1.05, 1.0), *loc*="upper left")

    plt.tight\_layout()

    plt.show()

def my\_summary(*x*):

    result = {

        'sum': x.sum(),

        'count': x.count(),

        'mean': x.mean(),

        'variance': x.var()

    }

*return* result

def fineMaxValueKey(*dict*):

    key = None

    largest = 0

*for* keys, item *in* dict.items():

*if* max(item) > largest:

            largest = max(item)

            key = keys

*return* key, largest

def clusterToIdx(*clusters*):

    idx\_list = [-1 *for* i *in* range(0, 20640)]

    idx = 0

*for* k *in* clusters:

*for* i *in* k:

            idx\_list[i] = idx

        idx = idx + 1

*return* idx\_list

*if* \_\_name\_\_ == "\_\_main\_\_":

    main()

* Result (output)

"=== 1. Data Load & Missing Data check"

텍스트, 배터리, 명판이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명텍스트이(가) 표시된 사진

자동 생성된 설명

"=== 2. split median\_house\_value & labeling"

텍스트이(가) 표시된 사진

자동 생성된 설명

"=== 3. drop not use data (total\_bedrooms)"

텍스트이(가) 표시된 사진

자동 생성된 설명

=== 4. Preprocessing

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

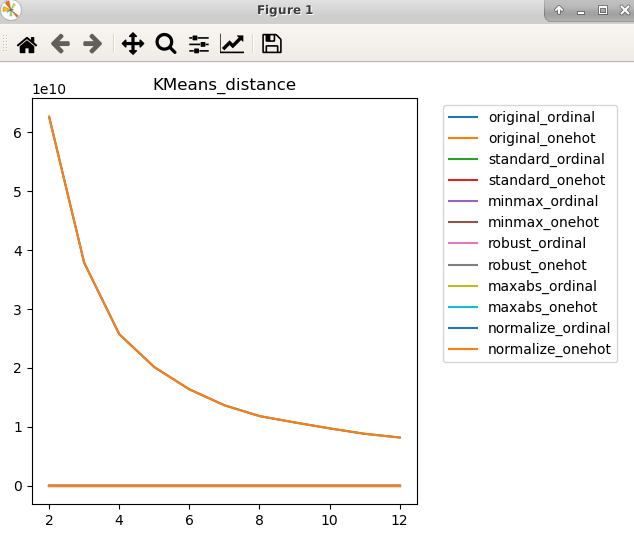
자동 생성된 설명

"=== 6. Result"

* **Feature case 1 (all feature)**

<K-mean>

1. Sum of distance in k-means (for elbow method)



|  |  |
| --- | --- |
| 2. k-means silhouette score | 3. k-means purity |
|  |  |

<EM clustering – GMM>

|  |  |
| --- | --- |
| 1. EM silhouette score | 2. EM purity |
|  |  |

<Clarans>

|  |  |
| --- | --- |
| 1. Clarans silhouette score | 2. Clarans purity |
|  |  |

<DBSCAN>

|  |  |
| --- | --- |
| 1. DBSCAN silhouette score | 2. DBSCAN purity |
|  |  |

<MeanShift>

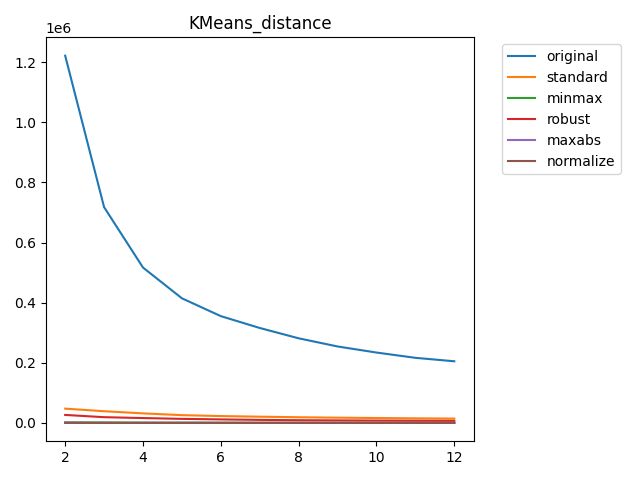
|  |  |
| --- | --- |
| 1. MeanShift silhouette score | 2. MeanShift purity |
|  |  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| cluster | k-mean | EM | Clarans | DBSCAN | MeanShift |
| Silhouette score | 0.864007 | 0.31492 | 0.57241 | 0.91926 | 0.88869 |
| Purity score | 0.8640 | 0.31225 | 0.20847 | 0.29123 | 1.0 |

* **Feature case 2**

<K-mean>

1. Sum of distance in k-means (for elbow method)



|  |  |
| --- | --- |
| 2. k-means silhouette score | 3. k-means purity |
|  |  |

<EM clustering – GMM>

|  |  |
| --- | --- |
| 1. EM silhouette score | 2. EM purity |
|  |  |

<Clarans>

|  |  |
| --- | --- |
| 1. Clarans silhouette score | 2. Clarans purity |
|  |  |

