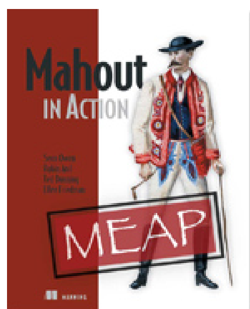
MANNING PUBLICATIONS



Mahout in Action
Sean Owen, Robin Anil, Ted Dunning, and Ellen Friedman

There are many clustering algorithms in Mahout, and some work well for a given dataset while others don't. K-Means is a very generic clustering algorithm, which can be molded easily to fit almost all situations. In this article based on chapter 9 of Mahout in Action, the authors discuss Fuzzy K-Means, an algorithm that tries to generate overlapping clusters from a dataset.

You may also be interested in…

Running Fuzzy K-Means Clustering

There are many clustering algorithms in Mahout, and some work well for a given dataset while others don't. K-Means is a very generic clustering algorithm, which can be molded easily to fit almost all situations. It's also simple to understand and can easily be executed on parallel computers.

K-Means is to clustering what Vicks is to cough syrup. It's a simple algorithm and is more than 50 years old. Stuart Lloyd first proposed the standard algorithm in 1957 as a technique for pulse code modulation. However, it wasn't until 1982 before it got published[1]. It's widely used as a clustering algorithm in many fields of science. The algorithm requires the user to set the number of clusters, k, as the input parameter.

What we really want to talk about is the Fuzzy K-Means algorithm, but in order to explain it we first need you to know what K-Means is all about.

All you need to know about K-Means

K-Means algorithm puts a hard limitation on the number of clusters, k. This limitation might cause you to doubt the quality of this method. Fear not—this algorithm has proven to work very well for a wide range of real-world problems over the last 25 plus years of its existence. Even if the estimate of the value k is suboptimal, the clustering quality is not affected much by it.

Suppose we are clustering news articles to get top-level categories like politics, science, and sports. For that we might want to choose a small value of k, which is in the range of 10 to 20. If you're looking to cluster some subtopics, you need a larger value of k, like 50 to 100. Imagine there are one million news articles in our database and you are trying to find out groups of articles that discuss the same story. The number of such related stories would be much smaller than the entire corpus—perhaps, in the range of 100 articles per cluster. This means, we need a k value of 10000 to generate such a distribution. This surely would test the scalability of clustering and that's where Mahout shines the brightest.

For good quality clustering using K-Means, we will need to estimate the value of k. An approximate way of estimating k is to figure it out based on the data we have and the size of clusters we need. In the case above, if there are around 500 news articles published about every story, we should be starting our clustering with a k value like 2,000.

This is a crude way of estimating the number of clusters. Nevertheless, K-Means algorithm generates decent clustering even with this approximation. The type of distance measure used mainly determines the quality of K-Means clusters.

Let's look at K-Means algorithm in detail. Suppose we have n points, which we need to cluster into k groups. K-Means algorithm will start with an initial set of k centroid points. The algorithm does multiple rounds of the

[1] http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.131.1338

processing and refines this centroid location until the iteration max-limit criterion is reached or until the centroids converge to a fixed point from which it doesn't move very much. A single K-Means iteration is illustrated clearly in figure 1. The actual algorithm is a series of such iteration, till it encounters the criteria above.

There are two steps in this algorithm. The first step finds the points, which are nearest to each centroid point, and assigns them to that specific cluster. The second step recalculates the centroid point using the average of the coordinates of all the points in that cluster. Such a two-step algorithm is a classic case of what is known as Expectation Maximization (EM) Algorithm.[2] The algorithm is a two-step process, which is repeated until convergence is reached. The first step—the expectation (E) step—finds the expected points associated with a cluster. The second step, known as the maximization (M) step, improves the estimation of cluster center using the knowledge from the E step. A complete discourse on expectation maximization is beyond the scope of this article, but plenty of explanations and resources on EM are found online[3].

