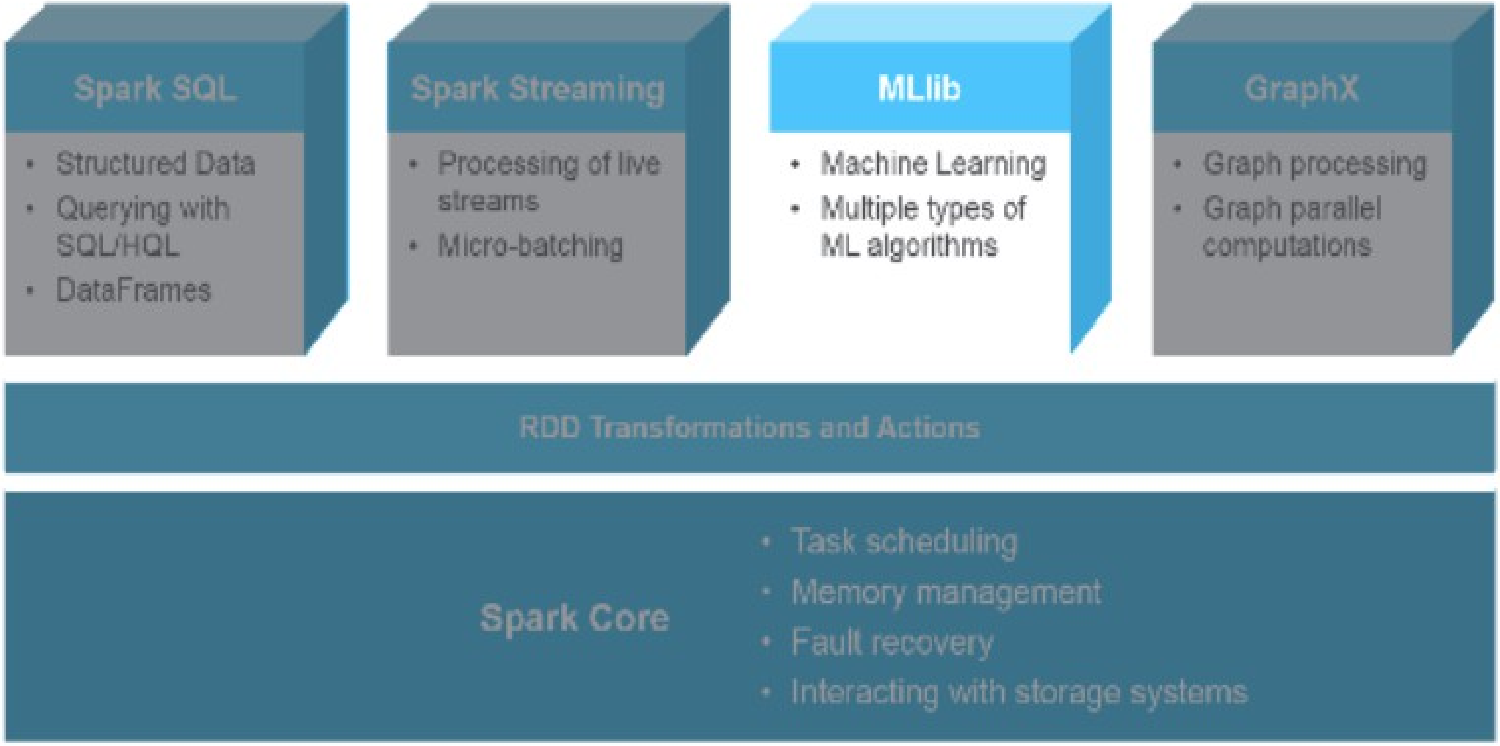
Hands-on: Exercise Machine Learning using Apache Spark MLlib

July 2016

Dr.Thanachart Numnonda
IMC Institute
thanachart@imcinstitute.com

What is MLlib?



Source: MapR Academy

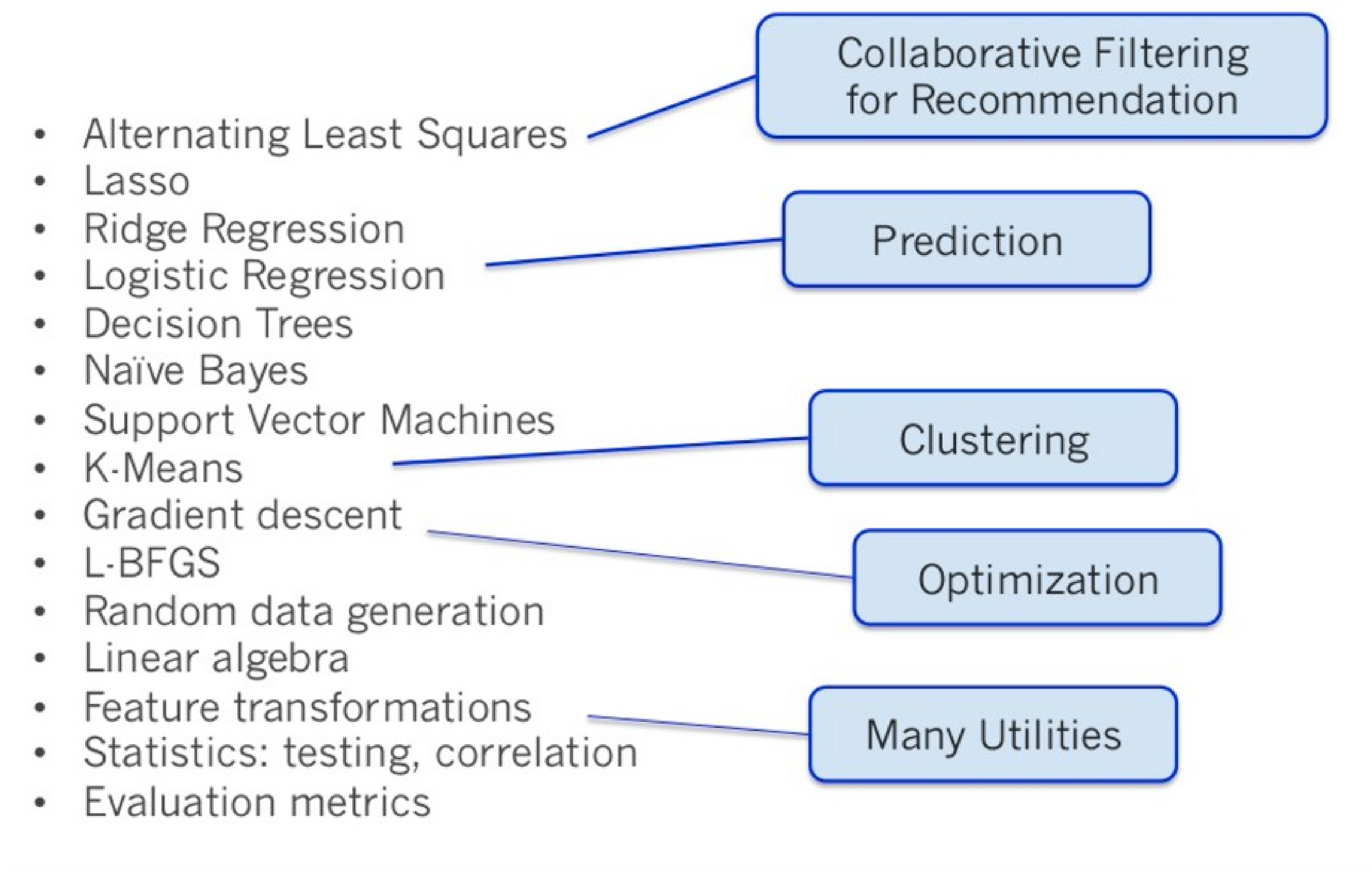
What is MLlib?

- MLlib is a Spark subproject providing machine learning primitives:
  - initial contribution from AMPLab, UC Berkeley
  - shipped with Spark since version 0.8
  - 33 contributors

Mllib Algorithms

- **Classification**: logistic regression, linear support vector machine(SVM), naive Bayes
- **Regression**: generalized linear regression (GLM)
- **Collaborative filtering**: alternating least squares (ALS)
- **Clustering**: k-means
- **Decomposition**: singular value decomposition (SVD), principal component analysis (PCA)

What is in MLlib?



- Alternating Least Squares
- Lasso
- Ridge Regression
- Logistic Regression
- Decision Trees
- Naïve Bayes
- Support Vector Machines
- K-Means
- Gradient descent
- L-BFGS
- Random data generation
- Linear algebra
- Feature transformations
- Statistics: testing, correlation
- Evaluation metrics

Source: Mllib:Spark's Machine Learning Library, A. Talwalkar

MLlib: Benefits

- Part of Spark
- Scalable
- Support: Python, Scala, Java
- Broad coverage of applications & algorithms
- Rapid developments in speed & robustness

Machine Learning

Machine learning is a scientific discipline that explores the construction and study of algorithms that can learn from data.

[Wikipedia]

Vectors

- A point is just a set of numbers. This set of numbers or coordinates defines the point's position in space.
- Points and vectors are same thing.
- Dimensions in vectors are called features
- Hyperspace is a space with more than three dimensions.
- Example: A person has the following dimensions:
  - Weight
  - Height
  - Age
- Thus, the interpretation of point (160,69,24) would be 160 lb weight, 69 inches height, and 24 years age.

Source:Spark Cookbook

Vectors in MLlib

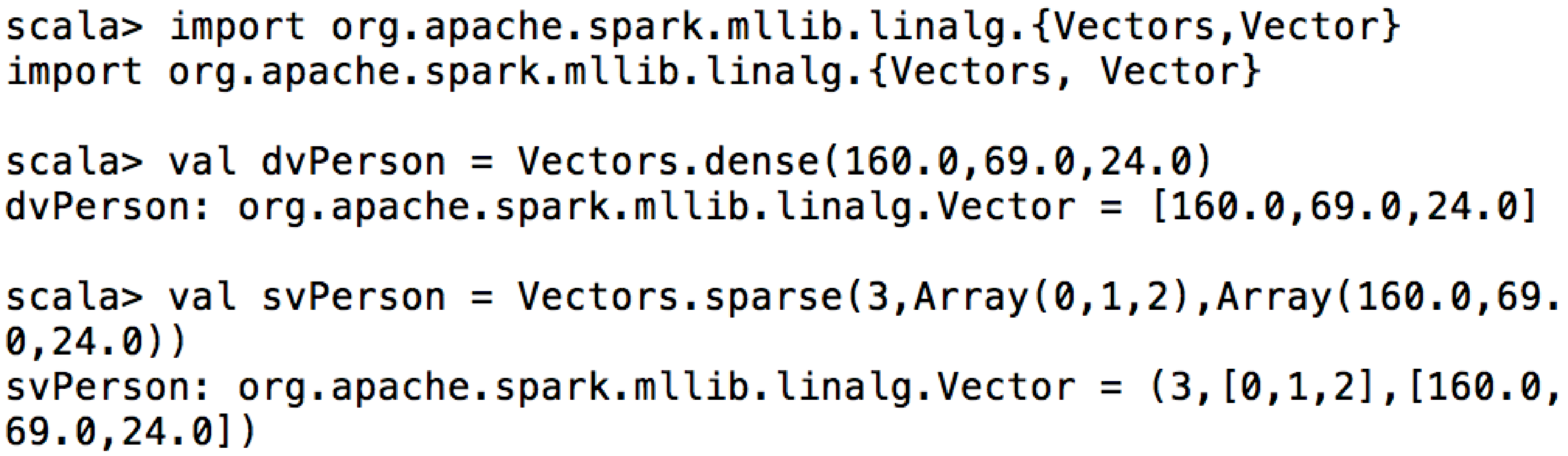
- Spark has local vectors and matrices and also distributed matrices.
  - Distributed matrix is backed by one or more RDDs.
  - A local vector has numeric indices and double values, and is stored on a single machine.
- Two types of local vectors in MLlib:
  - **Dense vector** is backed by an array of its values.
  - **Sparse vector** is backed by two parallel arrays, one for indices and another for values.
- Example
  - Dense vector: [160.0,69.0,24.0]
  - Sparse vector: (3,[0,1,2],[160.0,69.0,24.0])

Source:Spark Cookbook

Vectors in Mllib (cont.)

- Library
  - import org.apache.spark.mllib.linalg.{Vectors,Vector}
- Signature of **Vectors.dense:**
  - def dense(values: Array[Double]): Vector
- Signature of **Vectors.sparse**:
  - def sparse(size: Int, indices: Array[Int], values: Array[Double]): Vector

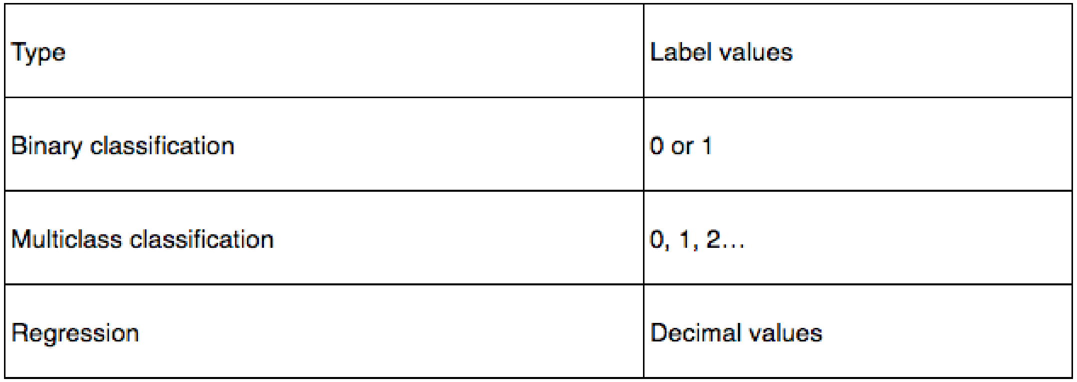
Example



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

scala> val dvPerson = Vectors.dense(160.0,69.0,24.0)
dvPerson: org.apache.spark.mllib.linalg.Vector = [160.0,69.0,24.0]

scala> val svPerson = Vectors.sparse(3,Array(0,1,2),Array(160.0,69.
0,24.0))
svPerson: org.apache.spark.mllib.linalg.Vector = (3,[0,1,2],[160.0,
69.0,24.0])
```

Labeled point

- Labeled point is a local vector (sparse/dense),), which has an associated label with it.
- Labeled data is used in supervised learning to help train algorithms.
- Label is stored as a double value in **LabeledPoint**.



Type	Label values
Binary classification	0 or 1
Multiclass classification	0, 1, 2…
Regression	Decimal values

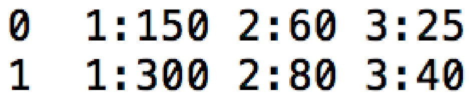
Source:Spark Cookbook

Example

```
scala> import org.apache.spark.mllib.linalg.{Vectors,Vector}
scala> import org.apache.spark.mllib.regression.LabeledPoint
scala>  val willBuySUV =
LabeledPoint(1.0,Vectors.dense(300.0,80,40))


scala> val willNotBuySUV =
LabeledPoint(0.0,Vectors.dense(150.0,60,25))


scala> val willBuySUV =
LabeledPoint(1.0,Vectors.sparse(3,Array(0,1,2),Array(300.0,80,
40)))


scala> val willNotBuySUV =
LabeledPoint(0.0,Vectors.sparse(3,Array(0,1,2),Array(150.0,60,
25)))
```

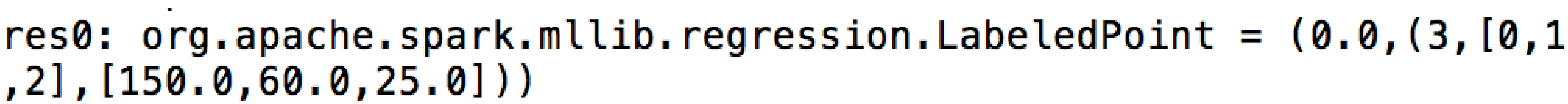
Example (cont)

```
# vi person_libsvm.txt
```



```
0  1:150 2:60 3:25
1  1:300 2:80 3:40
```

```
scala> import org.apache.spark.mllib.util.MLUtils

scala> import org.apache.spark.rdd.RDD

scala> val persons =
MLUtils.loadLibSVMFile(sc,"hdfs:///user/cloudera/person_libsvm
.txt")

scala> persons.first()
```



```
res0: org.apache.spark.mllib.regression.LabeledPoint = (0.0,(3,[0,1
,2],[150.0,60.0,25.0]))
```

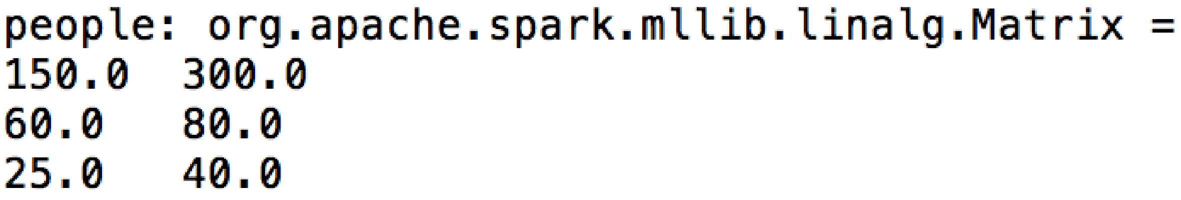
Matrices in MLlib

- Spark has local matrices and also distributed matrices.
  - Distributed matrix is backed by one or more RDDs.
  - A local matrix stored on a single machine.
- There are three types of distributed matrices in MLlib:
  - **RowMatrix**: This has each row as a feature vector.
  - **IndexedRowMatrix**: This also has row indices.
  - **CoordinateMatrix**: This is simply a matrix of MatrixEntry. A MatrixEntry represents an entry in the matrix represented by its row and column index

Source:Spark Cookbook

Example

