菜菜的scikit-learn课堂03

sklearn中的数据预处理和特征工程

小伙伴们晚上好~o(￣▽￣)ブ

我是菜菜，这里是我的sklearn课堂第三期，今晚的直播内容是数据预处理和特征工程~

我的开发环境是**Jupyter lab**，所用的库和版本大家参考：

Python 3.7.1（你的版本至少要3.4以上

Scikit-learn 0.20.0 （你的版本至少要0.19

Numpy 1.15.3, **Pandas** 0.23.4, **Matplotlib** 3.0.1, **SciPy** 1.1.0

请扫码进群领取课件和代码源文件，**扫描二维码后回复"K"就可以进群哦~**

菜菜的scikit-learn课堂03

sklearn中的数据预处理和特征工程

1 概述

1.1 数据预处理与特征工程

想象一下未来美好的一天，你学完了菜菜的课程，成为一个精通各种算法和调参调库的数据挖掘工程师了。某一天你从你的同事，一位药物研究人员那里，得到了一份病人临床表现的数据。药物研究人员用前四列数据预测一下最后一数据，还说他要出差几天，可能没办法和你一起研究数据了，希望出差回来以后，可以有个初步分析结果。于是你就看了看数据，看着很普通，预测连续型变量，好说，导随机森林回归器调出来，调参调呀调，MSE很小，跑了个还不错的结果。

012	232	33.5	0	10.7
020	121	16.9	2	210.1
027	165	24.0	0	427.6

几天后，你同事出差回来了，准备要一起开会了，会上你碰见了和你同事在同一个项目里工作的统计学家。他问起你的分析结果，你说你已经小有成效了，统计学家很吃惊，他说：“不错呀，这组数据问题太多，我都分析不出什么来。”

你心里可能咯噔一下，忐忑地回答说：“我没听说数据有什么问题呀。”

于是统计学家说：“诶？没人告诉你说，最后一列数据如果取个对数，结果会更好吗？”

你内心毫无波动：“没。”

统计学家：“诶那你肯定听说了第四列数据有点问题吧，这个特征的取值范围是1~10，0是表示缺失值的。而且他们输入数据的时候出错，很多10都被录入成0了，现在分不出来了。”

你：”……“

统计学家：”还有第二列和第三列数据基本是一样的，相关性太强了。“

你：”这个我发现了，不过这两个特征在预测中的重要性都不高，无论其他特征怎样出错，我这边结果里显示第一列的特征是最重要的，所以也无所谓啦。“

统计学家：“啥？第一列不就是编号吗？”

你：“不是吧。”

统计学家：“哦我想起来了！第一列就是编号，不过那个编号是我们根据第五列排序之后编上去的！这个第一列和第五列是由很强的联系，但是毫无意义啊！”

老血喷了一屏幕，数据挖掘工程师卒。

这个悲惨又可爱的故事来自《数据挖掘导论》，虽然这是故事里的状况十分极端，但我还是想把这段对话作为今天这章的开头，博大家一笑（虽然可能听完就泪流满面了）。在过去两周，我们已经讲了两个算法：决策树和随机森林，我们通过决策树带大家认识了sklearn，通过随机森林讲解了机器学习中调参的基本思想，现在可以说，只要上过前面两堂课的，人人都会调随机森林和决策树的分类器了，而我呢，也只需要跟着各大机器学习书籍的步伐，给大家一周一个算法带着讲解就是了。如果这样的话，结果可能就是，大家去工作了，遇到了一个不那么靠谱的同事，给了你一组有坑的数据，最后你就一屏幕老血吐过去，牺牲在数据行业的前线了。

数据不给力，再高级的算法都没有用。

我们在课堂中给大家提供的数据，都是经过层层筛选，适用于课堂教学的——运行时间短，预测效果好，没有严重缺失等等问题。尤其是sklearn中的数据，堪称完美。各大机器学习教材也是如此，都给大家提供处理好的数据，这就导致，很多人在学了很多算法之后，到了现实应用之中，发现模型经常就调不动了，因为现实中的数据，离平时上课使用的完美数据集，相差十万八千里。所以我决定，少讲一两个简单的算法，为大家专门拿一堂课来讲解建模之前的流程，**数据预处理和特征工程。**这样大家即可以学到数据挖掘过程中很重要但是却经常被忽视的一些步骤，也可以不受课堂的限制，如果自己有时间，可以尝试在真实数据上建模。

