Mini Project

CSE5002 智能数据分析

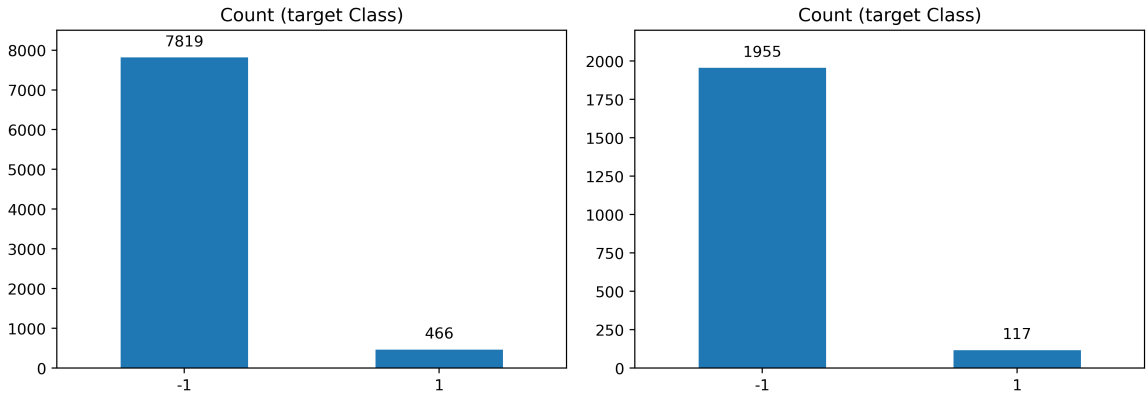
姓名: 李子强
学号: 11930674

2020 年 6 月 8 日

目 录

1 数据预处理

1.1 数据描述



(a) 训练集　　(b) 测试集

图 1: 标签分布

如图1所示，这是一个二分类问题。在训练集中，负类（7819）和正类（466）比例悬殊，比例达 16.78 比 1。如果直接全部标记为负类准确度也能达到 94.38%。所以对数据集采样的时候，一定需要考虑样本量平衡处理的问题。

1.2 特征相关性分析

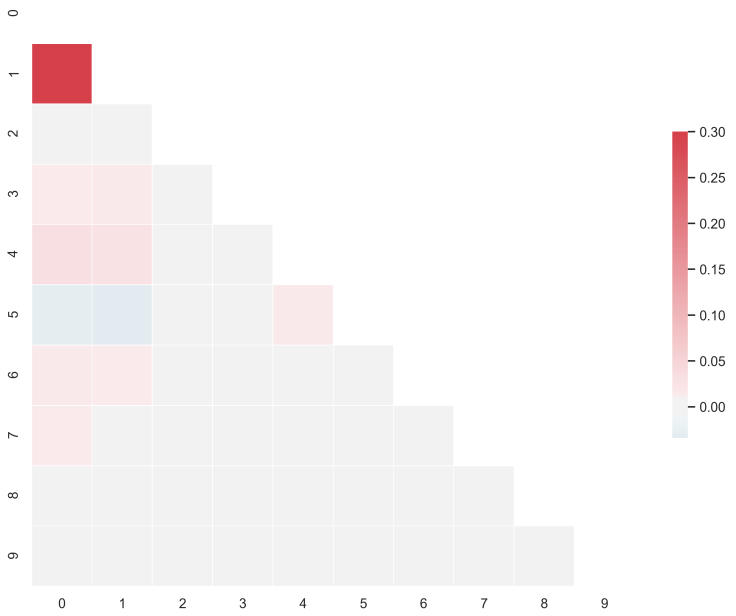


图 2: 训练集的相关性矩阵

数据集共有 10 个维度，使用皮尔逊相关系数分析两两特征之间的相关关系，画出图2。皮尔逊相关系数绝对值趋近于 0，两个随机变量趋于无关；绝对值趋近于 1 趋于相关。相关系数大于 0 为正相关，小于 0 为负相关。可以看出比较多特征之间是没有关系的，但是，第 0 维和第 1 维是强相关。

1.3 采样

通常非平衡数据的处理方式是欠采样（under-sampling）和过采样（oversampling）。欠采样是丢弃多数类的数据，因为只学习了一部分数据，可能会造成偏差增加，造成过拟合。过采样是重复或者合成少数类数据，实际上没有为模型引入更多数据，过分强调少数类数据，会放大少数类噪音对模型的影响。

SMOTE (Synthetic Minority Oversampling Technique) 是 2002 年 Chawla 等人提出的一种在随机采样的基础上改进的一种过采样算法 [1]。基本思想就是对少数类别样本进行分析和模拟，并将人工模拟的新样本添加到数据集中，进而使原始数据中的类别不再严重失衡。该算法的模拟过程采用了 K 最近邻 (kNN，k-NearestNeighbor)，模拟生成新样本的步骤：

1. 采样最邻近算法，计算出每个少数类样本 $\tilde{x}$ 的 K 个近邻 $\tilde{x}_i$;
2. 从 K 个近邻中随机挑选 N 个样本进行随机线性插值;
3. 构造新的少数类样本 $x_{new} = x + rand\,(0,1) \times (\tilde{x} - \tilde{x}_i)$;
4. 将新样本与原数据合成，产生新的数据集;

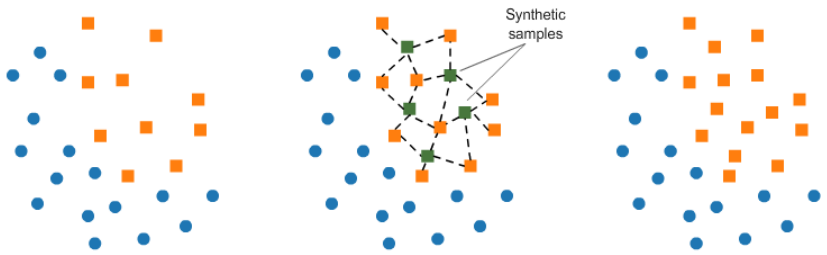


图 3: SMOTE 生成示例

除此之外，还有由 SMOTE 衍生出来的变体算法，比如：Border-line SMOTE。

另外，ADASYN (Adaptive Synthetic) 是 2008 年 He 等人提出的一种自适应综合过采样方法 [4]。其最大的特点是采用某种机制自动决定每个少数类样本需要产生多少合成样本，而不是像 SMOTE 那样对每个少数类样本合成同数量的样本。具体流程如下：

