

银行信贷违约检测 对不平衡数据集分类问题的研究

沈祥壮，田瑶，涂庆

中山大学

2019 年 12 月 8 日

数据格式

1 数据与问题描述

- 数据格式
- 问题描述

2 预处理与特征工程

- 预处理
- 分类型特征
- 数值型数据
- 隐私数据

3 算法

- XGBoost
- EasyEnsemble
- BalanceCascade

4 实验设计与结果

- 运行环境与流程
- 结果展示

5 编程工具分享

数据格式

厦门国际银行信贷数据

数据来源于 2019 厦门国际银行“数创金融杯”数据建模大赛
训练集 132029 条，测试集 23561 条。

用户基本信息		借贷相关信息		隐私数据
id	用户唯一标识	loanProduct	产品类型	ncloseCrediCard
target	违约为 1	lmt	预授信金额	unpayIndvLoad
certId	证件号	basicLevel	基础评级	unpayOtherLoan
gender	性别	bankCard	放款卡号	unpayNormalLoan
age	年龄	residentAddr	居住地	5yearBadloan
dist	地区	linkRela	联系人关系	x_0
edu	学历	setupHour	申请时段	x_1
job	单位类型	weekday	申请日（周几）	x_2
ethnic	民族	isNew	是否新增数据	...
highestEdu	最高学历			x_76
certValidBegin	证件号起始日期			x_77
certValidStop	证件号失效日期			x_78

问题描述

高维稀疏

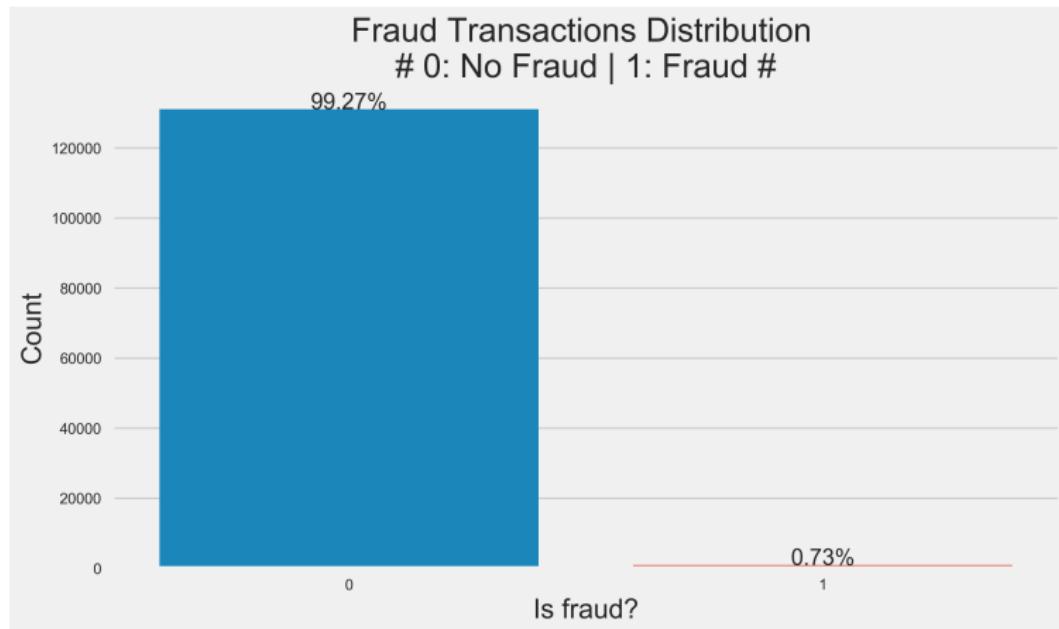
含有大量的类别特征，在进行编码 (One-Hot) 后，数据呈现出高维和稀疏的性质。

问题描述

极端不平衡数据的分类

设违约数据为正样本，记为 P ；不违约为负样本，记为 N 。

本数据集的正负样本比为 $\frac{|N|}{|P|} \approx 136$



预处理

1 数据与问题描述

- 数据格式
- 问题描述

2 预处理与特征工程

- 预处理
- 分类型特征
- 数值型数据
- 隐私数据

3 算法

- XGBoost
- EasyEnsemble
- BalanceCascade

4 实验设计与结果

- 运行环境与流程
- 结果展示

5 编程工具分享

预处理

重复值与缺失值

- 无重复数据。
- 缺失值分为两种：NA 与-999。
 - 仅 “bankCard” 字段含有 NA 型缺失值，而此特征对于分类并无太大作用，故去除。
 - 对于表现为-999 的缺失值，我们在特征工程时分别进行不同的处理。