```
scala> import org.apache.spark.mllib.linalg.{Vectors,Matrix,
Matrices}

scala> val people = Matrices.dense(3,2,Array(150d,60d,25d,
300d,80d,40d))
```

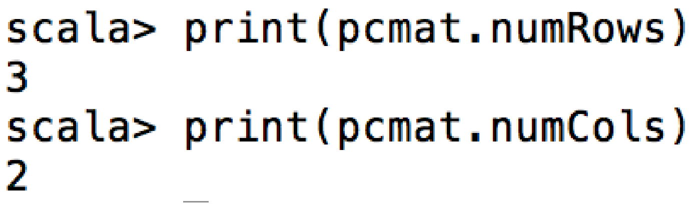


```
people: org.apache.spark.mllib.linalg.Matrix =
150.0  300.0
60.0   80.0
25.0   40.0
```

```
scala>  val personRDD =
sc.parallelize(List(Vectors.dense(150,60,25),
Vectors.dense(300,80,40)))

scala> import org.apache.spark.mllib.linalg.distributed.
{IndexedRow, IndexedRowMatrix,RowMatrix, CoordinateMatrix,
MatrixEntry}

scala> val personMat = new RowMatrix(personRDD)
```

Example

```
scala> print(personMat.numRows)

scala> val personRDD = sc.parallelize(List(IndexedRow(0L,
Vectors.dense(150,60,25)), IndexedRow(1L,
Vectors.dense(300,80,40))))

scala>  val pirmat = new IndexedRowMatrix(personRDD)

scala> val personMat = pirmat.toRowMatrix

scala> val meRDD = sc.parallelize(List(

  MatrixEntry(0,0,150), MatrixEntry(1,0,60),

MatrixEntry(2,0,25), MatrixEntry(0,1,300),

MatrixEntry(1,1,80),MatrixEntry(2,1,40) ))

scala> val pcmat = new CoordinateMatrix(meRDD)
```



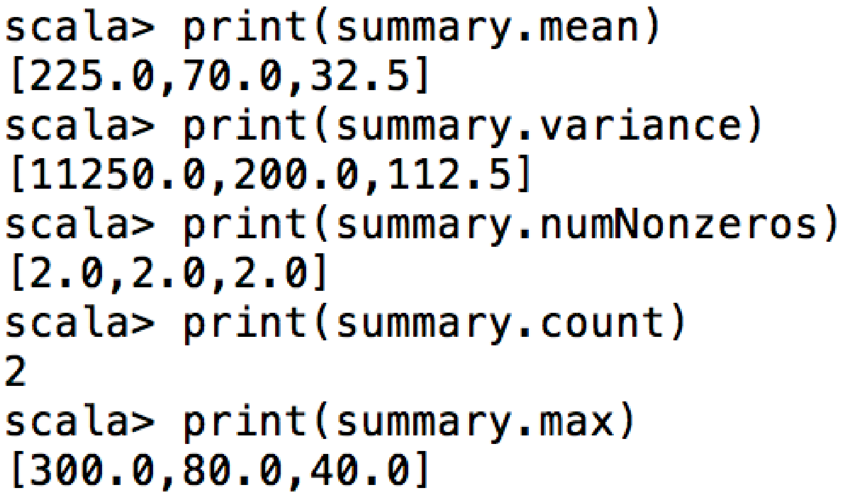
```
scala> print(pcmat.numRows)
3
scala> print(pcmat.numCols)
2
```

Statistic functions

- Central tendency of data—mean, mode, median
- Spread of data—variance, standard deviation
- Boundary conditions—min, max

Example

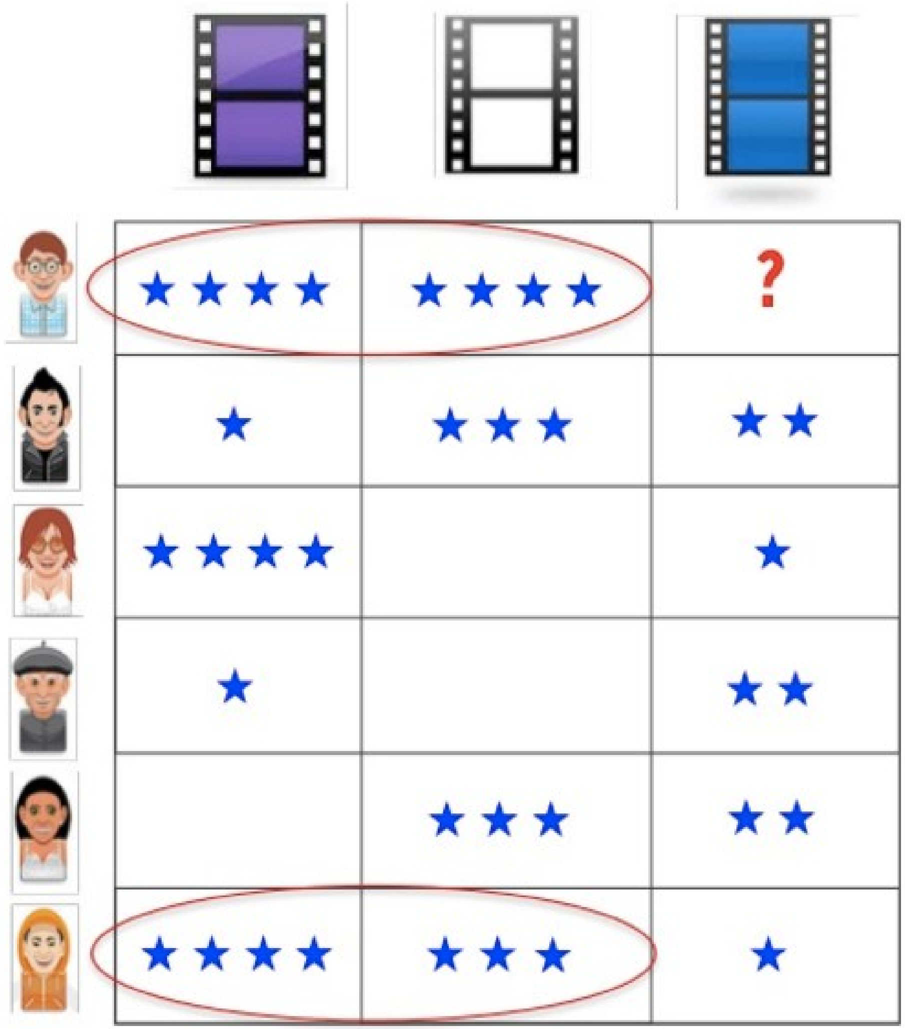
```
scala> import org.apache.spark.mllib.linalg.{Vectors,Vector}

scala> import org.apache.spark.mllib.stat.Statistics

scala> val personRDD =
sc.parallelize(List(Vectors.dense(150,60,25),
Vectors.dense(300,80,40)))

scala> val summary = Statistics.colStats(personRDD)
```



```
scala> print(summary.mean)
[225.0,70.0,32.5]
scala> print(summary.variance)
[11250.0,200.0,112.5]
scala> print(summary.numNonzeros)
[2.0,2.0,2.0]
scala> print(summary.count)
2
scala> print(summary.max)
[300.0,80.0,40.0]
```

Hands-on
Movie Recommendation

Recommendation



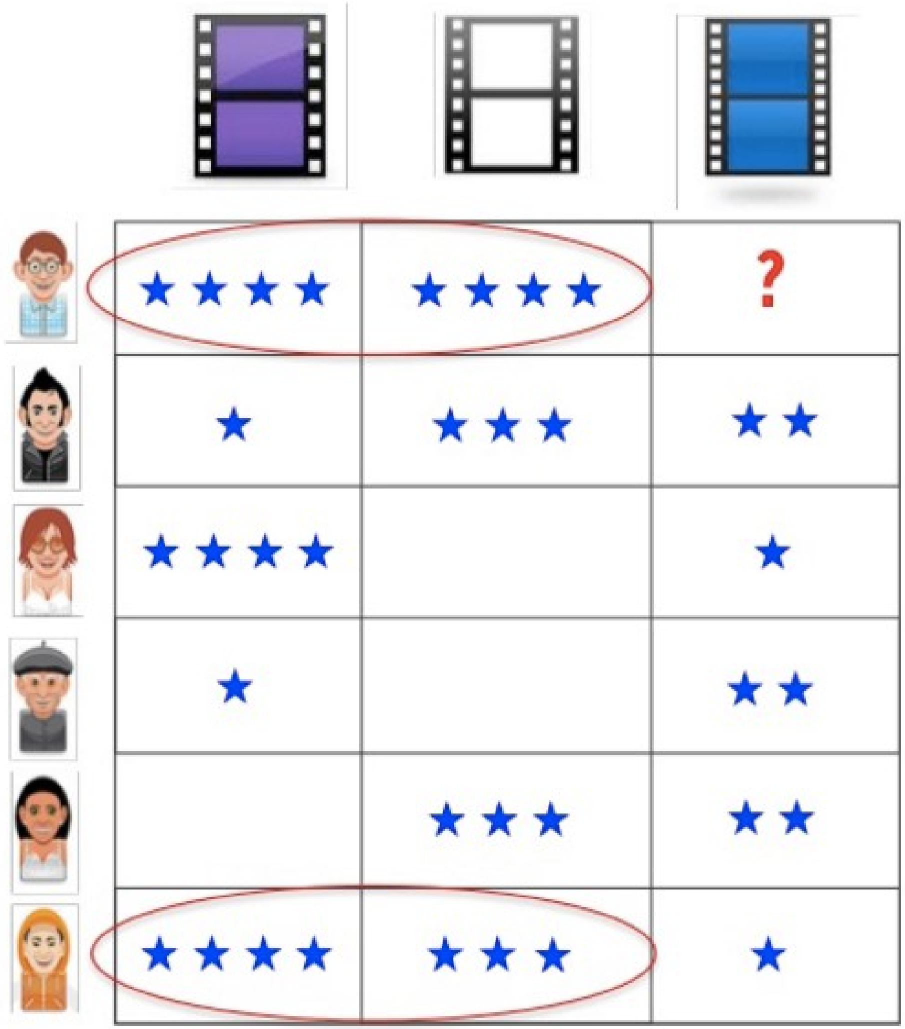
★★★★	★★★★	?
★	★★★	★★
★★★★		★
★		★★
	★★★	★★
★★★★	★★★	★

Goal: Recommend movies to users

NETFLIX

Source: Mllib:Spark's Machine Learning Library, A. Talwalkar

Recommendation: Collaborative Filtering



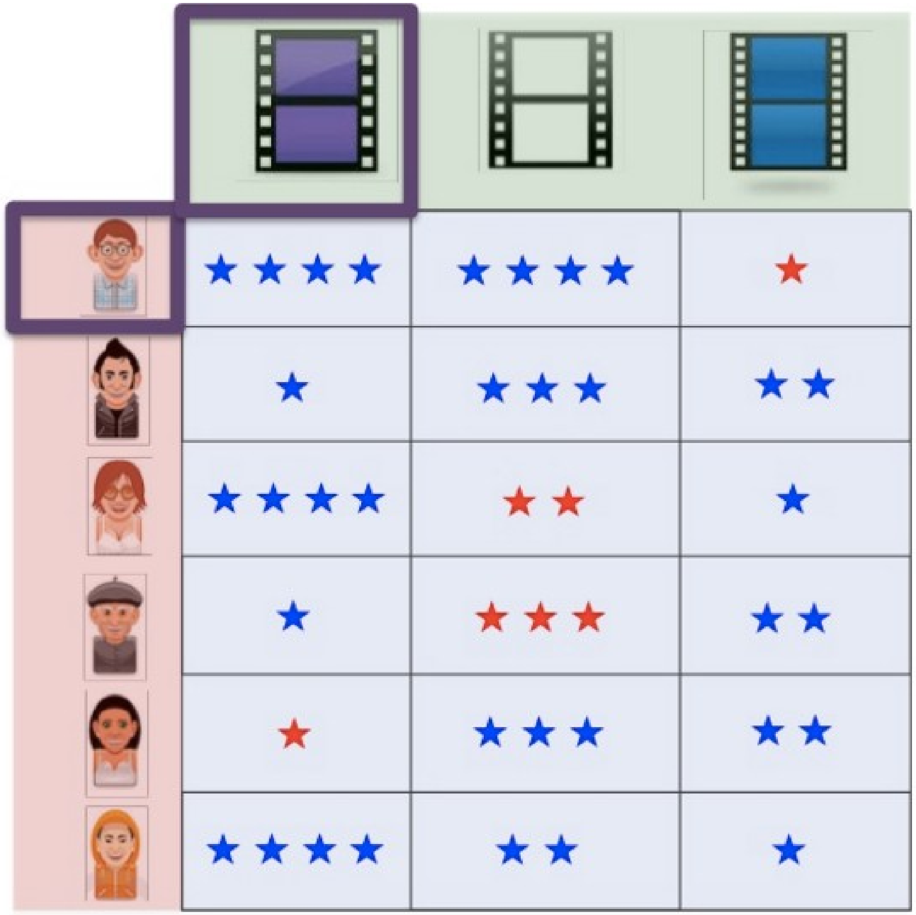
★★★★	★★★★	?
★	★★★	★★
★★★★		★
★		★★
	★★★	★★
★★★★	★★★	★

Goal: Recommend movies to users

NETFLIX

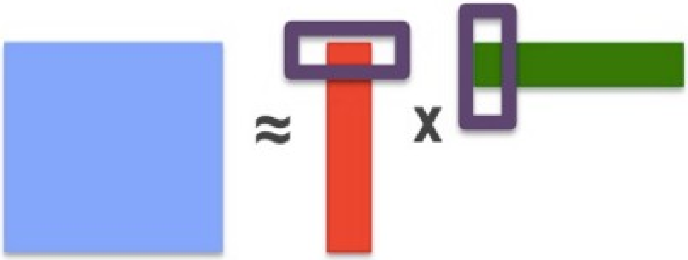
Source: Mllib:Spark's Machine Learning Library, A. Talwalkar

Recommendation



★★★★	★★★★	★
★	★★★	★★
★★★★	★★	★
★	★★★	★★
★	★★★	★★
★★★★	★★	★

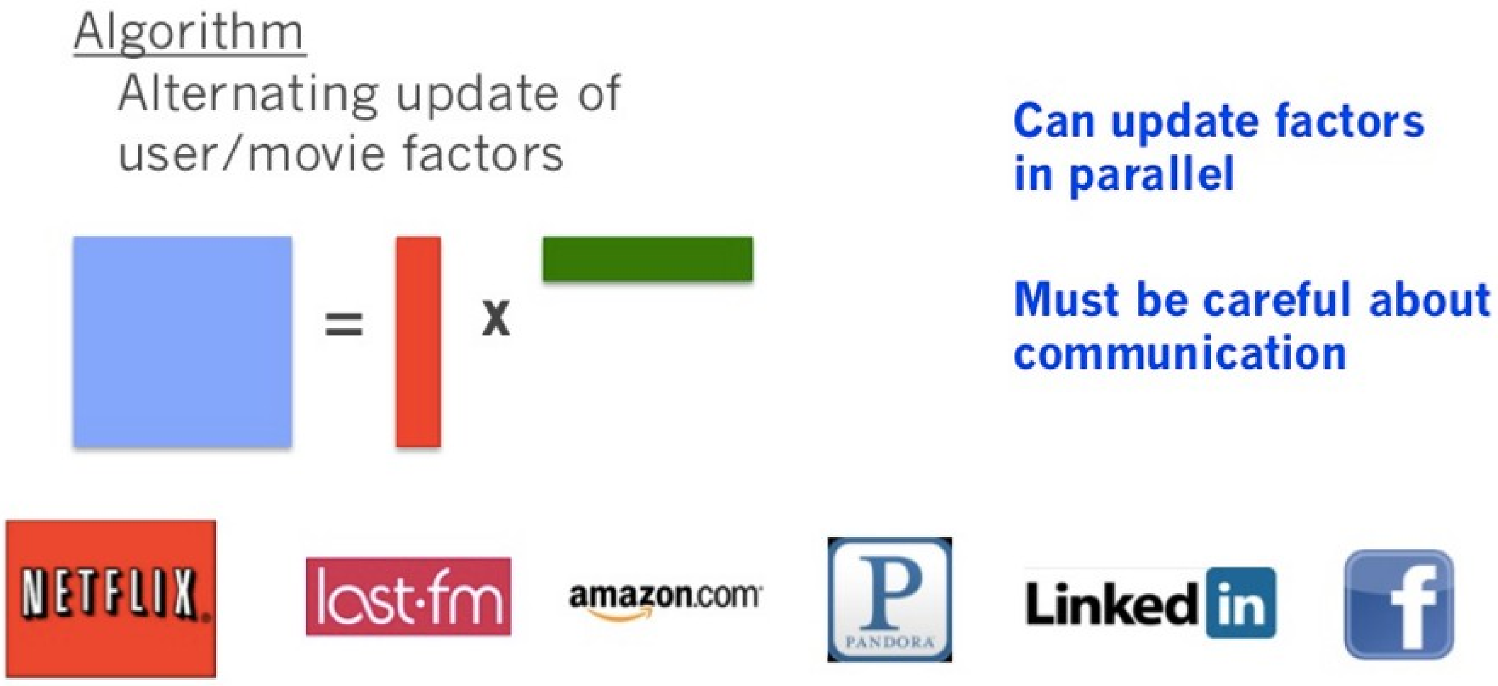
Solution: Assume ratings are determined by a small number of factors.



25M Users, 100K Movies
→ 2.5 trillion ratings
With 10 factors/user
→ 250M parameters

Source: Mllib:Spark's Machine Learning Library, A. Talwalkar

Recommendation: ALS



Algorithm

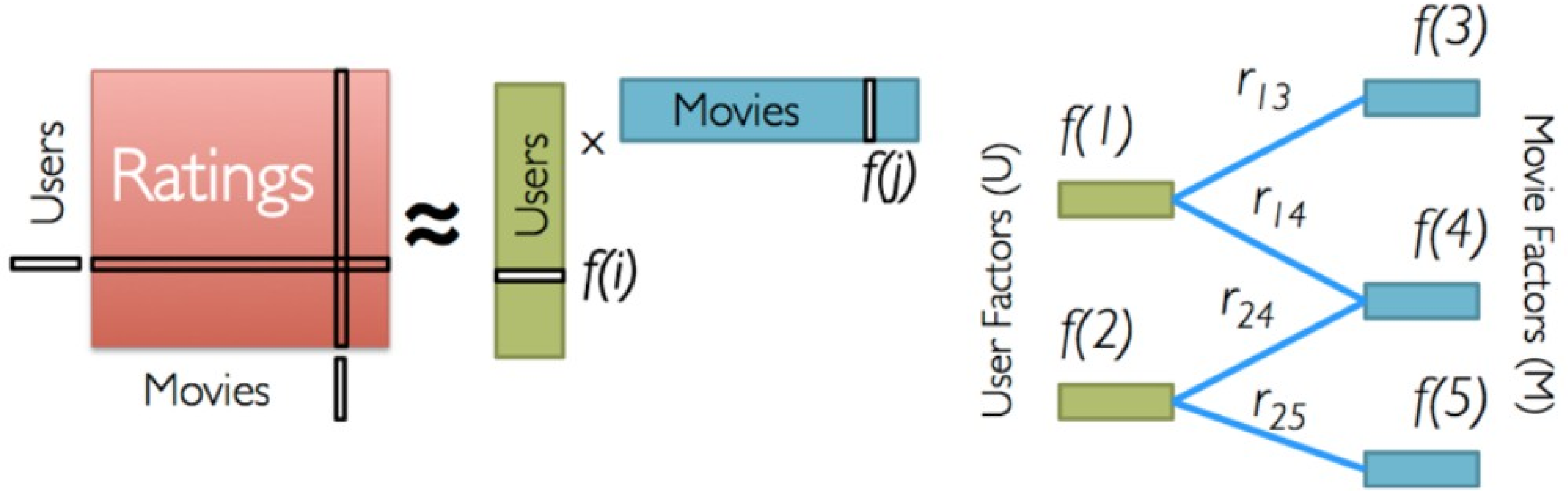
Alternating update of user/movie factors

Can update factors in parallel

Must be careful about communication

Source: Mllib:Spark's Machine Learning Library, A. Talwalkar

Alternating least squares (ALS)



Iterate:

$$f[i] = \arg\min_{w \in \mathbb{R}^d} \sum_{j \in \mathrm{Nbrs}(i)} \left(r_{ij} - w^T f[j]\right)^2 + \lambda ||w||_2^2$$

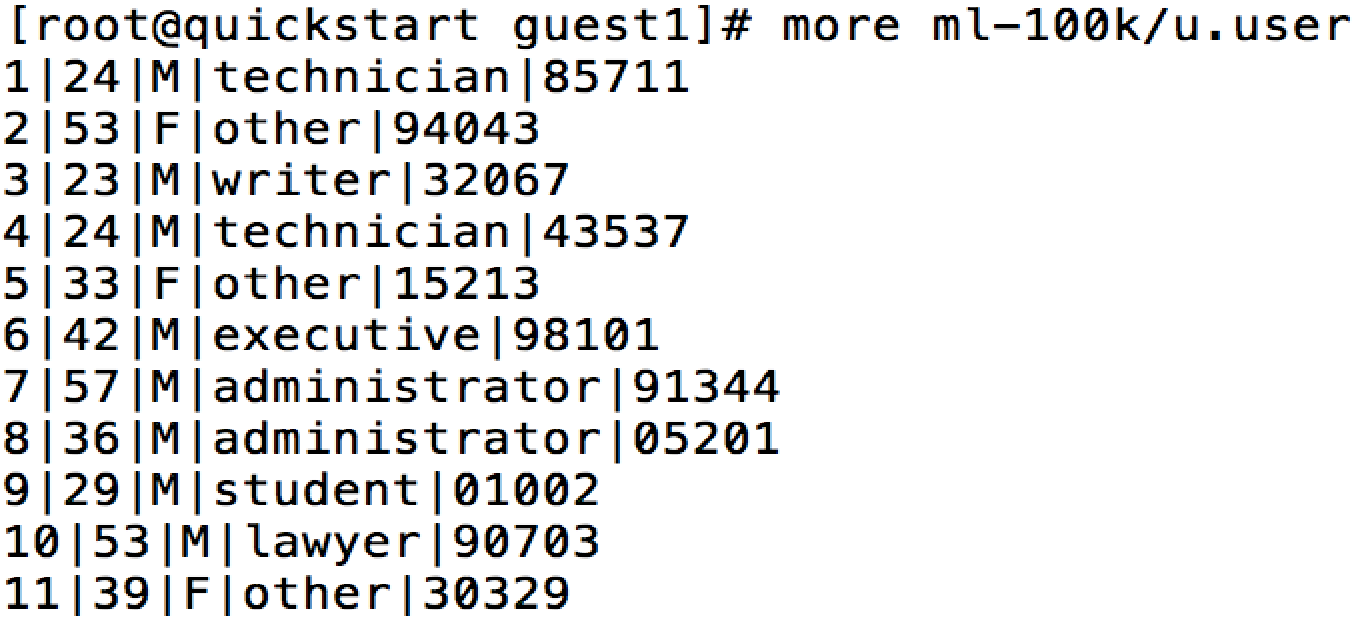
Source: MLlib: Scalable Machine Learning on Spark, X. Meng

MLlib: ALS Algorithm

- **numBlocks** is the number of blocks used to parallelize computation (set to -1 to autoconfigure)
- **rank** is the number of latent factors in the model
- **iterations** is the number of iterations to run
- **lambda** specifies the regularization parameter in ALS
- **implicitPrefs** specifies whether to use the explicit feedback ALS variant or one adapted for an implicit feedback data
- **alpha** is a parameter applicable to the implicit feedback variant of ALS that governs the baseline confidence in preference observations

MovieLen Dataset

1) Type command > `wget http://files.grouplens.org/datasets/movielens/ml-100k.zip`
2) Type command > `yum install unzip`
3) Type command > `unzip ml-100k.zip`
4) Type command > `more ml-100k/u.user`



