

1）数据分析（Exploratory Data Analysis）

在选择具体的算法之前，对数据中每一个特征的含义和产生原理有一定的了解：特征是连续的（real-valued）还是离散的（discrete）（**连续性特征和离散性特征**）？target的取值规则是什么？内部特征和target之间是否有某些有规律的定量关系，比如某个特征的值是缺失的时候则target是个固定值？如果特征是连续的，它的直方图（histogram）、均值和方差是如何分布的，折线图？如果特征是离散的，数据分布柱状图？不同的特征值之间是否存在某种顺序或大小关系？

**2）特征工程（Feature Engineering）**

特征选择是特征工程的第一步，它关系到机器学习算法的上限。在实践中特征工程、调整算法参数这两个步骤常需反复进行。

**特征的来源：**一块是业务已经整理好的各种特征数据去找出适合问题需要的特征，另一块是从业务特征中去寻找高级数据特征。

**特征的取舍：**最简单的方法是方差筛选法(sklearn.VarianceThreshold），即方差越大的特征，认为是有用的特征，如果方差比较小，接近于0那么该特征对模型训练没有任何作用。除此之外还可采用过滤法、包装法、嵌入法选择特征。过滤法选择特征：通过相关系数、假设检验、卡方检验和互信息统计学方法来衡量目标值和特征之间的关联程度，设定一个合适的阈值来选择是否舍弃某个特征。可以优先使用卡方检验和互信息来做特征选择。

**寻找高级特征：**通过已有的特征可以组合出更多高级特征，例如通过若干项特征的加减乘除组合来组合出新的、有价值的特征。

**缺失值处理：**对于连续的特征可以选择平均值或者中位数来填充；对于离散特征一般选择众数来填充。

**特殊的特征处理：**以日期为例，可以计算出于未来某个时间点的差值，从而将时间转化为连续值；第二种方法是按照时间的新旧将时间得到一个权重值，比如对于商品，三个月前购买的可以设置一个较低的权重，在三个月内但是三天前购买的可以设置一个较大的权重值，这样处理可以将时间转化为连续特征；第三种方式可以按照日期的单位，按照年月日时分秒进行扩维处理。

**连续特征的离散化处理：**对于连续特征有时候可以将其做离散化处理，这样可以将特征变得高维稀疏。常用的方法是根据阈值进行分组，比如我们根据连续值特征的分位数，将该特征分为高，中和低三个特征。将分位数从0-0.3的设置为高，0.3-0.7的设置为中，0.7-1的设置为高。

**标准化（Z-Score）**

公式为：(X-mean)/std计算时对每个属性/每列分别进行，处理后所有数据都聚集在0附近，方差为1，使用sklearn.preprocessing.scale()函数，可以直接将给定数据进行标准化。使用sklearn.preprocessing.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。

from sklearn import preprocessing

scaler = preprocessing.StandardScaler().fit(X\_train)

scaler.transform(X\_test) #可以直接使用训练集对测试集数据进行转换

**归一化（离差标准化）**

将特征数据缩放到一个指定的范围([0,1])之间，这可以通过preprocessing.MinMaxScaler类实现。（在实例化时可指定feature\_range=(min, max)最大最小值）

from sklearn import preprocessing

min\_max\_scaler = preprocessing.MinMaxScaler().fit(X\_train)

X\_train\_minmax = min\_max\_scaler.transform(X\_test)#将相同的缩放应用到测试集数据中

**正则化（Normalization）**

正则化的过程是将每个样本缩放到单位范数，对每个样本计算p-范数，然后对该样本中的每个元素除以该范数，这样处理后样本的p-范数等于1，其中向量X的p-范数的计算公式：，该方法主要用在文本分类和聚类中

from sklearn import preprocessing

normalizer = preprocessing.Normalizer().fit(X\_train)

normalizer.transform(X\_test)

**离散特征处理：独热编码（one-hot code）**

独热编码是指将具有N个属性值的特征转化为N个二元特征（只包含-1和1，或0和1）。扩充了数据的维度，基于Scikit-learn的例子：

from sklearn import preprocessing

enc = preprocessing.OneHotEncoder()

enc.fit([[0, 0, 3], [1, 1, 0], [0, 2, 1], [1, 0, 2]])

enc.transform([[0, 1, 3]]).toarray()

输出结果：array([[ 1.,  0.,  0.,  1.,  0.,  0.,  0.,  0.,  1.]])

