華中師範大學

自然语言处理
第五章 特征工程

主讲教师：曾江峰

华中师范大学 信息管理学院

jfzeng@ccnu.edu.cn

特征工程

- 特征是指区分事物的属性。特征工程是指通过规范化、标准化、鲁棒化和正则化等方法将数据转换成符合算法要求的数据。
- 本章重点介绍特征预处理、One-Hot编码、CountVectorizer、TF-IDF和LDA 。

特征工程

1. 特征预处理
2. One-Hot编码
3. CountVectorizer
4. TF-IDF
5. LDA

1. 特征预处理

- 当多个特征大小相差较大，或者某特征的方差相比其他特征数个数量级，容易影响或支配目标结果。特征预处理就是通过转换函数将这些特征数据转换成适合算法模型的过程。
- 这一过程对特征进行了集成、转换、规约等处理，主要包括特征的归一化、标准化、鲁棒化和正则化等方法。
- Sklearn的preprocessing模块提供相关的特征预处理函数。

1. 特征预处理

方法含义	方法名
归一化	preprocessing.MinMaxScaler
标准化	preprocessing.StandardScaler
鲁棒化	preprocessinf.RobustScaler
正则化	preprocessing.Normalize

表1-1 preprocessing模块常用方法

1. 特征预处理

1.1 归一化

不同特征往往具有不同的量纲，其差别较大的数值影响数据分析结果。为了让不同特征具有同等的重要性，往往需要进行归一化处理，通过对原始数据进行线性变换，将数据归一到[0,1]，即变换到固定的最小最大值的区间。归一化的数学计算公式如下：

$$X' = \frac{x - min}{max - min}$$

1. 特征预处理

1.1 归一化

Sklearn提供MinMaxScaler()方法进行归一化

```
MinMaxScaler(feature_range=(0,1))
```

参数	说明
feature_range=(0,1)	范围设置为0到1

表1-2 MinMaxScaler()参数说明

1. 特征预处理

1.1 归一化

【例1】归一化：现有三个样本，每个样本有四个特征，如表1-3所展示

	特征1	特征2	特征3	特征4
样本一	90	2	10	40
样本二	60	4	15	45
样本三	75	3	13	46

表1-3 样本特征

```python
# 例1：归一化
from sklearn.preprocessing import MinMaxScaler
def Normalization():    # 实例化一个转换器类
    Normalization = MinMaxScaler(feature_range = (0,1))    # 范围设置为0到1
    data = [[90,2,10,40],[60,4,15,45],[75,3,13,46]]
    print(data)
    # 调用fit_transform
    data_Normal = Normalization.fit_transform(data)
    print(data_Normal)
    return None
```

```python
if __name__ == '__main__':
    Normalization()
```

```
[[90, 2, 10, 40], [60, 4, 15, 45], [75, 3, 13, 46]]
[[1.         0.         0.         0.        ]
 [0.         1.         1.         0.83333333]
 [0.5        0.5        0.6        1.        ]]
```


1. 特征预处理

1.2 标准化

当有些特征的方差过大，会导致无法正确地去学习其他特征，标准化用于解决归一化容易受到样本中极大或者极小的异常值的影响。数据标准化（standardization)将数据按比例缩放到特定区间。标准化后，所有数据都聚集在0附近，方差为1。标准化的数学计算公式如下：

$$z = \frac{x - \mu}{\sigma}$$

其中，μ是均值，σ是标准差。

1. 特征预处理

1.2 标准化

方法一：采用Numpy模块实现

【例2】采用Numpy模块实现标准化

```python
# 例2：采用Numpy模块实现标准化
import numpy as np
def z_norm(data_list):
    data_len = len(data_list)
    if data_len == 0:
        raise'数据为空'
    data_list = np.array(data_list)
    mean_v = np.mean(data_list,axis = 0)
    std_v = np.std(data_list,axis = 0)
    print('该矩阵的初值为：\n{}\n 该矩阵的标准差为：\n{}\n'.format(mean_v,std_v))
    if std_v.all() == 0:
        raise '标准差为0'
    return (data_list - mean_v) / std_v
if __name__ == '__main__':
    data_list = [[1.5,-1.,2.],[2.,0.,0.]]
    print('矩阵初值为：\n{}'.format(data_list))
    print('z-score标准化:\n',z_norm(data_list))
```

```
矩阵初值为：
[[1.5, -1.0, 2.0], [2.0, 0.0, 0.0]]
该矩阵的初值为：
[ 1.75 -0.5   1.  ]
 该矩阵的标准差为：
[0.25 0.5  1.  ]

z-score标准化:
 [[-1. -1.  1.]
 [ 1.  1. -1.]]
```


1. 特征预处理

1.2 标准化

方法二：采用Sklearn模块提供的StandardScaler()实现

StandardScaler(copy,with_mean)

参数	说明
copy	取值为True或False，False意味着用归一化的值替代原来的值
with_mean	取值为True或False，False意味着是稀疏矩阵

表1-4 StandardScaler()参数说明

1. 特征预处理

1.2 标准化

方法二：采用Sklearn模块提供的StandardScaler()实现

【例3】采用Sklearn模块实现标准化

```
import numpy as np
from sklearn.preprocessing import StandardScaler
def Standardization():
    data_list = [[1.5,-1.,2.],[2.,0.,0.]]
    print('矩阵初值为：{}'.format(data_list))
    scaler = StandardScaler()
    data_Standard = scaler.fit_transform(data_list)
    print('该矩阵的均值为：{}\n 该矩阵的标准差为：{}'.format(scaler.mean_,np.sqrt(scaler.var_)))
    print('标准差标准化的矩阵为：{}'.format(data_Standard))
    return None
if __name__ == '__main__':
    Standardization()
```

```
矩阵初值为：[[1.5, -1.0, 2.0], [2.0, 0.0, 0.0]]
该矩阵的均值为：[ 1.75 -0.5   1.  ]
 该矩阵的标准差为：[0.25 0.5  1.  ]
标准差标准化的矩阵为：[[-1. -1.  1.]
 [ 1.  1. -1.]]
```