```
[root@quickstart guest1]# more ml-100k/u.user
1|24|M|technician|85711
2|53|F|other|94043
3|23|M|writer|32067
4|24|M|technician|43537
5|33|F|other|15213
6|42|M|executive|98101
7|57|M|administrator|91344
8|36|M|administrator|05201
9|29|M|student|01002
10|53|M|lawyer|90703
11|39|F|other|30329
```

Moving dataset to HDFS

1) Type command > cd ml-100k

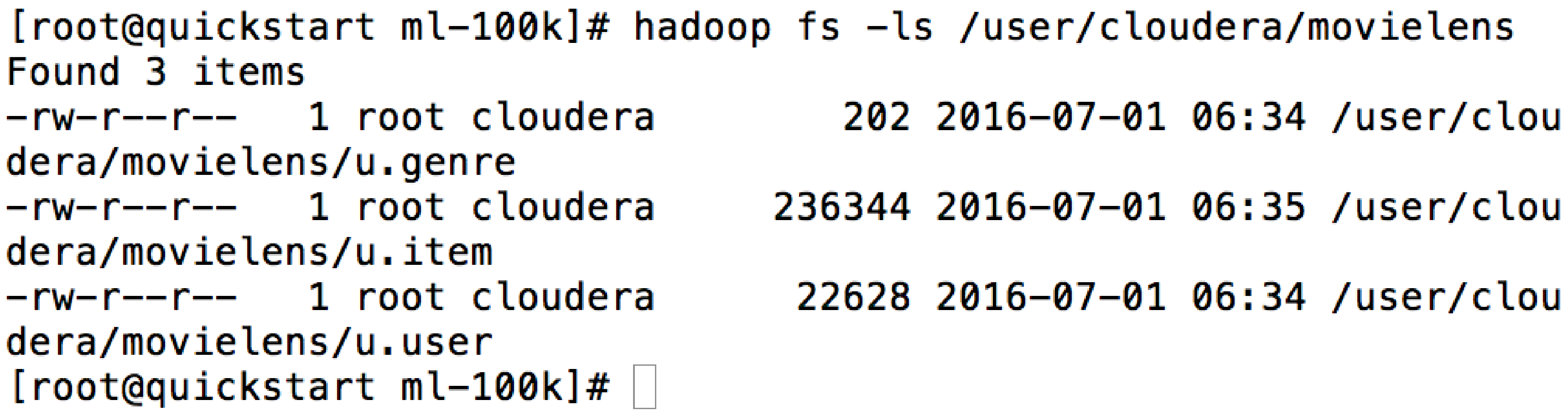
2) Type command > hadoop fs -mkdir /user/cloudera/movielens

3) Type command > hadoop fs -put u.user /user/cloudera/movielens
4) Type command > hadoop fs -put u.data /user/cloudera/movielens

4) Type command > hadoop fs -put u.genre /user/cloudera/movielens

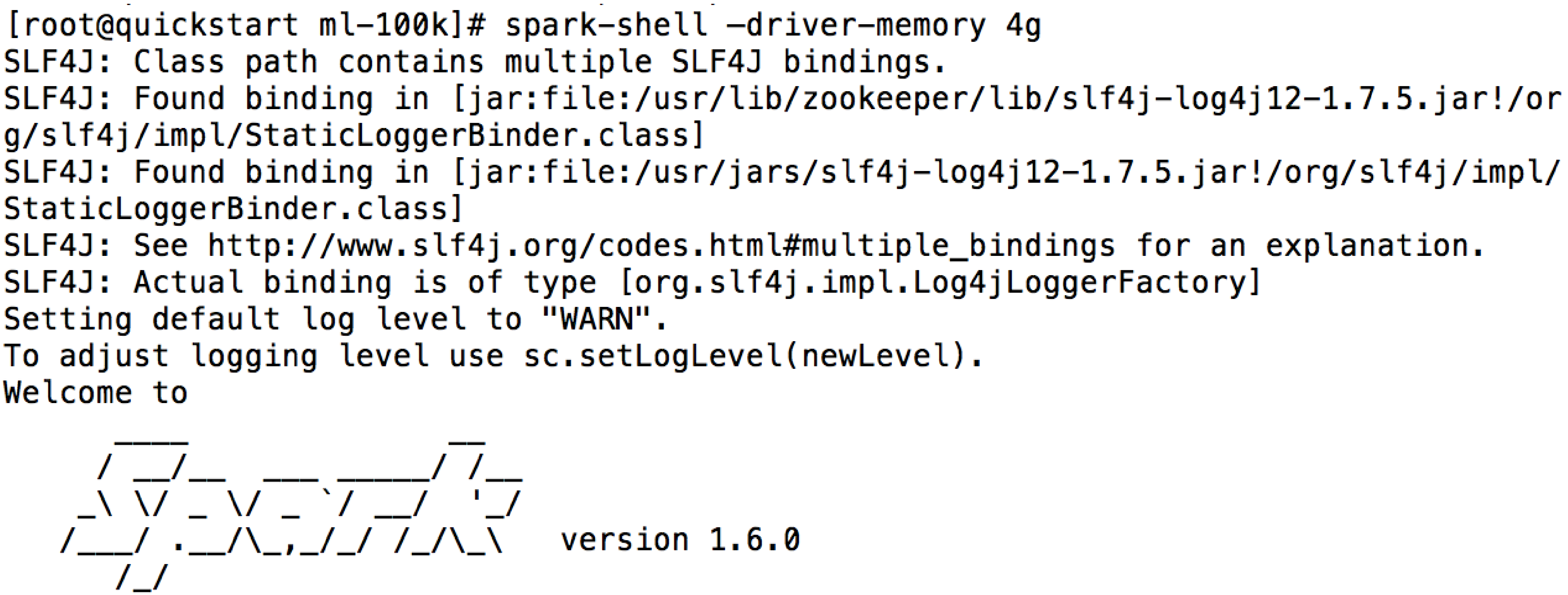
5) Type command > hadoop fs -put u.item /user/cloudera/movielens

6) Type command > hadoop fs -ls /user/cloudera/movielens



```
[root@quickstart ml-100k]# hadoop fs -ls /user/cloudera/movielens
Found 3 items
-rw-r--r--   1 root cloudera        202 2016-07-01 06:34 /user/clou
dera/movielens/u.genre
-rw-r--r--   1 root cloudera     236344 2016-07-01 06:35 /user/clou
dera/movielens/u.item
-rw-r--r--   1 root cloudera      22628 2016-07-01 06:34 /user/clou
dera/movielens/u.user
[root@quickstart ml-100k]# 
```

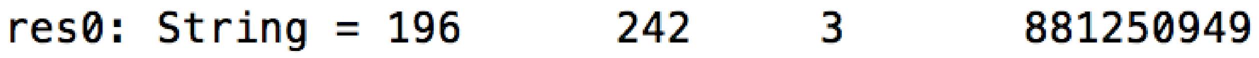
Start Spark-shell with extra memory



```
[root@quickstart ml-100k]# spark-shell -driver-memory 4g
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/lib/zookeeper/lib/slf4j-log4j12-1.7.5.jar!/or
g/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/jars/slf4j-log4j12-1.7.5.jar!/org/slf4j/impl/
StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel).
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 1.6.0
      /_/
```

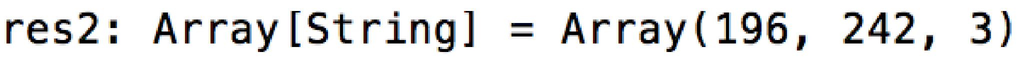
Extracting features from the MovieLens dataset

```
scala> val rawData =
sc.textFile("hdfs:///user/cloudera/movielens/u.data")

scala> rawData.first()
```



```
res0: String = 196      242     3       881250949
```

```
scala> val rawRatings = rawData.map(_.split("\t").take(3))

scala> rawRatings.first()
```



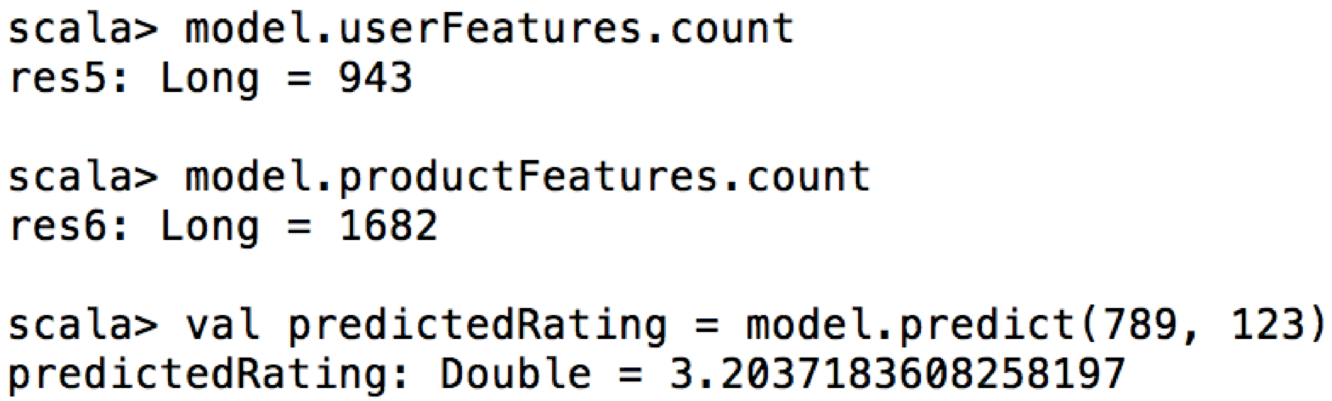
```
res2: Array[String] = Array(196, 242, 3)
```

```
scala> import org.apache.spark.mllib.recommendation.Rating

scala> val ratings = rawRatings.map { case Array(user, movie,
rating) =>Rating(user.toInt, movie.toInt, rating.toDouble) }
scala> ratings.first()
```

Training the recommendation model

