同济大学

TONGJI UNIVERSITY

实 验 报 告

实验名称: 网贷平台的风控建模与调优

课程名称: 网络数据风控技术

专　　业:	数据科学与大数据技术
学生姓名:	胡逸凡
学　　号:	2153592
指导老师:	叶晨
成　　绩:	
实验日期:	2023 年 11 月

目录

1 实验背景

信用贷（原蚂蚁借呗）是支付宝平台提供的小额信用贷款服务。互联网小额信用贷款平台（简称网贷平台）根据风险控制和准入标准对支付宝实名用户进行筛选，为筛选出的优质客户提供不同的借款额度。网贷平台的快速发展得益于互联网平台建立了自建的场景流量信用评分体系，形成了获客和贷款的闭环体验。本实验需要通过约 400 个数据维度评估用户当前的信用状态，结合脱敏处理过的样本数据，对贷款风险进行建模，预测用户 6 个月内的逾期率，为平台提供关键的决策依据。

2 问题分析与数据准备

2.1 问题分析

模型主要目标是进行违约预测，即判断借款人是否会违约（1 = 贷款违约，0 = 正常还款）。这是一个二元分类问题，需要建立一个预测模型来对每笔借款进行违约风险的评估，模型的输入为用户基本信息、网络行为、借款记录等，输出为用户违约的概率，通过设置概率阈值，从而判断违约 label 为 1 或 0。

2.2 数据说明

数据集和测试集均包含 3 个 GBK 格式的 csv 文件，且均包含 3 万条 Idx。观察数据可以发现，数据集中存在大量具有相似地理特征的样本，同时包含大量非数值型特征，并存在许多缺失值。由于特征数量庞大，需要进行特征工程，对包括但不限于地理位置和日期等非数值特征进行处理。另外，还需要选择不同的模型对数据集进行训练和预测，并对其效果进行评估。在此基础上，采用一系列模型融合的方法，以提高模型在训练集上的性能表现。

具体 csv 文件内容如下：

- Master.csv
  信贷客户的申报信息和部分第三方数据，以及需要预测的目标标签 target。每一行代表一笔借据，即一条成交借款样本，每个样本包含 200 多个字段。

a.idx：每笔借款的 unique key，可与另外 2 个文件里的 idx 相匹配
b.UserInfo_*：借款人特征字段
其中 UserInfo_2,，UserInfo_4，UserInfo_8，UserInfo_20 为城市信息，UserInfo_7，UserInfo_19 为省级信息。
c.WeblogInfo_*：网络行为字段
d.Education_Info*：学历学籍字段
e.ThirdParty_Info_PeriodN_*：第三方数据时间段 N 字段
f.SocialNetwork_*：社交网络字段
g.LinstingInfo：借款成交时间
h.Target：违约标签（1 = 贷款违约，0 = 正常还款）。测试集里不包含 target 字段。

- Log_Info.csv
信贷客户在网贷平台上的登陆记录
a.ListingInfo：借款成交时间
b.LogInfo1：操作代码
c.LogInfo2：操作类别
d.LogInfo3：登陆时间
e.idx：每一笔借款的 unique key

- Userupdate_Info.csv
部分客户在网贷平台上的修改记录
a.ListingInfo1：借款成交时间
b.UserupdateInfo1：修改内容
c.UserupdateInfo2：修改时间
d.idx：每一笔借款的 unique key

3 数据清洗与预处理

3.1 训练集与测试集融合

第一次尝试时，我首先对训练集和测试集分别进行数据清洗、预处理和特征工程，在处理完毕后进行 one-hot 编码时，由于训练集和测试集 shape 不一致，我进行了取交集的操作，使得 feature 个数大大减少的同时，忽略了很多相关性更高的 feature，效果并不理想，最高在 73.21 分。

之后，我在进行全部数据处理工作之前，首先将训练集和测试集融合，得到融合数据集 (pandas.DataFrame，60000×n，n 为 featrue 并集个数)，再对融合数据集进行数据处理以及 one-hot 编码，分数获得提升。

```
'''
读取测试集和训练集
'''
master = pd.concat([train_master, test_master], ignore_index=True)
userupdateinfo = pd.concat([train_userupdateinfo,test_userupdateinfo],
    ignore_index=True)
loginfo = pd.concat([train_loginfo,test_loginfo],ignore_index=True)
```

3.2 缺失值处理

首先统计 master 中各个 feature 的缺失值比例，如表一所示。观察到

表 1: The missing proportion of each feature in master.

