集成学习模型汇报

——以公司培训人员留职预测为例

20大数据管理 胡宇辰、刘栋

目录

CONTENTS


西南财经大学
SOUTHWESTERN UNIVERSITY OF FINANCE AND ECONOMICS
1925

01

理论部分

01 理论——方差与偏差

定义

使用样本数相同的不同训练集产生的**方差**

$$var(x) = E_D[(f(x;D) - \overline{f}(x))^2]$$

期望输出与真实标记的差别称为**偏差**（bias）

$$bias^2(x) = (\overline{f}(x) - y)^2$$

噪声

$$\epsilon^2 = E_D[(y_D - y)^2]$$

01 理论——方差与偏差

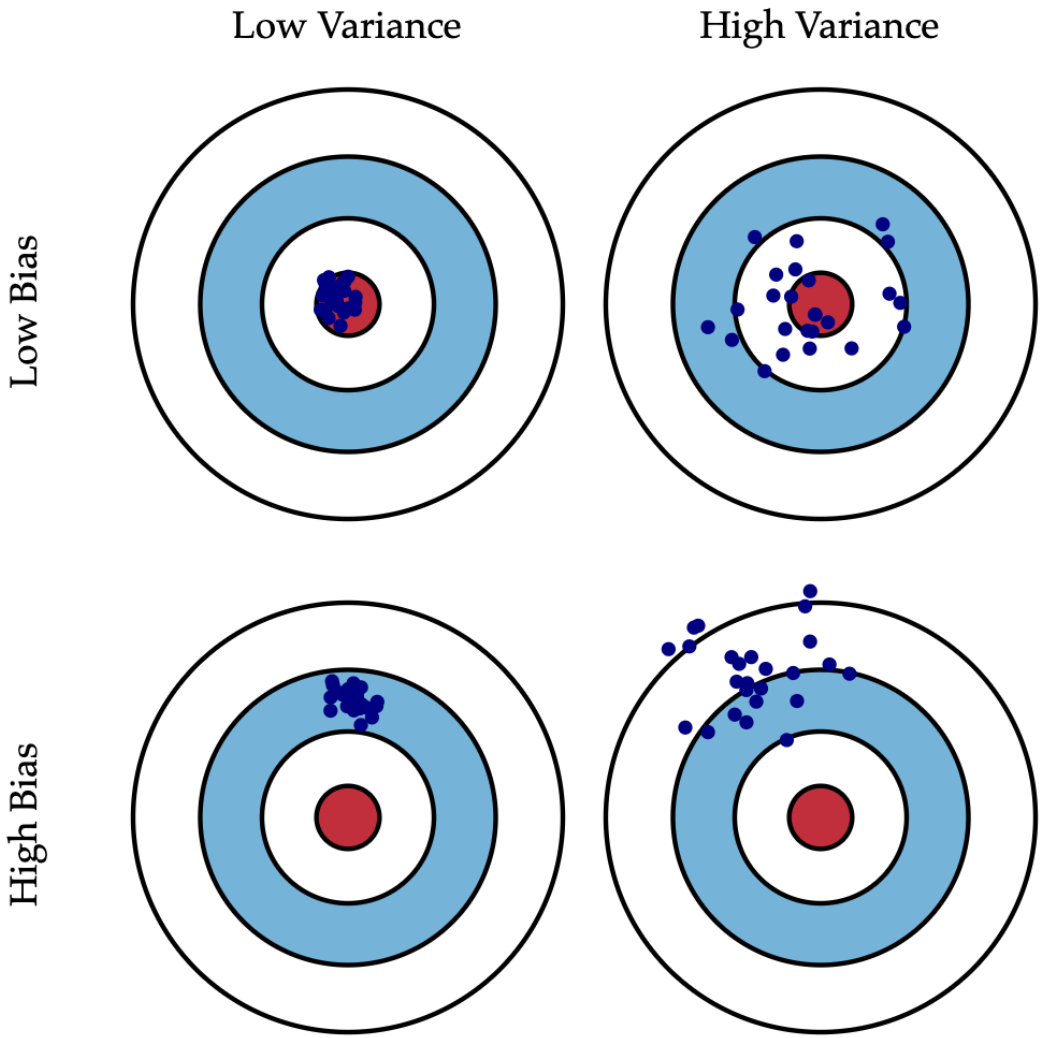
偏差-方差分解

$$
\begin{aligned}
E(f;D) &= \mathbb{E}_D\left[(f(\boldsymbol{x};D)-y_D)^2\right] \\
&= \mathbb{E}_D\left[(f(\boldsymbol{x};D)-\bar{f}(\boldsymbol{x})+\bar{f}(\boldsymbol{x})-y_D)^2\right] \\
&= \mathbb{E}_D\left[(f(\boldsymbol{x};D)-\bar{f}(\boldsymbol{x}))^2\right]+\mathbb{E}_D\left[(\bar{f}(\boldsymbol{x})-y_D)^2\right] \\
&\quad +\mathbb{E}_D\left[2\left(f(\boldsymbol{x};D)-\bar{f}(\boldsymbol{x})\right)\left(\bar{f}(\boldsymbol{x})-y_D\right)\right] \\
&= \mathbb{E}_D\left[(f(\boldsymbol{x};D)-\bar{f}(\boldsymbol{x}))^2\right]+\mathbb{E}_D\left[(\bar{f}(\boldsymbol{x})-y_D)^2\right] \\
&= \mathbb{E}_D\left[(f(\boldsymbol{x};D)-\bar{f}(\boldsymbol{x}))^2\right]+\mathbb{E}_D\left[(\bar{f}(\boldsymbol{x})-y+y-y_D)^2\right] \\
&= \mathbb{E}_D\left[(f(\boldsymbol{x};D)-\bar{f}(\boldsymbol{x}))^2\right]+\mathbb{E}_D\left[(\bar{f}(\boldsymbol{x})-y)^2\right]+\mathbb{E}_D\left[(y-y_D)^2\right] \\
&\quad +2\mathbb{E}_D\left[(\bar{f}(\boldsymbol{x})-y)(y-y_D)\right] \\
&= \mathbb{E}_D\left[(f(\boldsymbol{x};D)-\bar{f}(\boldsymbol{x}))^2\right]+(\bar{f}(\boldsymbol{x})-y)^2+\mathbb{E}_D\left[(y_D-y)^2\right],
\end{aligned}
$$

