# 机器学习05【机器学习】

# 主要内容

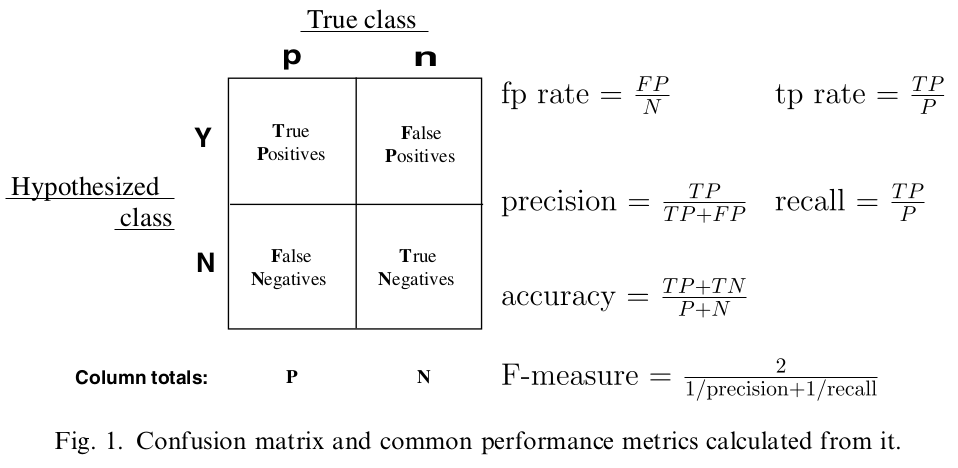
1. 模型评估方式
2. 理解协同过滤思想
3. 理解推荐系统架构
4. 理解推荐系统流程

# 学习目标

|  |  |  |
| --- | --- | --- |
| 节数 | 知识点 | 要求 |
| 第一节 （模型评估） | 混淆矩阵 | 掌握 |
| ROC和AUC | 掌握 |
| 第二节 （决策树与随机森林） | 决策树分类原则 | 掌握 |
| 信息增益 | 掌握 |
| 信息增益率 | 掌握 |
| 随机森林 | 掌握 |
| 第三节 （决策树与随机森林代码） | 决策树代码 | 掌握 |
| 随机森林代码 | 掌握 |
| 第四节（协同过滤） | 基于用户的协同过滤 | 掌握 |
| 基于商品的协同过滤 | 掌握 |
| 第五节（推荐系统介绍） | 推荐系统介绍 | 掌握 |
| 推荐的应用场景 | 掌握 |
| 第六节（推荐系统流程） | 推荐系统架构 | 掌握 |
| 推荐系统的详细流程 | 掌握 |
| 冷启动问题 | 掌握 |
| 第七节（推荐流程-数据模拟） | 模拟数据 | 掌握 |
| 将数导入Hive中 | 掌握 |

# 模型评估

1. **混淆矩阵：**



对以上混淆矩阵的解释：

P：样本数据中的正例数。

N：样本数据中的负例数。

Y：通过模型预测出来的正例数。

N：通过模型预测出来的负例数。

True Positives:真阳性，表示实际是正样本预测成正样本的样本数。

Falese Positives:假阳性，表示实际是负样本预测成正样本的样本数。

False Negatives:假阴性，表示实际是正样本预测成负样本的样本数。

True Negatives:真阴性，表示实际是负样本预测成负样本的样本数。

 :真阳性率(True Positive Rate，TPR)，也叫灵敏度(Sensitivity)，召回率(Recall)。即： ，正确的预测出的正例数占样本中正例总数的比例。真阳性率越大越好，越大代表在正样本中预测为正例的越多。

 :假阳性率(False Positive Rate，FPR)，也叫误诊率。错误的预测出的正例数占样本中负例的比例。假阳性率越小越好，越小代表在负样本中预测为正例的越少。

 :正确率(Precision),也叫精确率， ，通过模型预测出来真正是正例的正例数占模型预测出来是正例数的比例，越大越好。

 :准确率(accuracy)， ,模型预测正确的例数占总样本的比例。越大越好。

举例：假设现在有60个正样本，40个负样本，我们通过模型找出正样本50个，其中40个是真正的正样本，那么上面几个指标如下：

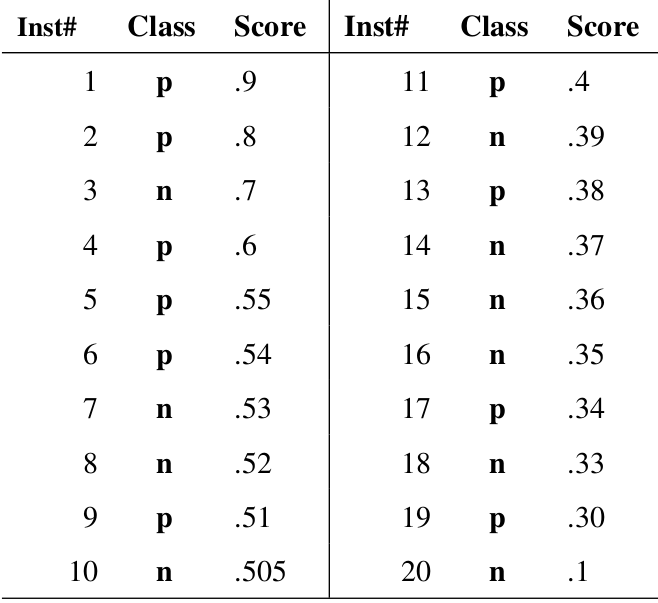
|  |  |
| --- | --- |
| TP=40 | FP=10 |
| FN=20 | TN=30 |

可知，一个模型的TP和TN越大越好。准确率=70/100=70%。精确率=40/50=80%。召回率=40/60=2/3。

1. **ROC和AUC:**

ROC(Receiver Operating Characteristic)曲线和AUC(Area Under the Curve)值常被用来评价一个二值分类器(binary classifier) 的优劣。

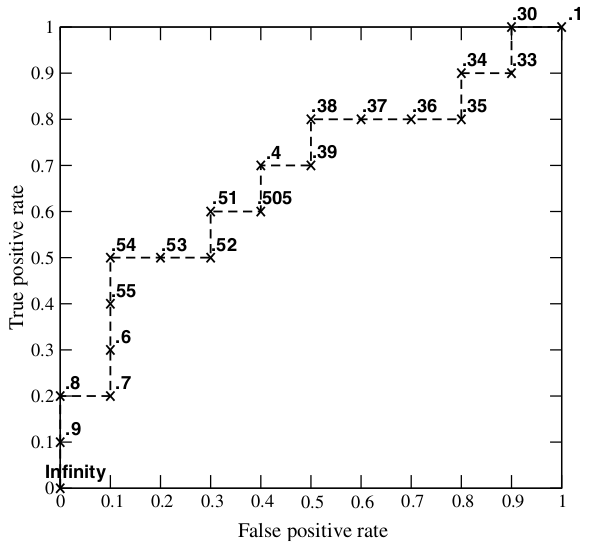
ROC曲线是以假阳性率FPR为横轴，以真阳性率TPR为纵轴的一个曲线图像。图像中的每一点是一个分类阈值，根据一些连续的分类阈值可以得到ROC的图像，如下图：有20个样本，其中真实正例有10个，用p表示，负例有10个，用n表示。Inst# 代表样本编号，Class代表样本真实的类别，Score表示利用模型得出每个测试样本属于真实样本的概率。依次将Score概率从大到小排序，得到下表：



