E6893 Big Data Analytics Lecture 6:

Big Data Analytics Algorithms -- III

Ching-Yung Lin, Ph.D.

Adjunct Professor, Dept. of Electrical Engineering and Computer Science

Mgr., Dept. of Network Science and Big Data Analytics, IBM Watson Research Center

October 9th, 2014

Course Structure

Class Data	Number	Topics Covered
09/04/14	1	Introduction to Big Data Analytics
09/11/14	2	Big Data Analytics Platforms
09/18/14	3	Big Data Storage and Processing
09/25/14	4	Big Data Analytics Algorithms -- I
10/02/14	5	Big Data Analytics Algorithms -- II (recommendation)
10/09/14	6	Big Data Analytics Algorithms — III (clustering)
10/16/14	7	Big Data Analytics Algorithms — IV (classification)
10/23/14	8	Linked Big Data – Graph Computing
10/30/14	9	Big Data Visualization
11/06/14	10	Mobile Data Collection, Analysis, and Interface
11/13/14	11	Hardware, Processors, and Cluster Platforms
11/20/14	12	Big Data Next Challenges – IoT, Cognition, and Beyond
11/27/14		*Thanksgiving Holiday*
12/04/14	13	*Final Projects Discussion (Optional)*
12/11/14 & 12/12/14	14-15	**Two-Day Big Data Analytics Workshop – Final Project Presentations**

Review — Key Components of Mahout

Collaborative Filtering

- User-Based Collaborative Filtering - single machine
- Item-Based Collaborative Filtering - **single machine** / MapReduce
- Matrix Factorization with Alternating Least Squares - **single machine** / MapReduce
- Matrix Factorization with Alternating Least Squares on Implicit Feedback- **single machine** / MapReduce
- Weighted Matrix Factorization, SVD++, Parallel SGD - **single machine**

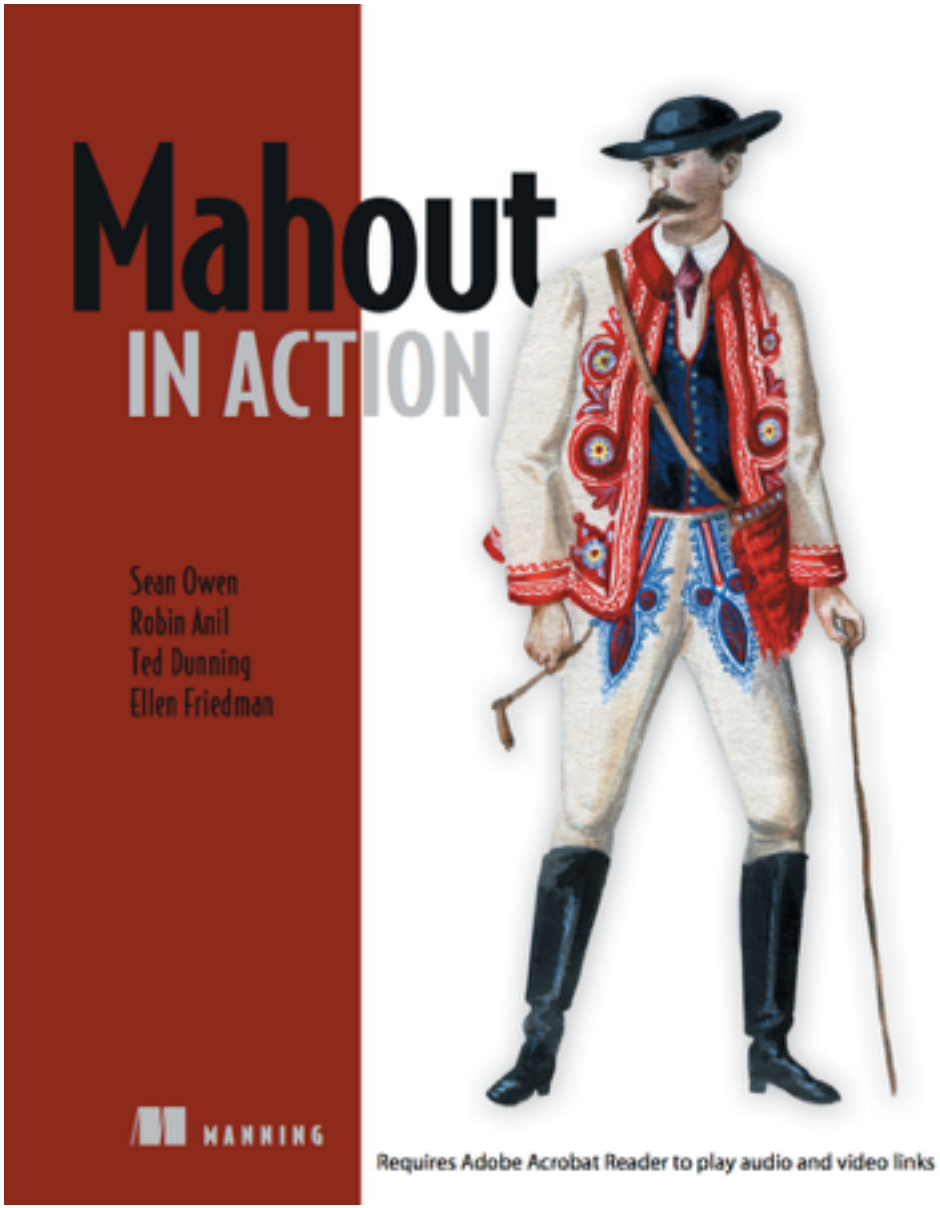
Classification

- Logistic Regression - trained via SGD - single machine
- Naive Bayes/ Complementary Naive Bayes - MapReduce
- Random Forest - **MapReduce**
- Hidden Markov Models - single machine
- Multilayer Perceptron - **single machine**

Clustering

- Canopy Clustering - **single machine** / MapReduce (deprecated, will be removed once Streaming k-Means is stable enough)
- k-Means Clustering - single machine / MapReduce
- Fuzzy k-Means - single machine / MapReduce
- Streaming k-Means - single machine / MapReduce
- Spectral Clustering - MapReduce

Mahout reference book



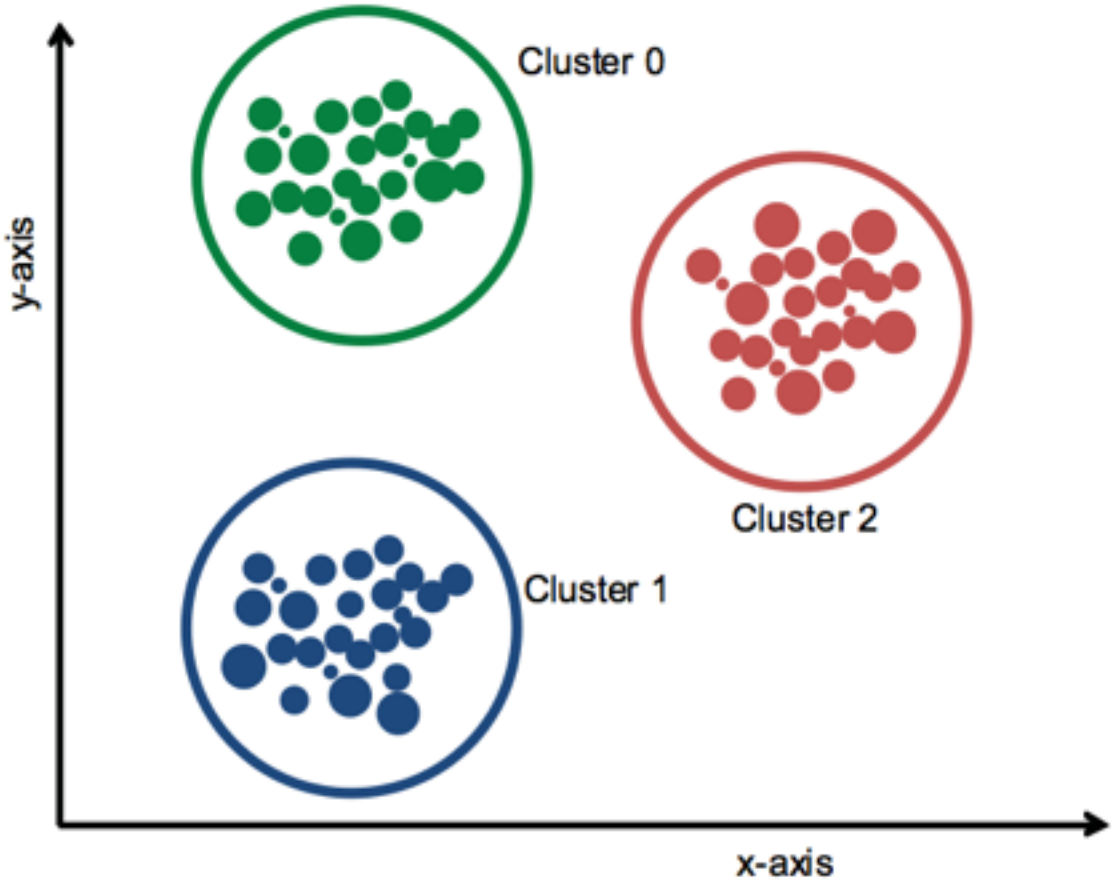
Requires Adobe Acrobat Reader to play audio and video links