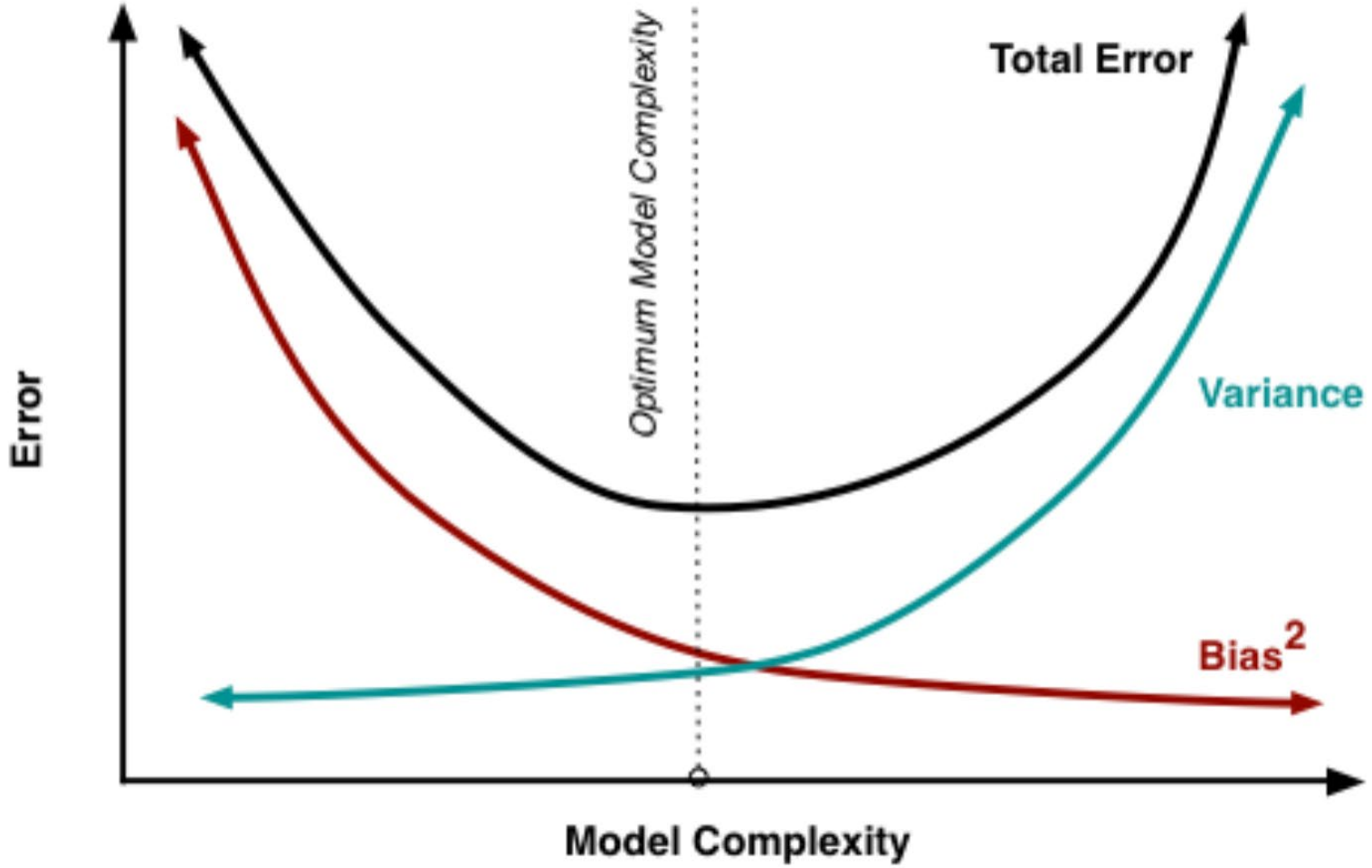
知乎 @Microstrong

01 理论——方差与偏差

理解方差和偏差



01 理论——方差与偏差


Error
Optimum Model Complexity
Total Error
Variance
Bias^2
Model Complexity

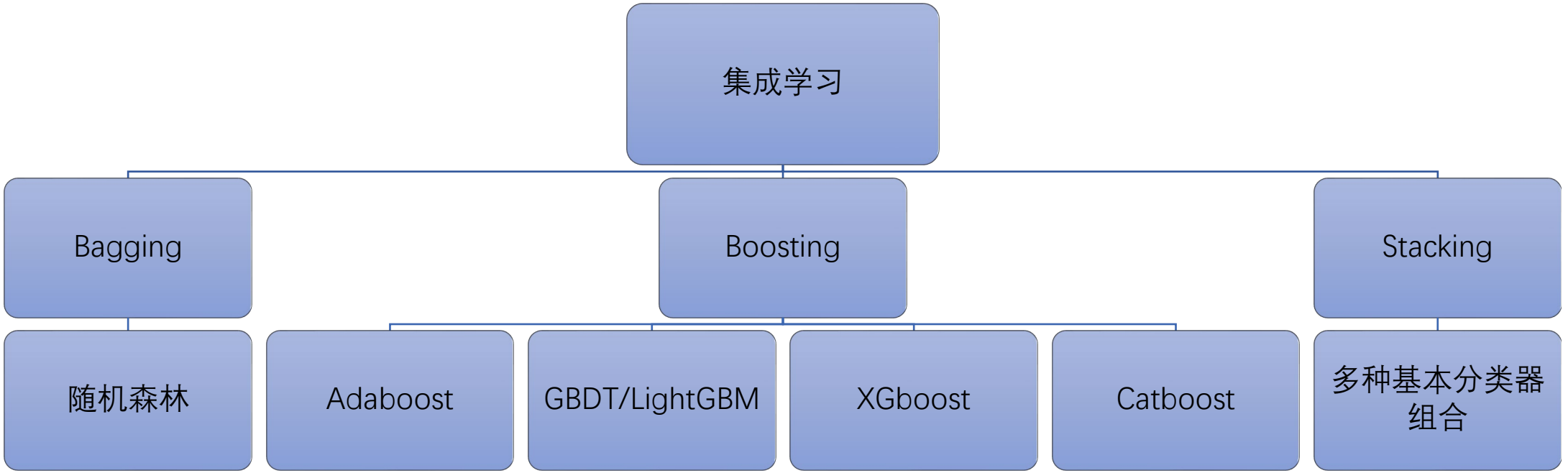
01 理论——方差与偏差

整体思路：首先，要知道偏差和方差是无法完全避免的，只能尽量减少其影响。

1. 在避免偏差时，**需尽量选择正确的模型**，一个非线性问题而我们一直用线性模型去解决，那无论如何，高偏差是无法避免的。
2. 有了正确的模型，我们还要**慎重选择数据集的大小**，通常数据集越大越好，但大到数据集已经对整体所有数据有了一定的代表性后，再多的数据已经不能提升模型了，反而会带来**计算量的增加**。而训练数据**太小**一定是不好的，这会带来**过拟合**，模型复杂度太高，方差很大，不同数据集训练出来的模型变化非常大。
3. 最后，要选择**合适的模型复杂度**，复杂度高的模型通常对训练数据有很好的拟合能力。