从第一个样本开始直到第20个样本，依次将Score当做分类阈值threshold。当预测测试样本属于正样本的概率大于或等于该threshold时，我们认为该样本是正样本，否则是负样本。

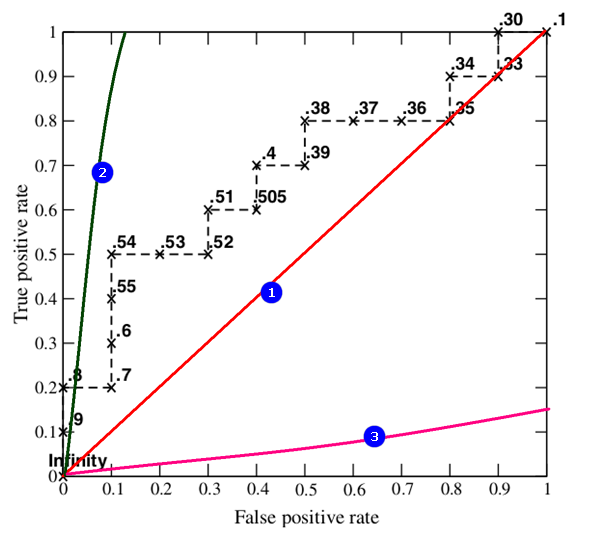
如：拿到第一个样本，该样本真实类别是p,Score=0.9，将0.9看成分类阈值threshold，那么该样本预测是正例，TPR=1/10,FPR=0/10=0，拿到第二个样本，该样本真实类别是p,Score=0.8,将0.8作为threshold,该样本预测是正例，TPR=2/10,FPR=0/10=0 … … 以此类推,当拿到第7个样本时，该样本真实类别是n,Score=0.53，将0.53看成分类阈值threshold，预测为正例，但是预测错误，将本该属于负例的样本预测为正例，那么当阈值为0.53时，共预测7个样本，预测正确的样本标号为1,2,4,5,6。预测错误的样本标号为：3,7。那么此时，TPR=5/10=0.5,FPR=2/10=0.2。

按照以上方式，每选择一个阈值threshold时，都能得出一组TPR和FPR,即ROC图像上的一点。通过以上，可以得到20组TPF和FPR，可以得到ROC图像如下，当threshold取值越多，ROC曲线越平滑。



上图图像当样本真实类别为正例时，模型预测该样本为正例那么图像向上画一步（TPR方向）。如果该样本真实类别是负例，模型预测该样本为正例那么图像向右画一步（FPR方向）。

下图中，如果ROC的图像是通过（0,0）点和（1.1）点的一条直线也就是①线，那么当前模型的预测能力是0.5，即：模型在预测样本时，预测对一次，预测错一次，会形成①曲线。如果ROC曲线是②线，那么该模型预测数据的真阳性率大于假阳性率，也就是模型预测对的次数多，预测错的次数少，模型越好。当模型的ROC曲线为③线时，模型的假阳性率比真阳性率大，模型预测错的次数多，预测对的次数少，还不如随机瞎蒙的概率0.5。综上所述，ROC的曲线越是靠近纵轴，越陡，该模型越好。那么如何根据ROC来量化评价一个模型的好坏，这就要用到AUC面积。



AUC面积是ROC曲线与横轴（假阳性率，FPR）围成的面积,也就是曲线下方的面积。AUC面积越大越好，代表模型分类效果更准确。

计算AUC的公式：



其中， 是属于正例的样本。M：测试样本中的正例数。N：测试样本中的负例数。 代表将测试样本（正例和负例都有）中的Score值按照正序排序，找到样本属于正例的索引号累加和。

AUC=1，完美的分类器，采用这个预测模型时，不管设定什么样的阈值都能正确的预测结果。绝大多数情况下，不存在这种分类器。

0.5<AUC<1,优于随机猜测，可以调节分类阈值，使AUC越靠近1，模型效果越好。

AUC=0.5,和随机分类一样，就是随机瞎蒙，模型没有预测价值。

AUC<0.5,比随机分类还差，大多数情况下成功避开了正确的结果。

AUC这种评估方式较计算准确率的评估方式更好。假设有两个模型M1与M2,两个模型的准确率都是80%，假设默认阈值0.5时，M1模型预测正例的概率多数位于0.51左右，但不小于0.5。M2模型预测正例的概率多数位于0.9附近，那么同样是80%的正确率下，M2模型将结果预测的更彻底，反映到AUC面积中，M2中预测正例的概率多数位于0.9左右，对应的 比较大，相应的AUC值比较大，而M1的AUC相对较小。所以AUC这种评估模型的方式更能说明模型好有多好，能更好的计算模型的纯度。

# 决策树和随机森林

决策树和随机森林都是非线性有监督的分类模型。

决策树是一种树形结构，树内部每个节点表示一个属性上的测试，每个分支代表一个测试输出，每个叶子节点代表一个分类类别。通过训练数据构建决策树，可以对未知数据进行分类，

随机森林是由多个决策树组成，随机森林中每一棵决策树之间没有关联，在得到一个随机森林后，当有新的样本进入的时候，随机森林中的每一棵决策树分别进行判断，分析出该样本属于哪一类，然后最后看哪一类被选择最多，就预测该样本属于这一类。

