User Clustering Recommendation System

Arushi Gupta

September 27, 2024

`https://github.com/agupta0/user-clustering-reccomendation`

Abstract

This document provides a detailed overview of the User Clustering Recommendation System project, including its objectives, methodology, implementation steps, and key learnings. It serves as a reference for anyone interested in understanding clustering techniques in machine learning.

1 Introduction

In this project, we implemented a user clustering recommendation system utilizing the K-Means clustering algorithm. The primary goal was to group users based on their preferences to enable tailored recommendations.

2 Objectives

- Understand the concept of clustering in unsupervised machine learning.
- Implement the K-Means clustering algorithm using Python and its libraries.
- Analyze the clustering results to gain insights into user preferences.
- Document the entire process for future reference.

3 Clustering Overview

Clustering is an unsupervised machine learning technique that aims to group similar data points together based on certain features. Unlike supervised learning, clustering does not use labeled data. The K-Means algorithm is one of the most popular clustering methods, which partitions data into K distinct clusters.

3.1 Introduction K-Means

K-means clustering is a popular unsupervised learning algorithm that partitions a dataset into k distinct clusters. Each cluster is represented by its centroid, which is the mean of the points within the cluster.

3.2 Algorithm Steps

The K-means algorithm works iteratively by following these steps:

1. **Initialization**: Select k initial centroids, either randomly or using methods like k-means++.
2. **Assignment**: Assign each data point to the nearest centroid based on Euclidean distance.
    - Note: the length of the shortest line between two points is known as Euclidean distance
    - aka orthogonal or Pythagorean distance
3. **Update**: Update the centroids by computing the mean of the points assigned to each cluster.
4. **Repeat**: Repeat the assignment and update steps until convergence (i.e., centroids do not change).

3.3 Objective Function

The goal of the K-means algorithm is to minimize the within-cluster variance, which is a measure of how close the data points are to their cluster centroid. The objective function, which quantifies this variance, is defined as:

$$J = \sum_{i=1}^{k} \sum_{x \in C_i} ||x - \mu_i||^2$$

Where:

- k is the number of clusters.
- C_i is the set of points assigned to the i-th cluster.
- μ_i is the centroid (mean) of the i-th cluster.
- $||x - \mu_i||^2$ is the squared Euclidean distance between a data point x and the centroid μ_i.
    - It has been some time since I've worked with vectors in linear algebra and I found that a quick recap of how the standard distance of a line formula becomes the formula above is helpful.

Derivation of Squared Euclidean Distance $||x - \mu_i||^2$

The Euclidean distance between two points $x = (x_1, x_2, \ldots, x_n)$ and $y = (y_1, y_2, \ldots, y_n)$ in n-dimensional space is given by the formula:

$$d(x, y) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + \cdots + (x_n - y_n)^2}$$

This formula computes the straight-line distance between the two points in Euclidean space.

In vector notation, where x and y are vectors $\mathbf{x}$ and $\mathbf{y}$, respectively, we can express the Euclidean distance as the norm of the difference between these vectors:

$$d(\mathbf{x}, \mathbf{y}) = ||\mathbf{x} - \mathbf{y}||$$

Here, $||\mathbf{x} - \mathbf{y}||$ is the Euclidean norm, or ℓ_2-norm, of the difference vector $\mathbf{x} - \mathbf{y}$. The Euclidean norm is defined as:

$$||\mathbf{x} - \mathbf{y}|| = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + \cdots + (x_n - y_n)^2}$$

In K-means clustering, we are interested in minimizing the squared Euclidean distance, which avoids the computational cost of taking the square root. The squared Euclidean distance between the two points is simply the square of the Euclidean distance:

$$||\mathbf{x} - \mathbf{y}||^2 = (x_1 - y_1)^2 + (x_2 - y_2)^2 + \cdots + (x_n - y_n)^2$$

Now, consider a point x and the centroid μ_i of cluster C_i in K-means clustering. The squared distance between the point x and the centroid μ_i is:

$$||x - \mu_i||^2 = (x_1 - \mu_{i1})^2 + (x_2 - \mu_{i2})^2 + \cdots + (x_n - \mu_{in})^2$$

Here:

- $x = (x_1, x_2, \ldots, x_n)$ represents the data point.
- $\mu_i = (\mu_{i1}, \mu_{i2}, \ldots, \mu_{in})$ represents the centroid of cluster C_i.