<DBSCAN>

|  |  |
| --- | --- |
| 1. DBSCAN silhouette score | 2. DBSCAN purity |
|  |  |

<MeanShift>

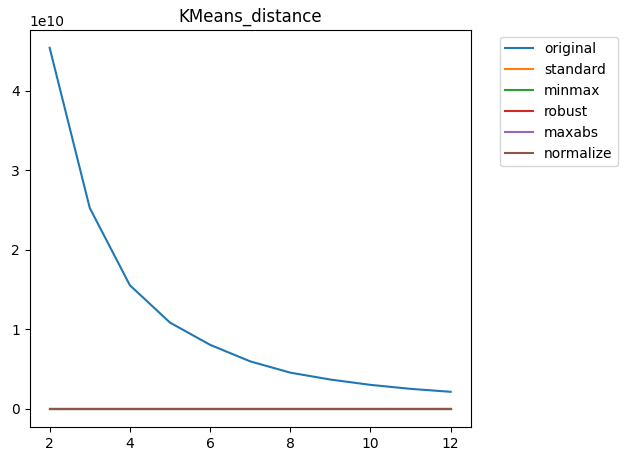
|  |  |
| --- | --- |
| 1. MeanShift silhouette score | 2. MeanShift purity |
|  |  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| cluster | k-mean | EM | Clarans | DBSCAN | MeanShift |
| Silhouette score | 0.5606 | 0.5602 | 0.343255 | 0.5982 | 0.5571 |
| Purity score | 0.3088 | 0.3209 | 0.227422 | 0.2849 | 0.4130 |

* **Feature case 3**

<K-mean>

1. Sum of distance in k-means (for elbow method)



|  |  |
| --- | --- |
| 2. k-means silhouette score | 3. k-means purity |
|  |  |

<EM clustering – GMM>

|  |  |
| --- | --- |
| 1. EM silhouette score | 2. EM purity |
|  |  |

<Clarans>

|  |  |
| --- | --- |
| 1. Clarans silhouette score | 2. Clarans purity |
|  |  |

<DBSCAN>

|  |  |
| --- | --- |
| 1. DBSCAN silhouette score | 2. DBSCAN purity |
|  |  |

<MeanShift>

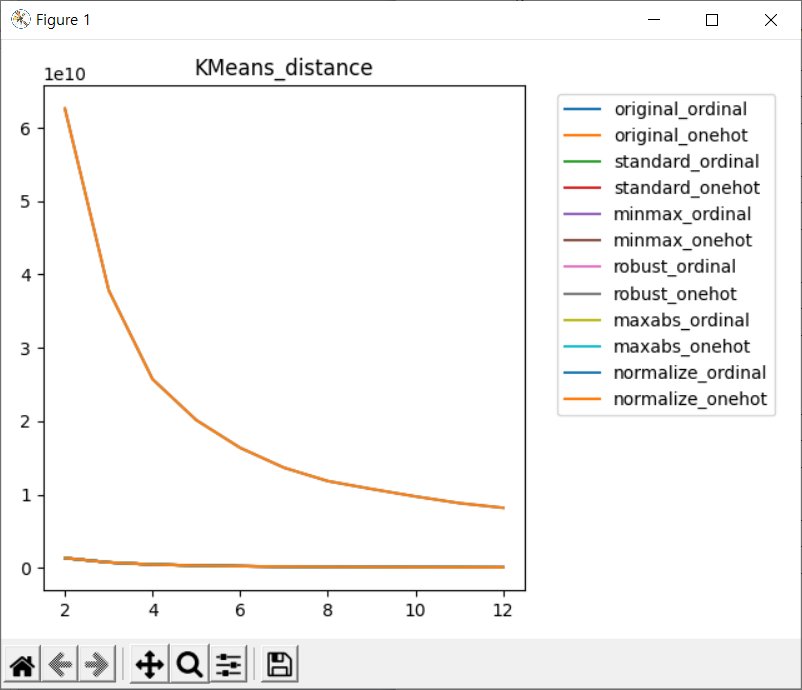
|  |  |
| --- | --- |
| 1. MeanShift silhouette score | 2. MeanShift purity |
|  |  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| cluster | k-mean | EM | Clarans | DBSCAN | MeanShift |
| Silhouette score | 0.9171 | 0.8114 | 0.53089 | 0.9357 | 0.7888 |
| Purity score | 0.2914 | 0.3058 | 0.20208 | 0.2488 | 0.9942 |

* **Feature case 4**

<K-mean>

1. Sum of distance in k-means (for elbow method)



|  |  |
| --- | --- |
| 2. k-means silhouette score | 3. k-means purity |
|  |  |

<EM clustering – GMM>

|  |  |
| --- | --- |
| 1. EM silhouette score | 2. EM purity |
|  |  |

<Clarans>

|  |  |
| --- | --- |
| 1. Clarans silhouette score | 2. Clarans purity |
|  |  |

<DBSCAN>

|  |  |
| --- | --- |
| 1. DBSCAN silhouette score | 2. DBSCAN purity |
|  |  |

<MeanShift>

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1. MeanShift silhouette score | | | 2. MeanShift purity | | | | |
|  | | |  | | | | |
| cluster | k-mean | EM | | Clarans | DBSCAN | MeanShift |
| Silhouette score | 0.7188 | 0.5542 | | 0.42186 | 0.9329 | 0.7783 |
| Purity score | 0.2069 | 0.3021 | | 0.21036 | 0.2665 | 1.0 |