1. 特征预处理

1.3 鲁棒化

当数据包含许多异常值，离群值较多时，使用均值和方差缩放不能取得较好效果，可以使用鲁棒性缩放（RobustScaler）进行处理。RobustScaler使用中位数和四分位数进行数据的转换，会直接将异常值剔除。

1. 特征预处理

1.3 鲁棒化

Sklearn提供sklearn.preprocessing.RobustScaler()实现鲁棒化

RobustScaler(quantile_range,with_centering,with_scaling)

参数	说明
with_centering	布尔值（默认值为True），在缩放之前将数据居中
with_scaling	布尔值（默认值为True），将数据缩放到四分位数范围
quantile_range	元组（默认值为（25.0,75.0）），用于计算分位数范围

表1-5 RobustScaler()参数说明

1. 特征预处理

1.3 鲁棒化

【例4】鲁棒化

```
# 例4: 鲁棒化
from sklearn.preprocessing import RobustScaler
X = [[1.,-2.,2.],[-2.,1.,3.],[4.,1.,-2.]]
transformer = RobustScaler().fit(X)
RobustScaler(quantile_range = (25.0,75.0),with_centering = True,with_scaling = True)
print(transformer.transform(X))
```

```
[[ 0.  -2.   0. ]
 [-1.   0.   0.4]
 [ 1.   0.  -1.6]]
```


1. 特征预处理

1.4 正则化

Preprocessing模块提供normalize()函数实现正则化

normalize(X,norm='l2')

参数	说明
X	样本数据
norm='l2'	L2范式

表1-6 normalize()参数说明

1. 特征预处理

1.4 正则化

【例5】正则化

```
# 例5: 正则化
from sklearn.preprocessing import normalize
X = [[1.,-1.,2.],[2.,0.,0.],[0.,1.,-1.]]
X_normalized = normalize(X,norm = 'l2')
print(X_normalized)
```

```
[[ 0.40824829 -0.40824829  0.81649658]
 [ 1.          0.          0.        ]
 [ 0.          0.70710678 -0.70710678]]
```


1. 特征预处理

1.5 示例

【例6】学生信息特征预处理

```
# 例6：学生信息处理
import pandas as pd
import numpy as np
from collections import Counter
from matplotlib import pyplot as plt
import seaborn as sns
plt.rcParams['font.sans-serif'] = ['SimHei']   # 中文字体设置-黑体
plt.rcParams['axes.unicode_minus'] = False   # 解决保存图像是负号"-"显示为方块的问题
sns.set(font = 'SimHei')   # 解决Seaborn中文显示问题
```

```
data = pd.read_excel('C:\\Users\\颖舟\\Desktop\\dummy.xls')
print(data)
```

```
   姓名   学历     成绩     能力   学校
0  小红   博士   90.0  100.0   同济
1  小黄   硕士   90.0   89.0   交大
2  小绿   本科   80.0   99.0   同济
3  小白   硕士   90.0   99.0   复旦
4  小紫   博士  100.0   79.0   同济
5  小城   本科   80.0   99.0   交大
6  校的  NaN    NaN    NaN  NaN
```

```
print('data head:\n',data.head())   # 序列的前n行，默认值为5
```

```
data head:
    姓名   学历     成绩     能力   学校
0  小红   博士   90.0  100.0   同济
1  小黄   硕士   90.0   89.0   交大
2  小绿   本科   80.0   99.0   同济
3  小白   硕士   90.0   99.0   复旦
4  小紫   博士  100.0   79.0   同济
```

```
print('data shape:\n',data.shape)   # 查看数据的行列大小
```

```
data shape:
 (7, 5)
```

```
print('data describe:\n',data.describe())
```

```
data describe:
                成绩          能力
count    6.000000    6.000000
mean    88.333333   94.166667
std      7.527727    8.495097
min     80.000000   79.000000
25%     82.500000   91.500000
50%     90.000000   99.000000
75%     90.000000   99.000000
max    100.000000  100.000000
```


1. 特征预处理

1.5 示例

【例6】学生信息特征预处理

```
# 列级别的判断，但凡某一列有null值或空的，则为真
data.isnull().any()
```

```
姓名     False
学历      True
成绩      True
能力      True
学校      True
dtype: bool
```

```
# 将列中为空或者null的个数统计出来，并将缺失值最多的排在前面
total = data.isnull().sum().sort_values(ascending = False)
print('total:\n',total)
```

```
total:
 学历     1
成绩     1
能力     1
学校     1
姓名     0
dtype: int64
```

```
# 输出百分比
precent = (data.isnull().sum() / data.isnull().count()).sort_values(ascending = False)
missing_data = pd.concat([total,precent],axis = 1,keys = ['Total','Precent'])
missing_data.head(20)
```

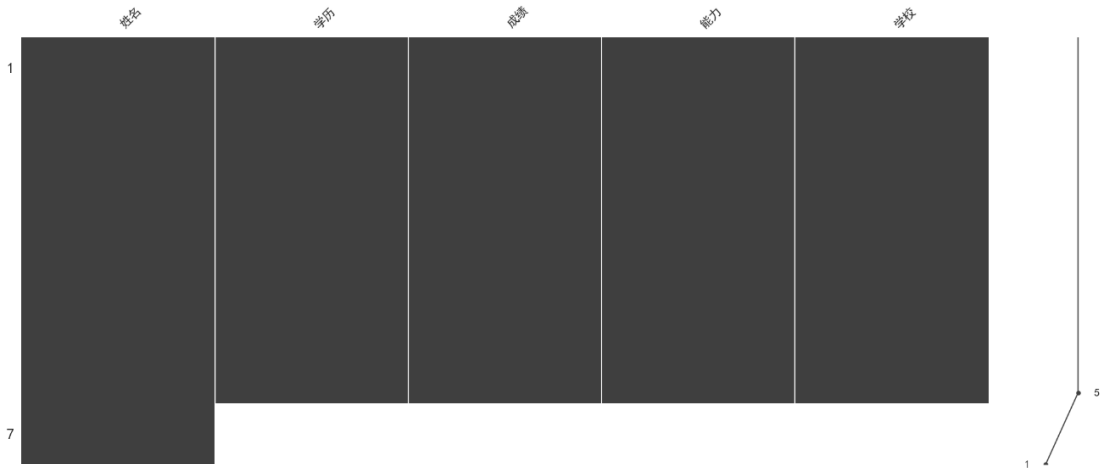
	Total	Precent
学历	1	0.142857
成绩	1	0.142857
能力	1	0.142857
学校	1	0.142857
姓名	0	0.000000

1. 特征预处理

1.5 示例

【例6】学生信息特征预处理

```
import missingno   # missingno是一个可视化缺失值的库
missingno.matrix(data)
plt.show()
```




```
# 至少有一半以上是非空的列筛选出来
data = data.dropna(thresh = data.shape[0] * 0.5,axis = 1)
print(data)
```

	姓名	学历	成绩	能力	学校
0	小红	博士	90.0	100.0	同济
1	小黄	硕士	90.0	89.0	交大
2	小绿	本科	80.0	99.0	同济
3	小白	硕士	90.0	99.0	复旦
4	小紫	博士	100.0	79.0	同济
5	小城	本科	80.0	99.0	交大
6	校的	NaN	NaN	NaN	NaN

```
# 如果某一行全部是na才删除，默认情况下是只保留没有空值的行
data.dropna(how = 'all',axis = 1)
print(data)
```

	姓名	学历	成绩	能力	学校
0	小红	博士	90.0	100.0	同济
1	小黄	硕士	90.0	89.0	交大
2	小绿	本科	80.0	99.0	同济
3	小白	硕士	90.0	99.0	复旦
4	小紫	博士	100.0	79.0	同济
5	小城	本科	80.0	99.0	交大
6	校的	NaN	NaN	NaN	NaN

1. 特征预处理

1.5 示例

【例6】学生信息特征预处理

```
# 统计重复记录数
data.duplicated().sum()
data.drop_duplicates()
data.columns
```

```
Index(['姓名', '学历', '成绩', '能力', '学校'], dtype='object')
```

```
# 第一步，将整个data的连续性字段和离散型字段进行归类
id_col = ['姓名']
cat_col = ['学历','学校']    # 离散型无序
cont_col = ['成绩','能力']    # 数值型
print(data[cat_col])    # 离散型的数据部分
print(data[cont_col])    # 连续型的数据部分
```

```
    学历   学校
0   博士   同济
1   硕士   交大
2   本科   同济
3   硕士   复旦
4   博士   同济
5   本科   交大
6  NaN  NaN
     成绩     能力
0   90.0  100.0
1   90.0   89.0
2   80.0   99.0
3   90.0   99.0
4  100.0   79.0
5   80.0   99.0
6    NaN    NaN
```


1. 特征预处理

1.5 示例

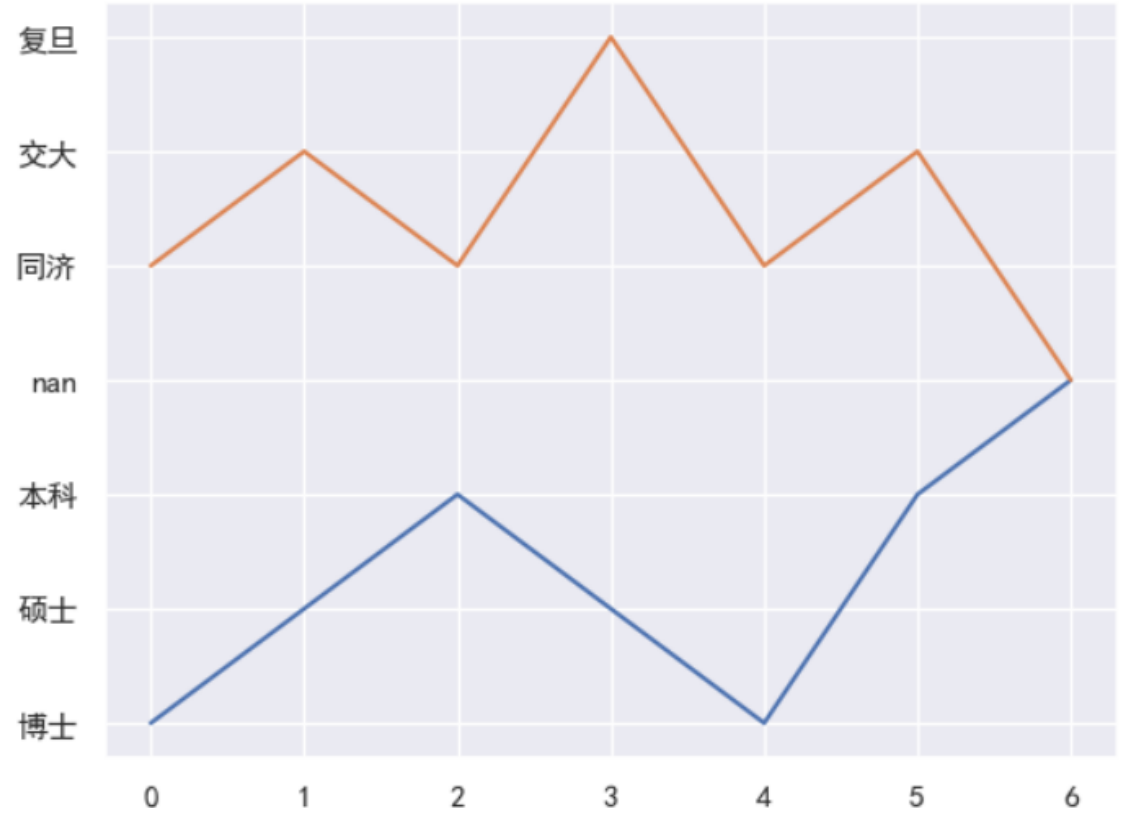
【例6】学生信息特征预处理

```
#计算出现的频次
for i in cat_col:
    print(pd.Series(data[i]).value_counts())
    plt.plot(list(data[i]))
#对于离散型数据，对其获取哑变量
dummies=pd.get_dummies(data[cat_col])
print('哑变量:\n',dummies)
```

```
硕士    2
博士    2
本科    2
Name: 学历, dtype: int64
同济    3
交大    2
复旦    1
Name: 学校, dtype: int64
```

