CS 644: Introduction to Big Data

Daqing Yun (daqing.yun@njit.edu)
New Jersey Institute of Technology

Edison SC@NERSC

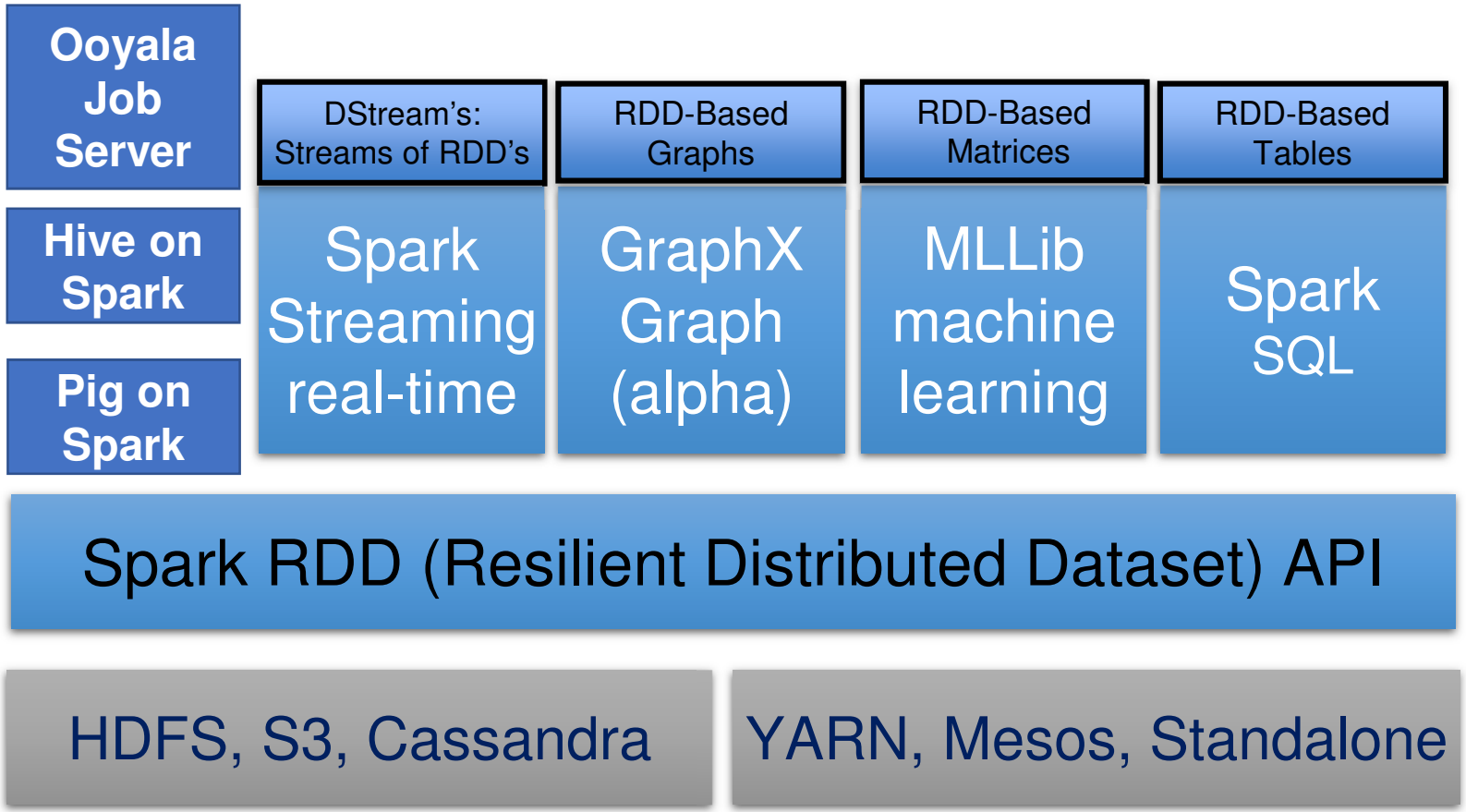
Big Data Analytics Algorithms

- Similarity (Distance) Measurements
- Clustering
  - K-Means
- Classification
  - k Nearest Neighbor
  - Logistics Regression (via Gradient Descent)
  - Decision Tree Induction
  - Naïve Bayes
  - Support Vector Machine
- Regression
  - Linear Regression (Single and Multiple) and Non-linear Regression
- Ensemble Methods
  - Random Forest
  - AdaBoost
- Principle Component Analysis
- *Neural Networks*

Hadoop-related Apache Projects

- **Ambari™**: A web based tool for provisioning, managing, and monitoring Hadoop clusters. It also provides a dashboard for viewing cluster health and ability to view MapReduce, Pig, and Hive applications visually.
- **Avro™**: A data serialization system.
- **Cassandra™**: A scalable multi-master database with no single points of failure.
- **Chukwa™**: A data collection system for managing large distributed systems.
- **HBase™**: A scalable, distributed database that supports structured data storage for large tables.
- **Hive™**: A data warehouse infrastructure that provides data summarization and ad hoc querying
- **Mahout™**: A scalable machine learning and data mining library.
- **Pig™**: A high-level data-flow language and execution framework for parallel computation
- **Spark™**: A fast and general compute engine for Hadoop data. Spark provides a simple and expressive programming model that supports a wide range of applications, including ETL, machine learning, stream processing, and graph computation.
- **Tez™**: A generalized data-flow programming framework, built on Hadoop YARN, which provides a powerful and flexible engine to execute an arbitrary DAG of tasks to process data for both batch and interactive use-cases.
- **ZooKeeper™**: A high-performance coordination service for distributed applications.

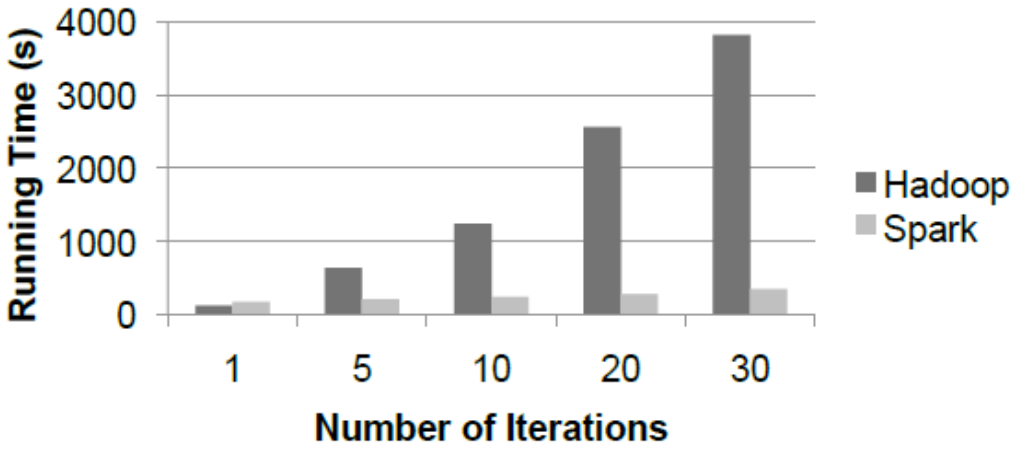
Spark and related efforts



http://www.meetup.com/spark-users/files/

Spark Concepts

- Spark focuses on one such class of applications:
  - Those that reuse a working set of data across multiple parallel operations.
  - This includes many iterative ML algorithms, as well as interactive data analysis tools.



Linear regression performance – Spark vs Hadoop

Spark: Cluster Computing with Working Sets. Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, Ion Stoica. *HotCloud 2010*. June 2010.

MLlib – Spark's Scalable ML library

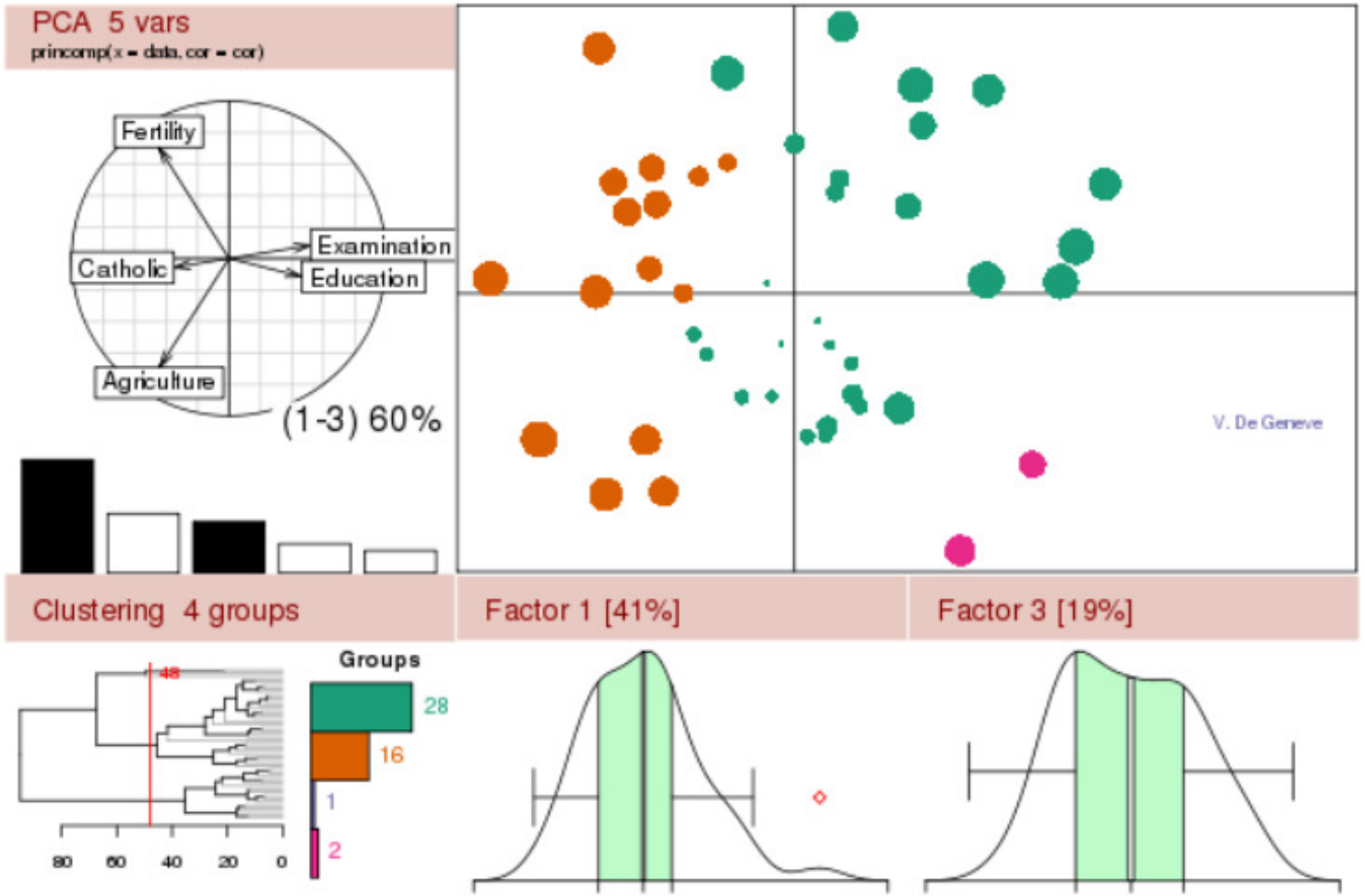
- Algorithms
  - Classification: logistic regression, naïve Bayes, ...
  - Regression: generalized linear regression, survival regression, ...
  - Decision trees, random forests, and gradient-boosted trees
  - Recommendation: alternating least squares (ALS)
  - Clustering: K-means, Gaussian mixtures (GMMs), ...
  - Topic modeling: latent Dirichlet allocation (LDA)
  - Frequent item sets, association rules, and sequential pattern mining
- ML workflow utilities
  - Feature transformations: standardization, normalization, hashing, ...
  - ML Pipeline construction
  - Model evaluation and hyper-parameter tuning
  - ML persistence: saving and loading models and Pipelines
- Others
  - Distributed linear algebra: SVD, PCA,...
  - Statistics: summary statistics, hypothesis testing,...

Sparse vector support
Linear algebra: SVD and PCA

https://spark.apache.org/docs/latest/ml-guide.html

The R Project for Statistical Computing

https://www.r-project.org/



Hadoop Integration with R

- **RHive** -- framework serves as a bridge between R language and Hive. RHive delivers the rich statistical libraries and algorithms of R to data stored in Hadoop by extending Hive's SQL-like query language (HiveQL) with R specific functions. Through the RHive functions, you can use HiveQL to apply R statistical models to data in your Hadoop cluster that you have cataloged using Hive.

- **RHadoop** -- open source framework available to R programmers, a collection of packages intended to help manage the distribution and analysis of data with Hadoop.
  - **rhdfs**: provides basic connectivity to the Hadoop Distributed File System. R programmers can browse, read, write, and modify files stored in HDFS from within R. Install this package only on the node that will run the R client.
  - **rhbase**: provides basic connectivity to the HBASE distributed database, using the Thrift server. R programmers can browse, read, write, and modify tables stored in HBASE from within R. Install this package only on the node that will run the R client.
  - **plyrmr**: enables the R user to perform common data manipulation operations, as found in popular packages such as plyr and reshape2, on very large data sets stored on Hadoop. Like rmr, it relies on Hadoop MapReduce to perform its tasks, but it provides a familiar plyr-like interface while hiding many of the MapReduce details. Install this package on every node in the cluster.
  - **rmr2**: allows R developer to perform statistical analysis in R via Hadoop MapReduce functionality on a Hadoop cluster. Install this package on every node in the cluster.
  - **ravro**: adds the ability to read and write avro files from local and HDFS file system and adds an avro input format for rmr2. Install this package only on the node that will run the R client.

https://github.com/RevolutionAnalytics/RHadoop/wiki

http://nexr.github.io/RHive/

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

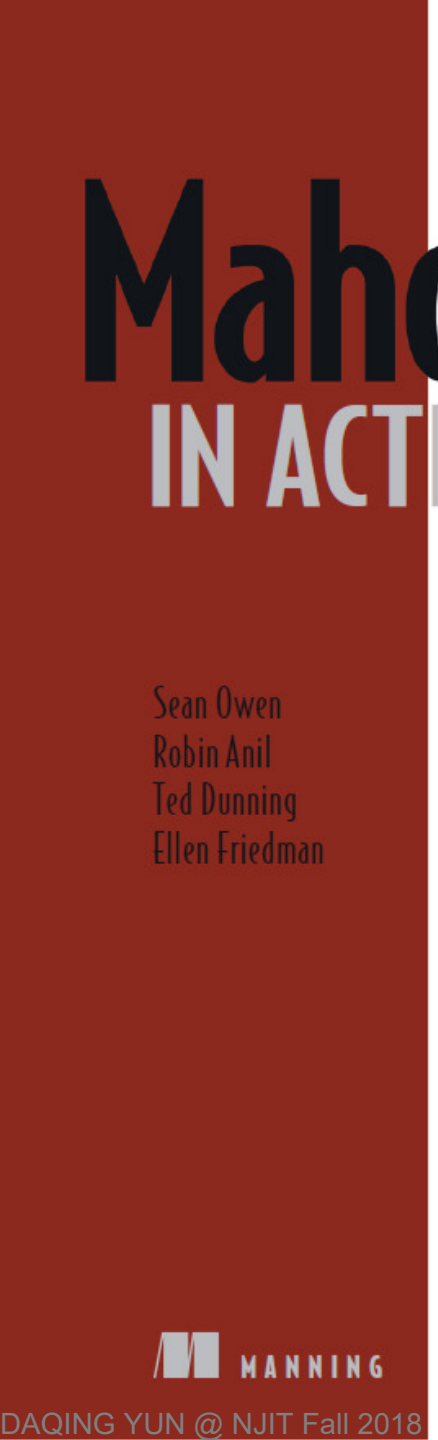
- Logistic Regression - trained via SGD - single machine
- Naive Bayes/ Complementary Naive Bayes - MapReduce
- Random Forest - **MapReduce**
- Hidden Markov Models - single machine
- Multilayer Perceptron - **single machine**