feature	missing proportion(%)
WeblogInfo_1	96.77
WeblogInfo_3	96.77
UserInfo_12	63.03
UserInfo_13	63.03
UserInfo_11	63.03
WeblogInfo_20	26.83
WeblogInfo_21	10.24
WeblogInfo_19	9.88
WeblogInfo_2	5.53
WeblogInfo_4	5.50
WeblogInfo_5	5.50

WeblogInfo_1 与 WeblogInfo_3 几乎全部缺失，直接剔除。

对于之后的缺失值较高的 feature，通过绘制堆叠条形图，展示特征缺失和不缺失内部的 target 的不同取值的分布情况，可以观察到缺失值对 target 的影响大小。之后，对于数值型 feature（UserInfo_11，UserInfo_12，UserInfo_13 ），将缺失值设置为 2.0，需要注意，这三个特征的缺失值均大于 50%，直接设置为 2.0 是合理的；对与字符型 feature（WeblogInfo_19，WeblogInfo_20，WeblogInfo_21），将缺失值设置为” 不详”。

对于其他有缺失的 feature，我对数值型且数值具有一定意义的 UserInfo_14，UserInfo_15，UserInfo_16，UserInfo_17 采用均值填充，其余采用众数填充。

```
## 用众数填充缺失值
categoric_cols = ['UserInfo_1', 'UserInfo_2', 'UserInfo_3',
    'UserInfo_4', 'UserInfo_5', 'UserInfo_6', 'UserInfo_7',
                'UserInfo_8', 'UserInfo_9', 'UserInfo_11',
                    'UserInfo_12', 'UserInfo_13', 'UserInfo_19',
                    'UserInfo_20',
                'UserInfo_21', 'UserInfo_22', 'UserInfo_23',
                    'UserInfo_24', 'Education_Info1', 'Education_Info2',
                'Education_Info3', 'Education_Info4', 'Education_Info5',
                    'Education_Info6', 'Education_Info7',
                'Education_Info8', 'WeblogInfo_19', 'WeblogInfo_20',
                    'WeblogInfo_21', 'SocialNetwork_1',
                'SocialNetwork_2', 'SocialNetwork_7', 'SocialNetwork_12']
for col in categoric_cols:
    mode_cols = master[col].mode()[0]
    master.loc[(master[col].isnull(), col)] = mode_cols
## 用均值填充缺失值
numeric_cols = ['UserInfo_14', 'UserInfo_15', 'UserInfo_16',
     'UserInfo_17']
for col in master.columns:
    if col in numeric_cols and col != u'Idx' and col != u'target' and
        col != 'ListingInfo':
       mean_cols = master[col].mean()
       master.loc[(master[col].isnull(), col)] = mean_cols
```

最后，考虑各个 feature 的标准差，将标准差小于 0.1 的剔除，因为标准差小说明分布均匀对 target 影响小。

3.3 离群点处理

对于数值型特征，我并没有使用箱线图考察离群点，而是直接由 3σ 准则，将 3σ 以外的数据用均值代替。

```
std_threshold = 3
special = ['Idx', 'target']
```

```
for i in range(len(master)):
    # 数值类型且不是Idx和target
    if pd.api.types.is_numeric_dtype(i) and i not in special:
        mean = master[i].mean()
        std = master[i].std()
        outliers = master[abs(master[i] - mean) > std_threshold * std]
        master.loc[abs(master[i] - mean) > std_threshold * std, i] = mean
```

3.4 归一化处理

对于数值型特征，采用 scaling，放缩到 0-1 之间。

```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
numeric_features = all.select_dtypes(include=['float64', 'int64'])
numeric_features = numeric_features.drop(columns=['Idx'])
all[numeric_features.columns] =
    scaler.fit_transform(all[numeric_features.columns])
```

