基于机器学习-集成学习算法

蚂蚁金服-借呗额度欺诈问题分析

Haiwang Luo 罗海王

阿里 Python 数据分析第五周报告

目　　录

1. 实验目的及要求

1。新的任务我们来做一个蚂蚁金服中借呗额度欺诈的问题，

2。数据集包括了 2015 年 9 月份两天时间内的交易数据，284807 笔交易中，一共有 492 笔是欺诈行为。

3。输入数据一共包括了 28 个特征 V1，V2，……V28 对应的取值，以及交易时间 Time 和交易金额 Amount。

4。为了保护数据隐私，我们不知道 V1 到 V28 这些特征代表的具体含义，只知道这 28 个特征值是通过 PCA 变换得到的结果。

5。另外字段 Class 代表该笔交易的分类，Class=0 为正常（非欺诈），Class=1 代表欺诈。

6。我们的目标是针对这个数据集构建一个信用卡欺诈分析的分类器。还是通过加载数据，准备数据和分类阶段这三个步骤去做本次项目的分析吧

2. 集成学习的知识梳理

2.1 集成学习介绍

2.1.1 何为集成方法？

集成学习是一种机器学习范式。在集成学习中，我们会训练多个模型（通常称为「弱学习器」）解决相同的问题，并将它们结合起来以获得更好的结果。最重要的假设是：当弱模型被正确组合时，我们可以得到更精确和/或更鲁棒的模型。集成方法的思想是通过将这些弱学习器的偏置和/或方差结合起来，从而创建一个「强学习器」（或「集成模型」），从而获得更好的性能。

2.1.2 组合弱学习器

很重要的一点是：我们对弱学习器的选择应该和我们聚合这些模型的方式相一致。如果我们选择具有低偏置高方差的基础模型，我们应该使用一种倾向于减小方差的聚合方法；而如果我们选择具有低方差高偏置的基础模型，我们应该使用一种倾向于减小偏置的聚合方法。

2.1.3 如何组合这些模型的问题？- 偏差和方差

广义的偏差（bias）描述的是预测值和真实值之间的差异，方差（variance）描述距的是预测值作为随机变量的离散程度。模型的偏差和方差：bagging 和 stacking 中的基模型为强模型（偏差低方差高），boosting 中的基模型为弱模型。

2.2 自助聚合 bagging

该方法通常考虑的是同质弱学习器，相互独立地并行学习这些弱学习器，并按照某种确定性的平均过程将它们组合起来。 Bagging 是 Bootstrap Aggregating 的缩写。Bagging 是为了得到泛化能力强的集成，因而就需要让各个子学习器之间尽可能独立，但是如果将样本分为了不同的不重合子集，那么每个基学习器学习的样本就会不足。所以它采用一种自助采样的方法（boostrap sampling）每次从数据集中随机选择一个 subset，然后放回初始数据集，下次取时，该样本仍然有一定概率取到。然后根据对每个 subset 训练出一个基学习器，然后将这些基学习器进行结合。对于分类任务可以通过 vote 来输出结果，回归任务可以求平均值。Bagging 的代表是 Random Forest，RF 是在决策树作为基学习器通过 Bagging 思想建立的。Random Forest 是一种基于 Bagging 思想的 Ensemble learning 方法，它实际上就是 Bagging + 决策树。Random Forest 可以用来做分类也可以做回归，做分类时最后多棵树的分类器通过 voting 来决定分类结果；做回归时，由多棵树预测值的 averaging 来决定预测结果。

2.3 提升法 boosting

该方法通常考虑的也是同质弱学习器。它以一种高度自适应的方法顺序地学习这些弱学习器（每个基础模型都依赖于前面的模型），并按照某种确定性的策略将它们组合起来。

Boosting 是一种将弱学习器转换为强学习器的算法，它的机制是：先从初始训练集训练出一个基学习器，然后根据基学习器的表现对训练样本进行调整，使得先前基学习器做错的训练样本在后续的训练中得到更多的关注，然后基于调整后的样本分布来训练下一个基学习器。

