

# 数据挖掘与机器学习

2017年2月16日 13:05

## 各概念阐述

### 数据挖掘



**数据挖掘：**也就是data mining，是一个很宽泛的概念，也是一个新兴学科，旨在**如何从海量数据中挖掘出有用的信息来。**

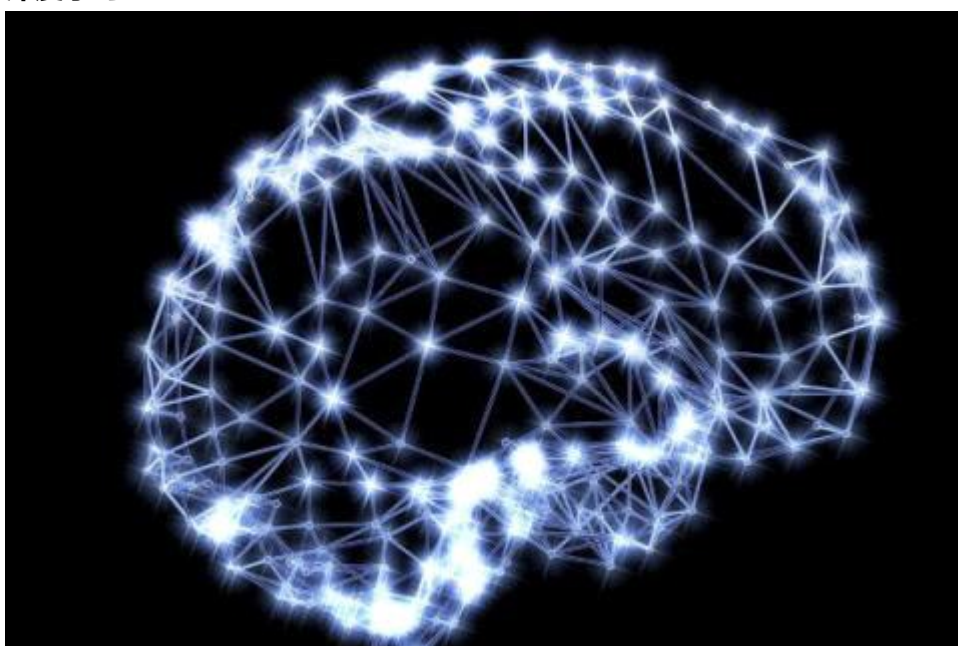
数据挖掘这个工作BI（商业智能）可以做，统计分析可以做，大数据技术可以做，市场营销也可以做，或者用excel分析数据，发现了一些有用的信息，然后这些信息可以指导你的business，这也属于数据挖掘。

### 机器学习



**机器学习：**machine learning，是计算机科学和统计学的交叉学科，基本目标是学习一个 $x \rightarrow y$ 的函数（映射），来做分类、聚类或者回归的工作。之所以经常和数据挖掘合在一起讲是因为现在好多数据挖掘的工作是通过机器学习提供的算法工具实现的，例如广告的ctr预估，PB级别的点击日志在通过典型的机器学习流程可以得到一个预估模型，从而提高互联网广告的点击率和回报率；个性化推荐，还是通过机器学习的一些算法分析平台上的各种购买，浏览和收藏日志，得到一个推荐模型，来预测你喜欢的商品。

## 深度学习



**深度学习：**deep learning，机器学习里面现在比较火的一个topic，本身是神经网络算法的衍

生，在图像，语音等富媒体的分类和识别上取得了非常好的效果，所以各大研究机构和公司都投入了大量的人力做相关的研究和开发。

总结：数据挖掘是个很宽泛的概念，数据挖掘常用方法大多来自于机器学习这门学科，深度学习也是来源于机器学习的算法模型，本质上是原来的神经网络。

**人工智能：**

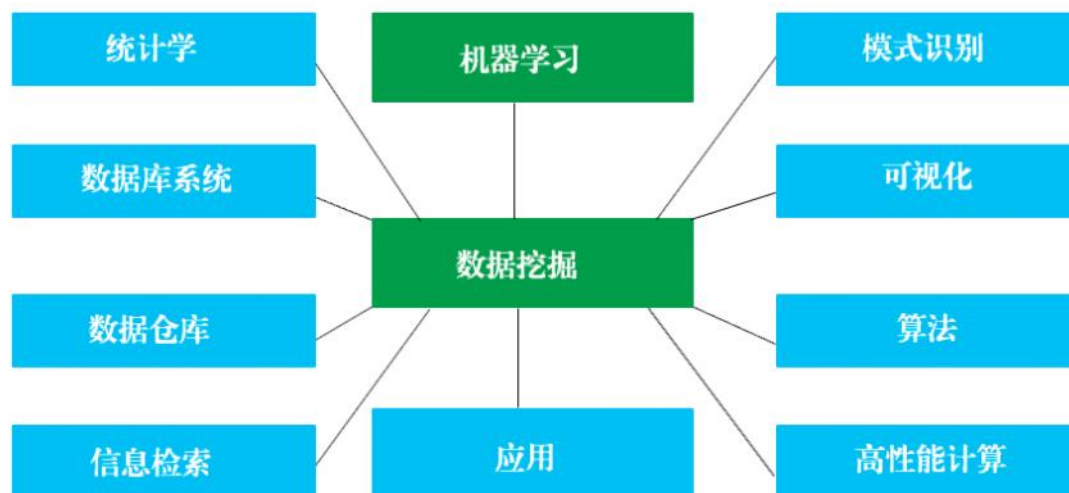


“人工智能”一词最初是在1956年Dartmouth学会上提出的。从那以后，研究者们发展了众多理论和原理，人工智能的概念也随之扩展。人工智能（Artificial Intelligence），英文缩写为AI。它是研究、开发用于模拟、延伸和扩展人的智能的理论、方法、技术及应用系统的一门新的技术科学。人工智能是计算机科学的一个分支，它企图了解智能的实质，并生产出一种新的能以人类智能相似的方式做出反应的智能机器，该领域的研究包括机器人、语言识别、图像识别、自然语言处理和专家系统等。人工智能从诞生以来，理论和技术日益成熟，应用领域也不断扩大，可以设想，未来人工智能带来的科技产品，将会是人类智慧的“容器”。

人工智能是对人的意识、思维的信息过程的模拟。人工智能不是人的智能，但能像人那样思考、也可能超过人的智能。



## 数据挖掘体系





# 机器学习

2017年7月21日 11:54

## 什么是机器学习

机器学习是是一门多领域交叉学科。涉及概率论、统计学、逼近论、凸分析、算法复杂度理论等多门学科。机器学习的算法在数据挖掘里被大量使用。

此外它是人工智能的核心，是使计算机具有智能的根本途径，其应用遍及人工智能的各个领域。

## 机器学习的应用

### 市场分析和管理的

比如：目标市场，客户关系管理（CRM），市场占有率分析，交叉销售，市场分割

1.比如做目标市场分析：

构建一系列的“客户群模型”，这些顾客具有相同特征：兴趣爱好，收入水平，消费习惯，等等。确定顾客的购买模式

CTR估计（广告点击率预测）比如通过逻辑回归来实现。

2.比如做交叉市场分析：

货物销售之间的相互联系和相关性，以及基于这种联系上的预测

### 风险分析和管理，风险预测，客户保持，保险业的改良，质量控制，竞争分析

1.比如做公司分析和风险管理：

财务计划——现金流转分析和预测

资源计划——总结和比较资源和花费

竞争分析——对竞争者和市场趋势的监控

对顾客按等级分组和基于等级的定价过程

对定价策略应用于竞争更激烈的市场中

保险公司对于保险费率的厘定

### 欺骗检测和异常模式的监测（孤立点）

欺诈行为检测和异常模式

1.比如对欺骗行为进行聚类 and 建模，并进行孤立点分析

2.汽车保险：相撞事件的分析

- 3.洗钱：发现可疑的货币交易行为
- 4.医疗保险：职业病人，医生或以及相关数据分析
- 5.电信：电话呼叫欺骗行为，根据呼叫目的地，持续事件，日或周呼叫次数，分析该模型发现与期待标准的偏差
- 6.零售产业：比如根据分析师估计有38%的零售额下降是由于雇员的不诚实行为造成的
- 7.反恐