首先计算需要合成的样本总量：

$$G = (S_{maj} - S_{min}) \times \beta$$

其中 S_{maj} 为多数类样本数量，S_{min} 为少数类样本数量，$\beta \in [0,1]$ 为系数。G 即为总共想要合成的少数类样本数量，如果 $\beta = 1$ 则是合成后各类别数目相等。

对于每个少类别样本 $\mathbf{x}_i$，找出其 K 近邻个点，并计算：

$$\Gamma_i = \frac{\Delta_i \,/\, K}{Z}$$

其中 Δ_i 为 K 近邻个点中多数类样本的数量，Z 为规范化因子以确保 Γ 构成一个分布。这样若一个少数类样本 $\mathbf{x}_i$ 的周围多数类样本越多，则其 Γ_i 也就越高。

最后对每个少类别样本 $\mathbf{x}_i$ 计算需要合成的样本数量 g_i，再用 SMOTE 算法合成新样本：

$$g_i = \Gamma_i \times G$$

可以看到 ADASYN 利用分布 Γ 来自动决定每个少数类样本所需要合成的样本数量，这等于是给每个少数类样本施加了一个权重，周围的多数类样本越多则权重越高。ADASYN 的缺点是易受离群点的影响，如果一个少数类样本的 K 近邻都是多数类样本，则其权重会变得相当大，进而会在其周围生成较多的样本。

但是对于本实验的二分类问题 – 只有一个多数类和一个少数类，使用 SMOTE 类型的算法即可。

使用过采样（或 SMOTE）+ 强正则模型（如 XGBoost）可能比较适合不平衡的数据。拿到一个新的数据时，可以不妨直接先试试这个方法，作为基准（Baseline）

2 分类模型

2.1 逻辑回归

逻辑回归（Logistic Regression）或称对数几率回归是一种用于解决二分类（0 or 1）问题的机器学习方法，用于估计某种事物的可能性。用 Sigmoid 函数代替阶跃函数解决零点处不可导的问题，并且单调可微。

$$y = \frac{1}{1+e^{-z}}$$

带入 $z = \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} + \boldsymbol{b}$

$$y = \frac{1}{1+e^{-(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}+\boldsymbol{b})}}$$

$$\Longrightarrow \quad \ln\frac{y}{1-y} = \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} + \boldsymbol{b} \tag{1}$$

式1的左侧可以看作 $\boldsymbol{x}$ 作为正例的相对可能性。因为其公式中带有对数因此得名。

2.2 支持向量机

支持向量机（Support Vector Machine），同样也是最初用于解决二分类问题的方法，通过在样本空间中构造一个超平面将不同类别的样本分开，并且要选取对样本局部扰动容忍最好的超平面。

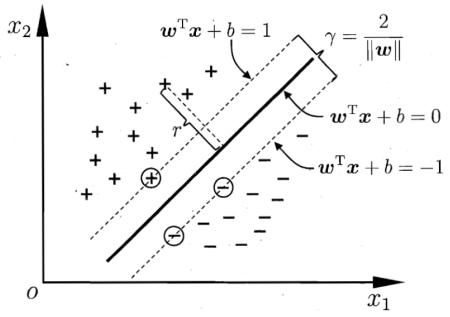


图 4: 支持向量与间隔

其中满足 $\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} + \boldsymbol{b} = \pm 1$ 的样本点，它们被称作支持向量，两个不同类的支持向量到超平面的距离之和为

$$\gamma = \frac{2}{\|\boldsymbol{w}\|} \tag{2}$$

它被称为间隔。支持向量机的训练目标是最大化间隔。可以将问题表示成，

$$\min_{\boldsymbol{w},b} \ \frac{1}{2}\|\boldsymbol{w}\|^2$$
$$\text{s.t. } y_i(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x_i} + \boldsymbol{b}) \geq 1, \quad i = 1, 2, \cdots, m.$$

同时，上述问题可以由拉格朗日乘子法转化为对偶问题。进一步地，如果选取合适的核函数，可以将非线性问题转化为线性问题求解。

2.3 最近邻方法

最近邻方法，也称 KNN 算法或译 K-近邻算法（k-nearest neighbors algorithm）其实是思想特别简单的算法，即是给定测试样本，基于某种距离度量找出训练集中与其最靠近的 k 个训练样本，然后基于这 k 个邻居训练样本的信息来进行预测。

通常在分类任务中可使用投票法，即选择这 k 个样本中出现最多的类标记作为预测结果；在回归任务中可使用平均法，即将这 k 个样本的实值输出标记的平均值作为预测结果，还可基于距离远近进行加权平均或者加权投票，距离越近的样本权重越大。

2.4 随机森林

随机森林（Random Forest）是集成学习中 Bagging 的一种。Bagging 的思想是生成基学习器之间需要有较大差异，组成的强学习器的方差减小。Bagging 多次对样本抽样产生不同的采样集。随机森林以决策树为基学习器，在训练过程中引入随机特征选择，从 d 个特征中选择 k 个特征，然后再从这个子集中选择一个最优特征用于划分。

随机森林简单、容易实现、可并行、计算开销小，令人惊奇的是，它在很多现实任务中展现出强大的性能。

2.5 Boosting 类

Boosting 是一族可将弱学习器提升为强学习器的算法。这族算法的工作机制类似：先从初始训练集训练出一个基学习器，再根据基学习器的表现，对训练样本分布进行调整，使得先前基学习器做错的训练样本在后续受到更多关注，然后基于调整后的样本分布来训练下一个基学习器；如此重复进行，直至基学习器数目达到事先指定的值 T，最终这个 T 个基学习器进行加权结合。

Boosting 族比较出名的是 AdaBoost。从偏差-方差分解的角度看，Boosting 主要关注降低偏差。

3 实验和结果

3.1 评价标准

		True condition	
	Total population	Condition positive	Condition negative
Predicted condition	Predicted condition positive	**True positive**	**False positive**, Type I error
	Predicted condition negative	**False negative**, Type II error	**True negative**

