内容简介

关于决策数和信息增益的内容介绍，可以回顾初级课程案例 2 和案例 3 数据建模部分。决策树是具有特征提取与特征描述功能的有监督学习算法，在分类、预测、规则提取等领域有着广泛的应用，在具备了决策树的基本理论和代码实现能力之后，本节重点探讨两方面内容。

1. 主要内容

- 为控制决策树复杂度，通过网格搜索 DecisionTreeClassifier 中超参数 max_depth、min_samples_split 最优组合，从而实现决策树剪枝；
- 进一步探索不同数据分割与重构方法对决策树算法准确率的影响。

2. 学习目标

学完本节，能解决以下问题：

- 如何完成关于决策树超参数的网格搜索？
- 如何定义一个函数，来分别实现不同的数据分割与数据重构方法？

决策树剪枝

决策树剪枝是决策树防止过拟合的主要方法。为了尽可能提高准确率，决策树的结点划分过程会不断重复，有时会造成决策树分支过多，出现过拟合的情况。因此，修剪掉分支前后分类误差相差不大的子树，能够降低决策树的复杂度，降低过拟合的风险。

决策树剪枝的基本策略有预剪枝和后剪枝。

预剪枝（prepruning）是指在决策树生成过程中，对每个结点在划分前先进行估计，如果当前结点的划分不能带来决策树泛化性能提升，则停止划分并将当前结点标记为叶结点。

后剪枝（postpruning）则是先根据训练集生成一棵完整的决策树，然后自底向上地对非叶结点进行考察，如果该结点对应的子树替换为叶结点能带来决策树泛化性能提升，则将该子树替换为叶结点。

泛化性能的提升则可以通过基于验证集的准确率来判断。

在 python 的 scikit-learn 机器学习库中，决策树算法的参数有 9 个，可以通过调整这些参数对决策树进行剪枝，控制决策树的复杂度。

其中，最常修改的参数是 max_depth 和 min_samples_split。

max_depth 表示决策树的最大层数；min_samples_split 表示划分当前结点所能够使用的最小样本量。我们可以通过网格搜索法来寻找合适的 max_depth 和 min_samples_split。

决策树算法总结

1、如何完成关于决策树超参数的网格搜索？

本次探讨的决策树超参数网格搜索是针对决策树剪枝而进行的，决策树剪枝是决策树防止过拟合的主要方法。

而决策树剪枝的基本策略有预剪枝和后剪枝：预剪枝（prepruning）是指在决策树生成过程中，对每个结点在划分前先进行估计；后剪枝（postpruning）则是先根据训练集生成一棵完整的决策树，然后自底向上地对非叶结点进行考察。

在 python 的 scikit-learn 机器学习库中，决策树算法可以通过调整 max_depth（最大层数）和 min_samples_split（当前结点的最小样本量）两个参数对决策树进行剪枝。具体的代码实现过程如下所示：

```
# 网格搜索超参数
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import cross_val_score

for max_depth in [30,40,50]:
    for min_samples_split in [2,3,4]:
        # 创建决策树
        dt = DecisionTreeClassifier(max_depth=max_depth, min_samples_split=min_samples_split,
                                    random_state=0)
        # 交叉验证计算验证集的准确率
        scores = cross_val_score(dt, X_train_ros, y_train_ros, scoring='accuracy', cv=4)
        score_v = np.mean(scores)
        # 训练模型并计算测试集的准确率
        dt.fit(X_train_ros, y_train_ros)
        score_t = dt.score(X_test, y_test)
        print("max_depth=%d, min_samples_split=%d, 验证集准确率=%.4f, 测试集准确率=%.4f" %
              (max_depth, min_samples_split, score_v, score_t))
```

```
max_depth=30, min_samples_split=2, 验证集准确率=0.9316, 测试集准确率=0.7881
max_depth=30, min_samples_split=3, 验证集准确率=0.9328, 测试集准确率=0.7712
max_depth=30, min_samples_split=4, 验证集准确率=0.9328, 测试集准确率=0.7797
max_depth=40, min_samples_split=2, 验证集准确率=0.9365, 测试集准确率=0.7881
max_depth=40, min_samples_split=3, 验证集准确率=0.9361, 测试集准确率=0.7712
max_depth=40, min_samples_split=4, 验证集准确率=0.9377, 测试集准确率=0.7797
max_depth=50, min_samples_split=2, 验证集准确率=0.9365, 测试集准确率=0.7881
max_depth=50, min_samples_split=3, 验证集准确率=0.9361, 测试集准确率=0.7712
max_depth=50, min_samples_split=4, 验证集准确率=0.9377, 测试集准确率=0.7797
```