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

https://mahout.apache.org/

Mahout reference book



Requires Adobe Acrobat Reader to play audio and video links

Similarity (Distance) Measurements

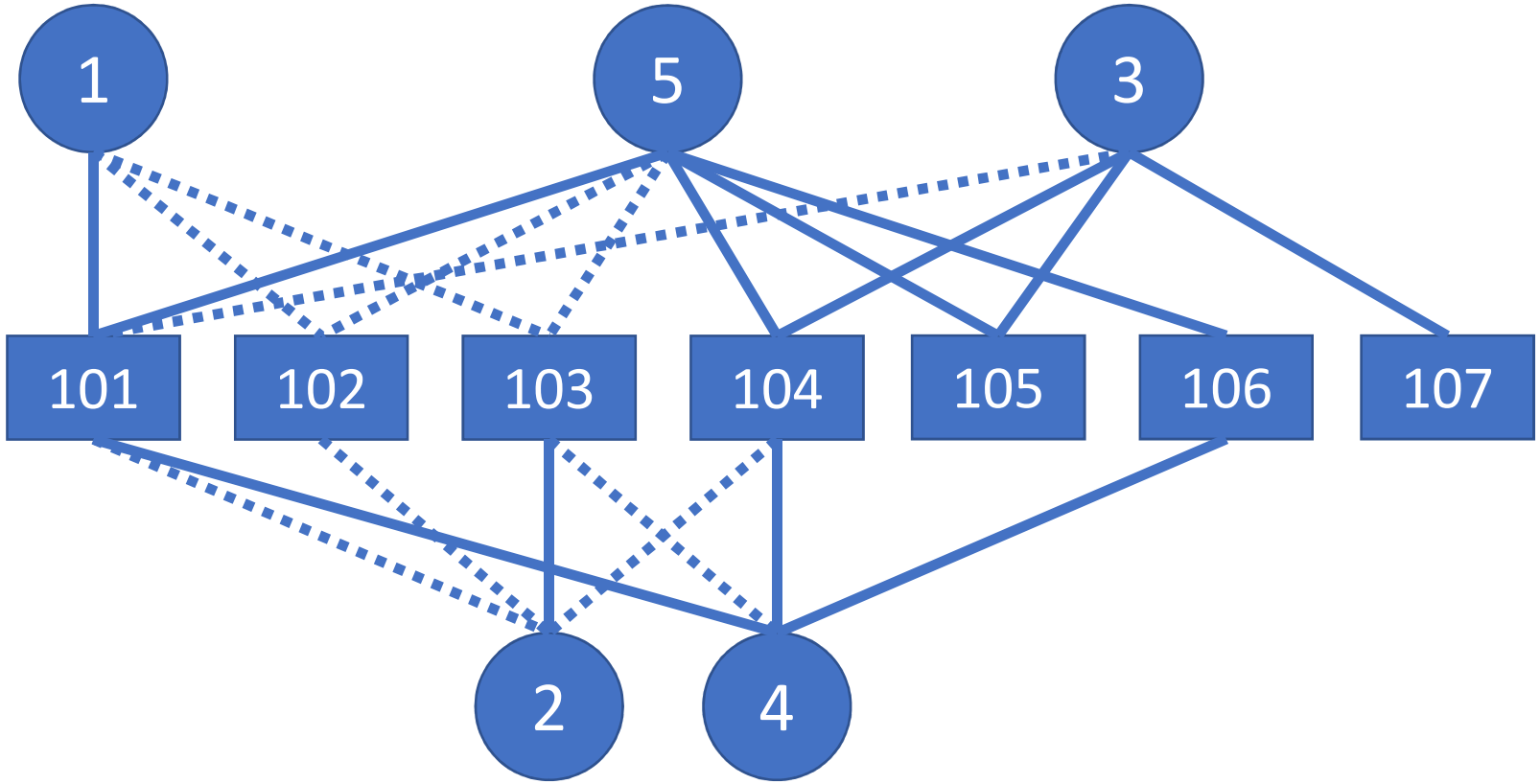
Measurement of Similarity

Why?

- Used/needed by many algorithms, e.g., clustering, nearest neighbor classification, etc.
- Initial (large) data set, in many cases, is not needed once similarities have been computed
- Transforming the data to a similarity (dissimilarity) space and then performing the analysis

Recommender -- inputs

User	Item	Rating
1,	101,	5.0
1,	102,	3.0
1,	103,	2.5
2,	101,	2.0
2,	102,	2.5
2,	103,	5.0
2,	104,	2.0
3,	101,	2.5
3,	104,	4.0
3,	105,	4.5
3,	107,	5.0
4,	101,	5.0
4,	103,	3.0
4,	104,	4.5
4,	106,	4.0
5,	101,	4.0
5,	102,	3.0
5,	103,	2.0
5,	104,	4.0
5,	105,	3.5
5,	106,	4.0



- Solid lines: positively related
- Dashed lines: negatively related

User-based Recommendation – Scenario I

- Adult: I'm looking for a CD for a teenager.
- Employee: OK, what does this teenager like?
- Adult: Oh, you know, what all the young kids like this days.
- Employee: What kind of music or bands?
- Adult: It's all noise to me. I don't know.
- Employee: Uh, well…I guess lots of young people are buying this boy band album here by New 2 Town?
- Adult: Sold!

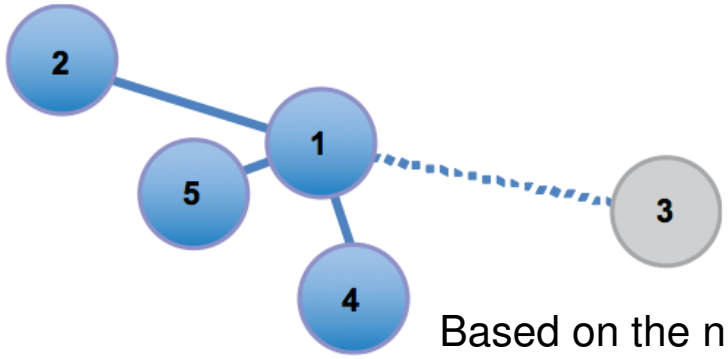
Recommendation Algorithm

```
for every item i that u has no preference for yet
  for every other user v that has a preference for i
    compute a similarity s between u and v
    incorporate v's preference for i, weighted by s, into a running average
return the top items, ranked by weighted average
```

- It would be terribly slow to examine every item.
- In reality, a neighborhood of most similar users is computed first, and only items known to those users are considered

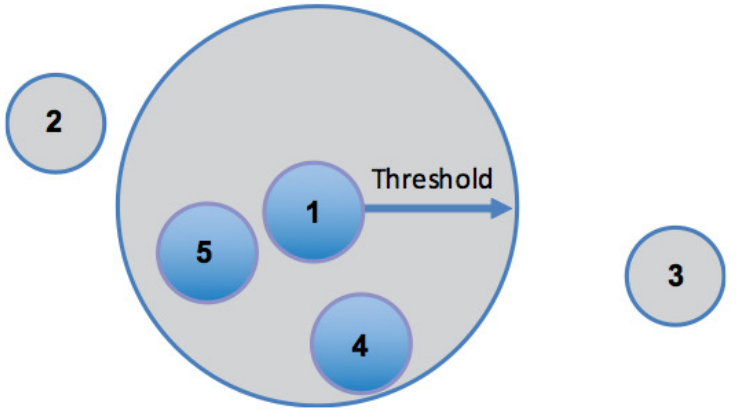
```
for every other user w
  compute a similarity s between u and w
  retain the top users, ranked by similarity, as a neighborhood n
for every item i that some user in n has a preference for,
     but that u has no preference for yet
  for every other user v in n that has a preference for i
    compute a similarity s between u and v
    incorporate v's preference for i, weighted by s, into a running average
```

Selecting neighborhood



Based on the number of neighbors

```
new NearestNUserNeighborhood(100, similarity, model);
```



Based on a fixed threshold, e.g., 0.7 or **0.5**

```
new ThresholdUserNeighborhood(0.7, similarity, model)
```

A simple user-based recommender program with Mahout

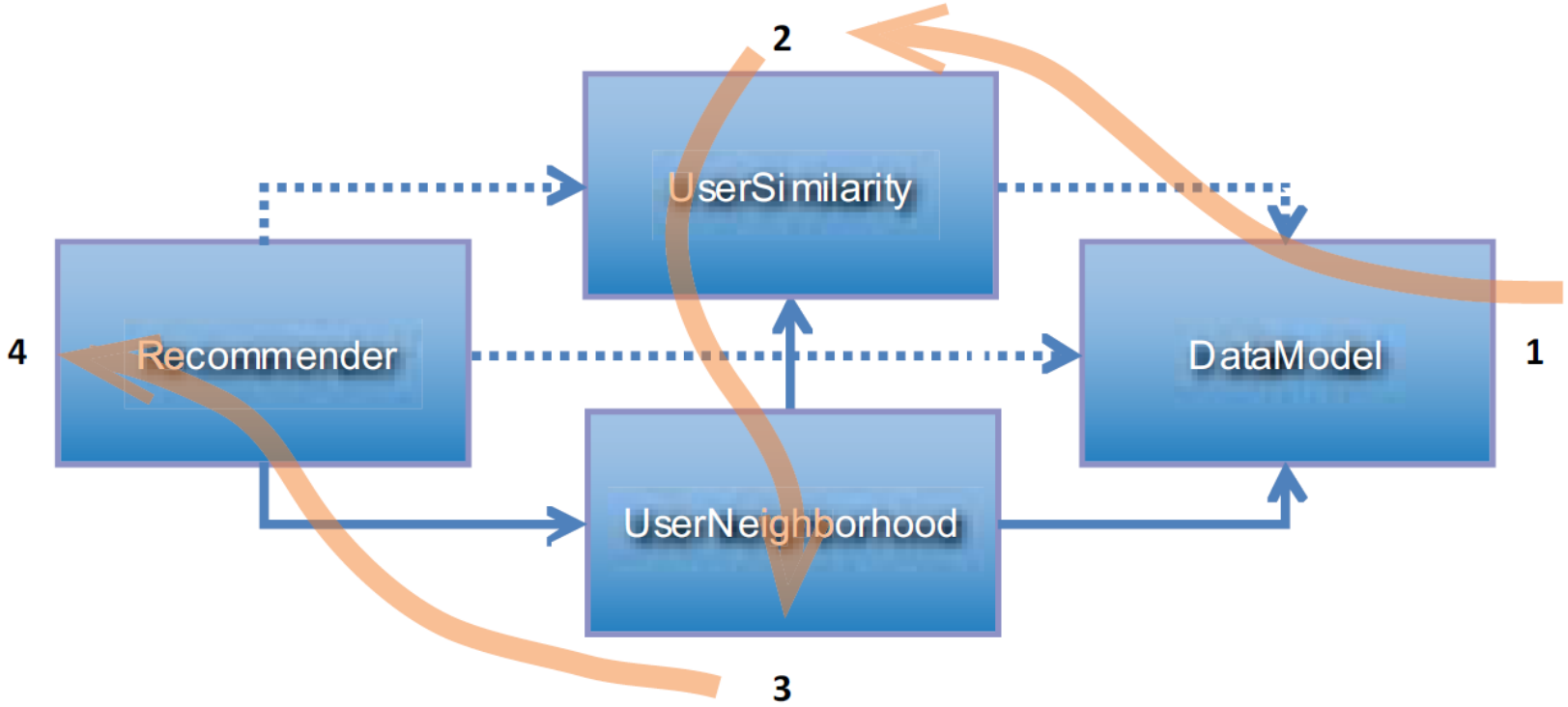
```
class RecommenderIntro {

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

    DataModel model =
      new FileDataModel (new File("intro.csv"));

    UserSimilarity similarity =
      new PearsonCorrelationSimilarity (model);
    UserNeighborhood neighborhood =
      new NearestNUserNeighborhood (2, similarity, model);

    Recommender recommender = new GenericUserBasedRecommender (
        model, neighborhood, similarity);

    List<RecommendedItem> recommendations =
        recommender.recommend(1, 1);

    for (RecommendedItem recommendation : recommendations) {
      System.out.println(recommendation);
    }

  }

}
```

Load data file

Create recommender engine

For user 1, recommend 1 item

Process and output of the example




```
RecommendedItem [item:104, value:4.257081]
```

Recommendation for Person 1:
Item 104 > Item 106
Item 107 is not favored

Performance evaluation of the simple recommender

Using GroupLens data (http://grouplens.org) -- 10 million rating MovieLens dataset.

```
DataModel model = new GroupLensDataModel (new File("ratings.dat"));
RecommenderEvaluator evaluator =
  new AverageAbsoluteDifferenceRecommenderEvaluator ();
RecommenderBuilder recommenderBuilder = new RecommenderBuilder() {
  @Override
  public Recommender buildRecommender(
        DataModel model) throws TasteException {
    UserSimilarity similarity = new PearsonCorrelationSimilarity(model);
    UserNeighborhood neighborhood =
      new NearestNUserNeighborhood(100, similarity, model);
    return new GenericUserBasedRecommender(
        model, neighborhood, similarity);
  }
};
double score = evaluator.evaluate(
    recommenderBuilder, null, model, 0.95, 0.05);
System.out.println(score);
```

10 nearest neighbors, average difference: 0.98
100 nearest neighbors, average difference: 0.89
500 nearest neighbors, average difference : 0.75

95% of training; 5% of testing

- Spearman: 0.8
- Tanimoto: 0.82
- Log-Likelihood: 0.73
- Euclidean: 0.75
- Pearson (weighted): 0.77
- Pearson: 0.89

Item-based Recommendation – Scenario II

- Adult: I’m looking for a CD for a teenage boy.
- Employee: What kind of music or bands does he like?
- Adult: He wears a Bowling In Hades T-shirt all the time and seems to have all of their albums. Anything else you’d recommend?
- Employee: Well, about everyone I know that likes Bowling In Hades seems to like the new Rock Mobster album.

Item-based Recommendation Algorithm

```
for every item i that u has no preference for yet
  for every item j that u has a preference for
    compute a similarity s between i and j
    add u's preference for j, weighted by s, to a running average
return the top items, ranked by weighted average
```

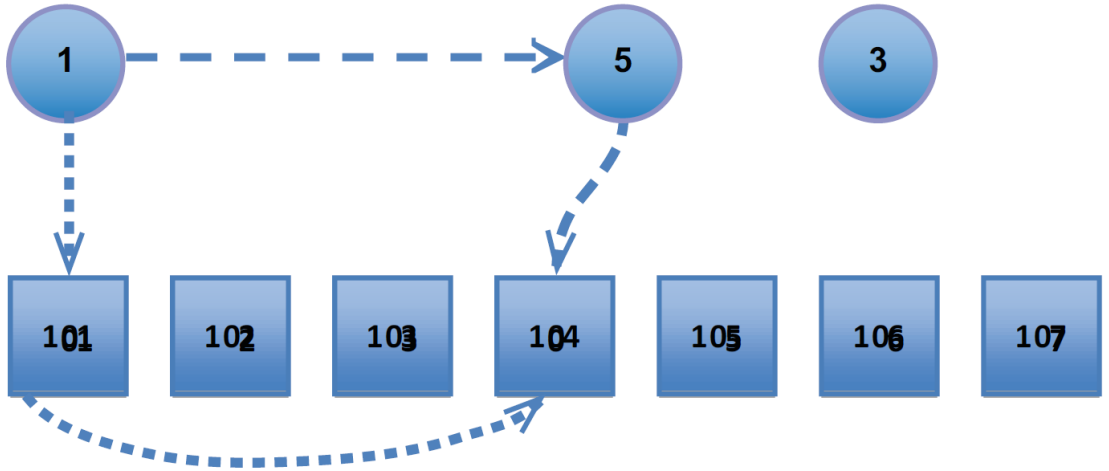


Illustration of the difference between user-based and item-based recommendation:

- User-based recommendation (large dashes) finds similar users, and sees what they like.
- Item-based recommendation (short dashes) sees what the user likes, and then finds similar items.

Code and Performance of Item-Based Recommendation

```
public Recommender buildRecommender(DataModel model)
    throws TasteException {
  ItemSimilarity similarity = new PearsonCorrelationSimilarity(model);
  return new GenericItemBasedRecommender(model, similarity);
}
```

Performance:

Implementation	Similarity
PearsonCorrelationSimilarity	0.75
PearsonCorrelationSimilarity + weighting	0.75
EuclideanDistanceSimilarity	0.76
EuclideanDistanceSimilarity + weighting	0.78
TanimotoCoefficientSimilarity	0.77
LogLikelihoodSimilarity	0.77

One thing you may notice is that this recommender setup runs significantly faster. That's not surprising, given that the data set has about 70,000 users and 10,000 items. Item-based recommenders are generally faster when there are fewer items than users.