## 文本挖掘

- 1.新闻组
- 2.电子邮件（垃圾邮件的过滤）可以通过贝叶斯来实现
- 3.文档归类
- 4.评论自动分析
- 5.垃圾信息过滤
- 6.网页自动分类等

## 天文学

例如：JPL实验室和Palomar天文台层借助于数据挖掘工具

## 推荐系统

当当网的图书推荐  
汽车之家的同类汽车推荐  
淘宝的同类商品推荐  
新浪的视频推荐  
百度知道的问题推荐  
社交推荐  
职位推荐

## 智能博弈

棋谱学习

## 频繁模式挖掘

购物篮商品分析，典型案例：啤酒-尿布





## 模式识别

1.语音识别

2.图像识别

指纹、虹膜纹识别

脸像识别

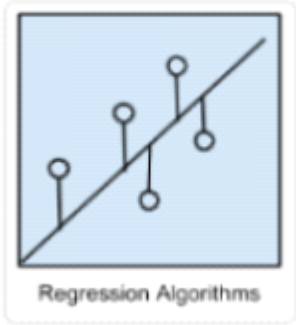
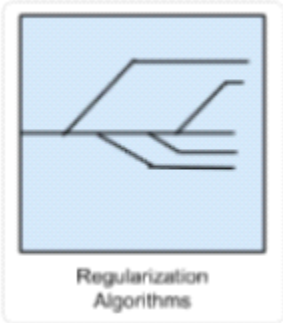
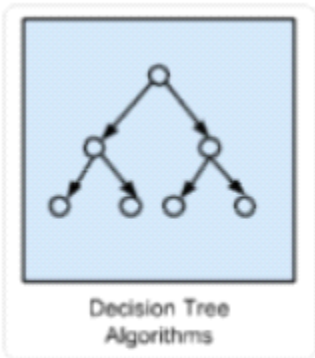
车牌识别

动态图像识别

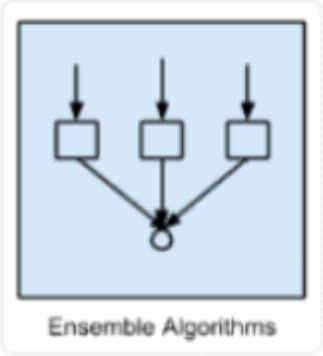
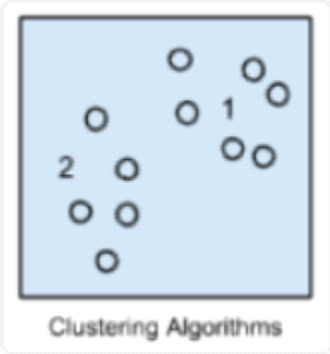
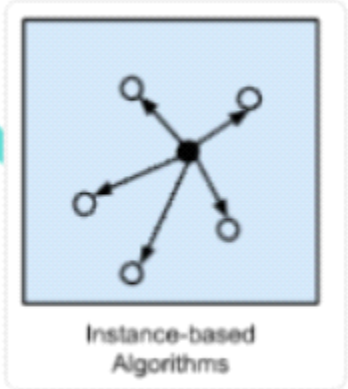
小波分析

# 机器学习算法概述

2017年10月25日 10:30

模型类别	说明	算法
回归模型	<p><b>Regression</b></p>  <p>回归模型研究的问题 因变量（<math>y</math>）和一个或多个自变量（<math>x</math>）的函数关系，可以用于预测，是现代预测学的基础。此外也可以用于分类。以前属于统计学范畴，现在也归到机器学习的范畴</p>	<p><b>1.最小二乘法</b> <b>2.逻辑回归</b> <b>3.逐步回归</b> 4.多元自适应样条法 是利用样条函数的张量积作为基础函数，分为前向过程，剪枝等过程。在处理大量数据，高维数据时表现良好 5.本地权重评估估计法 引入数据窗口概念，一般应用在量化投资，金融分析等领域</p>
正则化模型	 <p>正则化模型的思想是基于一个基础模型（比如最小二乘法）引入惩罚措施，目的是使模型具有更好的泛化能力</p>	<p><b>1.岭回归</b> <b>2.LASSO回归</b> <b>3.弹性网</b> <b>4.最小角回归</b></p>
决策树模型	 <p>决策树建立的模型不是函数式，而是一个决策树，既可以解决分类问题，也可以结局预测问题</p>	<p><b>1.CART树</b> <b>2.ID3算法树</b> <b>3.C4.5算法树</b> 4.卡方自动交叉校验树 5.M5算法树 通过方差诱导思想来实现树的分裂，当方差或误差小于一定阈值时，停止树的分裂</p>



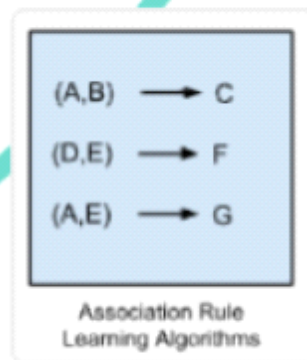
集成模型	<p><b>Ensemble algorithms</b></p>  <p>集成模型的特点将多个弱模型组合在一起。所以可以提高模型的精度和准确度。所以深受欢迎。</p>	<p><b>1.Boosting</b>  <b>2.Bagging ( 装袋算法 )</b>  3.层叠泛化算法  4.梯度提升机算法  5.梯度提升回归树算法  <b>6.随机森林</b></p>
聚类模型	<p><b>Clustering Algorithms</b></p>  <p>聚类算法的特点一般是基于距离度量来对数据做聚类分析，聚类的类别事先是不知道的。</p>	<p><b>1.K-Means</b>  2.最大期望法</p>
基于实例模型 ( 判别模型 )	<p><b>Instance-based Algorithms</b></p>  <p>判别模型模型的特点基于样本数据建立判别函数，通过判别函数判别新样本的类归属问题</p>	<p><b>1.KNN(k-最近邻法)</b>  2.学习向量量化算法  3.自组织映射法  4.本地权重学习法</p>

支持向量机模型	<div data-bbox="405 114 940 163" data-label="Section-Header"> <h2>Support Vector Machines</h2> </div> <div data-bbox="413 219 855 423" data-label="Image"> </div> <div data-bbox="373 463 1110 701" data-label="Text"> <p>支持向量机主要解决分类问题，在数据升维过程中，可能带来维数灾难问题，而SVM引入核函数概念，可以解决高维计算问题，所以性能很好。此外还包含凸优化理论，拉格朗日乘子法等知识。可以应用于手写体识别，语音识别等领域。</p> </div>	SVM 特征空间的映射 核函数 凸优化理论 拉格朗日乘子法
贝叶斯模型	<div data-bbox="402 734 802 781" data-label="Section-Header"> <h2>Bayesian Algorithms</h2> </div> <div data-bbox="434 826 798 1220" data-label="Figure"> </div> <div data-bbox="373 1229 1110 1373" data-label="Text"> <p>这个模型的核心思想是基于贝叶斯公式（定理），是一个种概率模型，可以应用自动推理，文本分析里的垃圾信息过滤</p> </div>	1.朴素贝叶斯分类器 2.贝叶斯信念网络
降维模型	<div data-bbox="397 1408 989 1449" data-label="Section-Header"> <h2>Dimensionality Reduction Algorithms</h2> </div> <div data-bbox="700 1487 979 1812" data-label="Image"> </div> <div data-bbox="373 1854 1110 1948" data-label="Text"> <p>模型的核心思想是做数据的降维，因为数据维数越高，计算代价越大。</p> </div>	1.主成分分析（PCA）

