CSCE 5300 Introduction to Big Data and Data Science

Machine Learning (Basic Concepts)

Spark Mllib

Agenda

- Introduction to Machine Learning
- Why MLlib
- K-means
- Logistic Regression
- Classification

Introduction to Machine Learning

Definition

- Machine learning is a study of computer algorithms that improve automatically through experience.

Terminology

- **Observation**
  - The basic unit of data.

- **Features**
  -Features is that can be used to describe each observation in a quantitative manner.

- **Feature vector**
  -is an n-dimensional vector of numerical features that represents some object.

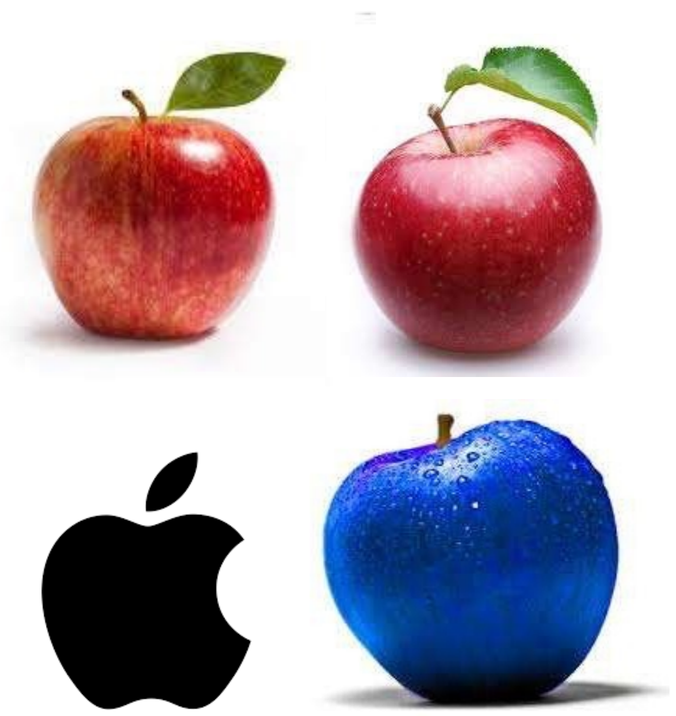
- **Training/ Testing/ Evaluation set**
  - Set of data to discover potential predictive relationships

Learning(Training)

- **Features:**
  1.Color: Red/ Green
  2. Type: Fruit
  3.Shape: nearly Circle
  etc…

Learning(Training)

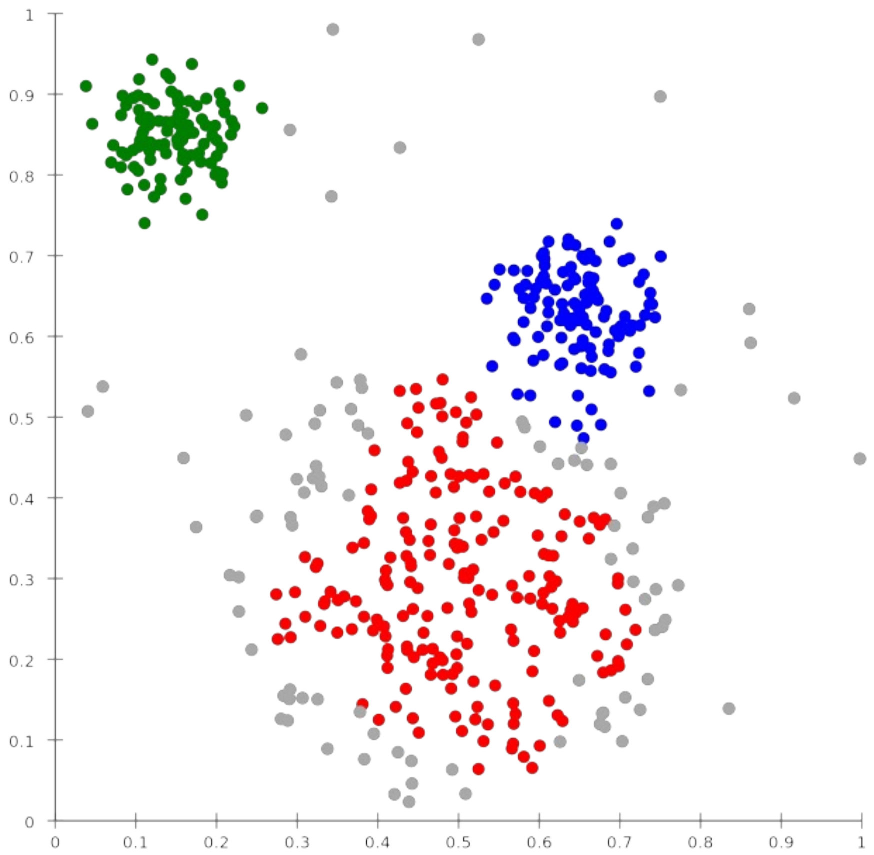
ID	Color	Type	Shape	is Apple (Label)
1	Red	Fruit	Cirle	Y
2	Red	Fruit	Cirle	Y
3	Black	Logo	nearly Circle	N
4	Blue	N/A	Cirle	N



Categories of Machine Learning

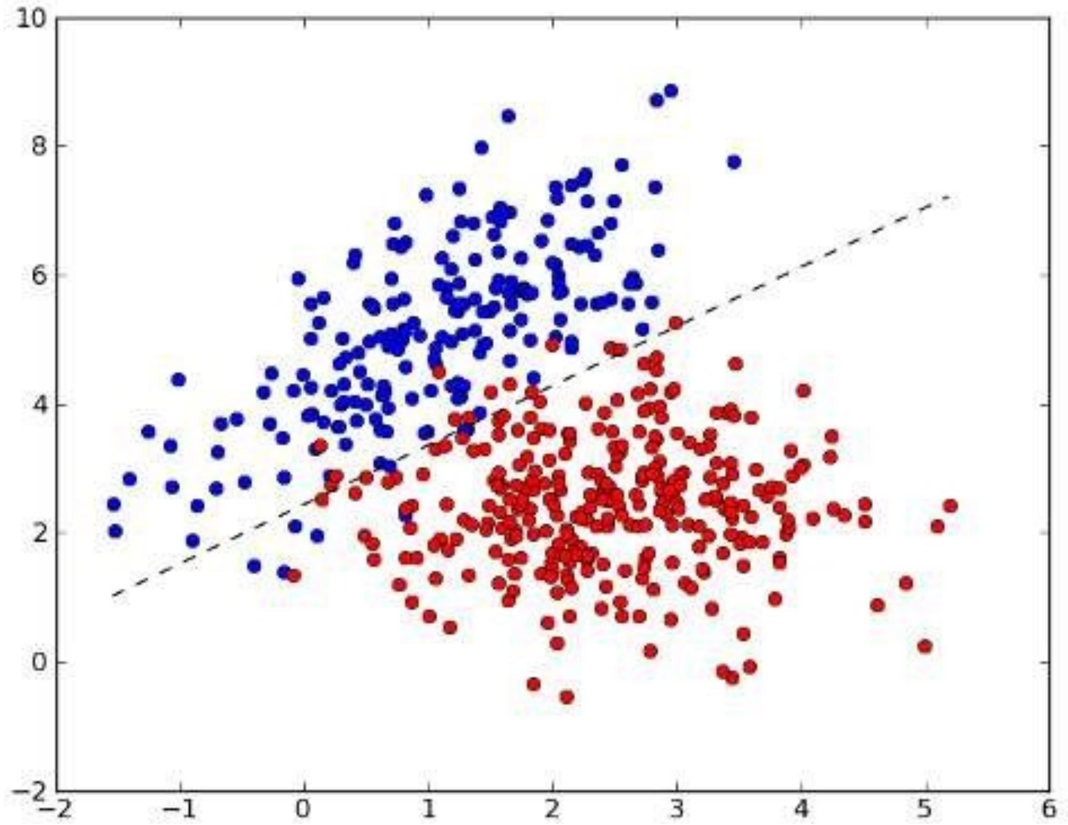
- **Classification:** predict class from observations.
- **Clustering:** group observations into meaningful groups.
- **Regression:** predict value from observations.

