通过特征提取，我们能得到未经处理的特征，这时的特征可能有以下问题：

- 不属于同一量纲：即特征的规格不一样，不能够放在一起比较。无量纲化可以解决这一问题。
- 信息冗余：对于某些定量特征，其包含的有效信息为区间划分，例如学习成绩，假若只关心“及格”或不“及格”，那么需要将定量的考分，转换成“1”和“0”表示及格和未及格。二值化可以解决这一问题。
- 定性特征不能直接使用：某些机器学习算法和模型只能接受定量特征的输入，那么需要将定性特征转换为定量特征。最简单的方式是为每一种定性值指定一个定量值，但是这种方式过于灵活，增加了调参的工作。通常使用哑编码的方式将定性特征转换为定量特征**：假设有N种定性值，则将这一个特征扩展为N种特征，当原始特征值为第i种定性值时，第i个扩展特征赋值为1，其他扩展特征赋值为0。哑编码的方式相比直接指定的方式，不用增加调参的工作，对于线性模型来说，使用哑编码后的特征可达到非线性的效果。
- 存在缺失值：因为各种各样的原因，真实世界中的许多数据集都包含缺失数据，这类数据经常被编码成空格、NaNs，或其他占位符。
- 信息利用率低：不同的机器学习算法和模型对数据中信息的利用是不同的，之前提到在线性模型中，使用对定性特征哑编码可以达到非线性的效果。类似地，对定量变量多项式化，或者进行其他的转换，都能达到非线性的效果。

无量纲化

标准化

数据的标准化是将数据按比例缩放，使之落入一个小的特定区间。在某些比较和评价的指标处理中经常会用到，去除数据的单位限制，将其转化为无量纲的纯数值，便于不同单位或量级的指标能够进行比较和加权。 公式为：(X-mean)/std 计算时对每个属性/每列分别进行。 将数据按属性（按列进行）减去其均值，并除以其方差。得到结果是，对于每个属性（每列）来说所有数据都聚集在0附近，方差为1。

```
from sklearn.datasets import load_iris
import numpy as np

X = np.array([[ 1., -1.,  2.],
              [ 2.,  0.,  0.],
              [ 0.,  1., -1.]])
from sklearn import preprocessing
X_scaled = preprocessing.scale(X)
print(X_scaled)
print(X_scaled.mean(axis=0))
print(X_scaled.std(axis=0))
```

out

```
[[ 0.         -1.22474487  1.33630621]
 [ 1.22474487  0.         -0.26726124]
 [-1.22474487  1.22474487 -1.06904497]]
[ 0.  0.  0.]
[ 1.  1.  1.]
```

sklearn 还提供了StandardScaler类，使用该类的好处在于可以保存训练集中的参数（均值、方差）直接使用其对象转换测试集数据。

```
scaler = preprocessing.StandardScaler().fit(X)
print(scaler)

print(scaler.mean_)

print(scaler.scale_)

print(scaler.transform(X))
scaler.transform([[-1.,  1., 0.]])
```

out

```
StandardScaler(copy=True, with_mean=True, with_std=True)
[ 1.          0.          0.33333333]
[ 0.81649658  0.81649658  1.24721913]
[[ 0.         -1.22474487  1.33630621]
 [ 1.22474487  0.         -0.26726124]
 [-1.22474487  1.22474487 -1.06904497]]
Out[9]:
array([[-2.44948974,  1.22474487, -0.26726124]])
```

区间缩放

另一种常用的方法是将属性缩放到一个指定的最大和最小值（通常是1-0）之间，这可以通过preprocessing.MinMaxScaler类实现。

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。

$$x' = \frac{x - Min}{Max - Min}$$

```
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)
print(X_train_minmax)
```

out

```
[[ 0.5         0.          1.        ]
 [ 1.          0.5         0.33333333]
 [ 0.          1.          0.        ]]
```

归一化

归一化是依照特征矩阵的行处理数据，其目的在于样本向量在点乘运算或其他核函数计算相似性时，拥有统一的标准，也就是说都转化为“单位向量”。规则为l2的归一化公式如下：

$$x' = \frac{x}{\sqrt{\sum_j^m x[j]^2}}$$

该方法主要应用于文本分类和聚类中。例如，对于两个TF-IDF向量的l2-norm进行点积，就可以得到这两个向量的余弦相似性。

```
X_normalized = preprocessing.normalize(X_train, norm='l2')
print(X_normalized)
normalizer = preprocessing.Normalizer().fit(X_train)
normalizer.transform(X_train)
```

out

```
[[ 0.40824829 -0.40824829  0.81649658]
 [ 1.          0.          0.        ]
 [ 0.          0.70710678 -0.70710678]]
Out[16]:
array([[ 0.40824829, -0.40824829,  0.81649658],
       [ 1.        ,  0.        ,  0.        ],
       [ 0.        ,  0.70710678, -0.70710678]])
```

