# 机器学习及 SparkMLlib 简介

# 目 录

# 机器学习及 SparkMLlib 简介

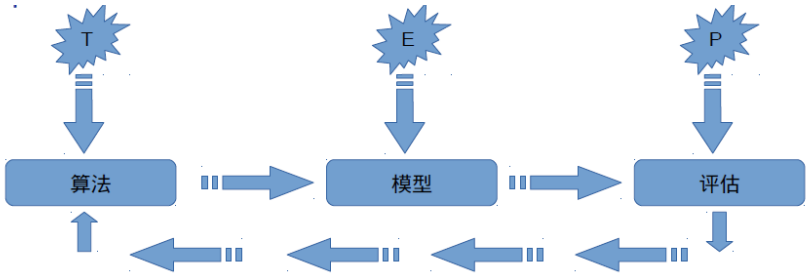
# 1 机器学习概念

## 1.1 机器学习的定义

在维基百科上对机器学习提出以下几种定义：

- “机器学习是一门人工智能的科学，该领域的主要研究对象是人工智能，特别是如何在经验学习中改善具体算法的性能”。
- “机器学习是对能通过经验自动改进的计算机算法的研究”。
- “机器学习是用数据或以往的经验，以此优化计算机程序的性能标准。” 一种经常引用的英文定义是：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。

可以看出机器学习强调三个关键词：算法、经验、性能，其处理过程如下图所示。




上图表明机器学习是数据通过算法构建出模型并对模型进行评估，评估的性能如果达到要求就拿这个模型来测试其他的数据，如果达不到要求就要调整算法来重新建立模型，再次进行评估，如此循环往复，最终获得满意的经验来处理其他的数据。

## 1.2 机器学习的分类

### 1.2.1 监督学习

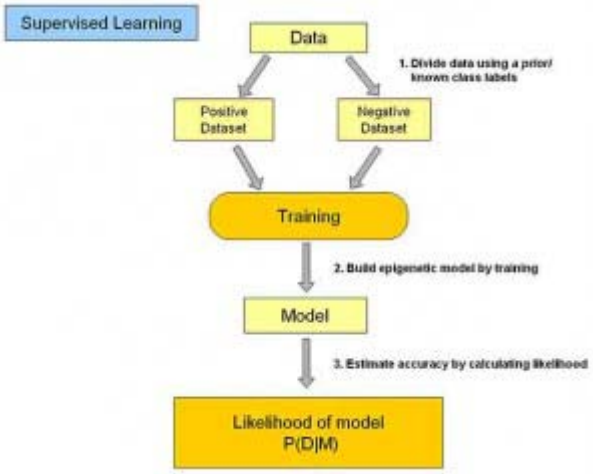
监督是从给定的训练数据集中学习一个函数（模型），当新的数据到来时，可以根据这个函数（模型）预测结果。监督学习的训练集要求包括输入和输出，也可以说是特征和目标。训练集中的目标是由人标注（标量）的。在监督式学习下，输入数据被称为“训练数据”，每组训练数据有一个明确的标识或结果，如对防垃圾邮件系统中“垃圾邮件”、“非垃圾邮件”，对手写数字识别中

的“1”、“2”、“3”等。在建立预测模型时，监督式学习建立一个学习过程，将预测结果与“训练数据”的实际结果进行比较，不断调整预测模型，直到模型的预测结果达到一个预期的准确率。常见的监督学习算法包括回归分析和统计分类：

- 二元分类是机器学习要解决的基本问题，将测试数据分成两个类，如垃圾邮件的判别、房贷是否允许等问题的判断。
- 多元分类是二元分类的逻辑延伸。例如，在因特网的流分类的情况下，根据问题的分类，网页可以被归类为体育、新闻、技术等，依此类推。

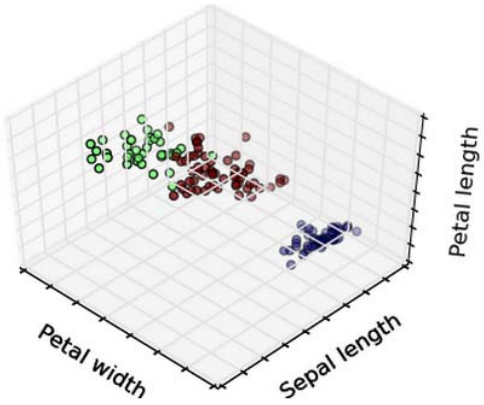
监督学习常常用于分类，因为目标往往是让计算机去学习我们已经创建好的分类系统。数字识别再一次成为分类学习的常见样本。一般来说，对于那些有用的分类系统和容易判断的分类系统，分类学习都适用。

监督学习是训练神经网络和决策树的最常见技术。神经网络和决策树技术高度依赖于事先确定的分类系统给出的信息。对于神经网络来说，分类系统用于判断网络的错误，然后调整网络去适应它；对于决策树，分类系统用来判断哪些属性提供了最多的信息，如此一来可以用它解决分类系统的问题。




## 1.2.2 无监督学习

与监督学习相比，无监督学习的训练集没有人为标注的结果。在非监督式学习中，数据并不被特别标识，学习模型是为了推断出数据的一些内在结构。常见的应用场景包括关联规则的学习以及聚类等。常见算法包括 Apriori 算法和 k-Means 算法。这类学习类型的目标不是让效用函数最大化，而是找到训练数据中的近似点。聚类常常能发现那些与假设匹配的相当好的直观分类，例如基于人口统计的聚合个体可能会在一个群体中形成一个富有的聚合 ,以及其他的贫穷的聚合。