Boosting 的代表是 Adam Boosting。Adaboost 是 Boosting 算法中的代表，它的思想也便是基于 Boosting 思想的。在 adaboost 的运算过程中，一开始在训练样本时，为每个子样本赋予一个权重，一开始这些权重都是相等的，然后在训练数据集上训练出一个弱分类器，并计算这个弱分类器在每个子样本上的错误率，在第二次对这同一数据集进行训练时，将会根据分类器的错误率对子数据集中各个权重进行调整，分类正确的权重降低，分类错误的权重上升，这些权重的总和不变。最终得到的分类器会基于这些训练的弱分类器的分类错误率来分配不同的决定系数，从而使权重更新时，错误样本具有更高的权重。最后以此来更新各个样本的权重，直至达到迭代次数或者错误率为 0。所以 Adaboost 会对那些影响准确率的数据额外关注，从而会降低 bias，而导致 overfit。

2.4 堆叠法 stacking

该方法通常考虑的是异质弱学习器，并行地学习它们，并通过训练一个「元模型」将它们组合起来，并通过训练一个「元模型」将它们组合起来，根据不同弱模型的预测结果输出一个最终的预测结果。stacking 是一种将弱学习器集成进行输出的策略，其中，在 stacking 中，所有的弱学习器被称作 0 级（0 level）学习器，他们的输出结果被一个 1 级（1 level）学习器接受，然后再输出最后的结果。这是实际上是一种分层的结构，前面提到的就是一种最基本的二级 Stacking。另外，在 bagging 或者 boosting 中，所有的弱学习器一般都要求是相同的模型，如决策树，而 stacking 中可以是不同的模型，如 KNN、SVM、LR、RF 等。

2.5 对集成学习的三种模式的理解

bagging 的重点在于获得一个方差比其组成部分更小的集成模型，而 boosting 和 stacking 则将主要生成偏置比其组成部分更低的强模型（即使方差也可以被减小）。自助聚合 bagging：（并行集成）是为了得到泛化能力强的集成，因而就需要让各个子学习器之间尽可能独立，

提升法 boosting：（序列集成）是一种将弱学习器转换为强学习器的算法，根据基学习器的表现对训练样本进行调整，然后基于调整后的样本分布来训练下一个基学习器。

堆叠法 stacking： 考虑的是异质弱学习器，并行地学习它们、并通过训练一个「元模型」将它们组合起来，根据不同弱模型的预测结果输出一个最终的预测结果。

2.6. 对集成学习算法的进一步探索

2.6.1 随机森林的剪枝

集成学习在实践中的训练效果很好 如果在测试集中的表现都是 100% 那应该是过拟合了 考虑一下剪枝

* n_estimators：这是森林中树木的数量，即基评估器的数量。n_estimators 越大，模型的效果往往越好。

* max_features：决策树划分时考虑的最大特征数。max_features 值越大，模型学习能学习到的信息越多，越容易过拟合。

* max_depth：决策树最大深度。常用的可以取值 10-100 之间。值越大，决策树越复杂，越容易过拟合。

* min_samples_split：内部节点再划分所需最小样本数。值越大，决策树越简单，越不容易过拟合。

* min_samples_leaf：叶子节点最少样本数。值越大，叶子节点越容易被被剪枝，决策树越简单，越不容易过拟合。

* max_leaf_nodes: 最大叶子节点数。值越小，叶子节点个数越少，可以防止过拟合。

2.6.2 Adaboost 剪枝

* n_estimators:基分类器提升（循环）次数，默认是 50 次，这个值过大，模型容易过拟合；值过小，模型容易欠拟合。

* learning_rate:学习率，表示梯度收敛速度，默认为 1，如果过大，容易错过最优值，如果过小，则收敛速度会很慢

* algorithm:boosting 算法，也就是模型提升准则，有两种方式 SAMME, 和 SAMME.R 两种，默认是 SAMME.R，两者的区别主要是弱学习器权重的度量。SAMME 是对样本集预测错误的概率进行划分的，SAMME.R 是对样本集的预测错误的比例，即错分率进行划分的，默认是用的 SAMME.R。