1. 认识决策树

术语：

根节点：最顶层的分类条件

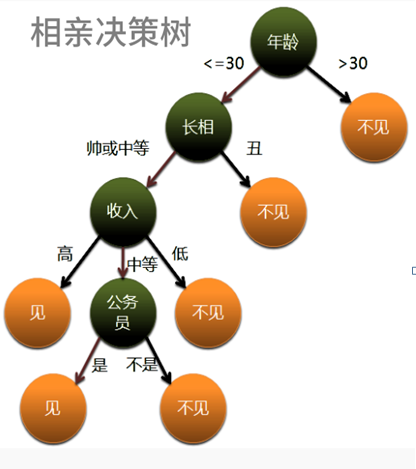
叶节点：代表每一个类别号

中间节点：中间分类条件

分支：代表每一个条件的输出

二叉树：每一个节点上有两个分支

多叉树：每一个节点上至少有两个分支



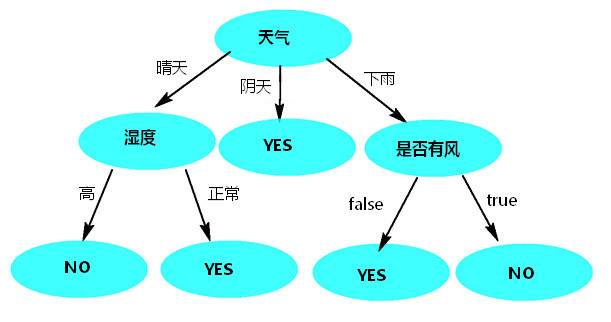
1. 决策树分类原则

如下图数据集：

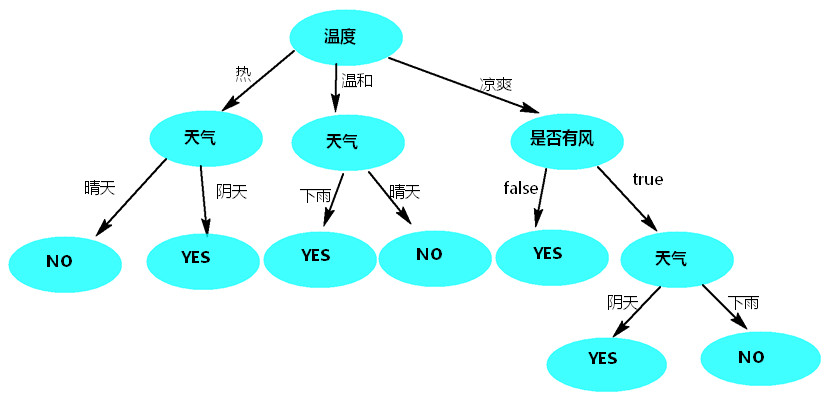


要按照前4列的信息，使用决策树预测车祸的发生，如何选择根节点呢？

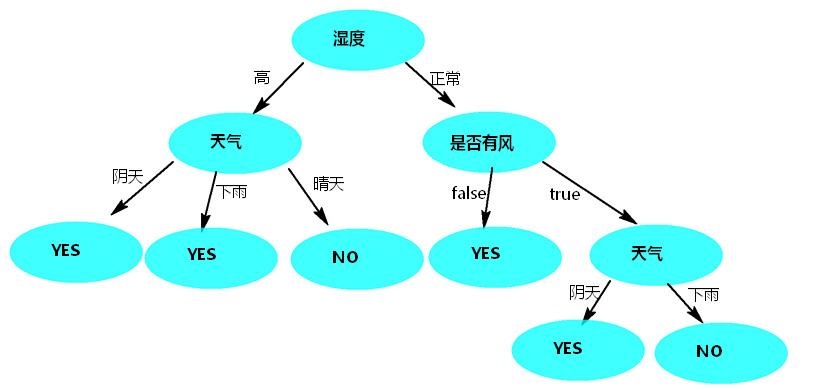
按照“天气”列作为根节点，使用决策树预测，如图：



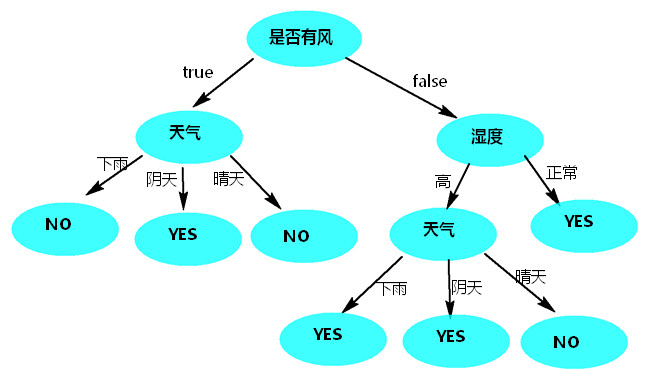
按照“温度”列作为根节点，使用决策树预测，如图：



按照“湿度”列作为根节点，使用决策树预测，如图：



使用“风”列作为根节点，使用决策树预测，如图：

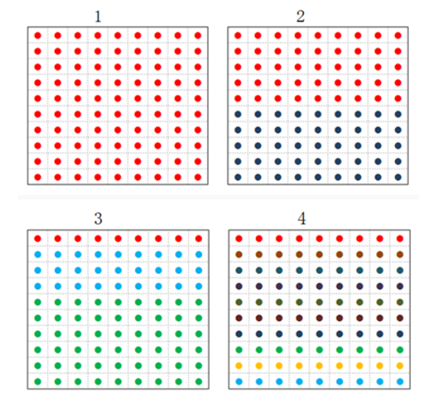


通过以上发现，只有使用天气作为根节点时，决策树的高度相对低而且树的两边能将数据分类的更彻底（其他列作为根节点时，树两边分类不纯粹，都有天气）。

**决策树的生成原则**：数据不断分裂的递归过程，每一次分裂，尽可能让类别一样的数据在树的一边，当树的叶子节点的数据都是一类的时候，则停止分类。这样分类的数据，每个节点两边的数据不同，将相同的数据分类到树的一侧，能将数据分类的更纯粹。减少树的高度和训练决策树的迭代次数。**注意**：训练决策树的数据集要离散化，不然有可能造成训练出来的树有些节点的分支特别多，容易造成过拟合。

1. 选择分类条件

下图：



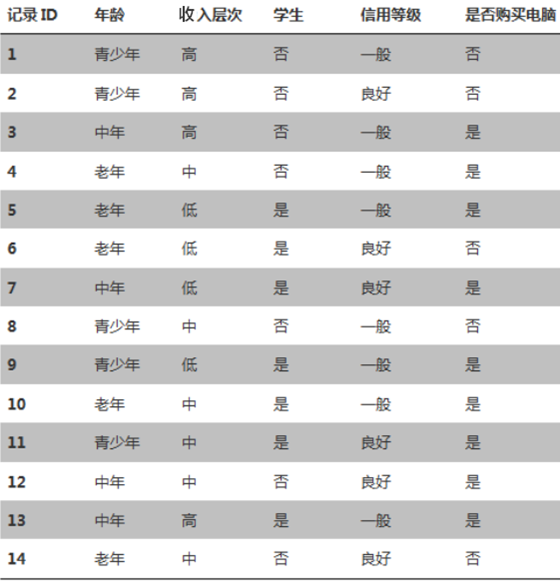
上图中箱子①中有100个红球。箱子②中有50个红球和50个黑球。箱子③中有10个红球和30个篮球，60个绿球。箱子④中各个颜色均有10中球。发现箱子①中球类单一，信息量少，比较纯粹，箱子④中，球的类别最多，相对①来说比较混乱，信息量大。

