Mini Project

CSE5002 智能数据分析

姓名: 张晓文
学号: 11930639

2020 年 6 月 9 日

目 录

1 数据预处理

1.1 数据描述

数据集由 train.data 和 test.data 两个文件组成，每个文件可看作一个 $N \times (d+1)$ 的矩阵，其中 N 为样本数，d 为特征数。数据集共有 10 个特征，分别对应数据集的前 10 列；最后一列代表样本类别，取值仅为 1 或-1，即此为二分类问题。其中 train 有 8285 个样本 (因此矩阵有 8285 行，11 列)，用于训练;test 有 2072 个样本。表格1和2分别列出了每一个特征的最大值和最小值，可以看到，除了第 4 个特征在训练集和测试集的最大值分别为 1.96310 和 3.444400 外, 其余列大致分布在区间 [-1,2] 内，可不做归一化。

Column Index	0	1	2	3	4	5	6	7	8	9
Minimum	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
Maximum	0.97803	1	1	0.90172	1.96310	-0.93067	-0.18695	0.25	-0.56799	1.14290

表 1: train.data 各列最值表

Column Index	0	1	2	3	4	5	6	7	8	9
Minimum	-0.80899	-0.97776	-1	-1	-1	-1	-1	-1	-1	-1
Maximum	1	1	-0.25	-0.066779	3.4444	-0.8992	-0.59563	-0.655	0.62739	-0.57143

表 2: test.data 各列最值表

1.2 特征相关性分析

在本次 project 中，我使用了 pandas 工具包来读取数据集，其中 pandas.DataFrame 类提供了计算相关系数的接口，可以用来进行特征相关性分析，如下面代码所示，其中 *dframe* 为读取 train.data 文件后的 DataFrame，*corr()* 为计算列相关性的方法。将分析结果画成热力图，如图1所示，格子颜色越深，对应的两个特征相关性越强。其中特征 0 与特征 1 的相关度达到了 0.96，故可只选取其中一个特征用于之后的训练。为了方便截取，之后我们选取特征 1 到特征 9 进行训练。

```
corr = dframe.corr().round(2)
mask = np.tril(np.ones(corr.shape)).astype(np.bool)
corr_lt = corr.where(mask) # only keep lower triangle
sns_plot = sns.heatmap(corr_lt, vmin=0, vmax=1, annot=True, cmap="Blues")
sns_plot.get_figure().savefig('img/feature_corr.png')
```

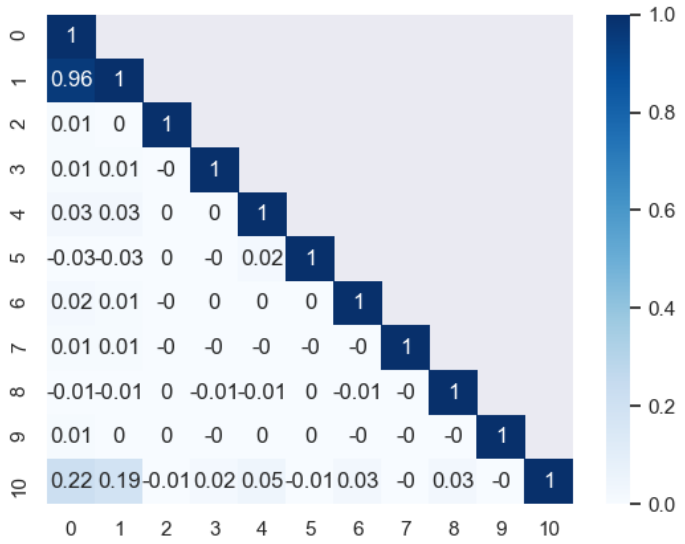

图 1: 特征相关性热力图

1.3 SMOTE 重复采样

经统计，train.data 包含标签为-1 的样本 7819 个，标签为 1 的样本 466 个；test.data 包含包含标签为-1 的样本 1955 个，标签为 1 的样本 117 个。少数类样本数目少于多数类样本数目的 6%，数据比例严重失衡。造成数据分布不均的原因有两种：一是在数据的真实分布中，比例失衡事实客观存在；二是样本数目太少，不能反映数据的真实分布。如不对数据进行处理，则训练模型可能将所有测试数据都判为多数类，仍能得到高达 95% 的准确率。为了提高模型的整体效果，可以对数据进行过采样或欠采样来平衡样本中的数据分布。

过采样的核心思想是增加少数类的样本数目。实现方法有随机对少数类进行有放回采样，但由于少数类重复样本多，可能面临着过拟合的风险。另一种常用方法为 SMOTE 算法 (Synthetic Minority Oversampling Technique)，如算法1所示。

Algorithm 1: SMOTE

Result: 新样本集

1. 对少数类中样本 x，计算它到其它少数类样本的欧式距离，得到 k 近邻。
2. 根据失衡比例设置采样倍率 N；对每一个少数类样本 x，从它的 k 近邻随机选择 N 个样本。
3. 对每一个被选中的近邻 x_n，与原样本 x 按照公式 $x_{new} = x + rand(0,1) \cdot |x - x_n|$ 构建新样本。

我使用了 imbalanced-learn[1]提供的 SMOTE 接口来增加训练集中少数类样本数目。imbalanced-learn 是一个专门解决数据集不同类型样本数目失衡的 python 工具包，它实现了许多重采样技术，并与 scikit-learn 兼容。SMOTE 类的接口定义如下，其中 *sampling_strategy* 为 float 类型时，代表重采样后，少数类样本数目和多数类样本数目的比值。

```
class imblearn.over_sampling.SMOTE(sampling_strategy='auto', random_state=None, k_neighbors=5,
                                                    m_neighbors='deprecated', out_step='deprecated
                                                   ', kind='deprecated', svm_estimator='deprecated
                                                   ', n_jobs=1, ratio=None)
```

我对 *sampling_strategy* 进行了调参，完整代码详见 *smote_ratio.py* 文件。在调参过程中，我用了 $sklearn.model_selection.StratifiedKFold$ 对 train.data 中的数据进行 5 折分层采样，然后分别对每次的训练数据进行 SMOTE 采样。*sampling_strategy* 候选值为 $[0, 0.2, 0.4, 0.45, 0.5, 0.55, 0.6, 0.8, 1]$，因为预实验显示在 0.5 效果最好，故在其附近多取了两个值。用来评估采样效果的模型是 $sklearn.svm.SVC(kernel = 'linear')$。图2的第一列子图展示了当 *sampling_strategy* 取不同值时，类 1 的 precision 和 recall 的变

[1]https://github.com/scikit-learn-contrib/imbalanced-learn

化，第二列子图展示了类-1 的 precision 和 recall 的变化。观察到当 $sampling\_strategy = 0.5$ 是一个重要拐点，少数类的 precision 突破了 0 并且达到峰值 (17.5% 以上)，recall 也突破了 0，之后随着 $sampling\_strategy$ 的增长而增长。与此同时，多数类的 precision 从 94% 左右上升，但 recall 开始从 100% 下跌。图3所示的模型整体准确率 $sampling\_strategy = 0.5$ 时下跌到约 90%。也在综上所述，$sampling\_strategy = 0.5$ 是最佳参数值，此时模型无论从整体还是具体到类，在可接受范围内都具有较高的准确率。