Clustering

Clustering a collection involves three things:

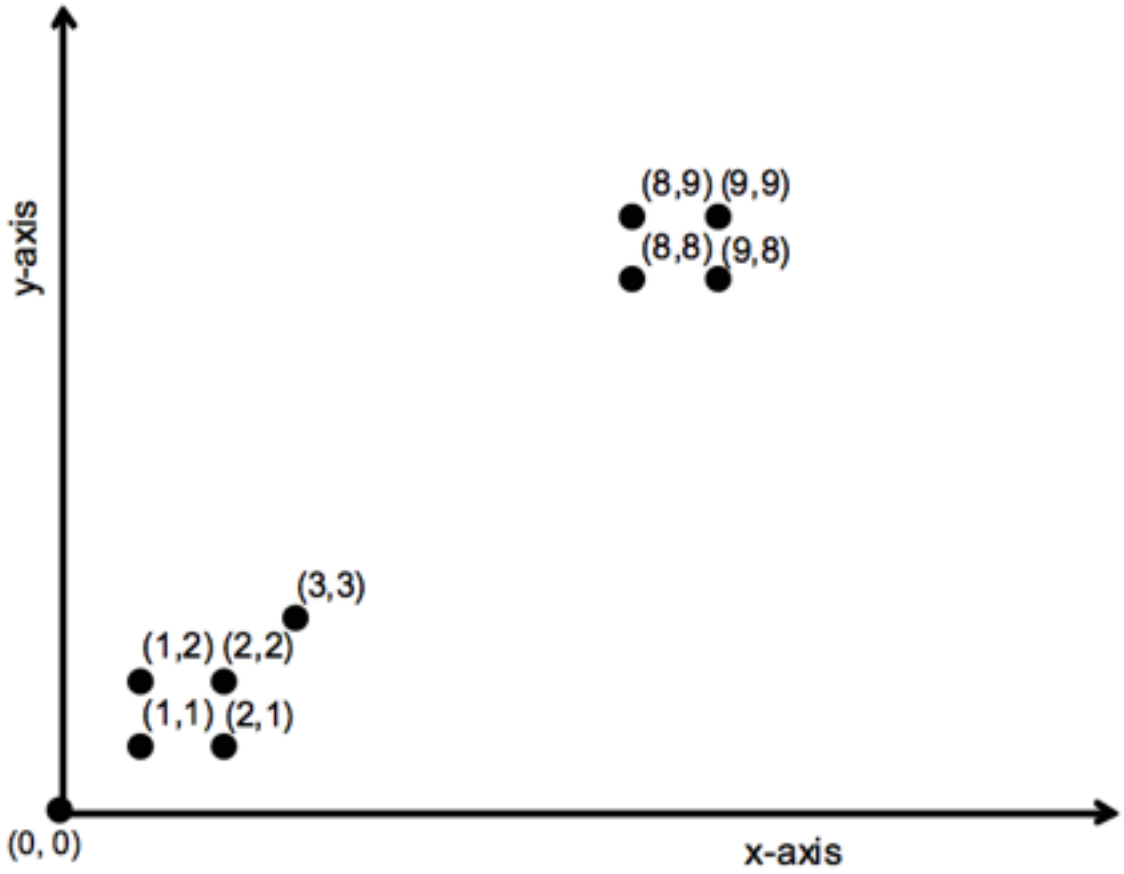
- *An algorithm*—This is the method used to group the books together.
- *A notion of both similarity and dissimilarity*—In the previous discussion, we relied on your assessment of which books belonged in an existing stack and which should start a new one.
- *A stopping condition*—In the library example, this might be the point beyond which books can't be stacked anymore, or when the stacks are already quite dissimilar.

Clustering — on feature plane

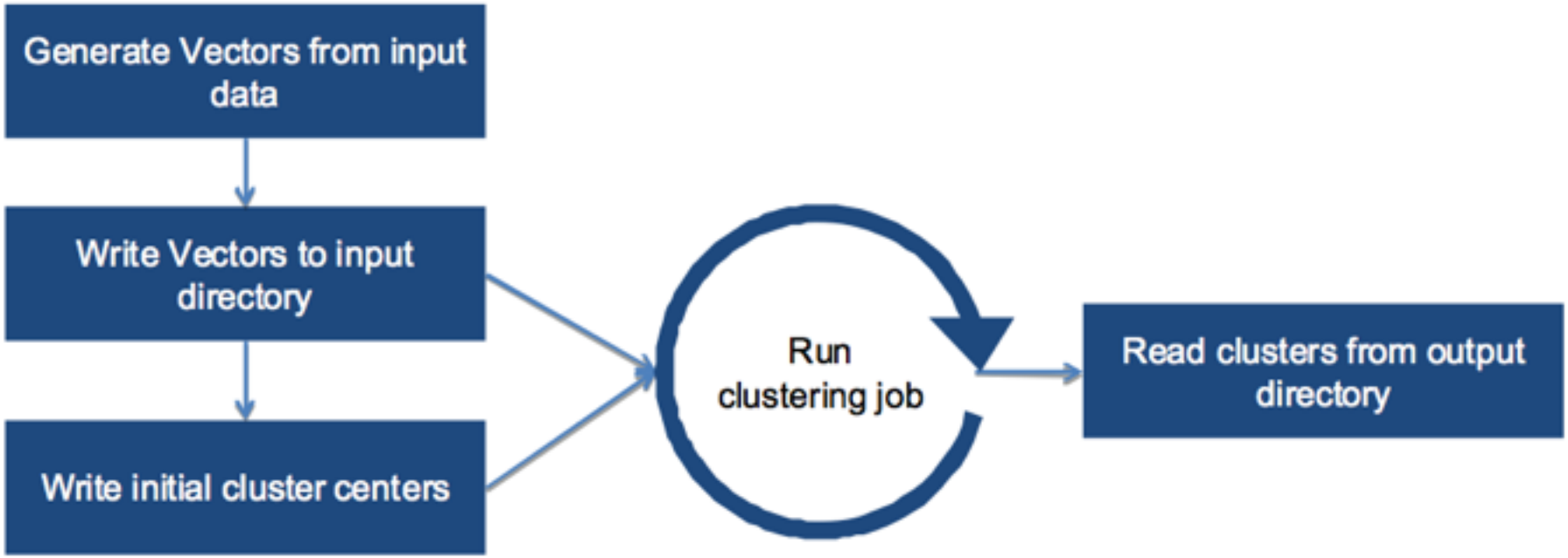


Clustering example

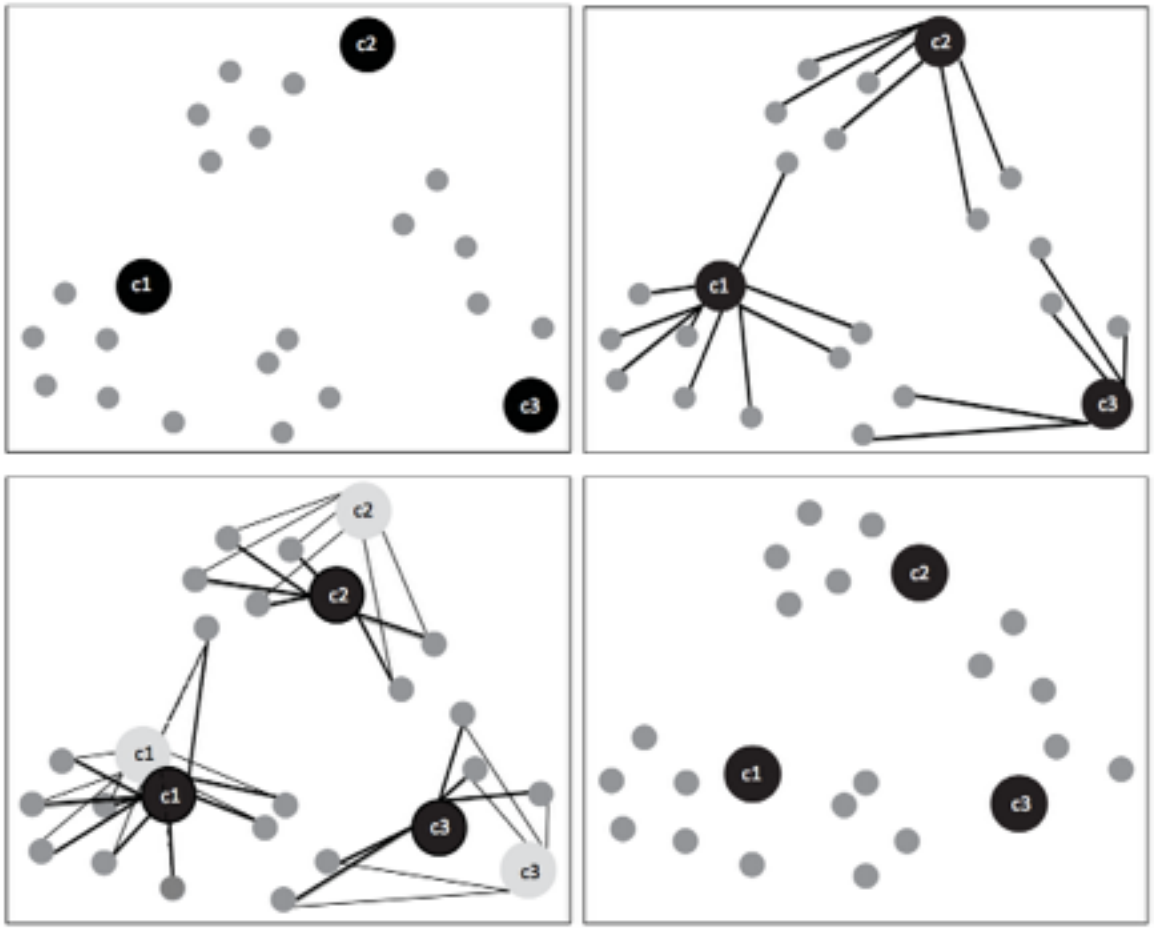
(1,1)
(2,1)
(1,2)
(2,2)
(3,3)
(8,8)
(8,9)
(9,8)
(9,9)



