本文主要是对照scikit-learn的preprocessing章节结合代码简单的回顾下预处理技术的几种方法，主要包括标准化、数据最大最小缩放处理、正则化、特征二值化和数据缺失值处理。内容比较简单，仅供参考！

首先来回顾一下下面要用到的基本知识。

一、知识回顾

均值公式：

x¯=1n∑i=1nxi

方差公式：

s2=1n∑i=1n(xi−x¯)2

0-范数，向量中非零元素的个数。

1-范数：

||X||=∑i=1n|xi|

2-范数：

||X||2=(∑i=1nx2i)12

p-范数的计算公式：

||X||p=(|x1|p+|x2|p+...+|xn|p)1p

数据标准化：当单个特征的样本取值相差甚大或明显不遵从高斯正态分布时，标准化表现的效果较差。实际操作中，经常忽略特征数据的分布形状，移除每个特征均值，划分离散特征的标准差，从而等级化，进而实现数据中心化。

二、标准化(Standardization)，或者去除均值和方差进行缩放

公式为：(X-X\_mean)/X\_std 计算时对每个属性/每列分别进行.

将数据按其属性(按列进行)减去其均值，然后除以其方差。最后得到的结果是，对每个属性/每列来说所有数据都聚集在0附近，方差值为1。

首先说明下sklearn中preprocessing库里面的scale函数使用方法：

sklearn.preprocessing.scale(X, axis=0, with\_mean=True,with\_std=True,copy=True)

1

1

根据参数的不同，可以沿任意轴标准化数据集。

参数解释：

X：数组或者矩阵

axis：int类型，初始值为0，axis用来计算均值 means 和标准方差 standard deviations. 如果是0，则单独的标准化每个特征（列），如果是1，则标准化每个观测样本（行）。

with\_mean: boolean类型，默认为True，表示将数据均值规范到0

with\_std: boolean类型，默认为True，表示将数据方差规范到1

一个简单的例子

假设现在我构造一个数据集X，然后想要将其标准化。下面使用不同的方法来标准化X：

方法一：使用sklearn.preprocessing.scale()函数

方法说明：

X.mean(axis=0)用来计算数据X每个特征的均值；

X.std(axis=0)用来计算数据X每个特征的方差；

preprocessing.scale(X)直接标准化数据X。

将代码整理到一个文件中：

from sklearn import preprocessing

import numpy as np

X = np.array([[ 1., -1., 2.],

[ 2., 0., 0.],

[ 0., 1., -1.]])

# calculate mean

X\_mean = X.mean(axis=0)

# calculate variance

X\_std = X.std(axis=0)

# standardize X

X1 = (X-X\_mean)/X\_std

# use function preprocessing.scale to standardize X

X\_scale = preprocessing.scale(X)

1

2

3

4

5

6

7

8

9

10

11

12

13

1

2

3

4

5

6

7

8

9

10

11

12

13

最后X\_scale的值和X1的值是一样的，前面是单独的使用数学公式来计算，主要是为了形成一个对比，能够更好的理解scale()方法。

方法2：sklearn.preprocessing.StandardScaler类

该方法也可以对数据X进行标准化处理，实例如下：

from sklearn import preprocessing

import numpy as np

X = np.array([[ 1., -1., 2.],

[ 2., 0., 0.],

[ 0., 1., -1.]])

scaler = preprocessing.StandardScaler()

X\_scaled = scaler.fit\_transform(X)

1

2

3

4

5

6

7

1

2

3

4

5

6

7

这两个方法得到最后的结果都是一样的。

三、将特征的取值缩小到一个范围（如0到1）

除了上述介绍的方法之外，另一种常用的方法是将属性缩放到一个指定的最大值和最小值(通常是1-0)之间，这可以通过preprocessing.MinMaxScaler类来实现。

使用这种方法的目的包括：

1、对于方差非常小的属性可以增强其稳定性；

2、维持稀疏矩阵中为0的条目。

下面将数据缩至0-1之间，采用MinMaxScaler函数

from sklearn import preprocessing

import numpy as np

X = np.array([[ 1., -1., 2.],

[ 2., 0., 0.],

[ 0., 1., -1.]])

min\_max\_scaler = preprocessing.MinMaxScaler()

X\_minMax = min\_max\_scaler.fit\_transform(X)

1

2

3

4

5

6

7

1

2

3

4

5

6

7

最后输出：

array([[ 0.5 , 0. , 1. ],

[ 1. , 0.5 , 0.33333333],

[ 0. , 1. , 0. ]])

1

2

3

1

2

3

测试用例：

>>> X\_test = np.array([[ -3., -1., 4.]])