* random_state: 随机种子设置

3. XGBoost 的引入

3.1 传统的 GBDT

(gradient boosting decision tree)也就是梯度提升决策树:

这是一种基于树的集成算法.多棵树的集合就构成了 GBDT。其实 GBDT 是对残差的拟合。GBDT 的目标函数是预测值和真实值差的累加，也就是误差累加，可以看出每一步计算都依赖于上面所有步的误差，效率比较低。

3.2 xgboost 特点

3.2.1 传统 GBDT 以 CART 作为基分类器，xgboost 还支持线性分类器，这个时候 xgboost 相当于带 L1 和 L2 正则化项的逻辑斯蒂回归（分类问题）或者线性回归（回归问题）。-- 可以通过 booster [default=gbtree]设置参数:gbtree: tree-based models/gblinear: linear models

3.2.2 传统 GBDT 在优化时只用到一阶导数信息，xgboost 则对代价函数进行了二阶泰勒展开，同时用到了一阶和二阶导数。顺便提一下，xgboost 工具支持自定义代价函数，只要函数可一阶和二阶求导。-- 对损失函数做了改进（泰勒展开，一阶信息 g 和二阶信息 h）

3.2.3 xgboost 在代价函数里加入了正则项，用于控制模型的复杂度。正则项里包含了树的叶子节点个数、每个叶子节点上输出的 score 的 L2 模的平方和。从 Bias-variance tradeoff 角度来讲，正则项降低了模型 variance，使学习出来的模型更加简单，防止过拟合，这也是 xgboost 优于传统 GBDT 的一个特性。-- 正则化包括了两个部分，都是为了防止过拟合，剪枝是都有的，叶子结点输出 L2 平滑是新增的。

3.2.4 shrinkage and column subsampling。shrinkage 缩减类似于学习速率，在每一步 tree boosting 之后增加了一个参数 n（权重），通过这种方式来减小每棵树的影响力，给后面的树提供空间去优化模型。column subsampling 列(特征)抽样，说是从随机森林那边学习来的，防止过拟合的效果比传统的行抽样还好（行抽样功能也有），并且有利于后面提到的并行化处理算法。

3.2.5 split finding algorithms(划分点查找算法) approximate algorithm— 近似算法，提出了候选分割点概念，先通过直方图算法获得候选分割点的分布情况，然后根据候选分割点将连续的特征信息映射到不同的 buckets 中，并统计汇总信息。Weighted Quantile Sketch—分布式加

权直方图算法。可并行的近似直方图算法。树节点在进行分裂时，我们需要计算每个特征的每个分割点对应的增益，即用贪心法枚举所有可能的分割点。当数据无法一次载入内存或者在分布式情况下，贪心算法效率就会变得很低，所以 xgboost 还提出了一种可并行的近似直方图算法，用于高效地生成候选的分割点。

3.2.6 对缺失值的处理。对于特征的值有缺失的样本，xgboost 可以自动学习出它的分裂方向。

3.2.7 Built-in Cross-Validation（内置交叉验证)

3.2.8 continue on Existing Model（接着已有模型学习）

3.2.9 High Flexibility（高灵活性）

3.2.10 并行化处理 —系统设计模块,块结构设计等

3.3 xgboost 模型参数

* max_depth:int |每个基本学习器树的最大深度，可以用来控制过拟合。典型值是 3-10

* learning_rate=0.1：即是 eta，为了防止过拟合，更新过程中用到的收缩步长，使得模型更加健壮。典型值一般设置为：0.01-0.2。

* n_estimators=100,估计器的数量

* objective：定义学习任务及相应的学习目标，可选目标函数如下：

“reg:linear” —— 线性回归

“reg:logistic” —— 逻辑回归

“binary:logistic” —— 二分类的逻辑回归问题，输出为概率

“binary:logitraw” —— 二分类的逻辑回归问题，输出的结果为 wTx

“count:poisson” —— 计数问题的 poisson 回归，输出结果为 poisson 分布。在 poisson 回归中，max_delta_step 的缺省值为 0.7。(used to safeguard optimization)

