Homework06

November 19, 2021

[25]:
```
from sklearn import metrics
from sklearn.metrics import pairwise_distances
from sklearn import datasets
from sklearn.cluster import KMeans
import numpy as np

def kmeanalgor():

    X = np.array([[1,2],[1,4],[1,0],[10,2],[10,4],[10,0]])
    kmeans = KMeans(n_clusters = 2, random_state = 0).fit(X)
    labels = kmeans.labels_
    centers = kmeans.cluster_centers_
    predicted = kmeans.predict([[0,0],[12,3]])
    print("Labels: ",labels)
    print("Cluster centers: ", centers)
    print("Predicted class: ",predicted)

kmeanalgor()

##2

from sklearn import metrics

def RandIndex():

    labels_true = [0,0,0,1,1,1]
    labels_pred = [0,0,1,1,2,2]

    rand_index = metrics.rand_score(labels_true, labels_pred)
    adjusted_rand_index = metrics.adjusted_rand_score(labels_true, labels_pred)
    print("Rand_index: ",rand_index)
    print("Adjusted_rand_index: ", adjusted_rand_index)

RandIndex()

def MutualInfo():
```

```
    labels_true = [0,0,0,1,1,1]
    labels_pred = [0,0,1,1,2,2]

    ami = metrics.adjusted_mutual_info_score(labels_true,labels_pred)
    nmi = metrics.normalized_mutual_info_score(labels_true,labels_pred)
    mis = metrics.mutual_info_score(labels_true, labels_pred)

    print("Adjusted Mutual Info Score: ", ami)
    print("Normalized Mutual Info Score: ", nmi)
    print("Mutual Info Score: ",mis)

MutualInfo()

## 3

def Threescores():

    labels_true = [0,0,0,1,1,1]
    labels_pred = [0,0,1,1,2,2]

    h = metrics.homogeneity_score(labels_true, labels_pred)
    c = metrics.completeness_score(labels_true, labels_pred)
    v = metrics.v_measure_score(labels_true, labels_pred)
    hcv = metrics.homogeneity_completeness_v_measure(labels_true, labels_pred)

    print("Homogeneity Score: ", h)
    print("Completeness Score: ", c)
    print("V_measure Score: ", v)
    print("Homogeneity, comleteness, and V-measure: ", hcv)

Threescores()

## 4

def FMIscore():

    labels_true = [0,0,0,1,1,1]
    labels_pred = [0,0,1,1,2,2]

    fmi = metrics.fowlkes_mallows_score(labels_true, labels_pred)

    print("Fowlkes-Mallows score: ", fmi)

FMIscore()

## 5
```

```
def Sscore():

    X,y = datasets.load_iris(return_X_y=True)
    kmeans_model = KMeans(n_clusters=3, random_state=1).fit(X)
    labels = kmeans_model.labels_
    s = metrics.silhouette_score(X,labels)
    print("Silhouette Score: ",s)

Sscore()

## 6

def CHscore():

    X,y = datasets.load_iris(return_X_y=True)
    kmeans_model = KMeans(n_clusters=3, random_state=1).fit(X)
    labels = kmeans_model.labels_
    ch = metrics.calinski_harabasz_score(X,labels)
    print("Calinski-Harabasz Score: ",ch)

CHscore()

## 7

def DBscore():

    X,y = datasets.load_iris(return_X_y=True)
    kmeans_model = KMeans(n_clusters=3, random_state=1).fit(X)
    labels = kmeans_model.labels_
    db = metrics.davies_bouldin_score(X,labels)
    print("Davies Bouldin Score: ",db)

DBscore()

## Mining

from os import listdir
from os.path import isfile, join
import random

def read_data(train_path_class0, train_path_class1,test_path_class0,␣
 ↪test_path_class1):
    #read X_train, Y_train
    X_train_class0 = [open(train_path_class0 + "/" + f, encoding = "utf-8",␣
 ↪errors = "ignore").read() for f in listdir(train_path_class0) if␣
 ↪isfile(join(train_path_class0,f))]
    Y_train_class0 = [0]*len(X_train_class0)
```

```
    X_train_class1 = [open(train_path_class1 + "/" + f, encoding = "utf-8",␣
 ↪errors = "ignore").read() for f in listdir(train_path_class1) if␣
 ↪isfile(join(train_path_class1,f))]
    Y_train_class1 = [1]*len(X_train_class1)
    X_train = X_train_class0 + X_train_class1