Note: fit了4个数据3个特征，而transform了1个数据3个特征。第一个特征两种值(0: 10, 1: 01)，第二个特征三种值(0: 100, 1: 010, 2: 001)，第三个特征四种值(0: 1000, 1: 0100, 2: 0010, 3: 0001)。所以转换[0, 1, 3]为[ 1.,  0.,  0.,  1.,  0.,  0.,  0.,  0.,  1.]。

**离散特征的离散化处理：标签编码（LabelEncoder）**

机器学习的训练数据均为数值类的，所以文本的数据需要通过标签编码映射为连续的自然数序列，离散的数值也可以映射。

from sklearn.preprocessing import LabelEncoder

le = LabelEncoder().fit(X\_train)

le.transform(X\_test) #将相同的编码规则应用到测试集数据中

**对连续的数据集进行降维，离散的数据独热编码或其他。决策树可以不用做标准化处理，所有基于概率分布模型的算法都可不需要标准化或者归一化处理。**

**异常特征的样本清洗：**可以用KMeans聚类将训练样本分成若干个簇，如果某一个簇里面的样本数很少而且簇质心和其他所有的簇都很远，那么这个簇里面的样本极有可能是异常特征样本了。我们可以将其从训练集过滤掉；第二种是异常点检测方法，主要是使用iForest或者one class SVM，使用异常点检测的机器学习算法来过滤所有的异常点。

**处理不平衡数据：**对于样本分布不均衡的数据（如训练集中A类别样本占比90%，B类别样本占比10%）如果不考虑类别不平衡问题，训练出来的模型对于类别B的预测准确率会很低。处理不平衡数据有两种办法分别是权重法和采样法：

权重法：对训练集中的每个类别加一个权重，如果该类别的样本数多，那么它的权重值低，反之越小。

**采样法：**数据采样分为上/下采样，下采样：以数据量少的一方样本数量为标准；上采样以数据量多的一方的样本数量为标准，把样本数量较少的类的样本数量生成和样本数量多的一方相同。SMOTE算法通过人工合成的方法来生成少类别的样本。对于某一个缺少样本的类别，它会随机找出几个该类别的样本，再找出最靠近这些样本的若干个该类别样本，组成一个候选合成集合，然后在这个集合中不停的选择距离较近的两个样本，在这两个样本之间，比如中点，构造一个新的该类别样本。（**保证训练集样本数均衡**）

# 集成学习（ensemble learning）

集成学习的定义：集成学习不是一个单独的机器学习算法，而是通过构建并结合多个机器学习器来完成学习任务，集成学习可以用于分类问题集成，回归问题集成，特征选取集成，异常点检测集成等等。

集成学习面对的两个问题：集成学习有两个主要的问题：如何得到若干个个体学习器；如何选择一种结合策略将这些若干个学习器组合成一个强学习器。

集成学习之个体学习器：个体学习器都是一个种类的（同质）或者不是同一个种类的（异质）。同质个体学习器的应用是最广泛的，使用最多的同质个体学习器模型是CART决策树和神经网络。

同质个体学习分类：同质个体学习器按照个体学习器之间是否存在依赖关系可以分为两类：第一类是个体学习之间存在强依赖关系，一系列个体学习器基本都需要串行生成，代表算法是boosting系列算法；第二类是个体学习器可以并行生成，代表算法是bagging和随机森林系列算法

**集成学习之boosting：**首先是从训练集用初始权重训练出一个弱学习器1，再根据若学习器的学习误差率表现来更新训练样本的权重，使得之前弱学习器1的学习误差率高的训练样本点的权重变高，使得这些误差率高的点在后面的弱学习器2中得到更多的重视，然后基于调整权重后的训练集来训练弱学习器2，如此重复，直到弱学习器数达到事先指定的数目T，最终将这T个弱学习器通过集合策略进行整合，得到最终的强学习器。Boosting系列算法里最著名算法主要有AdaBoost算法和提升树(boosting tree)系列算法。，提升树系列算法中应用最广泛的是GBDT梯度提升树（Gradient Boosting Tree）。

