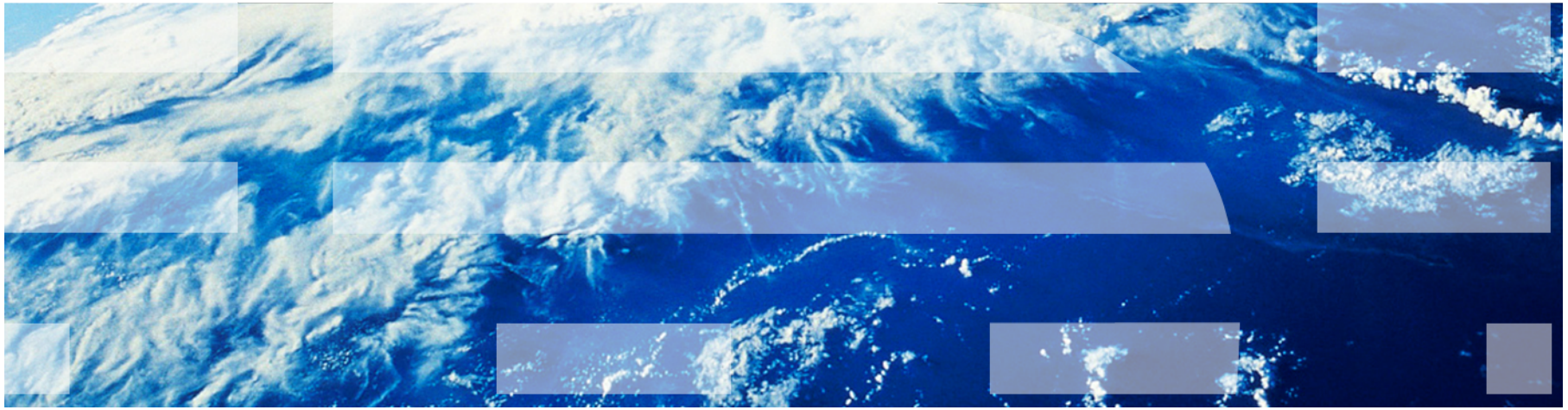
Columbia University

E6893 Big Data Analytics Lecture 3:

Big Data Analytics Algorithms — I

Ching-Yung Lin, Ph.D.

Adjunct Professor, Dept. of Electrical Engineering and Computer Science



September 20, 2019

Analytics 1 — Recommendation

Key Components of Mahout in Hadoop

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

Key Components of Spark MLlib

Columbia University

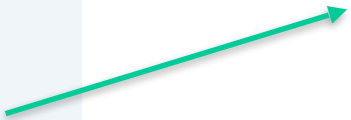
MLlib: Main Guide

- Basic statistics
- Pipelines
- Extracting, transforming and selecting features
- Classification and Regression
- Clustering
- Collaborative filtering
- Frequent Pattern Mining
- Model selection and tuning
- Advanced topics

Spark ML Basic Statistics

MLlib: Main Guide

- Basic statistics
- Pipelines
- Extracting, transforming and selecting features
- Classification and Regression
- Clustering
- Collaborative filtering
- Frequent Pattern Mining
- Model selection and tuning
- Advanced topics



- Correlation
- Hypothesis testing

- Correlation: Calculating the correlation between two series of data is a common operation in Statistics
  - Pearson's Correlation
  - Spearman's Correlation

Example of Popular Similarity Measurements

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

Pearson Correlation Similarity

$$\rho_{X,Y} = \frac{\operatorname{cov}(X,Y)}{\sigma_X \sigma_Y} = \frac{E[(X-\mu_X)(Y-\mu_Y)]}{\sigma_X \sigma_Y}$$

Data:

```
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	1.000
User 2	2.0	2.5	5.0	-0.764
User 3	2.5	-	-	-
User 4	5.0	-	3.0	1.000
User 5	4.0	3.0	2.0	0.945

missing data

On Pearson Similarity

Three problems with the Pearson Similarity:

1. 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.)
2. If two users overlap on only one item, no correlation can be computed.
3. 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.

Spearman Correlation Similarity

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

$$d_i = x_i - y_i$$

Pearson value on the relative ranks

Example for ties

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

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

Basic Spark Data format

Data: 1.0, 0.0, 3.0

// straightforward

```
scala> val dv: Vector = Vectors.dense(1.0, 0.0, 3.0)
dv: org.apache.spark.mllib.linalg.Vector = [1.0,0.0,3.0]
```

// number of parameters, location of non-zero indices, and non-zero values

```
scala> val sv1: Vector = Vectors.sparse(3, Array(0, 2), Array(1.0, 3.0))
sv1: org.apache.spark.mllib.linalg.Vector = (3,[0,2],[1.0,3.0])
```

// number of parameters, Sequence of non-value values (index, value)

```
scala> val sv2: Vector = Vectors.sparse(3, Seq((0, 1.0), (2, 3.0)))
sv2: org.apache.spark.mllib.linalg.Vector = (3,[0,2],[1.0,3.0])
```

Correlation Example in Spark

1.0, 0.0, 0.0, -2.0
4.0, 5.0, 0.0, 3.0
6.0, 7.0, 0.0, 8.0
9.0, 0.0, 0.0, 1.0

```
from pyspark.ml.linalg import Vectors
from pyspark.ml.stat import Correlation

data = [(Vectors.sparse(4, [(0, 1.0), (3, -2.0)]),),
        (Vectors.dense([4.0, 5.0, 0.0, 3.0]),),
        (Vectors.dense([6.0, 7.0, 0.0, 8.0]),),
        (Vectors.sparse(4, [(0, 9.0), (3, 1.0)]),)]
df = spark.createDataFrame(data, ["features"])

r1 = Correlation.corr(df, "features").head()
print("Pearson correlation matrix:\n" + str(r1[0]))

r2 = Correlation.corr(df, "features", "spearman").head()
print("Spearman correlation matrix:\n" + str(r2[0]))
```

Euclidean Distance Similarity

$$\mathrm{d}(\mathbf{p},\mathbf{q}) = \mathrm{d}(\mathbf{q},\mathbf{p}) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2 + \cdots + (q_n - p_n)^2} = \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2}.$$

Similarity = 1 / (1 + d)

	Item 101	Item 102	Item 103	Distance	Similarity to user 1
User 1	5.0	3.0	2.5	0.000	1.000
User 2	2.0	2.5	5.0	3.937	0.203
User 3	2.5	-	-	2.500	0.286
User 4	5.0	-	3.0	0.500	0.667
User 5	4.0	3.0	2.0	1.118	0.472

Cosine Similarity

$$\text{similarity} = \cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n} (A_i)^2} \times \sqrt{\sum_{i=1}^{n} (B_i)^2}}$$

Cosine similarity and Pearson similarity get the same results if data are normalized (mean == 0).

Caching User Similarity

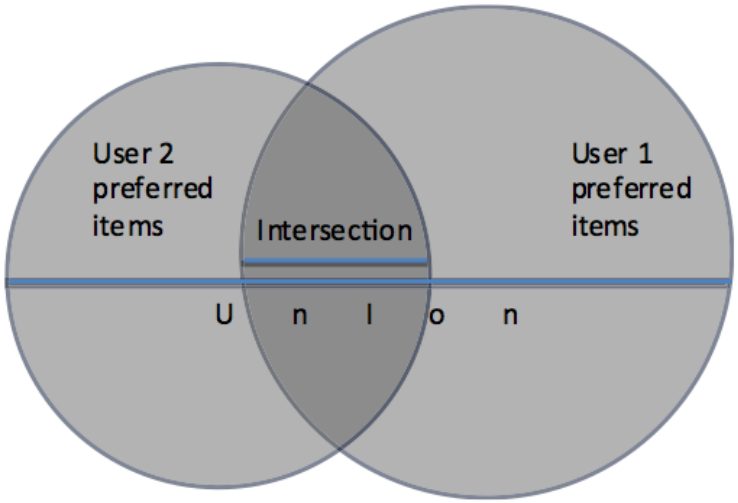
```
UserSimilarity similarity = new CachingUserSimilarity(
    new SpearmanCorrelationSimilarity(model), model);
```

Spearman Correlation Similarity is time consuming.
Need to use Caching ==> remember s user-user similarity which was previously computed.

Tanimoto (Jaccard) Coefficient Similarity

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}$$

Discard preference values



	Item 101	Item 102	Item 103	Item 104	Item 105	Item 106	Item 107	Similarity to user 1
User 1	X	X	X					1.0
User 2	X	X	X	X				0.75
User 3	X			X	X		X	0.17
User 4	X		X	X		X		0.4
User 5	X	X	X	X	X	X		0.5

Tanimoto similarity is the same as Jaccard similarity. But, Tanimoto distance is not the same as Jaccard distance.

Log-Likelihood Similarity

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

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

	Item 101	Item 102	Item 103	Item 104	Item 105	Item 106	Item 107	Similarity to user 1
User 1	X	X	X					0.90
User 2	X	X	X	X				0.84
User 3	X			X	X		X	0.55
User 4	X		X	X		X		0.16
User 5	X	X	X	X	X	X		0.55

Performance measurements

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

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

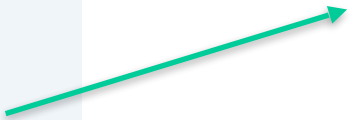
	Item 1	Item 2	Item 3
Actual	3.0	5.0	4.0
Estimate	3.5	2.0	5.0
Difference	0.5	3.0	1.0
Average difference	$= (0.5 + 3.0 + 1.0) / 3 = 1.5$		
Root-mean-square	$= \sqrt{((0.5^2 + 3.0^2 + 1.0^2) / 3)} = 1.8484$		

Spark ML Basic Statistics

Columbia University

MLlib: Main Guide

- Basic statistics
- Pipelines
- Extracting, transforming and selecting features
- Classification and Regression
- Clustering
- Collaborative filtering
- Frequent Pattern Mining
- Model selection and tuning
- Advanced topics



- Correlation
- Hypothesis testing

- Hypothesis testing: Hypothesis testing is a powerful tool in statistics to determine whether a result is statistically significant. Spark ML currently supports Pearson's Chi-squared ($\chi 2$) tests for independence.
- `ChiSquareTest` conducts Pearson's independence test for every feature against the label.

Chi-Square Tests

Table . Incidence of three types of malaria in three tropical regions.

	Asia	Africa	South America	Totals
Malaria A	31	14	45	90
Malaria B	2	5	53	60
Malaria C	53	45	2	100
Totals	86	64	100	250

http://math.hws.edu/javamath/ryan/ChiSquare.html

Chi-Square Tests

Observed	Expected	|O -E|	(O — E)2	(O — E)2/ E
31	30.96	0.04	0.0016	0.0000516
14	23.04	9.04	81.72	3.546
45	36.00	9.00	81.00	2.25
2	20.64	18.64	347.45	16.83
5	15.36	10.36	107.33	6.99
53	24.00	29.00	841.00	35.04
53	34.40	18.60	345.96	10.06
45	25.60	19.40	376.36	14.70
2	40.00	38.00	1444.00	36.10

Chi Square = 125.516

Degrees of Freedom = (c - 1)(r - 1) = 2(2) = 4

http://math.hws.edu/javamath/ryan/ChiSquare.html

Chi-Square Tests

probability level (alpha)

Df	0.5	0.10	0.05	0.02	0.01	0.001
1	0.455	2.706	3.841	5.412	6.635	10.827
2	1.386	4.605	5.991	7.824	9.210	13.815
3	2.366	6.251	7.815	9.837	11.345	16.268
4	3.357	7.779	9.488	11.668	13.277	18.465
5	4.351	9.236	11.070	13.388	15.086	20.517

Reject Ho because 125.516 is greater than 9.488 (for alpha = 0.05)

We would reject the null hypothesis that there is no relationship between location and type of malaria. Our data tell us there is a relationship between type of malaria and location.

http://math.hws.edu/javamath/ryan/ChiSquare.html

Chi-Square Tests in Spark

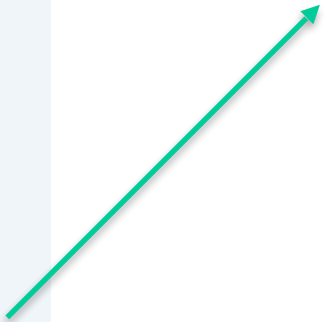
```
from pyspark.ml.linalg import Vectors
from pyspark.ml.stat import ChiSquareTest

