King Saud University College of Computer and Information Sciences

Information Technology department

IT362: Data Science Principles

Data Science Project Logbook

Team members
Yara Bahmaid
Aljohara Albanyan
Athbah Alaliwei
Shaden Alturki
Aafia Muhammad

Modelling and Communication

Clustering Task:

Date: [1/5/2024] to [3/5/2024]

Task: The task involved scaling specific float columns in a pandas DataFrame using Min-Max scaling. Additionally, a new DataFrame needed to be created with selected columns.

Code:

```
import pandas as pd

import numpy as np

from sklearn.preprocessing import MinMaxScaler

import pandas as pd


# Assuming you already have your data in the 'df' DataFrame


# Select the float columns for scaling

float_columns = ['Score', 'Number of reviews', 'Price', 'Star rating']


# Scale the float columns

scaler = MinMaxScaler()

df[float_columns] = scaler.fit_transform(df[float_columns])


# Create a new DataFrame with only the specified columns

selected_columns = ['Score', 'Number of reviews', 'Price', 'Star rating',

                'Parking']

data = df[selected_columns]


data.describe()
```

Difficulties:

1. Identifying Float Columns: Determining which columns in the DataFrame were of float type and thus required scaling.

2. Applying Min-Max Scaling: Implementing Min-Max scaling accurately across the selected float columns.

3. Creating New DataFrame: Generating a new DataFrame with only the specified columns.

What we did to solve the problem:

1. Identifying Float Columns: We first created a list of float column names based on prior knowledge of the dataset or by programmatically identifying them through methods like `df.dtypes`.

2. Applying Min-Max Scaling: Utilizing the `MinMaxScaler` from scikit-learn, we applied scaling to the identified float columns, ensuring a uniform scale between 0 and 1.

3. Creating New DataFrame: We constructed a new DataFrame containing only the specified columns using pandas' column selection functionality.

What we found:

1. Float Columns: We discovered the float columns in the DataFrame were 'Score', 'Number of reviews', 'Price', and 'Star rating'.

2. Min-Max Scaling: By employing Min-Max scaling, we normalized the values in these columns to a range between 0 and 1, preserving the relative differences between them.

3. New DataFrame: We successfully created a new DataFrame named 'data' comprising the selected columns 'Score', 'Number of reviews', 'Price', 'Star rating', and 'Parking' for further analysis.To compare between K-Means and Agglomerative Clustering algorithms and choose the best resulting clusters.

Task:

The task involves implementing KMeans clustering algorithm using Python's libraries such as NumPy, Matplotlib, Pandas, and Scikit-learn. The goal is to cluster data points into distinct groups based on their features.

Code:

```
import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

from sklearn.cluster import KMeans

from sklearn.metrics import silhouette_score

from IPython.display import clear_output
from sklearn.decomposition import PCA

# Set random seed for reproducibility
np.random.seed(42)

# Function to generate random centroids
def random_centroids(data, k):
    centroids = []
    for i in range(k):
        centroid = data.apply(lambda x: np.random.uniform(x.min(), x.max()))
        centroids.append(centroid)
    return pd.concat(centroids, axis=1)

# Function to calculate Euclidean distance
def get_labels(data, centroids):
    distances = centroids.apply(lambda x: np.sqrt(((data - x) ** 2).sum(axis=1)))
    return distances.idxmin(axis=1)

# Function to plot clusters using PCA
def plot_clusters(data, labels, centroids, iteration):
    pca = PCA(n_components=2)
    data_2d = pca.fit_transform(data)
    centroids_2d = pca.transform(centroids.T)
    clear_output(wait=True)
    plt.title(f'Iteration {iteration}')
    plt.scatter(x=data_2d[:,0], y=data_2d[:,1], c=labels)
    plt.scatter(x=centroids_2d[:,0], y=centroids_2d[:,1])
    plt.show()
```

```
# Function to update centroids
def new_centroids(data, labels, k):
    centroids = []
    for i in range(k):
        cluster_data = data[labels == i]
        centroid = cluster_data.mean()
        centroids.append(centroid)
    return pd.concat(centroids, axis=1)

# Parameters for KMeans clustering
max_iterations = 100
centroid_count = 3
centroids = random_centroids(data, centroid_count)
old_centroids = pd.DataFrame()
iteration = 1

# Initialize variables to keep track of the best result
best_kmeans_labels = None
best_kmeans_centroids = None
best_silhouette_score = -1

# KMeans clustering loop
while iteration < max_iterations and not centroids.equals(old_centroids):
    old_centroids = centroids

    labels = get_labels(data, centroids)
    centroids = new_centroids(data, labels, centroid_count)
    # Calculate silhouette score
    silhouette = silhouette_score(data, labels)

    # Print current KMeans result
    print(f"Iteration {iteration}: Silhouette Score = {silhouette}")

    # Update best result if silhouette score is higher
    if silhouette > best_silhouette_score:
        best_silhouette_score = silhouette
        best_kmeans_labels = labels
        best_kmeans_centroids = centroids

    # Plot clusters using PCA
    plot_clusters(data, labels, centroids, iteration)

    iteration += 1

# Print the best KMeans result
print("Best KMeans Result:")
print(f"Silhouette Score = {best_silhouette_score}")
print("Cluster Centers:")
print(best_kmeans_centroids)
```