哑变量：

	学历_博士	学历_本科	学历_硕士	学校_交大	学校_同济	学校_复旦
0	1	0	0	0	1	0
1	0	0	1	1	0	0
2	0	1	0	0	1	0
3	0	0	1	0	0	1
4	1	0	0	0	1	0
5	0	1	0	1	0	0
6	0	0	0	0	0	0



1. 特征预处理

1.5 示例

【例6】学生信息特征预处理

```
# 对于连续型数据的统计
data[cont_col].describe()
```

	成绩	能力
count	6.000000	6.000000
mean	88.333333	94.166667
std	7.527727	8.495097
min	80.000000	79.000000
25%	82.500000	91.500000
50%	90.000000	99.000000
75%	90.000000	99.000000
max	100.000000	100.000000

```
# 对于连续型数据，看偏度，将大于0.75的数值用log转换，使之符合正态分布
skewed_feats = data[cont_col].apply(lambda x: (x.dropna()).skew())    # 计算偏度
skewed_feats = skewed_feats[skewed_feats > 0.75]
skewed_feats = skewed_feats.index
data[skewed_feats] = np.log1p(data[skewed_feats])
# print(skewed_feats)
```

```
#对于连续型数据，对其进行标准化
from sklearn import preprocessing
scaled = preprocessing.scale(data[cont_col])
scaled = pd.DataFrame(scaled,columns = cont_col)
print(scaled)
```

```
       成绩        能力
0  0.242536  0.752210
1  0.242536 -0.666243
2 -1.212678  0.623260
3  0.242536  0.623260
4  1.697749 -1.955746
5 -1.212678  0.623260
6       NaN       NaN
```

《自然语言处理入门》

1. 特征预处理

1.5 示例

【例6】学生信息特征预处理

```
m = dummies.join(scaled)
data_cleaned = data[id_col].join(m)
print('标准化：\n',data_cleaned)
```

标准化：

	姓名	学历_博士	学历_本科	学历_硕士	学校_交大	学校_同济	学校_复旦	成绩	能力
0	小红	1	0	0	1	0	0.242536	0.752210	
1	小黄	0	0	1	1	0	0	0.242536	-0.666243
2	小绿	0	1	0	0	1	0	-1.212678	0.623260
3	小白	0	0	1	0	0	1	0.242536	0.623260
4	小紫	1	0	0	0	1	0	1.697749	-1.955746
5	小城	0	1	0	1	0	0	-1.212678	0.623260
6	校的	0	0	0	0	0	0	NaN	NaN

