第3章 特征工程

机器学习所使用的数据和特征决定了机器学习算法构建的模型的上限，更好的特征使得模型能够训练出更好的结果。为了使构建的模型尽可能逼近最优，需要在建模前对特征进行处理。特征工程是使用专业背景知识和技巧处理数据，使得特征能在机器学习算法上发挥更好作用的过程，特征工程包含特征变换和特征选择等步骤。

3.1 特征变换

通常情况下，使用原始数据直接建模的效果往往不好，为了使建立的模型简单精确，需要对原始数据进行特征变换，把原始的特征转化为更为有效的特征，常用的特征变换方法有标准化、独热编码和离散化等。

3.1.1 标准化

不同特征之间往往具有不同的量纲，由此所造成的数值间的分布差异可能会很大，在涉及空间距离计算或梯度下降法等情况时，不对其进行处理会影响数据分析结果的准确性。为了消除特征之间量纲和取值范围造成的影响，需要对数据进行标准化处理。常用数据标准化方法有离差标准化、标准差标准化、小数定标标准化和函数转换等。

1. 离差标准化

离差标准化是对原始数据的一种线性变换，结果是将原始数据的数值映射到[0,1]内，转换公式如式（3-1）所示。

$$X^* = \frac{x - \min(x)}{\max(x) - \min(x)} \tag{3-1}$$

在式(3-1)中，$\max(x)$为样本数据的最大值，$\min(x)$为样本数据的最小值，$\max(x) - \min(x)$为极差。离差标准化保留了原始数据之间的联系，是消除量纲和数据取值范围影响最简单的方法，但受离群点影响较大，适用于分布较为均匀的数据。

2. 标准差标准化

标准差标准化也称为零均值标准化或 z 分数标准化，是当前使用最广泛的数据标准化方法。经过该方法处理的数据均值为 0，标准差为 1，转换公式如式（3-2）所示。

$$X^* = \frac{x - \bar{x}}{\delta} \tag{3-2}$$

在式（3-2）中，$\bar{x}$ 为原始数据的均值，δ 为原始数据的标准差。标准差标准化适用于数据的最大值和最小值未知的情况，或数据中包含超出取值范围的离群点的情况。

3. 小数定标标准化

通过移动数据的小数位数，将数据映射到[−1,1]，移动的小数位数取决于数据绝对值的最大值，转换公式如式（3-3）所示，在式（3-3）中，k 表示数据整数位个数。

$$X^* = \frac{x}{10^k} \tag{3-3}$$

例 3-1 1888 年瑞士政府对瑞士法语的 47 个省份的生育能力和经济指标进行测量，并进行了相应的类别划分，得到的部分数据如表 3-1 所示。

表 3-1 部分瑞士省份的生育能力和经济指标测量数据

省份	生育能力	农业男性百分比	军队最高教育水平百分比	小学以上教育百分比	天主教百分比	婴儿死亡率	类别
Courtelary	80.2	17	15	12	9.96	22.2	1
Delemont	83.1	45.1	6	9	84.84	22.2	2
Franches-Mnt	92.5	39.7	5	5	93.4	20.2	2
Moutier	85.8	36.5	12	7	33.77	20.3	1
Neuveville	76.9	43.5	17	15	5.16	20.6	1

利用 Python 对表 3-1 的测量数据进行标准化，如代码 3-1 所示。

代码 3-1 数据标准化

In[1]:

```
import pandas as pd
import numpy as np
# 导入数据
data = pd.read_csv('../data/swiss.csv',encoding = 'GBK')
data = data.iloc[:,1:7]
# 极差标准化
def MaxMinScale(data):
    m_scale = (data-data.min( ))/(data.max( )-data.min( ))
    return m_scale
data_m_scale = MaxMinScale(data)
print('离差标准化之前的前 10 行数据为：\n',data.ix[0:9,:],'\n',
      '离差标准化之后的前 10 行数据为：\n',data_m_scale.ix[0:9,:])
```

Out[1]:

```
离差标准化之前的前 10 行数据为：
   0    1    2    3
0  5.1  3.5  1.4  0.2
1  4.9  3.0  1.4  0.2
2  4.7  3.2  1.3  0.2
3  4.6  3.1  1.5  0.2
4  5.0  3.6  1.4  0.2
5  5.4  3.9  1.7  0.4
6  4.6  3.4  1.4  0.3
7  5.0  3.4  1.5  0.2
8  4.4  2.9  1.4  0.2
9  4.9  3.1  1.5  0.1
```

离差标准化之后的前 10 行数据为：

	生育能力	农业男性百分比	军队最高教育水平百分比	小学以上教育百分比	天主教百分比	婴儿死亡率
0	0.786087	0.178531	0.352941	0.211538	0.079816	0.721519
1	0.836522	0.496045	0.088235	0.153846	0.845069	0.721519
2	1.000000	0.435028	0.058824	0.076923	0.932550	0.594937
3	0.883478	0.398870	0.264706	0.115385	0.323148	0.601266
4	0.728696	0.477966	0.411765	0.269231	0.030761	0.620253
5	0.714783	0.385311	0.176471	0.115385	0.903628	1.000000
6	0.848696	0.779661	0.382353	0.115385	0.926929	0.810127
7	0.998261	0.752542	0.323529	0.134615	0.970976	0.892405
8	0.824348	0.588701	0.264706	0.115385	0.976188	0.645570
9	0.833043	0.497175	0.382353	0.230769	0.911906	0.860759

In[2]:

```
#标准差标准化
def StandarScale(data):
    S_scale = (data-data.mean( ))/(data.std( ))
    return S_scale
data_s_scale = StandarScale(data)
print('标准差标准化之前的前 10 行数据为：\n',data.ix[0:9,:],'\n',
      '标准差标准化之后的前 10 行数据为：\n',data_s_scale.ix[0:9,:])
```

Out[2]:

标准差标准化之前的前 10 行数据为：

	生育能力	农业男性百分比	军队最高教育水平百分比	小学以上教育百分比	天主教百分比	婴儿死亡率
0	80.2	17.0	15	12	9.96	22.2
1	83.1	45.1	6	9	84.84	22.2
2	92.5	39.7	5	5	93.40	20.2
3	85.8	36.5	12	7	33.77	20.3
4	76.9	43.5	17	15	5.16	20.6
5	76.1	35.3	9	7	90.57	26.6
6	83.8	70.2	16	7	92.85	23.6
7	92.4	67.8	14	8	97.16	24.9
8	82.4	53.3	12	7	97.67	21.0
9	82.9	45.2	16	13	91.38	24.4

标准差标准化之后的前 10 行数据为：

	生育能力	农业男性百分比	军队最高教育水平百分比	小学以上教育百分比	天主教百分比	婴儿死亡率
0	0.805131	−1.482068	−0.186686	0.106213	−0.747727	0.775037
1	1.037285	−0.244794	−1.314805	−0.205787	1.047748	0.775037
2	1.789785	−0.482562	−1.440152	−0.621786	1.253000	0.088388
3	1.253428	−0.623462	−0.562726	−0.413786	−0.176810	0.122720
4	0.540955	−0.315244	0.064007	0.418212	−0.862821	0.225718
5	0.476913	−0.676299	−0.938766	−0.413786	1.185142	2.285664
6	1.093322	0.860387	−0.061340	−0.413786	1.239812	1.255691
7	1.781779	0.754712	−0.312033	−0.309787	1.343157	1.702013
8	0.981248	0.116261	−0.562726	−0.413786	1.355386	0.363047
9	1.021274	−0.240391	−0.061340	0.210212	1.204564	

In[3]:

```
#小数定标标准化
def DecimalScale(data):
    k = np.ceil(np.log10(data.abs( ).max( )))
    D_scale = data/(10**k)
    return D_scale
data_d_scale = DecimalScale(data)
print('小数定标标准化之前的前 10 行数据为：\n',data.ix[0:9,:],'\n',
      '小数定标标准化之后的前 10 行数据为：\n',data_d_scale.ix[0:9,:])
```

Out[3]:

小数定标标准化之前的前 10 行数据为：

	生育能力	农业男性百分比	军队最高教育水平百分比	小学以上教育百分比	天主教百分比	婴儿死亡率
0	80.2	17.0	15	12	9.96	22.2
1	83.1	45.1	6	9	84.84	22.2
2	92.5	39.7	5	5	93.40	20.2
3	85.8	36.5	12	7	33.77	20.3
4	76.9	43.5	17	15	5.16	20.6
5	76.1	35.3	9	7	90.57	26.6
6	83.8	70.2	16	7	92.85	23.6
7	92.4	67.8	14	8	97.16	24.9
8	82.4	53.3	12	7	97.67	21.0
9	82.9	45.2	16	13	91.38	24.4

小数定标标准化之后的前 10 行数据为：

	生育能力	农业男性百分比	军队最高教育水平百分比	小学以上教育百分比	天主教百分比	婴儿死亡率
0	0.802	0.170	0.15	0.12	0.0996	0.222
1	0.831	0.451	0.06	0.09	0.8484	0.222
2	0.925	0.397	0.05	0.05	0.9340	0.202
3	0.858	0.365	0.12	0.07	0.3377	0.203
4	0.769	0.435	0.17	0.15	0.0516	0.206
5	0.761	0.353	0.09	0.07	0.9057	0.266
6	0.838	0.702	0.16	0.07	0.9285	0.236
7	0.924	0.678	0.14	0.08	0.9716	0.249
8	0.824	0.533	0.12	0.07	0.9767	0.210
9	0.829	0.452	0.16	0.13	0.9138	0.244

注：由于数据较多，此处部分结果已省略。

根据代码 3-1 的结果可以看出，离差标准化前后数据的整体分布情况并未发生变化，原来取值较大的数据，离差标准化之后的值仍然较大，而且经过离差标准化以后数据集中于(0,1)之间；标准差标准化前后数据的整体分布情况未发生改变，然而标准差标准化之后的数据取值不局限于(0,1)之间，而且存在负值；对比以上三种方法发现，离差标准化方法简单，便于理解，数据标准化以后取值在(0,1)之间，标准差标准化受数据分布影响较小，小数定标标准化方法适用范围较广，且受数据分布影响较小，该方法适用程度适中。

4. 函数转换

函数转换是使用数学函数对原始数据进行转换，改变原始数据的特征，使特征变得更适合建模，常用的函数转换方法包括平方、开方、取对数、差分运算等，分别如式（3-4）、式（3-5）、式（3-6）、式（3-7）所示。

$$X^* = x^2 \tag{3-4}$$

$$X^* = \sqrt{x} \tag{3-5}$$

$$X^* = \log(x) \tag{3-6}$$

$$\nabla f(x_k) = f(x_{k+1}) - f(x_k) \tag{3-7}$$

函数转换常用来将不具有正态分布的数据转换成具有正态分布的数据。在时间序列分析中，简单的对数变换或者差分运算常常就可以将非平稳序列转换成平稳序列。还可以使用对数函数转换和反正切函数转换等函数转换方法对数据进行标准化。对数函数转换是指利用以10为底的对数函数对数据进行转换，即 $X^* = \log_{10}(x)$；反正切函数转换即使用 $X^* = \frac{2\arctan(x)}{\pi}$ 对数据进行转换。如果要求反正切函数转换的结果全部落入[0,1]，那么要求原始数据全部大于等于0，否则小于0的数据会被映射到[−1,0]。