2、如何定义一个函数，来分别实现不同的数据分割与数据重构方法？

首先需要明确在不同的数据分割与数据重构中，需要改变是哪些参数。在本节中，涉及到数据分割 train_test_split()中的参数 stratify，以及四种不同的数据重构方法：RandomOverSampler、RandomUnderSampler、ADASYN、SMOTE。因此可以将

四种不同的数据重构方法共同命名为 sampler，取值为这四种不同数据重构方法。即函数为 dt_score(X, y, stratify=None, sampler=None)，(X,y)作为输入的数据集。

在函数内部数据分割 train_test_split()直接令 stratify 取值为 stratify，通过改变 dt_score()函数中 stratify 的取值来改变 train_test_split()中 stratify 的取值；而函数内部的数据重构，可以通过判断参数 sampler 是否具有取值，来决定是否需要数据重构，如果有取值，这用具体的取值方法 fit_resample()即可进行不同的数据重构。

dt_score 函数的定义代码如下所示：

```
# 决策树分类
def dt_score(X, y, stratify=None, sampler=None):

    # 创建决策树
    dt = DecisionTreeClassifier(max_depth=40, min_samples_split=2, random_state=0)
    # 数据分割
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.1, shuffle=True, random_state=0, stratify=stratify)

    if sampler!=None:
        # 数据重构
        X_train_bal, y_train_bal = sampler.fit_resample(X_train, y_train)
        # 训练模型
        dt.fit(X_train_bal, y_train_bal)
        # 计算平衡训练集和测试集的准确率
        score_train = dt.score(X_train_bal, y_train_bal)
        score_test = dt.score(X_test, y_test)
    else:
        # 训练模型
        dt.fit(X_train, y_train)
        # 计算训练集和测试集的准确率
        score_train = dt.score(X_train, y_train)
        score_test = dt.score(X_test, y_test)

    return score_train, score_test
```

最终 return 模型分别在训练集和测试集上的准确度：score_train、score_test

不同数据重构对模型的影响如下所示：

```
# 数据重构对模型的影响
print("原训练集准确率=%.4f,测试集准确率=%.4f" %
      (dt_score(X, y)[0], dt_score(X, y)[1]))
print("简单重采样准确率=%.4f,测试集准确率=%.4f" %
      (dt_score(X, y, sampler=ros)[0], dt_score(X, y, sampler=ros)[1]))
print("SMOTE重采样准确率=%.4f,测试集准确率=%.4f" %
      (dt_score(X, y, sampler=sm)[0], dt_score(X, y, sampler=sm)[1]))
print("ADASYN重采样准确率=%.4f,测试集准确率=%.4f" %
      (dt_score(X, y, sampler=ada)[0], dt_score(X, y, sampler=ada)[1]))
print("简单亚采样准确率=%.4f,测试集准确率=%.4f" %
      (dt_score(X, y, sampler=rus)[0], dt_score(X, y, sampler=rus)[1]))
```

```
原训练集准确率=0.9953,测试集准确率=0.7203
简单重采样准确率=1.0000,测试集准确率=0.7881
SMOTE重采样准确率=1.0000,测试集准确率=0.7458
ADASYN重采样准确率=1.0000,测试集准确率=0.8220
简单亚采样准确率=1.0000,测试集准确率=0.5000
```

选取 ADASYN 重采样下进行参数 stratify 是否具有取值探讨，如下所示：

```
# 数据分割方式对模型的影响
print("标签分割：训练集准确率=%.4f,测试集准确率=%.4f" %
      (dt_score(X, y, stratify=y, sampler=ada)[0], dt_score(X, y, stratify=y, sampler=ada)[1]))
print("随机分割：训练集准确率=%.4f,测试集准确率=%.4f" %
      (dt_score(X, y, sampler=ada)[0], dt_score(X, y, sampler=ada)[1]))
```

```
标签分割：训练集准确率=1.0000,测试集准确率=0.6864
随机分割：训练集准确率=1.0000,测试集准确率=0.8220
```

最终发现随机分割的模型效果更好一些。

内容简介

关于朴素贝叶斯模型的相关内容，上一个案例已经较为全面的了解过。本节中将继续使用高斯朴素贝叶斯模型对公司概述与公司所属行业字段进行分析，寻找最佳的分类模型。

1. 主要内容

- 关于 sklearn 中 SelectKBest 特征选择，新增卡方检验计算单变量的卡方值，从而筛选掉与类别变量不相关的特征；
- 网格搜索超参数，以及不同数据分割方法和数据重构方法对模型的影响。

