Supervised and Unsupervised Learning Algorithms Using Spark

Lecture 6

Nouman M Durrani

Acknowledgement to all authors whose materials have been used

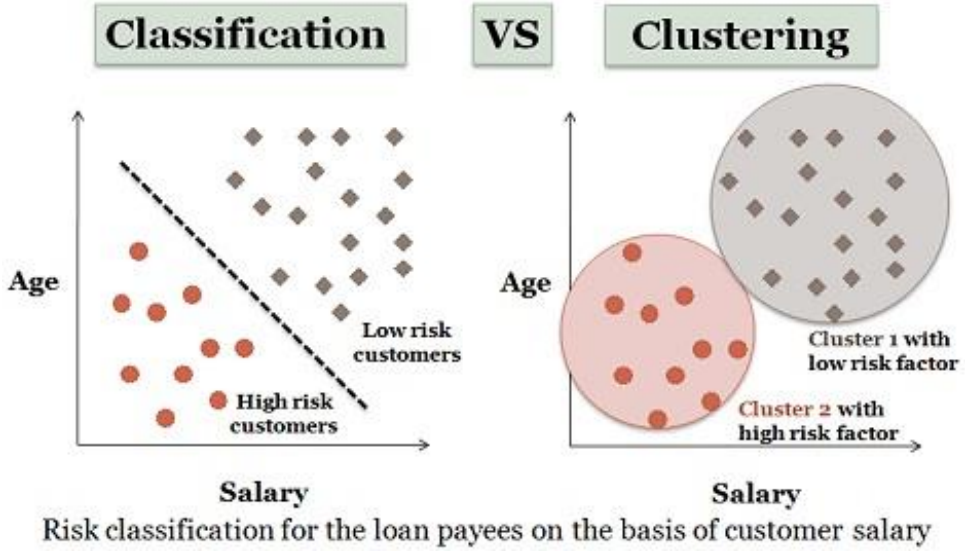
Basics: Supervised Learning Vs. Unsupervised Learning

- When the training is provided to the system
- For example: K-NN Classification, Naïve Base Classification, Linear Regression etc;

- **Unsupervised learning** does not involve training or learning

- For Example: k-means Clustering, Hierarchical Clustering

Basics: Classification and Clustering



Risk classification for the loan payees on the basis of customer salary

- Classification is used in supervised learning technique
- Clustering is used in unsupervised learning

- The similarity is measured by the **similarity function**

- Shorter the distance higher the similarity,
- Longer the distance higher the dissimilarity

Classification learns from existing categorizations and then assigns unclassified items to the best category.

Basics: Regression Algorithms

- Attempt to estimate the mapping function (f) from the input variables (x) to numerical or continuous output variables (y).

- For example, when you are asked to predict their prices, that is a regression task because price will be a continuous output.

- Examples of the common regression algorithms include linear regression, Support Vector Regression (SVR), and regression trees.

Basics: Classification Algorithms

- Attempt to estimate the mapping function (f) from the input variables (x) to discrete or categorical output variables (y).

- **For example**, a classification algorithm can try to predict whether the prices for the houses “sell more or less than the recommended retail price.”
- **For example**, the customer who applies for a loan may be classified as a safe and risky according to his/her age and salary.

- Examples of classification algorithms include logistic regression, Naïve Bayes, decision trees, and K Nearest Neighbors.

The clustering Problem:

- Given an integer k and a set of n data points in R^d, choose k centers so as to minimize φ

- K-means is the most popular clustering algorithm used in scientific and industrial applications" [3]

k-means Clustering

- A type of unsupervised learning, used when you have unlabeled data
- Find groups in the data, with the number of groups represented by the variable **k**.
- Data points are clustered based on feature similarity.

k-means Clustering

1. Arbitrarily choose an initial k centers $C=\{c_1, c_2, \cdots, c_k\}$.
2. For each $i \in \{1, \ldots, k\}$, set the cluster C_i to be the set of points in X that are closer to c_i than they are to c_j for all $j \neq i$.
3. For each $j \in \{1, \ldots, k\}$, set c_j to be the center of mass of all points in C_i: $c_i = \frac{1}{|C_i|} \sum_{x \in C_i} x$.
4. Repeat Steps 2 and 3 until C no longer changes.

Business Uses

- The *K*-means clustering algorithm is used to find groups which have not been explicitly labeled in the data.
- This can be used to confirm business assumptions about what types of groups exist or to identify unknown groups in complex data sets.