Thus, the expression $||x - \mu_i||^2$ represents the sum of squared differences between the coordinates of the point x and the corresponding coordinates of the centroid μ_i. This is known as the squared Euclidean distance, which is used in K-means clustering to measure the closeness of points to their assigned cluster centroids.

3.3.1 Variance in Clustering

In statistics, **variance** measures how far a set of points are spread out from their mean. In the context of clustering, the variance within a cluster C_i is defined as the average of the squared distances between each point and the centroid of that cluster:

$$\text{Var}(C_i) = \frac{1}{|C_i|} \sum_{x \in C_i} ||x - \mu_i||^2$$

Minimizing the within-cluster variance ensures that the points are tightly grouped around the centroid. In K-means, the objective function sums the variance across all k clusters, with the goal of minimizing this total sum.

3.3.2 Origin of the Objective Function

The objective function is based on the concept of the **sum of squared errors (SSE)** or **within-cluster sum of squares (WCSS)**. The Euclidean distance is squared to penalize points that are far from the centroid, ensuring tighter clusters. The objective function:

$$J = \sum_{i=1}^{k} \sum_{x \in C_i} ||x - \mu_i||^2$$

originates from minimizing the total variance across all clusters, and it reflects the goal of reducing the spread of points within each cluster while maintaining well-separated clusters.

By minimizing J, the K-means algorithm ensures that the clusters are compact (points within a cluster are close to each other) and distinct (points in different clusters are far apart).

3.4 Advantages and Disadvantages

3.5 Advantages

- Simple and easy to implement.
- Fast convergence for small to medium-sized datasets.
- Works well when clusters have a spherical shape.

3.6 Disadvantages

- Sensitive to the initial placement of centroids.
- May converge to a local optimum rather than a global one.
- Requires predefining the number of clusters k.
- Not suitable for clusters with complex shapes or uneven sizes.

3.7 Applications

K-means clustering is commonly used in:

- Market segmentation.
- Image compression.
- Document categorization.
- Anomaly detection.

3.8 K-Mean algorithm tldr.

The K-Means algorithm works as follows:

1. Choose the number of clusters, K.
2. Randomly initialize K centroids.
3. Assign each data point to the nearest centroid.
4. Recalculate the centroids as the mean of the assigned data points.
5. Repeat the assignment and update steps until convergence.

4 Methodology

4.1 Data Preparation

Now that we've discussed a lot of theory we can return to the project! We thankfully don't have to implement this from scratch and can implement amazing libraries created by the developers who came before us! The project begins with data collection and preparation. We used a hypothetical dataset with user preferences. The dataset was structured with columns representing user attributes (e.g., age, preferences).

4.2 Libraries Used

The following Python libraries were used in this project:

- **Pandas**: For data manipulation and analysis.
- **NumPy**: For numerical computations.
- **Scikit-learn**: For implementing the K-Means algorithm.

