MTH 522: Advanced Mathematical Statistics

Applying Different techniques to US Arrests Data

04/23/2023

Issues

In this report, we have utilized different techniques, including two types of clustering and a principal component analysis, to categorize and label the data without explicitly telling the model what the data means or labelling it. Clustering involves grouping data points together into clusters based on their proximity to one another, with two types of clustering techniques, k-means clustering and hierarchical clustering, employed in this analysis. The primary difference between these techniques is that k-means clustering requires the number of clusters to be specified in advance, while hierarchical clustering is a bottom-up approach that starts with a cluster around every point and expands the clusters until there is one cluster surrounding the entire dataset, without requiring the number of clusters to be predetermined.

Principal component analysis is another important technique that allows for the representation of data using fewer variables than the original dataset. By reducing the number of variables, we can represent the data more succinctly, making it easier to analyze and interpret.

Such techniques enable the computer to find patterns in the data that may not be immediately apparent to humans, and these patterns can then be used to create a model based on the data. In the context of the US arrest dataset, we faced the challenge of finding connections between states and specific crimes, including Assault, Rape, Murder, and Urban Pop.

Findings

Beginning with the principal component analysis, it was discovered that our PC1, or principal component 1, concentrated on the relationships between the variables of murder, rape, and assault and was able to account for 62% of the variation in the original data by itself. Then there was PC2, which focused almost exclusively on the Urban Pop variable, accounting for a further 24% of the data; so, by themselves, these two variables could account for around 86% of the data. Additionally, a visualization revealed that states like New York and California flow in the direction of the Rape variable, indicating that rape is prevalent in these states.

Moving on to the clustering, we began by utilizing k-means clustering. Here, we discovered that around four clusters, with k = 4, looked to cluster this US Arrest data well because the clusters were separated well and built around various groups of the data. Four clusters were found to be a good fit for this dataset after testing various k values and cluster sizes. Hierarchical clustering also confirms this. A cluster size of four was an acceptable cut off for clustering our data after we constructed our hierarchical cluster and showed a dendrogram.

Appendix A: Methods

The first step is to import the dataset using the pandas read_csv() function and assign it to the variable 'arrests_data'. Then the data is standardized using the StandardScaler() function from the scikit-learn library, which transforms the data so that each variable has a mean of 0 and a standard deviation of 1. The standardized data is then assigned to 'arrests_data_scaled'. Next, the PCA object is created and fitted to the standardized data using the fit_transform() method. This creates a new set of variables, called principal components (PCs), which are linear combinations of the original variables. A scatter plot is then created with PC1 on the x-axis and PC2 on the y-axis, using the scatter() method from matplotlib.pyplot library. The scatter plot shows the scores of each observation (state) on the two principal components. Arrow plots are added to show the relationship between the original variables and the principal components. The variables' coefficients for PC1 and PC2 are calculated using the pca.components_.T method and assigned to 'pcs'. For each variable, an arrow is drawn on the graph with its x and y values corresponding to its coefficients on PC1 and PC2. The variable name is added to the arrow using the text() method. The state names are used as labels for the scatter plot points using the text() method. The plot's axes are labeled and titled using the set_xlabel(), set_ylabel() and set_title() methods. Finally, the plot is displayed using the show() method from matplotlib.pyplot library. The loadings, or coefficients, of each variable for each component are printed using the pandas DataFrame method and assigned to 'loadings'. The amount of variance explained by each principal component is printed using the explained_variance_ratio_ attribute of the PCA object. Overall, the code performs PCA on the USArrests dataset and visualizes the scores of each state on the first two principal components, as well as the relationship between the original variables and the principal components. The loadings of each variable on each principal component are also printed, along with the amount of variance explained by each component.

We will talk about k-means clustering on the USArrests dataset with 2, 3, and 4 clusters. The algorithm is implemented using scikit-learn's KMeans function with the n_clusters parameter set to the number of clusters desired. The data is passed to the function using the arrests_data dataframe. For each cluster number, the code creates a scatter plot of the data points, with the x-axis representing the Murder variable and the y-axis representing the Assault variable. Each data point is colored based on the cluster label assigned by the k-means algorithm. The legend identifies the colors with the cluster labels. To visually distinguish the clusters, the code adds a border around each cluster grouping. This is accomplished by calculating the minimum and maximum values for the Murder and Assault variables within each cluster and drawing a rectangle around these values using the plt.Rectangle function.

