Implementing Sliding Window Algorithms for k-Clustering

SUBMITTED BY -

Aryaman Chauhan (2020B5A72006P)

Devansh (2020B5A72001P)

Harshvardhan Jouhary (2020B2A31623P)

SUBMITTED TO -

Dr. Aneesh S Chivukula

INDEX

Acknowledgment

We would like to express our deepest gratitude to all those who have contributed to the successful completion of the project. Their unwavering support, expertise, and dedication have been instrumental in making this project a reality.

First and foremost, we extend our heartfelt appreciation to our professor Dr. Aneesh S Chivukula, for their invaluable guidance, encouragement, and continuous support throughout the project's duration. Their expertise and insightful feedback have been instrumental in shaping the project's direction and ensuring its successful execution.

We are immensely grateful to our project team members for their hard work, collaboration, and enthusiasm. Each team member's unique skills and contributions have played a vital role in achieving the project's objectives.

Project description

The primary aim of project is to implement sliding window algorithm for K-clustering algorithm using Python programming language. The algorithms will be designed to handle continuous data streams and efficiently update the clustering results as new data points arrive.

We have taken inspiration from research paper published in **NeurIPS 2020** .

The project will focus on developing sliding window algorithms for k-clustering, where k represents the number of clusters. The algorithms will be designed to work with data streams that contain high-dimensional and potentially unbounded data points. The sliding window approach will enable the algorithms to process only the most recent data within a defined window size, thereby reducing memory requirements and processing time.

Here we have developed a sliding window variant of the k-means clustering algorithm that can efficiently update clusters in response to incoming data points within the sliding window.

We have applied this algorithm on dataset “Mall_Customers” where we analyse the customer base of mall and identify patterns within it. The data is received in form of .csv file. This file contains the attributes “Gender”, “Age”, “Annual Income”,” Spending Score”. The “Spending score” is a scale for measing how much a customer spends in buying goods from mall.

We will analyse this dataset which can be updates with time, since more customer get added and removed each day and implement K-clustering on it.

Project Team:

Devansh: Researched and Implemented design project

Aryaman Chauhan: Researched and Implemented design project

Harshvardhan Jouhary: Created project report

Project Implementation

We primarily used pandas and sklearn library to implement KMeans clustering. From here we will explain our code step by step.

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
import warnings
warnings.filterwarnings("ignore")
```

Here was have imported the required libraries: pandas for data handling, numpy for numerical operations, matplotlib for plotting, and KMeans from scikit-learn for performing k-means clustering.

- **k_clustering() function**

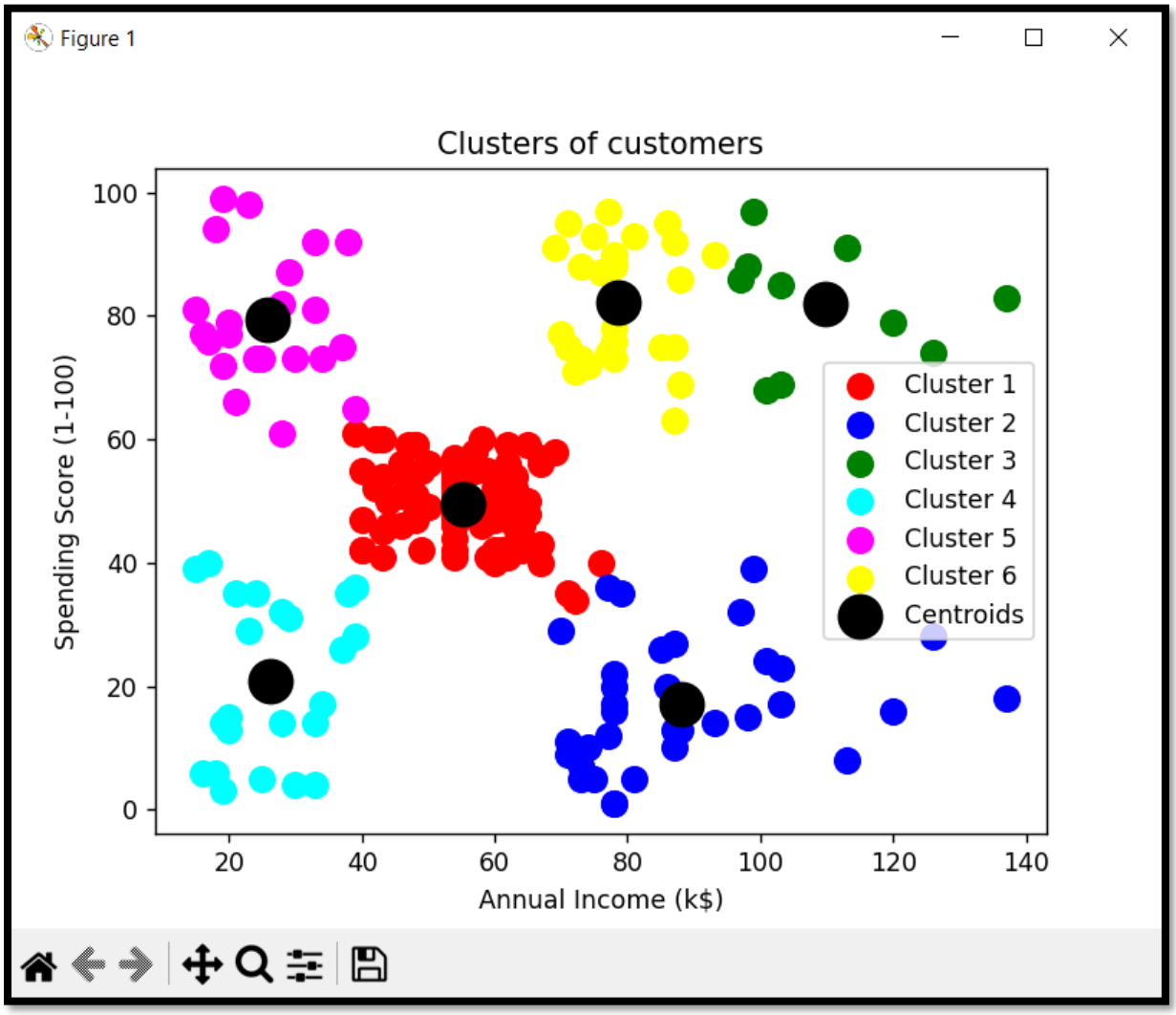
```python
def k_clustering(window_size):
    dataset = pd.read_csv("Mall_Customers.csv")
    X=dataset.iloc[window_size:,[3,4]].values