    Y_train = Y_train_class0 + Y_train_class1

    #shuffle X_train and Y_train
    Z = list(zip(X_train, Y_train))
    random.shuffle(Z)
    X_train, Y_train = zip(*Z)

    #read X_test, Y_test
    X_test_class0 = [open(test_path_class0 + "/" + f, encoding = "utf-8",␣
 ↪errors = "ignore").read() for f in listdir(test_path_class0) if␣
 ↪isfile(join(test_path_class0,f))]
    Y_test_class0 = [0]*len(X_test_class0)
    X_test_class1 = [open(test_path_class1 + "/" + f, encoding = "utf-8",␣
 ↪errors = "ignore").read() for f in listdir(test_path_class1) if␣
 ↪isfile(join(test_path_class1,f))]
    Y_test_class1 = [1]*len(X_test_class1)
    X_test = X_test_class0 + X_test_class1
    Y_test = Y_test_class0 + Y_test_class1

    #shuffle X_test and Y_test
    Z = list(zip(X_test, Y_test))
    random.shuffle(Z)
    X_test,Y_test = zip(*Z)

    print("***read data***")
    return X_train, Y_train, X_test, Y_test

train_path_class0 = "C:/Users/Carl/Desktop/PythonV/HW6/20news-bydate/
 ↪20news-bydate-train/alt.atheism"

train_path_class1 = "C:/Users/Carl/Desktop/PythonV/HW6/20news-bydate/
 ↪20news-bydate-train/comp.graphics"

test_path_class0 = "C:/Users/Carl/Desktop/PythonV/HW6/20news-bydate/
 ↪20news-bydate-test/alt.atheism"

test_path_class1 = "C:/Users/Carl/Desktop/PythonV/HW6/20news-bydate/
 ↪20news-bydate-test/comp.graphics"

X_train, Y_train, X_test, Y_test = read_data(train_path_class0,␣
 ↪train_path_class1, test_path_class0, test_path_class1)
```

```
## Use naive bayes

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score

def train(X_train, Y_trian, X_test, Y_test):
    #Tokenizing the texts
    count_vect = CountVectorizer()
    X_train_counts = count_vect.fit_transform(X_train)

    #Calculate TfidfTransformer
    tfidf_transformer = TfidfTransformer()
    X_train_tfidf = tfidf_transformer.fit_transform(X_train_counts)

    #Learning
    clf = MultinomialNB().fit(X_train_tfidf,Y_train)

    #Predicting
    X_test_counts = count_vect.transform(X_test)
    X_test_tfidf = tfidf_transformer.transform(X_test_counts)
    Y_predicted = clf.predict(X_test_tfidf)

    print(accuracy_score(Y_test,Y_predicted))

train(X_train, Y_train, X_test, Y_test)



####################----------------------------------##############################

### Question 2

print ("******* THIS IS QUESTION 2 *********")

from sklearn import metrics

def Question2():

    # k_means
    X,y = datasets.load_wine(return_X_y=True)

    kmeans = KMeans(n_clusters = 3, random_state = 0).fit(X)
    labels = kmeans.labels_
    centers = kmeans.cluster_centers_
```

```
#     print("Labels: ",labels)
#     print("Cluster centers: ", centers)

    labels_true = labels
    labels_pred = y
    rand_index = metrics.rand_score(labels_true, labels_pred)
    adjusted_rand_index = metrics.adjusted_rand_score(labels_true, labels_pred)
    print("Rand_index: ",rand_index)
    print("Adjusted_rand_index: ", adjusted_rand_index)

    ami = metrics.adjusted_mutual_info_score(labels_true,labels_pred)
    nmi = metrics.normalized_mutual_info_score(labels_true,labels_pred)
    mis = metrics.mutual_info_score(labels_true, labels_pred)

    print("Adjusted Mutual Info Score: ", ami)
    print("Normalized Mutual Info Score: ", nmi)
    print("Mutual Info Score: ",mis)

    h = metrics.homogeneity_score(labels_true, labels_pred)
    c = metrics.completeness_score(labels_true, labels_pred)
    v = metrics.v_measure_score(labels_true, labels_pred)
    hcv = metrics.homogeneity_completeness_v_measure(labels_true, labels_pred)

    print("Homogeneity Score: ", h)
    print("Completeness Score: ", c)