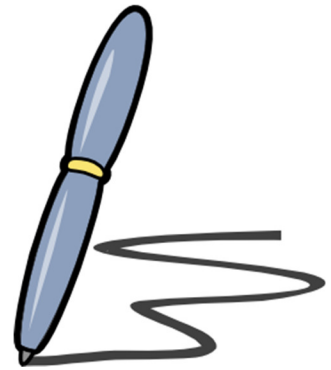
Measurement of Similarity

Definition

- Similarity between two objects is a numerical measure of the degree to which the two objects are alike
- Higher for pairs of objects that are more alike
- Transformations are often applied to convert a similarity to a dissimilarity

Measurement of Similarity

- Pearson Correlation Similarity
- Euclidean Distance Similarity
- Cosine Measure Similarity
- Spearman Correlation Similarity
- Tanimoto Coefficient Similarity (Jaccard Coefficient)
- Log-Likelihood Similarity



Pearson Correlation Similarity

Covariance of the two variables divided by the **product of their standard deviations**

$$\rho(X,Y) = \frac{\mathrm{cov}(X,Y)}{\sigma_X \sigma_Y} = \frac{E\left[(X-\mu_X)(Y-\mu_Y)\right]}{\sigma_X \sigma_Y}$$

$$\mu_X = E[X]$$

$$\mu_Y = E[Y]$$

$$\sigma_X^2 = E\left[(X-E[X])^2\right] = E\left[X^2\right] - E[X]^2$$

$$\sigma_Y^2 = E\left[(Y-E[Y])^2\right] = E\left[Y^2\right] - E[Y]^2$$

$$E\left[(X-\mu_X)(Y-\mu_Y)\right] = E\left[(X-E[X])(Y-E[Y])\right] = E[XY] - E[X]E[Y]$$

$$\rho(X,Y) = \frac{\sum XY - \frac{(\sum X)(\sum Y)}{N}}{\sqrt{\left(\sum X^2 - \frac{(\sum X)^2}{N}\right)\left(\sum Y^2 - \frac{(\sum Y)^2}{N}\right)}}$$

Pearson Correlation Similarity

$$\rho(X,Y)=\frac{\sum XY-\frac{(\sum X)(\sum Y)}{N}}{\sqrt{\left(\sum X^2-\frac{(\sum X)^2}{N}\right)\left(\sum Y^2-\frac{(\sum Y)^2}{N}\right)}}$$

Ratings of common items:
X = [5.0, 3.0, 2.5]
Y = [2.0, 2.5, 5.0]

Calculate each component:

$$\sum XY = 5.0\cdot 2.0+3.0\cdot 2.5+2.5\cdot 5.0=30$$
$$\sum X = 5.0+3.0+2.5=10.5$$
$$\sum Y = 2.0+2.5+5.0=9.5$$
$$\sum X^2 = 5.0^2+3.0^2+2.5^2=40.25$$
$$\sum Y^2 = 2.0^2+2.5^2+5.0^2=35.25$$
$$N=3$$

$$\rho(X,Y)=\frac{30-\frac{10.5\cdot 9.5}{3}}{\sqrt{\left(40.25-\frac{10.5^2}{3}\right)\left(35.25-\frac{9.5^2}{3}\right)}}=-0.7643$$

User	Item	Rating
1,	101,	5.0
1,	102,	3.0
1,	103,	2.5
2,	101,	2.0
2,	102,	2.5
2,	103,	5.0
2,	104,	2.0
3,	101,	2.5
3,	104,	4.0
3,	105,	4.5
3,	107,	5.0
4,	101,	5.0
4,	103,	3.0
4,	104,	4.5
4,	106,	4.0
5,	101,	4.0
5,	102,	3.0
5,	103,	2.0
5,	104,	4.0
5,	105,	3.5
5,	106,	4.0

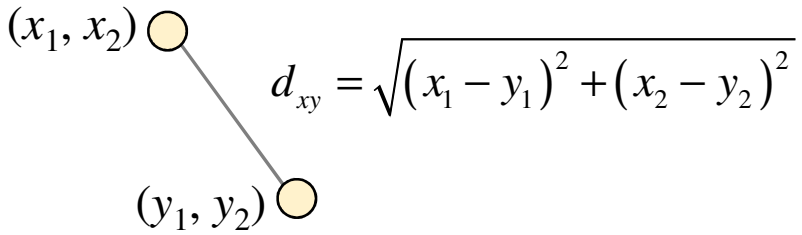
	Item 101	Item 102	Item 103	Correlation with user 1
User 1	5.0	3.0	2.5	
User 2	2.0	2.5	5.0	-0.764
User 3	2.5	-	-	-
User 4	5.0	-	3.0	
User 5	4.0	3.0	2.0	

Three problems with the Pearson Similarity

- Not take into account of the number of items in which two users' preferences overlap. (e.g., 2 overlap items ==> 1, more items may not be better.)
- If two users overlap on only one item, no correlation can be computed.
- The correlation is undefined if either series of preference values are identical.

Adding Weighting WEIGHTED as 2nd parameter of the constructor can cause the resulting correlation to be pushed towards 1.0, or -1.0, depending on how many points are used.

Euclidean Distance Similarity



$$d(X,Y) = d(Y,X)$$

$$= \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + \cdots + (x_n - y_n)^2}$$

$$= \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

$$\rho(X,Y) = \frac{1}{1+d} = \frac{1}{1 + \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}}$$

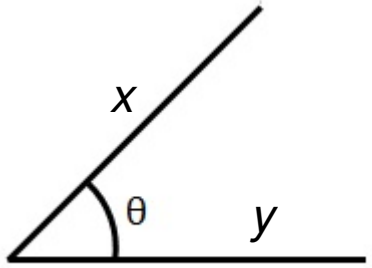
User	Item	Rating
1,	101,	5.0
1,	102,	3.0
1,	103,	2.5
2,	101,	2.0
2,	102,	2.5
2,	103,	5.0
2,	104,	2.0
3,	101,	2.5
3,	104,	4.0
3,	105,	4.5
3,	107,	5.0
4,	101,	5.0
4,	103,	3.0
4,	104,	4.5
4,	106,	4.0
5,	101,	4.0
5,	102,	3.0
5,	103,	2.0
5,	104,	4.0
5,	105,	3.5
5,	106,	4.0

	Item 101	Item 102	Item 103	Distance	Similarity to user 1
User 1	5.0	3.0	2.5	0.000	
User 2	2.0	2.5	5.0	3.937	0.203
User 3	2.5	-	-	2.500	
User 4	5.0	-	3.0	0.500	
User 5	4.0	3.0	2.0	1.118	

Cosine Measure Similarity

Cosine similarity and Pearson similarity get the same results if data are normalized (i.e., mean is zero)

$$\rho(X,Y) = \cos(\theta_{X,Y}) = \frac{X \cdot Y}{\|X\| \cdot \|Y\|} = \frac{\sum_{i=1}^{n}(x_i \cdot y_i)}{\sqrt{\sum_{i=1}^{n} x_i^2} \cdot \sqrt{\sum_{i=1}^{n} y_i^2}}$$



Geometric illustration of the cosine measure

x = (3, 2, 0, 5, 0, 0, 0, 2, 0, 0)

y = (1, 0, 0, 0, 0, 0, 0, 1, 0, 2)

xy = 3x1 + 2x0 + … + 0x2 = 5

||x|| = $(3x3 + 2x2 + … + 0x0)^{1/2}$ = 6.48

||y|| = $(1x1 + 0x0 + … + 2x2)^{1/2}$ = 2.24

Cosine Similarity of x and y is: (5/(6.48x2.24)) = 0.34

Spearman Correlation Similarity

Calculated based on pearson value about **relative ranks**

$$\rho(X,Y) = 1 - \frac{6\sum d_i^2}{n(n^2-1)}$$

$$d_i = x_i - y_i$$

Variable X_i	Position in the ascending order	Rank x_i
0.8	1	1
1.2	2	$\frac{2+3}{2} = 2.5$
1.2	3	$\frac{2+3}{2} = 2.5$
2.3	4	4
18	5	5

Example for ties

X = [1, 2, 3]
Y = [3, 2, 1]
d = [2, 0, 2]

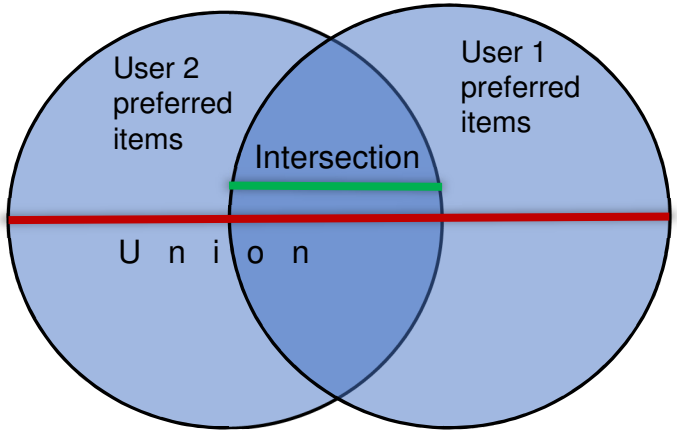
User Item Rating

```
1,   101,  5.0
1,   102,  3.0
1,   103,  2.5

2,   101,  2.0
2,   102,  2.5
2,   103,  5.0
2,   104,  2.0

3,   101,  2.5
3,   104,  4.0
3,   105,  4.5
3,   107,  5.0

4,   101,  5.0
4,   103,  3.0
4,   104,  4.5
4,   106,  4.0

5,   101,  4.0
5,   102,  3.0
5,   103,  2.0
5,   104,  4.0
5,   105,  3.5
5,   106,  4.0
```

	Item 101	Item 102	Item 103	Correlation to user 1
User 1	3.0	2.0	1.0	
User 2	1.0	2.0	3.0	–1.0
User 3	1.0	-	-	
User 4	2.0	-	1.0	
User 5	3.0	2.0	1.0	

Tanimoto (Jaccard) Coefficient Similarity

The similarity of sets by looking at the relative size of their intersection



User	Item	Rating
1,	101,	5.0
1,	102,	3.0
1,	103,	2.5
2,	101,	2.0
2,	102,	2.5
2,	103,	5.0
2,	104,	2.0
3,	101,	2.5
3,	104,	4.0
3,	105,	4.5
3,	107,	5.0
4,	101,	5.0
4,	103,	3.0
4,	104,	4.5
4,	106,	4.0
5,	101,	4.0
5,	102,	3.0
5,	103,	2.0
5,	104,	4.0
5,	105,	3.5
5,	106,	4.0

Discard preference values

$$\rho(X,Y) = \frac{|X \cap Y|}{|X \cup Y|}$$

$$= \frac{f_{11}}{f_{01} + f_{10} + f_{11}}$$

Number of matching presences

Number of attributes not involved in 00 matches

	Item 101	Item 102	Item 103	Item 104	Item 105	Item 106	Item 107	Similarity to user 1
User 1	X	X	X					
User 2	X	X	X	X				0.75
User 3	X			X	X		X	
User 4	X		X	X		X		
User 5	X	X	X	X	X	X		

Tanimoto (Jaccard) Coefficient Similarity

The similarity of sets by looking at the relative size of their intersection

- Similarity of documents – *character* level, not similar *meaning*
- Plagiarism detection
  - May not be exactly the same, but share large portions of the text
  - Plagiarizer may extract some parts of a document and later alter a few words or alter the order in which sentences of the original appear
  - No simple process of comparing documents character by character will detect a sophisticated plagiarism
- Mirror pages
  - Web pages are quite similar, but are rarely identical
- Articles from the same source
  - Finding web pages when they are textually similar, although not identical

Log-Likelihood Similarity

Assess how unlikely it is that the overlap between two users is just due to chance

$$\rho = -2\ln\left(\frac{\text{likelihood for null model}}{\text{likelihood for alternative model}}\right)$$

$$= -2\ln(\text{likelihood for null model}) + 2\ln(\text{likelihood for alternative model})$$

```
User   Item   Rating

1,     101,   5.0
1,     102,   3.0
1,     103,   2.5

2,     101,   2.0
2,     102,   2.5
2,     103,   5.0
2,     104,   2.0

3,     101,   2.5
3,     104,   4.0
3,     105,   4.5
3,     107,   5.0

4,     101,   5.0
4,     103,   3.0
4,     104,   4.5
4,     106,   4.0

5,     101,   4.0
5,     102,   3.0
5,     103,   2.0
5,     104,   4.0
5,     105,   3.5
5,     106,   4.0
```

	Item 101	Item 102	Item 103	Item 104	Item 105	Item 106	Item 107	Similarity to user 1
User 1	X	X	X					
User 2	X	X	X	X				
User 3	X			X	X		X	
User 4	X		X	X		X		
User 5	X	X	X	X	X	X		

Log-Likelihood Similarity

Log-Likelihood Similarity between User1 and User3:

	Item 101	Item 102	Item 103	Item 104	Item 105	Item 106	Item 107	Similarity to user 1
User 1	X	X	X					
User 2	X	X	X	X				
User 3	X			X	X		X	
User 4	X		X	X		X		
User 5	X	X	X	X	X	X		

Based on user preferences

	User 3 (Y)	User 3 (N)	SUM
User 1 (Y)	$k_{1,1} = 1$	$k_{1,2} = 2$	3
User 1 (N)	$K_{2,1} = 3$	$k_{2,2} = 1$	4
SUM	4	3	7

Row sum

Column sum

Sum of matrix: $S = \sum_{j=1}^{2}\sum_{i=1}^{2} k_{i,j} = 7$

Entropy of matrix:

$$H\left(\begin{bmatrix} k_{1,1}, k_{1,2} \\ k_{2,1}, k_{2,2} \end{bmatrix}\right) = -\sum_{j=1}^{2}\sum_{i=1}^{2}\left(\frac{k_{i,j}}{S}\cdot\log\left(\frac{k_{i,j}}{S}\right)\right) = 1.2770$$

Entropy of row sum:

$$H\left(\begin{bmatrix} \sum k_{1,\cdot} \\ \sum k_{2,\cdot} \end{bmatrix}\right) = -\sum_{i=1}^{2}\left(\frac{\sum k_{i,\cdot}}{S}\cdot\log\left(\frac{\sum k_{i,\cdot}}{S}\right)\right) = 0.6829$$

Entropy of col sum:

$$H\left(\begin{bmatrix} \sum k_{\cdot,1} \\ \sum k_{\cdot,2} \end{bmatrix}\right) = -\sum_{i=1}^{2}\left(\frac{\sum k_{\cdot,i}}{S}\cdot\log\left(\frac{\sum k_{\cdot,i}}{S}\right)\right) = 0.6829$$

Log-Likelihood Ratio (LLR):

$$LLR = 2\cdot S\cdot\left(H\left(\begin{bmatrix} k_{1,1}, k_{1,2} \\ k_{2,1}, k_{2,2} \end{bmatrix}\right) - H\left(\begin{bmatrix} \sum k_{1,\cdot} \\ \sum k_{2,\cdot} \end{bmatrix}\right) - H\left(\begin{bmatrix} \sum k_{\cdot,1} \\ \sum k_{\cdot,2} \end{bmatrix}\right)\right)$$

$$= 2\cdot 7\cdot(1.277 - 0.6829 - 0.6829)$$

$$= 1.2432$$

Log-Likelihood Similarity: $\rho(\text{User1}, \text{User2}) = 1 - \frac{1}{1+LLR}$

$$= 1 - \frac{1}{1+1.2432} = 0.5542$$

Log-Likelihood Similarity

- Mahout includes the Log-Likelihood Similarity
  - https://goo.gl/RMPQWL

	User 3 (Y)	User 3 (N)	SUM
User 1 (Y)	$k_{1,1} = 1$	$k_{1,2} = 2$	3
User 1 (N)	$K_{2,1} = 3$	$k_{2,2} = 1$	4
SUM	4	3	7

	Item 101	Item 102	Item 103	Item 104	Item 105	Item 106	Item 107	Similarity to user 1
User 1	X	X	X					
User 2	X	X	X	X				
User 3	X			X	X		X	0.55
User 4	X		X	X		X		
User 5	X	X	X	X	X	X		

Clustering (K-Means)

Clustering

- Given a set of data points, each having a set of attributes, and a *similarity* measure among them, find clusters such that
  - Data points in one cluster are more similar to one another
  - Data points in separate clusters are less similar to one another

- Similarity (distance) measures:
  - *Euclidean distance* if attributes are continuous
  - Other problem-specific measures

Tan, Steinbach, and Kumar. Introduction to Data Mining

Clustering a collection involves three things

An algorithm -- This is the method used to group the objects together.

A notion of both similarity and dissimilarity -- This determines which objects belong to an existing group (cluster) and which should start a new one.

