第5章

特征工程、降维与超参数调优

特征工程、降维与超参数调优是机器学习工程应用中的三个重要问题。在优惠券核销示例中讨论过，输入模型进行训练的一般不是实例的属性，而是从实例的属性数据中提取出的特征。从属性数据中提取出特征的过程叫作“特征工程”。在掌握机器学习的算法之后，特征工程就是最具有挑战性的工作了。

降维技术可以解决因为特征过多而带来的样本稀疏、计算量大等问题。

机器学习模型的参数有两种，一种是从样本中学习得到；另一种无法靠模型自身得到，需要人为设定。需要人为设定的参数称为超参数(Hyperparameters)，如 k-means 算法中的 k 值，分类树模型中树的层次，随机森林算法中树的个数等。超参数一般控制模型的主体框架，超参数的改变会对模型建立和预测产生很大的影响。超参数调优是寻找使模型整体最优的超参数的过程。

5.1 特征工程

特征工程的目标是从实例的原始数据中提取出供模型训练的合适特征。特征的提取与问题的领域知识密切相关。特征工程在机器学习过程中的位置和作用如图 5-1 所示。训练样本通过特征工程提取出合适的样本特征向量，供模型训练、评估使用。测试样本通过相同的特征工程提取出相同样式的特征向量，输入模型后，得到预测结果。

总体来说，特征提取是一种创造性的活动，没有固定的规则可循，本节仅讨论与特征提取相关的辅助环节。一般来说，需要先从总体上理解数据，必要时可通过可

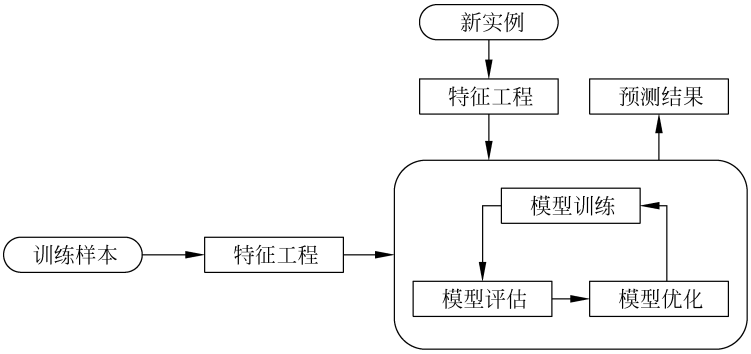


图 5-1　特征工程在机器学习过程中的位置和作用

视化来帮助理解，然后运用领域知识进行分析和联想，然后处理数据提取出特征。并不是所有提取出来的特征都会对模型预测有正面帮助，还需要通过预测结果来对比分析。

在机器学习应用中，特征提取是很重要的环节。对于文本、图像、语音等复杂数据，人工提取特征是非常困难的事情。为了追求好的效果，人们想了很多办法来提取它们的特征，提取出的特征数量甚至达到了上万个。近年来，自然语言处理、图像识别、语音识别等领域的自动提取特征研究取得了的重大突破，发展成为机器学习的一个重要分支：深度学习(Deep Learning)。正因为深度学习的进展，机器学习的应用门槛大为降低，使之得到了广泛应用。有关深度学习的内容，将在第 8 章专门讨论。

5.1.1　数据总体分析

得到样本数据后，先要对数据的总体概况进行初步分析。分析数据的总体概况，一般是根据经验进行，没有严格的步骤和程序，内容主要包括查看数据的维度、属性和类型，对数据进行简要统计，分析数据类别分布(分类任务)。

下面以优惠券使用预测数据为例，对数据的总体分析进行简单示例，主要用到了 Pandas 库中的一些函数。

1. 查看数据

Pandas 库中的 head 函数用来列出表中的前面若干项的数据，可以用来查看数据表的组成和部分样例。示例代码及输出见代码 5-1 中的前 4 行。

代码 5-1　数据总体分析示例(数据理解.ipynb)

```
1. >>> import pandas as pd
2. >>> df = pd.read_csv('E:\mlDataSets\o2o\ccf_offline_stage1_train.csv')
3. >>> df.head(10)
```

	User_id	Merchant_id	Coupon_id	Discount_rate	Distance	Date_received	Date
0	1439408	2632	null	null	0	null	20160217
1	1439408	4663	11002	150:20	1	20160528	null
2	1439408	2632	8591	20:1	0	20160217	null
3	1439408	2632	1078	20:1	0	20160319	null
4	1439408	2632	8591	20:1	0	20160613	null
5	1439408	2632	null	null	0	null	20160516
6	1439408	2632	8591	20:1	0	20160516	20160613
7	1832624	3381	7610	200:20	0	20160429	null
8	2029232	3381	11951	200:20	1	20160129	null
9	2029232	450	1532	30:5	0	20160530	null

```
>>> df.shape
(1754884, 7)
>>> df.describe(include = 'all')
```

	User_id	Merchant_id	Coupon_id	Discount_rate	Distance	Date_received	Date
count	1.754884e+06	1.754884e+06	1754884	1754884	1754884	1754884	1754884
unique	NaN	NaN	9739	46	12	168	183
top	NaN	NaN	null	null	0	null	null
freq	NaN	NaN	701602	701602	826070	701602	977900
mean	3.689255e+06	4.038808e+03	NaN	NaN	NaN	NaN	NaN
std	2.123428e+06	2.435963e+03	NaN	NaN	NaN	NaN	NaN
min	4.000000e+00	1.000000e+00	NaN	NaN	NaN	NaN	NaN
25%	1.845052e+06	1.983000e+03	NaN	NaN	NaN	NaN	NaN
50%	3.694446e+06	3.532000e+03	NaN	NaN	NaN	NaN	NaN
75%	5.528759e+06	6.329000e+03	NaN	NaN	NaN	NaN	NaN
max	7.361032e+06	8.856000e+03	NaN	NaN	NaN	NaN	NaN

```
>>> df.groupby('Merchant_id').size()
1          14
2          11
3          18
          ...
8856      250
Length: 8415, dtype: int64
```

2. 数据维度

代码 5-1 中的第 5 行,用 Pandas 库的 DataFrame 对象的 shape 属性查看数据维度。可以看到,该数据集有 1 754 884 行数据(实例),每行有 7 列数据(属性)。通过这些观察,可以对数据集的维度有一个大概的了解。

3. 数据统计

代码 5-1 中的第 7 行,用 describe 方法对数据进行了统计。该方法对于数值型的列给出平均值、最小值、最大值、标准差、较小值、中值、较大值等统计指标。在该例中,数值型的列是用户 ID 和商户 ID,其统计指标并没有什么意义,但仍然可以观察一下其最大值、最小值,以及代表偏离均值的标准差的大小等。

对于对象型的列,则给出唯一值个数、众数及其次数三个统计指标。如 Discount_

rate，一共有 46 个不一样的值出现，其中出现次数最多的是 null，一共出现了 701 602 次。

4. 查看数据类别分布

代码 5-1 中的第 9 行，用 groupby 方法查看某列数据的分类情况。可以看到，对于 Merchant_id 列，一共有 8415 个商户，每个商户的消费记录条数都列出来了。