例 3-2 利用 Python 对表 3-1 介绍的测量数据使用函数转换进行标准化处理，如代码 3-2 所示。

代码 3-2 使用函数转换标准化数据

In[1]:

```
import pandas as pd
import numpy as np
# 导入数据
data = pd.read_csv('../data/swiss.csv',encoding = 'GBK')
data = data.iloc[:,1:7]
# 对数函数转换
def LogNorm(data):
    L_norm = np.log10(data)
    return L_norm
data_l_norm = LogNorm(data)
print('对数函数转换前的前 10 行数据为：\n',data.ix[0:9,:],'\n',
      '对数函数转换后的前 10 行数据为：\n',data_l_norm.ix[0:9,:])
```

Out[1]:

对数函数转换前的前 10 行数据为：

	生育能力	农业男性百分比	军队最高教育水平百分比	小学以上教育百分比	天主教百分比	婴儿死亡率
0	80.2	17.0	15	12	9.96	22.2
1	83.1	45.1	6	9	84.84	22.2
2	92.5	39.7	5	5	93.40	20.2
3	85.8	36.5	12	7	33.77	20.3
4	76.9	43.5	17	15	5.16	20.6
5	76.1	35.3	9	7	90.57	26.6
6	83.8	70.2	16	7	92.85	23.6
7	92.4	67.8	14	8	97.16	24.9

```
8   82.4    53.3    12    7   97.67   21.0
9   82.9    45.2    16    13  91.38   24.4
```

对数函数转换后的前 10 行数据为：

```
        生育能力    农业男性百分比    军队最高教育水平百分比    小学以上教育百分比    天主教百分比    婴儿死亡率
0   1.904174    1.230449    1.176091    1.079181    0.998259    1.346353
1   1.919601    1.654177    0.778151    0.954243    1.928601    1.346353
2   1.966142    1.598791    0.698970    0.698970    1.970347    1.305351
3   1.933487    1.562293    1.079181    0.845098    1.528531    1.307496
4   1.885926    1.638489    1.230449    1.176091    0.712650    1.313867
5   1.881385    1.547775    0.954243    0.845098    1.956984    1.424882
6   1.923244    1.846337    1.204120    0.845098    1.967782    1.372912
7   1.965672    1.831230    1.146128    0.903090    1.987488    1.396199
8   1.915927    1.726727    1.079181    0.845098    1.989761    1.322219
9   1.918555    1.655138    1.204120    1.113943    1.960851    1.387390
```

In[2]:

```
# 反正切函数转换
import math
def TanNorm(data):
    T_norm = pd.DataFrame(np.zeros([len(data),len(data.columns)]))
    for i in range(len(data)):
        for j in range(len(data.columns)):
            T_norm.ix[i,j] = math.atan(data.ix[i,j])*2/np.pi
    return T_norm
data_t_norm = TanNorm(data)
print('反正切函数转换前的前 10 行数据为：\n',data.ix[0:9,:],'\n',
      '反正切函数转换后的前 10 行数据为：\n',data_t_norm.ix[0:9,:])
```

Out[3]:

反正切函数转换前的前 10 行数据为：

```
    生育能力    农业男性百分比    军队最高教育水平百分比    小学以上教育百分比    天主教百分比    婴儿死亡率
0   80.2    17.0    15    12    9.96    22.2
1   83.1    45.1    6     9     84.84   22.2
2   92.5    39.7    5     5     93.40   20.2
3   85.8    36.5    12    7     33.77   20.3
4   76.9    43.5    17    15    5.16    20.6
5   76.1    35.3    9     7     90.57   26.6
6   83.8    70.2    16    7     92.85   23.6
7   92.4    67.8    14    8     97.16   24.9
8   82.4    53.3    12    7     97.67   21.0
9   82.9    45.2    16    13    91.38   24.4
```

反正切函数转换后的前 10 行数据为：

```
        0       1       2       3       4       5
0   0.992063    0.962595    0.957621    0.947071    0.936296    0.971343
1   0.992339    0.985887    0.894863    0.929553    0.992497    0.971343
2   0.993118    0.983968    0.874334    0.874334    0.993184    0.968510
3   0.992581    0.982563    0.947071    0.909666    0.981154    0.968665
4   0.991722    0.985368    0.962595    0.957621    0.878135    0.969120
```

5	0.991635	0.981970	0.929553	0.909666	0.992971	0.976078
6	0.992403	0.990932	0.960263	0.909666	0.993144	0.973041
7	0.993110	0.990611	0.954604	0.920833	0.993448	0.974447
8	0.992274	0.988057	0.947071	0.909666	0.993482	0.969708
9	0.992321	0.985918	0.960263	0.951125	0.993034	0.973924

注：由于数据较多，此处部分结果已省略。

根据代码 3-2 所示的结果可以看出，数据经过反正切函数转换后取值范围落到(−1,1)之间，由于原始数据均大于等于 0，所以数据经过反正切函数转换后取值范围落到(0,1)之间，且两种方法均未改变数据整体分布情况。

3.1.2 独热编码

在机器学习中经常会遇到类型数据，例如，性别分为男、女，手机运营商分为移动、联通和电信。在这种情况下，通常会选择将其转化为数值代入模型，如 0、1 和−1、0、1，这时往往默认为连续型数值进行处理，然而这样会影响模型的效果。

独热编码即 One-Hot 编码，又称为一位有效编码，是处理类型数据较好的方法，主要是使用 *N* 位状态寄存器来对 *N* 个状态进行编码，每个状态都有它独立的寄存器位，并且在任意时刻都只有一个编码位有效，对于每一个特征，如果它有 *m* 个可能值，那么经过独热编码后，就变成了 *m* 个二元特征，并且这些特征之间是互斥的，每一次都只有一个被激活，这时原来的数据经过独热编码后会变成稀疏矩阵。对于性别特征（男或女），利用独热编码后可以表示为 10 和 01，如图 3-1 所示。

