Spark Machine Learning

# How do machines learn?

Regardless of whether the learner is a human or a machine, the basic learning can be divided into three components:

**Data input:** It utilizes observation, memory storage, and recall to provide a factual basis for further reasoning.

**Abstraction**: It involves the translation of data into broader representations.

**Generalization**: It uses abstracted data to form a basis for action.

.

## Steps to apply machine learning to data

Any machine learning task can be broken down into a series of more manageable steps.

**Collecting data**: Whether the data is written on paper, recorded in text files and spreadsheets, or stored in an SQL database, you will need to gather it in an electronic format suitable for analysis. This data will serve as the learning material an algorithm uses to generate actionable knowledge.

**Exploring and preparing the data:** The quality of any machine learning project is based largely on the quality of data it uses. This step in the machine learning process tends to require a great deal of human intervention. An often cited statistic suggests that 80 percent of the effort in machine learning is devoted to data. Much of this time is spent learning more about the data and its nuances during a practice called data exploration.

**Training a model on the data**: The specific machine learning task will inform the selection of an appropriate algorithm, and the algorithm will represent the data in the form of a model.

**Evaluating model performance**: Because each machine learning model results in a biased solution to the learning problem, it is important to evaluate how well the algorithm learned from its experience. Depending on the type of model used, you might be able to evaluate the accuracy of the model using a test dataset, or you may need to develop measures of performance specific to the intended application.

**Improving model performance:** If better performance is needed, it becomes necessary to utilize more advanced strategies to augment the performance of the model. Sometimes, it may be necessary to switch to a different type of model altogether. You may need to supplement your data with additional data, or perform additional preparatory work as in step two of this process.

# Choosing a machine learning algorithm

The process of choosing a machine learning algorithm involves matching the characteristics of the data to be learned to the biases of the available approaches. Choice of a machine learning algorithm is largely dependent upon the type of data you are analyzing and the proposed task at hand, it is often helpful to be thinking about this process while you are gathering, exploring, and cleaning your data.

# Supervised Learning Algorithms

Nearest Neighbor Classification

naive Bayes Classification

Decision Trees Classification

Classification Rule Learners Classification

Linear Regression Numeric prediction

Regression Trees Numeric prediction

Model Trees Numeric prediction

# Unspervised Learning

Association Rules Pattern detection

k-means Clustering Clustering

**Spark MLlib**

MLlib is Spark’s library of machine learning functions.

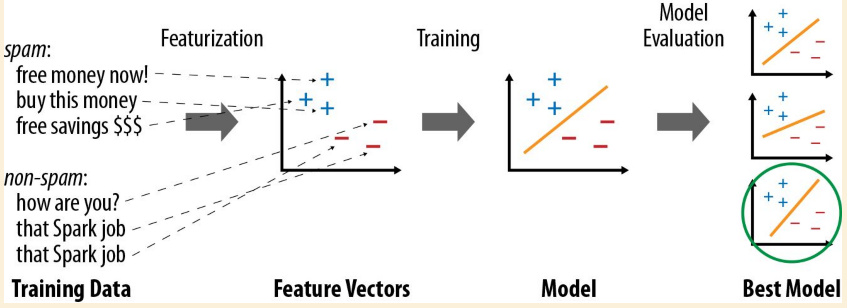
MLlib is designed to run in parallel on clusters.

MLlib contains a variety of learning algorithms and is accessible from all of Spark’s programming languages

MLlib let you invoke various algorithms on distributed datasets, representing all data as RDDs.

*Note: MLlib contains only parallel algorithms that run well on clusters. Some classic ML algorithms are not included because they were not designed for parallel platforms, but in contrast MLlib contains several recent research algorithms for clusters, such as distributed random forests, K-means | |, and alternating least squares. This choice means that MLlib is best suited for running each algorithm on a large dataset.*

**Steps in Machine Learning**



**Data Types**

MLlib contains a few specific data types, located in the org.apache.spark.mllib package (Java/ Scala) or pyspark.mllib (Python). The main ones are:

**Vector:** A mathematical vector. MLlib supports both dense vectors, where every entry is stored, and sparse vectors, where only the nonzero entries are stored to save space. Vectors can be constructed with the mllib.linalg.Vectors class.

**LabeledPoint:** A labeled data point for supervised learning algorithms such as classification and regression. Includes a feature vector and a label (which is a floating-point value). Located in the mllib.regression package.

**Rating:** A rating of a product by a user, used in the mllib.recommendation package for product recommendation. Various Model classes Each Model is the result of a training algorithm, and typically has a predict() method for applying the model to a new data point or to an RDD of new data points.

**Algorithms**

**Feature Extraction**

The mllib.feature package contains several classes for common feature transformations, include algorithms to construct feature vectors from text (or from other tokens), and ways to normalize and scale features.

**TF-IDF :** Term Frequency– Inverse Document Frequency, or TF-IDF, is a simple way to generate feature vectors from text documents. It computes two statistics for each term in each document: the term frequency (TF), which is the number of times the term occurs in that document, and the inverse document frequency (IDF), which measures how (in) frequently a term occurs across the whole document corpus. The product of these values, TF IDF, shows how relevant a term is to a specific document (i.e., if it is common in that document but rare in the whole corpus).

> > > from pyspark.mllib.feature import HashingTF

> > > sentence = "hello hello world"

*> > > words = sentence.split() # Split sentence into a list of terms*

*> > > tf = HashingTF( 10000) # Create vectors of size S = 10,000*

*> > > tf.transform( words) SparseVector( 10000, {3065: 1.0, 6861: 2.0})*

*> > > rdd = sc.wholeTextFiles(" data"). map( lambda (name, text): text.split())*

*> > > tfVectors = tf.transform( rdd) # Transforms an entire RDD*

**Scaling** : Most machine learning algorithms consider the magnitude of each element in the feature vector, and thus work best when the features are scaled so they weigh equally (e.g., all features have a mean of 0 and standard deviation of 1). Once you have built feature vectors, you can use the StandardScaler class in MLlib to do this scaling, both for the mean and the standard deviation. You create a StandardScaler, call fit() on a dataset to obtain a StandardScalerModel (i.e., compute the mean and variance of each column), and then call transform () on the model to scale a dataset.

Example

from pyspark.mllib.feature

import StandardScaler

vectors = [Vectors.dense([-2.0, 5.0, 1.0]), Vectors.dense([ 2.0, 0.0, 1.0])]

dataset = sc.parallelize( vectors)

scaler = StandardScaler( withMean = True, withStd = True)

model = scaler.fit( dataset)

result = model.transform( dataset)

# Result: {[-0.7071, 0.7071, 0.0], [0.7071, -0.7071, 0.0]}