特征二值化

对于某些定量特征，其包含的有效信息为区间划分，例如学习成绩，假若只关心“及格”或不“及格”，那么需要将定量的考分，转换成“1”和“0”表示及格和未及格。定量特征二值化的核心在于设定一个阈值，大于阈值的赋值为1，小于等于阈值的赋值为0，公式表达如下：

$$x' = \begin{cases} 1, x > threshold \\ 0, x \leq threshold \end{cases}$$

使用preproccessing库的Binarizer类对数据进行二值化的代码如下：

```
from sklearn.preprocessing import Binarizer

X = [[ 1., -1.,  2.],
     [ 2.,  0.,  0.],
     [ 0.,  1., -1.]]
binarizer = Binarizer().fit(X)
print(binarizer.transform(X))
```

output

```
[[ 1.  0.  1.]
 [ 1.  0.  0.]
 [ 0.  1.  0.]]
```

分类特征编码

特征更多的时候是分类特征，而不是连续的数值特征。 比如一个人的特征可以是`["male", "female"]`， `["from Europe", "from US", "from Asia"]`， `["uses Firefox", "uses Chrome", "uses Safari", "uses Internet Explorer"]`。 这样的特征可以高效的编码成整数，例如 `["male", "from US", "uses Internet Explorer"]`可以表示成 `[0, 1, 3]`， `["female", "from Asia", "uses Chrome"]`就是`[1, 2, 1]`。 这个的整数特征表示并不能在scikit-learn的估计器中直接使用，因为这样的连续输入，估计器会认为类别之间是有序的，但实际却是无序的。(例如：浏览器的类别数据则是任意排序的)。 一个将分类特征转换成scikit-learn估计器可用特征的可选方法是使用one-of-K或者one-hot编码,`OneHotEncoder`是该方法的一个实现。该方法将每个类别特征的 `m` 可能值转换成`m`个二进制特征值，当然只有一个是激活值。例如：

```
from sklearn.preprocessing import OneHotEncoder
print(OneHotEncoder().fit_transform([[0, 0, 3], [1, 1, 0], [0, 2, 1], [1, 0,
2]]).toarray())
```

output

```
[[ 1.  0.  1.  0.  0.  0.  0.  0.  1.]
 [ 0.  1.  0.  1.  0.  1.  0.  0.  0.]
 [ 1.  0.  0.  0.  1.  0.  1.  0.  0.]
 [ 0.  1.  1.  0.  0.  0.  0.  1.  0.]]
```

缺失值处理

因为各种各样的原因，真实世界中的许多数据集都包含缺失数据，这类数据经常被编码成空格、NaNs，或者是其他的占位符。但是这样的数据集并不能`scikit-learn`学习算法兼容，因为大多的学习算法都默认假设数组中的元素都是数值，因而所有的元素都有自己的意义。 使用不完整的数据集的一个基本策略就是舍弃掉整行或整列包含缺失值的数据。但是这样就付出了舍弃可能有价值数据（即使是不完整的）的代价。 处理缺失数值的一个更好的策略就是从已有的数据推断出缺失的数值。 `Imputer`类提供了缺失数值处理的基本策略，比如使用缺失数值所在行或列的均值、中位数、众数来替代缺失值。该类也兼容不同的缺失值编码。 接下来是一个如何替换缺失值的简单示例，缺失值被编码为`np.nan`,使用包含缺失值的列的均值来替换缺失值。

```
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))
```

output

```
[[ 4.          2.        ]
 [ 6.          3.66666667]
 [ 7.          6.        ]]
```

数据变换

很多情况下，考虑输入数据中的非线性特征来增加模型的复杂性是非常有效的。常见的数据变换有基于多项式的、基于指数函数的、基于对数函数的。 使用`preproccessing`库的`PolynomialFeatures`类对数据进行多项式转换的代码如下：

```
import numpy as np
from sklearn.preprocessing import PolynomialFeatures
X = np.arange(6).reshape(3, 2)
PolynomialFeatures(2).fit_transform(X)
```

output

```
array([[  1.,   0.,   1.,   0.,   0.,   1.],
       [  1.,   2.,   3.,   4.,   6.,   9.],
       [  1.,   4.,   5.,  16.,  20.,  25.]])
```

基本的数据预处理就包含以上的方法。

文中涉及源码在这里：源码

参考

sklearn preprocess 特征工程到底是什么？

关于使用sklearn进行数据预处理 —— 归一化/标准化/正则化

统计数据归一化与标准化

标准化和归一化什么区别？

特征工程到底是什么？

sklearn preprocess