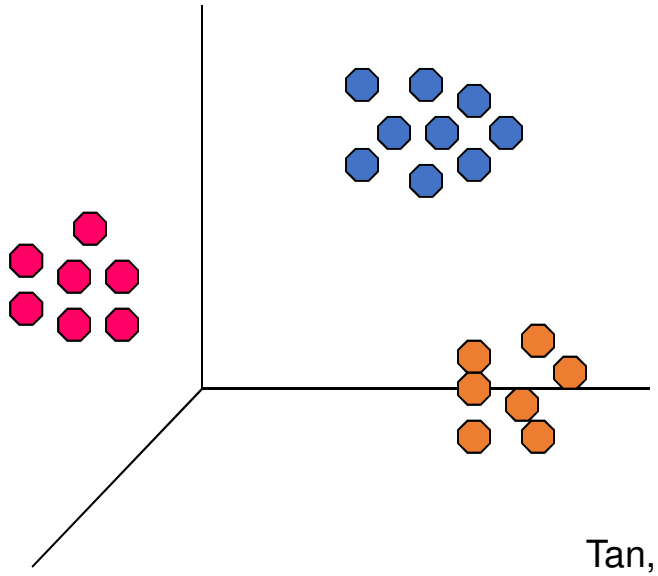
A stopping condition -- This might be the point beyond which objects cannot be grouped (clustered) anymore, or when the objects are already quite dissimilar.

Clustering - Example

- Euclidean distance based clustering in 3D space

Intra-cluster distances are minimized

Inter-cluster distances are maximized

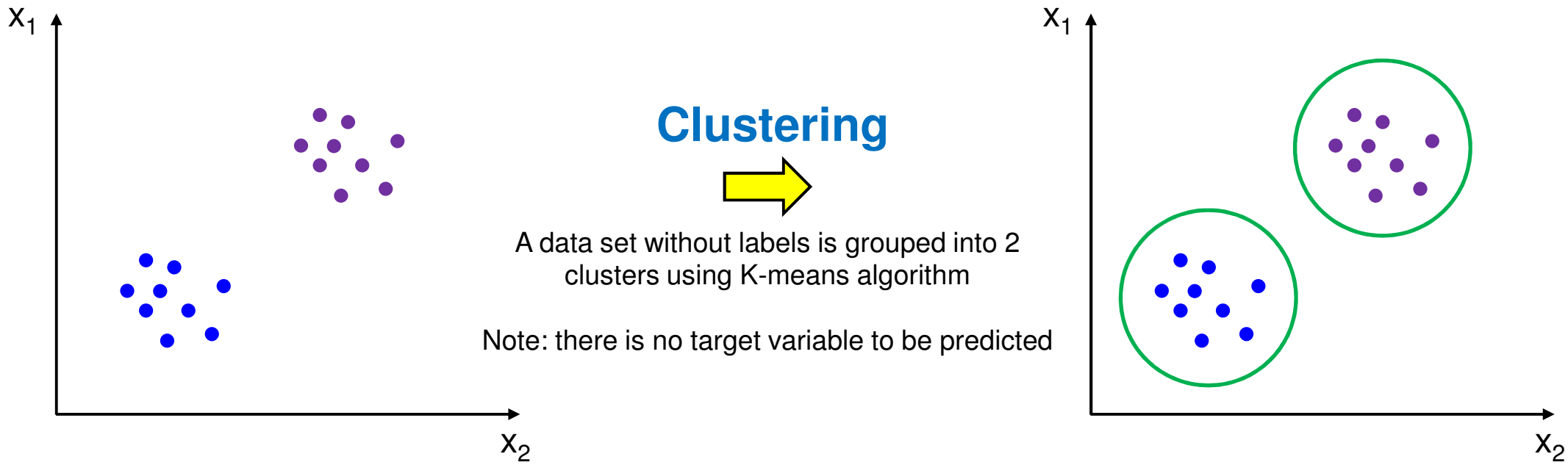


Tan, Steinbach, and Kumar. Introduction to Data Mining

Unsupervised Learning

Discover the structure within the un-labeled data

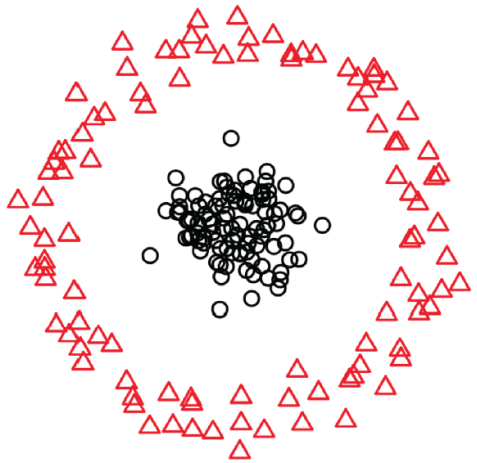
- Unsupervised machine learning is the machine learning task of inferring a function to describe hidden structure from "**unlabeled**" data
- Examples of unsupervised learning tasks include
  - **Clustering** (where we try to discover underlying groupings of examples)
  - **Anomaly detection** (where we try to infer if some examples in a dataset do not conform to some expected pattern)

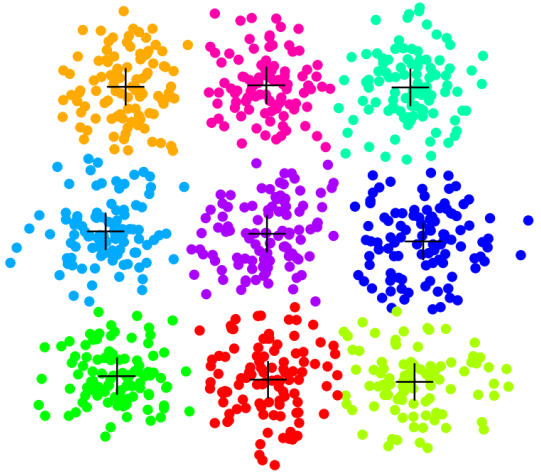


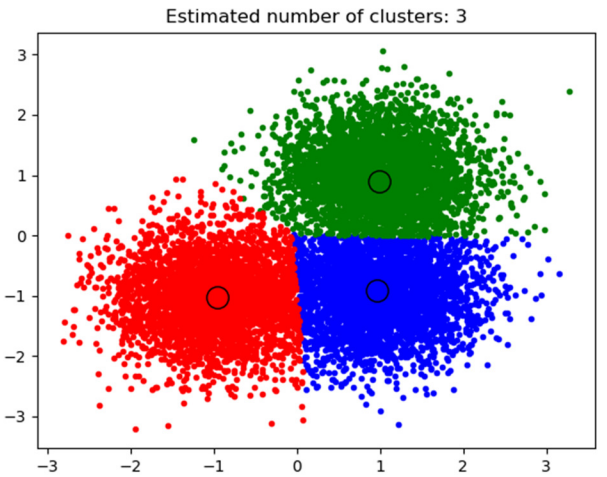
Clustering Analysis

A cluster is a subset of data items which are similar

- Clustering is a technique for finding similarity groups in a data, called clusters
- It attempts to group individuals in a population together by similarity, but not driven by a specific purpose
- Clustering is often called an unsupervised learning, as you don't have prescribed labels in the data and no class values denoting a priori grouping of the data instances are given
- A cluster is a collection of data items which are "similar" between them, and "dissimilar" to items in other clusters







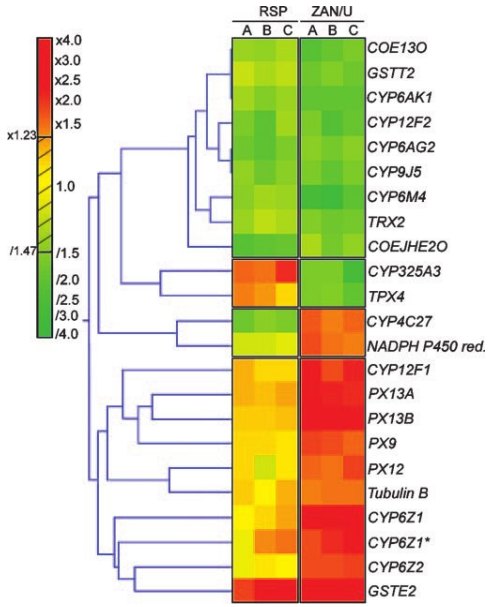
http://bigdata-madesimple.com/possibly-the-simplest-way-to-explain-k-means-algorithm/
http://www.mit.edu/~9.54/fall14/slides/Class13.pdf
http://scikit-learn.org/stable/auto_examples/cluster/plot_mean_shift.html
http://www.stat.cmu.edu/~ryantibs/statml/lectures/clustering.pdf

Application of Clustering

- Clustering is widely used in many applications
- Clustering is hard to evaluate, but very useful in practice

Market Segmentation
Discover customer groups and use them for targeted marketing programs

Social Network Analysis
Help you find a group of coherent friends on the social networks



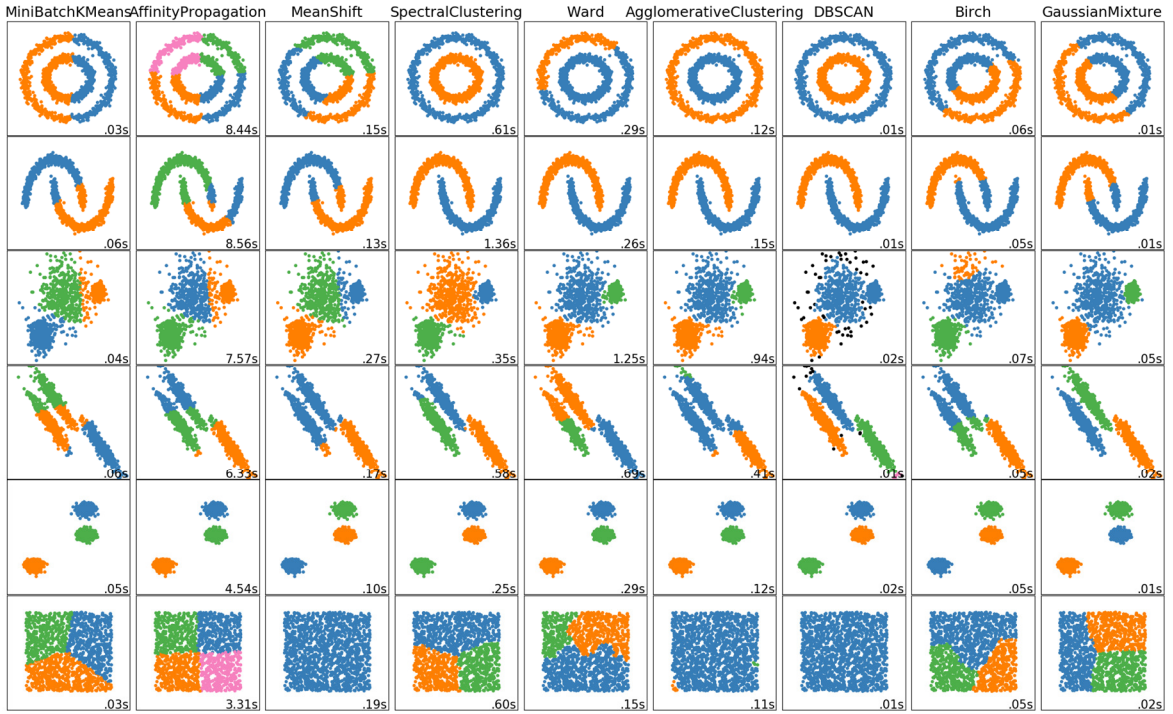
Genomics
Finding groups of gene with similar expressions

Clustering Methods

Clustering has a long history and still is in active research, there are possibly over 100 published clustering algorithms

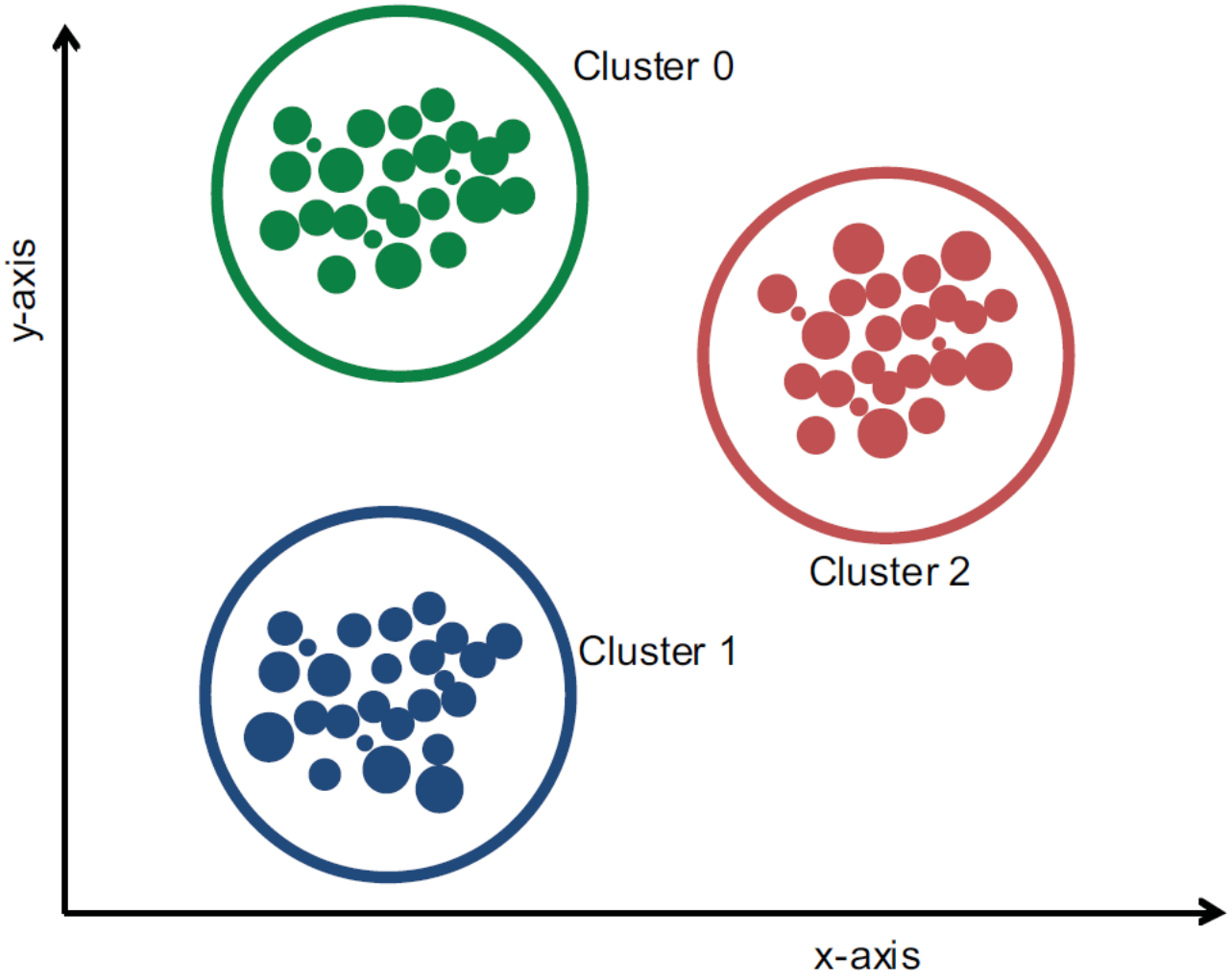
- **Partitioning Methods (Centroid-based Clustering)**
  - Find mutually exclusive clusters of spherical shape
  - Distance-based
  - K-Means is the most popular algorithm due to its simplicity and efficiency
- **Connectivity-based Methods (Hierarchical Clustering)**
  - Agglomerative hierarchical clustering (“bottom-up”)
  - Divisive hierarchical clustering (“top-down”)
- **Density-based Methods**
  - Popular examples are DBSCAN and its variants (e.g., OPTICS)
- **Grid-based Methods**
  - Use a multi-resolution grid data structure
- **Distribution-based Clustering**
  - A popular example is Expectation-Maximization algorithm

Comparison of different clustering algorithms on toy dataset



http://scikit-learn.org/stable/auto_examples/cluster/plot_cluster_comparison.html