	性别
1	男
2	男
3	女
4	男
5	女

独热编码

	性别男	性别女
1	1	0
2	1	0
3	0	1
4	1	0
5	0	1

图 3-1　使用独热编码处理性别特征

独热编码有以下优点。

（1）将离散型特征的取值扩展到了欧式空间，离散型特征的某个取值对应欧式空间的某个点。

（2）对离散型特征使用独热编码后，会让特征之间的距离计算更为合理。

例 3-3　利用 Python 进行独热编码，如代码 3-3 所示。

代码 3-3　独热编码

In[1]:
```
# 独热编码
from sklearn import preprocessing
enc = preprocessing.OneHotEncoder( )
enc.fit([[0, 0, 3], [1, 2, 0], [0, 1, 1], [1, 0, 2]])
print('[0,0,0]独热编码结果为：\n', enc.transform([[0, 0, 0]]).toarray( ), '\n',
```

```
        '[0,1,2]独热编码结果为：\n', enc.transform([[0, 1, 2]]).toarray( ), '\n',
        '[1,2,3]独热编码结果为：\n', enc.transform([[1, 2, 3]]).toarray( ))
```

```
Out[1]: [0,0,0]独热编码结果为：
        [[1. 0. 1. 0. 0. 1. 0. 0. 0.]]
        [0,1,2]独热编码结果为：
        [[1. 0. 0. 1. 0. 0. 0. 1. 0.]]
        [1,2,3]独热编码结果为：
        [[0. 1. 0. 0. 1. 0. 0. 0. 1.]]
```

在代码 3-3 中，OneHotEncoder 类使用的训练数据集为[[0,0,3],[1,2,0],[0,1,1],[1,0,2]]，每一列代表一个属性，独热编码时对每一个属性单独编码，如第一列中只有 0 和 1 两个数值，此时 0 编码为 10，1 编码为 01，而且不论数值排列顺序如何，独热编码时均按照从小到大的顺序排列。在代码 3-3 的结果中，[1,2,3] 独热编码后的前两位(0,1)对 1 编码，(0,0,1)对 2 编码，(0,0,0,1)对 3 编码，如图 3-2 所示。

	x	y	z
1	0	0	0
2	0	1	2
3	1	2	3

独热编码后

	0	1	0	1	2	0	1	2	3
1	1	0	1	0	0	1	0	0	0
2	1	0	0	1	0	0	0	1	0
3	0	1	0	0	1	0	0	0	1

图 3-2　独热编码结果

3.1.3　离散化

离散化是指将连续型特征（数值型）转换成离散型特征（类别型）的过程，需要在数据的取值范围内设定若干离散的划分点，将取值范围划分为一系列区间，最后用不同的符号或标签代表每个子区间。例如，将年龄离散化为年龄段，如图 3-3 所示。

序号	年龄
1	18
2	23
3	35
4	54
5	42
6	21
7	60
8	63
9	41
10	38

离散化

序号	年龄段
1	(17,27]
2	(17,27]
3	(27,36]
4	(45,54]
5	(36,45]
6	(17,27]
7	(54,63]
8	(54,63]
9	(36,45]
10	(36,45]

图 3-3　将年龄离散化为年龄段

部分只能接收离散型数据的算法，需要将数据离散化后才能正常运行，如 ID3、Apriori 等。而使用离散化与独热编码搭配的方法，还能够降低数据的复杂度，将其变得稀疏，增加算法运行速度。常用的离散化方法主要有 3 种：等宽法、等频法和基于聚类分析的方法。

1. 等宽法

等宽法是将数据的值域分成具有相同宽度区间的离散化方法，区间的个数由数据本身的特点决定或者用户指定。pandas 提供了 cut 函数，可以进行连续型数据的等宽离散化，其基础语法格式如下。

```
pandas.cut(x, bins, right=True, labels=None, retbins=False, precision=3, include_lowest=False)
```

cut 函数常用参数及其说明如表 3-2 所示。

表 3-2　cut 函数常用参数及其说明

参数名称	说明
x	接收数组或 Series，表示需要进行离散化处理的数据，无默认值
bins	接收 int、list、array、tuple，若为 int，则代表离散化后的类别数目；若为序列类型的数据，则表示进行切分的区间，每两个数间隔为一个区间，无默认值
right	接收 boolean，表示右侧是否为闭区间，默认为 True
labels	接收 list 和 array，表示离散化后各个类别的名称，默认为空
retbins	接收 boolean，表示是否返回区间标签，默认为 False
precision	接收 int，表示显示的标签的精度，默认为 3

例 3-4　对表 3-1 介绍的测量数据集中的婴儿死亡率使用 cut 函数进行等宽离散化处理，如代码 3-4 所示。

代码 3-4　等宽离散化处理

In[1]:

```
import pandas as pd
# 导入数据
data = pd.read_csv('../data/swiss.csv',encoding = 'GBK')
data = data.iloc[:,1:7]
#等宽离散化
sepal = pd.cut(data.ix[:,5],3)
print('婴儿死亡率离散化后的 3 条记录分布为：\n',sepal.value_counts( ))
```

Out[1]:

```
婴儿死亡率离散化后的 3 条记录分布为：
(16.067, 21.333]    32
(21.333, 26.6]      12
(10.784, 16.067]     3
Name: 婴儿死亡率, dtype: int64
```

根据代码 3-4 的结果可以看出，使用等宽离散化对数据分布具有较高要求，若数据分布不均匀，则各个类的数目也会变得非常不均匀，有些区间包含许多数据，而另外一些区间的数据极少，这会严重损坏所建立的模型。