关联规则模型

## Association Rule Learning Algorithms

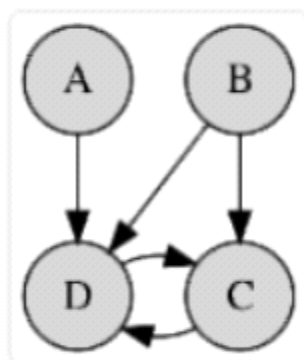
*Apriori algorithm*



模型的核心思想是挖掘数据之间的关联关系，典型的案例：啤酒-尿布 案例

图模型

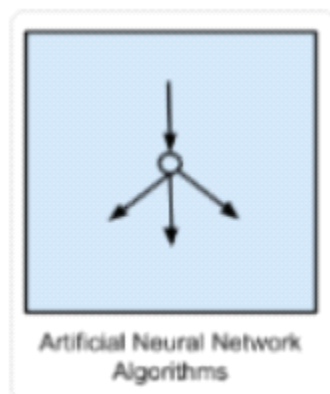
## Graphical Models



核心思想通过图的形式来建模

1. 贝叶斯网络
2. 马尔科夫随机域
3. 链图
4. 祖先图

人工神经网络模型

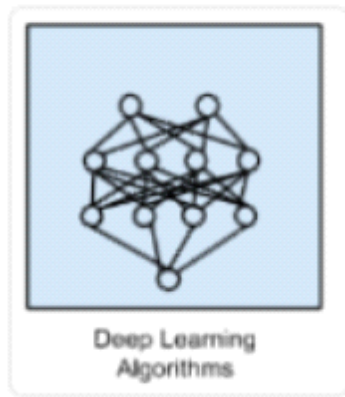


核心思想是模拟人的神经元来建模，含有接收数据+处理数据+传输函数模型

1. BP神经网络

深度学习

## Deep Learning



1. 深玻尔兹曼机
2. 深信念网络

本质是神经网络的延伸，具有一定的模型深度

总体分类，可以分两大来：

1. 监督学习算法
2. 无监督学习算法

# 监督学习和无监督学习

2017年7月21日 18:13

## 监督学习概念介绍

监督学习是指：利用一组**已知类别**的样本调整分类器的参数，使其达到所要求性能的过程，也称为监督训练或有导师训练。

监督学习是从标记的训练数据来推断一个功能的机器学习任务。训练数据包括一套训练示例。在监督学习中，每个实例都是由一个输入对象（通常为矢量）和一个期望的输出值（也称为监督信号）组成。监督学习算法是分析该训练数据，并产生一个推断的功能，其可以用于映射出新的实例。一个最佳的方案将允许该算法来正确地决定那些看不见的实例的类标签。这就要求学习算法是在一种“合理”的方式从一种从训练数据到看不见的情况下形成。

监督学习中在给予计算机学习样本的同时，还告诉计算各个样本所属的类别。若所给的学习样本不带有类别信息,就是无监督学习。任何一种学习都有一定的目的,对于模式识别来说，就是要通过有限数量样本的学习，使分类器在对无限多个模式进行分类时所产生的错误概率最小。

### 常见的监督学习算法

- 1.线性回归
- 2.逻辑回归
- 3.朴素贝叶斯
- 4.KNN(最近邻算法)
- 5.决策树
- 6.支持向量机
- 7.某些可用于分类或预测功能的神经网络模型

## 无监督学习概念介绍

现实生活中常常会有这样的问题：缺乏足够的先验知识，因此难以人工标注类别或进行人工类别标注的成本太高。很自然地，我们希望计算机能代我们完成这些工作，或至少提供一些帮助。根据**类别未知(没有被标记)**的训练样本解决模式识别中的各种问题，称之为无监督学习。比如“鸡尾酒会问题(cocktail party problem)”就是一个无监督学习问题。实际上，可以把无监督学习看做是聚类问题。

### 常见的无监督学习算法

- 1.系统聚类
- 2.K-means

3.K-中值聚类

3.K-众数法

4.某些神经网络模型，比如BP神经网络等

5.受限玻尔兹曼机



# 强化学习

2018年4月9日 8:59

## 强化学习概述

Reinforcement learning (RL) is an area of [machine learning](#) inspired by [behaviourist psychology](#), concerned with how [software agents](#) ought to take [actions](#) in an *environment* so as to maximize some notion of cumulative *reward*. The problem, due to its generality, is studied in many other disciplines, such as [game theory](#), [control theory](#), [operations research](#), [information theory](#), [simulation-based optimization](#), [multi-agent systems](#), [swarm intelligence](#), [statistics](#) and [genetic algorithms](#). In the operations research and control literature, the field where reinforcement learning methods are studied is called *approximate dynamic programming*<sup>[[citation needed](#)]</sup>. The problem has been studied in the [theory of optimal control](#), though most studies are concerned with the existence of optimal solutions and their characterization, and not with the learning or approximation aspects<sup>[[citation needed](#)]</sup>. In [economics](#) and [game theory](#), reinforcement learning may be used to explain how equilibrium may arise under [bounded rationality](#)<sup>[[citation needed](#)]</sup>.

强化学习（RL）也属于机器学习的范畴，灵感来自于行为主义心理学。

强化学习的思想是：引入奖励和惩罚机制，并告知模型如何采取行动，从而最大限度地获取奖励。

强化学习已经在许多其他学科中得到了研究，如博弈论、控制论、运筹学、信息论、基于仿真的优化、多智能体系统、群体智能、统计学和遗传算法等。

在运筹学和控制文献中，研究强化学习方法的领域称为**近似动态规划**。

在经济学和博弈论中，强化学习可以用来解释在有限理性下平衡是如何产生的。

## 案例——机器人浇注



### 监督学习倒水

诸如浇注这样的机器人行为，是通过观看人类的一个第三方示范演示进行学习的，并引入学习行为的奖励函数，比如浇注入目标获得奖励值，未浇注入目标扣减奖励值，以此来训练机器，使其行为满足获得最大奖励值。

## 案例——AI玩超级玛丽



在屏幕中，你可以看到一些简单的2D场景。其中，你可以看一个小人，那就是马里奥，还能看到场景中的所有其他物体。

### 输出

AI有四种方式与游戏进行互动：

- 1) 向左走
- 2) 向右走
- 3) 蹲下
- 4) 跳跃

这些就是AI的输出。AI可以根据输入来决定应该选择哪个输出。

### 奖励

AI一开始可能并不清楚游戏中潜在的奖励，但很快就会得到第一手经验。每个输出的奖励在游

戏中是不同的。如果你只是向左或向右走，那么奖励很低。如果你靠近一个硬币，则奖励稍高一些。如果你跳进一个神秘的盒子，那么奖励会再高一些。但是，如果你被敌人击中，那么奖励是负的，不用说，负数奖励更像是一种惩罚。

## **AI学习如何玩超级马里奥**

作为一名新手，AI从按右箭头开始游戏。这样，你就会得到马里奥向右移动的奖励。然而，随着你继续按右箭头，马里奥最终会碰到一个板栗仔，这样，AI被会被奖励死亡！

马里奥因为碰到板栗仔而奖励死亡。

不用担心，AI可以重新开始。这一次，当程序接收板栗仔向AI走来的输入时，AI可以尝试其他输出以获得不同的奖励。经过几次尝试之后，AI就会意识到，要获得最高奖励的输出就需要跳到板栗仔的头顶，或者直接跳过它。AI现在开始学习如何玩超级马里奥了哦——

当马里奥跳过板栗仔的时候，就会奖励继续活着

这个过程展示了AI程序一开始是一块干净的白板，不知道自己应该要做什么。然后，通过奖励函数（导师或监督），使AI不断地训练，从而获取更高的奖励以及避免惩罚，最后得到理想的模型。

所以，只要给予足够的时间，机器学习算法将能够训练自己以成功完成给定的任务。尽管这可能需要花费很长时间，但最重要的还是要为你的程序提供高质量的训练数据，以获得更准确的结果。

# Spark MLlib介绍

2018年2月13日 10:35

## 概述

**MLlib** is Apache Spark's scalable machine learning library.

## Ease of Use

Usable in Java, Scala, Python, and R.

