机器学习001-数据预处理技术（均值移除，范围缩放，归一化，二值化，独热编码） -

(【本文所使用的Python库和版本号】: Python 3.5, Numpy 1.14, scikit-learn 0.19, matplotlib 2.2)

数据预处理的必要性：在真实世界中，经常需要处理大量的原始数据，这些原始数据是机器学习算法无法理解的，所以为了让机器学习算法理解原始数据，需要对数据进行预处理。

为什么要标准化处理？一个百分制的变量与一个5分制的变量在一起怎么比较？只有通过数据标准化，都把它们标准到同一个标准时才具有可比性 。

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。

最常用的数据预处理技术：

1. 均值移除（Mean removal）

去除均值和方差进行缩放。（均值为0，标准差为1）

```
###########对数据集进行Normalization#########################
import numpy as np
from sklearn import preprocessing

data=np.array([[3, -1.5, 2, -5.4],
               [0, 4,-0.3,2.1],
               [1, 3.3, -1.9, -4.3]]) # 原始数据矩阵 shape=(3,4)

data_standardized=preprocessing.scale(data) #0均值处理

print(data_standardized.shape)
print('Mean={}'.format(data_standardized.mean(axis=0))) #对列求均值（浮点计算有误差，接近0）
print('Mean2={}'.format(np.mean(data_standardized,axis=0))) #对列求均值（浮点计算有误差，接近0）
# 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）。
print('标准化后：')
print(data_standardized)
print('标准差={}'.format(np.std(data_standardized,axis=0)))
```

----------输---------出----------

(3, 4) Mean=[5.55111512e-17 -1.11022302e-16 -7.40148683e-17 -7.40148683e-17] Mean2=[5.55111512e-17 -1.11022302e-16 -7.40148683e-17 -7.40148683e-17] standardized: [[1.33630621 -1.40451644 1.29110641 -0.86687558] [-1.06904497 0.84543708 -0.14577008 1.40111286] [-0.26726124 0.55907936 -1.14533633 -0.53423728]] STD=[1. 1. 1. 1.]

----------完----------

##########小********结##########**

1, 值移除之后的矩阵每一列的均值约为0，而std为1。这样做的目的是确保每一个特征列的数值都在类似的数据范围之间，防止某一个特征列数据天然的数值太大而一家独大。

2, 可以直接调用preprocessing模块中成熟的 方法来对一个numpy 矩阵进行均值移除。

3, 求一个numpy矩阵的平均值（或std，min,max等）至少有两种方法，如代码中第9行和第10行所示。

##

2. 范围缩放（Scaling）

必要性：数据点中每个特征列的数值范围可能变化很大，因此，有时需要将特征列的数值范围缩放到合理的大小。

```
###########对数据集进行范围缩放#########################
import numpy as np
from sklearn import preprocessing

data=np.array([[3, -1.5, 2, -5.4],
               [0, 4,-0.3,2.1],
               [1, 3.3, -1.9, -4.3]]) # 原始数据矩阵 shape=(3,4)

data_scaler=preprocessing.MinMaxScaler(feature_range=(0,1)) # 缩放到 (0,1) 之间
data_scaled=data_scaler.fit_transform(data)
print('scaled matrix: ********************************')
print(data_scaled)
```

----------输---------出----------

scaled matrix: ********** [[1. 0. 1. 0.] [0. 1. 0.41025641 1.] [0.33333333 0.87272727 0. 0.14666667]]

----------完----------

3. 归一化（Normalization）

用于需要对特征向量的值进行调整时，以保证每个特征向量的值都缩放到相同的数值范围。机器学习中最常用的归一化形式就是将特征向量调整为L1范数，归一化就是L1/L2为1

3.1 范数

$$P\text{范数} = (|x_1|^p + |x_2|^p + \ldots)^{\frac{1}{p}}$$

$$x = [[1,-1,2],[2,0,0],[0,1,-1]]$$

对第一行求L_1范数：$L_1 = (1+1+2) = 4$

对第一行计算:$[[1/4,-1/4,2/4]]$

对第二行求L_2范数：$L_2 = (1+1+4)^{\frac{1}{2}} = \sqrt{6}$

对第二行计算:$[[1/\sqrt{6},-1/\sqrt{6},2/\sqrt{6}]]$

@xuebaohua

给定向量x=（x1，x2，...xn）
L1范数：向量各个元素（向量分量）绝对值之和
L2范数：向量各个元素（向量分量）的平方求和然后求平方根

```
###########对数据集进行Normalization#########################
import numpy as np
from sklearn import preprocessing

data=np.array([[3, -1.5, 2, -5.4],
               [0, 4,-0.3,2.1],
               [1, 3.3, -1.9, -4.3]]) # 原始数据矩阵 shape=(3,4)

data_L1_normalized=preprocessing.normalize(data,norm='l1')
print('L1 normalized matrix: **********')
print(data_L1_normalized)
print('sum of matrix: {}'.format(np.sum(data_L1_normalized)))

data_L2_normalized=preprocessing.normalize(data,norm='l2') # 默认：l2
print('L2 normalized matrix: **********')
print(data_L2_normalized)
print('sum of matrix: {}'.format(np.sum(data_L2_normalized)))
```