01 理论——集成学习介绍

集成学习 (Ensemble Learning) 算法的基本思想就是将多个分类器组合，从而实现一个预测效果更好的集成分类器。

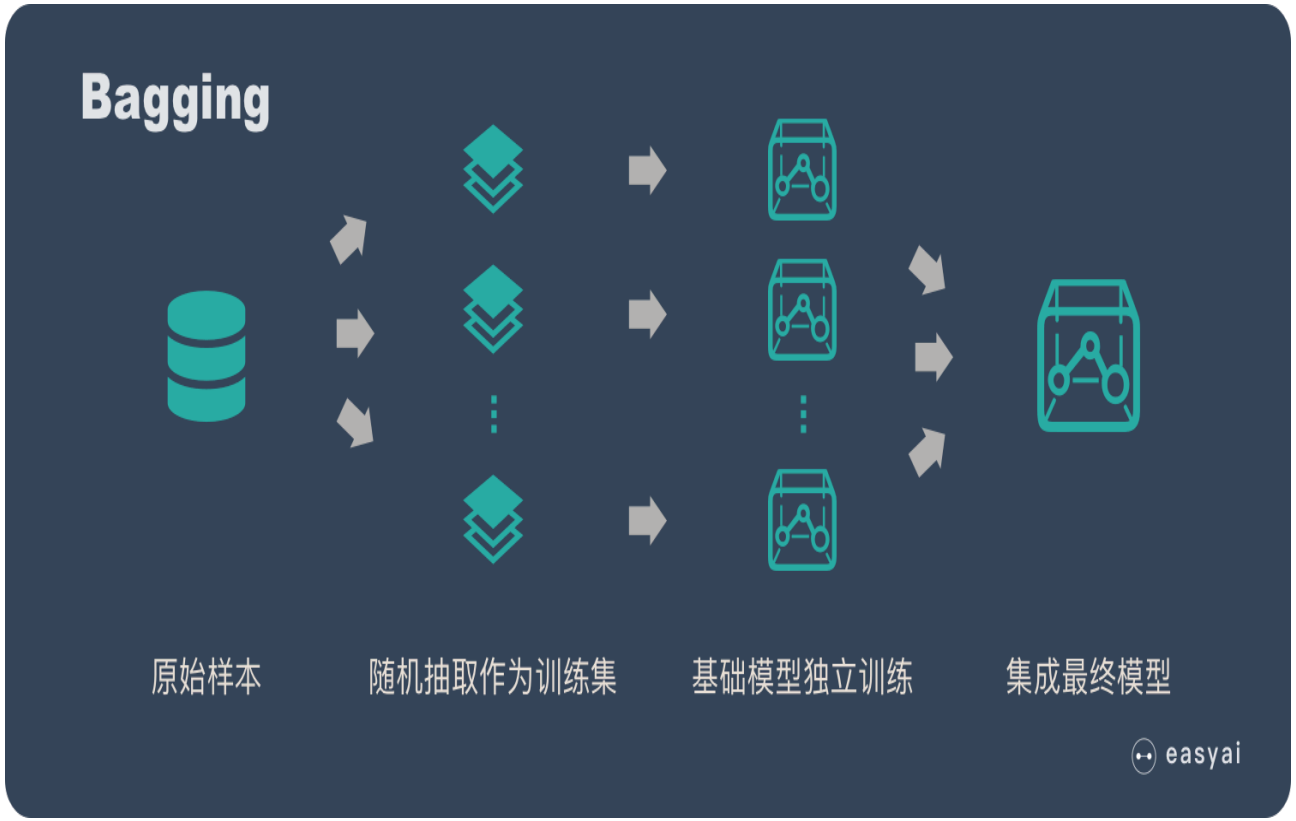


01 理论——Bagging

Bagging 的核心思路是 — — 民主。

Bagging 的思路是所有基础模型都一致对待，每个基础模型手里都只有一票。然后使用民主投票的方式得到最终的结果。

大部分情况下，**经过 bagging 得到的结果方差（variance）更小**。



1.从原始样本集中抽取训练集。每轮从原始样本集中使用**Bootstraping**的方法抽取n个训练样本（在训练集中，有些样本可能被多次抽取到，而有些样本可能一次都没有被抽中）。共进行k轮抽取，得到k个训练集。（k个训练集之间是相互独立的）

2.每次使用一个训练集得到一个模型，**k个训练集共得到k个模型**。（注：这里并没有具体的分类算法或回归方法，我们可以根据具体问题采用不同的分类或回归方法，如决策树、感知器等）