Cluster the observations with no Labels



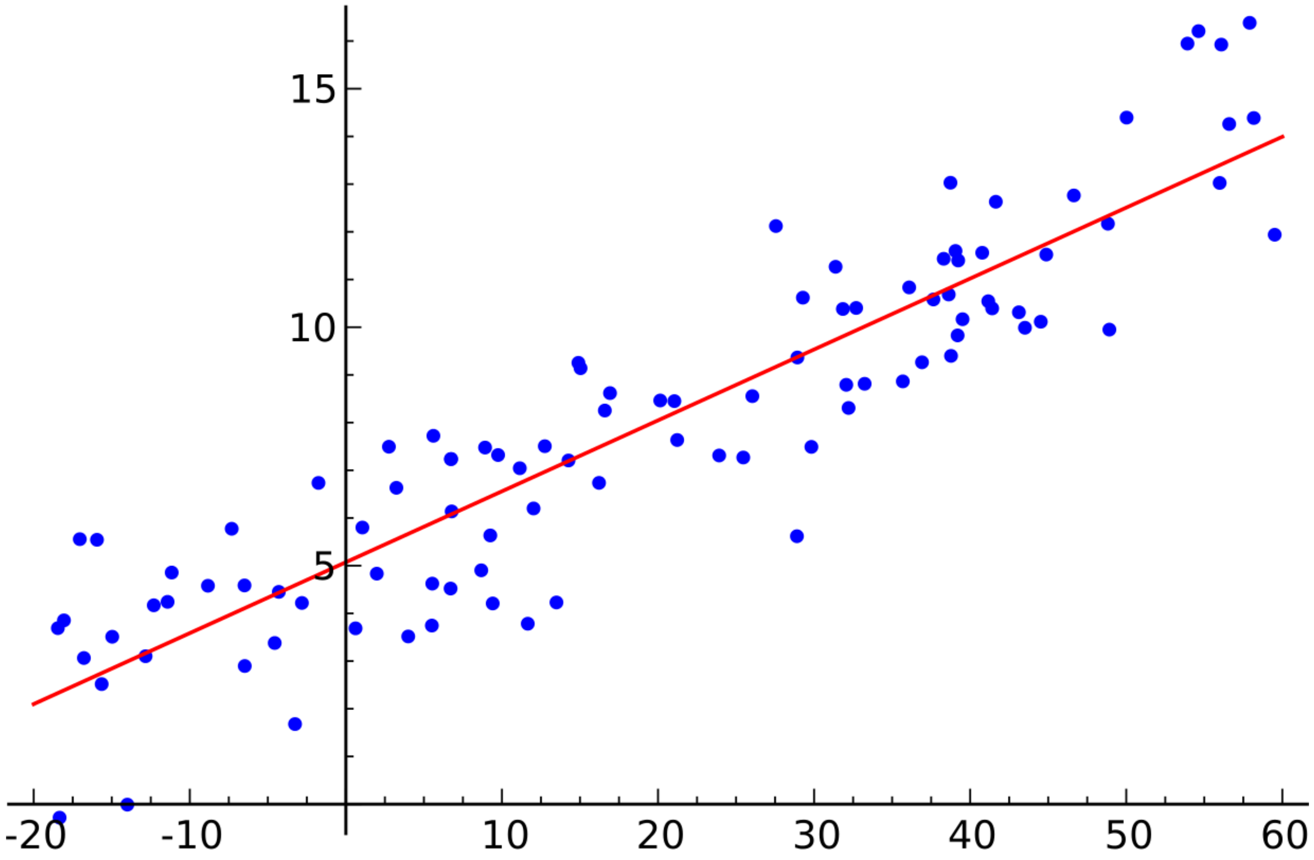
https://en.wikipedia.org/wiki/Cluster_analysis

Cut the observations



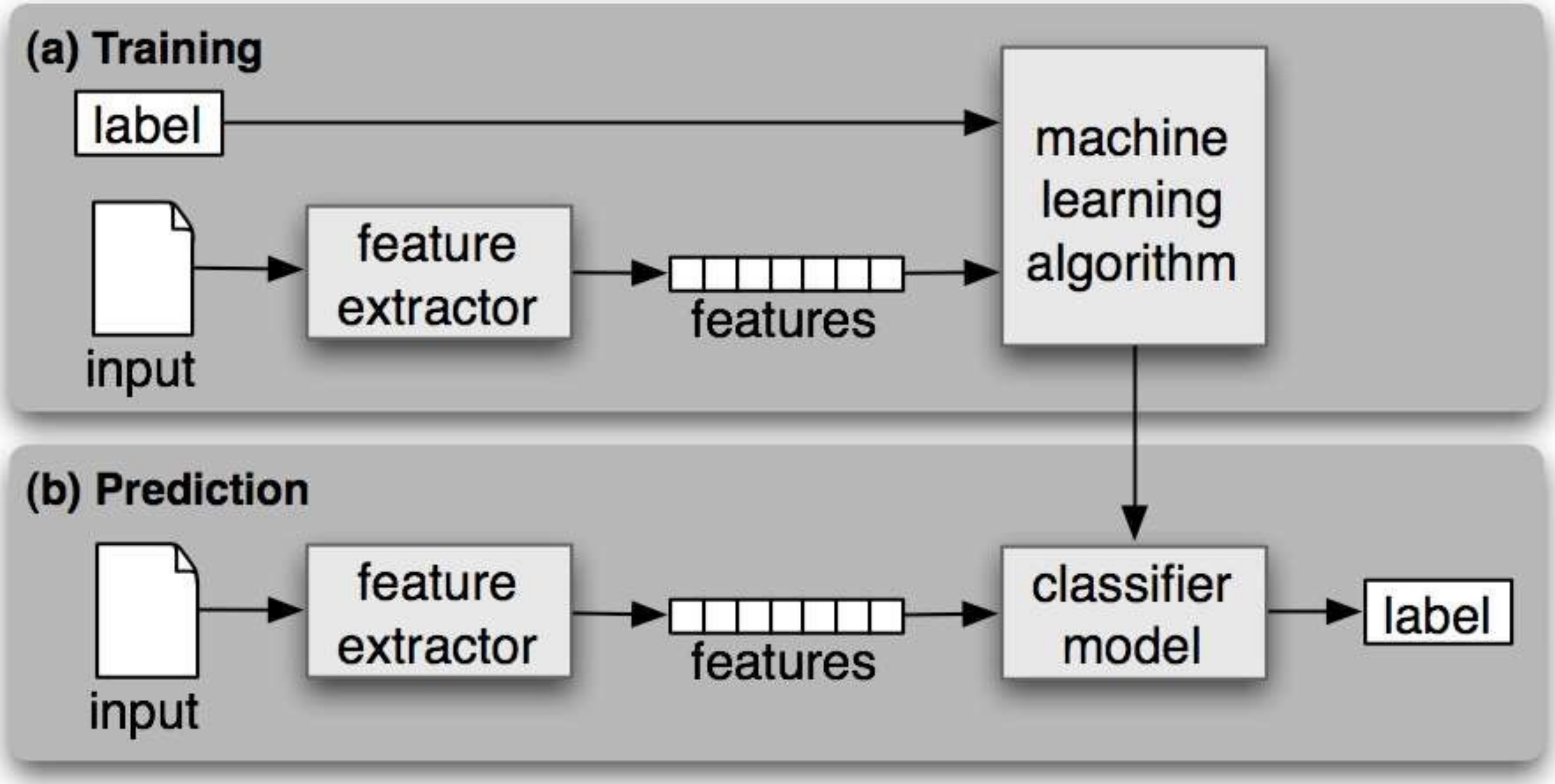
http://stats.stackexchange.com/questions/159957/how-to-do-one-vs-one-classification-for-logistic-regression