5.1.2　数据可视化

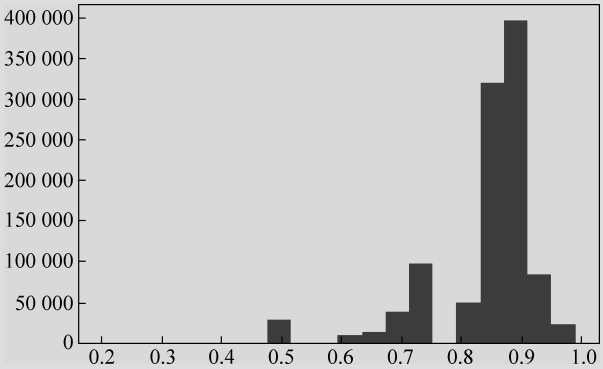
数据可视化通过直观的方式增加对数据的理解，帮助提取有用特征。

1. 特征取值分布

特征的取值分布情况可以为分析特征提供重要信息。下面以优惠券核销中的折扣率为例来画出数据分布图，从整体上理解折扣率的分布情况，示例代码见代码 5-2。

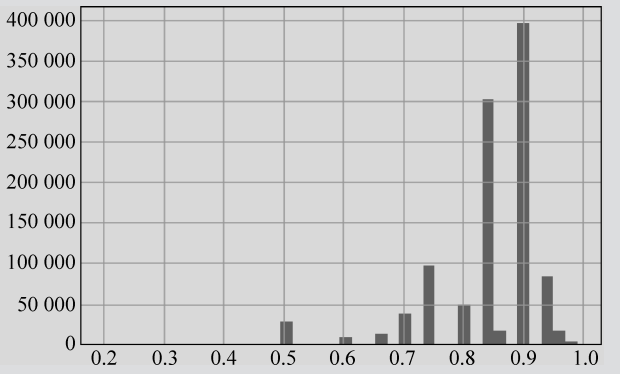
代码 5-2　特征取值分布可视化示例（数据可视化.ipynb）

```
1. import pandas as pd
2. df = pd.read_csv('E:\mlDataSets\o2o\ccf_offline_stage1_train.csv')
3. dr = df[['Discount_rate']]
4. drate = dr[dr.Discount_rate != 'null']
5.
6. def calc_discount_rate(s):
7.     s = str(s)
8.     s = s.split(':')
9.     if len(s) == 1:
10.        return float(s[0])
11.    else:
12.        return 1.0 - float(s[1])/float(s[0])
13. drate = drate.Discount_rate.astype('str').apply(calc_discount_rate)
14.
15. import matplotlib.pyplot as plt
16. plt.hist(drate, 20)
17. plt.show()
```




```
18.
19. drate.hist(bins = 40)
20. plt.show()
```

21.



第 4 行去掉折扣率为空的数据。因为有的是用满减的优惠方式,第 6～12 行定义一个函数来计算折扣率。

第 16 行用 pyplot 的 hist 方法,将折扣率的取值范围(0～1)分为 20 等分,对每一等分上的取值个数画直方图。

第 19 行用 DataFrame 的 hist 方法画直方图,等价于上面的方法。这次是分为 40 等分。

可以看出折扣率的分布,近似于正态分布,并以 0.9、0.85、0.95 为主。

查看特征值分布,还可以画饼图,不再赘述,可自行练习。

2. 离散型特征与离散型标签的关系

样本特征的值与该样本的标签的关系,是机器学习最为关心的事情。可视化可以直观地展现标签值随某特征取值的变化而变化的情况。

特征的取值分为离散和连续两类,同样标签也分为离散和连续两类。分类任务中的标签是离散型的,回归任务中的标签是连续型的。在分类任务中,标签可分为二分类和多分类的。可采用马赛克图(Mosaic Plot)[①]来可视化离散型特征值与离散型标签的关系。

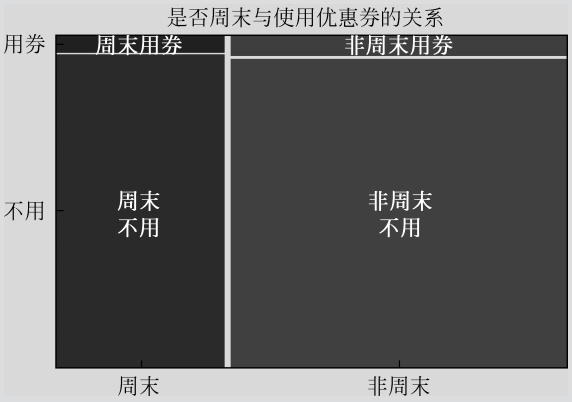
前面的示例从接收优惠券的日期中提取出了是否为周末的特征。来看看该特征与是否使用优惠券之间关系的可视化,见代码 5-3。

代码 5-3　离散型特征与离散型标签关系可视化示例(数据可视化.ipynb)

```
1. df = pd.read_csv(r'..\tianchi2o\feature_offline_in15days_2018_01_06.csv')
2. from statsmodels.graphics.mosaicplot import mosaic
3. wk = df.is_weekend.astype('str').apply(lambda x: '周末' if x == '1' else '非周末')
4. label = df.coupon_apply.astype('str').apply(lambda x: '用券' if x == '1' else '不用')
5. mosaic_data = pd.concat([wk, label], axis = 1)
6. mosaic(data = mosaic_data, index = ['is_weekend', 'coupon_apply'], gap = 0.01, title = u
   '是否周末与使用优惠券的关系')
```

① http://www.statsmodels.org/stable/generated/statsmodels.graphics.mosaicplot.mosaic.html

```
7.
8. plt.rc('font', family = 'SimHei', size = 13)
9. plt.show()
```




```
10.
```

第 2 行从 statsmodels.graphics.mosaicplot 包中导入 Mosaic 绘图工具。为了便于观察，用中文来显示周末和标签。

可以直观地看出，周末领取的券与非周末领取券的比例情况。可以看出周末领取的券比非周末领取的券核销的比例只稍高一点。那么如果分为周一到周日七天，是否能观察出更多的结论呢？读者可自行分析。

3. 连续型特征与离散型标签的关系

观察连续型特征与离散型标签的关系，常用盒图(Box-plot)。对于单个变量，盒图描述的是其分布的四分位图：上边缘、上四分位数、中位数、下四分位数和下边缘，如图 5-2 所示。上边缘是最大数，上四分位数是由大到小排在 1/4 的那个值，中位数是排在中间的那个数，下四分位数是排在 3/4 的那个数，下边缘是最小数。单个变量的盒图便于观察变量值的分布中心、扩展和偏移，另外还可以发现离群的异常值的存在。

