

Example of an output:

News Source: Reuters, Likes: 54, Cluster: 3

Writeup:

This project aimed to implement K-Means clustering based on the number of likes an article received, using a dataset that includes four primary news sources: Al Jazeera, BBC, CNN, and Reuters. Using the “Global News Engagement on Social Media” dataset from Kaggle, I explored insights about how different news sources perform in terms of social engagement, specifically using likes. Clustering by likes could be useful for understanding how different news sources engage their audience, or for analyzing patterns in the popularity of articles.

Initially, I began with importing the external crates “ndarray,” to enable the use of multi-dimensional arrays, and “csv” for reading CSV files. Then, I imported all the necessary types and functions including “Array1” and “Array2,” which are used for 1D and 2D arrays, “Error,” which is used for error handling, “ReaderBuilder” which allows configuring CSV file reading, “file” to handle files, and lastly, “HashMap” to map file paths to new sources.

Next, I began creating my functions to use in K-Means clustering. First, I create a function called “initialize_centroids.” The purpose of this function is to initialize “k” centroids by selecting the first “k” rows from the data. It works by accepting two arguments, and a 2D array of input data, which are the number of likes, and k, which is the number of clusters. The function then creates “Vec<Array1<f64>>” to store the centroids, looping over the range 0 to k and selecting the i-th row from the data array to be the centroid. The row is then cloned and added to the centroids vector. Then, I created a function called “distance” which calculates the Euclidean distance between two points which are the inputs, p1 and p2. This works by using the Euclidean distance formula with p1 and p2 which is the squared difference between p1 and p2, sums the squared differences, and returns the square root of the sum. My next step was to create another function called “find_closest_centroid” which calculates the distance between a point and each centroid to find the closest one. It does this by taking two arguments, a data point and the list of centroids. Then the algorithm iterates over all centroids, calculates the distance from the point using the previously created “distance” function, and finds and returns the index of the centroid with the smallest distance as it is the closest centroid. Next, once all points have been assigned to a cluster, the “recompute_centroids” function recomputes the centroids of each cluster based on the current cluster. This is done by averaging all the points that belong to each cluster. Computationally, this is done by intaking three arguments: an array dataset, a list of the cluster assignment for each data point, and k. First, it creates two lists, new_centroids, which store the new center for each cluster and counts which count how many points belong to each cluster. Then, for every point in the dataset, it adds the point's values to the corresponding cluster's new centroid and increases the count for how many points belong to that cluster. After iterating through all the data, the algorithm divides the total sum of each cluster's points by how many points belong to that cluster, giving the average value for each cluster, which becomes the new centroid for that cluster. This updates the center of the clusters so that they are more accurate in portraying the points assigned to them. My last function is called “has_converged” which checks if centroids are converged, meaning they no longer change significantly between iterations. This method works by taking three arguments, the list of centroids from the previous iteration, the list of centroids from the current iteration, and tolerance which is used as a threshold to determine if the centroids are still changing or not. The algorithm checks if the distance between each pair of old and new centroids is less than the tolerance value. If all centroids are sufficiently close to their previous positions, this means the centroids are no

longer changing significantly. In this case, the algorithm has "converged," and further iterations will not improve the results so the the algorithm stops because the clusters have stabilized.

My main function is where the data is loaded from CSV files, prepared for clustering, and the K-means algorithm is run to classify the data points into clusters. After clustering, it prints the results. To begin, using a hashmap, the code begins by mapping the CSV file paths, the key, to a custom name, the value. The next step is to read the CSV files. The algorithm reads the files and extracts the necessary data, filtering out rows where the number of "likes" is zero or less. The data is then prepared for clustering by extracting the "likes" data and transforming it into a 2D array. K-means clustering is then first performed, where $k = 4$ because there are four possible news sources an article could be from. Using the `initialize_centroids` function, the centroids are randomly initialized. Then the K-Means iterations are performed where each point is assigned to the closest centroid and then the centroids are recomputed. Then, it is checked for convergence. The process repeats until convergence. Finally, each news is mapped to its cluster, and the results are printed of each news source, the number of likes, and the cluster assignment for each data point.

Lastly, I created two test functions, "`test_cnn_clustering`" and "`test_bbc_clustering`," that simulate clustering on CNN and BBC datasets. The tests load the CSV files and perform clustering for the specific dataset. Then, the test verifies that the news source is correctly classified and that each data point has been assigned to a valid cluster. Ultimately, these functions help confirm that the clustering algorithm behaves as expected for individual datasets.