PHW2

고효진 - 안세훈 - 안정우 - 허서윤

**1. Source Code**

| import numpy as np  import pandas as pd  import matplotlib.pyplot as plt  from matplotlib import cm  from sklearn.preprocessing import MinMaxScaler  from sklearn.preprocessing import StandardScaler  from sklearn.preprocessing import RobustScaler  from sklearn.preprocessing import MaxAbsScaler  from sklearn.preprocessing import LabelEncoder  from sklearn.preprocessing import OneHotEncoder  from sklearn.cluster import KMeans  from sklearn.cluster import DBSCAN  from sklearn.cluster import MeanShift  from sklearn.mixture import GaussianMixture  from sklearn.cluster import estimate\_bandwidth  from pyclustering.cluster.clarans import clarans;  from sklearn.metrics.cluster import homogeneity\_score, completeness\_score, v\_measure\_score  from sklearn import metrics  from sklearn.metrics import silhouette\_score  from sklearn.metrics import silhouette\_samples  from sklearn.model\_selection import GridSearchCV  from sklearn.neighbors import NearestNeighbors  from kneed import KneeLocator  import warnings  warnings.filterwarnings(action='ignore')  # Preprocessing  def findMissingValue(df):  # check missing value  # only 'totla\_bedrooms' has missing values, fill median  df.total\_bedrooms.fillna(df.total\_bedrooms.median(), inplace=True)  return df  # function for set hyper parameters and run find\_best  def setCombination():  # Scaler List  standard = StandardScaler()  minMax = MinMaxScaler()  robust = RobustScaler()  maxAbs = MaxAbsScaler()  scalers = {"standard scaler": standard, "minMax scaler": minMax, "robust scaler": robust, "maxAbs scaler": maxAbs}  # Encoder List  label = LabelEncoder()  oneHot = OneHotEncoder()  encoders = {"label encoder": label, "one-hot encoder": oneHot}  # Model List  kmeans = KMeans()  gmm = GaussianMixture()  dbscan = DBSCAN()  meanshift = MeanShift()  models = {"kmeans": kmeans,  #"clarans": [],  "gmm": gmm,  "dbscan": dbscan,  "meanshift": meanshift  }  # Parameters  params\_dict = {"kmeans": {"n\_clusters": [x for x in range(3, 5)]},  "gmm": {"n\_components": [x for x in range(3, 5)]},  "dbscan": {"eps": [0.1, 0.5]},  "meanshift": {"bandwidth": [1, 2]}  }  return scalers, encoders, models, params\_dict  def knee\_method(X):  nearest\_neighbors = NearestNeighbors(n\_neighbors=11)  neighbors = nearest\_neighbors.fit(X)  distances, indices = neighbors.kneighbors(X)  distances = np.sort(distances[:, 10], axis=0)  fig = plt.figure(figsize=(5, 5))  plt.plot(distances)  plt.xlabel("Points")  plt.ylabel("Distance")  plt.savefig("Distance\_curve.png", dpi=300)  plt.title("Distance curve")  plt.show()  i = np.arange(len(distances))  knee = KneeLocator(i, distances, S=1, curve='convex', direction='increasing', interp\_method='polynomial')  fig = plt.figure(figsize=(5, 5))  knee.plot\_knee()  plt.xlabel("Points")  plt.ylabel("Distance")  plt.show()  print(distances[knee.knee])  def purity\_scorer(target, y\_pred):  contingency\_matrix = metrics.cluster.contingency\_matrix(target, y\_pred)  score = np.sum(np.amax(contingency\_matrix, axis=0)) / np.sum(contingency\_matrix)  def silhouette\_scorer(estimator, X):  labels = estimator.fit\_predict(X)  score = silhouette\_score(X, labels, metric='euclidean')  return score  def display\_silhouette\_plot(X, labels):  sil\_score = metrics.silhouette\_score(X, labels, metric='euclidean')  n\_clusters\_ = len(set(labels)) - (1 if -1 in labels else 0)  print("For n\_clusters =", n\_clusters\_, "The average silhouette score is :", sil\_score)  # compute the silhouette scores for each sample  sample\_silhouette\_values = silhouette\_samples(X, labels)  fig, ax1 = plt.subplots()  fig.set\_size\_inches(18, 7)  ax1.set\_xlim([-0.1, 1])  ax1.set\_ylim([0, len(X) + (n\_clusters\_ + 1) \* 10])  y\_lower = 10  for i in range(n\_clusters\_):  ith\_cluster\_silhouette\_values = sample\_silhouette\_values[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)  ax1.text(-0.05, y\_lower + 0.5 \* size\_cluster\_i, str(i))  y\_lower = y\_upper + 10  ax1.set\_title("The silhouette plot for the various clusters")  ax1.set\_xlabel("The silhouette coefficient values")  ax1.set\_ylabel("Cluster label")  ax1.axvline(x=sil\_score, color="red", linestyle='--')  ax1.set\_yticks([])  ax1.set\_xticks([-0.1, 0, 0.2, 0.4, 0.6, 0.8, 1])  plt.suptitle(("Silhouette analysis for clustering on sample data with n\_clusters = %d" % n\_clusters\_),  