Mourya Meda

Clustering

The dataset I have selected for the following implementation of k-means is the Iris dataset from the UCI Machine Learning Repository. The Iris dataset is in the following format:

```
Attribute Information:
   1. sepal length in cm
   2. sepal width in cm
   3. petal length in cm
   4. petal width in cm
   5. class:
      -- Iris Setosa
      -- Iris Versicolour
      -- Iris Virginica
```

In the following implementation I decided to only look at the quantitative data and ignore the class attribute. This helped calculate the Euclidean distance in a simple manner of comparing two ArrayLists<Doubles>.

The main quality metric I used is cohesion for each of the clusters and Sum Squared Error:

```
/**
 * Calculates quality measures such as SSE through cohesion
 */
public static void calculateQuality() {
    double cohesion = 0;

    for(Cluster cluster : clusterList) {
        // calculate cohesion
        for (Point point : cluster.getPoints()){
            cohesion += point.getDistanceToCentroid();
        }
        cluster.setCohesion(cohesion);
        sse+= cohesion;
    }
}
```

In order to first preprocess the data I had to convert the data into a data structure that can be easily handled. I created a Point class that holds and ArrayList<Doubles> for the quantitative data (sepal length in cm, sepal width in cm, petal length in cm, petal width in cm). Furthermore, the Point class is also defined by a String type which is the class of the tuple. This class String is useful when calculating the centroid point and defining its type. I took the mode of all the class of the points in the cluster and assigned that to the centroid. Other than that, it plays no further role.

The constructer for the Point class is as follows:

```
public Point(ArrayList<Double> x, String type)
{
    this.setPoints(x);
    this.setType(type);
    this.distanceToCentroid = 0;
}
```

Another class created is Cluster which holds ArrayList<Points>, the id of the cluster, its centroid, and its cohesion:

```
public Cluster(int id) {
    this.id = id;
    this.points = new ArrayList<Point>();
    this.centroid = null;
    this.cohesion = 0;
}
```

Other than removing the class attribute when running the k-means algorithm, there have been no other preprocessing of the data, as it was very simple and did not need any to take into account such as missing or extreme data points.

Surprisingly the SSE of Weka differed significantly compared to my program. However the size of each clusters when k = 2 is very close:

Weka:

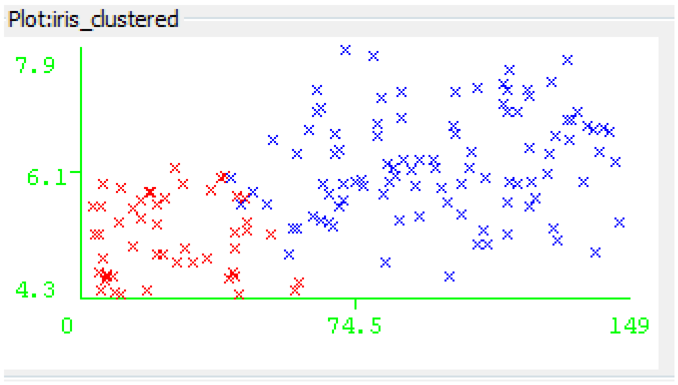
	Cluster 0	Cluster 1	SSE	Iterations
Size	100	50	12.143688281579722	7

k-Means Implementation:

	Cluster 0	Cluster 1	SSE	Iterations
Size	53	97	159.47239643973742	6

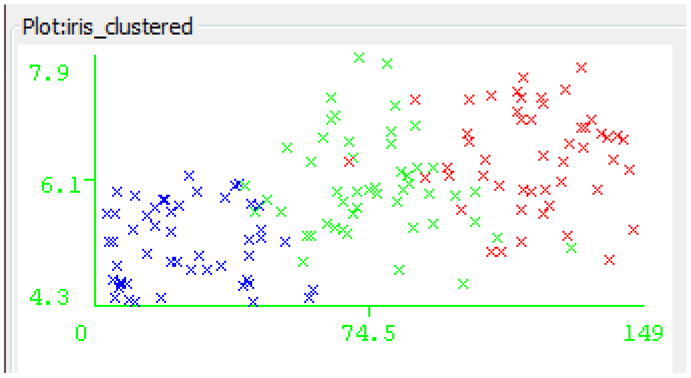
The significant difference in SSE is surprising and have still not figured out why that is the case. I predict it may be the result of my random choice of centroid to begin with, or in my calculations of the SSE.

Some of the insights I gained from working with the Iris dataset is that the higher the k the SSE increases. This could be because of the significant clustering of two clear groups found when plotted:



However, this is not the case when it comes to Weka in which is shows improvement with SSE. Once again, this leads me to believe there may be some different manner in which the centroids are picked initially and SSE is calculated.

When the dataset is run with density-based spatial clustering of applications with noise, there is a huge clarity in the clustering and is much more accurate in determining the class separations:



With the Epsilon being only 0.9 and min points of 6. This algorithm is very accurate in noticing any outliers and noisy data by utilizing density and grouping together points that are closely packed. Something k-means is weak against.