图 5: 二分类问题的混淆矩阵（Confusion Matrix）

基于混淆矩阵可以使用准确率（Precision）和召回率（Recall）来评价模型在不平衡数据上的分类精度。F-score（F1）是准确率和召回率的调和平均值。还有常用的评价指标 AUCROC（Area Under Receiver-Operator Characteristic curve），需要注意的是它其实是有偏的，不适用于不平衡场景下的模型评估 [3]。另一个指标 AUCPRC（Area Under Curve of Precision-Recall Curve）[3] 指准确率-召回率曲线下的面积。这些评价准则不会被不同类别中样本的数量所影响，因此通常被认为是“无偏的”，可以在类别不平衡的场景下使用。

由图 6，可以看出，在 ROC 曲线中，由于单调，曲线越向左上角弯曲越好，而 PR 曲线不是单调的，那么给定对于给定一定的 recall，precision 越大越好，换言之 AUCPRC 越大越好。ROC 曲线由于兼顾正例与负例，所以适用于评估分类器的整体性能，相比而言 PR 曲线完全聚焦于正例。

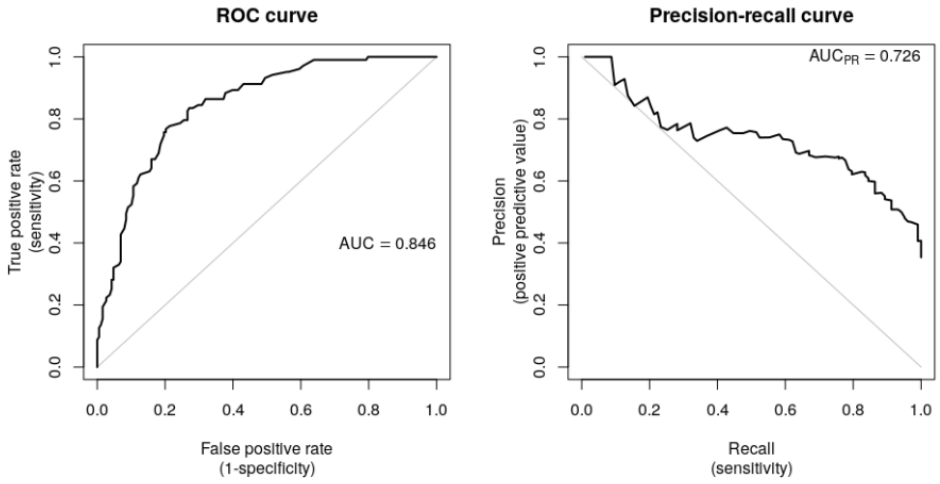


图 6: ROC 曲线与 PR 曲线

$$\text{Recall} = \frac{TP}{TP + FN} \qquad \text{Precision} = \frac{TP}{TP + FP}$$

$$\text{F1} - \text{score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

3.2 训练与测试

根据已有数据：训练集 8285 个样本，其中 7819 个负类，466 个正类；测试集 2071 个样本，其中 1955 个负类，117 个正类。训练时，将训练集的 8285 个样本随机按照 3:1 的比例分成训练集 (train set) 和验证集 (validation set)。测试集 (test set) 的 2071 个样本，仅用于训练结束后的性能测试，不参与模型训练。

3.3 实验结果

3.3.1 逻辑回归

逻辑回归采用 `sklearn.linear_model` 中的 `LogisticRegression` 训练，分四分之一的训练集 (train set) 数据作为验证集 (validation set)，使用 `GridSearchCV` 搜索最优参数：

- `C`：正则化系数 λ 的倒数，`float` 类型，默认为 1.0。越小的数值表示越强的正则化。搜索范围：$[0.001, 0.01, 0.1, 1, 10, 100, 1000]$。
- `penalty`：惩罚项，`str` 类型，可选参数为 "l1" 和 "l2"，默认为 "l2"。搜索范围："l1" 和 "l2"。

其他不需要搜索的参数设置：

- `class_weight`：用于标示分类模型中各种类型的权重，可以是一个字典或者 "balanced" 字符串，默认为不输入，也就是不考虑权重，即为 None。如果选择输入的话，可以选择 balanced 让类库自己计算类型权重，或者自己输入各个类型的权重。如果 `class_weight` 选择 balanced，那么类库会根据训练样本量来计算权重。某种类型样本量越多，则权重越低，样本量越少，则权重越高。当 `class_weight` 为 balanced 时，类权重计算方法如下：`n_samples / (n_classes * np.bincount(y))`。`n_samples` 为样本数，`n_classes` 为类别数量，`np.bincount(y)` 会输出每个类的样本数。
- `solver`：优化算法选择参数，只有五个可选参数，即 newton-cg, lbfgs, liblinear, sag, saga。默认为 liblinear。`solver` 参数决定了对逻辑回归损失函数的优化方法。对于本数据集，较小数据集的二元分类 liblinear 即可。
- `n_jobs`：并行数，或者可视作内核数。`int` 类型，默认为 1。为 1 时，用 CPU 的一个内核运行程序；为 2 时，用 CPU 的 2 个内核运行程序。为-1 的时候，用所有 CPU 的内核运行程序。

使用 `GridSearchCV` 搜索 `C` 和 `penalty`。搜索过程采用 5-Fold，评价标准采用 `sklearn` 中定义的 'f1_macro'，marco 意为每一类求 F1（准确率和召回率的调和平均），然后直接取无权平均（各类别 F1 的权重相同）。

搜索结果：`C` 取 0.01, `penalty` 取 'l1'。图 7 是最优参数在测试集 (test set) 中的混淆矩阵。表 1 为对应的评价数值。

	Precision	Recall	F1-score
-1	0.97051	0.77442	0.86145
1	0.13867	0.60684	0.22576
macro avg	0.55459	0.69063	0.54360

表 1: 逻辑回归的评价

3.3.2 支持向量机（SVM）

`sklearn` 中可以选择线性 SVM，也可以采用 RBF （径向基函数核）为核函数，转化为非线性问题求解。本实验对 2 种 SVM 都进行了训练，选择最优参数。从图 8 看出，虽然线性 SVM 在少数类上召回率没有使用 RBF 的核 SVM 好，但是线性 SVM 在 F1-score 上的表现更好。

