Spark ML Tutorial

## What is Machine Learning?

*“Field of study that gives computers the ability to learn without being explicitly programmed.” – Arthur Samuel (1959)*

*"A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T, as measured by P, improves with experience E.” - Tom Mitchell (1998)*

# Machine Learning Algorithms: Classification

1. Supervised Learning
2. Unsupervised Learning

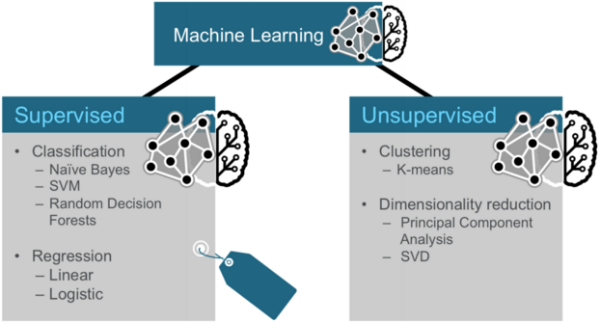
# Supervised Learning:

**Supervised learning** is the **machine learning** task of inferring a function from labeled training data. The training data consist of a set of training examples. In **supervised learning**, each example is a pair consisting of an input object (typically a vector) and a desired output value (also called the supervisory signal).

In Supervised Learning, right answers are given. Based on the previous known data, we have to predict a continuous valued output (Regression) or classify the input into groups (Classification).

# Unsupervised Learning

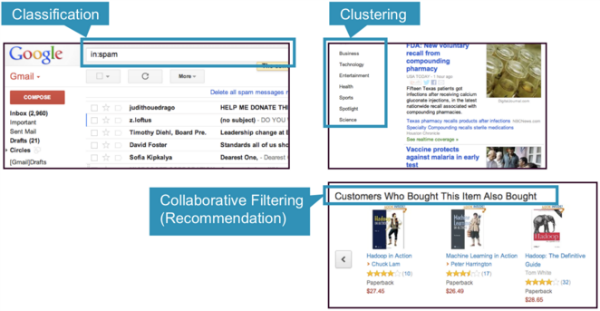
**Unsupervised machine learning** is the [machine learning](https://en.wikipedia.org/wiki/Machine_learning) task of inferring a function to describe hidden structure from "unlabeled" data (a classification or categorization is not included in the observations). Since the examples given to the learner are unlabeled, there is no evaluation of the accuracy of the structure that is output by the relevant algorithm—which is one way of distinguishing unsupervised learning from [supervised learning](https://en.wikipedia.org/wiki/Supervised_learning) and [reinforcement learning](https://en.wikipedia.org/wiki/Reinforcement_learning).



Credit: https://mapr.com/blog/apache-spark-machine-learning-tutorial/

# Three Categories of Techniques for Machine Learning

Three common categories of machine learning techniques are Classification, Clustering and Collaborative Filtering.



Credit: https://mapr.com/blog/apache-spark-machine-learning-tutorial/

* **Classification:** Gmail uses a machine learning technique called classification to designate if an email is spam or not, based on the data of an email: the sender, recipients, subject, and message body. Classification takes a set of data with known labels and learns how to label new records based on that information.
* **Clustering:** Google News uses a technique called clustering to group news articles into different categories, based on title and content. Clustering algorithms discover groupings that occur in collections of data.
* **Collaborative Filtering:** Amazon uses a machine learning technique called collaborative filtering (commonly referred to as recommendation), to determine which products users will like based on their history and similarity to other users.

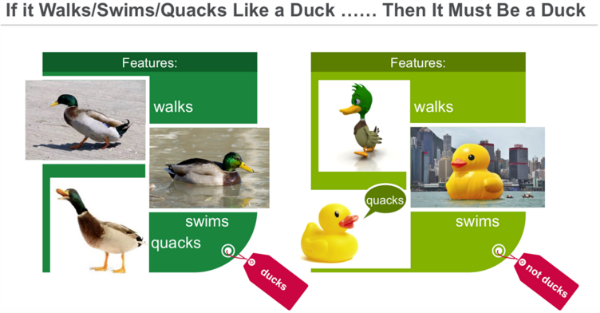
# Classification

Classification is a family of supervised machine learning algorithms that designate input as belonging to one of several pre-defined classes. Some common use cases for classification include:

* credit card fraud detection
* email spam detection

Classification data is labeled, for example, as spam/non-spam or fraud/non-fraud. Machine learning assigns a label or class to new data.