对于字符型特征，对于省市需要进行归一化处理（UserInfo_2,，UserInfo_4, UserInfo_8, UserInfo_20 为城市信息, UserInfo_7, UserInfo_19 为省级信息), 将省市中省、市、自治区删减掉, 只保留名称; 对与其他字符型有的需要去掉空格, 有的需要全部转为小写, 如 UserInfo_9, UserupdateInfo1。

```
import re
## 去掉空格
master['UserInfo_9'] = master['UserInfo_9'].apply(lambda x: x.strip())
## 去掉大小写
userupdateinfo['UserupdateInfo1'] =
    userupdateinfo['UserupdateInfo1'].apply(lambda x: x.lower())

def encodingstr(s):
    regex = re.compile(r'.+市')
    if regex.search(s):
        s = s[:-1]
        return s
    else:
```

```
        return s

def encodingstr2(s):
    special = ['内蒙古自治区', '宁夏回族自治区', '广西壮族自治区',
        '新疆维吾尔自治区']
    ret = ['内蒙古', '宁夏', '广西', '新疆']
    for i in range(len(special)):
        if s == special[i]:
            return ret[i]
    regex1 = re.compile(r'.+省')
    regex2 = re.compile(r'.+市')
    if regex1.search(s):
        s = s[:-1]
        return s
    elif regex2.search(s):
        s = s[:-1]
        return s
    else:
        return s

master['UserInfo_2'] = master['UserInfo_2'].apply(lambda x:
    encodingstr(x))
master['UserInfo_4'] = master['UserInfo_4'].apply(lambda x:
    encodingstr(x))
master['UserInfo_7'] = master['UserInfo_7'].apply(lambda x:
    encodingstr2(x))
master['UserInfo_19'] = master['UserInfo_19'].apply(lambda x:
    encodingstr2(x))
master['UserInfo_8'] = master['UserInfo_8'].apply(lambda x:
    encodingstr(x))
master['UserInfo_20'] = master['UserInfo_20'].apply(lambda x:
    encodingstr(x))
userupdateinfo.to_csv('./userupdateinfo.csv', index=False,
    encoding='utf-8')
```

通过对数字型和字符型的归一化，我们消除不同量纲的影响，减小了 one-hot 后 feature 的数量，提高模型的泛化能力，另外，归一化也降低异常值的影响。

4 特征工程

4.1 地理位置信息特征构建

地理位置信息特征构建主要分 2 个部分，其一是将具体城市转为一、二、三线等类别特征，其二是分析各个省份对 target 违约率的影响，将违约率较高（我定义了每个省份 grade，阈值设为大于 45）的省份构造二值特征。

通过转化为一线、新一线、二线、三线、四线城市和不详，大大减少了城市数量，在 ont-hot 编码后的特征维度大大减小。这样操作是合理的，因为如果使用具体城市进行 one-hot 编码，编码结果稀疏，模型无法很好地学习地理位置信息，另外，有部分城市出现数目少，比如 a 城市有 3 位客户均违约，若采用具体城市名称会造成严重过拟合。

```
# x线城市
def get_city_tier(city):
    first_tier = ['北京', '上海', '广州', '深圳']
    new_first_tier = [...]
    second_tier = [...]
    third_tier = [...]
    fourth_tier = [...] # 具体在报告中省略

    if city in first_tier:
        return 1
    elif city in new_first_tier:
        return 2
    elif city in second_tier:
        return 3
    elif city in third_tier:
        return 4
    elif city in fourth_tier:
        return 5
    elif city == '不详':
        return 6
    return None

city_col = ['UserInfo_2', 'UserInfo_4', 'UserInfo_8', 'UserInfo_20']
for col in city_col:
    master[col] = master[col].apply(get_city_tier)
```

```
    mode_value = master[col].mode()[0]
    master[col].fillna(mode_value, inplace=True)
```

通过将违约率超过阈值的省份构造新的二值特征（0= 是该省，1= 不是该省），我们更好的体现了省的地理信息对 target 的影响。

这里需要注意的是，起初我并没有构造新的二值特征，而是直接将省份替换为该省的违约率得分，从而把字符型 feature 转换为数值型 feature，在验证集上效果较好，但是在测试集效果不佳，原因是直接替换为数值会造成严重过拟合，比如 b 省 2 位客户中 1 位违约，则将该省替换为 50（违约率 0.5），在测试集上使得 b 省客户的违约率预测更偏向违约的可能。

```
# 是否为**省
my_alpha = 45

province_col = ['UserInfo_7', 'UserInfo_19']
province_info = pd.DataFrame(columns=['Province', 'y_sum', 'num',
    'grade'])

co = master['UserInfo_7']
sum_y = master.groupby(co)['target'].sum().reset_index()
num = master.groupby(co).size().reset_index()
province_info['Province'] = master.groupby(co).groups.keys()
province_info['y_sum'] = sum_y['target'].astype(int)
province_info['num'] = num[0].astype(int)
province_info['grade'] = (province_info['y_sum'] / province_info['num']
    * 1000).astype(int)
# province_info['grade'] = province_info['y_sum'] / province_info['num']

filtered_province_info = province_info[province_info['grade'] >
    my_alpha]
province_names = filtered_province_info['Province']
# 遍历每个省份名称，如果 'UserInfo_7'
    列的值等于该省份名称，对应列设置为1，否则为0
for col in province_col:
    col = master[col]
    for province in province_names:
        column_name = f'is_{province}' # 根据省份名称创建列名
        master[column_name] = (col == province).astype(int)
```