```
scala> import org.apache.spark.mllib.recommendation.ALS
scala> val model = ALS.train(ratings, 50, 10, 0.01)
```

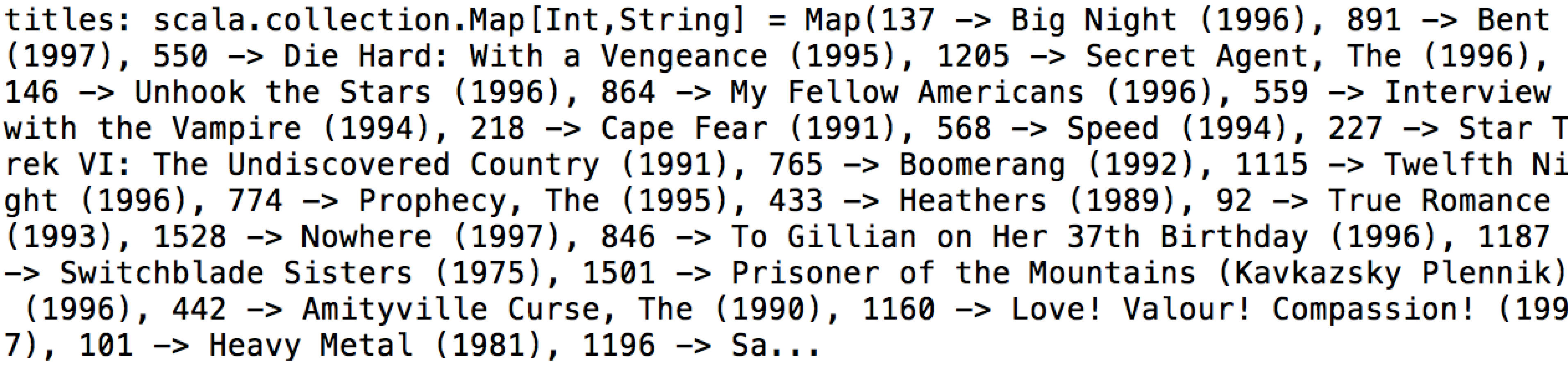
Note: We'll use rank of 50, 10 iterations, and a lambda parameter of 0.01



```
scala> model.userFeatures.count
res5: Long = 943

scala> model.productFeatures.count
res6: Long = 1682

scala> val predictedRating = model.predict(789, 123)
predictedRating: Double = 3.2037183608258197
```

Inspecting the recommendations

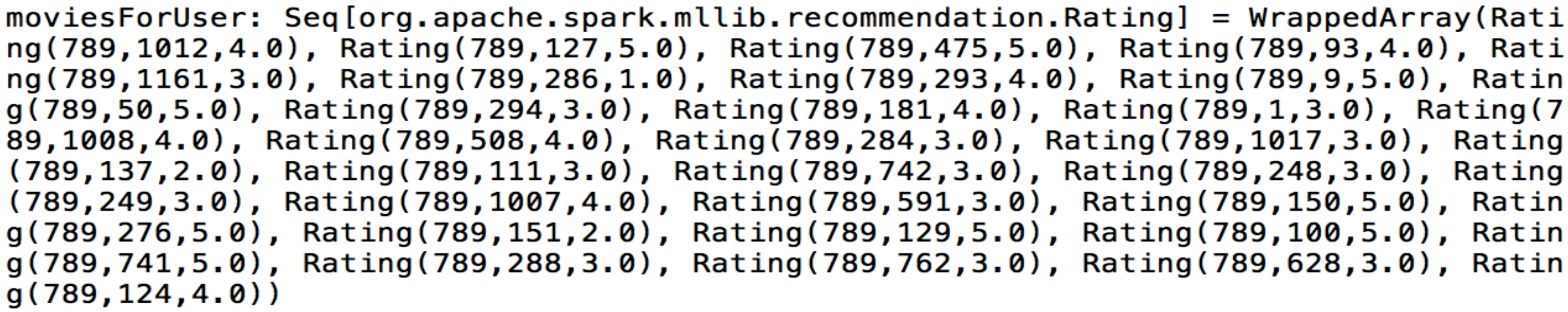
```
scala> val movies =
sc.textFile("hdfs:///user/cloudera/movielens/u.item")

scala> val titles = movies.map(line =>
line.split("\\|").take(2)).map(array
=>(array(0).toInt,array(1))).collectAsMap()
```



```
titles: scala.collection.Map[Int,String] = Map(137 -> Big Night (1996), 891 -> Bent
(1997), 550 -> Die Hard: With a Vengeance (1995), 1205 -> Secret Agent, The (1996),
146 -> Unhook the Stars (1996), 864 -> My Fellow Americans (1996), 559 -> Interview
with the Vampire (1994), 218 -> Cape Fear (1991), 568 -> Speed (1994), 227 -> Star T
rek VI: The Undiscovered Country (1991), 765 -> Boomerang (1992), 1115 -> Twelfth Ni
ght (1996), 774 -> Prophecy, The (1995), 433 -> Heathers (1989), 92 -> True Romance
(1993), 1528 -> Nowhere (1997), 846 -> To Gillian on Her 37th Birthday (1996), 1187
-> Switchblade Sisters (1975), 1501 -> Prisoner of the Mountains (Kavkazsky Plennik)
 (1996), 442 -> Amityville Curse, The (1990), 1160 -> Love! Valour! Compassion! (199
7), 101 -> Heavy Metal (1981), 1196 -> Sa...
```

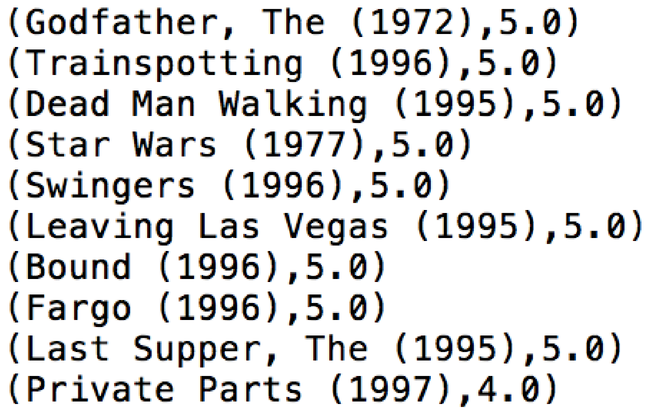
Inspecting the recommendations (cont.)

```
scala> val moviesForUser = ratings.keyBy(_.user).lookup(789)
```



```
moviesForUser: Seq[org.apache.spark.mllib.recommendation.Rating] = WrappedArray(Rati
ng(789,1012,4.0), Rating(789,127,5.0), Rating(789,475,5.0), Rating(789,93,4.0), Rati
ng(789,1161,3.0), Rating(789,286,1.0), Rating(789,293,4.0), Rating(789,9,5.0), Ratin
g(789,50,5.0), Rating(789,294,3.0), Rating(789,181,4.0), Rating(789,1,3.0), Rating(7
89,1008,4.0), Rating(789,508,4.0), Rating(789,284,3.0), Rating(789,1017,3.0), Rating
(789,137,2.0), Rating(789,111,3.0), Rating(789,742,3.0), Rating(789,248,3.0), Rating
(789,249,3.0), Rating(789,1007,4.0), Rating(789,591,3.0), Rating(789,150,5.0), Ratin
g(789,276,5.0), Rating(789,151,2.0), Rating(789,129,5.0), Rating(789,100,5.0), Ratin
g(789,741,5.0), Rating(789,288,3.0), Rating(789,762,3.0), Rating(789,628,3.0), Ratin
g(789,124,4.0))
```

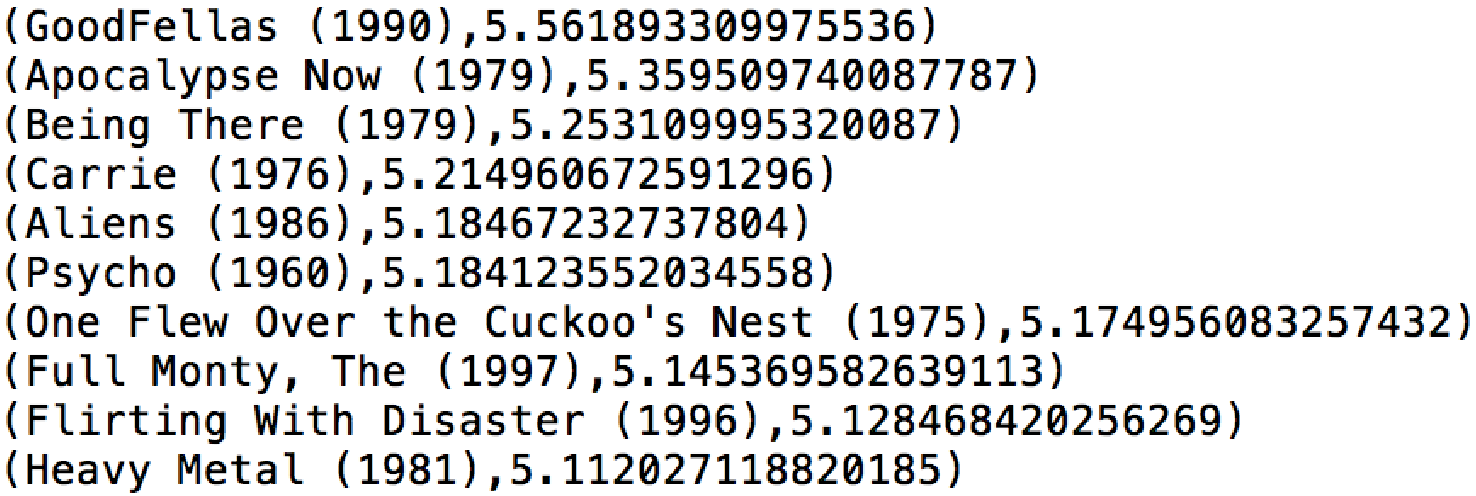
```
scala> moviesForUser.sortBy(-_.rating).take(10).map(rating =>
(titles(rating.product), rating.rating)).foreach(println)
```



```
(Godfather, The (1972),5.0)
(Trainspotting (1996),5.0)
(Dead Man Walking (1995),5.0)
(Star Wars (1977),5.0)
(Swingers (1996),5.0)
(Leaving Las Vegas (1995),5.0)
(Bound (1996),5.0)
(Fargo (1996),5.0)
(Last Supper, The (1995),5.0)
(Private Parts (1997),4.0)
```

Top 10 Recommendation for userid 789

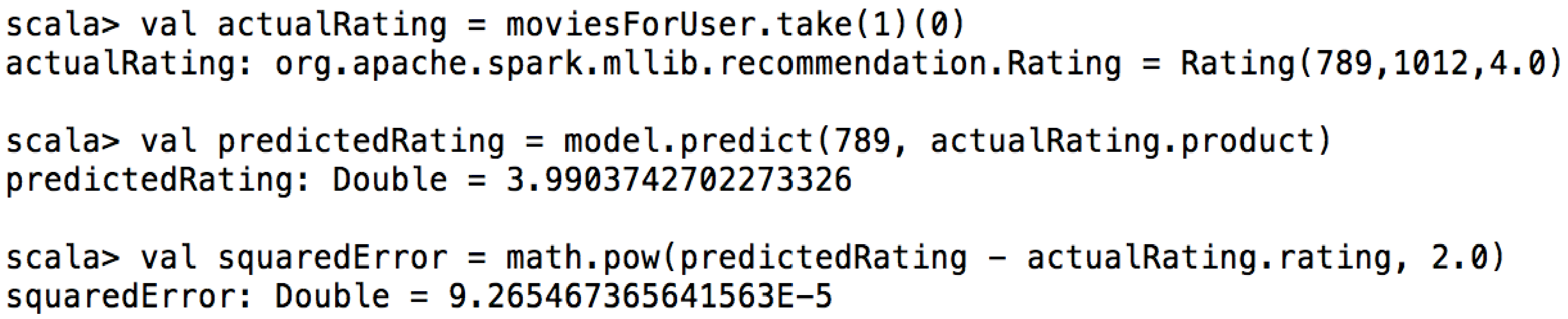
```
scala> val topKRecs = model.recommendProducts(789,10)

scala> topKRecs.map(rating => (titles(rating.product),
rating.rating)).foreach(println)
```



```
(GoodFellas (1990),5.561893309975536)
(Apocalypse Now (1979),5.359509740087787)
(Being There (1979),5.253109995320087)
(Carrie (1976),5.214960672591296)
(Aliens (1986),5.18467232737804)
(Psycho (1960),5.184123552034558)
(One Flew Over the Cuckoo's Nest (1975),5.174956083257432)
(Full Monty, The (1997),5.145369582639113)
(Flirting With Disaster (1996),5.128468420256269)
(Heavy Metal (1981),5.112027118820185)
```

Evaluating Performance: Mean Squared Error

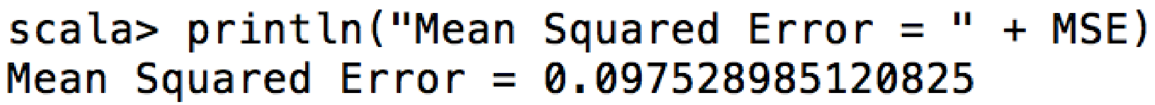
```
scala> val actualRating = moviesForUser.take(1)(0)

scala> val predictedRating = model.predict(789,
actualRating.product)

scala> val squaredError = math.pow(predictedRating -
actualRating.rating, 2.0)
```



```
scala> val actualRating = moviesForUser.take(1)(0)
actualRating: org.apache.spark.mllib.recommendation.Rating = Rating(789,1012,4.0)

scala> val predictedRating = model.predict(789, actualRating.product)
predictedRating: Double = 3.9903742702273326

scala> val squaredError = math.pow(predictedRating - actualRating.rating, 2.0)
squaredError: Double = 9.265467365641563E-5
```

Overall Mean Squared Error

```
scala> val usersProducts = ratings.map{ case Rating(user,
product, rating)  => (user, product)}

scala> val predictions = model.predict(usersProducts).map{

 case Rating(user, product, rating) => ((user, product),
rating)}

scala> val ratingsAndPredictions = ratings.map{

  case Rating(user, product, rating) => ((user, product),
rating)

}.join(predictions)

scala> val MSE = ratingsAndPredictions.map{

    case ((user, product), (actual, predicted)) =>
math.pow((actual - predicted), 2)

}.reduce(_ + _) / ratingsAndPredictions.count
```

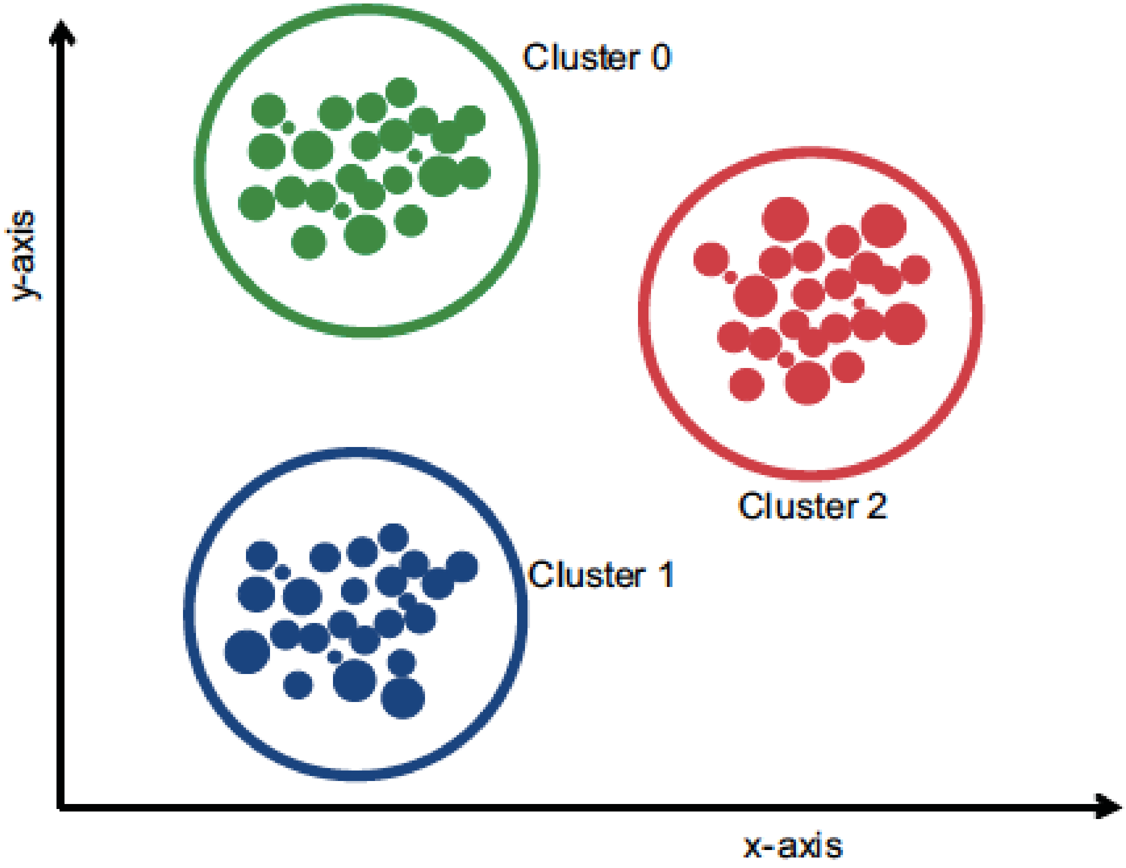