fontsize=14, fontweight='bold')  plt.show()  return sil\_score  def featureCombination(df, index):  if index == 0:  feature = df[['housing\_median\_age', 'total\_rooms', 'total\_bedrooms']]  elif index == 1:  feature = df[['population', 'households', 'median\_income']]  elif index == 2:  feature = df[['population', 'ocean\_proximity']]  elif index == 3:  feature = df[['population', 'median\_income']]  elif index == 4:  feature = df[['households', 'median\_income']]  elif index == 5:  feature = df[['total\_rooms', 'total\_bedrooms']]  elif index == 6:  feature = df[['median\_income', 'ocean\_proximity']]  elif index == 7:  feature = df[['median\_income', 'total\_rooms']]  return feature  # function for store combination that has the best accuracy  def findBestCombination(df, scalers, encoders, models, params\_dict):  best\_combination = {}  best\_score = 0  # Sample Data  for index in range(1):  X = featureCombination(df, index)  feature = X.columns.tolist()  print(f'\n[feature: {feature}]')  num\_cols = X.select\_dtypes(include=['int64', 'float64']).columns.to\_list() # numerical value  cat\_cols = X.select\_dtypes(include=['object']).columns.to\_list() # categorical value  # find the best parameter by using grid search  for scaler\_key, scaler in scalers.items():  scaled\_X = scaler.fit\_transform(X[num\_cols])  print(f'\n[scaler: {scaler\_key}]')  # Label encoding if there are categorical values.  if 'ocean\_proximity' in X.columns :  label=LabelEncoder()    encoded\_X=label.fit\_transform(X[cat\_cols])  print(encoded\_X)  encoded\_X=encoded\_X.reshape(-1,1)    # add encoded\_x to scaled\_x  scaled\_X=np.concatenate((scaled\_X, encoded\_X), axis=1)  print(f'[encoder: label encoder]\n')  #print(scaled\_X )  knee\_method(scaled\_X)  for model\_key, model in models.items():  print(f'\n[model: {model\_key}]')  cv = [(slice(None), slice(None))]  if (model\_key == 'clarans'):  clarans\_obj = clarans(scaled\_X, 6, 3, 5)  clarans\_obj.process()  clst = clarans\_obj.get\_clusters()  label = []  for i in range(len(clst)):  for j in clst[i]:  label.insert(j, i)  if (best\_score < score):  best\_score = score  best\_X = scaled\_X  best\_label = label  target\_dict = {'silhouette': score,  'scaler': scaler\_key,  'model': model\_key,  'param': {'n\_cluster':6,'local minima':3},  'feature': feature  }      else: # Grid Search    grid = GridSearchCV(estimator=model,  param\_grid=params\_dict[model\_key],  scoring=silhouette\_scorer,  cv=cv)  grid.fit(scaled\_X)  print(f'best\_parameters: {grid.best\_params\_}')  score = grid.best\_score\_  if (best\_score < score):  best\_score = score  best\_X = scaled\_X  best\_label = grid.best\_estimator\_    target\_dict = {'silhouette': score,  'scaler': scaler\_key,  'model': model\_key,  'param': grid.best\_params\_,  'feature': feature  }    list\_size = 10  list\_size -= 1  flag = False  # save accuracy  if model\_key not in best\_combination.keys():  best\_combination[model\_key] = []  if len(best\_combination[model\_key]) <= list\_size:  best\_combination[model\_key].append(target\_dict)  # insert accuracy  elif best\_combination[model\_key][-1]['silhouette'] < score:  for i in range(1, list\_size):  if best\_combination[model\_key][list\_size - 1 - i]['silhouette'] > score:  best\_combination[model\_key].insert(list\_size - i, target\_dict)  best\_combination[model\_key].pop()  flag = True  break  if flag is False:  best\_combination[model\_key].insert(0, target\_dict)  best\_combination[model\_key].pop()  print(f'silhouette score: {score}', end='')  print()  return best\_combination, best\_X, best\_label  # read data  df = pd.read\_csv("housing.csv")  # preprocessing  df = findMissingValue(df)  # set scalers, models, params, k values  scalers, encoders, models, params\_dict = setCombination()  # get best combination dictionary  best\_result, best\_X, best\_label = findBestCombination(df, scalers, encoders, models, params\_dict)  print("\n\n-----------result-----------")  print("[Best Results]")  best\_score = 0  for model\_name, result\_list in best\_result.items():  print(model\_name)  for result in result\_list:  print(result)  if (best\_score < result['silhouette']):  best\_score = result['silhouette']  best\_combi = result  print()  print("[Best Combination]")  print(best\_combi)  display\_silhouette\_plot(best\_X, best\_label.fit\_predict(best\_X)) |
| --- |