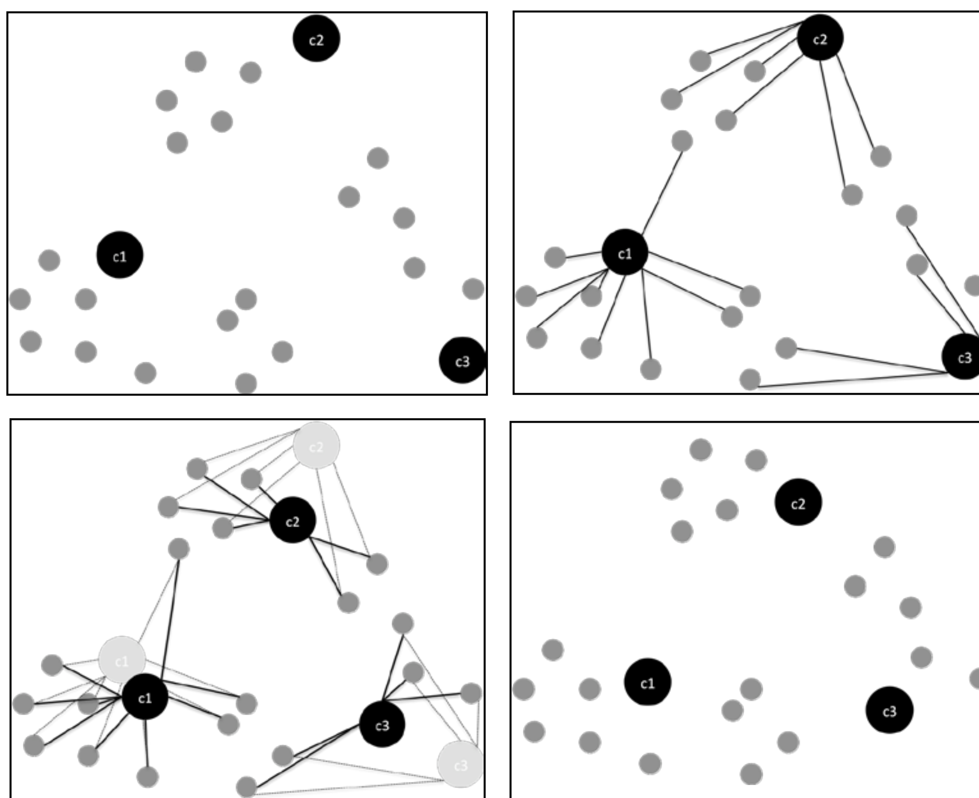


Figure 1 K-Means clustering in action. Starting with 3 random points as centroids (top-left), the Map stage (top-right) assigns each point to the cluster nearest to it. In the reduce stage (bottom-left), the associated points are averaged out to produce the new location of the centroid, leaving us with the final configuration (bottom-right). After each iteration, the final configuration is fed back in to the same loop till the centroids converge.

Running K-Means clustering

The K-Means clustering algorithm is run using either the `KMeansClusterer` or the `KMeansDriver` class. The former one does an in-memory clustering of the points while the latter is an entry point to launch K-Means as a Map/Reduce job. Both methods can be run like a regular Java program and can read and write data from the disk. They can also be executed on a Hadoop cluster reading and writing data to a distributed file system.

---

[2] http://en.wikipedia.org/wiki/Expectation-maximization_algorithm
[3] http://www.cc.gatech.edu/~dellaert/em-paper.pdf gives an easier explanation on EM Algorithm in terms of lower bound maximization.

For this example, we are going to use a random point generator function to create the points. It generates the points in the Vector format as a normal distribution around a given center. The points are scattered around in a natural manner. These points are going to be clustered using the in-memory K-Means clustering implementation in Mahout.

The `generateSamples` function in the listing 9.1 below takes a center say `(1,1)`, the standard deviation `(2)`, and creates a set of n (400) random points around the center, which behaves like a normal distribution. Similarly we will create two other sets with centers `(1,0)` and `(0,2)` and standard deviation `0.5` and `0.1` respectively. In listing 1, we run the `KMeansClusterer` using the following parameters:

- The input points are in the `List<Vector>` format
- The `DistanceMeasure` is `EuclideanDistanceMeasure`
- The threshold of convergence is `0.01`
- `The number of clusters k is 3`
- `The clusters were chosen using a RandomSeedGenerator`

Listing 1 In-memory clustering example using the K-Means algorithm

```
private static void generateSamples(List<Vector> vectors, int num,
  double mx, double my, double sd) {
  for (int i = 0; i < num; i++) {
    sampleData.add(new DenseVector(
      new double[] {
        UncommonDistributions.rNorm(mx, sd),
        UncommonDistributions.rNorm(my, sd)
      }
    ));
  }
}
public static void KMeansExample() {
  List<Vector> sampleData = new ArrayList<Vector>();

  generateSamples(sampleData, 400, 1, 1, 3);                  #1
  generateSamples(sampleData, 300, 1, 0, 0.5);
  generateSamples(sampleData, 300, 0, 2, 0.1);

  List<Vector> randomPoints = RandomSeedGenerator.chooseRandomPoints(
    points, k);
  List<Cluster> clusters = new ArrayList<Cluster>();

  int clusterId = 0;
  for (Vector v : randomPoints) {
    clusters.add(new Cluster(v, clusterId++));
  }

  List<List<Cluster>> finalClusters = KMeansClusterer.clusterPoints(
    points, clusters, new EuclideanDistanceMeasure(), 3, 0.01); #2
  for(Cluster cluster : finalClusters.get(finalClusters.size() - 1)) {
    System.out.println("Cluster id: " + cluster.getId() + " center: "
                     + cluster.getCenter().asFormatString());   #3
  }
}
```