如何量化以上每个箱子中信息的纯粹和混乱（信息量的大小）指标，可以使用信息熵或者基尼系数。

**1). 信息熵：**信息熵是香农在1948年提出来量化信息信息量的指标，熵的定义如下：



其中，n代表当前类别有多少类别，代表当前类别中某一类别的概率。例如下图，



计算“是否购买电脑”这列的信息熵，当前类别“是否购买电脑”有2个类别，分别是“是”和“否”，那么“是否购买电脑”类别的信息熵如下：



通过以上计算可以得到，某个类别下信息量越多，熵越大，信息量越少，熵越小。假设“是否购买电脑”这列下只有“否”这个信息类别，那么“是否购买电脑”这列的信息熵为：



上图中，如果按照“年龄”，“收入层次”，“学生”，“信用等级”列使用决策树来预测“是否购买电脑”。如何选择决策树的根节点分类条件，就是找到某列作为分类条件时，使“是否购买电脑”这列分类的更彻底，也就是找到在某个列作为分类条件下时，“是否购买电脑”信息熵相对于没有这个分类条件时信息熵降低最大（降低最大，就是熵越低，分类越彻底），这个条件就是分类节点的分类条件。这里要使用到条件熵和信息增益。

**条件熵**：在某个分类条件下某个类别的信息熵叫做条件熵，类似于条件概率，在知道Y的情况下，X的不确定性。条件熵一般使用 表示，代表在Y条件下，X的信息熵。上图中假设在“年龄”条件下，“是否购买电脑”的信息熵为：“年龄”列每个类别下对应的“是否购买电脑”信息熵的和。

H(是否购买电脑|年龄)=H(是否购买电脑|青少年)+H(是否购买电脑|中年)+H(是否购买电脑|老年)

所以：

**信息增益**：代表熵的变化程度。分类前的信息熵减去分类后的信息熵。如特征Y对训练集D的信息增益为。

在“年龄”条件下，“是否购买电脑”的信息增益为：

g(是否购买电脑，年龄)=H(是否购买电脑)-H(是否购买电脑,年龄)

=0.94-0.69=0.25

由以上可知，按照“记录ID”，“年龄”，“收入层次”，“学生”，“信用等级”列使用决策树来预测“是否购买电脑”，选择分类根分类条件时步骤：

a.计算“是否购买电脑”的信息熵

b.计算在已知各个列的条件熵

H(是够购买电脑|年龄)，H(是够购买电脑|收入层次)，H(是够购买电脑|是否学生)，H(是够购买电脑|信用等级)

c.求各个条件下的信息增益，选择信息增益大的作为分类条件。选择中间节点时，以此类推。

在构建决策树时，选择信息增益大的属性作为分类节点的方法也叫ID3分类算法。

**2).基尼系数**：基尼系数也可以表示样本的混乱程度。公式如下：



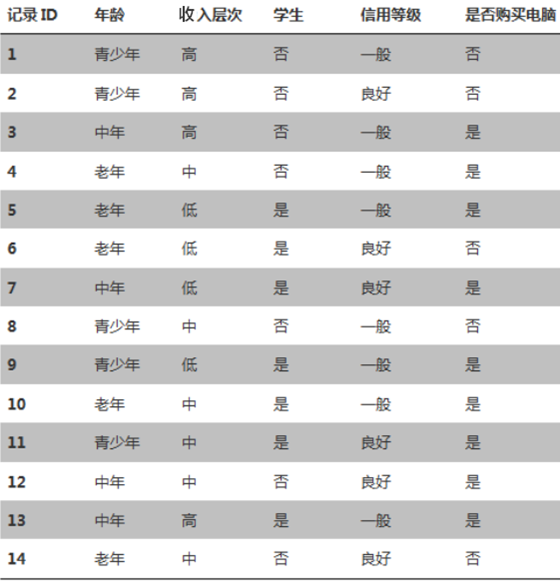
其中，k代表当前类别有K个类别。代表当前类别中某一类别的概率，1-代表不是当前这个类别的概率。

上图中计算“是否购买电脑”这列的基尼系数：



基尼系数越小代表信息越纯，类别越少，基尼系数越大，代表信息越混乱，类别越多。基尼增益的计算和信息增益相同。假设某列只有一类值，这列的基尼系数为0。

1. 信息增益率



在上图中，如果将“记录ID”也作为分类条件的话，由于“记录ID”对于“是否购买电脑”列的条件熵为0，可以得到“是否购买电脑”在“记录ID”这个分类条件下信息增益最大。如果选择“记录ID”作为分类条件，容易造成分支特别多，对已有记录ID的数据可以分类出结果，对于新的记录ID有可能不能成功的分类出结果。

使用信息增益来筛选分类条件，更倾向于选择更混杂的属性。容易出现过拟合问题。可以使用信息增益率来解决这个问题。

信息增益率的公式：gr(D,A) = g(D,A)/H(A)，在某个条件下信息增益除以这个条件的信息熵。

例如:在“记录ID”条件下，“是否购买电脑”的信息增益最大，信息熵H(记录ID)也比较大，两者相除就是在“记录ID”条件下的信息增益率，结果比较小，消除了当某些属性比较混杂时，使用信息增益来选择分类条件的弊端。使用信息增益率来构建决策树的算法也叫C4.5算法。一般相对于信息增益来说，选择信息增益率选择分类条件比较合适。

如果决策树最后一个条件依然没能将数据准确分类，那么在这个节点上就可以使用概率来决定。看看哪些情况出现的多，该情况就是该节点的分类结果。

1. 使用决策树来做回归或者预测值



如上图，使用学历、收入、身高、行业使用决策树来预测收到的邮件数。可以将邮件数分为几类（也可以按照其他列，将邮件数分类），比如邮件数<=23封属于A类，邮件数大于23<邮件数<=30为B类，A类中取邮件的平均数，B类中也取邮件的平均数。就是可以将某些列作为分类条件划分邮件数的类别，再取邮件数的平均数，这样可以使用决策树来预测大概值的范围。

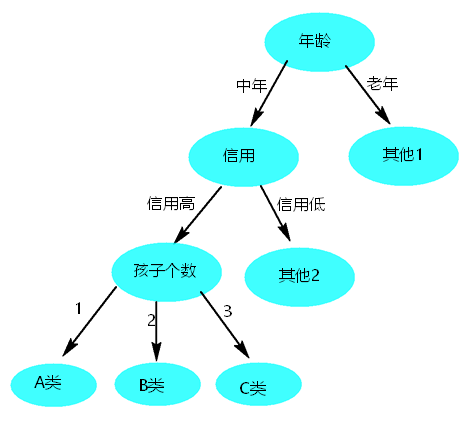
1. 决策树预剪枝和后剪枝

决策树对训练集有很好的分类能力，但是对于未知的测试集未必有好的分类能力，导致模型的泛化能力弱，可能发生过拟合问题，为了防止过拟合问题的出现，可以对决策树进行剪枝。剪枝分为预剪枝和后剪枝。