    print("V_measure Score: ", v)
    print("Homogeneity, comleteness, and V-measure: ", hcv)

    fmi = metrics.fowlkes_mallows_score(labels_true, labels_pred)

    print("Fowlkes-Mallows score: ", fmi)

Question2()

### Question 3

print ("******* THIS IS QUESTION 3 *********")

from os import listdir
from os.path import isfile, join
import random

def read_data(train_path_class0, train_path_class1,test_path_class0,␣
 ↪test_path_class1):
    #read X_train, Y_train
```

```
    X_train_class0 = [open(train_path_class0 + "/" + f, encoding = "utf-8",␣
↪errors = "ignore").read() for f in listdir(train_path_class0) if␣
↪isfile(join(train_path_class0,f))]
    Y_train_class0 = [0]*len(X_train_class0)
    X_train_class1 = [open(train_path_class1 + "/" + f, encoding = "utf-8",␣
↪errors = "ignore").read() for f in listdir(train_path_class1) if␣
↪isfile(join(train_path_class1,f))]
    Y_train_class1 = [1]*len(X_train_class1)
    X_train = X_train_class0 + X_train_class1
    Y_train = Y_train_class0 + Y_train_class1

    #shuffle X_train and Y_train
    Z = list(zip(X_train, Y_train))
    random.shuffle(Z)
    X_train, Y_train = zip(*Z)

    #read X_test, Y_test
    X_test_class0 = [open(test_path_class0 + "/" + f, encoding = "utf-8",␣
↪errors = "ignore").read() for f in listdir(test_path_class0) if␣
↪isfile(join(test_path_class0,f))]
    Y_test_class0 = [0]*len(X_test_class0)
    X_test_class1 = [open(test_path_class1 + "/" + f, encoding = "utf-8",␣
↪errors = "ignore").read() for f in listdir(test_path_class1) if␣
↪isfile(join(test_path_class1,f))]
    Y_test_class1 = [1]*len(X_test_class1)
    X_test = X_test_class0 + X_test_class1
    Y_test = Y_test_class0 + Y_test_class1

    #shuffle X_test and Y_test
    Z = list(zip(X_test, Y_test))
    random.shuffle(Z)
    X_test,Y_test = zip(*Z)

    print("***reading data***")
    return X_train, Y_train, X_test, Y_test

train_path_class0 = "C:/Users/Carl/Desktop/PythonV/HW6/20news-bydate/
↪20news-bydate-train/alt.atheism"

train_path_class1 = "C:/Users/Carl/Desktop/PythonV/HW6/20news-bydate/
↪20news-bydate-train/comp.graphics"

test_path_class0 = "C:/Users/Carl/Desktop/PythonV/HW6/20news-bydate/
↪20news-bydate-test/alt.atheism"
```

```
test_path_class1 = "C:/Users/Carl/Desktop/PythonV/HW6/20news-bydate/
 ↪20news-bydate-test/comp.graphics"

X_train, Y_train, X_test, Y_test = read_data(train_path_class0,␣
 ↪train_path_class1, test_path_class0, test_path_class1)

### Use naive bayes

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.naive_bayes import MultinomialNB
from sklearn.naive_bayes import GaussianNB
from sklearn.naive_bayes import BernoulliNB
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.ensemble import BaggingClassifier
from sklearn import tree
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score

def train(X_train, Y_trian, X_test, Y_test):
    #Tokenizing the texts
    count_vect = CountVectorizer()
    X_train_counts = count_vect.fit_transform(X_train)

    #Calculate TfidfTransformer
    tfidf_transformer = TfidfTransformer()
    X_train_tfidf = tfidf_transformer.fit_transform(X_train_counts)

    #LearningList
    clf = MultinomialNB().fit(X_train_tfidf,Y_train)
    gas = GaussianNB().fit(X_train_tfidf.toarray(),Y_train)
    ber = BernoulliNB().fit(X_train_tfidf,Y_train)
    treetree = tree.DecisionTreeClassifier().fit(X_train_tfidf,Y_train)
    nmo = MLPClassifier().fit(X_train_tfidf,Y_train)
    ada = AdaBoostClassifier(n_estimators=100).fit(X_train_tfidf,Y_train)
    gb = GradientBoostingClassifier(n_estimators=50).fit(X_train_tfidf,Y_train)
    rf = RandomForestClassifier(n_estimators=50).fit(X_train_tfidf,Y_train)