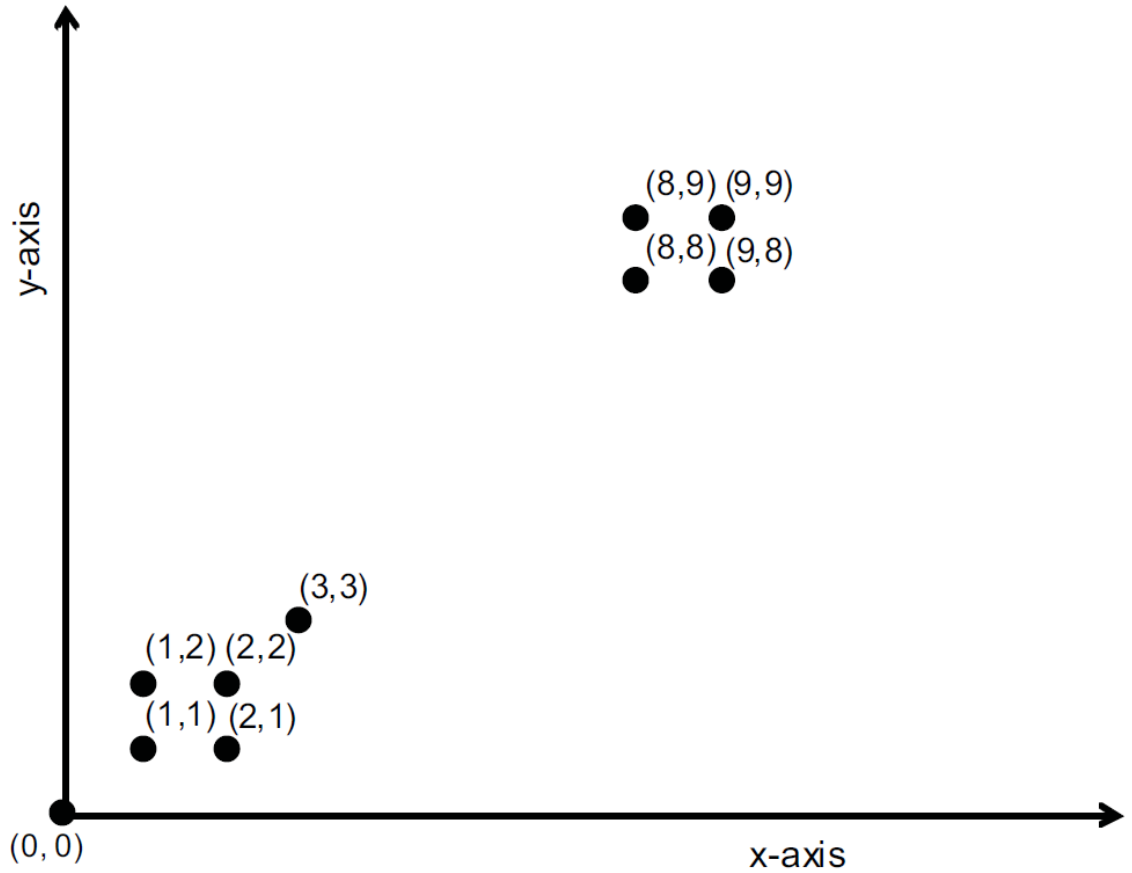
Clustering on Feature Plane



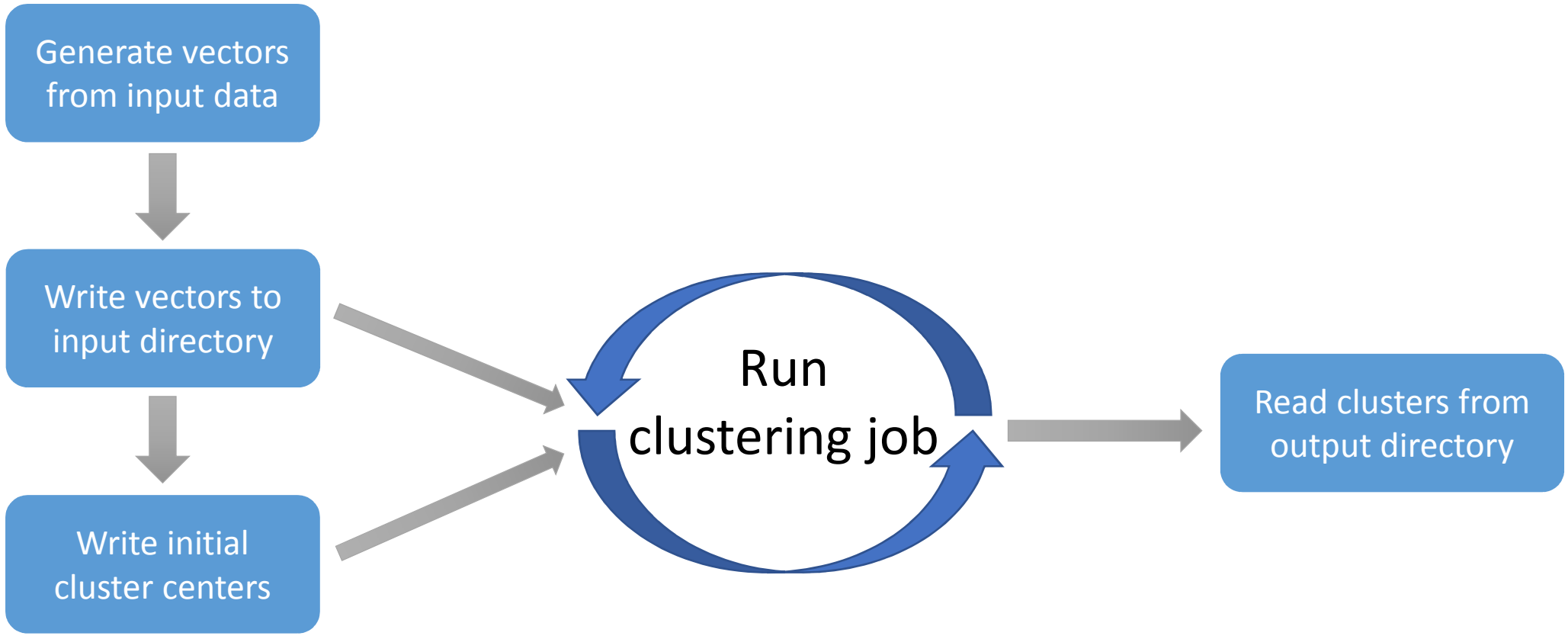
- Points in an x-y plane.
- The circles represent clusters, and the points on the plane could be viewed as three logical groups.
- Clustering algorithms can help identify those groups.

Clustering Example

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



Steps on Clustering



K-means Clustering

- Partition data into k number of mutually exclusive clusters
- How well a point fits into a cluster is determined by the distance from that point to the cluster's center

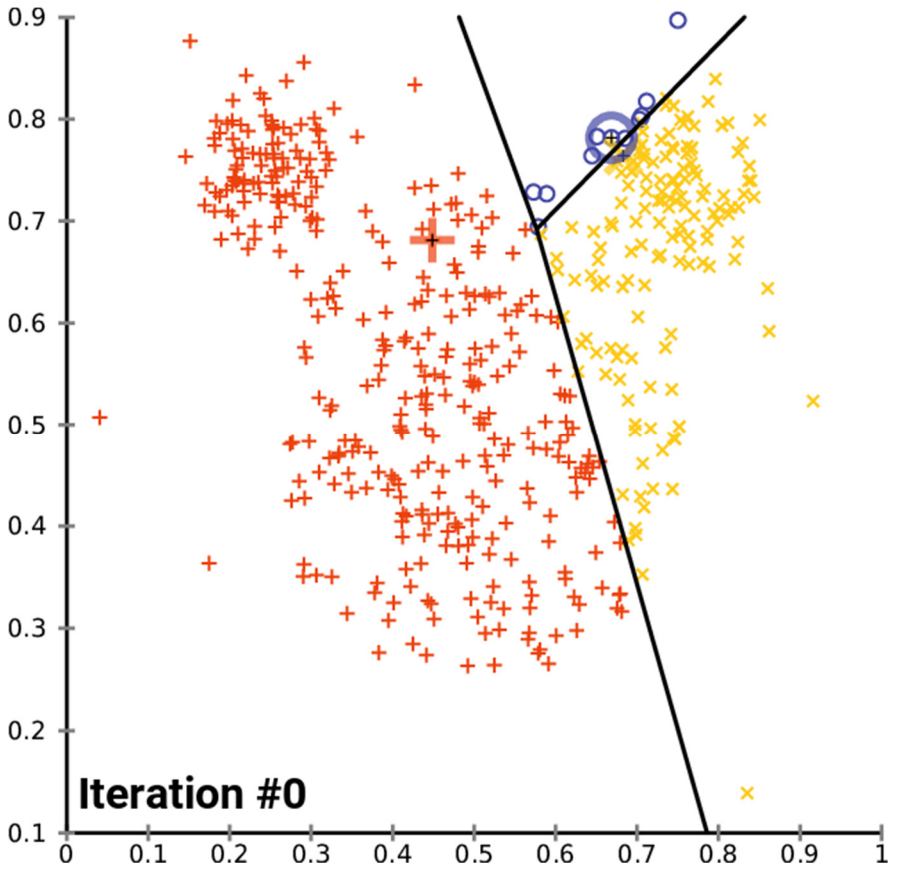
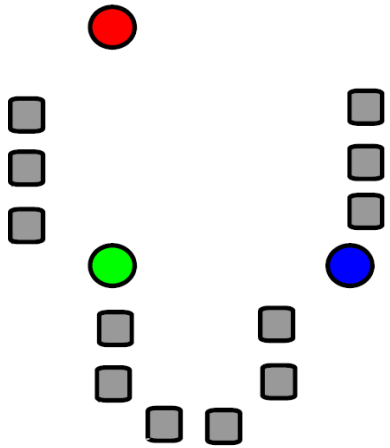


Image: https://en.wikipedia.org/wiki/K-means_clustering#/media/File:K-means_convergence.gif

K-means

- Graphical view of K-means clustering process

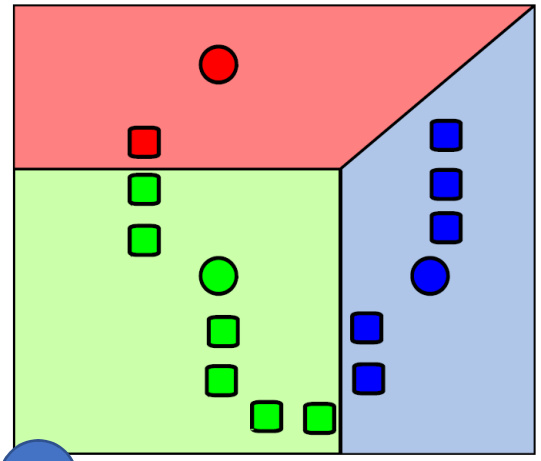
Choose the initial centroids



1

k (*k*=3) initial "means" are randomly generated within the data domain (shown in color)

Cluster assignment step

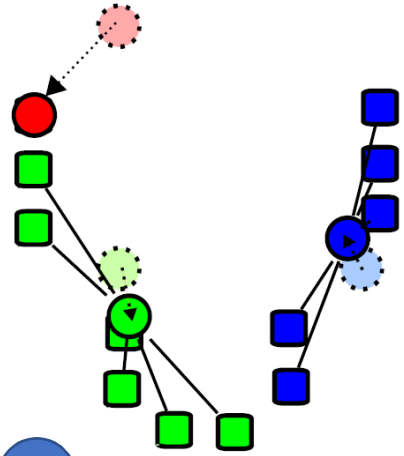


2

k clusters are created by associating every observation with the nearest mean

In cluster assignment step, the algorithm goes through each of the data points and assigns it to the closest cluster

Move centroid step

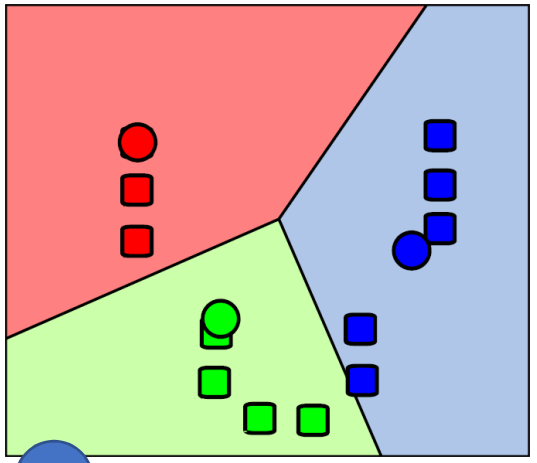


3

The centroid of each of the *k* clusters becomes the new mean

In move centroid step, the algorithm moves the centroids to the average of the points in the cluster, i.e., it calculates the average of all points in a cluster and moves the centroid to the average location

Repeat



4

Steps 2 and 3 are repeated until convergence has been reached

In other words, it repeats until the centroids do not move significantly anymore

Source: wikipedia

K-means (Llyod's) Clustering Algorithm

- Intend to partition n objects into k clusters in which each object belongs to the cluster with the nearest mean

1. Randomly initialize k cluster centroids $\mu_1, \mu_2, \ldots, \mu_k \in \mathbb{R}^n$
2. Repeat until convergence {

Centroid for cluster:
μ_j = mean of the cluster

- For every example *i*, set

$$c^{(i)} := j \text{ that minimizes } \left\| x^{(i)} - \mu_j \right\|^2$$

Euclidean distance

Cluster assignment step

Assign each data point to the nearest cluster whose mean has the least squared Euclidean distance

- For every *k*, set

$$\mu_k := \text{average (mean) of points assigned to the cluster } k$$

$$\mu_k := \frac{1}{|C_k|} \sum_{i \in C_k} x^{(i)}$$

Update centroid step → move centroid

Calculate the new means to be the centroids of the observations in the new clusters

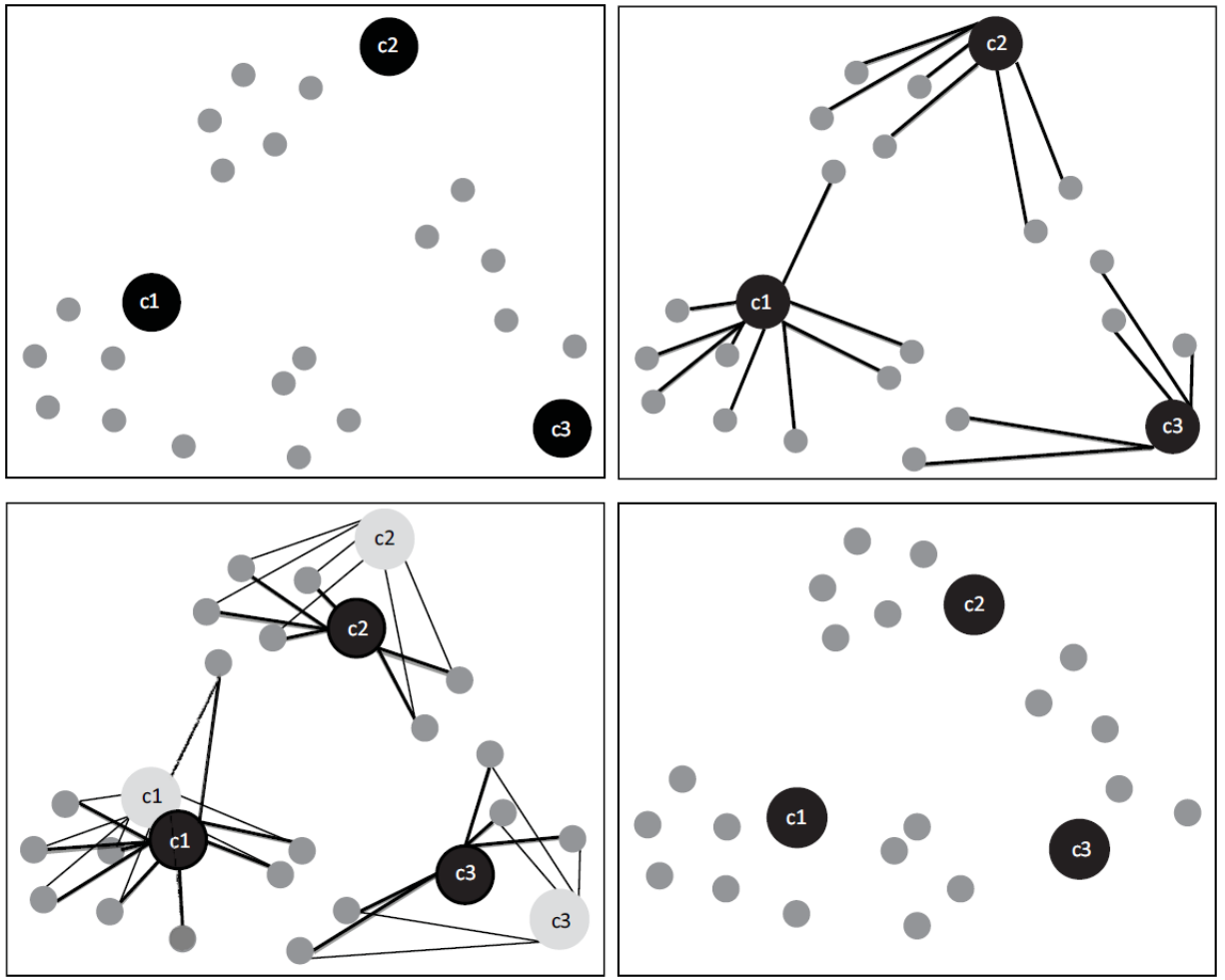
} // end of repeat

http://www.saedsayad.com/clustering_kmeans.htm
https://www.youtube.com/watch?v=LvgcfMOyREE

