算法数据挖掘算法讨论

数据挖掘能分析出数据中的有用信息，给企业带来显著的经济效益，这使得数据挖掘技术越来越普及。如在销售数据中发掘顾客的消费习惯，并可从交易记录中找出顾客偏好的产品组合，其他包括找出流失顾客的特征与推出新产品的时机点等都是零售业常见的实例；利用数据挖掘分析顾客群的消费行为与交易纪录，结合基本数据，并依其对品牌价值等级的高低来区隔顾客，进而达到差异化营销的目的。

数据挖掘的基本任务主要体现在以下五个部分：**分类和回归、聚类、关联规则、时序模式、数据降维、推荐系统**。

1，分类和回归

分类和回归是数据挖掘中最常用的算法，常用的分类和回归算法有：线性回归，SVM，决策树，朴素贝叶斯，KNN，逻辑回归，人工神经网络，卷积神经网络等等。

* 1. 分类和回归的应用场景

分类和回归在商业上能有如下的应用场景：

1）如何将信用卡申请人分为低、中、高风险群？

2）如何预测哪些顾客在未来半年内会取消该公司服务，哪些电话用户会申请增值服务？

3）如何预测银行可以安全地贷给贷款人的贷款量？

4）哪些使用2G通信网络的手机用户有可能转换到3G通信网络？

5）如何有效预测房地产开发中存在的风险？

6）市场经理需要进行数据分析，以便帮助他预测具有某些特征的顾客会购买一台新的计算机。

* 1. 分类与回归的概念

分类（Classification）：指将数据映射到预先定义好的群组或类。

因为在分析测试数据之前，类别就已经确定了，所以分类通常被称为有监督的学习。分类算法要求基于数据属性值来定义类别，通常通过已知所属类别的数据的特征来描述类别。分类就是构造一个分类函数（分类模型），把具有某些特征的数据项映射到某个给定的类别上。

在分类中一般训练数据集是带有类标号的，也就是说在分类之前，要划分的类别是已经确定的。通常分类模型是以分类规则、决策树或数学表达式的形式给出。

回归（Regression）：用属性的历史数据预测未来趋势。

回归首先假设一些已知类型的函数（例如线性函数、Logistic函数等）可以拟合目标数据，然后利用某种误差分析确定一个与目标数据拟合程度最好的函数。

回归模式的函数定义与分类模式相似，主要差别在于分类模式采用离散预测值（例如类标号），而回归模式采用连续的预测值。在这种观点下，分类和回归都是预测问题。但数据挖掘业界普遍认为：用预测法预测类标号为分类，预测连续值（例如使用回归方法）为预测。许多问题可以用线性回归解决，许多非线性问题可以通过对变量进行变化，从而转换为线性问题来解决。

* 1. 常见的分类和回归算法

1.3.1 线性回归

**原理**：首先要了解什么是线性模型，给定由d个属性描述的实例 q其中是指x在第i个属性上的取值，线性模型试图学得一个通过属性的线性组合来进行预测的函数，即

其中,b是我们要学习的参数，是我们训练学习的目的，在得到这些参数以后就能得到模型就可以确定了。

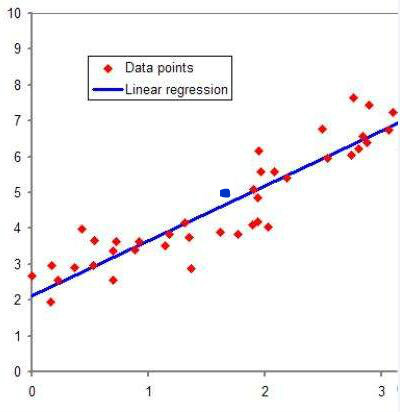
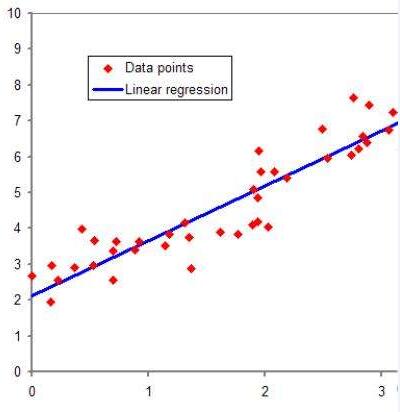
线性回归就是试图学习一个线性模型以尽可能准的预测实值输出标记。

若要求得最好的,b，我们首先需要定义一个能够判断模型优劣的损失函数。

损失函数定义为：

其中为所求模型的结果，为样本真正的结果，将他们的平方差之和作为损失函数。最后通过最小化损失函数作为最优解的判断。

如图一所示，线性回归通过对给出的数据，模拟出一条线性直线来表示数据的模型，之后再出现新的样本时，就能直接预测出该样本的输出值了。



图一

**实现**：1，现在有很多流行的开源工具都能够使用线性回归算法，如tensorflow，caffe，Torch等框架都支持线性回归的方法。

2，Spark中可以使用MLlib库直接调用线性回归的算法。由LinearRegressionWithSGD类的object定义了train函数来训练线性回归的数据。train函数的定义如下所示：

|  |
| --- |
| package org.apache.[Spark](http://lib.csdn.net/base/spark).mllib.regression    def train(        input: RDD[LabeledPoint],        numIterations: Int,        stepSize: Double,        miniBatchFraction: Double): LinearRegressionModel = {      new LinearRegressionWithSGD(stepSize, numIterations, miniBatchFraction).run(input)    } |

其中参数input为输入样本，numIterations为迭代次数，stepSize为步长，miniBatchFraction为迭代因子。创建一个LinearRegressionWithSGD对象，初始化梯度下降算法。

1.3.2 决策树

原理：决策树是一类常见的分类的机器学习方法。以二分法为例，从给定的训练集中学习一个模型用以对新的实例进行分类。

在商业中通常都会有像是/否这样的两难选择，比如，应不应该给这个客户发信用卡，或者应不应该给这个客户发高级信用卡等等。此时就可以使用决策树的方法来尝试。然而在决定一件事的时候有很多的属性，究竟哪些属性重要，哪些属性不重要呢？这里需要引用“信息增益”来判定属性的重要程度。

信息增益：为了精确地定义信息增益，我们先定义信息论中广泛使用的一个度量标准。一个属性A相对样例集合S的信息增益记作Gain(S,A)，通过比较属性的的大小来决定哪个属性比较重要。

下面以天气（阴、晴）和风力（强、弱）两个属性决定打球与否的例子来看看如何计算它们的信息增益。

|  |  |  |
| --- | --- | --- |
| 天气 | 风力 | 结果 |
| 晴 | 小 | 不打 |
| 晴 | 大 | 不打 |
| 晴 | 小 | 打 |
| 晴 | 小 | 打 |
| 阴 | 小 | 打 |
| 阴 | 大 | 不打 |
| 阴 | 大 | 打 |
| 晴 | 小 | 不打 |
| 阴 | 小 | 打 |
| 阴 | 小 | 打 |
| 阴 | 大 | 打 |
| 晴 | 大 | 打 |
| 晴 | 小 | 打 |
| 晴 | 大 | 不打 |

上面的14个实例，将打球作为正例，不打球作为负例。

首先计算要去打球的熵：

然后考虑天气的情况：

天气：晴有7天，7天中有3天去打球，有4天不打球；阴有7天，其中6天去打球1天没打球，那么就可以计算天气状况下对打球的影响：

天气为晴天的熵为：

天气为阴天的熵为：

那么属性weather对打球的信息增益为：

属性weather对打球的信息增益为0,151,按照相同的方法可以计算出属性风力Wind对打球的影响，结果是：

通过计算可以看出对于打球来说天气的影响要比风力的影响要大，所以在建立决策树的时候应该把天气属性放在风力属性的前面。

在得到了各个属性的重要程度以后，就可以通过属性来构建决策树了。

ID3是常用的构建决策树的算法之一。它的算法步骤是：

ID3算法：  
1）对当前样本集合，计算所有属性的信息增益；  
2）选择信息增益最大的属性作为测试属性，把测试属性取值相同的样本划为同一个子样本集；  
3）若子样本集的类别属性只含有单个属性，则分支为叶子节点，判断其属性值并标上相应的符号，然后返回调用处；否则对子样本集递归调用本算法。

决策树的优劣分析：

优势：  
- 非黑盒  
- 轻松去除⽆关属性 （ Gain = 0）  
- 测试起来很快 （ O(depth)）  
劣势：  
- 只能线性分割数据  
- 贪婪算法（可能找不到最好的树）

后来人们在决策树的条件下对其进行改进，如Bagging,Boosting,Fandom Forest, XGBoost等。

实现：1，有很多开源的库如：scikit learn等，还有开源的框架也能实现决策树的功能如tensorflow，caffe，Torch等。

2，在spark的MLlib也可以很方便的直接调用决策树的方法。实现方法如下：

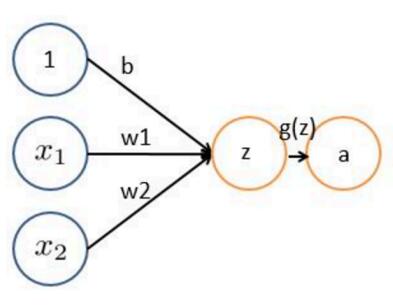
|  |
| --- |
| sc = SparkContext(appName="PythonDecisionTreeClassificationExample")  # 加载和解析数据文件为RDD  dataPath = "/home/zhb/Desktop/work/DecisionTreeShareProject/app/sample\_libsvm\_data.txt" print(dataPath)  data = MLUtils.loadLibSVMFile(sc,dataPath)  # 将数据集分割为训练数据集和测试数据集  (trainingData,testData) = data.randomSplit([0.7,0.3])  print("train data count: " + str(trainingData.count()))  print("test data count : " + str(testData.count()))  # 训练决策树分类器  # categoricalFeaturesInfo 为空，表示所有的特征均为连续值  model = DecisionTree.trainClassifier(trainingData, numClasses=2, categoricalFeaturesInfo={}, impurity='gini', maxDepth=5, maxBins=32)  # 测试数据集上预测  predictions = model.predict(testData.map(lambda x: x.features))  # 打包真实值与预测值  labelsAndPredictions = testData.map(lambda lp: lp.label).zip(predictions) |

1.3.3 人工神经网络和卷积神经网络

人工神经网络（ANN）是人类使用计算机模拟对人脑的神经网络，来对数据进行处理。它是以计算机技术为基础，结合数学、物理学以及信息处理等学科来对人脑神经网络的组成和工作原理进行研究，建立抽象模型来实现的。在人工神经网络中利用方向传导算法（BP算法）可以让一个人工神经网络模型从大量训练样本中学习统计规律，从而对未知事件做预测。

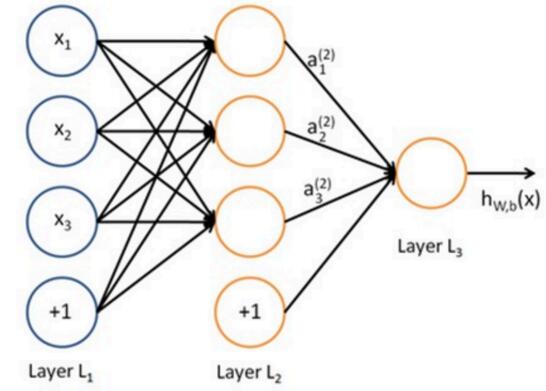
在了解人工神经网络之前，需要了解感知器的感念。

如下图是一个简单的感知器：



其中x1,x2为输入值，w1,w2,b为要求的权重，g(z)为激活函数。

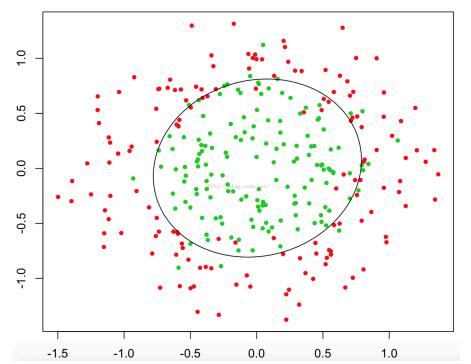
感知器中通过对输入的数据信息的拟合，得到想要的模型。但是感知器的作用范围很有限，因此可以增加其节点（也就是神经元）的个数以及层数来提高该算法的能力。



如上图所示，ANN一般都会有一个输入层L1，一个输出层L3以及若干个隐藏层。

层中每个神经元都与其相邻层的神经元全连接，每个连接都有一个权值，神经网络的目的就是通过数据的训练来获取权值。

线性回归和SVM对线性可分的分类问题会取得很好的效果，但是ANN可以得到非线性的分类结果。如下图所示：



ANN算法是一个正向前推+反向后推的过程。

ANN的步骤：

（1），给网络上的连接权值随着设置每一个（0~1）范围内的值

（2）将样本输入，下层网络对输入样本进行加权求和以及映射并把结果传递给下一层网络

（3）下层网络步骤（2）的行为，一直到输出层。

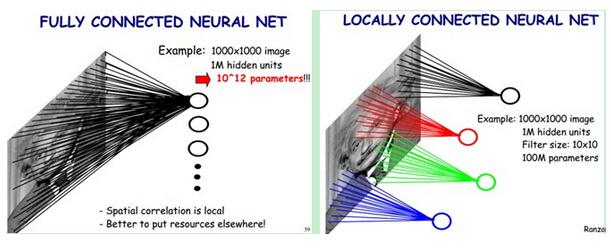
（4）输出层将计算的结果与真正的结果进行对比，通过最小化代价函数，计算出残差，然后反向传导来修复网络的连接权值。

人工神经网络的缺点：1）比较容易过拟合，参数比较难tune，而且需要不少trick；

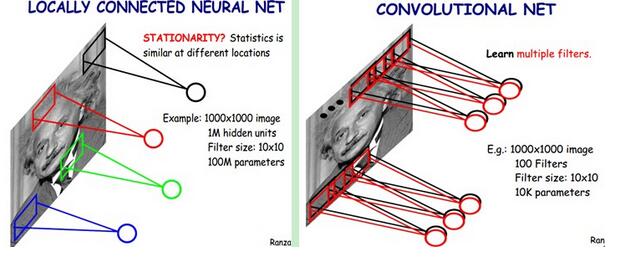
2）训练速度比较慢，在层次比较少（小于等于3）的情况下效果并不比其它方法更优；

卷积神经网络(CNN)：针对人工神经网络训练速度慢容易过拟合的缺点，出现了卷积神经网络。CNN通过引入局部感受野和权值共享两个方法来改善ANN的不足。下面介绍两个方法并以图形输入为例来进行比较。

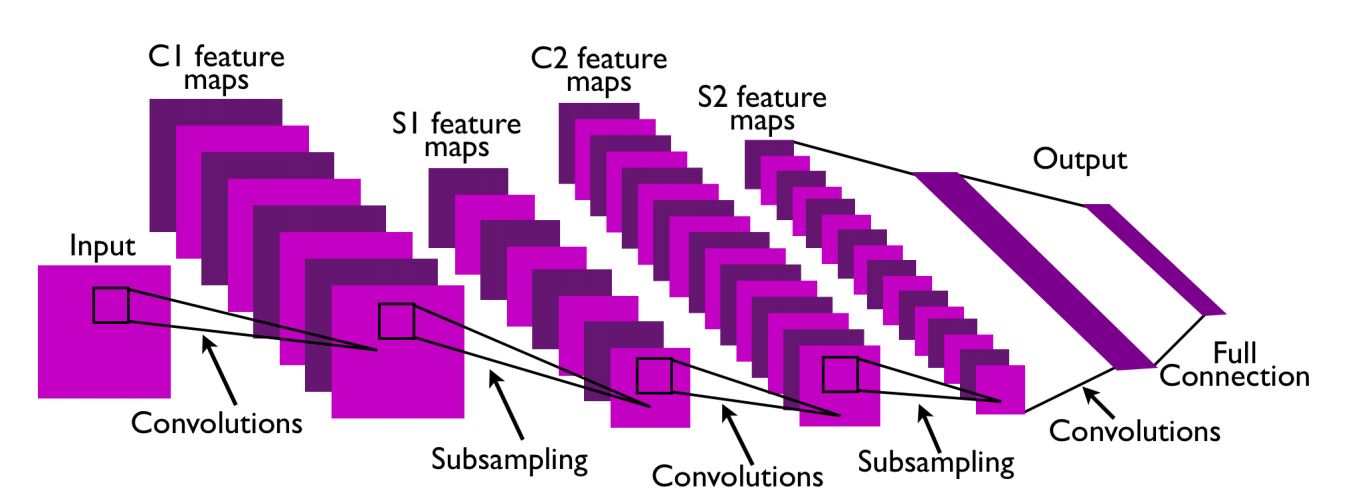
局部感受野：在ANN中每个神经元是与相邻层的所以神经元是一一连接的关系，所以链接权值的个数会随着层数的增加而极具的增加。而在CNN中神经元是其他层之间的联系是局部的，就像人是通过一个局部的感受野去感受外界图像一样，每一个神经元都不需要对全局图像做感受，每个神经元只感受局部的图像区域，然后在更高层，将这些感受不同局部的神经元综合起来就可以得到全局的信息了。ANN与CNN的连接方式如下图所示：



权值共享：在ANN中在同一层网络中不同的神经元其连接权值是不同的，这样每一层有多少的神经元就会有多少的连接权值，再与其他层的神经元相连接之后连接权值的个数还是会很多。在CNN中是通过相同层的神经元的权值共享来降低连接权值的个数，进一步加快算法的训练速度。ANN与CNN的权值方面的比较如下所示：

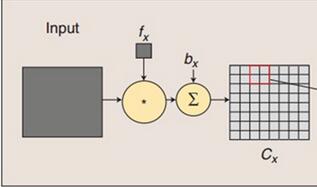


卷积神经网络的简单模型如下图所示：

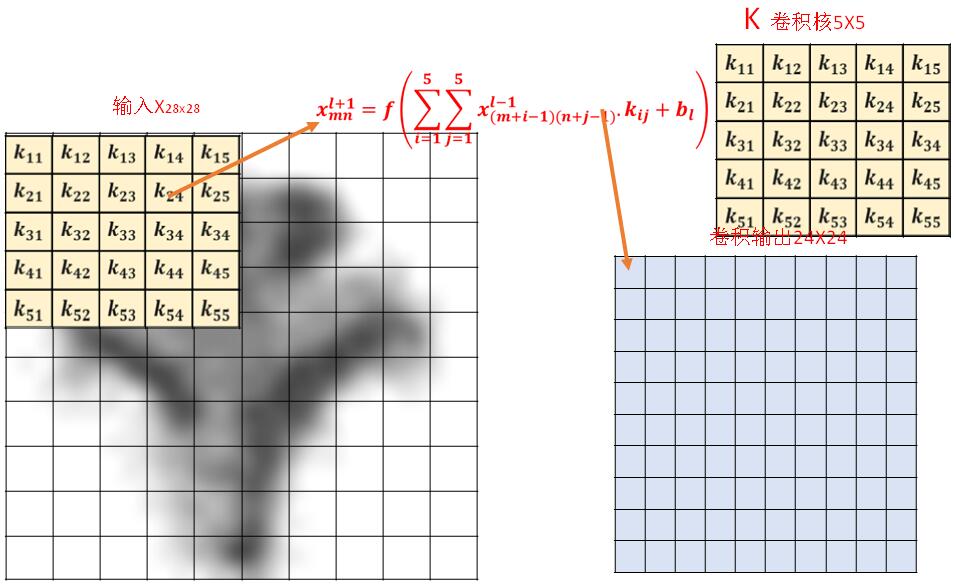


上面有一个输入层，两个降采样层，两个卷积层，两个全连接层以及一个输出层。该模型是一LeNet-5模型其主要的目标是对手写数字图片的识别。下面展开介绍各个层。

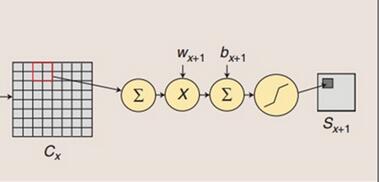
卷积层：卷积层为特征提取层，每个神经元的输入与前一层的局部感受野相连，并提取该局部的特征，一旦该局部特征被提取后，它与其他特征间的位置关系也随之确定下来； 如下图所示：



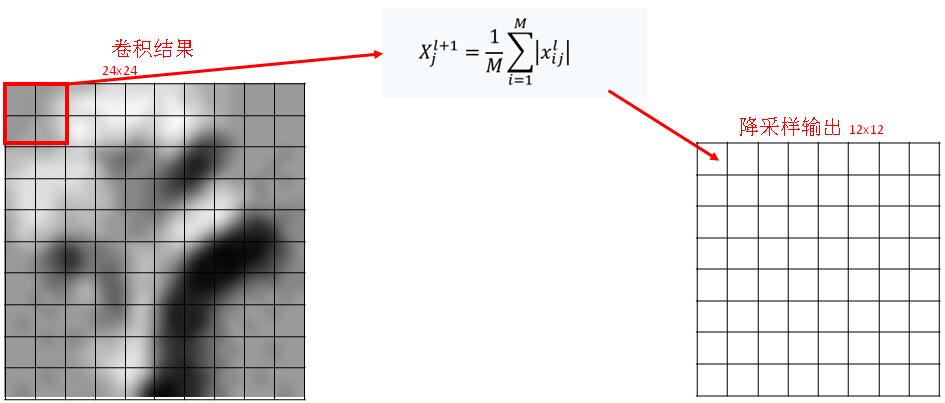
图像与卷积核进行卷积再加上偏移量得到新的像素值。具体如下图所示：



降采样层：利用图像局部相关性的原理，对图像进行子抽样，可以减少数据处理量同时保留有用信息。



如上图所示，是求和降采样：将区域内的像素值相加然后乘上权值再加上偏移量最后使用激活函数将其映射到新成新的map。



CNN算法的训练过程主要如下：

其主要步骤还是分为两大阶段：向前传播阶段和向后传播阶段。

1，向前传播阶段：

输入层-》卷积层-》采样层-》卷积层-》采样层—》全连接层-》输出

2，向后传播阶段：主要思想是通过最小化残差来调整权值和偏置，但是由于权值共享等原因不像BP那样能统一对待，需要分情况讨论；

反向传导：

1）：输出层的残差：

2）：若该L层是卷积层，下一层L+1为采样层。

a,将采样层的残差进行扩充：

b,利用卷积计算卷积层的残差：

3）：该L层为采样层下一层L+1层为卷积层

假设L层有N个feature map，L+1层有M个feature map。那么L层每个feature map 的残差就等于L+1层所以残差与卷积核之和。

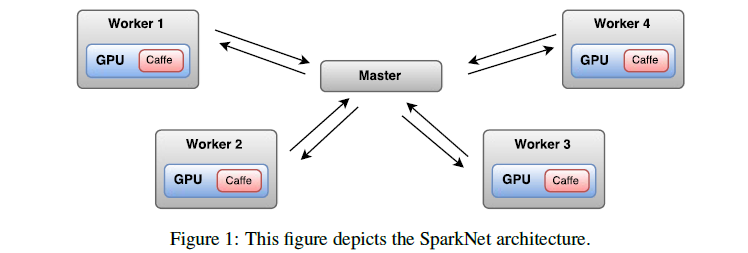
在得到各个层的残差之后就可以更新权重和偏置了。

用法：1，有很多开源的框架也能实现神经网络的功能如tensorflow，caffe，Torch等。

2，在Spark中并没有直接能够用于调用深度学习的包，但是可以使用SparkNet来使用深度学习的技术。

SparkNet 是基于Spark的深度神经网络架构，提供了便捷的接口能够去访问Spark RDDs；同时提供Scala接口去调用caffe；还拥有一个轻量级的tensor 库；使用了一个简单的并行机制来实现SGD的并行化，使得SparkNet能够很好的适应集群的大小并且能够容忍极高的通信延时；它易于部署，并且不需要对参数进行调整；它还能很好的兼容现有的caffe模型；

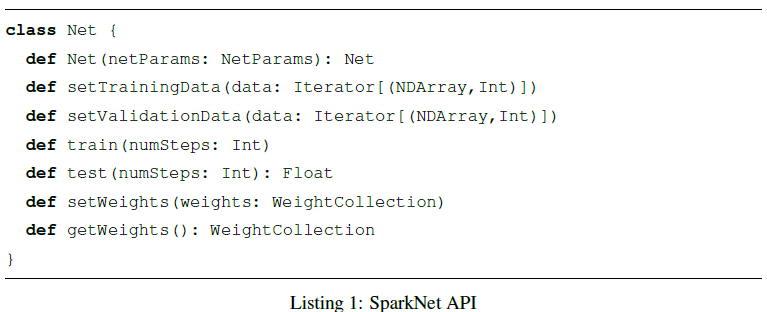
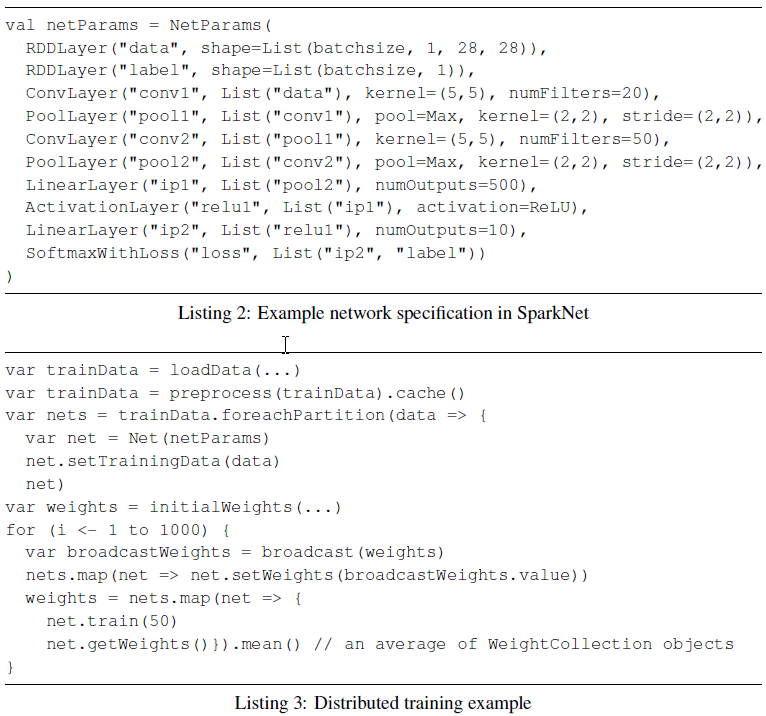
下面这张图是SparkNet的架构：



从上图可以看出，Master 向每个worker 分发任务之后，各个worker都单独的使用Caffe（利用GPU）来进行训练。每个worker完成任务之后，把参数传回Master。

SparkNet 是建立在[Apache Spark](http://lib.csdn.net/base/10" \t "_blank" \o "Apache Spark知识库)和Caffe深度学习库的基础之上的。SparkNet 用[Java](http://lib.csdn.net/base/17" \t "_blank" \o "Java EE知识库)来访问Caffe的数据，用Scala来访问Caffe的参数，用ScalaBuff来使得Caffe网络在运行时保持动态结构。SparkNet能够兼容Caffe的一些模型定义文件，并且支持Caffe模型参数的载入。

下面简单贴一下SparkNet的api和模型定义、模型训练代码。

1.4 评价指标

评价指标：我们在得到模型之后，还应该知道模型的性能到底是好是坏，也就是要对模型的性能进行评估。下面对分类和回归的评价指标进行讨论。

（1）分类算法的评价指标

以二分类为标准，常用的分类指标有：预测的正确率和错误率、召回率、准确率-召回率曲线下方的面积、ROC曲线和ROC曲线下面的面积等。

正确率：训练样本中被正确分类的数目除以样本的总数。

错误率：训练样本中被错误分类的数目除以样本的总数。

准确率和召回率：准确率定义为真阳性的数目除以真阳性和假阳性的总数，其中真阳性是指被正确预测的类别为1的样本，假阳性是错误预测为类别1的样本。精确率表示的是预测为正样本中有多少是真正的正样本。召回率定义为真阳性的数目除以真阳性和假阴性的和，其中假阴性是类别为1却被预测为0的样本。召回率是指所有正样本中有多少预测正确的。准确率-召回率的计算如下图所示：



准确率和召回率通常一起组合使用，形成准确率-召回率（PR）曲线。并以曲线下的面积作为衡量模型好坏的标准。

ROC曲线与ROC曲线在概念上和PR曲线类似，它是对分类器的真阳性率−假阳性率的图形化解释。和准确率和召回率类似， ROC曲线（图5-9）表示了分类器性能在不同决策阈值下TPR对FPR的折衷。曲线上每个点代表分类器决策函数中不同的阈值。

（2）回归模型的评价指标

对回归模型而言，因为目标变量是任一实数，所以我们的模型不大可能精确预测到目标变量。然而，我们可以计算预测值和实际值的误差，并用某种度量方式进行评估。一些用于评估回归模型的方法包括：均方误差（MSE， Mean Squared Error）、 均方根误差（RMSE， Root Mean Squared Error）、 平均绝对误差（MAE， Mean Absolute Error）、 R-平方系数（R-squared coefficient）等。

均方误差（MSE）：所有样本预测值和实际值平方差之和，最后除以样本总数。

均方根误差（RMSE）：就是在均方误差上面加上一个二次根号。

平方绝对误差（MAE）：就是预测值和实际值的差的绝对值的平均值。

2 聚类

2.1 聚类的应用场景

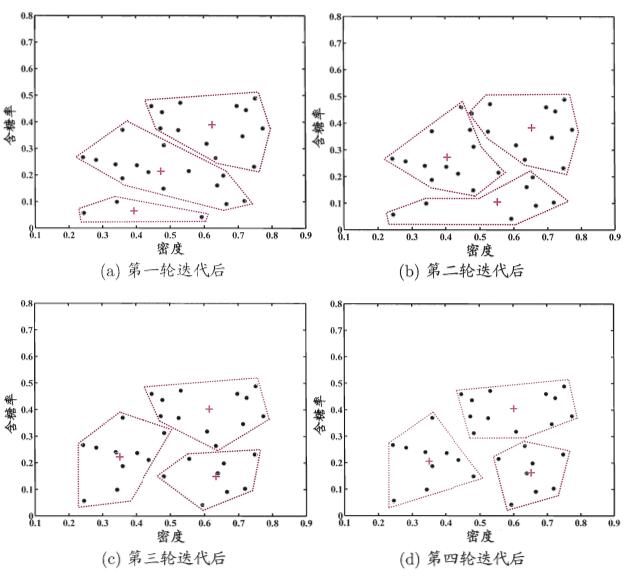
经常会碰到这样的问题：  
1）如何通过一些特定的症状归纳某类特定的疾病？  
2）谁是银行信用卡的黄金客户？  
3）谁喜欢打国际长途，在什么时间，打到哪里？  
4）对住宅区进行聚类，确定自动提款机 ATM 的安放位置。  
5）如何对用户 WAP 上网行为进行分析，通过客户分群进行精确营销？  
6）促销时针对哪一类客户，这类客户具有哪些特征？这类问题往往是在促销前首要解决的问题，对整个客户做分群，将客户分组在各自的群组里，然后对每个不同的群组，采取不同的营销策略。

2.2 聚类的概念

聚类（Clustering）是在没有给定划分类的情况下，根据信息相似度进行信息聚类的一种方法，因此聚类又称为无指导的学习。与分类不同，分类需要先定义类别和训练样本，是有指导的学习。聚类就是将数据划分或分割成相交或者不相交的群组的过程，通过确定数据之间在预先指定的属性上的相似性，就可以完成聚类任务。聚类的输入是一组未被标记的数据，根据数据自身的距离或相似度进行划分。划分的原则是保持最大的组内相似性和最小的组间相似性，也就是使不同聚类中的数据尽可能地不同，而同一聚类中的数据尽可能地相似。

原理：聚类是将数据集中的样本划分成若干个通常是不相交的子集，每个子集称为“簇”，它在无监督的学习中应用很广。

形式化的说，假定样本集包含m个无标记的样本，每个样本是一个n维的特征向量，则聚类算法将样本集D划分为k个不相交的簇{|l=1,2,…,k},簇与簇之间是没有交集的，那么聚类最终被分割成包括m个元素的簇标记向量的表示。



如上图所示，通过k-means聚类算法能够将一些点分割成若干的区域。

2.3 性能度量和距离计算

在介绍聚类算法k-means聚类算法之前，先介绍距离计算和性能度量两个概念。性能度量：性能度量是一种度量准则，用于表示被聚类划分的样本簇的优劣。对于最终的聚类结果我们追求“簇内相似度高”且“簇间相似度低”的标准。

聚类性能度量大致有两类，一类是将聚类的结果与某个“参考模型”进行比较，称为“外部指标”，一类是直接考察聚类结果而不利用任何参考模型，称为“内部指标”。

1，外部指标：假设数据集为，假定通过聚类给出的簇划分为,参考模型给出的簇的划分为：。

a=|SS|,b=|SD|,c=|DS|,d=|DD|

可以将a,b,c,d理解成： a表示两个样本对(xi，xj)在我们得到的聚簇中是属于同一聚簇的，而且在参考模型中(xi，xj)也属于同一聚簇的样本对的个数。b表示两个样本对(xi，xj)在我们得到的聚簇中是属于同一聚簇的，但是在参考模型中(xi，xj)是不属于同一聚簇的样本对的个数。c表示两个样本对(xi，xj)在我们得到的聚簇中是不属于同一聚簇的，但是在参考模型中(xi，xj)是属于同一聚簇的样本对的个数。d表示两个样本对(xi，xj)在我们得到的聚簇中是不属于同一聚簇的，但是在参考模型中(xi，xj)是也不属于同一聚簇的样本对的个数

在得到a,b,c,d之后可以使用下面这些外部指标来判定一个聚类算法的优劣：

Jaccard系数：

FM指数：

上述性能度量的结果均在[0,1]区间，值越大越好。

2，内部指标

在得到数据集D的簇划分之后，可以定义如下一些变量：

、、以及。

其中，dist(..)用于计算两个样本间的距离，下面会有介绍；代表C的中心点，表示簇C内样本间的平均距离，表示簇C内样本间的最远距离，对应于簇和簇最近样本间的距离，对应于簇和簇中心点间的距离。那么常用的聚类性能度量内部指标：

DB指数：

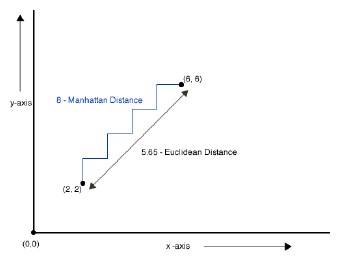
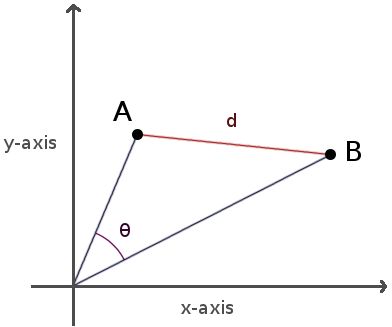
Dunn指数：

DBI的值越小越好，DI的值越大越好。

距离计算：两个样本之间的距离计算对于聚类算法来说非常重要。距离的计算有欧式距离和曼哈顿距离等等。

欧式距离就是计算两个样本点之间的在各个维度上的差值的平方差：如图二(a)中所示。

曼哈顿距离也称为街区距离，即两个点想城市里的两栋建筑一样，不能直接到达，需要按照一定的规则到达，如图二(b)所示。



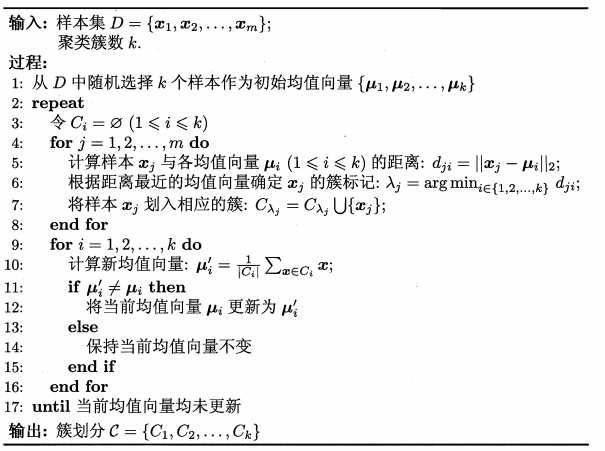
图二(a)欧式距离 (b)曼哈顿距离

2.4 常见的聚类算法

2.4.1 k-mans聚类

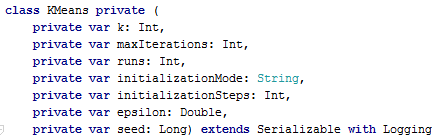
k-manss算法：对于给定的样本集,k-means算法针对聚类所得簇划分最小化平方差： ，其中 是簇的均值向量。

E的值越小则簇内的样本相似度就越高，k-means采用贪心算法，通过迭代来将E最小化。

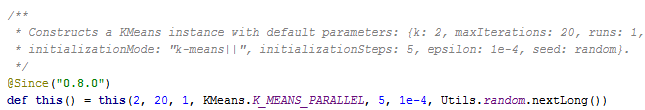
k-means算法的流程如下所示：

用法：1，有很多有开源的框架都能实现包括k-means的聚类的功能如tensorflow，caffe，Torch等。

2，在Spark中可以直接通过MLlib调用k-means聚类的算法。MLlib的k-means算法实现如下所示：



然后通过如下的默认构造函数，可以看到这些可调参数具有如下初始值。



参数的含义解释如下：

* k 表示期望的聚类的个数。
* maxInterations 表示方法单次运行最大的迭代次数。
* runs 表示算法被运行的次数。K-means 算法不保证能返回全局最优的聚类结果，所以在目标数据集上多次跑 K-means 算法，有助于返回最佳聚类结果。
* initializationMode 表示初始聚类中心点的选择方式, 目前支持随机选择或者 K-means||方式。默认是 K-means||。
* initializationSteps表示 K-means||方法中的部数。
* epsilon 表示 K-means 算法迭代收敛的阀值。
* seed 表示集群初始化时的随机种子。

通常应用时，我们都会先调用 KMeans.train 方法对数据集进行聚类训练，这个方法会返回 KMeansModel 类实例，然后我们也可以使用 KMeansModel.predict 方法对新的数据点进行所属聚类的预测，这是非常实用的功能。

2.5 聚类算法的评价指标

聚类的评价指标可以使用上面已经提到的“内部指标”和“外部指标”进行评价。

3 关联规则

3.1 关联规则的应用场景

我们经常会碰到这样的问题：

1）商业销售上，如何通过交叉销售得到更大的收入？

2）保险方面，如何分析索赔要求发现潜在的欺诈行为？

3）银行方面，如何分析顾客消费行业，以便有针对性地向其推荐感兴趣的服务？

4）哪些制造零件和设备设置与故障事件关联？

5）哪些病人和药物属性与结果关联？

6）哪些商品是已经购买商品A的人最有可能购买的？

7) 可用于消费市场价格分析，猜测顾客的消费习惯；

8) 网络安全领域中的入侵检测技术；

9) 可用在用于高校管理中，根据挖掘规则可以有效地辅助学校管理部门有针对性的开展贫困助学工作；

10) 可用在移动通信领域中，指导运营商的业务运营和辅助业务提供商的决策制定。

除此之外，人们希望从大量的商业交易记录中发现有价值的关联知识，以帮助进行商品目录的设计、交叉营销或其他有关的商业决策。在商业销售上，关联规则可用于交叉销售，以得到更大的收入；在保险业务方面，如果出现了不常见的索赔要求组合，则可能为欺诈行为，需要进一步调查；在医疗方面，可找出可能的治疗组合；在银行方面，对顾客进行分析，可以推荐感兴趣的服务等。这些都属于关联规则挖掘问题，关联规则挖掘的目的是在一个数据集中找出各项之间的关系，从大量的数据中挖掘出有价值的描述数据项之间相互联系的有关知识。随着收集和存储在[数据库](http://www.2cto.com/database/)中的数据规模越来越大，人们对从这些数据中挖掘相应的关联知识越来越有兴趣。

3.2 关联规则的概念

关联规则（Association）：揭示数据之间的相互关系，而这种关系没有在数据中直接表示出来。关联规则的主要目的是找出数据集中的频繁模式即多次重复出现的模式；和并发关系即同时出现的关系；频繁和并发关系也称作关联。

应用关联规则最经典的案例就是购物篮分析，通过分析顾客购物篮商品之间的关联，可以挖掘出顾客的购物习惯，从而帮助零售商更好地制定有针对性的营销策略。以下举一个简单的关联规则的例子：

婴儿尿不湿🡪啤酒[支持度=10%，置信度=70%]

这个规则表明，在所有顾客当中，有10%的顾客同时购买了婴儿尿不湿和啤酒，而所有购买了婴儿尿不湿的顾客中，占70%的人同时还购买了啤酒。发现这个关联规则之后，超市零售商决定把婴儿尿不湿和啤酒拜放在一起进行促销，结果明显提升了销售额。

支持度和置信度是衡量关联规则强度的两个重要指标，它们分别反应着所发现规则的有用性和确定性。

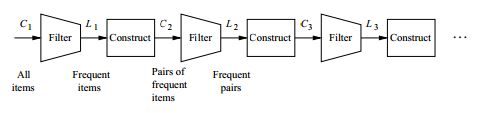
3.3 关联规则的常用算法

3.3.1 Apriori算法

Apriori算法具体分为以下两步进行：

（1）生成所有的频繁项目集。一个频繁项目集是一个支持度高于最小支持度阈值的项目集。

（2）从频繁项目集中生成所有的可信关联规则。这里的可信关联规则是指置信度大于最小置信度阈值的规则。



Ci是备选第i次备选项目数据集，Li是选出来的合格的频繁i项集，该流程迭代到Ci为空为止。

**用法**：Spark的MLlib库并不能调用Apriori。不过可以使用JAVA调用Weka库中的Apriori方法。

|  |
| --- |
| Instances data = null; try { BufferedReader reader = new BufferedReader( new FileReader( "TestStudenti.arff" ) ); data = new Instances(reader); reader.close(); data.setClassIndex(data.numAttributes()-1); } catch ( IOException e ) { e.printStackTrace(); } double deltaValue = 0.05; double lowerBoundMinSupportValue = 0.1; double minMetricValue = 0.5; int numRulesValue = 20; double upperBoundMinSupportValue = 1.0; String resultapriori; Apriori apriori = new Apriori(); apriori.setDelta(deltaValue); apriori.setLowerBoundMinSupport(lowerBoundMinSupportValue); apriori.setNumRules(numRulesValue); apriori.setUpperBoundMinSupport(upperBoundMinSupportValue); apriori.setMinMetric(minMetricValue); try{ apriori.buildAssociations( data ); } catch ( Exception e ) { e.printStackTrace(); } resultapriori = apriori.toString(); System.out.println(resultapriori); } } |

3.3 关联规则的评价指标

关联规则的评价指标可以通过通过对比预测的推荐给客户会购买的比例与实际的客户购买的情况的残差来判断模型的好坏。或者使用均方差、均方根差来判断（均方差、均方根差详细情况可参考1.4节）。

4 时序模式

4.1 时序模式的应用场景

1）下个月的商品销量、销售额或库存量是多少？

2）明天广州市的最高用电负荷是多少？

3）市场潜力预测

4）气象预测

5）股市涨跌预测

4.2 时序模式的概念

时序模式：描述基于时间或其他序列的经常发生的规律或趋势，并对其建模。

与回归一样，它也用已知的数据预测未来的值，但这些数据的区别是变量所处时间的不同。序列模式将关联模式和时间序列模式结合起来，重点考虑数据之间在时间维度上的关联性。时序模式包含时间序列分析和序列发现。

\* 时间序列分析：用已有的数据序列预测未来。在时间序列分析中，数据的属性值是随着时间不断变化的。回归不强调数据间的先后顺序，而时间序列要考虑时间特性，尤其要考虑时间周期的层次，如天、周、月、年等，有时还要考虑日历的影响，如节假日等。

\* 序列发现：用于确定数据之间与时间相关的序列模式。这些模式与在数据（或者事件）中发现的相关的关联规则很相似，只是这些序列是与时间相关的。

4.3 常用的时序算法

4.3.1 自回归积分移动平均过程（ARIMA）

ARIMA的基本思想是：将预测对象随时间的推移而形成的数据序列视为一个随机序列，用一定的数学模型来近似描述这个序列。这个模型一旦被识别后就可以从时间序列的过去值和现在值来预测未来。

ARIMA算法的过程如下：

（1） 获取被观测系统时间序列数据；   
（2） 对数据绘图，观测是否为平稳时间序列。若为非平稳时间序列要先进行d阶差分运算后化为平稳时间序列，此处的d即为ARIMA(p,d,q)模型中的d；若为平稳序列，则用ARMA(p,q)模型。所以ARIMA(p,d,q) 模型区别于ARMA(p,q)之处就在于前者的自回归部分的特征多项式含有d个单位根。   
（3） 对得到的平稳时间序列分别求得其自相关系数ACF 和偏自相关系数PACF，通过对自相关图和偏自相关图的分析，得到最佳的阶层 p 和阶数 q。由以上得到的d、q、p ，得到ARIMA模型。   
（4） 模型诊断，进行诊断分析，以证实所得模型确实与所观察到的数据特征相符。若不相符，重新回到第（3）步。

**应**用：ARIMA在Spark中并没有对于的函数用来调用，在市面上用R语言写ARIMA的比较多，或者使用软件SPSS建立ARIMA预测模型

原生态的spark中并没有支持时序模式的算法，可以调用第三方的库spark-ts来使用时序算法。

4.3.2 指数平滑

指数平滑法是在[移动平均法](http://wiki.mbalib.com/wiki/%E7%A7%BB%E5%8A%A8%E5%B9%B3%E5%9D%87%E6%B3%95)基础上发展起来的一种[时间序列分析预测法](http://wiki.mbalib.com/wiki/%E6%97%B6%E9%97%B4%E5%BA%8F%E5%88%97%E5%88%86%E6%9E%90%E9%A2%84%E6%B5%8B%E6%B3%95)，它是通过计算指数平滑值，配合一定的时间序列预测模型对现象的未来进行[预测](http://wiki.mbalib.com/wiki/%E9%A2%84%E6%B5%8B)。其原理是任一期的指数平滑值都是本期实际观察值与前一期指数平滑值的加权平均。

指数平滑的基本公式如下所示：

St--时间t的平滑值； yt--时间t的实际值； St−1 --时间t-1的平滑值； α--平滑常数，其取值范围为[0,1]。

在指数平滑中对离预测期较近的观察值赋予较大的权数，对离预测值较远的观察值赋予较小的权数，权数由近到远按指数规律递减，所以叫做指数平滑法。

根据平滑次数不同，指数平滑法分为：一次指数平滑法、二次指数平滑法和三次指数平滑法等。但它们的基本思想都是：预测值是以前观测值的加权和，且对不同的数据给予不同的权，新数据给较大的权，旧数据给较小的权。

当时间序列的变动出现直线趋势时，用一次指数平滑法来预测仍存在着明显的滞后偏差。因此，也需要进行修正。修正的方法也是在一次指数平滑的基础上再作二次指数平滑，利用滞后偏差的规律找出曲线的发展方向和发展趋势，然后建立直线趋势预测模型。故称为二次指数平滑法。并且在二次平滑之后可以使用直线模型来作为预测二次平滑的预测模型.

若时间序列的变动呈现出二次曲线趋势，则需要用三次指数平滑法。三次指数平滑是在二次指数平滑的基础上再进行一次平滑，三次指数平滑法的预测模型为一个一元二次的抛物线模型。

**应用：**能做时间序列的软件很多，如SAS、R、SPSS或者python的pandas等。原生态的spark中并没有支持时序模式的算法，可以调用第三方的库spark-ts来使用时序算法。

4.3 时序算法的评价指标

对于时序算法来说可以通过对实际值与模型的预测值之间的算法关系来判断模型的好坏。这些算法关系包括：均方误差，均方根误差，平方绝对误差（详细可参考1.4节）等。

5，数据降维

5.1 数据降维应用的场景

1）探索性数据分析；

2）提取特征去训练其他机器学习模型；

3）降低大型模型在预测阶段的存储和计算需求（例如，一个执行预测的生产系统）；

4）把大量文档缩减为一组隐含话题；

5）当数据维度很高时，使得学习和推广模型更加容易（例如，当处理文本、声音、图像、视频等非常高维的数据时）。

5.2 数据降维的概念

不同于我们之前学习的回归、分类和聚类，降维方法并不是用来做模型预测的。降维方法从一个D维的数据输入提取出k维表示， k一般远远小于D。因此，降维方法本身是一种预处理方法，或者说是一种特征转换的方法，而不是模型预测的方法。

降维方法中尤为重要的是，被抽取出的维度表示应该仍能捕捉大部分的原始数据的变化和结构。这源于一个基本想法：大部分数据源包含某种内部结构，这种结构一般来说是未知的（常称为隐含特征或潜在特征），但如果能发现结构中的一些特征，我们的模型就可以学习这种结构并从中预测，而不用从大量无关的充满噪音特征的原始数据中去学习预测。简言之，缩减维度可以排除数据中的噪音并保留数据原有的隐含结构。

5.3 常用的数据降维算法

5.3.1 主成分分析(PCA)

PCA是处理一个数据矩阵，抽取矩阵中k个主向量，主向量彼此不相关。计算结果中，第一个主向量表示输入数据的最大变化方向。之后的每个主向量依次代表不考虑之前计算过的所有方向时最大的变化方向。因此，返回的k个主成分代表了输入数据可能的最大变化。事实上，每一个主成分向量上有着和原始数据矩阵相同的特征维度。因此需要使用映射来做一次降维，原来的数据被投影到主向量表示的k维空间。

PCA算法的实现步骤如下：

1. 计算样本的协方差矩阵；
2. 计算协方差矩阵的特征值和特征向量；
3. 选取较大的特征值对应的特征向量组成模式矢量；
4. 通过矢量模式与去均值之后的维度相点乘得到降维后的数据

**用法：**

|  |
| --- |
| /\*\*  \* 数据降维  \* 主成分分析PCA  \* 设法将原来具有一定相关行（比如 P个指标）的指标  \* 重新组合成一组新的互相无关的综合指标来代替原来的指标，从而实现数据降维的目的  \*/  object PCA {  val conf = new SparkConf() //创建环境变量  .setMaster("local") //设置本地化处理  .setAppName("PCA") //设定名称  val sc = new SparkContext(conf)  def main(args: Array[String]) {  val data = sc.textFile("./src/main/spark/DataDimensionReduction/a.txt")  .map(\_.split(" ").map(\_.toDouble))  .map(line => Vectors.dense(line))  val rm = new RowMatrix(data)  val pc = rm.computePrincipalComponents(3)//提取主成分，设置主成分个数为３  val mx = rm.multiply(pc)//创建主成分矩阵  mx.rows.foreach(println)  }  } |

5.4 数据降维的评价指标

一般数据降维的目的是通过降维算法将样本维度很大的数据进行降维处理，若是数据被降维了且降维之后数据在运行时的准确率之类的评价指标并没有变坏太多，就说明降维算法是有效的。

所以可以使用：1，数据降维前的总维度-数据降维后的总维度

2，数据降维后的运行效果来评价一个降维模型的好坏。

1和2相结合来评价一个降维算法

6，推荐引擎

6.1 推荐引擎的应用场景

1）可选项众多：可选的物品越多，用户就越难找到想要的物品。如果用户知道他们想要什么，那搜索能有所帮助。然而最适合的物品往往并不为用户所事先知道。这时，通过向用户推荐相关物品，其中某些可能用户事先不知道，将能帮助他们发现新物品。  
2）偏个人喜好：当人们主要根据个人喜好来选择物品时，推荐引擎能利用集体智慧，根据其他有类似喜好用户的信息来帮助他们发现所需物品。