Steps on clustering

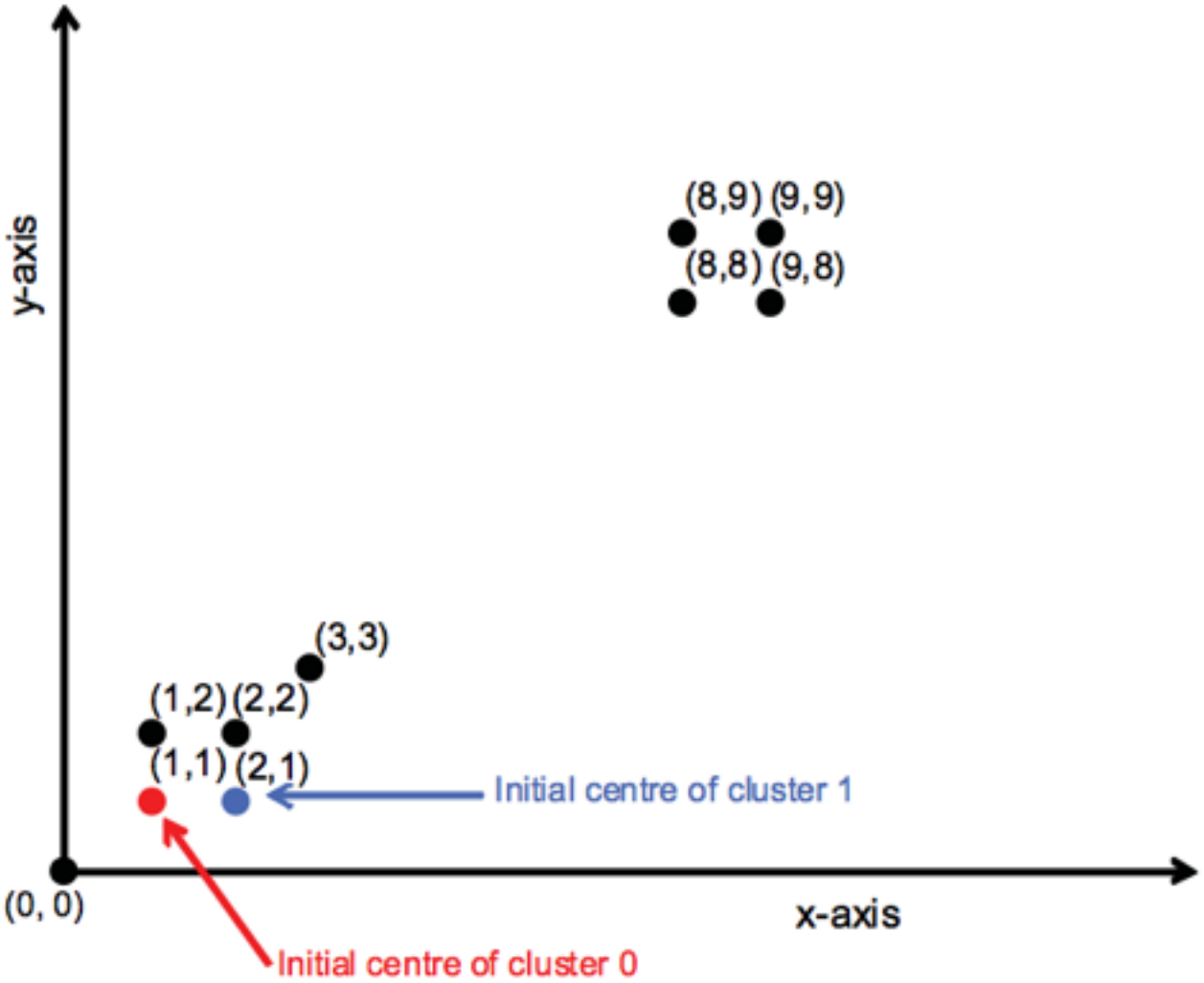


K-mean clustering



K-means clustering in action. Starting with three 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 you with the final configuration (bottom right). After each iteration, the final configuration is fed back into the same loop until the centroids come to rest at their final positions.

Making initial cluster centers



Parameters to Mahout k-mean clustering algorithm

- The `SequenceFile` containing the input vectors.
- The `SequenceFile` containing the initial cluster centers. In this case, we seed two clusters, so there are two centers.
- The similarity measure to be used. We use `EuclideanDistanceMeasure` as the measure of similarity here, and we explore other similarity measures later in this chapter.
- The `convergenceThreshold`. If in a particular iteration the centers of the clusters don't change beyond this threshold, no further iterations are done.
- The number of iterations to be done.
- The `Vector` implementation used in the input files.

HelloWorld clustering scenario

```
public static final double[][] points = { {1, 1}, {2, 1}, {1, 2},
                                          {2, 2}, {3, 3}, {8, 8},
                                          {9, 8}, {8, 9}, {9, 9}};

public static void writePointsToFile(List<Vector> points,
                                     String fileName,
                                     FileSystem fs,
                                     Configuration conf) throws IOException {
  Path path = new Path(fileName);
  SequenceFile.Writer writer = new SequenceFile.Writer(fs, conf,
      path, LongWritable.class, VectorWritable.class);
  long recNum = 0;
  VectorWritable vec = new VectorWritable();
  for (Vector point : points) {
    vec.set(point);
    writer.append(new LongWritable(recNum++), vec);
  }
  writer.close();
}

public static List<Vector> getPoints(double[][] raw) {
  List<Vector> points = new ArrayList<Vector>();
  for (int i = 0; i < raw.length; i++) {
    double[] fr = raw[i];
    Vector vec = new RandomAccessSparseVector(fr.length);
    vec.assign(fr);
    points.add(vec);
  }
  return points;
}
```

HelloWorld Clustering scenario - II

```
public static void main(String args[]) throws Exception {

  int k = 2;

  List<Vector> vectors = getPoints(points);

  File testData = new File("testdata");
  if (!testData.exists()) {
    testData.mkdir();
  }
  testData = new File("testdata/points");
  if (!testData.exists()) {
    testData.mkdir();
  }

  Configuration conf = new Configuration();
  FileSystem fs = FileSystem.get(conf);
  writePointsToFile(vectors,
      "testdata/points/file1", fs, conf);

  Path path = new Path("testdata/clusters/part-00000");
  SequenceFile.Writer writer
      = new SequenceFile.Writer(
          fs, conf,      path, Text.class, Cluster.class);

  for (int i = 0; i < k; i++) {
    Vector vec = vectors.get(i);
    Cluster cluster = new Cluster(
        vec, i, new EuclideanDistanceMeasure());
    writer.append(new Text(cluster.getIdentifier()), cluster);
  }
  writer.close();
```

Specify number of clusters to be formed

Create input directories for data

Write initial centers

HelloWorld Clustering scenario - III

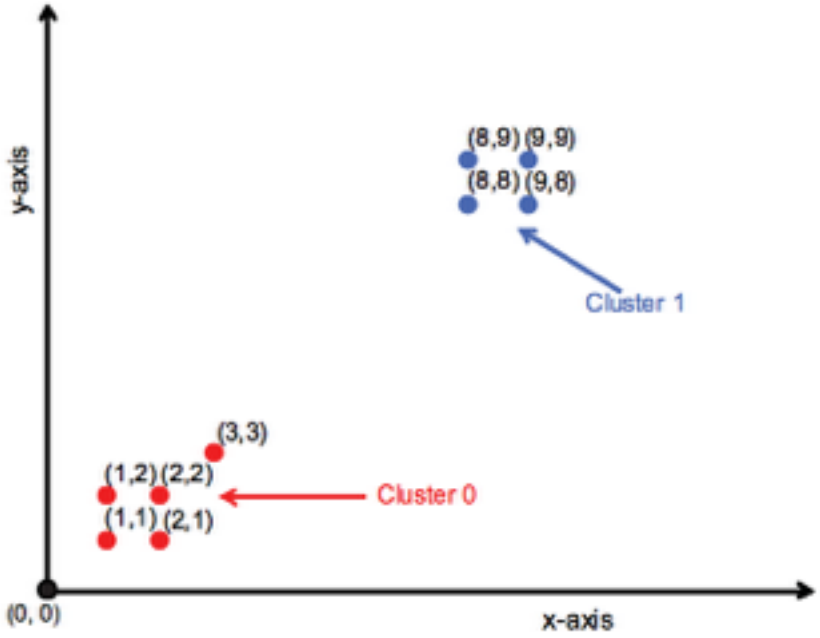
```
  KMeansDriver.run(conf, new Path("testdata/points"),
    new Path("testdata/clusters"),
    new Path("output"), new EuclideanDistanceMeasure(),
        0.001, 10, true, false);

  SequenceFile.Reader reader
      = new SequenceFile.Reader(fs,
          new Path("output/" + Cluster.CLUSTERED_POINTS_DIR
              + "/part-m-00000"), conf);

  IntWritable key = new IntWritable();
  WeightedVectorWritable value = new WeightedVectorWritable();
  while (reader.next(key, value)) {
    System.out.println(
        value.toString() + " belongs to cluster "
            + key.toString());
  }
  reader.close();
}
```

