



# sklearn中的数据预处理

🕒 Reading time ~4 minutes

## 1. 介绍

sklearn.preprocessing提供了各种公共函数，来将raw feature vector转换成另外一种更适合评估器工作的格式。

## 2. 标准化(Standardization)、平均移除法 ( mean removal ) 和方差归一化 ( variance scaling )

数据集的标准化，在scikit中，对于众多机器学习评估器来说是必须的；如果各独立特征不进行标准化，结果标准正态分布数据差距很大：比如使用均值为0、方差为1的高斯分布。

实际上，我们经常忽略分布的形状，只是简单地通过移除每个feature的均值，将数据转换成中间值，接着通过分割非常数项的feature进行归一化。

例如，在机器学习的目标函数 ( objective function )，比如SVM中的RBF或者线性模型中的L1和L2正则项，其中使用的元素，前提是所有的feature都是以0为中心，且方差的阶 ( order ) 都一致。如果一个feature的方差，比其它feature的阶 ( order ) 都大，那么它将在目标函数中占支配地位，从而使得estimator从其它feature上学习不到任何东西。

scale函数提供了一个快速而简单的方式：

```
>>> from sklearn import preprocessing
>>> import numpy as np
>>> X = np.array([[ 1., -1., 2.],
...               [ 2., 0., 0.],
...               [ 0., 1., -1.]])
>>> X_scaled = preprocessing.scale(X)
```

```
>>> X_scaled
array([[ 0. ..., -1.22...,  1.33...],
       [ 1.22...,  0. ..., -0.26...],
       [-1.22...,  1.22..., -1.06...]])
```

我们可以看到，归一化后的数据，均值为0，方差为1：

```
>>> X_scaled.mean(axis=0)
array([ 0.,  0.,  0.])

>>> X_scaled.std(axis=0)
array([ 1.,  1.,  1.])
```

preprocessing模块提供了另一个工具类：StandardScaler，它实现了Transformer API，来计算在一个训练集上的平均值和标准差（standard deviation），同样需要在测试集上使用相同的转换。该类也可以应用在sklearn.pipeline.Pipeline上。

```
>>> scaler = preprocessing.StandardScaler().fit(X)
>>> scaler
StandardScaler(copy=True, with_mean=True, with_std=True)

>>> scaler.mean_
array([ 1. ...,  0. ...,  0.33...])

>>> scaler.scale_
array([ 0.81...,  0.81...,  1.24...])

>>> scaler.transform(X)
array([[ 0. ..., -1.22...,  1.33...],
       [ 1.22...,  0. ..., -0.26...],
       [-1.22...,  1.22..., -1.06...]])
```

scaler实例，可以用在新数据上，并可以以相同的方式在训练集上转换：

```
>>> scaler.transform([[-1.,  1.,  0.]])
array([[ -2.44...,  1.22..., -0.26...]])
```

通过在StandardScaler的构造函数中设置with\_mean=False 或者 with\_std=False，可以禁止均值中心化（centering）和归一化（scaling）。

## 2.1 将feature归一化到一个范围内

另一种标准化方式是，将feature归一化到给定的最大、最小值范围内，比如：[0,1]之间，这样，每个feature的最大绝对值为1. 可以使用：MinMaxScaler或者MaxAbsScaler。

使用归一化的动机是，阻止sparse中的0元素，让含有小标准差的feature变得更健壮。

下例：归一化至[0, 1]

```
>>> X_train = np.array([[ 1., -1., 2.],
...                     [ 2., 0., 0.],
...                     [ 0., 1., -1.]])
...
>>> min_max_scaler = preprocessing.MinMaxScaler()
>>> X_train_minmax = min_max_scaler.fit_transform(X_train)
>>> X_train_minmax
array([[ 0.5, 0., 1.],
       [ 1., 0.5, 0.33333333],
       [ 0., 1., 0.]])
```

可以通过查看scaler的属性，来看下训练集上的转换是否合理。

```
>>> min_max_scaler.scale_
array([ 0.5, 0.5, 0.33...])

>>> min_max_scaler.min_
array([ 0., 0.5, 0.33...])
```