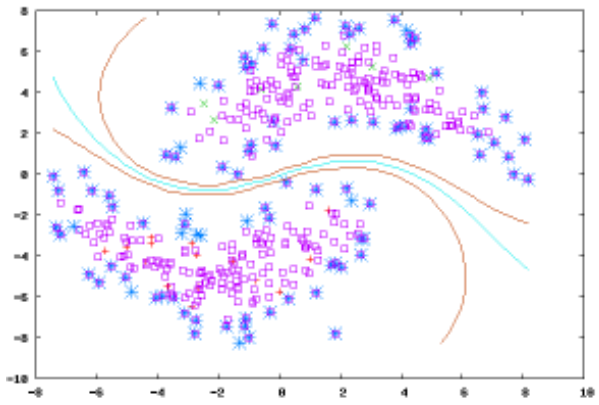

非监督学习看起来非常困难：目标是我们不告诉计算机怎么做，而是让它（计算机）自己去学习怎样做一些事情。非监督学习一般有两种思路：第一种思路是在指导 Agent 时不为其指定明确的分类，而是在成功时采用某种形式的激励制度。需要注意的是，这类训练通常会置于决策问题的框架里，因为它的目标不是产生一个分类系统，而是做出最大回报的决定。这种思路很好地概括了现实世界，Agent 可以对那些正确的行为做出激励，并对其他的行为进行处罚。

因为无监督学习假定没有事先分类的样本，这在一些情况下会非常强大，例如，我们的分类方法可能并非最佳选择。在这方面一个突出的例子是 Backgammon（西洋双陆棋）游戏，有一系列计算机程序（例如 neuro-gammon 和 TD-gammon）通过非监督学习自己一遍又一遍地玩这个游戏，变得比最强的人类棋手还要出色。这些程序发现的一些原则甚至令双陆棋专家都感到惊讶，并且它们比那些使用预分类样本训练的双陆棋程序工作得更出色。

### 1.2.3 半监督学习

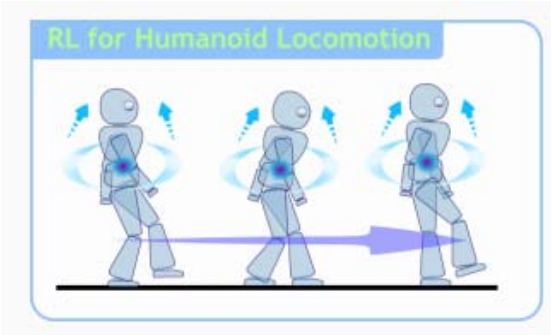
半监督学习（Semi-supervised Learning）是介于监督学习与无监督学习之间一种机器学习方式，是模式识别和机器学习领域研究的重点问题。它主要考虑如何利用少量的标注样本和大量的未标注样本进行训练和分类的问题。半监督学习对于减少标注代价，提高学习机器性能具有非常重大的实际意义。主要算法有五类：基于概率的算法；在现有监督算法基础上进行修改的方法；直接依赖于聚类假设的方法等，在此学习方式下，输入数据部分被标识，部分没有被标识，这种学习模型可以用来进行预测，但是模型首先需要学习数据的内在结构以便合理地组织数据来进行预测。应用场景包括分类和回归，算法包括一些对常用监督式学习算法的延伸，这些算法首先试图对未标识数据进行建模，在此基础上再对标识的数据进行预测，如图论推理算法（Graph Inference）或者拉普拉斯支持向量机（Laplacian SVM）等。

半监督学习分类算法提出的时间比较短，还有许多方面没有更深入的研究。半监督学习从诞生以来，主要用于处理人工合成数据，无噪声干扰的样本数据是当前大部分半监督学习方法使用的数据，而在实际生活中用到的数据却大部分不是无干扰的，通常都比较难以得到纯样本数据。



### 1.2.4 强化学习

强化学习通过观察来学习动作的完成，每个动作都会对环境有所影响，学习对象根据观察到的周围环境的反馈来做出判断。在这种学习模式下，输入数据作为对模型的反馈，不像监督模型那样，输入数据仅仅是作为一个检查模型对错的方式，在强化学习下，输入数据直接反馈到模型，模型必须对此立刻做出调整。常见的应用场景包括动态系统以及机器人控制等。常见算法包括 Q-Learning 以及时间差学习（Temporal difference learning）。




在企业数据应用的场景下，人们最常用的可能就是监督式学习和非监督式学习的模型。在图像识别等领域，由于存在大量的非标识的数据和少量的可标识数据，目前半监督式学习是一个很热的话题。而强化学习更多地应用在机器人控制及其他需要进行系统控制的领域。

## 1.3 机器学习的常见算法

常见的机器学习算法有：

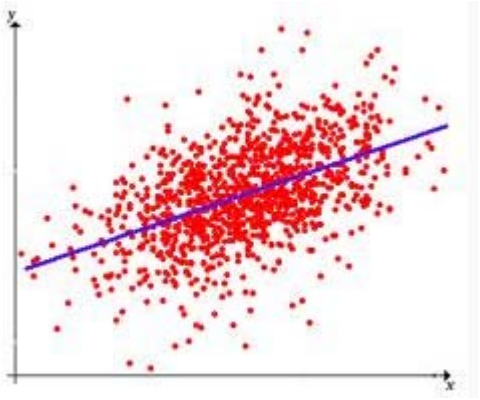
- 构造条件概率：回归分析和统计分类；
- 人工神经网络；
- 决策树；
- 高斯过程回归；
- 线性判别分析；

- 最近邻居法；
- 感知器；
- 径向基函数核；
- 支持向量机；
- 通过再生模型构造概率密度函数；
- 最大期望算法；
- graphical model：包括贝叶斯网和 Markov 随机场；
- Generative Topographic Mapping；
- 近似推断技术；
- 马尔可夫链蒙特卡罗方法；
- 变分法；
- 最优化：大多数以上方法，直接或者间接使用最优化算法。

根据算法的功能和形式的类似性，我们可以把算法分类，比如说基于树的算法，基于神经网络的算法等等。当然，机器学习的范围非常庞大，有些算法很难明确归类到某一类。而对于有些分类来说，同一分类的算法可以针对不同类型的问题，下面用一些相对比较容易理解的方式来解析一些主要的机器学习算法：