data = [(0.0, Vectors.dense(0.5, 10.0)),
        (0.0, Vectors.dense(1.5, 20.0)),
        (1.0, Vectors.dense(1.5, 30.0)),
        (0.0, Vectors.dense(3.5, 30.0)),
        (0.0, Vectors.dense(3.5, 40.0)),
        (1.0, Vectors.dense(3.5, 40.0))]
df = spark.createDataFrame(data, ["label", "features"])

r = ChiSquareTest.test(df, "features", "label").head()
print("pValues: " + str(r.pValues))
print("degreesOfFreedom: " + str(r.degreesOfFreedom))
print("statistics: " + str(r.statistics))
```

Spark ML Clustering

Columbia University

MLlib: Main Guide

- Basic statistics
- Pipelines
- Extracting, transforming and selecting features
- Classification and Regression
- Clustering
- Collaborative filtering
- Frequent Pattern Mining
- Model selection and tuning
- Advanced topics



- K-means
  - Input Columns
  - Output Columns
- Latent Dirichlet allocation (LDA)
- Bisecting k-means
- Gaussian Mixture Model (GMM)
  - Input Columns
  - Output Columns

Example: clustering







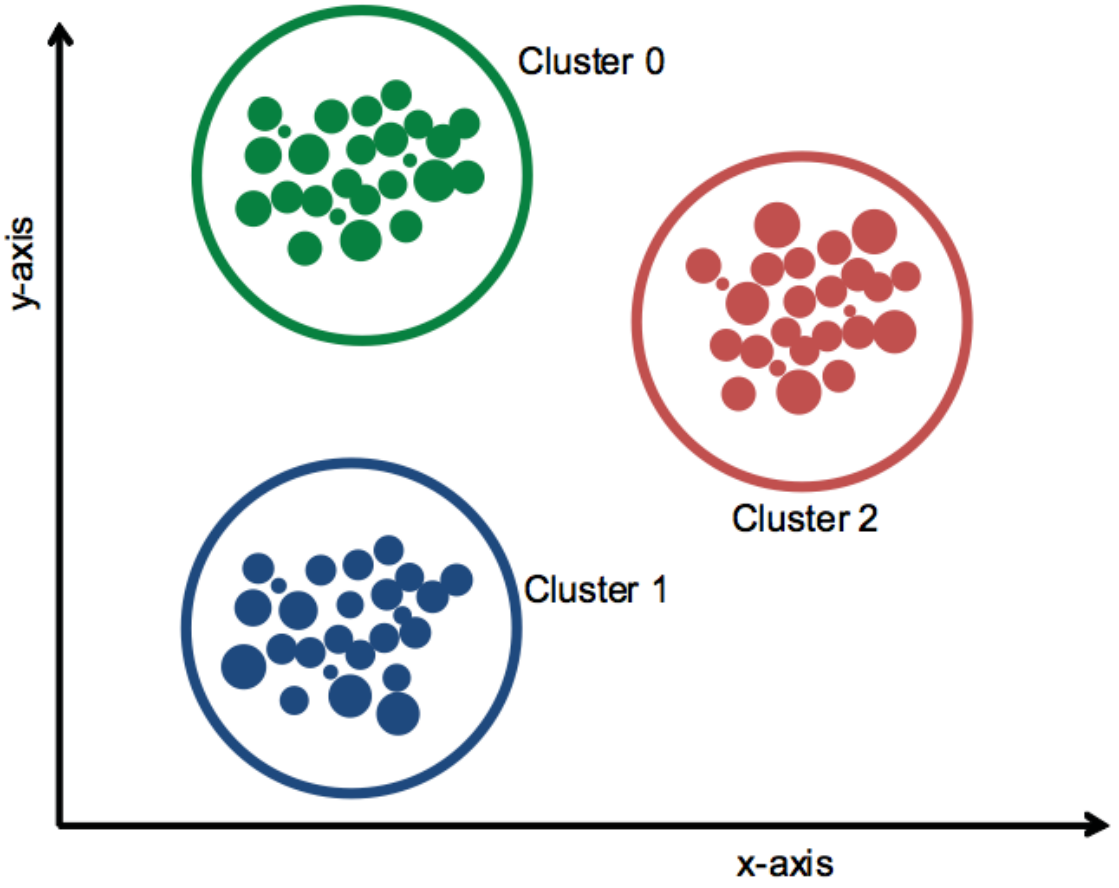

Feature Space

Clustering

Clustering a collection involves three things:

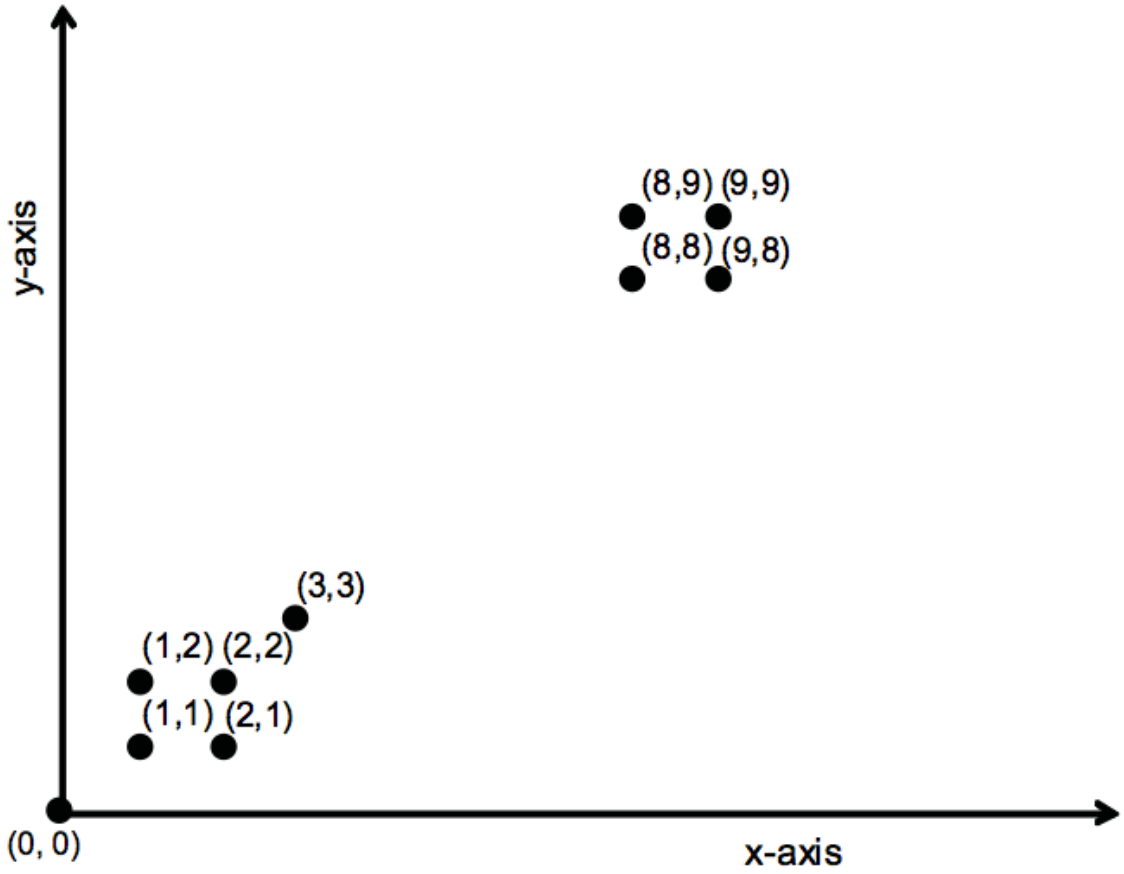
- *An algorithm*—This is the method used to group the books together.
- *A notion of both similarity and dissimilarity*—In the previous discussion, we relied on your assessment of which books belonged in an existing stack and which should start a new one.
- *A stopping condition*—In the library example,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

