数据分析几类经典方法

李千目

K-Nearest Neighbor算法

思想：**通过找事物属性上的相似去揣测类别上的相同**。

简单来说，就是类比法。

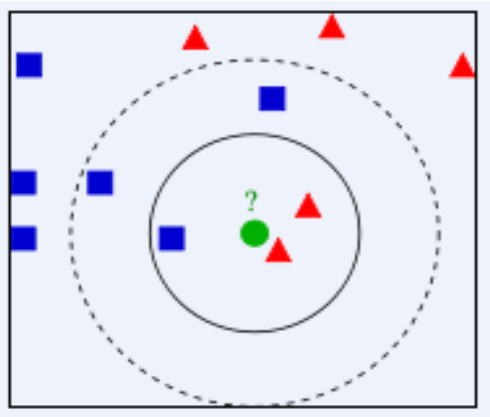
K-Nearest Neighbor从**训练数据集**里面找出和**待分类的数据对象**最相近的K个对象。这K个对象中出现次数最多的那个类别，就可以被当做这个待分类的数据对象的类别。

所谓K近邻算法，即是给定一个训练数据集，对新的输入实例，在训练数据集中找到与该实例最邻近的K个实例（也就是上面所说的K个邻居）， 这K个实例的多数属于某个类，就把该输入实例分类到这个类中。

如果一个样本在特征空间中的k个最相似(即特征空间中最邻近)的样本中的大多数属于某一个类别，则该样本也属于这个类别。

例如，我们对一个植物样本进行分类的时候，通过对比这个样本和实验室的标本之间的各个特征，例如长度、直径、颜色等属性，然后找到最相似的几个标本。这几个标本的类别就应该是我们手上这个植物样本的类别。如果这几个标本的类别也不同，我们取里面出现次数最多的那个类别。

K-Nearest Neighbor算法



如图所示，有两类不同的样本数据，分别用蓝色的小正方形和红色的小三角形表示，而图正中间的那个绿色的圆所标示的数据则是待分类的数据。也就是说，现在， 我们不知道中间那个绿色的数据是从属于哪一类（蓝色小正方形or红色小三角形），下面，我们就要解决这个问题：给这个绿色的圆分类。

好，从它的邻居下手。但一次性看多少个邻居呢？

如果K=3，绿色圆点的最近的3个邻居是2个红色小三角形和1个蓝色小正方形，少数从属于多数，基于统计的方法，判定绿色的这个待分类点属于红色的三角形一类。

如果K=5，绿色圆点的最近的5个邻居是2个红色三角形和3个蓝色的正方形，还是少数从属于多数，基于统计的方法，判定绿色的这个待分类点属于蓝色的正方形一类。

K-Nearest Neighbor算法

当无法判定当前待分类点是从属于已知分类中的哪一类时，我们可以依据统计学的理论看它所处的位置特征，衡量它周围邻居的权重，而把它归为(或分配)到权重更大的那一类。这就是K近邻算法的核心思想。

K近邻算法使用的模型实际上对应于对特征空间的划分。

算法的三个基本要素：K值的选择，距离度量和分类决策规则；

1. K值的选择会对算法的结果产生重大影响。

K值较小意味着只有与输入实例较近的训练实例才会对预测结果起作用，但容易发生过拟合；

K值较大，优点是可以减少学习的估计误差，但缺点是学习的近似误差增大，这时与输入实例较远的训练实例也会对预测起作用，使预测发生错误。

再直观点的解释就是当k过小(比如等于1)时，预测结果很容易受到某个噪声点的影响，当k过大(比如等于训练集总数据量N)时，预测结果始终为训练实例中数量最多的类。

在实际应用中，K值一般选择一个较小的数值，通常采用交叉验证的方法来选择最优的K值。随着训练实例数目趋向于无穷和K=1时，误差率不会超过贝叶斯误差率的2倍，如果K也趋向于无穷，则误差率趋向于贝叶斯误差率。

一般而言，k是不大于20的整数，可通过交叉验证法来选取最优k值。

K-Nearest Neighbor算法

该算法中的分类决策规则往往是多数表决，即由输入实例的K个最临近的训练实例中的多数类决定输入实例的类别

距离度量一般采用Lp距离，当p=2时，即为欧氏距离，在度量之前，应该将每个属性的值规范化，这样有助于防止具有较大初始值域的属性比具有较小初始值域的属性的权重过大。