#1 Generates 3 sets of points each with a different center and standard deviation
#2 Runs `KMeansClusterer` using the `CosineDistanceMeasure`
#3 Reads the center of the cluster and prints it

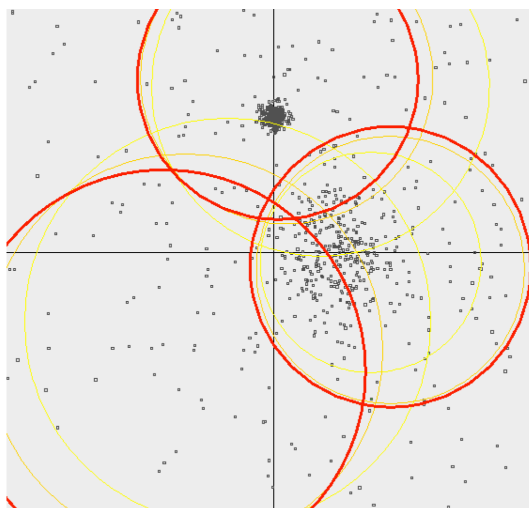


Figure 2 K-Means Clustering. We start with k as 3 and try to cluster 3 normal distributions we have generated. The thin lines denote the clusters estimated in previous iterations; here we can clearly see the clusters shifting.

The `DisplayKMeans` class kept in the "examples" folder of the Mahout code is a great tool to visualize the algorithm in a two-dimensional plane. It shows how the clusters shift their position after each iteration. It is also a great example of how clustering is done using `KMeansClusterer`. Just run the `DisplayKMeans` as a Java Swing application and view the output of the example given in figure 2.

Note that the K-Means in-memory clustering implementation works with a list of `Vector` objects. The amount of memory used by this program depends on the total size of all the vectors. The sizes of clusters are larger compared to the size of the vectors in the case of sparse vectors or the same size for dense vectors. As a rule of thumb, assume that the number of vectors that could be fit in memory equals the number of data points + k centers. If the data is huge, we cannot run this implementation.

This is where MapReduce shines. Using the MapReduce infrastructure, we can split this clustering algorithm to run on multiple machines, with each Mapper getting a subset of the points and nearest cluster computed in a streaming fashion.

Next, we'll focus on Fuzzy K-Means algorithm.

What's so fuzzy about Fuzzy K-Means?

As the name says, the Fuzzy K-Means algorithm does a fuzzy form of K-Means clustering. Instead of exclusive clustering in K-Means, Fuzzy K-Means tries to generate overlapping clusters from the dataset. In the academic community, it's also known by the name Fuzzy C-Means algorithm. We can think of it as an extension of K-Means. K-Means tries to find the hard clusters (a point belonging to one cluster), whereas Fuzzy K-Means discovers the soft clusters. In a soft cluster, any point can belong to more than one cluster with a certain affinity value towards each. This affinity is proportional to the distance of point to the centroid of the cluster. Like K-Means, Fuzzy K-Means works on those objects that can be represented in n-dimensional vector space and has a distance measure defined.

The algorithm is available in the `FuzzyKMeansClusterer` or `FuzzyKMeansDriver` class. The former is an in-memory and the latter a MapReduce implementation. We are going to use a random point generator function to create the points scattered in a two-dimensional plane.

In listing 2, we run the in-memory version using the `FuzzyKMeansClusterer` with the following parameters:

- The input `Vector` data in the `List<Vector>` format.
- The `DistanceMeasure` is `EuclideanDistanceMeasure`.
- The threshold of convergence is `0.01`.
- `The number of clusters k is 3.`

- The fuzziness parameter m is 3.

Listing 2 In-memory clustering example of Fuzzy K-Means clustering

```
public static void FuzzyKMeansExample() {
  List<Vector> sampleData = new ArrayList<Vector>();

  generateSamples(sampleData, 400, 1, 1, 3);                #1
  generateSamples(sampleData, 300, 1, 0, 0.5);
  generateSamples(sampleData, 300, 0, 2, 0.1);

  List<Vector> randomPoints = RandomSeedGenerator.chooseRandomPoints(
    points, k);
  List<SoftCluster> clusters = new ArrayList<SoftCluster>();

  int clusterId = 0;
  for (Vector v : randomPoints) {
    clusters.add(new SoftCluster(v, clusterId++));
  }

  List<List<SoftCluster>> finalClusters = FuzzyKMeansClusterer
     .clusterPoints(points, clusters, new EuclideanDistanceMeasure(),
         0.01, 3, 10); #2
  for(SoftCluster cluster : finalClusters.get(finalClusters.size() - 1)) {
    System.out.println("Fuzzy Cluster id: " + cluster.getId()
        + " center: " + cluster.getCenter().asFormatString());   #3
  }

}
```

#1 Generates 3 sets of points using different parameters
#2 Runs FuzzyKMeansClusterer
#3 Reads the center of the fuzzy-clusters and prints it.