Clustering — on feature plane

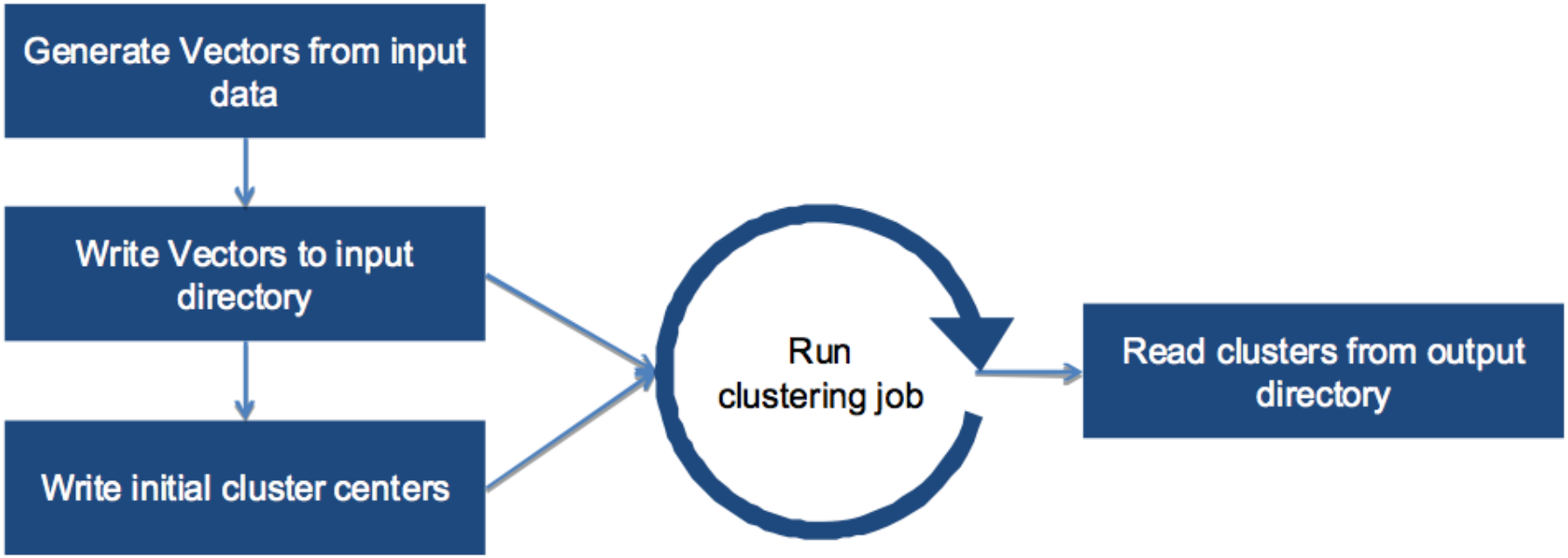


Clustering example

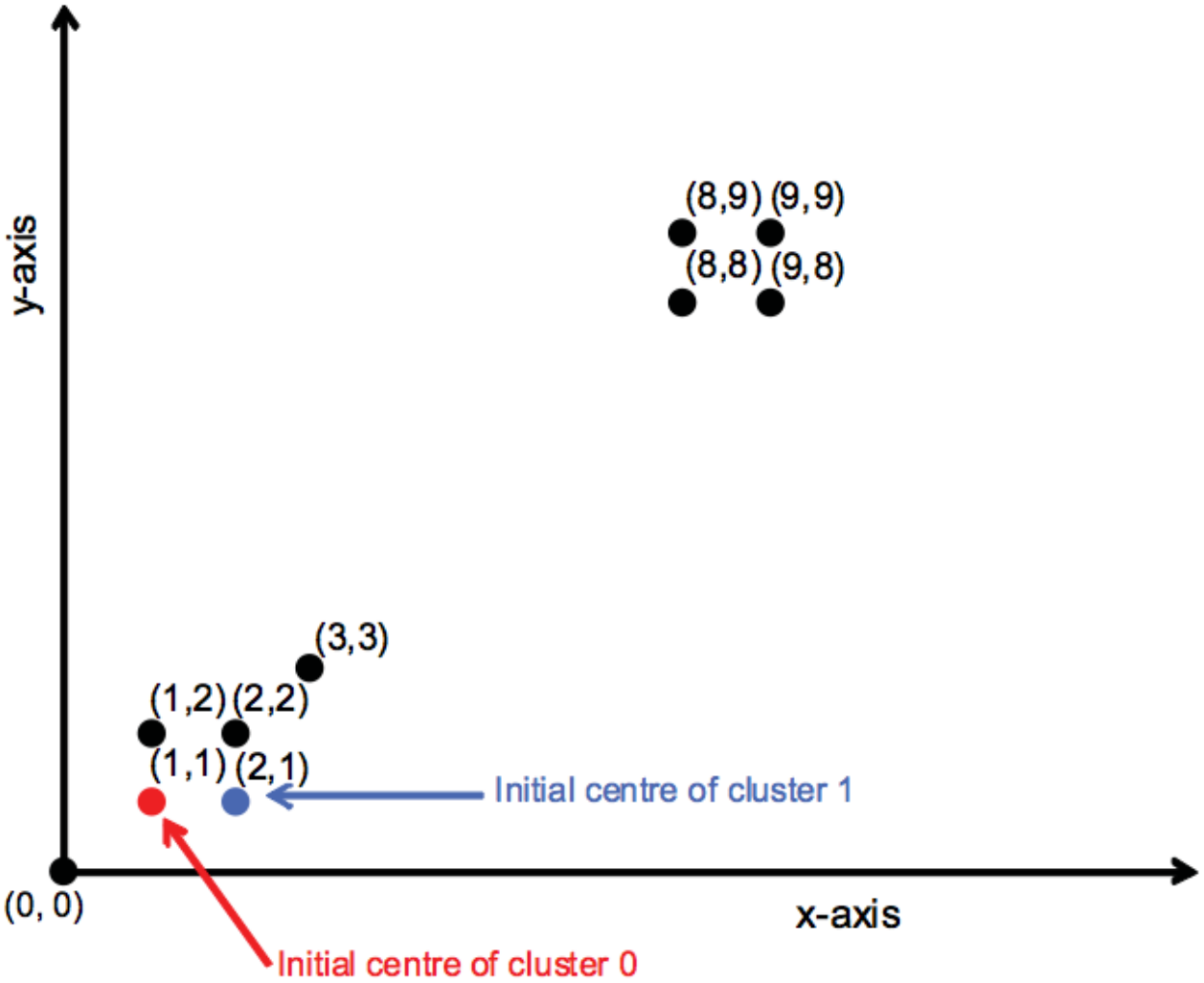
(1,1)
(2,1)
(1,2)
(2,2)
(3,3)
(8,8)
(8,9)
(9,8)
(9,9)



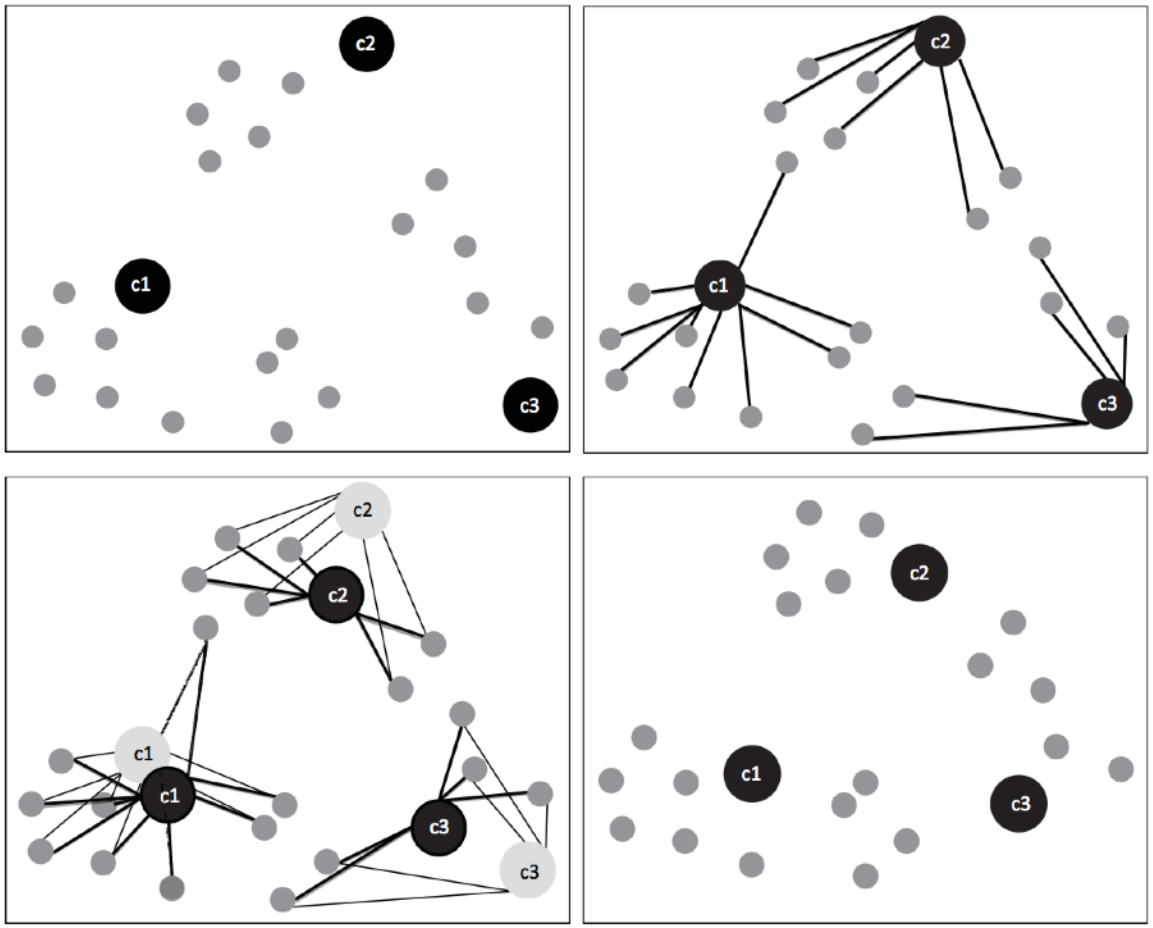
Steps on clustering



Making initial cluster centers



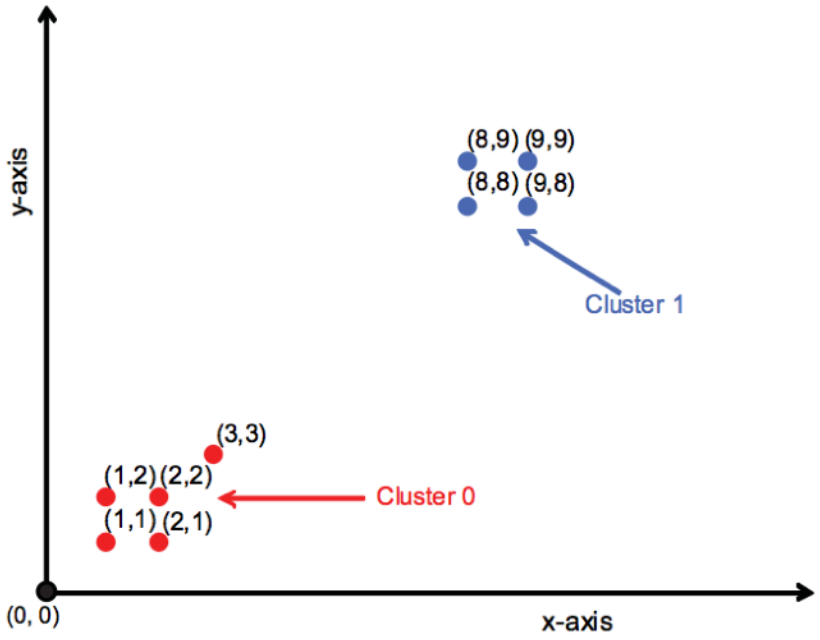
K-mean clustering



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

HelloWorld clustering scenario result

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



Testing difference distance measures

Euclidean distance measure

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

Squared Euclidean distance measure

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

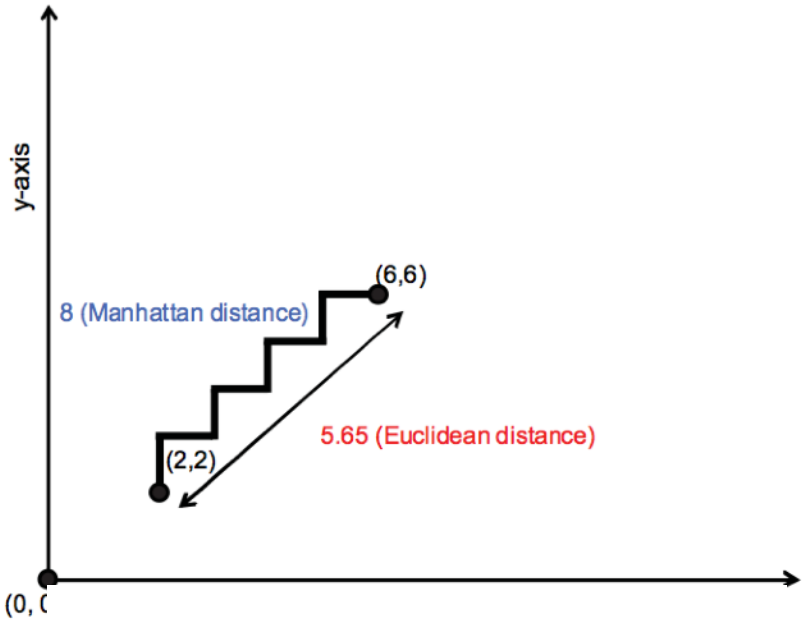
Manhattan distance measure

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

Manhattan and Cosine distances

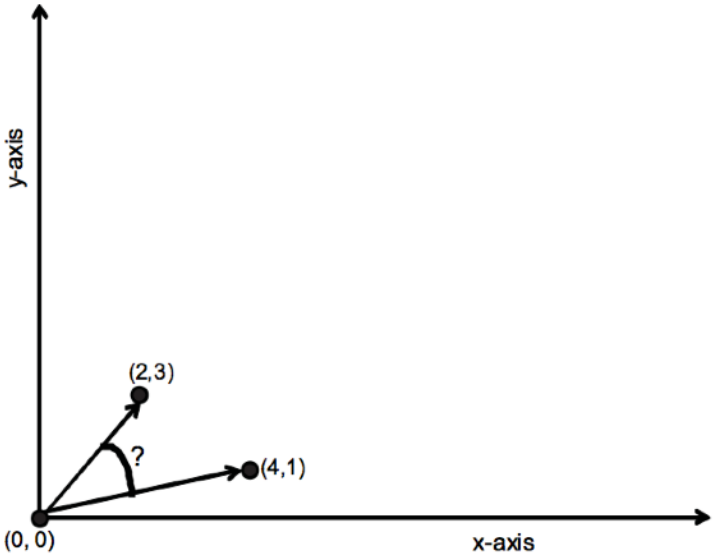
Manhattan distance measure

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



Cosine distance measure

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



Tanimoto distance measure

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

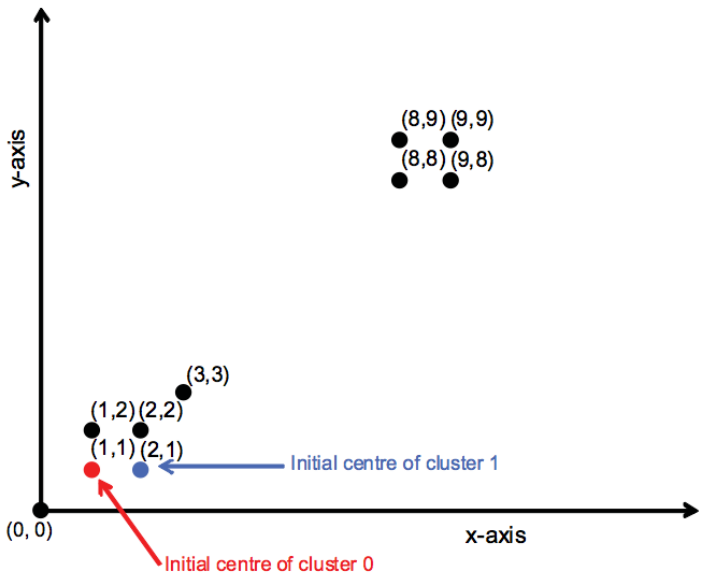
Weighted distance measure

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

Results comparison

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

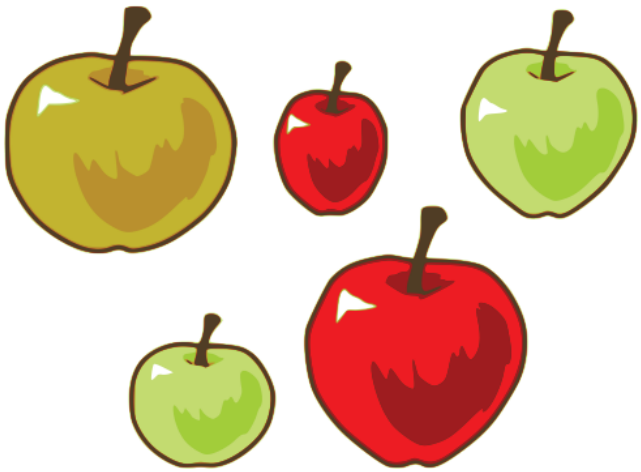
(1,1)
(2,1)
(1,2)
(2,2)
(3,3)
(8,8)
(8,9)
(9,8)
(9,9)



Sample code of K-mean clustering in Spark