Next, we will talk about hierarchical clustering on the "arrests_data" dataset using the complete linkage method. The first step is to create a linkage matrix using the "linkage" function from the "scipy.cluster.hierarchy" module. The linkage matrix contains information about the hierarchical clustering structure of the data, and it is

computed using the pairwise distances between observations. In this case, the "complete" method is used to calculate the distances, which uses the maximum distance between all pairs of observations in the two clusters being merged. The resulting linkage matrix is stored in the "hier_cluster" variable. Next, a dendrogram is plotted using the "dendrogram" function from the same module. The dendrogram is a tree-like diagram that shows the hierarchical relationships between the observations. The x-axis shows the observations, and the y-axis represents the distance between them. The "truncate_mode" parameter is used to cut the tree at a certain level to make it easier to interpret. Here, the "level" option is used with a "p" value of 50, which means that the tree will be cut at a distance corresponding to the 50th percentile of all distances. The "leaf_rotation" and "leaf_font_size" parameters are used to make the labels of the observations easier to read. After the dendrogram is plotted, the "cut_tree" function is used to cut the tree into four clusters. The resulting cluster labels are stored in the "clusters" variable. The "cut_tree" function cuts the tree at a certain level based on the number of clusters specified by the "n_clusters" parameter.

Appendix B: Results

	PC1	PC2	PC3	PC4
Murder	0.535899	0.418181	-0.341233	0.649228
Assault	0.583184	0.187986	-0.268148	-0.743407
UrbanPop	0.278191	-0.872806	-0.378016	0.133878
Rape	0.543432	-0.167319	0.817778	0.089024

Fig: Coefficients of each variable

```
Explained variance by PC1: 62.01%
Explained variance by PC2: 24.74%
Explained variance by PC3: 8.91%
Explained variance by PC4: 4.34%
```