Difficulties: Implementing KMeans clustering involves several challenges:

1. Initializing centroids: Randomly initializing centroids can lead to suboptimal clustering results.
2. Convergence: Ensuring that the algorithm converges to stable centroids within a reasonable number of iterations.
3. Evaluation: Determining the optimal number of clusters (k) and evaluating the quality of clustering.

What we did to solve the problem:

1. Initializing centroids: We addressed this by creating a function random_centroids() to generate random centroids within the range of the data points.
2. Convergence: We monitored the convergence of the algorithm by comparing the centroids at each iteration with the centroids from the previous iteration. The loop terminates when either the maximum number of iterations is reached or when the centroids stop changing.
3. Evaluation: We used the silhouette score as a metric to evaluate the quality of clustering. The silhouette score measures how similar an object is to its own cluster (cohesion) compared to other clusters (separation). We stored the best clustering result based on the highest silhouette score.

What We Found:

1. The KMeans algorithm successfully clustered the data points into distinct groups.
2. The best KMeans result was achieved with a certain silhouette score, indicating the quality of the clustering.
3. The cluster centers represent the centroids of the clusters, providing insight into the characteristics of each cluster.

Task: The task was to perform hierarchical clustering on a dataset using Python libraries such as NumPy, Matplotlib, and scikit-learn. Specifically, the goal was to cluster the data into a predefined number of clusters and visualize the clusters in a 2D space.

Code:
```
import numpy as np

import matplotlib.pyplot as plt

from sklearn.decomposition import PCA

from IPython.display import clear_output

from sklearn.cluster import AgglomerativeClustering



# Function to plot clusters using PCA

def plot_clusters_2d(data_2d, labels, centroids_2d, title):

    clear_output(wait=True)

    plt.title(title)

    plt.scatter(x=data_2d[:,0], y=data_2d[:,1], c=labels)

    plt.scatter(x=centroids_2d[:,0], y=centroids_2d[:,1], marker='X', s=150, c='red')
    plt.show()

# Hierarchical clustering
best_k = 3  # Assuming you choose the best k based on the KMeans results
hierarchical = AgglomerativeClustering(n_clusters=best_k)
hierarchical.fit(data)
hierarchical_labels = hierarchical.labels_

# Fit PCA model with original data
pca = PCA(n_components=2)
data_2d = pca.fit_transform(data)

# Transform centroids using fitted PCA model
centroids_2d = pca.transform(np.array(data.groupby(hierarchical_labels).mean()))

# Plot Hierarchical clustering
plot_clusters_2d(data_2d, hierarchical_labels, centroids_2d, "Hierarchical")
```

Difficulties:

Choosing the optimal number of clusters (k): Unlike KMeans clustering, hierarchical clustering doesn't inherently provide a clear way to determine the optimal number of clusters. We needed a method to decide on the appropriate number of clusters for the data.

Visualizing high-dimensional data: The data likely had more than two dimensions, making it challenging to visualize the clusters. We needed to reduce the dimensionality of the data for visualization purposes.

Interpreting hierarchical clustering results: Hierarchical clustering generates a hierarchical tree of clusters, which can be difficult to interpret and visualize effectively.

How we solved them:

Choosing the optimal number of clusters (k): We used the results from a previous KMeans clustering step to determine the best number of clusters. By leveraging the insights gained from KMeans, we could make an educated guess for the number of clusters in hierarchical clustering.

Visualizing high-dimensional data: We applied Principal Component Analysis (PCA) to reduce the dimensionality of the data to two dimensions. PCA helps maintain the most significant variance in the data while reducing the number of dimensions, enabling easier visualization.

Interpreting hierarchical clustering results: We utilized a function to plot the clusters in a 2D space, making it easier to interpret the results. Additionally, we calculated the centroids of each cluster and plotted them to provide further insight into the cluster distribution.

What we found:

By using a combination of KMeans clustering results and hierarchical clustering with PCA visualization, we could effectively cluster and visualize the data in a 2D space.

The choice of the optimal number of clusters greatly influenced the clustering results and subsequent interpretation.

PCA proved to be a valuable tool for visualizing high-dimensional data, enabling better understanding and interpretation of the clustering outcomes.

Task: Evaluate clustering methods and choose the best one based on the silhouette score.