Some examples of use cases are:

1. Behavioral segmentation:
    - Segment by purchase history
    - Segment by activities on application, website, or platform
    - Define personas based on interests
    - Create profiles based on activity monitoring

2. Inventory categorization:

- Group inventory by sales activity
- Group inventory by manufacturing metrics

3. Sorting sensor measurements:

- Detect activity types in motion sensors
- Group images
- Separate audio
- Identify groups in health monitoring

4. Detecting bots or anomalies:

- Separate valid activity groups from bots
- Group valid activity to clean up outlier detection

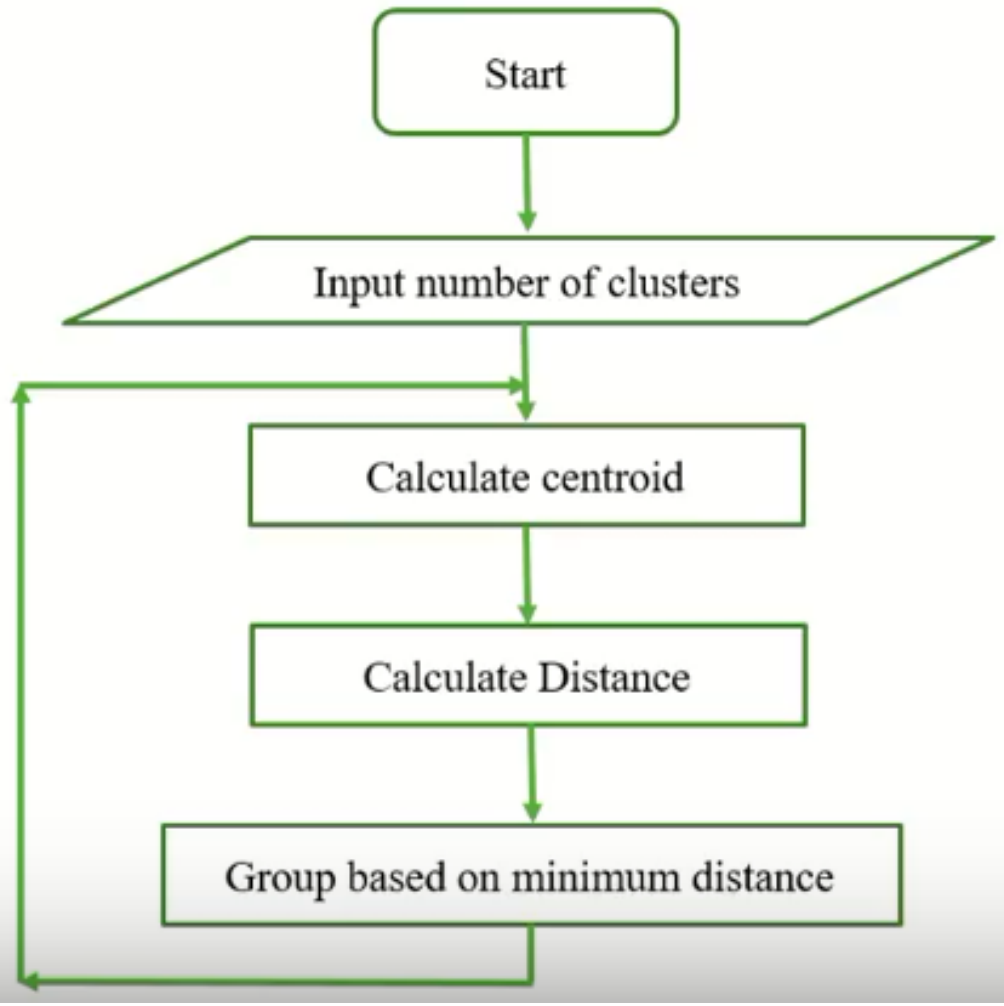
In addition, monitoring if a tracked data point switches between groups over time can be used to detect meaningful changes in the data.

K- MEAN CLUSTERING

- **E**xploratory data analysis technique.
- **I**mplements non hierarchical method of grouping objects together.
- **D**etermines the centroid using the Euclidean method for distance calculation.
- **G**roups the objects based on minimum distance.

It analyze and explore the whole dataset.

K- MEAN CLUSTERING



K- MEAN CLUSTERING

- Apply K-Mean Clustering for the following data sets for two clusters. Tabulate all the assignments.