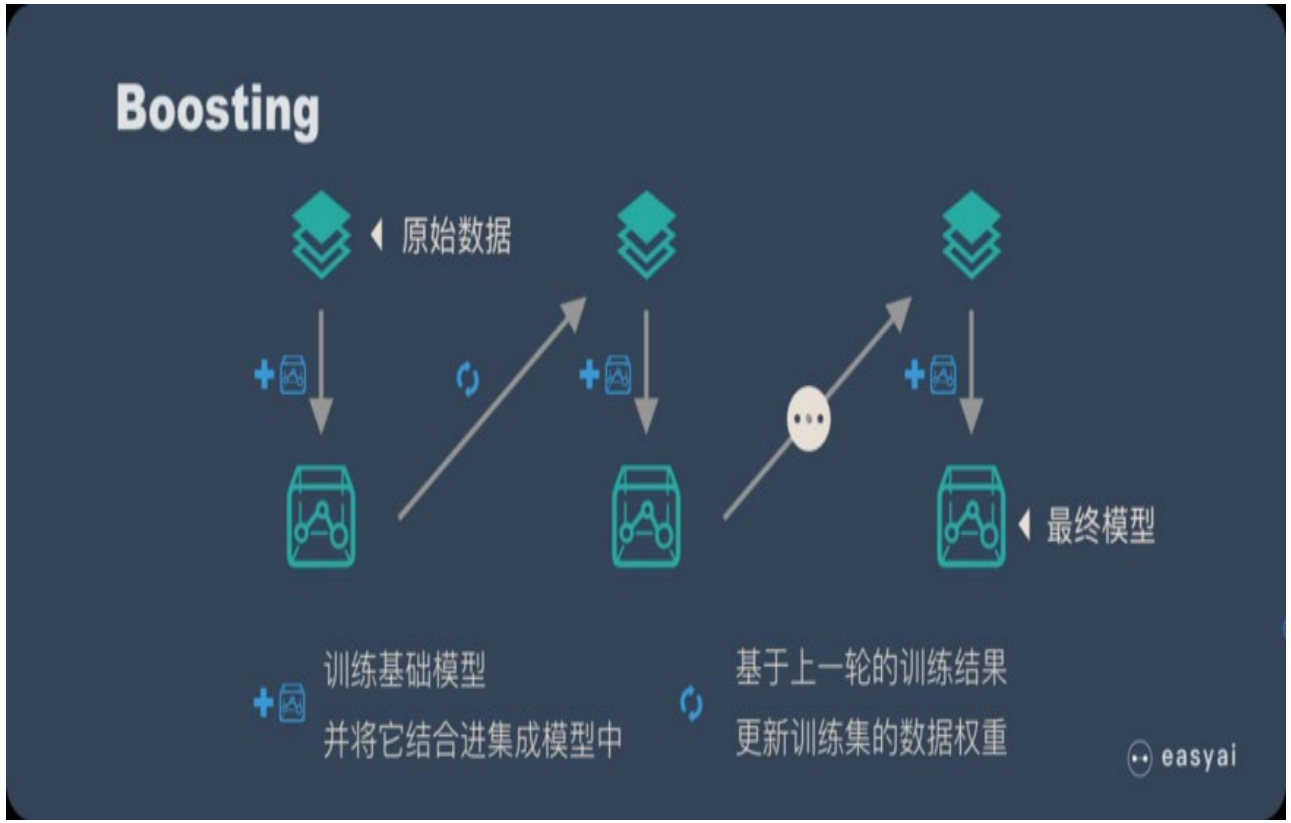
3.对分类问题：将上步得到的k个模型**采用投票的方式得到分类结果**；对回归问题，计算上述模型的均值作为最后的结果。（所有模型的重要性相同）

01 理论——Boosting

Boosting 的核心思路是 — — 挑选精英。

Boosting 和 bagging 最本质的差别在于他对基础模型不是一致对待的，而是经过不停的考验和筛选来挑选出「精英」，然后给精英更多的投票权，表现不好的基础模型则给较少的投票权，然后综合所有人的投票得到最终结果。

大部分情况下，**经过 boosting 得到的结果偏差更小**。



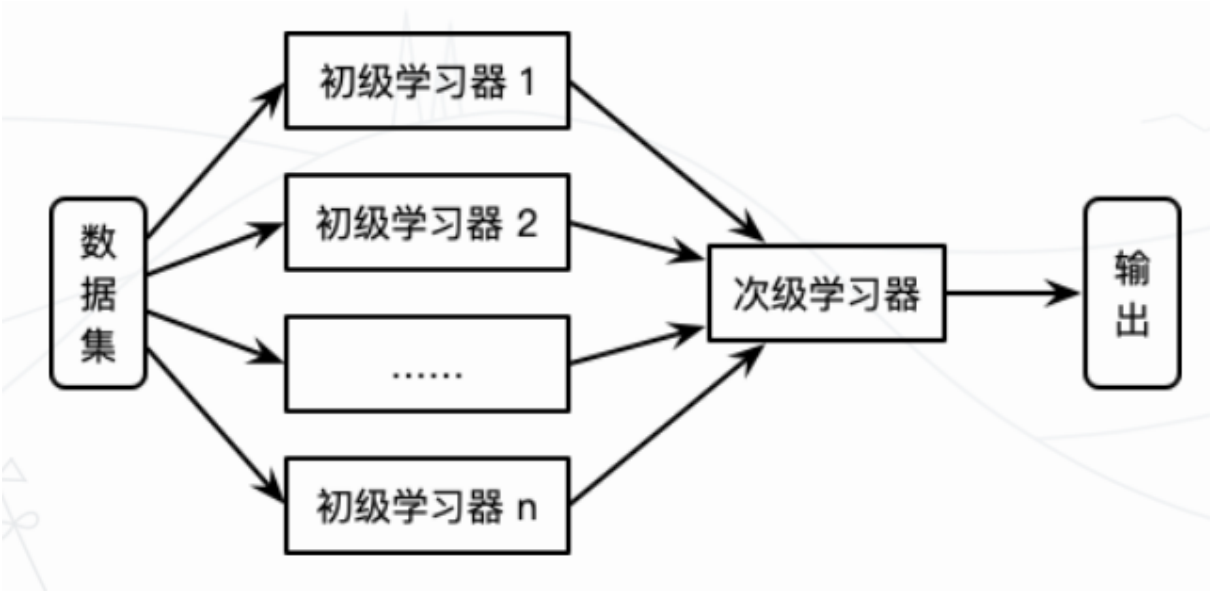
1.通过加法模型将**基础模型进行线性的组合**。
2.每一轮训练都**提升那些错误率小的基础模型权重，同时减小错误率高的模型权重**。
3.在每一轮改变**训练数据的权值或概率分布**，通过提高那些在前一轮被弱分类器分错样例的权值，减小前一轮分对样例的权值，来使得分类器对误分的数据有较好的效果。

01 理论——Stacking

Stacking 的核心思路：

- 利用初级学习算法**对原始数据集进行学习**，同时生成一个**新的数据集**。
- 根据从初级学习算法生成的新数据集，利用**次级学习算法**学习并得到最终的输出。