The `DisplayFuzzyKMeans` class in the "examples" folder of the Mahout code is a good tool to visualize this algorithm on a 2-dimensional plane. `DisplayFuzzyKMeans` runs as a Java swing application and produces an output as given in the figure 3.

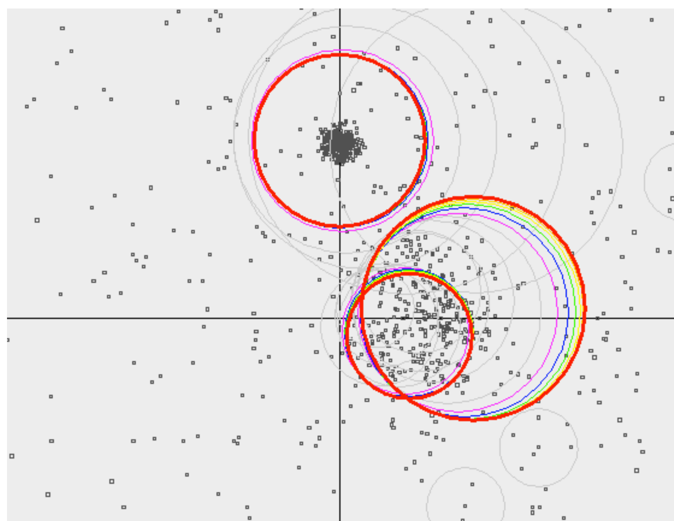


Figure 3 Fuzzy K-Means clustering. The clusters look like they are overlapping each other and the degree of overlap is decided by the fuzziness parameter.

MapReduce implementation of Fuzzy K-Means

Before running the MapReduce implementation, let's create a checklist for running Fuzzy K-Means clustering against the Reuters dataset like we did for K-Means. We have:

- The dataset in the Vector format.
- The RandomSeedGenerator to seed the initial k clusters.
- The distance measure is SquaredEuclideanDistanceMeasure.
- A large value of convergenceThreshold –d 1.0 because we are using the squared value of the distance measure.
- The maxIterations is the default value of –x 10.
- The coefficient of normalization or the fuzziness factor, a value greater than 1.0, –m.

To run the Fuzzy K-Means clustering over the input data, use the Mahout launcher using the fkmeans program name as follows:

```
$bin/mahout fkmeans
-i reuters-vectors -c reuters-fkmeans-centroids
-o reuters-fkmeans-clusters -d 1.0 -k 21 -m 2 -w
-dm org.apache.mahout.common.distance.SquaredEuclideanDistanceMeasure
```

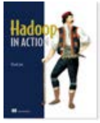
Like K-Means, FuzzyKMeansDriver will automatically run the RandomSeedGenerator if the number of clusters (k) flag is set. Once the random centroids are generated, Fuzzy K-Means clustering will use it as the input set of k centroids. The algorithm runs multiple iterations over the dataset until the centroids converges, each time creating the output in the folder cluster-*. Finally, it runs another job, which generates the probabilities of a point belonging to a particular cluster based on the distance measure and the fuzziness parameter (m).

It's a good idea to inspect the clusters using the ClusterDumper tool. ClusterDumper shows the top words of the cluster as per the centroid. To get the actual mapping of points to the clusters, we need to read the SequenceFiles in the points/ folder. Each entry in the sequence file has a key, which is the identifier of the vector, and a value, which is the list of cluster centroids with an associated numerical value, which tells us how well the point belongs to that particular centroid.

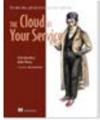
Summary

The Mahout implementation of the popular K-Means algorithm works great for small and big datasets. Fuzzy K-Means clustering gives more information related to partial membership of a document into various clusters. Fuzzy K-Means has better convergence properties than just K-Means. We tuned our clustering module to use Fuzzy K-Means to help identify this soft membership information.

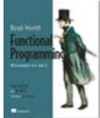
Here are some other Manning titles you might be interested in:



Hadoop in Action
Chuck Lam



The Cloud at Your Service
Jothy Rosenberg and Arthur Mateos



Real-World Functional Programming
Tomas Petricek with Jon Skeet

Last updated: May 9, 2011