分类型特征

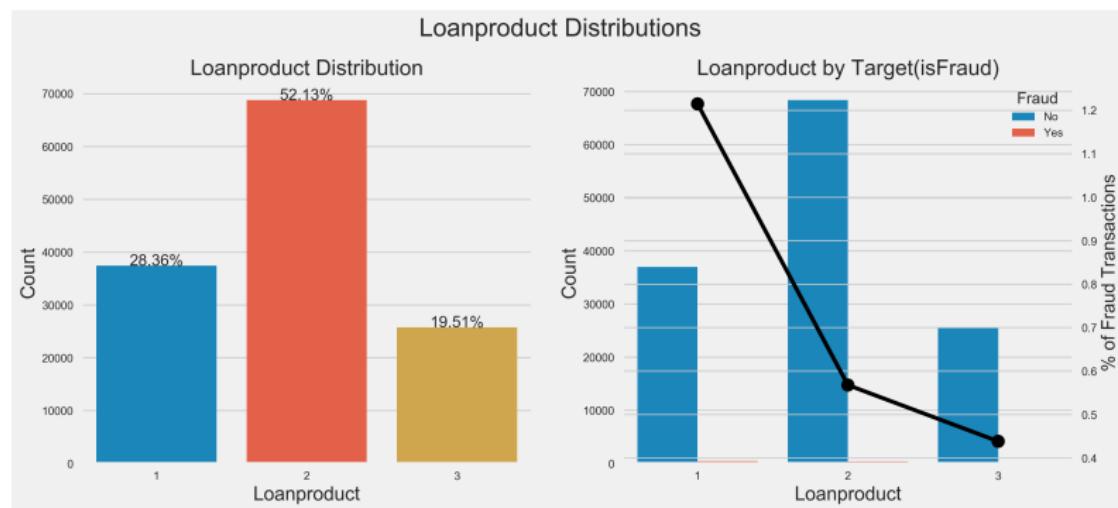
删除或重新编码

- 删除无关特征：id, certId, dist, residentAddr, isNew
- 对其他分类特征进行重新编码：
 - 直接重新编码：loanProduct, job, linkRela, ncloseCreditCard, unpayIndvLoan, unpayOtherLoan, unpayNormalLoan, 5yearBadloan
 - 转换后编码：basicLevel, setupHour, weekday, ethnic

