数据分析与数据挖掘第三次作业第二大题 – b问分析文档

数据分析与数据挖掘

DAM COURSE, SPRING 2018

BY

1552674 李　源

Tongji University
School of Software Engineering

1 代码运行结果

这里我参照了文档要求，选择了如下指标作为评判分类器预测结果，分别为：总体的 precision 值，recall 值以及 auc值。同时为了弥补数据集正负样本不均匀的问题，我分别采用了朴素随机欠采样和基于 SMOTE 算法的过采样，与不做正负样本平衡进行比较。

1.1 不做正负样本平衡结果

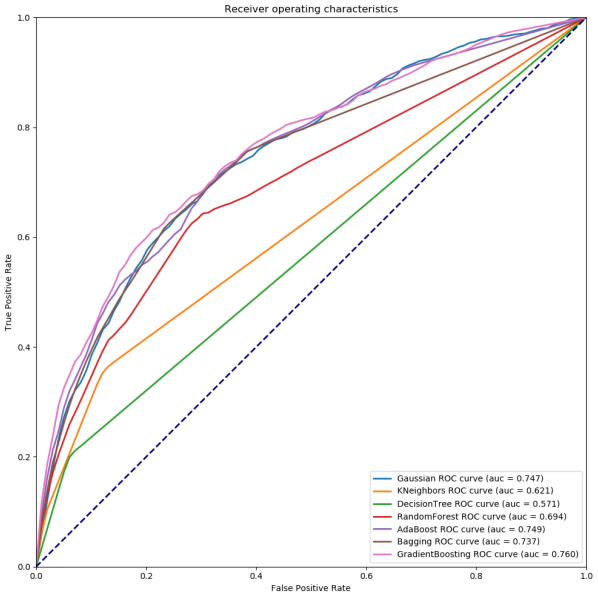
七种分类器的评判结果如下：

```
Gaussian
Overall precision: 0.187
Overall recall: 0.379
Overall auc: 0.747
**********
KNeighbors
Overall precision: 0.258
Overall recall: 0.095
Overall auc: 0.621
**********
DecisionTree
Overall precision: 0.170
Overall recall: 0.196
Overall auc: 0.571
**********
RandomForest
Overall precision: 0.300
Overall recall: 0.061
Overall auc: 0.694
**********
AdaBoost
Overall precision: 0.000
Overall recall: 0.000
Overall auc: 0.749
**********
Bagging
Overall precision: 0.334
Overall recall: 0.082
Overall auc: 0.737
**********
GradientBoosting
Overall precision: 0.000
Overall recall: 0.000
Overall auc: 0.760
**********
```

七种分类器的 precision、recall、auc 结果汇总如下（：

分类器	评价标准		
	precision	recall	auc
Gaussian	18.7%	37.9%	0.747
KNeighbors	35.8%	9.5%	0.621
DecisionTree	17.0%	19.6%	0.571
RandomForest	30.0%	6.1%	0.694
AdaBoost	0.0%	0.0%	0.749
Bagging	33.4%	8.2%	0.737
GradientBoosting	0.0%	0.0%	0.760

七种分类器的ROC曲线及AUC值如下：

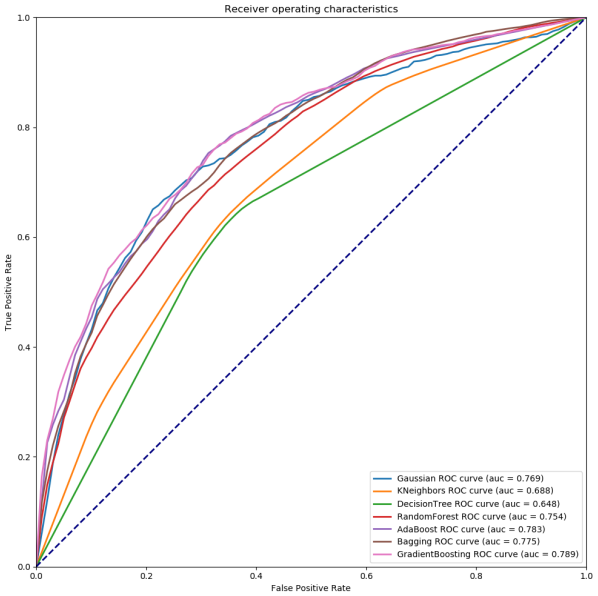


1.2 欠采样结果

七种分类器的 precision、recall、auc 结果汇总如下：

分类器	评价标准		
	precision	recall	auc
Gaussian	81.0%	42.6%	0.769
KNeighbors	65.5%	64.4%	0.688
DecisionTree	65.5%	63.3%	0.648
RandomForest	71.2%	62.4%	0.754
AdaBoost	72.1%	66.8%	0.783
Bagging	71.7%	66.8%	0.775
GradientBoosting	72.8%	68.6%	0.789