Lp距离即闵可夫斯基距离：当p=1时，又称曼哈顿距离；当p=2时，又称欧式距离；当p趋向于无穷大时，又称切比雪夫距离。

该方法的不足之处是计算量较大，因为对每一个待分类的文本都要计算它到全体已知样本的距离，才能求得它的K个最近邻点。

目前常用的解决方法是事先对已知样本点进行剪辑，事先去除对分类作用不大的样本。

该算法比较适用于样本容量比较大的类域的自动分类，而那些样本容量较小的类域采用这种算法比较容易产生误分。

C4.5和CART（Classification and Regression Tree）

都是基于决策树的分类算法。

决策树本质上是一个对训练数据进行划分的数据结构。

决策树中每个结点代表一个关于属性的问题。对这个问题的不同回答决定了一个数据对象被分到树的不同分支。最终所有的训练数据都会被塞入到某个叶子结点。这和常见的二分搜索树、B+树等数据结构没有本质的区别。

与其它树的数据结构用途不同的是，决策树最终的目的是要回答一个数据对象的类别。

一个未知类别的数据对象被决策树划分到某个叶子节点，这个叶子结点内的数据类别就可以被当作这个数据的类别。

如果叶子结点的数据有几个不同的类别，与K-Nearest Neighbor一样，我们取出现次数最多的那个类别。

C4.5和CART（Classification and Regression Tree）

为了尽可能保持回答的一致性，我们希望每个叶子结点内的数据类别尽量保持一致。这样，决策树在回答一个数据类别的时候，就更有把握一些。

因此，我们希望决策树每个叶子结点的类别的分布越纯越好。当一个结点内所有数据的类别都一致的时候，这样纯度最高；当一个结点所有数据的类别都两两不相同的时候，这个纯度最低。

纯度可以用不同的指标来测量，常见的两个纯度指标是熵（Entropy）和基尼指数（Gini Index）

C4.5和CART的训练算法利用如何让熵（或者基尼指数）减得更多（纯度增加更大）来决定如何构造这个决策树。

C4.5还考虑了如何避免过度耦合（overfitting）等问题

C4.5算法优点：产生的分类规则易于理解，准确率较高。

缺点：在构造树的过程中，需要对数据集进行多次的顺序扫描和排序，因而导致算法的低效。此外，C4.5只适合于能够驻留于内存的数据集，当训练集大得无法在内存容纳时程序无法运行。

Naive Bayes

是通过贝叶斯定理来进行分类的。朴素贝叶斯将数据的属性和数据的类别看做两个随机变量（X和Y），然后问题转换成为找出一个给定属性X，哪个Y出现的概率最大，也就是贝叶斯定律中的后验概率P(Y|X)。

在贝叶斯定律里面，一个数据的产生，是有了这个数据的类别，然后再产生这个数据的各个属性。因此，P(Y)被叫做先验概率。

给定了数据的属性，再反过来去推测其类别就是后验概率，P(Y|X)。

根据贝叶斯定理，后验概率可以由先验概率和条件概率计算出来。而先验概率和条件概率可以由训练数据统计而得。

朴素贝叶斯之所以叫朴素，因为这个算法假设给定数据对象的类别Y，不同属性的出现是互相独立的（Conditional Independent）。

Naive Bayes

最为广泛的两种分类模型是决策树模型(Decision Tree Model)和朴素贝叶斯模型（Naive Bayesian Model，NBM）。和决策树模型相比，朴素贝叶斯分类器(Naive Bayes Classifier,或NBC)发源于古典数学理论，有着坚实的数学基础，以及稳定的分类效率。同时，NBC模型所需估计的参数很少，对缺失数据不太敏感，算法也比较简单。理论上，NBC模型与其他分类方法相比具有最小的误差率。但是实际上并非总是如此，这是因为NBC模型假设属性之间相互独立，这个假设在实际应用中往往是不成立的，这给NBC模型的正确分类带来了一定影响。