2. 学习目标

学完本节，能解决以下问题：

- 卡方检验与方差分析的概念以及之间的区别是什么？
- 如何完成高斯朴素贝叶斯模型中超参数组合的网格搜索？
- 如何定义一个函数，来分别实现不同的数据分割与数据重构方法？

卡方检验

卡方检验是对于定类变量总体分布进行检验的一种非参数检验方法。它可以根据样本数据，推断其总体的分布与期望分布或某一理论分布是否存在显著差异，其原假设与备择假设如下：

H_0：样本来自的总体分布与期望分布或某一理论分布无显著差异。

H_1：样本来自的总体分布与期望分布或某一理论分布有显著差异。

在具体应用中，卡方检验可以检验两个变量之间有没有关系。

例如，现在有一组实验者是否抽烟和是否患肺癌的数据如下：

	肺癌人数	健康人数	合计	患癌率
抽烟组	40	60	100	40%
不抽烟组	30	70	100	30%
合计	70	130	200	35%

抽烟组和不抽烟组的患癌率分别是 40%和 30%，这其中的差异有可能是变量“是否抽烟”和“是否患癌”之间存在相关性，也有可能是误差导致的。为了研究两个变量之间是否真的存在相关性，设置原假设为变量之间不存在相关性，即抽烟组和不抽烟组的实际患癌率均是 35%。然后通过卡方检验的计算公式进行计算：

$$\chi^2 = \sum_{i=1}^{k} \frac{(f_0 - f_e)^2}{f_e}$$

其中：

f_0是根据样本计算出的实际次数

f_e是每个实际次数相应的预期次数

卡方值越大，说明样本来自的总体分布与期望分布差距较大，即两个变量相关性较强；卡方值越小，说明两个变量相关性较弱。

在之前的案例中我们提到过，朴素贝叶斯模型不具有特征选择的功能，需要通过类 SelectKBest 确定最佳的特征数 K。因为朴素贝叶斯为分类模型，可选的计分函数包括 f_classif，chi2 和 mutual_info_classif 三种。其中，后两者适用于输入特征非负的条件，且最后一项计算复杂度相对较高，所以案例中采用 chi2 函数。

这一函数统计每一个特征和类别之间的卡方值，取卡方值最高的 K 个特征，从而筛选掉与类别变量不相关的特征。

朴素贝叶斯算法总结

1、卡方检验与方差分析的概念以及之间的区别是什么？

卡方检验是对于定类变量总体分布进行检验的一种非参数检验方法；而方差分析是从总体上判断多组数据的均值μ之间的差异是否显著。

从计算方法上可以发现，卡方检验是通过计算总体中不同取值的实际次数与预期次数差异得到的卡方值，计算公式如下：

$$\chi^2 = \sum_{i=1}^{k} \frac{(f_0 - f_e)^2}{f_e}$$

而方差分析是通过计算组间方差与组内方差的比值来比较均值之间差异的显著性。

其中卡方检验的预期次数一般是通过假设不存在相关性而得到，然后通过公式看是否推翻原假设。

从特征选择 SelectKBest 代码实现上，都是参数 score_func 的不同取值来决定的，score_func=f_classif 表示使用方差分析作为计分函数，而卡方检验是取值为 chi2。

除此以外，卡方检验适用于输入特征非负的条件，而方差分析则没有这个限制。

2、如何完成高斯朴素贝叶斯模型中超参数组合的网格搜索？

本关于高斯朴素贝叶斯模型的超参数网格搜索不做过多总结，可参照中级课程案例三相关内容，这里交叉验证不使用 GridSearchCV，而使用 cross_val_score，例如案例中的代码：

```
# 网格搜索超参数
from sklearn.naive_bayes import GaussianNB
from sklearn.feature_selection import SelectKBest, chi2

for k in [6000,7000,8000]:
    for priors in [(0.20,0.58,0.14,0.08), (0.25,0.25,0.25,0.25)]:
        # 特征选择
        selector = SelectKBest(score_func=chi2, k=k)
        X_train_ros_new = selector.fit_transform(X_train_ros, y_train_ros)
        X_test_new = selector.transform(X_test)

        # 使用高斯朴素贝叶斯模型建模
        gnb = GaussianNB(priors=priors)
        # 交叉验证计算验证集的准确率
        scores = cross_val_score(gnb, X_train_ros_new, y_train_ros, scoring='accuracy', cv=4)
        score_v = np.mean(scores)
        # 训练模型并计算测试集的准确率
        gnb.fit(X_train_ros_new, y_train_ros)
        score_t = gnb.score(X_test_new, y_test)

        print("k=%d, priors=%s, 验证集准确率=%.4f, 测试集准确率=%.4f" %
              (k, priors, score_v, score_t))
```

```
k=6000, priors=(0.2, 0.58, 0.14, 0.08), 验证集准确率=0.9654, 测试集准确率=0.8220
k=6000, priors=(0.25, 0.25, 0.25, 0.25), 验证集准确率=0.9654, 测试集准确率=0.8220
k=7000, priors=(0.2, 0.58, 0.14, 0.08), 验证集准确率=0.9609, 测试集准确率=0.8305
k=7000, priors=(0.25, 0.25, 0.25, 0.25), 验证集准确率=0.9609, 测试集准确率=0.8305
k=8000, priors=(0.2, 0.58, 0.14, 0.08), 验证集准确率=0.9630, 测试集准确率=0.8220
k=8000, priors=(0.25, 0.25, 0.25, 0.25), 验证集准确率=0.9630, 测试集准确率=0.8220
```