在模型调参上，和逻辑回归类似使用 `GridSearchCV` 搜索 `C`，`kernel`，`gamma`，`penalty`，等常见参数。

搜索结果：

- 线性 SVM `C` 取 10, `penalty` 取 'l2';
- 非线性 SVM `C` 取 0.01, `kernel` 取 'rbf'，`gamma` 取 'auto'。

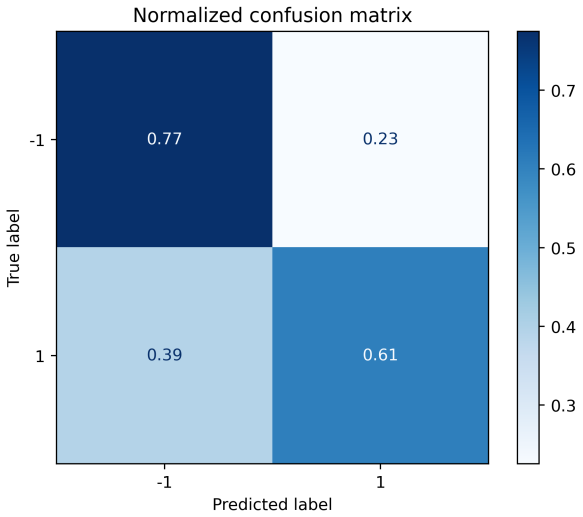


图 7: 逻辑回归的混淆矩阵（测试集）

	Precision	Recall	F1-score
-1	0.96357	0.86598	0.91218
1	0.16825	0.45299	0.24537
macro avg	0.56591	0.65949	0.57877

表 2: 线性 SVM 的评价

	Precision	Recall	F1-score
-1	0.97051	0.75754	0.85090
1	0.13187	0.61538	0.21719
macro avg	0.55119	0.68646	0.53405

表 3: RBF 核 SVM 的评价

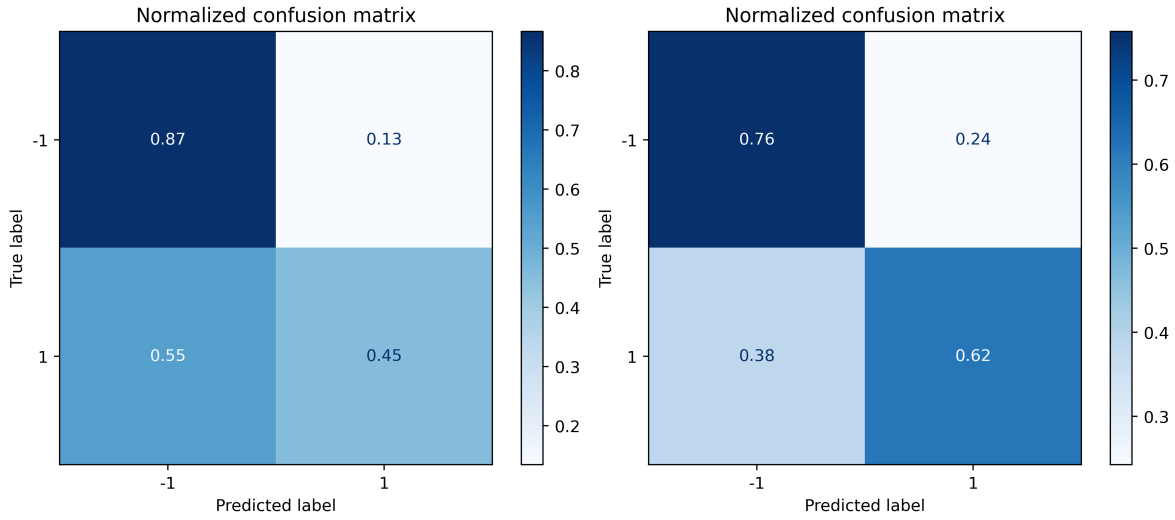


图 8: 线性 SVM，RBF 核 SVM 的混淆矩阵（测试集）

3.3.3 最近邻方法

最近邻方法的表现非常依赖于样本之间的距离这一概念的定义，常见定义是欧式距离 $\sqrt{\sum_{i=1}^{n}(x_i-y_i)^2}$，曼哈顿距离 $\sqrt{\sum_{i=1}^{n}|x_i-y_i|}$；另一个会影响模型表现的是选择邻居的个数，以及对应的权重。这里同样使用 `GridSearchCV` 搜索其会影响表现的参数：

- `n_neighbors`：选取几个最近的邻居，默认是 5，搜索范围放宽到 $[3,5,7,9]$，

- `weights`：邻居的权重是否考虑距离，可以选 'uniform' 不考虑，或者 'distance' 根据距离决定权重，
- `p`，Minkowski metric 的指数参数。当 $p = 1$ 时，与曼哈顿距离等价，当 $p = 2$ 时与欧氏距离等价。

根据搜索结果：`n_neighbors`: 3, `p`: 1, `weights`: 'distance'，得到以下图9和表4。

	Precision	Recall	F1-score
-1	0.94403	0.97494	0.95924
1	0.07547	0.03419	0.04706
macro avg	0.50975	0.50456	0.50315

表 4: KNN 的评价

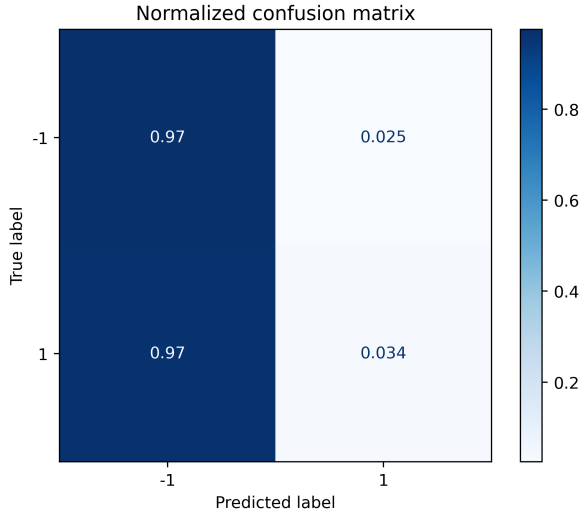


图 9: KNN 的混淆矩阵（测试集）

3.3.4 随机森林

随机森林是典型的集成学习的实现，期望由多个弱学习器集成得到一个表现优秀的强学习器。所以随机森林常见的参数有如下：

- `n_estimators`：弱学习器的个数。一般来说 `n_estimators` 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；
- `max_depth`：默认为 None，如果采用默认值，决策树在建立子树的时候不会限制子树的深度，限制深度可以防止过拟合；
- `bootstrap`：是否有放回的采样。