Sample No	X	Y
1	185	72
2	170	56
3	168	60
4	179	68
5	182	72
6	188	77

k-means Clustering

- Given k = 2

Initial Centroid

Cluster	X	Y
k1	185	72
k2	170	56

- Calculate Euclidean distance using the given equation.

$$\text{Distance } [(x,y), (a,b)] = \sqrt{(x-a)^2+(x-b)^2}$$

Cluster 1 (185,72)= $\sqrt{(185-185)^2+(72-72)^2} = 0$

Distance from Cluster 2 = $\sqrt{(170-185)^2+(56-72)^2}$

(170,56) = $\sqrt{(-15)^2+(-16)^2}$

= $\sqrt{255+256}$

= $\sqrt{481}$

= 21.93

Distance from Cluster 1 = $\sqrt{(185-170)^2+(72-56)^2}$

(185,72) = $\sqrt{(15)^2+(16)^2}$

= $\sqrt{255+256}$

= $\sqrt{481}$

= 21.93

Cluster 2 (170,56)= $\sqrt{(170-170)^2+(56-56)^2} = 0$

Cluster	Centroid		
	X	Y	ASSIGNMENT
k1	0	21.93	1
k2	21.93	0	2

Initial Centroid

Cluster	X	Y
k1	185	72
k2	170	56

- Calculate Euclidean distance for the next dataset (168,60)

$$\text{Distance } [(x,y), (a,b)] = \sqrt{(x-a)^2+(x-b)^2}$$

Distance from Cluster 1 $= \sqrt{(168-185)^2+(60-72)^2}$

(185,72) $= \sqrt{(-17)^2+(-12)^2}$

$= \sqrt{283+144}$

$= \sqrt{433}$

$= 20.808$

Distance from Cluster 2 $= \sqrt{(168-170)^2+(60-56)^2}$

(170,56) $= \sqrt{(-2)^2+(-4)^2}$

$= \sqrt{4+16}$

$= \sqrt{20}$

$= 4.472$

k-means Clustering

Dataset	Euclidean Distance		
	Cluster 1	Cluster 2	ASSIGNMENT
(168,60)	20.808	4.472	2

- Update the cluster centroid.

Cluster	X	Y
k1	185	72
k2	= (170 + 168)/ 2 = 169	= (60+56)/ 2 = 58

- Calculate Euclidean distance for the next dataset (179,68)

$$\text{Distance } [(x,y), (a,b)] = \sqrt{(x-a)^2+(x-b)^2}$$

Distance from Cluster 1 $= \sqrt{(179-185)^2+(68-72)^2}$

(185,72) $= \sqrt{(-6)^2+(-4)^2}$

$= \sqrt{36+16}$

$= \sqrt{52}$

$= 7.211103$

- Calculate Euclidean distance for the next dataset (179,68)

$$\text{Distance } [(x,y), (a,b)] = \sqrt{(x-a)^2+(x-b)^2}$$

Distance from Cluster 2 $= \sqrt{(179-169)^2+(68-58)^2}$

(169,58) $= \sqrt{(10)^2+(10)^2}$

$= \sqrt{100+100}$

$= \sqrt{200}$

$= 14.14214$

The k-means Clustering

Dataset	Euclidean Distance		
	Cluster 1	Cluster 2	ASSIGNMENT
(179,68)	7.211103	14.14214	1

- Update the cluster centroid.

Cluster	X	Y
k1	= 185+179/2 =182	= 72+68/2 =70
k2	169	58

- Calculate Euclidean distance for the next dataset (182,72)

$$\text{Distance } [(x,y), (a,b)] = \sqrt{(x-a)^2+(x-b)^2}$$

$$\text{Distance from Cluster 1} = \sqrt{(182-182)^2+(72-70)^2}$$

(182,70)

$$= \sqrt{(0)^2+(2)^2}$$
$$= \sqrt{0+4}$$
$$= \sqrt{4}$$
$$= 2$$

- Calculate Euclidean distance for the next dataset (182,72)

$$\text{Distance } [(x,y), (a,b)] = \sqrt{(x-a)^2+(x-b)^2}$$

$$\text{Distance from Cluster 2} = \sqrt{(182-169)^2+(72-58)^2}$$

(169,58)

$$= \sqrt{(13)^2+(14)^2}$$
$$= \sqrt{169+196}$$
$$= \sqrt{365}$$
$$= 19.10$$

The k-means Clustering

Dataset	Euclidean Distance		
	Cluster 1	Cluster 2	ASSIGNMENT
(182,72)	2	19.10	1

- Update the cluster centroid.

Cluster	X	Y
k1	= 182+182/2 =182	= 70+72/2 = 71
k2	169	58

The k-means Clustering

- Final Assignment

Dataset No	X	Y	Assignment
1	185	72	1
2	170	56	2
3	168	60	2
4	179	68	1
5	182	72	1
6	188	77	1

The k-means++ algorithm

We propose a specific way of choosing centers for the **k-means** algorithm. In particular, let $D(x)$ denote the shortest distance from a data point to the closest center we have already chosen. Then, we define the following algorithm, which we call **k-means++**.