    et = ExtraTreesClassifier(n_estimators=100).fit(X_train_tfidf,Y_train)
    bag = BaggingClassifier(n_estimators=100).fit(X_train_tfidf,Y_train)

    #Predicting
    X_test_counts = count_vect.transform(X_test)
    X_test_tfidf = tfidf_transformer.transform(X_test_counts)
    ### Predict List
```

```
    Y_predicted = clf.predict(X_test_tfidf)
    Y_gas = gas.predict(X_test_tfidf.toarray())
    Y_ber = ber.predict(X_test_tfidf)
    Y_treetree = treetree.predict(X_test_tfidf)
    Y_nmo = nmo.predict(X_test_tfidf)
    Y_ada = ada.predict(X_test_tfidf)
    Y_gb = gb.predict(X_test_tfidf)
    Y_rf = rf.predict(X_test_tfidf)
    Y_et = et.predict(X_test_tfidf)
    Y_bag = bag.predict(X_test_tfidf)

    print("This is MultinomialNB -->", accuracy_score(Y_test,Y_predicted))
    print("This is GaussianNB -->", accuracy_score(Y_test,Y_gas))
    print("This is BernoulliNB -->", accuracy_score(Y_test,Y_ber))
    print("This is DecisionTreeClassifier -->",␣
↪accuracy_score(Y_test,Y_treetree))
    print("This is MLPClassifier -->", accuracy_score(Y_test,Y_nmo))
    print("This is AdaBoostClassifier -->", accuracy_score(Y_test,Y_ada))
    print("This is GradientBoostingClassifier -->", accuracy_score(Y_test,Y_gb))
    print("This is RandomForestClassifier -->", accuracy_score(Y_test,Y_rf))
    print("This is ExtraTreesClassifier -->", accuracy_score(Y_test,Y_et))
    print("This is BaggingClassifier -->", accuracy_score(Y_test,Y_bag))
    Acc_list =␣
↪[accuracy_score(Y_test,Y_predicted),accuracy_score(Y_test,Y_gas),accuracy_score(Y_test,Y_be

    print("Max accuracy is : --->", max(Acc_list), " which is MLPClassifier ",␣
↪Acc_list.index(max(Acc_list))+1, "th one")

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

```
Labels:  [1 1 1 0 0 0]
Cluster centers:  [[10.  2.]
 [ 1.  2.]]
Predicted class:  [1 0]
Rand_index:  0.6666666666666666
Adjusted_rand_index:  0.24242424242424243
Adjusted Mutual Info Score:  0.2987924581708901
Normalized Mutual Info Score:  0.5158037429793889
Mutual Info Score:  0.4620981203732969
Homogeneity Score:  0.6666666666666669
Completeness Score:  0.420619835714305
V_measure Score:  0.5158037429793889
Homogeneity, comleteness, and V-measure:  (0.6666666666666669,
0.420619835714305, 0.5158037429793889)
Fowlkes-Mallows score:  0.4714045207910317
Silhouette Score:  0.5528190123564091
Calinski-Harabasz Score:  561.62775662962
```

```
Davies Bouldin Score:  0.6619715465007542
***read data***
0.9689703808180536
******* THIS IS QUESTION 2 *********
Rand_index:  0.718656763791024
Adjusted_rand_index:  0.37111371823084754
Adjusted Mutual Info Score:  0.4226866642766121
Normalized Mutual Info Score:  0.4287568597645354
Mutual Info Score:  0.4657066646034707
Homogeneity Score:  0.42870141389448585
Completeness Score:  0.42881231997856467
V_measure Score:  0.4287568597645355
Homogeneity, comleteness, and V-measure:  (0.42870141389448585,
0.42881231997856467, 0.4287568597645355)
Fowlkes-Mallows score:  0.5835370218944976
******* THIS IS QUESTION 3 *********
***reading data***
This is MultinomialNB --> 0.9689703808180536
This is GaussianNB --> 0.9619181946403385
This is BernoulliNB --> 0.9280677009873061
This is DecisionTreeClassifier --> 0.8758815232722144
This is MLPClassifier --> 0.9788434414668548
This is AdaBoostClassifier --> 0.9294781382228491
This is GradientBoostingClassifier --> 0.9365303244005642
This is RandomForestClassifier --> 0.9365303244005642
This is ExtraTreesClassifier --> 0.9421720733427362
This is BaggingClassifier --> 0.9351198871650211
Max accuracy is : ---> 0.9788434414668548  which is MLPClassifier  5 th one
```

[ ]:

[ ]: