

用scikit-learn做数据预处理



Jeriah (/u/4e1f3ce26c72) [+ 关注](#)

2017.03.04 22:58* 字数 1197 阅读 233 评论 0 喜欢 4

(/u/4e1f3ce26c72)

数据预处理是进行机器学习的必要环节，对原始数据进行加工，比如标准化、归一化、二进制化、特征编码、插补缺失值、生成多项式特征等。

scikit-learn (<http://scikit-learn.org/stable/index.html>) 是 Python 下的机器学习工具包，集成了预处理、降维、聚类、分类等方法。我们用此工具来介绍数据预处理。

主要应用的是 preprocessing 类。

标准化 Standardization

常用的标准化方法为 **z-score** 法，目的是将传入的矩阵变为每列均值为0、方差为1的标准型，因为机器学习中的函数许多是以0为中心的，例如 *sigmoid* 函数，方差为1可以使数据分布均匀，防止某个特征数据过大影响模型的训练。

$$X_{ij} = \frac{(X_{ij} - mean)}{variance}$$

z-score 公式

我们用 sklearn 中的 preprocessing 类中的 scale 方法简单进行标准化。

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



或者可以将计算均值和方差与转换的步骤分开。

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

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

fit 是计算过程，transform 是转换过程。transform 的一个好处是可以继续使用 fit 的数据，例如 transform 训练数据之后之后可以直接 transform 测试数据。

有时标准化时要求把数据缩放到某一个范围中，preprocessing 类也提供了 MinMaxScaler 方法，设置 MinMaxScaler 的参数 feature_range=(min, max)。当然最常用的就是 (0,1) 范围。

```
>>> X_train = np.array([[ 1., -1.,  2.],
...                     [ 2.,  0.,  0.],
...                     [ 0.,  1., -1.]])
...
>>> min_max_scaler = preprocessing.MinMaxScaler(feature_range=(0, 1))
>>> 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.        ]])
```

归一化 Normalization

归一化是将每个样本缩放到单位范数（可以粗略地理解为向量的函数）的过程。如果后面要使用如二次型（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。归一化的计算方法是 1除以p-范数。归一化之后的样本的 p-范数等于1。



向量的 p-范数

1,文字表达:

若 x 为 n 维向量, 那么定义 p -范数为:

当 $p=1, 2, \infty$ 时候是比较常用的范数。

1-范数是向量各个分量绝对值之和。

2-范数 (Euclid 范数) 就是通常所说的向量的长度。

∞ -范数 是通常所说的最大值范数, 指的是向量各个分量绝对值的最大值。

2,数学表达:

令 $x=(x_1, x_2, \dots, x_n)^T$ (T 是转置的意思)

1-范数: $\|x\|_1 = |x_1| + |x_2| + \dots + |x_n|$

2-范数: $\|x\|_2 = (|x_1|^2 + |x_2|^2 + \dots + |x_n|^2)^{1/2}$

∞ -范数: $\|x\|_\infty = \max(|x_1|, |x_2|, \dots, |x_n|)$

易得推论: $\|x\|_\infty \leq \|x\|_2 \leq \|x\|_1 \leq \sqrt{2} \|x\|_2 \leq \|x\|_\infty$

向量的p-范数

preprocessing 类下的 normalize 方法可以直接归一化, 并可以选择 l1 范数与 l2 范数, 默认是 l2 范数。

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

normalize 也同样有 fit 方法与 transform 方法。



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

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

二进制化 Binarization

二进制化的方法是设定一个阈值，样本值比阈值大的为1，小的为0。

文本处理领域愿意使用二进制化，虽然归一化和 TF-IDF 加工的特征表现得更好，但使用二进制化来简化数据更为普遍（可能简化概率推理）。

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

阈值调整需要设置 threshold 参数，下面代码是设置阈值1.1的情况。

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

类别特征编码 Encoding categorical features

比如现在有特征 ["male", "female"] ["from Europe", "from US", "from Asia"], ["uses Firefox", "uses Chrome", "uses Safari", "uses Internet Explorer"], 平常我们将特征进行整数编码。

["male", "from US", "uses Internet Explorer"] 编码应该是 [0, 1, 3]。



["female", "from Asia", "uses Chrome"]的编码应该是[1,2,1]。

但是 scikit-learn 估计器不能直接使用这种整数型的编码，它的输入最好是连续型的，估计器也可能把整数编码特征特征误解为有序的数据，这会有害于模型训练。

类别特征编码用的是 OneHotEncoder 方法，占第几位的特征值就是1，其余为0。例如 ["male", "female"] 有两个特征，male 占第1个位置，剩下的0，male 就是 [1,0]，female 就是 [0,1]；["from Europe", "from US", "from Asia"]有3个特征，"from Europe"就是 [1,0,0]，"from US" 就是 [0,1,0]，"from Asia"就是 [0,0,1]，以此类推。

["male", "from US", "uses Internet Explorer"]的整数编码[0,1,3]的二进制编码就是 [1,0, 0,1,0, 0,0,0,1]。

代码实现用的是 preprocessing 类下的 OneHotEncoder 方法。

```
>>> enc = preprocessing.OneHotEncoder()
>>> enc.fit([[0, 0, 3], [1, 1, 0], [0, 2, 1], [1, 0, 2]])
OneHotEncoder(categorical_features='all', dtype=<... 'numpy.float64'>,
              handle_unknown='error', n_values='auto', sparse=True)
>>> enc.transform([[0, 1, 3]]).toarray()
array([[ 1.,  0.,  0.,  1.,  0.,  0.,  0.,  0.,  1.]])
```

插补缺失值 Imputation of missing values

以往对于数据中有丢失值的情况，做法是把该数据所在行和列都删掉，这样的后果是容易丢失重要的数据。

imputer 的方法利用平均值、中位数或出现频率最高的数据对缺失的数据进行填补。代码如下，NaN 即丢失的值。

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

strategy 告诉了本代码用 mean 平均值找补，fit 是用它括号里的数据计算平均值，transform 是应用变换。



生成多项式特征 Generating polynomial features

生成多项式的预处理方式是为了给输入数据增加复杂性。处理后的特征会获得高阶的数据和交叉的数据。

用到的是 preprocessing 下的 PolynomialFeatures 方法。

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

PolynomialFeatures(2) 指的是最多生成二次项。

X的特征值由 (X1, X2) 变成了 (1, X1, X2, X1^2, X1X2, X2^2)。

机器学习 (/nb/10411500)

举报文章 © 著作权归作者所有



Jeriah (/u/4e1f3ce26c72) ♂

写了 10613 字, 被 1 人关注, 获得了 12 个喜欢
(/u/4e1f3ce26c72)

+ 关注

起舞弄清影 何似在人间

♡ 喜欢 (/sign_in?utm_source=desktop&utm_medium=not-signed-in-like-button) | .





更多分享


被以下专题收入, 发现更多相似内容


(http://cwb.assets.jianshu.io/notes/9780085/weibo/image_e2e6c7265221.pg)




 Python语... (/c/9bc3ae683403?utm_source=desktop&utm_medium=notes-included-collection)

 @IT·互联网 (/c/V2CqjW?utm_source=desktop&utm_medium=notes-included-collection)

 机器学习与数据挖掘 (/c/9ca077f0fae8?utm_source=desktop&utm_medium=notes-included-collection)

 数据处理 (/c/cc7226c9d1dc?utm_source=desktop&utm_medium=notes-included-collection)

 程序员 (/c/NEt52a?utm_source=desktop&utm_medium=notes-included-collection)