数据挖掘的五大流程：

1. 获取数据

2. **数据预处理**
数据预处理是从数据中检测，纠正或删除损坏，不准确或不适用于模型的记录的过程
可能面对的问题有：数据类型不同，比如有的是文字，有的是数字，有的含时间序列，有的连续，有的间断。也可能，数据的质量不行，有噪声，有异常，有缺失，数据出错，量纲不一，有重复，数据是偏态，数据量太大或太小
数据预处理的目的：让数据适应模型，匹配模型的需求

3. **特征工程**：
特征工程是将原始数据转换为更能代表预测模型的潜在问题的特征的过程，可以通过挑选最相关的特征，提取特征以及创造特征来实现。其中创造特征又经常以降维算法的方式实现。
可能面对的问题有：特征之间有相关性，特征和标签无关，特征太多或太小，或者干脆就无法表现出应有的数据现象或无法展示数据的真实面貌
特征工程的目的：1) 降低计算成本，2) 提升模型上限

4. 建模，测试模型并预测出结果

5. 上线，验证模型效果

1.2 sklearn中的数据预处理和特征工程

sklearn中包含众多数据预处理和特征工程相关的模块，虽然刚接触sklearn时，大家都会为其中包含的各种算法的广度深度所震惊，但其实sklearn六大板块中有两块都是关于数据预处理和特征工程的，两个板块互相交互，为建模之前的全部工程打下基础。

Classification

Identifying to which category an object belongs to.

Applications: Spam detection, Image recognition.
Algorithms: SVM, nearest neighbors, random forest, … — Examples

Regression

Predicting a continuous-valued attribute associated with an object.

Applications: Drug response, Stock prices.
Algorithms: SVR, ridge regression, Lasso, … — Examples

Clustering

Automatic grouping of similar objects into sets.

Applications: Customer segmentation, Grouping experiment outcomes
Algorithms: k-Means, spectral clustering, mean-shift, … — Examples

Dimensionality reduction

Reducing the number of random variables to consider.

Applications: Visualization, Increased efficiency
Algorithms: PCA, feature selection, non-negative matrix factorization. — Examples

Model selection

Comparing, validating and choosing parameters and models.

Goal: Improved accuracy via parameter tuning
Modules: grid search, cross validation, metrics. — Examples

Preprocessing

Feature extraction and normalization.

Application: Transforming input data such as text for use with machine learning algorithms.
Modules: preprocessing, feature extraction. — Examples

- 模块preprocessing：几乎包含数据预处理的所有内容
- 模块Impute：填补缺失值专用
- 模块feature_selection：包含特征选择的各种方法的实践
- 模块decomposition：包含降维算法

2 数据预处理 Preprocessing & Impute

2.1 数据无量纲化

在机器学习算法实践中，我们往往有着将不同规格的数据转换到同一规格，或不同分布的数据转换到某个特定分布的需求，这种需求统称为将数据“无量纲化”。譬如梯度和矩阵为核心的算法中，譬如逻辑回归，支持向量机，神经网络，无量纲化可以加快求解速度；而在距离类模型，譬如K近邻，K-Means聚类中，无量纲化可以帮我们提升模型精度，避免某一个取值范围特别大的特征对距离计算造成影响。（一个特例是决策树和树的集成算法们，对决策树我们不需要无量纲化，决策树可以把任意数据都处理得很好。）

数据的无量纲化可以是线性的，也可以是非线性的。线性的无量纲化包括**中心化**（Zero-centered或者Mean-subtraction）处理和**缩放处理**（Scale）。中心化的本质是让所有记录减去一个固定值，即让数据样本数据平移到某个位置。缩放的本质是通过除以一个固定值，将数据固定在某个范围之中，取对数也算是一种缩放处理。