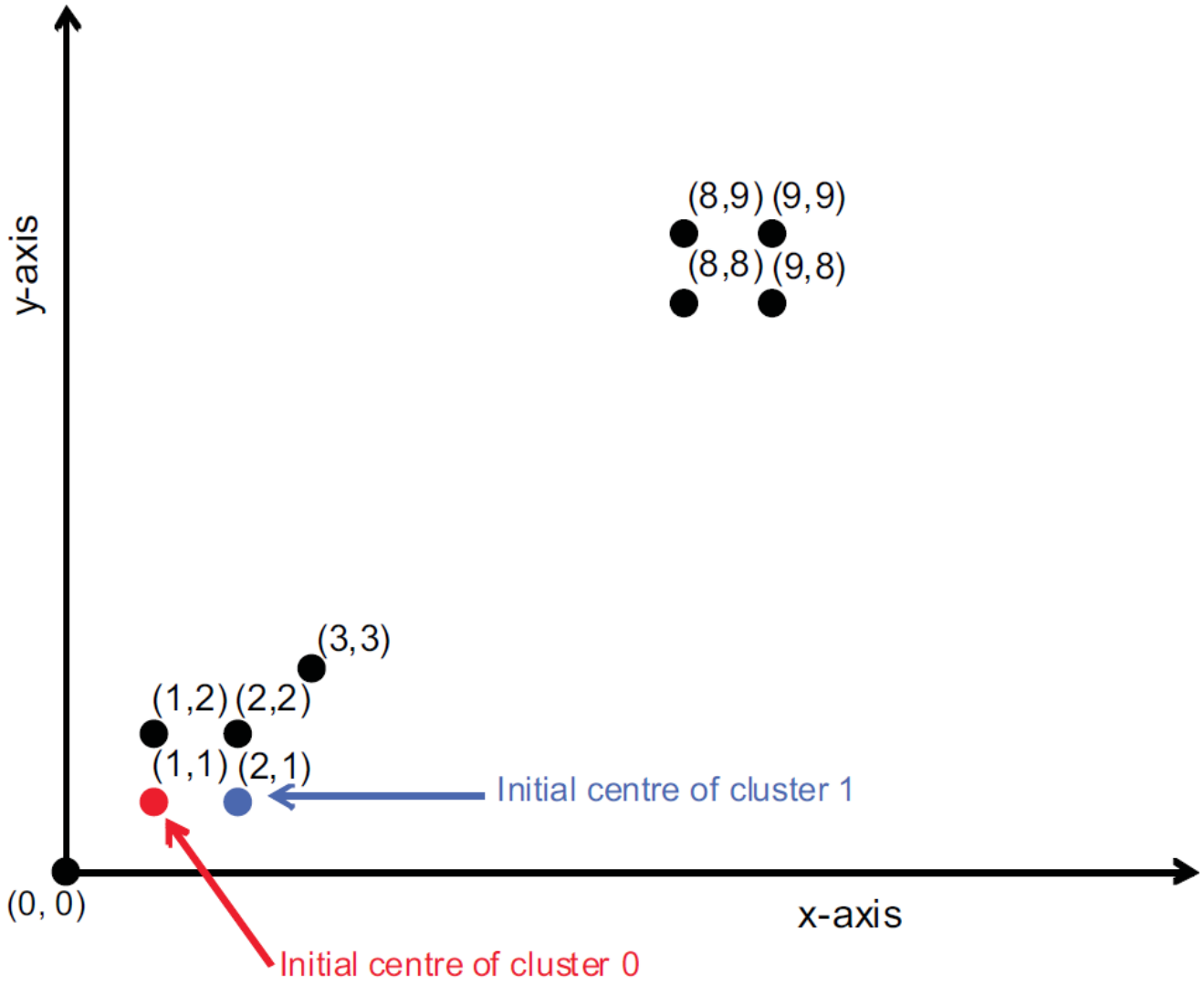
K-means Clustering

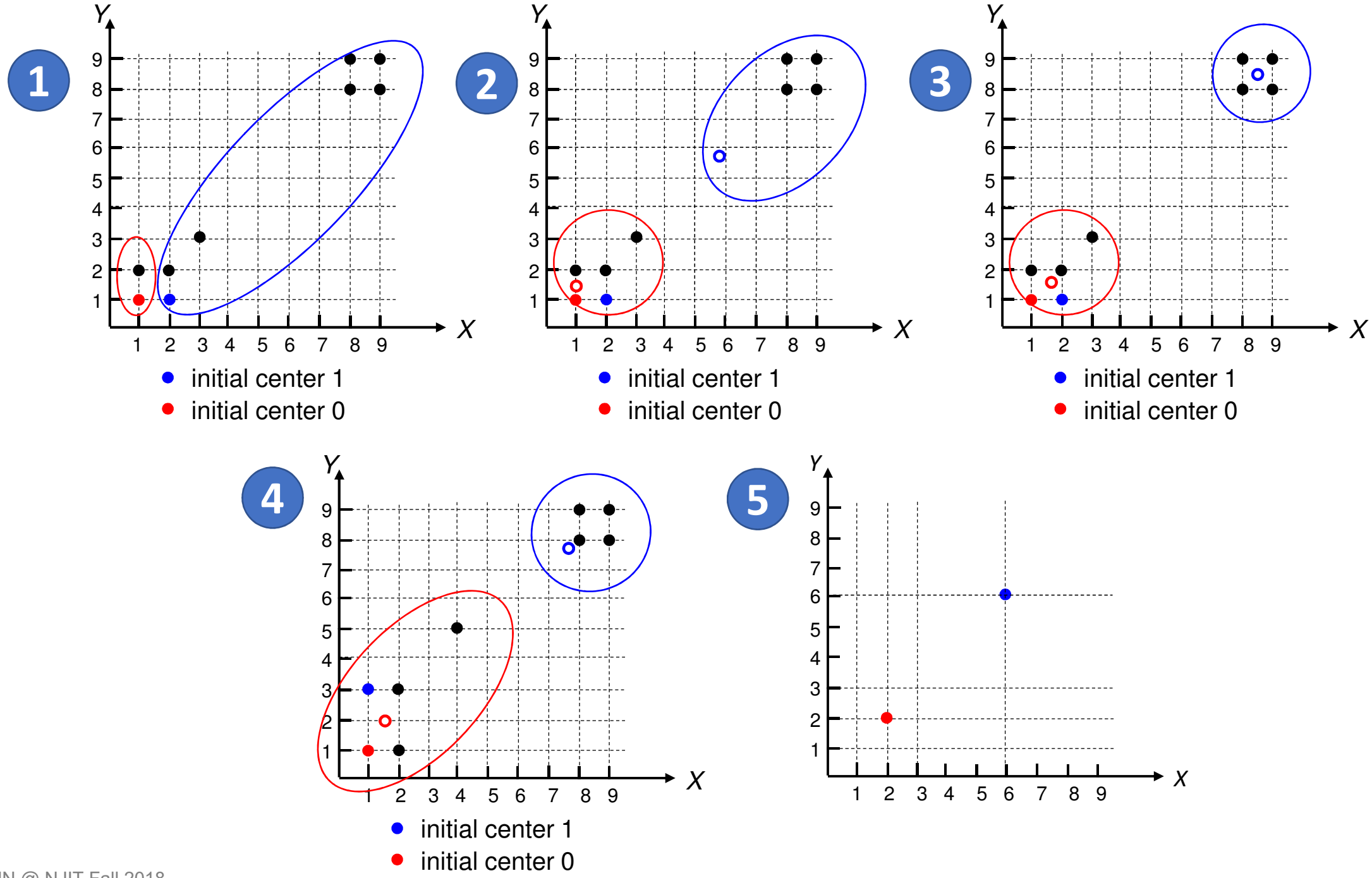
K-means clustering in action

- 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.

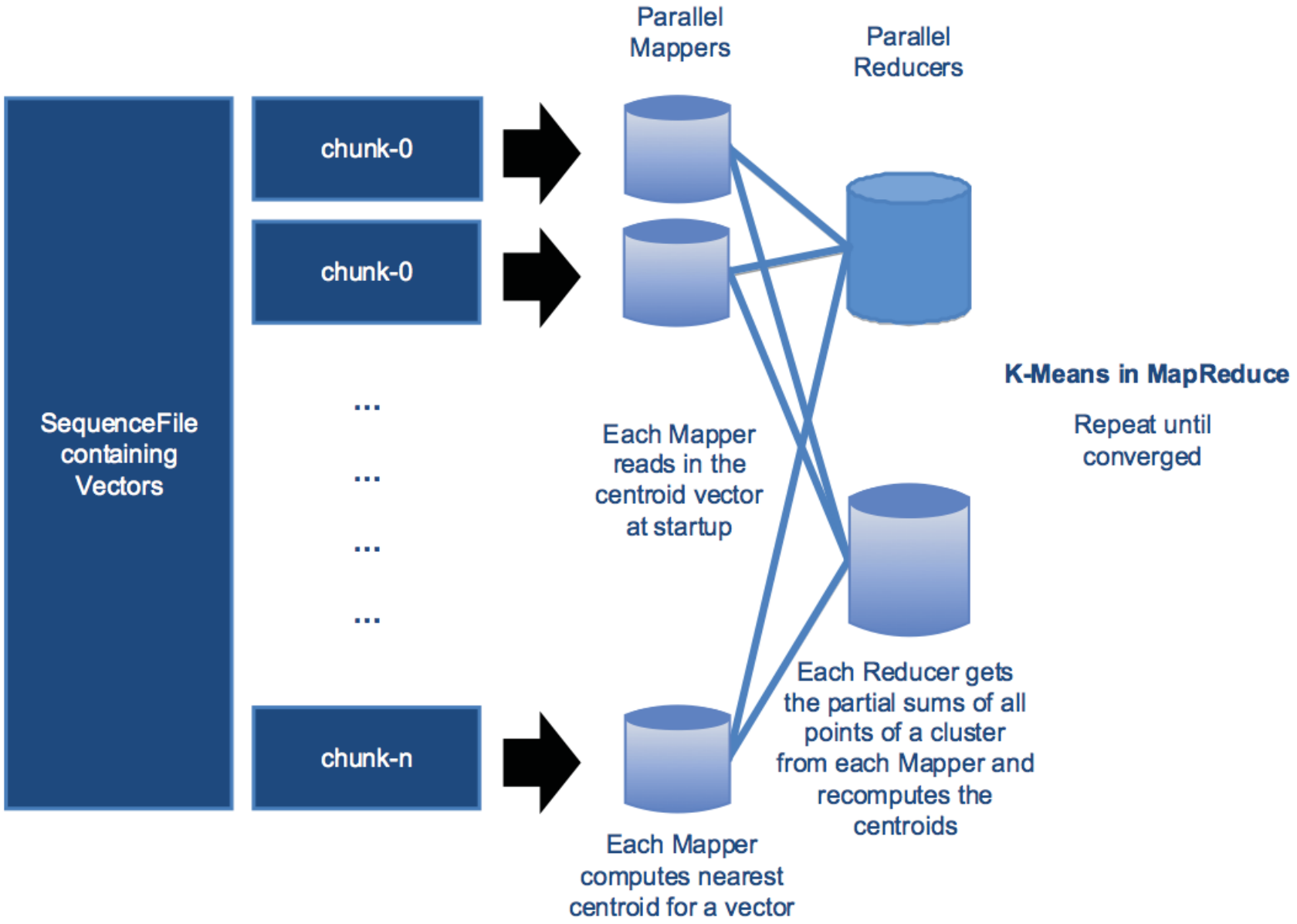


Making initial clustering centers is an important step in k-means clustering




1
2
3
4
5
Y
X
initial center 1
initial center 0

K-means clustering running as MapReduce job



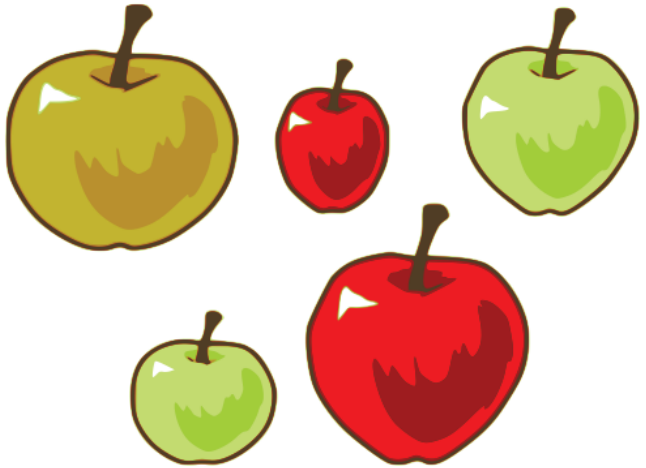
Hadoop K-means clustering jobs

- In Mahout, the MapReduce version of the k-means algorithm is instantiated using `KMeansDriver` class, which has just a single entry point -- the `runJob` method.
- The algorithm takes the following input parameters:
  - 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.

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 pre-allocated 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 are called as `SparseVector`.
  - `SequentialAccessSparseVector` is implemented as two parallel arrays, one of integers and the others 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 => 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]

The k-means clustering job entry point

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

TIP 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 Hello World clustering code

```
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;
}
```

The Hello World clustering code

```
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

The Hello World clustering code

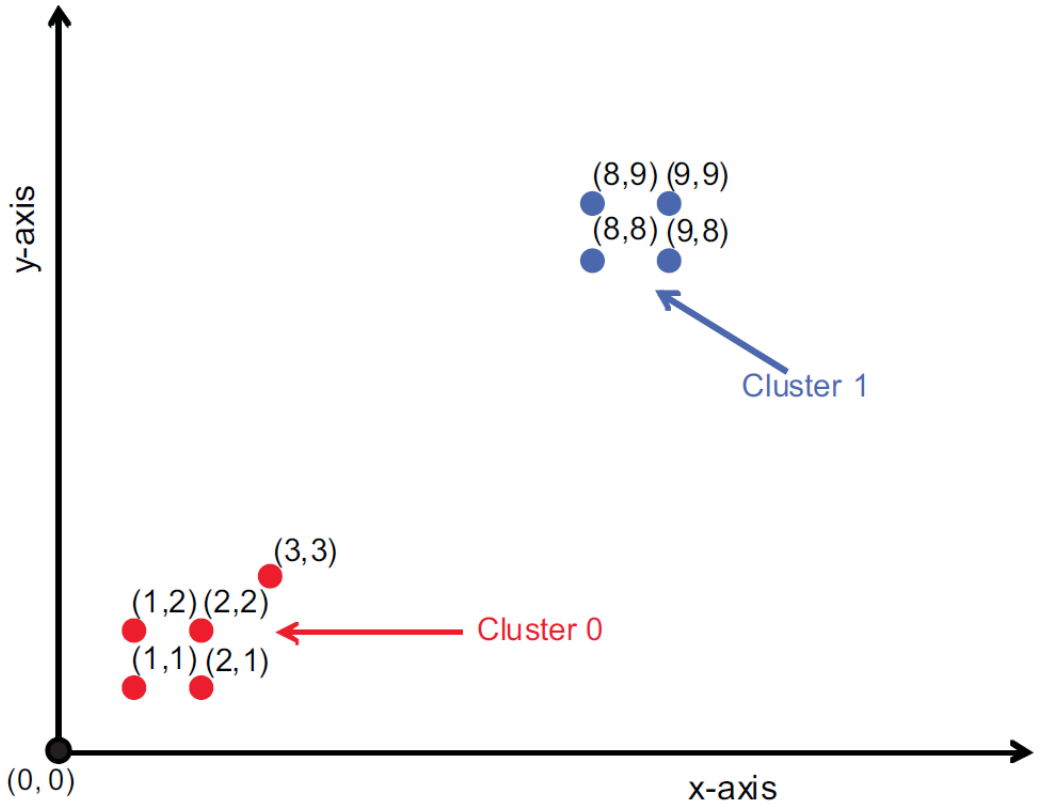
```
KMeansDriver.run(conf, new Path("testdata/points"),
  new Path("testdata/clusters"),
  new Path("output"), new EuclideanDistanceMeasure(),
      0.001, 10, true, false);

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

The Hello World example -- results

```
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
```



The output of the Hello World k-means clustering program. Even with distant centers, the k-means algorithm was able to correctly iterate and correct the centers based on the Euclidean distance measure.

