MACHINE LEARNING ASSIGNMENT – 05

Praveena Goli(700743010)

1. Programming elements: Clustering & Dimensionality reduction

In class programming:

1. Principal Component Analysis a. Apply PCA on CC dataset. b. Apply k-means algorithm on the PCA result and report your observation if the silhouette score has improved or not? c. Perform Scaling+PCA+K-Means and report performance.

import pandas as pd

import numpy as np

from sklearn.preprocessing import StandardScaler

from sklearn.decomposition import PCA

# Load the dataset

df = pd.read\_csv('/content/CC GENERAL.csv')

# Drop the irrelevant columns

df.drop(['CUST\_ID', 'TENURE'], axis=1, inplace=True)

# Fill the missing values with the column mean

df.fillna(df.mean(), inplace=True)

# Standardize the data

scaler = StandardScaler()

df\_scaled = scaler.fit\_transform(df)

# Apply PCA

pca = PCA(n\_components=10)

pca.fit(df\_scaled)

# Get the explained variance ratio

explained\_variance\_ratio = pca.explained\_variance\_ratio\_

# Get the cumulative sum of explained variance

cumulative\_explained\_variance\_ratio = np.cumsum(explained\_variance\_ratio)

# Print the explained variance ratios and the cumulative sum

print(explained\_variance\_ratio)

print(cumulative\_explained\_variance\_ratio)

[0.28845814 0.21570572 0.09330079 0.07548528 0.06652726 0.05389941

0.04544392 0.04156174 0.03280202 0.02534919]

[0.28845814 0.50416386 0.59746464 0.67294993 0.73947718 0.7933766

0.83882052 0.88038226 0.91318428 0.93853347]

The output of the above code will be a plot of the silhouette scores for each k. Based on the plot, you can choose the number of clusters. The higher the silhouette score, the better the clustering.

If the silhouette score has improved after applying PCA, it means that the PCA has reduced the dimensionality of the data while retaining most of the information. This can lead to better clustering performance and faster computation time.

from sklearn.cluster import KMeans

from sklearn.metrics import silhouette\_score

# Apply PCA

pca = PCA(n\_components=2)

pca\_result = pca.fit\_transform(df\_scaled)

# Apply k-means for k=2 to 10 and get the silhouette scores

silhouette\_scores = []

for k in range(2, 11):

    kmeans = KMeans(n\_clusters=k, random\_state=42)

    kmeans.fit(pca\_result)

    score = silhouette\_score(pca\_result, kmeans.labels\_)

    silhouette\_scores.append(score)

    print(f"k={k}, silhouette score={score}")

import matplotlib.pyplot as plt

# Plot the silhouette scores

plt.plot(range(2, 11), silhouette\_scores)

plt.xlabel('Number of Clusters')

plt.ylabel('Silhouette Score')

plt.show()

import pandas as pd

import numpy as np

from sklearn.preprocessing import StandardScaler

from sklearn.decomposition import PCA

from sklearn.cluster import KMeans

from sklearn.metrics import silhouette\_score

# Load the dataset

df = pd.read\_csv('/content/CC GENERAL.csv')

# Drop the irrelevant columns

df.drop(['CUST\_ID', 'TENURE'], axis=1, inplace=True)

# Fill the missing values with the column mean

df.fillna(df.mean(), inplace=True)

# Scale the data

scaler = StandardScaler()

df\_scaled = scaler.fit\_transform(df)

# Perform PCA

pca = PCA(n\_components=10)

df\_pca = pca.fit\_transform(df\_scaled)

# Apply K-Means clustering

kmeans = KMeans(n\_clusters=5, random\_state=42)

kmeans.fit(df\_pca)

labels = kmeans.labels\_

# Calculate the silhouette score

silhouette\_avg = silhouette\_score(df\_pca, labels)

print(f"The average silhouette score is : {silhouette\_avg}")

The average silhouette score is : 0.2314725220610872

Graphical user interface, application

Description automatically generated

the average silhouette score for the clustering. The higher the silhouette score, the better the clustering performance.

1. Use pd\_speech\_features.csv
2. a. Perform Scaling b. Apply PCA (k=3)
3. c. Use SVM to report performance

import pandas as pd

df = pd.read\_csv('/content/pd\_speech\_features.csv', header=1)

X = df.iloc[:, :-1].values

y = df.iloc[:, -1].values

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

from sklearn.decomposition import PCA

pca = PCA(n\_components=3)

X\_pca = pca.fit\_transform(X\_scaled)

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_pca, y, test\_size=0.2, random\_state=42)

from sklearn.svm import SVC

from sklearn.metrics import accuracy\_score

svm = SVC(kernel='rbf', random\_state=42)

svm.fit(X\_train, y\_train)

y\_pred = svm.predict(X\_test)

accuracy = accuracy\_score(y\_test, y\_pred)

print(f"Accuracy: {accuracy}")

Accuracy: 0.8013245033112583

The **accuracy** variable contains the accuracy of the SVM model on the testing data. You can adjust the SVM hyperparameters and PCA parameters to try to improve the accuracy of the model.

Note that the performance of SVM depends on the choice of hyperparameters and the data itself. It's always a good idea to cross-validate the model to get a more accurate estimate of its performance.

1. Apply Linear Discriminant Analysis (LDA) on Iris.csv dataset to reduce dimensionality of data to k=2

import pandas as pd

df = pd.read\_csv('/content/Iris.csv')

X = df.iloc[:, :-1].values

y = df.iloc[:, -1].values

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

from sklearn.discriminant\_analysis import LinearDiscriminantAnalysis as LDA

lda = LDA(n\_components=2)

X\_lda = lda.fit\_transform(X\_scaled, y)

Here, we set **n\_components=2** to reduce the dimensionality of the data to 2. The **fit\_transform()** method applies LDA on the scaled features and returns the transformed features with reduced dimensionality.

Now, the **X\_lda** array contains the transformed features with reduced dimensionality. We can use this array as input to various machine learning algorithms.

Note that LDA is a supervised method and requires the labels of the data to be known. rIn this case, the **y** variable contains the labels of the **Iris.csv** dataset.

1. Briefly identify the difference between PCA and LDA

PCA (Principal Component Analysis) and LDA (Linear Discriminant Analysis) are both methods for dimensionality reduction, but they have different objectives and are used in different scenarios.

PCA is an unsupervised method that seeks to find the most important features or directions in the data that capture the maximum amount of variance. It does not take into account the labels of the data and simply tries to find a low-dimensional representation of the data that preserves as much information as possible. PCA is often used for data visualization, noise reduction, and feature extraction.

LDA, on the other hand, is a supervised method that seeks to find the most discriminative features or directions in the data that maximize the separation between the classes. It takes into account the labels of the data and tries to find a low-dimensional representation of the data that maximizes the inter-class distance and minimizes the intra-class distance. LDA is often used for classification and feature extraction.

In summary, while both PCA and LDA are methods for dimensionality reduction, PCA is an unsupervised method that seeks to capture the maximum amount of variance in the data, while LDA is a supervised method that seeks to maximize the separation between the classes in the data.

Git Hub Link: https://github.com/Goli18/ML\_05.git

Video Link : https://github.com/Goli18/ML\_05/blob/main/ML\_05%20-%20Word%202023-04-12%2022-57-59.mp4