1a. Take one center c_1, chosen uniformly at random from $\mathcal{X}$.
1b. Take a new center c_i, choosing $x \in \mathcal{X}$ with probability $\frac{D(x)^2}{\sum_{x \in \mathcal{X}} D(x)^2}$.
1c. Repeat Step 1b. until we have taken k centers altogether.
2-4. Proceed as with the standard **k-means** algorithm.

We call the weighting used in Step 1b simply "D^2 weighting".

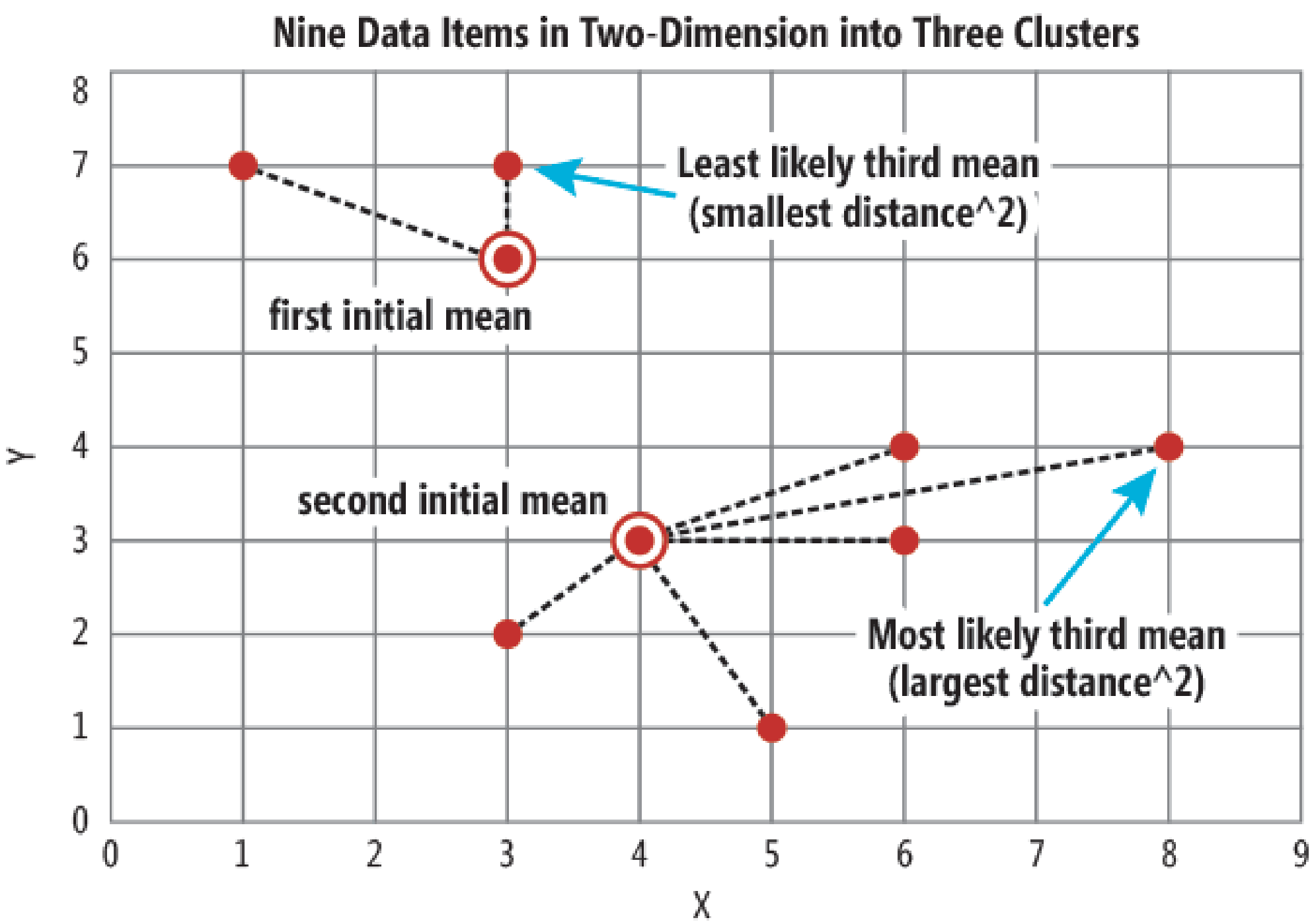
The k-means ++ Algorithm

- The first step is to choose a data point at random. Call this point s_1. Next, compute the squared distances

$$D_i^2 = ||Y_i - s_1||^2.$$

- Now choose a second point s_2 from the data. The probability of choosing Y_i is $D_i^2 / \sum_j D_j^2$
- Now recompute the distance as $D_i^2 = \min\left\{||Y_i - s_1||^2, ||Y_i - s_2||^2\right\}$.
- Now choose a third point s_3 from the data where the probability of choosing Y_i is $D_i^2 / \sum_j D_j^2$.
- We continue until we have k points s_1, s_2, s_3,…, s_k.
- Finally, we run k-means clustering using s_1, s_2, s_3,…, s_k as starting values. Call the resulting centers c_1, c_2, c_3,…,c_k.
- Arthur and Vassilvitskii proved that the expected value is over the randomness in the algorithm
- .