```
scala> println("Mean Squared Error = " + MSE)
Mean Squared Error = 0.097528985120825
```

Clustering using K-Means

Clustering use cases

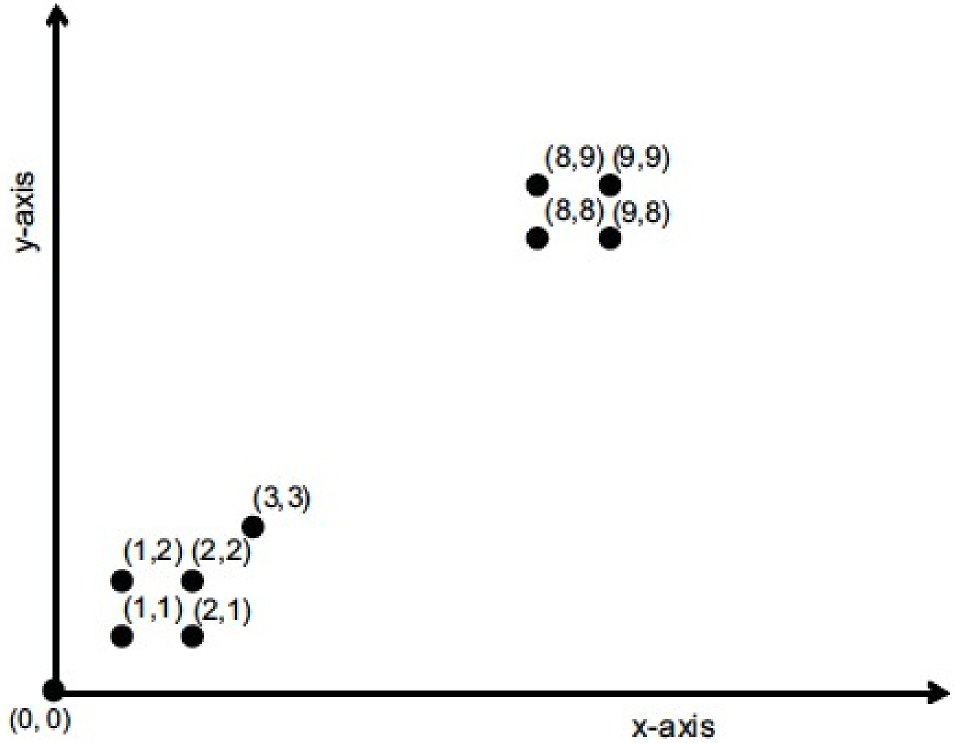
- Market segmentation
- Social network analysis: Finding a coherent group of people in the social network for ad targeting
- Data center computing clusters
- Real estate: Identifying neighborhoods based on similar features
- Text analysis: Dividing text documents, such as novels or essays, into genres



Source: Mahout in Action

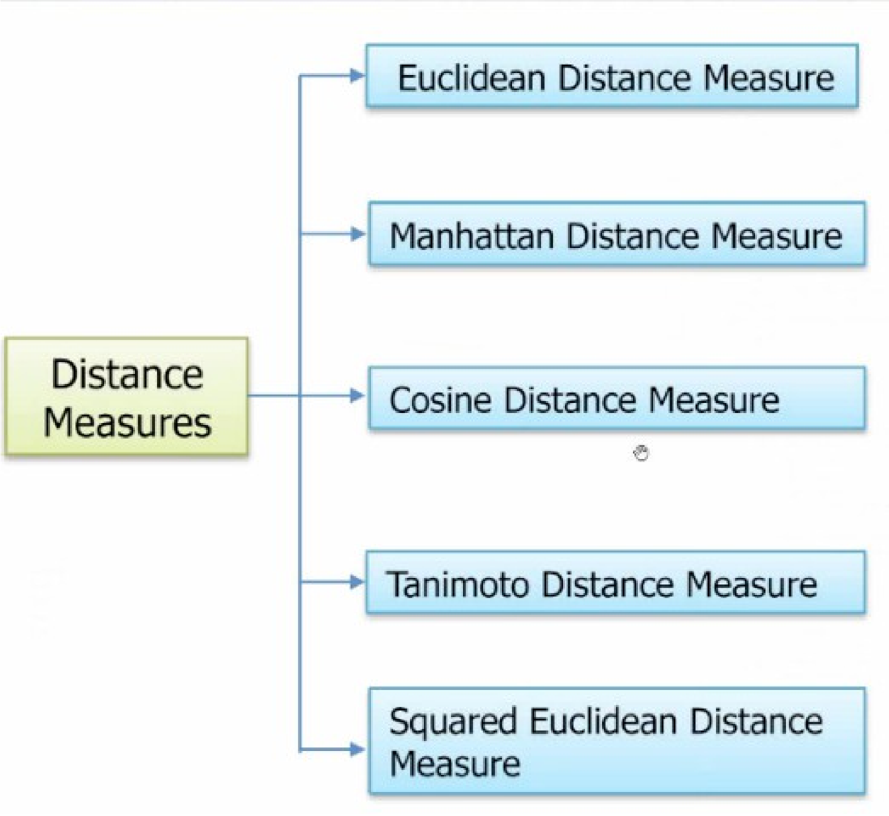
Sample Data

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



Source: Mahout in Action

Distance Measures


Distance Measures
Euclidean Distance Measure
Manhattan Distance Measure
Cosine Distance Measure
Tanimoto Distance Measure
Squared Euclidean Distance Measure

Source www.edureka.in/data-science

Distance Measures

- Euclidean distance measure

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

- Squared Euclidean distance measure

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

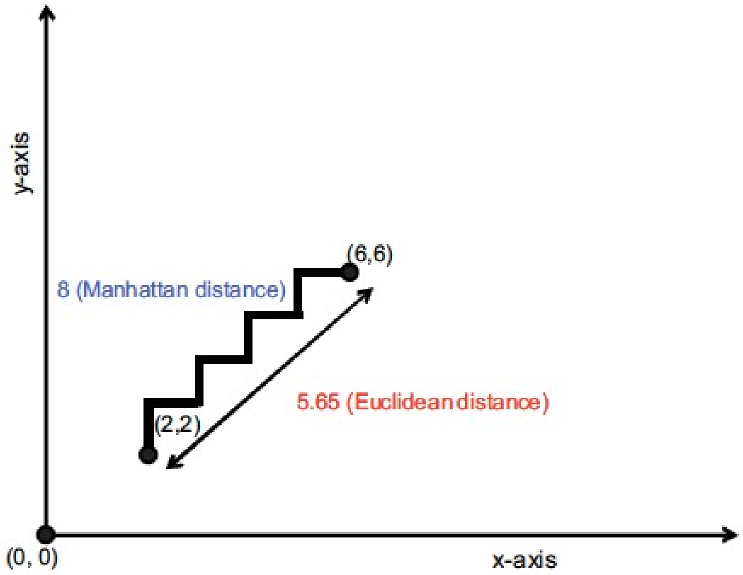
- Manhattan distance measure

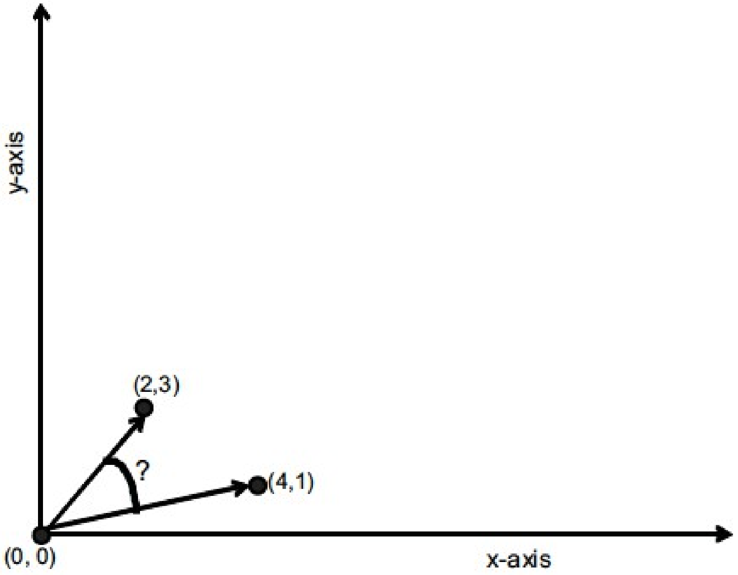
$$d = |a_1 - b_1| + |a_2 - b_2| + \dots + |a_n - b_n|$$

- Cosine distance measure

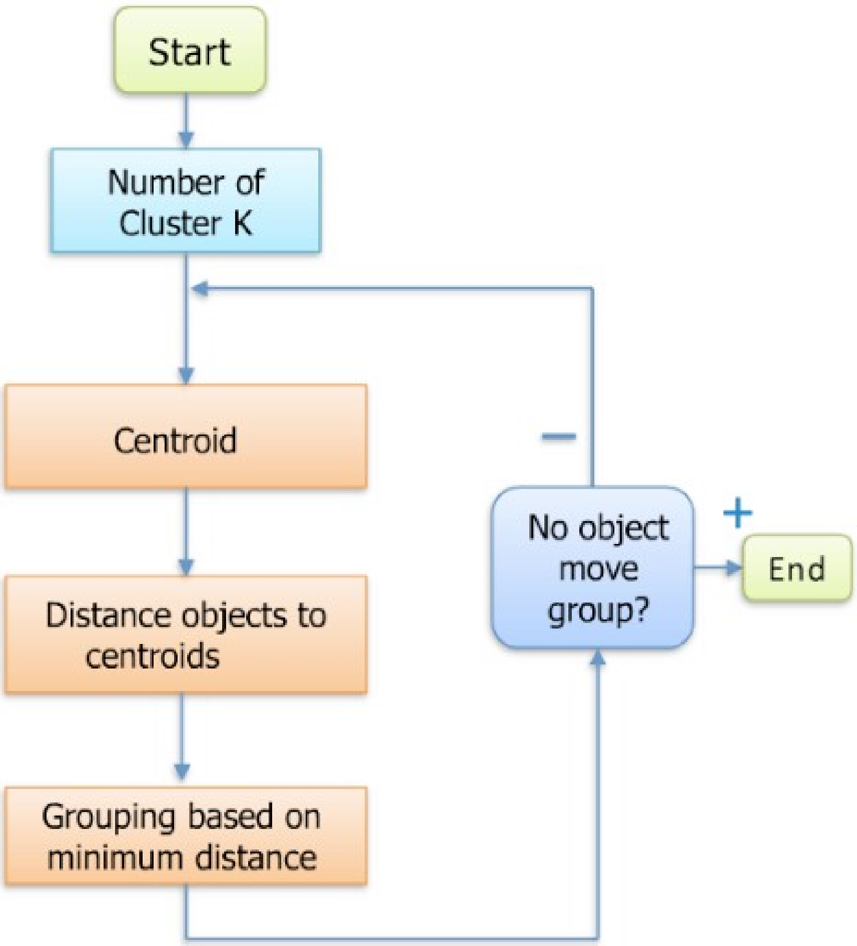
$$d = 1 - \frac{(a_1 b_1 + a_2 b_2 + \dots + a_n b_n)}{(\sqrt{(a_1^2 + a_2^2 + \dots + a_n^2)}\sqrt{(b_1^2 + b_2^2 + \dots + b_n^2)})}$$

Distance Measures



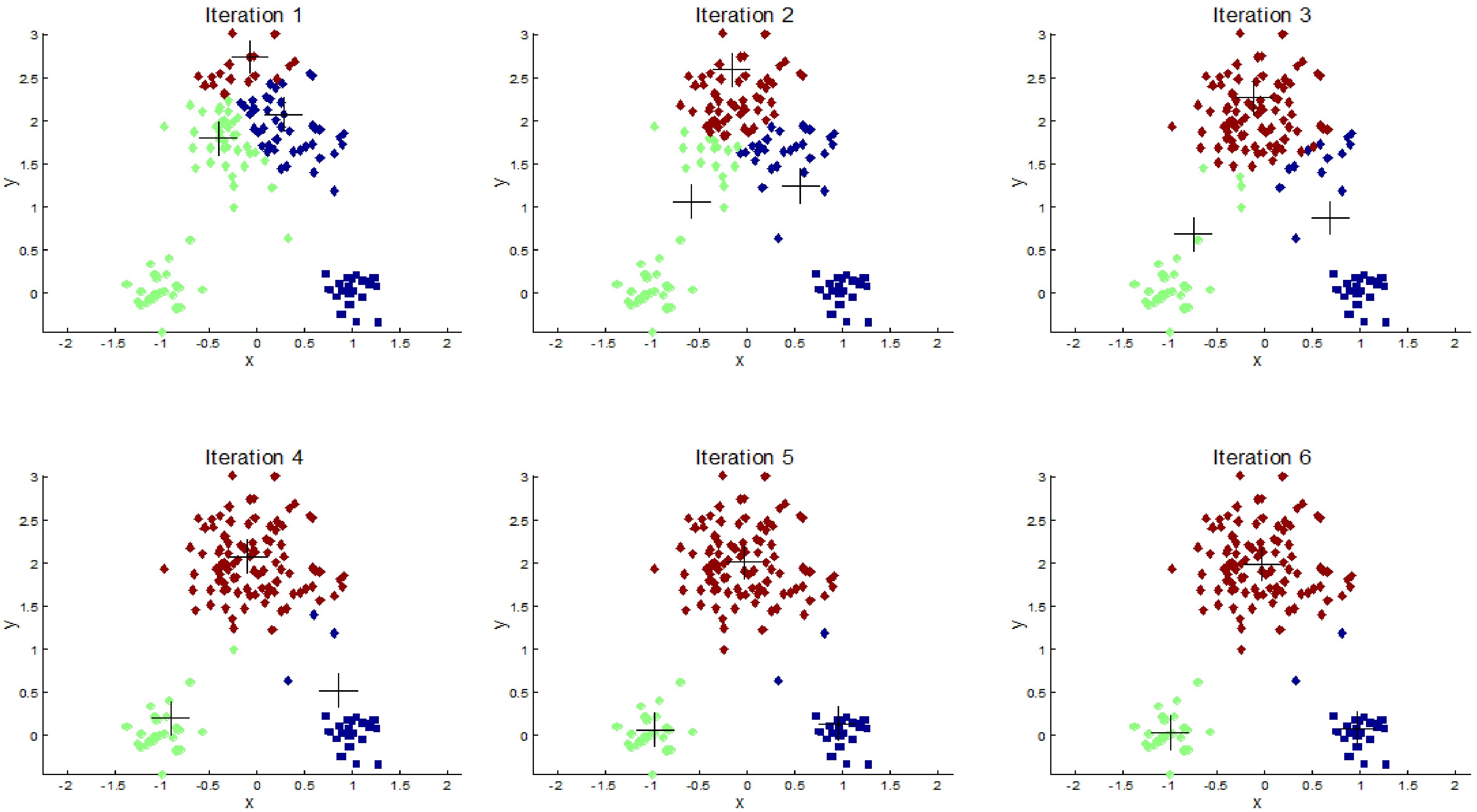


K-Means Clustering

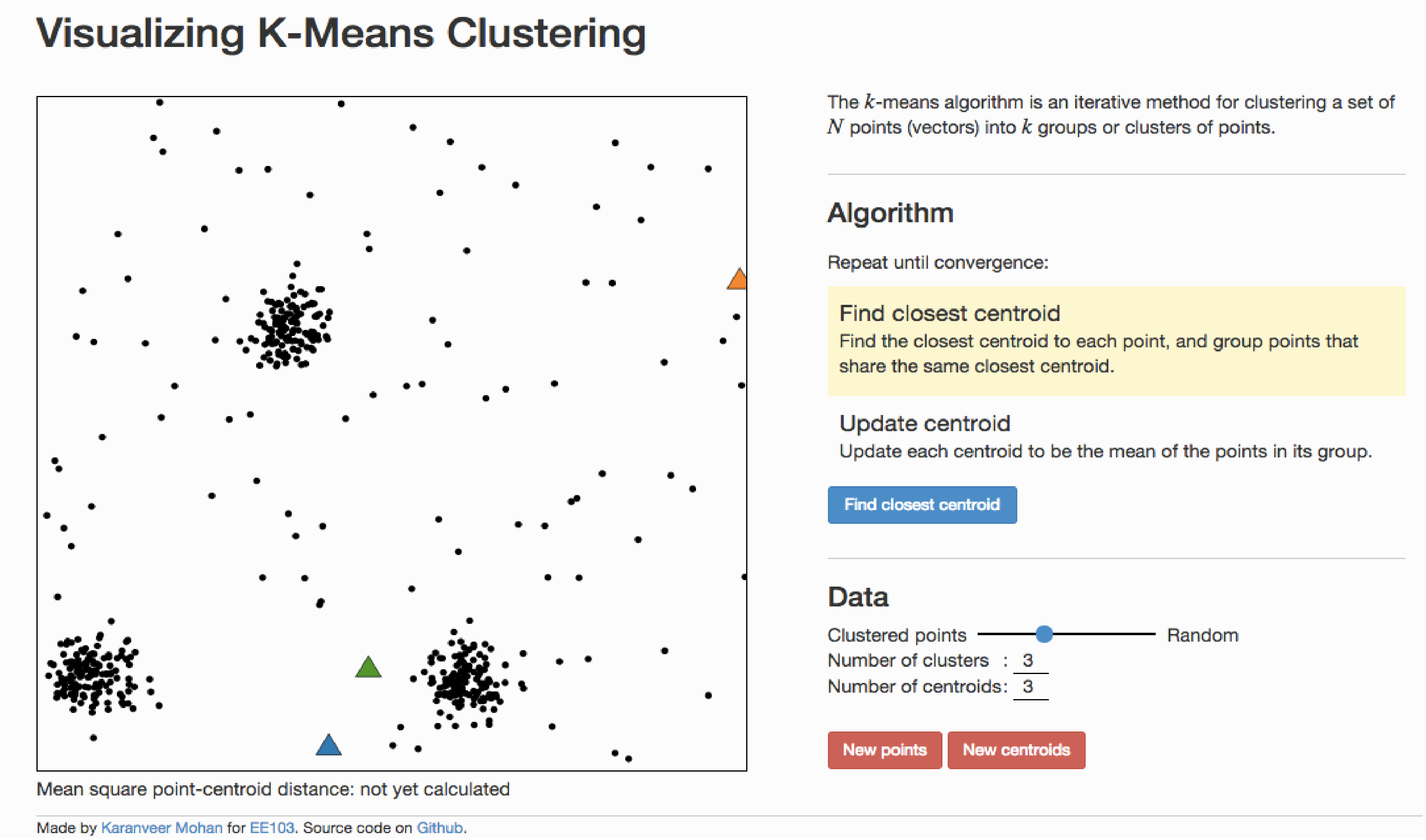

Start
Number of Cluster K
Centroid
Distance objects to centroids
Grouping based on minimum distance
No object move group?
−
+
End

Source: www.edureka.in/data-science

Example of K-Means Clustering



http://stanford.edu/class/ee103/visualizations/kmeans/kmeans.html



Visualizing K-Means Clustering

The k-means algorithm is an iterative method for clustering a set of N points (vectors) into k groups or clusters of points.

Algorithm

Repeat until convergence:

Find closest centroid

Find the closest centroid to each point, and group points that share the same closest centroid.

Update centroid

Update each centroid to be the mean of the points in its group.

Data

Clustered points ——●—— Random

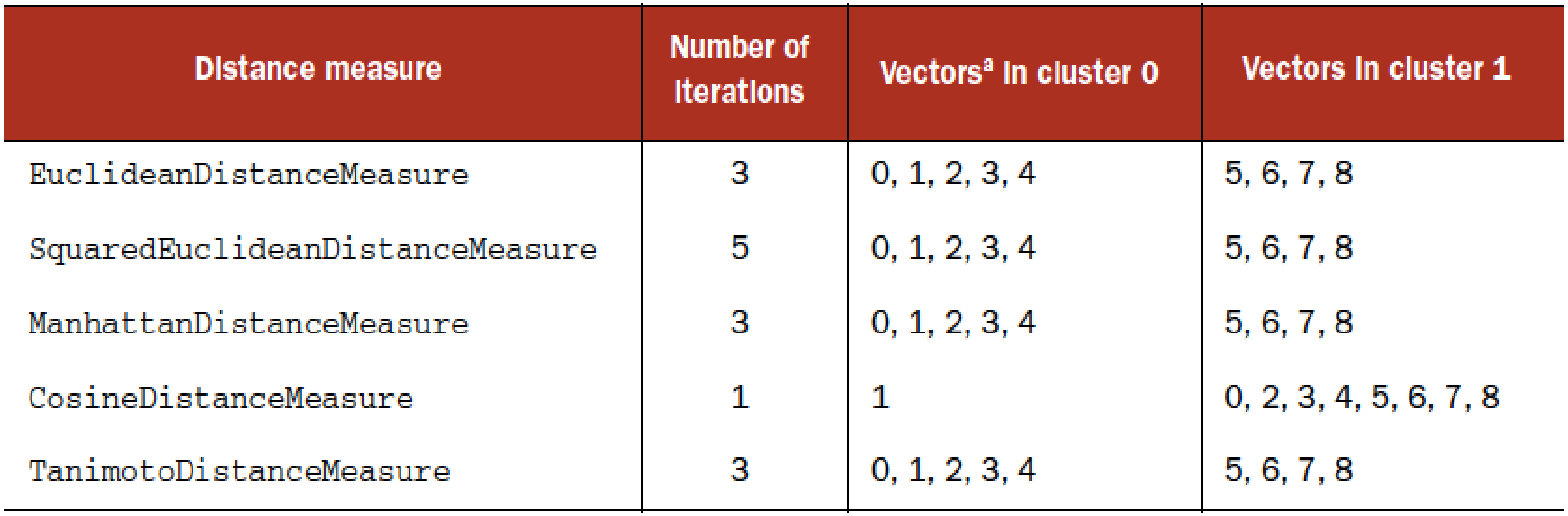
Number of clusters : 3

Number of centroids: 3

Mean square point-centroid distance: not yet calculated

Made by Karanveer Mohan for EE103. Source code on Github.

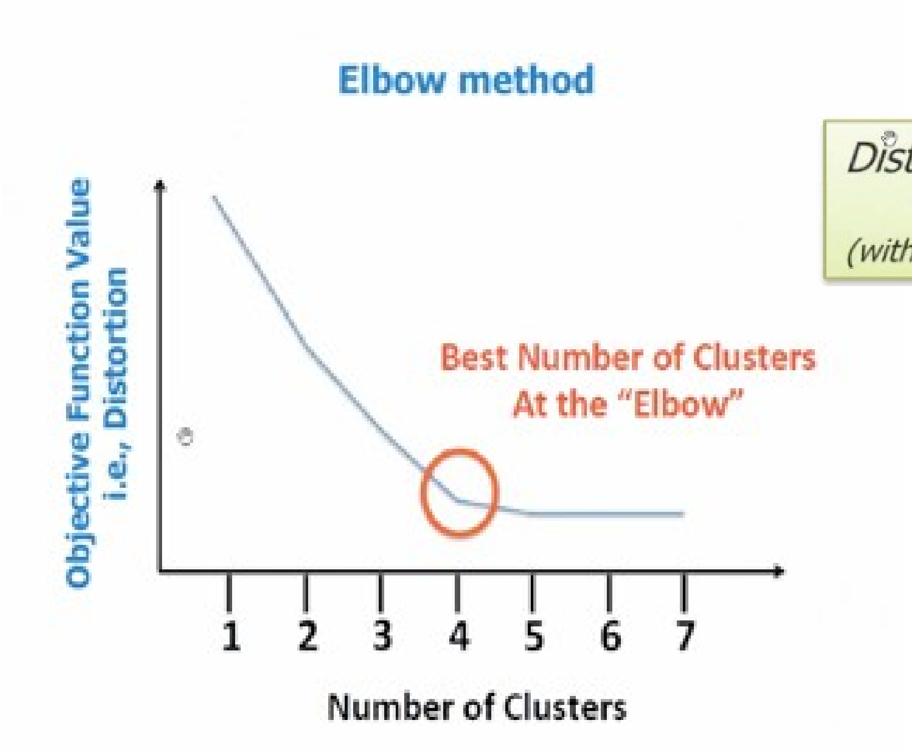
K-Means with different distance measures



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

Source: Mahout in Action

Choosing number of clusters



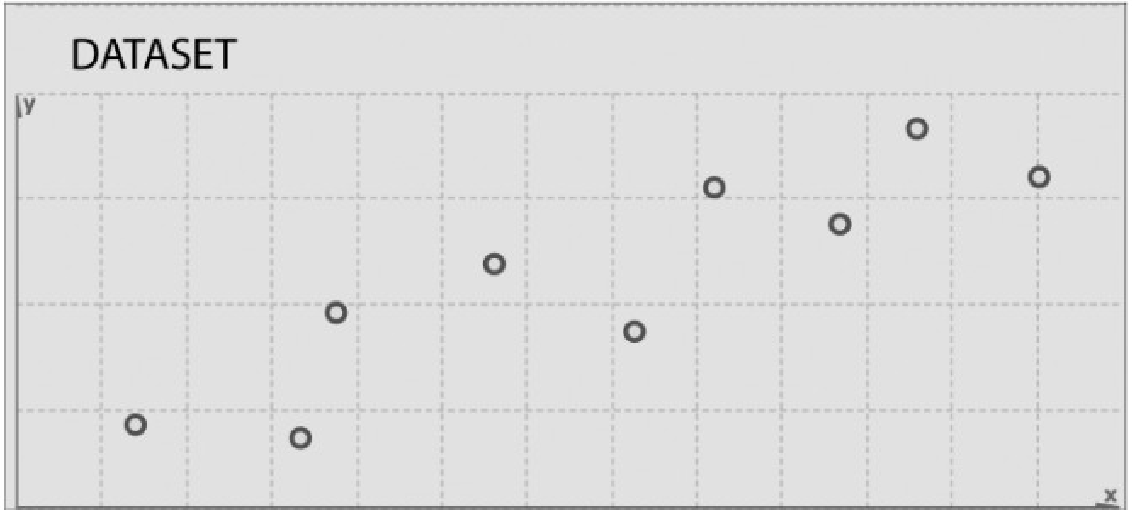
$$\textit{Distortion} = \sum_{i=1}^{m}(x_i - c_i)^2 = \sum_{j=1}^{k} \sum_{i \in OwnedBy(\mu_j)} (x_i - \mu_j)^2$$

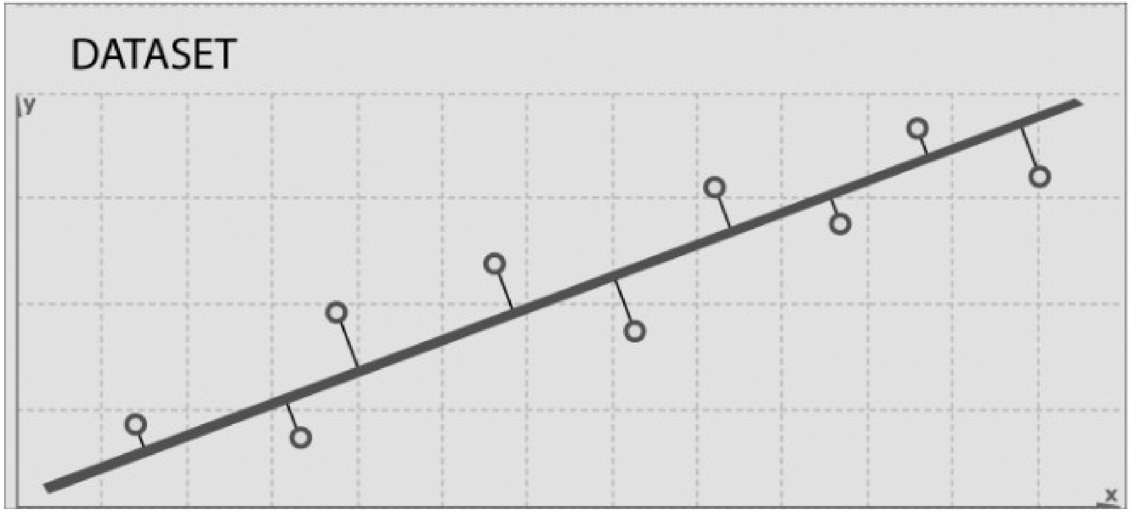
(within cluster sum of squares)

Dimensionality reduction

- Process of reducing the number of dimensions or features.
- Dimensionality reduction serves several purposes
  - Data compression
  - Visualization
- The most popular algorithm: Principal component analysis (PCA).

Dimensionality reduction





Source: Spark Cookbook

Dimensionality reduction with SVD

- Singular Value Decomposition (SVD): is based on a theorem from linear algebra that a rectangular matrix A can be broken down into a product of three matrices

$$A = USV^T$$

$$U^T U = 1$$

$$V^T V = 1$$

Dimensionality reduction with SVD

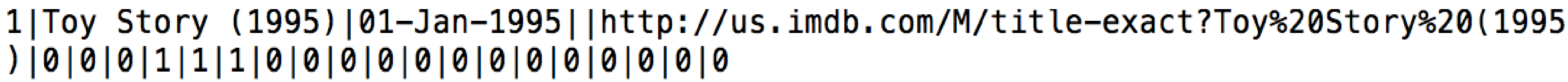
- The basic idea behind SVD
  - Take a high dimension, a highly variable set of data points
  - Reduce it to a lower dimensional space that exposes the structure of the original data more clearly and orders it from the most variation to the least.
- So we can simply ignore variation below a certain threshold to massively reduce the original data, making sure that the original relationship interests are retained.

Hands-on

Clustering on MovieLens Dataset

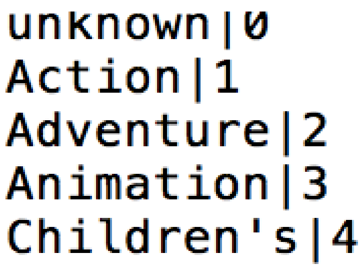
Extracting features from the MovieLens dataset

