**第七章 K-近邻算法**

K-近邻算法（K-Nearest Neighbors， KNN）是一个理论上比较成熟的方法，也是最简单的机器学习算法之一。

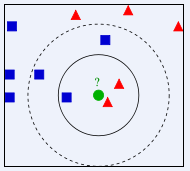
它的工作原理是：存在一个样本数据集合，也称作训练样本集，并且样本数据集中每个样本都有标签，即我们知道样本集中每一样本与所属类别的对应关系。输入新的样本数据，将新数据的样本特征与样本集中数据对应的特征进行比较，然后算法提取样本集中特征最相似数据（最近邻）的分类标签。一般来说我们只选择样本数据集中前个最相似的数据，这就是K-近邻算法的出处。最后，选择个最相似数据中出现次数最多的分类，作为新数据的分类。

KNN是一种非参数机器学习模型（Non-Parametric Machine Learning Model），决策树亦是，它通过记忆训练集数据来对新的样本数据进行分类。它也被称为基于实例的学习（Instance-Based Learning）。

这种模型也被称为lazy learning，因为它不像回归、随机森林等方法，它在训练阶段不需要学习任何东西。

基于实例（Instance-Based）：先记住所有实例（训练数据），然后用相似度算法来泛化到新数据中；

基于模型（Model-Based）：基于训练数据学习一个模型（函数），然后基于该模型来做预测。



（百度百科）

我们要判别上图中那个绿色的圆是属于哪一类数据。我们从它的邻居下手。但一次性看多少个邻居呢？从上图中，你还能看到：

如果，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，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。

于此我们看到，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。值的选择、距离度量和分类决策规则是该算法的三个基本要素：

值的选择会对算法的结果产生重大影响。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；如果值较大，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。在实际应用中，值一般选择一个较小的数值，通常采用交叉验证（或网格搜寻）的方法来选择最优的值。随着训练实例数目趋向于无穷和时，误差率不会超过贝叶斯误差率的2倍，如果也趋向于无穷，则误差率趋向于贝叶斯误差率。

距离度量一般采用距离，当时，即为欧氏距离，在度量之前，应该将每个属性的值规范化（归一化），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。

分类决策规则往往是多数表决，即由输入实例的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，也就是闵可夫斯基距离（Minkowski Distance）：

给定向量，它们的距离定义为

也就是向量的-范数。

当时，距离就变成了曼哈顿距离（Manhattan Distance）：

当时，距离就变成了欧式距离（Euclidean Distance）：

该算法在分类时主要的不足是：

1. 不平衡数据：当样本不平衡时，如一个类的样本容量很大，而其他类样本容量很小时，有可能导致当输入一个新样本时，该样本的K个邻居中大容量类的样本占多数。 该算法只计算“最近的”邻居样本，某一类的样本数量很大，那么或者这类样本并不接近目标样本，或者这类样本很靠近目标样本。无论怎样，数量并不能影响运行结果。可以采用权值的方法（和该样本距离小的邻居权值大）来改进。

2. 大容量数据：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，才能求得它的个最近邻点。所以，对于大数据，KNN算法不是很有效。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。

3. 高维数据：由于维数灾难（Curse of Dimensionality），高维数据也会影响KNN算法的效果。

KNN算法不仅可以用于分类，还可以用于回归。通过找出一个样本的k个最近邻居，将这些邻居的属性的平均值赋给该样本，就可以得到该样本的属性。更有用的方法是将不同距离的邻居对该样本产生的影响给予不同的权值(weight)，如权值与距离成反比。