$$\mathbb{E}[R(\hat{c}_1, \ldots, \hat{c}_k)] \leq 8(\log k + 2) \min_{c_1,\ldots,c_k} R(c_1, \ldots, c_k).$$


Nine Data Items in Two-Dimension into Three Clusters
Least likely third mean
(smallest distance^2)
first initial mean
second initial mean
Most likely third mean
(largest distance^2)
Y
X
0
1
2
3
4
5
6
7
8
9

The k-means ++ Algorithm

- The first initial mean at (3, 6) was randomly selected.
- Then the distance-squared from each of the other 8 data items to the first mean was computed, and using that information, the second initial mean at (4, 3) was selected.
- To select a data item as the third initial mean, the squared distance from each data point to its closest mean is computed.
- The distances are shown as dashed lines.
- Using these squared distance values, the third mean will be selected so that data items with small squared distance values have a low probability of being selected, and data items with large squared distance values have a high probability of being selected.
- This technique is sometimes called proportional fitness selection.

The k-means ++ Algorithm: Proportional fitness selection using Roulette wheel selection

- Proportional fitness selection is the heart of the k-means++ initialization mechanism.
- There are several ways to implement proportional fitness selection.
- Here we use Roulette wheel selection for proportional fitness selection.
- Suppose there are four candidate items (0, 1, 2, 3) with associated values (20.0, 10.0, 40.0, 30.0).
- The sum of the values is 20.0 + 40.0 + 10.0 + 30.0 = 100.0.
- Proportional fitness selection will pick item 0 with probability 20.0/100.0 = 0.20; pick item 1 with probability 10.0/100.0 = 0.10; pick item 2 with probability 40.0/100.0 = 0.40; and pick item 3 with probability 30.0/100.0 = 0.30.

The k-means ++ Algorithm: Proportional fitness selection using Roulette wheel selection

- If the probabilities of selection are stored in an array as (0.20, 0.10, 0.40, 0.30), the cumulative probabilities can be stored in an array with values (0.20, 0.30, 0.70, 1.00).
- Now, suppose a random p is generated with value 0.83.
- If i is an array index into the cumulative probabilities array, when i = 0, cum[i] = 0.20, which isn't greater than p = 0.83, so i increments to 1.
- Now cum[i] = 0.30, which is still not greater than p, so i increments to 2.
- Now cum[i] = 0.70, which is still not greater than p, so i increments to 3.
- Now cum[i] = 1.00, which is greater than p, so i = 3 is returned as the selected item.

The k-means and k-means ++ Algorithm implantation using Spark

- The implementation in spark.mllib has the following parameters:

- k is the number of desired clusters. Note that it is possible for fewer than k clusters to be returned, for example, if there are fewer than k distinct points to cluster.
- maxIterations is the maximum number of iterations to run.
- initializationMode specifies either random initialization or initialization via k-means++.
- initializationSteps determines the number of steps in the k-means++ algorithm.
- epsilon determines the distance threshold within which we consider k-means to have converged.
- initialModel is an optional set of cluster centers used for initialization. If this parameter is supplied, only one run is performed.

```scala
import org.apache.spark.mllib.clustering.{KMeans, KMeansModel}
import org.apache.spark.mllib.linalg.Vectors

// Load and parse the data
val data = sc.textFile("data/mllib/kmeans_data.txt")
val parsedData = data.map(s => Vectors.dense(s.split(' ').map(_.toDouble))).cache()

// Cluster the data into two classes using KMeans
val numClusters = 2
val numIterations = 20
val clusters = KMeans.train(parsedData, numClusters, numIterations)

// Evaluate clustering by computing Within Set Sum of Squared Errors
val WSSSE = clusters.computeCost(parsedData)
println(s"Within Set Sum of Squared Errors = $WSSSE")

// Save and load model
clusters.save(sc, "target/org/apache/spark/KMeansExample/KMeansModel")
val sameModel = KMeansModel.load(sc, "target/org/apache/spark/KMeansExample/KMeansModel")
```

Streaming k-means implementation using Spark

- When data arrive in a stream, estimate clusters dynamically,
- spark.mllib provides support for streaming k-means clustering,
  - parameters to control the decay (or “forgetfulness”) of the estimates.
- The generalization of the mini-batch k-means update rule.
- For each batch of data, we assign all points to their nearest cluster, compute new cluster centers, then update each cluster using:

$$c_{t+1} = \frac{c_t n_t \alpha + x_t m_t}{n_t \alpha + m_t} \quad (1)$$

$$n_{t+1} = n_t + m_t \quad (2)$$

where c_t is the previous center for the cluster, n_t is the number of points assigned to the cluster thus far, x_t is the new cluster center from the current batch, and m_t is the number of points added to the cluster in the current batch.