注：对于初级学习器，可以是相同类型也可以是不同类型的。在新的数据集中，初级学习器的输出被用作次级学习器的输入特征，初始样本的标记仍被用作次级学习器学习样本的标记。Stacking 算法的流程如下图所示：



01 理论——模型对比

Bagging和Boosting的区别总结如下：

1. **样本选择上：** Bagging方法的训练集是从原始集中**有放回的选取**，所以从原始集中选出的各轮训练集之间是独立的；而Boosting方法需要**每一轮的训练集不变**，只是训练集中每个样本在分类器中的**权重发生变化**。而权值是根据上一轮的分类结果进行调整

2. **样例权重上：** Bagging方法使用均匀取样，所以**每个样本的权重相等**；而Boosting方法根据错误率不断调整样本的权值，**错误率越大则权重越大**

3. **预测函数上：** Bagging方法中所有**预测函数的权重相等**；而Boosting方法中**每个弱分类器都有相应的权重**，对于分类误差小的分类器会有更大的权重

4. **并行计算上：** Bagging方法中各个预测函数**可以并行生成**；而Boosting方法各个预测函数**只能顺序生成**，因为后一个模型参数需要前一轮模型的结果。

01 理论——模型介绍

随机森林:
https://easyai.tech/ai-definition/random-forest/

AdaBoost: (提升错分数据点的权重)
https://easyai.tech/ai-definition/adaboost/

GBDT: (拟合梯度的残差)
https://zhuanlan.zhihu.com/p/45145899

LightGBM: (基于histogram的决策树算法\直方图做差加速)
https://blog.csdn.net/wuzhongqiang/article/details/105350579

XGBoost: (基分类器的选择\二阶泰勒展开\正则项\列抽样\缺失值处理\)
https://blog.csdn.net/wuzhongqiang/article/details/104854890

Catboost: (分类特征 \预测偏移)
https://mp.weixin.qq.com/s/xloTLr5NJBgBspMQtxPoFA

02

实验部分

02 实验——背景介绍

数据背景：

通常大部分数据公司会开展一些数据相关的培训课程，一方面可以为公司获得一定的名声和收益，另外一方面数据公司可以通过培训来寻找有意向留职的员工，本课程论文旨在利用集成学习相关方法，通过对参加培训者的相关特征，预测培训者寻找新工作或将为公司工作的可能性。

02 实验——数据介绍

数据来源：

本文所用数据来自https://www.kaggle.com,包含19158条数据,共14个变量。具体变量及相关信息如下表所示：

变量名	解释	变量类型	举例
enrollee_id	培训者的唯一ID	————	
city	城市代码	离散型	city_103、city_140
city_development_index	城市发展指数	连续型	0、92、0.776
gender	性别	离散型	Male\Female
relevent_experience	培训者的相关经验	离散型	Has relevent experience\ No relevent experience
enrolled_university	已注册的大学课程类型	离散型	no_enrollment\ Part time course
education_level	受教育程度	离散型	Graduate\ Masters
major_discipline	培训者的专业	离散型	STEM\ Business Degree
experience	工作经验	离散型	15\17\2\5
company_size	当前公司的规模	连续型	50-90\10000+
company_type	当前公司的类型	离散型	Pvt Ltd\ Funded Startup
last_new_job	上一份工作与当前工作的时间差	连续型	4\1\3\ never
training_hours	已完成培训的时长	连续型	36\47\8
target	是否找工作	0-1变量	

02 实验——数据预处理

数据类型转换：

```
def gender_to_numeric(x):
    if x=='Female': return 2
    if x=='Male':   return 1
    if x=='Other':   return 0

def rel_experience(x):
    if x=='Has relevent experience': return 1
    if x=='No relevent experience':   return 0

def enrollment(x):
    if x=='no_enrollment'    : return 0
    if x=='Full time course':   return 1
    if x=='Part time course':   return 2

def edu_level(x):
    if x=='Graduate'        :   return 0
    if x=='Masters'         :   return 1
    if x=='High School'     :   return 2
    if x=='Phd'             :   return 3
    if x=='Primary School'  :   return 4
```

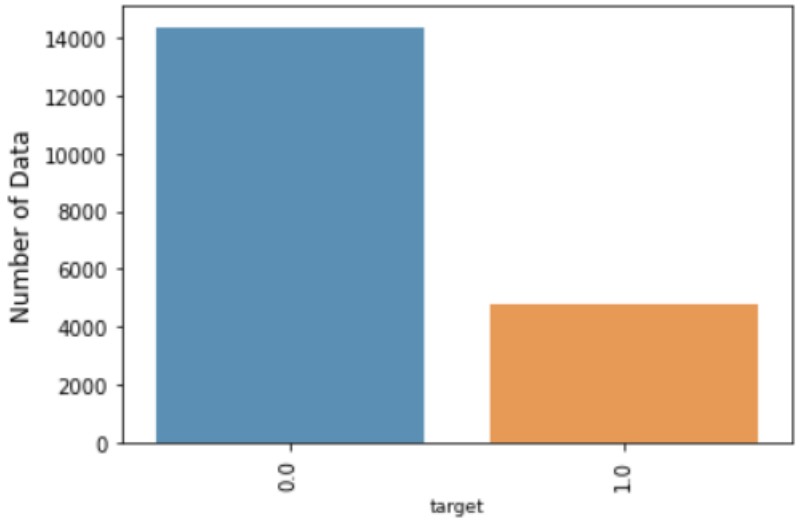
```
train['gender'] = train['gender'].apply(gender_to_numeric)
train['relevent_experience'] = train['relevent_experience'].apply(rel_experience)
train['enrolled_university'] = train['enrolled_university'].apply(enrollment)
train['education_level'] = train['education_level'].apply(edu_level)
train['major_discipline'] = train['major_discipline'].apply(major)
train['experience'] = train['experience'].apply(experience)
train['company_type'] = train['company_type'].apply(company_t)
train['company_size'] = train['company_size'].apply(company_s)
train['last_new_job'] = train['last_new_job'].apply(last_job)
train['city'] = train['city'].apply(city)
```