Fig: Total variance for each principal component

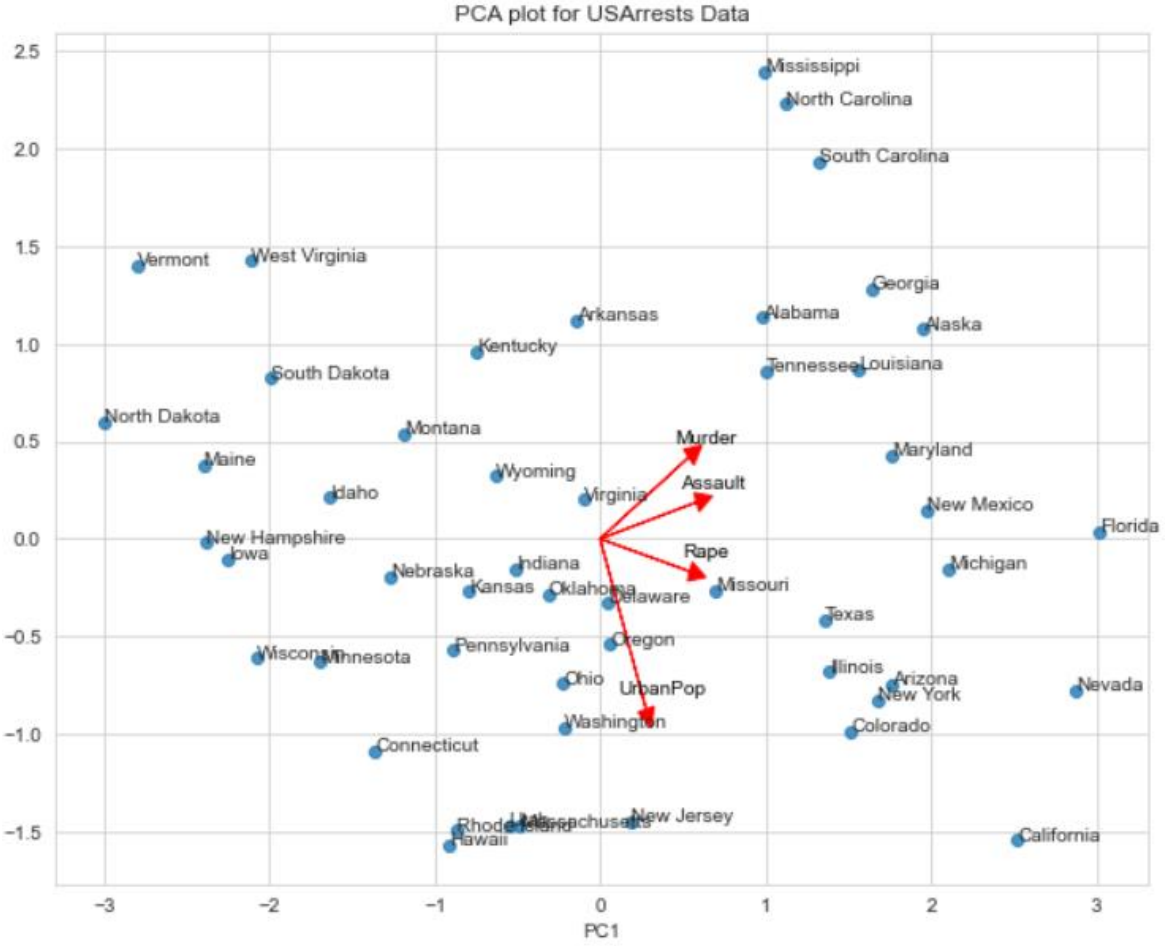
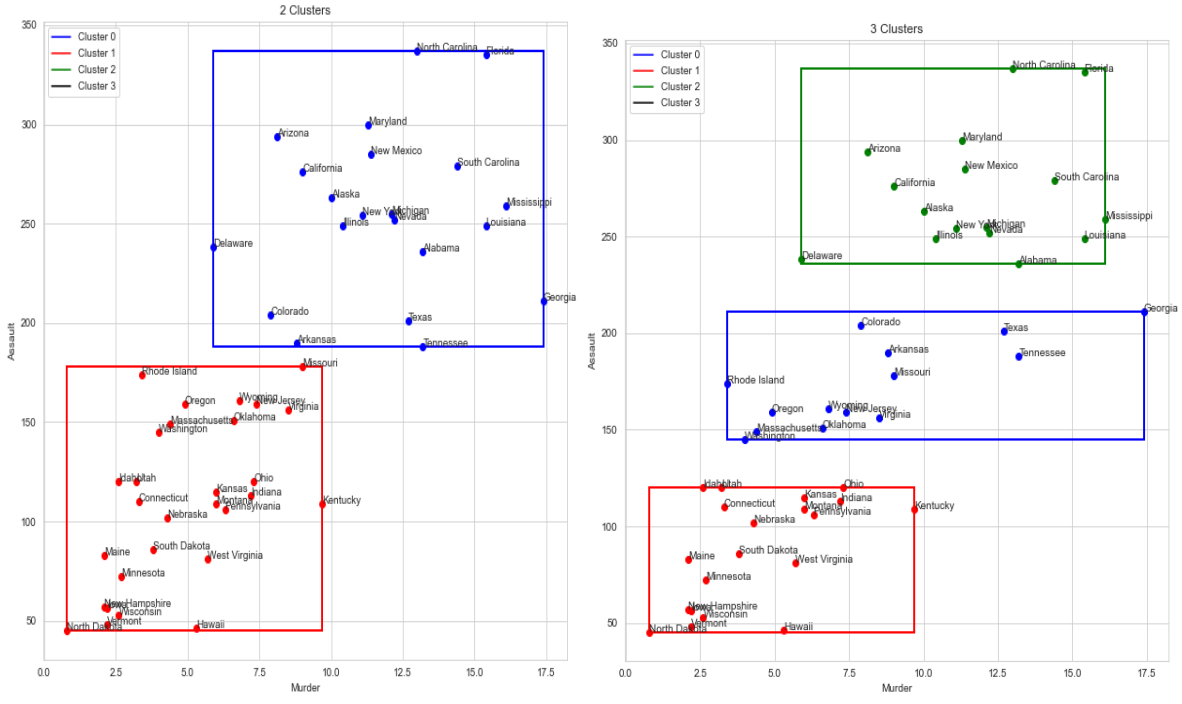
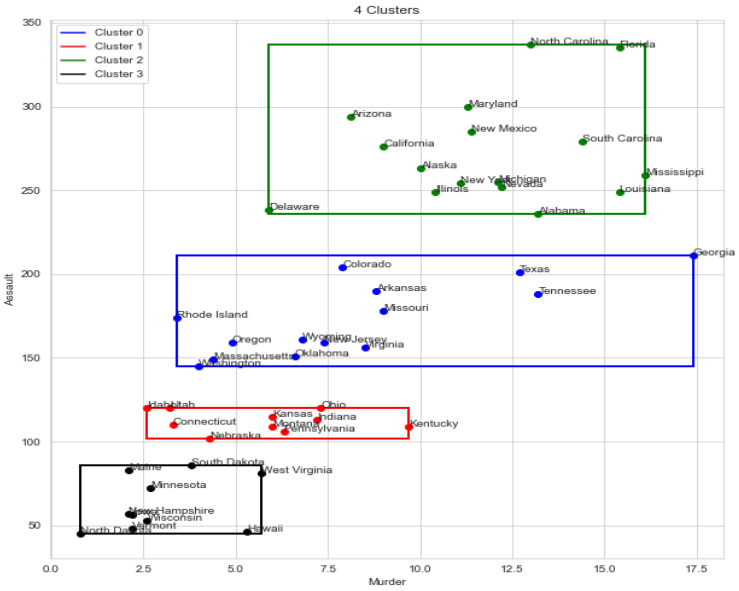