最终我们选择 k=7000,priors=(0.2,0.58,0.14,0.08)作为最优超参数组合。

3、如何定义一个函数，来分别实现不同的数据分割与数据重构方法？

这里与决策树模型中不同数据分割与重构对模型的影响内容基本一致，定义函数gnb_score()的过程如下：

```
# 朴素贝叶斯分类
def gnb_score(X, y, stratify=None, sampler=None):

    # 最优超参数构建模型
    selector = SelectKBest(score_func=chi2, k=7000)
    gnb = GaussianNB(priors=(0.20,0.58,0.14,0.08))
    # 数据分割
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.1, shuffle=True, random_state=0, stratify=stratify)

    if sampler!=None:
        # 数据重构
        X_train_bal, y_train_bal = sampler.fit_resample(X_train, y_train)
        # 特征选择
        selector.fit(X_train_ros, y_train_ros)
        X_train_bal_new = selector.transform(X_train_bal)
        X_test_new = selector.transform(X_test)
        # 训练朴素贝叶斯模型
        gnb.fit(X_train_bal_new, y_train_bal)
        # 计算特征选择后的平衡训练集和测试集的准确率
        score_train = gnb.score(X_train_bal_new, y_train_bal)
        score_test = gnb.score(X_test_new, y_test)
    else:
        # 特征选择
        selector.fit(X_train_ros, y_train_ros)
        X_train_new = selector.transform(X_train)
        X_test_new = selector.transform(X_test)
        # 训练朴素贝叶斯模型
        gnb.fit(X_train_new, y_train)
        # 计算特征选择后的训练集和测试集的准确率
        score_train = gnb.score(X_train_new, y_train)
        score_test = gnb.score(X_test_new, y_test)

    return score_train, score_test
```

其中特征选择 selector()拟合时，依然使用的是随机分割和简单重采样的数据集，这与特征选择工具 SelectKBest 网格搜索超参数 k 时使用的数据集相同，使得特征选择时不受不同数据分割和数据重构的影响。

不同数据重构对模型的影响如下所示：

```
# 数据重构对模型的影响
print("原训练集准确率=%.4f,测试集准确率=%.4f" %
      (gnb_score(X, y)[0], gnb_score(X, y)[1]))
print("简单重采样准确率=%.4f,测试集准确率=%.4f" %
      (gnb_score(X, y, sampler=ros)[0], gnb_score(X, y, sampler=ros)[1]))
print("SMOTE重采样准确率=%.4f,测试集准确率=%.4f" %
      (gnb_score(X, y, sampler=sm)[0], gnb_score(X, y, sampler=sm)[1]))
print("ADASYN重采样准确率=%.4f,测试集准确率=%.4f" %
      (gnb_score(X, y, sampler=ada)[0], gnb_score(X, y, sampler=ada)[1]))
print("简单亚采样准确率=%.4f,测试集准确率=%.4f" %
      (gnb_score(X, y, sampler=rus)[0], gnb_score(X, y, sampler=rus)[1]))
```

```
原训练集准确率=0.9944,测试集准确率=0.8220
简单重采样准确率=0.9967,测试集准确率=0.8305
SMOTE重采样准确率=0.9980,测试集准确率=0.8305
ADASYN重采样准确率=0.9979,测试集准确率=0.8305
简单亚采样准确率=0.9972,测试集准确率=0.5508
```