Testing Different Distance Measurements

- **Euclidean distance measure**
  - A ruler-measured distance

$$d = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + \ldots + (x_n - y_n)^2}$$

- **Squared Euclidean distance measure**

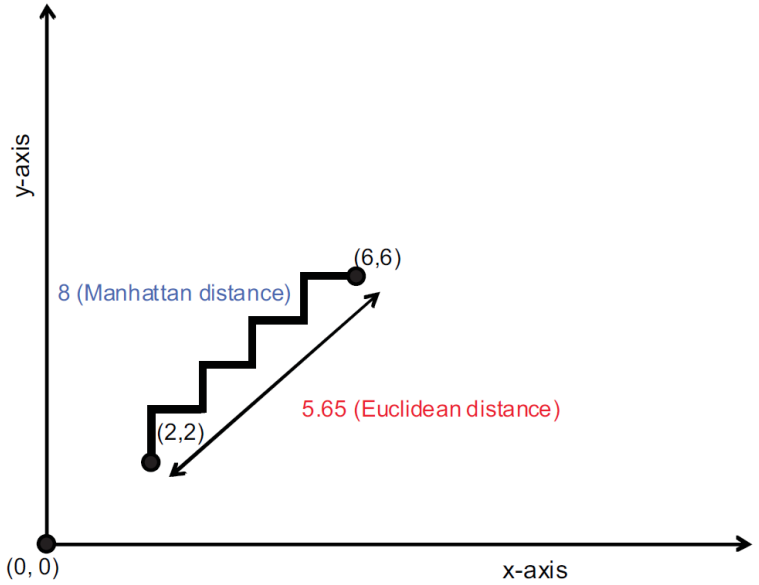
$$d = (x_1 - y_1)^2 + (x_2 - y_2)^2 + \ldots + (x_n - y_n)^2$$

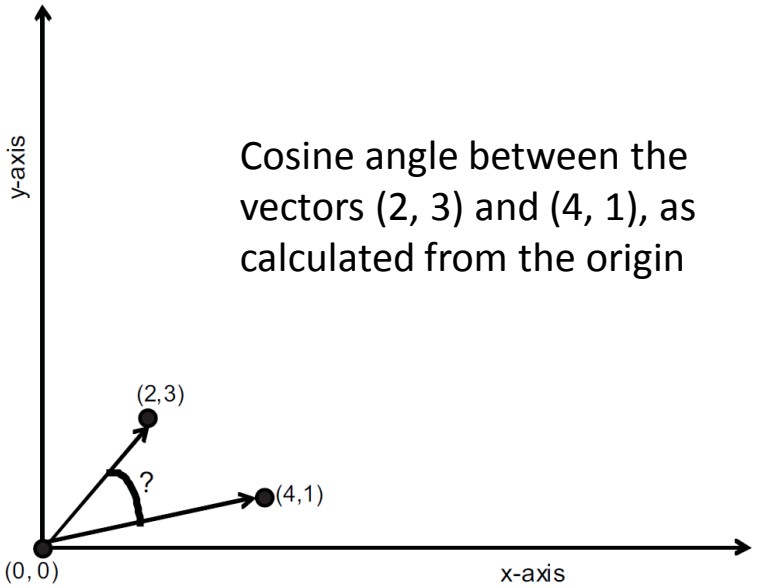
- **Manhattan distance measure**
  - The sum of the absolute differences

$$d = |x_1 - y_1| + |x_2 - y_2| + \ldots + |x_n - y_n|$$

- **Cosine distance measure**
  - Direction difference

$$d = 1 - \frac{(x_1 y_1 + x_2 y_2 + \ldots + x_n y_n)}{\left(\sqrt{x_1^2 + x_2^2 + \ldots + x_n^2}\right)\left(\sqrt{y_1^2 + y_2^2 + \ldots + y_n^2}\right)}$$





Cosine angle between the vectors (2, 3) and (4, 1), as calculated from the origin

Testing Different Distance Measurements

- **Tanimoto distance measure**
  - Captures the information about the angle and the relative distance between the points

$$d = 1 - \frac{\left(x_1 y_1 + x_2 y_2 + \ldots + x_n y_n\right)}{\left(\sqrt{x_1^2 + x_2^2 + \ldots + x_n^2}\right) + \left(\sqrt{y_1^2 + y_2^2 + \ldots + y_n^2}\right) - \left(x_1 y_1 + x_2 y_2 + \ldots + x_n y_n\right)}$$

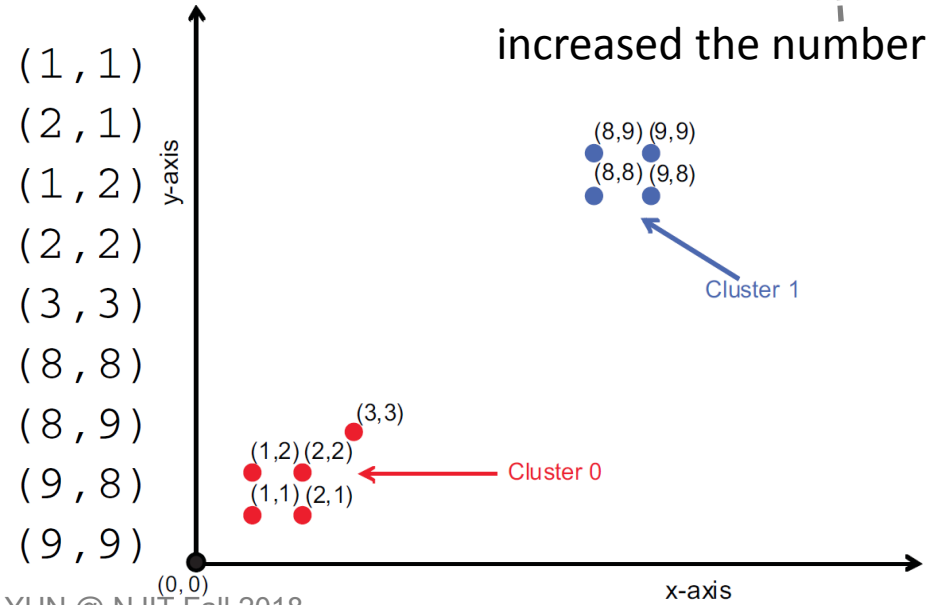
- **Weighted distance measure**
  - Mahout also provides a `WeightedDistanceMeasure` class, and implementations of Euclidean and Manhattan distance measures that use it.
  - A weighted distance measure is an advanced feature in Mahout that allows you to give weights to different dimensions in order to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

Result of clustering using various 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

increased the number of iterations

these integer number are the names/IDs of the given data points



Cosine distance measure:

- Does not work well for this dataset (not necessarily bad), in domain such as text clustering, it can work well

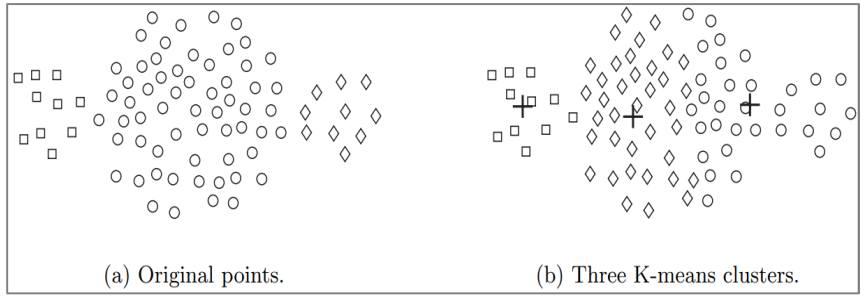
K-means -- Weakness

- **The number of cluster k must be specified in advance**
  - k is an input parameter; an inappropriate choice of k may yield poor results
  - No global theoretical method to find the optimal k – compare the outcomes of multiple runs with different k and then choose the best one based on some predefined criterion
  - In general, a large k probably decreases the error but increases the risk of overfitting
- **Sensitive to initial centroids selection, which leads to unwanted solution**
  - Random choice of initial cluster centers is one of the major drawback
  - Different runs on the same dataset may start with different initial centers – lead to different results – hard to reproduce the results
  - Run many times, starting with different random centroids each time – compare a numerical measure, e.g., distortion (the sum of the squared differences between each data point and its corresponding centroid) – pick the clustering that gave the lowest cost (distortion)
  - K-means++ algorithm
- **K-means can only handle numerical data**
  - Categorical data needs to be encoded or separated in a way that can still work with the algorithm

https://www.datascience.com/blog/k-means-clustering
http://www.saedsayad.com/clustering_kmeans.htm
F. Provost and T. Fawcett, Data Science for Business, O'Reilly Media, 2013
https://www.youtube.com/watch?v=PpH_hv55GNQ
Tan, Steinbach, and Kumar. Introduction to Data Mining

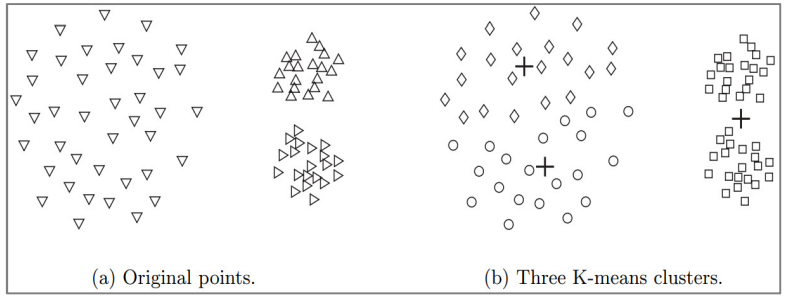
K-means -- Weakness

- **The algorithm may get stuck in the local optima**
  - Given enough time, K-means always converge, although to a local optima (namely, a locally best clustering), which highly depends on the initialization of the centroids
- **Sensitive to outliers and noise – results in an inaccurate partition**
  - When the squared error criterion is used, outliers can unduly influence the clusters that are found
  - Arithmetic mean is not robust to outliers – very far data from the centroid may pull it away from the real one
  - It is often helpful to discover outliers and eliminate them beforehand – unless the outliers are of the primary interest, e.g., unusually profitable customers may be the most interesting points
- K-means cannot handle non-globular clusters or clusters of different sizes and densities
  - K-means has difficulty detecting the "natural" clusters, when clusters have non-spherical shapes or widely different sizes or densities



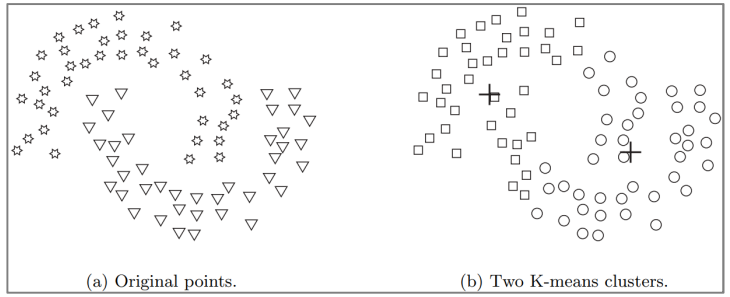
K-means with clusters of different sizes

K-means cannot find the three natural clusters because one of the clusters is much larger than the other two, and hence, the larger cluster is broken, while one of the smaller clusters is combined with a portion of the larger cluster



K-means with clusters of different densities

K-means fails to find the three natural clusters because the two smaller clusters are much denser than the larger cluster



K-means with non-globular clusters

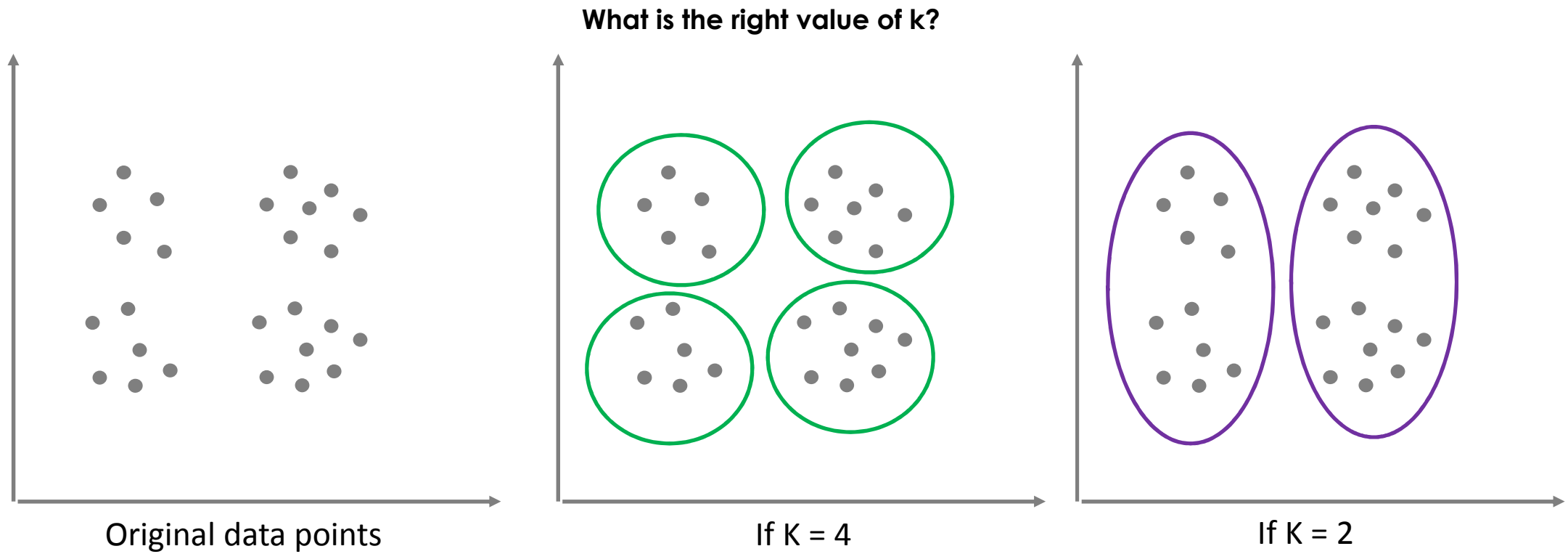
K-means finds two clusters that mix portions of the two natural clusters because the shape of the natural clusters is not globular

http://scikit-learn.org/stable/modules/clustering.html https://www.youtube.com/watch?v=PpH_hv55GNQ

Tan, Steinbach, and Kumar. Introduction to Data Mining

K-means – How to choose the value of *k*?

- k must be chosen for k-means – still an active research area and no definitive answer
- The most common way of choosing k is still by **manually** looking at visualizations or the output of the clustering algorithm
- Why is it not easy to determine the right value of k? -- because it is often generally ambiguous about how many clusters there are in the dataset

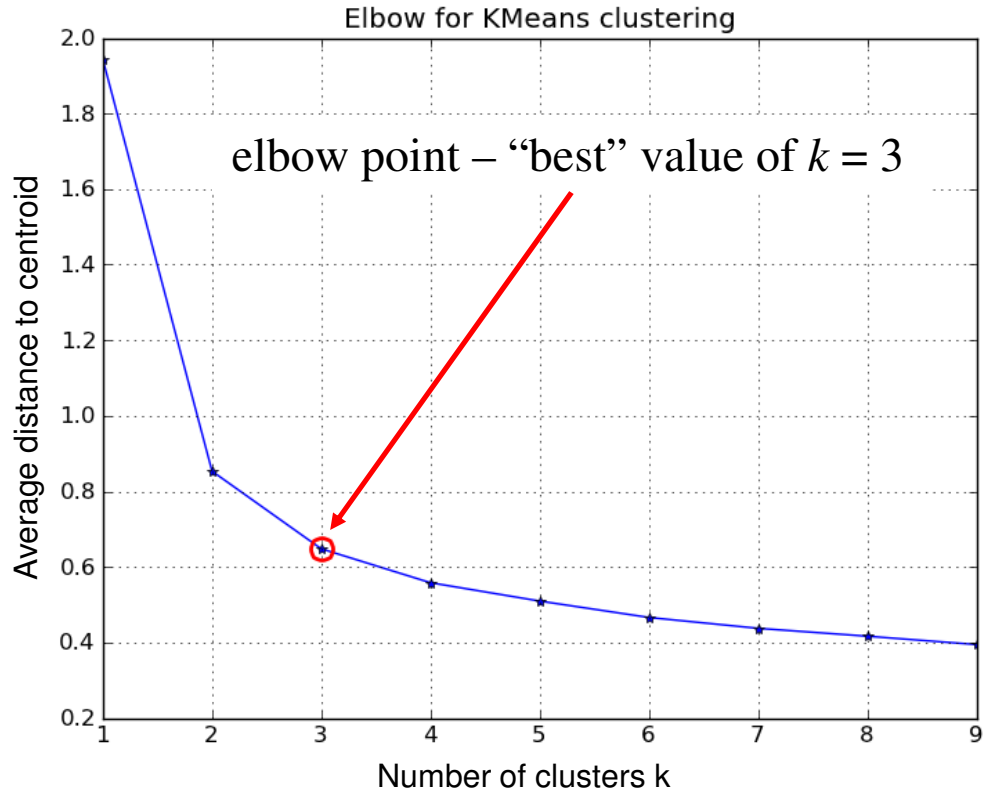


K-means – How to choose the value of *k*?