七种分类器的ROC曲线及AUC值如下：



七种分类器的评判结果如下：

```
Gaussian
Overall precision: 0.810
Overall recall: 0.426
Overall auc: 0.769
**********
KNeighbors
Overall precision: 0.655
Overall recall: 0.644
Overall auc: 0.688
**********
DecisionTree
Overall precision: 0.655
Overall recall: 0.633
Overall auc: 0.648
**********
RandomForest
Overall precision: 0.712
Overall recall: 0.624
Overall auc: 0.754
**********
AdaBoost
Overall precision: 0.721
Overall recall: 0.668
Overall auc: 0.783
**********
Bagging
Overall precision: 0.717
Overall recall: 0.672
Overall auc: 0.775
**********
GradientBoosting
Overall precision: 0.728
Overall recall: 0.686
Overall auc: 0.789
**********
```

1.3 基于SMOTE算法的过采样结果

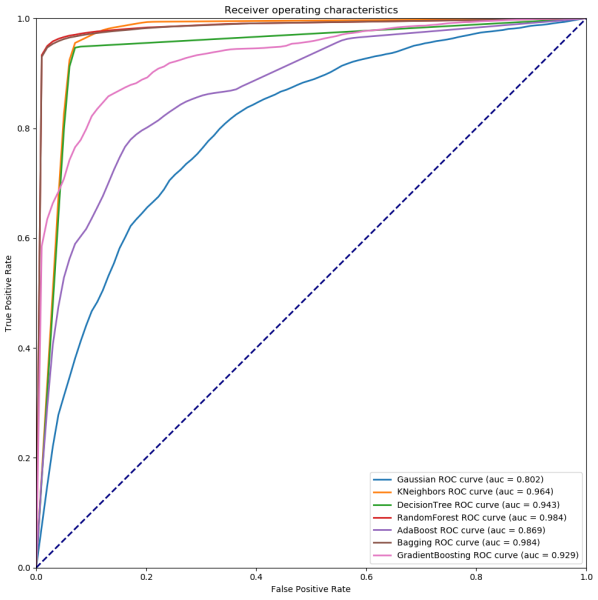
七种分类器的 precision、recall、auc 结果汇总如下：

分类器	评价标准		
	precision	recall	auc
Gaussian	81.4%	47.9%	0.802
KNeighbors	88.8%	98.1%	0.964
DecisionTree	93.9%	94.6%	0.943
RandomForest	97.8%	95.2%	0.984
AdaBoost	80.3%	79.9%	0.869
Bagging	97.3%	95.3%	0.984
GradientBoosting	88.1%	83.8%	0.929

七种分类器的评判结果如下：

```
Gaussian
Overall precision: 0.814
Overall recall: 0.479
Overall auc: 0.802
**********
KNeighbors
Overall precision: 0.888
Overall recall: 0.981
Overall auc: 0.964
**********
DecisionTree
Overall precision: 0.939
Overall recall: 0.946
Overall auc: 0.943
**********
RandomForest
Overall precision: 0.978
Overall recall: 0.952
Overall auc: 0.984
**********
AdaBoost
Overall precision: 0.803
Overall recall: 0.799
Overall auc: 0.869
**********
Bagging
Overall precision: 0.973
Overall recall: 0.953
Overall auc: 0.984
**********
GradientBoosting
Overall precision: 0.881
Overall recall: 0.838
Overall auc: 0.929
**********
```