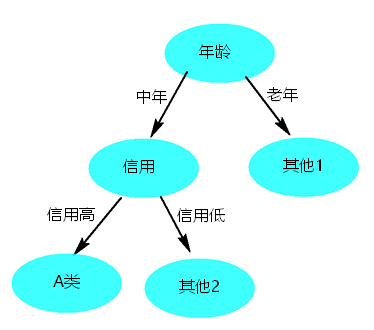
预剪枝：就是在构建决策树的时候提前停止。比如指定树的深度最大为3，那么训练出来决策树的高度就是3,预剪枝主要是建立某些规则限制决策树的生长，降低了过拟合的风险，降低了建树的时间，但是有可能带来欠拟合问题。

后剪枝：后剪枝是一种全局的优化方法，在决策树构建好之后，然后才开始进行剪枝。后剪枝的过程就是删除一些子树，这个叶子节点的标识类别通过大多数原则来确定，即属于这个叶子节点下大多数样本所属的类别就是该叶子节点的标识。选择减掉哪些子树时，可以计算没有减掉子树之前的误差和减掉子树之后的误差，如果相差不大，可以将子树减掉。一般使用后剪枝得到的结果比较好。

剪枝可以降低过拟合问题，如下图：



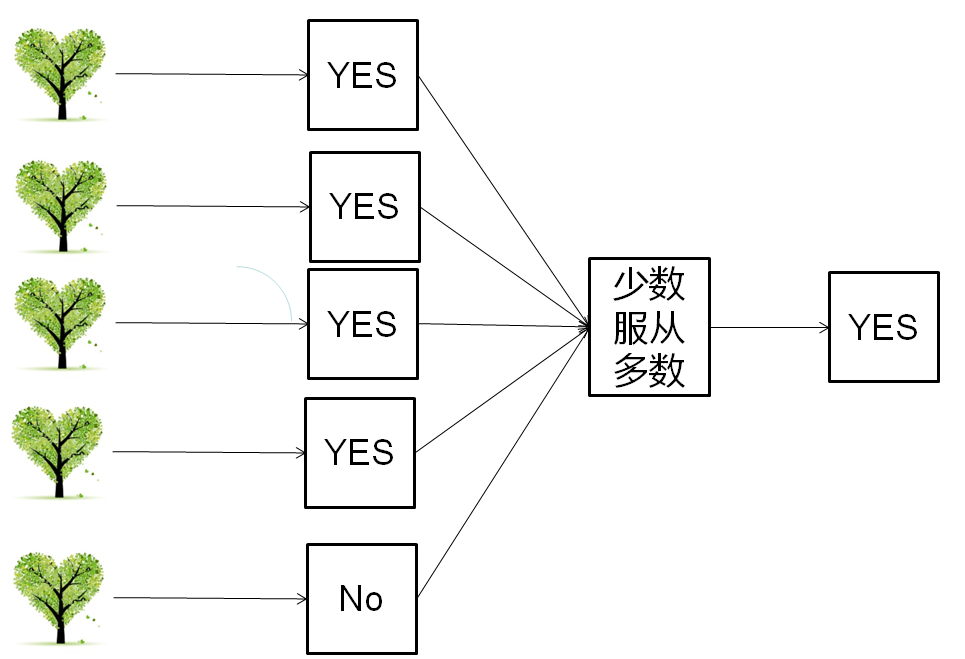
当来一条数据年龄为中年，信用高，孩子个数是4个时，没有办法分类。可以通过剪枝，降低过拟合问题。



1. 随机森林

随机森林是由多个决策树组成。是用随机的方式建立一个森林，里面由很多决策树组成。随机森林中每一棵决策树之间都是没有关联的。得到随机森林之后，对于一个样本输入时，森林中的每一棵决策树都进行判断，看看这个样本属于哪一类，最终哪一类得到的结果最多，该输入的预测值就是哪一类。

随机森林中的决策树生成过程是对样本数据进行行采样和列采样，可以指定随机森林中的树的个数和属性个数，这样当训练集很大的时候，随机选取数据集的一部分，生成一棵树，重复上面过程，可以生成一堆形态各异的树，这些决策树构成随机森林。



随机森林中的每个决策树可以分布式的训练，解决了单棵决策树在数据量大的情况下预算量大的问题。当训练样本中出现异常数据时，决策树的抗干扰能力差，对于随机森林来说也解决了模型的抗干扰能力。

# 决策树和随机森林代码

1. **决策树：**

|  |
| --- |
| **object** ClassificationDecisionTree {  **def** main(args: Array[*String*]): Unit = {  **val** conf = **new** SparkConf()  conf.setAppName("analysItem")  conf.setMaster("local[3]")  **val** sc = **new** SparkContext(conf)  **val** data = MLUtils.loadLibSVMFile(sc, "汽车数据样本.txt")  // Split the data into training and test sets (30% held out for testing)  **val** splits = data.randomSplit(Array(0.7, 0.3))  **val** (trainingData, testData) = (splits(0), splits(1))  //指明类别  **val** numClasses=2  //指定离散变量，未指明的都当作连续变量处理  //1,2,3,4维度进来就变成了0,1,2,3  //这里天气维度有3类,但是要指明4,这里是个坑,后面以此类推  **val** categoricalFeaturesInfo=Map[Int,Int](0->4,1->4,2->3,3->3)  //设定评判标准 "gini"/"entropy"  **val** impurity="entropy"  //树的最大深度,太深运算量大也没有必要 剪枝 防止模型的过拟合！！！  **val** maxDepth=3  //设置离散化程度,连续数据需要离散化,分成32个区间,默认其实就是32,分割的区间保证数量差不多 这个参数也可以进行剪枝  **val** maxBins=32  //生成模型  **val** model =DecisionTree.trainClassifier(trainingData,numClasses,categoricalFeaturesInfo,impurity,maxDepth,maxBins)  //测试  **val** labelAndPreds = testData.map { point =>  **val** prediction = model.predict(point.features)  (point.label, prediction)  }  **val** testErr = labelAndPreds.filter(r => r.\_1 != r.\_2).count().toDouble / testData.count()  println("Test Error = " + testErr)  println("Learned classification tree model:\n" + model.toDebugString)  }  } |