You classify something based on pre-determined features. Features are the “if questions” that you ask. The label is the answer to those questions. In this example, if it walks, swims, and quacks like a duck, then the label is "duck."



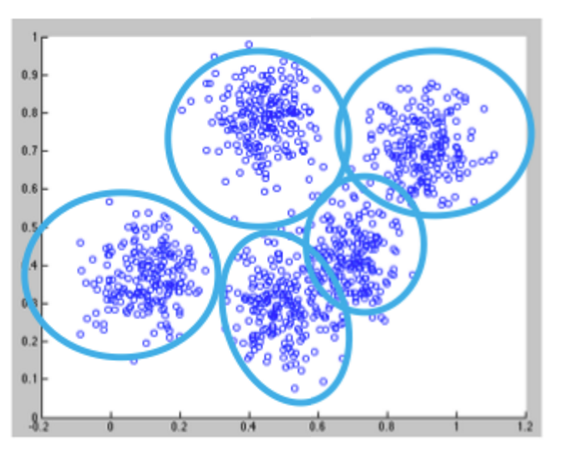
# Clustering

In clustering, an algorithm groups objects into categories by analyzing similarities between input examples. Clustering uses include:

* **Search results** grouping
* **Grouping of customers**
* **Anomaly** detection
* Text categorization



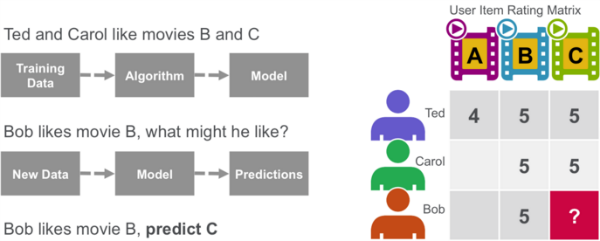
Clustering uses unsupervised algorithms, which do not have the outputs in advance.



Clustering using the K-means algorithm begins by initializing all the coordinates to centroids. With every pass of the algorithm, each point is assigned to its nearest centroid based on some distance metric, usually Euclidean distance. The centroids are then updated to be the “centers” of all the points assigned to it in that pass. This repeats until there is a minimum change in the centers.

# Collaborative Filtering

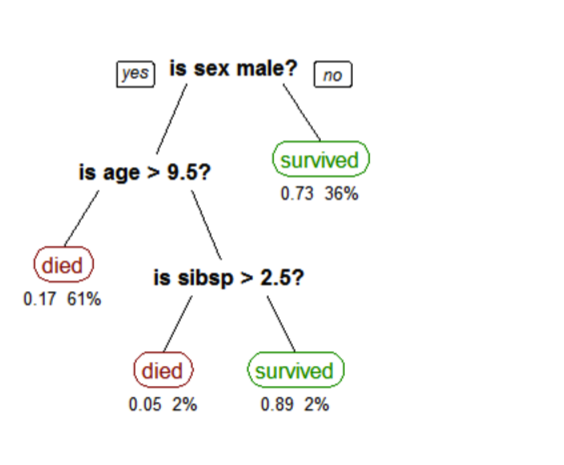
Collaborative filtering algorithms recommend items (this is the filtering part) based on preference information from many users (this is the collaborative part). The collaborative filtering approach is based on similarity; people who liked similar items in the past will like similar items in the future. The goal of a collaborative filtering algorithm is to take preferences data from users, and to create a model that can be used for recommendations or predictions. Ted likes movies A, B, and C. Carol likes movies B and C. We take this data and run it through an algorithm to build a model. Then when we have new data such as Bob likes movie B, we use the model to predict that C is a possible recommendation for Bob.