MLlib fits into [Spark](#)'s APIs and interoperates with [NumPy](#) in Python (as of Spark 0.9) and R libraries (as of Spark 1.5). You can use any Hadoop data source (e.g. HDFS, HBase, or local files), making it easy to plug into Hadoop workflows.

```
data = spark.read.format("libsvm")\
    .load("hdfs://...")
```

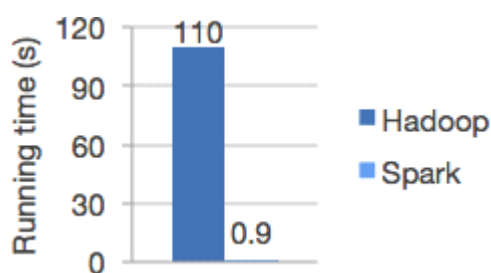
```
model = KMeans(k=10).fit(data)
```

Calling MLlib in Python

## Performance

High-quality algorithms, 100x faster than MapReduce.

Spark excels at iterative computation, enabling MLlib to run fast. At the same time, we care about algorithmic performance: MLlib contains high-quality algorithms that leverage iteration, and can yield better results than the one-pass approximations sometimes used on MapReduce.



Logistic regression in Hadoop and Spark

# Easy to Deploy

Runs on existing Hadoop clusters and data.

If you have a Hadoop 2 cluster, you can run Spark and MLlib without any pre-installation. Otherwise, Spark is easy to run [standalone](#) or on [EC2](#) or [Mesos](#). You can read from [HDFS](#), [HBase](#), or any Hadoop data source.

## Algorithms

MLlib contains many algorithms and utilities.

ML algorithms include:

- Classification: logistic regression, naive Bayes,...
- Regression: generalized linear regression, survival regression,...
- Decision trees, random forests, and gradient-boosted trees
- Recommendation: alternating least squares (ALS)
- Clustering: K-means, Gaussian mixtures (GMMs),...
- Topic modeling: latent Dirichlet allocation (LDA)
- Frequent itemsets, association rules, and sequential pattern mining

ML workflow utilities include:

- Feature transformations: standardization, normalization, hashing,...
- ML Pipeline construction
- Model evaluation and hyper-parameter tuning
- ML persistence: saving and loading models and Pipelines

Other utilities include:

- Distributed linear algebra: SVD, PCA,...
- Statistics: summary statistics, hypothesis testing,...

MLlib是一个构建在Spark上的、专门针对大数据处理的并发式高速机器学习库，其特点是采用较为先进的迭代式、内存存储的分析计算，使得数据的计算处理速度大大高于普通的数据处理引擎。

MLlib机器学习库还在不停地更新中，Apache的相关研究人员仍在不停地为其中添加更多的机器学习算法。目前MLlib中已经有通用的学习算法和工具类，包括统计、分类、回归、聚类、降维等。

MLlib采用Scala语言编写，Scala语言是运行在JVM上的一种函数式编程语言，特点就是可移植性强，“一次编写，到处运行”是其最重要的特点。借助于RDD数据统一输入格式，让用户可以在不同的IDE上编写数据处理程序，通过本地化测试后可以在略微修改运行参数后直接在集群上运行。对结果的获取更为可视化和直观，不会因为运行系统底层的不同而造成结果的差异与改变。



# MLlib基本数据模型

2018年2月16日 11:21

## 概述

RDD是MLlib专用的数据格式，它参考了Scala函数式编程思想，并大胆引入统计分析概念，将存储数据转化成向量和矩阵的形式进行存储和计算，这样将数据定量化表示，能更准确地整理和分析结果。

## 多种数据类型

MLlib先天就支持较多的数据格式，从最基本的Spark数据集RDD到部署在集群中的**向量和矩阵**。同样，MLlib还支持部署在本地计算机中的本地化格式。

下表给出了MLlib支持的数据类型。

类型名称	释义
Local vector	本地向量集。主要向Spark提供一组可进行操作的数据集合
Labeled point	向量标签。让用户能够分类不同的数据集合
Local matrix	本地矩阵。将数据结合以矩阵形式存储在本地计算机中
Distributed matrix	分布式矩阵。将矩阵集合以矩阵形式存储在分布式计算机中

以上就是MLlib支持的数据类型，其中分布式矩阵根据不同的作用和应用场景，又分为四种不同的类型

## 一、本地向量

MLlib使用的本地化存储类型是向量，这里的向量主要由两类构成：**稀疏型数据集**（spares）和**密集型数据集**（dense）。例如一个向量数据(9,5,2,7)，按密集型数据格式可以被设定成(9,5,2,7)进行存储，数据集被作为一个集合的形式整体存储。而对于稀疏型数据，可以按向量的大小存储为(4, Array(0,1,2,3), Array(9,5,2,7))。

### 案例一：

```
import org.apache.spark.{SparkConf,SparkContext}

def main(args:Array[String]):Unit={
  //--建立密集型向量
  //--dense可以将其理解为MLlib专用的一种集合形式，它与Array类似
  val vd=Vectors.dense(2,0,6)//
  println(vd)

  //①参:size。sparse方法是将给定的数据Array数据(9,5,2,7)分解成指定的size个部分进行处理，本例中是7个
  //③参:输入数据。本例中是Array ( 9,5,2,7)
  //②参:输入数据对应的下标，要求递增，并且最大值要小于等于size

  val vs=Vectors.sparse(7,Array(0,1,3,6),Array(9,5,2,7))
  println(vs(6))

}

}
```

## 二、向量标签的使用

### 向量标签的使用

向量标签用于对MLlib中机器学习算法的不同值做标记。例如分类问题中，可以将不同的数据集分成若干份，以整型数0、1、2.....进行标记，即程序的编写者可以根据自己的需要对数据进行标记。

### 案例一：

```
import org.apache.spark.mllib.linalg.{Vector,Vectors}
import org.apache.spark.mllib.regression.LabeledPoint
object Demo03{

defmain(args:Array[String]):Unit={
```

```

val vd=Vectors.dense(2,0,6)
//--对密集向量建立标记点
val pos=LabeledPoint(1,vd)
//--打印标记点内容数据
println(pos.features)
//--打印既定标记
println(pos.label)
}
}

```

Features用于显示打印标记点所代表的`数据内容`，而Label用于显示`标记数`。

## 案例二：

### 文件数据：

```

2.3 3.2 1
1.6 2.3 2
5.1 2.1 1
6.2 7.2 1
3.2 1.2 2
9.2 2.1 1

```

### 代码：

```

object Driver {

  def main(args: Array[String]): Unit = {

    val conf=new SparkConf().setMaster("local").setAppName("vd")
    val sc=new SparkContext(conf)

    val data=sc.textFile("d://ml/labeled.txt")

    val parseData=data.map { x =>
      val parts=x.split(" ")

      LabeledPoint(parts(2).toDouble,Vectors.dense(parts(0).toDouble,parts(1).toDouble))
    }

    //查看标签数据
    parseData.foreach { x => println(x) }
  }
}

```

//只查看特征数据 ( 自变量数据)

```
parseData.foreach { x => println(x.features) }
```

//只查看标签数据(因变量数据)

```
parseData.foreach { x => println(x.label) }
```

```
}
```

```
}
```

### 案例三：

**数据集：**

1 1:2 2:3 3:4

2 1:1 2:2 3:3

1 1:1 2:3 3:3

1 1:3 2:1 3:3

说明：label index1:value1 index2:value2 ...