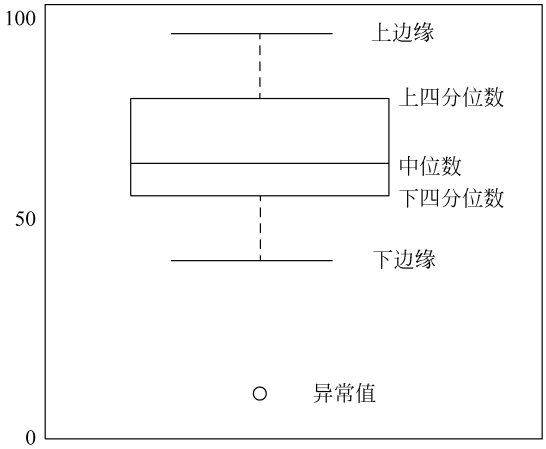


图 5-2　盒图示例

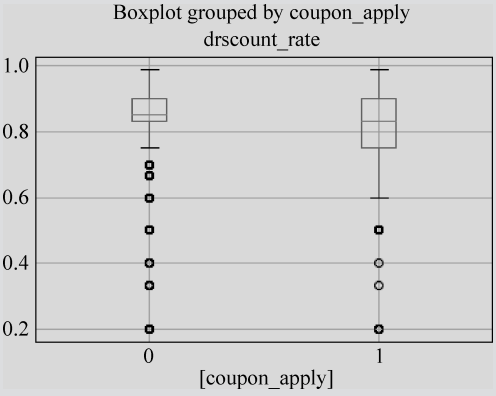
还是以折扣率为例，来看看核销与没有核销优惠券的折扣率的分布，见代码 5-4。

代码 5-4　连续型特征与离散型标签关系可视化示例(数据可视化.ipynb)

```
df = pd.read_csv('E:\mlDataSets\o2o\ccf_offline_stage1_train.csv')
df = df[df.Coupon_id!= 'null']
df['discount_rate'] = df.Discount_rate.apply(calc_discount_rate)
from datetime import date
def get_label(s):
    s = s.split(':')
    if s[0] == 'null':
        return 0
    elif (date(int(s[0][0:4]),int(s[0][4:6]),int(s[0][6:8])) - date(int(s[1][0:4]),int(s[1][4:6]),int(s[1][6:8]))).days <= 15:
        return 1
    else:
        return 0＃15 天以外用券消费时，也返回 0

df['coupon_apply'] = df.Date.astype('str') + ':' +  df.Date_received.astype('str')
df.coupon_apply = df.coupon_apply.apply(get_label)
df = df[['discount_rate','coupon_apply']]

df.boxplot(by = 'coupon_apply')
plt.show()
```



左边为没有核销的优惠券分布，右边为核销的优惠券分布。可以看出，核销的优惠券的分布更加分散一些，整体相对偏下，可见折扣越多对人们吸引力就越大。

4. 离散型特征与连续型标签的关系

盒图也可以用来观察离散型特征与连续型标签的关系，将输出的分类改为输入的分类，对每个输入的分类的输出画成一个盒图，然后将所有输入分类的盒图放在一起观察。

密度图(Density Plot)也可用来可视化类似关系。在密度图中，将每个离散的特征值画一条曲线，多条曲线放在一起进行比较，如图 5-3 所示。每个离散特征值的曲线的横坐标设为连续的标签值，纵坐标设为对应标签值的密度。

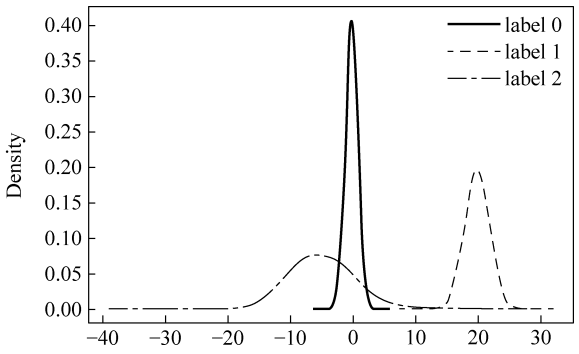


图 5-3　密度图示例(见彩插)

密度图与前面介绍过的直方图类似,描述的也是数据的分布情况,可以看成将直方图区间无限细分后形成的平滑曲线。

5. 连续型特征与连续型标签的关系

连续型特征与连续型标签的关系是常用的画图方式,即将输入、输出值对应在平面上作点,可采用 Matplotlib 和 Pandas 中的 scatter()函数。

5.1.3　数据预处理

数据预处理包括的内容很多,且处理方式依个人思路不同,体现出创新性。下面列出一些常见的预处理。

1. 有效特征与无效特征的取舍

通过观察和分析,如果发现某个特征与标签值有关联关系,则应该纳入模型训练。而与标签值无关的特征,则应该排除,否则会干扰模型学习。

作为唯一取值的样本 ID 往往是无效特征,需要舍弃。优惠券核销预测项目中有两类 ID,分别是用户 ID 和商户 ID。这两类 ID 不是每个样本唯一的,所以未必是无效特征。

对于商户 ID,猜测商户 ID 应该跟核销结果有关,因为各商户的打折促销手段不一样。可以把各商户发放的优惠券核销情况进行统计,计算一下核销率,看一看各商户的核销率的差别。对于用户 ID,同样可以计算某用户领取优惠券后的核销率。可以观察出两类 ID 对预测结果是否有影响,即是否是有效特征。

有的算法可以分析出每个特征的重要程度,如代码 4-12 中的第 10、11 行是随机森林算法分析的各特征的重要程度。

在特征选择和训练时效上,有时候需要做平衡。对于某些不重要的特征,在训练时效要求高的情况下,需要放弃。

2. 独热编码

有些样本的特征是分类特征。分类特征是在一个集合里没有次序的有限个值,如人

的性别、班级编号等。对分类特征常见的编码方式是整数,如男女性别分别表示为1、0,一班、二班、三班等分别表示为1、2、3等。但是,整数编码天然存在次序,而原来的分类特征是没有次序的。如果算法不考虑它们的差别,则会带来意想不到的后果。比如,班级分别用1、2、3、4等来编码时,如果机器学习算法忽略了次序问题,就会认为一班和二班之间的距离是1,而一班和三班之间的距离是2。

为了防止此类错误的出现,常采用独热(One-Hot)编码。假如分类特征有 n 个类别,独热编码则使用 n 位来对它们进行编码。例如,假设有四个班,则一到四班分别编码为0001、0010、0100、1000,每个编码只有一位有效。如此,任意两个班之间的距离计算都是1。

对独热码的处理,sklearn 在 preprocessing 包中提供了 OneHotEncoder 类。

3. 特征值变换

为了适合算法需要,有时需要对特征值进行某种变换。常用的变换包括平方、开方、取对数和差分运算等,即

$$\begin{aligned} x_{\text{new}} &= x^2 \\ x_{\text{new}} &= \sqrt{x} \\ x_{\text{new}} &= \log x \\ \nabla f(x_k) &= f(x_{k+1}) - f(x_k) \end{aligned} \tag{5-1}$$

举一个说明数据变换的常用例子。如果某次考试后,老师发现百分制的成绩及格率太低,想把及格率提高到一些,但不能改变成绩的次序,那怎么办呢?一个办法就是将原始成绩的平方根值乘10作为最终成绩。

4. 特征值分布处理

有些算法对特征的取值分布比较敏感,需要预先对取值的分布进行调整。在讨论 k-means 算法时已经分析过此情况,并介绍了归一化方法(2.1.3节)。除了归一化方法,对特征值的分布进行处理的常用方法还有标准化和正则化等。