## Decision Trees

Decision trees create a model that predicts the class or label based on several input features. Decision trees work by evaluating an expression containing a feature at every node and selecting a branch to the next node based on the answer. A decision tree for predicting survival on the Titanic is shown below. The feature questions are the nodes, and the answers “yes” or “no” are the branches in the tree to the child nodes.

* Q1: is sex male?
  + yes
  + Q2: is age > 9.5?
    - No
    - Is sibsp >2.5?
      * No
      * died



A tree showing survival of passengers on the [Titanic](https://en.wikipedia.org/wiki/Titanic) ("sibsp" is the number of spouses or siblings aboard). The figures under the leaves show the probability of survival and the percentage of observations in the leaf.

Training Set

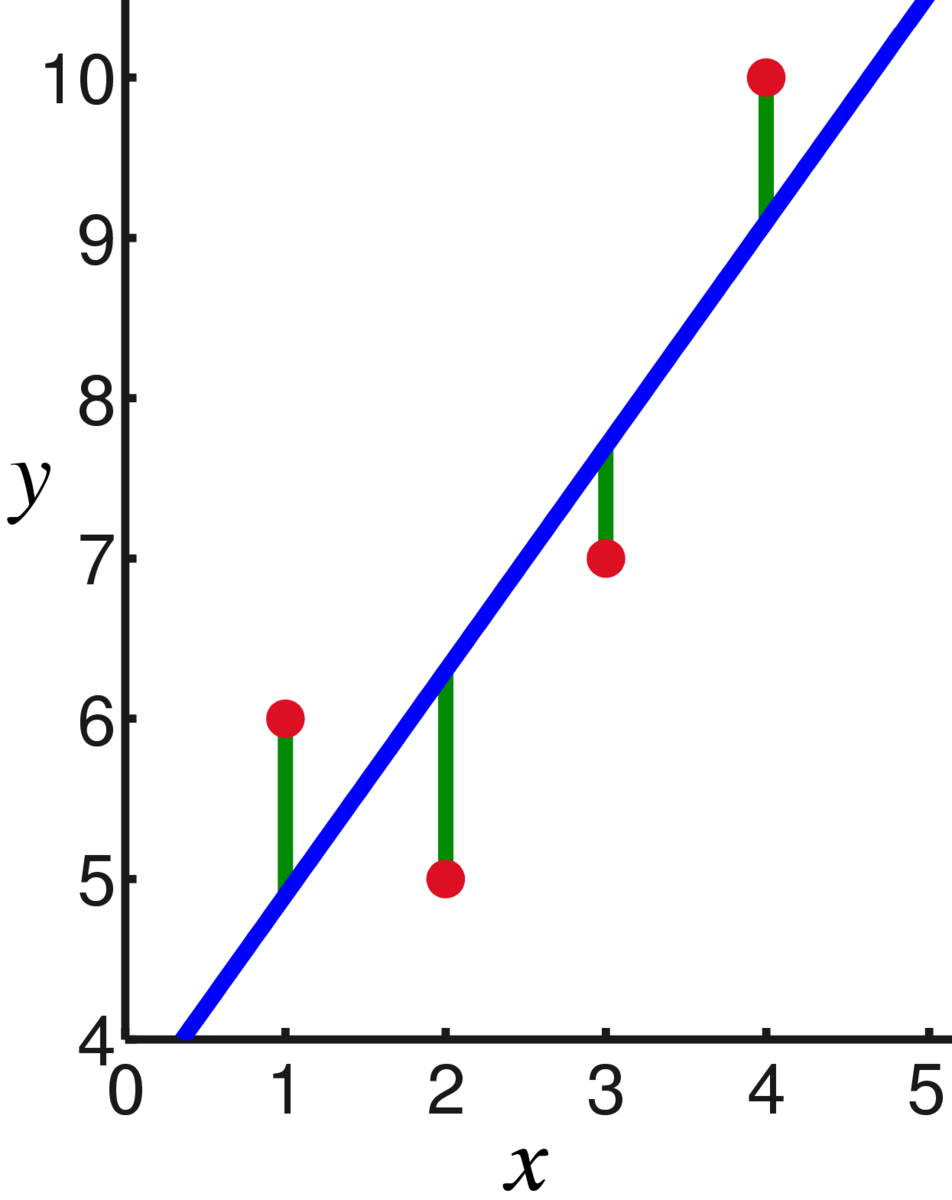
Learning Algorithm

Hypothesis (h) Algorithm

Estimated Price

Size of House

# Linear Regression

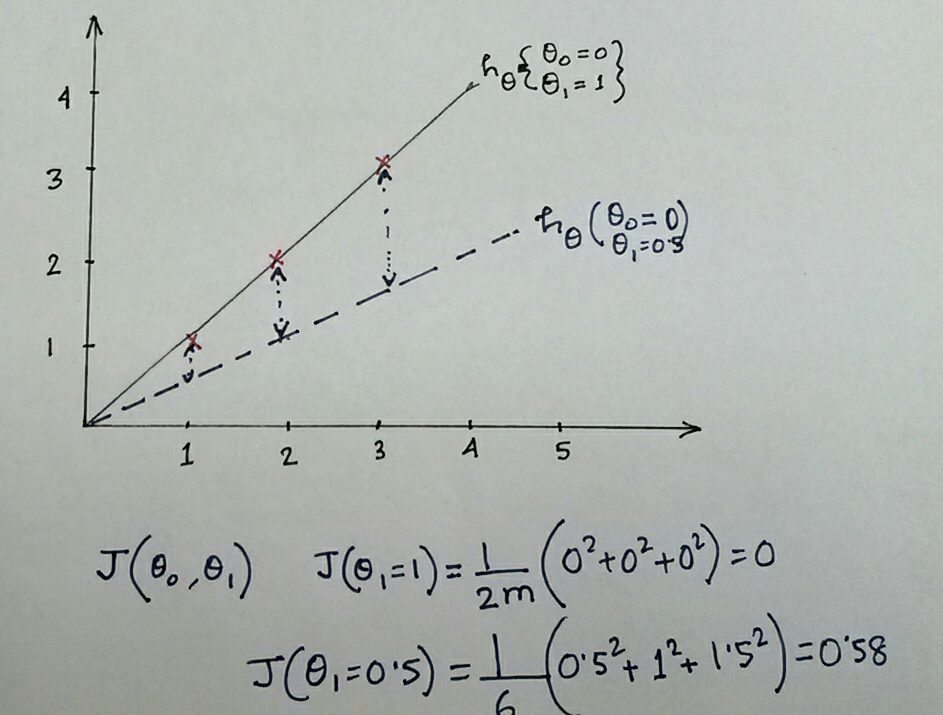


Credit: Wikipedia: Linear Regression

Hypothesis

x = Feature

θ = parameter



Cost function J(θ) can be defined as:

Instead of one variable, the model can be dependent on multiple features or just instead of linear, it may be quadratic or any higher order polynomial. For n such different features, the hypothesis can be written as:

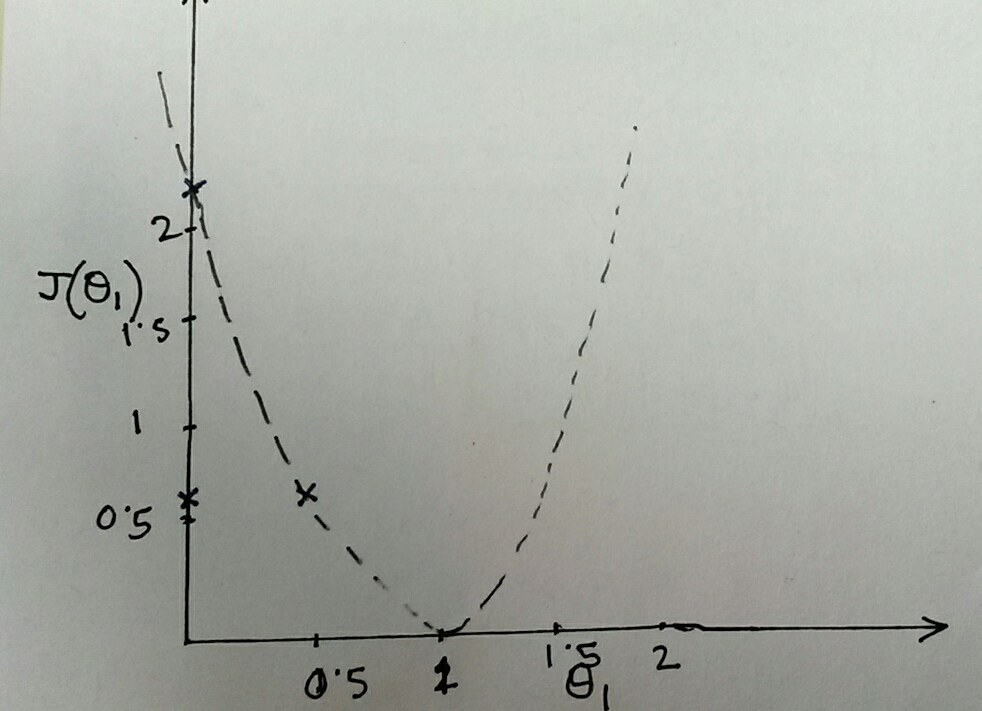
n = number of features

= n + 1 Parameters

The feature vector x and parameter vector θ can be defined as follows:

The above equation is also called as **Multivariate (or Multiple Feature) linear equation**.

Cost function J(θ) can be defined as:



## Gradient Descent

Simultaneously update for every j=0,1, 2…, n

Combining above two equations, we get the following equation:

To use the equation and apply for different features, we get

and so on

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | #Impressions | AdTextQuality | Contains Image | #clicks |
| x0 | x1 | x2 | x3 | y |
| 1 | 2000 | 0.8 | 1 | 125 |
| 1 | 2400 | 0.6 | 0 | 100 |
| 1 | 3000 | 0.5 | 1 | 120 |
| 1 | 1500 | 0.4 | 0 | 25 |
| 1 | 2200 | 0.9 | 1 | 150 |

To get the parameter vector θ, we have to inverse the matrix:

In Octave/Matlab, we can solve this by directly performing a pseudo inverse:

theta = pinv(X’ \* X) \* X’ \* y

# Spark Linear Regression

## Start a Spark Shell:

$ spark-shell --master yarn-client --num-executors 8 --executor-cores 2 --queue marathon --driver-memory 4G --executor-memory 8G --jars /export/home/bparakka/spark\_jars/spark-csv\_2.11-1.2.0.jar,/export/home/bparakka/spark\_jars/spark-avro\_2.10-2.0.1.jar,/export/home/bparakka/spark\_jars/commons-csv-1.2.jar,/export/home/bparakka/spark\_jars/dali-data-spark-5.0.9-all.jar

Let us build the Linear Regression Model:

scala> import org.apache.spark.SparkContext

scala> import org.apache.spark.SparkConf

scala> import org.apache.spark.sql.SQLContext

// Import LinearRegression Module

scala> import org.apache.spark.ml.regression.LinearRegression

scala> val conf = new SparkConf().setAppName("LinearRegressionDemo")

scala> val sc = new SparkContext(conf)

scala> val sqlCtx = new SQLContext(sc)

Holdem provides SqlContext and Spark Context

* Spark context available as sc.
* SQL context available as sqlContext.

scala> val training = sqlContext.read.format("libsvm").load("/user/jmukherj/data/mllib/sample\_linear\_regression\_data.txt")

// In case, you do not have access to data, try uploading data to hadoop and change the user name in the path

// From gateway commandline: hdfs dfs -put  /export/home/jmukherj/spark-demo/data .

scala> val lr = new LinearRegression().setMaxIter(10).setRegParam(0.3).setElasticNetParam(0.8)

lr: org.apache.spark.ml.regression.LinearRegression = linReg\_7db7cf28a785

// Train the model

scala> val lrModel = lr.fit(training)

// Print the coefficients of Linear Regression

scala> println(s"Coefficients: ${lrModel.coefficients} Intercept: ${lrModel.intercept}")