Run k-means algorithm

Read output, print vector, cluster ID

HelloWorld clustering scenario result

```
1.0: [1.000, 1.000] belongs to cluster 0
1.0: [2.000, 1.000] belongs to cluster 0
1.0: [1.000, 2.000] belongs to cluster 0
1.0: [2.000, 2.000] belongs to cluster 0
1.0: [3.000, 3.000] belongs to cluster 0
1.0: [8.000, 8.000] belongs to cluster 1
1.0: [9.000, 8.000] belongs to cluster 1
1.0: [8.000, 9.000] belongs to cluster 1
1.0: [9.000, 9.000] belongs to cluster 1
```



Testing difference distance measures

Euclidean distance measure

$$d = \sqrt{(a_1 - b_1)^2 + (a_2 - b_2)^2 + \ldots + (a_n - b_n)^2}$$

Squared Euclidean distance measure

$$d = (a_1 - b_1)^2 + (a_2 - b_2)^2 + \ldots + (a_n - b_n)^2$$

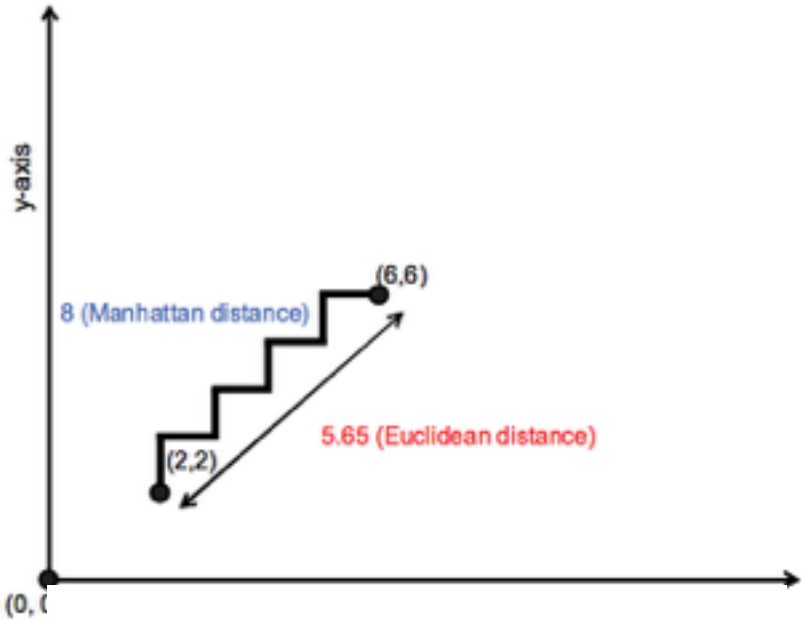
Manhattan distance measure

$$d = |a_1 - b_1| + |a_2 - b_2| + \ldots + |a_n - b_n|$$

Manhattan and Cosine distances

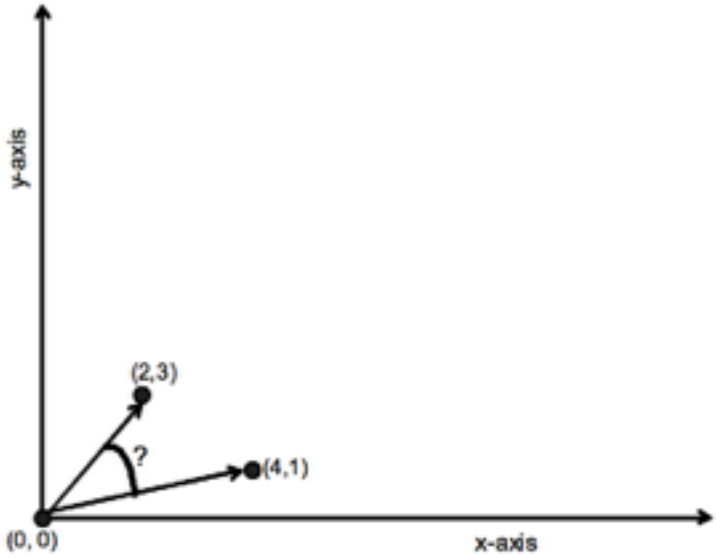
Manhattan distance measure

$$d = |a_1 - b_1| + |a_2 - b_2| + \ldots + |a_n - b_n|$$



Cosine distance measure

$$d = 1 - \frac{(a_1 b_1 + a_2 b_2 + \ldots + a_n b_n)}{(\sqrt{(a_1^2 + a_2^2 + \ldots + a_n^2)}\sqrt{(b_1^2 + b_2^2 + \ldots + b_n^2)})}$$



Tanimoto distance and weighted distance

Tanimoto distance measure

$$d = 1 - \frac{(a_1 b_1 + a_2 b_2 + \ldots + a_n b_n)}{\sqrt{(a_1^2 + a_2^2 + \ldots + a_n^2)} + \sqrt{(b_1^2 + b_2^2 + \ldots + b_n^2)} - (a_1 b_1 + a_2 b_2 + \ldots + a_n b_n)}$$

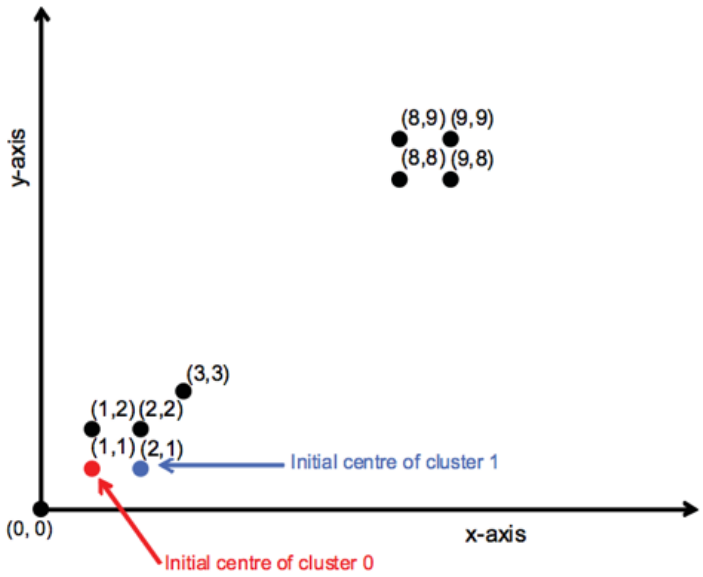
Weighted distance measure

Mahout also provides a `WeightedDistanceMeasure` class, and implementations of Euclidean and Manhattan distance measures that use it. A weighted distance measure is an advanced feature in Mahout that allows you to give weights to different dimensions in order to either increase or decrease the effect of a dimension

Results comparison

Distance measure	Number of iterations	Vectors[a] in cluster 0	Vectors in cluster 1
EuclideanDistanceMeasure	3	0, 1, 2, 3, 4	5, 6, 7, 8
SquaredEuclideanDistanceMeasure	5	0, 1, 2, 3, 4	5, 6, 7, 8
ManhattanDistanceMeasure	3	0, 1, 2, 3, 4	5, 6, 7, 8
CosineDistanceMeasure	1	1	0, 2, 3, 4, 5, 6, 7, 8
TanimotoDistanceMeasure	3	0, 1, 2, 3, 4	5, 6, 7, 8

(1,1)
(2,1)
(1,2)
(2,2)
(3,3)
(8,8)
(8,9)
(9,8)
(9,9)



Data preparation in Mahout — vectors

In Mahout, vectors are implemented as three different classes, each of which is optimized for different scenarios: `DenseVector`, `RandomAccessSparseVector`, and `SequentialAccessSparseVector`.