```
scala> val rawData =
sc.textFile("hdfs:///user/cloudera/movielens/u.item")

scala> println(movies.first)
```



```
1|Toy Story (1995)|01-Jan-1995||http://us.imdb.com/M/title-exact?Toy%20Story%20(1995
)|0|0|0|1|1|1|0|0|0|0|0|0|0|0|0|0|0|0|0
```

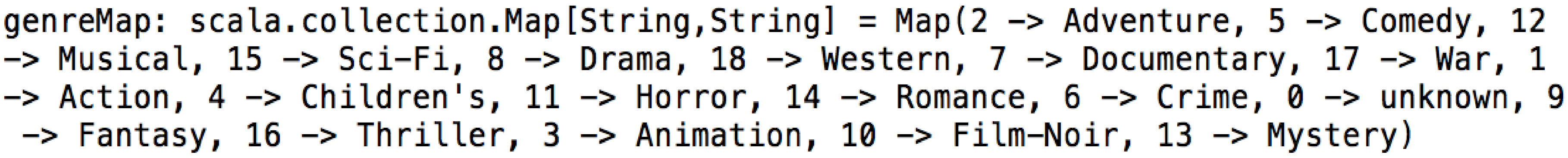
```
scala> val genres =
sc.textFile("hdfs:///user/cloudera/movielens/u.genre")

scala> genres.take(5).foreach(println)
```



```
unknown|0
Action|1
Adventure|2
Animation|3
Children's|4
```

Extracting features from the MovieLens dataset (cont.)

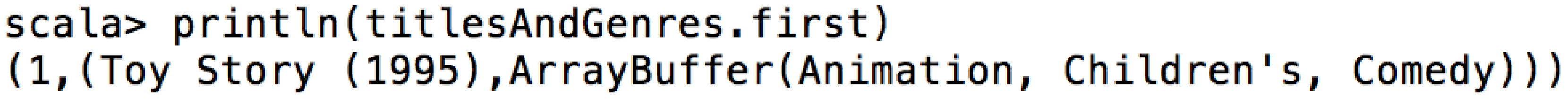
```
scala> val genreMap = genres.filter(!_.isEmpty).map(line =>
line.split("\\|")).map(array=> (array(1),
array(0))).collectAsMap
```



```
genreMap: scala.collection.Map[String,String] = Map(2 -> Adventure, 5 -> Comedy, 12
-> Musical, 15 -> Sci-Fi, 8 -> Drama, 18 -> Western, 7 -> Documentary, 17 -> War, 1
-> Action, 4 -> Children's, 11 -> Horror, 14 -> Romance, 6 -> Crime, 0 -> unknown, 9
 -> Fantasy, 16 -> Thriller, 3 -> Animation, 10 -> Film-Noir, 13 -> Mystery)
```

Extracting features from the MovieLens dataset (cont.)

```
scala> val titlesAndGenres = movies.map(_.split("\\|")).map
{ array =>
  val genres = array.toSeq.slice(5, array.size)
  val genresAssigned = genres.zipWithIndex.filter { case (g,
idx) =>
    g == "1"
  }.map { case (g, idx) =>
    genreMap(idx.toString)
  }
  (array(0).toInt, (array(1), genresAssigned))
}
```



```
scala> println(titlesAndGenres.first)
(1,(Toy Story (1995),ArrayBuffer(Animation, Children's, Comedy)))
```

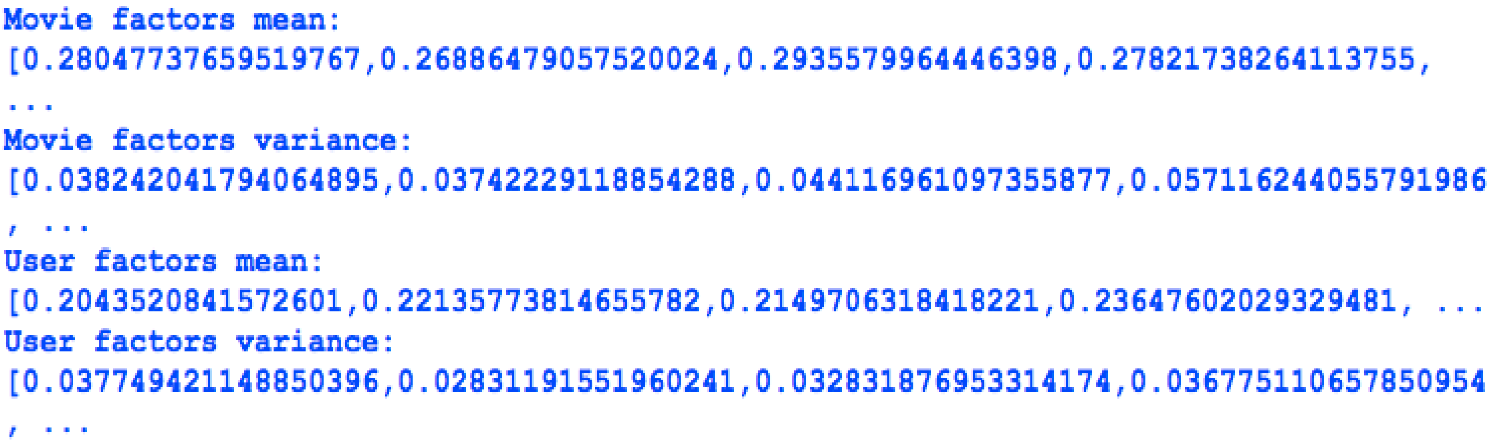
Training the recommendation model