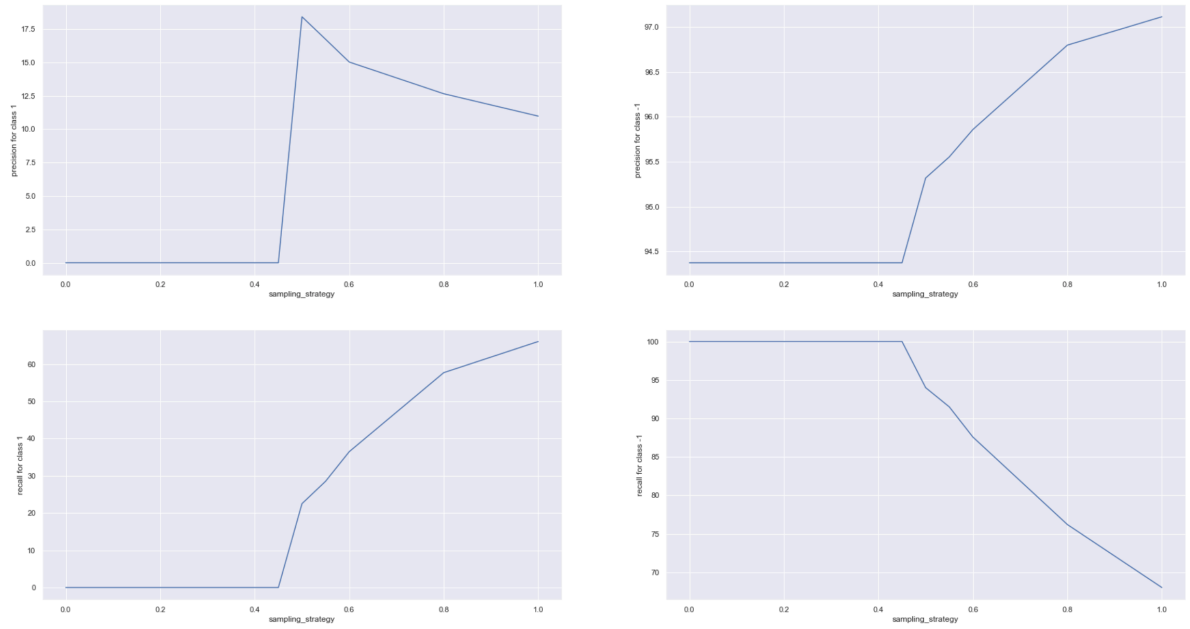

图 2: SMOTE Precision and Recall

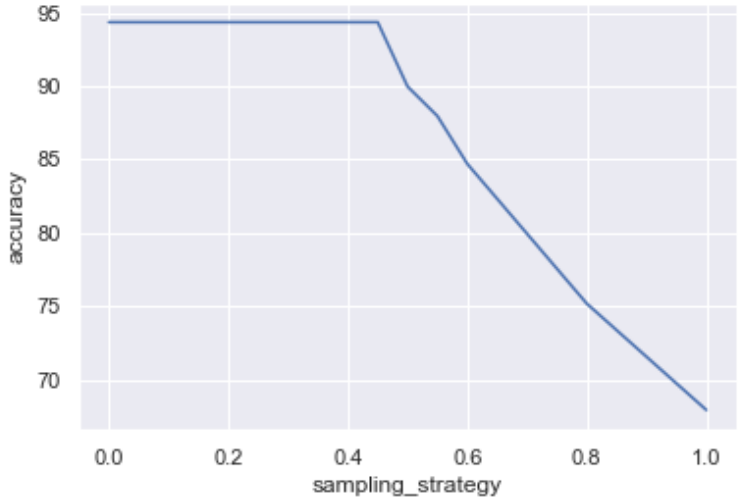

图 3: SMOTE Accuracy

2 分类模型

2.1 逻辑回归

逻辑回归是二分类问题的一个线性模型。与线性回归假设因变量 y 服从高斯分布不同，逻辑回归假设 y 服从伯努利分布。逻辑回归函数包含逻辑函数 $y = \frac{1}{1=e^{-z}}$ 和线性回归函数 $z =< w, x > +b$ 两部分，其中逻辑函数的值表示后验概率 $pr(y = 1|x)$，当 y 0.5 时，认为样本属于正类，否则属于负类。逻辑函数或者说 sigmoid 函数的曲线如图4所示。目标函数是最小化 $l(\hat{w}) = \sum_{i=1}^{n}(-y_i < \hat{w}, \hat{x_i} > +log(1+exp(< \hat{w}, \hat{x_i} >)))$，它相当于最大化 log-likelihood function $l(w, b) = \sum_{i=1}^{n} log\ pr(y_i|x_i; w, b)$. 可以使用梯度下降法或者牛顿法来解 $\hat{w}^* = arg\ min_{\hat{w} \in R^{d+1}} l(\hat{w})$.

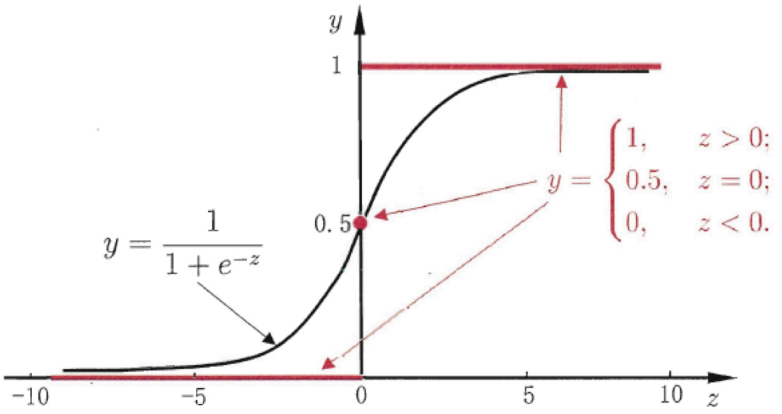

图 4: sigmoid function

2.2 支持向量机

给定一组训练实例，每个训练实例被标记为属于两个类别中的一个或另一个，SVM 训练算法建立一个将新的实例分配给两个类别之一的模型，使其成为非概率二元线性分类器。SVM 模型是将实例表示为空间中的点，这样映射就使得单独类别的实例被尽可能宽的明显的间隔分开，如图5。然后，将新的实例映射到同一空间，并基于它们落在间隔的哪一侧来预测所属类别。

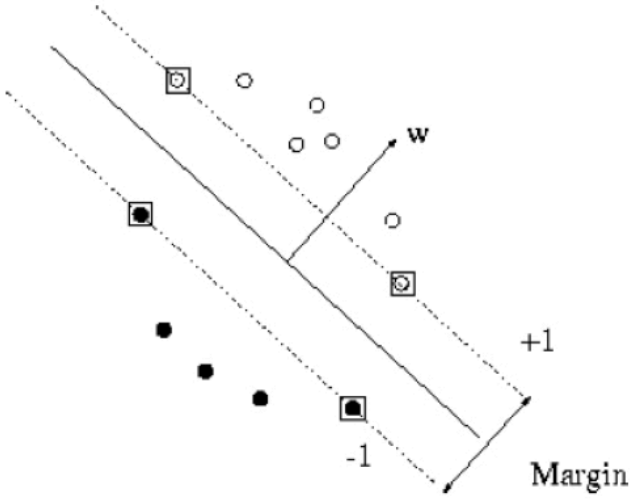

图 5: SVM margin

它的原优化问题为找一组 w, b，使得 $\frac{||w||_2^2}{2}$ 最小化，并且对任意 i 有 $y_i(< w, x_i > +b)$ 1。它的对偶问题是 $max_\alpha W(\alpha) = \sum_{i=1}^n \alpha_i - \frac{1}{2}\sum_{i,j=1}^n \alpha_i\alpha_j y_i y_j < x_i, x_j >$，使得 α_i $0, \sum_{i=1}^n \alpha_i y_i = 0$；解出最优的 α^* 后，可计算 $w^* = \sum_{i=1}^n \alpha_i^* y_i x_i$, $b^* = y_i - \sum_{i=1}^n \alpha_i^* y_i < x_i, x_j >$。分离超平面的计算公式为 $w^* x + b^* = 0$, 分类决策函数为 $f(x) = sign(w^* x + b^*)$.

对于输入空间中的非线性分类问题，可以通过非线性变换将它转化为某个维特征空间中的线性分类问题，在高维特征空间中学习线性支持向量机。由于在线性支持向量机学习的对偶问题里，目标函数和分类决策函数都只涉及实例和实例之间的内积，所以不需要显式地指定非线性变换，而是用核函数替换当中的内积。核函数表示通过一个非线性转换后的两个实例间的内积，公式为 $K(x, z) = \phi(x)\phi(z)$. 在线性支持向量机学习的对偶问题中，用核函数 [公式] 替代内积，求解得到的就是非线性支持向量机，决策函数为 $f(x) = sign(\sum_{i=1}^n \alpha_i^* y_i K(x, x_i) + b^*)$

2.3 最近邻方法

K 近邻算法，即是给定一个训练数据集，对新的输入实例，在训练数据集中找到与该实例最邻近的 K 个实例，这 K 个实例的多数属于某个类，就把该输入实例分类到这个类中，如图6。最近邻的定义是通过不同距离函数来定义，最常用的是欧式距离。为了保证每个特征同等重要性，最好对特征进行归一化。

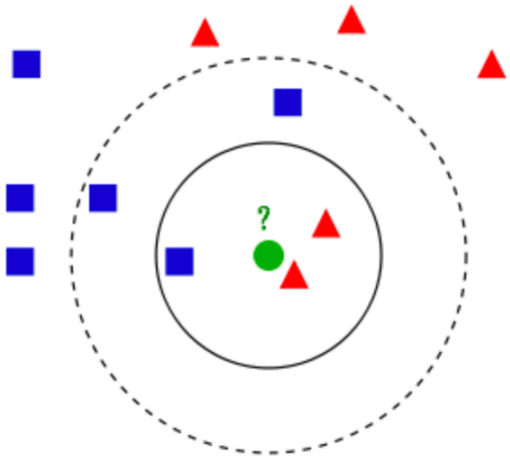

图 6: KNN

2.4 随机森林

随机森林是 Bagging 的扩展变体，它不仅对训练样本进行多次有放回采样，用来训练不同的弱分类器，还在决策树的训练过程中引入了随机特征选择 (feature selection)。对决策树的每个节点，首先从该节点的特征里选出包含 k 个特征的子集，然后从子集里选出最优特征进行划分。随机森林的优点是简单、易实现且开销小，虽然引入特征选择使得个体分类器性能降低，从而导致随机森林起始性能往往较差，但是随着弱分类器数目增加，随机森林通常会收敛到更低的泛化误差。

2.5 Boosting 类

Boosting 的基本思想是结合多个弱分类器来创造一个强分类器。它的特点在于使用了 adaptive 的样本权重。在它的循环算法中 (算法2)，每一步都会产生一个弱分类器，然后增加分错的训练样本的权重，使得它下一次有更大的可能被抽中，因此每一个弱分类器的生成都是朝着损失函数的负梯度方向，若干步后