在实验中发现，限制深度 `max_depth` 会使得多数类表现下降，但是少数类表现提升明显。根据搜索结果：`n_estimators`: 170, `max_depth`: 7, `bootstrap`: True，得到以下图10和表5。

3.3.5 AdaBoost

类似的没有参数 `class_weight`，需要手动加入对非平衡类的处理。后面有专门一节（3.5）讨论非平衡处理的影响。

针对参数选择：

	Precision	Recall	F1-score
-1	0.97303	0.75652	0.85122
1	0.13768	0.64957	0.22720
macro avg	0.55535	0.70305	0.53921

表 5: 随机森林的评价

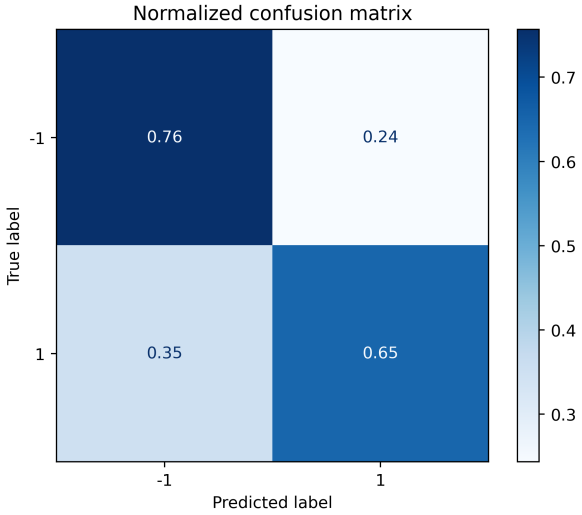


图 10: 随机森林的混淆矩阵（测试集）

- `n_estimators`：整数型，可选参数，默认为 50。弱学习器的最大迭代次数，或者说最大的弱学习器的个数。一般来说 n_estimators 太小，容易欠拟合，n_estimators 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，常常将 n_estimators 和下面介绍的参数 learning_rate 一起考虑。
- `algorithm`：可选参数，默认为 SAMME.R。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 使用了概率度量的连续值，迭代一般比 SAMME 快，因此 AdaBoostClassifier 的默认算法 algorithm 的值也是 SAMME.R。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。
- `learning_rate`：浮点型，可选参数，默认为 1.0。每个弱学习器的权重缩减系数，取值范围为 0 到 1，对于同样的训练集拟合效果，较小的 learning_rate 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。

从表6和图11可以看出，结果相对于前面的模型在少数类的表现非常差，

3.3.6 Gradient Boosting

与 AdaBoost 算法类似，Gradient Boosting 算法也是 Boosting 下的分支。Boosting 是 Ensemble Learning 算法的一个类别。Boost 的意思为“提升”，这类算法的思想是“给定仅比随机猜测略好的弱学习算法，将其提升为强学习算法”。Gradient Boosting 是一种 Boosting 的思想，它本质是，每一次建立模型是在之前建立模型损失函数的梯度下降方向。

同样的，Gradient Boosting 需要调整的参数也是类似的，比如：`n_estimators` 和 `learning_rate`。

	Precision	Recall	F1-score
-1	0.94396	0.99949	0.97093
1	0.50000	0.00855	0.01681
macro avg	0.72198	0.50402	0.49387

表 6: AdaBoost 的评价

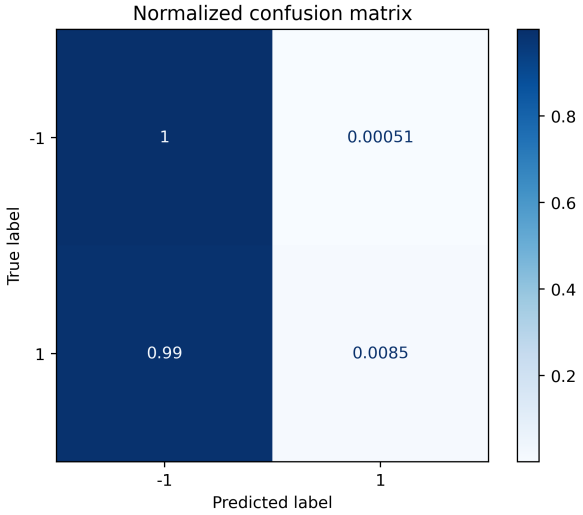


图 11: AdaBoost 的混淆矩阵（测试集）

- `loss`：损失函数。分类模型和回归模型的损失函数是不一样的。

  对于分类模型，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"。在原理篇中对这些分类损失函数有详细的介绍。一般来说，推荐使用默认的"deviance"。它对二元分离和多元分类各自都有比较好的优化。而指数损失函数等于把我们带到了 Adaboost 算法。

从表7和图12可以看出，相似的结果相对于前面的模型在少数类的表现非常差。

	Precision	Recall	F1-score
-1	0.94391	0.99847	0.97042
1	0.25000	0.00855	0.01653
macro avg	0.59695	0.50351	0.49347

表 7: GradientBoosting 的评价

3.3.7 LightGBM

LightGBM 是一种基于决策树的 Gradient Boosting 框架 [5]，2017 年由微软提出。在梯度提升决策树 (Gradient Boosting Decision Tree, GBDT) 上改进，突出轻量 light 的思想，加快的训练速度，精度却没有损失太多。

使用 `lightgbm` 中 `scikit-learn` 接口 – `LGBMClassifier`。`scikit-learn` 接口版本参数和原生参数稍有不同，不过应该无碍。需要调整的重要参数：`n_estimators`, `learning_rate`,