七种分类器的ROC曲线及AUC值如下：



2 分析讨论

这里主要对特征选择、结果分析、欠采样和过采样，以及参数调节这4个方面进行分析。

2.1 特征选择

针对b问，我参考助教提供的feature_ref.pdf文件，一共抽取了如下的特征，共计32个：

特征种类	分组方法	特征含义	特征个数
TYPE.1 count/ratio - count	UI、monthly 分组	购买/被购买的次数	3
	UI、whole 分组	购买/被购买的次数	1
TYPE.1 count/ratio - product diversity	U、monthly 分组	购买的 I 的数量	3
	U、whole 分组	购买的 I 的数量	1
	U、monthly 分组	购买的 B 的数量	3
	U、whole 分组	购买的 B 的数量	1
	U、monthly 分组	购买的 C 的数量	3
	U、whole 分组	购买的 C 的数量	1
TYPE.1 count/ratio - penetration	I、monthly 分组	购买过的 U 的数量	3
	I、whole 分组	购买过的 U 的数量	1
TYPE.2 AGG feature - month AGG		monthly 特征，求AGG	12
TYPE.2 AGG feature - user AGG	I 分组	购买过的 U 的AGG	4
TYPE.2 AGG feature - B/C/I AGG	U 分组	被购买过的 I 的AGG	4

针对这32个特征，我参考论文的方法，使用 leave-out auc 和 sklearn 提供的 feature_importance_ 来判断每一个特征的效果，其结果如下：

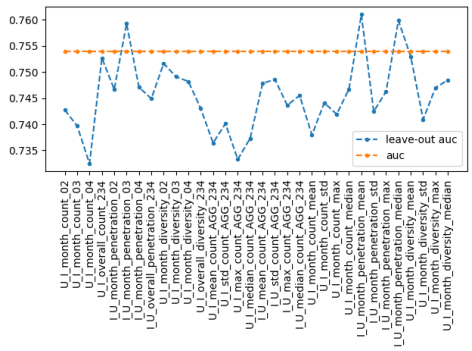


Figure 2.1: leave-out auc

而对于 sklearn 提供的 feature_importance_，其值最小的前五特征如下：

特征名称	feature_importance_
U_I_median_count_AGG_234	0.001
I_U_median_count_AGG_234	0.004
U_I_month_count_max	0.006
U_I_month_count_median	0.008
U_I_month_count_std	0.010

结合两个评价标准，我发现 TYPE.2 类中的涉及到 meidan 和 max 相关的特征对于分类效果的贡献很小，个别的特征甚至其 leave-out auc 是负值，说明其可能会导致分类的效果更差。而对于 TYPE.1 中的特征，其 leave-out auc 降低较多，说明其特征中涉及到的信息对于分类器的影响很大，包含了很多分类器所必需的信息。

实际上，我也测试过加入 TYPE.3 和 TYPE.4 的部分特征，但是其结果不尽人意，，添加后计算得到的 auc 值 分别为 0.721 和 0.732（进行了朴素随机欠采样，分类器为 RandomForestClassifier），明显是低于原始的 0.755。我的一个想法是，后两者特征，更多地是考虑了单一偏向 vipno 或者 pluno 的方面，而没有考虑到两者结合后的情况。

2.2 Adaboost 和 GradientBoosting 的参数调节

sklearn 中的 sklearn.model_selection 包提供了 GridSearchCV 方法来尝试找出最优的参数。因为这是一个二分类问题，所以我使用 roc_auc 来作为不同参数的评判标准，并且进行 5-fold 交叉验证。

针对 Adaboost 分类器，我考虑了2个参数，分别是弱分类器的个数、步长。其待选择的参数如下：721

```
param_test1 = {'n_estimators': range(30, 101, 10),
               'learning_rate': np.arange(0.01, 0.1, 10)}
```

Figure 2.2: Adaboost 待选择参数

最后选择出的结果如下：

```
Best param: {'learning_rate': 0.01, 'n_estimators': 100}
Best score: 0.6591674318947045
```

Figure 2.3: 弱分类器的个数、步长结果

针对 GradientBoosting 分类器，考虑到了弱分类器的个数、步长、决策树最大深度、最大特征数以及子采样的比例。其待选择的参数如下：

```
param_test1 = {'n_estimators': range(10, 61, 10),
               'learning_rate': np.arange(0.01, 0.1, 10)}
param_test2 = {'max_depth': range(3, 14, 2)}
param_test3 = {'max_features': range(7, 20, 2),
               'subsample': [0.6, 0.7, 0.75, 0.8, 0.85, 0.9]}
```

Figure 2.4: GradientBoosting 待选择参数

最后选择出的结果如下：

```
Best param: {'learning_rate': 0.01, 'n_estimators': 60}
Best score: 0.754463830221406
```

Figure 2.5: 弱分类器的个数、步长结果

```
Best param: {'max_depth': 3}
Best score: 0.7608917457402306
```