**Python示例：**

变量、引用和对象

在Python中对每一个变量赋值时，就创建了一个指向等号右边的引用。

a = [4, 3, 2, 5]

a为变量，[4, 3, 2, 5]为对象。

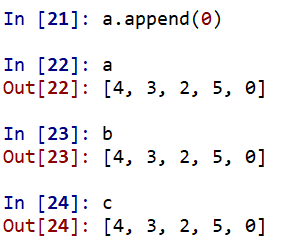
定义新的变量：

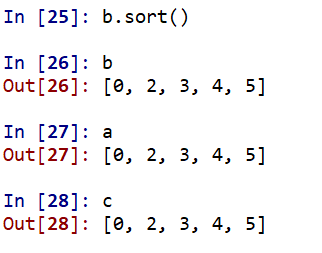
b = a

c = b

在Python中，变量a，b，c都指向了相同的对象[4, 3, 2, 5]，即







1. 变量无类型，对象有类型：「类型」这个概念不是存在于变量中，而是存在于对象中。变量本身就是通用的，它只是恰巧在某个时间点上引用了当时的特定对象而已。就比如说在表达式中，我们用的那个变量会立马被它当时所引用的特定对象所替代。

2. 变量引用对象：变量通过一根线，连着对象，变量用自己所拥有的能力，把对象和自己连接起来（指针连接对象空间），引用建立了变量和对象之间的映射关系，这就是引用。引用完成，就实现了赋值。

表1 一个二类问题的混淆矩阵（Confusion Matrix）

|  |  |  |  |
| --- | --- | --- | --- |
|  | | 预测结果 | |
| 1 | 0 |
| 真实结果 | 1 | 真正例（TP） | 假负例（FN） |
| 0 | 假正例（FP） | 真负例（TN） |

**自己编写代码来计算TP、FN、FP、TN和Confusion Matrix**

真实结果：y；预测结果：y\_pred

正例：y[y == 1]

预测结果应对应的正例：y\_pred[y == 1]

TP = len(y\_pred[y==1][ y\_pred[y == 1]==1])

TN = len(y\_pred[y==0][ y\_pred[y == 0]==0])

FN = len(y\_pred[y==1][ y\_pred[y == 1]==0])

FP = len(y\_pred[y==0][ y\_pred[y == 0]==1])

真阳性率（True Positive Rate， TPR）：TPR = TP / ( TP+FN )（敏感性: Sensitivity）

真阴性率（True Negative Rate，TNR）：TNR= TN / (FP + TN) （特异性：Specificity）

准确率（Accuracy）：Acc = ( TP + TN ) /(TP+TN+FP+FN)

精确度 (Precision)：TP / ( TP+FP )

召回率（Recall）：TP / (TP + FN )

from sklearn.neighbors import KNeighborsClassifier

knn\_fit = KNeighborsClassifier(n\_neighbors = 3, p = 2, metric = 'minkowski')

参数n\_neighbors： int, optional (default = 5)

参数p : integer, optional (default = 2)，Lp距离的指数p

参数metric： string or callable, default 'minkowski'，距离度量

Breast Cancer数据集

<http://mlr.cs.umass.edu/ml/datasets/Breast+Cancer+Wisconsin+%28Diagnostic%29>

# Attribute Domain

-- -----------------------------------------

1. Sample code number id number

2. Clump Thickness 1 - 10 肿块厚度

3. Uniformity of Cell Size 1 - 10 细胞大小的均匀性

4. Uniformity of Cell Shape 1 - 10 细胞形状的均匀性

5. Marginal Adhesion 1 - 10 边缘粘

6. Single Epithelial Cell Size 1 - 10 单上皮细胞的大小

7. Bare Nuclei 1 - 10 裸核

8. Bland Chromatin 1 - 10 乏味染色体

9. Normal Nucleoli 1 - 10 正常核

10. Mitoses 1 - 10 有丝分裂

11. Class: (2 for benign, 4 for malignant) 良性的，恶性的

Missing attribute values: 16 denoted by "?".