就可以逼近损失函数局部最小值。

Algorithm 2: Boosting

Result: base classifiers
Input the original data set D
Input the number of bootstrap samples k
Initialise the weights for samples $w \leftarrow (\frac{1}{N}, \frac{1}{N}, ..., \frac{1}{N})$
i = 1
while $i\ k$ **do**
  Create a boostrap sample D_i of size N from D according to w
  Train a base model on D_i
  Increase the weights of incorrectly classified examples
  Reduce the weights of correctly classified examples
  Normalise w
end

3 实验及结果

3.1 评价标准

在本次项目中，用到的评分标准有准确率 (accuracy), 精确率 (precision), 召回率 (recall), f1 score 和混淆矩阵。

准确率 (accuracy) 是分类正确的样本占总样本个数的比例，计算公式为 $Accuracy = \frac{n\_correct}{n\_total}$，其中 $n\_correct$ 是分类正确的样本数，n_total 是总样本数。当不同类别样本的比例非常不均衡时，占比大的类别往往成为影响准确率的最主要因素，故在调参时不宜使用准确率作为评分标准。

精确率 (precision) 是分类器不将负样本标记为正样本的能力，计算公式为 $precision = \frac{TP}{TP+FP}$，其中 TP 代表 True Positive，即分类正确的正样本数，FP 代表 False Positive，即被误分为正样本的负样本数。精确率越高，模型对负样本的区分能力越强。

召回率 (recall) 指实际为正的样本中被预测为正的样本所占实际为正的样本的比例，计算公式为 $recall = \frac{TP}{TP+FN}$，其中 FN 是 False Negative 的缩写，表示被误分为负样本的正样本数。召回率越高，模型对正样本的识别能力越强。

F1 score 是精确率和召回率的调和平均值，计算公式为 $F1 = \frac{2 \cdot precision \cdot recall}{precision+recall}$，取值范围在区间 [0,1] 内。精确率和召回率对 F1 score 的相对贡献是相等的，F1 score 越高，说明模型越稳健。因此选其作为调参的评分标准。

混淆矩阵记录统计了样本真实类别与分类模型预测类别的数目。对于二元分类问题，混淆矩阵如图7所示。

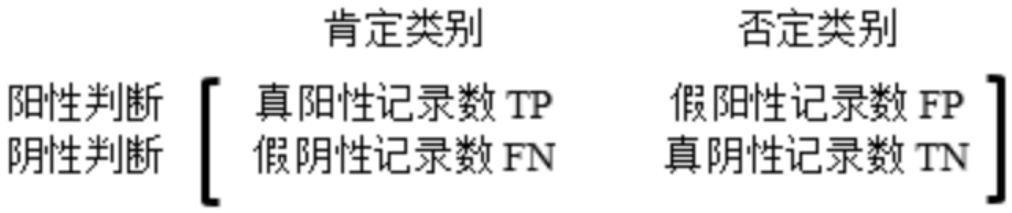

图 7: 二元分类混淆矩阵结构

宏平均 (macro average) 指先对每一个类统计指标值，然后在对所有类求算术平均值。

微平均 (micro average) 指对数据集中的每一个实例不分类别进行统计建立全局混淆矩阵，然后计算相应指标，类似于加权平均。

3.2 实验结果

本节针对每一种模型，分别介绍它们的调参过程，以及在 test.data 上的结果。搜索最佳参数使用了 scikit-learn 的 $sklearn.model_selection.GridSearchCV$，接口定义如下所示，其中最重要的参数是 $estimator$，代表训练模型；$param\_grdi$，代表需要最优化的参数取值；$scoring$，代表评分标准，我选取 f1 作为综合评分标准。它适用于小数据集，输入模型与参数，能在较合理的时间内返回参数的最佳组合。

```
class sklearn.model_selection.GridSearchCV(estimator, param_grid, *, scoring=None, n_jobs=None
                                            , iid='deprecated', refit=True, cv=None,
                                            verbose=0, pre_dispatch='2*n_jobs', error_score
                                            =nan, return_train_score=False)
```

3.2.1 逻辑回归

训练模型来自 $sklearn.linear\_model.LogisticRegression$, API 如下所示

```
class sklearn.linear_model.LogisticRegression(penalty='l2', *, dual=False, tol=0.0001, C=1.0,
                                               fit_intercept=True, intercept_scaling=1,
                                               class_weight=None, random_state=None, solver='
                                               lbfgs', max_iter=100, multi_class='auto',
                                               verbose=0, warm_start=False, n_jobs=None,
                                               l1_ratio=None)
```

我选择调节的参数是 C 和 $solver$。其中 C 代表 inverse of regularization strength, 取值越小，惩罚力度越大，模型越不容易过拟合，候选值为 $[0.1, 1, 5, 10, 13, 15, 20, 25, 30]$。$solver$ 代表优化算法，候选值为 $['liblinear', 'sag', 'lbfgs', 'newton-cg']$，完整代码见 $grid\_search\_logistic\_regression.py$ 文件。图8为 C 和 $solver$ 的不同取值对 f1 分数的影响，最优参数组合为 $C = 15$, $solver = sag$。

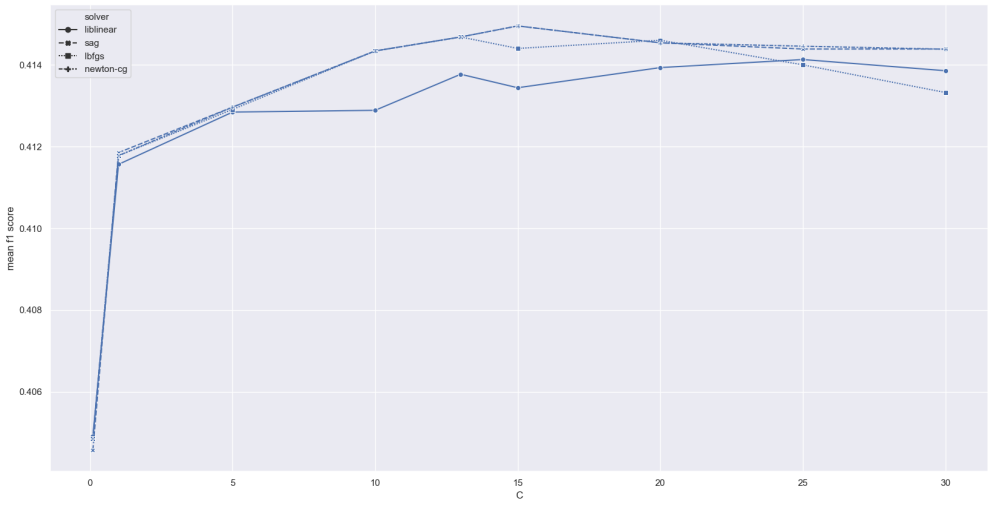

图 8: Logistic Regression 模型参数得分表

用 train.data 所有的样本训练调参后的逻辑回归模型，并对 test.data 的样本做预测。表3给出了该分类器在类 1 和类-1 的 precision、recall、f1 分数，并且分别计算了各项的宏平均和加权平均，其中 support 是两类的实际样本数。可以看到多数类的各项分数都在 0.9 以上，少数类的 precision 分数是 0.19，recall 是 0.38，虽然比多数类差许多，但未经过重采样前的 0 相比，已经是很大的提升。表4是对该模型整体表现评分，可以看到该模型的准确度大约在 0.89 左右，虽然比重采样前的 0.95 左右低，但还在可接受范围内，并且在少数类上的表现有较大提升。此外，根据混淆矩阵，-1 类分对了 1770 个，分错 185 个；1 类分对了 44 个，分错了 73 个。

类名	precision	recall	f1-score	support
-1	0.96	0.91	0.93	1955
1	0.19	0.38	0.25	117
macro avg	0.58	0.64	0.59	2072
weighted avg	0.92	0.88	0.89	2072

表 3: Logistic Regression 模型各类分数表

accuracy	weighted f1	micro precision	mac precision	macro recall	mic recall
0.875	0.894	0.875	0.576	0.641	0.875

表 4: Logistic Regression 模型整体分数表

3.2.2 支持向量机

我选择了线性 SVM 与 rbf 核的非线性 SVM 模型。

线性 SVM 模型来自 *sklearn.svm.LinearSVC*，API 如下所示

```
class sklearn.svm.LinearSVC(penalty='l2', loss='squared_hinge', *, dual=True, tol=0.0001, C=1.
                                                    0, multi_class='ovr', fit_intercept=True,
                                                    intercept_scaling=1, class_weight=None, verbose
                                                    =0, random_state=None, max_iter=1000)
```

我选择调节的参数是 C，它是正则化参数，与正则化强度成反比，必须为正数。我选取的候选值为 $[0.1, 1, 5, 10, 13, 15, 20, 25, 30]$，完整代码见 *grid_search_linear_svc.py* 文件。图9为 C 的不同取值对 f1 分数的影响，最优参数为 $C = 10$。

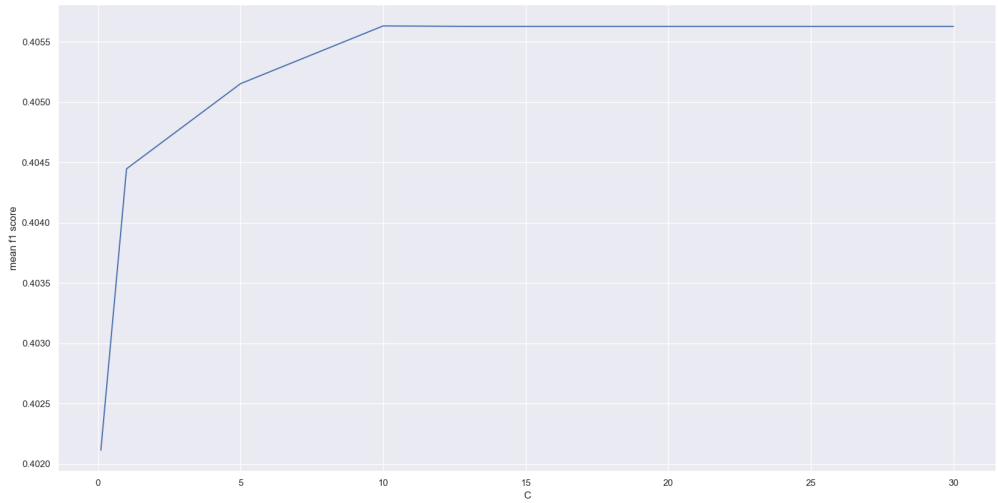

图 9: Linear SVC 模型参数得分表

用 train.data 所有的样本训练调参后的线性 SVM 模型，并对 test.data 的样本做预测。表5给出了该分类器在类 1 和类-1 的 precision、recall、f1 分数，并且分别计算了各项的宏平均和加权平均，其中 support 是两类的实际样本数。可以看到多数类的各项分数都在 0.9 以上，少数类的 precision、recall 和 f1 score 均比逻辑回归模型高 0.01 左右，优势并不明显。表6是对该模型整体表现评分，可以看到该模型的准确度比逻辑回归高 0.023，多分对了大约 47 个样本。根据混淆矩阵，-1 类分对了 1787 个，分错 168 个；1 类分对了 42 个，分错了 75 个。

类名	precision	recall	f1-score	support
-1	0.96	0.91	0.94	1955
1	0.20	0.36	0.26	117
macro avg	0.58	0.64	0.60	2072
weighted avg	0.92	0.88	0.90	2072

表 5: Linear SVC 模型各类分数表

accuracy	weighted f1	micro precision	mac precision	macro recall	mic recall
0.883	0.898	0.883	0.580	0.637	0.883

表 6: Linear SVC 模型整体分数表

rbf 核 SVM 模型来自 *sklearn.svm.SVC*, API 如下所示

```
class sklearn.svm.SVC(*, C=1.0, kernel='rbf', degree=3, gamma='scale', coef0=0.0, shrinking=
                                                    True, probability=False, tol=0.001, cache_size=
                                                    200, class_weight=None, verbose=False, max_iter
                                                    =-1, decision_function_shape='ovr', break_ties=
                                                    False, random_state=None)
```

我选择调节的参数是 C 和 $gamma$。其中 C 同前面一样，为正则参数，候选值为 $[300, 400, 500, 600, 700]$。$gamma$ 是 rbf 核系数，候选值为 $['scale', 0.1, 1]$，完整代码见 *grid_search_svc.py* 文件。图10为 C 和 $gamma$ 的不同取值对 f1 分数的影响，最优参数组合为 $C = 500$ ，$solver = 'scale'$。

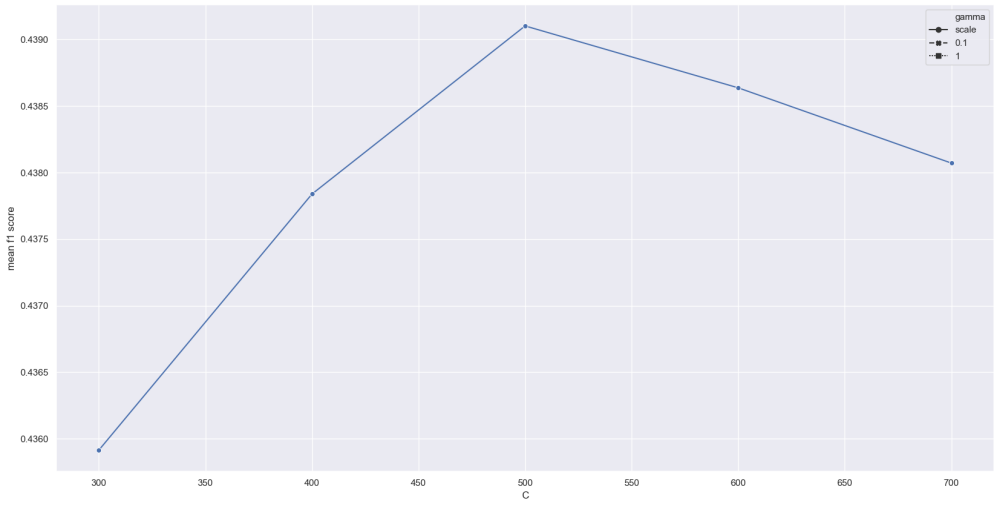

图 10: SVC 模型参数得分表

用 train.data 所有的样本训练调参后的模型，并对 test.data 的样本做预测。表7给出了该分类器在类 1 和类-1 的 precision、recall、f1 分数，并且分别计算了各项的宏平均和加权平均，其中 support 是两类的实际样本数。表8是对该模型整体表现评分，可以看到该模型的准确度大约在 0.891 左右，比前逻辑回归和线性 SVM 差一些。根据混淆矩阵，类-1 分对了 1758 个，分错了 197 个，类 1 分对了 46 个，分错了 71 个。

类名	precision	recall	f1-score	support
-1	0.96	0.90	0.93	1955
1	0.19	0.39	0.26	117
macro avg	0.58	0.65	0.59	2072
weighted avg	0.92	0.87	0.89	2072

表 7: SVM 模型各类分数表

accuracy	weighted f1	micro precision	mac precision	macro recall	mic recall
0.871	0.891	0.871	0.575	0.646	0.871

表 8: SVM 模型整体分数表

3.2.3 最近邻方法

模型来自 $sklearn.neighbors.KNeighborsClassifier$，API 如下所示

```
class sklearn.neighbors.KNeighborsClassifier(n_neighbors=5, *, weights='uniform', algorithm='
                                              auto', leaf_size=30, p=2, metric='minkowski',
                                              metric_params=None, n_jobs=None, **kwargs)
```

我选择调节的参数是 $n_neighbors$，使用的近邻数。我选取的候选值为 $[iforiinrange(2,13)]$, 完整代码见 $grid\_search\_knn.py$ 文件。图11为 C 的不同取值对 f1 分数的影响，呈锯齿状，最优参数为 $C=5$。

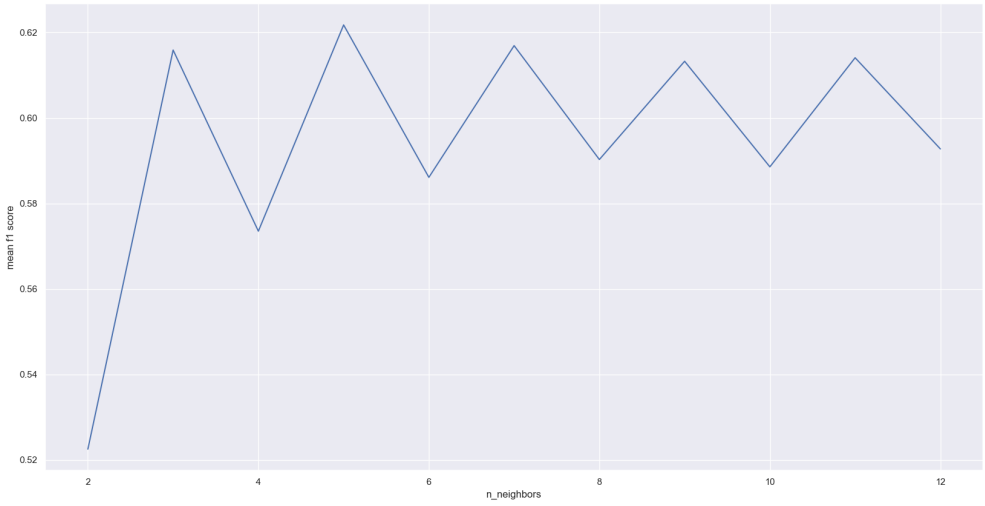

图 11: KNN 模型参数得分表

用 train.data 所有的样本训练调参后的模型，并对 test.data 的样本做预测。表9给出了该分类器在类 1 和类-1 的 precision、recall、f1 分数，并且分别计算了各项的宏平均和加权平均，其中 support 是两类的实际样本数。其中多数类的 recall 和 f1 score 低于 0.90，少数类的 precision 只有 0.07，而前三个模型至少为 0.19；两类的 recall 分数也有所下降。表10是对该模型整体表现评分，可以看到该模型的准确度仅为 0.792，显著低于之前的模型。根据混淆矩阵，类-1 分对了 1616 个，分错了 339 个，类 1 分对了 24 个，分错了 93 个。综上，最近邻方法模型区分类 1 和类-1 的能力比之前的模型弱，两类都有更多的样本被分错。