```
from pyspark.ml.clustering import KMeans
from pyspark.ml.evaluation import ClusteringEvaluator

# Loads data.
dataset = spark.read.format("libsvm").load("data/mllib/sample_kmeans_data.txt")

# Trains a k-means model.
kmeans = KMeans().setK(2).setSeed(1)
model = kmeans.fit(dataset)

# Make predictions
predictions = model.transform(dataset)

# Evaluate clustering by computing Silhouette score
evaluator = ClusteringEvaluator()

silhouette = evaluator.evaluate(predictions)
print("Silhouette with squared euclidean distance = " + str(silhouette))

# Shows the result.
centers = model.clusterCenters()
print("Cluster Centers: ")
for center in centers:
    print(center)
```

vectorization example



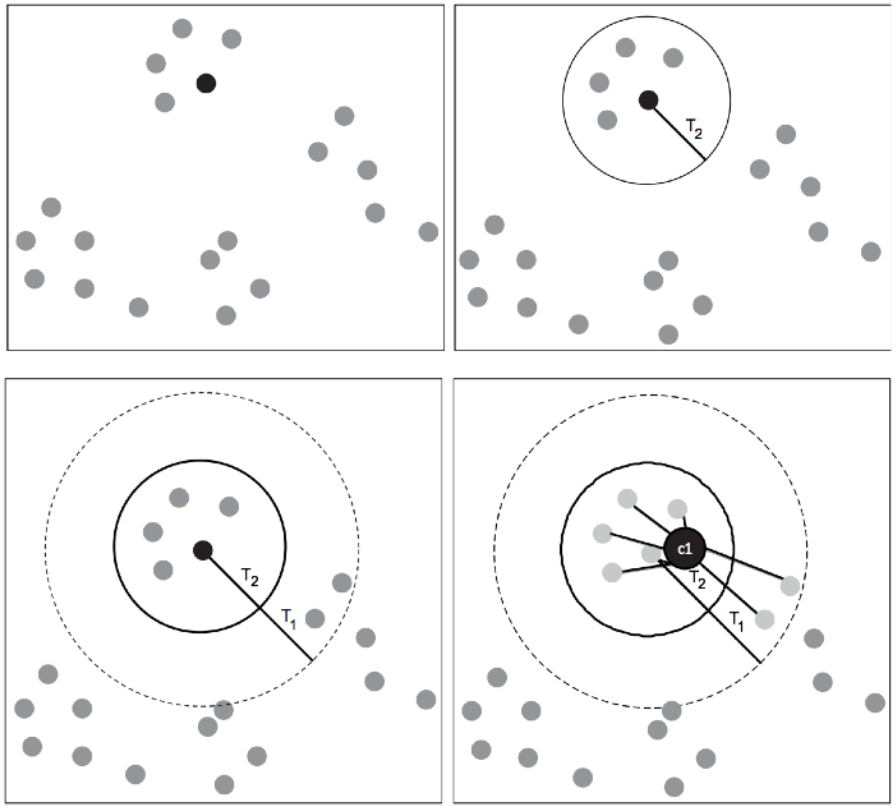
0: weight
1: color
2: size

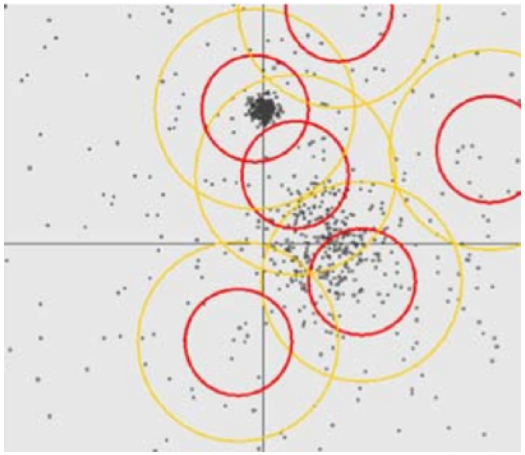
```
[0 => 100 gram, 1 => red, 2 => small]
```

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

Canopy clustering to estimate the number of clusters

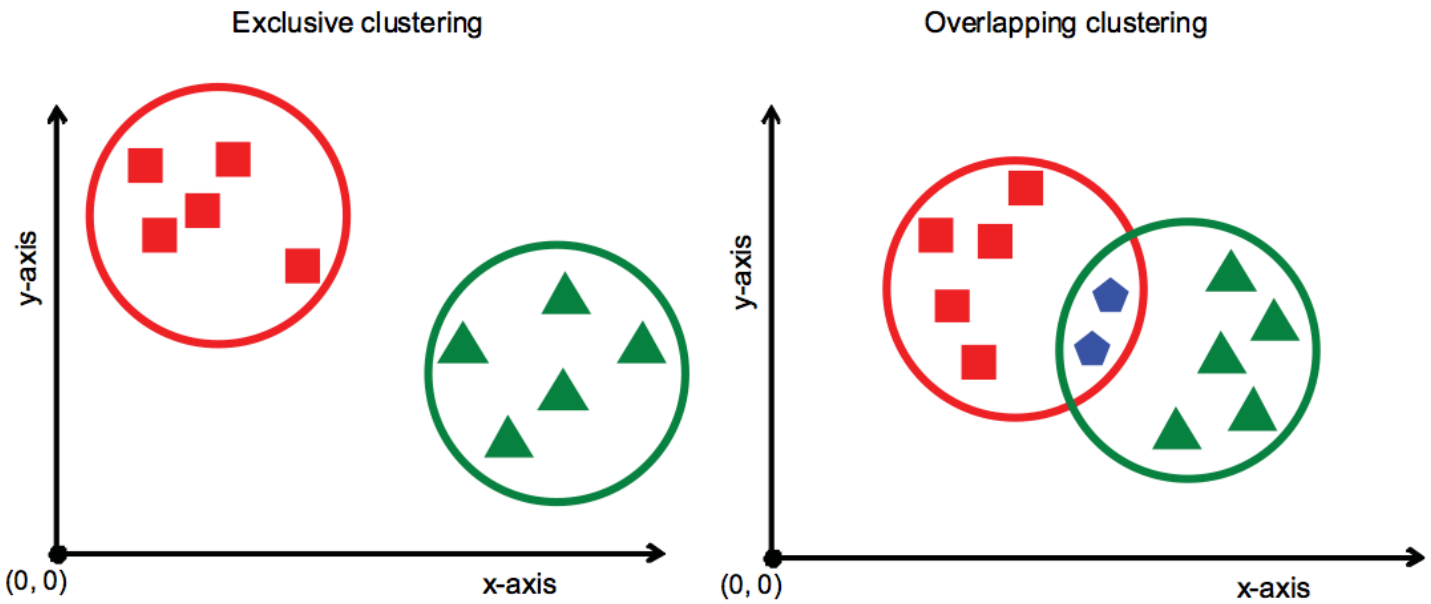
Tell what size clusters to look for. The algorithm will find the number of clusters that have approximately that size. The algorithm uses two distance thresholds. This method prevents all points close to an already existing canopy from being the center of a new canopy.



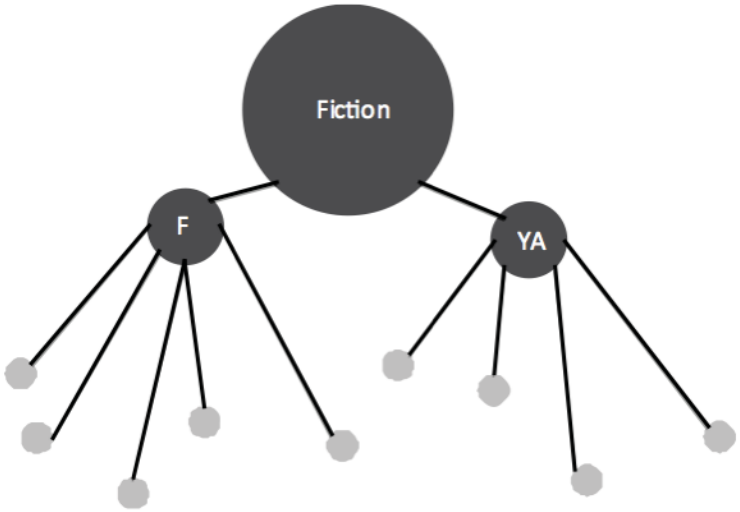


Canopy clustering: if you start with a point (top left) and mark it as part of a canopy, all the points within distance T_2 (top right) are removed from the data set and prevented from becoming new canopies. The points within the outer circle (bottom-right) are also put in the same canopy, but they're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

Other clustering algorithms



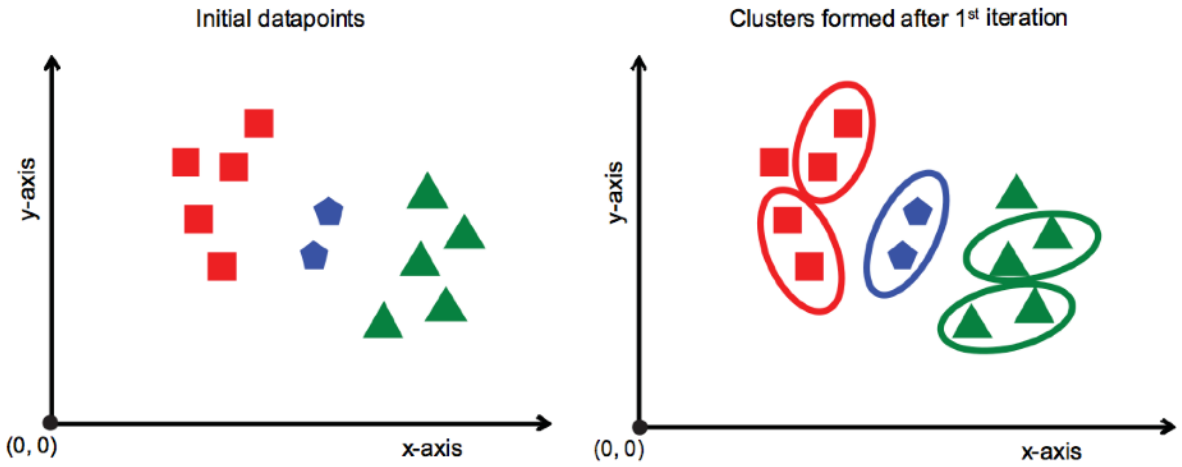
Hierarchical clustering



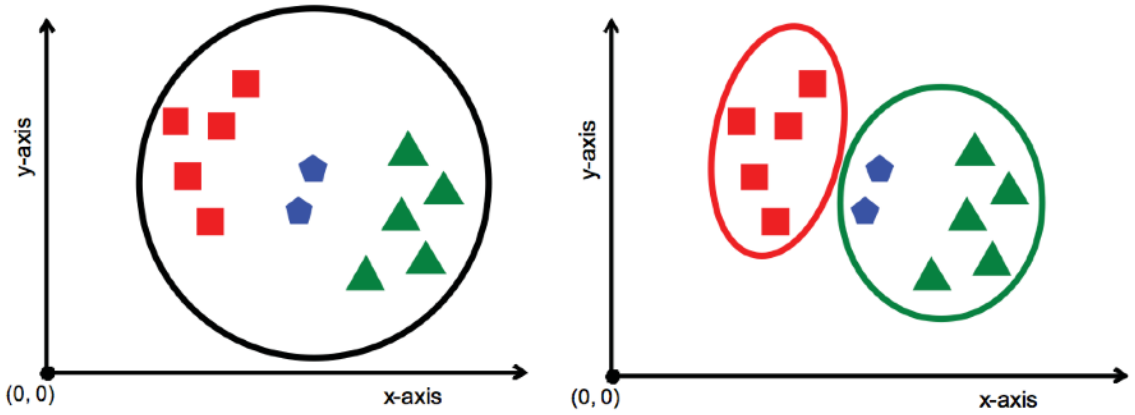
Different clustering approaches

FIXED NUMBER OF CENTERS

BOTTOM-UP APPROACH: FROM POINTS TO CLUSTERS VIA GROUPING



TOP-DOWN APPROACH: SPLITTING THE GIANT CLUSTER



Spark ML Classification and Regression

Columbia University

MLlib: Main Guide

- Basic statistics
- Pipelines
- Extracting, transforming and selecting features
- Classification and Regression
- Clustering
- Collaborative filtering
- Frequent Pattern Mining
- Model selection and tuning
- Advanced topics

- Classification
  - Logistic regression
    - Binomial logistic regression
    - Multinomial logistic regression
  - Decision tree classifier
  - Random forest classifier
  - Gradient-boosted tree classifier
  - Multilayer perceptron classifier
  - Linear Support Vector Machine
  - One-vs-Rest classifier (a.k.a. One-vs-All)
  - Naive Bayes
- Regression
  - Linear regression
  - Generalized linear regression
    - Available families
  - Decision tree regression
  - Random forest regression
  - Gradient-boosted tree regression
  - Survival regression
  - Isotonic regression

MLlib: Main Guide

- Basic statistics
- Pipelines
- Extracting, transforming and selecting features
- Classification and Regression
- Clustering
- Collaborative filtering
- Frequent Pattern Mining
- Model selection and tuning
- Advanced topics

- Linear methods
- Decision trees
  - Inputs and Outputs
    - Input Columns
    - Output Columns
- Tree Ensembles
  - Random Forests
    - Inputs and Outputs
      - Input Columns
      - Output Columns (Predictions)
  - Gradient-Boosted Trees (GBTs)
    - Inputs and Outputs