- **preprocessing.MinMaxScaler**

当数据(x)按照最小值中心化后，再按极差（最大值 - 最小值）缩放，数据移动了最小值个单位，并且会被收敛到[0,1]之间，而这个过程，就叫做**数据归一化**(Normalization，又称Min-Max Scaling)。注意，Normalization是归一化，不是正则化，真正的正则化是regularization，不是数据预处理的一种手段。归一化之后的数据服从正态分布，公式如下：

$$x^* = \frac{x - min(x)}{max(x) - min(x)}$$

在sklearn当中，我们使用**preprocessing.MinMaxScaler**来实现这个功能。MinMaxScaler有一个重要参数，feature_range，控制我们希望把数据压缩到的范围，默认是[0,1]。

```
from sklearn.preprocessing import MinMaxScaler

data = [[-1, 2], [-0.5, 6], [0, 10], [1, 18]]

#不太熟悉numpy的小伙伴，能够判断data的结构吗？
#如果换成表是什么样子？
import pandas as pd
pd.DataFrame(data)

#实现归一化
scaler = MinMaxScaler()                                 #实例化
scaler = scaler.fit(data)                               #fit，在这里本质是生成min(x)和max(x)
result = scaler.transform(data)                         #通过接口导出结果
result

result_ = scaler.fit_transform(data)                    #训练和导出结果一步达成

scaler.inverse_transform(result)                        #将归一化后的结果逆转

#使用MinMaxScaler的参数feature_range实现将数据归一化到[0,1]以外的范围中

data = [[-1, 2], [-0.5, 6], [0, 10], [1, 18]]
scaler = MinMaxScaler(feature_range=[5,10])             #依然实例化
```

```
result = scaler.fit_transform(data)                 #fit_transform一步导出结果
result

#当x中的特征数量非常多的时候，fit会报错并表示，数据量太大了我计算不了
#此时使用partial_fit作为训练接口
#scaler = scaler.partial_fit(data)
```

BONUS: 使用numpy来实现归一化

```
import numpy as np
X = np.array([[-1, 2], [-0.5, 6], [0, 10], [1, 18]])

#归一化
X_nor = (X - X.min(axis=0)) / (X.max(axis=0) - X.min(axis=0))
X_nor

#逆转归一化
X_returned = X_nor * (X.max(axis=0) - X.min(axis=0)) + X.min(axis=0)
X_returned
```

- **preprocessing.StandardScaler**

当数据(x)按均值(μ)中心化后，再按标准差(σ)缩放，数据就会服从为均值为0，方差为1的正态分布（即标准正态分布），而这个过程，就叫做**数据标准化**(Standardization，又称Z-score normalization)，公式如下：

$$x^* = \frac{x - \mu}{\sigma}$$

```
from sklearn.preprocessing import StandardScaler
data = [[-1, 2], [-0.5, 6], [0, 10], [1, 18]]

scaler = StandardScaler()                           #实例化
scaler.fit(data)                                    #fit，本质是生成均值和方差

scaler.mean_                                        #查看均值的属性mean_
scaler.var_                                         #查看方差的属性var_

x_std = scaler.transform(data)                      #通过接口导出结果

x_std.mean()                                        #导出的结果是一个数组，用mean()查看均值
x_std.std()                                         #用std()查看方差

scaler.fit_transform(data)                          #使用fit_transform(data)一步达成结果

scaler.inverse_transform(x_std)                     #使用inverse_transform逆转标准化
```

对于StandardScaler和MinMaxScaler来说，空值NaN会被当做是缺失值，在fit的时候忽略，在transform的时候保持缺失NaN的状态显示。并且，尽管去量纲化过程不是具体的算法，但在fit接口中，依然只允许导入至少二维数组，一维数组导入会报错。通常来说，我们输入的X会是我们的特征矩阵，现实案例中特征矩阵不太可能是一维所以不会存在这个问题。

- **StandardScaler和MinMaxScaler选哪个?**

看情况。大多数机器学习算法中，会选择StandardScaler来进行特征缩放，因为MinMaxScaler对异常值非常敏感。在PCA，聚类，逻辑回归，支持向量机，神经网络这些算法中，StandardScaler往往是最好的选择。

MinMaxScaler在不涉及距离度量、梯度、协方差计算以及数据需要被压缩到特定区间时使用广泛，比如数字图像处理中量化像素强度时，都会使用MinMaxScaler将数据压缩于[0,1]区间之中。

建议先试试看StandardScaler，效果不好换MinMaxScaler。

除了StandardScaler和MinMaxScaler之外，sklearn中也提供了各种其他缩放处理（中心化只需要一个pandas广播一下减去某个数就好了，因此sklearn不提供任何中心化功能）。比如，在希望压缩数据，却不影响数据的稀疏性时（不影响矩阵中取值为0的个数时），我们会使用MaxAbsScaler；在异常值多，噪声非常大时，我们可能会选用分位数来无量纲化，此时使用RobustScaler。更多详情请参考以下列表。