label是此数据集中每一行给定的标签，而后的index是标签所标注的这一行中的不同的索引值，而紧跟在各自index后的value是不同索引所形成的数据值。

**代码：**

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

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

```
import org.apache.spark._
```

```
import org.apache.spark.mllib.util.MLUtils
```

```
object Demo04{
```

```
def main(args:Array[String]):Unit={
```

```
val conf=new SparkConf().setMaster("local").setAppName("TestLabel")
```

```
val sc=new SparkContext(conf)
```

```
val data=MLUtils.loadLibSVMFile(sc,"d://data.txt")
```

```
data.foreach(println)
```

```
}  
}
```

**最后的结果：**

```
(1.0,(3,[0,1,2],[2.0,3.0,3.0]))  
(2.0,(3,[0,1,2],[5.0,8.0,9.0]))  
(1.0,(3,[0,1,2],[7.0,6.0,7.0]))  
(1.0,(3,[0,1,2],[3.0,2.0,1.0]))
```

根据打印结果，可以很明显地看出，loadLibSVMFile方法将数据分解成一个稀疏向量进行下一步的操作。

## 提示

MLUtils.loadLibSVMFile数据集标记的index是从1开始，读者可以试试从0

## 三、本地矩阵的使用

大数据运算中，为了更好地提升计算效率，可以更多地使用矩阵运算进行数据处理。部署在单机中的本地矩阵就是一个很好的存储方法。

举一个简单的例子，例如一个数组Array（1,2,3,4,5,6），将其分为2行3列的矩阵，可用如下程序4-4进行处理。

### 案例一：

```
import org.apache.spark.mllib.linalg.{Matrix,Matrices}  
  
object Demo05{  
  
  def main(args:Array[String]):Unit={  
    val mx=Matrices.dense(2,3,Array(1,2,3,4,5,6))  
    println(mx)  
  }  
}
```

打印结果如下：

```
1.0  3.0  5.0  
2.0  4.0  6.0
```

从结果来看，数组Array（1,2,3,4,5,6）被重组成一个新的2行3列的矩阵。Matrices.dense方法是矩阵重组的调用方法，第一个参数是新矩阵行数，第二个参数是新矩阵的列数，第三个参数为传入的数据值。

## 分布式矩阵的使用

一般来说，采用分布式矩阵进行存储的情况都是数据量非常大的情况

### 1. 行矩阵

行矩阵是最基本的一种矩阵类型。行矩阵是以行作为基本方向的矩阵存储格式，列的作用相对较小。可以将其理解为行矩阵是一个巨大的特征向量的集合。每一行就是一个具有相同格式的向量数据，且每一行的向量内容都可以单独取出来进行操作。

#### 案例一：

**数据：**

1 2 3

4 5 6

**代码：**

```
import org.apache.spark.mllib.linalg.distributed.RowMatrix
object Demo06{

def main(args: Array[String]): Unit = {
    val conf=new SparkConf().setMaster("local").setAppName("vd")
    val sc=new SparkContext(conf)

    val data=sc.textFile("d://ml/rowMatrix.txt")
    //转成RDD，并且行数据的格式是Vector
    val parseData=data.map(_._split(" ").map(_._toDouble)).map(array=>Vectors.dense(array))
    //读入行矩阵
    val rm=new RowMatrix(parseData)
    //打印行数
    println(rm.numRows)
    //打印列数
    println(rm.numCols())
}
```



```
}  
}
```

## 2. 带有行索引的行矩阵

单纯的行矩阵对其内容无法进行直接显示，当然可以通过调用其方法显示内部数据内容。有时候，为了方便在系统调试的过程中对行矩阵的内容进行观察和显示，MLlib提供了另外一个矩阵形式，即带有行索引的行矩阵。

### 案例一：

**数据：**

```
1 2 3  
4 5 6
```

**代码：**

```
import org.apache.spark._  
import org.apache.spark.mllib.linalg.{Vector, Vectors}  
import org.apache.spark.mllib.linalg.distributed.{IndexedRowMatrix, RowMatrix}  
import org.apache.spark.mllib.linalg.distributed.{IndexedRow, RowMatrix, IndexedRowMatrix}  
object Demo06{  
  
  def main(args: Array[String]): Unit = {  
    val conf=new SparkConf().setMaster("local").setAppName("vd")  
    val sc=new SparkContext(conf)  
  
    val data=sc.textFile("d://ml/rowMatrix.txt")  
  
    var index=0  
  
    //转成RDD，并且行数据的格式是Vector  
    val parseData=data.map(_._split(" ").map(_._toDouble))  
    .map(array=>Vectors.dense(array)).map { vector =>  
  
      index+=1  
      new IndexedRow(index,vector)  
    }  
  }  
}
```

```
val rm=new IndexedRowMatrix(parseData)
```

```
//打印矩阵所有的行数据
```

```
rm.rows.foreach {println}
```

```
//打印指定索引的行数据
```

```
rm.rows.filter { x => x.index==1 }.foreach { x => print(x) }
```

```
}
```

**打印结果如下：**

```
IndexedRow(3,[1.0,2.0,3.0])
```

```
IndexedRow(3,[4.0,5.0,6.0])
```

# MLlib统计量基础

2018年2月16日 15:09

## 概述

数理统计中，基本统计量包括数据的平均值、方差，这是一组求数据统计量的基本内容。在MLlib中，统计量的计算主要用到Statistics类库。

从下表可以看到，Statistics类中不同的方法代表不同的统计量的求法，下面根据不同的内容分别加以介绍。

类型名称	释义
colStats	以列为基础计算统计量的基本数据
corr	对两个数据集进行相关系数计算，根据参量的不同，返回值格式有差异

## 计算基本统计量

这里主要调用colStats方法，接受的是RDD类型数据。

这里需要注意的是，其工作和计算是以列为基础进行计算，调用不同的方法可以获得不同的统计量值，其方法内容如下表所示。

方法名称	释义
count	行内数据个数
Max	最大值
Min	最小值
Mean	均值
normL1	曼哈顿距离
normL2	欧几里得距离
numNonzeros	不包含0值的个数
variance	标准差

### 案例一 计算基本统计量

数据：

- 1
- 2
- 3

4

5

### 代码：

```
import org.apache.spark._
import org.apache.spark.mllib.linalg.{Vector, Vectors}
import org.apache.spark.mllib
import org.apache.spark.mllib.stat.Statistics
object Demo07{

def main(args:Array[String]):Unit={

val conf=new SparkConf().setMaster("local").setAppName("testSummary")
val sc=new SparkContext(conf)
val rdd=sc.textFile("d://testSummary.txt").map(_._split(" ").map(_._toDouble)).map(array=>
Vectors.dense(array))
val summary=Statistics.colStats(rdd)
println(summary.mean)//计算均值
println(summary.variance)//计算方差
println(summary.max)//最大值
println(summary.min)//最小值
println(summary.count)//行数

}

}
```

### 案例二 距离计算

除了一些基本统计量的计算，此方法中还包括两种距离的计算，分别是normL1和normL2，代表着曼哈顿距离和欧几里得距离。这两种距离主要是用以表达数据集内部数据长度的常用算法。

### 代码：

```
.....
println(summary.normL1)//计算曼哈顿距离 [15.0]
println(summary.normL2)//计算欧式距离 [7.416198487095663]
```

## 曼哈顿距离：

曼哈顿距离用来表明两个点在标准坐标系上的绝对轴距总和。其公式如下：

$$x = x_1 + x_2 + x_3 + \dots + x_n$$

## 二、计算相关系数

相关系数是一种用来反映变量之间相关关系密切程度的统计指标，在现实中一般用于对两组数据的拟合和相似程度进行定量化分析。常用的一般是皮尔逊相关系数，MLlib中默认的相关系数求法也是使用皮尔逊相关系数法。