Streaming k-means implementation using Spark

- with α=1 all data will be used from the beginning;
- with α =0 only the most recent data will be used.
- This is analogous to an exponentially-weighted moving average.

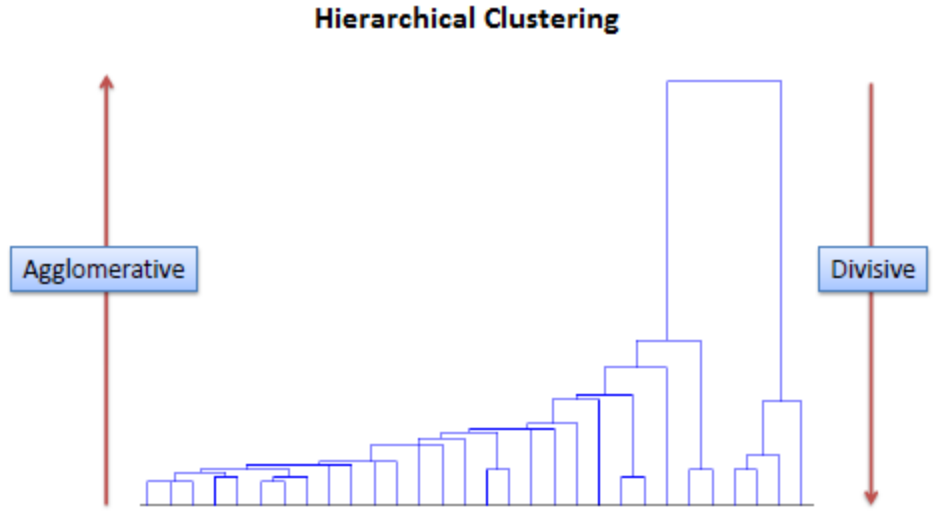
- t + halfLife will have dropped to 0.5.

Streaming k-means implementation using Spark

- Add new text files with data the cluster centers will update.
- Each training point should be formatted as [x1, x2, x3], and each test data point should be formatted as (y, [x1, x2, x3]), where y is some useful label or identifier (e.g. a true category assignment).
- Anytime a text file is placed in /training/data/dir the model will update.
- Anytime a text file is placed in /testing/data/dir you will see predictions. With new data, the cluster centers will change!

- **Find full example code at** "examples/src/main/python/mllib/streaming_k_means_example.py" in the Spark repo.
- **Find full example code at** "examples/src/main/scala/org/apache/spark/examples/mllib/StreamingKMeansExample.scala" in the Spark repo.

Hierarchical clustering



Creating clusters that have a predetermined ordering from top to bottom.

Divisive method

- Assign all of the observations to a single cluster
- Partition the cluster to two least similar clusters.
- Finally, proceed recursively on each cluster until there is one cluster.
- More accurate hierarchies than agglomerative algorithms in some circumstances
  - but conceptually more complex.

Hierarchical clustering

Agglomerative method

- In agglomerative or bottom-up clustering method we assign each observation to its own cluster.
- Then, compute the similarity (e.g., distance) between each of the clusters
- join the two most similar clusters, a single cluster left.

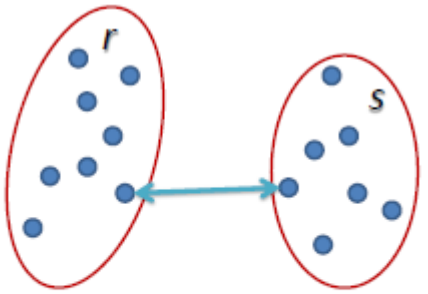
Hierarchical clustering

	BA	FI	MI	NA	RM	TO
BA	0	662	877	255	412	996
FI	662	0	295	468	268	400
MI	877	295	0	754	564	138
NA	255	468	754	0	219	869
RM	412	268	564	219	0	669
TO	996	400	138	869	669	0

Determine the proximity matrix containing the distance between each point

The matrix is updated to display the distance between each cluster.

Three methods differ in how the distance between each cluster is measured.