2. 等频法

等频法是将相同数量的记录放在每个区间的离散化方法，能够保证每个区间的数量基本一致。cut 函数虽然不能够直接实现等频离散化，但是可以通过定义将相同数量的记录放进每个区间。

例 3-5 对表 3-1 介绍的测量数据集中的婴儿死亡率使用等频离散化，如代码 3-5 所示。

代码 3-5 等频离散化

In[1]:

```
import pandas as pd
import numpy as np
# 导入数据
data = pd.read_csv('../data/swiss.csv',encoding = 'GBK')
data = data.iloc[:,1:7]
#等频离散化
def SameRateCut(data,k):
    w=data.quantile(np.arange(0,1+1.0/k,1.0/k))
    data=pd.cut(data,w)
    return data
result=SameRateCut(data.ix[:,5],3).value_counts( )
print('婴儿死亡率等频离散化后各个类别数目分布状况为：','\n',result)
```

Out[1]:

```
婴儿死亡率等频离散化后各个类别数目分布状况为：
(20.8, 26.6]        16
(18.967, 20.8]      15
(10.8, 18.967]      15
Name: 婴儿死亡率, dtype: int64
```

根据代码 3-5 所示结果可以看出，相比较于等宽离散化，等频离散化避免了类分布不均匀的问题，但同时却也有可能将数值非常接近的两个值分到不同的区间以满足每个区间数据个数相同。

3. 基于聚类分析的方法

基于聚类分析的方法是将连续型数据用聚类算法（如 K-Means 算法等）进行聚类，然后通过聚类得到的簇对数据进行离散化的方法，合并到一个簇的连续型数据为一个区间。基于聚类分析的方法需要用户指定簇的个数，用来决定产生的区间数。

例 3-6 对表 3-1 介绍的测量数据集中的婴儿死亡率进行离散化，如代码 3-6 所示。

代码 3-6 基于聚类分析的方法的数据离散化

In[1]:

```
import pandas as pd
import numpy as np
# 导入数据
data = pd.read_csv('../data/swiss.csv',encoding = 'GBK')
data = data.iloc[:,1:7]
# 基于聚类分析的方法
```

```
def KmeanCut(data,k):
    from sklearn.cluster import KMeans   # 引入 K-Means
    kmodel=KMeans(n_clusters=k,n_jobs=3)   # 建立模型，n_jobs 是并行数
    kmodel.fit(data.reshape((len(data), 1)))   # 训练模型
    c=pd.DataFrame(kmodel.cluster_centers_).sort_values(0)   # 输出聚类中心并排序
    w=c.rolling(2).mean()   # 相邻两项求中点，作为边界点
    w=pd.DataFrame([0]+list(w[0])+[data.max( )])   # 把首末边界点加上
    w.fillna(value=c.min(),inplace=True)
    w = list(w.ix[:,0])
    data=pd.cut(data,w)
    return data
result=KmeanCut(np.array(data.ix[:,5]),3).value_counts()
print('婴儿死亡率聚类离散化后各个类别数目分布状况为：','\n',result)
```

Out[1]:

```
婴儿死亡率聚类离散化后各个类别数目分布状况为：
(0.0, 15.286]          2
(15.286, 17.42]        5
(17.42, 21.56]        28
(21.56, 26.6]         12
dtype: int64
```

基于 K-Means 算法进行聚类分析的离散化方法可以很好地根据现有数据的分布状况进行聚类，但是由于 K-Means 算法本身的缺陷，用该方法进行离散化时依旧需要指定离散化后类别的数目。此时需要配合聚类算法评价方法，找出最优的聚类簇数目。

3.2 特征选择

特征选择也称为特征子集选择，是从原始特征中选择出一些最有效特征以降低数据集维度的过程，是提高机器学习算法性能的一个重要手段。特征选择能够剔除不相关或者冗余的特征，从而达到减少特征个数、提高模型精确度、减少运行时间的目的。

3.2.1 子集搜索与评价

子集搜索法在原始特征中选择出最优的特征子集，避免特征过多时遇到指数爆炸问题。该方法在选择特征时采取从候选特征子集中不断迭代生成更优候选子集的方法，使得时间复杂度大大减小。该方法主要涉及如何生成候选子集和如何评价候选子集的好坏两个关键环节。

生成候选子集可以使用贪心算法，主要分为以下 3 种策略。

前向搜索：初始将每个特征视为一个候选子集，然后从当前所有候选子集中选择出最佳的特征子集；接着在上一轮中选出的特征子集中添加一个新的特征，同样选出最佳特征子集，直至选不出比上一轮更好的特征子集。

后向搜索：初始将所有特征视为一个候选特征子集，接着尝试剔除上一轮特征子集中一

个特征并选出当前最优特征子集，直至选不出比上一轮更好的特征子集。

双向搜索：将前向搜索和后向搜索结合起来，即在每一轮中都有添加操作和剔除操作。

在选择候选子集时，可以利用信息增益对特征子集的好坏进行评价，信息增益越大越有助于分类。

3.2.2 过滤式选择

过滤式选择先对数据集进行特征选择，然后对学习器进行训练，特征的选择与后续学习器无关。Relief 方法是一种著名的过滤式特征选择方法，该方法设计了一个“相关统计量”来度量特征的重要性，该统计量是一个向量，其每个分量分别对应一个初始特征，其重要性取决于相关统计量分量之和。