- `DenseVector` can be thought of as an array of doubles, whose size is the number of features in the data. Because all the entries in the array are preallocated regardless of whether the value is 0 or not, we call it *dense*.
- `RandomAccessSparseVector` is implemented as a `HashMap` between an `integer` and a `double`, where only nonzero valued features are allocated. Hence, they're called as `SparseVectors`.
- `SequentialAccessSparseVector` is implemented as two parallel arrays, one of integers and the other of doubles. Only nonzero valued entries are kept in it. Unlike the `RandomAccessSparseVector`, which is optimized for random access, this one is optimized for linear reading.

vectorization example

0: weight
1: color
2: size

```
[0 => 100 gram, 1 => red, 2 => small]
```

Apple	Weight (kg) (0)	Color (1)	Size (2)	Vector
Small, round, green	0.11	510	1	[0.11, 510, 1]
Large, oval, red	0.23	650	3	[0.23, 650, 3]
Small, elongated, red	0.09	630	1	[0.09, 630, 1]
Large, round, yellow	0.25	590	3	[0.25, 590, 3]
Medium, oval, green	0.18	520	2	[0.18, 520, 2]

Mahout codes to create vectors of the apple example

```
public static void main(String args[]) throws Exception {
  List<NamedVector> apples = new ArrayList<NamedVector>();

  NamedVector apple;
  apple = new NamedVector(
      new DenseVector(new double[] {0.11, 510, 1}),
      "Small round green apple");
  apples.add(apple);
  apple = new NamedVector(
    new DenseVector(new double[] {0.23, 650, 3}),
      "Large oval red apple");
  apples.add(apple);
  apple = new NamedVector(
    new DenseVector(new double[] {0.09, 630, 1}),
      "Small elongated red apple");
  apples.add(apple);
  apple = new NamedVector(
    new DenseVector(new double[] {0.25, 590, 3}),
      "Large round yellow apple");
  apples.add(apple);
  apple = new NamedVector(
    new DenseVector(new double[] {0.18, 520, 2}),
      "Medium oval green apple");
```

Associates a name with the vector

Mahout codes to create vectors of the apple example — II

```
Configuration conf = new Configuration();
FileSystem fs = FileSystem.get(conf);

Path path = new Path("appledata/apples");
SequenceFile.Writer writer = new SequenceFile.Writer(fs, conf,
    path, Text.class, VectorWritable.class);
VectorWritable vec = new VectorWritable();
for (NamedVector vector : apples) {
  vec.set(vector);
  writer.append(new Text(vector.getName()), vec);
}
writer.close();

SequenceFile.Reader reader = new SequenceFile.Reader(fs,
    new Path("appledata/apples"), conf);

Text key = new Text();
VectorWritable value = new VectorWritable();
while (reader.next(key, value)) {

  System.out.println(key.toString() + " "
      + value.get().asFormatString());
}
reader.close();
}
```

Serializes vector data

Deserializes vector data

Vectorization of text

Vector Space Model: Term Frequency (TF)

For example, if the word *horse* is assigned to the 39,905th index of the vector, the word *horse* will correspond to the 39,905th dimension of document vectors. A document's vectorized form merely consists, then, of the number of times each word occurs in the document, and that value is stored in the vector along that word's dimension. The dimension of these document vectors can be very large.

Stop Words: *a, an, the, who, what, are, is, was,* and so on.

Stemming:

A stemmer for English, for example, should identify the string "cats" (and possibly "catlike", "catty" etc.) as based on the root "cat", and "stemmer", "stemming", "stemmed" as based on "stem". A stemming algorithm reduces the words "fishing", "fished", and "fisher" to the root word, "fish". On the other hand, "argue", "argued", "argues", "arguing", and "argus" reduce to the stem "argu" (illustrating the case where the stem is not itself a word or root) but "argument" and "arguments" reduce to the stem "argument".

Most Popular Stemming algorithms

Columbia University

Lookup algorithms

A simple stemmer looks up the inflected form in a lookup table. The advantages of this approach is that it is simple, fast, and easily handles exceptions. The disadvantages are that all inflected forms must be explicitly listed in the table: new or unfamiliar words are not handled, even if they are perfectly regular (e.g. iPads ~ iPad), and the table may be large. For languages with simple morphology, like English, table sizes are modest, but

Suffix-stripping algorithms

Suffix stripping algorithms do not rely on a lookup table that consists of inflected forms and root form relations. Instead, a typically smaller list of "rules" is stored which provides a path for the algorithm, given an input word form, to find its root form. Some examples of the rules include:

- if the word ends in 'ed', remove the 'ed'
- if the word ends in 'ing', remove the 'ing'
- if the word ends in 'ly', remove the 'ly'

Term Frequency — Inverse Document Frequency (TF-IDF)

The value of word is reduced more if it is used frequently across all the documents in the dataset.

To calculate the inverse document frequency, the document frequency (DF) for each word is first calculated. Document frequency is the number of documents the word occurs in. The number of times a word occurs in a document isn't counted in document frequency. Then, the inverse document frequency or IDF_i for a word, w_i, is

$$IDF_i = \frac{1}{DF_i}$$

$$W_i = TF_i \cdot IDF_i = TF_i \cdot \frac{N}{DF_i} \quad \text{or} \quad W_i = TF_i \cdot \log\frac{N}{DF_i}$$

n-gram

It was the best of time. it was the worst of times. ==> bigram

It was
was the
the best
best of
of times
times it
it was
was the
the worst
worst of
of times

Mahout provides a log-likelihood test to reduce the dimensions of n-grams

Examples — using a news corpus

Reuters-21578 dataset: 22 files, each one has 1000 documents except the last one.

http://www.daviddlewis.com/resources/testcollections/ reuters21578/

Extraction code:

```
mvn -e -q exec:java
-Dexec.mainClass="org.apache.lucene.benchmark.utils.ExtractReuters"
-Dexec.args="reuters/ reuters-extracted/"
```

Using the extracted folder, run the `SequenceFileFromDirectory` class. You can use the launcher script from the Mahout root directory to do the same:

```
bin/mahout seqdirectory -c UTF-8
-i examples/reuters-extracted/ -o reuters-seqfiles
```

This will write the Reuters articles in the `SequenceFile` format. Now the only step left is to convert this data to vectors. To do that, run the `SparseVectorsFromSequence-Files` class using the Mahout launcher script:

```
bin/mahout seq2sparse -i reuters-seqfiles/ -o reuters-vectors -ow
```

Mahout dictionary-based vectorizer

Option	Flag	Description	Default value
Overwrite (`bool`)	`-ow`	If set, the output folder is overwritten. If not set, the output folder is created if the folder doesn't exist. If the output folder does exist, the job fails and an error is thrown. Default is unset.	N/A
Lucene analyzer name (`String`)	`-a`	The class name of the analyzer to use.	`org.apache.lucene.analysis.standard.StandardAnalyzer`
Chunk size (`int`)	`-chunk`	The chunk size in MB. For large document collections (sizes in GBs and TBs), you won't be able to load the entire dictionary into memory during vectorization, so you can split the dictionary into chunks of the specified size and perform the vectorization in multiple stages. It's recommended you keep this size to 80 percent of the Java heap size of the Hadoop child nodes to prevent the vectorizer from hitting the heap limit.	`100`
Weighting (`String`)	`-wt`	The weighting scheme to use: `tf` for term-frequency based weighting and `tfidf` for TF-IDF based weighting.	`tfidf`
Minimum support (`int`)	`-s`	The minimum frequency of the term in the entire collection to be considered as a part of the dictionary file. Terms with lesser frequency are ignored.	2

Mahout dictionary-based vectorizer — II

Option	Flag	Description	Default value
Minimum document frequency (int)	-md	The minimum number of documents the term should occur in to be considered a part of the dictionary file. Any term with lesser frequency is ignored.	1
Max document frequency percentage (int)	-x	The maximum number of documents the term should occur in to be considered a part of the dictionary file. This is a mechanism to prune out high frequency terms (stop-words). Any word that occurs in more than the specified percentage of documents is ignored.	99
N-gram size (int)	-ng	The maximum size of n-grams to be selected from the collection of documents.	1

Mahout dictionary-based vectorizer — III

Option	Flag	Description	Default value
Minimum log-likelihood ratio (LLR) (`float`)	`-ml`	This flag works only when *n*-gram size is greater than 1. Very significant *n*-grams have large scores, such as 1000; less significant ones have lower scores. Although there's no specific method for choosing this value, the rule of thumb is that *n*-grams with a LLR value less than 1.0 are irrelevant.	`1.0`
Normalization (`float`)	`-n`	The normalization value to use in the L_p space. A detailed explanation of normalization is given in section 8.4. The default scheme is to not normalize the weights.	0
Number of reducers (`int`)	`-nr`	The number of reducer tasks to execute in parallel. This flag is useful when running a dictionary vectorizer on a Hadoop cluster. Setting this to the maximum number of nodes in the cluster gives maximum performance. Setting this value higher than the number of cluster nodes leads to a slight decrease in performance. For more details, read the Hadoop documentation on setting the optimum number of reducers.	1
Create sequential access sparse vectors (`bool`)	`-seq`	If set, the output vectors are created as `SequentialAccessSparseVectors`. By default the dictionary vectorizer generates `RandomAccessSparseVectors`. The former gives higher performance on certain algorithms like k-means and SVD due to the sequential nature of vector operations. By default the flag is unset.	N/A

Outputs & Steps

```
$ ls reuters-vectors/
df-count/
dictionary.file-0
frequency.file-0
tfidf-vectors/
tf-vectors/
tokenized-documents/

wordcount/
```

1. Tokenization using Lucene StandardAnalyzer
2. n-gram generation step
3. converts the tokenized documents into vectors using TF
4. count DF and then create TF-IDF

A practical setting of flags

- `-a`—Use `org.apache.lucene.analysis.WhitespaceAnalyzer` to tokenize words based on the whitespace characters between them.
- `-chunk`—Use a chunk size of 200 MB. This value won't produce any effect on the Reuters data, because the dictionary sizes are usually in the 1 MB range.
- `-wt`—Use the `tfidf` weighting method.
- `-s`—Use a minimum support value of `5`.
- `-md`—Use a minimum document frequency value of `3`.
- `-x`—Use a maximum document frequency percentage of 90 percent to aggressively prune away high-frequency words.
- `-ng`—Use an *n*-gram size of `2` to generate both unigrams and bigrams.
- `-ml`—Use a minimum log-likelihood ratio (LLR) value of `50` to keep only very significant bigrams.
- `-seq`—Set the `SequentialAccessSparseVectors` flag.