4.2 成交时间特征构建

主要考虑处理时间特征，也主要分为两部分。其一直接对时间处理，将月、日提出为新特征；其二考虑时间的隐藏信息，构建用户信息更新的次数、用户信息更新的种类数、用户分几次更新、用户借款成交与信息更新时间跨度，总共 4 个新特征。通过这两步，使得时间特征离散化。

考虑月、日的特征可以寻找与具体时间有关的周期性，以及本实验中我们无法通过事件驱动判断违约率是否会突然升降，只能通过具体时间反映现实世界中可能发生的事件造成的影响。

```
# 把月、日单独拎出来，放到3列中
master['month'] = pd.DatetimeIndex(master.ListingInfo).month
master['day'] = pd.DatetimeIndex(master.ListingInfo).day
master['day'].head()
master.drop(['ListingInfo'], axis=1, inplace=True)
master['target'] = master['target'].astype(str)
master.to_csv('./master.csv', index=False, encoding='utf-8')
```

新建与实践有关的隐藏信息，**在我看来，也是对用户的分类**，以考察违约率。它通过考察用户 4 个方面的行为，达成此目的。

```
from collections import defaultdict
import datetime as dt

## userupdateinfo表
userupdate_info_number = defaultdict(list) ### 用户信息更新的次数
userupdate_info_category = defaultdict(set) ###用户信息更新的种类数
userupdate_info_times = defaultdict(list) ### 用户分几次更新
userupdate_info_date = defaultdict(list) ####
    用户借款成交与信息更新时间跨度

with open('./userupdateinfo.csv', 'r') as f:
    f.readline()
    for line in f.readlines():
        cols = line.strip().split(",") ### cols 是list结果
        userupdate_info_date[cols[0]].append(cols[1])
        userupdate_info_number[cols[0]].append(cols[2])
        userupdate_info_category[cols[0]].add(cols[2])
        userupdate_info_times[cols[0]].append(cols[3])
```

```
    print(u'提取信息完成')

userupdate_info_number_ = defaultdict(int)
userupdate_info_category_ = defaultdict(int)
userupdate_info_times_ = defaultdict(int)
userupdate_info_date_ = defaultdict(int)
```

4.3 类别特征编码

采用 one-hot 编码对处理完成的数据集进行展开，展开为每个特征的每个类别的独立二值特征，从而将分类信息编码到特征中。这样较好地保留原始类别信息，提升模型识别和理解不同的类别的能力以提升分类准确度。ont-hot 编码后得到 4040 维的特征。

```
all['Idx'] = all['Idx'].astype(np.int64)
all['target'].fillna(0, inplace=True) # 填充测试集target的缺失值
all['target'] = all['target'].astype(np.int64)
all = pd.get_dummies(all)
```

之后，对这些特征进行特征选择（在下一部分重点阐释），再对模型数据进行 scaling，提升模型的泛化能力。

4.4 特征选择

特征选择是机器学习中的关键步骤，它涉及确定哪些特征对于建立模型和取得良好性能是最重要的。我采用将前一步得到的 4040 维的数据输入 XGBoost 得到 feature importances，从而通过 topN 方法进行特征选择。特征重要性如图所示。

```
feature_imp = modelfit(xgb1, train_data, y_train)

colu = train_data.columns
n = N # topN
top_n_features = feature_imp.index[:n]
top_n_features = [int(feature[1:]) for feature in top_n_features]
top_n_features = colu[top_n_features]
train_subset2 = train_data[top_n_features]
```

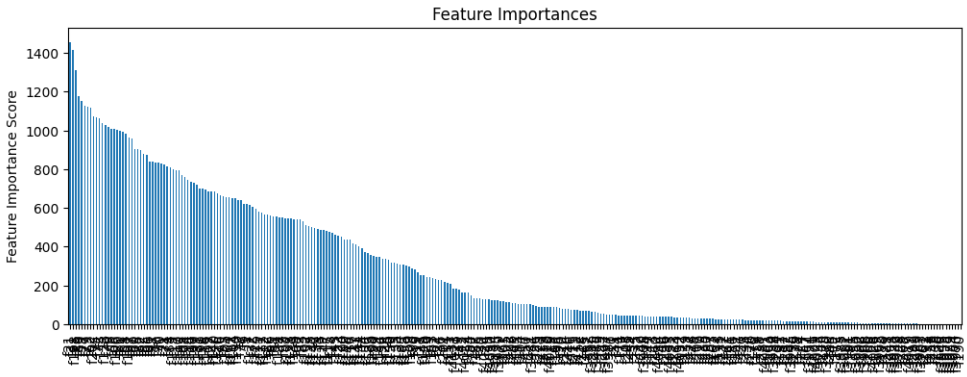

图 1: 特征重要性

```
test_subset2 = test_data[top_n_features]
if not 'Idx' in train_subset2.columns:
    train_subset2.insert(0, 'Idx', taidx) # 重新插入Idx列
if not 'Idx' in test_subset2.columns:
    test_subset2.insert(0, 'Idx', teidx) # 重新插入Idx列
train_data = train_subset2
test_data = test_subset2
```