分类型特征

直接重新编码

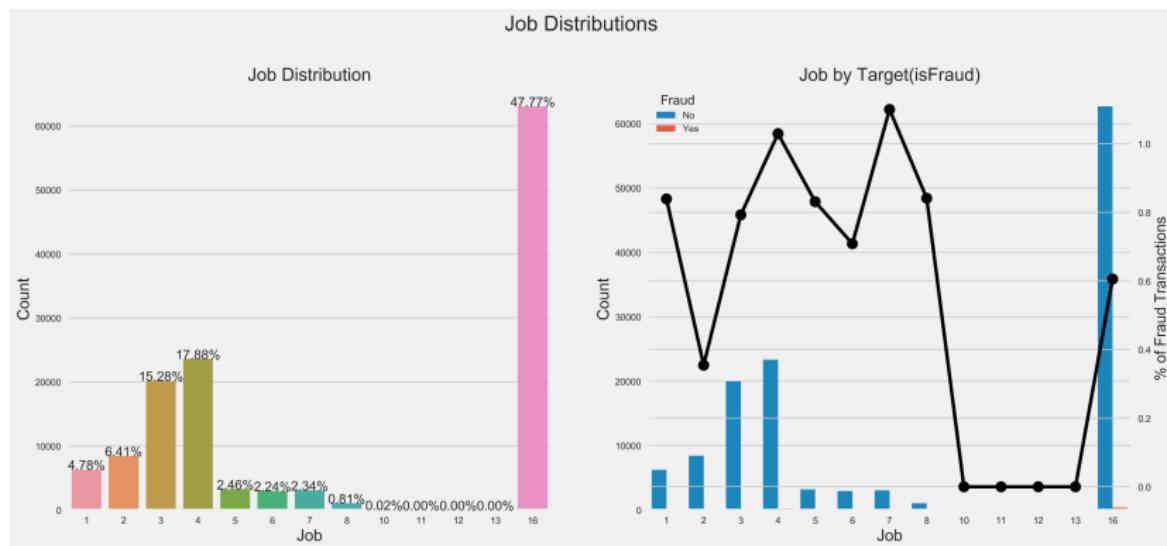
部分类别变量的不同值对应的欺诈率有较为明显的区别。



分类型特征

直接重新编码

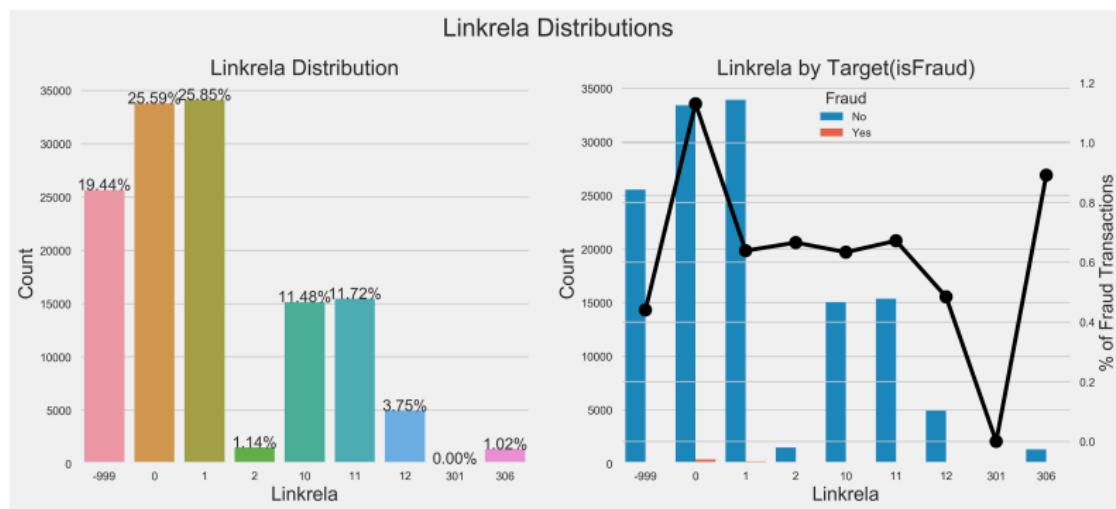
部分类别变量的不同值对应的欺诈率有较为明显的区别。



分类型特征

直接重新编码

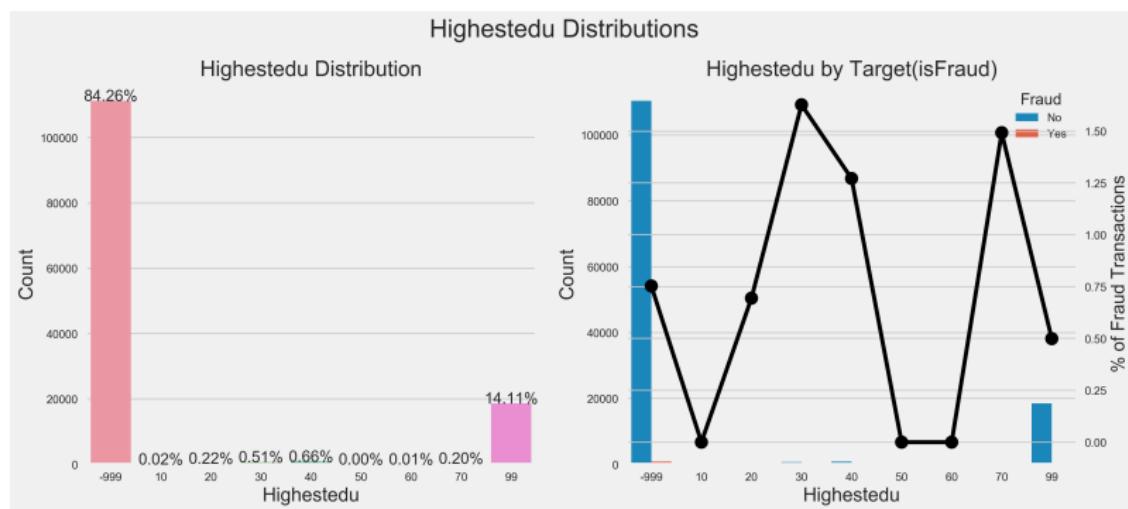
部分类别变量的不同值对应的欺诈率有较为明显的区别。



分类型特征

直接重新编码

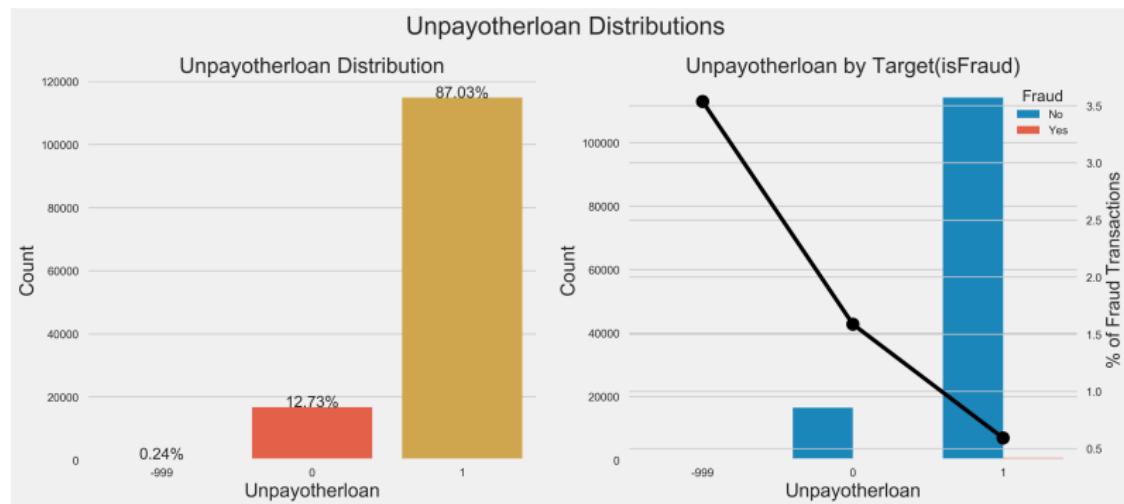
部分类别变量的不同值对应的欺诈率有较为明显的区别。



分类型特征

直接重新编码

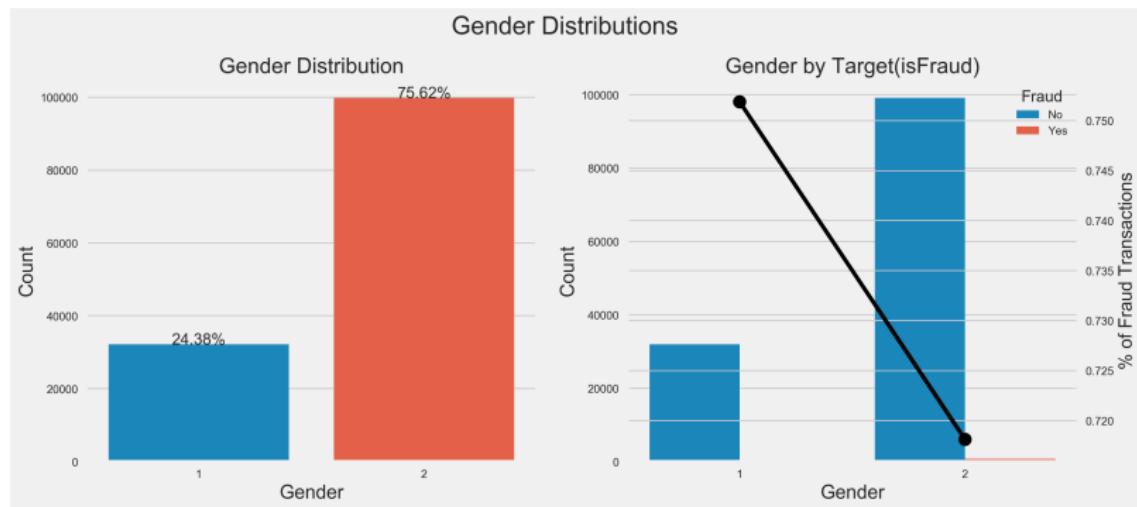
注意这类特征，其缺失的数据均有较高的欺诈率，所以我们
将-999 编码为一个新的类别。



分类型特征

直接重新编码

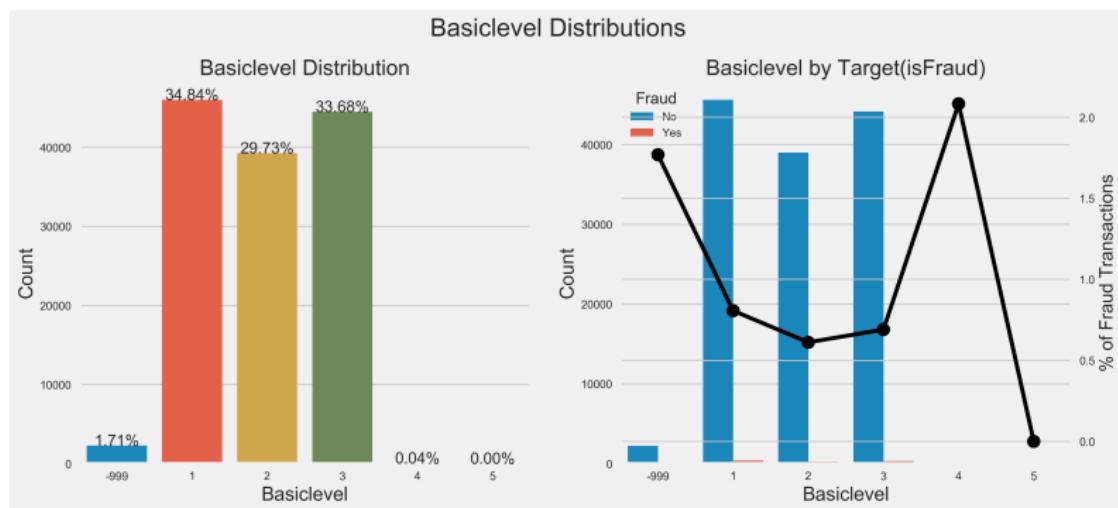
而有些变量区分则不明显，即重叠比较严重。（这里采取保守策略，即保留那些区分度不高的特征）



分类型特征

变换后编码

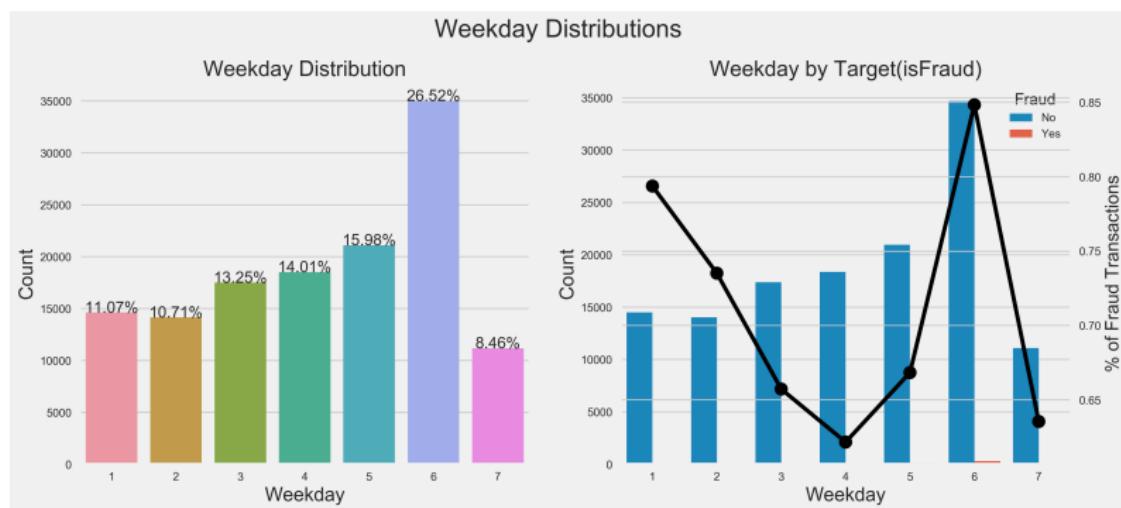
缺失值 (-999) 和 4 编码为 1, 其他编码为 0 (尽可能降低维度)



分类型特征

变换后编码

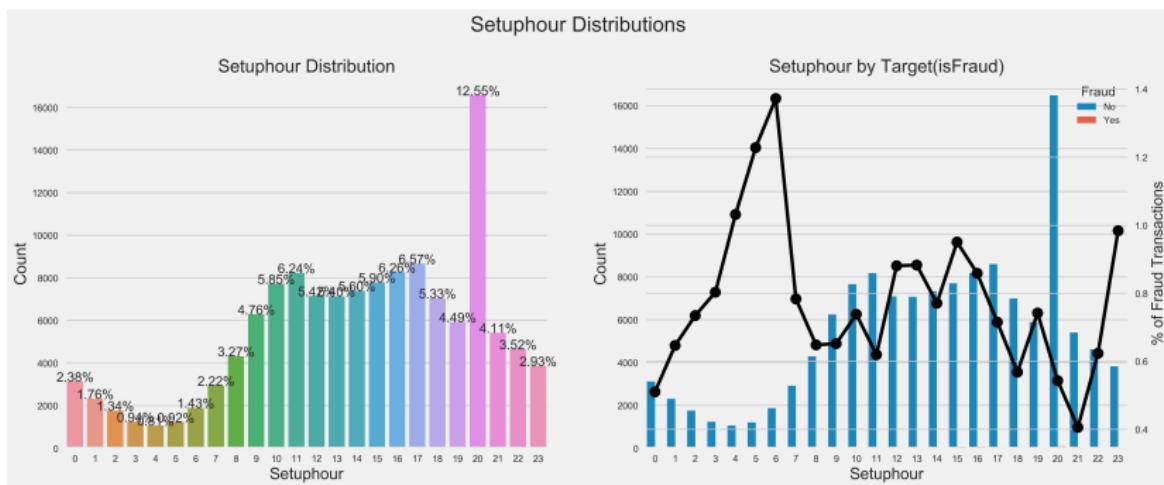
时间分段编码，周一周二交易量较少，诈骗率较高



分类型特征

变换后编码

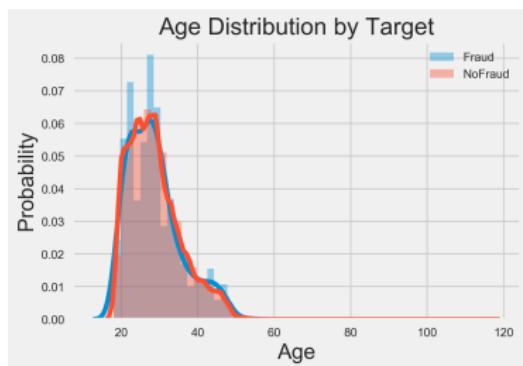
时间分段编码，晚九点到第二天早九点交易量较少，诈骗率较高



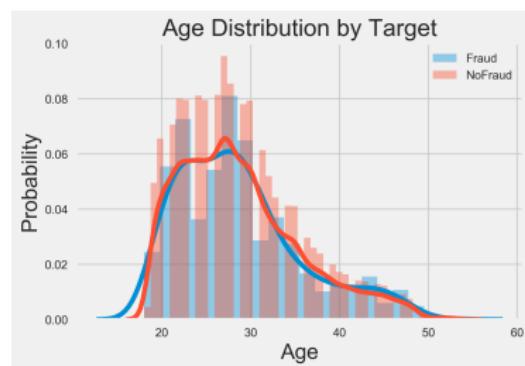
数值型数据

去除极端值后标准化

一般的直接标准化，特殊的去除 N（多数类）中的极端数据再标准化



图：去除极端值前



图：去除极端值后

数值型数据

DATA LEAKAGE!

- ① $(\text{allData} - \text{mean}(\text{allData})) / \text{sd}(\text{allData})$
- ② $(\text{trainData} - \text{mean}(\text{trainData})) / \text{sd}(\text{trainData})$
 $(\text{testData} - \text{mean}(\text{testData})) / \text{sd}(\text{testData})$
- ③ $(\text{trainData} - \text{mean}(\text{trainData})) / \text{sd}(\text{trainData})$
 $(\text{testData} - \text{mean}(\text{trainData})) / \text{sd}(\text{trainData})$

Data leakage is not just about leaking y into your predictions, it is also about leaking information from the future.(stackexchange)

Target leakage

参考 Hastie, ESL7.10 Cross-Validation,CatBoost(Liudmila, NIPS 2018)

Leaking information from the future

参考 <https://stats.stackexchange.com/questions/174823>

隐私数据

保持不变

对变量 x_0 到 x_{78} , 我们没有任何已知的信息, 保持不变。

1 数据与问题描述

- 数据格式
- 问题描述

2 预处理与特征工程

- 预处理
- 分类型特征
- 数值型数据
- 隐私数据

3 算法

- XGBoost
- EasyEnsemble
- BalanceCascade

4 实验设计与结果

- 运行环境与流程
- 结果展示

5 编程工具分享

XGBoost

XGBoost 对不平衡数据的处理

- 通过 XGBoost 源码：

`if (info.labels[i] == 1.0f) w *= param_.scale_pos_weight` 可以看到 XGBoost 通过加强正样本梯度的影响，即通过代价敏感学习来处理样本不平衡问题。

- 设置 `scale_pos_weight` 与 `sample_weight` 来执行两种不同形式的代价敏感学习。
- 这里选用 `scale_pos_weight`，参数设置为 $\frac{|N|}{|P|} \approx 136$

Unsupervised Downampling

- 简单，并行支持好。
- N 中部分数据信息为充分利用，可能欠拟合。

Algorithm 1 The EasyEnsemble algorithm.

- 1: {Input: A set of minority class examples \mathcal{P} , a set of majority class examples \mathcal{N} , $|\mathcal{P}| < |\mathcal{N}|$, the number of subsets T to sample from \mathcal{N} , and s_i , the number of iterations to train an AdaBoost ensemble H_i }
- 2: $i \Leftarrow 0$
- 3: **repeat**
- 4: $i \Leftarrow i + 1$
- 5: Randomly sample a subset \mathcal{N}_i from \mathcal{N} , $|\mathcal{N}_i| = |\mathcal{P}|$.
- 6: Learn H_i using \mathcal{P} and \mathcal{N}_i . H_i is an AdaBoost ensemble with s_i weak classifiers $h_{i,j}$ and corresponding weights $\alpha_{i,j}$. The ensemble's threshold is θ_i , i.e.

$$H_i(x) = \text{sgn} \left(\sum_{j=1}^{s_i} \alpha_{i,j} h_{i,j}(x) - \theta_i \right).$$
- 7: **until** $i = T$
- 8: Output: An ensemble:

$$H(x) = \text{sgn} \left(\sum_{i=1}^T \sum_{j=1}^{s_i} \alpha_{i,j} h_{i,j}(x) - \sum_{i=1}^T \theta_i \right).$$

BalanceCascade

Supervised Downsampling

- 参与算法的数据量递减，算法用时短，适用于大规模数据集
- 类似 Boosting 的 Supervised Downsampling，使得算法容易过拟合。

Algorithm 2 The BalanceCascade algorithm.

- 1: {Input: A set of minority class examples \mathcal{P} , a set of majority class examples \mathcal{N} , $|\mathcal{P}| < |\mathcal{N}|$, the number of subsets T to sample from \mathcal{N} , and s_i , the number of iterations to train an AdaBoost ensemble H_i }
 - 2: $i \Leftarrow 0$, $f \Leftarrow \sqrt[T-1]{\frac{|\mathcal{P}|}{|\mathcal{N}|}}$, f is the false positive rate (the error rate of misclassifying a majority class example to the minority class) that H_i should achieve.
 - 3: **repeat**
 - 4: $i \Leftarrow i + 1$
 - 5: Randomly sample a subset \mathcal{N}_i from \mathcal{N} , $|\mathcal{N}_i| = |\mathcal{P}|$.
 - 6: Learn H_i using \mathcal{P} and \mathcal{N}_i . H_i is an AdaBoost ensemble with s_i weak classifiers $h_{i,j}$ and corresponding weights $\alpha_{i,j}$. The ensemble's threshold is θ_i i.e.

$$H_i(x) = \text{sgn} \left(\sum_{j=1}^{s_i} \alpha_{i,j} h_{i,j}(x) - \theta_i \right).$$
 - 7: Adjust θ_i such that H_i 's false positive rate is f .
 - 8: Remove from \mathcal{N} all examples that are correctly classified by H_i .
 - 9: **until** $i = T$
 - 10: Output: A single ensemble:

$$H(x) = \text{sgn} \left(\sum_{i=1}^T \sum_{j=1}^{s_i} \alpha_{i,j} h_{i,j}(x) - \sum_{i=1}^T \theta_i \right).$$
-

运行环境与流程

1 数据与问题描述

- 数据格式
- 问题描述

2 预处理与特征工程

- 预处理
- 分类型特征
- 数值型数据
- 隐私数据

3 算法

- XGBoost
- EasyEnsemble
- BalanceCascade

4 实验设计与结果

- 运行环境与流程
- 结果展示

5 编程工具分享

运行环境与流程

运行环境 [Kaggle kernel]

- system: Linux
- CORES: 4
- RAM: 18.628563 GB
- release : 4.9.0-11-amd64
- machine : x86_64
- brand: Intel(R) Xeon(R) CPU @ 2.30GHz
- hz_advertised: 2.3000 GHz
- hz_actual: 2.3000 GHz

运行环境与流程

实验流程

对上述三种算法均采用下述流程进行实现：

- ① 对训练集分割，80% 用于训练，20% 验证集
- ② 使用 5 折交叉验证，调参
- ③ 选取最优参数，对全部的训练集进行训练
- ④ 对测试集预测，线上提交 (AUC 衡量)

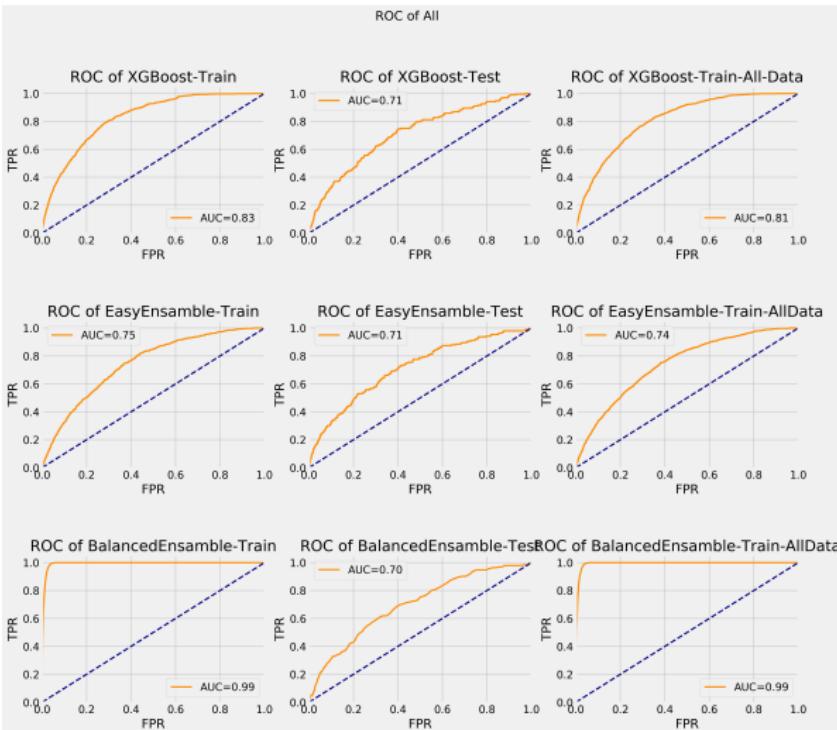
运行环境与流程

Why AUC?

- ① ROC 下面积
- ② 随机从正类和负类中各抽取一个样本，算法给正类的 score 大于负类的概率
- ③ 正负类配对样本（样本的笛卡尔积，共 $|N||P|$ 对）中，正类的 score 大于负类的概率。而 $AUC * |N||P|$ 即为 Mann-Whitney U 统计量。

结果展示

150-Estimator-ROC



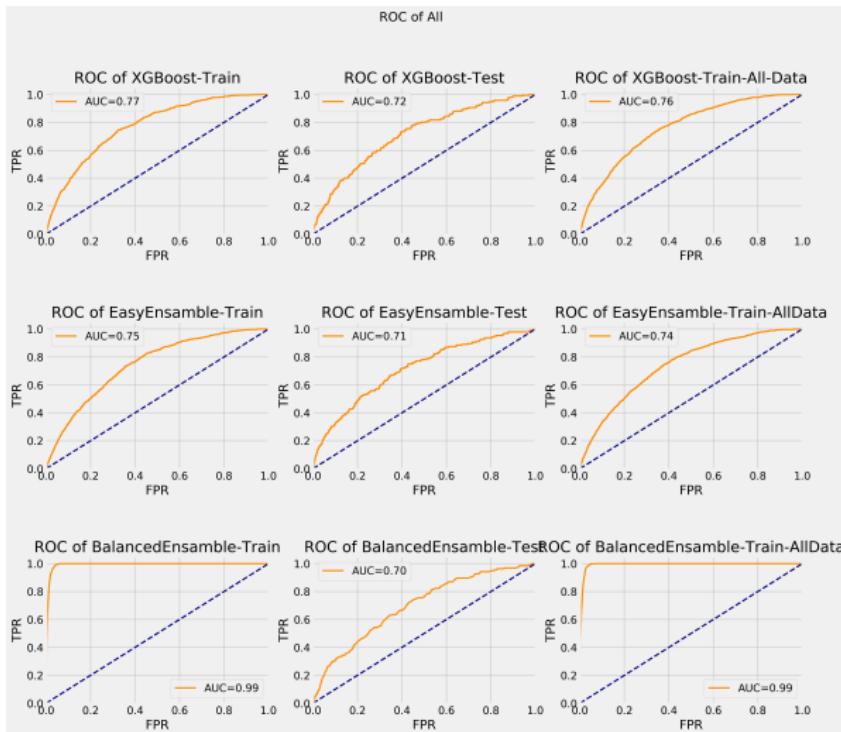
结果展示

150-Estimator-Time

Time(seconds)	Search	Train	Prediction
XGBoost	306.152	20.553	0.16
EasyEnsamble	1633.848	62.603	29.182
BalancedCascade	1166.192	81.818	1.145

结果展示

300-Estimator-ROC



结果展示

300-Estimator-Time

Time(seconds)	Search	Train	Prediction
XGBoost	1512.681	22.744	0.158
EasyEnsamble	3635.758	142.838	64.232
BalancedCascade	2764.941	194.82	2.833

结果展示

线上 10.65%

XGBoost 与 EasyEnsamble 线上 AUC 均能达到 0.77
(EasyEnsamble 略高), BalancedCascade 达到 0.76。此外榜首
AUC 为 0.79。

结果展示

实验结果分析

准确率分析

- ① XGBoost，在 150 个基分类器的时候表现出较强的过拟合现象，主要是由于其 Boosting 的机制存在 **target leakage** (Liudmila, NIPS 2018)。在增加基分类器的数量后，由于 XGBoost 的行列采样机制引入更多随机性，降低了过拟合。此外，由于代价敏感的引入，使得其对于不平衡数据集也有良好的表现。
- ② EasyEnsamble 采用**无监督的降采样**，很好地平衡了数据集，且由于其对负样本的采样是随机的，所以基本不存在过拟合问题。
- ③ BalancedCascade 由于其采用**有监督的降采样**，容易引起过拟合。因为中途被剔除的数据可能也是有用的 (F.-Z. Marcos, 2004)。这里在极度不平衡数据集的表现并非如论文 (Xu-Ying Liu, 2008) 所说那么好。

结果展示

实验结果分析

运行效率分析

- ① XGBoost 运行效率的提升来自两方面，分别是算法理论和工程实现。算法理论上主要是 Column Subsampling, Histogram-based Algorithm(特征分割点-> 分割区间) 以及 Sparsity-aware Split Finding，减少了计算量。更主要的是工程上的实现带来了效率提升，底层由 C 和 C++ 实现，Cache-aware block structure + Out-of-core tree learning 且支持并行（如寻找最优分割时）。
- ② EasyEnsamble 较慢的原因主要是工程实现。因为所用库 imblearn 由 Python 实现，虽然对并行支持很好，也是无法弥补与 C++ 效率上的差距。
- ③ BalancedCascade，也是用库 imblearn，所以总体也是偏慢。但是由于其剔除样本的性质，所以其对特大数据集的效率会比 EasyEnsamble 高。