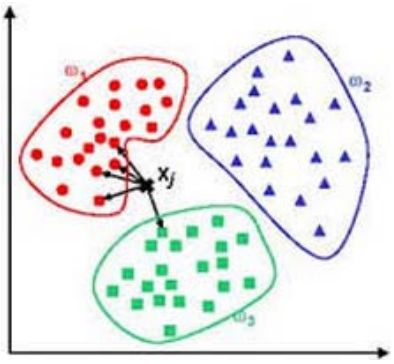
### 1.3.1 回归算法

回归算法是试图采用对误差的衡量来探索变量之间的关系的一类算法。回归算法是统计机器学习的利器。在机器学习领域，人们说起回归，有时候是指一类问题，有时候是指一类算法，这一点常常会使初学者有所困惑。常见的回归算法包括：最小二乘法（Ordinary Least Square），逻辑回归（Logistic Regression），逐步式回归（Stepwise Regression），多元自适应回归样条（Multivariate Adaptive Regression Splines）以及本地散点平滑估计（Locally Estimated Scatterplot Smoothing）。



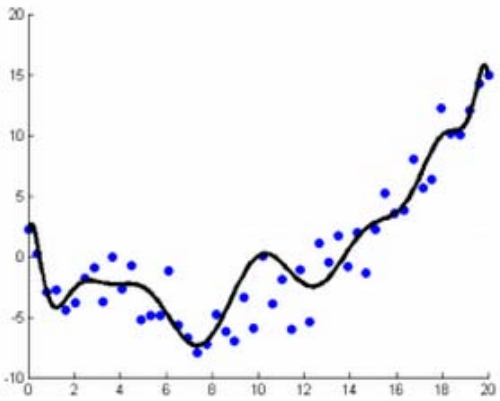
## 1.3.2 基于实例的算法

基于实例的算法常常用来对决策问题建立模型，这样的模型常常先选取一批样本数据，然后根据某些近似性把新数据与样本数据进行比较。通过这种方式来寻找最佳的匹配。因此，基于实例的算法常常也被称为“赢家通吃”学习或者“基于记忆的学习”。常见的算法包括 k-Nearest Neighbor (KNN)、学习矢量量化（Learning Vector Quantization， LVQ）以及自组织映射算法（Self-Organizing Map，SOM）



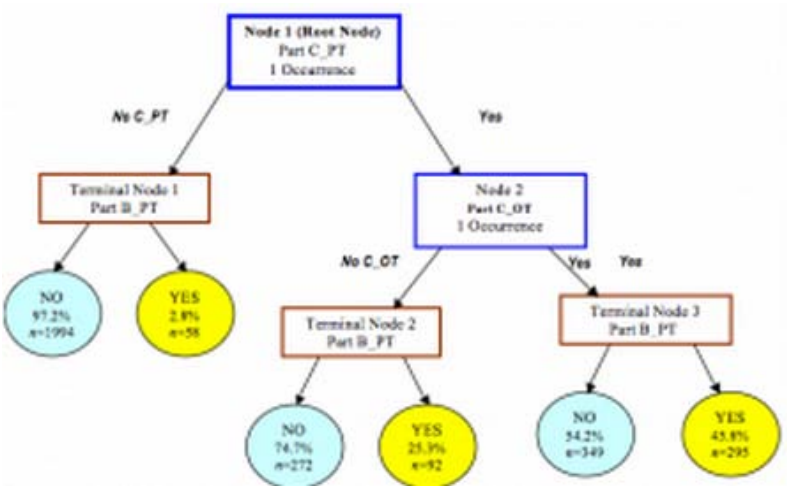
## 1.3.3 正则化方法

正则化方法是其他算法（通常是回归算法）的延伸，根据算法的复杂度对算法进行调整。正则化方法通常对简单模型予以奖励而对复杂算法予以惩罚。常见的算法包括 :Ridge Regression、Least Absolute Shrinkage and Selection Operator（LASSO）以及弹性网络（Elastic Net）。



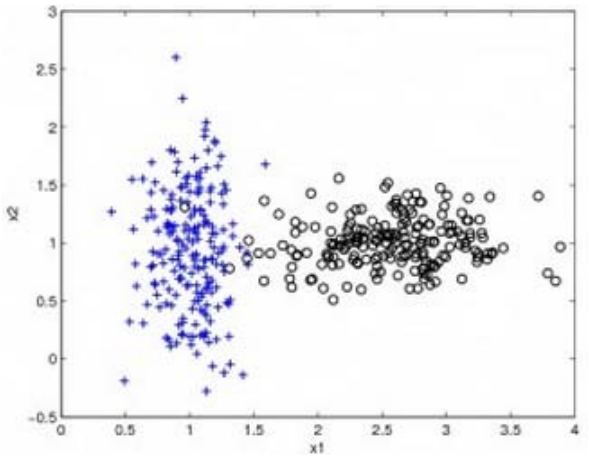
### 1.3.4 决策树学习

决策树算法根据数据的属性采用树状结构建立决策模型， 决策树模型常常用来解决分类和回归问题。常见的算法包括：分类及回归树（Classification And Regression Tree， CART）、 ID3 (Iterative Dichotomiser 3)、C4.5、Chi-squared Automatic Interaction Detection (CHAID)、Decision Stump、机森林（Random Forest）、多元自适应回归样条（MARS）以及梯度推进机（Gradient Boosting Machine，GBM）。



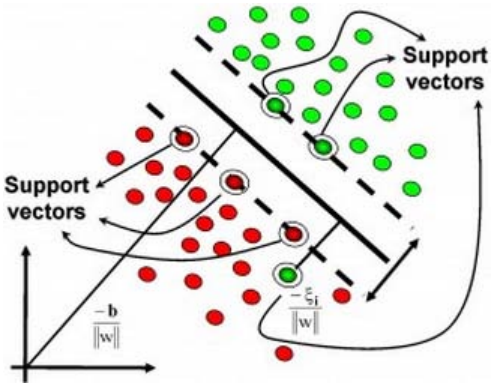

### 1.3.5 贝叶斯学习

贝叶斯方法算法是基于贝叶斯定理的一类算法，主要用来解决分类和回归问题。常见算法包括：朴素贝叶斯算法、平均单依赖估计（Averaged One-Dependence Estimators， AODE）以及 Bayesian Belief Network（BBN）。