Boosting类得所有算法都面对的问题：如何计算学习误差率e、如何得到弱学习器的权重系数a、如何更新样本权重w和使用哪一种弱学习器的结合策略。任何学习器都可以用于Adaboosting，使用最广泛的Adaboosting弱学习器是决策树和神经网络，Adaboost的主要缺点对异常样本敏感，优点是分类精度高、构造简单、不容易发生过拟合。该算法的详细思路可阅读：<https://www.cnblogs.com/pinard/p/6133937.html>

**集成学习之bagging：**与boosting原理不同，bagging的弱学习器之间没有依赖关系，可以并行生成。Bagging的个体学习器的训练集是通过随机采样得到的，通过T次随机采样就可以得到T个样本集，使用这T个样本集分别独立训练处T个弱学习器，再对这T个弱学习器通过集合策略得到最终的强学习器。这里的随机采样是采用有放回的随机采样。随机森林系列是bagging的一个特化进阶版，随机森林的弱学习器都是决策树，所谓进阶石随机森林在bagging的样本随机采样基础上，又加上了特征的随机选择。

集成学习之结合策略：

对于数值类的回归预测问题，通常使用的结合策略是加权平均法（对T个弱学习器分配不同的权重值），既对于若干个弱学习器的输出进行加权平均得到最终的预测输出。

对于分类问题的预测，通常使用的是投票法，最简单的投票法是相对多数投票法，如果不止一个类别获得最高投票则随机选择一个作为最终的类别。还有复杂的投票法如：绝对多数投票和加权投票法。

上述两种方法都是对弱学习器的结果做平均或者投票，可能学习误差较大，于是就有了学习法这种方法，对于学习法，代表方法是stacking，当使用stacking的结合策略时， 我们不是对弱学习器的结果做简单的逻辑处理，而是再加上一层学习器，也就是说将弱学习器的学习结果作为输入，将训练集的输出作为输出，重新训练一个学习器来得到最终结果。

4）模型调参（网格寻优）：对于集成算法的超参数分为框架参数和弱学习器参数

超参数就是需要人工设定的参数，比如决策树的深度等。Python提供的超参数寻优算法：网格搜索(grid search)

网格搜索实际是暴力搜索，首先将需要调整的参数设置为一组候选值(字典类型)，然后网格搜索会穷举各种参数的组合，根据设定的评分机制找到最好的那一组设置。

from sklearn.model\_selection import GridSearchCV #导包

param\_test = { 'min\_samples\_split': list(range(50,201,20)),

'max\_depth':list(range(3,14,2))} #参数取值字典列表

gsearch1 = GridSearchCV(

estimator= RandomForestClassifier(max\_depth=8), #设置随机森林的固定

param\_grid = param\_test, #代入需要调整的超参数字典列表

scoring='roc\_auc', #参数的评价指标

cv=5) #K折交叉验证

gsearch1.fit(X,y) #训练模型

cv\_result = pd.DataFrame.from\_dict(clf.cv\_results\_)

with open('cv\_result.csv','w') as f:

cv\_result.to\_csv(f)

print( gsearch1.best\_params\_, #输出最优参数

gsearch1.best\_score\_) #输出最优参数对应的scoring分数

网格调参详见：<https://www.cnblogs.com/nwpuxuezha/p/6618205.html>。

5）常用算法的参数调整：

**5.1 RF随机森林**（RF包括RandomForestClassifier和RandomForestRegressor）

RF需要调参的参数包括两部分：Bagging框架的参数和弱学习器决策树的参数，以下是boosting框架的参数，弱学习器决策树的参数参见附录。

1) n\_estimators: 也就是弱学习器的最大迭代次数。一般来说太小，容易欠拟合，n\_estimators太大，计算量会太大，并且n\_estimators到一定的数量后，再增大n\_estimators获得的模型提升会很小，所以一般选择一个适中的数值。默认是100。