Fig: Plot for PCA analysis





k-mean Clusters for k=2,3,4

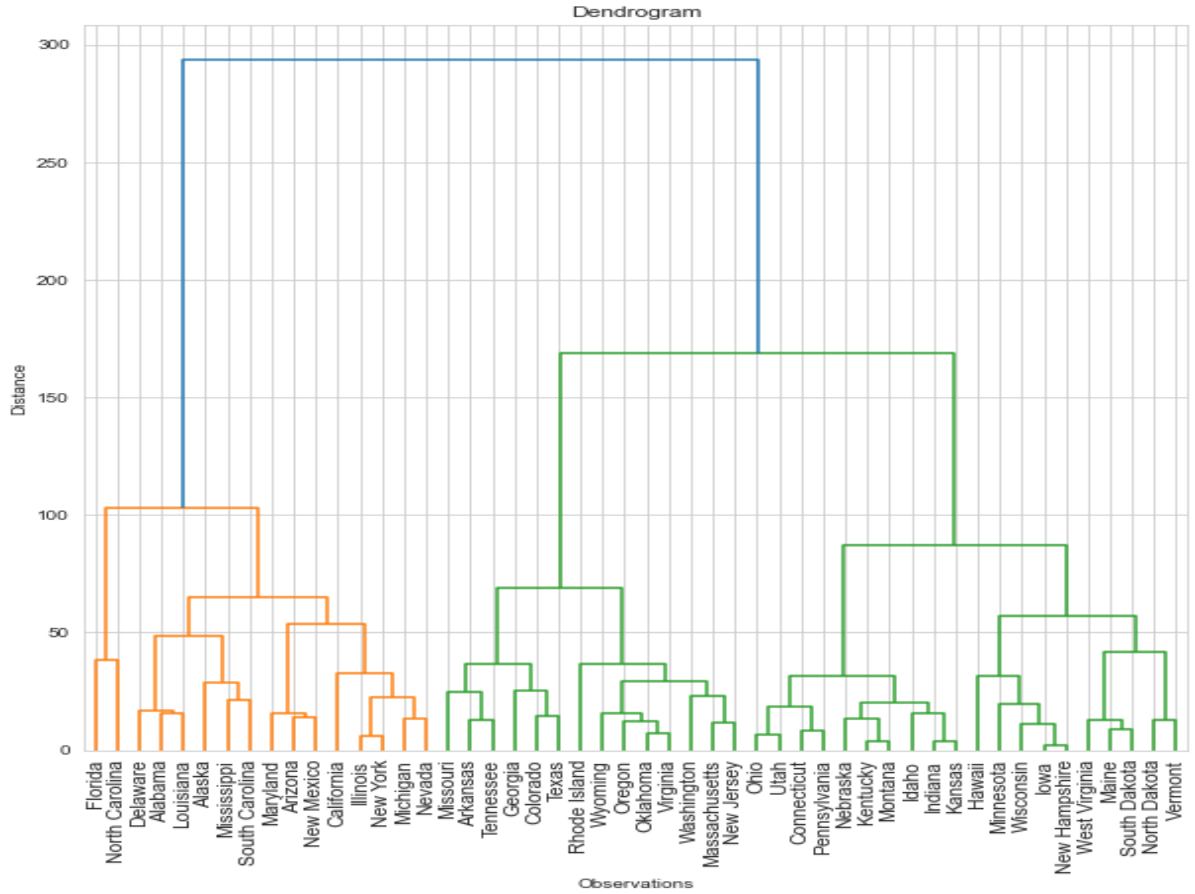


Fig: Dendrogram

Appendix C: Code

```
import pandas as pd
import numpy as np
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
from scipy.cluster.hierarchy import linkage, dendrogram, cut_tree
from sklearn.cluster import KMeans
from sklearn.cluster import AgglomerativeClustering
# Load data
arrests_data = pd.read_csv("USArrests.csv", index_col=0)
arrests_data

# Standardize the data
scaler = StandardScaler()
arrests_data_scaled = scaler.fit_transform(arrests_data)

# PCA
# Create PCA object
pca = PCA()
# Fit and transform the data
arrests_data_pca = pca.fit_transform(arrests_data_scaled)
fig, ax = plt.subplots(figsize=(10, 8))

# Scatter plot the scores (x and y) onto PCs 1 and 2
ax.scatter(arrests_data_pca[:, 0], arrests_data_pca[:, 1], alpha=0.8)

# Create variable coefficients for PC1 and PC2
pcs = pca.components_.T

# Use variable coefficients to create variable arrows on the graph
for i, (x, y) in enumerate(zip(pcs[:, 0], pcs[:, 1])):
    ax.arrow(0, 0, x, y, head_width=0.1, head_length=0.1, linewidth=1, color='r')
    ax.text(x + 0.1, y + 0.1, arrests_data.columns[i], color='black', ha='center',
va='center')

# Use state names as labels for the scatter plot points
for i, state in enumerate(arrests_data.index):
    ax.text(arrests_data_pca[i, 0], arrests_data_pca[i, 1], state, fontsize=10)

# Set labels and title
ax.set_xlabel("PC1")
ax.set_ylabel("PC2")
ax.set_title("PCA plot for USArrests Data")
plt.show()
```

```
# Print the loadings (coefficients) of each variable for each component
loadings = pd.DataFrame(pcs, columns=["PC1", "PC2", "PC3", "PC4"],
index=arrests_data.columns)
print("Loadings:\n", loadings)


# Output the amount of variance explained by each principal component
print("Explained variance by PC1: {:.2%}".format(pca.explained_variance_ratio_[0]))
print("Explained variance by PC2: {:.2%}".format(pca.explained_variance_ratio_[1]))
print("Explained variance by PC3: {:.2%}".format(pca.explained_variance_ratio_[2]))
print("Explained variance by PC4: {:.2%}".format(pca.explained_variance_ratio_[3]))