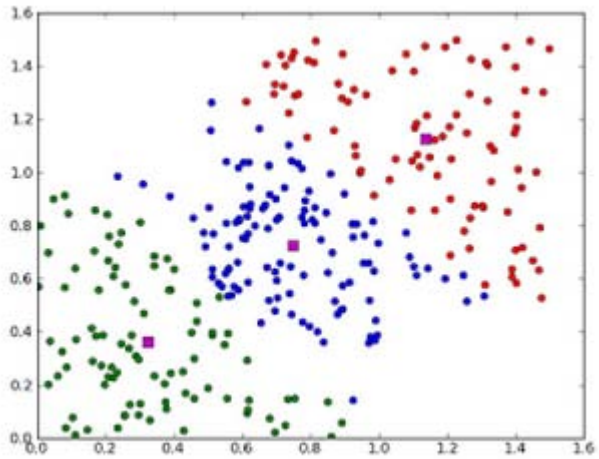

### 1.3.6 基于核的算法

基于核的算法中最著名的莫过于支持向量机（SVM）了。基于核的算法把输入数据映射到一个高阶的向量空间， 在这些高阶向量空间里， 有些分类或者回归问题能够更容易解决。 常见的基于核的算法包括：支持向量机（Support Vector Machine，SVM）、径向基函数（Radial Basis Function，RBF)以及线性判别分析（Linear Discriminate Analysis，LDA)等。



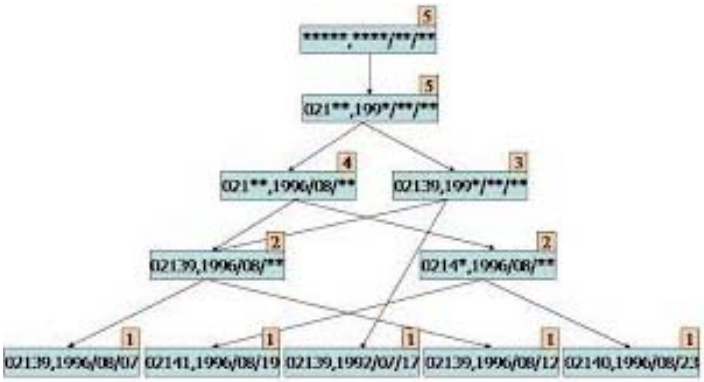

## 1.3.7 聚类算法

聚类就像回归一样，有时候人们描述的是一类问题，有时候描述的是一类算法。聚类算法通常按照中心点或者分层的方式对输入数据进行归并。所有的聚类算法都试图找到数据的内在结构，以便按照最大的共同点将数据进行归类。常见的聚类算法包括 k-Means 算法以及期望最大化算法（Expectation Maximization，EM）。



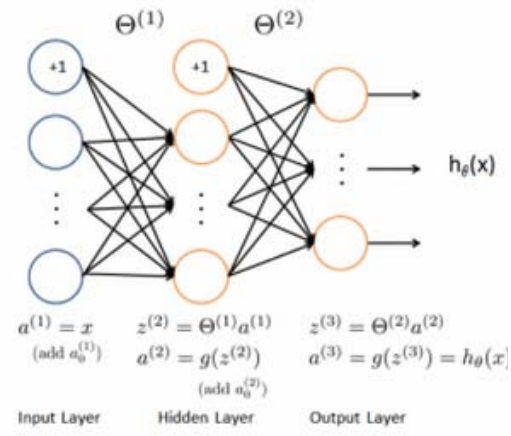
## 1.3.8 关联规则学习

关联规则学习通过寻找最能够解释数据变量之间关系的规则，来找出大量多元数据集中有用的关联规则。常见算法包括 Apriori 算法和 Eclat 算法等。



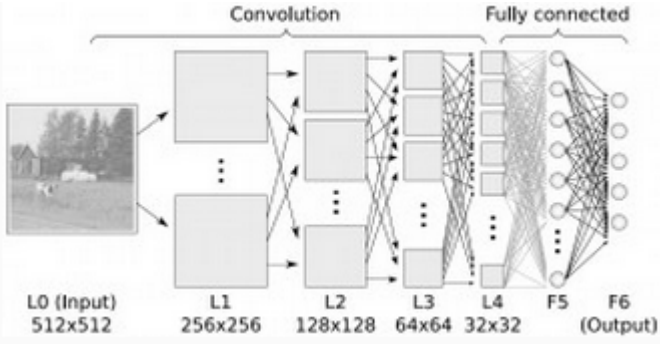
### 1.3.9 人工神经网络算法

人工神经网络算法模拟生物神经网络，是一类模式匹配算法。通常用于解决分类和回归问题。人工神经网络是机器学习的一个庞大的分支，有几百种不同的算法（其中深度学习就是其中的一类算法，我们会单独讨论）。重要的人工神经网络算法包括：感知器神经网络（Perceptron Neural Network）、反向传递（Back Propagation）、Hopfield 网络、自组织映射（Self-Organizing Map, SOM）、学习矢量量化（Learning Vector Quantization，LVQ）。