类名	precision	recall	f1-score	support
-1	0.95	0.83	0.88	1955
1	0.07	0.21	0.10	117
macro avg	0.51	0.52	0.49	2072
weighted avg	0.90	0.79	0.84	2072

表 9: KNN 模型各类分数表

accuracy	weighted f1	micro precision	mac precision	macro recall	mic recall
0.792	0.838	0.792	0.506	0.516	0.792

表 10: KNN 模型整体分数表

3.2.4 随机森林

随机森林模型来自 *sklearn.ensemble.RandomForestClassifier*, 其 API 定义如下:

```
class sklearn.ensemble.RandomForestClassifier(n_estimators=100, *, criterion='gini', max_depth
                                               =None, min_samples_split=2, min_samples_leaf=1,
                                                min_weight_fraction_leaf=0.0, max_features='
                                               auto', max_leaf_nodes=None,
                                               min_impurity_decrease=0.0, min_impurity_split=
                                               None, bootstrap=True, oob_score=False, n_jobs=
                                               None, random_state=None, verbose=0, warm_start=
                                               False, class_weight=None, ccp_alpha=0.0,
                                               max_samples=None)
```

调参过程参考了刘建平的博客[2]，顺序如下（代码见 grid_search_random_forest.py）:

1. 首先考虑树的数量 *n_estimator*，如果太小容易欠拟合，太大又容易过拟合；取值范围选择了 $range(1, 101, 10)$；如图12, 最优值为 11。
2. 调节树的最大深度 *max_depth* 和内部节点再划分所需最小样本数 *min_samples_split*，取值范围分别为 $range(3, 14, 2)$ 和 $range(30, 200, 30)$;如图13, 最优值分别为 $max\_depth = 13$ 和 $min\_samples\_split = 30$。
3. *min_samples_split* 还和叶子节点最少样本数 *min_samples_leaf* 有关联，需要一起调参；取值范围分别为 $range(10, 100, 20)$ 和 $range(10, 50, 10)$；如图14, 结果为 $min\_samples\_split = 30$ 和 $min\_samples\_leaf = 20$。
4. 最后对最大特征数 *max_features* 进行调参；如图15, 取值范围是 [0.2, 0.4,0.6,0.8,1]，最优值取 0.8。

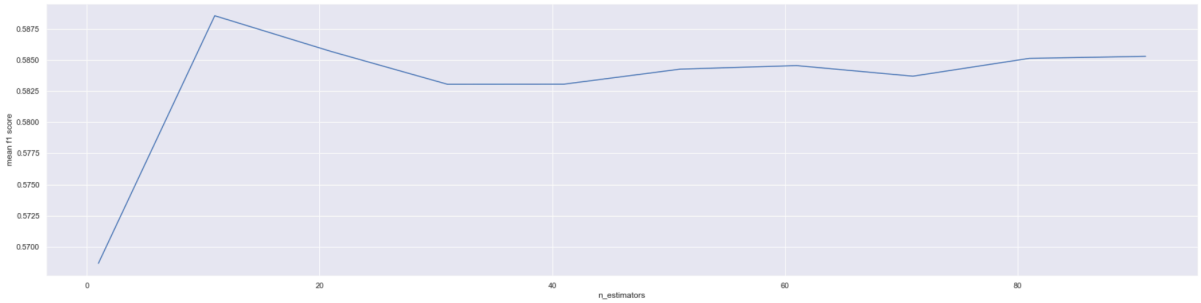

图 12: 随机森林模型 n_estimators 参数得分

[2]https://www.cnblogs.com/pinard/p/6160412.html

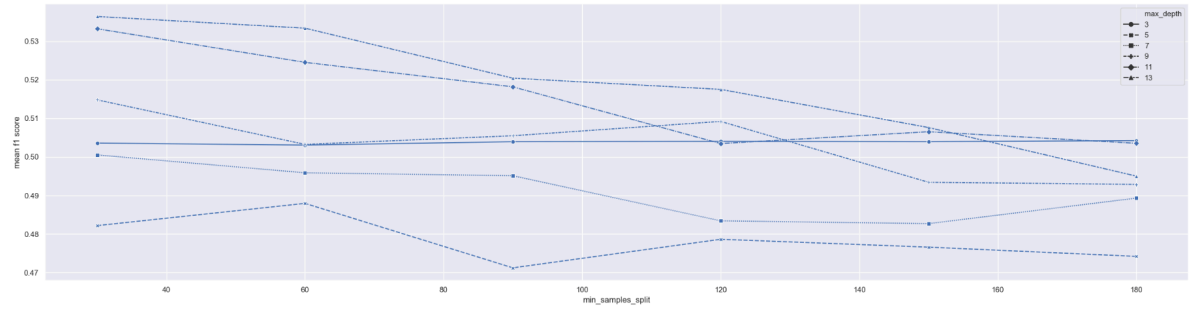

图 13: 随机森林模型 max_depth 和 min_samples_split 得分

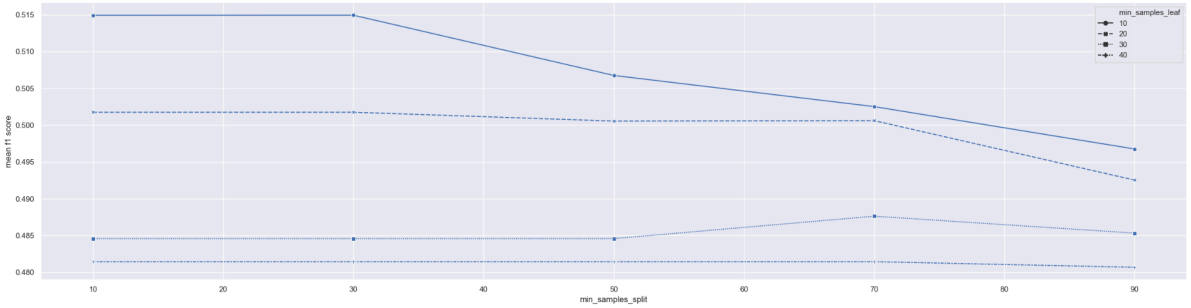

图 14: 随机森林模型 min_samples_split 和 min_samples_leaf 得分

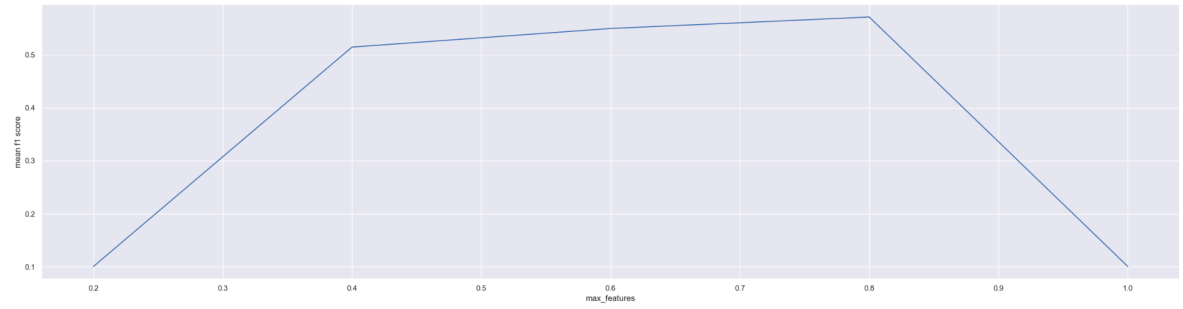

图 15: 随机森林模型 max_features 参数得分

用 train.data 训练好的模型，在 test.data 的表现如表11和12所示。随机森林在少数类上的 precision 得分为 0.17，稍逊于逻辑回归和 SVM 模型。根据混淆矩阵，类-1 分对了 1758 个，分错了 197 个，类 1 分对了 39 个，分错了 78 个。

类名	precision	recall	f1-score	support
-1	0.96	0.90	0.93	1955
1	0.17	0.33	0.22	117
macro avg	0.56	0.62	0.57	2072
weighted avg	0.91	0.87	0.89	2072

表 11: Random Forest 模型各类分数表

accuracy	weighted f1	micro precision	mac precision	macro recall	mic recall
0.867	0.888	0.867	0.561	0.616	0.867

表 12: Random Forest 模型整体分数表

3.2.5 AdaBoost

模型来自 *sklearn.ensemble.AdaBoostClassifier*, API 如下所示

```
class sklearn.ensemble.AdaBoostClassifier(base_estimator=None, *, n_estimators=50,
                                          learning_rate=1.0, algorithm='SAMME.R',
                                          random_state=None)
```