Find a model to describe the observations



https://commons.wikimedia.org/wiki/File:Linear_regression.svg

Machine Learning Pipelines



http://www.nltk.org/book/ch06.html

What is MLlib

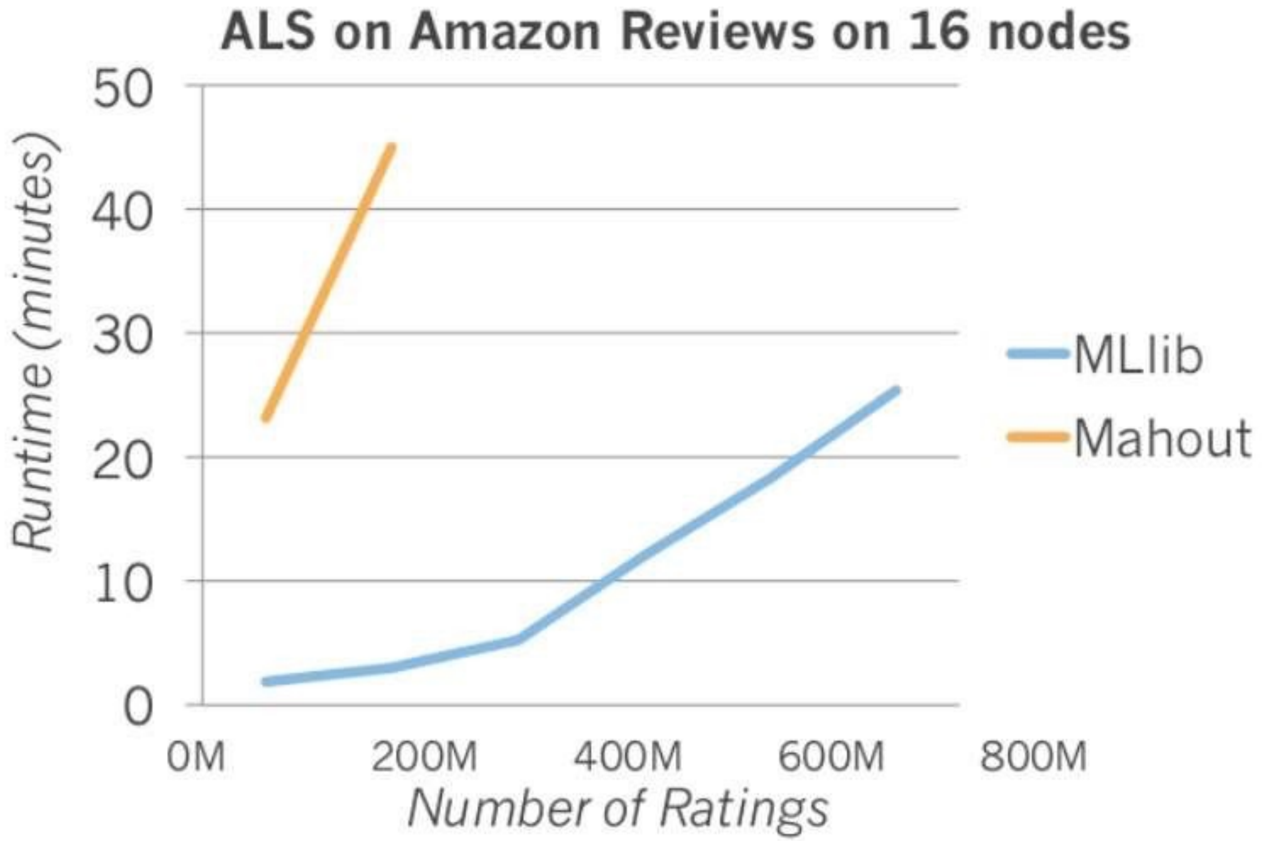
What is MLlib

- **MLlib** is an Apache Spark component focusing on **machine learning**:
  - MLlib is Spark’s core ML library
  - Developed by MLbase team in AMPLab
  - 80+ contributions from various organization
  - Support Scala, Python, and Java APIs

Algorithms in MLlib

- **Statistics**: Description, correlation
- **Clustering:** k-means
- **Collaborative filtering:** ALS
- **Classification:** SVMs, naive Bayes, decision tree.
- **Regression:** linear regression, logistic regression
- **Dimensionality:** (Singular Value Decomposition) SVD, Principle Component Analysis (PCA)

Performance



On a dataset with 660M users, 2.4M items, and 3.5B ratings
MLlib runs in 40 minutes with 50 nodes

Data Type

- Dense vector
- Sparse vector
- Labeled point

Dense & Sparse

- **Raw Data:**

ID	A	B	C	D	E	F
1	1	0	0	0	0	3
2	0	1	0	1	0	2
3	1	1	1	0	1	1

$$\text{dense} : \underbrace{1.}\,\underbrace{0.}\,\underbrace{0.}\,\underbrace{0.}\,\underbrace{0.}\,\underbrace{0.}\,\underbrace{3.}$$

$$\text{sparse} : \begin{cases} \text{size} : 7 \\ \text{indices} : \underbrace{0}\,\underbrace{6} \\ \text{values} : \underbrace{1.}\,\underbrace{3.} \end{cases}$$

Dense vs Sparse

- Training Set:
  - number of example: 12 million
  - number of features: 500
  - sparsity: 10%

	Dense	Sparse
Storage	47GB	7GB
Time	240s	58s

- Not only save storage, but also received a 4x speed up

Labeled Point

- Dummy variable (1,0)
- Categorical variable (0, 1, 2, …)

```
from pyspark.mllib.linalg import SparseVector
from pyspark.mllib.regression import LabeledPoint

# Create a labeled point with a positive label and a dense feature vector.  pos =
LabeledPoint(1.0, [1.0, 0.0, 3.0])

# Create a labeled point with a negative label and a sparse feature  vector.
neg = LabeledPoint(0.0, SparseVector(3, [0, 2], [1.0, 3.0]))
```