“multi:softmax” —— 让 XGBoost 采用 softmax 目标函数处理多分类问题，同时需要设置参数 num_class（类别个数）。返回预测的类别(不是概率)。

“multi:softprob” —— 和 softmax 一样，但是输出的是 ndata * nclass 的向量，可以将该向量 reshape 成 ndata 行 nclass 列的矩阵。每行数据表示样本所属于每个类别的概率。

“rank:pairwise” —— set XGBoost to do ranking task by minimizing the pairwise loss

* booster: default="gbtree"，可选 gbtree 和 gblinear，gbtree 使用基于树的模型进行提升计算，gblinear 使用线性模型进行提升计算

* n_jobs：线程数目

* gamma：0，损失阈值，在树的一个叶节点上进行进一步分裂所需的最小损失减少量，越大，算法越保守。取值范围为：[0,∞]。在节点分裂时，只有分裂后损失函数的值下降了，才会分裂这个节点。Gamma 指定了节点分裂所需的最小损失函数下降值。这个参数的值越大，算法越保守。这个参数的值和损失函数息息相关，所以是需要调整的。

* min_child_weight=1, 拆分节点权重和阈值，如果节点的样本权重和小于该阈值，就不再进行拆分。在现行回归模型中，这个是指建立每个模型所需要的最小样本数。越大，算法越保守，可以用来减少过拟合。 取值范围为：[0,∞]

* max_delta_step=0, 每棵树的最大权重估计。如果它的值被设置为 0，意味着没有约束；如果它被设置为一个正值，它能够使得更新的步骤更加保守。通常这个参数是没有必要的，但是如果在逻辑回归中类别极其不平衡这时候他有可能会起到帮助作用。把它范围设置为 1-10 之间也许能控制更新。 取值范围为：[0,∞]

* scale_pos_weight=1,用来控制正负样本的比例，平衡正负样本权重，处理样本不平衡。在类别高度不平衡的情况下，将参数设置大于 0，可以加快收敛。

3.3.1 xgboost 的优势

- 正则化项防止过拟合
- xgboost 不仅使用到了一阶导数，还使用二阶导数，损失更精确，还可以自定义损失
- XGBoost 的并行优化，XGBoost 的并行是在特征粒度上的
- 考虑了训练数据为稀疏值的情况，可以为缺失值或者指定的值指定分支的默认方向，这能大大提升算法的效率
- 支持列抽样，不仅能降低过拟合，还能减少计算

4. 实验伪代码及功能描述

4.1 数据导入与预处理

```
import pandas as pd
from sklearn.model_selection import train_test_split

# 第一步：加载数据(已经经过PCA 降维了)
# data =
pd.read_csv('https://www.dropbox.com/s/qk3u8j529tgj72d/alipay_huabei_FS1.csv?dl=0')
data = pd.read_csv('/Users/haiwangluo/Desktop/alipython_files/alipay_huabei_FS1.csv')
# 筛选特征值
x_data = data.iloc[:, 1: -2]    # [行开始索引:行结束索引， 列开始索引:列结束索引]
# print(x_data.head())
# 筛选目标值
x_target = data['Class']
# print(x_target.value_counts())

# 第二步：准备数据
x_train, x_test, y_train, y_test = train_test_split(x_data, x_target, test_size=0.3,
stratify=x_target, random_state=1)
```