### 案例一 计算皮尔逊相关系数

#### 数据：

testCorrX.txt

1 2 3 4 5

testCorrY.txt

2 4 7 9 10

#### 代码：

```
import org.apache.spark.{SparkConf,SparkContext}

object Demo08{

  def main(args:Array[String]):Unit={
    val conf=new SparkConf().setMaster("local").setAppName("testCorrect")
    val sc=new SparkContext(conf)
    val rddX=sc.textFile("d://testCorrX.txt").flatMap(_.split(" ").map(_.toDouble))
    val rddY=sc.textFile("d://testCorrY.txt").flatMap(_.split(" ").map(_.toDouble))
    val correlation:Double=Statistics.corr(rddX,rddY)//计算两组数据之间的相关系数
    println(correlation)//0.9877569118027772

  }
}
```

# 最小二乘法

2018年2月18日 12:22

## 介绍

最小二乘法（又称最小平方法）是一种数学优化技术。**它通过最小化误差的平方和寻找数据的最佳函数匹配。**利用最小二乘法可以简便地求得未知的数据，并使得这些求得的数据与实际数据之间误差的平方和为最小。最小二乘法还可用于曲线拟合。其他一些优化问题也可通过最小化能量或最大化熵用最小二乘法来表达。

## 背景故事

最小二乘法(Least Squares Method,简记为LSE)是一个比较古老的方法，源于天文学和测地学上的应用需要。在早期数理统计方法的发展中，这两门科学起了很大的作用。丹麦统计学家霍尔把它们称为“数理统计学的母亲”。此后近三百年来，它广泛应用于科学实验与工程技术中。美国统计史学家斯蒂格勒(S. M. Stigler)指出, 最小二乘方法是19世纪数理统计学的压倒一切的主题。1815年时,这方法已成为法国、意大利和普鲁士在天文和测地学中的标准工具,到1825年时已在英国普遍使用。

## 朱赛普·皮亚齐



追溯到1801年，意大利天文学家朱赛普·皮亚齐发现了第一颗小行星谷神星。经过40天的跟踪观测后，由于谷神星运行至太阳背后，使得皮亚齐失去了谷神星的位置。随后全世界的科学家利用皮亚齐的观测数据开始寻找谷神星，但是根据大多数人计算的结果来寻找谷神星都没有结果。



**高斯****勒让德**

时年24岁的高斯也计算了谷神星的轨道。奥地利天文学家海因里希·奥尔伯斯根据高斯计算出来的轨道重新发现了谷神星。高斯于其1809年的著作《关于绕日行星运动的理论》中。在此书中声称他自1799年以来就使用最小二乘方法,由此爆发了一场与勒让德的优先权之争。

近代学者经过对原始文献的研究,认为两人可能是独立发明了这个方法,但首先见于书面形式的,以勒让德为早。然而,现今教科书和著作中,多把这个发明权归功于高斯。其原因,除了高斯有更大的名气外,主要可能是因为其正态误差理论对这个方法的重要意义。勒让德在其著作中,对最小二乘方法的优点有所阐述。然而,缺少误差分析。我们不知道,使用这个方法引起的误差如何,就需建立一种误差分析理论。高斯于1823年在误差 $e_1, \dots, e_n$ 独立同分布的假定下,证明了最小二乘方法的一个最优性质:在所有无偏的线性估计类中,最小二乘方法是其中方差最小的!

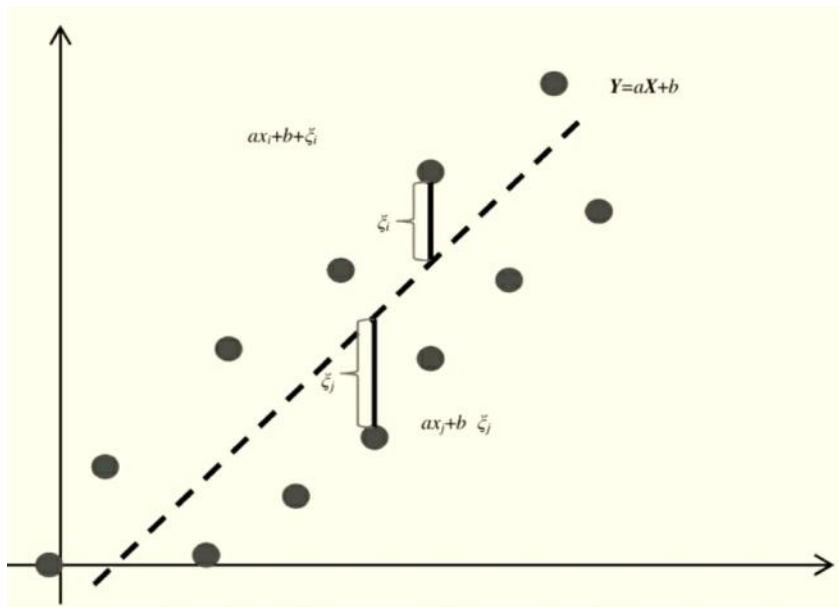
现行的最小二乘法是勒让德(A. M. Legendre)于1805年在其著作《计算彗星轨道的新方法》中提出的。它的主要思想就是选择未知参数,使得理论值与观测值之差的平方和达到最小:

$$H = \sum_{i=0}^m (y - y_i)^2$$

[原理及推导过程](#)

下面我们来看一下最简单的线性情况。

如下图所示，对于某个数据集 $(x_i, y_i)$  ( $i=0,1,\dots,n$ )，我们需要找到一条趋势线（图中的虚线），能够表达出数据集 $(x_i, y_i)$ 这些点所指向的方向。



我们先用一个直线函数表示这条趋势线：

$$Y=aX+b$$

数据集的点一定位于这条趋势线的上下两侧，或者与趋势线重合。我们把某个样本点 $x_i$ 到这条趋势线的垂直距离定义为残差 $\xi_i$ ，那么过这一点与趋势线平行的样本函数为 $y_i=ax_i+b+\xi_i$ 。如果这个样本点位于趋势线的上侧，在残差 $\xi_i>0$ ，反之则 $\xi_i<0$ ，如果样本点位于趋势线上则 $\xi_i=0$ 。

现在，我们求解这条趋势线。因为是线性函数，所以也就是求解 $a$ 、 $b$ 这两个值。

下面我们将带有残差的直线函数修改为下面的形式：

$$\xi_i=y_i-ax_i-b$$

因为残差 $\xi_i$ 有正负号的问题，所以我们统一用平方和来计算，即残差平方和：

$$Q = \sum_{i=1}^n \xi_i^2 = \sum_{i=1}^n (y_i - ax_i - b)^2$$

很明显这个二次函数是一个凸函数（单峰函数），我们接下来对该函数求极值，即它的一阶导数等于0。

$$\frac{\partial Q}{\partial a} = 2 \sum_{i=1}^n (-x_i)(y_i - ax_i - b) = -2 \sum_{i=1}^n (x_i y_i - ax_i^2 - bx_i) = 0$$

$$\frac{\partial Q}{\partial b} = 2 \sum_{i=1}^n (-1)(y_i - ax_i - b) = -2(\sum_{i=1}^n y_i - \sum_{i=1}^n ax_i - nb) = 0$$

接下来，将两个方程联立，我们令：

$$n\bar{x} = \sum_{i=1}^n x_i; n\bar{y} = \sum_{i=1}^n y_i$$

解得a、b的值为：

$$a = \frac{\sum_{i=1}^n x_i y_i - n\bar{x}\bar{y}}{\sum_{i=1}^n x_i^2 - n\bar{x}^2}$$

$$b = \bar{y} - a\bar{x}$$

上式中，x与y都已知，所以可求得a,b。

# 案例—预测商品需求量

2018年3月8日 10:36

## 说明

某种商品的需求量（ $y$ ，吨）、价格（ $x_1$ ，元/千克）和消费者收入（ $x_2$ ，元）观测值如下表所示。

$$y = \theta_1 X_1 + \theta_2 X_2 + \theta_0$$

y	x1	x2
100	5	1000
75	7	600
80	6	1200
70	6	500
50	8	30
65	7	400
90	5	1300
100	4	1100
110	3	1300
60	9	300
	10	200