标准化(Z-Score)方法是针对特征值概率分布敏感的算法而进行的调整特征值概率分布的方法。它是对某个特征进行的操作,先计算出该特征的均值和方差,然后对所有样本的该特征值减去均值并除以标准差。如样本的第 j 个特征 $x^{(j)}$ 的均值估计为 $\text{mean}x^{(j)} = \frac{1}{m}\sum_{i=1}^{m} x_i^{(j)}$,方差估计为 $\text{var}x^{(j)} = \frac{1}{m}\sum_{i=1}^{m}(x_i^{(j)} - \text{mean}x^{(j)})^2$,则 $x_i^{(j)}$ 的标准化操作为

$$\text{Z-Score}(x_i^{(j)}) = \frac{x_i^{(j)} - \text{mean}x^{(j)}}{\sqrt{\text{var}x^{(j)}}} \tag{5-2}$$

标准化操作的实质是将取值分布聚集在0附近,方差为1。实现标准化操作的有 sklearn.preprocessing 包的 scale()函数和 StandardScaler 类。

正则化(Normalization)是对样本进行的操作,它先计算每个样本的所有特征值的 p

范数，然后将该样本中的每个特征除以该范数，其结果是使得每个样本的特征值组成的向量的 p 范数等于 1。范数的计算见式(2-11)。实现正则化操作的有 sklearn. preprocessing 包的 normalize()函数和 Normalizer()类。

5. 缺失数据处理

有的算法能够自己处理缺失值。当使用不能处理缺失值的算法时，或者是想自己处理缺失值时，有以下两种方法：一是删除含有缺失的样本或者特征，这种方法会造成信息损失，可能会训练出不合理的模型；二是补全缺失值，这是常用的方法，即用某些值来代替缺失值。补全缺失值的目的是想最大限度地利用数据，以提高预测成功率。对于补全缺失值，有两种做法：一种做法是将所有的缺失值都分为一类，相当于将缺失值的这些样本作为一个子集来训练；另一种做法是插补(Imputation)，即用最可能的值来代替缺失值。插补的做法，实际上也是预测，也就是说，是采用机器学习的方法来补全机器学习所需要的样本。

常用的插补方法有：①均值、众数插补，对于连续型特征，可采用均值来插补缺失数据，对于离散性的特征，可采用众数来插补缺失数据。②建模预测，利用其他特征值来建模预测缺失的特征值，它的做法与机器学习的方法完全一样，将缺失特征作为待预测的标签，将未缺失的样本划分训练集和验证集并训练模型。③插值，利用本特征的其他值来建模预测缺失值，常用的方法是拉格朗日插值法和牛顿插值法(在 scipy. interpolate. lagrange 包中提供了拉格朗日插值法函数)。

6. 异常数据处理

异常数据是明显偏离其他值的特征值。异常数据也称为离群点。通过统计分析可以发现离群点。统计分析中，常采用 3σ 原则来筛选离群点。3σ 是正态分布中，与均值超过 3 倍标准差的值。在正态分布的假设下，距离平均值 3σ 之外的值出现的概率要小于 0.003，属于极个别的小概率事件。

当发现离群点后，首先要结合领域知识进行分析，确定该值的出现到底是不是合理的。有些异常值的出现可能蕴含着规律的改变，是发现特殊规律的契机。如果是合理的异常值，则不需要处理，可以直接进行训练。如果是不合理的异常值，可删除该值，并按缺失值的处理方法进行处理。

5.2　线性降维

从实例的属性数据中提取特征时，有可能会提取特别多的特征，即出现了高维数据。在高维时，样本在空间中分布会很稀疏，距离计算也要复杂得多。高维数据带来的问题被称为“维数灾难”(Curse of Dimensionality)。此外，还有可能提取出两种线性相关的特征，即出现了特征冗余。降维是解决上述问题的方法，它将高维空间中的点映射到低维空间中。可通过多种方法来达到降维的目的，比如通过随机森林算法分析出特征的重要

排序,直接去掉排名靠后的特征,这是一种比较“硬”的方法。降维算法是专门用于降维的算法,可以分为线性和非线性的。线性的降维算法是基于线性变换来降维,主要有奇异值分解、主成分分析等算法。

在数据可视化时,也经常用到降维算法,比如将高维数据降成直观可视的二维平面上或三维立体空间中的数据。

5.2.1 奇异值分解

矩阵的奇异值分解(Singular Value Decomposition)是矩阵论的重要内容,在机器学习算法中有重要应用,此处仅讨论相关结论及其在降维中的应用,有关概念和推导的详细内容可参考矩阵论相关书籍。

$m \times n$ 的矩阵 $\boldsymbol{A}$ 的奇异值分解式为

$$\boldsymbol{A} = \boldsymbol{U} \begin{bmatrix} \boldsymbol{\Sigma} & \boldsymbol{O} \\ \boldsymbol{O} & \boldsymbol{O} \end{bmatrix} \boldsymbol{V}^{\mathrm{T}} \tag{5-3}$$

其中,$\boldsymbol{U}$ 是 $m \times m$ 的酉矩阵;$\boldsymbol{V}$ 是 $n \times n$ 的酉矩阵,即 $\boldsymbol{U}^{\mathrm{T}}\boldsymbol{U} = \boldsymbol{I}$,$\boldsymbol{V}^{\mathrm{T}}\boldsymbol{V} = \boldsymbol{I}$。$\begin{bmatrix} \boldsymbol{\Sigma} & \boldsymbol{O} \\ \boldsymbol{O} & \boldsymbol{O} \end{bmatrix}$ 是 $m \times n$ 的矩阵,$\boldsymbol{\Sigma} = \mathrm{diag}(\sigma_1, \sigma_2, \cdots, \sigma_r)$ 为对角矩阵,$\sigma_1 \geqslant \sigma_2 \geqslant \cdots \geqslant \sigma_r$ 为矩阵 $\boldsymbol{A}$ 的全部非零奇异值。

一般来说,奇异值序列下降非常快,奇异值分解用于降维就是利用了这一点,它用最大的 r 个奇异值及对应的左右奇异向量来近似原矩阵,即

$$\boldsymbol{A}_{m \times n} = \boldsymbol{U}_{m \times m} \begin{bmatrix} \boldsymbol{\Sigma} & \boldsymbol{O} \\ \boldsymbol{O} & \boldsymbol{O} \end{bmatrix}_{m \times n} \boldsymbol{V}^{\mathrm{T}}_{n \times n} \approx \boldsymbol{U}_{m \times r} \boldsymbol{\Sigma}_{r \times r} \boldsymbol{V}^{\mathrm{T}}_{r \times n} \tag{5-4}$$

如果 $\boldsymbol{A}_{m \times n}$ 是按行排列的样本,即 m 是样本数量,n 是特征数量,对上式右乘 $\boldsymbol{V}_{n \times r}$ 可得到降维后的样本特征矩阵:

$$\boldsymbol{A}'_{m \times r} = \boldsymbol{A}_{m \times n} \boldsymbol{V}_{n \times r} = \boldsymbol{U}_{m \times r} \boldsymbol{\Sigma}_{r \times r} \boldsymbol{V}^{\mathrm{T}}_{r \times n} \boldsymbol{V}_{n \times r} = \boldsymbol{U}_{m \times r} \boldsymbol{\Sigma}_{r \times r} \tag{5-5}$$