      - Input Columns
      - Output Columns (Predictions)

Classification — definition

DEFINITION Computer classification systems are a form of machine learning that use learning algorithms to provide a way for computers to make decisions based on experience and, in the process, emulate certain forms of human decision making.

Classification example: using SVM to recognize a Toyota Camry

Non-ML

Rule 1.Symbol has something like bull's head
Rule 2.Big black portion in front of car.
Rule 3. …..????

ML — Support Vector Machine

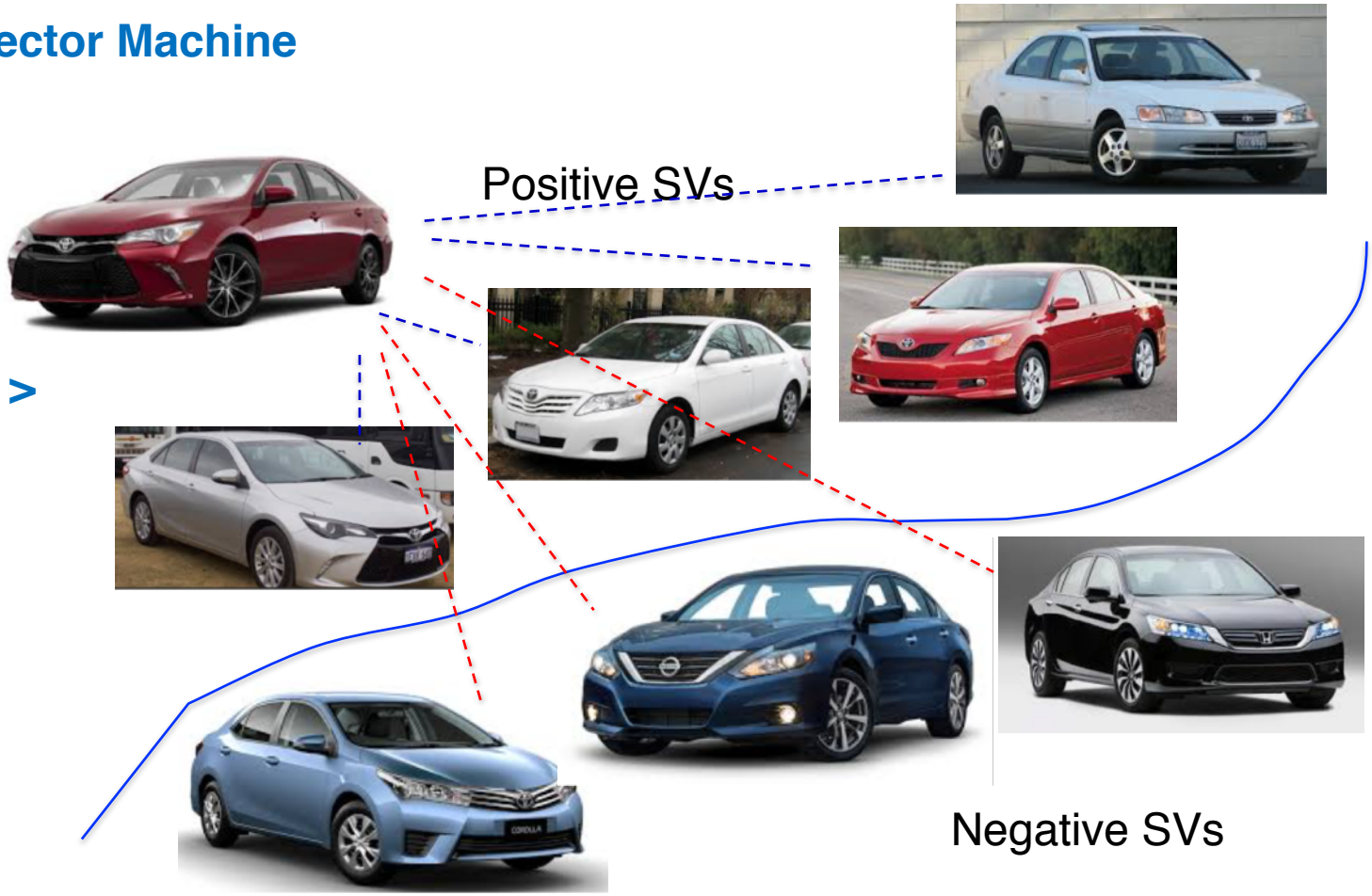
Feature Space

Positive SVs

Negative SVs

Classification example: using SVM to recognize a Toyota Camry

ML — Support Vector Machine

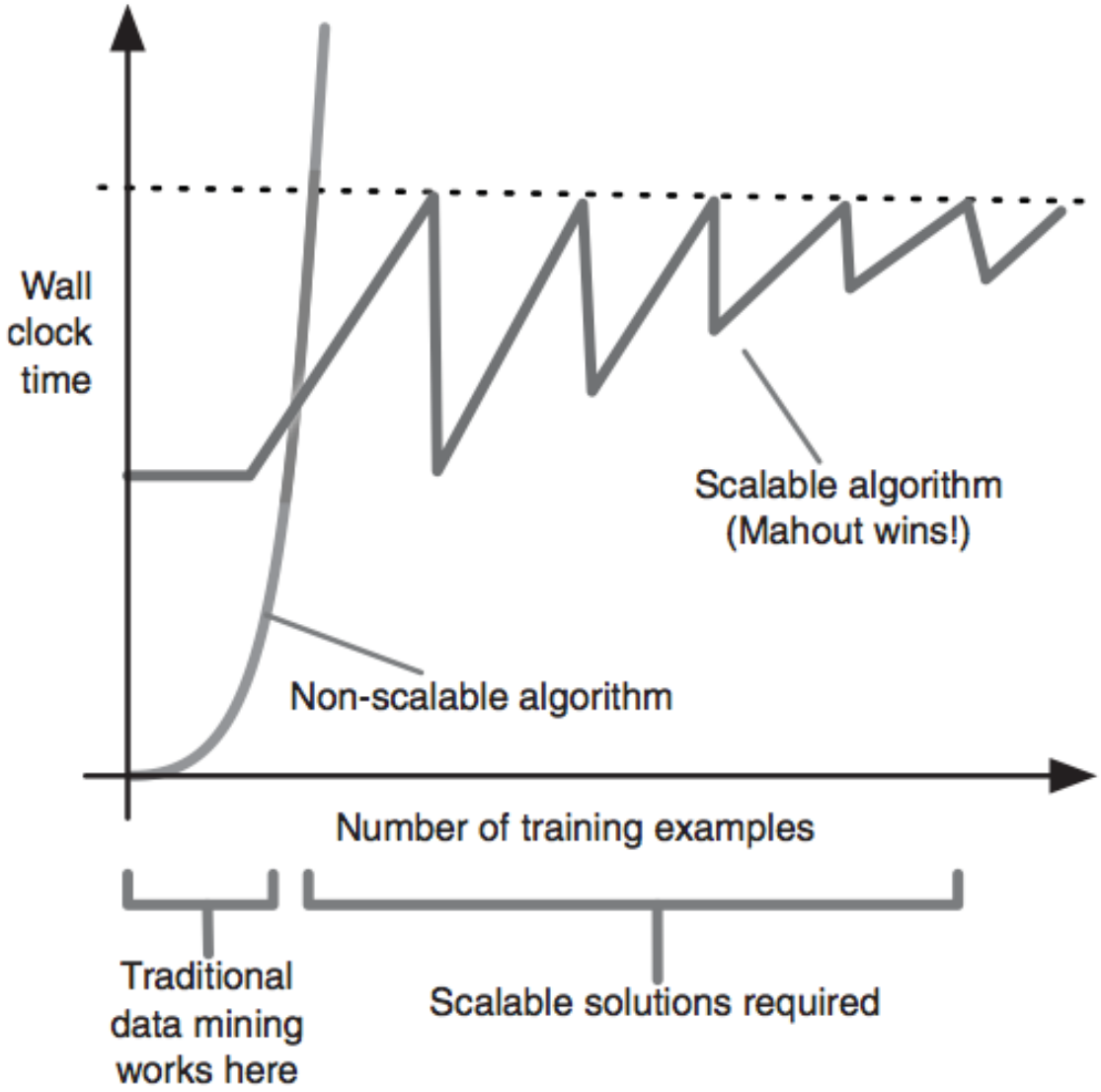


When to use Big Data System for classification?

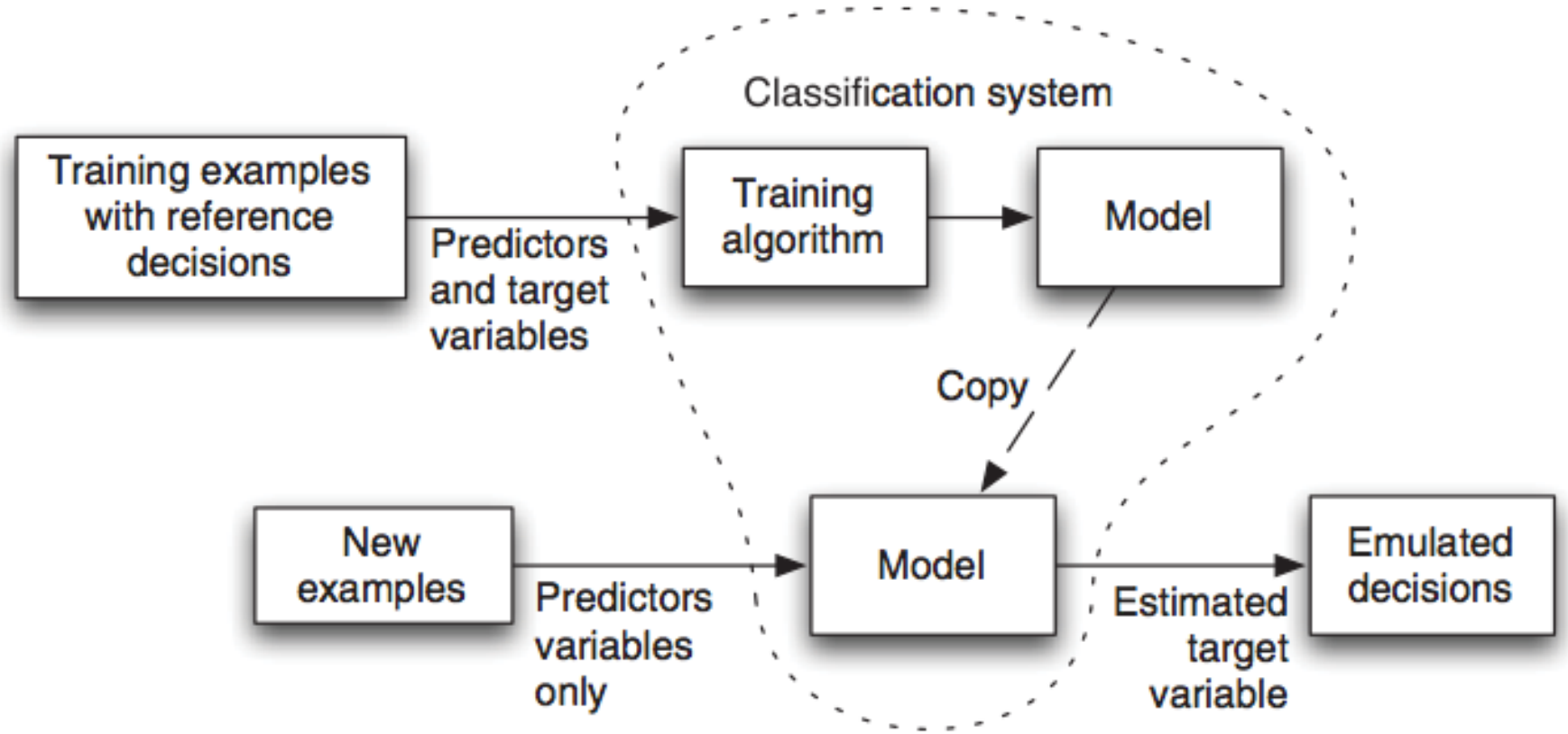
Columbia University

System size in number of examples	Choice of classification approach
< 100,000	Traditional, non-Mahout approaches should work very well. Mahout may even be slower for training.
100,000 to 1 million	Mahout begins to be a good choice. The flexible API may make Mahout a preferred choice, even though there is no performance advantage.
1 million to 10 million	Mahout is an excellent choice in this range.
> 10 million	Mahout excels where others fail.

The advantage of using Big Data System for classification



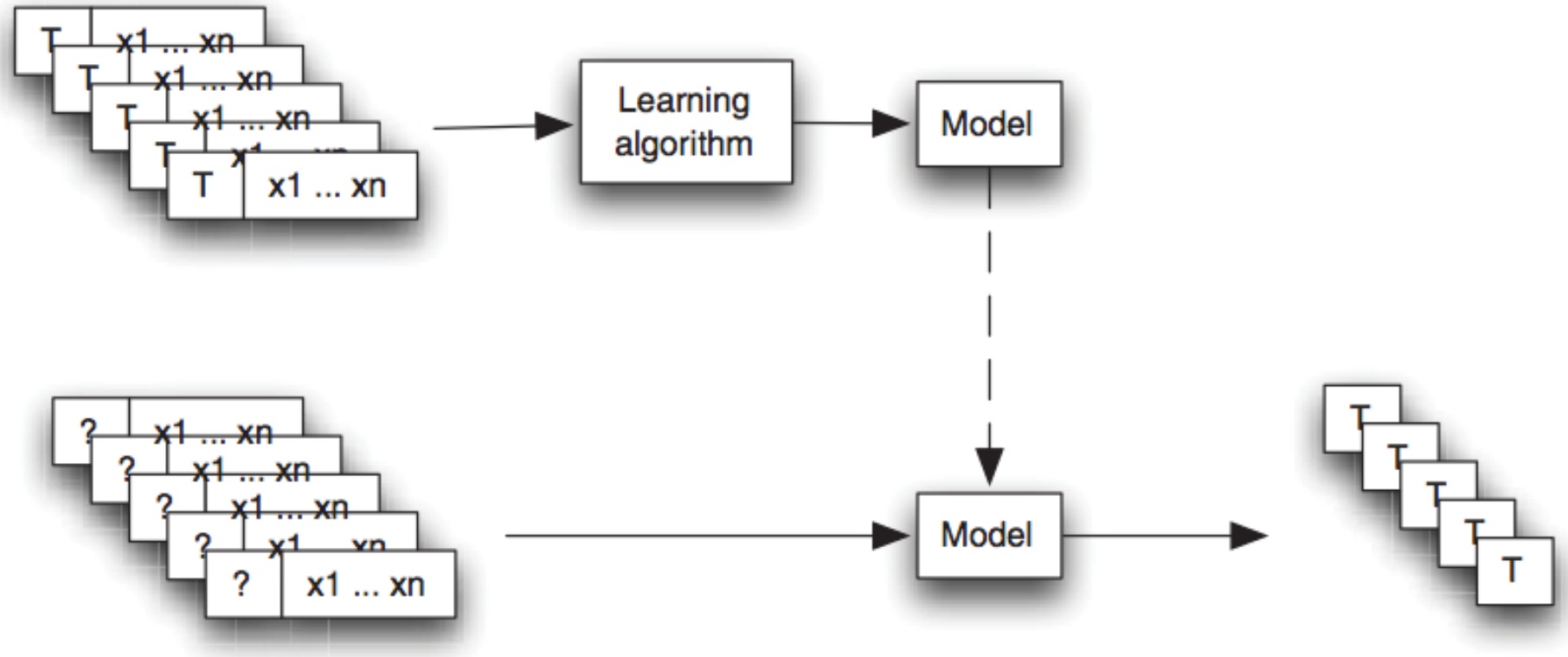
How does a classification system work?



Key terminology for classification

Key Idea	Description
Model	A computer program that makes decisions; in classification, the output of the training algorithm is a model.
Training data	A subset of training examples labeled with the value of the target variable and used as input to the learning algorithm to produce the model.
Test data	A withheld portion of the training data with the value of the target variable hidden so that it can be used to evaluate the model.
Training	The learning process that uses training data to produce a model. That model can then compute estimates of the target variable given the predictor variables as inputs.
Training example	An entity with features that will be used as input for learning algorithm.
Feature	A known characteristic of a training or a new example; a feature is equivalent to a characteristic.
Variable	In this context, the value of a feature or a function of several features. This usage is somewhat different from the use of *variable* in a computer program.
Record	A container where an example is stored; such a record is composed of fields.
Field	Part of a record that contains the value of a feature (a variable).
Predictor variable	A feature selected for use as input to a classification model. Not all features need be used. Some features may be algorithmic combinations of other features.
Target variable	A feature that the classification model is attempting to estimate: the target variable is categorical, and its determination is the aim of the classification system.

Input and Output of a classification model



Four types of values for predictor variables

Type of value	Description
Continuous	This is a floating-point value. This type of value might be a price, a weight, a time, or anything else that has a numerical magnitude and where this magnitude is the key property of the value.