### 1.3.10 深度学习算法

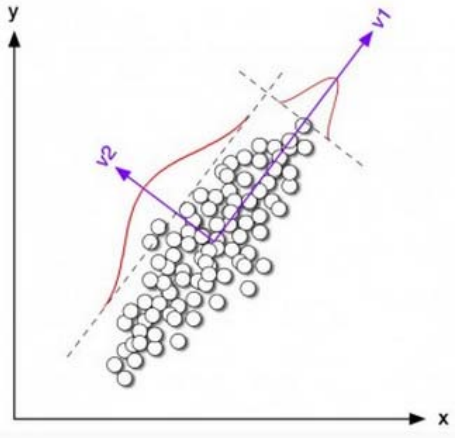
深度学习算法是对人工神经网络的发展，在近期赢得了很多关注，特别是百度也开始发力深度学习后，更是在国内引起了很多关注。在计算能力变得日益廉价的今天，深度学习试图建立大得多也复杂得多的神经网络。很多深度学习的算法是半监督式学习算法，用来处理存在少量未标识数据的大数据集。常见的深度学习算法包括：受限波尔兹曼机（Restricted Boltzmann Machine，RBN）、Deep Belief Networks（DBN）、卷积网络（Convolutional Network）、堆栈式自动编码器（Stacked Auto-encoders）。




### 1.3.11 降低维度算法

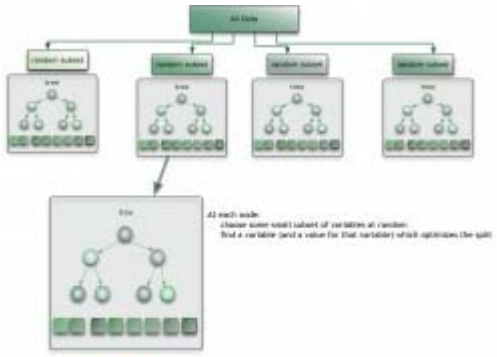
像聚类算法一样，降低维度算法试图分析数据的内在结构，不过降低维度算法是以非监督学习的方式，试图利用较少的信息来归纳或者解释数据。这类算法可以用于高维数据的可视化或者用来简化数据以便监督式学习使用。常见的算法包括：主成份分析（Principle Component

Analysis， PCA）、偏最小二乘回归（Partial Least Square Regression，PLS）、 Sammon 映射、多维尺度（Multi-Dimensional Scaling, MDS）、投影追踪（Projection Pursuit）等。




### 1.3.12 集成算法

集成算法用一些相对较弱的学习模型独立地对同样的样本进行训练，然后把结果整合起来进行整体预测。集成算法的主要难点在于究竟集成哪些独立的较弱的学习模型以及如何把学习结果整合起来。这是一类非常强大的算法，同时也非常流行。常见的算法包括：Boosting、Bootstrapped Aggregation（Bagging）、AdaBoost、堆叠泛化（Stacked Generalization，Blending）、梯度推进机（Gradient Boosting Machine, GBM）、随机森林（Random Forest）。



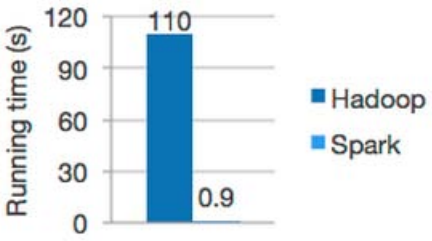
# 2 Spark MLlib 介绍

Spark 之所以在机器学习方面具有得天独厚的优势，有以下几点原因：

（1）机器学习算法一般都有很多个步骤迭代计算的过程，机器学习的计算需要在多次迭代后获得足够小的误差或者足够收敛才会停止，迭代时如果使用 Hadoop 的 MapReduce 计算框架，每次计算都要读/写磁盘以及任务的启动等工作，这回导致非常大的 I/O 和 CPU 消耗。而 Spark 基于内存的计算模型天生就擅长迭代计算，多个步骤计算直接在内存中完成，只有在必要时才会操作磁盘和网络，所以说 Spark 正是机器学习的理想的平台。

（2）从通信的角度讲，如果使用 Hadoop 的 MapReduce 计算框架，JobTracker 和 TaskTracker 之间由于是通过 heartbeat 的方式来进行的通信和传递数据，会导致非常慢的执行速度，而 Spark 具有出色而高效的 Akka 和 Netty 通信系统，通信效率极高。

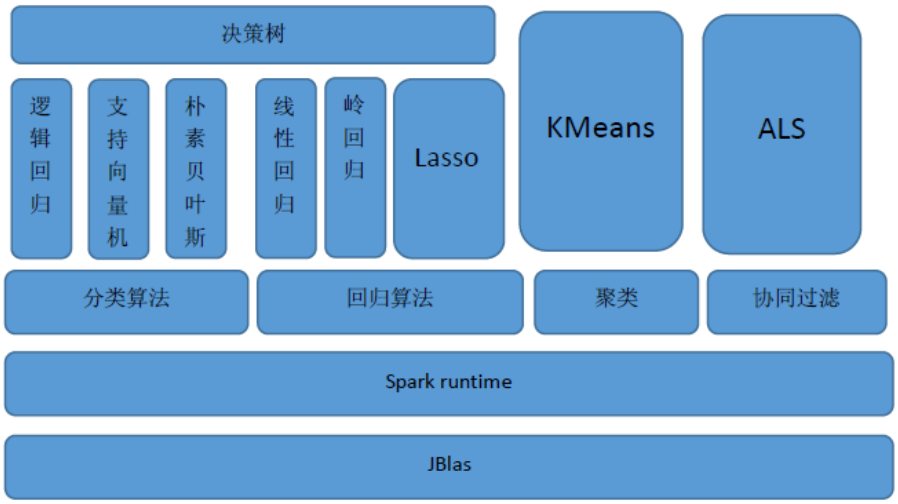
MLlib(Machine Learnig lib) 是 Spark 对常用的机器学习算法的实现库，同时包括相关的测试和数据生成器。Spark 的设计初衷就是为了支持一些迭代的 Job，这正好符合很多机器学习算法的特点。在 Spark 官方首页中展示了 Logistic Regression 算法在 Spark 和 Hadoop 中运行的性能比较，如图下图所示。