如果MinMaxScaler给出了显式的范围：feature\_range=(min, max)。那么对应完整的公式为：

```
X_std = (X - X.min(axis=0)) / (X.max(axis=0) - X.min(axis=0))
```

```
X_scaled = X_std / (max - min) + min
```

MaxAbsScaler以相类似的方式运作，它的归一化会介于[-1, 1]之间，通过在每个feature上分隔最大值来实现。这意味着数据已经是0为中心的，或者是稀疏数据。

这里我们使用一个示例数据：

```
>>> X_train = np.array([[ 1., -1., 2.],
...                     [ 2., 0., 0.],
...                     [ 0., 1., -1.]])
...
>>> max_abs_scaler = preprocessing.MaxAbsScaler()
>>> X_train_maxabs = max_abs_scaler.fit_transform(X_train)
>>> X_train_maxabs      # doctest +NORMALIZE_WHITESPACE^
array([[ 0.5, -1., 1.],
       [ 1., 0., 0.],
       [ 0., 1., -0.5]])
>>> X_test = np.array([[ -3., -1., 4.]])
>>> X_test_maxabs = max_abs_scaler.transform(X_test)
>>> X_test_maxabs
array([[ -1.5, -1., 2.]])
>>> max_abs_scaler.scale_
array([ 2., 1., 2.]])
```

如果你不想创造对象，scale模块提供了另外的函数：minmax\_scale 和 maxabs\_scale。

## 2.2 归一化sparse矩阵

如果对sparse数据进行中心化，会摧毁数据的稀疏性，十分敏感。我们可以对sparse数据进行特殊的归一化，尤其各种feature以不同的归一化方式进行。

**MaxAbsScaler** 和 **maxabs\_scale** 是专门处理稀疏数据的，强烈推荐这种方式。然而，scale 和 **StandardScaler** 可以接受scipy.sparse的metrics作为输入，只要在构造函数中显示传入 with\_centering=False。否则会抛出ValueError，打破sparsity，并且在执行时由于分配大量内存而经常crash。RobustScaler 不能对稀疏数据进行fit，但是你可以在稀疏数据输入上使用transform方法。

注意：scaler接受压缩式稀疏数据行（Compressed Sparse Rows），以及压缩式稀疏数据列（Compressed Sparse Columns），分别参见：`scipy.sparse.csr_matrix` 和 `scipy.sparse.csc_matrix`。其它稀疏输入可以转换成CSR表示。为了避免不必要的内存拷贝，推荐你使用**CSR**或者**CSC**表示法。

最后，如果该中心化后的数据如预期般足够小，可以通过sparse matrices的toarray方法转成一个数组。

## 2.3 归一化异常数据

如果你的数据包含许多异常项（outliers），使用均值和方差的效果不会很好。在这种情况下，我们可以使用**robust\_scale** 和 **RobustScaler**作为替代。它可以在你给定的中心或者范围内给出健壮的估计。

### Scaling vs Whitening

对于独自进行中心化和归一化来说有时并不够好。因为下游的模型可能会基于feature的线性独立性做出一些假设。

你可以使用 `sklearn.decomposition.PCA` 或 `sklearn.decomposition.RandomizedPCA`，并设置**whiten=True**以便移除feature之间的线性相关性。

## 2.4 kernel matrices的中心化

如果你有一个kernel为K的kernel矩阵，通过定义函数phi计算在特征空间内的内积（点乘），`KernelCenterer`可以对kernel矩阵进行转换，以便它在移除空间的均值后，能包含通过phi函数定义的特征空间的内积。

## 3.正态分布化（Normalization）

Normalization用来将各个样本归一化为norm为1的正态分布。如果你要使用二项式形式（比如点乘、或者其它kernel来确定样本相似性）

该假设是向量空间模型（VSM）的基础，经常用于文本分类和内容聚类。

函数 `normalize` 提供了一个简单的方法来操作类似数组的数据集，使用l1或l2范式：

```
>>> 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...]])
```

