分类中解决类别不平衡问题

前面我们已经初步认识了，什么是类别不平衡问题。其实，在现实环境中，采集的数据（建模样本）往往是比例失衡的。比如网贷数据，逾期人数的比例是极低的（千分之几的比例）；奢侈品消费人群鉴定等。

1 类别不平衡数据集基本介绍

在这一节中，我们一起看一下，当遇到数据类别不平衡的时候，我们该如何处理。在Python中，有Imblearn包，它就是为处理数据比例失衡而生的。

- 安装Imblearn包

```
pip3 install imbalanced-learn
```

第三方包链接：https://pypi.org/project/imbalanced-learn/

- 创造数据集

```
from sklearn.datasets import make_classification
import matplotlib.pyplot as plt

#使用make_classification生成样本数据
X, y = make_classification(n_samples=5000,
                           n_features=2,  # 特征个数= n_informative () + n_redundant + n_repeated
                           n_informative=2,  # 多信息特征的个数
                           n_redundant=0,   # 冗余信息，informative特征的随机线性组合
                           n_repeated=0,  # 重复信息，随机提取n_informative和n_redundant 特征
                           n_classes=3,  # 分类类别
                           n_clusters_per_class=1,  # 某一个类别是由几个cluster构成的
                           weights=[0.01, 0.05, 0.94],  # 列表类型，权重比
                           random_state=0)
```

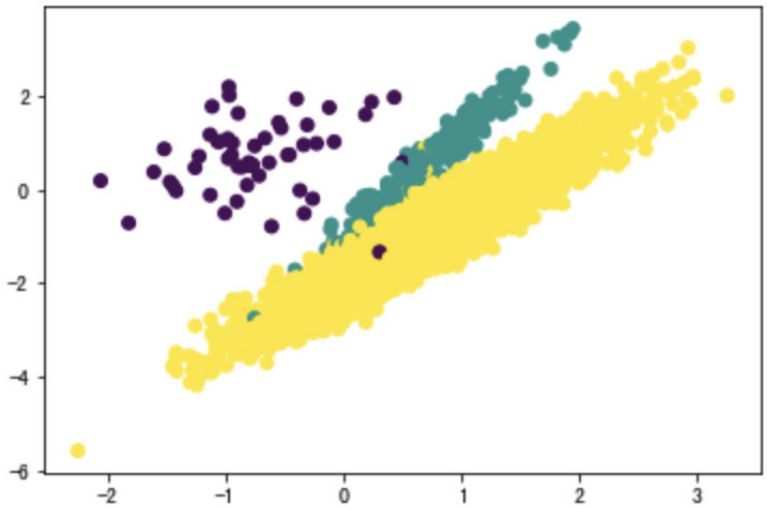
- 查看各个标签的样本

```
#查看各个标签的样本量
from collections import Counter
Counter(y)

# Counter({2: 4674, 1: 262, 0: 64})
```

- 数据集可视化

```
# 数据集可视化
plt.scatter(X[:, 0], X[:, 1], c=y)
plt.show()
```



可以看出样本的三个标签中，1，2的样本量极少，样本失衡。下面使用imblearn进行过采样。

接下来，我们就要基于以上数据，进行相应的处理。

关于类别不平衡的问题，主要有两种处理方式：

- 过采样方法
  - 增加数量较少那一类样本的数量，使得正负样本比例均衡。
- 欠采样方法
  - 减少数量较多那一类样本的数量，使得正负样本比例均衡。

2 解决类别不平衡数据方法介绍

2.1 过采样方法

2.1.1 什么是过采样方法

对训练集里的少数类进行“过采样”（oversampling），**即增加一些少数类样本使得正、反例数目接近，然后再进行学习。**

2.1.2 随机过采样方法

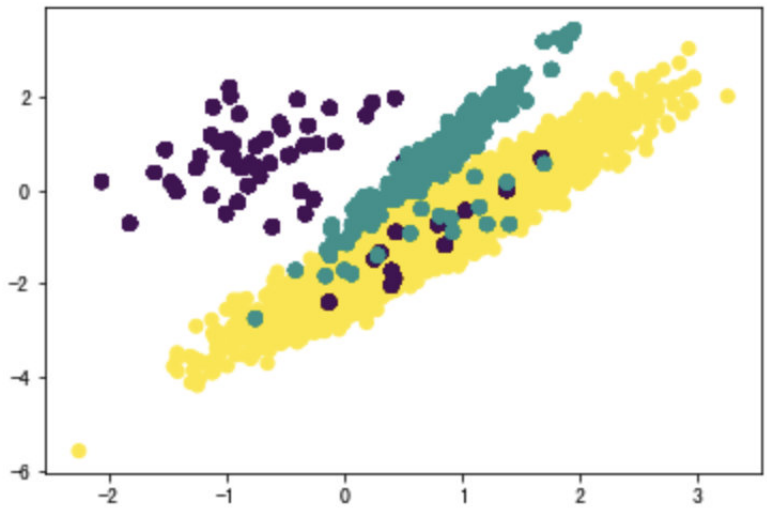
随机过采样是在少数类 S_{min} 中随机选择一些样本，然后**通过复制所选择的样本生成样本集 E**，将它们添加到 S_{min} 中来扩大原始数据集从而得到新的少数类集合 $S_{new-min}$ 。新的数据集 $S_{new-min} = S_{min} + E$ 。