1. **随机森林：**

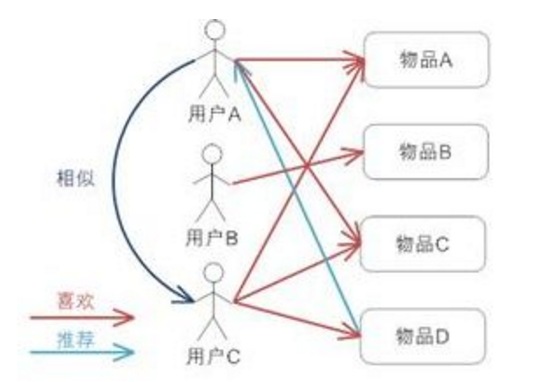
|  |
| --- |
| **object** ClassificationRandomForest {  **def** main(args: Array[*String*]): Unit = {  **val** conf = **new** SparkConf()  conf.setAppName("analysItem")  conf.setMaster("local[3]")  **val** sc = **new** SparkContext(conf)  //读取数据  **val** data = MLUtils.loadLibSVMFile(sc,"汽车数据样本.txt")  //将样本按7：3的比例分成  **val** splits = data.randomSplit(Array(0.7, 0.3))  **val** (trainingData, testData) = (splits(0), splits(1))  //分类数  **val** numClasses = 2  // categoricalFeaturesInfo 为空，意味着所有的特征为连续型变量  **val** categoricalFeaturesInfo =Map[Int, Int](0->4,1->4,2->3,3->3)  //树的个数  **val** numTrees = 3  //特征子集采样策略，auto 表示算法自主选取  //"auto"根据特征数量在4个中进行选择  // 1,all 全部特征 2,sqrt 把特征数量开根号后随机选择的 3,log2 取对数个 4,onethird 三分之一  **val** featureSubsetStrategy = "auto"  //纯度计算 "gini"/"entropy"  **val** impurity = "entropy"  //树的最大层次  **val** maxDepth = 3  //特征最大装箱数,即连续数据离散化的区间  **val** maxBins = 32  //训练随机森林分类器，trainClassifier 返回的是 RandomForestModel 对象  **val** model = RandomForest.trainClassifier(trainingData, numClasses, categoricalFeaturesInfo,  numTrees, featureSubsetStrategy, impurity, maxDepth, maxBins)  //打印模型  println(model.toDebugString)  //保存模型  //model.save(sc,"汽车保险")  //在测试集上进行测试  **val** count = testData.map { point =>  **val** prediction = model.predict(point.features)  // Math.abs(prediction-point.label)  (prediction,point.label)  }.filter(r => r.\_1 != r.\_2).count()  println("Test Error = " + count.toDouble/testData.count().toDouble)  println()  }  } |

# 协同过滤

协同过滤（Collaborative Filtering）技术，是推荐系统中应用最为广泛的技术之一，协同过滤算法主要有两种，一种是基于用户的协同过滤算法(UserBaseCF)，另一种是基于物品的协同过滤算法(ItemBaseCF）。

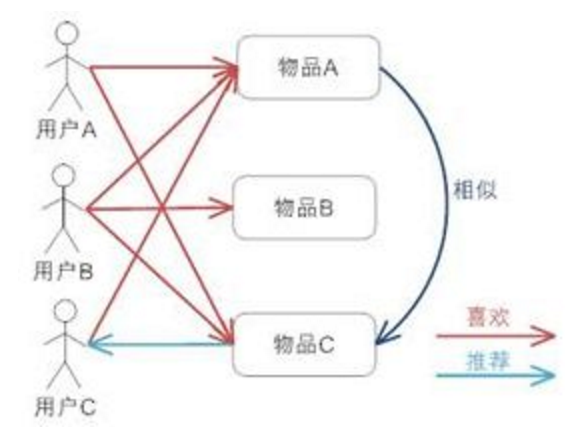
1. **基于用户的协同过滤**

基于用户的协同过滤：跟你喜好相似的人喜欢的东西你也很有可能喜欢。如图假设用户A喜欢商品A，C。用户B喜欢商品B。用户C喜欢商品A,C,D。我们可以发现用户A和C的行为和偏好是比较类似的。用户C喜欢物品D，那么就可以将物品D推荐给用户A。



1. **基于商品的协同过滤**

基于商品的协同过滤：跟你喜欢的东西类似的东西你也可能喜欢。用户A喜欢商品A和C，用户B喜欢商品A,B,C。用户C喜欢物品A。从所有用户的历史喜好中假设商品A与商品C比较类似，也就是喜欢商品A的都喜欢商品C，那么基于这个结论我们可以将商品C推荐给用户C。 多数情况下基于商品的协同过滤经常使用。



# App推荐系统

1. **app推荐系统介绍**

个性化商品营销和信息过载是推荐系统产生的根本原因。app推荐的最根本就是根据用户历史下载的App去找到与这个App相似的app，推荐给该用户。如何找到与此App相似的app?这就需要基于所有用户的下载历史计算挖掘出基于物品相似的矩阵。

app推荐系统是基于用户的隐式转换数据和基于商品的协同过滤来实现app个性化推荐。当用户登录应用商店时，根据之前用户的行为信息来推荐对应感兴趣的app。

思考：假设自己实现一个商品推荐系统需要什么样的数据？

商品的基本信息、用户商品购买记录、用户对购买商品的评价。

1. **推荐的应用场景**

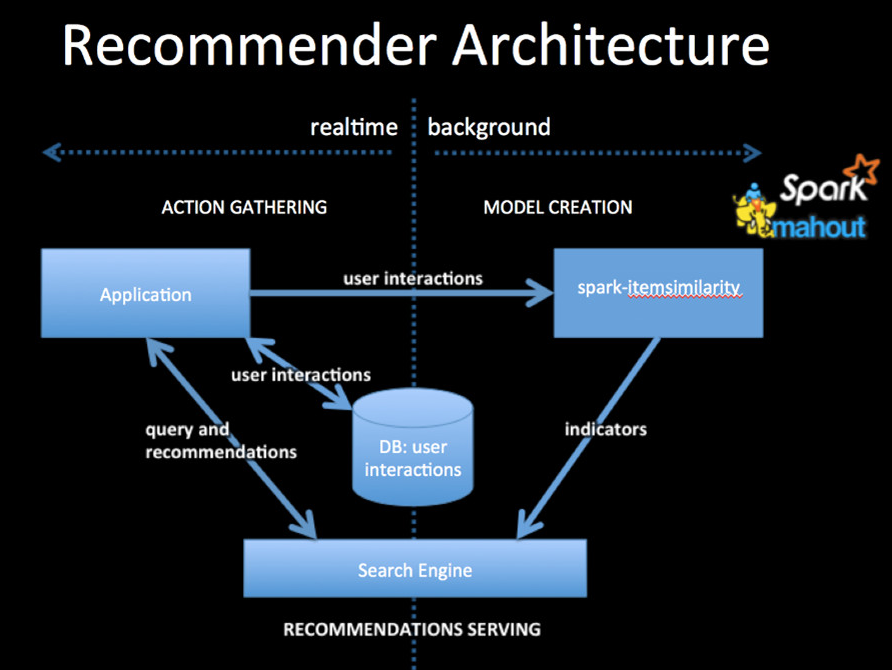
推荐的应用场景有很多种，例如：商品信息推荐，新闻推荐、app推荐、好友推荐、音乐推荐、广告推荐等。以上场景都是推荐系统的应用场景。