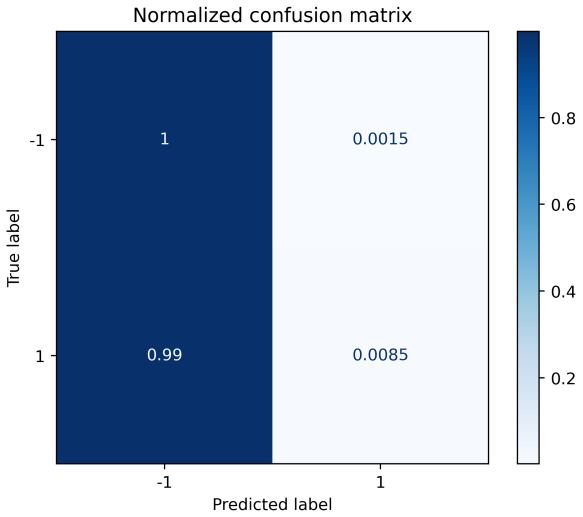


图 12: GradientBoosting 的混淆矩阵（测试集）

- `boosting_type`: boosting 类型。默认 “gbdt”; 可选: ”gbdt”:Gradient Boosting Decision Tree, ”dart”:Dropouts meet Multiple Additive Regression Trees , ”goss”:Gradient-based One-Side Sampling, ”rf”: Random Forest。

根据基础参数 `class_weight = 'balanced'` , `objective='binary'`。经过参数搜索，得搜索结果: `n_estimators`: 15, `max_depth`: 5，得到以下图13和表8。

	Precision	Recall	F1-score
-1	0.96985	0.77340	0.86056
1	0.13645	0.59829	0.22222
macro avg	0.55315	0.68585	0.54139

表 8: LightGBM 的评价

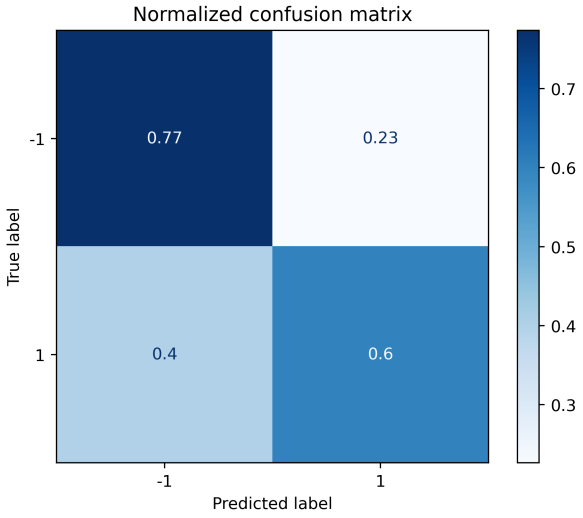


图 13: LightGBM 的混淆矩阵（测试集）

3.3.8 XGBoost

XGBoost 是陈天奇在 2016 年提出了一种可扩展性的端到端基于树的 Boosting 系统 [2]，这个系统可以处理稀疏性数据，通过分布式加权直方图算法去近似学习树，这个系统也提供基于缓存的加速模式、数据压缩、分片功能。

XGBoost 没有提供 `class_weight='balanced'` 参数，但是提供替代参数 `scale_pos_weight`，控制正负类权重的平衡，提供一个 `float` 值，典型的计算方式是 `sum(negative instances) / sum(positive instances)`。

使用 `xgboost` 中 `scikit-learn` 接口 – `XGBClassifier`。需要调整的重要参数：`n_estimators`, `max_depth`。这两个参数前面模型已有解释，故这里不再解释。

根据设定的基础参数 `scale_pos_weight` 经过参数搜索，搜索结果为：`n_estimators`: 90, `max_depth`: 9，得到以下图14和表9。

	Precision	Recall	F1-score
-1	0.95942	0.81023	0.87854
1	0.11876	0.42735	0.18587
macro avg	0.53909	0.61879	0.53220

表 9: XGBoost 的评价

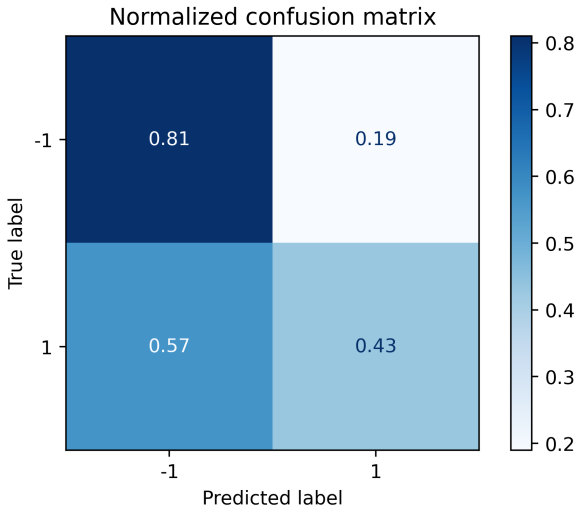


图 14: XGBoost 的混淆矩阵（测试集）

3.4 模型效果对比

模型效果对比均进行了过采样或者设定了 `class_weight='balanced'` 参数等类似的平衡少数类的的措施。正类（少数类），precision、recall、F1 分数最好的分类器分别是线性 SVM、AdaBoost、线性 SVM。简单的模型反而相对表现好。但是总体上来说，少数类的准确率在 13% 左右，召回率在 56%，F1 分数在 21%。

负类（多数类）本身在测试样本上就有优势，不同分类器在召回率表现都差异不大；在召回率上就区别较大了，表现最好还是线性 SVM，其次是 XGBoost；在 F1 分数上前两位也服从这个排序。总体上来说，多数类的准确率在 96% 左右，召回率在 77%，F1 分数在 85%。

如果需要兼顾正类和负类的表现，宏平均值（类表现的算术平均数）是一种度量。只看准确率或 F1 分数线性 SVM 是最好的，只看召回率随机森林是最好的。对于不同分类器来说，平均准确率在 13% 左右，平均召回率在 56%，平均 F1 分数在 21%，相对差异不是太大。

GBDT 和 LightGBM 结果表现非常接近，这可以理解，因为 LightGBM 是在 GBDT 上改进的，但是可能数据质量或者数量不过无法体现 LightGBM 的改进之处。

