Extending the PDC-DP-means Algorithm to Streaming data

Amit Ezer 316097575, Tzvi Greenfeld 209300581

1 Introduction

In this project, we adapted the DPMeans clustering method from the pdc_dp_means package to handle streaming data. This adaptation centers around the way we initialize and update the centroids and the number of clusters for each batch of incoming data.

At the start, we initialize the algorithm with default values, using the "k-means++" strategy for the initial centroids and setting a default number of clusters.

As we start to process the data stream, the core of our adaptation takes effect. We loop over each batch of data (for synthetic data, we consider 2D points, and for images, we consider pixel values). Instead of running DPMeans with fixed initial parameters, we update these parameters based on the result of the previous batch. This allows the algorithm to adapt to the changing data stream.

Specifically, if there are prior centroids available, we compute a weighted average of these centroids. We assign smaller weights to older centroids, using a decay factor of 0.9 to decrease the weight for each successive older centroid. This ensures that more recent centroids have a greater influence on the initialization for the current batch, allowing the algorithm to adapt more effectively to recent changes in the data.

Listing 1: implementation of decay factor

```
weights = [np.pow(0.9, i + 1) for i in range(len(centroids_agger))]
prev_centroids = np.average(centroids_agger, axis=1, weights=weights)
```

We then run the DPMeans clustering on the current batch with the updated initial centroids and number of clusters. After fitting the model and predicting the cluster assignments, we extract the new centroids from the fitted model. We store the data, cluster assignments, and centroids (or the image processed according to these clusters) for potential output and further use.

Finally, we update the 'previous centroids' and 'previous number of clusters' based on the current batch. This includes stacking the new centroids to the list of previous centroids, ensuring we consider them in the initialization of the next batch. The number of clusters is updated to the number of unique centroids obtained in the current iteration.

This continuous updating of initialization parameters ensures that our DP-Means clustering method remains responsive to the streaming nature of the data, where the underlying distribution might change over time. Thus, we create a form of "streaming DPMeans" which can handle evolving data streams.

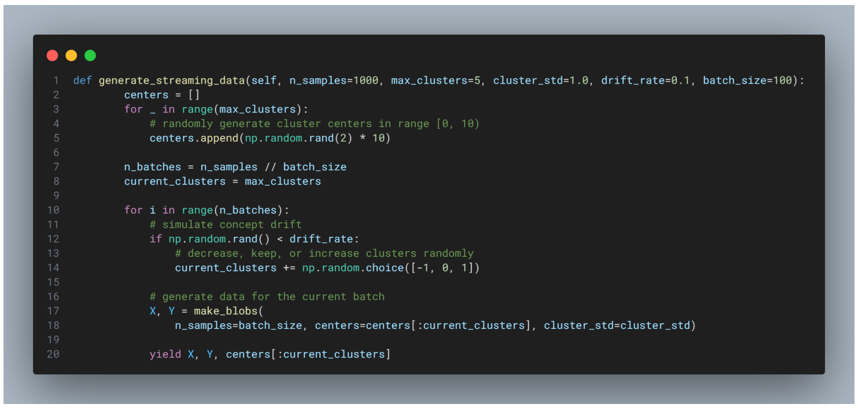
2 Synthetic Data

2.1 Generation

The synthetic data was partitioned into a series of batches, each composed of a defined number of samples derived from Gaussian blobs—clusters of points in the 2D space. The initial set-up featured a maximum number of clusters, each with a random center within the defined space. Notably, this process was designed to permit variability in the cluster count between consecutive batches, adding a layer of complexity and dynamism to the data.

One of the key features of our synthetic data generation was the introduction of ’concept drift’. This allowed the data to ’move’ or ’drift’ between batches, a common phenomenon in real-world streaming data. Concept drift was controlled by a predefined drift rate, which determined the probability of a change, or ’drift’, occurring between batches.