选取 SMOTE 重采样方法，继续对不同数据分割方法对模型的影响进行探讨，如下所示：

```
# 数据分割方式对模型的影响
print("标签分割：训练集准确率=%.4f,测试集准确率=%.4f" %
      (gnb_score(X, y, stratify=y, sampler=sm)[0], gnb_score(X, y, stratify=y, sampler=sm)[1]))
print("随机分割：训练集准确率=%.4f,测试集准确率=%.4f" %
      (gnb_score(X, y, sampler=sm)[0], gnb_score(X, y, sampler=sm)[1]))
```

```
标签分割：训练集准确率=0.9976,测试集准确率=0.7458
随机分割：训练集准确率=0.9980,测试集准确率=0.8305
```

参数 stratify=y 时，代表按照分类标签分割，而不设置参数 stratify 时，默认为随机分割。最终发现随机分割的模型效果更好一些。

内容简介

在完成了对于决策树和朴素贝叶斯算法的探讨后，本节继续引入一个新的算法：支持向量机 SVM 算法，包括以下内容。

1. 主要内容

- SVM 算法介绍；
- 网格搜索 SVM 算法的最优超参数；
- 探讨不同的数据分割方法和数据重构方法对 SVM 模型的影响。

2. 学习目标

学完本节，能解决以下问题：

- SVM 算法的基本原理是什么？
- 如何通过网格搜索 SVM 算法的最优超参数组合？
- 如何定义一个函数，来分别实现不同的数据分割与数据重构方法？
- 在本案例中哪个模型在何种情况下其模型效果最佳？

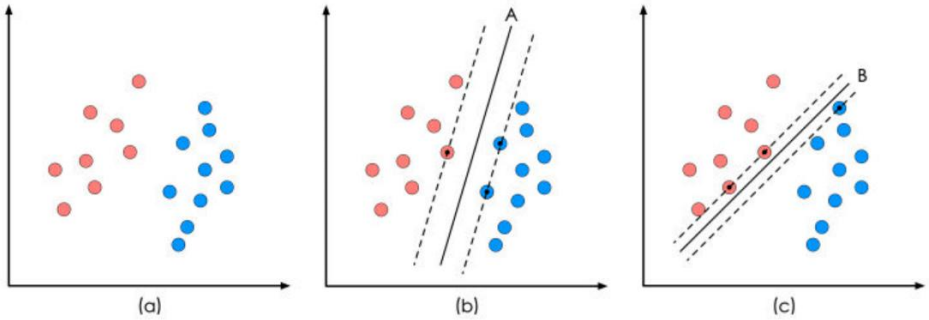
SVM 算法介绍

SVM(Support Vector Machine)-支持向量机，属于有监督机器学习算法，是一种很流行的分类模型，具体应用例如分类图片中植物种类、文本分类等。

以较为简单的二分类问题为例：

给定训练样本集 D = $\{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2),\ldots, (\boldsymbol{x}_m, y_m)\}$, $y_i \in \{-1, +1\}$

SVM 算法就是基于训练集 D 在样本空间中找到一个划分超平面，将不同类别的样本分开。如图所示：



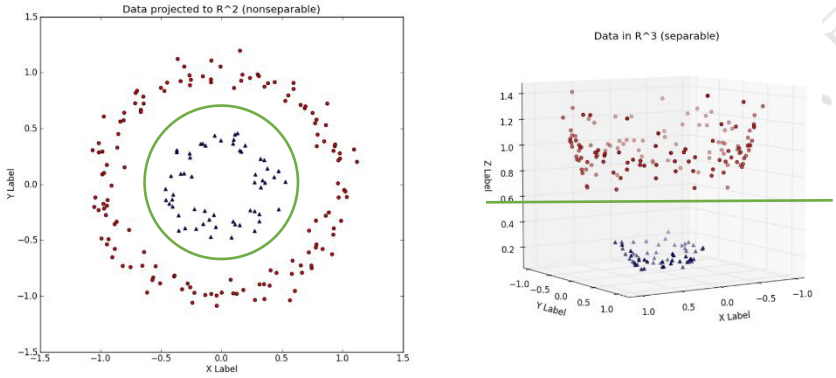
红点和蓝点表示两类样本，在二维空间中这两类显然是可以被一条直线分开的（在高维空间中，划分类别样本的是更高维的超平面，因此将其统称为超平面），SVM 就是要找到最好的一个划分超平面。

SVM 算法中，最优超平面选择的依据是**间隔**(margin)。对比上图(b)和(c)，超平面 A 的间隔要大于 B，也就是说，在保证图中直线斜率不变的情况下移动直线，会在原直线 A 和 B 的两侧找到两个极限位置（即超过该位置就会产生错分现象），即图中虚线所在的位置。