# 推荐系统流程

1. **推荐系统架构**

推荐系统中一般都会分为两部分，一部分是离线训练模型，另一部分是在线使用模型推荐。离线训练模型需要经过数据采集、清洗、特征抽取等步骤，然后训练模型，这里训练模型一般是周期性训练模型，比如一天训练一次，或者一周训练一次，因为模型的数据不同，训练出来模型的效果不同，将模型训练好之后，可以将模型保存到HDFS中。在线使用模型时，从HDFS中加载模型，将对应的实时数据获取过来组织特征，然后带入模型使用，得到对应的推荐结果即可。

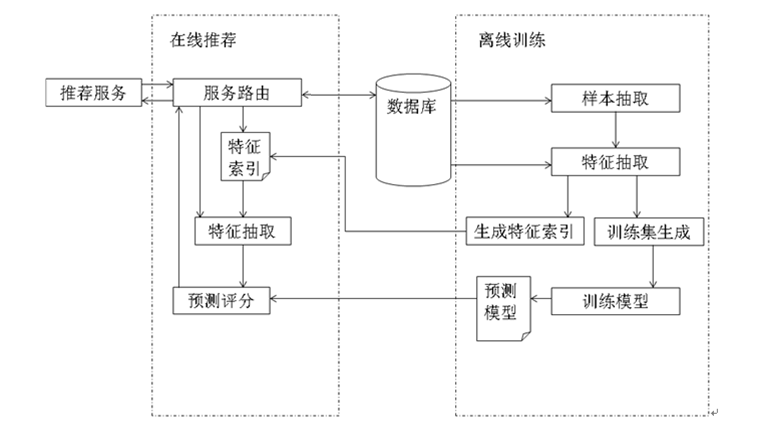
一般在做推荐系统时使用的架构如下图，左侧为推荐系统在线推荐部分，右侧为推荐系统模型离线训练部分，另外最下层是实时数据与离线数据结合部分。过程如下：



离线部分：当用户访问application时，将用户的行为数据收集起来放入Hbase或者Hive中，基于此数据构建训练集，离线训练推荐系统模型，将训练好的模型参数保存到Redis或者模型文件中，待使用。

在线部分：当用户访问application时，根据用户的基本信息去数据库中拿到用户的最近行为数据，基于模型数据实时计算出用户的推荐列表。

1. **App推荐系统架构**



离线训练：

1. 在准备训练集时，如果数据是放在HBase中，数据量太大，可以通过Hive映射，随机样本抽取一部分数据当做训练数据的源数据，只要抽取的数据能代表整体即可，这样即增大了构成的源数据集的时间跨度又可能使训练模型的时间会大大缩短。如果全量数据不多，也可以将所有数据直接当做源数据集。
2. 如果Hive或者Hbase中的数据列太多，可以对数据进行特征抽取，构成训练模型的训练集。同时将所有的特征保存到特征文件中构成特征索引，供推荐系统在线推荐使用。
3. 训练集准备好之后，对于训练集中的每条数据来说就是用户针对当前App的行为信息，结果上来看，要么下载了App，要么没有下载App,这就是二分类问题，可以使用逻辑回归结合训练集训练模型。
4. 当模型训练好之后，就是一些特征参数，可以将模型保存到Redis或者文件中，供实时推荐部分调用。

在线推荐：

1. 用户访问系统时，通过服务路由可以拿到用户的id，根据用户id首先去数据库中拿到当前用户的一些基础数据信息，将基础数据结合特征文件将有用的特征过滤出来，进行基础数据的格式化。
2. 加载模型文件数据，将用户格式化后的特征数据带入模型文件得到推荐列表。
3. **App推荐系统使用数据**

App推荐系统使用的数据应该包括如下：

1. 要推荐物品或内容的元数据，例如关键字，基本描述等。
2. 用户对物品或者信息的偏好，根据应用本身的不同，可能包括用户对物品的评分，用户查看物品的记录，用户的购买记录等。其实这些用户的偏好信息可以分为两类：

* 显式的用户反馈：这类是用户在网站上自然浏览或者使用网站以外，显式的提供反馈信息，例如用户对物品的评分，或者对物品的评论。
* 隐式的用户反馈：这类是用户在使用网站时产生的数据，隐式的反应了用户对物品的喜好，例如用户购买了某物品，用户查看了某物品的信息等等。

基于以上数据分析，App推荐系统使用的数据有：

App基本信息表、用户app历史下载表、用户浏览app信息表。

1. **推荐系统详细流程**

想要为用户推荐App，按照协同过滤的思想，要构建所有App的同现矩阵，还要知道每个用户对每个App的评分，这里无法获取用户对app的评分。我们根据逻辑回归模型使用app与app关联当做特征，基于用户浏览app信息表和用户历史下载表来基于所有用户数据构建出app之间的关联权重（特征的权重分值当做app与app关联的重要程度）。进而计算出每个App值得推荐的总分值，按照分值从大到小排序，将分值大的推荐给用户即可。

这里我们不可能拿到用户对所有App的评分。但是我们可以对用户下载或者浏览的每个App与该用户历史下载的每个App,构建不同App的同现情况，把同现情况当做特征维度，然后根据用户是否下载了该App（是或否），针对二分类问题，使用逻辑回归计算每两个App同现的权重，也就是每个特征维度的权重值。两个App同现的权重越大，说明两个App的关联性越强，比如：下载A应用的用户，都下载了B应用，而极少数用户下载B的同时还下载了C应用，那么AB同现的次数远远大于AC同现的次数，可以针对下载A的用户，推荐应用B。反映到逻辑回归中就是AB特征的权重大于AC特征的权重。

我们可以针对不同的用户将商家有的每个App与该用户下载历史App构建App同现特征，然后去训练好的模型中取对应维度的权重值累加，最终计算出每个App值得为该用户推荐的分值，进而得到推荐列表。

当一个新用户登录系统后，没有App的历史下载记录，那么就没有办法构建每个App与历史下载App的同现特征，就没有办法为用户计算出推荐列表，这就是推荐系统中的冷启动问题。

**冷启动**：当新用户登录系统后，没有办法生成对应的推荐列表。

如何解决冷启动问题？

在构建训练集时，不仅构建每个App与当前用户的下载历史App的同现特征，还要构建针对每个App各个维度的基本特征。这样训练出来的模型，当用户是新用户时，可以基于App本身的特征权重为用户推荐App列表，解决冷启动问题。