Single Linkage

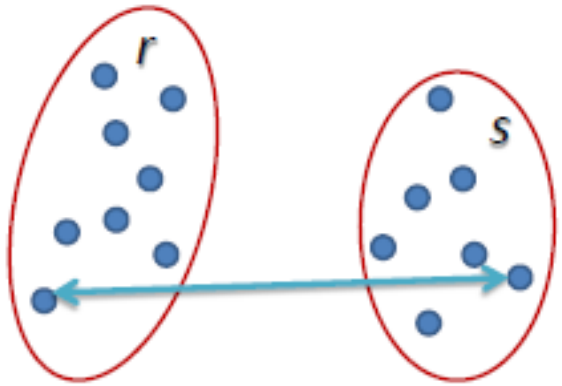
- The distance between two clusters is the shortest distance between two points in each cluster.
- For example, the distance between clusters “r” and “s” to the left is equal to the length of the arrow between their two closest points.

$$L(r,s) = \min(D(x_{ri}, x_{sj}))$$

Hierarchical clustering

Complete Linkage

- The distance between two clusters is defined as the longest distance between two points in each cluster.
- For example, the distance between clusters “r” and “s” to the left is equal to the length of the arrow between their two furthest points.

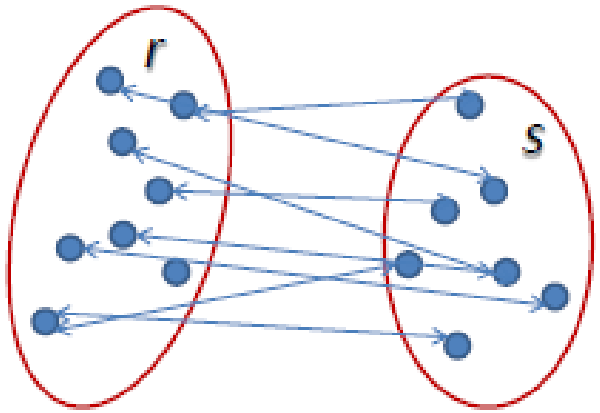


$$L(r,s) = \max(D(x_{ri}, x_{sj}))$$

Hierarchical clustering

Average Linkage

- The distance between two clusters is defined as the average distance between each point in one cluster to every point in the other cluster.
- For example, the distance between clusters “r” and “s” to the left is equal to the average length each arrow between connecting the points of one cluster to the other.



$$L(r,s) = \frac{1}{n_r n_s} \sum_{i=1}^{n_r} \sum_{j=1}^{n_s} D(x_{ri}, x_{sj})$$

Hierarchical clustering

- Begin with the disjoint clustering having level L(0) = 0 and sequence number m = 0.
- Find the least dissimilar pair of clusters in the current clustering, say pair (r), (s), according to

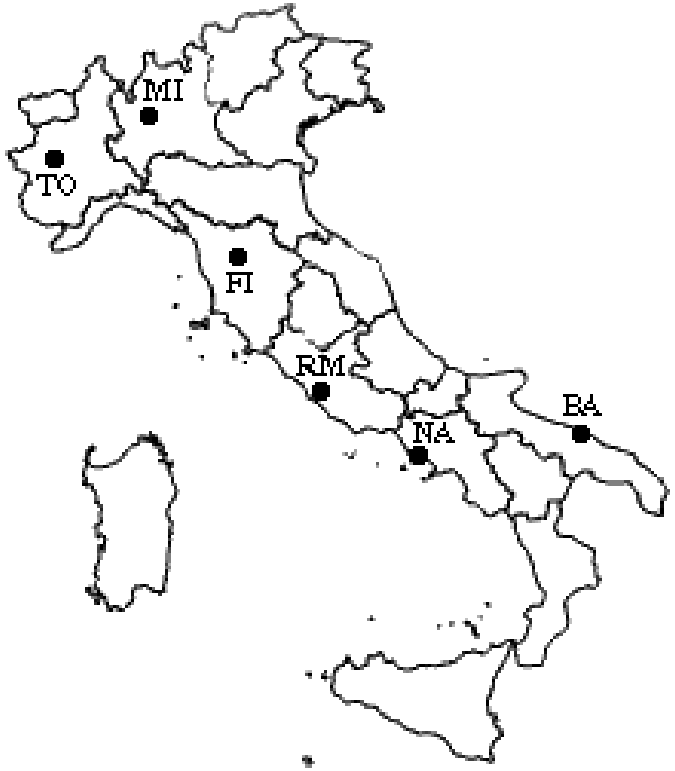
  d[(r),(s)] = min d[(i),(j)] where the minimum is over all pairs of clusters in the current clustering.
- Increment the sequence number : m = m +1. Merge clusters (r) and (s) into a single cluster to form the next clustering m. Set the level of this clustering to

  L(m) = d[(r),(s)]
- Update the proximity matrix, D, by deleting the rows and columns corresponding to clusters (r) and (s) and adding a row and column corresponding to the newly formed cluster. The proximity between the new cluster, denoted (r,s) and old cluster (k) is defined in this way:

  d[(k), (r,s)] = min d[(k),(r)], d[(k),(s)]
- If all objects are in one cluster, stop. Else, go to step 2.