    i=best_cluster(X)
    kmeans = KMeans(n_clusters = i, init = 'k-means++', random_state = 42)
    y_kmeans = kmeans.fit_predict(X)
    #print(y_kmeans)
    #print(X)
    colors = ['red', 'blue', 'green', 'cyan', 'magenta', 'yellow', 'orange', 'purple', 'brown', 'pink']
    for cluster_num in range(i):
        plt.scatter(X[y_kmeans == cluster_num, 0], X[y_kmeans == cluster_num, 1], s=100, c=colors[cluster_num],
                    label=f'Cluster {cluster_num + 1}')
    plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s=300, c='black', label='Centroids')

    plt.title('Clusters of customers')
    plt.xlabel('Annual Income (k$)')
    plt.ylabel('Spending Score (1-100)')
    plt.legend()
    plt.show()
```

To implement K means clustering on dataset we have created a separate function named **k_clustering()** which will be called via loop from main. It takes a **window_size** as an input parameter, which is used to read a specific portion of the dataset for clustering, which will implement sliding window functionality.

Data is taken from **"Mall_Customers.csv"**. The **iloc** method is then used to select the features for clustering. The X variable contains the selected features (columns 3 and 4) as a numpy array. Only column 3 and 4 contains significant data for analysis hence we omitted the remaining columns.

We used KMeans method in sklearn.cluster library to implement K means clustering.The default algorithm used by this method is *"lloyd"*.

To find the optimal number of cluster we created another function **best_cluster(),** which takes data as input and returns optimal value of cluster, we store the return value in 'i'.



The function then plots each cluster separately with a different color using **plt.scatter()**. It iterates over the clusters and plots the data points for each cluster with the corresponding color from the **colors** list. The cluster centroids are plotted as black markers.

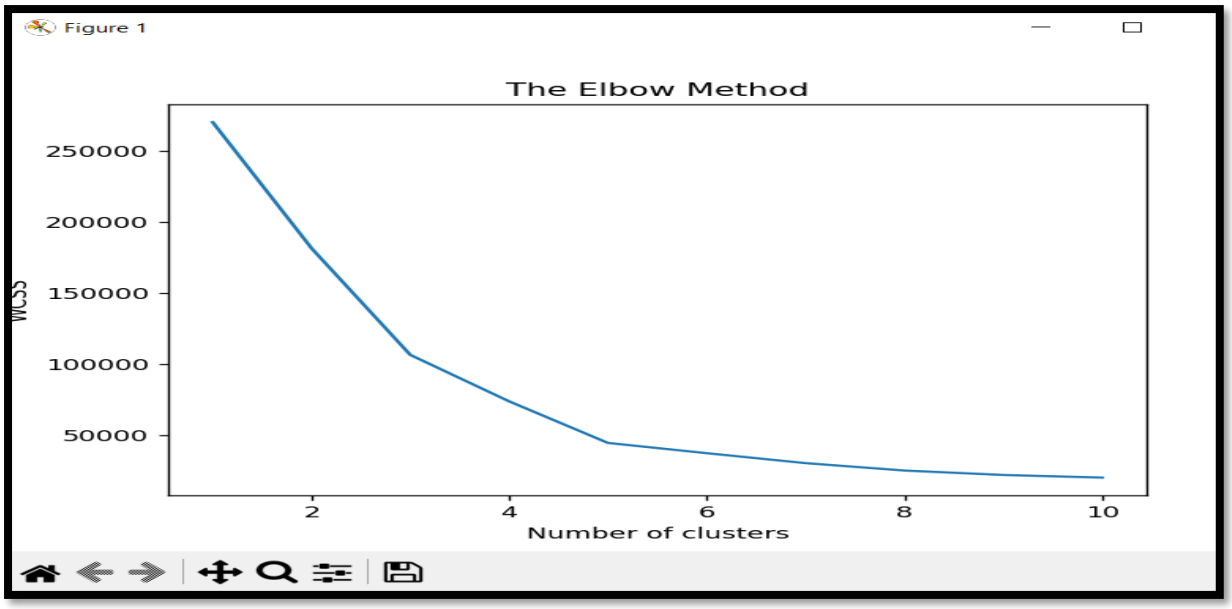
- **best_cluster() function**