4.3 Implementation Steps

4.3.1 Loading the Data

To start, we load the dataset using Pandas:

```
import pandas as pd

# Load the dataset
data = pd.read_csv('user_data.csv')
```

4.3.2 Data Preprocessing

The data was preprocessed to handle any missing values and scale the features for better clustering results. Data scaling ensures that each feature contributes equally to the distance calculations in K-Means.

```
from sklearn.preprocessing import StandardScaler

# Handle missing values and scale data
data.fillna(data.mean(), inplace=True)  % Filling missing values with the mean
scaler = StandardScaler()                % Initializing the scaler
scaled_data = scaler.fit_transform(data) % Scaling the data
```

4.3.3 Defining the K-Means Model

Next, we define the K-Means clustering model. We choose the number of clusters, K. In this case, we will use 3 clusters:

```
from sklearn.cluster import KMeans

# Define the K-Means model
kmeans = KMeans(n_clusters=3, random_state=42)
% Setting random state for reproducibility
```

4.3.4 Fitting the Model

We fit the model to the scaled data, which involves finding the optimal cluster centers and assigning each data point to a cluster.

```
# Fitting the model on the scaled data
kmeans.fit(scaled_data)
```

4.3.5 Predicting Clusters

After fitting the model, we can predict which cluster each user belongs to. This helps in understanding user segmentation based on preferences.

```
# Predicting the cluster for each user
clusters = kmeans.predict(scaled_data)

# Adding the cluster information to the original DataFrame
data['cluster'] = clusters
```

4.3.6 Visualization

Finally, we visualize the clusters to analyze the results. This helps in understanding how well the K-Means algorithm has grouped the users based on their preferences.

```
import matplotlib.pyplot as plt

# Visualization of clusters (assuming two features for
    simplicity)
plt.scatter(data['feature1'], data['feature2'], c=data['
    cluster'], cmap='viridis')
plt.title('User Clusters')
plt.xlabel('Feature 1')  # Replace with actual feature
    names
plt.ylabel('Feature 2')  # Replace with actual feature
    names
plt.show()
```

5 Key Learnings

Throughout this project, the following key concepts were learned:

- The importance of data preprocessing in machine learning workflows.
- Understanding the K-Means algorithm and its parameters, such as the number of clusters.
- The significance of feature scaling in improving clustering performance.
- How to visualize and interpret clustering results effectively.

6 Challenges Faced

During the implementation of this project, several challenges were encountered:

- Determining the optimal number of clusters was difficult.
- Handling missing values required careful consideration to avoid bias in clustering results.

7 Conclusion

The User Clustering Recommendation System successfully demonstrated how to group users based on their preferences using the K-Means clustering algorithm. This project not only enhanced my understanding of clustering techniques but also provided hands-on experience with Python libraries used in data science.

8 Future Work

Potential future enhancements for this project could include:

- Exploring different clustering algorithms (e.g., DBSCAN, Hierarchical Clustering).
- Implementing a user interface to interact with the clustering system.
- Using real-world datasets for more complex analysis rather than simplified version used here.

References

[1] S. P. Lloyd, "A Lloyd's Algorithm for K-Means Clustering," *IEEE Transactions on Computers*, vol. C-31, no. 6, pp. 881-884, 1982. DOI: 10.1109/TC.1982.1676439.

[2] Wolfram Research, "Euclidean Distance," *Wolfram MathWorld*, Available at: `https://mathworld.wolfram.com/EuclideanDistance.html`.

[3] N. Yaraghi, "K-Means Clustering Explained: An Easy Guide to Cluster Analysis," YouTube, Available at: `https://www.youtube.com/watch?v=YEwt6BJROug`.

[4] Pedregosa et al., "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825-2830, 2011. Available at: `https://scikit-learn.org/stable/`.

[5] W. McKinney, "Data Structures for Statistical Computing in Python," in *Proceedings of the 9th Python in Science Conference*, 2010, pp. 51-56. Available at: `https://pandas.pydata.org/`.

[6] T. E. Oliphant, *A Guide to NumPy*, USA: Trelgol Publishing, 2006. Available at: `https://numpy.org/`.

[7] OpenAI, "ChatGPT: Optimizing Language Models for Dialogue." Available at: `https://openai.com/chatgpt`. Accessed in 2024. ChatGPT was used to assist in explaining key concepts and guidance throughout the project.