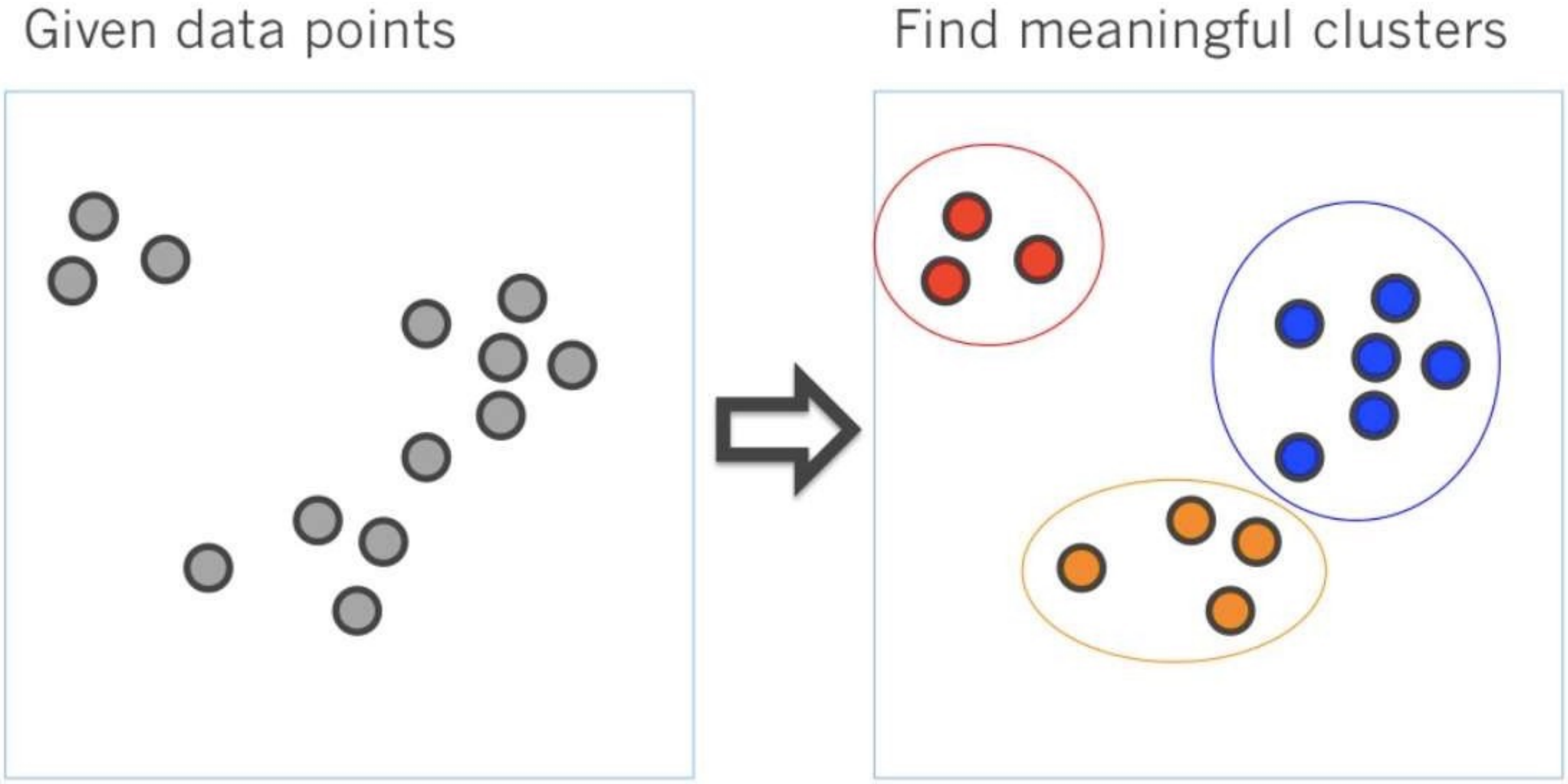
Clustering Model
K-Means

K-means

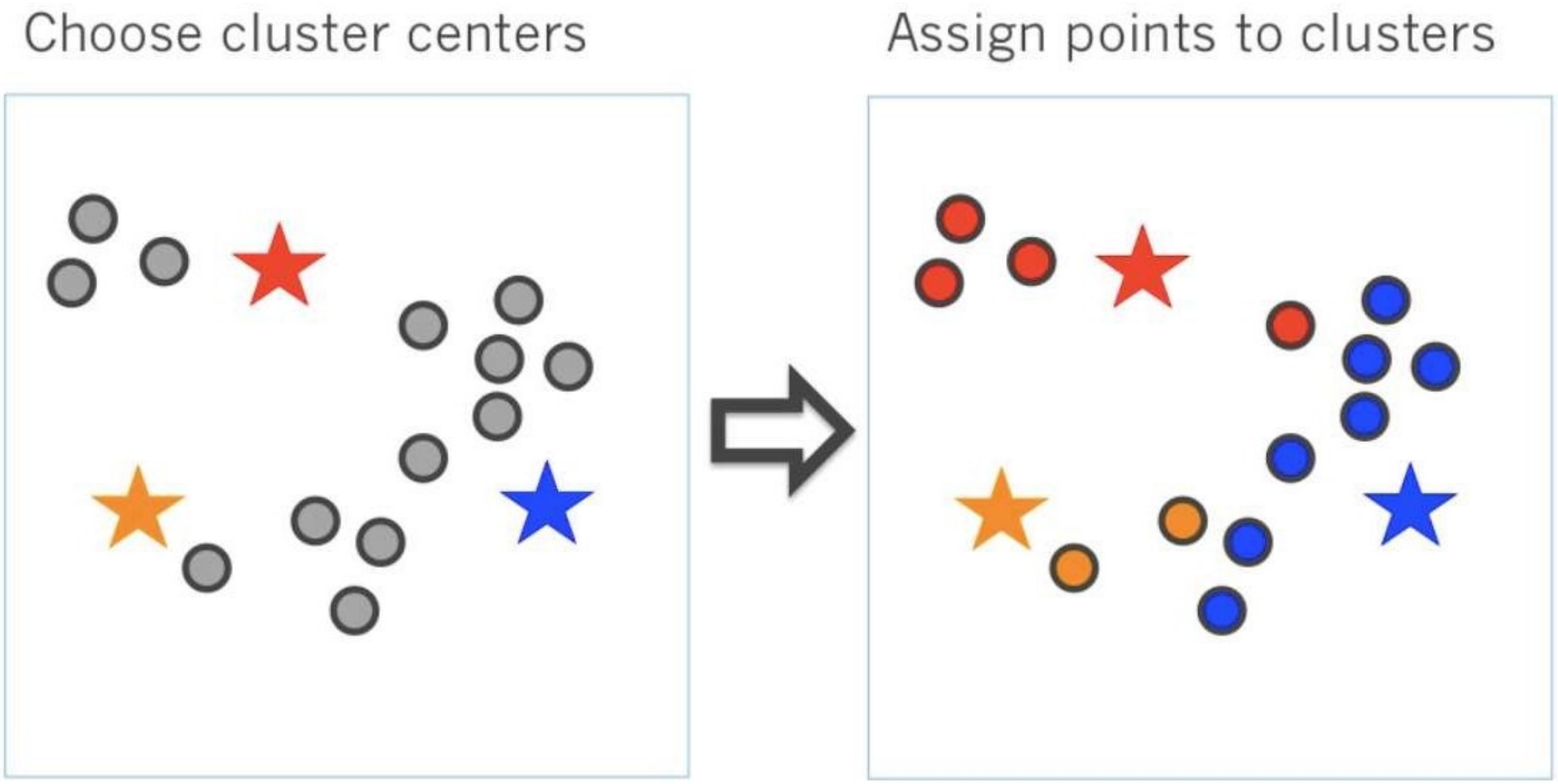
- K-means clustering aims to partition **n** observations into **k** clusters in which each observation belongs to the cluster with the nearest mean, serving as a prototype of the cluster.

$$\underset{\mathbf{S}}{\arg\min} \sum_{i=1}^{k} \sum_{\mathbf{x} \in S_i} \left\| \mathbf{x} - \boldsymbol{\mu}_i \right\|^2$$