```
scala> :paste
import org.apache.spark.mllib.recommendation.ALS
import org.apache.spark.mllib.recommendation.Rating
val rawData =
sc.textFile("hdfs:///user/cloudera/movielens/u.data")
val rawRatings = rawData.map(_.split("\t").take(3))
val ratings = rawRatings.map{ case Array(user, movie,
rating) => Rating(user.toInt, movie.toInt,
rating.toDouble) }
ratings.cache
val alsModel = ALS.train(ratings, 50, 10, 0.1)
import org.apache.spark.mllib.linalg.Vectors
val movieFactors = alsModel.productFeatures.map { case (id,
factor) => (id, Vectors.dense(factor)) }
val movieVectors = movieFactors.map(_._2)
val userFactors = alsModel.userFeatures.map { case (id,
factor) => (id, Vectors.dense(factor)) }
val userVectors = userFactors.map(_._2)
```

Normalization

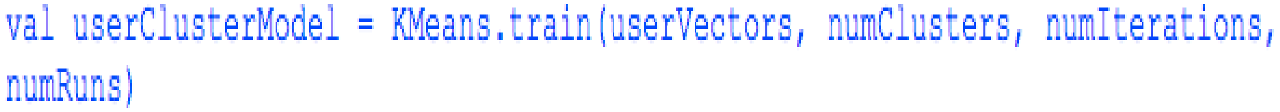
```
scala> :paste
import org.apache.spark.mllib.linalg.distributed.RowMatrix
val movieMatrix = new RowMatrix(movieVectors)
val movieMatrixSummary =
movieMatrix.computeColumnSummaryStatistics()
val userMatrix = new RowMatrix(userVectors)
val userMatrixSummary =
userMatrix.computeColumnSummaryStatistics()
println("Movie factors mean: " + movieMatrixSummary.mean)
println("Movie factors variance: " +
movieMatrixSummary.variance)
println("User factors mean: " + userMatrixSummary.mean)
println("User factors variance: " +
userMatrixSummary.variance)
```

Output from Normalization



```
Movie factors mean:
[0.28047737659519767,0.26886479057520024,0.2935579964446398,0.27821738264113755,
...
Movie factors variance:
[0.038242041794064895,0.03742229118854288,0.04411696109735877,0.057116244055791986
, ...
User factors mean:
[0.2043520841572601,0.22135773814655782,0.2149706318418221,0.23647602029329481, ...
User factors variance:
[0.037749421148850396,0.02831191551960241,0.032831876953314174,0.036775110657850954
, ...
```

Training a clustering model

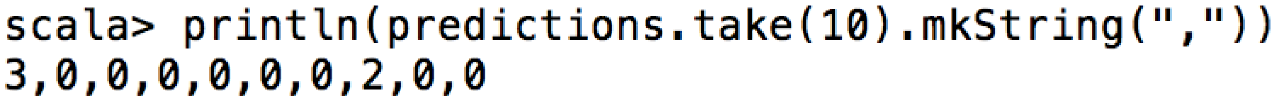


```
val userClusterModel = KMeans.train(userVectors, numClusters, numIterations,
numRuns)
```

```
scala> import org.apache.spark.mllib.clustering.KMeans
scala> val numClusters = 5
scala> val numIterations = 10
scala> val numRuns = 3
scala> val movieClusterModel = KMeans.train(movieVectors,
numClusters, numIterations, numRuns)
```

Making predictions using a clustering model

```
scala> val movie1 = movieVectors.first

scala> val movieCluster = movieClusterModel.predict(movie1)

scala> val predictions =
movieClusterModel.predict(movieVectors)
```



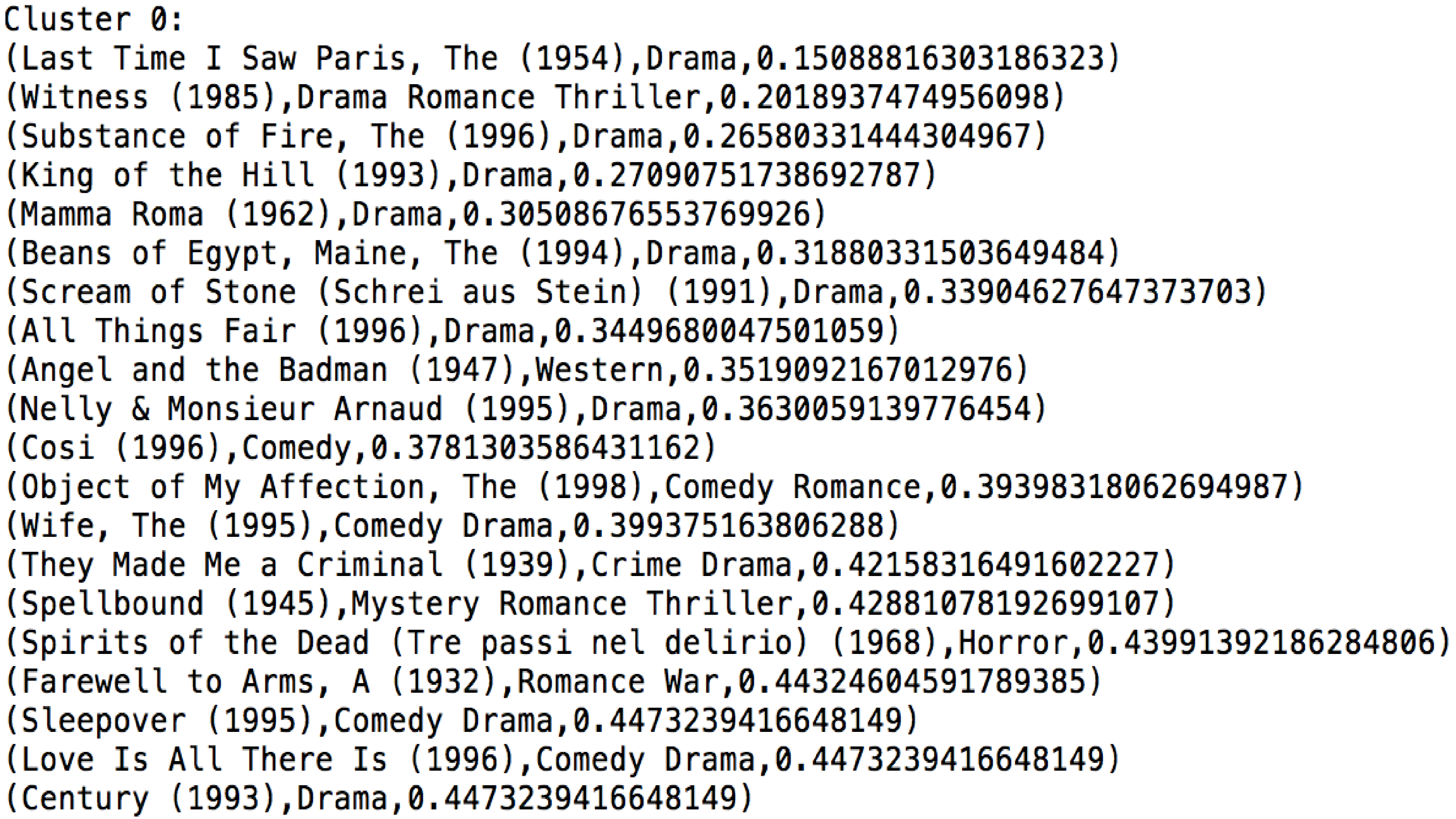
```
scala> println(predictions.take(10).mkString(","))
3,0,0,0,0,0,0,2,0,0
```

Interpreting cluster predictions

```
scala> :paste

import breeze.linalg._

import breeze.numerics.pow

def computeDistance(v1: DenseVector[Double], v2: 
DenseVector[Double]) = pow(v1 - v2, 2).sum

val titlesWithFactors = titlesAndGenres.join(movieFactors)

val moviesAssigned = titlesWithFactors.map { case (id, 
((title, genres), vector)) =>

  val pred = movieClusterModel.predict(vector)

  val clusterCentre = movieClusterModel.clusterCenters(pred)

  val dist = 
computeDistance(DenseVector(clusterCentre.toArray), 
DenseVector(vector.toArray))

   (id, title, genres.mkString(" "), pred, dist)

}
```

Interpreting cluster predictions (cont.)

```
val clusterAssignments = moviesAssigned.groupBy { case (id,
title, genres, cluster, dist) => cluster }.collectAsMap

for ( (k, v) <- clusterAssignments.toSeq.sortBy(_._1)) {

  println(s"Cluster $k:")

  val m = v.toSeq.sortBy(_._5)

  println(m.take(20).map { case (_, title, genres, _, d) =>
(title, genres, d) }.mkString("\n"))

  println("=====\n")

}
```



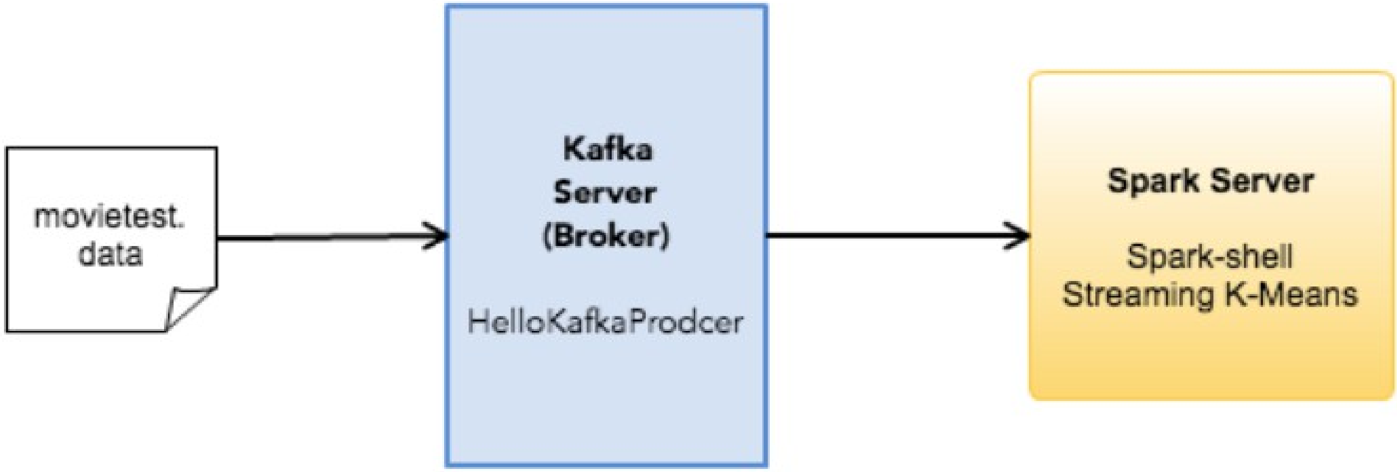
```
Cluster 0:
(Last Time I Saw Paris, The (1954),Drama,0.1508816303186323)
(Witness (1985),Drama Romance Thriller,0.2018937474956098)
(Substance of Fire, The (1996),Drama,0.2658031444304967)
(King of the Hill (1993),Drama,0.2709075173869278 7)
(Mamma Roma (1962),Drama,0.3050867655376992 6)
(Beans of Egypt, Maine, The (1994),Drama,0.3188033150364948 4)
(Scream of Stone (Schrei aus Stein) (1991),Drama,0.3390462764737370 3)
(All Things Fair (1996),Drama,0.3449680047501059)
(Angel and the Badman (1947),Western,0.3519092167012976)
(Nelly & Monsieur Arnaud (1995),Drama,0.3630059139776454)
(Cosi (1996),Comedy,0.3781303586431162)
(Object of My Affection, The (1998),Comedy Romance,0.3939831806269498 7)
(Wife, The (1995),Comedy Drama,0.399375163806288)
(They Made Me a Criminal (1939),Crime Drama,0.4215831649160222 7)
(Spellbound (1945),Mystery Romance Thriller,0.4288107819269910 7)
(Spirits of the Dead (Tre passi nel delirio) (1968),Horror,0.4399139218628480 6)
(Farewell to Arms, A (1932),Romance War,0.4432460459178938 5)
(Sleepover (1995),Comedy Drama,0.447323941664814 9)
(Love Is All There Is (1996),Comedy Drama,0.447323941664814 9)
(Century (1993),Drama,0.447323941664814 9)
```

Real-time Machine Learning using Streaming K-Means

Online learning with Spark Streaming

- **Streaming regression**
  - trainOn: This takes DStream[LabeledPoint] as its argument.
  - predictOn: This also takes DStream[LabeledPoint].
- **Streaming KMeans**
  - An extension of the mini-batch K-means algorithm

Streaming K-Means Program



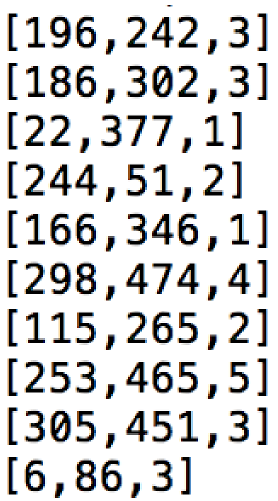
MovieLen Training Dataset

- The rows of the training text files must be vector data in the form

  [x1,x2,x3,...,xn]

1) Type command > wget https://s3.amazonaws.com/imcbucket/data/movietest.data

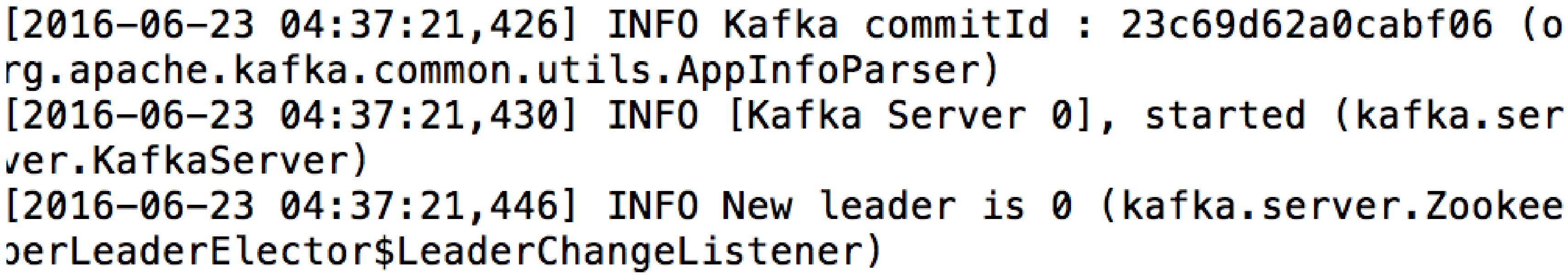
2) Type command > more movietest.data



```
[196,242,3]
[186,302,3]
[22,377,1]
[244,51,2]
[166,346,1]
[298,474,4]
[115,265,2]
[253,465,5]
[305,451,3]
[6,86,3]
```

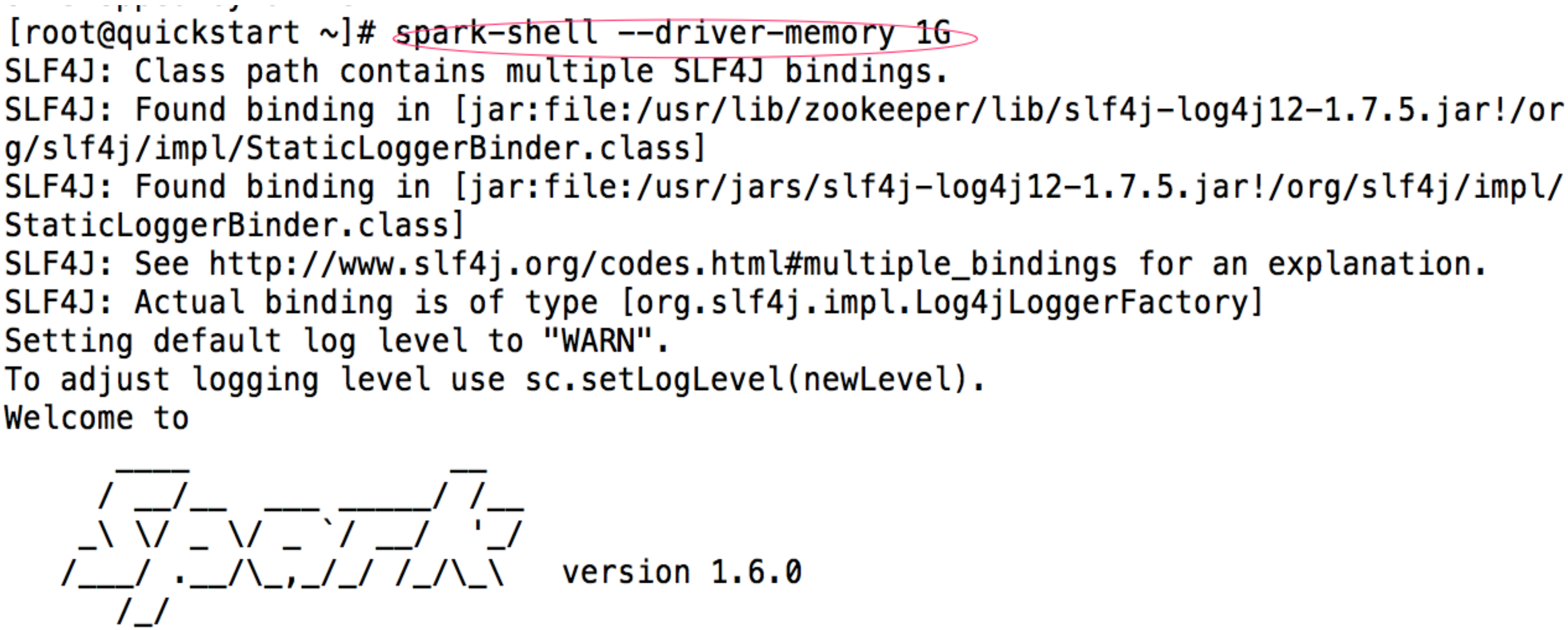
Install & Start Kafka Server