```
def best_cluster(X):
    wcss = []
    for i in range(1, 11):
        kmeans = KMeans(n_clusters=i, init='k-means++', random_state=42)
        kmeans.fit(X)
        wcss.append(kmeans.inertia_)

    wcss_diff = np.diff(wcss)
    wcss_diff_percentage_change = (wcss_diff[:-1] - wcss_diff[1:]) / wcss_diff[:-1]
    elbow_index = np.argmax(wcss_diff_percentage_change < 0.1) + 1
    optimal_k = elbow_index+1
    best_wcss = wcss[elbow_index]
    print("Optimal number of clusters (k):", optimal_k)
    print("Best WCSS value:", best_wcss)
    plt.plot(range(1, 11), wcss)
    plt.title('The Elbow Method')
    plt.xlabel('Number of clusters')
    plt.ylabel('WCSS')
    plt.show()
    return optimal_k
```

This method uses Elbow Method to find the optimal number of clusters (k).

Within-cluster sum of square(wcss) is calculated for each number of cluster and stored in the wcss list. The code then plots the wccs values against the number of clusters to help determine the optimal number of cluster visually. The “Elbow” point is where the plot stars to bend, indicating the optimal number of clusters.

Instead of manually studying graph and determining best cluster, we have completely automated this process via coding. We first calculate the differences between consecutive WCSS values and then calculate the relative percentage change for each difference. When the percentage change is minimal we select that elbow point. For understanding it properly, we decided to print graph for this step.



The plot bend at 6 hence it was most optimal choice.Beyond this point, the curve starts to flatten, suggesting that adding more clusters may not significantly reduce the WCSS. Therefore, the best WCSS value in this case would be the WCSS value corresponding to k=6.

Once we have obtained best number of cluster(in our case it was 6) we will perform k-means clustering and visualize it.

- **main method**

```
54 ▶ if __name__ == "__main__":
55       while True:
56           user_input = input("Do you want to run programme(yes/no)? ").strip().lower()
57           if user_input == "no":
58               print("Exiting the programme")
59               break
60           elif user_input == "yes":
61               window_size = int(input("Enter Current Window size: "))
62               print("Window size = ",window_size)
63               k_clustering(window_size)
64           else:
65               print("Invalid input. Please enter 'yes' or 'no'.")
66
```

In the main part of the code, it asks the user whether to run the program and proceed with clustering. If the user enters "yes," it takes the window size as input and calls the **k_clustering(window_size)** function to perform the clustering. The program continues to ask the user if they want to run the program until the user enters "no."

We also ask the user to input window size via terminal. **window_size** is used to implement sliding window functionality, we can also automate this process by calculating **window_size** from date and time but here we decided to request it from user.

```
Do you want to run programme(yes/no)? yes
Enter Current Window size: 0
Window size =  0
```

Different window size will result in different cluster formations. For Window size 0 we get optimal cluster value as 6 when is printed in terminal.

```
Optimal number of clusters (k): 6
Best WCSS value: 37233.81451071001
```

But for window size 10 we get

```
Do you want to run programme(yes/no)? yes
Enter Current Window size: 10
Window size =  10
Optimal number of clusters (k): 4
Best WCSS value: 63302.75011246064
```

Cluster value is 4

Conclusion

We have successfully implemented Sliding Window Algorithms for k-Clustering. This project aimed to address the challenges of handling dynamic data streams efficiently and updating clustering results. By utilizing a sliding window approach, we can easily update clustering result, making it well-suited for real-time applications. The algorithms' ability to adapt to changing data distributions has proven valuable in maintaining clustering accuracy even in dynamic environments.

While we have accomplished our immediate goals, we recognize that there is room for further enhancement. The data updating can be made more efficient and automated by making a **time function** which automatically select latest data only, removing the manual input required by user to set **window_size**.

Future work could focus on exploring different clustering algorithms suitable for sliding window paradigms or investigating ways to optimize the algorithms for specific data types or application domains.

References

- https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html
- https://cs.paperswithcode.com/paper/sliding-window-algorithms-for-k-clustering
- https://github.com/google-research/google-research/tree/master/sliding_window_clustering
- https://paperswithcode.com/conference/neurips-2020-12
- https://realpython.com/k-means-clustering-python/
- https://itnext.io/sliding-window-algorithm-technique-6001d5fbe8b3