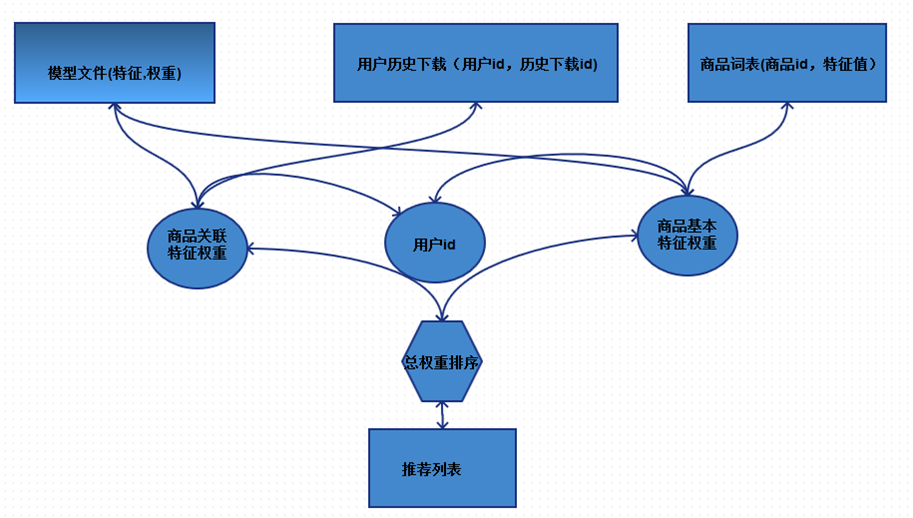
* 离线训练模型：

根据用户的下载或者浏览历史可以拿到用户对App的操作详细信息，比如：用户是否下载了该App、用户手机是否支付了该App、当前用户的手机型号、当前用户浏览或者下载的App的大小、版本号、设计者、评价星级、下载数量等信息。将这些信息当做App的基本特征。

根据用户下载和浏览每个App的历史，与该用户的历史下载App列表形成每个关联特征，作为App的关联特征。

综合上面基本特征和关联特征当做训练集的features，是否下载App作为训练集的Label，使用逻辑回归离线训练模型。

* 在线使用模型：



用户登录系统之后，根据用户ID，获取当前用户历史下载App列表，将商店中每个App与用户下载列表中的每个App构成关联特征，同时每个App还有基本特征，得到关联特征和基本特征之后可以去训练好的模型文件中拿到相应特征对应的权重值，将每个关联特征权重累加得到关联特征的总值（这就是根据用户偏好得到的当前App值得推荐给该用户的分值），将基本特征权重累加得到基本特征的总值，最后将关联特征总值和基本特征总值相加得到一个总分值，该分值就是该App值得推荐给该用户的推荐值。

商店中的每个App最终都会计算出一个值得为当前用户推荐的总分值，再按照分值取前N大，取出最值得推荐给该用户的App，构成推荐列表。

注意：在训练模型构建“商品基本特征”维度时，可能有“设计者”，“手机型号”等这种非数字化的可分类的文本特征属性（比如“性别”维度下有男女两类，“职业”类别下有老师、学生、工人等可分类的数据），由于模型只能使用数字化的数据来训练，可以对训练集含有这种可分类的文本特征属性自己定义一个特征变换规则：如果男用1表示，女用0表示。老师用0表示，学生用1表示，工人用2表示，那么使用模型时，相对应的将测试数据中当前维度的文本特征属性也要按照相对应的同一个特征变换规则来变换，对最终的结果没有影响。这里对文本特征属性设置不同的数字来表示不同的类别时，如果设置的值的跳度大（比如男用1表示，女用1000来表示），影响的是训练模型中按照梯度下降寻找最优解的迭代次数。

# 推荐流程-数据模拟

推荐整体从数据处理开始，默认数据从关系型数据到每天增量导入到hive，在hive中通过中间表和调用python文件等一系列操作，将数据处理为算法数学建模的入口数据，这里只是模拟一下，所以用一个scala文件产生所有准备数据，并直接load到hive中去做数据处理。

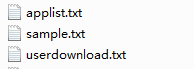
数据处理完以后开始数学建模，通过recommend.scala文件对逻辑回归算法的调用，产生模型文件，将三个模型文件加载到Redis中，启动项目，访问测试。整个过程默认已经有hive环境，intellij idea的环境，并且可以执行scala文件。

整体流程如下：

Scala文件产生数据🡺load到hive，处理数据🡺recommond.scala调用逻辑回归算法计算模型，生成模型文件🡺将模型文件加载到Redis中，运行项目🡺测试

1. 数据模拟

通过DataGenerator类创建数据，参见附件DataGenerator.scala文件，传入参数两个，数据条数和输出目录。会生成对应的三个文件：



1. Hive建表

真实的生产场景涉及到大概五十张表的字段，这里全部简化流程，直接给出最终的三张表：

1. App基本信息表
2. 用户App历史下载表
3. 正负例样本表(用户浏览app下载表)

**建表语句：**

**App基本信息表：**

* 1. CREATE EXTERNAL TABLE IF NOT EXISTS dim\_rcm\_hitop\_id\_list\_ds
  2. (
  3. hitop\_id STRING,
  4. name STRING,
  5. author STRING,
  6. sversion STRING,
  7. ischarge SMALLINT,
  8. designer STRING,
  9. font STRING,
  10. icon\_count INT,
  11. stars DOUBLE,
  12. price INT,
  13. file\_size INT,
  14. comment\_num INT,
  15. screen STRING,
  16. dlnum INT
  17. )row format delimited fields terminated by '\t';

**用户App历史下载表：**

1. CREATE EXTERNAL TABLE IF NOT EXISTS dw\_rcm\_hitop\_userapps\_dm
2. (
3. device\_id STRING,
4. devid\_applist STRING,
5. device\_name STRING,
6. pay\_ability STRING
7. )row format delimited fields terminated by '\t';

**正负例样本表：**

1. CREATE EXTERNAL TABLE IF NOT EXISTS dw\_rcm\_hitop\_sample2learn\_dm
2. (
3. label STRING,
4. device\_id STRING,
5. hitop\_id STRING,
6. screen STRING,
7. en\_name STRING,
8. ch\_name STRING,
9. author STRING,
10. sversion STRING,
11. mnc STRING,
12. event\_local\_time STRING,
13. interface STRING,
14. designer STRING,
15. is\_safe INT,
16. icon\_count INT,
17. update\_time STRING,
18. stars DOUBLE,
19. comment\_num INT,
20. font STRING,
21. price INT,
22. file\_size INT,
23. ischarge SMALLINT,
24. dlnum INT
25. )row format delimited fields terminated by '\t';
26. 加载数据

分别向三张表加载数据：

* 1. 商品词表：
  2. load data local inpath '/root/test/applist.txt' into table dim\_rcm\_hitop\_id\_list\_ds;
  3. 用户历史下载表：
  4. load data local inpath '/root/test/userdownload.txt' into table dw\_rcm\_hitop\_userapps\_dm;
  5. 正负例样本表：
  6. load data local inpath '/root/test/sample.txt' into table dw\_rcm\_hitop\_sample2learn\_dm;

# 本节作业

1. ROC和AUC模型评估方式。
2. 理解协同过滤的思想
3. 理解app推荐架构和原理