```
# wget http://www-us.apache.org/dist/kafka/0.9.0.1/kafka_2.10-
0.9.0.1.tgz

# tar xzf kafka_2.10-0.9.0.1.tgz

# cd kafka_2.10-0.9.0.1

# bin/kafka-server-start.sh config/server.properties&
```



```
[2016-06-23 04:37:21,426] INFO Kafka commitId : 23c69d62a0cabf06 (o
rg.apache.kafka.common.utils.AppInfoParser)
[2016-06-23 04:37:21,430] INFO [Kafka Server 0], started (kafka.ser
ver.KafkaServer)
[2016-06-23 04:37:21,446] INFO New leader is 0 (kafka.server.Zookee
perLeaderElector$LeaderChangeListener)
```

Start Spark-shell with extra memory



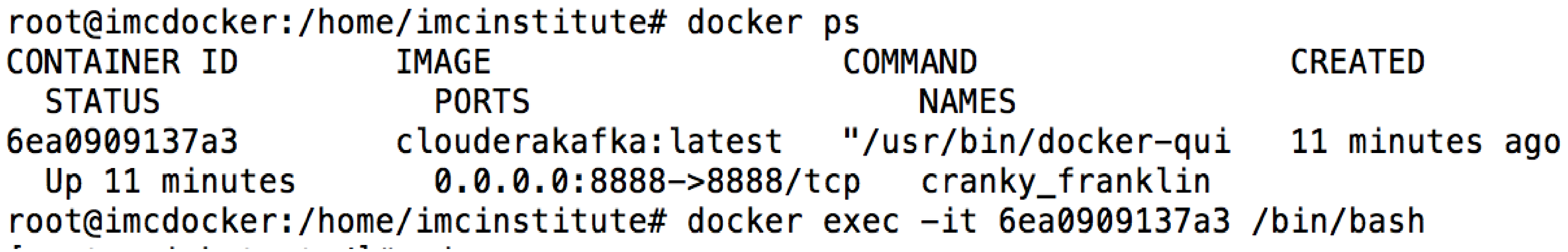
```
[root@quickstart ~]# spark-shell --driver-memory 1G
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/lib/zookeeper/lib/slf4j-log4j12-1.7.5.jar!/or
g/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/jars/slf4j-log4j12-1.7.5.jar!/org/slf4j/impl/
StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel).
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 1.6.0
      /_/
```

Streaming K-Means

```
$ scala> :paste
import org.apache.spark.mllib.linalg.Vectors
import org.apache.spark.mllib.regression.LabeledPoint
import org.apache.spark.mllib.clustering.StreamingKMeans
import org.apache.spark.SparkConf
import org.apache.spark.streaming.{Seconds, StreamingContext}
import org.apache.spark.storage.StorageLevel
import StorageLevel._
import org.apache.spark._
import org.apache.spark.streaming._
import org.apache.spark.streaming.StreamingContext._
import org.apache.spark.streaming.kafka.KafkaUtils
val ssc = new StreamingContext(sc, Seconds(2))
val kafkaStream = KafkaUtils.createStream(ssc,
"localhost:2181","spark-streaming-consumer-group", Map("java-
topic" -> 5))

val lines = kafkaStream.map(_._2)

val ratings = lines.map(Vectors.parse)

val numDimensions = 3

val numClusters = 5

val model = new StreamingKMeans()

  .setK(numClusters)

  .setDecayFactor(1.0)

  .setRandomCenters(numDimensions, 0.0)

model.trainOn(ratings)

model.predictOn(ratings).print()

ssc.start()

ssc.awaitTermination()
```

Running HelloKafkaProducer on another windows

- Open a new ssh windows



```
root@imcdocker:/home/imcinstitute# docker ps
CONTAINER ID        IMAGE                      COMMAND                  CREATED
  STATUS              PORTS                          NAMES
6ea0909137a3        clouderakafka:latest       "/usr/bin/docker-qui    11 minutes ago
  Up 11 minutes       0.0.0.0:8888->8888/tcp     cranky_franklin
root@imcdocker:/home/imcinstitute# docker exec -it 6ea0909137a3 /bin/bash
```

Java Code: Kafka Producer

```
import java.util.Properties;
import kafka.producer.KeyedMessage;
import kafka.producer.ProducerConfig;
import java.io.*;

public class HelloKafkaProducer {
    final static String TOPIC = "java-topic";
    public static void main(String[] argv){
        Properties properties = new Properties();

properties.put("metadata.broker.list","localhost:9092");

properties.put("serializer.class","kafka.serializer.StringEncoder");
```

Java Code: Kafka Producer (cont.)

```
        try(BufferedReader br = new BufferedReader(new
FileReader(argv[0]))) {
                StringBuilder sb = new StringBuilder();
                ProducerConfig producerConfig = new
ProducerConfig(properties);
                kafka.javaapi.producer.Producer<String,String>
producer = new kafka.javaapi.producer.Producer<String,
String>(producerConfig);
                String line = br.readLine();

                while (line != null) {
                        KeyedMessage<String, String> message
=new KeyedMessage<String, String>(TOPIC,line);
                        producer.send(message);
                        line = br.readLine();
                }
```

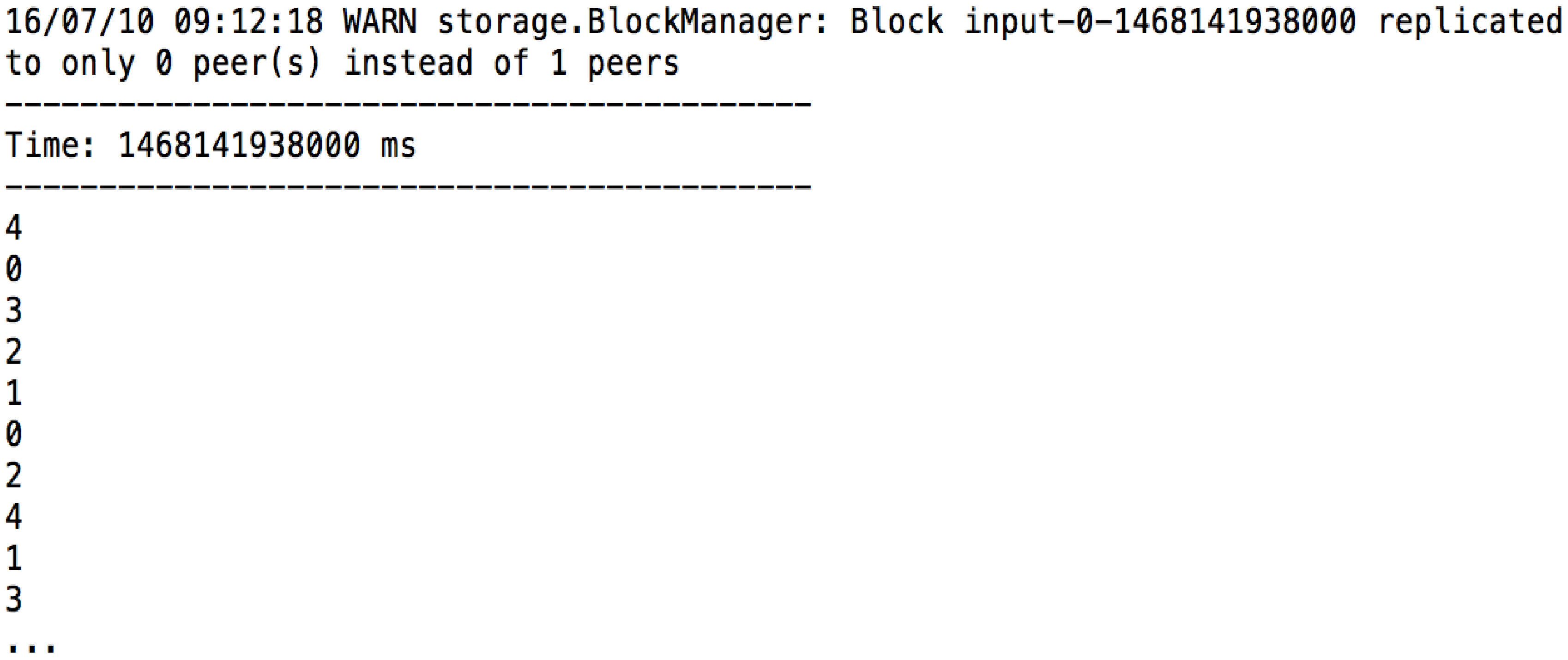
Java Code: Kafka Producer (cont.)

```
               producer.close();
        } catch (IOException ex) {
                ex.printStackTrace();
        }
     }
}
```

Compile & Run the program

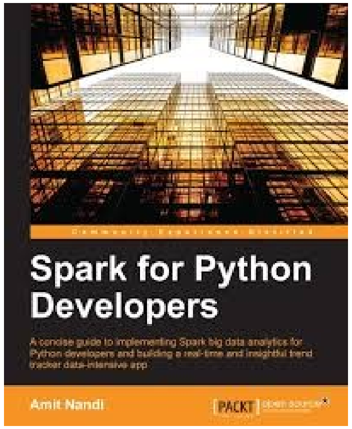
```
// Using a vi Editor to edit the sourcecode
# vi HelloKafkaProducer.java
// Alternatively
# wget
https://s3.amazonaws.com/imcbucket/apps/HelloKafkaProducer.java

// Compile progeram
# export CLASSPATH=".:/root/kafka_2.10-0.9.0.1/libs/*"
# javac HelloKafkaProducer.java

//prepare the data
# cd
# wget https://s3.amazonaws.com/imcbucket/input/pg2600.txt
# cd kafka_2.10-0.9.0.1
// Run the program
# java HelloKafkaProducer  /root/movietest.data
```

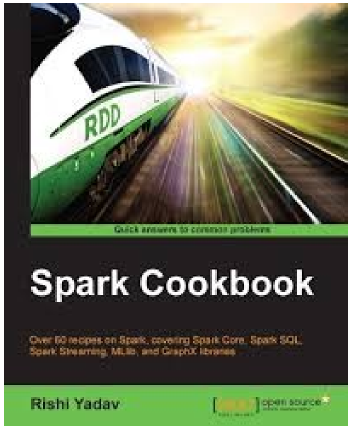
Example Result

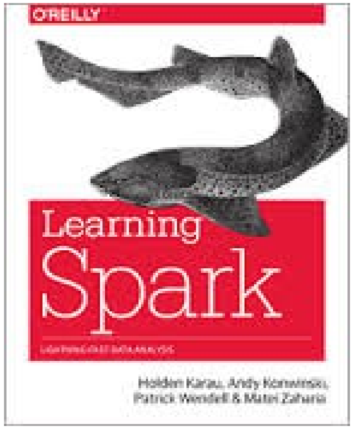


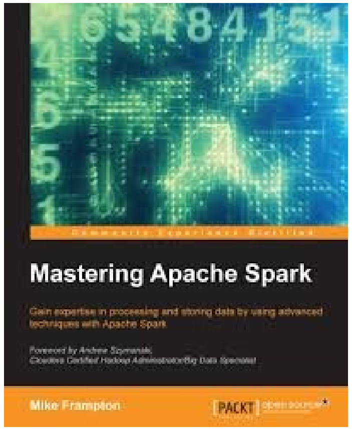
```
16/07/10 09:12:18 WARN storage.BlockManager: Block input-0-1468141938000 replicated
to only 0 peer(s) instead of 1 peers
-------------------------------------------
Time: 1468141938000 ms
-------------------------------------------
4
0
3
2
1
0
2
4
1
3
...
```

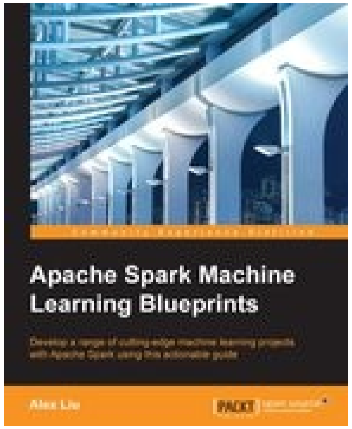
Recommended Books

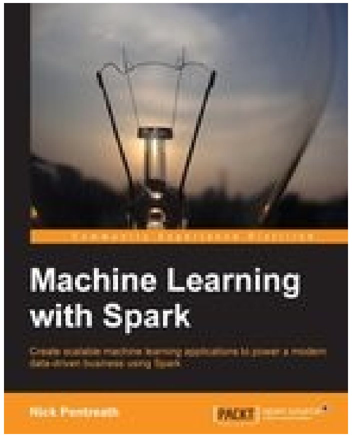

Community Experience Distilled
Spark for Python Developers
A concise guide to implementing Spark big data analytics for Python developers and building a real-time and insightful trend tracker data-intensive app
Amit Nandi
PACKT open source

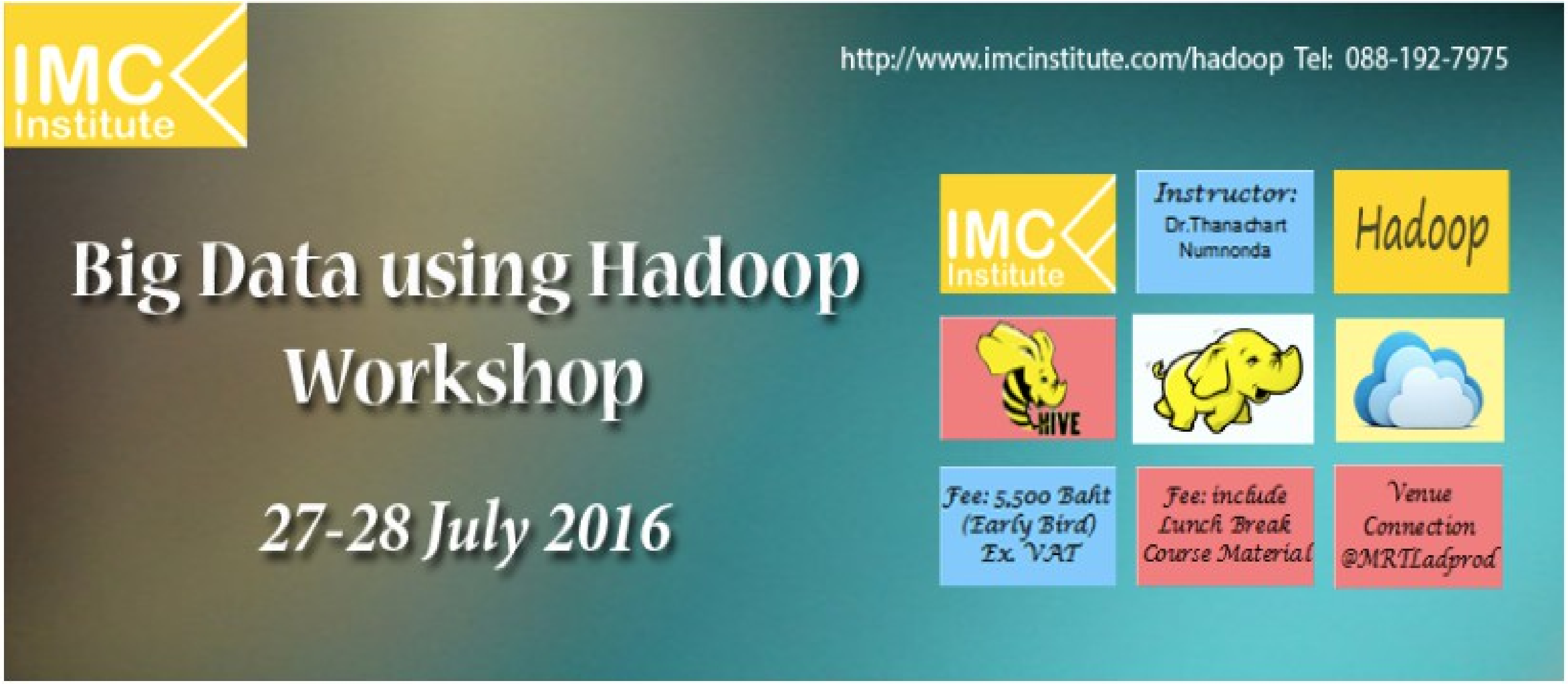

RDD
Quick answers to common problems
Spark Cookbook
Over 60 recipes on Spark, covering Spark Core, Spark SQL, Spark Streaming, MLlib, and GraphX libraries
Rishi Yadav
open source


O'REILLY
Learning Spark
LIGHTNING-FAST DATA ANALYSIS
Holden Karau, Andy Konwinski, Patrick Wendell & Matei Zaharia


Community Experience Distilled
Mastering Apache Spark
Gain expertise in processing and storing data by using advanced techniques with Apache Spark
Foreword by Andrew Szymanski, Cloudera Certified Hadoop Administrator/Big Data Specialist
Mike Frampton
PACKT open source


Community Experience Distilled
Apache Spark Machine Learning Blueprints
Develop a range of cutting-edge machine learning projects with Apache Spark using this actionable guide
Alex Liu
PACKT open source


Community Experience Distilled
Machine Learning with Spark
Create scalable machine learning applications to power a modern data-driven business using Spark
Nick Pentreath
PACKT open source



IMC Institute

http://www.imcinstitute.com/hadoop Tel: 088-192-7975

Big Data using Hadoop Workshop

27-28 July 2016

IMC Institute

Instructor:
Dr.Thanachart Numnonda

Hadoop

Fee: 5,500 Baht
(Early Bird)
Ex. VAT

Fee: include
Lunch Break
Course Material

Venue
Connection
@MRTLadprod

Thank you

www.imcinstitute.com
www.facebook.com/imcinstitute