Coefficients: [0.0,0.32242680527120665,-0.3434032467598714,1.9150727151422058,0.05243196193697364,0.7655821177677931,0.0,-0.15066337184764722,-0.21543856849661647,0.2197798181186232] Intercept: 0.1599151948298551

// Summarizing the model

scala> val trainingSummary = lrModel.summary

scala> println(s"numIterations: ${trainingSummary.totalIterations}")

numIterations: 7

scala> println(s"objectiveHistory: ${trainingSummary.objectiveHistory.toList}")

scala> trainingSummary.residuals.show()

+--------------------+

|           residuals|

+--------------------+

|   -9.88820375668679|

|   0.552848441428345|

|  -5.204455166107957|

| -20.566897334292467|

|  -9.449054571607757|

|  -6.909871700295163|

| -10.003746360543401|

|  2.0625840179837054|

|   3.111029172261603|

|  -15.89234518431549|

|   -5.03672751832655|

|   6.483562402235833|

|  12.430572797349361|

| -20.319408828276753|

|  -2.004174515673998|

| -17.869321007390845|

|   7.645833501989321|

| -2.2664040343089678|

|-0.10226933724422116|

| -1.3802150366530652|

+--------------------+

only showing top 20 rows

scala> println(s"RMSE: ${trainingSummary.rootMeanSquaredError}")

RMSE: 10.18912588716637

scala> println(s"r2: ${trainingSummary.r2}")

r2: 0.02285212242080381

# Logistic Regression

Logistic Regression in Spark

scala> import org.apache.spark.ml.classification.LogisticRegression

scala> val logr = new LogisticRegression().setMaxIter(10).setRegParam(0.3).setElasticNetParam(0.8)

// Let us fit the model

scala> val lrModel = logr.fit(training)

lrModel: org.apache.spark.ml.classification.LogisticRegressionModel = logreg\_2d44bcad9da5

scala> println(s"Coefficients: ${lrModel.coefficients} Intercept: ${lrModel.intercept}")

Coefficients: (692,[244,263,272,300,301,328,350,351,378,379,405,406,407,428,433,434,455,456,461,462,483,484,489,490,496,511,512,517,539,540,568],[-7.353983524188245E-5,-9.102738505589519E-5,-1.9467430546904612E-4,-2.0300642473487006E-4,-3.147618331486081E-5,-6.842977602660782E-5,1.5883626898246297E-5,1.4023497091376187E-5,3.543204752496839E-4,1.1443272898171345E-4,1.0016712383667136E-4,6.014109303795461E-4,2.840248179122746E-4,-1.1541084736508883E-4,3.859968863129012E-4,6.350195574241052E-4,-1.1506412384575748E-4,-1.5271865864986881E-4,2.8049338089941963E-4,6.070117471191622E-4,-2.0084596632474579E-4,-1.4210755792901366E-4,2.739010341160868E-4,2.773045624496799E-4,-9.8380270272694E-5,-3.8085224435179237E-4,-2.5315198008556106E-4,2.774771477075417E-4,-2.443619763919299E-4,-0.001539474468759762,-2.307332841133226E-4]) Intercept: 0.22456315961250506

Reference:

1. <http://web.cs.ucla.edu/~mtgarip/linear.html>
2. <https://stackoverflow.com/questions/33842982/pyspark-linear-regression-example-from-official-documentation-bad-results>
3. <https://spark.apache.org/docs/2.1.0/ml-classification-regression.html>
4. <https://mylandingpage.website/blog/2016/02/05/practical-apache-spark-in-10-minutes-mlib/>
5. <https://spark.apache.org/docs/1.6.1/mllib-linear-methods.html>
6. <https://spark.apache.org/downloads.html>
7. <https://spark.apache.org/docs/1.6.1/ml-classification-regression.html#linear-regression>

# Appendix

Print the content of a file (on HDFS)

scala> val training = sqlContext.read.format("libsvm").load("/user/jmukherj/data/mllib/sample\_libsvm\_data.txt")

scala> training.collect().foreach(println)