**2. The structure of the program**

**<Used Functions>**

findMissingValue(df)

- drop NAN

- Split feature and target

- return drop\_df, numerical data, categorical data

setCombination(X, y)

- set scaler list

- set encoder list

- set model list

- set parameters

- return scalers, encoders, models, pramas\_dict

purity\_scorer(target, labels)

- return purity score

silhouette\_scorer(estimator, X)

- return silhouette score

dispaly\_silhouette\_plot(X, labels)

- display silhouette plot

- return silhouette score

featureCombination(df, index)

- select feature

- return selected\_feature

findBestCombination(df, scalers, encoders, models, params\_dict)

- find the best parameter by using grid search

- return best combination dictionary, best\_score for silhouette, best\_score for purity

**<Procedure (Main Function)>**

For in Features-> [N features]

For in Scalers -> [Standard Scaler, MinMax Scaler, Robust Scaler, MaxAbsScaler]

if there are categorical value, Label encoding

For in Models -> [kmeans, gmm, clarans, dbscan, meanshift

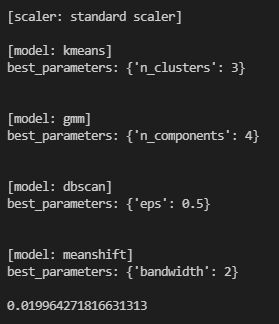
For in Parameters -> Grid Search(for each parameters)

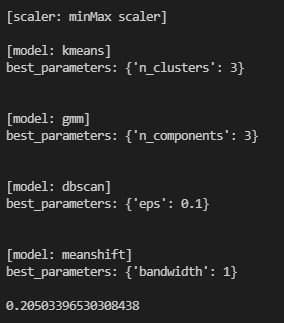
test method **-> silhouette, purity**

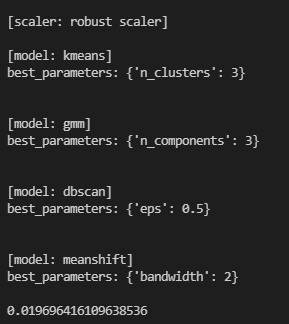
Print best combination

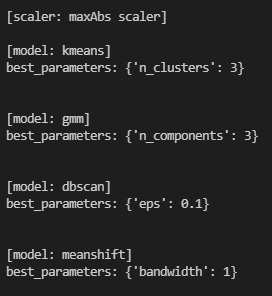
**3. Output**

Result for each iteration(representative image)