02 实验——数据预处理

缺失值处理：

```
train['gender'] = train['gender'].fillna((train['gender'].mean()))
train['enrolled_university'] = train['enrolled_university'].fillna((train['enrolled_university'].mean()))
train['major_discipline'] = train['major_discipline'].fillna((train['major_discipline'].mean()))
train['company_size'] = train['company_size'].fillna((train['company_size'].mean()))
train['company_type'] = train['company_type'].fillna((train['company_type'].mean()))
train['education_level'] = train['education_level'].fillna((train['education_level'].mean()))
train['experience'] = train['experience'].fillna((train['experience'].mean()))
train['last_new_job'] = train['last_new_job'].fillna((train['last_new_job'].mean()))
```

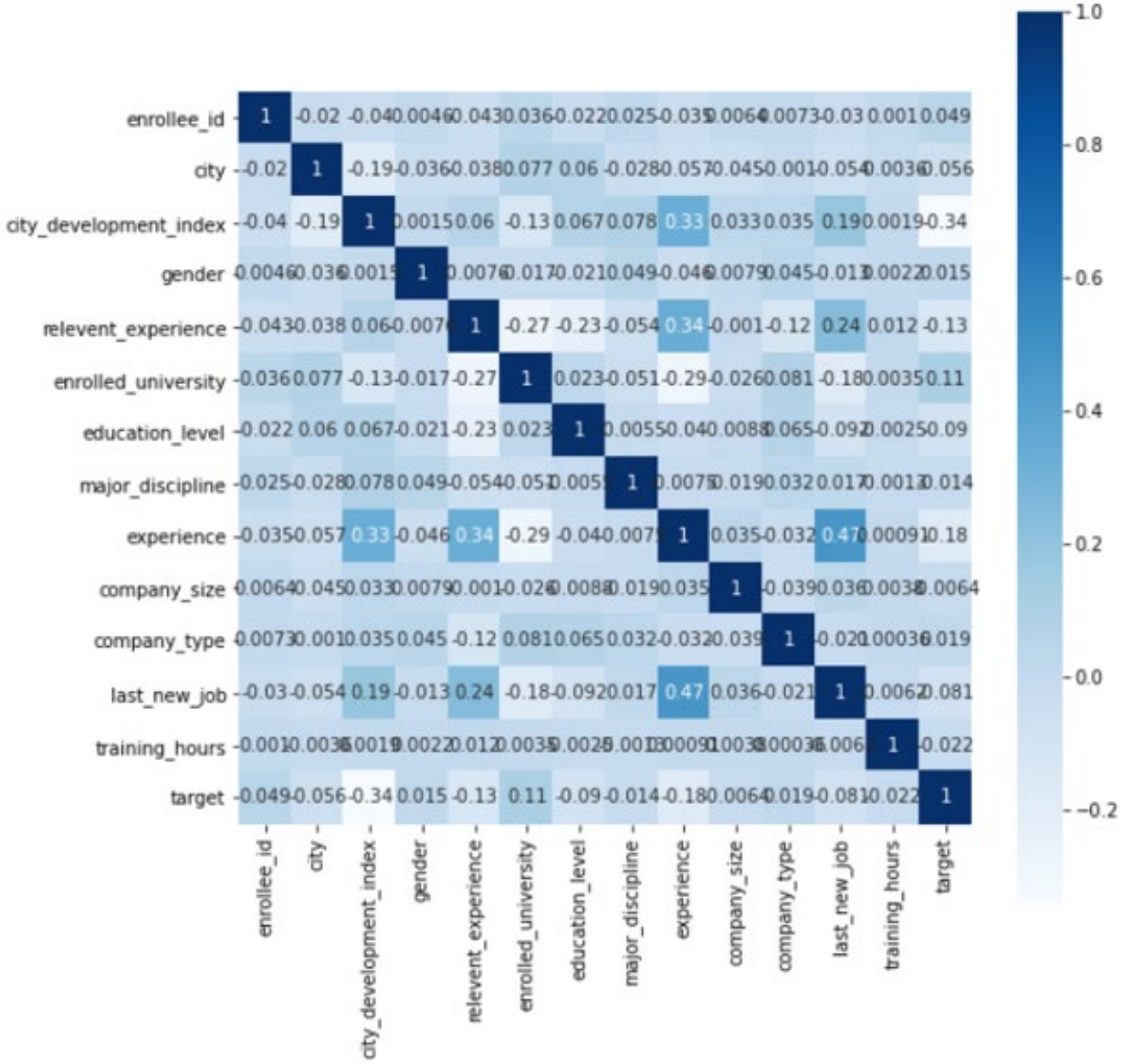
02 实验——初步探索数据



样本不平衡查看：

正负例样本比例大约在1：3左右，样本不平衡现象不严重，而且也符合现实情况，因此未对样本做不平衡处理。

02 实验——初步探索数据



变量相关性查看：

变量之间相关性基本都在**0.3**以下，其中‘last_new_job’与‘experience’相关性达到**0.47**，但也是可接受范围以内，因此未对变量做相关性处理。