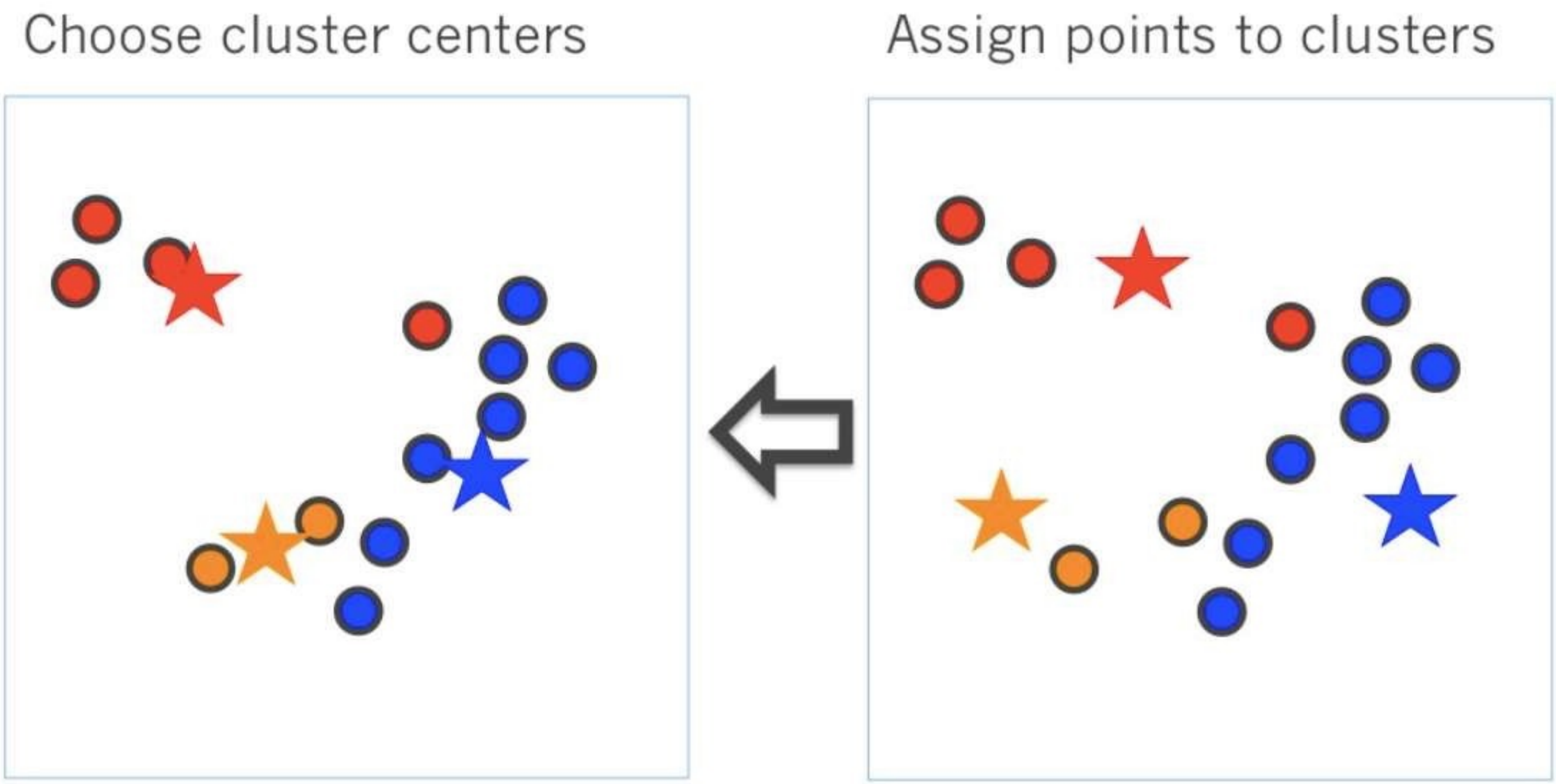
Clustering with K-Means



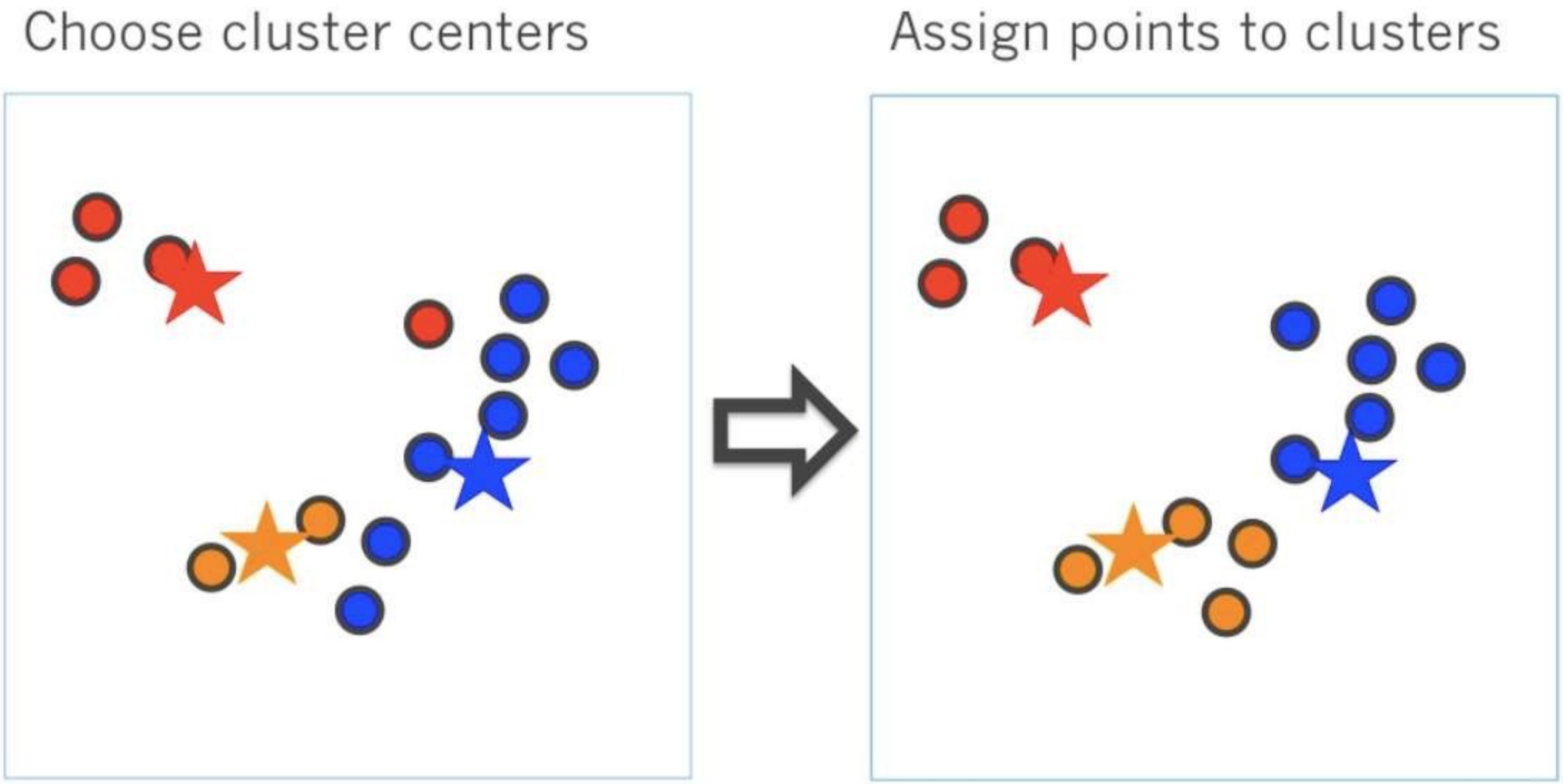
Clustering with K-Means



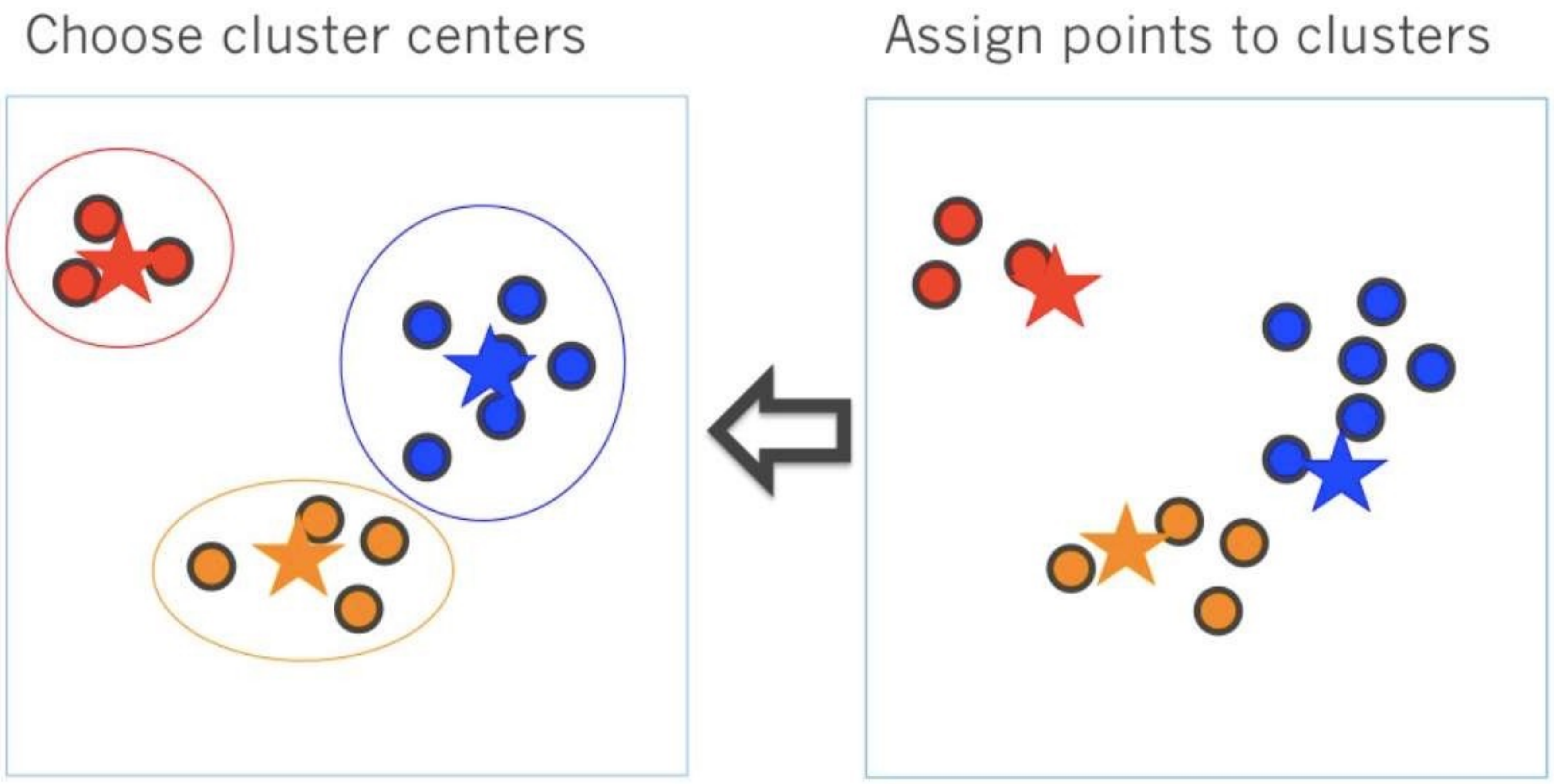
Clustering with K-Means



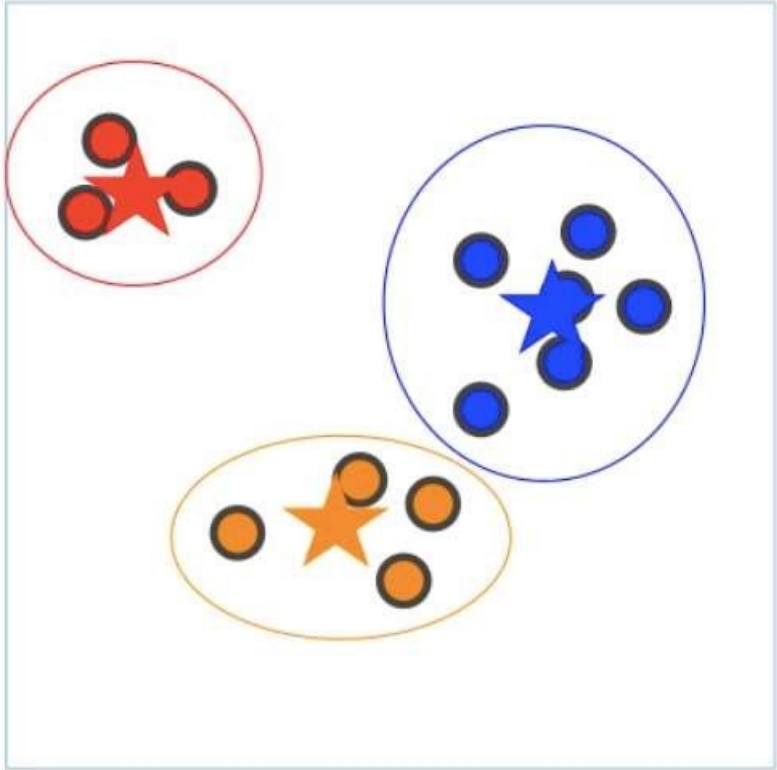
Clustering with K-Means



Clustering with K-Means



Clustering with K-Means



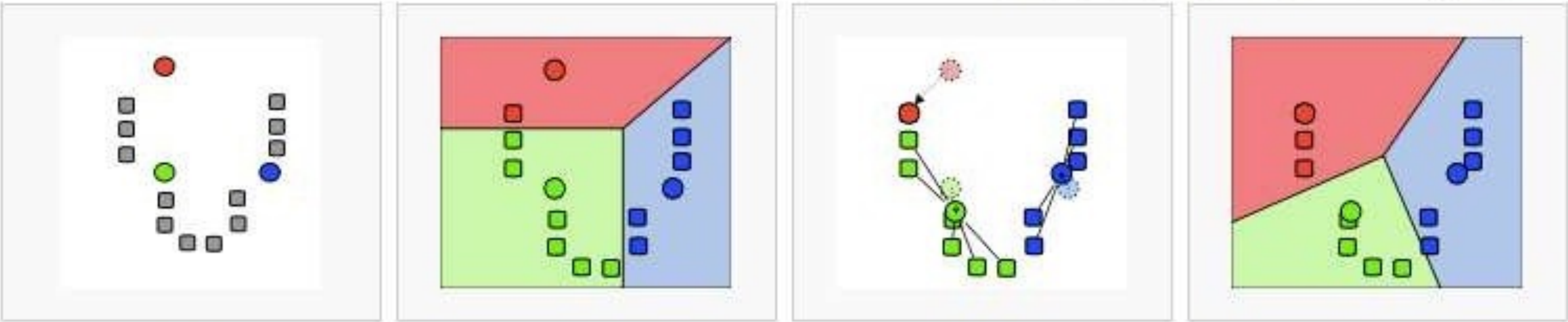
Data distributed by instance (point/row)

Smart initialization

Limited communication
(# clusters << # instances)

Summary

Demonstration of the standard algorithm



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

2. *k* clusters are created by associating every observation with the nearest mean. The partitions here represent the Voronoi diagram generated by the means.

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

4. Steps 2 and 3 are repeated until convergence has been reached.