Hierarchical Clustering

	BA	FI	MI	NA	RM	TO
BA	0	662	877	255	412	996
FI	662	0	295	468	268	400
MI	877	295	0	754	564	138
NA	255	468	754	0	219	869
RM	412	268	564	219	0	669
TO	996	400	138	869	669	0



Hierarchical Clustering

	BA	FI	MI/TO	NA	RM
BA	0	662	877	255	412
FI	662	0	295	468	268
MI/TO	877	295	0	754	564
NA	255	468	754	0	219
RM	412	268	564	219	0

Hierarchical Clustering

	BA	FI	MI/TO	NA/RM
BA	0	662	877	255
FI	662	0	295	268
MI/TO	877	295	0	564
NA/RM	255	268	564	0

Hierarchical Clustering

	BA/NA/RM	FI	MI/TO
BA/NA/RM	0	268	564
FI	268	0	295
MI/TO	564	295	0

min d(i,j) = d(BA/NA/RM,FI) = 268 => merge BA/NA/RM and FI into a new cluster called BA/FI/NA/RM
L(BA/FI/NA/RM) = 268
m = 4

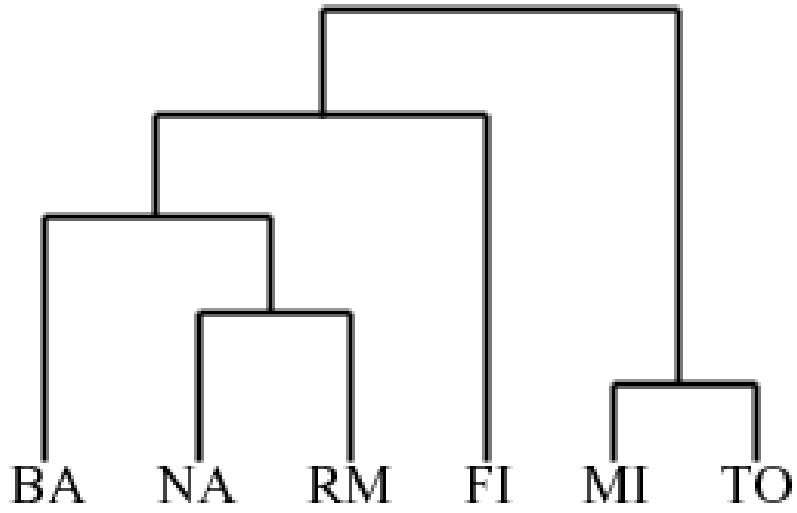
Hierarchical Clustering

	BA/FI/NA/RM	MI/TO
BA/FI/NA/RM	0	295
MI/TO	295	0

Hierarchical Clustering

Finally, we merge the last two clusters at level 295.

The process is summarized by the following hierarchical tree:



Problems with the Hierarchical Clustering:

The main weaknesses of agglomerative clustering methods are:

- they do not scale well: time complexity of at least $O(n^2)$, where n is the number of total objects;
- they can never undo what was done previously.

Bisecting k-means

- Bisecting K-means can often be much faster than regular K-means,
  - generally produce a different clustering.
- Bisecting k-means is a kind of hierarchical clustering.
- Hierarchical clustering seeks to build a hierarchy of clusters.
- Strategies for hierarchical clustering generally fall into two types:
  - Agglomerative
  - Divisive

Bisecting k-means algorithm implementation using Spark

Bisecting k-means algorithm is a kind of divisive algorithms.

The implementation in MLlib has the following parameters:

- *k*: the desired number of leaf clusters (default: 4).
- The actual number could be smaller if there are no divisible leaf clusters.
- *maxIterations*
- *minDivisibleClusterSize*
- *seed*

```scala
import org.apache.spark.mllib.clustering.BisectingKMeans
import org.apache.spark.mllib.linalg.{Vector, Vectors}

// Loads and parses data
def parse(line: String): Vector = Vectors.dense(line.split(" ").map(_.toDouble))
val data = sc.textFile("data/mllib/kmeans_data.txt").map(parse).cache()

// Clustering the data into 6 clusters by BisectingKMeans.
val bkm = new BisectingKMeans().setK(6)
val model = bkm.run(data)

// Show the compute cost and the cluster centers
println(s"Compute Cost: ${model.computeCost(data)}")
model.clusterCenters.zipWithIndex.foreach { case (center, idx) =>
  println(s"Cluster Center ${idx}: ${center}")
}
```