02 实验——训练集划分及标准化处理

划分训练集与测试集：

以15%的数据为测试集，85%的数据为训练集划分数据

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
ss = StandardScaler()
Y = train['target']
X = train.drop(columns=['target'])
X = ss.fit_transform(X)#特征标准化
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.15, random_state=7)
```

02 实验

实验一：留职预测实验

选用了常用的六种集成算法模型对数据进行训练，预测培训者是否会在培训后留职。

实验二：模型对比实验

通过以集成方法与基分类器的组合对比不同方法下模型的准确性、AUC、运行时间等

02 实验一

未调参实验结果:

	随机森林	Adaboost	GBDT	LightGBM	XGboost	Catboost
准确率	0.7773	0.7676	0.7791	0.7832	0.7707	0.7836
AUC	0.6847	0.5978	0.6800	0.7157	0.6726	0.6990
召回率	0.4979	0.2552	0.4800	0.5793	0.4745	0.5283
结果截图	准确率ACC: 0.7773 ********** 混淆矩阵: [[1873 276] [364 361]] ********** AUC: 0.6847 ********** 召回率Recall: 0.4979 ********** F1-score: 0.5301	准确率ACC: 0.7676 ********** 混淆矩阵: [[2021 128] [540 185]] ********** AUC: 0.5978 ********** 召回率Recall: 0.2552 ********** F1-score: 0.3565	准确率ACC: 0.7791 ********** 混淆矩阵 [[1891 258] [377 348]] ********** AUC: 0.6800 ********** 召回率Recall: 0.4800 ********** F1-score: 0.5229	ACC: 0.7832 ********** 混淆矩阵: [[1831 318] [305 420]] ********** AUC: 0.7157 ********** Recall: 0.5793 ********** F1-score: 0.5742	ACC: 0.7707 ********** 混淆矩阵: [[1871 278] [381 344]] ********** AUC: 0.6726 ********** Recall: 0.4745 ********** F1-score: 0.5108	ACC: 0.7836 ********** confusion_matrix: [[1869 280] [342 383]] ********** AUC: 0.6990 ********** Recall: 0.5283 ********** F1-score: 0.5519

02 实验一

调参说明：

（1）调参目的：偏差和方差的协调

（2）一般来说集成模型的重要参数分成两个部分——框架参数与基模型参数。

模型框架参数：

弱学习器的个数、学习率、泛化能力参数等

基模型参数：

弱学习器的基本参数，例如随机森林中的决策树树的最大特征数、最大树深、叶子节点最少样本数等

（3）调参方法：网格搜索

02 实验一

模型重要参数说明：

	随机森林	Adaboost	GBDT	LightGBM	XGboost	Catboost
重要参数	‘n_estimators‘:树的数目 ‘max_depth‘最大树深 'min_samples_split’:内部节点再划分所需最小样本数 'min_samples_leaf’:叶子节点最少样本数 'max_features’:最大特征数	‘base_estimator’:基分类器 'algorithm’:弱学习器权重的度量 'n_estimators’: 弱学习器的最大迭代次数 'learning_rate‘:学习率 当选择基分类器为Cart树时基分类器参数与随机森林相同	‘n_estimators’:最大迭代次数 'subsample':子采样的比例 'max_depth’:树深 ‘min_samples_split’:同前 ‘min_samples_leaf’:同前 ‘max_features’:同前	'num_leaves’:叶子节点数 'min_data_in_leaf’:叶子可能具有的最小记录数 'bagging_fraction’:每次迭代时用的数据比例 'lambda_l1’:正则化参数 'lambda_l2’:正则化参数 ':	'max_depth’:最大树深 ‘min_child_weight’:最小叶子节点样本权重和 'gamma‘: 节点分裂所需的最小损失函数下降值 'subsample‘:随机采样的比例 'colsample_bytree‘:每棵随机采样的列数的占比	‘depth’:树深 'learning_rate‘:学习率 'l2_leaf_reg’:L2正则化系数 'iterations’:最大迭代次数

02 实验一

调参后结果

	随机森林	Adaboost	GBDT	LightGBM	XGboost	Catboost
准确率	0.7909(0.7773)	0.7752(0.7676)	0.7804(0.7791)	0.7878(0.7832)	0.7881(0.7707)	0.7864(0.7836)
AUC	0.7185(0.6847)	0.6235(0.5978)	0.7106(0.6800)	0.7137(0.7157)	0.7153(0.6726)	0.7054(0.6990)
召回率	0.5724(0.4979)	0.3172(0.2552)	0.5697(0.4800)	0.5641(0.5793)	0.5683(0.4745)	0.5421(0.5283)
运行时间	1.34913s	1.99675s	1.35414s	0.57785s	0.98348s	0.94517s