可见,样本特征矩阵从 $m \times n$ 维度降到了 $m \times r$。

numpy.linalg 包中提供了奇异值分解函数 svd。来看一个奇异值分解降维的简单例子,示例代码见代码 5-5。

代码 5-5 奇异值分解降维示例(奇异值分解降维.ipynb)

```
>>> import numpy as np
# 用一个 3×4 的矩阵简单观察奇异值分解的效果
>>> A = np.mat([[1,2,3,4],[5,6,7,8],[9,10,11,12]])
>>> print(A)
[[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]]
# U 为左奇异向量,V 为右奇异向量,sigma 为奇异值对角矩阵
>>> U, sigma, VT = np.linalg.svd(A)
>>> print("U", U)
```

```
11. U [[ -0.20673589  0.88915331  0.40824829]
12.  [ -0.51828874  0.25438183 -0.81649658]
13.  [ -0.82984158 -0.38038964  0.40824829]]
14. >>> print("sigma", sigma)
15. sigma [2.54368356e+01   1.72261225e+00   1.73014261e-15]
16. >>> print("VT", VT)
17. VT [[ -0.40361757 -0.46474413 -0.52587069 -0.58699725]
18.  [ -0.73286619 -0.28984978  0.15316664  0.59618305]
19.  [ 0.5089281   -0.82947951  0.1321747   0.1883767 ]
20.  [ 0.20246527  0.10937892 -0.82615365  0.51430946]]
21. #降为一维,sigma_c,U_c,VT_c 为保存的矩阵
22. >>> sigma_c = np.diag(sigma[0:1])
23.>>> print(sigma_c)
24. [[ 25.43683563]]
25. >>> U_c = U[:,0:1]
26. >>> print(U_c)
27. [[ -0.20673589]
28.  [ -0.51828874]
29.  [ -0.82984158]]
30. >>> VT_c = VT[0:1,:]
31. >>> print(VT_c)
32. [[ -0.40361757 -0.46474413 -0.52587069 -0.58699725]]
33. #近似还原
34. >>> print("conv A", U_c * sigma_c * VT_c)
35. conv A [[2.12250651   2.44395316   2.76539981   3.08684646]
36.  [5.32114289   6.12701254   6.93288219   7.73875183]
37.  [8.51977928   9.81007192  11.10036456  12.3906572 ]]
38. #降维
39. >>> print(A * VT_c.T)
40. [[ -5.25870689]
41.  [ -13.18362544]
42.  [ -21.10854399]]
43. print(U_c * sigma_c)
44. [[ -5.25870689]
45.  [ -13.18362544]
46.  [ -21.10854399]]
```

第 15 行为输出的奇异值,可见第 1 个奇异值远远大于后面两个,所以可以去掉后面两个来降成一维。对比第 5 行的原始矩阵和第 35 行的还原矩阵,可以发现存在少量误差。从第 22 行到第 32 行示例了保存的矩阵,可见保存的数值数量少于原矩阵数值数量。实际应用中的大矩阵降维会大大减少需要保存的数量。

可以看到,在第 15 行输出的奇异值中,第 3 个奇异值很小,如果将其去掉,对原矩阵影响很小。读者可以自行修改参数,降成二维,看看还原后的矩阵与原矩阵的差别。

上面从实验的角度说明了去掉很小的奇异值及其奇异向量给原矩阵带的影响很小。但奇异值的含义并不好解释,所以称该方法为“黑盒”方法。

奇异值分解降维的基本流程如算法 5-1 所示。

算法 5-1　奇异值分解降维算法的基本流程

步　数	操　作
1	设定降维的维数 r
2	对样本特征矩阵 A 进行奇异值分解
3	取最大的 r 个奇异值及对应的右奇异向量组成降维矩阵
4	对样本特征矩阵 A 右乘降维矩阵，得到低维样本特征矩阵


视频

5.2.2　主成分分析

主成分分析(Principal Components Analysis，PCA)是最常使用的降维方法之一。顾名思义，主成分分析是找出主要成分来代替原来数据。主成分分析应用非常广泛，本节从容易理解的低维空间开始讨论，然后推广到高维空间中。

主成分分析降维，实际上是丢弃样本点在空间中的某些坐标分量。来看一个从二维降成一维的示例，如图 5-4 所示。在二维平面上有 $\boldsymbol{x}_1$、$\boldsymbol{x}_2$、$\boldsymbol{x}_3$、$\boldsymbol{x}_4$ 四个点，具体坐标如图中所示，它们满足中心化要求，即 $\sum_{i=1}^{4}\boldsymbol{x}_i=0$。

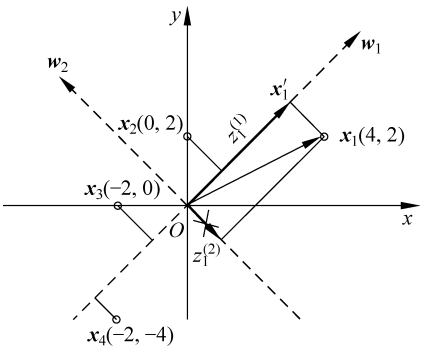


图 5-4　二维平面上的主成分分析示例(见彩插)

$\boldsymbol{x}$ 和 $\boldsymbol{y}$ 是平面常用的标准正交基向量(1,0)和(0,1)。$\boldsymbol{w}_1$ 和 $\boldsymbol{w}_2$ 是平面的另一组标准正交基向量，即满足 $\|\boldsymbol{w}_i\|_2=\boldsymbol{w}_i\boldsymbol{w}_i^{\mathrm{T}}=1$，$\boldsymbol{w}_i\boldsymbol{w}_j^{\mathrm{T}}=0$，这里的 $\|\cdot\|_2$ 是指向量的 2 范数。

点 $\boldsymbol{x}_1$ 在基 $\boldsymbol{w}_1$ 和 $\boldsymbol{w}_2$ 下的投影(坐标值)如图中粗虚线所示。如果丢弃其中一个坐标，不妨设为 $\boldsymbol{w}_2$ 上的坐标(图中打叉的投影)，那么点 $\boldsymbol{x}_1$ 就变成了新坐标系中 $\boldsymbol{w}_1$ 轴上的点 $\boldsymbol{x}_1'$，它在 $\boldsymbol{w}_1$ 轴上的坐标值记为 $z_1^{(1)}$(下标 1 表示来自对应点 $\boldsymbol{x}_1$，上标(1)表示在基 $\boldsymbol{w}_1$ 上的投影)。$\boldsymbol{x}_1'$ 用向量表示为 $\boldsymbol{x}_1'=z_1^{(1)}\boldsymbol{w}_1$，即方向由基 $\boldsymbol{w}_1$ 确定，模为 $z_1^{(1)}$。