Logistic regression in Hadoop and Spark

可以看出在 Logistic Regression 的运算场景下，Spark 比 Hadoop 快了 100 倍以上！

MLlib 目前支持 4 种常见的机器学习问题: 分类、回归、聚类和协同过滤，MLlib 在 Spark 整个生态系统中的位置如图下图所示。

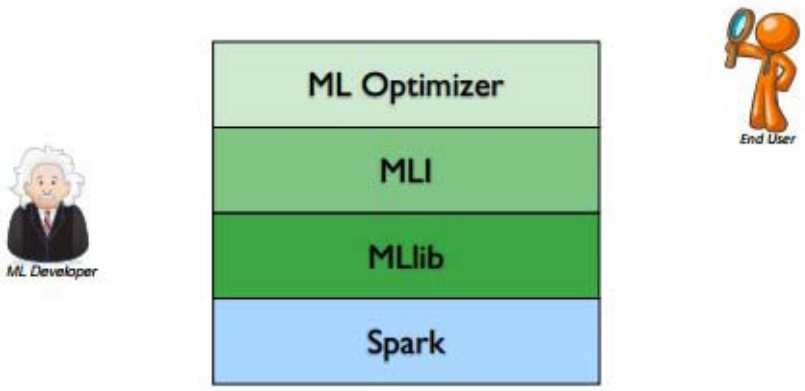



MLlib 基于 RDD，天生就可以与 Spark SQL、GraphX、Spark Streaming 无缝集成，以 RDD 为基石，4 个子框架可联手构建大数据计算中心！

MLlib 是 MLBase 一部分，其中 MLBase 分为四部分：MLlib、MLI、ML Optimizer 和 MLRuntime。

- ML Optimizer 会选择它认为最适合的已经在内部实现好了的机器学习算法和相关参数，来处理用户输入的数据，并返回模型或别的帮助分析的结果；

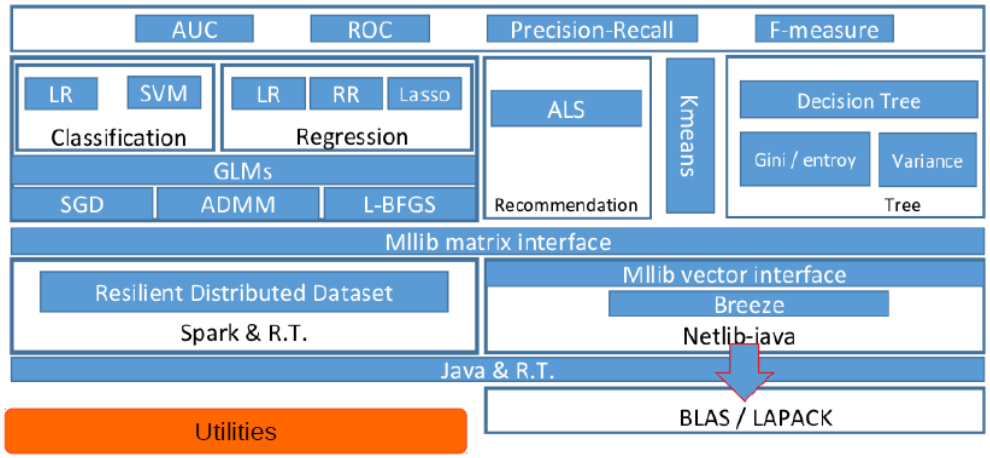
- MLI 是一个进行特征抽取和高级 ML 编程抽象的算法实现的 API 或平台；
- MLlib 是 Spark 实现一些常见的机器学习算法和实用程序，包括分类、回归、聚类、协同过滤、降维以及底层优化，该算法可以进行可扩充； MLRuntime 基于 Spark 计算框架，将 Spark 的分布式计算应用到机器学习领域。




# 3 Spark MLlib 架构解析

从架构图可以看出 MLlib 主要包含三个部分：

- **底层基础**：包括 Spark 的运行库、矩阵库和向量库；
- **算法库**：包含广义线性模型、推荐系统、聚类、决策树和评估的算法；
- **实用程序**：包括测试数据的生成、外部数据的读入等功能。




## 3.1 MLlib 的底层基础解析

底层基础部分主要包括向量接口和矩阵接口，这两种接口都会使用 Scala 语言基于 Netlib 和 BLAS/LAPACK 开发的线性代数库 Breeze。

MLlib 支持本地的密集向量和稀疏向量，并且支持标量向量。

MLlib 同时支持本地矩阵和分布式矩阵，支持的分布式矩阵分为 RowMatrix、IndexedRowMatrix、CoordinateMatrix 等。

关于密集型和稀疏型的向量 Vector 的示例如下所示。

```
import org.apache.spark.mllib.linalg.{Vector, Vectors}

// Create a dense vector (1.0, 0.0, 3.0).
val dv: Vector = Vectors.dense(1.0, 0.0, 3.0)
// Create a sparse vector (1.0, 0.0, 3.0) by specifying its indices and values corresponding to nonzero entries.
val sv1: Vector = Vectors.sparse(3, Array(0, 2), Array(1.0, 3.0))
// Create a sparse vector (1.0, 0.0, 3.0) by specifying its nonzero entries.
val sv2: Vector = Vectors.sparse(3, Seq((0, 1.0), (2, 3.0)))
```

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

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

疏矩阵在含有大量非零元素的向量 Vector 计算中会节省大量的空间并大幅度提高计算速度，如下图所示。

Training set:

- 12 million examples
- 500 features
- sparsity: 10%

| | dense | sparse |
|---|---|---|
| storage | 47GB | 7GB |
| time | 240s | 58s |