Relief 方法的关键是如何确定衡量特征重要性的统计量，给定训练集 $\{(\boldsymbol{x}_1,y_1),(\boldsymbol{x}_2,y_2),\cdots,(\boldsymbol{x}_m,y_m)\}$，对于每一个样本 $\boldsymbol{x}_i$，Relief 方法先在 $\boldsymbol{x}_i$ 同类样本中寻找其最近邻 $\boldsymbol{x}_{i,nh}$，称为“猜中近邻”，再从 $\boldsymbol{x}_i$ 的异类样本中寻找其最近邻 $\boldsymbol{x}_{i,nm}$，称为“猜错近邻”，相关统计量对应于属性 j 的分量如式（3-8）所示。

$$\delta^j=\sum_i -\mathrm{diff}\left(x_i^j,x_{i,\mathrm{nh}}^j\right)^2+\mathrm{diff}\left(x_i^j,x_{i,\mathrm{nm}}^j\right)^2 \tag{3-8}$$

在式（3-8）中，x_a^j 表示样本 $\boldsymbol{x}_a$ 在属性 j 上的取值，$\mathrm{diff}\left(x_a^j,x_b^j\right)$ 取决于属性 j 的类型，若属性 j 为离散型，则 $x_a^j=x_b^j$ 时 $\mathrm{diff}\left(x_a^j,x_b^j\right)$ 为 0，否则为 1；若 j 为连续型，则 $\mathrm{diff}\left(x_a^j,x_b^j\right)=\left|x_a^j-x_b^j\right|$，需要注意的是 x_a^j 和 x_b^j 已规范化到[0,1]。

除 Relief 方法以外，方差选择法、相关系数法、卡方检验和互信息法等也是较为常用的过滤式选择方法。

方差选择法：该方法利用方差进行特征选择，特征对应的方差越小，意味着该特征的识别能力越差。在极端情况下，特征对应的方差为 0，此时意味着该特征在所有样本上都是一个值。

相关系数法：该方法利用相关系数进行特征选择，计算各特征与目标特征的相关系数和相关系数的 P 值，然后选择出 K 个最好的特征。

卡方检验：该方法利用卡方检验值作为特征评分标准，卡方检验值越大，相关性越强。卡方检验值是评价定性自变量对定性因变量相关性的统计量。

互信息法：该方法使用互信息作为特征评分标准，互信息指两个事件集合之间的相关性，互信息作为特征词和类别之间的测度，若特征词属于该类，则它们的互信息最大。

3.2.3 包裹式选择

与过滤式选择不同，包裹式选择在选择特征的同时，将后续的学习器作为特征选择的评价准则。包裹式选择根据目标函数，每次选择若干特征或排除若干特征。这一方法的核心思想是基于某一种模型，并给定模型评价方法，针对特征空间中的不同特征子集，计算子集的预测效果，预测效果最好的就是最优特征子集。包裹式选择可以视为为某种学习器量身定做的特征选择方法，由于在每一轮迭代中都需要训练学习器，因此在获得较好性能的同时也产

生了较大的开销。

LVW 算法和 RFE 算法是较为经典的两种包裹式选择方法，两种方法简单介绍如下。

LVW 算法基于拉斯维加斯方法的框架，假设数据集为 D，特征集为 A，则 LVW 算法每次从特征集 A 中随机产生一个特征子集 A1，然后使用交叉验证的方法估计学习器在特征子集 A1 上的误差，若该误差小于之前获得的最小误差，或者与之前的误差相等但特征子集 A1 中包含的特征数更少，则将特征子集 A1 保留，LVW 算法过程如图 3-4 所示。

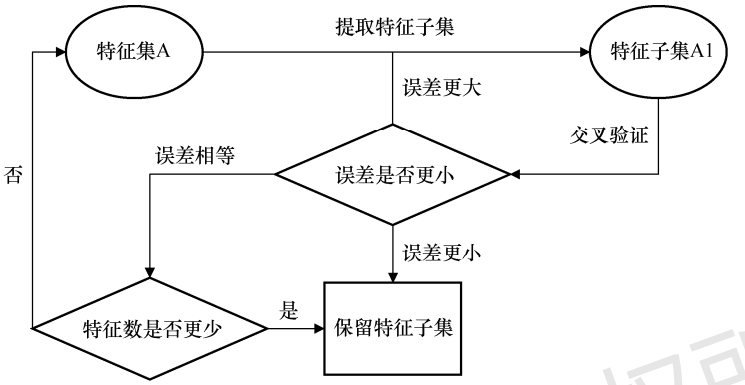

图 3-4　LVW 算法过程

RFE 算法即递归特征消除法，RFE 算法首先要设定一个特征保留个数，然后选择一个基模型对特征集合进行多轮训练，每轮训练后移除若干权值系数的特征，再基于新的特征子集合进行训练，直至特征个数达到预设值，RFE 算法过程如图 3-5 所示。

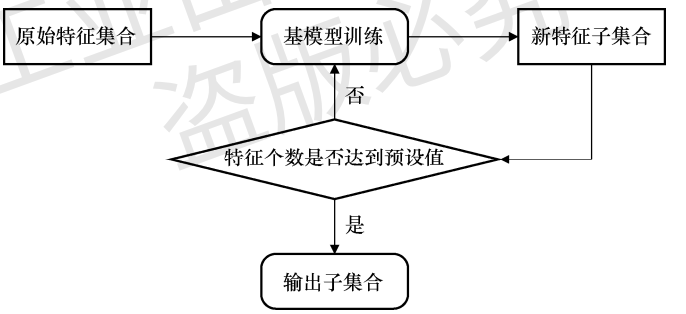

图 3-5　RFE 算法过程

3.2.4　嵌入式选择与 L_1 范数正则化

与包裹式选择使用学习器作为特征选择的评价准则不同，嵌入式选择将特征选择的过程与学习器的训练过程融为一体。给定数据集 $D=\{(\boldsymbol{x}_1,y_1),(\boldsymbol{x}_2,y_2),\cdots,(\boldsymbol{x}_m,y_m)\}$，其中 $\boldsymbol{x}\in\mathbb{R}^d, y\in\mathbb{R}$。考虑简单线性回归模型，以平方误差为损失函数，优化目标函数如式（3-9）所示。

$$\min_{\boldsymbol{w}}\sum_{i=1}^{m}(y_i-\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_i)^2 \tag{3-9}$$