- 通过代码实现随机过采样方法：

```
# 使用imblearn进行随机过采样
from imblearn.over_sampling import RandomOverSampler
ros = RandomOverSampler(random_state=0)
X_resampled, y_resampled = ros.fit_resample(X, y)
#查看结果
Counter(y_resampled)

#过采样后样本结果
# Counter({2: 4674, 1: 4674, 0: 4674})

# 数据集可视化
plt.scatter(X_resampled[:, 0], X_resampled[:, 1], c=y_resampled)
plt.show()
```



- 缺点：
  - 对于随机过采样，由于需要对少数类样本进行复制来扩大数据集，**造成模型训练复杂度加大。**
  - 另一方面也容易**造成模型的过拟合问题**，因为随机过采样是简单的对初始样本进行复制采样，这就使得学习器学得的规则过于具体化，不利于学习器的泛化性能，造成过拟合问题。

为了解决随机过采样中造成模型过拟合问题，又能保证实现数据集均衡的目的，出现了过采样法代表性的算法SMOTE算法。

2.1.3 过采样代表性算法-SMOTE

SMOTE全称是Synthetic Minority Oversampling即合成少数类过采样技术。

SMOTE算法是对随机过采样方法的一个改进算法，由于随机过采样方法是直接对少数类进行重采用，会使训练集中有很多重复的样本，容易造成产生的模型过拟合问题。而SMOTE算法的基本思想:

> 对每个少数类样本 $\boldsymbol{x_i}$ ，从它的最近邻中随机选择一个样本 $\hat{\boldsymbol{x_i}}$ （ $\hat{\boldsymbol{x_i}}$ 是少数类中的一个样本），然后在 $\boldsymbol{x_i}$ 和 $\hat{\boldsymbol{x_i}}$ 之间的连线上随机选择一点作为新合成的少数类样本。

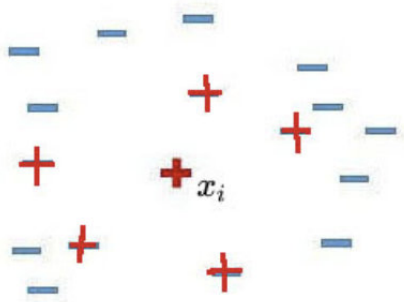
SMOTE算法合成新少数类样本的算法描述如下：

- 1) 对于少数类中的每一个样本 $\boldsymbol{x_i}$ ，以欧氏距离为标准计算它到少数类样本集 $\boldsymbol{S_{min}}$ 中所有样本的距离，得到其k近邻。
- 2) 根据样本不平衡比例设置一个采样比例以确定采样倍率N，对于每一个少数类样本 $\boldsymbol{x_i}$ ，从其k近邻中随机选择若干个样本，假设选择的是 $\hat{\boldsymbol{x_i}}$ 。
- 3) 对于每一个随机选出来的近邻 $\hat{\boldsymbol{x_i}}$ ，分别与 $\boldsymbol{x_i}$ 按照如下公式构建新的样本。

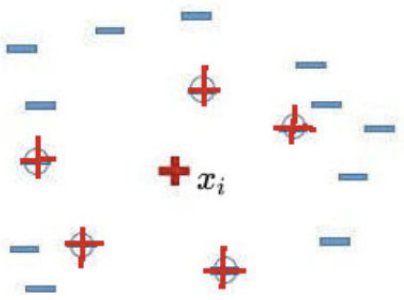
$$x_{new} = x_i + rand(0,1) \times (\hat{x_i} - x_i)$$

我们用图文表达的方式，再来描述一下SMOTE算法。

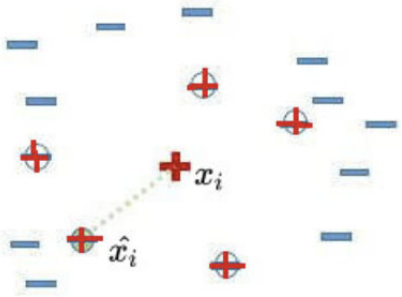
1) 先随机选定一个少数类样本 $\boldsymbol{x_i}$ 。



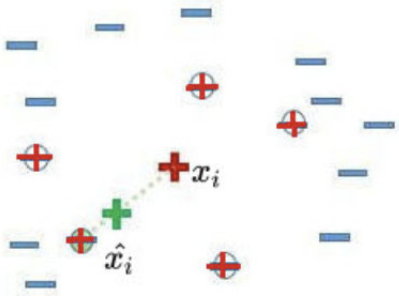
2) 找出这个少数类样本 $\boldsymbol{x_i}$ 的K个近邻（假设K=5），5个近邻已经被圈出。