Run the vectorizer using the preceding options in the Mahout launcher script:

```
bin/mahout seq2sparse -i reuters-seqfiles/ -o reuters-vectors-bigram -ow
-a org.apache.lucene.analysis.WhitespaceAnalyzer
-chunk 200 -wt tfidf -s 5 -md 3 -x 90 -ng 2 -ml 50 -seq
```

normalization

Some documents may pop up showing they are similar to all the other documents because it is large. ==> Normalization can help.

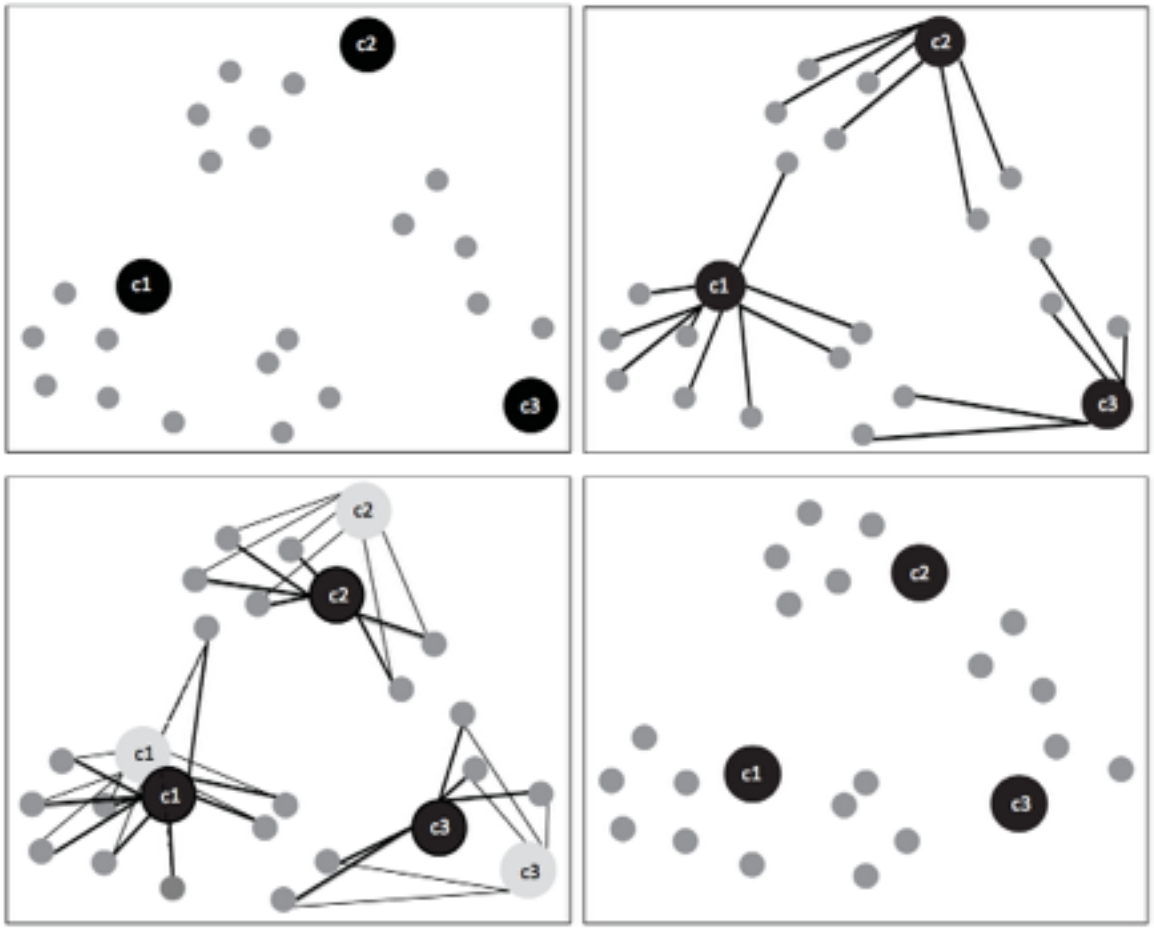
In Mahout, normalization uses what is known in statistics as a p-norm. For example, the p-norm of a 3-dimensional vector, [x, y, z], is

$$\frac{x}{(|x|^p + |y|^p + |z|^p)^{1/p}}, \frac{y}{(|x|^p + |y|^p + |z|^p)^{1/p}}, \frac{z}{(|x|^p + |y|^p + |z|^p)^{1/p}}$$

Clustering methods provided by Mahout

- K-means clustering
- Centroid generation using canopy clustering
- Fuzzy k-means clustering and Dirichlet clustering
- Topic modeling using latent Dirichlet allocation as a variant of clustering

K-mean clustering



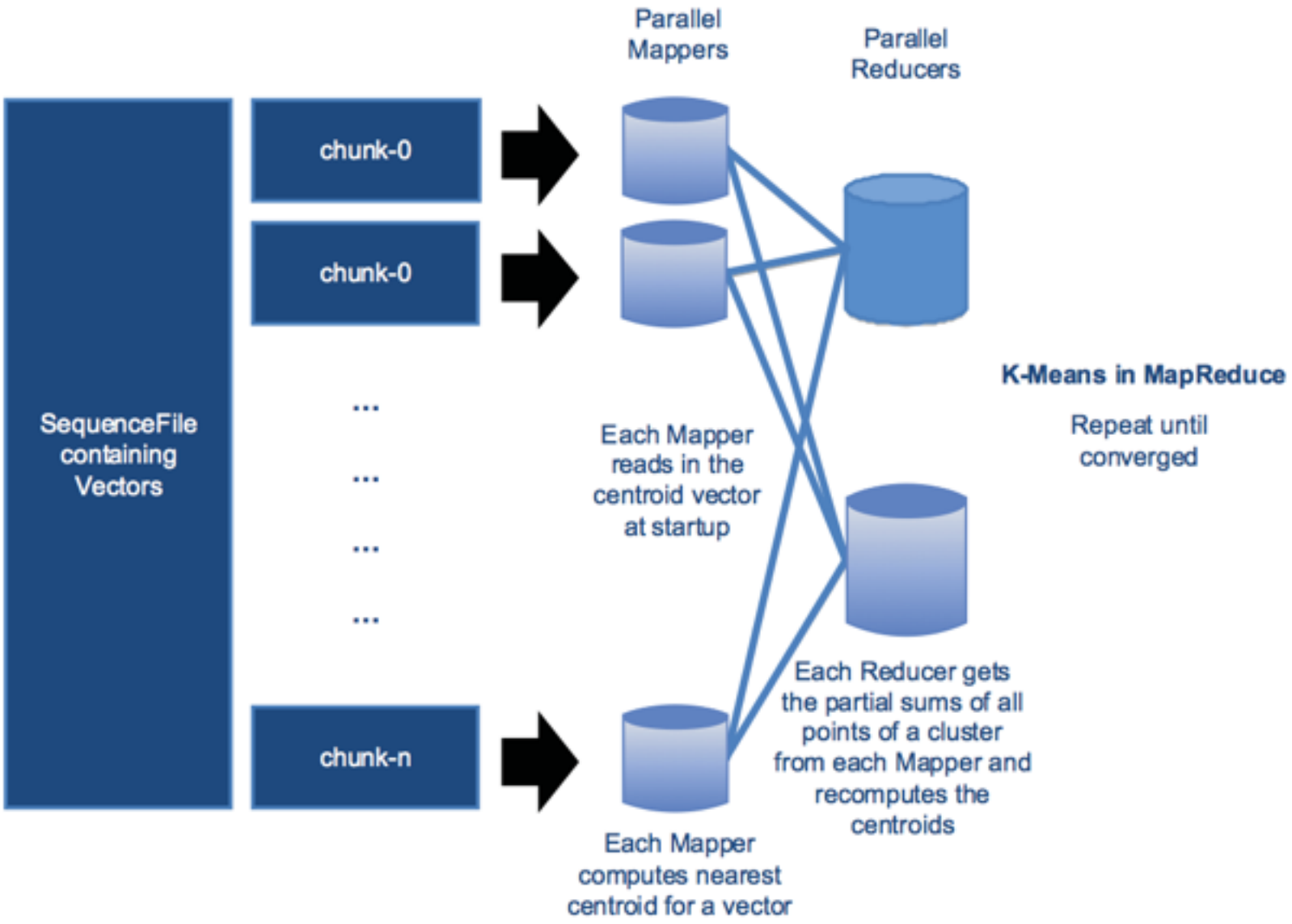
K-means clustering in action. Starting with three random points as centroids (top left), the map stage (top right) assigns each point to the cluster nearest to it. In the reduce stage (bottom left), the associated points are averaged out to produce the new location of the centroid, leaving you with the final configuration (bottom right). After each iteration, the final configuration is fed back into the same loop until the centroids come to rest at their final positions.

Hadoop k-mean clustering jobs

In Mahout, the MapReduce version of the k-means algorithm is instantiated using the `KMeansDriver` class. The class has just a single entry point—the `runJob` method.

