Homework 6
Software Quality
Group 06

Okikioluwa Ojo (100790236)

Date: Mar 5, 2024

GitHub Link: https://github.com/okikio-school/Homework-6

You will use the following open source java Machine Learning package to do some data clustering. http://java-ml.sourceforge.net/

You read the getting started section to set up this library. The assignment is for you to compare the performance of three clustering algorithms (including Kmeans and you choose any two other algorithms). Use the Iris flower dataset we discussed in class.

Note you will use the same method signature to do the clustering … in spite of a different algorithm being employed? Can you tell why???

You need to explore how to evaluate each of the three algorithms. You need to read the documentation to explore how and what to evaluate? Did you use the same method signature to evaluate as well? Why?

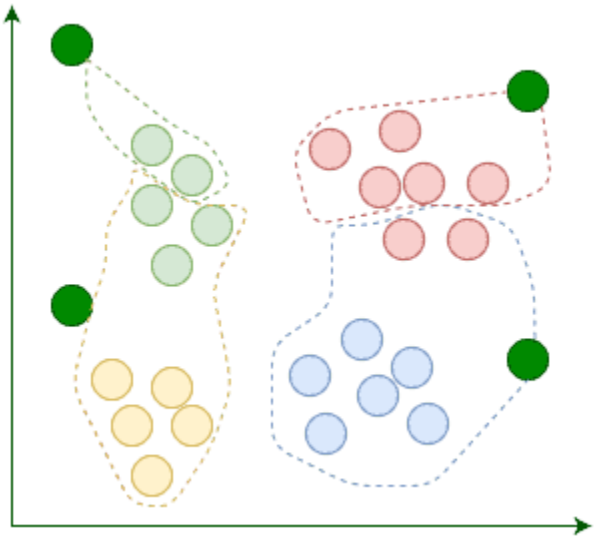
The clustering algorithm can be summarized as finding a pattern and reinforcing it into clusters, the basic shape of the algorithm is one that takes data and returns a modified set, where the data is placed into clusters.

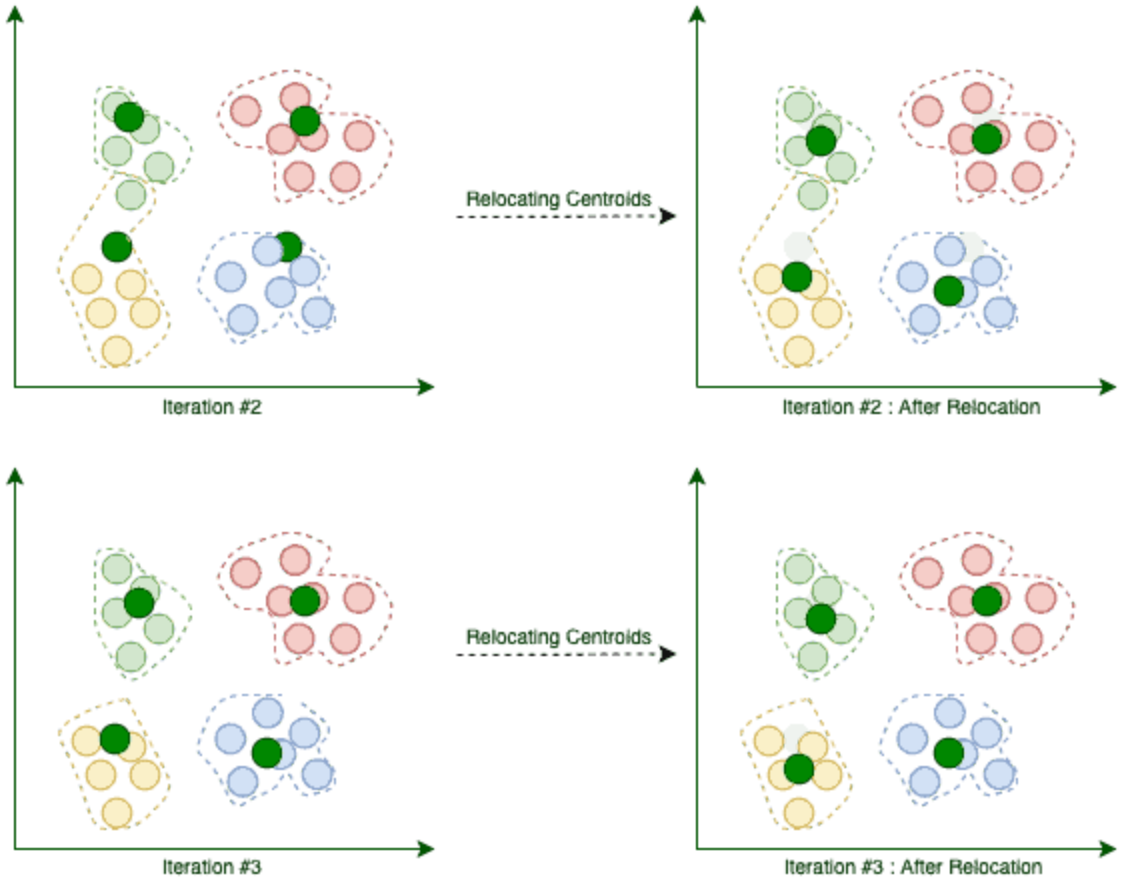
- Kmeans is a centroid-based clustering algorithm, it partitions datasets into a predefined set of clusters which is what we refer to when as `k` in KMeans (in the demo code I use 4 clusters, which is the default JavaML comes with). The KMeans algorithm iteratively assigns each data point (instance) to the nearest cluster center (this is the centroid part of the algorithm), it uses distance to identify the cluster center. The centroids are recalculated until they stabilize (e.g. the centroids no longer really change position so we classify the centroid as having reached stability).
    - KMeans aims to partition the data set into distinct non-overlapping clusters, where each instance belongs to the cluster with the nearest mean (nearest centroid), as KMeans minimizes the variance within a cluster. Basically we want more of the same type of data within the cluster.
    - KMeans starts by initializing K centroids randomly.
    - Each instance is assigned to the nearest centroid based on distance.
    - The centroids of the clusters are recalculated as the mean of all points assigned to the cluster.
    - This process repeats until the centroids no longer change significantly