无量纲化	功能	中心化	缩放	详解
.StandardScaler	标准化	均值	方差	通过减掉均值并将数据缩放到单位方差来标准化特征，标准化完毕后的特征服从标准正态分布，即方差为1，均值为0
.MinMaxScaler	归一化	最小值	极差	通过最大值最小值将每个特征缩放到给定范围，默认[0,1]
.MaxAbsScaler	缩放	N/A	最大值	通过让每一个特征里的数据，除以该特征中绝对值最大的数值的绝对值，将数据压缩到[-1,1]之间，这种做法并没有中心化数据，因此不会破坏数据的稀疏性。数据的稀疏性是指，数据中包含0的比例，0越多，数据越稀疏。
.RobustScaler	无量纲化	中位数	四分位 数范围	使用可以处理异常值，对异常值不敏感的统计量来缩放数据。 这个缩放器删除中位数并根据百分位数范围（IRQ:Interquartile Range）缩放数据。IQR是第一分位数（25%）和第三分维数（75%）之间的范围。数据集的标准化是通过去除均值，缩放到单位方差来完成，但是异常值通常会对样本的均值和方差造成负面影响，当异常值很多噪音很大时，用中位数和四分位数范围通常会产生更好的效果。
.Normalizer	无量纲化	N/A	sklearn中未明确，依范数原理应当是: l1：样本向量的长度/样本中每个元素绝对值的和 l2：样本向量的长度/样本中每个元素的欧氏距离	将样本独立缩放到到单位范数。每个至少带一个非0值的样本都回被独立缩放，使得整个样本（整个向量）的的长度都为l1范数或l2范数。这个类可以处理密集数组(numpy arrays)或scipy中的稀疏矩阵（scipy.sparse），如果你希望避免复制/转换过程中的负担，请使用CSR格式的矩阵。 将输入的数据缩放到单位范数是文本分类或聚类中的常见操作。例如，两个l2正则化后的TF-IDF向量的点积是向量的余弦相似度，并且是信息检索社区常用的向量空间模型的基本相似性度量。 使用参数norm来确定要正则化的范数方向，可以选择"l1","l2"以及"max"三种选项，默认l2范数。 这个评估器的fit接口什么也不做，但在管道中使用依然是很有用的。
.Power Transformer	非线性 无量纲化	N/A	N/A	应用特征功率变换使数据更接近正态分布。 功率变换是一系列参数单调变换，用于使数据更像高斯。这对于建模与异方差性（非常数方差）或其他需要正态性的情况相关的问题非常有用。要求输入的数据严格为正，power_transform()通过最大似然估计来稳定方差和并确定最小化偏度的最佳参数。 默认情况下，标准化应用于转换后的数据。
.Quantile Transformer	非线性 无量纲化	N/A	N/A	使用百分位数转换特征，通过缩小边缘异常值和非异常值之间的距离来提供特征的非线性变换。可以使用参数output_distribution = "normal"来将数据映射到标准正态分布。
.KernelCenterer	中心化	均值	N/A	将核矩阵中心化。设K(x，z)是由phi(x)^T phi(z)定义的核，其中phi是将x映射到希尔伯特空间的函数。 KernelCenterer在不明确计算phi(x)的情况下让数据中心化为0均值。 它相当于使用 sklearn.preprocessing.StandardScaler(with_std = False)来将phi(x)中心化。

2.2 缺失值

机器学习和数据挖掘中所使用的数据，永远不可能是完美的。很多特征，对于分析和建模来说意义非凡，但对于实际收集数据的人却不是如此，因此数据挖掘之中，常常会有重要的字段缺失值很多，但又不能舍弃字段的情况。因此，数据预处理中非常重要的一项就是处理缺失值。

```
import pandas as pd
data = pd.read_csv(r"C:\work\learnbetter\micro-class\
                   week 3 Preprocessing\Narrativedata.csv",index_col=0)

data.head()
```

在这里，我们使用从泰坦尼克号提取出来的数据，这个数据有三个特征，一个数值型，两个字符型，标签也是字符型。从这里开始，我们就使用这个数据给大家作为例子，让大家慢慢熟悉sklearn中数据预处理的各种方式。

- **impute.SimpleImputer**

class `sklearn.impute.SimpleImputer` (*missing_values=nan*, *strategy='mean'*, *fill_value=None*, *verbose=0*, *copy=True*)