User_Clustering_Recommendation

September 27, 2024

[1]:
```
pip install pandas
```

```
Requirement already satisfied: pandas in
/opt/homebrew/Cellar/jupyterlab/4.2.5_1/libexec/lib/python3.12/site-packages
(2.2.3)
Requirement already satisfied: numpy>=1.26.0 in
/opt/homebrew/lib/python3.12/site-packages (from pandas) (2.1.1)
Requirement already satisfied: python-dateutil>=2.8.2 in
/opt/homebrew/Cellar/jupyterlab/4.2.5_1/libexec/lib/python3.12/site-packages
(from pandas) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in
/opt/homebrew/Cellar/jupyterlab/4.2.5_1/libexec/lib/python3.12/site-packages
(from pandas) (2024.2)
Requirement already satisfied: tzdata>=2022.7 in
/opt/homebrew/Cellar/jupyterlab/4.2.5_1/libexec/lib/python3.12/site-packages
(from pandas) (2024.2)
Requirement already satisfied: six>=1.5 in
/opt/homebrew/Cellar/jupyterlab/4.2.5_1/libexec/lib/python3.12/site-packages
(from python-dateutil>=2.8.2->pandas) (1.16.0)
Note: you may need to restart the kernel to use updated packages.
```

[4]:
```
pip install scikit-learn
```

```
Collecting scikit-learn
  Downloading scikit_learn-1.5.2-cp312-cp312-macosx_12_0_arm64.whl.metadata (13
kB)
Requirement already satisfied: numpy>=1.19.5 in
/opt/homebrew/lib/python3.12/site-packages (from scikit-learn) (2.1.1)
Collecting scipy>=1.6.0 (from scikit-learn)
  Downloading scipy-1.14.1-cp312-cp312-macosx_14_0_arm64.whl.metadata (60 kB)
Collecting joblib>=1.2.0 (from scikit-learn)
  Downloading joblib-1.4.2-py3-none-any.whl.metadata (5.4 kB)
Collecting threadpoolctl>=3.1.0 (from scikit-learn)
  Downloading threadpoolctl-3.5.0-py3-none-any.whl.metadata (13 kB)
Downloading scikit_learn-1.5.2-cp312-cp312-macosx_12_0_arm64.whl (11.0 MB)
    11.0/11.0 MB
17.0 MB/s eta 0:00:00a 0:00:01
Downloading joblib-1.4.2-py3-none-any.whl (301 kB)
Downloading scipy-1.14.1-cp312-cp312-macosx_14_0_arm64.whl (23.1 MB)
```

```
   23.1/23.1 MB
17.3 MB/s eta 0:00:00a 0:00:01
Downloading threadpoolctl-3.5.0-py3-none-any.whl (18 kB)
Installing collected packages: threadpoolctl, scipy, joblib, scikit-learn
Successfully installed joblib-1.4.2 scikit-learn-1.5.2 scipy-1.14.1
threadpoolctl-3.5.0
Note: you may need to restart the kernel to use updated packages.
```