结果截图	准确率ACC: 0.7909 ********** 混淆矩阵confusion_matrix: [[1858 291] [310 415]] ********** AUC: 0.7185 ********** 召回率Recall: 0.5724 ********** F1-score: 0.5800	准确率ACC: 0.7752 ********** 混淆矩阵: [[1998 151] [495 230]] ********** AUC: 0.6235 ********** 召回率Recall: 0.3172 ********** F1-score: 0.4159	ACC: 0.7804 ********** confusion_matrix: [[1830 319] [312 413]] ********** AUC: 0.7106 ********** Recall: 0.5697 ********** F1-score: 0.5669	ACC: 0.7878 ********** 混淆矩阵 [[1855 294] [316 409]] ********** AUC: 0.7137 ********** Recall: 0.5641 ********** F1-score: 0.5728	ACC: 0.7881 ********** confusion_matrix: [[1853 296] [313 412]] ********** AUC: 0.7153 ********** Recall: 0.5683 ********** F1-score: 0.5750	ACC: 0.7864 ********** confusion_matrix: [[1867 282] [332 393]] ********** AUC: 0.7054 ********** Recall: 0.5421 ********** F1-score: 0.5614

02 实验二

（1）模型对比实验

通过以集成方法与基分类器的组合对比不同方法下模型的准确性、AUC、运行时间等

（2）比较策略：

集成方法选择:

Bagging、Adaboost、Stacking

基分类器选择：

SVM、贝叶斯分类器、逻辑回归、决策树、感知机

采取两两组合的方式生成共15个模型进行比较

（其中因为Stacking方法较为特殊，因此在选择其分类器时将其中一种基分类器作为第二层分类器即最终分类器，其他分类器作为第一层分类器）

（3）其他说明：

- **因代码能力有限，实验过程中采用的包均为sklearn上面的已有模型包**
- **因时间有限，并未对模型进行调参，仅做初步比较**

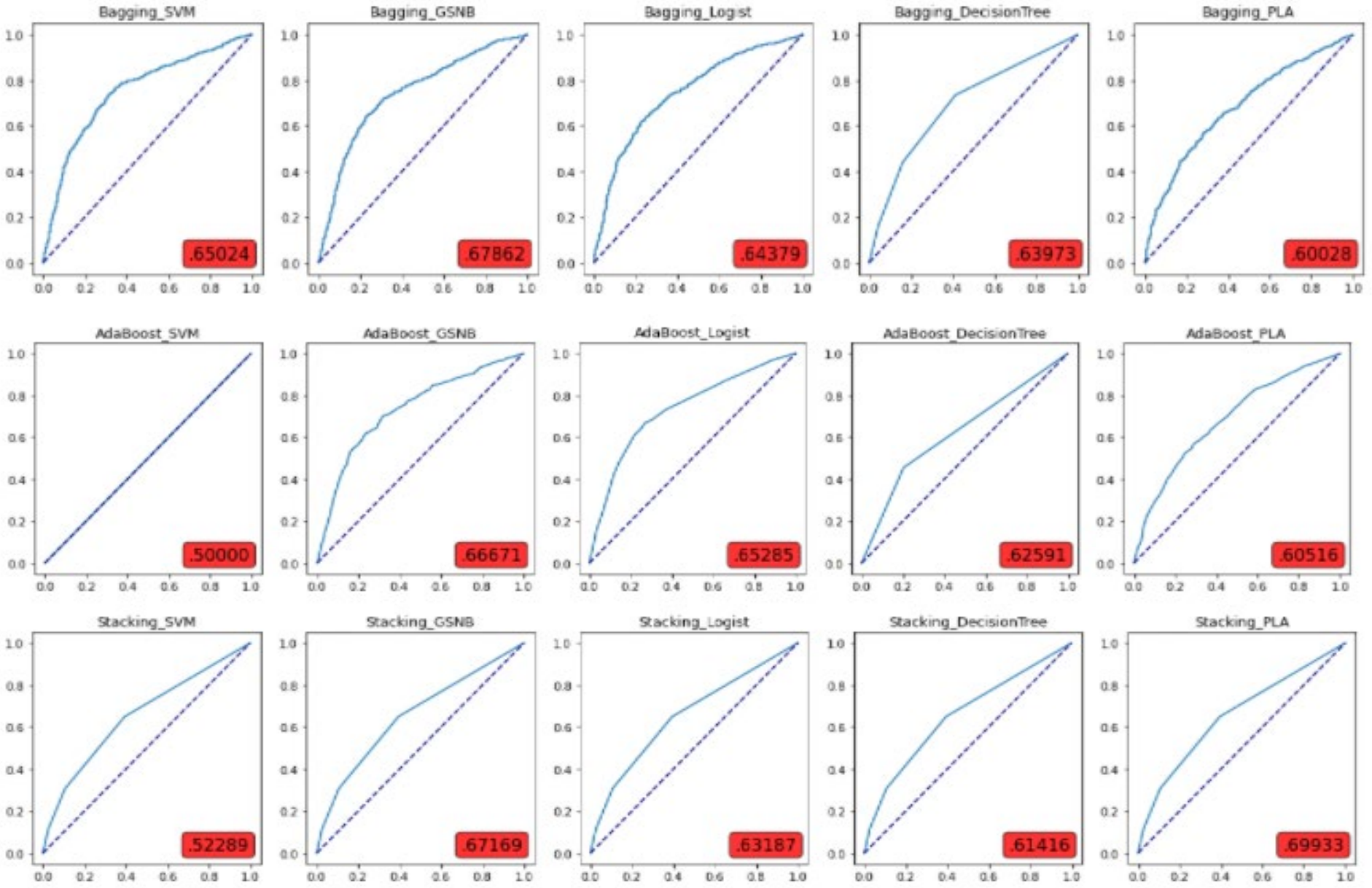
02 实验二

实验结果截图：

Bagging_SVM	Bagging_GSNB	Bagging_Logist	Bagging_DecisionTree	Bagging_PLA
准确率ACC：0.7784 召回率Recall：0.3945 F1-score：0.4731 运行时间是：28.50283s	准确率ACC：0.7575 召回率Recall：0.5200 F1-score：0.5196 运行时间是：0.18076s	准确率ACC：0.7714 召回率Recall：0.3738 F1-score：0.4520 运行时间是：0.17091s	准确率ACC：0.7630 召回率Recall：0.3876 F1-score：0.4521 运行时间是：0.21018s	准确率ACC：0.7557 召回率Recall：0.2828 F1-score：0.3687 运行时间是：0.18478s
AdaBoost_SVM	**AdaBoost_GSNB**	**AdaBoost_Logist**	**AdaBoost_DecisionTree**	**AdaBoost_PLA**
准确率ACC：0.7477 召回率Recall：0.0000 F1-score：0.0000 运行时间是：98.56738s	准确率ACC：0.7575 召回率Recall：0.4676 F1-score：0.4931 运行时间是：0.15050s	准确率ACC：0.7658 召回率Recall：0.4248 F1-score：0.4779 运行时间是：0.09646s	准确率ACC：0.7105 召回率Recall：0.4552 F1-score：0.4424 运行时间是：0.08846s	准确率ACC：0.7328 召回率Recall：0.3476 F1-score：0.3962 运行时间是：0.09782s
Stacking_SVM	**Stacking_GSNB**	**Stacking_Logist**	**Stacking_DecisionTree**	**Stacking_PLA**
准确率ACC：0.7509 召回率Recall：0.0855 F1-score：0.1476 运行时间是：11.24583s	准确率ACC：0.7756 召回率Recall：0.4676 F1-score：0.5125 运行时间是：21.91703s	准确率ACC：0.7697 召回率Recall：0.3517 F1-score：0.4352 运行时间是：20.83815s	准确率ACC：0.7081 召回率Recall：0.4276 F1-score：0.4249 运行时间是：20.70788s	准确率ACC：0.4854 召回率Recall：0.9145 F1-score：0.4727 运行时间是：20.55499s

02 实验二

实验结果截图:



02 实验二

注：红色字体代表横向最优值，黄色背景代表纵向最优值

	Bagging_SVM	Bagging_GSNB	Bagging_Logist	Bagging_DecisionTree	Bagging_PLA
准确率	0.7784	0.7575	0.7714	0.763	0.7557
召回率	0.3945	0.52	0.3738	0.3876	0.2828
AUC	0.65024	0.67862	0.64379	0.63973	0.60028
运行时间	28.50283s	0.18076s	0.17091s	0.21018s	0.18478s
	AdaBoost_SVM	AdaBoost_GSNB	AdaBoost_Logist	AdaBoost_DecisionTree	AdaBoost_PLA
准确率	0.7477	0.7575	0.7658	0.7105	0.7328
召回率	0	0.4676	0.4248	0.4552	0.3476
AUC	0.5	0.66671	0.65285	0.62591	0.60516
运行时间	98.56738s	0.15050s	0.09646s	0.08846s	0.09782s
	Stacking_SVM	Stacking_GSNB	Stacking_Logist	Stacking_DecisionTree	Stacking_PLA
准确率	0.7509	0.7756	0.7697	0.7081	0.4854
召回率	0.0855	0.4676	0.3517	0.4276	0.9145
AUC	0.52289	0.67169	0.63187	0.61416	0.69933
运行时间	11.24583s	21.91703s	20.83815s	20.70788s	20.55499s