1. 特征预处理

1.5 示例

【例6】学生信息特征预处理

```
# 变量之间的相关性
print('变量之间的相关性:\n',data_cleaned.corr())
```

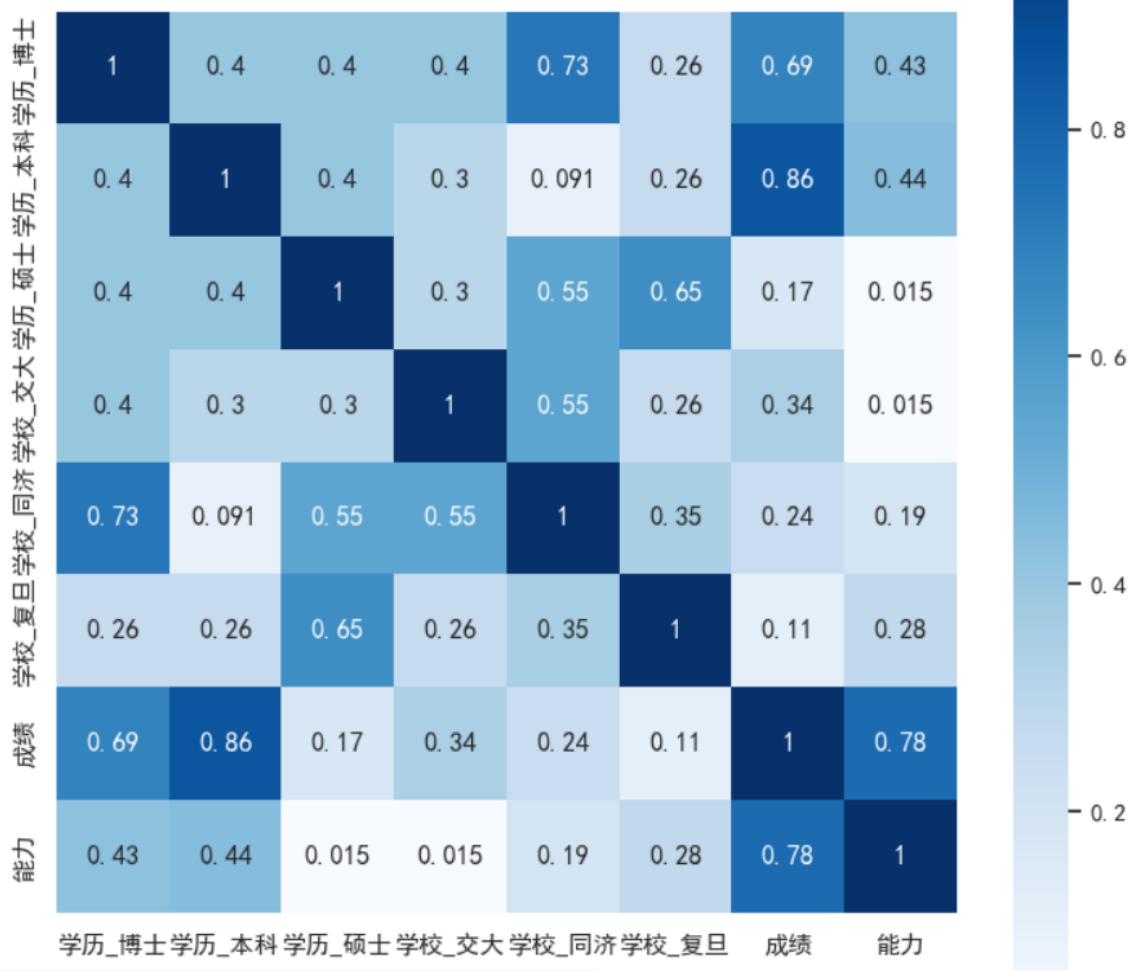
```
变量之间的相关性:
          学历_博士     学历_本科     学历_硕士     学校_交大     学校_同济     学校_复旦       成绩  \
学历_博士  1.000000 -0.400000 -0.400000 -0.400000  0.730297 -0.258199  0.685994
学历_本科 -0.400000  1.000000 -0.400000  0.300000  0.091287 -0.258199 -0.857493
学历_硕士 -0.400000 -0.400000  1.000000  0.300000 -0.547723  0.645497  0.171499
学校_交大 -0.400000  0.300000  0.300000  1.000000 -0.547723 -0.258199 -0.342997
学校_同济  0.730297  0.091287 -0.547723 -0.547723  1.000000 -0.353553  0.242536
学校_复旦 -0.258199 -0.258199  0.645497 -0.258199 -0.353553  1.000000  0.108465
成绩      0.685994 -0.857493  0.171499 -0.342997  0.242536  0.108465  1.000000
能力     -0.425514  0.440711 -0.015197 -0.015197 -0.193425  0.278730 -0.776663

              能力
学历_博士 -0.425514
学历_本科  0.440711
学历_硕士 -0.015197
学校_交大 -0.015197
学校_同济 -0.193425
学校_复旦  0.278730
成绩     -0.776663
能力      1.000000
```


1. 特征预处理

1.5 示例

【例6】学生信息特征预处理




```python
# 以下是相关性的热力图
def corr_heat(df):
    dfData = abs(df.corr())
    plt.subplots(figsize = (9,9))   # 设置画面大小
    sns.heatmap(dfData, annot = True, vmax = 1, square = True, cmap = 'Blues')
    # plt.savefig('C:\\Users\\Desktop\\BluesStateRelation.png')
    plt.show()
corr_heat(data_cleaned)
```

2. One-Hot编码

2.1 认识One-Hot编码

机器学习算法往往无法直接处理文本数据，需要把文本数据转换为数值型数据，独热编码把文本转换为数值，也称One-Hot编码，又称为一位有效编码。One-Hot编码保证了每一个取值只会使得一种状态处于“激活态”，也就是说这N种状态中只有一个状态位值为1，其他状态位都是0。

独热编码具有操作简单，容易理解的优势。但是，独热编码完全割裂了词与词之间的联系，而且在大语料集下，每个向量的长度过大，占据大量内存。

2. One-Hot编码

2.1 认识One-Hot编码

【例7】One-Hot编码

步骤1：确定编码对象["中国", "美国", "日本", "美国"]

步骤2：确定分类变量：中国、美国、日本共3种类别

步骤3：进行特征编码：中国—0，美国—1，日本—2

结果：["中国", "美国", "日本", "美国"]进行One-Hot编码为[[1,0,0],[0,1,0],[0,0,1],[0,1,0]]

2. One-Hot编码

2.2 Pandas实现

pandas的get_dummies()函数实现one-hot编码

pandas.get_dummies(data,sparse=False)

参数	说明
data	数组类型、Series、DataFrame等
sparse	稀疏矩阵

表2-1 get_dummies()参数说明

2. One-Hot编码

2.2 Pandas实现

【例8】采用Pandas实现One-Hot编码

```
# 例8：采用Pandas实现One-Hot编码
import pandas as pd
s = pd.Series(list('abcd'))
print(s)
s1 = pd.get_dummies(s,sparse = True)
print(s1)
```

```
0    a
1    b
2    c
3    d
dtype: object
   a  b  c  d
0  1  0  0  0
1  0  1  0  0
2  0  0  1  0
3  0  0  0  1
```


