Distributed Systems

Lab 1

Parallel K-Means using Hadoop

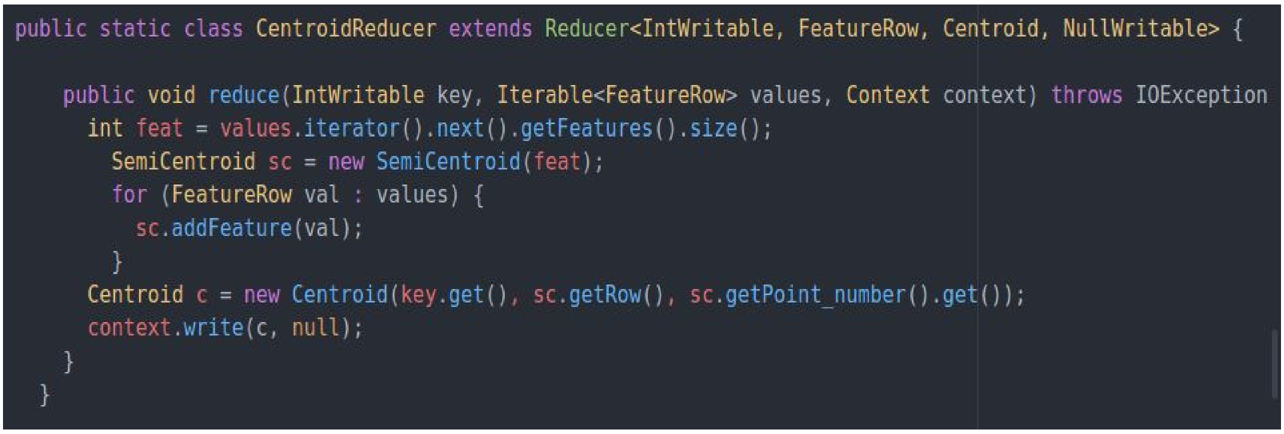
Members :

- Amr Mohamed Nasreldin Elsayed (47)
- Michael Raafat Mikhail (57)
- Amira Nabil ( )

Unparallel K-means Pseudo-Code:

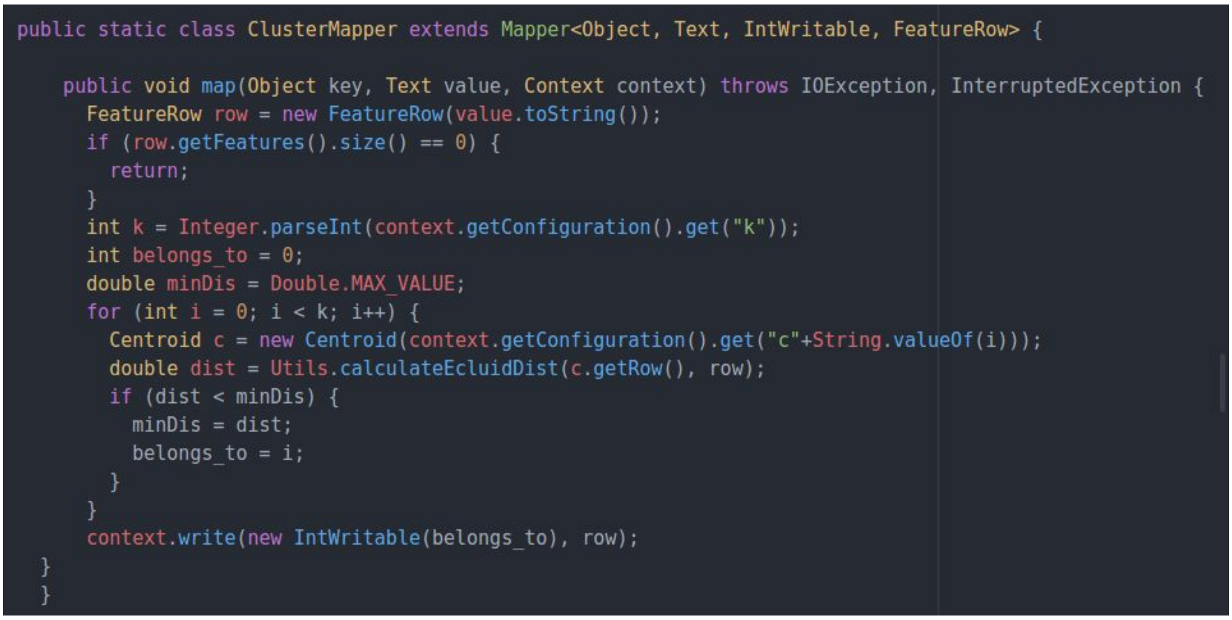
```
Function K-means ( K : number of clusters, D : dataset of samples) :
    1. Initialize k cluster centroid randomly : M(1), M(2), …. to M(k).
    2. Repeat Until Convergence:
        a. For every sample i in D :
            i. C(i) = $\text{argmin}_j$(ecludian_distance(D(i) - M(j))
        b. For j from 1 to K:
            i. M(j) = Mean(any sample i where C(i) == j)
    3. Return the cluster centroids.
```