优点

朴素贝叶斯算法假设了数据集属性之间是相互独立的，因此算法的逻辑性十分简单，并且算法较为稳定，当数据呈现不同的特点时，朴素贝叶斯的分类性能不会有太大的差异。换句话说就是朴素贝叶斯算法的健壮性比较好，对于不同类型的数据集不会呈现出太大的差异性。当数据集属性之间的关系相对比较独立时，朴素贝叶斯分类算法会有较好的效果。

缺点

属性独立性的条件同时也是朴素贝叶斯分类器的不足之处。数据集属性的独立性在很多情况下是很难满足的，因为数据集的属性之间往往都存在着相互关联，如果在分类过程中出现这种问题，会导致分类的效果大大降低。

Support Vector Machine(SVM)

Support Vector Machine(SVM)，是近年来使用得最广的分类算法，因为他在高维数据，例如图像和文本上的表现都好过其他很多算法。

与Naive Bayes不同之处在于，它不关心这个数据是如何产生的，它只关心如何区分这个数据的类别。所以，大家也称这种分类算法是discriminative（有判别力）的。

在SVM算法内，任何一个数据都被表示成一个向量，也就是高维空间中的一个点，每个属性代表一个维度。

SVM和大多数分类算法一样，假设如果一堆数据的类别相同，那么他们的其他属性值也应该相近。因此，高维空间上，不同的类别数据应该处于不同的空间区域。

SVM的训练算法就是找出区分这几个区域的空间分界面。能找到的分界面可能有很多个，SVM算法中选择的是两个区域之间最靠近正中间的那个分界面，或者说离几个区域都最远的那个分界面（maximum-margin hyper plane）。因为，现实数据可能是有噪声的。有了噪声，一个数据可能会在观测空间的位置的周围区域都出现。几个区域最远的那个分界面能够尽量保证有噪声的数据点不至于从一个区域跳到另外区域去。

这个最佳的分界面的寻找问题在SVM中被表示成一个有约束的优化问题（Constrained Optimization）。通过优化算法里的拉格朗日法可以求得这个最优的分界面。

Support Vector Machine(SVM)

支持向量机（Support Vector Machine, SVM）是一类按监督学习（supervised learning）方式对数据进行二元分类的广义线性分类器（generalized linear classifier），其决策边界是对学习样本求解的最大边距超平面（maximum-margin hyper plane）。

Apriori算法

Apriori算法的最早提出是为了寻找关联规则。

后来，因为Apriori算法有很清晰和简单的算法逻辑结构，所以Apriori逐步成为一种搜索算法思想，有点类似动态规划、贪心算法的概念。

在很多相关论文里面，算法以Apriori-like来修饰，但是算法的目的跟关联规则挖掘并没有关系。

关联规则通常表示成{A,B}→{C}，意思是如果A，B出现了，那么C也很大可能出现。

关联规则是通过历史数据求得。

显然，我们不能因为历史数据中出现了一条数据同时包含A，B，C就认为{A,B}→{C}成立。只有当足够的数据记录都包含A，B，C，我们才认为这个规则有一定置信度的。所以，关联规则第一步就是找出频繁的项集，项就是指这里的A，B，C等。项集就是包含这些项的集合。在这个例子里面，{A,B,C}就是一个频繁的项集，A，B，C属于一个项。如果数据库里面有n个不同的项，那么总共可能有2^n个项集。

显然我们不能一个一个去试，看其是否频繁。Apriori算法要解决的问题，就是如何快速找出频繁项集。

Apriori算法

Apriori的核心思想就是认为如果{A,B,C}是频繁的，那么它的子集也必须是频繁的。这就是Apriori算法里所描述的anti-monotonic property（递减性质）。

频繁的定义就是指出现的次数大于某个预先定义的阈值。因此，我们可以从只含一个项的集合开始搜索，然后剔除非频繁。然后再找只含2个项的集合，再剔除非频繁。

如果把算法的搜索看做一个搜索树，那么每次的剔除都是剔除一个树的分支，所以就可以大大减小搜索空间。这也是为什么大部分Apriori-like算法本质上都是搜索算法的原因。