2. One-Hot编码

2.3 Sklearn实现

【例9】Sklearn采用OneHotEncoder编码

```
# 例9：采用Sklearn实现One-Hot编码
from sklearn.preprocessing import OneHotEncoder
enc = OneHotEncoder()
enc.fit([[0, 0, 3],
         [1, 1, 0],
         [0, 2, 1],
         [1, 0, 2]])   # fit编码
ans1 = enc.transform([[0, 1, 3]])   # 输出稀疏矩阵
ans2 = enc.transform([[0, 1, 3]]).toarray()   #输出数组格式
print('稀疏矩阵\n',ans1)
print('数组格式\n',ans2)
```

```
稀疏矩阵
   (0, 0)        1.0
  (0, 3)         1.0
  (0, 8)         1.0
数组格式
 [[1. 0. 0. 1. 0. 0. 0. 0. 1.]]
```


2. One-Hot编码

特征提取：

数据往往具有不同的数据类型，如数值型、字符型、布尔型等。但是，机器学习模型只接收数值型和布尔型，需要特征提取进行转化。特征提取又称为特征抽取，是将任意数据（字典、文本或图像）转换为可用于机器学习的数字特征。

方法	说明
feature_extraction.DictVectorizer	将特征值映射列表转换为向量
feature_extraction.FeatureHasher	特征哈希
feature_extraction.text	文本相关特征抽取
feature_extraction.image	图像相关特征抽取
feature_extraction.text.CountVectorizer	将文本转换为每个词出现次数的向量
feature_extraction.text.TfidfVectorizer	将文本转换为tfidf 值的向量

表2-2 特征提取方法说明

《自然语言处理入门》

2. One-Hot编码

2.4 DictVectorizer

当数据以“字典”数据结构进行存储时，字典特征提取将字典内容转化成计算机可以处理的数值。sklearn提供DictVectorizer()函数实现字典特征提取

```
sklearn.feature_extraction.DictVectorizer(sparse=True)
```

参数	说明
sparse=True	返回稀疏矩阵，只将矩阵中非零值按位置表示

表2-3 DictVectorizer()参数说明

2. One-Hot编码

2.4 DictVectorizer

【例10】字典特征抽取

```python
# 例10：字典特征抽取
from sklearn.feature_extraction import DictVectorizer
def dictvec1():   # 定义一个字典列表，表示多个数据样本
    data = [{'city':'上海','temperature':100},
            {'city': '北京', 'temperature': 60},
            {'city': '深圳', 'temperature': 30} ]
    # 1.转换器
    DictTransform = DictVectorizer()
    # DictTransform = DictVectorizer(sparse = True)    # 输出为稀疏矩阵
    DictTransform = DictVectorizer(sparse = False)    # 输出为二维数组
    # 2.调用fit_transform()方法，返回sparse矩阵
    data_new = DictTransform.fit_transform(data)
    print(DictTransform.get_feature_names())
    print(data_new)
    return None
if __name__ == '__main__':
    dictvec1()
```

```
['city=上海', 'city=北京', 'city=深圳', 'temperature']
[[  1.   0.   0. 100.]
 [  0.   1.   0.  60.]
 [  0.   0.   1.  30.]]
```

3行表示3个向量，即3个样本。
4列表示2个特征（city和温度）的取值。其中，city共3个取值，取值为'上海', '北京', 深圳'，采用 one –hot编码。第一行中，‘上海’为真，取值为1，‘北京’、‘深圳’为假，取值为0；第二行中，‘北京’为真，取值为1，其余为0……依次类推。

3. CountVectorizer

文本特征提取：

- CountVectorizer与TfidfVectorizer是两个特征数值计算的常见方法。
- CountVectorizer只考虑每种词汇在文本中出现的频率
- TfidfVectorizer除了考量某一词汇在当前训练文本中出现的频率之外，同时关注包含这个词汇的其它训练文本数目的倒数。相比之下，训练文本的数量越多，TfidfVectorizer更有优势

3. CountVectorizer

3.1 认识CountVectorizer

CountVectorizer适合于主题较多的语料集。它构成特征矩阵的每一行表示一个训练文本的词频统计结果，只考虑每个单词在文本中出现的频率。其思想是，将文本中每个单词视为一个特征，构成一个词汇表，不考虑单词出现的先后次序，该方法又称为词袋法（Bag of Words)。词袋模型是最早的自然语言处理模型，将文章中的词语通过计数的方式转换为数字。

3. CountVectorizer

3.2 Sklearn调用CountVectorizer

Sklearn提供CountVectorizer()方法用于文本特征提取

feature_extraction.text.CountVectorizer(stop_words)

参数	说明
stop_words	停用词表

表3-1 CountVectorizer()参数说明

3. CountVectorizer

3.2 Sklearn调用CountVectorizer

【例11】CountVectorizer举例

```
# 例11: CountVectorizer举例
from sklearn.feature_extraction.text import CountVectorizer
texts=['orange banana apple grape','banana apple apple','grape','orange apple']
#1.实例化一个转换器类
cv = CountVectorizer()
#2.调用fit_transform()
cv_fit=cv.fit_transform(texts)
print(cv.vocabulary_)
print(cv_fit.shape)
print(cv_fit)
print(cv_fit.toarray())
```

```
{'orange': 3, 'banana': 1, 'apple': 0, 'grape': 2}
(4, 4)
  (0, 3)        1
  (0, 1)        1
  (0, 0)        1
  (0, 2)        1
  (1, 1)        1
  (1, 0)        2
  (2, 2)        1
  (3, 3)        1
  (3, 0)        1
[[1 1 1 1]
 [2 1 0 0]
 [0 0 1 0]
 [1 0 0 1]]
```