Categorical	A categorical value can have one of a set of prespecified values. Typically the set of categorical values is relatively small and may be as small as two, although the set can be quite large. Boolean values are generally treated as categorical values. Another example might be a vendor ID.
Word-like	A word-like value is like a categorical value, but it has an open-ended set of possible values.
Text-like	A text-like value is a sequence of word-like values, all of the same kind. Text is the classic example of a text-like value, but a list of email addresses or URLs is also text-like.

Sample data that illustrates all four value types

Name	Type	Value
from-address	Word-like	`George <george@fumble-tech.com>`
in-address-book?	Categorical (`TRUE`, `FALSE`)	`TRUE`
non-spam-words	Text-like	`Ted, Mahout, User, lunch`
spam-words	Text-like	`available`
unknown-words	Continuous	`0`
message-length	Continuous	`31`

Supervised vs. Unsupervised Learning

Columbia University

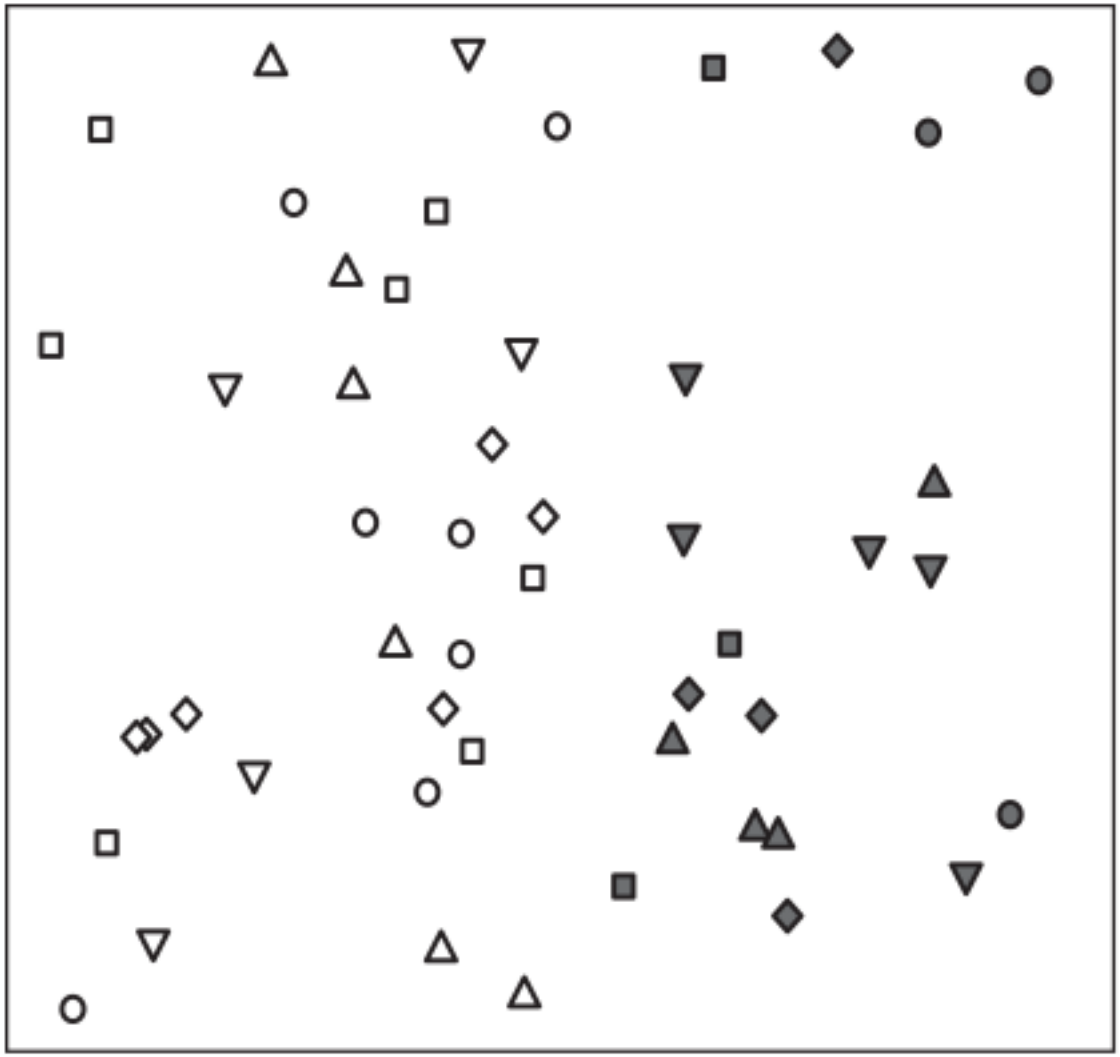
Classification algorithms are related to, but still quite different from, clustering algorithms such as the k-means algorithm described in previous chapters. Classification algorithms are a form of supervised learning, as opposed to unsupervised learning, which happens with clustering algorithms. A supervised learning algorithm is one that's given examples that contain the desired value of a target variable. Unsupervised algorithms aren't given the desired answer, but instead must find something plausible on their own.

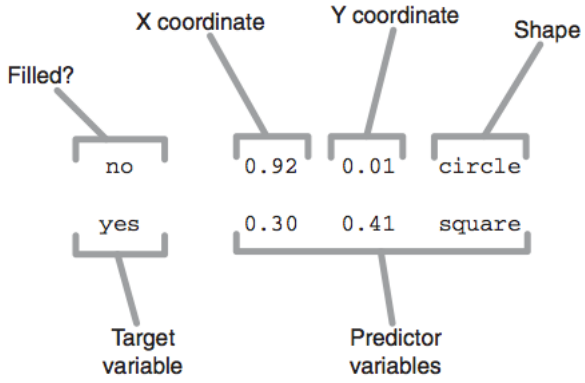
Supervised and unsupervised learning algorithms can often be usefully combined. A clustering algorithm can be used to create features that can then be used by a learning algorithm, or the output of several classifiers can be used as features by a clustering algorithm. Moreover, clustering systems often build a model that can be used to categorize new data. This clustering system model works much like the model produced by a classification system. The difference lies in what data was used to produce the model. For classification, the training data includes the target variables; for clustering, the training data doesn't include target variables.

Work flow in a typical classification project

Stage	Step
1. Training the model	Define target variable. Collect historical data. Define predictor variables. Select a learning algorithm. Use the learning algorithm to train the model.
2. Evaluating the model	Run test data. Adjust the input (use different predictor variables, different algorithms, or both).
3. Using the model in production	Input new examples to estimate unknown target values. Retrain the model as needed.

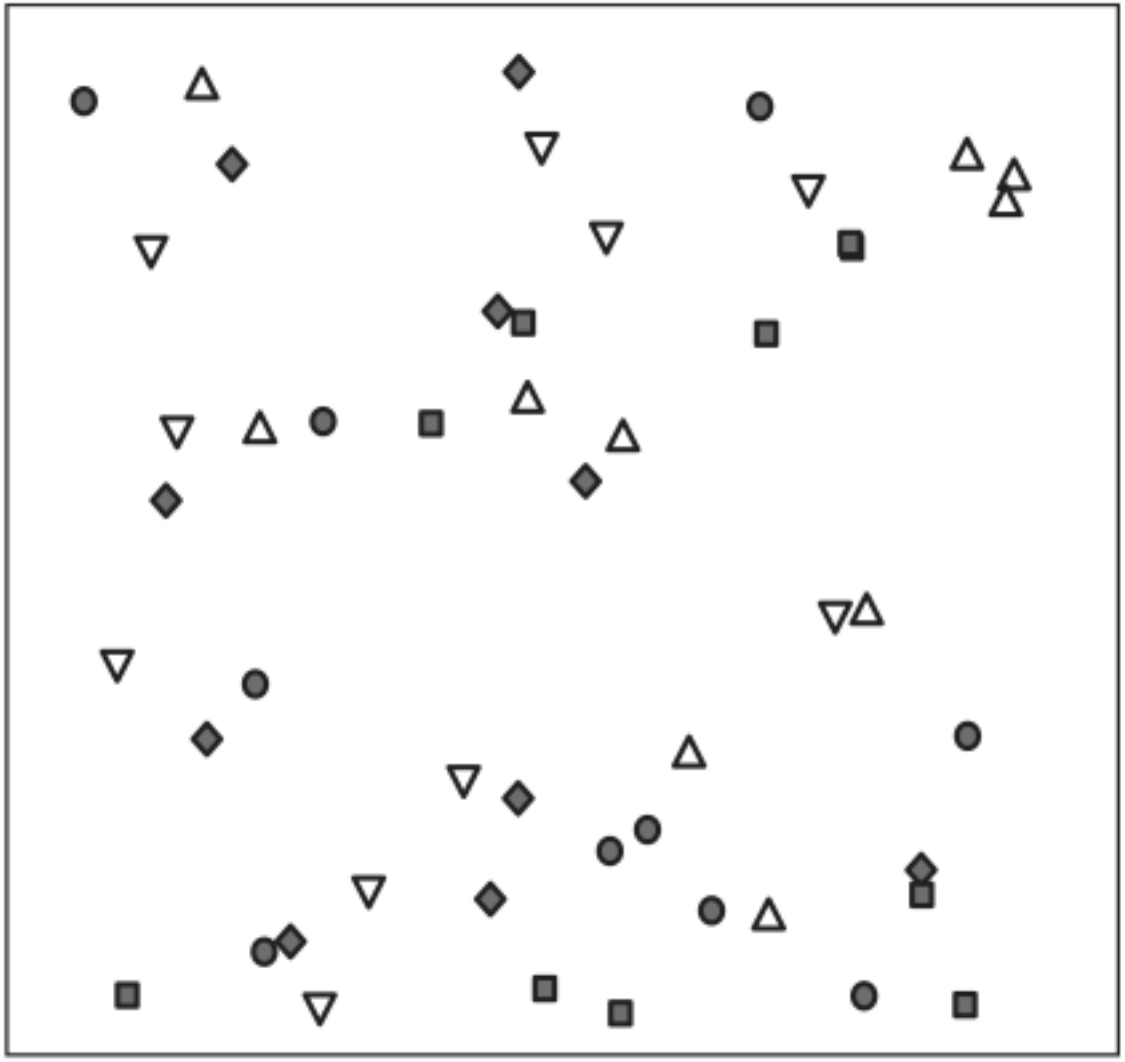
Classification Example 1 — Color-Fill





Position looks promising, especially the x-axis ==> predictor variable.
Shape seems to be irrelevant. Target variable is “color-fill” label.

Classification Example 2 — Color-Fill (another feature)



Fundamental classification algorithms

Example of fundamental classification algorithms:

- Naive Bayesian
- Complementary Naive Bayesian
- Stochastic Gradient Descent (SDG)
- Random Forest
- Support Vector Machines

Choose algorithm

Size of data set	Mahout algorithm	Execution model	Characteristics
Small to medium (less than tens of millions of training examples)	Stochastic gradient descent (SGD) family: `OnlineLogisticRegression`, `CrossFoldLearner`, `AdaptiveLogisticRegression`	Sequential, online, incremental	Uses all types of predictor variables; sleek and efficient over the appropriate data range (up to millions of training examples)