我选择调节的参数是 *n_estimators* 和 *learning_rate*。其中 *n_estimators* 候选值为 $range(1, 51, 10)$。*learning_rate* 候选值为 $[0.001, 0.01, 0.1, 1]$,完整代码见 *grid_search_adaboost.py* 文件。图16为 *n_estimators* 和 *learning_rate* 的不同取值对 f1 分数的影响，最优参数组合为 $n\_estimators = 1$, $learning\_rate = 0.001$。

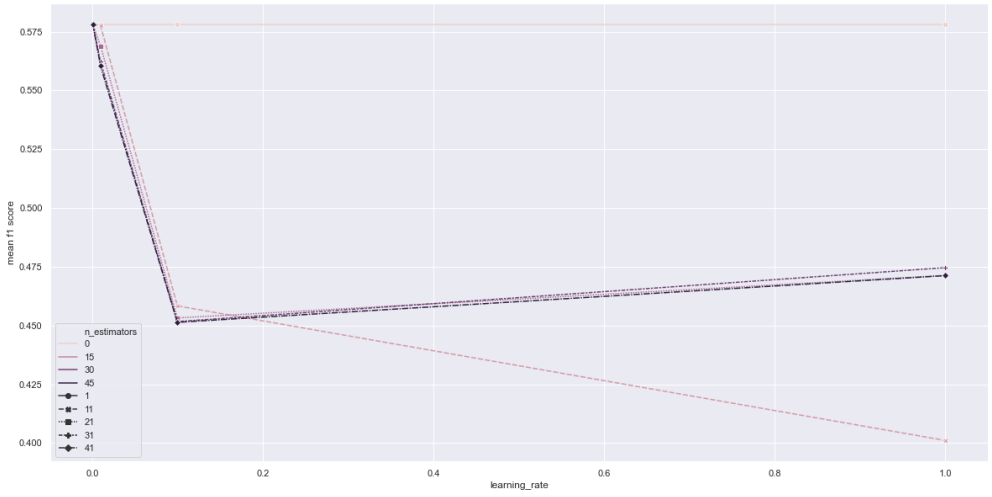

图 16: AdaBoost 模型参数得分

用 train.data 所有的样本训练调参后的模型，并对 test.data 的样本做预测。表13和表14。AdaBoost 模型明显提升了少数类的 recall 至 0.64，之前 recall 最高值仅有 0.39；但其 precision 只有 0.12。这说明有更多的少数类样本被分对了，但同时也有更多的多数类样本被分错了。因而其整体准确率为 0.871.

类名	precision	recall	f1-score	support
-1	0.97	0.71	0.82	1955
1	0.12	0.64	0.20	117
macro avg	0.54	0.68	0.51	2072
weighted avg	0.92	0.71	0.78	2072

表 13: AdaBoost 模型各类分数表

accuracy	weighted f1	micro precision	mac precision	macro recall	mic recall
0.871	0.891	0.871	0.544	0.646	0.871

表 14: AdaBoost 模型整体分数表

3.2.6 Gradient Boosting

我使用的模型为 *sklearn.ensemble.GradientBoostingClassifier*，其 API 如下：

```
class sklearn.ensemble.GradientBoostingClassifier(*, loss='deviance', learning_rate=0.1,
                                                   n_estimators=100, subsample=1.0, criterion='
                                                   friedman_mse', min_samples_split=2,
                                                   min_samples_leaf=1, min_weight_fraction_leaf=0.
                                                   0, max_depth=3, min_impurity_decrease=0.0,
                                                   min_impurity_split=None, init=None,
                                                   random_state=None, max_features=None, verbose=0
                                                   , max_leaf_nodes=None, warm_start=False,
                                                   presort='deprecated', validation_fraction=0.1,
                                                   n_iter_no_change=None, tol=0.0001, ccp_alpha=0.
                                                   0)
```

调参步骤如下（代码见 grid_search_gradientboost.py）：

1. 调节弱分类器数目 *n_estimators* 和 *learning_rate*，候选值分别为 $range(20, 121, 20)$ 和 $[0.01, 0.1, 0.5, 1]$；如图17，最优值组合为 $n\_estimators = 80$ 和 $learning\_rate = 1$。
2. 调节 *max_depth* 和 *min_samples_split*，候选值分别为 $range(3, 14, 2)$ 和 $range(100, 801, 200)$；如图18，最优组合是 $max\_depth = 7$ 和 $min\_samples\_split = 100$。

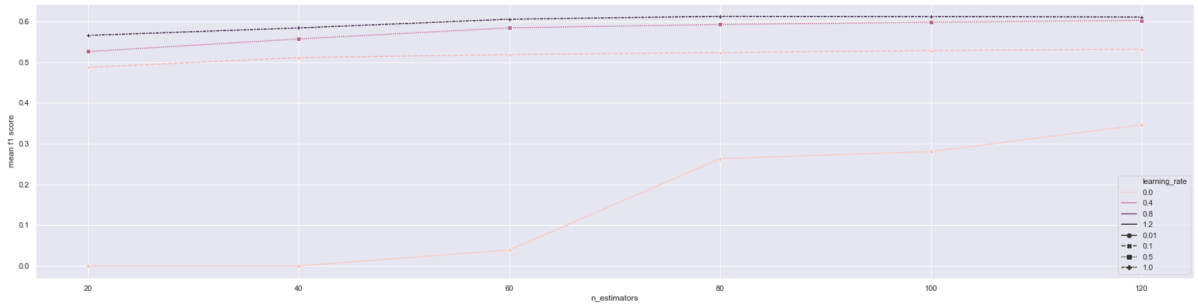

图 17: Gradient Boosting 模型参数 n_estimators 和 learning_rate 得分

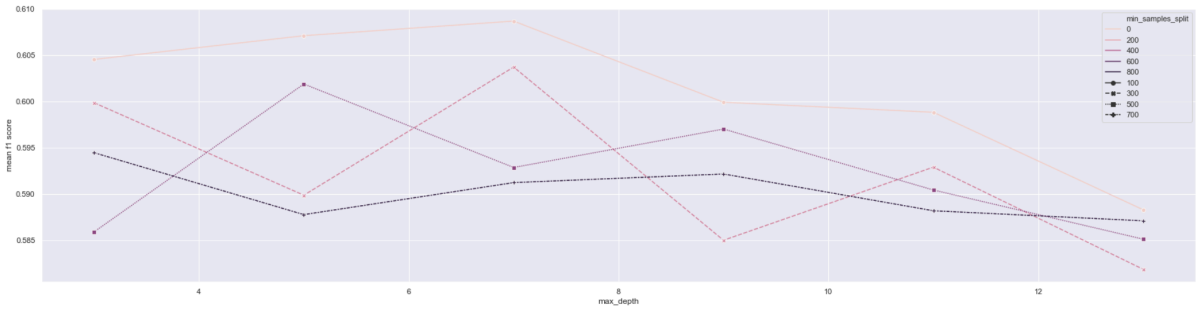

图 18: Gradient Boosting 模型参数 max_depth 和 min_samples_split 得分

训练好的模型在 test.data 上的评分如表15和 16所示。根据混淆矩阵，类-1 分对了 1658 个，分错了 297 个，类 1 分对了 26 个，分错了 91 个。它在类-1 和类 1 上的表现和最近邻模型相近，相比于其他模型，它们都分错了更多的类-1 和类 1.

类名	precision	recall	f1-score	support
-1	0.95	0.85	0.90	1955
1	0.08	0.22	0.12	117
macro avg	0.51	0.54	0.51	2072
weighted avg	0.90	0.81	0.85	2072

表 15: Gradient Boosting 模型各类分数表

accuracy	weighted f1	micro precision	mac precision	macro recall	mic recall
0.813	0.851	0.813	0.514	0.535	0.813

表 16: Gradient Boosting 模型整体分数表

3.2.7 LightGBM

LightGBM 模型来自同名包, 它是一个基于树的 gradient boosting 框架。使用到的 API 定义如下:

```
class lightgbm.LGBMClassifier(boosting_type='gbdt', num_leaves=31, max_depth=- 1,
                                                learning_rate=0.1, n_estimators=100,
                                                subsample_for_bin=200000, objective=None,
                                                class_weight=None, min_split_gain=0.0,
                                                min_child_weight=0.001, min_child_samples=20,
                                                subsample=1.0, subsample_freq=0,
                                                colsample_bytree=1.0, reg_alpha=0.0, reg_lambda
                                                =0.0, random_state=None, n_jobs=- 1, silent=
                                                True, importance_type='split', **kwargs)
```

调参顺序如下（代码见 grid_search_LGBM.py）:

1. max_depth 和 num_leaves; max_depth 不宜过深，否则容易过拟合，取值范围选择了 $[4, 6, 8]$; num_leaves 取值范围是 $[20, 30, 40]$; 如图19, 最优值为 $max\_depth = 8$ 和 $num\_leaves = 40$。
2. $min\_child\_samples$ 和 min_child_weight，取值范围分别为 $[18, 19, 20, 21, 22]$ 和 $[0.001, 0.002]$; 如图**??**, 最优值分别为 $min\_child\_samples = 18$ 和 $min\_child\_weight = 0.001$。
3. 调整 $feature\_fraction$, 防止过拟合;取值范围分别为 $[0.6, 0.8, 1]$;如图21, 结果为 $feature\_fraction = 0.8$。
4. 调整 $bagging\_fraction$ 和 $bagging\_freq$; 取值范围分别是 $[0.8, 0.9, 1]$ 和 $[2, 3, 4]$ 如图22, 最优值取 $bagging\_fraction = 0.9$ 和 $bagging\_freq = 3$。

5. 最后调整 *cat_smooth*，为设置每个类别拥有样本最小的个数，主要用于去噪；设置的取值范围是 $[0, 10, 20]$，如图23，该参数对 f1 score 影响不大，故可设置为 0.