样本数：699

良性的：458；恶性的：241

超参数K的选取：网格搜寻（Grid Search），K的取值分别为1，2，3，4，5，6，7，8。

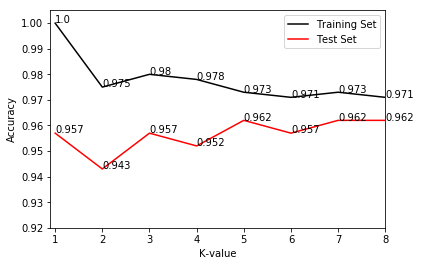


图1 K取不同值时对应的训练准确率和测试准确率

**第八章 推荐引擎**

**推荐引擎****（Recommendation Engine， RE）**

大型推荐系统在互联网上的应用非常广泛，比如在网上购物时，电商平台会根据你的个人信息和历史购买行为向你推荐商品；又比如使用手机收听音乐时，相应的App会根据你的喜好，推送歌曲。从模型的角度来看，这些推荐行为其实对应着一个分类问题。

* 站在用户的角度，推荐系统将所有物品分为两类，一类是这个用户感兴趣的，另一类是这个用户不感兴趣的。
* 反过来，站在物品的角度，推荐系统将用户分为两类，对该物品感兴趣的和对该物品不感兴趣的。

针对这样的分类问题，前面讨论的很多模型都能有效地解决，比如逻辑回归、支持向量学习机等。但这些模型都只能对一个角度进行建模，比如从用户的角度出发，则建模数据是物品的特征，而用户的喜好倾向由模型参数体现，也就是说，这时需要对每个用户单独进行建模。如果从物品的角度出发，也可以得到类似的结论。但问题是，上面的互联网应用场景有一个共同的特点就是用户数量和物品数量都很大，这导致经典的分类模型无法使用（否则需要同时搭建和训练几万，甚至几百万的模型）。

推荐引擎（Recommendation Engine， RE）是主动发现用户当前或潜在需求的定律，并主动推送信息给用户的信息网络。挖掘用户的喜好和需求，主动向用户推荐其感兴趣或者需要的对象。

推荐引擎不是被动查找，而是主动推送；不是独立媒体，而是媒体网络；不是检索机制，而是主动学习。推荐引擎利用基于内容、基于用户行为、基于社交关系网络等多种方法，为用户推荐其喜欢的商品或内容。

**基于内容的推荐（Content-Based Recommendation）**

基于内容的推荐是在推荐引擎出现之初应用最为广泛的推荐机制，它的核心思想是根据推荐物品或内容的元数据，发现物品或者内容的相关性，然后基于用户以往的喜好记录，推荐给用户相似的物品。

**基于协同滤波的推荐（Collaborative Filtering-Based Recommendation）**

协同滤波是基于用户行为的推荐，不需要知道推荐物品的内容。

而基于协同过滤的推荐，又分三个子类：

1、基于用户的推荐(通过共同口味与偏好找相似邻居用户，K-邻居算法，你朋友喜欢，你也可能喜欢)；

2、基于项目的推荐(发现物品之间的相似度，推荐类似的物品，你喜欢物品A，C与A相似，可能也喜欢C)；

3、基于模型的推荐(基于样本的用户喜好信息构造一个推荐模型，然后根据实时的用户喜好信息预测推荐)。

**相似度测量**：

余弦相似度（Cosine Similarity）、基于欧氏距离的相似度、皮尔逊相关系数、修正余弦相似度（Adjusted Cosine Similarity）。

向量，其中和分别别表示用户X和Y对物品的打分。如果没有打分，则为0。

用户X和Y余弦相似度（Cosine Similarity）定义为

余弦相似度体现的是每个向量的方向关系（角度），而非幅度。如果你想要幅度，则应计算欧几里德距离。

用户X和用户Y的公共评分集，也就是均被两者评分的物品的集合定义为。用户X和用户Y对物品集的打分分别为和，

用户X和用户Y的基于欧氏距离的相似度为。

皮尔逊相关系数（Pearson Correlation Coefficient）:

其中。

所以，皮尔逊相关系数就是减去平均值(去中心化)后做余弦相似性。

用户X和用户Y的皮尔逊相关系数

其中，。

修正余弦相似度（Adjusted Cosine Similarity）：

常规余弦相似度反映了方向的差异，而不是位置的差异。因此，使用余弦相似度指标无法考虑到用户评分这样的差异。调整后余弦相似度可以缓解这一问题，具体做法是从每对共同评分的配对减去各自用户的平均评分，其定义如下：