显然，丢弃一个坐标使得 $\boldsymbol{x}_1$ 只能用 $\boldsymbol{x}_1'$ 来代表(或者叫恢复)，因此要带来误差，该恢复误差可以用 $\boldsymbol{x}_1$ 和 $\boldsymbol{x}_1'$ 之差的 2-范数的平方来衡量，即 $\|\boldsymbol{x}_i-\boldsymbol{x}_i'\|_2^2$，在欧氏空间中即为 $\boldsymbol{x}_1$ 和 $\boldsymbol{x}_1'$ 的欧氏距离平方。

同样可分析其他点在丢弃 $\boldsymbol{w}_2$ 上的坐标时产生的误差。

希望降维带来的恢复误差尽可能小，即要使所有点的误差和最小，称为极小化恢复

误差。因此，优化模型为

$$\min_{\boldsymbol{w}_1,\boldsymbol{w}_2}\sum_{i=1}^{4}\|\boldsymbol{x}_i-\boldsymbol{x}'_i\|_2^2$$

$$\text{s.t.}\quad \boldsymbol{w}_i\boldsymbol{w}_i^{\mathrm{T}}=1,\quad \boldsymbol{w}_i\boldsymbol{w}_j^{\mathrm{T}}=0 \tag{5-6}$$

展开式(5-6)：

$$\begin{aligned}\sum_{i=1}^{4}\|\boldsymbol{x}_i-\boldsymbol{x}'_i\|_2^2&=\sum_{i=1}^{4}\|\boldsymbol{x}_i-z_i^{(1)}\boldsymbol{w}_1\|_2^2=\sum_{i=1}^{4}(\boldsymbol{x}_i-z_i^{(1)}\boldsymbol{w}_1)(\boldsymbol{x}_i-z_i^{(1)}\boldsymbol{w}_1)^{\mathrm{T}}\\&=\sum_{i=1}^{4}\boldsymbol{x}_i\boldsymbol{x}_i^{\mathrm{T}}-\sum_{i=1}^{4}\boldsymbol{x}_i(z_i^{(1)}\boldsymbol{w}_1)^{\mathrm{T}}-\sum_{i=1}^{4}z_i^{(1)}\boldsymbol{w}_1(\boldsymbol{x}_i)^{\mathrm{T}}+\sum_{i=1}^{4}z_i^{(1)}\boldsymbol{w}_1(z_i^{(1)}\boldsymbol{w}_1)^{\mathrm{T}}\\&=\sum_{i=1}^{4}\boldsymbol{x}_i\boldsymbol{x}_i^{\mathrm{T}}-\sum_{i=1}^{4}\boldsymbol{x}_i\boldsymbol{w}_1^{\mathrm{T}}(z_i^{(1)})^{\mathrm{T}}-\sum_{i=1}^{4}z_i^{(1)}(\boldsymbol{x}_i\boldsymbol{w}_1^{\mathrm{T}})^{\mathrm{T}}+\sum_{i=1}^{4}z_i^{(1)}(z_i^{(1)})^{\mathrm{T}}\\&=\sum_{i=1}^{4}\boldsymbol{x}_i\boldsymbol{x}_i^{\mathrm{T}}-\sum_{i=1}^{4}z_i^{(1)}(z_i^{(1)})^{\mathrm{T}}\end{aligned} \tag{5-7}$$

式中，$\boldsymbol{x}_i\boldsymbol{w}_1^{\mathrm{T}}$ 是 $\boldsymbol{x}_i$ 在标准正交基 $\boldsymbol{w}_1$ 上的投影，即 $z_i^{(1)}$。$\sum_{i=1}^{4}\boldsymbol{x}_i(\boldsymbol{x}_i)^{\mathrm{T}}$ 是常数，因此，要使式(5-7)最小，即要使 $\sum_{i=1}^{4}z_i^{(1)}(z_i^{(1)})^{\mathrm{T}}=\sum_{i=1}^{4}(z_i^{(1)})^2$ 最大。直观来看，就是要调整 $\boldsymbol{w}_1$ 的角度使这 4 个点在 $\boldsymbol{w}_1$ 上投影的平方和最大(也称为最大化投影方差)。

如何来计算呢？将投影用向量乘积表示，得到

$$\begin{aligned}\sum_{i=1}^{4}z_i^{(1)}(z_i^{(1)})^{\mathrm{T}}&=\sum_{i=1}^{4}(z_i^{(1)})^{\mathrm{T}}z_i^{(1)}=\sum_{i=1}^{4}\boldsymbol{w}_1\boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{x}_i\boldsymbol{w}_1^{\mathrm{T}}=\boldsymbol{w}_1\Big[\sum_{i=1}^{4}(\boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{x}_i)\Big]\boldsymbol{w}_1^{\mathrm{T}}\\&=\boldsymbol{w}_1\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}\boldsymbol{w}_1^{\mathrm{T}}\end{aligned} \tag{5-8}$$

式中，$\boldsymbol{X}=\begin{pmatrix}\boldsymbol{x}_1\\\boldsymbol{x}_2\\\boldsymbol{x}_3\\\boldsymbol{x}_4\end{pmatrix}$。

在满足中心化要求 $\sum_{i=1}^{4}\boldsymbol{x}_i=0$ 的前提下，$\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}$ 为协方差矩阵，是实对称矩阵，因此存在正交矩阵 $\boldsymbol{Q}=(\boldsymbol{q}_1,\boldsymbol{q}_2)$，使得 $\boldsymbol{Q}^{\mathrm{T}}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}\boldsymbol{Q}=\begin{pmatrix}\lambda_1&0\\0&\lambda_2\end{pmatrix}$，即对 $\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}$ 进行特征值分解。对 $\boldsymbol{Q}$ 的任一特征向量 $\boldsymbol{q}_i$，有 $\boldsymbol{q}_i^{\mathrm{T}}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}\boldsymbol{q}_i=\lambda_i$。

利用拉格朗日乘数法可以证明在约束条件 $\boldsymbol{w}_i\boldsymbol{w}_i^{\mathrm{T}}=1$，$\boldsymbol{w}_i\boldsymbol{w}_j^{\mathrm{T}}=0$ 下，式(5-8)取得的最大值为矩阵 $\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}$ 的最大特征值，此时，$\boldsymbol{w}_1$ 为对应的特征向量。最大的特征向量代表了最主要的投影成分。

来看图 5-4 中 4 个点($\boldsymbol{x}$ 和 $\boldsymbol{y}$ 基下的坐标如图中所示)的主成分降维的计算过程及结果，其代码见代码 5-6，特征值分解是用 numpy.linalg 包中的 eig 函数。

代码 5-6　主成分分析示例(主成分分析示例.ipynb)

```
>>> import numpy as np
>>> X = np.mat([[4,2],[0,2],[-2,0],[-2,-4]])
>>> print(X)
[[ 4  2]
[ 0  2]
[-2  0]
[-2 -4]]
>>> tr,W = np.linalg.eig(X.T * X)
>>> print(tr)
[40.  8.]
>>> print(W)
[[ 0.70710678 -0.70710678]
 [ 0.70710678  0.70710678]]
>>> w1 = W[:,0].T
>>> X * w1.T
matrix([[ 4.24264069],
        [ 1.41421356],
        [-1.41421356],
        [-4.24264069]])
```