4.2 使用 Pipeline 管道机制，流水线化构建算法模型

```
from sklearn.pipeline import Pipeline
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
from xgboost import XGBClassifier

estimator_models = [
    Pipeline(steps=[('rfc', RandomForestClassifier(random_state=1))]),
    Pipeline(steps=[('abc', AdaBoostClassifier(random_state=1))]),
    Pipeline(steps=[('xgbc', XGBClassifier(random_state=1))])
]
```

4. 3 使用 GridSearchCV 参数调优

4.3.1 构建 GridSearchCV 调参的函数

```
from sklearn.model_selection import train_test_split, GridSearchCV
import datetime
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

def gridsearchcv_works(x_train, x_test, y_train, y_test, estimator_name,
estimator_model, params):
    '''GridSearchCV 参数调优
    @param x_train: 训练集特征值
    @param x_test: 测试集特征值
    @param y_train: 训练集目标值
    @param y_test: 测试集目标值
    @param estimator_name: 算法模型的名称
    @param estimator_model: 算法模型
    @param params: 算法模型中需要调优的参数
    @return: 返回关于训练结果的列表
    '''
    # 调用 datetime 函数，-> 获取算法运行时间
    time_start = datetime.datetime.now()
    # 使用 GridSearchCV 调参数， 选择到对应的算法模型
    estimator = GridSearchCV(estimator_model, param_grid=params, scoring='accuracy',
cv=3)
    estimator.fit(x_train, y_train)
    y_predict = estimator.predict(x_test)
    print('%s 模型的最优参数组合为 %s' % (estimator_name, estimator.best_params_))
    # print('%s 模型的分类结果报告：\n%s' % (estimator_name,
classification_report(y_test, y_predict, target_names=['0 正常(非欺诈)', '1 代表欺诈
']))))
    # print('%s 模型的混淆矩阵为：\n%s' % (estimator_name, confusion_matrix(y_test,
y_predict)))

    # 把这些输出数据保存在列表中: 最优分数、准确率、精确率、召回率、F1 score、训练时间
    e_bs = estimator.best_score_  # 最优分数
    e_bs = float('%.5f' % e_bs)
    e_ac = accuracy_score(y_test, y_predict)  # 准确率
    e_ac = float('%.5f' % e_ac)
    e_ps = precision_score(y_test,y_predict) # 精确率
    e_ps = float('%.5f' % e_ps)
    e_rs = recall_score(y_test, y_predict) # 召回率、
    e_rs = float('%.5f' % e_rs)
    e_fs = f1_score(y_test, y_predict) # F1 score
    e_fs = float('%.5f' % e_fs)
    e_tp = datetime.datetime.now() - time_start # 训练时间

# 构建一个列表，并返回给外界使用
    list = [e_bs, e_ac, e_ps, e_rs, e_fs, e_tp]
    print(list)
    return list
```

4.3.2 设置分类器的参数

```
parameters_list = [
    {'rfc__max_depth': [9], 'rfc__n_estimators': [9], 'rfc__min_samples_leaf': [10],
'rfc__criterion': ['gini']},
    {'abc__n_estimators': [50], 'abc__learning_rate': [1.0], 'abc__algorithm':
['SAMME.R']},
    {'xgbc__max_depth': [11], 'xgbc__n_estimators': [54], 'xgbc__learning_rate':
[1.0], 'xgbc__min_child_weight': [1]}
]
```

4.3.3 调用函数，对具体的分类器进行 GridSearchCV 参数调优

```
gw1 = gridsearchcv_works(x_train, x_test, y_train, y_test, '随机森林算法',
estimator_models[0], parameters_list[0])
gw2 = gridsearchcv_works(x_train, x_test, y_train, y_test, 'AdaBoost 提升算法',
estimator_models[1], parameters_list[1])
gw3 = gridsearchcv_works(x_train, x_test, y_train, y_test, 'XGBoost 分类提升算法',
estimator_models[2], parameters_list[2])
```