In addition to the drifting data, we also implemented changes in the number of clusters between batches. Upon the occurrence of a concept drift, the number of clusters in the following batch could either increase, decrease, or remain unchanged, the outcome of which was decided randomly.



```python
def generate_streaming_data(self, n_samples=1000, max_clusters=5, cluster_std=1.0, drift_rate=0.1, batch_size=100):
        centers = []
        for _ in range(max_clusters):
            # randomly generate cluster centers in range [0, 10)
            centers.append(np.random.rand(2) * 10)

        n_batches = n_samples // batch_size
        current_clusters = max_clusters

        for i in range(n_batches):
            # simulate concept drift
            if np.random.rand() < drift_rate:
                # decrease, keep, or increase clusters randomly
                current_clusters += np.random.choice([-1, 0, 1])

            # generate data for the current batch
            X, Y = make_blobs(
                n_samples=batch_size, centers=centers[:current_clusters], cluster_std=cluster_std)

            yield X, Y, centers[:current_clusters]
```

Figure 1: The method used to generate synthetic data

2.2 Real Synthetic data clusters

In the figure below, we show the 'real' clusters generetaed, including drifting and varying number of clusters

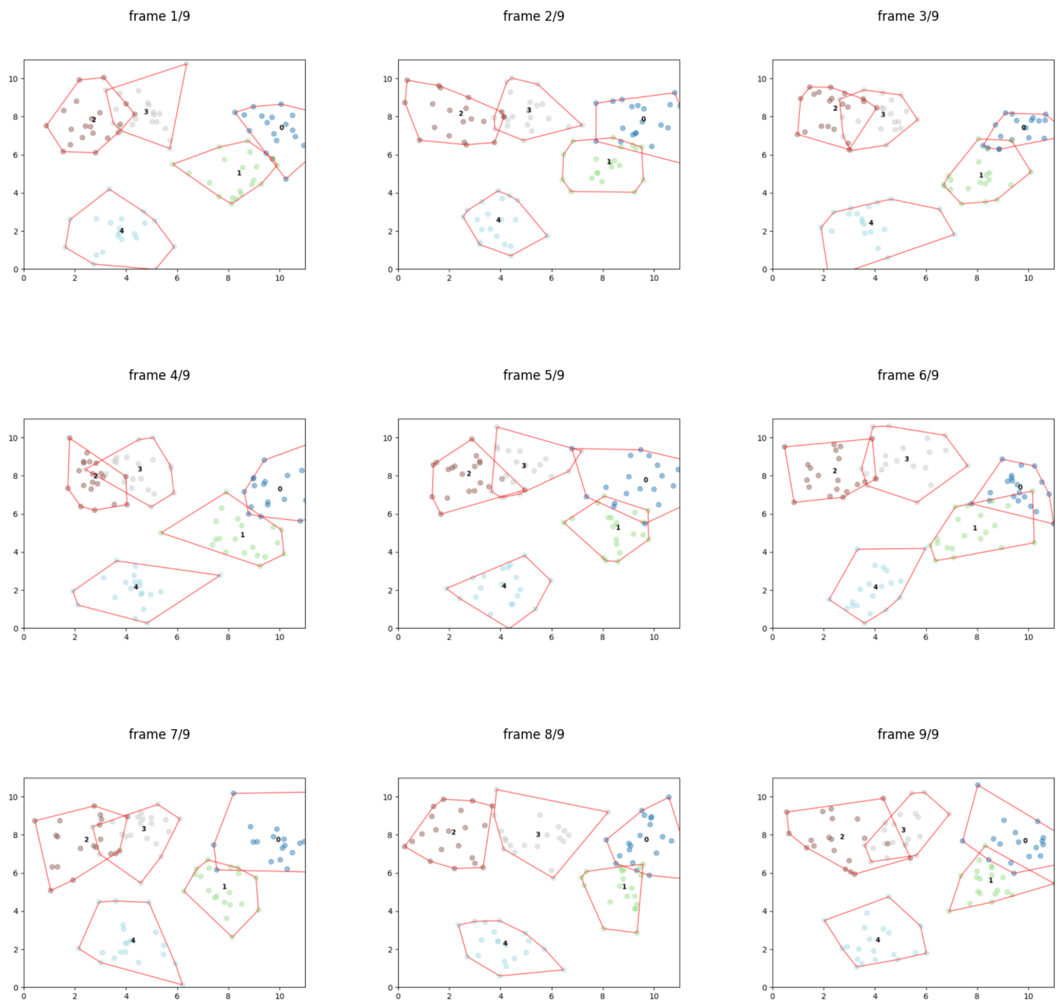


Figure 2: Visualization of clustering by gaussian blob

2.3 PDC-DP Synthetic data clusters

Here we can see the same data clustered by our algorithm, with $\lambda = 10$

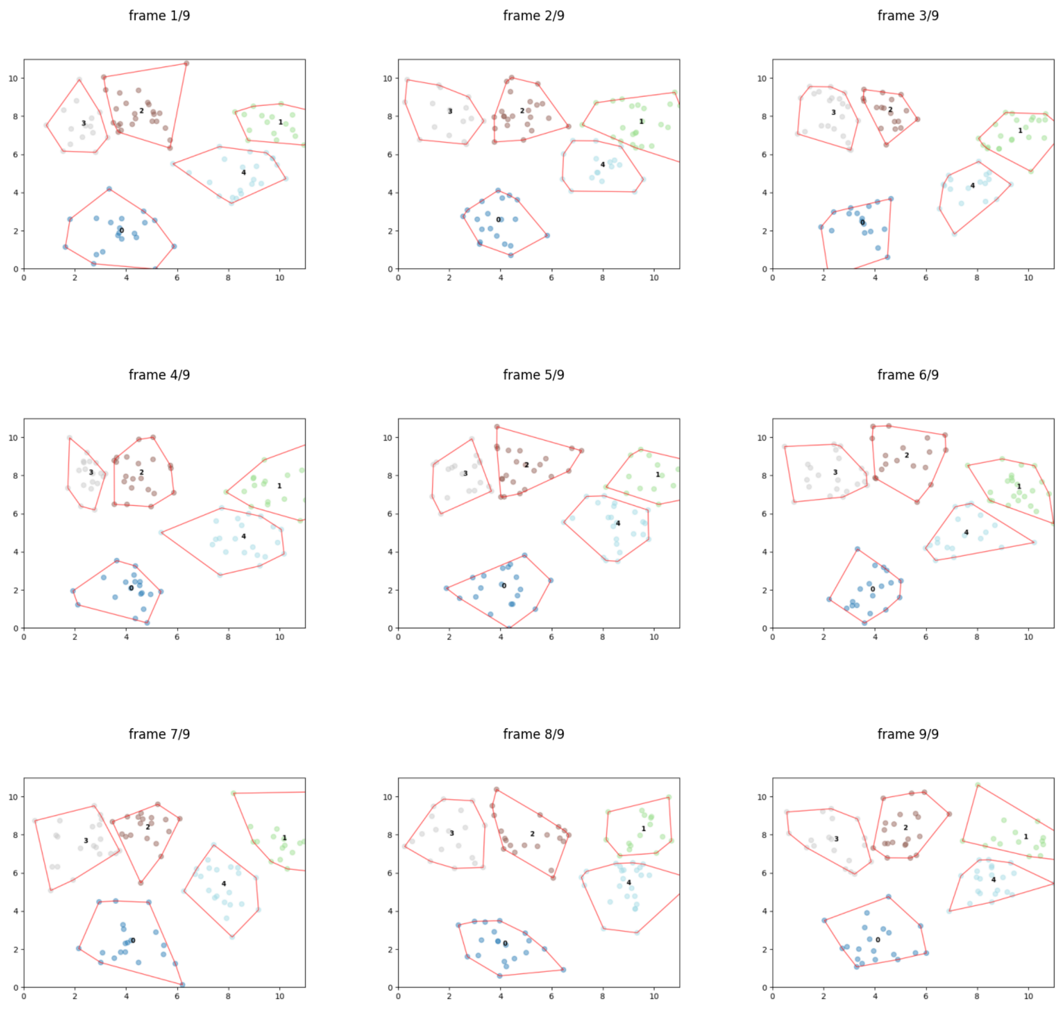


Figure 3: Visualization of clustering with PDC-DP

3 Real Data

3.1 Data Process

The real data used in this project consisted of a collection of videos featuring mandrills. The first step involved extracting individual frames from each video by traversing through the video frames at the specified fps given as a parameter of the program. These frames were then stored as a sequential list.

Our adaptation of the PDC-DP means clustering algorithm was applied with the objective of grouping pixels that exhibited similar colors. the output frame would consist of all the pixels colored according to their respective centroid color within the cluster. This process allowed us to create a video where each frame represented the pixels colored in their centroid color.

Finally, after executing the clustering algorithm on the list of frames and generating the output frames colored by their centroid colors, we reconstructed the frames back into a video format. This enabled us to visually observe the clustering results.