6.2 推荐引擎的概念

推荐引擎背后的想法是预测人们可能喜好的物品并通过探寻物品之间的联系来辅助这个过程。从这点上来说，它和同样也做预测的搜索引擎互补。但与搜索引擎不同，推荐引擎试图向人们呈现的相关内容并不一定就是人们所搜索的，其返回的某些结果甚至人们都没听说过。

一般来讲，推荐引擎试图对用户与某类物品之间的联系建模。比如使用推荐引擎来告诉用户有哪些电影他们可能会喜欢。如果这点做得很好，就能吸引用户持续使用我们的服务。同样，如果能准确告诉用户有哪些电影与某一电影相似，就能方便用户在站点上找到更多感兴趣的信息。这也能提升用户的体验、参与度以及站点内容对用户的吸引力。

实际上，推荐引擎的应用并不限于电影、书籍或是产品。本章内容同样适用于用户与物品关系或社交网络中用户与用户之间的关系。比方说向用户推荐他们可能认识或关注的用户。

6.3 推荐引擎常用的算法

6.3.1 协同过滤算法

协同过滤算法是最常用的推荐算法，是一种借助众包智慧的途径。它利用大量已有的用户偏好来估计用户对其未接触过的物品的喜好程度。其内在思想是相似度的定义。

在基于用户的方法的中，如果两个用户表现出相似的偏好（即对相同物品的偏好大体相同），那就认为他们的兴趣类似。要对他们中的一个用户推荐一个未知物品，便可选取若干与其类似的用户并根据他们的喜好计算出对各个物品的综合得分，再以得分来推荐物品。其整体的逻辑是，如果其他用户也偏好某些物品，那这些物品很可能值得推荐。

基于物品的方法来做推荐。这种方法通常根据现有用户对物品的偏好或是评级情况，来计算物品之间的某种相似度。这时，相似用户评级相同的那些物品会被认为更相近。一旦有了物品之间的相似度，便可用用户接触过的物品来表示这个用户，然后找出和这些已知物品相似的那些物品，并将这些物品推荐给用户。同样，与已有物品相似的物品被用来生成一个综合得分，而该得分用于评估未知物品的相似度。

如下图所示，展示了基于用户的协同过滤算法的表现形式：



从图中可以看出向用户推荐一个商品那么如何选择这个商品是一个很大的问题。在已有的信息中，用户3已经选择了物品1和物品5，用户2比较偏向于物品2和物品4，而用户1选择了物品1和物品5。那么我们就可以认为用户1和用户3在选择偏好上更加相似。所以完全有理由相信用户1和用户3都选择了相同的物品1和物品5那么向用户3推荐物品3是完全合理的。

而计算用户之间以及物品之间的的相似性可以采取欧式距离或者是余弦角度。以下面的表1是用户的打分表，用这张表来看看用户间的相似性。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 物品1 | 物品2 | 物品3 | 物品4 |
| 用户1 | 1 | 1 | 3 | 1 |
| 用户2 | 1 | 2 | 3 | 2 |
| 用户3 | 2 | 2 | 1 | 1 |

用户1和用户2之间相似度，使用欧式距离计算为:

用户1和用户2的相似度为0.414，用户1和用户3的相似度为：

用户1和用户3的相似度为0.287，从分值上可以看出的分值大于的分值，因此可以得到用户2比用户3更加想似于用户1。 那么在想用户1推荐物品的时候就可以参考用户3所拥有的物品。

而对于不熟悉的用户，在确实特定用户信息的情况下，根据用户已有的偏好数据去推荐一个未知物品是合理的。这就是基于物品的推荐算法。该算法是以已有的物品为线索去进行相似度计算从而推荐给特定的目标用户。如图2所示，展示了基于商品的推荐方法的表现形式。



从图中可以看出，这次同样是给用户3推荐一个物品，在不知道其他用户的情况下，通过计算的方式得出与以购买物品最相近的物品推荐给用户。

**用法：**Spark MLlib中还没有实现协同过滤推荐的算法，需要使用者自己去写。

6.3.2 交替最小二乘法法（ALS算法）

ALS是一种求解矩阵分解问题的最优化方法。它功能强大、效果理想而且被证明相对容易并行化。以用户给产品打分为例，它通过观察到的所有用户给产品的打分来推断每个用户的喜好并向用户推荐合适的产品。

ALS的实现原理是迭代式求解一系列最小二乘回归问题。在每一次迭代时，固定用户因子矩阵或是物品因子矩阵中的一个，然后用固定的这个矩阵以及评级数据来更新另一个矩阵。之后，被更新的矩阵被固定住，再更新另外一个矩阵。如此迭代，直到模型收敛（或是迭代了预设好的次数）。

将所给的矩阵W分解成两个较小的矩阵U和V，即是ALS的算法的关键。在ALS中首先对U和V矩阵随机化生成，之后固定某一个特定对象，去求另一个未随机化的矩阵对象。然后利用被求取的矩阵对象去求随机化矩阵对象。最后两个对象相互迭代计算，求取与实际矩阵差异达到程序设定的最小阈值位置。其具体步骤如下：



**实现**：在Spark的MLlib中可以直接调用ALS算法。MLlib的ALS算法有固定的数据格式：

|  |
| --- |
| case class Rating(user:Int, product:Int, rating:Double) |

其中Rating是固定的ALS输入格式，它要求是一个元组类型的数据，其中的数值分别为[Int,Int,Double]，因此在数据集建立时，用户名和物品名分别用数值代替，而最后的评分没有变化。

ALS数据模型是根据数据集训练获得，即ALS.train方法是最为重要的方法。ALS.train源码如下：

|  |
| --- |
| def train(  ratings:RDD[Rating],  rank:Int,  iterations:Int,  lambda:Double,  blocks:Int,  seed:Long,  ):MatrixFactorizationModel={  new ALS(blocks, blocks, rank, iterations, lambda, false, 1.0, seed).run(ratings)  } |

参数的意义：

* numBlocks:并行计算的block数（-1为自动配置）
* rank:模型中的隐藏因子数
* iterations:算法迭代次数
* lambda:ALS中的正则化参数
* implicitPref:使用显示反馈ALS变量或隐式反馈
* alpha:ALS隐式反馈变化率用于控制每次拟合修正的幅度

6.4 推荐引擎的评价指标

推荐引擎的评价指标可以使用均方差以及k值平均准确率来评价。

（1）均方差：是指所以样本预测到的评级与真实评级的差值的平方的总和。

（2）K值平均准确率（MAPK）：的意思是整个数据集上的K值平均准确率（Average Precision at Kmetric， APK）的均值。 APK是信息检索中常用的一个指标。它用于衡量针对某个查询所返回的 “前K个”文档的平均相关性。对于每次查询，我们会将结果中的前K个与实际相关的文档进行比较。用APK指标计算时，结果中文档的排名十分重要。如果结果中文档的实际相关性越高且排名也更靠前，那APK分值也就越高。由此，它也很适合评估推荐的好坏。因为推荐系统也会计算 “前K个”推荐物，然后呈现给用户。如果在预测结果中得分更高（在推荐列表中排名也更靠前）的物品实际上也与用户更相关，那自然这个模型就更好。 APK和其他基于排名的指标同样也更适合评估隐式数据集上的推荐。这里用MSE相对就不那么合适。当用APK来做评估推荐模型时，每一个用户相当于一个查询，而每一个“前K个”推荐物组成的集合则相当于一个查到的文档结果集合。用户对电影的实际评级便对应着文档的实际相关性。这样， APK所试图衡量的是模型对用户感兴趣和会去接触的物品的预测能力。

全局MAPK的求解要计算对每一个用户的APK得分，再求其平均。这就要为每一个用户都生成相应的推荐列表。