数据分析实践报告（金融风控）

小组成员

191250028 樊言鹏

191250059 蒋承霖

191850189 王涛

实践课题及方案

1.主题

赛题以金融风控中的个人信贷为背景，要求根据贷款申请人的数据信息预测其是否有违约的可能，以此判断是否通过此项贷款，这是一个典型的分类问题，竞赛采用AUC作为评价指标。

2.方案

通过对赛事相关内容的学习，我们将预测过程整体按照4个步骤进行：

1. 数据EDA
2. 特征工程
3. 建模调参
4. 模型融合

运行平台：jupyter

详细步骤

1.数据EDA

观察总体分布并处理缺失值

拿到数据集，首先参照竞赛题目了解数据字段含义、类型等，通过一些命令查看数据维度及字段数值的缺失情况，观察特征后：

- 对缺失数据较少的列按照数值基本特征选择不同的填充方法补充
- 等级、时间等数据数值化处理等

```
# 填写所有为空的值
def fill_na(data: pd.DataFrame):
    # 按照中位数填写数值类型
    numerical_feats = list(data.select_dtypes(exclude=['object']).columns)
    category_feats = list(data.select_dtypes(include=['object']).columns)
    for numerical_feat in numerical_feats:
        data[numerical_feat] = data[numerical_feat].fillna(data[numerical_feat].median())

    for category_feat in category_feats:
        data[category_feat] = data[category_feat].fillna(data[category_feat].mode()[0])  # 取出第一行的值填入即可

# 日期转换为int类型
def transform_datetime(df: pd.DataFrame):
    # 转化成时间格式
    df['issueDate'] = pd.to_datetime(df['issueDate'], format='%Y-%m-%d')
    start_date = datetime.datetime.strptime('2007-06-01', '%Y-%m-%d')
    # 构造时间特征
    df['issueDate'] = df['issueDate'].apply(lambda x: x - start_date).dt.days

# 时间为序列数据，转化为连续时间|就业年限
def transform_employment_length(data: pd.DataFrame):
    data['employmentLength'].replace(to_replace='10+ years', value='10 years', inplace=True)
    data['employmentLength'].replace('< 1 year', '0 years', inplace=True)
    data['employmentLength'] = data['employmentLength'].apply(lambda x: int(x.split()[0]))

# 时间为序列数据，转化为连续时间|开通时间
def transform_earliest_credit_line(data: pd.DataFrame):
    import calendar
    def month_year_to_month_count(s):
        month = list(calendar.month_abbr).index(s[0:3])
        year = int(s[-4:])
        return 12 * (year - 1944) + month

    data['earliesCreditLine'] = data['earliesCreditLine'].apply(lambda s: month_year_to_month_count(s))

# 等级为序列数据，转化为连续数值|等级
def transform_grade(data: pd.DataFrame):
    data['grade'] = data['grade'].map({'A': 1, 'B': 2, 'C': 3, 'D': 4, 'E': 5, 'F': 6, 'G': 7})

def bin(data: pd.DataFrame):
    data['loanAmnt'] = pd.qcut(data['loanAmnt'], 10, labels=False)
    data['interestRate'] = pd.qcut(data['interestRate'], 10, labels=False)
    data['ficoRangeLow'] = pd.qcut(data['ficoRangeLow'], 10, labels=False)
    data['ficoRangeHigh'] = pd.qcut(data['ficoRangeHigh'], 10, labels=False)
    data['totalAcc'] = pd.qcut(data['totalAcc'], 10, labels=False)
```

2.特征工程

我们借助SelectKBest函数使用相关系数法进行特征列的选择，精简掉无用的特征，以降低最终模型的复杂性。Pearson相关系数是一种最简单的，可以帮助理解特征和响应变量之间关系的方法，该方法衡量的是变量之间的线性相关性。

下面函数中的内容在之后的很多地方都会调用，进行列的筛选处理。

```
if('isDefault' in numerical_feats):
    is_train_data=True #含有标签，是训练数据
    y_train = data['isDefault']
    numerical_feats.remove('isDefault')
    x_train = data.drop(columns='isDefault')
    x_train = x_train[numerical_feats]
    select_k_best = SelectKBest(k=35).fit(x_train, y_train)
    column_indexes = select_k_best.get_support(indices=True)
    column_list = list(x_train.columns)
    chosen_columns = []
    for column_index in column_indexes:
        chosen_columns.append(column_list[column_index])
    print('chosen_columns：',len(chosen_columns),chosen_columns)
# 选中的数值型标签
category_columns = list(data.select_dtypes(include=['object']).columns)
label = 'isDefault'

# 保留列
reserved_columns = chosen_numerical_columns+category_columns
if(is_train_data):
    reserved_columns.append(label)
print('保留列：',len(reserved_columns),reserved_columns)

# 取反操作，去掉不在保留列的所有值。
to_drop_columns = []
column_list = list(data.columns)
for column in column_list:
    if column not in reserved_columns:
        to_drop_columns.append(column)

return data.drop(columns=to_drop_columns)
```