2) oob\_score :即是否采用袋外样本来评估模型的好坏。默认识False。个人推荐设置为True，因为袋外分数反应了一个模型拟合后的泛化能力。

随机森林算法原理：<https://www.cnblogs.com/pinard/p/6156009.html>。

**5.2 Adaboost算法**（两个：AdaBoostClassifier分类，AdaBoostRegressor回归）

Adaboost类需要调参的参数包括两部分：boosting框架的参数和弱学习器的参数，以下是boosting框架的参数，弱学习器决策树的参数参见附录。

1）base\_estimator：可以选择任何一个分类或者回归学习器，常用的一般是CART决策树或者神经网络MLP。默认是决策树。注意的点是，如果我们选择的AdaBoostClassifier算法是SAMME.R，则我们的弱分类学习器还需要支持概率预测，也就是在scikit-learn中弱分类学习器对应的预测方法除了predict还需要有predict\_proba。

2）algorithm：这个参数只有AdaBoostClassifier有。主要原因是scikit-learn实现了两种Adaboost分类算法，SAMME和SAMME.R。两者的主要区别是弱学习器权重的度量，SAMME使用了二元分类Adaboost算法的扩展，即用对样本集分类效果作为弱学习器权重，而SAMME.R使用了对样本集分类的预测概率大小来作为弱学习器权重。由于SAMME.R使用了概率度量，迭代一般比SAMME快，因此algorithm默认值是SAMME.R。我们一般使用默认的SAMME.R就够了，但是要注意的是使用了SAMME.R， 则弱分类学习器参数base\_estimator必须限制使用支持概率预测的分类器。SAMME算法则没有这个限制。

4) n\_estimators： 最大的弱学习器的个数。一般来说太小，容易欠拟合，太大，又容易过拟合，一般选择一个适中的数值。默认是50。在实际调参的过程中，常常和参数learning\_rate一起考虑。

5) learning\_rate:  每个弱学习器的权重缩减系数ν，ν的取值范围为0<ν≤1，对于同样的训练集拟合效果，较小的ν意味着需要更多的弱学习器的迭代次数。通常用步长和迭代最大次数一起来决定算法的拟合效果。所以这两个参数n\_estimators和learning\_rate要一起调参。一般来说，可以从一个小一点的ν开始调参，默认是1。

Adaboost算法原理：<https://www.cnblogs.com/pinard/p/6133937.html>。

**5.3 GBDT梯度提升树**（GradientBoostingClassifier和GradientBoostingRegressor）

GBDT算法限定了弱学习器只能使用CART回归树模型，是通过损失函数的负梯度来拟合，找到了一种通用的拟合损失误差的办法。需要调参的参数包括两部分：boosting框架的参数和弱学习器的参数，以下是boosting框架的参数，弱学习器决策树的参数参见上文。

1) n\_estimators： 最大的弱学习器的个数。一般来说太小，容易欠拟合，太大，又容易过拟合，一般选择一个适中的数值。默认是100。常常和参数learning\_rate一起考虑。

2) learning\_rate:  每个弱学习器的权重缩减系数ν，ν的取值范围为0<ν≤1，对于同样的训练集拟合效果，较小的ν意味着需要更多的弱学习器的迭代次数。通常用步长和迭代最大次数一起来决定算法的拟合效果。所以这两个参数n\_estimators和learning\_rate要一起调参。一般来说，可以从一个小一点的ν开始调参，默认是1。

3) subsample:子采样，取值为(0,1]，注意这里的子采样和随机森林不一样，随机森林使用的是放回抽样，而这里是不放回抽样。如果取值为1，则全部样本都使用，等于没有使用子采样。如果取值小于1，则只有一部分样本会去做GBDT的决策树拟合。选择小于1的比例可以减少方差，即防止过拟合，但是会增加样本拟合的偏差，因此取值不能太低。推荐在[0.5, 0.8]之间，默认是1.0，即不使用子采样。

4) loss: GBDT算法中的损失函数，分类问题和回归问题所采用的损失函数不同。分类模型，有对数似然损失函数"deviance"和指数损失函数"exponential"两者输入选择。默认是对数似然损失函数"deviance"。推荐使用默认的"deviance"。指数损失函数等于Adaboost算法。回归模型，有均方差"ls", 绝对损失"lad", Huber损失"huber"和分位数损失“quantile”。默认是均方差"ls"。一般来说，如果数据的噪音点不多，用默认的均方差"ls"比较好。如果是噪音点较多，则推荐用抗噪音的损失函数"huber"。而如果需要对训练集进行分段预测的时候，则采用“quantile”。