1.均值插补法

2.回归插补法

异常值的检测和处理

1.结合业务做异常值检测，比如给定年龄属相，定出合理范围。然后进行检测

2.通过画图，画箱线图来检测

3.通过一些统计量来统计，比如最大值，最小值等

1.处理方法，最好不好将异常值直接丢企，可以进行转储，以便于异常值的分析处理

2.转储完之后，将异常值变为空，然后插补

为了能够通过Mllib建模，我们首先需要对数据格式进行一定的处理，比如如下所示：

100|5 1000

75|7 600

80|6 1200

70|6 500

50|8 30

65|7 400

90|5 1300

100|4 1100

110|3 1300

60|9 300

#### 代码：

```
import org.apache.spark.SparkConf
import org.apache.spark.SparkContext
import org.apache.spark.mllib.regression.LabeledPoint
import org.apache.spark.mllib.linalg.Vectors
import org.apache.spark.ml.regression.LinearRegressionModel.LinearRegressionModelReader
import org.apache.spark.ml.regression.LinearRegressionSummary
import org.apache.spark.ml.regression.LinearRegressionTrainingSummary
import org.apache.spark.mllib.regression.LinearRegressionWithSGD
import org.apache.spark.mllib.regression.LinearRegressionModel
import org.apache.spark.ml.regression.LinearRegressionModel
import org.apache.spark.sql.SQLContext
import org.apache.spark.ml.feature.VectorAssembler
import org.apache.spark.ml.regression.LinearRegression
```

```
object Driver {
```

```
  def main(args: Array[String]): Unit = {
```

```
    val conf=new SparkConf().setMaster("local").setAppName("lr")
```

```

val sc=new SparkContext(conf)

val sqc=new SQLContext(sc)

val data=sc.textFile("d://ml/lritem.txt")

//--将数据转成 tuple格式,是为了后期转成SparkSql的DataFrame
val parseData=data.map { x =>
    val parts=x.split("\\|")
    val features=parts(1).split(" ")
    (parts(0).toDouble,features(0).toDouble,features(1).toDouble)
}

//--转成DF
val df=sqc.createDataFrame(parseData)

//--定义各列字段名字
val dfData=df.toDF("Y","X1","X2")
//--定义features 字段名
val featureColum=Array("X1","X2")

//--定义VectorAssemmbler格式, 这是MLlib线性回归模型要求的数据形式
//--此外, 要指定特征向量是哪几列, 以及定义特征向量对应的别称, 一般用:features"
val assembler=new VectorAssembler().setInputCols(featureColum).setOutputCol("features")

//--将DF转成VectorAssemmbler格式, 后面建模会用到
val vecDF=assembler.transform(dfData)

//--建模, 设定 自变量(features)和 因变量(Y)。
//--setFitIntercept(true) 表示是否需要计算截距项
//--fit(数据)是训练生成模型
val model=new LinearRegression().setFeaturesCol("features").setLabelCol("Y")
    .setFitIntercept(true).fit(vecDF)

//--transform(数据) 预测数据, 测试的数据的格式必须为:VectorAssemmbler
val predictionTable=model.transform(vecDF)

```

//--查询结果

```
val result=predictionTable.selectExpr("features","Y","round(prediction,1)as prediction")
```

```
result.foreach { x => println(x) }
```

//--以下为我们自定义的一组测试数据，进行测试

```
val
```

```
testData=assembler.transform(sqc.createDataFrame(sc.makeRDD(List((0.0,8,35))))).toDF("Y","X1","X2"))
```

```
val testPrediction=model.transform(testData).selectExpr("features","Y","round(prediction,1)as prediction")
```

```
testPrediction.foreach { x => println(x) }
```

```
println(model.summary.r2)
```

```
}
```

```
}
```

# 案例—预测谋杀率

2018年3月8日 10:37

## 说明

下表统计了某个国家某一时期的谋杀率（murder）和其他因素之间的影响，这些因素包括：Population人口，Income(收入)，Illiteracy（文盲程度），LifeExp（生活经验），HSGrad(文献中未做具体解释)，Frost(温度在冰点以下的平均天数)，Area(地区)

请尝试用MLlib，利用样本集（lrmurder-sample）建立预测模型，并利用模型对测试集（lrmurder-test）进行验证

Population	Income	Illiteracy	LifeExp	Murder	HSGrad	Frost	Area
3615	3624	2.1	69.05	15.1	41.3	20	50708
365	6315	1.5	69.31	11.3	66.7	152	566432
2212	4530	1.8	70.55	7.8	58.1	15	113417
2110	3378	1.9	70.66	10.1	39.9	65	51945
21198	5114	1.1	71.71	10.3	62.6	20	156361
2541	4884	0.7	72.06	6.8	63.9	166	103766
3100	5348	1.1	72.48	3.1	56	139	4862
579	4809	0.9	70.06	6.2	54.6	103	1982
8277	4815	1.3	70.66	10.7	52.6	11	54090
4931	4091	2	68.54	13.9	40.6	60	58073
868	4963	1.9	73.6	6.2	61.9	0	6425

## 📖 样本集：

3615, 3624, 2.1, 69.05, 15.1, 41.3, 20, 50708  
365, 6315, 1.5, 69.31, 11.3, 66.7, 152, 566432  
2212, 4530, 1.8, 70.55, 7.8, 58.1, 15, 113417  
2110, 3378, 1.9, 70.66, 10.1, 39.9, 65, 51945  
21198, 5114, 1.1, 71.71, 10.3, 62.6, 20, 156361  
2541, 4884, 0.7, 72.06, 6.8, 63.9, 166, 103766  
3100, 5348, 1.1, 72.48, 3.1, 56, 139, 4862  
579, 4809, 0.9, 70.06, 6.2, 54.6, 103, 1982  
8277, 4815, 1.3, 70.66, 10.7, 52.6, 11, 54090  
4931, 4091, 2, 68.54, 13.9, 40.6, 60, 58073  
868, 4963, 1.9, 73.6, 6.2, 61.9, 0, 6425



813, 4119, 0.6, 71.87, 5.3, 59.5, 126, 82677  
11197, 5107, 0.9, 70.14, 10.3, 52.6, 127, 55748  
5313, 4458, 0.7, 70.88, 7.1, 52.9, 122, 36097  
2861, 4628, 0.5, 72.56, 2.3, 59, 140, 55941  
2280, 4669, 0.6, 72.58, 4.5, 59.9, 114, 81787  
3387, 3712, 1.6, 70.1, 10.6, 38.5, 95, 39650  
3806, 3545, 2.8, 68.76, 13.2, 42.2, 12, 44930  
1058, 3694, 0.7, 70.39, 2.7, 54.7, 161, 30920  
4122, 5299, 0.9, 70.22, 8.5, 52.3, 101, 9891  
5814, 4755, 1.1, 71.83, 3.3, 58.5, 103, 7826  
9111, 4751, 0.9, 70.63, 11.1, 52.8, 125, 56817  
3921, 4675, 0.6, 72.96, 2.3, 57.6, 160, 79289  
2341, 3098, 2.4, 68.09, 12.5, 41, 50, 47296  
4767, 4254, 0.8, 70.69, 9.3, 48.8, 108, 68995  
746, 4347, 0.6, 70.56, 5, 59.2, 155, 145587  
1544, 4508, 0.6, 72.6, 2.9, 59.3, 139, 76483  
590, 5149, 0.5, 69.03, 11.5, 65.2, 188, 109889  
812, 4281, 0.7, 71.23, 3.3, 57.6, 174, 9027  
7333, 5237, 1.1, 70.93, 5.2, 52.5, 115, 7521  
1144, 3601, 2.2, 70.32, 9.7, 55.2, 120, 121412  
18076, 4903, 1.4, 70.55, 10.9, 52.7, 82, 47831  
5441, 3875, 1.8, 69.21, 11.1, 38.5, 80, 48798  
637, 5087, 0.8, 72.78, 1.4, 50.3, 186, 69273  
10735, 4561, 0.8, 70.82, 7.4, 53.2, 124, 40975  
2715, 3983, 1.1, 71.42, 6.4, 51.6, 82, 68782  
2284, 4660, 0.6, 72.13, 4.2, 60, 44, 96184  
11860, 4449, 1, 70.43, 6.1, 50.2, 126, 44966  
931, 4558, 1.3, 71.9, 2.4, 46.4, 127, 1049  
2816, 3635, 2.3, 67.96, 11.6, 37.8, 65, 30225  
681, 4167, 0.5, 72.08, 1.7, 53.3, 172, 75955  
4173, 3821, 1.7, 70.11, 11, 41.8, 70, 41328  
12237, 4188, 2.2, 70.9, 12.2, 47.4, 35, 262134  
1203, 4022, 0.6, 72.9, 4.5, 67.3, 137, 82096  
472, 3907, 0.6, 71.64, 5.5, 57.1, 168, 9267  
4981, 4701, 1.4, 70.08, 9.5, 47.8, 85, 39780  
3559, 4864, 0.6, 71.72, 4.3, 63.5, 32, 66570  
1799, 3617, 1.4, 69.48, 6.7, 41.6, 100, 24070  
4589, 4468, 0.7, 72.48, 3, 54.5, 149, 54464