#"banana apple apple"在（apple，banana，grape，orange）中，"banana"出现1次，"apple"出现2次，其余两种出现0次，因此得到的二维数组第二行为[2,1,0,0]

根据每个单词的首字母在26个字母中出现的先后次序进行排序。（apple，banana，grape，orange）排名为（0,1,2,3）。

(0, 3) 1 解释为第一字符串的序号为3的词语出现次数为1。0表示第一个字符串"orange banana apple grape"。3为'orange'。1表示出现次数1。

4. TF-IDF

4.1 认识TF-IDF

TF-IDF（Term Frequency-Inverse Document Frequency，词频与逆向文件频率）用于评估词语对于文件的重要程度。

TF-IDF不仅考虑了词频，更考虑词语的稀有程度，词语的重要程度正比于其在文档中出现的频次，反比于其存在于多少篇文章中。当一个词语在某文章中出现较高频次，在其他文章中出现较低频次，说明该词语具有较好的区分度。

4. TF-IDF

4.2 计算TF-IDF

步骤1：计算TF

$$\text{词频(TF)} = \frac{\text{某个词在文章中的出现次数}}{\text{文章的总词数}}$$

步骤2：计算IDF

$$\text{逆文档频率(IDF)} = \log\frac{\text{总样本数}}{\text{包含有该词的文档数} + 1}$$

步骤3：计算TF-IDF

$$\text{TF} - \text{IDF算法} = TF\text{算法} \cdot IDF\text{算法}$$

4. TF-IDF

4.2 计算TF-IDF

【例12】TF-IDF计算

（1）计算TF

某文件共有100个词语，“苹果”出现3次，“苹果”在该文件中的词频就是3/100=0.03。

（2）计算IDF

“苹果”在1000个文档中出现，全部文件总数是10000000个，逆向文件频率是log(10000000/1000) = 4。

（3）计算TF-IDF

Tf-idf的值就是0.03 × 4 = 0.12。

4. TF-IDF

4.3 Sklearn调用TF-IDF

Sklearn提供TfidfVectorizer()函数实现

TfidfVectorizer(stop_words, sublinear_tf, max_df)

参数	说明
stop_words	停用词表
sublinear_tf	取值为True或False
max_df	文件频率阈值

表4-1 TfidfVectorizerr()参数说明

4. TF-IDF

4.3 Sklearn调用TF-IDF

【例13】TfidfVectorizer举例

```
# 例13: TF-IDF举例
from sklearn.feature_extraction.text import TfidfVectorizer
texts=['orange banana apple grape','banana apple apple','grape','orange apple']
cv = TfidfVectorizer()
cv_fit=cv.fit_transform(texts)
print(cv.vocabulary_)
print(cv_fit)
print(cv_fit.toarray())
```

```
{'orange': 3, 'banana': 1, 'apple': 0, 'grape': 2}
  (0, 2)        0.5230350301866413
  (0, 0)        0.423441934145613
  (0, 1)        0.5230350301866413
  (0, 3)        0.5230350301866413
  (1, 0)        0.8508160982744233
  (1, 1)        0.5254635733493682
  (2, 2)        1.0
  (3, 0)        0.6292275146695526
  (3, 3)        0.7772211620785797
[[0.42344193 0.52303503 0.52303503 0.52303503]
 [0.8508161  0.52546357 0.         0.        ]
 [0.         0.         1.         0.        ]
 [0.62922751 0.         0.         0.77722116]]
```

《自然语言处理入门》

5. LDA

5.1 认识LDA

LDA(Latent Dirichlet Allocation，潜在狄利克雷分布)是概率生成模型的一个典型代表，也将其称为 LDA 主题模型。Blei等于2003年提出的一种文档主题生成模型。

LDA模型是一个三层贝叶斯网络模型，其核心思想是每个文档对应一个服从Dirichlet分布$\vec{\theta}$主题分布，每个主题对应的词分布服从Dirichlet分布$\vec{\varphi}$，其中文档-主题分布α参数和主题-词分布β参数服从Dirichlet分布$\vec{\alpha},\vec{\beta}$。由于**LDA能够降低文本表示维度，**在文本主题识别、文本分类以及文本相似度计算方面广泛应用。

5. LDA

5.1 认识LDA

生成模型是指一篇文章的每个词可通过“以一定概率选择某个主题，并从这个主题中以一定概率选择某个词语”的过程。“主题”就是一个文本所蕴含的中心思想，一个文本可以有一个主体，也可以有多个主题。主题由关键词来体现，可以将主题看作是一种关键词集合，同一个词在不同的主题背景下，出现的概率不同，如一篇文章出现“林丹”的名字，这篇文章有很大概率属于体育的主题，但也有小概率属于娱乐的主题。

5. LDA

5.1 认识LDA

LDA把文章看作是由词汇组合而成，通过不同的词汇概率分布来反映不同的主题。 假设现在有两篇文章《体育快讯》和《经济周报》，有三个主题“体育”，“娱乐”，“ 经济”。LDA认为《体育快讯》的词汇分布概率为{体育:0.7, 经济:0.2，娱乐:0.1}，而《经济周报》的词汇分布概率为{体育:0.2, 经济:0.7，娱乐:0.1 }。