	Support vector machine (SVM)	Sequential	Experimental still; sleek and efficient over the appropriate data range
Medium to large (millions to hundreds of millions of training examples)	Naive Bayes	Parallel	Strongly prefers text-like data; medium to high overhead for training; effective and useful for data sets too large for SGD or SVM
	Complementary naive Bayes	Parallel	Somewhat more expensive to train than naive Bayes; effective and useful for data sets too large for SGD, but has similar limitations to naive Bayes
Small to medium (less than tens of millions of training examples)	Random forests	Parallel	Uses all types of predictor variables; high overhead for training; not widely used (yet); costly but offers complex and interesting classifications, handles nonlinear and conditional relationships in data better than other techniques

Stochastic Gradient Descent (SGD)

Both statistical estimation and machine learning consider the problem of minimizing an objective function that has the form of a sum:

$$Q(w) = \sum_{i=1}^{n} Q_i(w),$$

where the parameter w is to be estimated and where typically each summand function $Q_i()$ is associated with the i-th observation in the data set (used for training).

- Choose an initial vector of parameters w and learning rate α.
- Randomly shuffle examples in the training set.
- Repeat until an approximate minimum is obtained:
  - For $i = 1, 2, ..., n$, do:
    - $w := w - \alpha \nabla Q_i(w)$.

Let's suppose we want to fit a straight line $y = w_1 + w_2 x$ to a training set of two-dimensional points $(x_1, y_1), \ldots, (x_n, y_n)$ using least squares. The objective function to be minimized is:

$$Q(w) = \sum_{i=1}^{n} Q_i(w) = \sum_{i=1}^{n} (w_1 + w_2 x_i - y_i)^2 .$$

The last line in the above pseudocode for this specific problem will become:

$$\begin{bmatrix} w_1 \\ w_2 \end{bmatrix} := \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} - \alpha \begin{bmatrix} \sum_{i=1}^{n} 2(w_1 + w_2 x_i - y_i) \\ \sum_{i=1}^{n} 2x_i(w_1 + w_2 x_i - y_i) \end{bmatrix}.$$

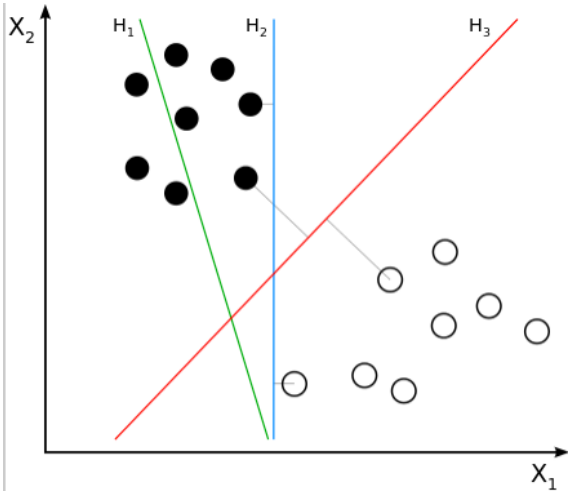
Characteristic of SGD

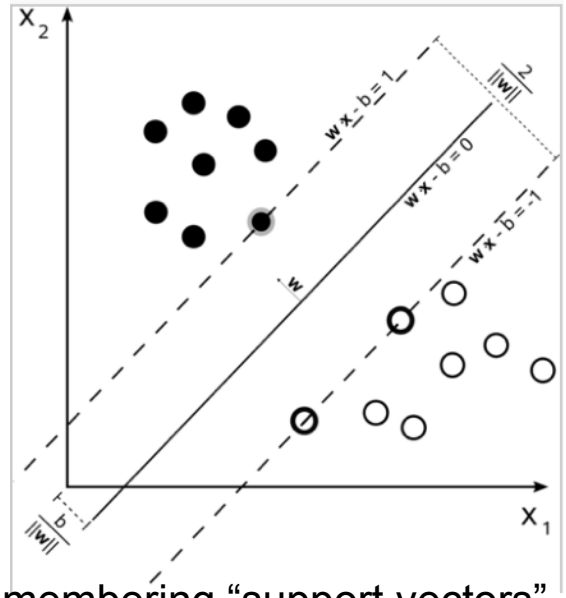
THE SGD ALGORITHM

Stochastic gradient descent (SGD) is a widely used learning algorithm in which each training example is used to tweak the model slightly to give a more correct answer for that one example. This incremental approach is repeated over many training examples. With some special tricks to decide how much to nudge the model, the model accurately classifies new data after seeing only a modest number of examples. Although SGD algorithms are difficult to parallelize effectively, they're often so fast that for a wide variety of applications, parallel execution isn't necessary.

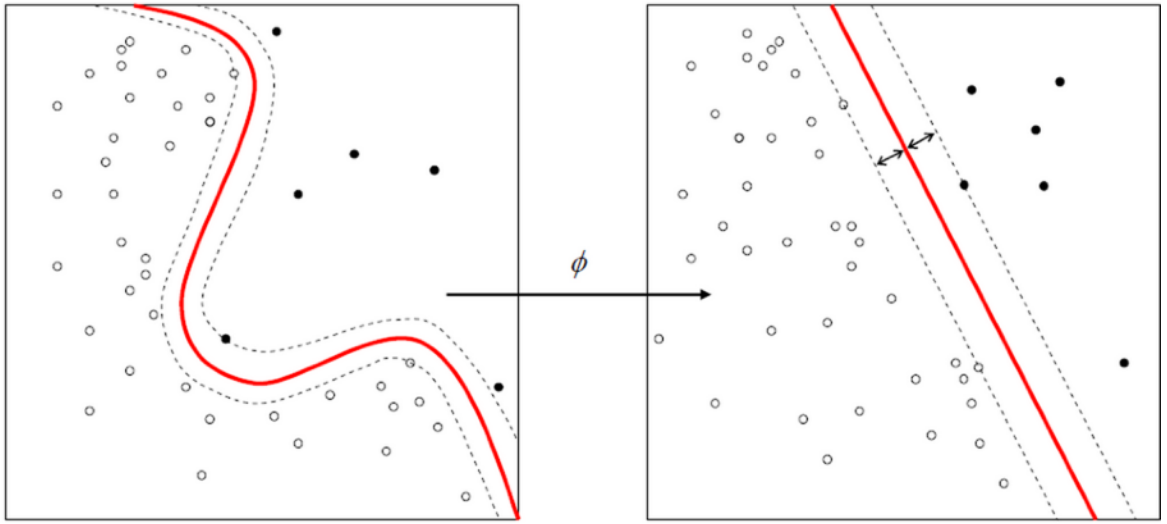
Importantly, because these algorithms do the same simple operation for each training example, they require a constant amount of memory. For this reason, each training example requires roughly the same amount of work. These properties make SGD-based algorithms scalable in the sense that twice as much data takes only twice as long to process.

Support Vector Machine (SVM)





maximize boundary distances; remembering “support vectors”



nonlinear kernels

Example SVM code in Spark