在虚线上的训练样本点被称为**支持向量**(support vector)，两个异类支持向量到超平面的距离之和即为间隔，具有最大间隔的超平面就是最优划分超平面。因为具有最大间隔的超平面，在引入新的样本点时，分类错误的概率更小，即模型具有更强的泛化能

力。SVM 就是要在给定训练样本集的情况下，找到最优划分超平面，从而得到一个分类器。

之前讨论所使用的训练样本是线性可分的，即存在一个划分超平面能将训练样本正确分类。但是在现实任务中，原始样本空间内也许不存在一个能正确划分两类样本的超平面，如图所示：



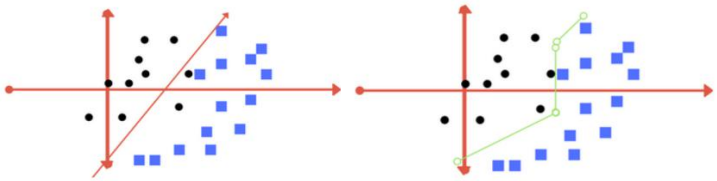
对这样的问题，可以将样本从原始空间映射到一个更高维的特征空间，使得样本在新的特征空间内线性可分。

例如，我们在原图左的基础上加上一维，即 z 轴，并令 $z = x^2 + y^2$。如右图所示，此时，类别样本就可以通过平面 z=b(b 为常数)来划分，从而可以在三维空间中找到最优划分超平面。这样的超平面在二维空间中则表现为一个曲线，而不是一条直线。这种从低维空间到高维空间的映射，即 z 的具体数学表示就称为**核函数**(kernel function)。常用的核函数包括线性核、多项式核、高斯核、拉普拉斯核、Sigmoid 核等。

SVM 算法的优化过程就是通过不断地修改超平面，来找到最大间隔，具体的优化过程涉及比较复杂的数学原理，例如结构风险最小化理论、拉格朗日对偶等内容，在此不多做介绍，感兴趣的同学可以参考链接 https://zhuanlan.zhihu.com/p/24638007，或者周志华《机器学习》。

在本次课程中我们简单讲解两个涉及到 SVM 调参过程中的概念和参数：参数 C 和 gamma。

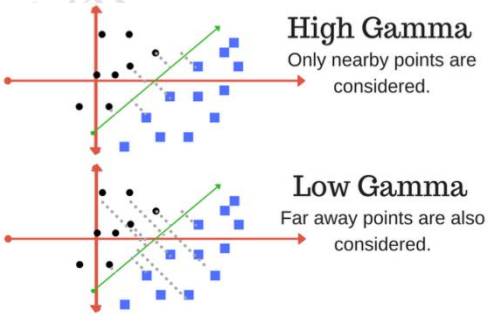
以上所举的两个例子都是较为理想的情况，在实际情况中，样本之间可能出现重叠或者相互“融合”的情况，如下图所示：



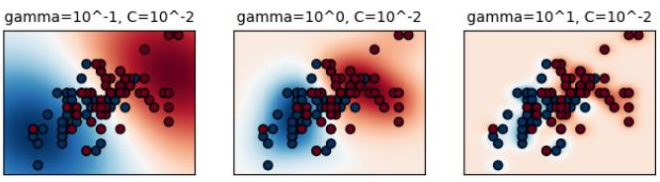
两种选择超平面的方式各有利弊，左图在允许一些错分发生的情况下，可以得到的超平面间隔比较大；右图在不允许错分发生的情况下，得到的超平面间隔比较小。

参数 C 的设置就是在准确率和泛化性能中确定一个平衡，C 越大，表示更看重准确率；C 越小，表示更看重泛化性能。

参数 gamma 就是衡量样本点影响远近的参数，gamma 越大，表示离直线越近的点会被考虑；gamma 越小，表示离直线越远的点会被考虑。如下图所示：



对于超平面是直线的问题，gamma 的作用可能不是很明显。当需要将样本空间映射到高维空间上时，超平面可能是复杂的曲线或者其他情况，如下图所示：



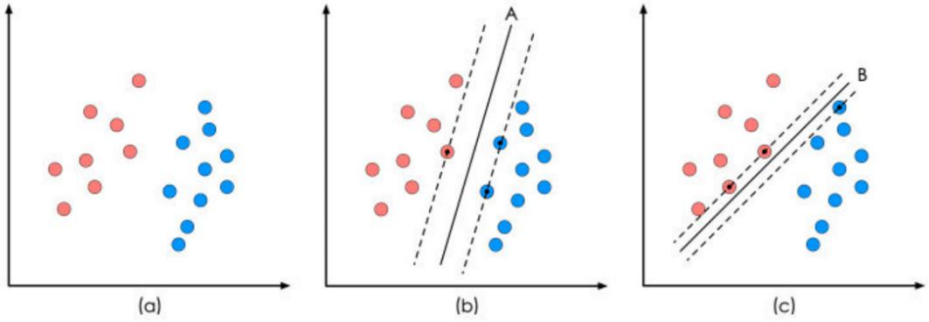
在 C 一定的情况下，gamma 取值对超平面的影响。图中蓝色和红色的阴影就表示