当原始特征数量过多而样本数目相对较少时，式（3-9）容易陷入过拟合，在式（3-9）中引入正则化项可以缓解这一问题，如 L_1 范数正则化和 L_2 范数正则化。L_1 范数正则化和 L_2 范

数正则化可以视为损失函数的惩罚项，用于对损失函数中的某些参数做一些限制。

若使用 L_1 范数正则化，则优化目标函数可以转化如式（3-10）所示。

$$\min_{\boldsymbol{w}} \sum_{i=1}^{m} (y_i - \boldsymbol{w}^{\mathrm{T}} \boldsymbol{x}_i)^2 + \lambda \|\boldsymbol{w}\|_1 \tag{3-10}$$

在式（3-10）中，$\lambda \|\boldsymbol{w}\|_1$ 为 L_1 范数正则化项，表示权值向量 $\boldsymbol{w}$ 中各个元素的绝对值之和。正则化参数 $\lambda > 0$，需要用户指定，此时优化目标函数称为 LASSO。

若使用 L_2 范数正则化，则优化目标函数可以转化如式（3-11）所示。

$$\min_{\boldsymbol{w}} \sum_{i=1}^{m} (y_i - \boldsymbol{w}^{\mathrm{T}} \boldsymbol{x}_i)^2 + \lambda \|\boldsymbol{w}\|_2^2 \tag{3-11}$$

在式（3-11）中，$\lambda \|\boldsymbol{w}\|_2^2$ 为 L_2 范数正则化项，表示权值向量 $\boldsymbol{w}$ 中各个元素的平方和。正则化参数 $\lambda > 0$，需要用户指定，此时优化目标函数称为“岭回归”。

L_1 范数正则化和 L_2 范数正则化都有助于降低过拟合风险，而且前者比后者更容易获得“稀疏解”，即前者求得的 w 会有更少的非零分量。

3.2.5 稀疏表示与字典学习

“稀疏性”可以理解为在数据集中存在很多 0 元素，这些 0 元素并不是以整行或整列的形式存在的。当数据集具有这样的稀疏表达形式时，对学习任务会有很多好处，例如，线性支持向量机之所以能在文本数据上有很好的性能，恰恰是因为文本数据在使用字频表示后具有高度的稀疏性，使大多数问题变得线性可分，同时稀疏样本不会造成存储上的巨大负担。

特征选择所考虑的问题是特征具有“稀疏性”，即数据中的许多列与当前的学习任务无关，通过特征选择去除这些列，此时学习器在训练的过程中仅需要在较小的数据集上进行，学习任务的难度可能会有所降低，涉及的计算和存储开销会减少，训练模型的可解释性也会提高。

以文档分类任务为例，每一个文档视为一个样本，每个字或词视为一个特征，字或词在文档中出现的频率视为特征取值。如果文档中文字全部是汉字，那么考虑《康熙字典》和《现代汉语常用字表》得到的数据集是不同的，然而，给定文档以后，有很多的字或词是不会出现在该文档中的，于是数据集中每一行都会出现大量的 0 元素，而且对于不同的文档，0 元素出现的列往往差异很大，如图 3-6 所示。

文档
机器学习是人工智能的一个重要分支 ……

阿	啊	…	机器	学习	是	人工	智	能	的	一个	重要	分支	…
0	0	…	1	1	1	1	1	1	1	1	1	1	…

阿	啊	…	机器	学习	是	人工	智能	的	一个	重要	分支	…
0	0	…	1	1	1	1	1	1	1	1	1	…

图 3-6　文档分类稀疏矩阵

在一般的学习任务中，并没有《现代汉语常用字表》可以利用，需要学习得到一个类似作用的“字典”。为普通稠密表达的样本找到合适的“字典”，可以将样本转化为合适的稀疏表达形式，从而使学习任务得以简化，模型复杂度得以降低，通常将这一过程称为“字典学习”，也称为“稀疏编码”。“字典学习”侧重于学得“字典”的过程，“稀疏编码”更侧重于对样本进行稀疏表达的过程，由于两者通常是在同一优化求解过程中完成的，因此可以将两者统称为“字典学习”。

给定数据集$\{\boldsymbol{x}_1,\boldsymbol{x}_2,\cdots,\boldsymbol{x}_m\}$，“字典学习”的最简单形式如式（3-12）所示。

$$\min_{\boldsymbol{B},\boldsymbol{\alpha}_i}\sum_{i=1}^{m}\left\|\boldsymbol{x}_i-\boldsymbol{B}\boldsymbol{\alpha}_i\right\|_2^2+\lambda\sum_{i=1}^{m}\left\|\boldsymbol{\alpha}_i\right\|_1 \tag{3-12}$$

在式（3-12）中，$\boldsymbol{B}\in\mathbb{R}^{d\times k}$为字典矩阵，$k$为字典的词汇量，通常由用户指定，$\boldsymbol{\alpha}_i\in\mathbb{R}^k$是样本$\boldsymbol{x}_i\in\mathbb{R}^d$的稀疏表示。$\min_{\boldsymbol{B},\boldsymbol{\alpha}_i}\sum_{i=1}^{m}\left\|\boldsymbol{x}_i-\boldsymbol{B}\boldsymbol{\alpha}_i\right\|_2^2$是希望有$\boldsymbol{\alpha}_i$能够很好地重构$x_i$，$\lambda\sum_{i=1}^{m}\left\|\boldsymbol{\alpha}_i\right\|_1$则是希望$\boldsymbol{\alpha}_i$尽量稀疏。