5. LDA

5.1 认识LDA

LDA是一种无监督机器学习技术，可以用来识别大规模文档集（document collection）或语料库（corpus）中潜藏的主题信息。它采用了词袋（bag of words）的方法，这种方法将每一篇文档视为一个词频向量，从而将文本信息转化为了易于建模的数字信息。

5. LDA

5.2 Gensim实现LDA

Gensim是一个通过衡量词组（如整句或文档）模式来挖掘文档语义结构的工具，具有如下三大核心概念：文集（语料）、向量和模型。

5. LDA

5.2 Gensim实现LDA

(1)语料(Corpus)：一组原始文本的集合。 Gensim 会从文本中推断出结构、主题等，通常是一个可迭代的对象（如列表）。

(2)向量(Vector)：由一组文本特征构成的列表，是一段文本在 Gensim 中的内部表达。向量中的每一个元素是一个(key, value)的元组。

(3)模型(Model)：定义了两个向量空间的变换。

5. LDA

5.2 Gensim实现LDA

【例14】Gensim实现LDA

```
# 例17：Gensim实现LDA
from gensim import corpora,models
import jieba.posseg as jp,jieba

# 文本集
texts = [
    '美国女排没有输给中国女排，是输给了郎平',
    '为什么越来越多的人买MPV，而放弃SUV？跑一趟长途就知道了',
    '美国排球无缘世锦赛决赛，听听主教练的评价',
    '中国女排晋级世锦赛决赛，全面解析主教练郎平的执教艺术',
    '跑了长途才知道，SUV和轿车之间的差距',
    '家用的轿车买什么好']
print('文本内容：')
print(texts)
```


5. LDA

5.2 Gensim实现LDA

【例14】Gensim实现LDA

```
# 分词过程，然后每句话/每段话构成一个单词的列表
flags = ('n','nr','ns','nt','eng','v','d')   # 词性
stopwords = ('没','就','知道','是','才','听听','坦言','全面','越来越','评价','放弃','人')
words_ls = []
for text in texts:
    words = [word.word for word in jp.cut(text) if word.flag in flags and word.word not in stopwords]
    words_ls.append(words)
print('分词结果：')
print(word_ls)

# 去重，存到字典
dictionary = corpora.Dictionary(words_ls)
print(dictionary)
```


5. LDA

5.2 Gensim实现LDA

【例14】Gensim实现LDA

```
# 按照（词ID: 词频）构成corpus
corpus = [dictionary.doc2bow(words) for words in words_ls]
print('语料为词ID：词频')
print(corpus)

# 设置了num_topics = 2两个主题
# 第一个是汽车相关主题，第二个是体育相关主题
print('两个主题：汽车和体育')
lda = models.ldamodel.LdaModel(corpus = corpus,id2word = dictionary,num_topics = 2)
for topic in lda.print_topics(num_words = 4):
    print(topic)
print(lda.inference(corpus))
```


5. LDA

5.2 Gensim实现LDA

【例14】Gensim实现LDA

```
text5 = '中国女排向三连冠发起冲击'
bow = dictionary.doc2bow([word.word for word in jp.cut(text5) if word.flag in flags and word.word not in stopwords])
ndarray = lda.inference([bow])[0]
print(text5)
for e,value in enumerate(ndarray[0]):
    print('\t主题%d推断值%.2f'%(e,value))
```

```
文本内容：
['美国女排没有输给中国女排，是输给了郎平', '为什么越来越多的人买MPV，而放弃SUV？跑一趟长途就知道了', '美国排球无缘世锦赛决赛，听听主教练的评价', '中国女排晋级世锦赛决赛，全面解析主教练郎平的执教艺术', '跑了长途才知道，SUV和轿车之间的差距', '家用的轿车买什么好']
分词结果：
[['美国', '女排', '没有', '输给', '中国女排', '输给', '郎平'], ['买', 'MPV', 'SUV', '跑', '长途'], ['美国', '排球', '无缘', '世锦赛', '决赛', '主教练'], ['中国女排', '晋级', '世锦赛', '决赛', '主教练', '郎平', '执教', '艺术'], ['跑', '长途', 'SUV', '轿车', '差距'], ['家用', '轿车', '买']]
Dictionary<22 unique tokens: ['中国女排', '女排', '没有', '美国', '输给']...>
```


5. LDA

5.2 Gensim实现LDA

【例14】Gensim实现LDA

```
语料为词ID：词频
[[(0, 1), (1, 1), (2, 1), (3, 1), (4, 2), (5, 1)], [(6, 1), (7, 1), (8, 1), (9, 1), (10, 1)], [(3, 1), (11,
1), (12, 1), (13, 1), (14, 1), (15, 1)], [(0, 1), (5, 1), (11, 1), (12, 1), (13, 1), (16, 1), (17, 1), (18,
1)], [(7, 1), (9, 1), (10, 1), (19, 1), (20, 1)], [(8, 1), (20, 1), (21, 1)]]
两个主题：汽车和体育
(0, '0.076*"长途" + 0.075*"跑" + 0.075*"SUV" + 0.061*"轿车"')
(1, '0.075*"中国女排" + 0.074*"郎平" + 0.065*"输给" + 0.058*"决赛"')
(array([[0.66322577, 7.3367505 ],
       [5.4315414 , 0.5684447 ],
       [5.9135604 , 1.0864153 ],
       [0.6273671 , 8.372607  ],
       [5.4445825 , 0.5554045 ],
       [2.2646556 , 1.7353303 ]], dtype=float32), None)
中国女排向三连冠发起冲击
        主题0推断值0.56
        主题1推断值1.44
```


Thank you!