Figure 2.6: 决策树最大深度结果

之后我将选择出的参数设置为对应分类器的参数，进行了预测，可以发现 Adaboost 和 GradientBoosting 分类器均比使用默认参数取得了一定的提升。

```
Best param: {'max_features': 19, 'subsample': 0.85}
Best score: 0.7814304662789511
```

Figure 2.7: 最大特征数、子采样的比例结果

2.3 预测结果分析

这里的分析主要是针对三种评价标准的评估，以及七种分类器最终结果的分析。

2.3.1 三种评价标准

针对本次问题，我一共选择了三种指标作为评价标准，分别是 precision、recall、auc。

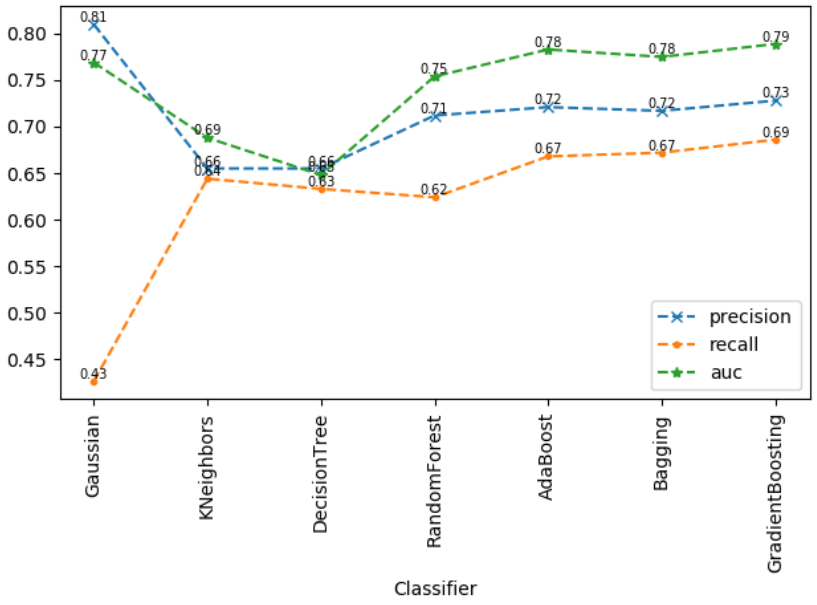


Figure 2.8: 评价结果（进行欠采样后）

这里我们可以注意到，针对 Gaussian 分类器，在 precision 的得分上和另外两种分类器的差异较大，对于 Gaussian 分类器，其在 precision 上的得分很差，为 42.6%，而 auc 则是达到七种分类器的最高值 0.810。

为什么会出现这样的情况呢？因为本问题实际上考虑的是用户是否再次购买某商品的情况。我查找了网上的文章，发现这类问题被称作 — “复购问题”，在 Kaggle 竞赛网站上曾经有一个这样的比赛 Instacart Market Basket Analysis（https://www.kaggle.com/c/instacart-market-basket-analysis）。

针对复购问题，会存在一个很经典的现象：样本的分布不均匀。通常来说，绝大多数的商品都会存在于混淆矩阵的 TN 部分，即对于某一个（vipno，pluno）对，在后一月并不会出现，而算法预测的结果也是其不会出现。

对于 precision 来说，其计算公式为 $pre = TP/(TP + FP)$，表现了正确预测为正占全部预测为正的比例。如果我仅仅考虑 precision，则可能出现的情况是我的算法会趋向于保守，那么这样得到的结果就可能导致最后算法预测为正的结果很少，那么最后决定实际推广的商品也会过少，可能会导致覆盖面不广的情况。

而实际上，recall 在这里也不是一个很好的评价指标，因为 $recall = TP/(TP + FN)$，如果我们尝试去提升 recall 的值，那么就会导致我的算法过于冒险，想尽可能多地去找出真正复购了的商品，那么这样最后就会导致推广的商品过多，覆盖面过广。

（这一部分结论参考了这篇博文：https://reata.github.io/kaggle/classification/kaggle-instacart-market-basket-analysis-retrospect）

博客中提出使用 f1-score 来评价，我这里运用了另一个评价指标 ROC 曲线下的面积 auc。对于 ROC 曲线，其是基于 sensitivity 和 specificity 来计算的，而这两个指标是条件概率，其不会被正样本的真实概率所影响到，也就是说，他们和做出的 ROC 曲线不会受到样本分布的影响，这样相对会更加客观。