在讲解随机森林的案例时，我们用这个类和随机森林回归填补了缺失值，对比了不同的缺失值填补方式对数据的影响。这个类是专门用来填补缺失值的。它包括四个重要参数：

参数	含义&输入
missing_values	告诉SimpleImputer，数据中的缺失值长什么样，默认空值np.nan
strategy	我们填补缺失值的策略，默认均值。 输入“mean”使用均值填补（仅对数值型特征可用） 输入“median"用中值填补（仅对数值型特征可用） 输入"most_frequent”用众数填补（对数值型和字符型特征都可用） 输入“constant"表示请参考参数“fill_value"中的值（对数值型和字符型特征都可用）
fill_value	当参数startegy为”constant"的时候可用，可输入字符串或数字表示要填充的值，常用0
copy	默认为True，将创建特征矩阵的副本，反之则会将缺失值填补到原本的特征矩阵中去。

```
data.info()
#填补年龄

Age = data.loc[:,"Age"].values.reshape(-1,1)            #sklearn当中特征矩阵必须是二维
Age[:20]

from sklearn.impute import SimpleImputer
imp_mean = SimpleImputer()                              #实例化，默认均值填补
imp_median = SimpleImputer(strategy="median")           #用中位数填补
imp_0 = SimpleImputer(strategy="constant",fill_value=0) #用0填补

imp_mean = imp_mean.fit_transform(Age)                  #fit_transform一步完成调取结果
imp_median = imp_median.fit_transform(Age)
imp_0 = imp_0.fit_transform(Age)

imp_mean[:20]
imp_median[:20]
imp_0[:20]

#在这里我们使用中位数填补Age
data.loc[:,"Age"] = imp_median

data.info()

#使用众数填补Embarked
Embarked = data.loc[:,"Embarked"].values.reshape(-1,1)
```

```
imp_mode = SimpleImputer(strategy = "most_frequent")
data.loc[:,"Embarked"] = imp_mode.fit_transform(Embarked)

data.info()
```

BONUS：用Pandas和Numpy进行填补其实更加简单

```
import pandas as pd
data = pd.read_csv(r"C:\work\learnbetter\micro-class\week 3
Preprocessing\Narrativedata.csv",index_col=0)

data.head()

data.loc[:,"Age"] = data.loc[:,"Age"].fillna(data.loc[:,"Age"].median())
#.fillna 在DataFrame里面直接进行填补

data.dropna(axis=0,inplace=True)
#.dropna(axis=0)删除所有有缺失值的行，.dropna(axis=1)删除所有有缺失值的列
#参数inplace，为True表示在原数据集上进行修改，为False表示生成一个复制对象，不修改原数据，默认False
```

2.3 处理分类型特征：编码与哑变量

在机器学习中，大多数算法，譬如逻辑回归，支持向量机SVM，k近邻算法等都只能够处理数值型数据，不能处理文字，在sklearn当中，除了专用来处理文字的算法，其他算法在fit的时候全部要求输入数组或矩阵，也不能够导入文字型数据（其实手写决策树和普斯贝叶斯可以处理文字，但是sklearn中规定必须导入数值型）。然而在现实中，许多标签和特征在数据收集完毕的时候，都不是以数字来表现的。比如说，学历的取值可以是["小学"，“初中”，“高中”，"大学"]，付费方式可能包含["支付宝"，“现金”，“微信”]等等。在这种情况下，为了让数据适应算法和库，我们必须将数据进行**编码**，即是说，**将文字型数据转换为数值型。**

- **preprocessing.LabelEncoder**：标签专用，能够将分类转换为分类数值

```
from sklearn.preprocessing import LabelEncoder

y = data.iloc[:,-1]                          #要输入的是标签，不是特征矩阵，所以允许一维

le = LabelEncoder()                          #实例化
le = le.fit(y)                               #导入数据
label = le.transform(y)                      #transform接口调取结果

le.classes_                                  #属性.classes_查看标签中究竟有多少类别
label                                        #查看获取的结果label

le.fit_transform(y)                          #也可以直接fit_transform一步到位

le.inverse_transform(label)                  #使用inverse_transform可以逆转
```

```
data.iloc[:,-1] = label                          #让标签等于我们运行出来的结果

data.head()

#如果不需要教学展示的话我会这么写：
from sklearn.preprocessing import LabelEncoder
data.iloc[:,-1] = LabelEncoder().fit_transform(data.iloc[:,-1])
```

