1. 数据集中各特征值为连续型，也就是有无数个可能的值。测量得到的数据就是这个样子，比如，测量结果可能是1、 1.2或1.25，等等。连续值的另一个特点是，如果两个值相近，表示相似度很大。一种萼片长1.2cm的植物跟一种萼片宽1.25cm的植物很相像。

与此相反，类别的取值为离散型。虽然常用数字表示类别，但是类别值不能根据数值大小比较相似性。 Iris数据集用不同的数字表示不同的类别，比如类别0、 1、 2分别表示Iris Setosa、 IrisVersicolour、 Iris Virginica。但是这不能说明前两种植物，要比第一种和第三种更相近——尽管单看表示类别的数字时确实如此。在这里，数字表示类别，只能用来判断两种植物是否属于同一种类别，而不能说明是否相似。

1. 数据集的特征为连续值，而我们即将使用的算法使用类别型特征值，因此我们需要把连续值转变为类别型，这个过程叫作离散化。特征值大小实际上与该特征的分类效果没有任何关系

最简单的离散化算法，莫过于确定一个阈值，将低于该阈值的特征值置为0，高于阈值的置为1。我们把某项特征的阈值设定为该特征所有特征值的均值。每个特征的均值计算方法如下。

attribute\_means = X.mean(axis=0)

我们得到了一个长度为4的数组，这正好是特征的数量。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 特征1 | 特征2 | 特征3 | 特征4 | 类别 |
|  |  |  |  | 0，1，2... |

注意区分特征和类别 此处类别的取值为离散型

X\_d = np.array(X >= attribute\_means, dtype='int')

注意，任何的数值转化都使用逻辑表达式！！！

1. 计算距离的选择

* 欧氏距离：但是如果某些特征比其他特征取值大很多，精确度就会比较差。此外，如果很多特征值为0，也就是所谓的稀疏矩阵，结果也不准确。
* 曼哈顿距离：虽然异常值也会影响分类结果，但是其所受的影响要比欧氏距离小得多
* 余弦距离：更适合解决异常值和数据稀疏问题

例如，你的数据集有很多特征，但是如果任意一对个体之间的欧氏距离都相等，那么你就没法通过欧氏距离进行比较了！曼哈顿距离在某些情况下具有更高的稳定性，但是如果数据集中某些特征值很大，用曼哈顿距离的话，这些特征会掩盖其他特征间的邻近关系。最后，再来说说余弦距离，它适用于特征向量很多的情况，但是它丢弃了向量长度所包含的在某些场景下可能会很有用的一些信息

# 特征工程

***类别型特征***二值化后就变成了数值型特征

***数值型特征***也可以通过离散化过程转换为类别型特征

特征可以是类别型的也可以是数值型的，看需要进行转换

1. 在开展数据挖掘工作前，有些基础性测试我们要做，比如确保特征值是不同的。如果特征值都相同，就跟没提供什么信息一样，挖掘就失去了意义。scikit-learn中的VarianceThreshold转换器可用来删除特征值的方差达不到最低标准的特征。

from sklearn.feature\_selection import VarianceThreshold

vt = VarianceThreshold()

Xt = vt.fit\_transform(X) # 会直接剔除不达最低方差的特征

输出每一列的方差。

print(vt.variances\_)

1. 选择最佳特征

特征很多的情况下，怎么选出最佳的几个，可有点难度。随着特征数量的增加，寻找子集的任务复杂度呈指数级增长。寻找最佳特征组合的时间复杂度同样是指数级增长的。其中一个变通方法是不要找表现好的子集，而只是去找表现好的单个特征（单变量），依据 是它们各自所能达到的精确度。分类任务通常是这么做的，我们一般只要测量变量和目标类别之间的某种相关性就行。

scikit-learn提供了几个用于选择单变量特征的转换器，其中SelectKBest返回k个最佳特征， SelectPercentile返回表现最佳的前r%个特征。这两个转换器都提供计算特征表现的一系列方法。单个特征和某一类别之间相关性的计算方法有很多。最常用的有卡方检验（χ2）。其他方法还有互信息和信息熵。

使用SelectKBest转换器类，用卡方函数打分，初始化转换器。

from sklearn.feature\_selection import SelectKBest

from sklearn.feature\_selection import chi2

transformer = SelectKBest(score\_func=chi2, k=3)

Xt\_chi2 = transformer.fit\_transform(X, y) # 调用fit\_transform方法，对相同的数据集进行预处理和转换。X为特征，y为类别,最后选出前k个分数最高的特征。

1. 创建特征，PCA

主成分分析算法（Principal Component Analysis， PCA）的目的是找到能用较少信息描述数据集的特征组合。它意在发现彼此之间没有相关性、能够描述数据集的特征，确切说这些特征的**方差**跟整体方差没有多大差距，这样的特征也被称为主成分。这也就意味着，借助这种方法，就能通过更少的特征捕获到数据集的大部分信息。

PCA跟其他转换器用法类似。它只有主成分数量这一个参数。它默认会返回数据集中的所有特征。然而， PCA会对返回结果根据方差大小进行排序，返回的第一个特征方差最大，第二个特征方差稍小，以此类推。因此，前几个特征往往就能够解释数据集的大部分信息。

from sklearn.decomposition import PCA

pca = PCA(n\_components=5)

Xd = pca.fit\_transform(X)

返回的结果Xd矩阵只有五个特征，但是不容小觑，我们看一下每个特征的方差。

np.set\_printoptions(precision=3, suppress=True)

pca.explained\_variance\_ratio\_

# [sklearn.decomposition](http://sklearn.apachecn.org/cn/0.19.0/modules/classes.html#module-sklearn.decomposition).PCA

**PCA**(n\_components=None, copy=True, whiten=False, svd\_solver='auto', tol=0.0, iterated\_power='auto', random\_state=None)

n\_components: 降维后的特征数

copy：如果为True,则后续使用fit(X).transform(X)；如果为False,则后续使用fit\_transform（X）

属性：