因此我在选择时，主要考虑出的是 auc 指标，同时在一定程度上考虑 precision 和 recall 值。比较三种处理数据集后的结果，可以发现 Bagging 取得了较好的效果，针对原始数据集，其 auc 为第三，但是 precision 为第一，远高于 auc 前二的 Adaboost 和 GradientBoosting 分类器。而针对利用 SMOTE 过采样后的结果，其在三个评价结果上均取得了第一的排名。

2.4 欠采样和过采样

前面已经提到了，本次的问题存在一个很典型的现象：样本分布不均匀。而我们所更关心的要预测事件，是属于少数类别的，即真正复购了的商品。当面临不平衡的数据集的时候，机器学习算法倾向于产生不太令人满意的分类器。

不平衡的数据集会影响到分类器的性能，比如说在本次中，DecisionTree 的效果不好，因为它会偏向于数量多的类别，而对于那些少数类，可能会被作为噪音而被忽略。因此其少数类可能会存在较高的误判率，参考前面的结果，其 precision 是最低的。

因此我在这里利用了欠采样和过采样的方法分别对数据集进行了修正。其中欠采样我使用的是朴素随机欠采样，过采样我使用的是基于 SMOTE 算法的合成少数类过采样。

2.4.1 朴素随机欠采样（Random Under-Sampling）

朴素随机欠采样的做法是，通过随机地消除多数类样本的实例，从而最后达到多数类和少数类的平衡。这样做的好处一是能够平衡正负样本的数量，并且减少数据集后，可以提升运行的性能。

但是其缺点也是明显的，因为对于它所丢弃的多数类样本，可能会包含一些很重要的潜在信息，并且选择出来的样本可能会有一定的偏差存在，从而最终导致分类器在实际的测试集上效果较差。

2.4.2 合成少数类过采样（SMOTE）

SMOTE 算法是提取了少数类中的一个数据子集作为一个实例，然后来模拟生成相似的实例。我在这里利用了 imblearn 这个包中的 SMOTE 算法来实现。因为它不同于朴素随机过采样，其生成的新样本不是原来样本的副本，这样能够在一定程度上缓解过拟合的问题。同时相对于朴素随机欠采样，SMOTE 没有丢失一些有价值的信息。

最后的结果可以参考前面的截图和图表，我发现在使用了欠采样和过采样后，对于所有的分类器的分类结果都有了一定程度上的提升。而过采样的提升效果更好。对于这样的结果，我认为是，本次的原始数据集的数量为6000余条，与欠采样减少数据集相比，过采样是提升了数据集的数量，而且这个提升并没达到可能会过拟合的程度，从而达到了更好的效果。

3 性能分析

这里简单地对七种分类器的运行时间进行了一个比较，其结果如下：

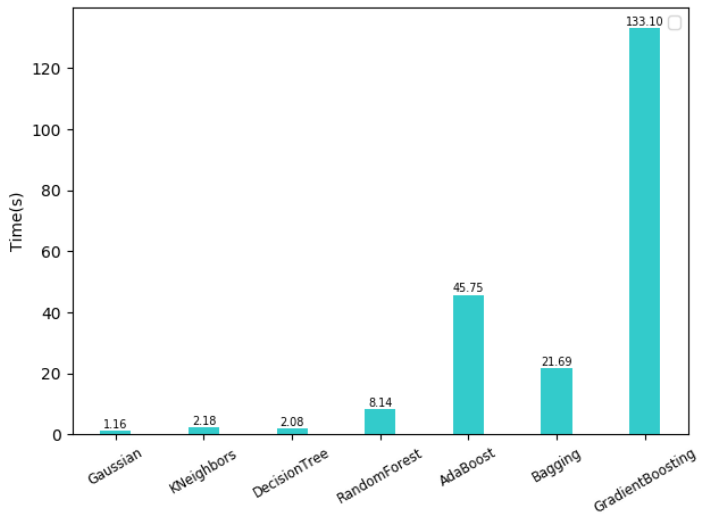


Figure 3.1: 运行时间

Adaboost 和 GradientBoosting 分类器消耗的时间最大，而较为简单的弱分类器的时间要明显短于复合分类器。不过 Adaboost 和 GradientBoosting 分类器的分类效果要优于弱分类器，其付出的时间消耗在一定程度上是值得的。