最终将 4040 维特征降维带 286 维，大大降低了之后训练模型时的时间开销，同时减少了影响较小的 feature 的噪声影响。

4.5 类别不平衡处理

训练集中两类 target 的正负样本比例接近 13:1，与金融欺诈中比例 10000:1 相比较小，但也出现了不平衡的问题，考虑将选择训练集使得比值小于 10。

我首先考虑了 imblearn 库中几种方法 [王改改 ([n. d.])]。我尝试了欠采样中的随机法，Prototype selection，NearMiss 以及过采样中的 Prototype generation，随机复制。通过提交到网站的得分来看，效果并不好，出现了严重的过拟合。

另外，我还尝试了 xgboost 和 lighgbm 中通过设置 scale_pos_weight 和 is_unbalance 参数来自动平衡正负样本权重，一般设置在正负样本比例在 1:6-1:10，有一定提升。

5 建模与优化

5.1 实验环境

python version：3.10.13
pycharm version：PyCharm 2023.2.4 (Professional Edition)
pytorch version：2.1.0
CUDA version：12.2

5.2 Logistic Regression

Logistic Regression 虽然被称为回归，但其实际上是分类模型，并常用于二分类，因其简单、可并行化、可解释强深受工业界喜爱。Logistic Regression 的本质是：假设数据服从这个分布，然后使用极大似然估计做参数的估计 [Pearl and Reed(1920)]。

然而，因为 Logistic Regression 的决策面试线性的，Logistic 回归无法有效解决非线性问题，也很难处理数据不平衡的问题，另外，对于高维数据性能不好，它作为一个线性的分类器，也处理不好特征之间相关的情况 [阿泽 ([n. d.])]。

在本次实验中，使用 Logistic Regression 效果最不佳，auc 分数仅有 68.72。

```
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression

lr = LogisticRegression(tol=1e-6)
parameters = {'penalty': ('l1', 'l2'), 'C': [0.01, 0.1, 1, 10, 100]}
clf_lr = GridSearchCV(lr, parameters, cv=3)
print('开始训练')
clf_lr.fit(train_data.values[:, 1:], y_train)
print('模型训练结束')
print(clf_lr)

clf_lr_accuracy = clf_lr.score(train_data.values[:, 1:], y_train)
```

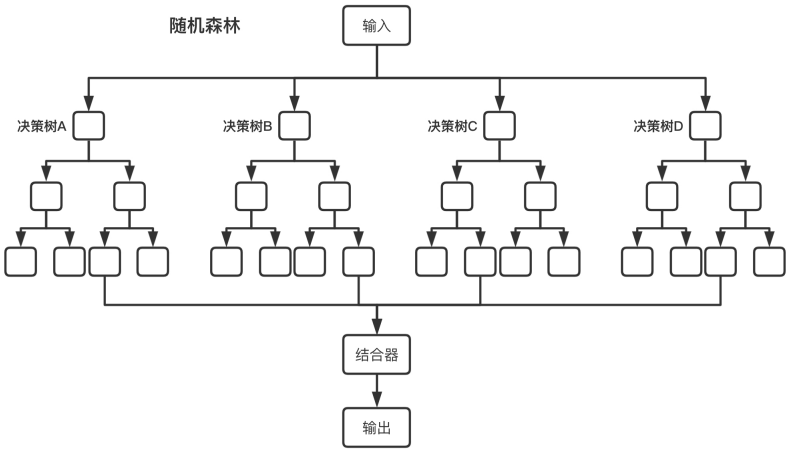

图 2: Random Forest

5.3 Random Forest

随机森林是一种 bagging 算法，利用多棵决策树对样本进行训练并预测的一种分类器，每棵树都是独立训练的。随机森林通过对每棵树的预测结果进行平均或投票来进行预测，从而提高了模型的鲁棒性和准确性 [于晓虹 and 楼文高 (2016)]。其模型图如下。

