

# 学习

用来记录在建立模型中，所需要的知识点

## pandas 知识点

### concat函数

该函数用于DataFrame对象的链接，下面举具体的例子

```
In [60]: # axis=0
import pandas as pd

df1 = pd.DataFrame({'name': ['小明', '小红'], 'color': ['蓝', '红']})
df2 = pd.DataFrame({'name': ['小黑', '小白'], 'color': ['黑', '白']})
df_concat = pd.concat([df1, df2], axis=0)

print('df1=')
print(df1)
print()
print('df2=')
print(df2)
print()
print('行合并结果为')
print('df_concat=')
print(df_concat)
```

```
df1=
  name color
0  小明  蓝
1  小红  红
```

```
df2=
  name color
0  小黑  黑
1  小白  白
```

```
行合并结果为
df_concat=
  name color
0  小明  蓝
1  小红  红
0  小黑  黑
1  小白  白
```

```
In [61]: # axis=1
import pandas as pd

df1 = pd.DataFrame({'name': ['小黑', '小白'], 'color': ['黑', '白']})
df2 = pd.DataFrame({'Age': [10, 10]})
df_concat = pd.concat([df1, df2], axis=1)

print('df1=')
```

```
print(df2)
print()
print('df2=')
print(df2)
print()
print('列合并结果为')
print('df_concat=')
print(df_concat)
df1=
  Age
0  10
1  10
```

```
df2=
  Age
0  10
1  10
```

```
列合并结果为
df_concat=
  name color Age
0  小黑   黑  10
1  小白   白  10
```

## copy函数

copy函数的参数deep=False时为浅复制，deep=True时为深复制。原来网上是说deep=False为默认值，但是个人实践发现，在python3.8版本下，deep=True为默认值。

```
In [62]: df = pd.DataFrame({'a':[1, 2], 'b':[3, 4]})
print('原来df=')
print(df)
print('然后用不同方式复制df....然后更改df.....')
df_True = df.copy(deep=True)
df_False = df.copy(deep=False)
df_Default = df.copy()
df['a'][0] = 10
print('更改后的df=')
print(df)
print('深复制得到df_True=')
print(df_True)
print('浅复制得到df_False=')
print(df_False)
print('默认复制(深复制)的得到的df_Default=')
print(df_Default)
```

```

原来df=
  a b
0 1 3
1 2 4
然后不同方式复制df....然后更改df.....
更改后的df=
  a b
0 10 3
1 2 4

```

## sklearn 知识点

### 转换器

#### 为什么要转换器

sklearn中的转换器是什么呢？假设目前这个场景，你想对测试集上输入部分做同训练集一样归一化的处理( $x' = \frac{x - \min}{\max - \min}$ )，那么一般会这样做。

```

In [63]: import numpy as np
          #自定义训练集和测试集
X_train = np.array([[1, 2], [3, 4]], dtype='float')
X_test = np.array([[2, 2], [3, 3]], dtype='float')
          #训练集每列的最大最小值
min0, max0 = min(X_train[:, 0]), max(X_train[:, 0])
min1, max1 = min(X_train[:, 1]), max(X_train[:, 1])
print('训练集X_train=')
print(X_train)
print('训练集X_train的第0列的最大值为{}, 最小值为{}'.format(max0, min0))
print('训练集X_train的第1列的最大值为{}, 最小值为{}'.format(max1, min1))
          #对训练集进行归一化处理
for i in range(X_train.shape[0]):
    for j in range(X_train.shape[1]):
        if j == 0:
            X_train[i][j] = (X_train[i][j] - min0) / (max0 - min0)
        else:
            X_train[i][j] = (X_train[i][j] - min1) / (max1 - min1)
print('归一化后的训练集为X_train=')
print(X_train)

```

```

训练集X_train=
[[1. 2.]
 [3. 4.]]
训练集X_train的第0列的最大值为3.0, 最小值为1.0
训练集X_train的第1列的最大值为4.0, 最小值为2.0
归一化后的训练集为X_train=
[[0. 0.]
 [1. 1.]]