`preprocessing`模块提供了一个工具类：`Normalizer`。

```
>>> normalizer = preprocessing.Normalizer().fit(X) # fit does nothing
>>> normalizer
Normalizer(copy=True, norm='l2')
```

`normalizer`实例可以作为转换器被用在样本向量上：

```
>>> 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. ...]])
```

对于稀疏矩阵输入来说：

`normalize`和 `Normalizer`同时接受dense array、或者`scipy.sparse`输入。

对于sparse来说，数据被转换成CSR形式（`scipy.sparse.csr_matrix`）。

## 4.二值化（Binarization）

Feature二值化可以将数值形（numerical）的feature进行阈值化得到boolean型数据。这对于下游的概率估计来说可能很有用（比如：数据分布为Bernoulli分布时）。例如，sklearn.neural\_network.BernoulliRBM的case也是。

在文本处理社区中，使用二值feature也很常用（可以简化概率模型），如果归一化的count（比如：词频TF）或者TF-IDF值feature经常很有用。

对于Normalizer来说，工具类Binarizer可以在sklearn.pipeline.Pipeline的早期使用。

```
>>> 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.]])
```

我们有可能调整binarizer的threshold：

```
>>> binarizer = preprocessing.Binarizer(threshold=1.1)
>>> binarizer.transform(X)
array([[ 0., 0., 1.],
       [ 1., 0., 0.],
       [ 0., 0., 0.]])
```

对于StandardScaler 或 Normalizer来说，preprocessing模块提供了另一个函数binarize。

## 5.将类别特征进行编码

经常，有些特征并不是连续的，可能是类别化的。比如：

- 性别：[“male”, “female”]

- 国家 : [“from Europe”, “from US”, “from Asia”]
- 使用的浏览器 : [“uses Firefox”, “uses Chrome”, “uses Safari”, “uses Internet Explorer”]

这样的feature可以用整型表示，例如：

- [“male”, “from US”, “uses Internet Explorer”]表示为：[0, 1, 3]
- [“female”, “from Asia”, “uses Chrome”]表示为：[1, 2, 1]

一些整型表示可以被直接用在sklearn的评估器上，当成连续输入，或者解释成类别。

将类别feature转换成sklearn评估器可用的feature的一种方式为：使用one-of-K，或者one-hot编码，它可以用 OneHotEncoder来实现。该评估器将m个可能的feature类别值转换成m个二元feature。

例如：

```
>>> enc = preprocessing.OneHotEncoder()
>>> enc.fit([[0, 0, 3], [1, 1, 0], [0, 2, 1], [1, 0, 2]])
OneHotEncoder(categorical_features='all', dtype=<... 'float'>,
              handle_unknown='error', n_values='auto', sparse=True)
>>> enc.transform([[0, 1, 3]]).toarray()
array([[ 1.,  0.,  0.,  1.,  0.,  0.,  0.,  0.,  1.]])
```

缺省的，每个feature可以取多少值可以从数据集中自动判断。通过使用n\_values参数可以显示指定。在我们的数据集中有两个性别（gender），三个可能的大洲（continents），还有4种浏览器（web browser）。我们去拟合该estimator，并进行数据转换。在结果中，头两个数表示性别的编码，接下去的3个数表示大洲的编码，最后4个则用于浏览器。

详见字典载入feature

## 6. 补充缺失值

出于其他原因，现实世界中，有许多数据集中包含着缺失值(missing values)，经常编码成空格，NaN或者其它占位符。这样的数据集对于sklearn来说是不兼容的，它认为的输入数据必须是全是数值型的。一个基本策略是，使用不完整的数据（抛弃掉那些带缺失值的行）。然而，缺失的数据中也有可能含有有价值的信息（尽管不完整）。另一个更好地策略是，插



入缺失值，比如：从已经数据中去模拟它们。

Imputer类提供了基本策略来补充缺失值，或者使用均值、中值、或者行中最常用的值、或者缺失值所在列中最常用的值。该类提供了不同的缺失值补充策略。

下面的代码段演示了如何使用np.nan替换缺失值，使用了包含了缺失值的列(axis 0)的均值：

```
>>> 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.   ]]
```