>>> X\_test\_minmax = min\_max\_scaler.transform(X\_test)

>>> X\_test\_minmax

array([[-1.5 , 0. , 1.66666667]])

1

2

3

4

1

2

3

4

注意：这些变换都是对列进行处理。

当然，在构造类对象的时候也可以直接指定最大最小值的范围：feature\_range=(min, max)，此时应用的公式变为：

X\_std=(X-X.min(axis=0))/(X.max(axis=0)-X.min(axis=0))

X\_minmax=X\_std/(X.max(axis=0)-X.min(axis=0))+X.min(axis=0))

1

2

1

2

四、正则化(Normalization)

正则化的过程是将每个样本缩放到单位范数(每个样本的范数为1)，如果要使用如二次型(点积)或者其它核方法计算两个样本之间的相似性这个方法会很有用。

该方法是文本分类和聚类分析中经常使用的向量空间模型（Vector Space Model)的基础.

Normalization主要思想是对每个样本计算其p-范数，然后对该样本中每个元素除以该范数，这样处理的结果是使得每个处理后样本的p-范数(l1-norm,l2-norm)等于1。

方法1：使用sklearn.preprocessing.normalize()函数

>>> X = [[ 1., -1., 2.],

... [ 2., 0., 0.],

... [ 0., 1., -1.]]

>>> X\_normalized = preprocessing.normalize(X, norm='l2')

>>> X\_normalized

array([[ 0.40..., -0.40..., 0.81...],

[ 1. ..., 0. ..., 0. ...],

[ 0. ..., 0.70..., -0.70...]])

1

2

3

4

5

6

7

8

1

2

3

4

5

6

7

8

方法2：sklearn.preprocessing.StandardScaler类

>>> normalizer = preprocessing.Normalizer().fit(X) # fit does nothing

>>> normalizer

Normalizer(copy=True, norm='l2')

1

2

3

1

2

3

然后使用正则化实例来转换样本向量：

>>> normalizer.transform(X)

array([[ 0.40..., -0.40..., 0.81...],

[ 1. ..., 0. ..., 0. ...],

[ 0. ..., 0.70..., -0.70...]])

>>> normalizer.transform([[-1., 1., 0.]])

array([[-0.70..., 0.70..., 0. ...]])

1

2

3

4

5

6

1

2

3

4

5

6

两种方法都可以，效果是一样的。

五、二值化(Binarization)

特征的二值化主要是为了将数据特征转变成boolean变量。在sklearn中，sklearn.preprocessing.Binarizer函数可以实现这一功能。实例如下：

>>> X = [[ 1., -1., 2.],

... [ 2., 0., 0.],

... [ 0., 1., -1.]]

>>> binarizer = preprocessing.Binarizer().fit(X) # fit does nothing

>>> binarizer

Binarizer(copy=True, threshold=0.0)

>>> binarizer.transform(X)

array([[ 1., 0., 1.],

[ 1., 0., 0.],

[ 0., 1., 0.]])

1

2

3

4

5

6

7

8

9

10

1

2

3

4

5

6

7

8

9

10

Binarizer函数也可以设定一个阈值，结果数据值大于阈值的为1，小于阈值的为0，实例代码如下：

>>> binarizer = preprocessing.Binarizer(threshold=1.1)

>>> binarizer.transform(X)

array([[ 0., 0., 1.],

[ 1., 0., 0.],

[ 0., 0., 0.]])

1

2

3

4

5

1

2

3

4

5

六、缺失值处理

由于不同的原因，许多现实中的数据集都包含有缺失值，要么是空白的，要么使用NaNs或者其它的符号替代。这些数据无法直接使用scikit-learn分类器直接训练，所以需要进行处理。幸运地是，sklearn中的Imputer类提供了一些基本的方法来处理缺失值，如使用均值、中位值或者缺失值所在列中频繁出现的值来替换。

下面是使用均值来处理的实例：

>>> import numpy as np

>>> from sklearn.preprocessing import Imputer

>>> imp = Imputer(missing\_values='NaN', strategy='mean', axis=0)

>>> imp.fit([[1, 2], [np.nan, 3], [7, 6]])

Imputer(axis=0, copy=True, missing\_values='NaN', strategy='mean', verbose=0)

>>> X = [[np.nan, 2], [6, np.nan], [7, 6]]

>>> print(imp.transform(X))

[[ 4. 2. ]

[ 6. 3.666...]

[ 7. 6. ]]

1

2

3

4

5

6

7

8

9

10

1

2

3

4

5

6

7

8

9

10

Imputer类同样支持稀疏矩阵：

>>> import scipy.sparse as sp

>>> X = sp.csc\_matrix([[1, 2], [0, 3], [7, 6]])

>>> imp = Imputer(missing\_values=0, strategy='mean', axis=0)

>>> imp.fit(X)

Imputer(axis=0, copy=True, missing\_values=0, strategy='mean', verbose=0)

>>> X\_test = sp.csc\_matrix([[0, 2], [6, 0], [7, 6]])

>>> print(imp.transform(X\_test))

[[ 4. 2. ]

[ 6. 3.666...]

[ 7. 6. ]]

1

2

3

4

5

6

7

8

9

10

1

2

3

4

5

6

7

8

9

10

本文讲解的比较接单，如果对这些不是很理解的话，请到scikit-learn的官网中查看英文版本：preprocessing.

References

Scikit-learn preprocessing.