----------输--------出----------

L1 normalized matrix: ********** [[0.25210084 -0.12605042 0.16806723 -0.45378151] [0. 0.625 -0.046875 0.328125] [0.0952381 0.31428571 -0.18095238 -0.40952381]] sum of matrix: 0.5656337535014005 L2 normalized matrix: ********** [[0.45017448 -0.22508724 0.30011632 -0.81031406] [0. 0.88345221 -0.06625892 0.46381241] [0.17152381 0.56602858 -0.32589524 -0.73755239]] sum of matrix: 0.6699999596689536

----------完----------

##########################小********结##################################**

1，Normaliztion之后所有的特征向量的值都缩放到同一个数值范围，可以确保数据点没有因为特征的基本性质而产生的较大差异，即确保所有数据点都处于同一个数据量，提高不同特征数据的可比性。

2，注意和均值移除的区别：均值移除是对每一个特征列都缩放到类似的数值范围，每一个特征列的均值为0，而Normalization是将全局所有数值都缩放到同一个数值范围。

###

4. 二值化（Binarization）

二值化用于将数值特征向量转换为布尔类型向量。

```
###########对数据集进行Binarization#########################
import numpy as np
from sklearn import preprocessing

data=np.array([[3, -1.5, 2, -5.4],
               [0, 4,-0.3,2.1],
               [1, 3.3, -1.9, -4.3]]) # 原始数据矩阵 shape=(3,4)

data_binarized=preprocessing.Binarizer(threshold=1.4).fit_transform(data) #小于等于 threshold 为
0
print('binarized matrix: **********************************')
print(data_binarized)
```

----------------------------------输---------出-----------------------------

binarized matrix: ********** [[1. 0. 1. 0.] [0. 1. 0. 1.] [0. 1. 0. 0.]]

---完----------------------------------

##########################小********结##################################**

1，二值化之后的数据点都是0或者1，所以叫做二值化。

2，计算方法是，将所有大于threshold的数据都改为1，小于等于threshold的都设为0。

3，经常用于出现某种特征（比如设为1），或者没有出现某种特征（设为0）的应用场合。

###

5. 独热编码（One-Hot Encoding）

通常，需要处理的数值都是稀疏地，散乱地分布在空间中，但我们并不需要存储这些大数值，这时就需要使用独热编码，独热编码实际上是一种收紧特征向量的工具。

```
###########对数据集进行独热编码#########################

import numpy as np
```

```
from sklearn import preprocessing

data=np.array([[0,2,1,12],
               [1,3,5,3],
               [2,3,2,12],
               [1,2,4,3]]) # 原始数据矩阵 shape=(4,4)

encoder=preprocessing.OneHotEncoder()
encoder.fit(data)
encoded_vector=encoder.transform([[2,3,5,3]]).toarray()
print('one-hot encoded matrix: ********************************')
print(encoded_vector.shape)
print(encoded_vector)
```

----------输---------出----------

one-hot encoded matrix: ********** (1, 11) [[0. 0. 1. 0. 1. 0. 0. 0. 1. 1. 0.]]

----------完----------

##########小********结##########**

1，独热编码可以缩小特征向量的维度，将稀疏的，散乱的数据集（比如代码块中的data，shape=(4,4)）收缩为11维致密矩阵（如输出结果，shape=(1,11)）。

2，编码方式为：根据原始数据集data构建编码器encoder，用编码器来对新数据进行编码。比如，第0列有三个不同值（0,1,2），故而有三个维度，即0=100，1=010，2=001；同理，第1列有两个不同值（2,3），故而只有两个维度，即2=10，3=01；同理，第2列有四个不同值（1,5,2,4），故而有四个维度，即1=1000，2=0100,4=0010,5=0001同理，第3列有两个不同值（3,12），故而只有两个维度，即3=10，12=01。所以在面对新数据[[2,3,5,3]]时，第0列的2就对应于001，第二列的3对应于01，第三列的5对应于0001，第四列的3对应于10，连接起来后就是输出的这个（1,11）矩阵，即为读了编码后的致密矩阵。

3，如果面对的新数据不存在上面的编码器中，比如[[2,3,5,4]]时，4不存在于第3列（只有两个离散值3和12），则输出为00，连接起来后是[[0. 0. 1. 0. 1. 0. 0. 0. 1. 0. 0.]]，注意倒数第二个数字变成了0

6.数据离散化

作用：将连续型数据离散化

```
ages = [20,33,54,23,66,77,88,99,26,63]
bins = [18,25,35,60,100]
labels = ['少年','青年','中年','老年']
new_ages = pd.cut(x=ages,bins=bins,labels=labels,retbins=True)#retbins:返回bins
print(new_ages)
```

----------输---------出----------

([少年, 青年, 中年, 少年, 老年, 老年, 老年, 老年, 青年, 老年] Categories (4, object): [少年 < 青年 < 中年 < 老年], array([18, 25, 35, 60, 100]))

##########

参考资料:

1, Python机器学习经典实例，Prateek Joshi著，陶俊杰，陈小莉译