由于我们最终采用了xgboost和catboost来建立模型，在这里对特征提取后的数据进行了one-hot编码、非序列化特征提取，得到两份可用于训练的数据

- ```
  def feature_xgb(data: pd.DataFrame):

      is_train_data = ('isDefault' in list(data.columns))
      data=feature_selection(data)
      # 对非序列化特征进行onehot编码
      category_features = list(data.select_dtypes(include=['object']).columns)
      return pd.get_dummies(data, columns=category_features, drop_first=True)
  ```

```
def save_train_feature_xgb():
    train = pd.read_csv('train.csv')
    train_featured = feature_xgb(train)
    train_featured.to_csv('middle-product/train_featured_xgb.csv',index=False)
    print('xgb训练集特征已存储')
```

得到

- catboost特征提取与存储：

```
def feature_catb(data: pd.DataFrame):
    is_train_data = ('isDefault' in list(data.columns))
    data = feature_selection(data)

    #对于待预测数据，需要将特征字符串化
    if not is_train_data:
        with open('middle-product/feature_param_catb.json', 'r') as file:
            dic = json.load(file)
            cat_features = dic['cat_features']
            file.close()
            print('cat_features加载完成！')
        for i in data.columns:
            if i in cat_features:
                data[i] = data[i].astype('str')

    return data


def save_train_feature_xgb():
    train = pd.read_csv('train.csv')
    train_featured = feature_xgb(train)
    train_featured.to_csv('middle-product/train_featured_xgb.csv',index=False)
    print('xgb训练集特征已存储')
```

3.建模调参

获得较优的参数：
```

使用贝叶斯优化(Bayesian Optimization)来获得xgboost限定条件下的优化参数，将赛题给定的训练集按照

```
def get_max():
    # 读取已经特征处理的数据
    train = pd.read_csv('middle-product/train_featured_xgb.csv')
    print('数据加载完成！')
    print(f'训练数据基本信息：{train.info()}')
    x_train = train.drop(columns='isDefault')
    y_train = train['isDefault']
    def xgb_cv(min_child_weight,
               max_depth, gamma,
               subsample,
               colsample_bytree,
               reg_alpha,
               reg_lambda,
               learning_rate,
               scale_pos_weight):
        # 建立模型
        model = xgb.XGBClassifier(
            objective='binary:logistic',  # 目标函数
            scale_pos_weight=scale_pos_weight,  # 解决样本个数不平衡的问题
            random_state=2023,
            n_estimators=100,
            min_child_weight=min_child_weight,
            max_depth=int(max_depth),
            gamma=gamma,
            subsample=subsample,
            colsample_bytree=colsample_bytree,
            reg_alpha=reg_alpha,
            reg_lambda=reg_lambda,
            learning_rate=learning_rate
        )

        val = cross_val_score(model, x_train, y_train, cv=5, scoring='roc_auc').mean()
        return val

    """定义优化参数"""
    bayes = BayesianOptimization(
        xgb_cv,
        {
            'min_child_weight': (0.001, 1),
            'max_depth': (5, 10),
            'gamma': (0.1, 1.0),
            'subsample': (0.6, 0.95),
            'colsample_bytree': (0.6, 0.95),
            'reg_alpha': (1e-2, 100),
            'reg_lambda': (0, 1000),
            'learning_rate': (0.01, 0.15),
            'scale_pos_weight':(0.5,2)
        }
    )

    """开始优化"""
    bayes.maximize(init_points=5,  # init_points：初始点，int，可选（默认值=5），探索开始探索之前的迭代次数
                   n_iter=25)  # 最大迭代次数，int，可选（默认值=25），方法试图找到最大值的迭代次数
    print("最大参数情况：",bayes.max)
```

为了获得更好的效果，我们利用上述函数得到的优化参数去获取较好的迭依次增大模型迭代次数，直至auc评估的准确性无明显提升

```
# 获得迭代次数
def get_iters(max_params: dict):
    # 读取已经特征处理的数据
    train = pd.read_csv('middle-product/train_featured_xgb.csv')
    print('数据加载完成！')
    print(f'训练数据基本信息：{train.info()}')
    x_train = train.drop(columns='isDefault')
    y_train = train['isDefault']