在本次实验中，由于数据量较大维度较高，采用默认的 100 棵决策树，需要耗费较长的训练时间，auc 分数仅有 64.98，出现了过拟合问题。

```
from sklearn.ensemble import RandomForestClassifier
from sklearn import metrics

rf = RandomForestClassifier(oob_score=True , random_state =42)
rf.fit(train_data.values[:,1:] ,y_train)
print(rf0.oob_score_)
y_predprob = rf.predict_proba(train_data.values[:,1:])[:,1]
print('AUC Score(Train): %f'%metrics.roc_auc_score(y_train ,
    y_predprob))
rf_predictions_df = pd.DataFrame({'ID': train_data['Idx'],
    'Prediction': y_predprob})
rf_predictions_df.to_csv('rf_prediction.csv', index=False)
```

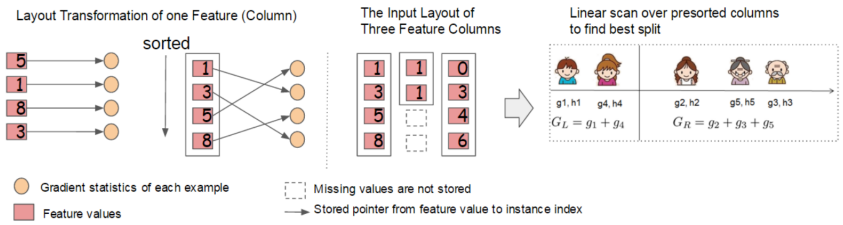

图 3: Block structure for parallel learning of XGBoost

5.4 XGBoost

XGBoost（eXtreme Gradient Boosting）极致梯度提升，是一种基于 GBDT 的算法或者说工程实现。XGBoost 具有高效、灵活和轻便的特点，在数据挖掘、推荐系统等领域得到广泛的应用。它同样使用多棵决策树来进行预测，其并行学习块结构如图。每棵树都是在前一棵树的残差基础上构建的，Boosting 思想即通过来不断迭代，增加弱分类器并根据一定的学习率进行学习，提高模型精度。 [Chen and Guestrin(2016)] 另外，由于 XGBoost 使用了正则化项来控制模型的复杂度，并采用了特殊的梯度下降方法进行训练，使得其在处理高维稀疏数据时，具有较好的效果。

本次实验中，除了利用 XGBoost 进行特征选择，也用它进行测试集的预测，模型参数通过 GridSearchCV 进行调整，事实上，模型参数对最终模型效果有很大影响。最终提交平台 auc 得分为 75.06。

```
def modeltest(alg, dtrain, y_train, dtest, useTrainCV=True, cv_folds=5,
    early_stopping_rounds=50):
    if useTrainCV:
        xgb_param = alg.get_xgb_params()
        xgtrain = xgb.DMatrix(dtrain.values[:, 1:], label=y_train)
        cvresult = xgb.cv(xgb_param, xgtrain,
            num_boost_round=alg.get_params()['n_estimators'],
            nfold=cv_folds,
                        early_stopping_rounds=early_stopping_rounds)
        alg.set_params(n_estimators=cvresult.shape[0])

    #建模
    alg.fit(dtrain.values[:, 1:], y_train, eval_metric='auc')

```

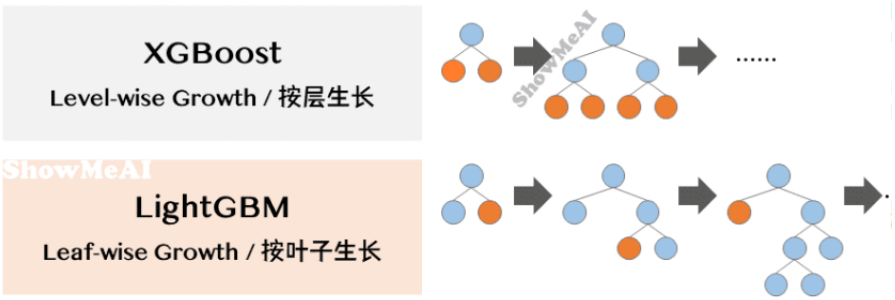

图 4: Comparison between XGBoost and LightGBM

```
xgb1 = XGBClassifier(
    learning_rate=0.05, # 学习率，一般取值范围在0.01到0.2之间
    n_estimators=200, # 弱分类器的数量，可以根据交叉验证调整
    max_depth=7, # 每棵树的最大深度，控制树的复杂度
    min_child_weight=1, # 每个子节点中所需的最小样本权重总和
    gamma=0.15, # 控制叶子节点分裂的程度，一般取值范围在0到1之间
    subsample=0.8, # 训练数据的子采样比例，一般取值范围在0.5到1之间
    colsample_bytree=0.8, #
        每棵树的特征子采样比例，一般取值范围在0.5到1之间
    objective='binary:logistic', # 学习任务的目标，二元分类问题使用
        'binary:logistic'
    nthread=4, # 并行线程数，通常设置为CPU的核心数
    scale_pos_weight=3, # 正负类别权重平衡，处理不平衡的数据集
    seed=42 # 随机种子，用于生成可重复的随机性
)

modeltest(xgb1, train_data, y_train, test_data)
```