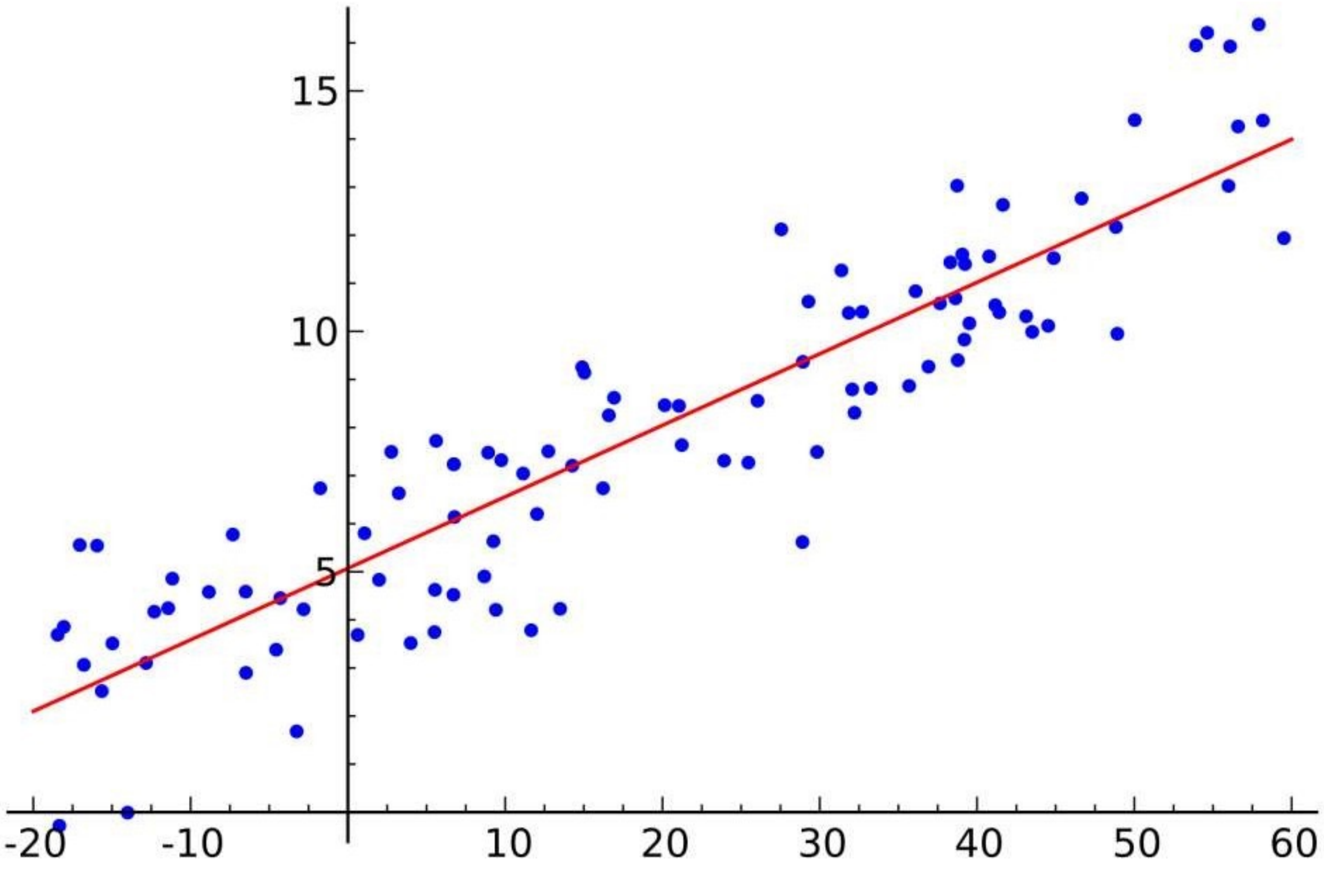
https://en.wikipedia.org/wiki/K-means_clustering

K-Means: Python

```
from pyspark.mllib.clustering import KMeans, KMeansModel  from
numpy import array
from math import sqrt

# Load and parse the data
data = sc.textFile("data/mllib/kmeans_data.txt")
parsedData = data.map(lambda line: array([float(x) for x in line.split(' ')]))

# Build the model (cluster the data)
clusters = KMeans.train(parsedData, 2, maxIterations=10,
           runs=10, initializationMode="random")

# Evaluate clustering by computing Within Set Sum of Squared Errors  def
error(point):
   center = clusters.centers[clusters.predict(point)]  return
   sqrt(sum([x**2 for x in (point - center)]))

WSSSE = parsedData.map(lambda point: error(point)).reduce(lambda x, y: x + y)
print("Within Set Sum of Squared Error = " + str(WSSSE))
```