40GB savings in storage, 4x speedup in computation

标量 LabledPoint 在实际中也被大量使用，例如判断邮件是否为垃圾邮件时就可以使用类似于以下的代码：

```
import org.apache.spark.mllib.linalg.Vectors
import org.apache.spark.mllib.regression.LabeledPoint

// Create a labeled point with a positive label and a dense feature vector.
val pos = LabeledPoint(1.0, Vectors.dense(1.0, 0.0, 3.0))

// Create a labeled point with a negative label and a sparse feature vector.
val neg = LabeledPoint(0.0, Vectors.sparse(3, Array(0, 2), Array(1.0, 3.0)))
```

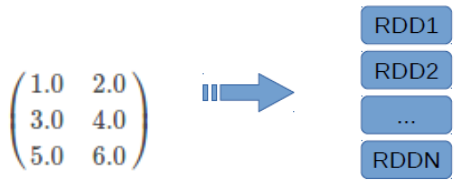
可以把表示为 1.0 的判断为正常邮件，而表示为 0.0 则作为垃圾邮件来看待。

对于矩阵 Matrix 而言，本地模式的矩阵如下所示。

$$\begin{pmatrix} 1.0 & 2.0 \\ 3.0 & 4.0 \\ 5.0 & 6.0 \end{pmatrix}$$

```
import org.apache.spark.mllib.linalg.{Matrix, Matrices}

// Create a dense matrix ((1.0, 2.0), (3.0, 4.0), (5.0, 6.0))
val dm: Matrix = Matrices.dense(3, 2, Array(1.0, 3.0, 5.0, 2.0, 4.0, 6.0))
```

分布式矩阵如下所示。




RowMatrix 直接通过 RDD[Vector]来定义并可以用来统计平均数、方差、协同方差等：

```
Import org.apache.spark.mllib.linalg.Vector
Import org.apache.spark.mllib.linalg.distributed.RowMatrix

val rows: RDD[Vector] = ... // an RDD of local vectors
// Create a RowMatrix from an RDD[Vector].
val mat: RowMatrix = new RowMatrix(rows)

// Get its size.
val m = mat.numRows()
val n = mat.numCols()
```

```
Import org.apache.spark.mllib.linalg.Matrix
Import org.apache.spark.mllib.linalg.distributed.RowMatrix
Import org.apache.spark.mllib.stat.MultivariateStatisticalSummary

val mat: RowMatrix = ... // a RowMatrix

// Compute column summary statistics.
val summary: MultivariateStatisticalSummary = mat.computeColumnSummaryStatistics()
println(summary.mean) // a dense vector containing the mean value for each column
println(summary.variance) // column-wise variance
println(summary.numNonzeros) // number of nonzeros in each column

// Compute the covariance matrix.
val cov: Matrix = mat.computeCovariance()
```

而 IndexedRowMatrix 是带有索引的 Matrix，但其可以通过 toRowMatrix 方法来转换为 RowMatrix，从而利用其统计功能，代码示例如下所示。

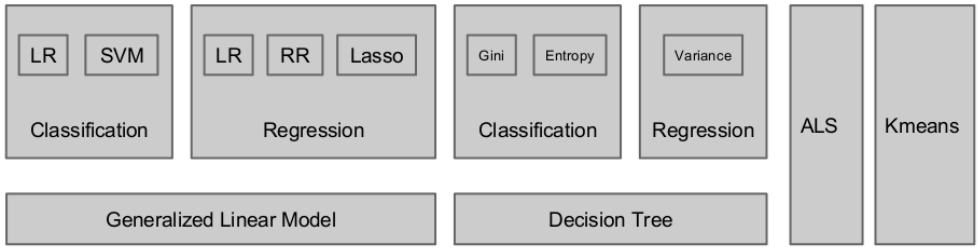
```
Import org.apache.spark.mllib.linalg.distributed.{IndexedRow, IndexedRowMatrix, RowMatrix}

val rows: RDD[IndexedRow] = ... // an RDD of indexed rows
// Create an IndexedRowMatrix from an RDD[IndexedRow].
val mat: IndexedRowMatrix = new IndexedRowMatrix(rows)

// Get its size.
val m = mat.numRows()
val n = mat.numCols()

// Drop its row indices.
val rowMat: RowMatrix = mat.toRowMatrix()
```

CoordinateMatrix 常用于稀疏性比较高的计算中，是由 RDD[MatrixEntry]来构建的，MatrixEntry 是一个 Tuple 类型的元素，其中包含行、列和元素值，代码示例如下所示：

```scala
import org.apache.spark.mllib.linalg.distributed.{CoordinateMatrix, MatrixEntry}

val entries: RDD[MatrixEntry] = ... // an RDD of matrix entries
// Create a CoordinateMatrix from an RDD[MatrixEntry].
val mat: CoordinateMatrix = new CoordinateMatrix(entries)

// Get its size.
val m = mat.numRows()
val n = mat.numCols()

// Convert it to an IndexRowMatrix whose rows are sparse vectors.
val indexedRowMatrix = mat.toIndexedRowMatrix()
```

## 3.2 MLlib 的算法库分析

下图是 MLlib 算法库的核心内容。




在这里我们分析一些 Spark 中常用的算法：

### 3.2.1 分类算法

分类算法属于监督式学习，使用类标签已知的样本建立一个分类函数或分类模型，应用分类模型，能把数据库中的类标签未知的数据进行归类。分类在数据挖掘中是一项重要的任务，目前在商业上应用最多，常见的典型应用场景有流失预测、精确营销、客户获取、个性偏好等。MLlib 目前支持分类算法有：逻辑回归、支持向量机、朴素贝叶斯和决策树。

案例：导入训练数据集，然后在训练集上执行训练算法，最后在所得模型上进行预测并计算训练误差。

```scala
import org.apache.spark.SparkContext
import org.apache.spark.mllib.classification.SVMWithSGD
import org.apache.spark.mllib.regression.LabeledPoint

// 加载和解析数据文件
val data = sc.textFile("mllib/data/sample_svm_data.txt")
val parsedData = data.map { line =>
  val parts = line.split(' ')
```

```
  LabeledPoint(parts(0).toDouble, parts.tail.map(x => x.toDouble).toArray)
}

// 设置迭代次数并进行进行训练
val numIterations = 20
val model = SVMWithSGD.train(parsedData, numIterations)

// 统计分类错误的样本比例
val labelAndPreds = parsedData.map { point =>
val prediction = model.predict(point.features)
(point.label, prediction)
}
val trainErr = labelAndPreds.filter(r => r._1 != r._2).count.toDouble / parsedData.count
println("Training Error = " + trainErr)
```