第 8 行是对 $\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}$ 为协方差矩阵求特征值和特征向量。第 14 行取最大特征值 40 对应的特征向量(0.70710678,0.70710678),即 $\boldsymbol{w}_1$,它可以写成分数形式$\left(\frac{1}{\sqrt{2}},\frac{1}{\sqrt{2}}\right)$,可见它的方向是 $\boldsymbol{x}$ 和 $\boldsymbol{y}$ 轴中间 45°。第 15 行是求各点在 $\boldsymbol{w}_1$ 上的投影(坐标),可见 $\boldsymbol{x}_1$ 在 $\boldsymbol{w}_1$ 上的投影为 4.24264069,读者可以用几何知识验证。

在此例中,是从二维降到一维,即用点到线的投影来代替平面上的点。如果在三维立体空间中,可将空间中的点投影到一个平面上或者一条线上。进一步推广,可以将多维空间中的点投影到一个低维的超平面上。

推广到多维空间的求解思路与此例类似,下面给出简略过程及结论。

设样本集为 $\boldsymbol{S}=\{\boldsymbol{x}_1,\boldsymbol{x}_2,\cdots,\boldsymbol{x}_m\}$ 包含 m 个无标签样本,每个样本 $\boldsymbol{x}_i=\{x_i^{(1)},x_i^{(2)},\cdots,x_i^{(n)}\}$ 是一个 n 维特征向量,并经过了中心化,即 $\sum_{i=1}^{m}\boldsymbol{x}_i=0$。中心化是对每个样本 $\boldsymbol{x}_i$ 减去均值 $\bar{\boldsymbol{x}}$ 的操作。经过投影变换后的新坐标系为 $\boldsymbol{W}=\begin{pmatrix}\boldsymbol{w}_1\\ \vdots\\ \boldsymbol{w}_n\end{pmatrix}$,其中 $\boldsymbol{w}_i$ 为标准正交基。将样本从 n 维降为 n' 维,即丢弃新坐标系中部分坐标,不妨设新坐标系为 $\boldsymbol{W}'=\begin{pmatrix}\boldsymbol{w}_1\\ \vdots\\ \boldsymbol{w}_{n'}\end{pmatrix}$,样本点在新坐标系中的投影为 $\boldsymbol{z}_i=\{z_i^{(1)},z_i^{(2)},\cdots,z_i^{(n')}\}$,其中 $z_i^{(j)}=\boldsymbol{x}_i\boldsymbol{w}_j^{\mathrm{T}}$ 是 $\boldsymbol{x}_i$ 在低维坐标系里第 j 维的投影。

用 $\boldsymbol{z}_i$ 来恢复原样本点 $\boldsymbol{x}_i$，得到恢复点为 $\boldsymbol{x}'_i=\sum_{i=1}^{n'} z_i^{(j)}\boldsymbol{w}_i=\boldsymbol{z}_i\boldsymbol{W}'$，则误差为 $\|\boldsymbol{x}_i-\boldsymbol{x}'_i\|_2^2$。基于最小化恢复误差的思想，可知优化模型为

$$\begin{aligned}&\min_{\boldsymbol{W}}\sum_{i=1}^{m}\|\boldsymbol{x}_i-\boldsymbol{x}'_i\|_2^2\\&\text{s.t.}\quad \boldsymbol{w}_i\boldsymbol{w}_i^{\mathrm{T}}=1,\quad \boldsymbol{w}_i\boldsymbol{w}_j^{\mathrm{T}}=0\end{aligned}\tag{5-9}$$

整理误差和，利用矩阵的迹的性质，可得

$$\sum_{i=1}^{m}\|\boldsymbol{x}_i-\boldsymbol{x}'_i\|_2^2=\sum_{i=1}^{m}\|\boldsymbol{x}_i-\boldsymbol{z}_i\boldsymbol{W}'\|_2^2=\sum_{i=1}^{m}\boldsymbol{x}_i\boldsymbol{x}_i^{\mathrm{T}}-\mathrm{tr}(\boldsymbol{W}'\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}\boldsymbol{W}'^{\mathrm{T}})\tag{5-10}$$

其中 tr(·)为矩阵的迹。即

$$\begin{aligned}&\min_{\boldsymbol{W}}-\mathrm{tr}(\boldsymbol{W}'\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}\boldsymbol{W}'^{\mathrm{T}})\\&\text{s.t.}\quad \boldsymbol{w}_i\boldsymbol{w}_i^{\mathrm{T}}=1,\quad \boldsymbol{w}_i\boldsymbol{w}_j^{\mathrm{T}}=0\end{aligned}\tag{5-11}$$

利用拉格朗日乘数法将条件极值转化为无条件极值，可证明：取协方差矩阵 $\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}$ 最大的 n' 个特征值对应的特征向量组成 $\boldsymbol{W}'$，可使式(5-11)取得最小值。

如果把恢复误差看成是样本点到超平面的距离平方(如图 5-4 中 $\boldsymbol{x}_1$ 到 $\boldsymbol{w}_1$ 的距离所示)，那么极小化恢复误差就是极小化样本点到超平面距离的平方。

上述分析过程，还有另一种解释，即最大化投影方差。投影方差是指样本点在超平面上的投影距离中心的偏离程度。最大化投影方差使这个偏离程度越大越好，即通过调整该超平面使各投影点最大可能地分开。以图 5-4 所示示例来对该解释进行简要讨论。因为各样本点已经进行了中心化，因此，原点 O 即为各样本点的均值点。对 $\boldsymbol{x}_1$ 来说，它的投影方差是 $(z_1^{(1)})^2$，它的恢复误差是 $(z_1^{(2)})^2$。不管基向量如何调整方向，点 $\boldsymbol{x}_1$ 到原点 O 的距离 $\|\boldsymbol{x}_1\|_2^2$ 不变，而 $\|\boldsymbol{x}_1\|_2^2=(z_1^{(1)})^2+(z_1^{(2)})^2$。因此，最大化投影方差，即为最小化恢复误差。

主成分分析降维算法的基本流程如算法 5-2 所示。

算法 5-2　主成分分析降维算法的基本流程

步　数	操　作
1	设定降维的维数 n'
2	计算所有样本的均值 $\bar{x}$，减各样本 x_i 完成中心化
3	计算样本的协方差矩阵 $\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}$，并进行特征值分解
4	取最大的 n' 个特征值对应的特征向量组成投影矩阵
5	用样本特征矩阵乘投影矩阵得到低维样本特征矩阵

sklearn 在 decomposition 包中提供了实现主成分分析的 PCA 类。

5.3 超参数调优

超参数调优需要依靠实验的方法,以及人的经验。对算法本身的理解越深入,对实现算法的过程了解越详细,积累了越多的调优经验,就越能够快速准确地调优。

实验的方法,就是设置了一组超参数之后,用训练集来训练并用验证集来检验,多次重复以上过程,取效果最好的那组超参数。训练数据的划分可以采用保持法,也可以采用 k-折交叉验证法。超参数调优的实验方法主要有两种:网格搜索和随机搜索。