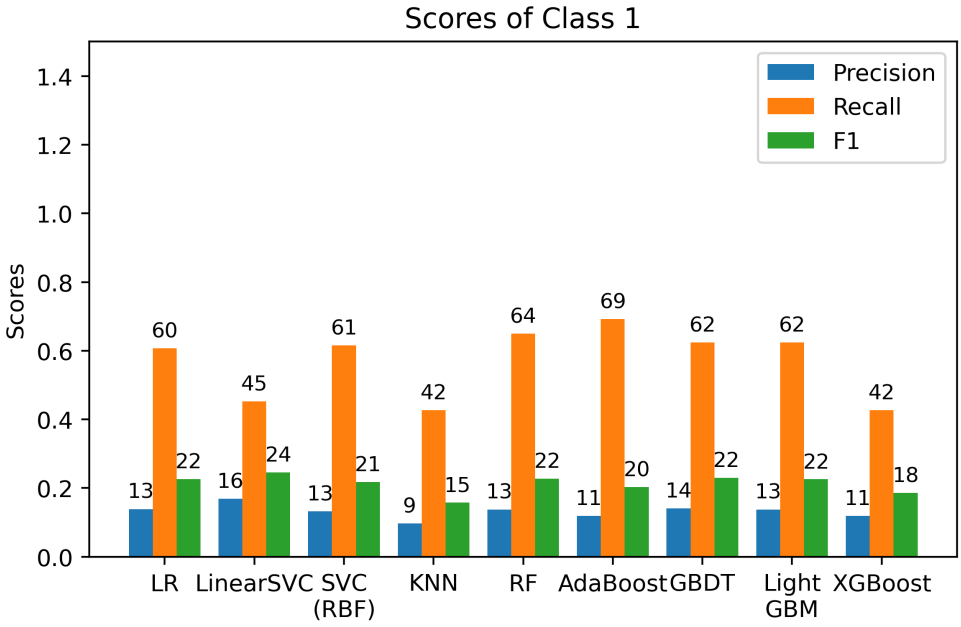


图 15: 正类（少数类）在不同分类器上的表现

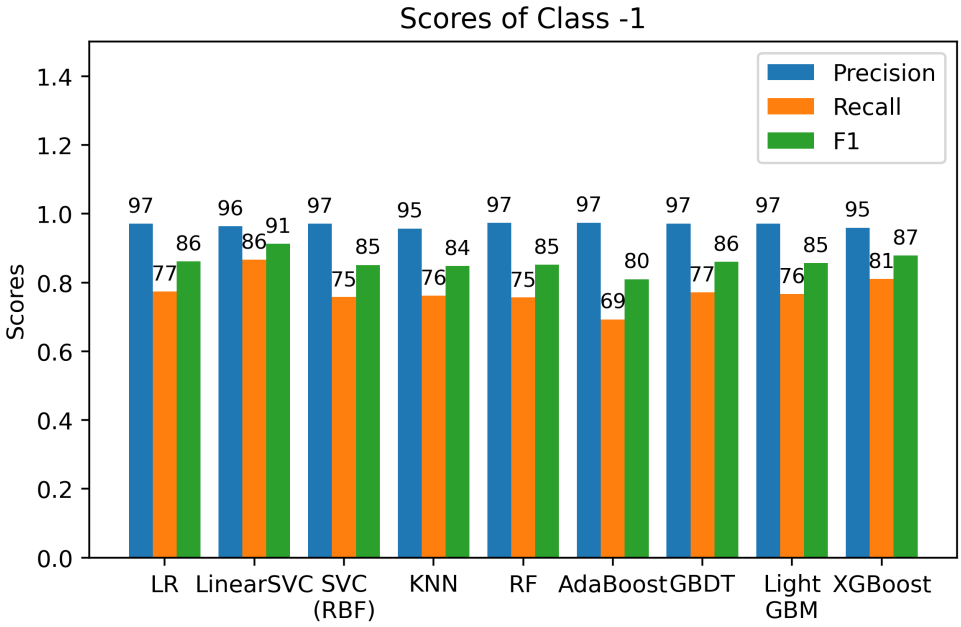


图 16: 负类（多数类）在不同分类器上的表现

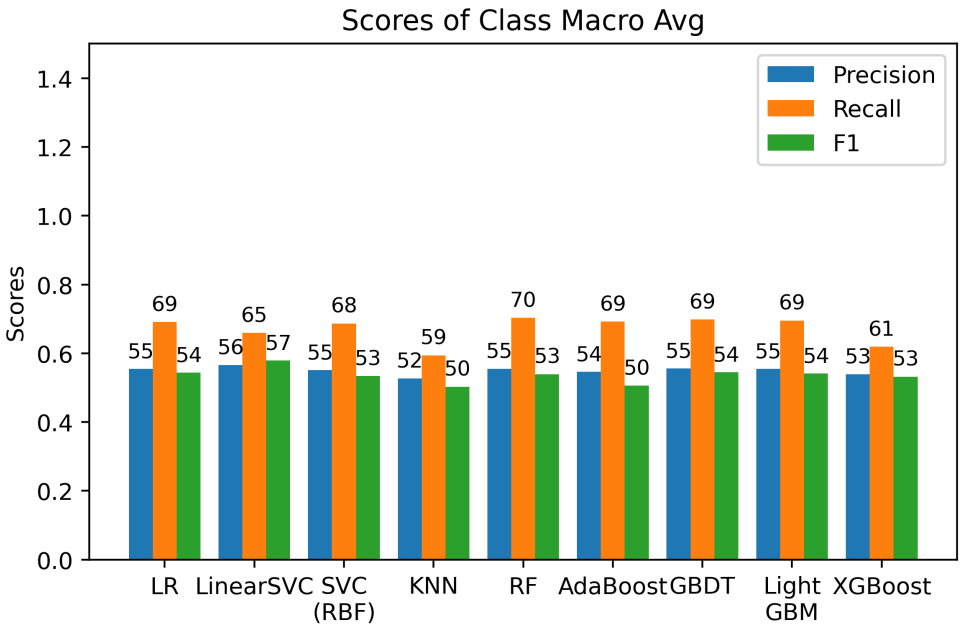


图 17: 宏平均值（类表现的算术平均数）在不同分类器上的表现

3.5 过采样效果对比

3.5.1 无调整 vs. 过采样

前面有些模型在训练时候没有可选参数 (`class_weight='balanced'`)。因为没有针对不平衡类优化，所以少数类表现特别差。下面使用过采样的方式实验是否能提高少数类的性能。

例如最近邻算法，应用 SMOTE，结果如图18和表10，与没有使用过采样的图9和表4相比，平均类准确率和召回率具有提升，在提升少数类表现的同时，多数类表现下降，这是放大少数类噪声造成的影响。