最终得到的超平面。可以看到，gamma 取值越大，超平面的确定考虑了更远处的样本，使得超平面更加的复杂，也更容易过拟合。

SVM 算法总结

1、SVM 算法的基本原理是什么？

SVM(Support Vector Machine)-支持向量机，是一种很流行的分类模型，具体应用例如分类图片中植物种类、文本分类等。其基本原理是基于训练集 D 在样本空间中找到一个划分超平面，将不同类别的样本分开。如图所示：



在虚线上的训练样本点被称为**支持向量**(support vector)，两个异类支持向量到超平面的距离之和即为间隔，具有最大间隔的超平面就是最优划分超平面。因为具有最大间隔的超平面，在引入新的样本点时，分类错误的概率更小，即模型具有更强的泛化能力。SVM 就是要在给定训练样本集的情况下，找到最优划分超平面，从而得到一个分类器。

而对于线性不可分的情况，SVM 算法将样本从原始空间映射到一个更高维的特征空间，使得样本在新的特征空间内线性可分，这种从低维空间到高维空间的映射的数学表示就称为**核函数**(kernel function)。常用的核函数包括线性核、多项式核、高斯核、拉普拉斯核、Sigmoid 核等。

本次课程中介绍了 SVM 两个参数：C 和 gamma。

参数 C 是在准确率和泛化性能中确定一个平衡，C 越大，表示更看重准确度；C 越小，表示更看重泛化性能。

参数 gamma 是衡量样本点影响远近的参数，gamma 越大，表示离直线越近的点会被考虑；gamma 越小，表示离直线越远的点会被考虑。

2、如何通过网格搜索 SVM 算法的最优超参数组合？

关于网格搜索超参数代码如下：

```
from sklearn.svm import SVC
import time
# 返回当前时间的时间戳
start = time.time()

# 拆分出验证集
X_t, X_v, y_t, y_v = train_test_split(X_train_ros, y_train_ros,
                                      shuffle=True, random_state=0)
# 网格搜索超参数
for C in [0.4,1,1.6,2.2,2.8]:
    # 创建SVM模型
    svc = SVC(C=C, kernel='linear', random_state=0)
    # 训练模型
    svc.fit(X_t, y_t)
    # 验证集和测试集的准确率
    score_v = svc.score(X_v, y_v)
    score_t = svc.score(X_test, y_test)
    print("C=%.1f, 验证集准确率=%.4f, 测试集准确率=%.4f" %
          (C, score_v, score_t))
# 返回当前时间的时间戳
end = time.time()

print("调参总用时:", end-start, "s")
```

```
C=0.4, 验证集准确率=0.9023, 测试集准确率=0.8051
C=1.0, 验证集准确率=0.9332, 测试集准确率=0.8136
C=1.6, 验证集准确率=0.9511, 测试集准确率=0.8220
C=2.2, 验证集准确率=0.9560, 测试集准确率=0.8305
C=2.8, 验证集准确率=0.9544, 测试集准确率=0.8305
调参总用时: 322.28082633018494 s
```

首先只搜索正则化超参数 C，其次由于本案例的特征维度较高，因此设置核函数为线性核函数’ linear’ 。由于线性核函数没有 gamma 参数，因此没有对其进行探讨。

由于 SVM 模型的计算成本过高，因此在拆分验证集的时候未使用交叉验证，只是简单的使用 train_test_split()函数，且验证集的大小使用 train_test_split()默认值 fig_size=0.25。同时增加前后时间戳来查看计算成本。

最终得到当 C=2.2 时模型效果最佳。

3、如何定义一个函数，来分别实现不同的数据分割与数据重构方法？

首先是定义使用不同数据分割与数据重构的函数 svm_score()，代码如下：

```
# SVM算法分类
def svm_score(X, y, stratify=None, sampler=None):

    # 使用最优超参数创建模型
    svc = SVC(C=2.2, kernel='linear', random_state=0)
    # 数据分割
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.1, shuffle=True, random_state=0, stratify=stratify)

    if sampler!=None:
        # 数据重构
        X_train_bal, y_train_bal = sampler.fit_resample(X_train, y_train)
        # 训练模型
        svc.fit(X_train_bal, y_train_bal)
        # 计算平衡训练集和测试集的准确率
        score_train = svc.score(X_train_bal, y_train_bal)
        score_test = svc.score(X_test, y_test)
    else:
        # 训练模型
        svc.fit(X_train, y_train)
        # 计算训练集和测试集的准确率
        score_train = svc.score(X_train, y_train)
        score_test = svc.score(X_test, y_test)

    return score_train, score_test
```