GBDT算法原理：<https://www.cnblogs.com/pinard/p/6140514.html>。

# 附录 Python

0、python的机器学习模型提供的接口：

公共借口：

model.fit() # 模型训练，监督模型来说是 fit(X,y)，非监督模型是 fit(X)。

监督模型的接口：

model.predict(X\_new) # 用训练的模型预测新样本并输出target

model.predict\_proba(X\_new) # 输出预测的target的概率值

model.score() # 对模型预测进行评估

非监督模型：

model.transform() # 从数据中学到新的基空间

model.fit\_transform() # 从数据中学到新的基，并将数据按照这个基转化

1、决策树算法常用参数及注释：

1)划分时考虑的最大特征数max\_features: 默认是"auto",意味着划分时最多考虑√N个特征；如果是"log2"意味着划分时最多考虑log2N个特征。如果是整数，代表考虑的特征绝对数。如果是浮点数，代表考虑特征百分比，即考虑取整后的特征数。一般用默认的"auto"就可以了，如果特征数非常多，可以灵活使用刚才描述的其他取值来控制划分时考虑的最大特征数，以控制决策树的生成时间。

2)决策树最大深度max\_depth:默认可以不输入，如果不输入的话，默认值是3。一般来说，数据少或者特征少的时候可以不管这个值。如果模型样本量多，特征也多的情况下，推荐限制这个最大深度，具体取值取决于数据的分布。常用的取值10-100。

3)内部节点再划分所需最小样本数min\_samples\_split:这个值限制了子树继续划分的条件，如果某节点的样本数少于该值，则不会继续再尝试选择最优特征来进行划分。默认是2.如果样本量不大，不需要管这个值。如果样本量数量级非常大，则推荐增大这个值。

4)叶子节点最少样本数min\_samples\_leaf:这个值限制了叶子节点最少的样本数，如果某叶子节点数目小于样本数，则会和兄弟节点一起被剪枝。 默认是1,可以输入最少的样本数的整数，或者最少样本数占样本总数的百分比。如果样本量不大，不需要管这个值。如果样本量数量级非常大，则推荐增大这个值。

5）叶子节点最小的样本权重和min\_weight\_fraction\_leaf：这个值限制了叶子节点所有样本权重和的最小值，如果小于这个值，则会和兄弟节点一起被剪枝。 默认是0，就是不考虑权重问题。一般来说，如果我们有较多样本有缺失值，或者分类树样本的分布类别偏差很大，就会引入样本权重，这时我们就要注意这个值了。

6)最大叶子节点数max\_leaf\_nodes:通过限制最大叶子节点数，可以防止过拟合，默认是"None”，即不限制最大的叶子节点数。如果加了限制，算法会建立在最大叶子节点数内最优的决策树。如果特征不多，可以不考虑这个值，但是如果特征分成多的话，可以加以限制，具体的值可以通过交叉验证得到。

7)节点划分最小不纯度min\_impurity\_split:这个值限制了决策树的增长，如果某节点的不纯度(基于基尼系数，均方差)小于这个阈值，则该节点不再生成子节点。即为叶子节点。一般不推荐改动默认值1e-7。

1、机器学习算法中的随机数生成：

1）numpy随机数生成API：（import numpy as np）

np.random.rand(m, n) #生成一个m X n维数组，取值在[0, 1]

np.random.randn(m, n) #生成一个m X n维数组，服从N(0, 1)的正态分布

a\*np.random.randn(m, n)+u #生成一个均值是u方差是a^2的正态分布

np.random.randint(min, max, size=[m, n]) #生成取值范围在[min, max]的m X n维数组

np.random. random\_sample([size]) #返回随机的浮点数，在半开区间 [0.0, 1.0)

例：(5-2)\*np.random.random\_sample([3, 4])+2 #返回范围在[2,5)之间的一个3 X 4维数组

2) scikit-learn随机数据生成API介绍：（from sklearn. datasets import samples\_generator）

make\_regression 生成回归模型的数据

make\_hastie\_10\_2,make\_classification/make\_multilabel\_classification生成分类模型数据

make\_blobs生成聚类模型数据

make\_gaussian\_quantiles生成分组多维正态分布的数据

详细随机数生成例子：<http://www.cnblogs.com/pinard/p/6047802.html>。