```
from pyspark.ml.classification import LinearSVC

# Load training data
training = spark.read.format("libsvm").load("data/mllib/sample_libsvm_data.txt")

lsvc = LinearSVC(maxIter=10, regParam=0.1)

# Fit the model
lsvcModel = lsvc.fit(training)

# Print the coefficients and intercept for linear SVC
print("Coefficients: " + str(lsvcModel.coefficients))
print("Intercept: " + str(lsvcModel.intercept))
```

Naive Bayes

Training set:

sex	height (feet)	weight (lbs)	foot size(inches)
male	6	180	12
male	5.92 (5'11")	190	11
male	5.58 (5'7")	170	12
male	5.92 (5'11")	165	10
female	5	100	6
female	5.5 (5'6")	150	8
female	5.42 (5'5")	130	7
female	5.75 (5'9")	150	9

Classifier using Gaussian distribution assumptions:

sex	mean (height)	variance (height)	mean (weight)	variance (weight)	mean (foot size)	variance (foot size)
male	5.855	3.5033e-02	176.25	1.2292e+02	11.25	9.1667e-01
female	5.4175	9.7225e-02	132.5	5.5833e+02	7.5	1.6667e+00

Test Set:

sex	height (feet)	weight (lbs)	foot size(inches)
sample	6	130	8

$$posterior(male) = \frac{P(male)\,p(height|male)\,p(weight|male)\,p(footsize|male)}{evidence}$$

$$evidence = P(male)\,p(height|male)\,p(weight|male)\,p(footsize|male) + P(female)\,p(height|female)\,p(weight|female)\,p(footsize|female)$$

$$P(male) = 0.5$$

$$p(\text{height}|\text{male}) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(\frac{-(6-\mu)^2}{2\sigma^2}\right) \approx 1.5789,$$

$$p(\text{weight}|\text{male}) = 5.9881 \cdot 10^{-6}$$
$$p(\text{foot size}|\text{male}) = 1.3112 \cdot 10^{-3}$$
$$\text{posterior numerator (male)} = \text{their product} = 6.1984 \cdot 10^{-9}$$

$$P(\text{female}) = 0.5$$
$$p(\text{height}|\text{female}) = 2.2346 \cdot 10^{-1}$$
$$p(\text{weight}|\text{female}) = 1.6789 \cdot 10^{-2}$$
$$p(\text{foot size}|\text{female}) = 2.8669 \cdot 10^{-1}$$
$$\text{posterior numerator (female)} = \text{their product} = 5.3778 \cdot 10^{-4}$$

==> female

Random Forest

Random forests are an ensemble learning method for classification (and regression) that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes output by individual trees.

The training algorithm for random forests applies the general technique of bootstrap aggregating, or bagging, to tree learners. Given a training set $X = x_1, \ldots, x_n$ with responses $Y = y_1$ through y_n, bagging repeatedly selects a bootstrap sample of the training set and fits trees to these samples:

For b = 1 through B:

1. Sample, with replacement, n training examples from X, Y; call these X_b, Y_b.
2. Train a decision or regression tree f_b on X_b, Y_b.

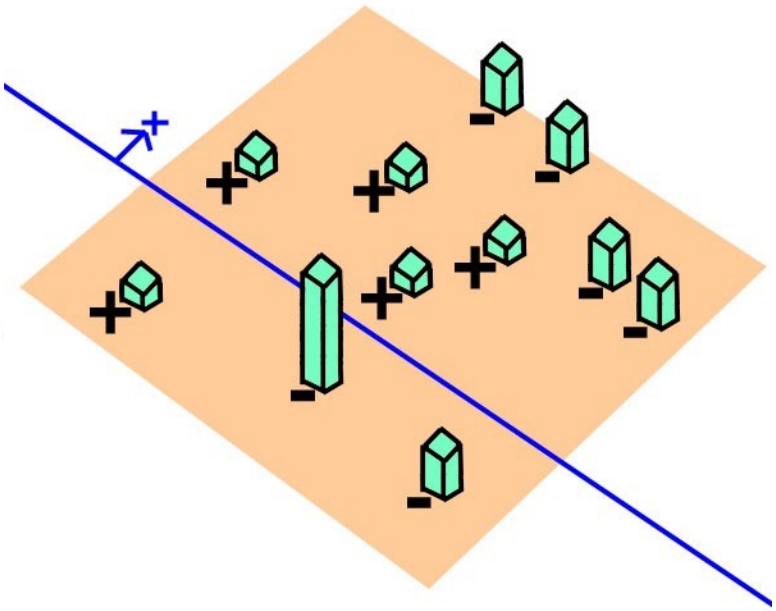
After training, predictions for unseen samples x' can be made by averaging the predictions from all the individual regression trees on x':

$$\hat{f} = \frac{1}{B}\sum_{b=1}^{B} \hat{f}_b(x')$$

or by taking the majority vote in the case of decision trees.

Random forest uses a modified tree learning algorithm that selects, at each candidate split in the learning process, a random subset of the features.

Adaboost Example



- Adaboost [Freund and Schapire 1996]
  - Constructing a "strong" learner as a linear combination of weak learners

- Start with a uniform distribution ("weights") over training examples
  (The weights tell the weak learning algorithm which examples are important)

- Obtain a weak classifier from the weak learning algorithm, $h_{j_t}: X \rightarrow \{-1, 1\}$

- Increase the weights on the training examples that were misclassified

- (Repeat)

The final classifier is a linear combination of the weak classifiers obtained at all iterations

$$f_{\text{final}}(\mathbf{x}) = sign\left(\sum_{s=1}^{S} \alpha_s h_s(x)\right)$$

Example — User Modeling using Time-Sensitive Adaboost

- Obtain simple classifier on each feature, e.g., setting threshold on parameters, or binary inference on input parameters.

- The system classify whether a new document is interested by a person via Adaptive Boosting (Adaboost):
  - The final classifier is a linear weighted combination of single-feature classifiers.
  - Given the single-feature simple classifiers, assigning weights on the training samples based on whether a sample is correctly or mistakenly classified. <== Boosting.
  - Classifiers are considered sequentially. The selected weights in previous considered classifiers will affect the weights to be selected in the remaining classifiers. Adaptive.
  - According to the summed errors of each simple classifier, assign a weight to it. The final classifier is then the weighted linear combination of these simple classifiers.

People select apples according to their shapes, sizes, other people's interest, etc.

Each attribute is a simple classifier used in Adaboost.

- Our new Time-Sensitive Adaboost algorithm:
  - In the AdaBoost algorithm, all samples are regarded equally important at the beginning of the learning process
  - We propose a time-adaptive AdaBoost algorithm that assigns larger weights to the latest training samples

Time-Sensitive Adaboost [Song, Lin, et al. 2005]

- In AdaBoost, the goal is to minimize the energy function:

$$\sum_{i=1}^{N} \exp\left(-c_i \sum_{s=1}^{S} \alpha_s h_s(x_i)\right)$$

  - All samples are regarded equally important at the beginning of the learning process

- Propose a time-adaptive AdaBoost algorithm that assigns larger weights to the latest documents to indicate their importance

$$\sum_{i=1}^{N} \exp\left(-c_i \sum_{s=1}^{S} \alpha_s \exp\left(-\tau \cdot (t - t_i)\right) h_s(x_i, t)\right)$$

- Weak learners
  - linear classifiers corresponding to the content, community and dynamic patterns

Algorithm: ***Time-Sensitive Adaboost***

Given: $(x_1, c_1, t_1), \mathrm{K}, (x_N, c_N, t_N)$ where $x_i \in X$, $c_i \subseteq \{-1, 1\}$, N is the size of samples in the training set; current time t, and τ

For $s = 1, \ldots, S$

Initialize $D_1(i) = 1 / \left(N \cdot \exp\left(\tau \cdot (t - t_i)\right)\right)$.

Set the weight α_s of the current weak hypothesis h_s according to its weighted error rate ε_s

$$\alpha_s = \frac{1}{2} \ln\left(\frac{1 - \varepsilon_s}{\varepsilon_s}\right)$$

where $\varepsilon_s = \sum_{i=1}^{N} D_s(i) h_s(x_i) c_i$.

Update $D_{s+1}(i) = \dfrac{D_s(i) \exp\left(-\alpha_s c_i \exp\left(-\tau \cdot (t - t_i)\right) h_s(x_i)\right)}{Z_s}$

where Z_s is a normalization term.

End

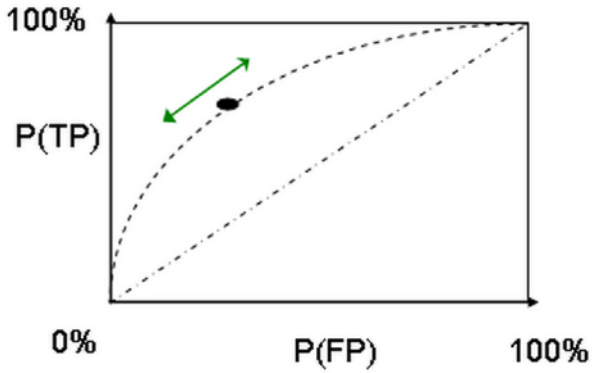
Find weak hypothesis by: $h_s = \arg\min_{h_j \in \mathrm{H}} \varepsilon_j$.

Output: the final hypothesis: $H(x) = \mathrm{sign}(F(x))$

where $F(x) = \sum_{s=1}^{S} \alpha_s h_s(x)$.

Evaluate the model

```
$  bin/mahout runlogistic --input donut.csv --model ./model \
        --auc --confusion
AUC = 0.57
confusion: [[27.0, 13.0], [0.0, 0.0]]
...
```



AUC (0 ~ 1):
- 1 — perfect
- 0 — perfectly wrong
- 0.5 — random

True positive	**False positive** (Type I error)
False negative (Type II error)	**True negative**

confusion matrix

Option	What it does
`--quiet`	Produces less status and progress output.
`--auc`	Prints AUC score for model versus input data after reading data.
`--scores`	Prints target variable value and scores for each input example.
`--confusion`	Prints confusion matrix for a particular threshold (see `--threshold`).
`--input <input>`	Reads data records from specified file or resource.
`--model <model>`	Reads model from specified file.

Homework #1 (Due 10/4/2019, 5pm)

Columbia University

See TA's instruction.

Questions?