In general, there is no method to determine exact value of k, but an accurate estimate might be obtained

Elbow Method

- Try different k, looking at the change in the average distance to the centroid, as k increases?
- Average falls rapidly until right k, then falls much more slowly
- It might be hard to determine where the elbow of the curve actually is



☹

Unfortunately, the elbow method often does not work well in practice because there may not be a well-defined elbow

The silhouette value is a measure of how similar an object is to its own cluster (cohesion) compared to other clusters (separation). Silhouette value lies in the range of [-1, 1]. Silhouette coefficients near +1 indicate that the sample is far away from the neighboring clusters

http://scikit-learn.org/stable/auto_examples/cluster/plot_kmeans_silhouette_analysis.html

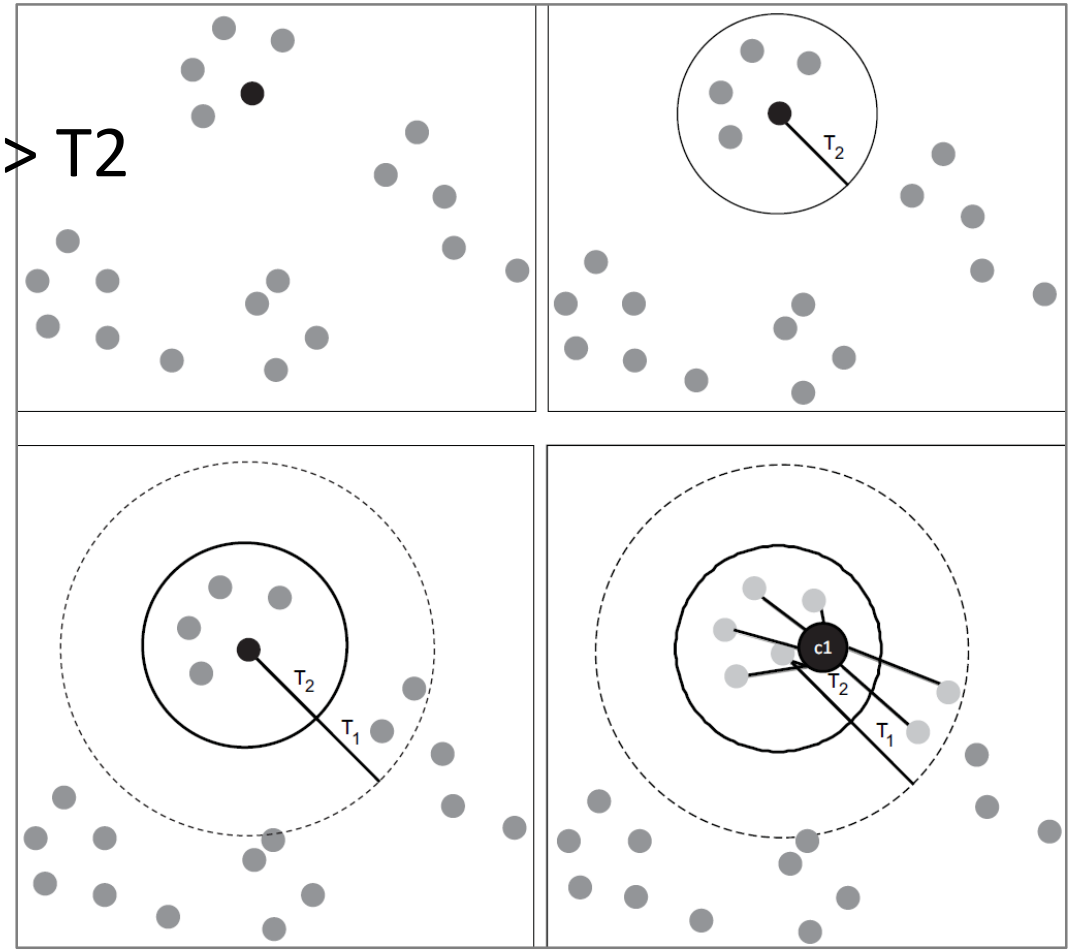
https://www.youtube.com/watch?v=lbR5br5yvrY

https://youtu.be/5TPldC_dC0s?list=PLmNPvQr9Tf-ZSDLwOzxpvY-HrE0yv-8Fy

Canopy Clustering to Estimate the Number of Clusters

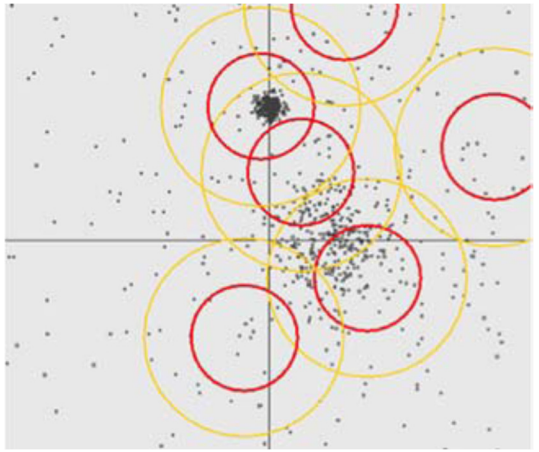
- Tell what size clusters to look for
- The algorithm will find the number of clusters that have approximately that size
- The algorithm uses two distance thresholds
- This method prevents all points close to an already existing canopy from being the center of a new canopy

T1 > T2



Canopy clustering:

- If you start with a point (top left) and mark it as part of a canopy, all the points within distance T2 (top right) are removed from the data set and prevented from becoming new canopies.
- The points within the outer circle (bottom-left) 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.



Mahout example of in-memory canopy generation visualized using the DisplayCanopy class, started with T1=3.0 and T2=1.5 and clustered the randomly generated points.

Run 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
```

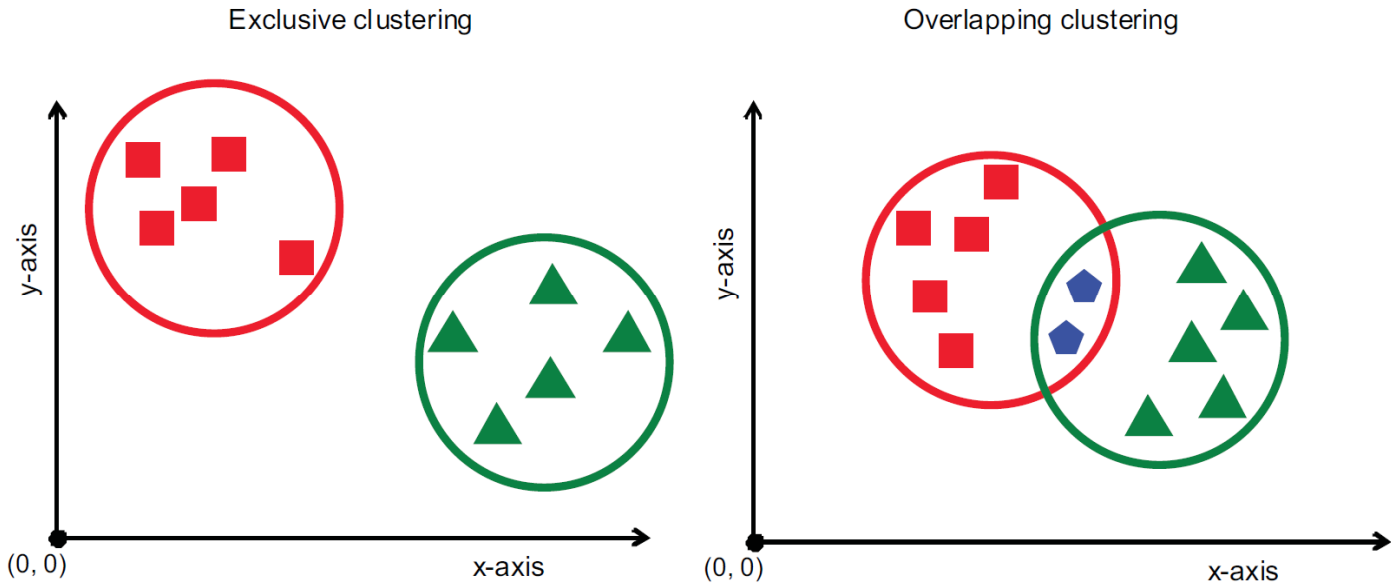
- Within a minute, `CanopyDriver` will generate the centroids in the output folder.
- Next, use these centroids to improve k-means clustering, use `TanimotoDistanceMeasure`

```
$ 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
```

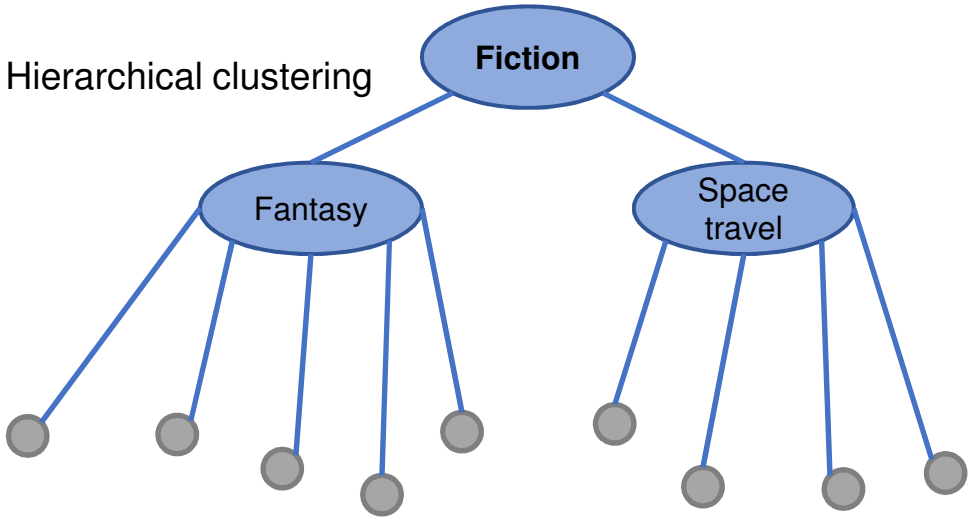
- After the clustering is done, use `ClusterDumper` to inspect the clusters. Some of them are listed here:

```
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
```

Other Clustering Algorithms

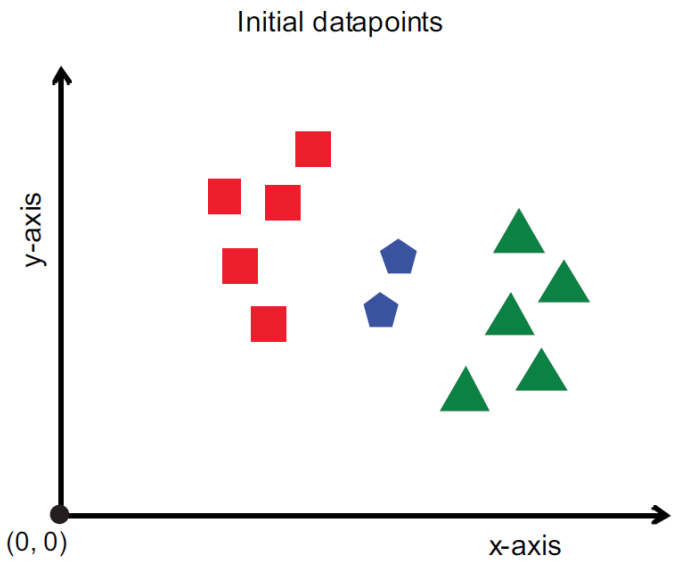


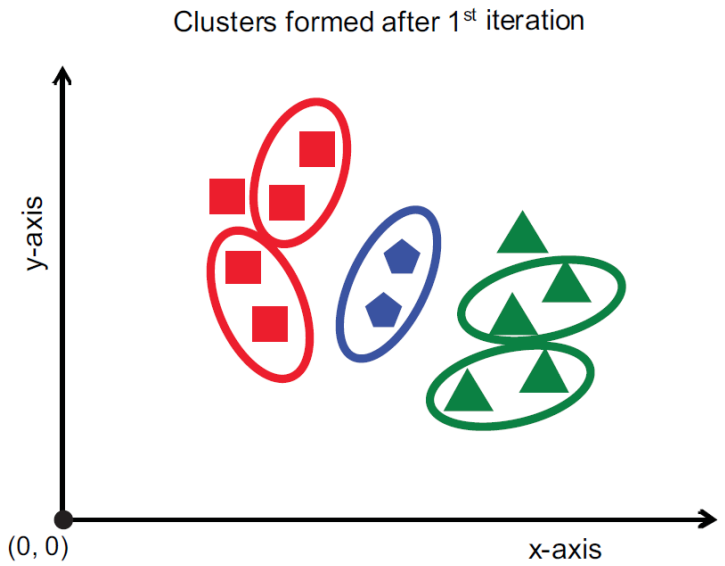
- In exclusive clustering, squares and triangles each have their own cluster, and each belongs only to one cluster.
- In overlapping clustering, some shapes, like pentagons, can belong to both clusters with some probability, so they're part of both clusters instead of having a cluster of their own.



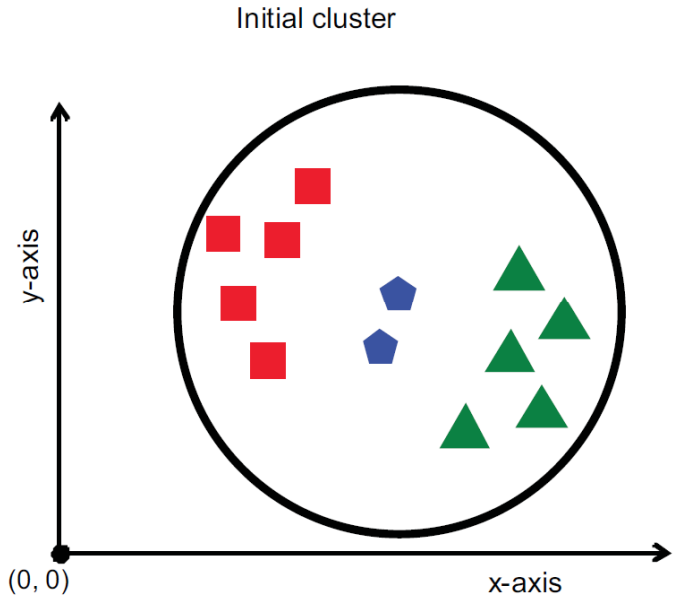
- Construct a fiction cluster by merging fantasy cluster and space travel cluster and other similar clusters

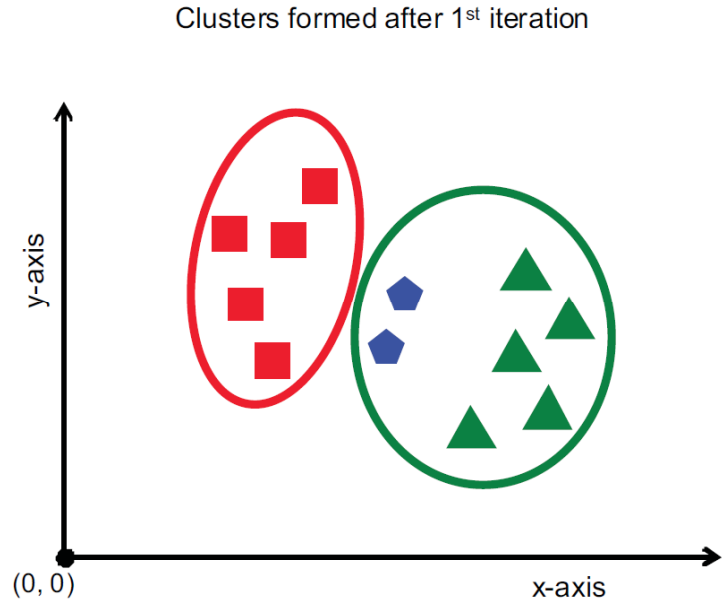
Different Clustering Approaches





Bottom-up clustering: after every iteration the clusters are merged to produce larger and larger clusters until it's not feasible to merge based on the given distance measure.





Top-down clustering: during each iteration, the clusters are divided into two by finding the best splitting until you get the clusters you desire.

Reading

- Chapter 8 of *Introduction to Data Mining* by P. Tan et al.
- Part I and Part II of *Mahout in Action* by S. Owen et al.

Thanks ! ☺

Questions ?