- **preprocessing.OrdinalEncoder**：特征专用，能够将分类特征转换为分类数值

```
from sklearn.preprocessing import OrdinalEncoder

#接口categories_对应LabelEncoder的接口classes_，一模一样的功能
data_ = data.copy()

data_.head()

OrdinalEncoder().fit(data_.iloc[:,1:-1]).categories_

data_.iloc[:,1:-1] = OrdinalEncoder().fit_transform(data_.iloc[:,1:-1])

data_.head()
```

- **preprocessing.OneHotEncoder**：独热编码，创建哑变量

我们刚才已经用OrdinalEncoder把分类变量Sex和Embarked都转换成数字对应的类别了。在舱门Embarked这一列中，我们使用[0,1,2]代表了三个不同的舱门，然而这种转换是正确的吗？

我们来思考三种不同性质的分类数据：

1） 舱门（S，C，Q）

三种取值S，C，Q是相互独立的，彼此之间完全没有联系，表达的是S≠C≠Q的概念。这是名义变量。

2） 学历（小学，初中，高中）

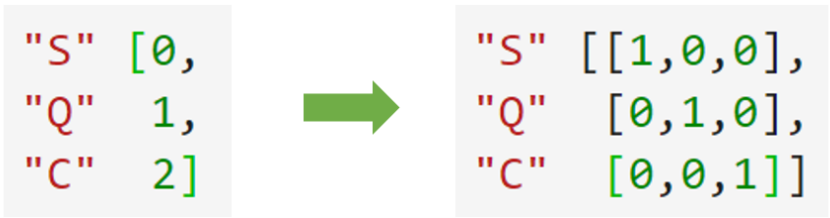
三种取值不是完全独立的，我们可以明显看出，在性质上可以有高中>初中>小学这样的联系，学历有高低，但是学历取值之间却不是可以计算的，我们不能说小学 + 某个取值 = 初中。这是有序变量。

3） 体重（>45kg，>90kg，>135kg）

各个取值之间有联系，且是可以互相计算的，比如120kg - 45kg = 90kg，分类之间可以通过数学计算互相转换。这是有距变量。

然而在对特征进行编码的时候，这三种分类数据都会被我们转换为[0,1,2]，这三个数字在算法看来，是连续且可以计算的，这三个数字相互不等，有大小，并且有着可以相加相乘的联系。所以算法会把舱门，学历这样的分类特征，都误会成是体重这样的分类特征。这是说，我们把分类转换成数字的时候，忽略了数字中自带的数学性质，所以给算法传达了一些不准确的信息，而这会影响我们的建模。

类别OrdinalEncoder可以用来处理有序变量，但对于名义变量，我们只有使用哑变量的方式来处理，才能够尽量向算法传达最准确的信息：



这样的变化，让算法能够彻底领悟，原来三个取值是没有可计算性质的，是“有你就没有我”的不等概念。在我们的数据中，性别和舱门，都是这样的名义变量。因此我们需要使用独热编码，将两个特征都转换为哑变量。

```
data.head()

from sklearn.preprocessing import OneHotEncoder
X = data.iloc[:,1:-1]

enc = OneHotEncoder(categories='auto').fit(X)
result = enc.transform(X).toarray()
result

#依然可以直接一步到位，但为了给大家展示模型属性，所以还是写成了三步
OneHotEncoder(categories='auto').fit_transform(X).toarray()

#依然可以还原
pd.DataFrame(enc.inverse_transform(result))

enc.get_feature_names()

result
result.shape

#axis=1,表示跨行进行合并，也就是将量表左右相连，如果是axis=0，就是将量表上下相连
newdata = pd.concat([data,pd.DataFrame(result)],axis=1)

newdata.head()

newdata.drop(["Sex","Embarked"],axis=1,inplace=True)

newdata.columns =
["Age","Survived","Female","Male","Embarked_C","Embarked_Q","Embarked_S"]

newdata.head()
```

特征可以做哑变量，标签也可以吗？可以，使用类sklearn.preprocessing.LabelBinarizer可以对做哑变量，许多算法都可以处理多标签问题（比如说决策树），但是这样的做法在现实中不常见，因此我们在这里就不赘述了。