1 数据与问题描述

- 数据格式
- 问题描述

2 预处理与特征工程

- 预处理
- 分类型特征
- 数值型数据
- 隐私数据

3 算法

- XGBoost
- EasyEnsemble
- BalanceCascade

4 实验设计与结果

- 运行环境与流程
- 结果展示

5 编程工具分享

Jupyter 插件

https://github.com/ipython-contrib/jupyter_contrib_nbextensions

The screenshot shows a Jupyter Notebook interface with the following details:

- Header:** File, Edit, View, Insert, Cell, Kernel, Navigate, Widgets, Help, Kernel Python 3.
- Left Sidebar (Contents):** A tree view of available notebooks:
 - 模型拟合 [0.774493]
 - 简单尝试
 - 调参
 - Easy Ensemble Classifier[0.76]
 - 简单尝试
 - 调参
 - Balanced Bagging Classifier
 - 简单尝试
 - 调参
- Cell 1 (In [1]):** Displays Python code for model selection and evaluation. It imports various sklearn modules and imblearn.ensemble classifiers. The code is as follows:

```
# import lib
import pandas as pd
import numpy as np
import xgboost as xgb
from sklearn.model_selection import train_test_split, StratifiedKFold, RandomizedSearchCV
from sklearn.decomposition import PCA
from sklearn.metrics import roc_auc_score, confusion_matrix, roc_curve, auc
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import balanced_accuracy_score
from sklearn.naive_bayes import BernoulliNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn import svm
from imblearn.ensemble import EasyEnsembleClassifier, BalancedBaggingClassifier
```

Output: executed in 2m 24s, finished 00:59:50 2019-11-23
Using TensorFlow backend.
- Cell 2 (In [2]):** Displays Python code for importing matplotlib and setting its style. The code is as follows:

```
import matplotlib.pyplot as plt
import matplotlib as mpl
mpl.style.use('fivethirtyeight')
```

Output: executed in 3.29s, finished 00:59:53 2019-11-23
- Cell 3 (In [3]):** Displays a partially visible line of code starting with "from sklearn.model_selection import KFold".

算法计时器:algotimer



算法计时器:algotimer

```
1 from algotimer import Timer, TimerPloter
2 ...
3 with Timer('GaussianNB, Train'):
4     gnb = GaussianNB()
5     clf = gnb.fit(iris.data, iris.target)
6
7 with Timer('KNN(K=3), Test'):
8     neigh = KNeighborsClassifier(n_neighbors=3)
9     clf = neigh.fit(iris.data, iris.target)
10 ...
11 ploter = TimerPloter()
12 ploter.plot()
```

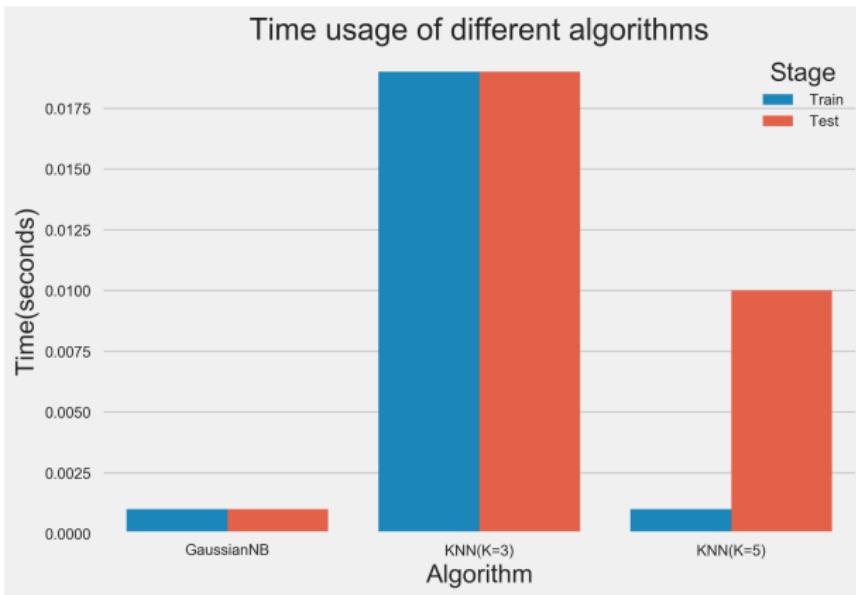
算法计时器:algotimer

得到 logging.csv 文件

```
1 GaussianNB , Train , 0.001
2 GaussianNB , Test , 0.001
3 KNN(K=3) , Train , 0.019
4 KNN(K=3) , Test , 0.019
5 KNN(K=5) , Train , 0.001
6 KNN(K=5) , Test , 0.01
```

算法计时器:algotimer

TimerPloter 做图



总结

已完成

数据的预处理，探索性数据分析，使用三种算法分别从算法层面（代价敏感学习）和数据层面（降采样）对不平衡数据集进行分类。对三种算法的时间效率和准确率做了水平对比，并对实验结果进行了较为深入的分析。

失败的尝试

- ① 尝试采用 PCA 降维，结果更差。分析原因可能如 lightGBM(Microsoft Research, NIPS 2017) 所说：*These approaches highly rely on the assumption that features contain significant redundancy*
- ② 尝试采用有监督和无监督的特征选择方法，结果变差。如去除低方差特征，基于决策树的特征选择等。分析可能与数据类别不平衡有关，其影响正常的特征选择。

数据与问题描述
○○○

预处理与特征工程
○○○○○○○○○○○○○○

算法
○○○

实验设计与结果
○○○○○○○○○○

编程工具分享
○○○○○●

Thanks