```

```

In [64]: #再对测试集做相同的处理
          print('原测试集X_test=')
          print(X_test)
          for i in range(X_test.shape[0]):
              for j in range(X_test.shape[1]):
                  if j == 0:

```

```

        X_test[i][j] = (X_test[i][j] - min0) / (max0 - min0)
    else:
        print(X_test[i][j])
        X_test[i][j] = (X_test[i][j] - min1) / (max1 - min1)
print('归一化后的测试集X_test=')
print(X_test)
原测试集X_test=
[[2. 2.]
 [3. 3.]]
2.0
3.0
归一化后的测试集X_test=
[[0.5 0.]
 [1. 0.5]]

```

上面对测试集归一化非常的麻烦，并且我们还需要记录下训练集的最大最小值，所以sklearn中使用了转换器来简略操作。

```

In [65]: import numpy as np
from sklearn.preprocessing import MinMaxScaler
#自定义训练集和测试集
X_train = np.array([[1, 2], [3, 4]], dtype='float')
X_test = np.array([[2, 2], [3, 3]], dtype='float')
ms = MinMaxScaler()
ms.fit(X_train)
X_test = ms.transform(X_test)
print(X_test)

[[0.5 0.]
 [1. 0.5]]

```

在这里其中转换器的fit操作相当于记录训练集中每一列的极值并保存起来，transform函数就相当于用保存的极值对测试集进行归一化

一般来说，fit就是用来保存通过训练集得到的关键信息，transform就是用fit得到的关键信息来处理测试集，fit\_transform则是合并2个函数

### 自定义转换器

虽然sklearn提供转换器可以满足大多数问题的需要，但是总会遇到已有转换器无法解决的问题，所以我们需要自定义转换器。下面我们来制作和MinMaxScaler转换器一样功能的转换器

```

In [66]: import numpy as np
import pandas as pd
#导入自定义转换器所需要的基类
from sklearn.base import BaseEstimator, TransformerMixin
#定义自定义转换器，重写fit和transform函数
class MyMinMaxScaler(BaseEstimator, TransformerMixin):
    def fit(self, X, y=None):
        self.min = np.min(X, axis=0)
        self.max = np.max(X, axis=0)
        return self
    def transform(self, X, y=None):
        X = X.copy()
        return (X-self.min) / (self.max-self.min)

```

```

X_train = np.array([[1, 2], [3, 4]], dtype='float')
X_test = np.array([[2, 2], [3, 3]], dtype='float')

ms = MyMinMaxScaler()
ms.fit(X_train)
X_test = ms.transform(X_test)
print(X_test)
[[0.5 0.]
 [1. 0.5]]

```

## 估计器

估计器是sklearn提供的应一个强大的类，它封装了某个模型，比如决策树模型、贝叶斯模型，使用者可以创建估计器对象创建模型，调用fit方法训练模型，调用predict或者predict\_proba来预测结果。例如：

```

import sklearn.tree as tree
#创建模型
clf = tree.DecisionTreeClassifier()
#训练模型
clf.fit(X_train)
#预测结果
clf.predict(X_test)

```

## 管道

### 连接n个转换器

注意:连接n个转换器后得到的管道视为一个转换器举个例子假设目前我们有两个转换器，我们会这样使用

```

In [67]: import numpy as np
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import MinMaxScaler
from sklearn.pipeline import Pipeline

# 自定义训练集、测试集
X_train = np.array([[1, np.nan, 3], [np.nan, 2, 3], [3, 5, np.nan]])
X_test = np.array([[2, np.nan, 3], [np.nan, 2, 3], [3, 4, np.nan]])

#定义转换器
imputer = SimpleImputer(strategy='most_frequent')
scaler = MinMaxScaler()

#通过训练集获取转换器关键参数
X_train_1 = imputer.fit_transform(X_train)
scaler.fit(X_train_1)

#对测试集做相同处理
X_test_1 = imputer.transform(X_test)
X_test_2 = scaler.transform(X_test_1)
print(X_test_2)

```

```
[[0.5  0.  0.  ]
 [0.  0.  0.  ]
 [1.  0.66666667 0.  ]]
```

同样这里比较麻烦，可以用Pipeline对象将多个转换器连接起来，起到一个转换器的效果

```
In [68]: # 自定义训练集、测试集
X_train = np.array([[1, np.nan, 3],[np.nan, 2, 3],[3, 5, np.nan]])
X_test = np.array([[2, np.nan, 3],[np.nan, 2, 3],[3, 4, np.nan]])

pipeline = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='most_frequent')),
    ('scaler', MinMaxScaler())
])

pipeline.fit(X_train)
X_test_ = pipeline.transform(X_test)
print(X_test_)

[[0.5  0.  0.  ]
 [0.  0.  0.  ]
 [1.  0.66666667 0.  ]]
```

连接n个转换器和1个评估器

注意:连接n个转换器和1个评估器得到的管道视为一个评估器

调用fit函数会依次调用转换器的fit\_transform函数，最后在调用估计器的fit函数进行训练调用predict或者predict\_proba函数则是会依次调用转换器的transform函数对测试集做和训练集相同的处理，最后在调用估计器的predict或predict\_proba函数进行预测