PageRank算法

PageRank因为谷歌搜索引擎而出名。

满足一个关键词的网页通常很多很多，如何安排这些网页的显示的前后顺序呢？

PageRank的想法就是，如果这个网页被很多其他网页引用，那么这个网页重要程度就很大，理应放得前面一些。如果一个网页只被很少网页引用甚至没有被引用，那么这个网页就不重要，可以放得靠后。这里的引用和论文之间的引用类似。我们评价一篇论文的好坏也是看其引用数量。

在网页里面，引用可以是一个超链接。当然，PageRank还可以用在其他图数据上，只要他们存在某种链接，就可以认为是这里的引用。

除此之外，PageRank还认为，如果一个网页被重要的网页引用，那么这个网页肯定比被不重要网页引用更重要。如果把每个网页的重要分数看成一个未知变量，这两个直观的假设可以整理成一个线性方程。那么PageRank根据此方程解出每个网页的重要分数。最后网页的排名就是按照这个重要分数由大到小排列。

换句话说，PageRank算法综合考虑链接的数量和网页的质量两个因素，将二者结合起来对网页进行排序。需要特别指出的是，PageRank计算出的网页重要性排名，是完全基于链接结构的，与用户输入的查询关键字是无关的。所以，很多时候PageRank是可以离线计算的。

Adaboost算法

Adaboost是一种集成学习算法。

其核心思想是在同一个训练集上，通过考察上一次实验中每个样本的分类是否正确，以及总体分类的准确率，自适应地调整每个样本的权值，迭代地生成若干个不同的分类器（弱分类器），最终将这些分类器组合起来，提升为一个强分类器。

AdaBoost为了减少分错的情况，有意识专门针对分错的数据进行训练。

这种思想就例如中学时期的试卷练习一样。同一份试卷的考题可以让学生做多次，但每次只需要重复上次学生做错的题。而上次做对了的题目，就不需要反复练习了。这样的练习可以突出重点，同时又节约时间。

K-means和Expectation Maximization算法

K-means是最常见的聚类算法。

Expectation Maximization是一种用来估计带有隐藏变量的模型参数的方法。

K-means背后的思想其实属于Expectation Maximization的一种。K-means先随机在数据集里面找k个簇中心点，然后把每个数据分到离它最近的中心所在的那个簇。然后再计算新的簇中心点。由此不断循环直到这个所有簇中心不再变化。

在K-means里面，这个模型的参数就是k个簇的中心。隐藏的变量就是每个数据点的类标。如果我们知道每个数据点的类标，那么只需要针对每个簇的数据点求一下算术平均，就可以估计出这个簇的中心。

但问题是，每个数据点的类标是隐藏的（未知的），所以我们无法直接估计出每个簇的簇中心。Expectation Maximization算法就是专门来针对这种有隐藏变量的模型估计的。

Expectation Maximization算法的估计是基于最大似然（Maximum Likelihood），也就是找出模型参数使得这个模型最能够描述当前观察数据。

换句话说，这个估计方法希望让观察的数据在这个估计的模型里面出现的概率是最高的。

K-means和Expectation Maximization算法

最大似然是一个原则。最大似然的估计方法是根据这个原则产生的，否则任何一个估计出来模型我们都可以说它好或者不好。

至于为什么用最大似然这个原则，这源于概率论与数理统计的一个直观假设：我们观察到的事件总比没有观察到的发生的概率高；我们多次观察到事件出现的概率总比很少观察到的高。

Expectation Maximization算法是两个不同步骤的不断循环。

一个步骤叫作Expectation，也就是固定当前模型的参数去估计最有可能隐藏变量的值。

另外一个步骤叫作Maximization，既是通过当前估计隐藏变量的值去估计模型的参数。然后不断迭代，直到模型的参数不再变化即算法停止。

Expectation和Maximization都是基于最大似然去找隐藏变量的值和模型的参数值。

在K-means里面，把每个数据点塞进K个簇中，离簇中心最近那个步骤就是这里的Expectation。通过计算簇内所有数据点的算术平均得到簇的中心就是Maximization。K-means的目的就是让每个簇内的数据点尽量靠近簇中心。从最大似然的角度来看，一个数据点越靠近其簇中心，它出现的概率越高。因此在Expectation的时候，K-means把每个数据点塞给最近的那个簇中心的簇。Maximization步簇中心是通过算术平均求得的。算术平均就是最大似然的估计。