  To evaluate the performance of KMeans we can use metrics like Inertia, and the Silhouette Score. Inertia refers to the sum of squared distances to their closest cluster center. Lower inertia implies a better algorithm. For the Silhouette Score, it measures an instance to other instances in its own cluster. The silhouette

ranges from -1 to 1, a high value indicates that the object matches its own cluster fairly well, and implies that the other clusters are poor matches.

Wonderful resource https://www.baeldung.com/java-k-means-clustering-algorithm





- Farthest First, is a simpler implementation of KMeans, it selects centroids that are as far away from each other as possible, without iterating. What we mean by without iteration is that we do not repeatedly refine the choice of centroid through multiple passes of the data. Instead, we identify centroids based on the principle of maximizing the minimum distance between any set of points chosen to be centroids; this helps us spread out the centroids as much as possible within the dataset from the outset. By not using iteration we keep the algorithm simple, fast, and efficient.
    - The algorithm starts by selecting a random point as the first centroid.
    - For each subsequent centroid the algorithm selects the data point that is farthest away from any centroid that's already been chosen.
    - We continue this process until the desired number of centroids (k) have been met.
    - One key thing to note is that unlike KMeans, which aims to minimize the differences between instances in a cluster, FartestFirst aims to have a

more diverse coverage for the dataset, all without needing to adjust the centroids based on cluster assignment.

Like KMeans, the Farthest First algorithm can be evaluated using inertia and silhouette score.

- Cobweb is an incremental clustering algorithm which uses a tree structure to dynamically organize data into hierarchical trees of clusters, generally meant for categorical data. When a new instance is introduced, the Cobweb algorithm either merges it into an existing cluster, creates a new cluster, splits an existing cluster or merges 2 clusters together. The decision is based on maximizing the probabilistic measure (category utility function).
    - Creates an empty tree.
    - For each data point (instance), the algorithm traverses the current tree to find the best cluster (each cluster counts as a node in the tree) that can accommodate the new instance, based on probabilistic measure (this is often called the category utility function) that evaluates how well the instance fits into a potential cluster.
    - Depending on the probability, Cobweb will either add the instance to a cluster node, create a new cluster if it doesn't fit into existing clusters, merge to clusters nodes into one if it improves the probabilistic measure, or split a cluster node into 2 separate clusters if it improves the probabilistic measure
    - This is then repeated for each instance, until the all the points have been defined under a cluster

  The category utility function is a criterion for determining the quality of a clustering structure based on how well each of the instances fit into the category of the cluster node. The measure assesses both the probability of predicting attribute values within clusters (basically if the category is teens, the algorithm should be able to accurately predict that the age from 13 - 19, if it can't as in the category utility function is incorrect it takes one of the actions listed above ^). Cobweb, is very similar to KMeans in the results they're aiming for, but unlike KMeans which aims to minimize differences in a cluster (passive), Cobweb aims to maximize similarities in a cluster (active)

  Evaluating Cobweb can be tricky, but one way of doing it is to measure the overall classification quality against known categories that exist in the data.

Can you print the output of each algorithm in a manner that shows the different clusters?