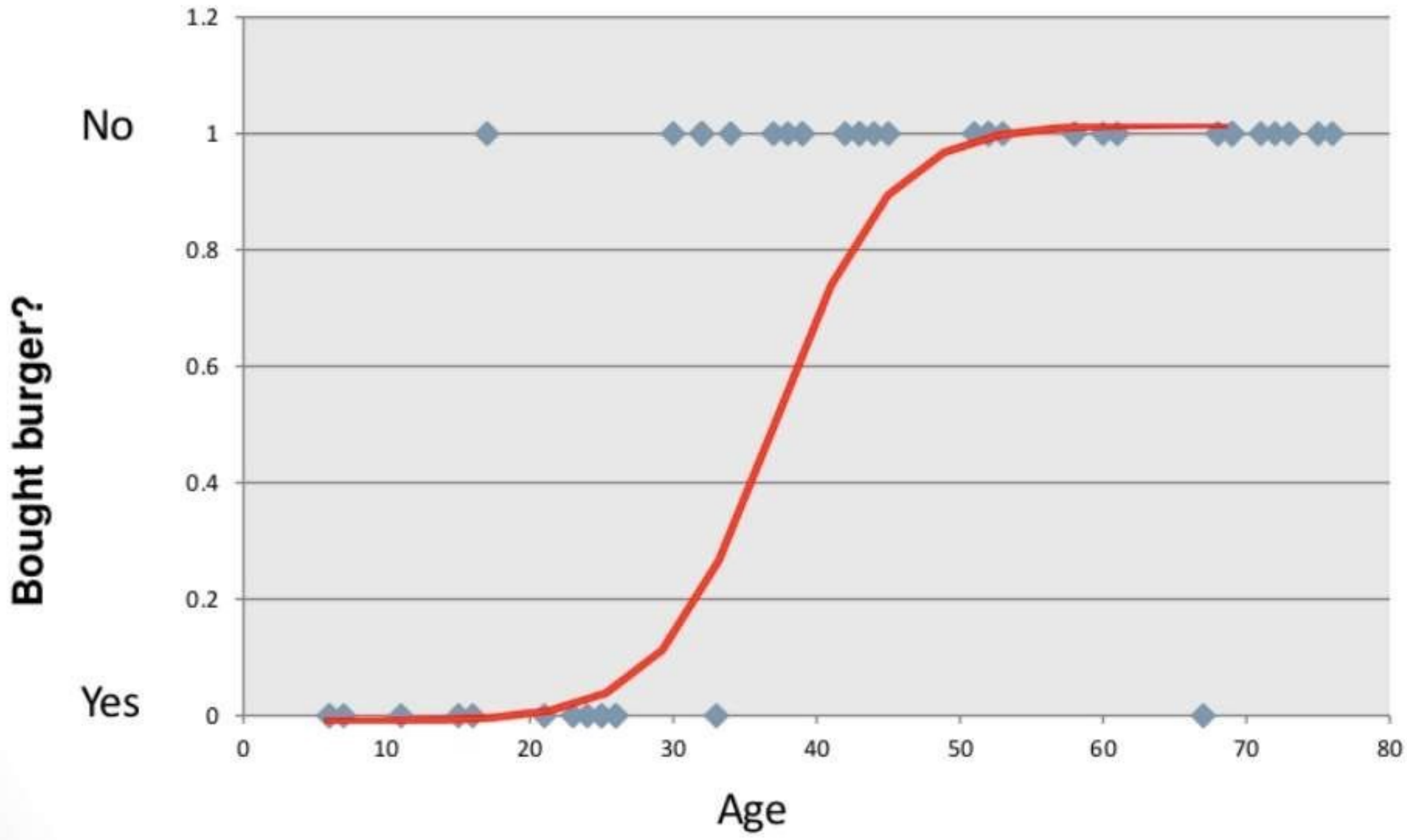
Classification Model
Logistic Regression

linear regression



https://commons.wikimedia.org/wiki/File:Linear_regression.svg

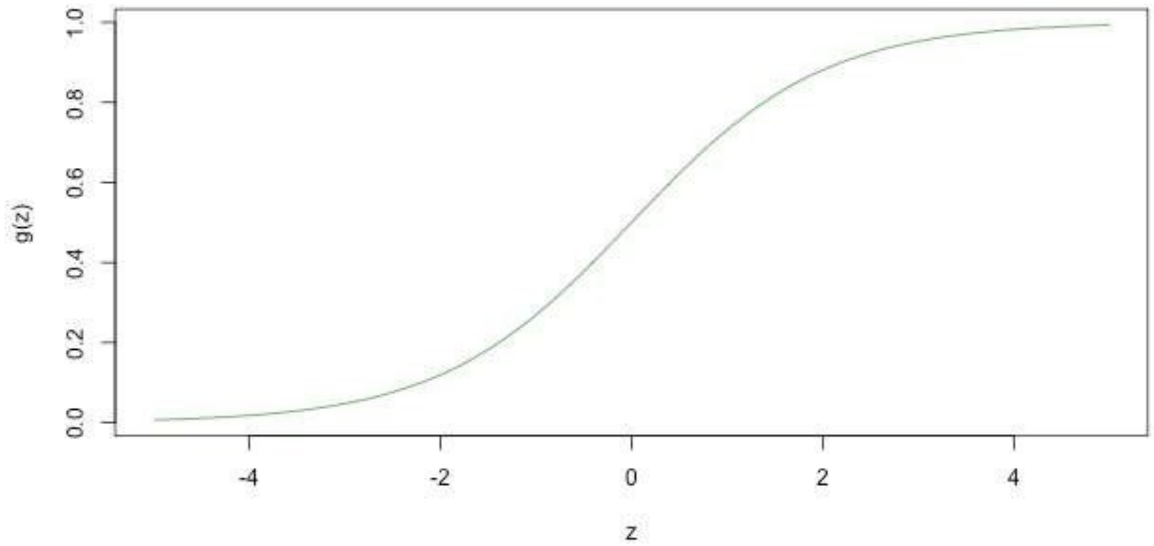
When outcome is only 1/0



http://www.slideshare.net/21_venkat/logistic-regression-17406472

Hypotheses function

- hypotheses: $h_\theta(x) = g(\theta^T x) = \frac{1}{1 + e^{-\theta^T x}}$
- sigmoid function: $g(z) = \frac{1}{1 + e^{-z}}$



Cost Function

- Maximum Likelihood estimation

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\left[\underbrace{y^{(i)}\log(h_\theta(x^{(i)}))}_{\text{if } y = 1} + \underbrace{(1-y^{(i)})\log(1-h_\theta(x^{(i)}))}_{\text{if } y = 0}\right] + \underbrace{\frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2}_{\text{regularized}}$$

Sample Code

```
from pyspark.mllib.classification import LogisticRegressionWithSGD
from pyspark.mllib.regression import LabeledPoint
from numpy import array

# Load and parse the data
def parsePoint(line):
    values = [float(x) for x in line.split(' ')]
    return LabeledPoint(values[0], values[1:])

data = sc.textFile("data/mllib/sample_svm_data.txt")
parsedData = data.map(parsePoint)

# Build the model
model = LogisticRegressionWithSGD.train(parsedData)

# Evaluating the model on training data
labelsAndPreds = parsedData.map(lambda p: (p.label, model.predict(p.features)))
trainErr = labelsAndPreds.filter(lambda (v, p): v != p).count() /
float(parsedData.count())
print("Training Error = " + str(trainErr))
```