例 3-7　利用 Python 对表 3-1 介绍的测量数据集进行特征选择，如代码 3-7 所示。

代码 3-7　特征选择

In[1]:

```
#特征选择
import pandas as pd
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
# 导入数据
data = pd.read_csv('../data/swiss.csv',encoding = 'GBK')
X = data.iloc[:,1:7]
y = data.iloc[:,7]
# 过滤式选择
print('特征选择前数据集形状为：',X.shape)
X_new=SelectKBest(chi2,k=5).fit_transform(X,y)
print('特征选择后数据集形状为：',X_new.shape)
```

Out[1]:

```
特征选择前数据集形状为：(47, 6)
特征选择后数据集形状为：(47, 5)
```

In[2]:

```
# 包裹式选择
from sklearn.feature_selection import RFE
from sklearn.linear_model import LinearRegression
names=X.columns
lr=LinearRegression( )
rfe=RFE(lr,n_features_to_select=1)#选择剔除 1 个
rfe.fit(X,y)
print('剔除排名为：\n',sorted(zip(map(lambda x:round(x,5), rfe.ranking_),names)))
```

Out[2]:

```
剔除排名为：
 [(1, '生育能力'), (2, '小学以上教育百分比'), (3, '天主教百分比'), (4, '农业男性百分比'), (5, '婴儿死亡率'), (6, '军队最高教育水平百分比')]
```

In[3]:

```
# 嵌入式
from sklearn.svm import    LinearSVC
from sklearn.feature_selection import SelectFromModel
print('L1 范数正则化前数据集形状为：',X.shape)
lsvc=LinearSVC(C=0.01,penalty='l1',dual=False).fit(X,y)
model=SelectFromModel(lsvc,prefit=True)
X_new=model.transform(X)
print('L1 范数正则化后数据集形状为：',X_new.shape)
```

Out[3]:

```
L1 范数正则化前数据集形状为：(47, 6)
L1 范数正则化后数据集形状为：(47, 5)
```

根据代码 3-7 的运行结果可以看出，嵌入式特征选择保留了 5 个特征，而包裹式选择（RFE 方法）在特征选择时需要人为选择剔除特征个数，本次选择剔除一个特征，过滤式特征需要选择保留特征个数，本次选择保留 5 个特征，根据代码的显示结果，删除特征婴儿死亡率最佳，由此可以发现不同的方法特征选择的结果是不同的。

小　　结

本章主要介绍特征工程中特征变换和特征选择相关的内容。其中，特征变换介绍了其作用和主要方法，主要方法有简单函数变换、标准化、归一化、独热编码和离散化；特征选择介绍了其作用和主要方法，主要方法包括子集搜索与评价、过滤式选择、包裹式选择、嵌入式选择与 L_1 范数正则化及稀疏表示与字典学习。

课 后 习 题

1. 选择题

（1）下列说法正确的是（　　）。

A. 特征变换和特征选择都可以达到降维的目的

B. 数据归一化等同于数据标准化

C. Python 中 cut 函数可以对连续数据进行等宽离散化和等频离散化

D. 子集搜索法只能利用信息增益进行评价

（2）下列关于标准化的说法正确的是（　　）。

A. 离差标准化和标准差标准化之后数据取值范围相同

B. 标准差标准化后数据取值全为正值

C. 离差标准化前后数据整体分布情况未发生变化

D. 小数定标标准化适用范围较窄

（3）特征选择时考虑用学习器作为评价准则的是（　　）。

A. 过滤式选择

B. 包裹式选择
C. 嵌入式选择
D. 稀疏表示与字典学习
（4）关于嵌入式选择与 L_1 范数正则化描述错误的是（　　）。
A. 嵌入式选择与学习器的训练过程无关
B. L_1 范数正则化后的目标函数称为 LASSO
C. L_2 范数正则化后的目标函数称为“岭回归”
D. L_1 范数正则化和 L_2 范数正则化都有助于降低过拟合风险
（5）特征选择可以利用下列哪个方法（　　）。
A. 标准化
B. 独热编码
C. 稀疏表示与字典学习
D. 函数转换
（6）关于离散化说法正确的是（　　）。
A. 等宽离散化和等频离散化可以用同一个函数实现
B. 基于聚类的方法只能用 K-Means 算法实现
C. 基于聚类的方法针对于一维数据
D. Python 中 cut 函数可以实现等频法离散化
（7）下列关于类型数据说法正确的是（　　）。
A. 将类型数据默认为连续数据进行不会影响模型效果
B. 独热编码是唯一有效的处理类型数据的方法
C. 独热编码不能处理非连续性数值特征
D. 独热编码后的数据变成稀疏矩阵
（8）下列关于归一化说法错误的是（　　）。
A. 归一化的目的之一是将原始数据变成(0,1)之间的数
B. 归一化可以利用线性函数、反正切函数转换等方法实现
C. 任何数据经反正切函数转换均可变成(0,1)之间的数
D. 数据归一化可以提升模型的收敛速度和精确度
（9）特征工程一定不会用到哪个方法（　　）。
A. 标准化
B. 独热编码
C. 过滤式选择
D. 缺失值处理
（10）下面说法不正确的是（　　）。
A. 数据稀疏表达时 0 越多越好
B. 常用的特征变换方法有标准化、归一化、函数变换和特征离散化等
C. 特征选择能够剔除不相关或者冗余的特征
D. 包裹式选择在选择特征的同时，将后续的学习器考虑进来作为特征选择的评价准则

2. 填空题

（1）特征工程主要任务是_____和特征选择。

（2）特征变换可以用到_____、_____、_____、独热编码和_____。

（3）嵌入式选择中减少过拟合风险可以利用_____、_____。

（4）特征选择可以利用_____、_____、_____、_____或_____方法。

（5）消除数据量纲影响可以利用_____和_____。