- The Hadoop configuration.
- The `SequenceFile` containing the input `Vectors`.
- The `SequenceFile` containing the initial `Cluster` centers.
- The similarity measure to be used. We'll use `EuclideanDistanceMeasure` as the measure of similarity and experiment with the others later.
- The `convergenceThreshold`. If in an iteration, the centroids don't move more than this distance, no further iterations are done and clustering stops.
- The number of iterations to be done. This is a hard limit; the clustering stops if this threshold is reached.

K-mean clustering running as MapReduce job



Homework #2

Recommendation:

1. Choose any two datasets from Yahoo Labs Ratings and Classification Data.
2. Try various recommendation algorithms provided by Mahout

Clustering:

Using datasets from:

1. Online news (e.g., New York Times article in September 2014)
2. Wikipedia articles
3. (optional) gather data from Twitter API, try clustering

Do clustering —> finding related documents

Hadoop k-mean clustering code

```
KmeansDriver.runJob(hadoopConf,
      inputVectorFilesDirPath, clusterCenterFilesDirPath,
      outputDir, new EuclideanDistanceMeasure(),
      convergenceThreshold, numIterations, true, false);
```

Mahout reads and writes data using the Hadoop `FileSystem` class. This provides seamless access to both the local filesystem (via java.io) and distributed filesystems like HDFS and S3FS (using internal Hadoop classes). This way, the same code that works on the local system will also work on the Hadoop filesystem on the cluster, provided the paths to the Hadoop configuration files are correctly set in the environment variables. In Mahout, the bin/mahout shell script finds the Hadoop configuration files automatically from the `$HADOOP_CONF` environment variable.

```
$ bin/mahout kmeans -i reuters-vectors/tfidf-vectors/ \
-c reuters-initial-clusters  \
-o reuters-kmeans-clusters \
-dm org.apache.mahout.common.distance.SquaredEuclideanDistanceMeasure \
-cd 1.0 -k 20 -x 20 -cl
```

The output

The directory listing of the output folder looks something like this:

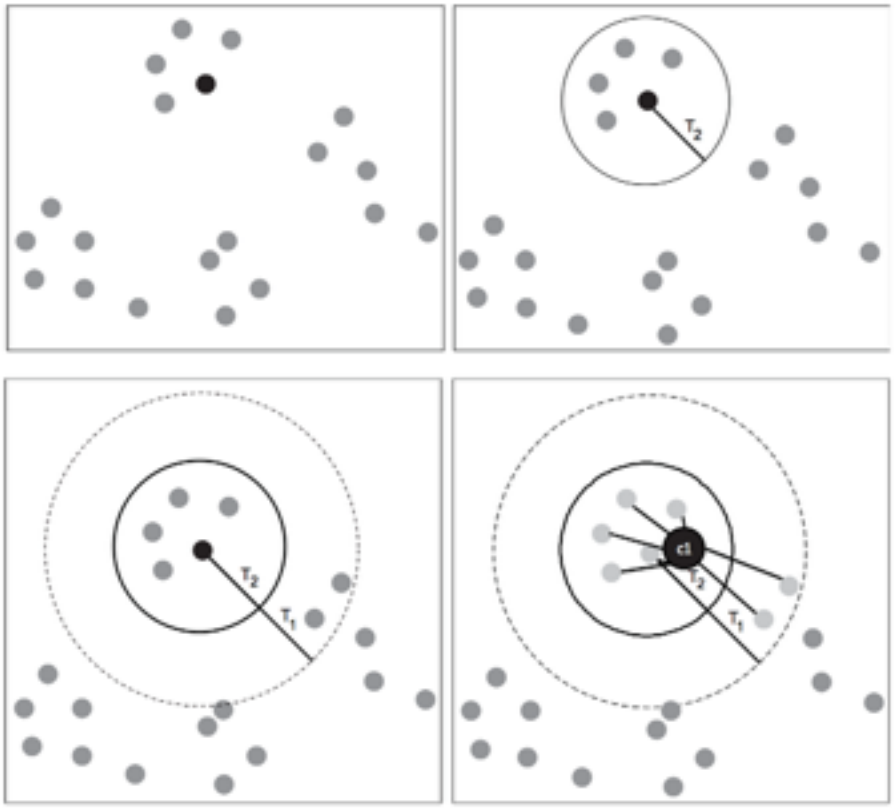
```
$ ls -l reuters-kmeans-clusters
drwxr-xr-x  4 user  5000  136 Feb 1 18:56 clusters-0
drwxr-xr-x  4 user  5000  136 Feb 1 18:56 clusters-1
drwxr-xr-x  4 user  5000  136 Feb 1 18:56 clusters-2
...
drwxr-xr-x  4 user  5000  136 Feb 1 18:59 clusteredPoints

$ bin/mahout clusterdump -dt sequencefile \
-d reuters-vectors/dictionary.file-* \
-s  reuters-kmeans-clusters/clusters-19 -b 10 -n 10
```

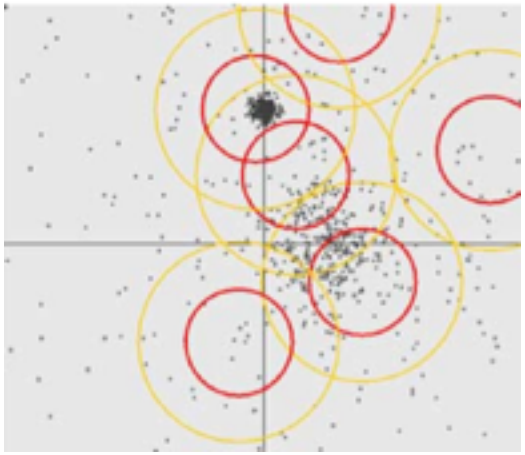
Running `ClusterDumper` on the output folder corresponding to the last iteration produces output similar to the following:

```
Id: 11736:
    Top Terms: debt, banks, brazil, bank, billion, he, payments, billion
     dlrs, interest, foreign

Id: 11235:
    Top Terms: amorphous, magnetic, metals, allied signal, 19.39, corrosion,
     allied, molecular, mode, electronic components
...
Id: 20073:
    Top Terms: ibm, computers, computer, att, personal, pc, operating system,
     intel, machines, dos
```

Canopy clustering to estimate the number of clusters

Tell what size clusters to look for. The algorithm will find the number of clusters that have approximately that size. The algorithm uses two distance thresholds. This method prevents all points close to an already existing canopy from being the center of a new canopy.





Canopy clustering: if you start with a point (top left) and mark it as part of a canopy, all the points within distance T_2 (top right) are removed from the data set and prevented from becoming new canopies. The points within the outer circle (bottom-right) are also put in the same canopy, but they're allowed to be part of other canopies. This assignment process is done in a single pass on a mapper. The reducer computes the average of the centroid (bottom right) and merges close canopies.

Running canopy clustering

To run canopy generation over the Reuters data set, execute the canopy program using the Mahout launcher as follows:

```
$ bin/mahout canopy -i reuters-vectors/tfidf-vectors \
-o reuters-canopy-centroids \
-dm org.apache.mahout.common.distance.EuclideanDistanceMeasure \
-t1 1500 -t2 2000
```

Within a minute, CanopyDriver will generate the centroids in the output folder.
Created less than 50 centroids.

```
$ bin/mahout kmeans -i reuters-vectors/tfidf-vectors \
-o reuters-kmeans-clusters \
-dm org.apache.mahout.common.distance.TanimotoDistanceMeasure  \
-c reuters-canopy-centroids/clusters-0 -cd 0.1 -ow -x 20 -cl
```

After the clustering is done, use ClusterDumper to inspect the clusters. Some of them are listed here:

```
Id: 21523:name:
    Top Terms:
tones, wheat, grain, said, usda, corn, us, sugar, export, agriculture
Id: 21409:name:
    Top Terms:
stock, share, shares, shareholders, dividend, said, its, common, board,
     company
Id: 21155:name:
    Top Terms:
oil, effective, crude, raises, prices, barrel, price, cts, said, dlrs
Id: 19658:name:
    Top Terms:
drug, said, aids, inc, company, its, patent, test, products, food
Id: 21323:name:
    Top Terms:
       7-apr-1987, 11, 10, 12, 07, 09, 15, 16, 02, 17
```

News clustering code

```
public class NewsKMeansClustering {
  public static void main(String args[]) throws Exception {
    int minSupport = 2;
    int minDf = 5;
    int maxDFPercent = 95;
    int maxNGramSize = 2;
    int minLLRValue = 50;
    int reduceTasks = 1;
    int chunkSize = 200;
    int norm = 2;
    boolean sequentialAccessOutput = true;

    String inputDir = "inputDir";

    Configuration conf = new Configuration();
    FileSystem fs = FileSystem.get(conf);

    String outputDir = "newsClusters";
    HadoopUtil.delete(new Path(outputDir));
    Path tokenizedPath = new Path(outputDir,
        DocumentProcessor.TOKENIZED_DOCUMENT_OUTPUT_FOLDER);
    MyAnalyzer analyzer = new MyAnalyzer();
```