编码与哑变量	功能	重要参数	重要属性	重要接口
.LabelEncoder	分类标签编码	N/A	.classes_：查看标签中究竟有多少类别	fit, transform, fit_transform, inverse_transform
.OrdinalEncoder	分类特征编码	N/A	.categories_：查看特征中究竟有多少类别	fit, transform, fit_transform, inverse_transform
.OneHotEncoder	独热编码，为名义变量创建哑变量	**categories**: 每个特征都有哪些类别，默认"auto"表示让算法自己判断，或者可以输入列表，每个元素都是一个列表，表示每个特征中的不同类别 **handle_unknown**：当输入了categories，且算法遇见了categories中没有写明的特征或类别时，是否报错。默认"error"表示请报错，也可以选择"ignore"表示请无视。如果选择"ignore"，则未再categories中注明的特征或类别的哑变量会全部显示为0。在逆转(inverse transform)中，未知特征或类别会被返回为None。	.categories_：查看特征中究竟有多少类别，如果是自己输入的类别，那就不需要查看了	fit, transform, fit_transform, inverse_transform, get_feature_names：查看生成的哑变量的每一列都是什么特征的什么取值

2.4 处理连续型特征：二值化与分段

- **sklearn.preprocessing.Binarizer**

根据阈值将数据二值化（将特征值设置为0或1），用于处理连续型变量。大于阈值的值映射为1，而小于或等于阈值的值映射为0。默认阈值为0时，特征中所有的正值都映射到1。二值化是对文本计数数据的常见操作，分析人员可以决定仅考虑某种现象的存在与否。它还可以用作考虑布尔随机变量的估计器的预处理步骤（例如，使用贝叶斯设置中的伯努利分布建模）。

```
#将年龄二值化

data_2 = data.copy()

from sklearn.preprocessing import Binarizer
X = data_2.iloc[:,0].values.reshape(-1,1)                    #类为特征专用，所以不能使用一维数组
transformer = Binarizer(threshold=30).fit_transform(X)

transformer
```

- **preprocessing.KBinsDiscretizer**

这是将连续型变量划分为分类变量的类，能够将连续型变量排序后按顺序分箱后编码。总共包含三个重要参数：

参数	含义&输入
n_bins	每个特征中分箱的个数，默认5，一次会被运用到所有导入的特征
encode	编码的方式，默认“onehot” "onehot"：做哑变量，之后返回一个稀疏矩阵，每一列是一个特征中的一个类别，含有该类别的样本表示为1，不含的表示为0 “ordinal”：每个特征的每个箱都被编码为一个整数，返回每一列是一个特征，每个特征下含有不同整数编码的箱的矩阵 "onehot-dense"：做哑变量，之后返回一个密集数组。
strategy	用来定义箱宽的方式，默认"quantile" "uniform"：表示等宽分箱，即每个特征中的每个箱的最大值之间的差为(特征.max() - 特征.min())/(n_bins) "quantile"：表示等位分箱，即每个特征中的每个箱内的样本数量都相同 "kmeans"：表示按聚类分箱，每个箱中的值到最近的一维k均值聚类的簇心得距离都相同

```
from sklearn.preprocessing import KBinsDiscretizer

X = data.iloc[:,0].values.reshape(-1,1)
est = KBinsDiscretizer(n_bins=3, encode='ordinal', strategy='uniform')
est.fit_transform(X)

#查看转换后分的箱：变成了一列中的三箱
set(est.fit_transform(X).ravel())

est = KBinsDiscretizer(n_bins=3, encode='onehot', strategy='uniform')
#查看转换后分的箱：变成了哑变量
est.fit_transform(X).toarray()
```

3 12周课程提纲

菜菜的sklearn课堂

日期	期数	主题	涉及的sklearn模块
11月7日	01期	决策树	tree
11月14日	02期	随机森林	ensemble
11月21日	03期	数据预处理和特征工程	preprocessing, impute, feature_selection
11月28日	04期	主成分分析	decomposition
12月5日	05期	逻辑回归	linear_model
12月12日	06期	K-Means	cluster
12月19日	07期	SVM（1）	svm
12月26日	08期	SVM（2）	svm
1月2日	09期	线性回归	linear_model
1月9日	10期	朴素贝叶斯	naive_bayes
1月16日	11期	sklearn中的数据产生	datasets
1月23日	12期	神经网络	neural_network

完整课程和课件大家请走：http://edu.cda.cn/course/982