# K Means

for n_cluster in range(2,5):
    kmeans = KMeans(n_clusters=n_cluster, random_state=42).fit(arrests_data)
    # Define colors for the clusters
    colors = {0: 'blue', 1: 'red', 2: 'green', 3: 'black'}

    # Plot the data points colored by their cluster labels
    fig, ax = plt.subplots(figsize=(10,10))
    for i, state in enumerate(arrests_data.index):
        ax.scatter(arrests_data.iloc[i,0], arrests_data.iloc[i,1],
c=colors[kmeans.labels_[i]])
        ax.text(arrests_data.iloc[i, 0], arrests_data.iloc[i, 1], state, fontsize=10)
    # Add a legend
    legend_elements = [plt.Line2D([0], [0], color=color, label=f'Cluster {cluster}') for
cluster, color in colors.items()]
    ax.legend(handles=legend_elements, loc='upper left')

    # Add a border around each cluster grouping
    x_min, x_max = ax.get_xlim()
    y_min, y_max = ax.get_ylim()
    for cluster in set(kmeans.labels_):
        cluster_indices = np.where(kmeans.labels_ == cluster)[0]
        x_cluster_min, x_cluster_max = arrests_data.iloc[cluster_indices, 0].min(),
arrests_data.iloc[cluster_indices, 0].max()
        y_cluster_min, y_cluster_max = arrests_data.iloc[cluster_indices, 1].min(),
arrests_data.iloc[cluster_indices, 1].max()
        rect = plt.Rectangle((x_cluster_min, y_cluster_min), x_cluster_max -
x_cluster_min, y_cluster_max - y_cluster_min, fill=False, edgecolor=colors[cluster],
linewidth=2)
        ax.add_patch(rect)

    ax.set_xlabel("Murder")
```

```
    ax.set_ylabel("Assault")
    ax.set_title(f'{n_cluster} Clusters')
    plt.show()

# Hierarchical Clustering
hier_cluster = linkage(arrests_data, method="complete")

# Plotting dendrogram
plt.figure(figsize=(10, 10))
plt.title("Dendrogram")
plt.xlabel("Observations")
plt.ylabel("Distance")
dendrogram(hier_cluster, labels=arrests_data.index, truncate_mode='level', p=50,
leaf_rotation=90., leaf_font_size=12)
plt.show()

# Cutting the dendrogram at k=4 to create 4 clusters
clusters = cut_tree(hier_cluster, n_clusters=4)

# Counting the number of observations in each cluster
unique, counts = np.unique(clusters, return_counts=True)
print(dict(zip(unique, counts)))
```