    # 数据集分割
    # x_train: 训练集的特征
    # x_test:  测试集的特征
    # y_train: 训练集的标签
    # t_test:  测试集的标签,
    x_train, x_test, y_train, y_test = train_test_split(x_train, y_train, test_size=0.1, random_state=2023)
    ### fit model for train data
    model = xgb.XGBClassifier(
        learning_rate=max_params['learning_rate'],  # 学习率
        n_estimators=8000,  # 树的个数--1000棵树建立xgboost，也就是迭代次数
        max_depth=int(max_params['max_depth']),  # 树的深度
        min_child_weight=max_params['min_child_weight'],  # 叶子节点最小权重
        gamma=max_params['gamma'],  # 惩罚项中叶子结点个数前的参数
        subsample=max_params['subsample'],  # 随机选择80%样本建立决策树
        colsample_bytree=max_params['colsample_bytree'],  # 随机选择80%特征建立决策树
        reg_alpha=max_params['reg_alpha'],
        reg_lambda=max_params['reg_lambda'],
        objective='binary:logistic',  # 目标函数
        scale_pos_weight=max_params['scale_pos_weight'],  # 0:640390 1:159610 解决样本个数不平衡的问题 sum(negative instances) / sum(posi
        random_state=2023,  # 随机数
    )

    model.fit(x_train,
              y_train,
              eval_set=[(x_test, y_test)],
              eval_metric="auc",
              verbose=20,
              early_stopping_rounds=200)
    print('训练完成')
```

catboost使用相同的方法来获得优化参数。

训练模型：

使用特征处理后的数据以及优化参数建模并持久化

```
def final_train_xgb():
    # 读取已经特征处理的数据
    # 读取已经特征处理的数据
    train = pd.read_csv('middle-product/train_featured_xgb.csv')
    print('数据加载完成！')

    print('*' * 10)
    x_train = train.drop(columns='isDefault')
    y_train = train['isDefault']

    print('xgb x_train:')
    x_train.info()

    ### fit model for train data
    model = xgb.XGBClassifier(
        learning_rate=0.12,  # 学习率
        n_estimators=800,  # 树的个数--1000棵树建立xgboost，也就是迭代次数
        max_depth=9,  # 树的深度
        min_child_weight=0.5,  # 叶子节点最小权重
        gamma=0.99,  # 惩罚项中叶子结点个数前的参数
        subsample=0.8,  # 随机选择80%样本建立决策树
        colsample_bytree=0.9,  # 随机选择80%特征建立决策树
        reg_alpha=12.0,
        reg_lambda=100,
        objective='binary:logistic',  # 目标函数
        scale_pos_weight=1.2,  # 0:640390 1:159610 解决样本个数不平衡的问题 sum(negative instances) / sum(positive instances)
        random_state=2023,  # 随机数
    )

    model.fit(x_train,
              y_train)
    print('训练完成')
    model.save_model('middle-product/model_xgb.json')
    print("最终训练")
```

4.模型融合与预测

为了结果更加准确，在VotingModel方法中采用加权的方法融合xgboost和catboost的结果进行预测

```
class VotingModel:
    #总权重
    _weight_sum=0
    #模型列表
    _model_list=[]

    def _load_model(self,model_type, model_file,model_weight):
        print('开始加载',model_file)
        if(model_type == 'xgb'):
            model = xgb.XGBClassifier()
        if(model_type == 'catb'):
            model = CatBoostClassifier()

        model.load_model(model_file)
        model_item = {}
        model_item['model_type']=model_type
        model_item['model'] = model
        model_item['model_weight'] = model_weight
        self._model_list.append(model_item)
        print('加载完成', model_file)

    def __init__(self,model_config):
        for item in model_config:
            model_weight = item['model_weight']
            model_type = item['model_type']
            model_file = item['model_file']
            self._weight_sum += model_weight
            self._load_model(model_type,model_file,model_weight)

        print('模型加载完成，权重和为：',self._weight_sum)
        print('模型配置为：',self._model_list)

    def _get_predict(self,model_type,test:pd.DataFrame,model1):
        test_predict = None
        if (model_type == 'xgb'):
            featured_test =  feature_xgb(test)
            print("xgb预测数据的特征:")
            featured_test.info()
            test_predict = model.predict_proba(featured_test)[:, -1]
        if(model_type == 'catb'):
            featured_test =  feature_catb(test)
            print("catb预测数据的特征:")
            featured_test.info()
            test_predict = model.predict(featured_test, prediction_type='Probability')[:, -1]

        return test_predict
```

将测试集testA.csv输入进行预测

```
def predict(xgb_weight=1,catb_weight=2):

    test = pd.read_csv('testA.csv')
    # print(type(test[['id']]))#<class 'pandas.core.frame.DataFrame'>
    # print(type(test['id']))#<class 'pandas.core.series.Series'>

    model_config = [
        {
            'model_type':'xgb',
            'model_file':'middle-product/model_xgb.json',
            'model_weight': xgb_weight
        },
        {
            'model_type': 'catb',
            'model_file': 'middle-product/model_catb.json',
            'model_weight': catb_weight
        },
    ]
    voting_model = VotingModel(model_config)
    out = test[['id']].copy()
    out['isDefault'] = voting_model.fit(test)

    print('预测输出')
    out.info()
    file_name = 'predict/predict' + '___xgb_weight-' +str(xgb_weight)+'___cat_weight-'+str(catb_weight) +'___'+ str(time.time())+'.csv'
    out.to_csv(file_name,index=False)
    print("已经完成预测")
predict(1,1)
```

5.实验结果

将最终结果并提交天池平台，auc评估0.7346分