4.3.4 画表格，把所有 model 的结果都集中到一起

```
# 列(0-5)：'最优分数', '准确率', '精确率', '召回率', 'F1 score', '训练时间'
col = ['最优分数', '准确率', '精确率', '召回率', 'F1 score', '训练时间']
df = pd.DataFrame(columns=col)

d1 = pd.Series({col[0]:gw1[0], col[1]:gw1[1], col[2]:gw1[2], col[3]:gw1[3],
col[4]:gw1[4], col[5]:gw1[5]}, name='随机森林算法')
d2 = pd.Series({col[0]:gw2[0], col[1]:gw2[1], col[2]:gw2[2], col[3]:gw2[3],
col[4]:gw2[4], col[5]:gw2[5]}, name='AdaBoost 提升算法')
d3 = pd.Series({col[0]:gw3[0], col[1]:gw3[1], col[2]:gw3[2], col[3]:gw3[3],
col[4]:gw3[4], col[5]:gw3[5]}, name='XGBoost 分类提升算法')

df = df.append(d1).append(d2).append(d3)
print(df)
```

4.4 实验图表产出

	最优分数	准确率	精确率	召回率	F1 score	训练时间
随机森林算法	0.99935	0.99953	0.90909	0.81081	0.85714	00:00:23.703496
AdaBoost提升算法	0.99916	0.99939	0.86364	0.77027	0.81429	00:02:08.782439
XGBoost分类提升算法	0.99956	0.99961	0.93233	0.83784	0.88256	00:02:02.965142

5 实验结论

- Bagging 与 boosting 模型都是很好的集成学习模型。
- Bagging 中的基于决策树模型的随机森林算法表现虽然不如 xgboost 算法，但强于常见的 adaboost 算法，且算法训练时间快，分类准确率较高。
- Xgboost 是集成学习算法中很强的一类算法，训练时间少于 adaboost，最优分数、准确率、精确率、召回率、f1-score 都是最高的。模型的拟合效果最好。另外 Xgboost 提供了大量可供调整的参数组合，更有利于参数调优，防止过拟合，模型的可使用性更强大。

6 参考资源

1。【机器学习】如何解决数据不平衡问题 https://www.cnblogs.com/charlotte77/p/10455900.html

2。常用的模型集成方法介绍：bagging、boosting 、stacking
https://www.jianshu.com/p/943f698c0215

3。Ensemble Learning 常见方法总结（Bagging、Boosting、Stacking、Blending）
https://blog.csdn.net/FrankieHello/article/details/81664135

4。使用 sklearn 进行集成学习——理论 https://www.cnblogs.com/jasonfreak/p/5657196.html

5. 集成学习总结 & Stacking 方法详解 https://blog.csdn.net/willduan1/article/details/73618677

6. 随机森林 sklearn FandomForest，及其调参
https://blog.csdn.net/geduo_feng/article/details/79558572

7. sklearn 集成学习 AdaBoostClassifier 参数详解
https://blog.csdn.net/JohnsonSmile/article/details/88759761

8。xgboost 入门与实战（原理篇） https://blog.csdn.net/sb19931201/article/details/52557382

9。Mac Ananconda Python 下载安装 xgboost【可参考多个报错类型】
https://blog.csdn.net/hahameier/article/details/105027178

10. xgboost 的 sklearn 接口和原生接口参数详细说明及调参指点
https://www.cnblogs.com/wzdLY/p/9831282.html

11. Scikit 中的特征选择，XGboost 进行回归预测，模型优化的实战
https://blog.csdn.net/sinat_35512245/article/details/79668363
XGboost 数据比赛实战之调参篇(完整流程)
https://blog.csdn.net/sinat_35512245/article/details/79700029

12. pandas 的 DataFrame 的 append 方法详细介绍
https://blog.csdn.net/sinat_29957455/article/details/84961936

13. 十分钟 AI 知识点】pandas 最详细教程 https://zhuanlan.zhihu.com/p/99889912