Classification Algorithms

Decision Tree

Decision Tree

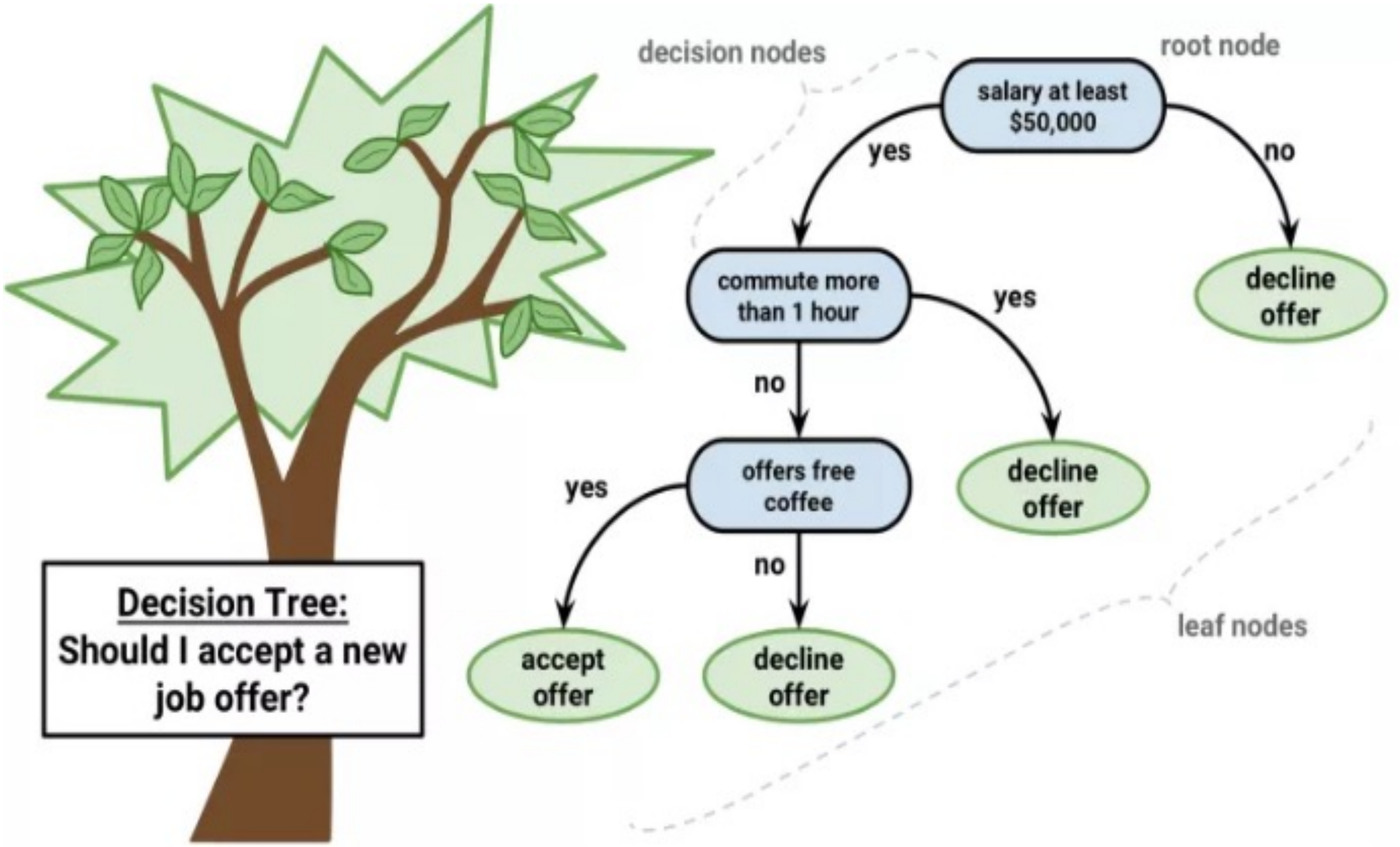
- Decision Tree algorithm belongs to the family of supervised learning algorithms
- Unlike other supervised learning algorithms, decision tree algorithm can be used for solving regression and classification problems too

Pseudocode

Place the best attribute of the dataset at the **root** of the tree.

Split the training set into **subsets**. Subsets should be made in such a way that each subset contains data with the same value for an attribute.

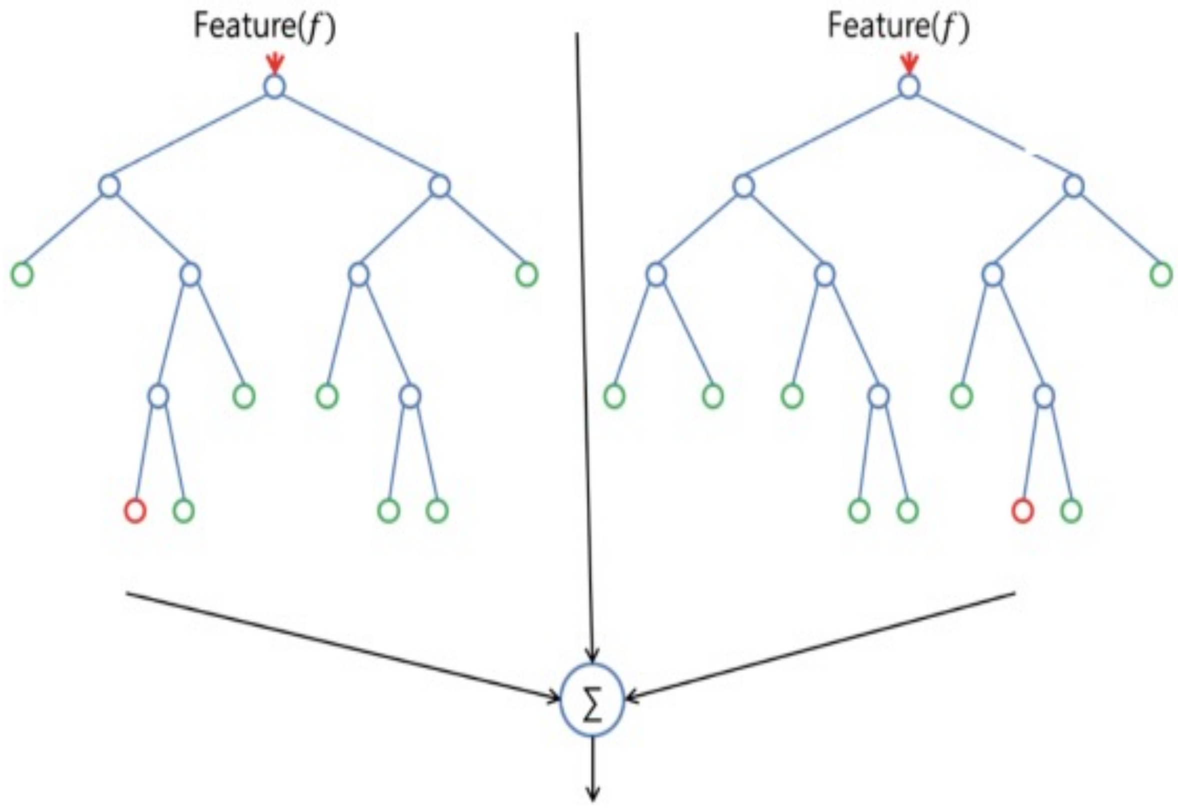
Repeat step 1 and step 2 on each subset until you find **leaf nodes** in all the branches of the tree.


decision nodes
root node
salary at least $50,000
yes
no
commute more than 1 hour
yes
decline offer
no
offers free coffee
decline offer
yes
no
accept offer
decline offer
leaf nodes
Decision Tree: Should I accept a new job offer?

Random Forest

Random Forest

Random forests or **random decision forests** are an ensemble learning method for classification, regression and other tasks, that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees


Feature(f)
Feature(f)
Σ

Naïve Bayes

Naive Method

- It is a classification technique based on Bayes' theorem.
- Naïve Bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature

Naive Method

- Why Naïve?
- Even if these features depend on each other or upon existance of the other features, all of these properties independently contribute to the probability of an object and that is why it is known as “Naïve”

Naive Method

Why use Naïve Bayes?

- Easy to build
- Useful for large datasets

Naive Method

- Naïve Bayes Formula

Prior Probability

Likelihood of the evidence 'E' if the Hypothesis 'H' is true

$$P(H|E) = \frac{P(H) * P(E|H)}{P(E)}$$

Posterior Probability of 'H' given the evidence

Priori probability that the evidence itself is true

References

- **Machine Learning Library (MLlib) Guide** http://spark.apache.org/docs/1.4.1/mllib-guide.html
- **MLlib: Spark's Machine Learning Library** http://www.slideshare.net/jeykottalam/mllib
- **Recent Developments in Spark MLlib and Beyond** http://www.slideshare.net/Hadoop_Summit
- **Introduction to Machine Learning** http://www.slideshare.net/rahuldausa/introduction-to-machine-_learning

SVM
https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47