	Precision	Recall	F1-score
-1	0.95691	0.76113	0.84786
1	0.09671	0.42735	0.15773
macro avg	0.52681	0.59424	0.50280

表 10: 使用 SMOTE 过采样和 KNN 的评价

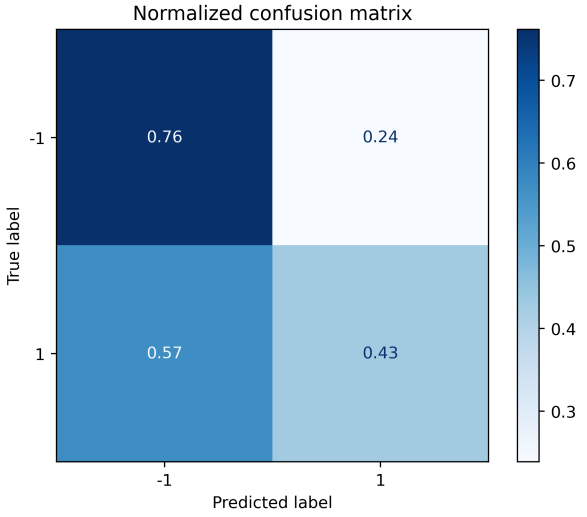


图 18: 使用 SMOTE 过采样和 KNN 的混淆矩阵（测试集）

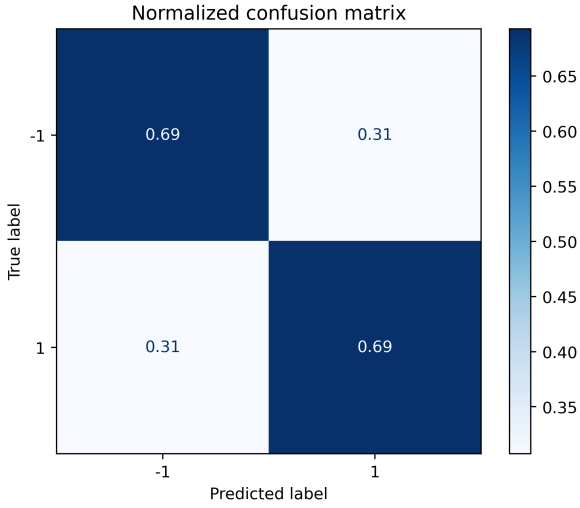


图 19: 使用 SMOTE 过采样和 AdaBoost 的混淆矩阵（测试集）

同样，针对没有可选参数 `class_weight='balanced'` 的模型 – AdaBoost 和 Gradient Boosting，都进行了实验查看过采样的影响，具有相似结果：AdaBoost 图19，Gradient Boosting 图20。

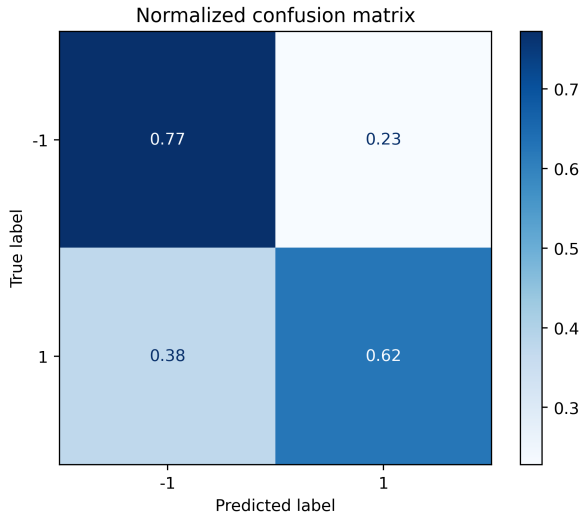


图 20: 使用 SMOTE 过采样和 GradientBoosting 的混淆矩阵（测试集）

3.5.2 class_weight='balanced' vs. 过采样

进一步的，查看了 class_weight='balanced' 的算法原理，是将不同样本数量的类，通过增加类权重，使得计算损失函数的时候，不同类的错误分类惩罚相同。

```
from sklearn.utils.class_weight import compute_class_weight

class_weight = 'balanced'
label = [0] * 9 + [1]*1 + [2, 2] # 9个类0, 1个类1, 2个类2
print(label)  # [0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 2, 2]
classes = [0, 1, 2]
weight = compute_class_weight(class_weight, classes, label)
print(weight)  # [ 0.44444444 4. 2. ]
print(.44444444 * 9)  # 3.99999996, 类0
print(4 * 1)  # 4, 类1
print(2 * 2)  # 4, 类2
```

如上代码片段所示，可以看到 class_weight 的调整是通过 compute_class_weight 这个函数把样本的平衡后的权重乘积（类样本数 × 类权重）调整到同一值，每个类别均如此。因此，样本数少的类相对权重就越大，保证了损失函数不会因为样本不均衡而忽略样本少的类。

至于还有易于混淆的参数 sample_weight 但是这参数目的目的不同，是为了调整样本采样时不同样本可信度的不同，导致需要在惩罚项上进行加权。sample_weight 是一个和长度为样本数量的向量，class_weight 是长度为类标签种数的向量。不过实际计算中 Actual sample weights = sample_weight * class_weight，如果没有 class_weight 或是设定为默认值，即 class_weight=1，实际使用差异不大。

所以没有 class_weight='balanced' 的参数设定，一样可以进行手动的样本过采样即可达到类似或等价的平衡效果。

参考文献

[1] Chawla, N. V., Bowyer, K. W., Hall, L. O., and Kegelmeyer, W. P. Smote: Synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research 16*, 1 (2002), 321–357.

[2] Chen, T., and Guestrin, C. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining* (2016), pp. 785–794.

[3] Davis, J., and Goadrich, M. The relationship between precision-recall and roc curves. In *Proceedings of the 23rd international conference on Machine learning* (2006), pp. 233–240.

[4] He, H., Bai, Y., Garcia, E. A., and Li, S. Adasyn: Adaptive synthetic sampling approach for imbalanced learning. In *2008 IEEE international joint conference on neural networks (IEEE world congress on computational intelligence)* (2008), IEEE, pp. 1322–1328.

[5] Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q., and Liu, T.-Y. Lightgbm: A highly efficient gradient boosting decision tree. In *Advances in neural information processing systems* (2017), pp. 3146–3154.