其次分别计算模型在不同数据重构集数据分割下的训练集和测试集的准确率，代码如下：

```
# 数据重构对模型的影响
start = time.time()
print("原训练集准确率=%.4f,测试集准确率=%.4f" %
      (svm_score(X, y)[0], svm_score(X, y)[1]))
print("简单重采样准确率=%.4f,测试集准确率=%.4f" %
      (svm_score(X, y, sampler=ros)[0], svm_score(X, y, sampler=ros)[1]))
print("SMOTE重采样准确率=%.4f,测试集准确率=%.4f" %
      (svm_score(X, y, sampler=sm)[0], svm_score(X, y, sampler=sm)[1]))
print("ADASYN重采样准确率=%.4f,测试集准确率=%.4f" %
      (svm_score(X, y, sampler=ada)[0], svm_score(X, y, sampler=ada)[1]))
print("简单亚采样准确率=%.4f,测试集准确率=%.4f" %
      (svm_score(X, y, sampler=rus)[0], svm_score(X, y, sampler=rus)[1]))
end = time.time()
print("总用时:", end-start, "s")
```

```
原训练集准确率=0.9944,测试集准确率=0.8390
简单重采样准确率=0.9976,测试集准确率=0.8390
SMOTE重采样准确率=0.9980,测试集准确率=0.8390
ADASYN重采样准确率=0.9979,测试集准确率=0.8390
简单亚采样准确率=0.9943,测试集准确率=0.6780
总用时: 1003.9283137321472 s
```

```
# 数据分割方式对模型的影响
print("标签分割：训练集准确率=%.4f,测试集准确率=%.4f" %
      (svm_score(X, y, stratify=y, sampler=sm)[0], svm_score(X, y, stratify=y, sampler=sm)[1]))
print("随机分割：训练集准确率=%.4f,测试集准确率=%.4f" %
      (svm_score(X, y, sampler=sm)[0], svm_score(X, y, sampler=sm)[1]))
```

```
标签分割：训练集准确率=0.9988,测试集准确率=0.7627
随机分割：训练集准确率=0.9980,测试集准确率=0.8390
```

最终确定 SVM 模型在 C=2.2 时，数据重构使用 SMOTE 重采样，数据分割使用随机分割方法下，模型效果最佳。

4、在本案例中哪个模型在何种情况下其模型效果最佳？

经过一系列对比分析，三个模型的最优情况下的结果如表所示：

模型	最高分类准确率（测试集）
决策树模型	0.8390
朴素贝叶斯模型	0.8305
SVM 模型	0.8220

因此 SVM 算法暂为最佳分类模型。继续探讨模型对于平衡数据集与非平衡数据集的准确率，例如代码：

```
# SVM算法最终评估
# 数据分割方式：随机分割
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.1, shuffle=True, random_state=0, stratify=None)
# 数据重构方式：SMOTE重采样
X_train_sm, y_train_sm = sm.fit_resample(X_train, y_train)
X_test_sm, y_test_sm = sm.fit_resample(X_test, y_test)
# 创建模型：C=2.2
svc = SVC(C=2.2, kernel='linear', random_state=0)
# 训练模型
svc.fit(X_train_sm, y_train_sm)

print("非平衡训练集准确率：%.4f" % svc.score(X_train, y_train))
print("非平衡测试集准确率：%.4f" % svc.score(X_test, y_test))

print("平衡训练集准确率：%.4f" % svc.score(X_train_sm, y_train_sm))
print("平衡测试集准确率：%.4f" % svc.score(X_test_sm, y_test_sm))
```

```
非平衡训练集准确率：0.9953
非平衡测试集准确率：0.8390
平衡训练集准确率：0.9980
平衡测试集准确率：0.7895
```

出现两种情况：①平衡训练集准确率略高于非平衡训练集准确率，产生原因是该模型由平衡数据训练而来，因此对平衡训练集友好度略高；②平衡测试集准确率低于非平

衡测试集准确率，产生的原因主要是测试集重采样时，新生成样本需要基于原测试集的样本，这一过程可能会扩大一些异常值的影响，从而降低了测试集的准确率。