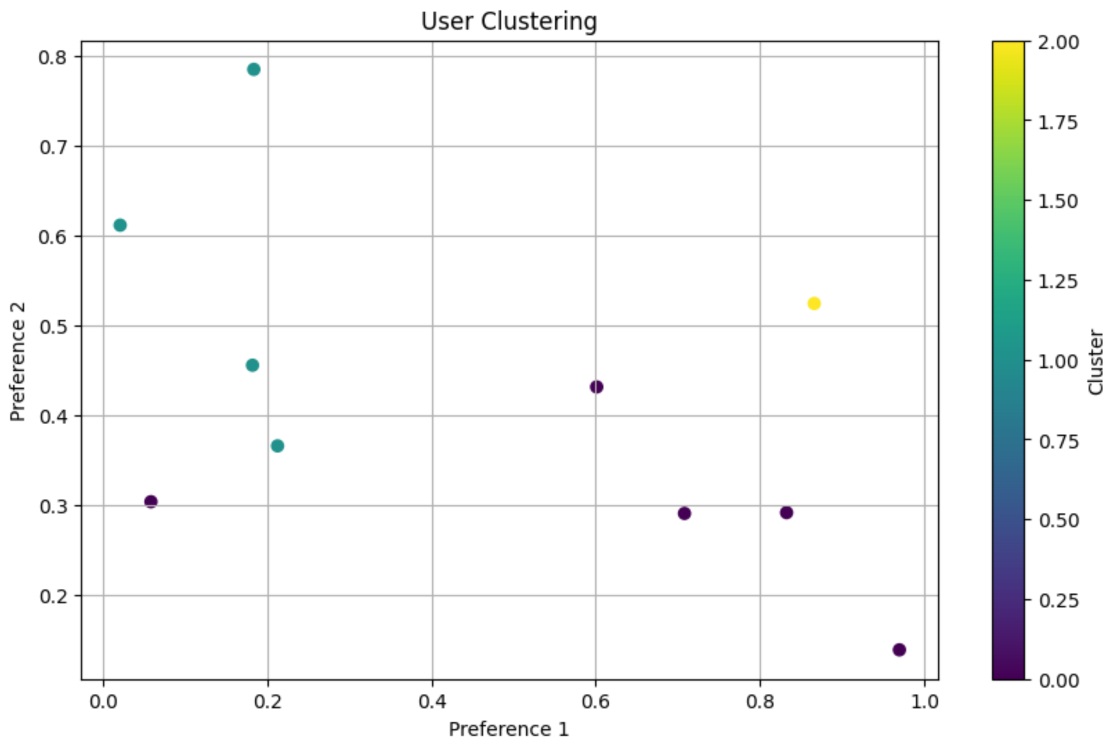
[5]:
```
pip install matplotlib
```

```
Requirement already satisfied: matplotlib in /opt/homebrew/Cellar/python-
matplotlib/3.9.2/libexec/lib/python3.12/site-packages (3.9.2)
Requirement already satisfied: contourpy>=1.0.1 in /opt/homebrew/Cellar/python-
matplotlib/3.9.2/libexec/lib/python3.12/site-packages (from matplotlib) (1.2.1)
Requirement already satisfied: cycler>=0.10 in /opt/homebrew/Cellar/python-
matplotlib/3.9.2/libexec/lib/python3.12/site-packages (from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /opt/homebrew/Cellar/python-
matplotlib/3.9.2/libexec/lib/python3.12/site-packages (from matplotlib) (4.53.1)
Requirement already satisfied: kiwisolver>=1.3.1 in /opt/homebrew/Cellar/python-
matplotlib/3.9.2/libexec/lib/python3.12/site-packages (from matplotlib) (1.4.5)
Requirement already satisfied: numpy>=1.23 in /opt/homebrew/lib/python3.12/site-
packages (from matplotlib) (2.1.1)
Requirement already satisfied: packaging>=20.0 in
/opt/homebrew/Cellar/jupyterlab/4.2.5_1/libexec/lib/python3.12/site-packages
(from matplotlib) (24.1)
Requirement already satisfied: pillow>=8 in /opt/homebrew/lib/python3.12/site-
packages (from matplotlib) (10.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in /opt/homebrew/Cellar/python-
matplotlib/3.9.2/libexec/lib/python3.12/site-packages (from matplotlib) (3.1.2)
Requirement already satisfied: python-dateutil>=2.7 in
/opt/homebrew/Cellar/jupyterlab/4.2.5_1/libexec/lib/python3.12/site-packages
(from matplotlib) (2.9.0.post0)
Requirement already satisfied: six>=1.5 in
/opt/homebrew/Cellar/jupyterlab/4.2.5_1/libexec/lib/python3.12/site-packages
(from python-dateutil>=2.7->matplotlib) (1.16.0)
Note: you may need to restart the kernel to use updated packages.
```