3) 随机从这K个近邻中选出一个样本 $\hat{\boldsymbol{x_i}}$ （用绿色圈出来了）。



4)在少数类样本 x_i 和被选中的这个近邻样本 $\hat{x}_i$ 之间的连线上，随机找一点。这个点就是人工合成的新的样本点（绿色正号标出）。



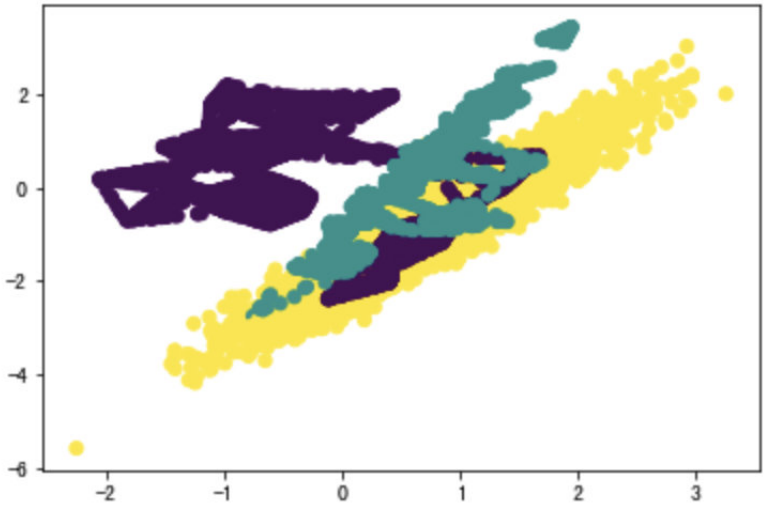
SMOTE算法摒弃了随机过采样复制样本的做法，可以防止随机过采样中容易过拟合的问题，实践证明此方法可以提高分类器的性能。

- 代码实现：

```
# SMOTE过采样
from imblearn.over_sampling import SMOTE
X_resampled, y_resampled = SMOTE().fit_resample(X, y)
Counter(y_resampled)

# 采样后样本结果
# [(0, 4674), (1, 4674), (2, 4674)]

# 数据集可视化
plt.scatter(X_resampled[:, 0], X_resampled[:, 1], c=y_resampled)
plt.show()
```



2.2 欠采样方法

2.2.1 什么是欠采样方法

直接对训练集中多数类样本进行“欠采样”（undersampling），即去**除一些多数类中的样本使得正例、反例数目接近，然后再进行学习。**

2.2.2 随机欠采样方法

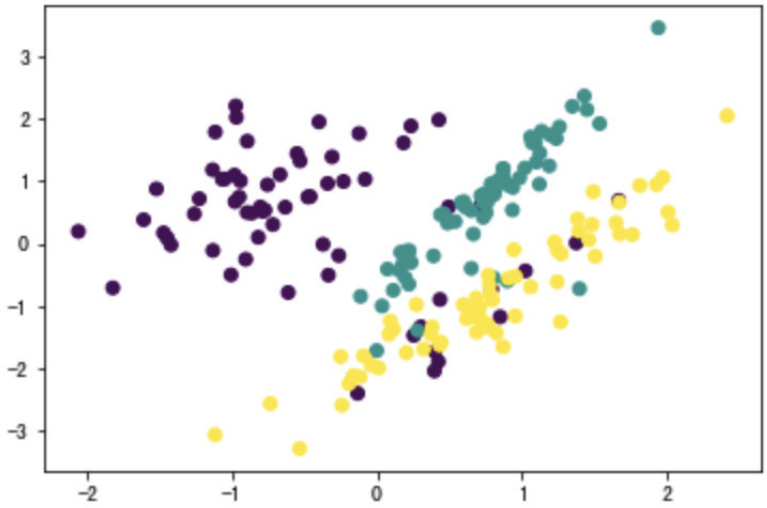
随机欠采样顾名思义即从多数类 S_{maj} 中随机选择一些样样本组成样本集 E 。然后将样本集 E 从 S_{maj} 中移除。新的数据集 $S_{new-maj} = S_{maj} - E$ 。

- 代码实现：

```
# 随机欠采样
from imblearn.under_sampling import RandomUnderSampler
rus = RandomUnderSampler(random_state=0)
X_resampled, y_resampled = rus.fit_resample(X, y)
Counter(y_resampled)

# 采样后结果
[(0, 64), (1, 64), (2, 64)]

# 数据集可视化
plt.scatter(X_resampled[:, 0], X_resampled[:, 1], c=y_resampled)
plt.show()
```



- 缺点：
  - 随机欠采样方法通过改变多数类样本比例以达到修改样本分布的目的，从而使样本分布较为均衡，但是这也存在一些问题。对于随机欠采样，**由于采样的样本集合要少于原来的样本集合，因此会造成一些信息缺失，即将多数类样本删除有可能会导致分类器丢失有关多数类的重要信息。**

官网链接：https://imbalanced-learn.readthedocs.io/en/stable/ensemble.html