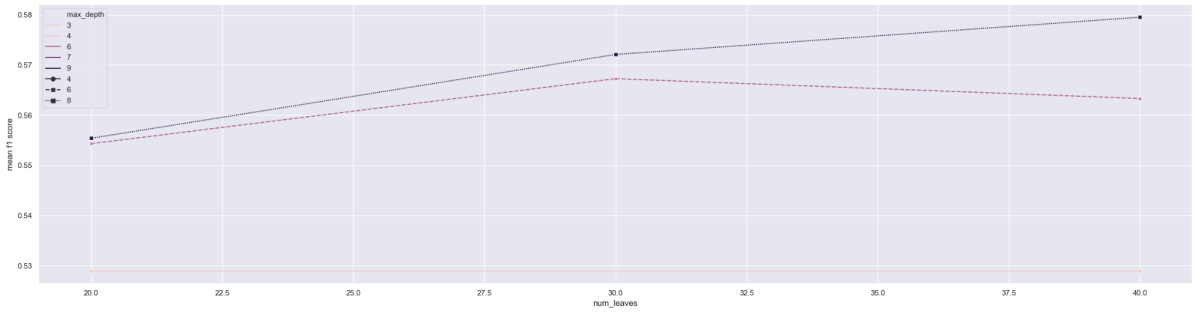

图 19: LGBM 模型 max_depth 和 num_leaves 参数得分

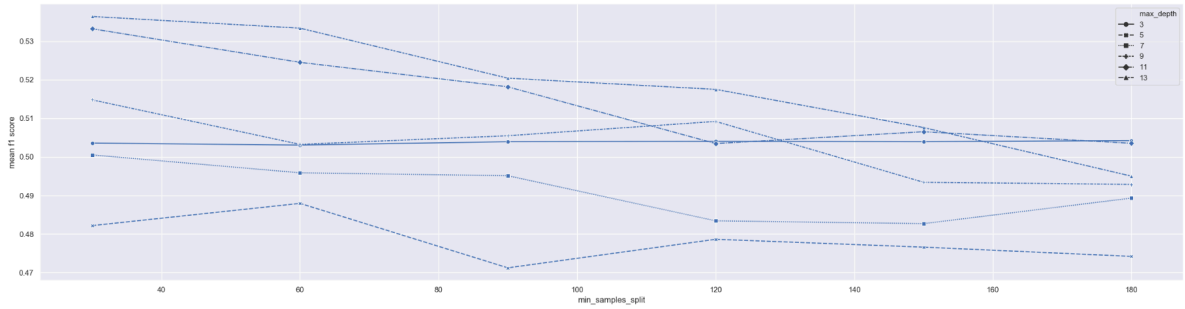

图 20: LGBM 模型 min_child_samples 和 min_child_weight 得分

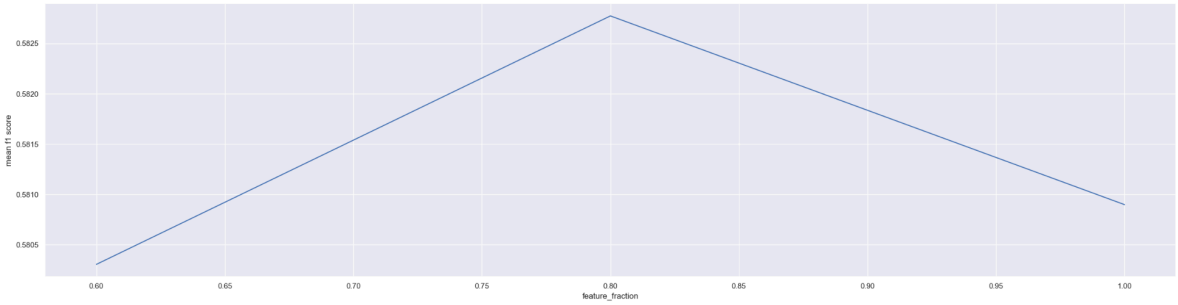

图 21: LGBM 模型 feature_fraction 得分

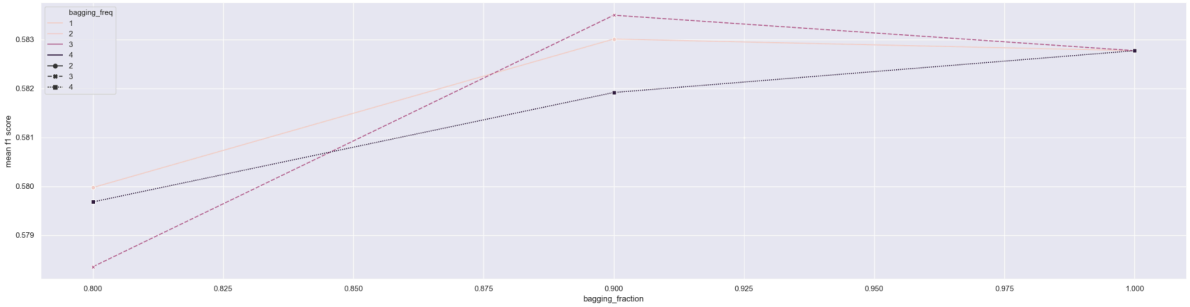

图 22: LGBM 模型 bagging_fraction 和 bagging_freq 得分

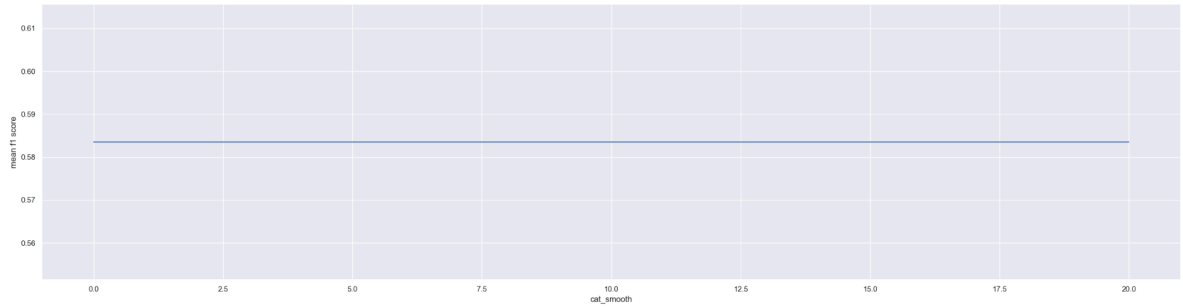

图 23: LGBM 模型 cat_smooth 得分

用 train.data 训练好的模型，在 test.data 的表现如表17和18所示。其中-1 类分对了 1752 个，分错 203 个；1 类分对了 37 个，分错了 80 个。LGBM 的表现与随机森林相似，稍逊于逻辑回归和 SVM 模型。

类名	precision	recall	f1-score	support
-1	0.96	0.90	0.93	1955
1	0.15	0.32	0.21	117
macro avg	0.56	0.61	0.57	2072
weighted avg	0.91	0.86	0.88	2072

表 17: LGBM 模型各类分数表

accuracy	weighted f1	micro precision	mac precision	macro recall	mic recall
0.863	0.885	0.863	0.555	0.606	0.863

表 18: LGBM 模型整体分数表

3.2.8 XGBoost

使用的模型为 $xgboost.XGBClassifier$，API 定义如下：

```
class xgboost.XGBClassifier(objective='binary:logistic', **kwargs)
```

调参顺序如下（代码见 grid_search_XGBoost.py）：

1. max_depth 和 min_child_weight；max_depth 不宜过深，否则容易过拟合，取值范围选择了 $[3,5,7,9]$；min_child_weight 取值范围是 $[1,3,5]$；如图24, 最优值为 $max\_depth = 9$ 和 $min\_child\_weight = 1$。

2. $learning\_rate$ 和 $n\_estimators$，取值范围分别为 $[0.001, 0.01, 0.1]$ 和 $range(200, 301, 20)$；如图25, 最优值分别为 $learning\_rate = 0.1$ 和 $n\_estimators = 220$。

3. 调整 $colsample\_bytree$ 和 $subsample$；取值范围分别为 $[0.5, 0.6, 0.8, 1]$ 和 $[0.5, 0.6, 0.8, 1]$；如图26, 结果为 $colsample\_bytree = 1$ 和 $subsample = 0.6$。

4. 调整 reg_alpha 和 reg_lambda；取值范围分别是 $[0.01, 0.1, 0.5, 0.8, 1]$ 和 $[0.01, 0.1, 0.5, 0.8, 1]$ 如图27, 最优值取 $reg\_alpha = 1$ 和 $reg\_lambda = 1$。