### 3.2.2 回归算法

回归算法属于监督式学习，每个个体都有一个与之相关联的实数标签，并且我们希望在给出用于表示这些实体的数值特征后，所预测出的标签值可以尽可能接近实际值。MLlib 目前支持回归算法有：线性回归、岭回归、Lasso 和决策树。

案例：导入训练数据集，将其解析为带标签点的 RDD，使用 LinearRegressionWithSGD 算法建立一个简单的线性模型来预测标签的值，最后计算均方差来评估预测值与实际值的吻合度。

```
import org.apache.spark.mllib.regression.LinearRegressionWithSGD
import org.apache.spark.mllib.regression.LabeledPoint

// 加载和解析数据文件
val data = sc.textFile("mllib/data/ridge-data/lpsa.data")
val parsedData = data.map { line =>
  val parts = line.split(',')
  LabeledPoint(parts(0).toDouble, parts(1).split(' ').map(x => x.toDouble).toArray)
}

//设置迭代次数并进行训练
val numIterations = 20
val model = LinearRegressionWithSGD.train(parsedData, numIterations)
```

```
// 统计回归错误的样本比例
val valuesAndPreds = parsedData.map { point =>
val prediction = model.predict(point.features)
(point.label, prediction)
}
val MSE = valuesAndPreds.map{ case(v, p) => math.pow((v - p), 2)}.reduce(_ +
_)/valuesAndPreds.count
println("training Mean Squared Error = " + MSE)
```

### 3.2.3 聚类算法

聚类算法属于非监督式学习，通常被用于探索性的分析，是根据“物以类聚”的原理，将本身没有类别的样本聚集成不同的组，这样的一组数据对象的集合叫做簇，并且对每一个这样的簇进行描述的过程。它的目的是使得属于同一簇的样本之间应该彼此相似，而不同簇的样本应该足够不相似，常见的典型应用场景有客户细分、客户研究、市场细分、价值评估。MLlib 目前支持广泛使用的 KMmeans 聚类算法。

案例：导入训练数据集，使用 KMeans 对象来将数据聚类到两个类簇当中，所需的类簇个数会被传递到算法中，然后计算集内均方差总和 (WSSSE)，可以通过增加类簇的个数 k 来减小误差。 实际上，最优的类簇数通常是 1，因为这一点通常是 WSSSE 图中的 “低谷点”。

```
import org.apache.spark.mllib.clustering.KMeans

// 加载和解析数据文件
val data = sc.textFile("kmeans_data.txt")
val parsedData = data.map( _.split(' ').map(_.toDouble))
// 设置迭代次数、类簇的个数
val numIterations = 20
val numClusters = 2

// 进行训练
val clusters = KMeans.train(parsedData, numClusters, numIterations)

// 统计聚类错误的样本比例
val WSSSE = clusters.computeCost(parsedData)
println("Within Set Sum of Squared Errors = " + WSSSE)
```

### 3.2.4 协同过滤

协同过滤常被应用于推荐系统，这些技术旨在补充用户-商品关联矩阵中所缺失的部分。MLlib当前支持基于模型的协同过滤，其中用户和商品通过一小组隐语义因子进行表达，并且这些因子也用于预测缺失的元素。

案例：导入训练数据集，数据每一行由一个用户、一个商品和相应的评分组成。假设评分是显性的，使用默认的 ALS.train()方法，通过计算预测出的评分的均方差来评估这个推荐模型。

```
import org.apache.spark.mllib.recommendation.ALS
import org.apache.spark.mllib.recommendation.Rating

// 加载和解析数据文件
val data = sc.textFile("mllib/data/als/test.data")
val ratings = data.map(_.split(',') match {
case Array(user, item, rate) => Rating(user.toInt, item.toInt, rate.toDouble)
})

// 设置迭代次数
val numIterations = 20
val model = ALS.train(ratings, 1, 20, 0.01)

// 对推荐模型进行评分
val usersProducts = ratings.map{ case Rating(user, product, rate) => (user, product)}
val predictions = model.predict(usersProducts).map{
case Rating(user, product, rate) => ((user, product), rate)
}
val ratesAndPreds = ratings.map{
case Rating(user, product, rate) => ((user, product), rate)
}.join(predictions)
val MSE = ratesAndPreds.map{
case ((user, product), (r1, r2)) => math.pow((r1- r2), 2)
}.reduce(_ + _)/ratesAndPreds.count
println("Mean Squared Error = " + MSE)
```

## 3.3 MLlib 的实用程序分析

实用程序部分包括数据的验证器、Label 的二元和多元的分析器、多种数据生成器、数据加载器。

```
val data: RDD[LabeledPoint] = MLUtils.loadLabeledData(sc, path)
val data: RDD[LabeledPoint] = MLUtils.loadLibSVMData(sc, path)
val data: RDD[LabeledPoint] =
  MLUtils.loadLibSVMData(sc, path , labelParser)
val data: RDD[LabeledPoint] =
  MLUtils.loadLibSVMData(sc, path, labelParser, numFeatures)
val data: RDD[LabeledPoint] =
  MLUtils.loadLibSVMData(sc, path , labelParser , numFeatures , minSplits)
val data: RDD[(String, String)] = sc.wholeTextFiles(dirPath , minSplits)
```

# 4 参考资料


（1）Spark 官网 mlllib 说明 http://spark.apache.org/docs/1.1.0/mllib-guide.html

（2）《机器学习常见算法分类汇总》 http://www.ctocio.com/hotnews/15919.html