Code:

```
kmeans_silhouette = silhouette_score(data, labels)

hierarchical_silhouette = silhouette_score(data, hierarchical_labels)


if kmeans_silhouette > hierarchical_silhouette:

    print("KMeans provides better clusters.")

    final_labels = labels

    cluster_centers = pd.DataFrame(centroids, index=data.columns)

else:

    print("Hierarchical clustering provides better clusters.")

    final_labels = hierarchical_labels

    cluster_centers = pd.DataFrame(index=data.columns)

    for label in range(best_k):

        cluster_centers[label] = data[hierarchical_labels == label].mean()


# Display the cluster centers

print("Cluster Centers:")
print(cluster_centers)
```

Difficulties:

Selection of Clustering Methods: Determining which clustering algorithms to evaluate.

Evaluation Metric: Choosing an appropriate metric for evaluating the quality of clusters.

Interpretation of Results: Interpreting the silhouette scores to make a decision on the best clustering method.

How We Solved Them:

Selection of Clustering Methods: We evaluated two commonly used clustering algorithms - KMeans and Hierarchical Clustering.

Evaluation Metric: We used the silhouette score, which measures the compactness and separation of clusters.

Interpretation of Results: We compared the silhouette scores of KMeans and Hierarchical Clustering and chose the method with the higher score as the best clustering approach.

What We Found:

After evaluating both KMeans and Hierarchical Clustering methods using silhouette scores:

If the silhouette score of KMeans clustering is higher than that of Hierarchical Clustering, we concluded that KMeans provides better clusters. In this case, we used the centroids of the clusters as cluster centers for KMeans.

If the silhouette score of Hierarchical Clustering is higher, we concluded that Hierarchical Clustering provides better clusters. In this case, we computed the mean of each feature within each cluster to determine the cluster centers.

Finally, we displayed the cluster centers for the chosen clustering method.

Integration of Large Language Models (LLMs) in Data Analysis

Date: [3-5-2024 to 5-5-2024]

1. Integration Decision:

We decided to integrate Large Language Models (LLMs) into our data analysis process to explore their potential in enhancing efficiency and productivity.

- **Rationale:**

LLMs have shown promise in automating various aspects of data analysis, including descriptive statistics, exploratory data analysis, and report generation.

Leveraging LLMs could streamline our workflow, enabling us to derive insights from the dataset more quickly and effectively.

2. Selection of LLM Platform:

We researched and selected a suitable LLM platform to integrate into our data analysis workflow.

- **Platform Evaluation:**

We evaluated multiple LLM platforms based on factors such as accessibility, functionality, and ease of integration.

After careful consideration, we chose ChatGPT 4.0 that offered a user-friendly chat interface and comprehensive natural language processing capabilities.

- **Rationale:**

The selected platform provided seamless integration with our existing tools and allowed for real-time interaction with the LLM, facilitating efficient data analysis.

3. LLM Integration Process:

We devised a plan to integrate the selected LLM platform into our data analysis process.

- **Integration Steps:**

We established protocols for accessing the LLM through the chat interface and formulated specific queries to extract relevant information from the dataset.

Training sessions were conducted to familiarize team members with the LLM platform and its capabilities, ensuring smooth integration into our workflow.

- **Rationale:**

Proper planning and training were essential to ensure effective utilization of the LLM and maximize its impact on our data analysis process.

4. Overcoming Challenges:

We encountered challenges during the initial stages of LLM integration and implemented strategies to address them.

- **Challenges Faced:**

Understanding the full potential of the LLM and determining its optimal use within our analysis workflow.

Ensuring data privacy and security while utilizing external LLM platforms for analysis tasks.

- **Strategies Implemented:**

We conducted thorough experimentation and exploration of the LLM's capabilities to identify its strengths and limitations.

Protocols were established to safeguard sensitive data and ensure compliance with privacy regulations during interactions with the LLM platform.

5. Impact Evaluation:

We assessed the impact of integrating LLMs into our data analysis process on productivity and efficiency.

- **Evaluation Metrics:**

Efficiency gains in terms of time saved on data analysis tasks.

Quality of insights derived from the LLM-driven analysis compared to traditional methods.

User feedback and satisfaction with the LLM integration process.

- **Rationale:**

Evaluating the impact of LLM integration allows us to gauge its effectiveness in improving our data analysis workflow and inform future decision-making regarding its utilization.

6. Future Directions:

Based on our experiences with LLM integration, we outlined potential areas for further exploration and improvement.

- **Future Considerations:**

Investigate advanced LLM functionalities, such as sentiment analysis and text summarization, to enhance the depth of analysis.

Explore opportunities for fine-tuning LLMs on domain-specific datasets to improve model performance and relevance to our analysis tasks.

- **Rationale:**

Continuous exploration and refinement of LLM integration strategies will enable us to harness the full potential of these advanced natural language processing technologies in our data analysis endeavors.