Imputer类提支持稀疏策略：

```
>>> 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.   ]]
```

注意，缺失值用0编码，并被显式地存于矩阵中。如果缺失值的数目比观测值还要多时，这种格式很合适。

Imputer可以用在Pipeline上，来构建一个支持插值（imputation）的组合estimator。详见  
Imputing missing values before building an estimator

## 7.生成多项式特征（polynomial features）

通常，由于考虑到输入数据的非线性feature的存在，你的模型变复杂。一个简单的方法是，使用多项式feature，它可以给出feature的高阶表示以及交叉项（interaction terms）。通过PolynomialFeatures实现：

```
>>> import numpy as np
>>> from sklearn.preprocessing import PolynomialFeatures
>>> X = np.arange(6).reshape(3, 2)
>>> X
array([[0, 1],
       [2, 3],
       [4, 5]])
>>> poly = PolynomialFeatures(2)
>>> poly.fit_transform(X)
array([[ 1.,  0.,  1.,  0.,  0.,  1.],
       [ 1.,  2.,  3.,  4.,  6.,  9.],
       [ 1.,  4.,  5., 16., 20., 25.]])
```

X的feature可以通过(X1,X2) 转换成

在一些情况下，只有feature间的交叉项是必须的（忽略高阶项），可以通过interaction\_only=True来实现：

```
>>> X = np.arange(9).reshape(3, 3)
>>> X
array([[0, 1, 2],
       [3, 4, 5],
       [6, 7, 8]])
>>> poly = PolynomialFeatures(degree=3, interaction_only=True)
>>> poly.fit_transform(X)
array([[ 1.,  0.,  1.,  2.,  0.,  0.,  2.,  0.],
       [ 1.,  3.,  4.,  5., 12., 15., 20., 60.],
       [ 1.,  6.,  7.,  8., 42., 48., 56., 336.]])
```

X的feature可以从(X1,X2,X3)转换成

注意：多项式feature只能显示地用在核方法（比如：`sklearn.svm.SVC`，`sklearn.decomposition.KernelPCA`）上的多项式核函数（polynomial Kernel functions）上。

Ridge regression的多项式内插（Polynomial interpolation）用它来创建多项式feature。

## 8.定制转换器

通常，你需要将一个存在的python函数转换成一个transformer来进行数据清洗和处理。你可以使用一个专用函数来实现一个转换器：FunctionTransformer。例如，我们可以在一个pipeline上使用一个log函数来进行转换：

```
>>> import numpy as np
>>> from sklearn.preprocessing import FunctionTransformer
>>> transformer = FunctionTransformer(np.log1p)
>>> X = np.array([[0, 1], [2, 3]])
>>> transformer.transform(X)
array([[ 0.        ,  0.69314718],
       [ 1.09861229,  1.38629436]])
```

详见原文.

参考：

1.<http://scikit-learn.org/stable/modules/preprocessing.html#preprocessing>

 LIKE  TWEET  +1

0 Comments d0evi1

 Login ▾ Recommend 2  Share

Sort by Best ▾



Start the discussion...

LOG IN WITH

OR SIGN UP WITH DISQUS 

Name

Be the first to comment.

## ALSO ON D0EVI1

## 我的推荐：机器学习资料 – d0evi1的博客

1 comment • a year ago

Yiyi Liu — 工具篇找不到网页了亲TAT

## libfm 1.4.2 manual – d0evi1的博客

1 comment • 10 months ago

Hongru Liu — 请问libfm学习后的结果在哪里会展示出来(如果不指定out的话)?

## sklearn中的模型评估 – d0evi1的博客




4 comments • a year ago

法布雷亚戈 — 谢谢。有空我改一下，现在免费可用的图片外链都越来越少了。  
----- 原始邮件 ...

## corpora.dictionary - 构建 word id 映射

1 comment • 7 months ago

鄭宜崴 — 想請問 當我們建立Dictionary或Corpus後要如何知道哪一個id 對應到 哪一個word ...

 Subscribe  Add Disqus to your siteAdd DisqusAdd  Privacy

© 2017 d0evi1. Powered by Jekyll using the HPSTR Theme.