Map-Reduce algorithm:



```java
public static class CentroidReducer extends Reducer<IntWritable, FeatureRow, Centroid, NullWritable> {

  public void reduce(IntWritable key, Iterable<FeatureRow> values, Context context) throws IOException
   int feat = values.iterator().next().getFeatures().size();
    SemiCentroid sc = new SemiCentroid(feat);
    for (FeatureRow val : values) {
     sc.addFeature(val);
    }
   Centroid c = new Centroid(key.get(), sc.getRow(), sc.getPoint_number().get());
   context.write(c, null);
  }
 }
```



```
public static class ClusterMapper extends Mapper<Object, Text, IntWritable, FeatureRow> {

    public void map(Object key, Text value, Context context) throws IOException, InterruptedException {
      FeatureRow row = new FeatureRow(value.toString());
      if (row.getFeatures().size() == 0) {
        return;
      }
      int k = Integer.parseInt(context.getConfiguration().get("k"));
      int belongs_to = 0;
      double minDis = Double.MAX_VALUE;
      for (int i = 0; i < k; i++) {
        Centroid c = new Centroid(context.getConfiguration().get("c"+String.valueOf(i)));
        double dist = Utils.calculateEcluidDist(c.getRow(), row);
        if (dist < minDis) {
          minDis = dist;
          belongs_to = i;
        }
      }
      context.write(new IntWritable(belongs_to), row);
  }
  }
```

<ins>We make some classes and util class that helps our algorithm:</ins>

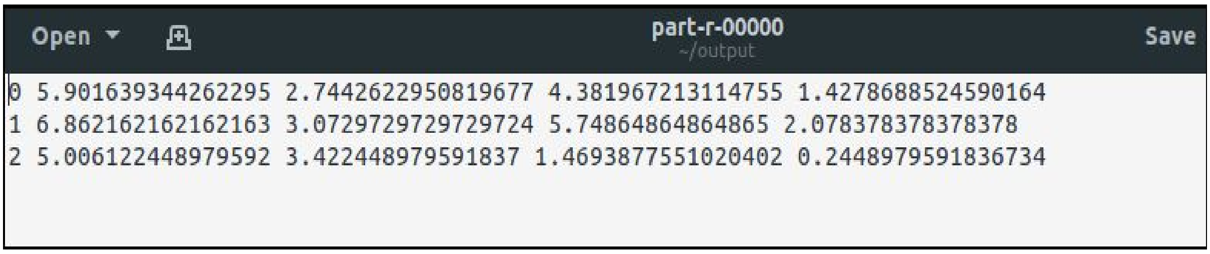
- Feature Row
- Centroid
- Utils

Challenges Faced:

- **Passing Feature Row per Sample in Mapper:**
  We decided to parse Text to String that represent value of features per sample, then converting these values to Feature Row.
- **How to get Initial Centroid:**
  We decided to set initial centroid randomly picking k-samples from the initial file.
- **How to pass results of each round:**
  Pass centroids as part of the configuration, meaning that after each round of output, read generated output to use it in the configuration after it.

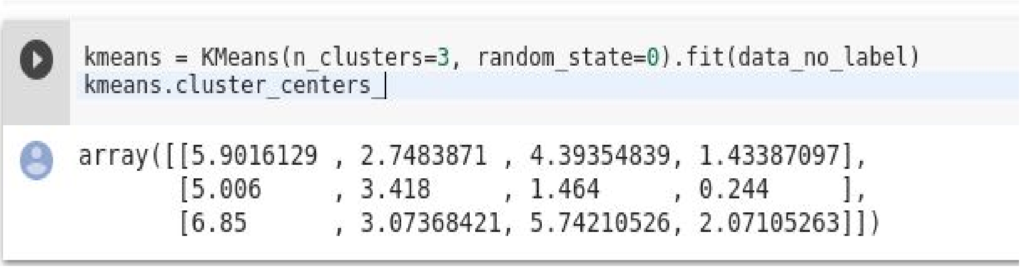
Evaluation Results:

Our Results are nearly the same as using Kmeans in Sklearn library in python.
We can see that both results converges to nearly same result.



```
0 5.901639344262295 2.7442622950819677 4.381967213114755 1.4278688524590164
1 6.862162162162163 3.0729729729729724 5.74864864864865 2.078378378378378
2 5.006122448979592 3.422448979591837 1.4693877551020402 0.2448979591836734
```

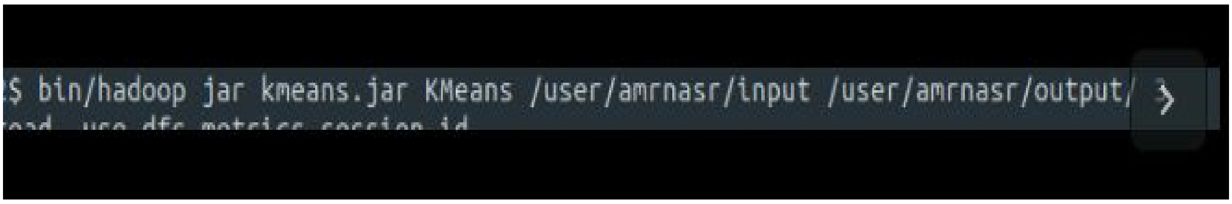
F_Measure is 0.6818495514147688
Conditional Entropy is 0.27302119105777406



```
kmeans = KMeans(n_clusters=3, random_state=0).fit(data_no_label)
kmeans.cluster_centers_
```

```
array([[5.9016129 , 2.7483871 , 4.39354839, 1.43387097],
       [5.006     , 3.418     , 1.464     , 0.244     ],
       [6.85      , 3.07368421, 5.74210526, 2.07105263]])
```

<u>**Notes:**</u> We formed a jar file for our K-means Algorithm as in WorldCount Example.



```
:$ bin/hadoop jar kmeans.jar KMeans /user/amrnasr/input /user/amrnasr/output/
```