5.5 LightGBM

LightGBM 是对 xgboost 的优化，两者对比如图，它们均为 GBDT 梯度提升树算法的框架，优化之处在于，lightgbm 选择了基于 histogram 的决策树算法，每次选取最大增益的节点进行分裂，具有更快的训练速度、更低的内存消耗、更好的准确率等优点。 [Ke et al.(2017)]

论文中提出 Exclusive Feature Bundle，高维数据通常是稀疏的，特征

的稀疏性给我们设计一种近乎无损的降低特征维数的方法提供了可能性，在稀疏特征空间中，许多特征是互斥的，即它们从不同时取非零值，我们把把互斥的特征捆绑成一个特征。文中还以 one-hot 编码举例，降低了时间复杂度，同时提高了模型的预测能力。因此，本次实验十分适合采用 LightGBM 算法。

本次实验中，利用 LightGBM 进行测试集的预测，模型参数通过 GridSearchCV 进行调整，最终提交平台 auc 得分为 75.19。

```
from lightgbm import LGBMClassifier
from sklearn import metrics

lgb_model = LGBMClassifier(
    boosting_type='gbdt',
    num_leaves=42,
    max_depth=15,
    learning_rate=0.01,
    n_estimators=1000,
    objective='binary',
    min_split_gain=0.01,
    min_child_weight=0.001,
    min_child_samples=20,
    subsample=1.0,
    subsample_freq=0,
    colsample_bytree=1.0,
    reg_alpha=0.4,
    reg_lambda=0.4,
    random_state=25,
    cat_smooth=1,
    max_bin=25
)

lgb_model.fit(train_data.values[:, 1:], y_train, eval_metric='auc')
```

5.6 CatBoost

在上次课同学分享之后，我了解了这个算法。CatBoost 的原理与 XGBoost 和 LightGBM 类似，都是通过将多个弱学习器组合成一个强学习器。

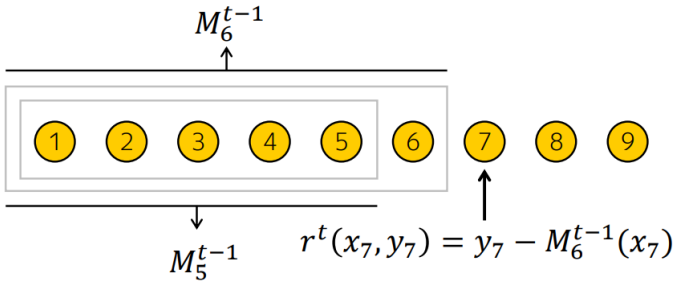

图 5: Ordered Boosting principle of CatBoost , examples are ordered according to σ.

论文中指出，CatBoost 最主要的创新点在于：Ordered Boosting 实现了有序提升，排列驱动以代替经典算法，如图所示，为论文中排序提升举的例子，为了得到无偏梯度估计，CatBoost 对每一个样本 x_i 都会训练一个单独的模型 M_i，模型 M_i 由使用不包含样本 x_i 的训练集训练得到。我们使用 M_i 来得到关于样本的梯度估计，并使用该梯度来训练基学习器并得到最终的模型。[Prokhorenkova et al.(2018)]。另外，CatBoost 基于对称决策树（oblivious trees）为基学习器实现的参数较少、支持类别型变量和高准确性的 GBDT 框架，主要解决的痛点是高效合理地处理类别型特征。

本次实验中，利用 CatBoost 进行测试集的预测，效果并不很令人惊喜，最终提交平台 auc 得分为 75.03。

```
from catboost import CatBoostClassifier, Pool, metrics, cv
from sklearn.model_selection import train_test_split


categorical_features_indices = np.where(train_data.dtypes !=
    np.float)[0]

model = CatBoostClassifier(
    custom_loss=[metrics.Accuracy()], #
        该指标可以计算logloss，并且在该规模的数据集上更加光滑
    random_seed=42,
    logging_level='Silent'
)

```

```
model.fit(
    X_train, y_train,
    cat_features=categorical_features_indices,
    eval_set=(X_val, y_val),
    plot=True
)

predictions = model.predict(test_data)
predictions_probs = model.predict_proba(test_data)
```

5.7 模型融合

暂时有模型融合的思路。

- Voting
  对于本次实验这个二分类问题，已有 3 个效果较好的基础模型（XGBoost, LightGBM, CatBoost），那么直接采取投票制的方法，投票多者确定为最终的分类。