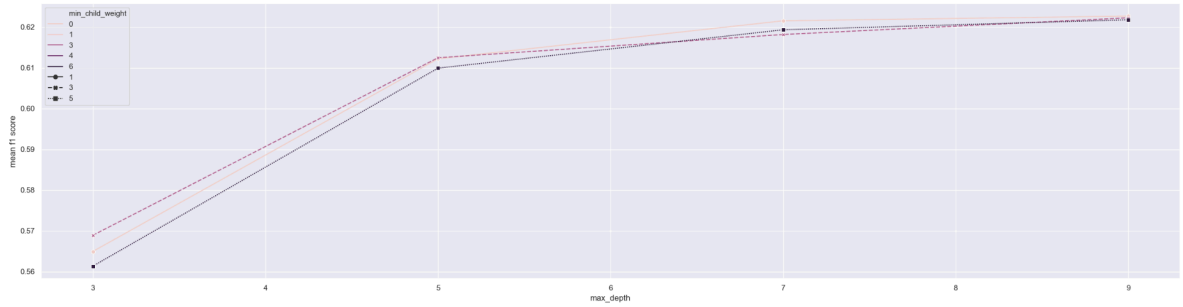

图 24: XGBoost 模型 max_depth 和 min_child_weight 参数得分

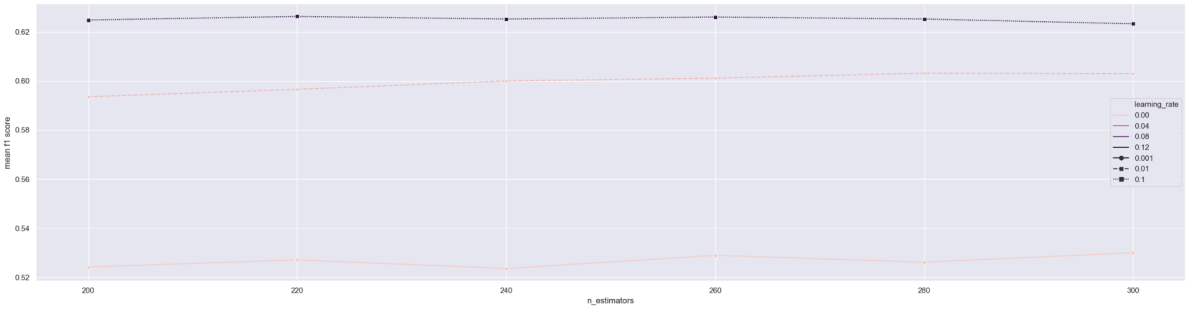

图 25: XGBoost 模型 learning_rate 和 n_estimators 得分

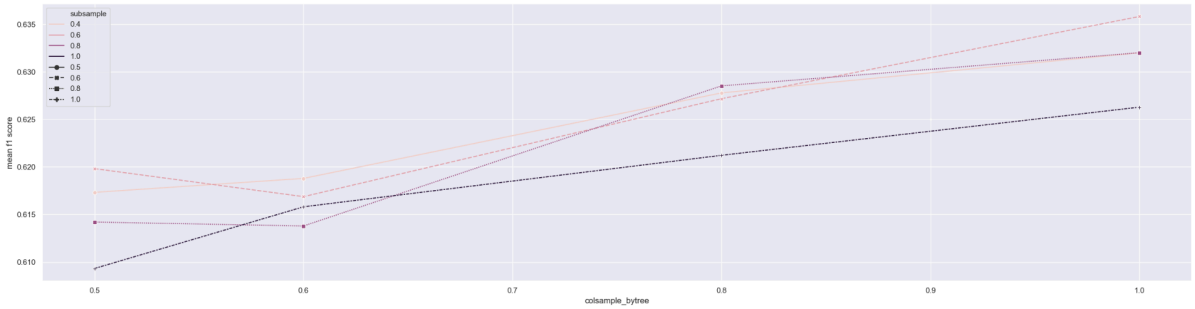

图 26: XGBoost 模型 colsample_bytree 和 subsample 得分

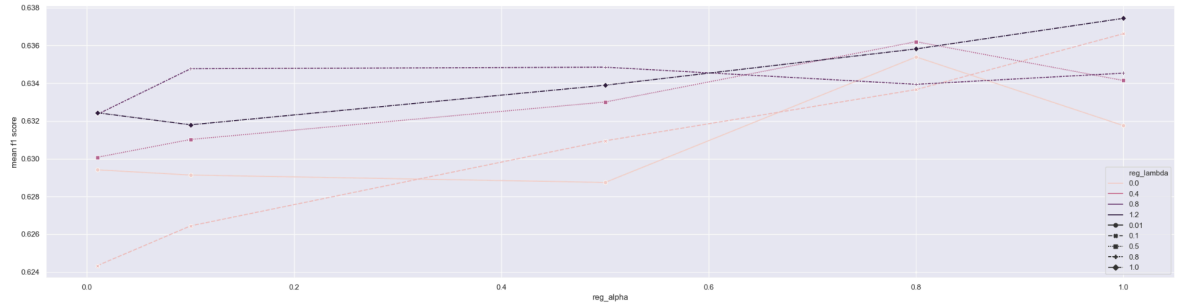

图 27: CGBoost 模 reg_alpha 和 reg_lambda 得分

用 train.data 训练好的模型, 在 test.data 的表现如表19和20所示。其中-1 类分对了 1752 个, 分错 203 个; 1 类分对了 37 个, 分错了 80 个。XGBoost 模型表现与 Gradient Boosting 相似, 不如 LightGBM 准确度高。

类名	precision	recall	f1-score	support
-1	0.95	0.88	0.91	1955
1	0.10	0.23	0.14	117
macro avg	0.53	0.55	0.53	2072
weighted avg	0.90	0.84	0.87	2072

表 19: XGBoost 模型各类分数表

accuracy	weighted f1	micro precision	mac precision	macro recall	mic recall
0.840	0.868	0.840	0.525	0.553	0.840

表 20: XGBoost 模型整体分数表

3.3 模型效果对比

首先我们从各模型得到的混淆矩阵入手, 对比各个模型在类 1 和类-1 上的表现, 如图28, 在类-1 上样本分对数超过 1750 的模型有: 逻辑回归、LightGBM、线性 SVM、rbf 核 SVM 以及随机森林; 在类 1 上样本分对数最多的模型是 AdaBoosting, 有 75 个, 分对数在 37 到 46 之间的模型有: 逻辑回归、线性 SVM、rbf 核 SVM、随机森林和 LightGBM, 分对数在 24 到 26 之间的模型有: gradient boosting、最

近邻和 XGBoost。综上，如果追求在少数类上表现最好的模型，则推荐 AdaBoosting；如果不希望牺牲太多在多数类上的表现，同时又想在少数类上工作良好，则推荐 rbf 核 SVM 模型。

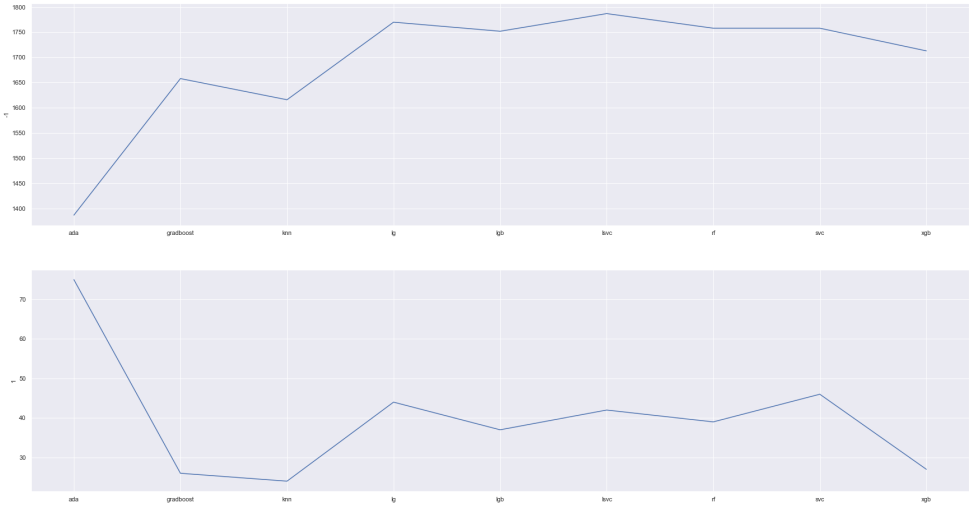

图 28: 各模型分每一类分对样本数

接下来我们从模型整体效果角度，观察不同评价标准的变化，如图29所示。准确度最高的模型是线性 SVM 模型，达到了 0.883，此外 rbf 核 SVM 也有不错的准确度 0.871；在图29中的 6 种评分标准中，可以发现逻辑回归和两个 SVM 模型排名稳占前四；而 AdaBoost 虽然在 macro_recall 一项表现优秀，但在其它评分标准下评分并不高。

图 29: 模型整体评分比较

从以上两个角度看，在这个二分类问题中，我最推崇 rbf 核 SVM 模型，其次是逻辑回归和线性 SVM 模型。

参考文献