3.2 Clustering

when processing the streaming data, we ran the PDC-DP means algorithm for each frame individually. However, unlike traditional batch processing, we incorporated parameters based on the results of the previous frames just as described in introduction section. This approach enabled the algorithm to adapt and accommodate recent changes in the data more effectively.

By utilizing information from the previous frames, the PDC-DP means algorithm could utilize the knowledge gained from prior clustering results. This allowed it to make more informed decisions about the clustering process for the current frame. The algorithm could dynamically adjust parameters based on the characteristics and clustering patterns observed in the previous frames.

This adaptation was crucial for handling streaming data, as it enabled the algorithm to be more responsive to concept drift and variations in the data. By incorporating knowledge from past frames, the PDC-DP means algorithm could adapt its clustering strategy and benefit over the runtime for real-time applications, ensuring that it accurately captured the changing patterns and dynamics of streaming data.

3.3 Side by side comparison

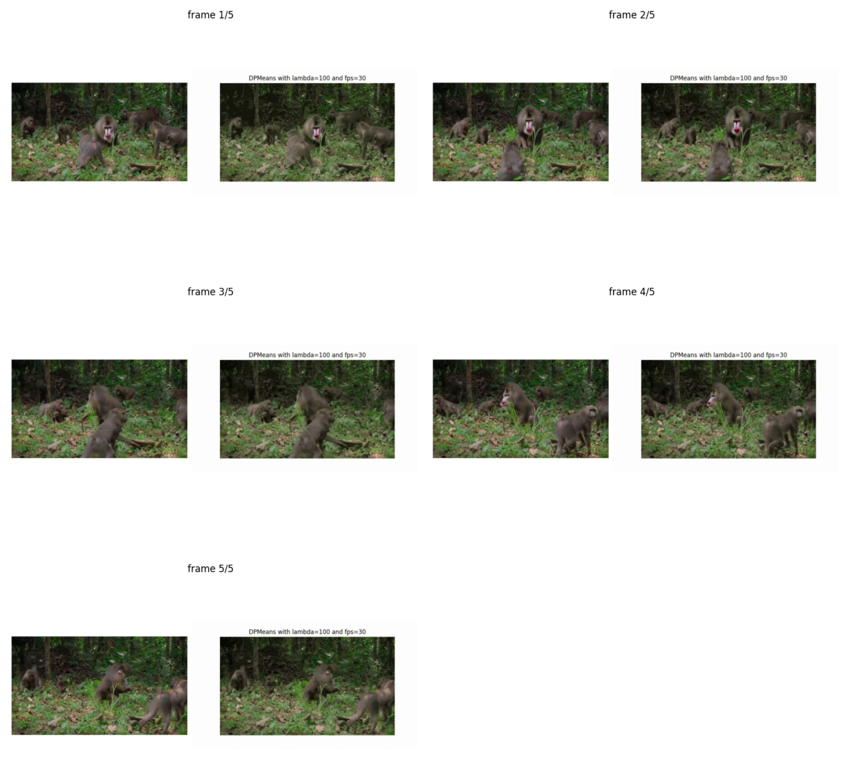

frame 1/5
DPMeans with lambda=100 and fps=30
frame 2/5
DPMeans with lambda=100 and fps=30
frame 3/5
DPMeans with lambda=100 and fps=30
frame 4/5
DPMeans with lambda=100 and fps=30
frame 5/5
DPMeans with lambda=100 and fps=30

Figure 4: Left: frame from the real video. Right: recolored image after clustering

(The video side by side comparison is available at 'report/many_mandrils_comaprsion.mp4')

3.4 Conclusions

The real data presented a number of challenges that were not present in the synthetic data. First, the real data was much more complex than the synthetic data. The mandrills in the videos moved around in a variety of ways, and their appearance changed over time (This inherent variability and evolution in the data reflected the concept of drift). This made it more difficult to cluster the data.

Despite these challenges, we were able to achieve good results with our adaptation of PDC-DP means algorithm on the real data. The algorithm was able to cluster the data effectively, even though the data was complex and subject to concept drift.

4 Further use

Clustering pixels by their colors on streaming data using the PDC-DP means algorithm can offers valuable benefits in video compression and object tracking. It enables efficient video compression by exploiting color similarities and reducing redundancy.

Additionally, it empowers robust object tracking by leveraging color clustering and adapting to changes in the data. These applications showcase the practical utility of clustering pixels by their colors in streaming data scenarios, demonstrating its effectiveness in both data compression and visual analysis tasks.