其中, 向量中的元素为用户X的平均评分，向量中的元素为用户Y的平均评分。

* 修正余弦相似度&余弦相似度的区别：

在余弦相似度上进行修正后得到修正余弦相似度，在计算的时候以用户的平均打分作为中心化标准进行修正每个用户在已有打分的item上的打分（忽略了未打分的item）。

* person系数&修正余弦相似度的区别：

去中心化的均值不同。

Person去中心化的标准是该用户与另一用户共同打分的物品的打分均值；

修正余弦相似度是该用户打分的所有物品的打分均值。

* 余弦相似度 & person系数与修正余弦相似度的区别：

余弦相似度计算时包括了用户所有打分信息，也就是说包括缺省的打分信息（缺省的地方赋0值）；

person系数与修正余弦相似度进行计算时只包括了两个用户都进行了打分的物品上的信息（忽略缺省的地方）。

通过上面任意一种方法找出用户相似度后，可以根据相似度的大小排序，找出前k个最相似的用户。找出相似用户后，可以直接对相似用户之间进行物品推荐，如A和B相似，且B有部分喜欢的电影A没看过，就可以为A推荐这部分电影。

**交替最小二乘（Alternating Least Squares, ALS）**

用户对物品的打分行为可以表示成一个评分矩阵，表示个用户对个物品的打分情况，其中，表示用户对物品的打分。如下表所示。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

用户不会对所有物品打分，图中表示用户没有打分的情况，所以这个矩阵很多元素都是空的，我们称其为“缺失值（Missing Value）”。这样的矩阵我们称之为稀疏矩阵（Sparse Matrix）。

在推荐系统中，我们希望得到用户对所有物品的打分情况，如果用户没有对一个物品打分，那么就需要预测用户是否会对物品打分，以及会打多少分。这就是所谓的“矩阵补全（填空）”。我们建模的目的就是要将评分矩阵“填满”（使用模型的预测值）。在此基础上，为用户推荐就是件很轻松的事情了。

一个的打分矩阵，可以用两个小矩阵和的乘积来近似：

其中称为用户喜好特征矩阵，称为产品特征矩阵。

我们使用用户喜好特征矩阵中的第个用户的特征向量和产品特征矩阵第个产品的特征向量来预测打分矩阵中的。我们可以得出一下的矩阵分解模型的损失函数为：

有了损失函数之后，就可以用优化方法来求解。通常的优化方法分为两种：交替最小二乘法（Alternative Least Squares）和随机梯度下降法（Stochastic Gradient Descent）。

ALS算法的思想就是：对目标函数，先随机生成然后固定它求解，再固定求解，这样交替进行下去，直到取得最优解。因为每步迭代都会降低误差，并且误差是有下界的，所以ALS一定会收敛。但由于问题是非凸的，ALS并不保证会收敛到全局最优解。但在实际应用中，ALS对初始点不是很敏感，是否全局最优解造成的影响并不大。

ALS算法执行步骤：

* 先随机生成用户喜好特征矩阵和产品特征矩阵。
* 固定，即认为是已知的常量，来求解：

由于上式中只有一个未知变量，因此的最优化问题转化为最小二乘问题，用最小二乘法求解的最优解。

上式两边关于求导得：

  令上式等于，得：

式中，为单位矩阵，为矩阵的第列列向量。

* 同理，用步骤 2 中类似的方法：

固定，则上式两边关于求导得：

  令上式为，可得：

式中，为单位矩阵，为矩阵的第行行向量。

循环执行步骤 2、3，直到损失函数的值收敛（或者设置一个迭代次数，迭代执行步骤 次后停止）。这样，就得到了最优解对应的矩阵。

**Python应用示例**

电影评分数据集

<https://grouplens.org/datasets/movielens/>

Movie Lens 数据集含有来自6000名用户对4000部电影的100万条评分数据。它分为三个表：评分、用户信息和电影信息。