- Averaging
  其实就是对 Voting 的每一个结果进行加权，给效果较好的 XGBoost 和 LightGBM 较大的权重 0.4，而 CatBoost 较小的权重 0.2，以加权平均数得到的概率确定最终的分类。

- Bagging
  对测试集的样本采用有放回的方式进行抽样，用抽样的样本建立子模型，对子模型进行训练，这个过程重复多次，最后对全部的子模型进行 Voting 或 Averaging 融合。伪代码如 Algorithm 1 所示。

- Stacking
  训练模型来学习使用底层学习器的预测结果，基模型在所有数据集上生成预测结果，次学习器会基于模型的预测结果进行再训练，并且用到了 K 折交叉进行验证。

Algorithm 1: The algorithm of Model Ensemble by Bagging.

Input : test set x(30000×286)

Parameters: Base learning algorithm ζ(XGBoost, LightGBM, CatBoost), epoch

Output : classification label(0,1), probability[0-1]

Initialize max_epoch as n;

for $epoch = 1$ *to* max_epoch **do**

  $h_t = \zeta(x)$;

end

Output $H(x) = \arg\max_{y \in y} \sum_{epoch=1}^{max\_epoch} E(h_t(x) = y)$

6 结论和心得体会

本次实验，在平台封榜前，我训练的模型已经达到满分（>75），在之前的实验过程中，我也遇到了很多问题，折磨我最久的也是令我印象最深刻问题可能很不起眼，它是训练集和测试集应该先融合再进行数据处理与特征工程，而不能分别处理，最后对 ont-hot 编码取交集，那样会导致相关性强的 feature 丢失。但是，直到现在我认为我的工作仍有不足之处，我的模型没有能达到很高的预测效果，我分析主要原因可能有二，其一是数据预处理和特征工程没有完全挖掘出各个标签与 target 之前的联系，其二是模型有待完善。

通过这次实验，我系统性的学习了集成学习，尤其是 Boosting 方法：XGBoost 在传统机器学习任务中表现卓越，尤其在结构化数据的分类、回归和排序任务上，它也具有强大的正则化功能，防止过拟合。LightGBM 在大规模数据集和高维度数据上表现更佳。它使用基于直方图的学习方法，减少了内存的使用。适用于处理文本分类、图像分类、推荐系统等领域的数据。CatBoost 在处理类别特征和缺失值方面表现出色，能够自动处理类别型特征的转换，鲁棒性强，对超参数不敏感，适用于含有类别特征和缺失值的任务，如电商推荐、医疗预测、金融风控。希望在今后的学习中能够继续深入研究各种机器学习方法，提升自己的科研能力。

参考文献

[Chen and Guestrin(2016)] Tianqi Chen and Carlos Guestrin. 2016. XGBoost: A Scalable Tree Boosting System. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (San Francisco, California, USA) *(KDD '16)*. Association for Computing Machinery, New York, NY, USA, 785–794. https://doi.org/10.1145/2939672.2939785

[Ke et al.(2017)] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. 2017. LightGBM: A Highly Efficient Gradient Boosting Decision Tree. In *Proceedings of the 31st International Conference on Neural Information Processing Systems* (Long Beach, California, USA) *(NIPS'17)*. Curran Associates Inc., Red Hook, NY, USA, 3149–3157.

[Pearl and Reed(1920)] Raymond Pearl and Lowell J. Reed. 1920. On the Rate of Growth of the Population of the United States since 1790 and Its Mathematical Representation. *Proceedings of the National Academy of Sciences of the United States of America* 6 6 (1920), 275–88. https://api.semanticscholar.org/CorpusID:39589064

[Prokhorenkova et al.(2018)] Liudmila Prokhorenkova, Gleb Gusev, Aleksandr Vorobev, Anna Veronika Dorogush, and Andrey Gulin. 2018. CatBoost: Unbiased Boosting with Categorical Features. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems* (Montréal, Canada) *(NIPS'18)*. Curran Associates Inc., Red Hook, NY, USA, 6639–6649.

[于晓虹 and 楼文高 (2016)] 于晓虹 and 楼文高. 2016. 基于随机森林的 P2P 网贷信用风险评价、预警与实证研究. 金融理论与实践 2 (2016), 6.

[王改改 ([n. d.])] 王改改. [n. d.]. 【图文结合】一文详解数据不平衡问题. https://zhuanlan.zhihu.com/p/296632599.

[阿泽 ([n. d.])] 阿泽. [n. d.]. 【机器学习】逻辑回归（非常详细）. https://zhuanlan.zhihu.com/p/74874291.