Custom Lucene Analyzer

News clustering example —> finding related articles

News clustering code — II

```
DocumentProcessor.tokenizeDocuments(new Path(inputDir),
  analyzer.getClass().asSubclass(Analyzer.class),
  tokenizedPath, conf);

DictionaryVectorizer.createTermFrequencyVectors(tokenizedPath,
  new Path(outputDir), conf, minSupport, maxNGramSize, minLLRValue,
  2, true, reduceTasks,
  chunkSize, sequentialAccessOutput, false);
TFIDFConverter.processTfIdf(
  new Path(outputDir ,
 DictionaryVectorizer.DOCUMENT_VECTOR_OUTPUT_FOLDER),
  new Path(outputDir), conf, chunkSize, minDf,
  maxDFPercent, norm, true, sequentialAccessOutput, false,
 reduceTasks);
Path vectorsFolder = new Path(outputDir, "tfidf-vectors");
Path canopyCentroids = new Path(outputDir ,
                                "canopy-centroids");
Path clusterOutput = new Path(outputDir , "clusters");

CanopyDriver.run(vectorsFolder, canopyCentroids,
  new EuclideanDistanceMeasure(), 250, 120,
  false, false);
KMeansDriver.run(conf, vectorsFolder,
  new Path(canopyCentroids, "clusters-0"),
  clusterOutput, new TanimotoDistanceMeasure(), 0.01,
  20, true, false);
```

Tokenize text

Calculate TF-IDF vectors

Run canopy centroid generation

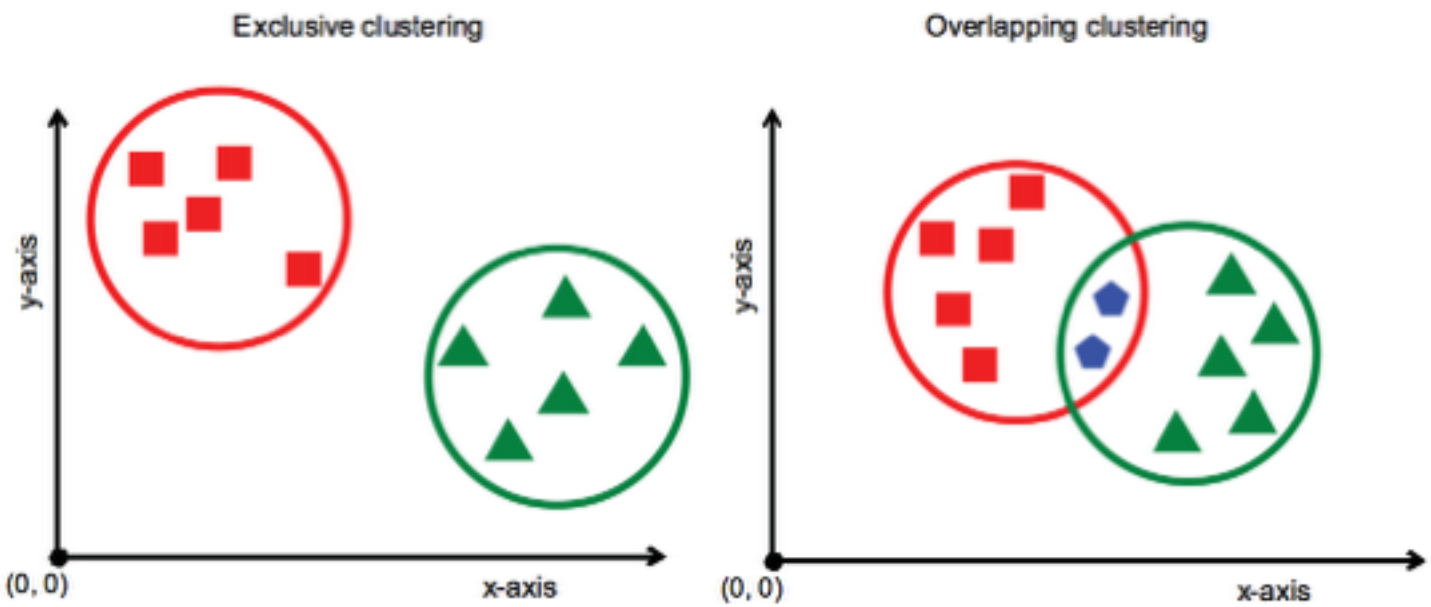
Run k-means algorithm

News clustering code — III

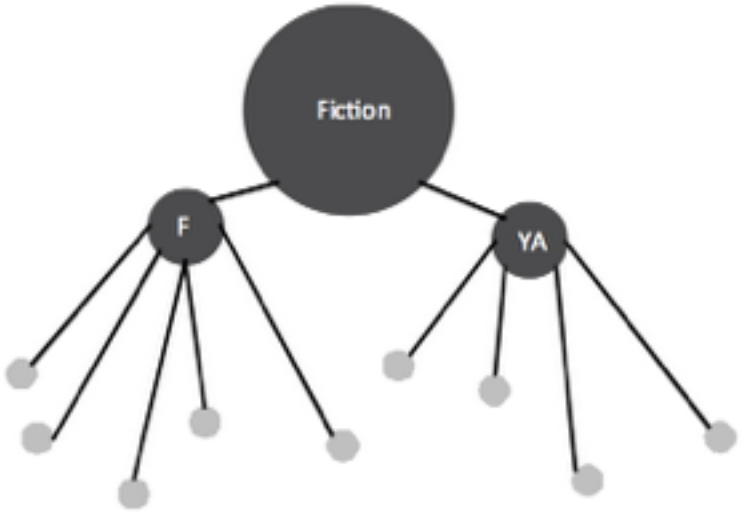
```
    SequenceFile.Reader reader = new SequenceFile.Reader(fs,
        new Path(clusterOutput
            + Cluster.CLUSTERED_POINTS_DIR + "/part-00000"), conf);

    IntWritable key = new IntWritable();
    WeightedVectorWritable value = new WeightedVectorWritable();
    while (reader.next(key, value)) {
       System.out.println(key.toString() + " belongs to cluster "
       + value.toString());
    }
    reader.close();
  }
}
```

Read Vector to Cluster mappings

Other clustering algorithms



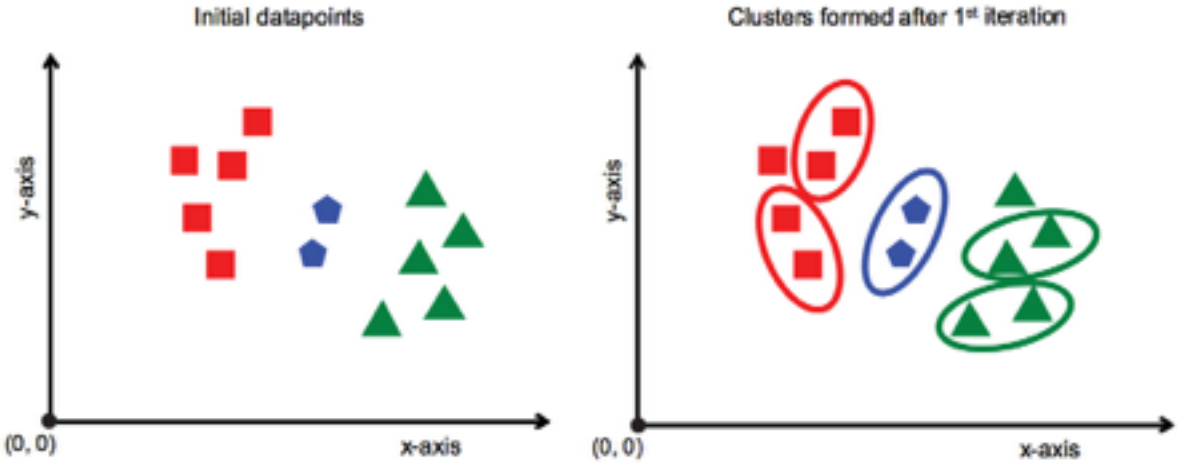
Hierarchical clustering



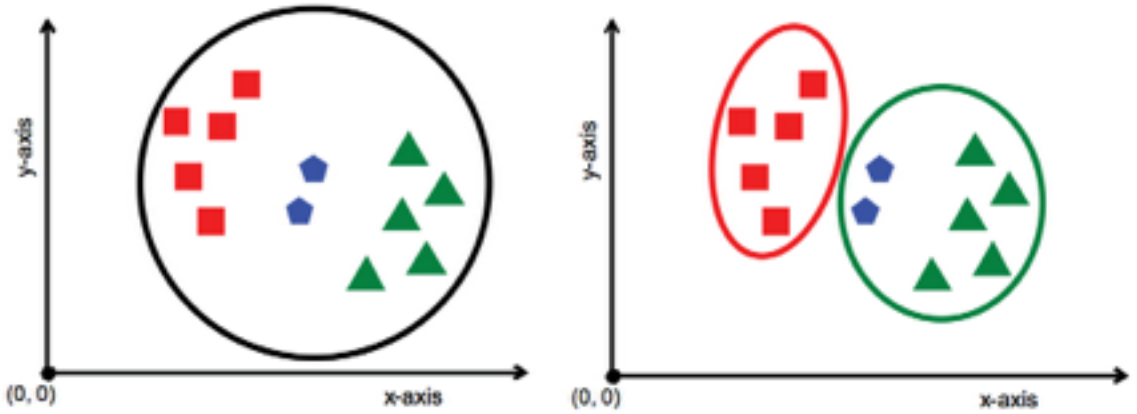
Different clustering approaches

FIXED NUMBER OF CENTERS

BOTTOM-UP APPROACH: FROM POINTS TO CLUSTERS VIA GROUPING



TOP-DOWN APPROACH: SPLITTING THE GIANT CLUSTER



Questions?