5.3.1 网格搜索

网格搜索方法类似于网格聚类的做法,它将各超参数形成的空间划分为若干小空间,在每一个小空间上取一组值作为代表进行实验。取效果最好的那组值作为最终的超参数值。

网格搜索的实现比较容易,下面用优惠券核销的例子来示例,对 sklearn 中随机森林算法 RandomForestClassifier 的 n_estimators 和 max_depth 两个参数进行网格搜索。

n_estimators 是弱学习器的最大迭代次数,或者说最大的弱学习器的个数,默认是 10。一般来说 n_estimators 太小,容易欠拟合,n_estimators 太大,又容易过拟合,一般选择一个适中的数值。

max_depth 是决策树最大深度。如果样本数量和特征都很多时,应限制最大深度,具体取值取决于数据的规模和分布。max_depth 常用 10～100 之间。

示例代码见代码 5-7。

代码 5-7 超参数调优网格搜索示例(use_rfc_gridsearch.py)

```
### 3.网格搜索,分类器采用随机森林算法
print('3.网格搜索')
print('\n\n------ 网格搜索', file = file_print_to)

time_start = time.time()

### - 3.1.设置二维网格,分别是 n_estimators 和 max_depth 两个参数
list_n_estimators = range(10, 71, 10)
list_max_depth = range(3,14,2)

### - 3.2.重复设置参数,并进行训练和预测
from sklearn.ensemble import RandomForestClassifier
for n_estimators in list_n_estimators:
    for max_depth in list_max_depth:
        print('\n n_estimators, max_depth:' + str(n_estimators) + ',' + str(max_depth))
```

```
            print('\n n_estimators, max_depth:' + str(n_estimators) + ',' + str(max_depth), \
                  file = file_print_to)
            rfc = RandomForestClassifier(random_state = 2, n_estimators = n_estimators, \
                                         max_depth = max_depth)
            rfc.fit(X_train, y_train)
            print('准确率: ' + str(rfc.score(X_verify, y_verify)), file = file_print_to)
            y_pred = rfc.predict(X_verify)
            from sklearn.metrics import confusion_matrix
            cm = confusion_matrix(y_verify, y_pred)
            print('平均准确率: ' + str(cal_average_perclass_accuracy(cm)), file = file_
    print_to)
            from sklearn.metrics import roc_auc_score
            predict_prob_y = rfc.predict_proba(X_verify)
            auc = roc_auc_score(y_verify, predict_prob_y[:, 1])
            print('\n 总 AUC:' + str(auc), file = file_print_to)
            rq = pd.DataFrame({'Coupon_id':features_verify['Coupon_id'], \
                               'coupon_apply':y_verify, 'prob_y':predict_prob_y[:, 1]})
            print('\n 平均 AUC is:' + str(meanAuc(rq)), file = file_print_to)

print('训练用时: ' + str(time.time() - time_start), file = file_print_to)
```

主要是利用两个 for 循环，分别对两个参数进行赋值，每次训练后，用验证集来计算准确率、平均准确率、总 AUC 和平均 AUC 四个指标，得到结果见表 5-1。通过阴影标出的数据分别是四个指标的最大值。

表 5-1　超参数调优网格搜索示例结果

n_estimators	max_depth					
	3	5	7	9	11	13
10	0.9094 0.5 0.7840 0.5496	0.9094 0.5 0.7928 0.5429	0.9095 0.5005 0.7775 0.5699	0.9103 0.5082 0.7348 0.5851	0.9093 0.5199 0.7157 0.5943	0.9014 0.5266 0.7133 0.5874
20	0.9094 0.5 0.7833 0.5526	0.9094 0.5 0.7924 0.5400	0.9094 0.5006 0.7811 0.5775	0.9107 0.5113 0.7515 0.5877	0.9114 0.5202 0.7153 0.5995	0.9038 0.5236 0.7058 0.5975
30	0.9094 0.5 0.7848 0.5405	0.9094 0.5 0.7939 0.5598	0.9095 0.5008 0.7813 0.5837	0.9105 0.5097 0.7508 0.5930	0.9113 0.5191 0.7184 0.6039	0.9061 0.5251 0.7044 0.6010

续表

n_estimators	max_depth					
	3	5	7	9	11	13
40	0.9094 0.5 0.7865 0.5384	0.9094 0.5 0.7959 0.5532	0.9095 0.5008 0.7839 0.5841	0.9104 0.5092 0.7572 0.5961	0.9112 0.5185 0.7267 0.6041	0.9099 0.5269 0.7099 0.6050
50	0.9094 0.5 0.7891 0.5436	0.9094 0.5 0.7955 0.5515	0.9094 0.5007 0.7797 0.5788	0.9104 0.5094 0.7524 0.6058	0.9114 0.5197 0.7255 0.6093	0.9085 0.5259 0.7034 0.6030
60	0.9094 0.5 0.7902 0.5413	0.9094 0.5 0.7955 0.5580	0.9094 0.5007 0.7810 0.5817	0.9104 0.5086 0.7551 0.6059	0.9114 0.5197 0.7279 0.6097	0.9095 0.5260 0.7084 0.6047
70	0.9094 0.5 0.7897 0.5441	0.9094 0.5 0.7952 0.5610	0.9094 0.5008 0.7837 0.5885	0.9105 0.5099 0.7569 0.6114	0.9114 0.5198 0.7305 0.6113	0.9096 0.5256 0.7069 0.6061

由表 5-1 可见，当采用不同评价指标时，最优值出现在不同网格。

需要注意的是，这种暴力的方法，只适合于小样本量、少参数的情况，否则效率很低。可以作适当改进：①在影响大的参数上作更细的划分，而在影响小的参数上作粗的划分；②先将网格粗切分，然后再对最好的网格进行细切分；③还有一种改进效率的贪心搜索方法，先在影响最大的参数上进行一维搜索，找到最优参数，然后固定它，再在余下参数中影响最大参数上进行一维搜索，如此下去，直到搜索完所有参数。这种贪心搜索方法的时间复杂度为参数总数的线性函数，而网格搜索方法的时间复杂度为参数总数的指数函数。但贪心搜索方法可能会收敛到局部最优值。

sklearn. grid_search. GridSearchCV 函数提供了网格搜索的功能。它要调用具体的分类器来进行分类，它要求被调用的分类器实现了 fit、predict、score 等方法。

5.3.2 随机搜索

随机搜索的思想和网格搜索比较相似，只是不固定分隔子空间，而是随机分隔。它将每个特征的取值都看成是一个分布，然后依概率从中取值。每轮实验中，每个特征取一个值，进行模型训练。随机搜索一般会比网格搜索要快一些，但是无法保证得到最优超参数值。

在 sklearn. model_selection. RandomizedSearchCV 中实现了随机搜索。

5.4　练习题

1. 用马赛克图分析优惠券核销示例中的周一到周日的领券核销情况。

2. 代码 5-5 所示的奇异值分解例子中，第 3 个奇异值很小，如果将其去掉，对原矩阵影响很小。自行修改代码 5-5 中的参数，将原矩阵降成二维，再还原。比较还原后的矩阵与原矩阵的差别。

3. 探索 sklearn 的 decomposition 包中的 PCA 类的应用。如用它对本章两个降维示例进行降维处理。