376, 4566, 0.6, 70.29, 6.9, 62.9, 173, 97203

## 测试集：

615, 3624, 2.1, 69.05, 0, 41.3, 20, 50708  
65, 6315, 1.5, 69.31, 0, 66.7, 152, 566432  
212, 4530, 1.8, 70.55, 0, 58.1, 15, 113417  
110, 3378, 1.9, 70.66, 0, 39.9, 65, 51945  
1198, 5114, 1.1, 71.71, 0, 62.6, 20, 156361  
541, 4884, 0.7, 72.06, 0, 63.9, 166, 103766  
100, 5348, 1.1, 72.48, 0, 56, 139, 4862  
79, 4809, 0.9, 70.06, 0, 54.6, 103, 1982  
277, 4815, 1.3, 70.66, 0, 52.6, 11, 54090  
931, 4091, 2, 68.54, 0, 40.6, 60, 58073  
868, 4963, 1.9, 73.6, 6.2, 61.9, 0, 6425  
813, 4119, 0.6, 71.87, 5.3, 59.5, 126, 82677  
11197, 5107, 0.9, 70.14, 10.3, 52.6, 127, 55748  
5313, 4458, 0.7, 70.88, 7.1, 52.9, 122, 36097  
2861, 4628, 0.5, 72.56, 2.3, 59, 140, 55941  
2280, 4669, 0.6, 72.58, 4.5, 59.9, 114, 81787  
3387, 3712, 1.6, 70.1, 10.6, 38.5, 95, 39650  
3806, 3545, 2.8, 68.76, 13.2, 42.2, 12, 44930  
1058, 3694, 0.7, 70.39, 2.7, 54.7, 161, 30920  
4122, 5299, 0.9, 70.22, 8.5, 52.3, 101, 9891  
5814, 4755, 1.1, 71.83, 3.3, 58.5, 103, 7826  
9111, 4751, 0.9, 70.63, 11.1, 52.8, 125, 56817  
3921, 4675, 0.6, 72.96, 2.3, 57.6, 160, 79289  
2341, 3098, 2.4, 68.09, 12.5, 41, 50, 47296  
4767, 4254, 0.8, 70.69, 9.3, 48.8, 108, 68995  
746, 4347, 0.6, 70.56, 5, 59.2, 155, 145587  
1544, 4508, 0.6, 72.6, 2.9, 59.3, 139, 76483  
590, 5149, 0.5, 69.03, 11.5, 65.2, 188, 109889  
812, 4281, 0.7, 71.23, 3.3, 57.6, 174, 9027  
7333, 5237, 1.1, 70.93, 5.2, 52.5, 115, 7521  
1144, 3601, 2.2, 70.32, 9.7, 55.2, 120, 121412  
18076, 4903, 1.4, 70.55, 10.9, 52.7, 82, 47831  
5441, 3875, 1.8, 69.21, 11.1, 38.5, 80, 48798  
637, 5087, 0.8, 72.78, 1.4, 50.3, 186, 69273  
10735, 4561, 0.8, 70.82, 7.4, 53.2, 124, 40975

2715, 3983, 1.1, 71.42, 6.4, 51.6, 82, 68782  
2284, 4660, 0.6, 72.13, 4.2, 60, 44, 96184  
11860, 4449, 1, 70.43, 6.1, 50.2, 126, 44966  
931, 4558, 1.3, 71.9, 2.4, 46.4, 127, 1049  
2816, 3635, 2.3, 67.96, 11.6, 37.8, 65, 30225  
681, 4167, 0.5, 72.08, 1.7, 53.3, 172, 75955  
4173, 3821, 1.7, 70.11, 11, 41.8, 70, 41328  
12237, 4188, 2.2, 70.9, 12.2, 47.4, 35, 262134  
1203, 4022, 0.6, 72.9, 4.5, 67.3, 137, 82096  
472, 3907, 0.6, 71.64, 5.5, 57.1, 168, 9267  
4981, 4701, 1.4, 70.08, 9.5, 47.8, 85, 39780  
3559, 4864, 0.6, 71.72, 4.3, 63.5, 32, 66570  
1799, 3617, 1.4, 69.48, 6.7, 41.6, 100, 24070  
4589, 4468, 0.7, 72.48, 3, 54.5, 149, 54464  
376, 4566, 0.6, 70.29, 6.9, 62.9, 173, 97203

#### 代码示例：

```
import org.apache.spark.SparkConf
import org.apache.spark.SparkContext
import scala.collection.mutable.ArrayBuffer
import org.apache.spark.sql.SQLContext
import org.apache.spark.ml.feature.VectorAssembler
import org.apache.spark.ml.regression.LinearRegression

object Driver {

  def main(args: Array[String]): Unit = {

    val conf=new SparkConf().setMaster("local").setAppName("lr-murder")

    val sc=new SparkContext(conf)

    val sqlc=new SQLContext(sc)

    val raw_data=sc.textFile("d://ml/lr.txt")

    val map_data=raw_data.map { x =>
```

```

val split_list=x.split(",")
(split_list(0).toDouble,split_list(1).toDouble,split_list(2).toDouble,
split_list(3).toDouble,split_list(4).toDouble,split_list(5).toDouble,
split_list(6).toDouble,split_list(7).toDouble)

}
val df=sql.createDataFrame(map_data)

val data = df.toDF("Population", "Income", "Illiteracy", "LifeExp",
    "Murder", "HSGrad", "Frost", "Area")
val colArray = Array("Population", "Income", "Illiteracy", "LifeExp",
    "HSGrad", "Frost", "Area")

val assembler = new VectorAssembler().setInputCols(colArray).setOutputCol("features")
val vecDF= assembler.transform(data)

// 建立模型，预测谋杀率Murder
// 设置线性回归参数
val lr1 = new LinearRegression()

val lr2 = lr1.setFeaturesCol("features").setLabelCol("Murder").setFitIntercept(true)
// RegParam : 正则化
val lr3 = lr2.setMaxIter(10).setRegParam(0.3).setElasticNetParam(0.8)

val lr = lr3

// 将训练集合代入模型进行训练
val lrModel = lr.fit(vecDF)

lrModel.extractParamMap()

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

// 模型进行评价
val trainingSummary = lrModel.summary
println(s"numIterations: ${trainingSummary.totalIterations}")
println(s"objectiveHistory: ${trainingSummary.objectiveHistory.toList}")

```

```
trainingSummary.residuals.show()
println(s"RMSE: ${trainingSummary.rootMeanSquaredError}")
println(s"r2: ${trainingSummary.r2}")

val predictions = lrModel.transform(vecDF)
println("输出预测结果")
val predict_result = predictions.selectExpr("features", "Murder", "round(prediction,1) as
prediction")
predict_result.foreach(println(_))

}
}
```