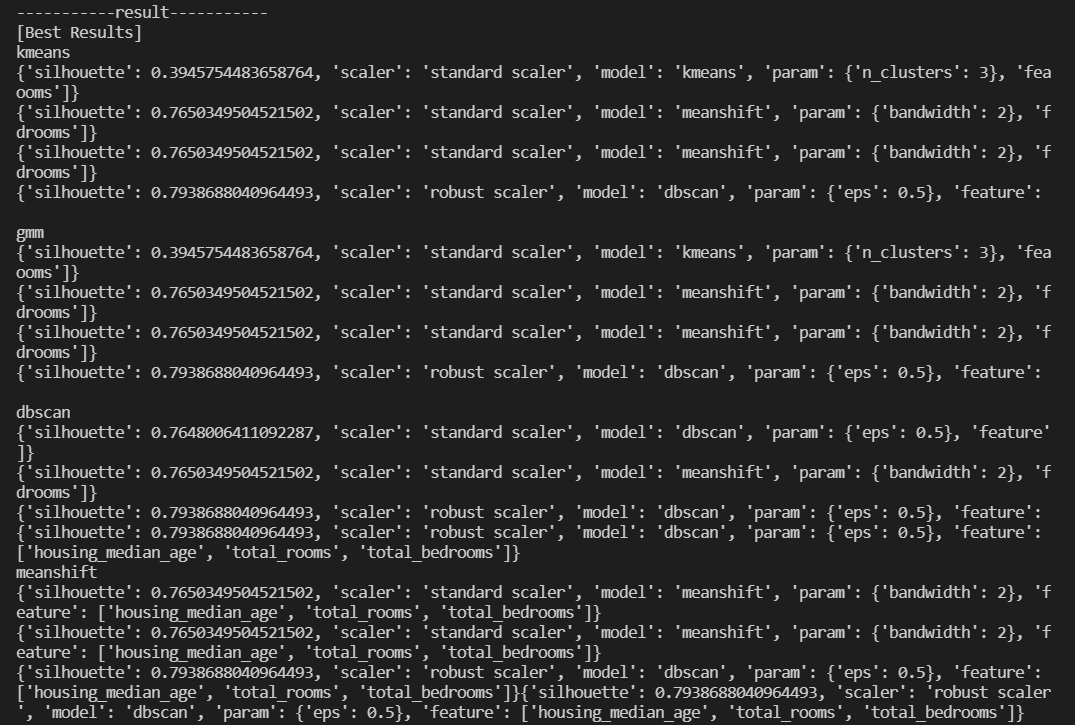


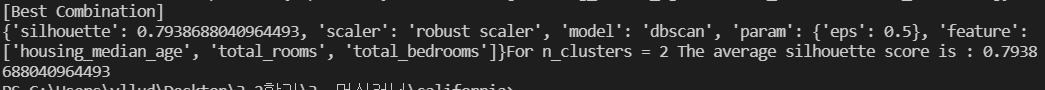




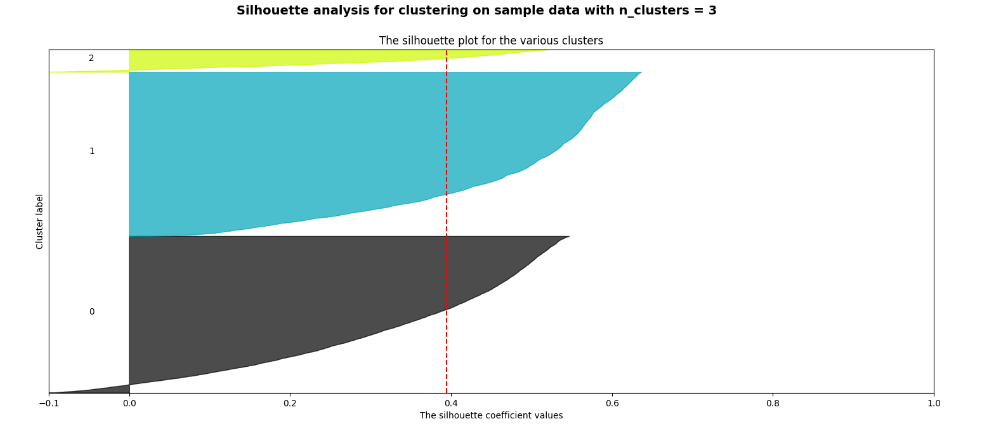


**Best result**

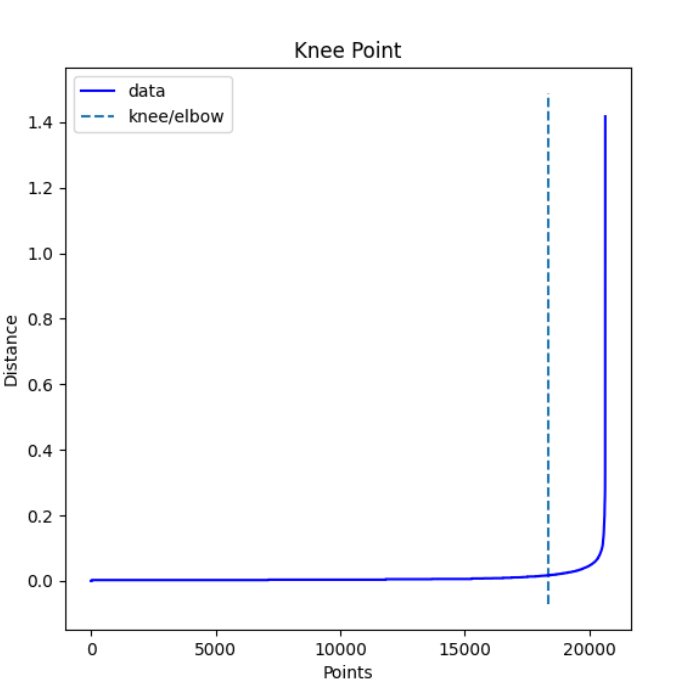




**silhouette score for best parameter**



**Knee point**



**4. Result**

Best combination feature is house\_median\_age, total\_rooms, total\_bedrooms. These are features related to **the condition of the house**. Best combination scaler is robust scaler, and clustering model is dbscan, param is 0.5. And cluster value is 2, average silhouette score is 0.7938688040964493.