[3]:
```python
import numpy as np # numpy for numerical data
import pandas as pd #pandas for dataframes (structured data)

#random seed ensures you can reprdouce results
np.random.seed(42)

# generate a dataset with 10 users (rows) and attributes (columns)
#age refers to user age
# preference_1 to preference_2 represents how much the user likes certain topics

data = pd.DataFrame({
    'user_id': range(1, 11),  #create 10 user ID's from 1 to 10
    'age' : np.random.randint(18, 45, size = 10), # 10 random ages between 18␣
 ↪and 45
    'preference_1': np.random.rand(10),         # Random preferences between 0␣
 ↪and 1 for topic 1
    'preference_2': np.random.rand(10),         # Random preferences between 0␣
 ↪and 1 for topic 2
    'preference_3': np.random.rand(10),         # Random preferences between 0␣
 ↪and 1 for topic 3
    'preference_4': np.random.rand(10)          # Random preferences between 0␣
 ↪and 1 for topic 4
})

# Setting 'user_id' as the index, because each row should correspond to a unique␣
 ↪user
data.set_index('user_id', inplace=True)

# Displaying the first 5 rows of the generated dataset
data.head()

# Normalizing the dataset: scaling age to be between 0 and 1, similar to the␣
 ↪preferences
from sklearn.preprocessing import MinMaxScaler

# Selecting only the features we want to normalize (age and preferences)
features = data[['age', 'preference_1', 'preference_2', 'preference_3',␣
 ↪'preference_4']]

# MinMaxScaler scales features to a range between 0 and 1
scaler = MinMaxScaler()
scaled_features = scaler.fit_transform(features)

# Convert the scaled features back into a DataFrame
scaled_data = pd.DataFrame(scaled_features, columns=features.columns, index=data.
 ↪index)

# Displaying the first 5 rows of the normalized dataset
scaled_data.head()

from sklearn.cluster import KMeans

# Defining the K-Means model
# Let's assume we want to group the users into 3 clusters for simplicity
kmeans = KMeans(n_clusters=3, random_state=42)

# Fitting the model on the scaled data
```

```
kmeans.fit(scaled_data)

# Predicting the cluster each user belongs to
clusters = kmeans.predict(scaled_data)

# Adding the cluster information to the original DataFrame
data['cluster'] = clusters

# Display the updated DataFrame
data[['age', 'preference_1', 'preference_2', 'preference_3', 'preference_4',␣
 ↪'cluster']].head()


# Calculating the average preferences for each cluster
cluster_means = data.groupby('cluster').mean()
print(cluster_means)


def recommend_items(user_data, cluster_means):
    # Get the user's cluster
    user_cluster = user_data['cluster']

    # Get the average preferences for the user's cluster
    recommended_preferences = cluster_means.loc[user_cluster]

    # Display recommendations
    return recommended_preferences

# Let's say we want to recommend items for the first user in the DataFrame
user_to_recommend = data.iloc[0]

# Get recommendations for this user
recommendations = recommend_items(user_to_recommend, cluster_means)
print(f"Recommendations for User 0:\n{recommendations}")

import matplotlib.pyplot as plt

# Visualizing the clusters
plt.figure(figsize=(10, 6))
plt.scatter(data['preference_1'], data['preference_2'], c=data['cluster'],␣
 ↪cmap='viridis', marker='o')
plt.title('User Clustering')
plt.xlabel('Preference 1')
plt.ylabel('Preference 2')
plt.colorbar(label='Cluster')
plt.grid()
plt.show()
```

```
           age  preference_1  preference_2  preference_3  preference_4
cluster
0         29.2      0.633925      0.291811      0.214823      0.391713
1         36.0      0.149538      0.554865      0.832615      0.488165
2         37.0      0.866176      0.524756      0.514234      0.097672
Recommendations for User 0:
age             29.200000
preference_1     0.633925
preference_2     0.291811
preference_3     0.214823
preference_4     0.391713
Name: 0, dtype: float64
```


User Clustering
Preference 1
Preference 2
Cluster