队伍编号：17

第二十届数据科学与统计建模竞赛

复赛论文

西南财经大学　　新网银行

2023 年 11 月

基于xgboost和集成机器学习的uplift建模

队　　长：李岚琦

队　　员1：朱子寒

队　　员2：李永康

队　　员3：

队长电话：18539212207

第二十届数据科学与统计建模竞赛

论文原创性及知识产权声明

本人郑重声明：所呈交的竞赛论文，是本团队（本人）在指导教师的指导下进行研究工作所取得的成果。除文中已经注明引用的内容外，本论文不含任何其他个人或集体已经发表或撰写过的作品成果。对本文的研究做出重要贡献的个人和集体，均已在文中以明确方式标明，因本竞赛论文引起的法律结果完全由本人承担。

本人同意竞赛期间撰写论文的知识产权属竞赛组委会及本人共有。本人完全了解竞赛组委会有关保留、使用竞赛论文的规定，竞赛组委会有权保留并向国家有关部门或机构送交论文的复印件和电子版，允许论文被查阅和借阅。本人授权竞赛组委会可以采用影印、缩印、数字化或其他复制手段保存和汇编本竞赛论文。

本竞赛论文属于

1、☐保密，在___年解密后适用本授权书。

2、☑不保密

特此声明。

团队成员签名：李岚琦，朱子寒，李永康

2023 年 11 月 22 日

基于 xgboost 和集成机器学习的 uplift 建模

摘要：

本论文依次通过数据查看、数据重要性查看、特征工程、机器学习建模、预测结果可视化，共五个步骤进行问题的求解。其中数据重要性查看步骤使用多种重要性排序方式的结合；特征工程步骤使用了 **featuretool** 特征生成器；建模步骤使用了 **causalml** 的 **R-Learner** 作为基础框架，使用了经过 **voting** 和 **stacking** 进行集成的 **xgboost** 作为具体回归器；在预测结果可视化步骤绘制 **auuc** 曲线。最终验证集结果可达 1. 17 左右。同时本次建模创新性地使用了负值填充、多维度特征提取、双钳位特征选择、因果集成模型等新方法。

Abstract:

This paper tackles the problem through five sequential steps: data exploration, importance analysis of data, feature engineering, machine learning modeling, and visualization of predictive results. The importance analysis of data employs a combination of various importance ranking methods. Feature engineering utilizes the feature generator from Featuretools. The modeling step utilizes the R-Learner from causalml as the foundational framework, incorporating XGBoost as a specific regressor through voting and stacking. In the visualization of predictive results, AUUC curves are plotted. The final validation set results achieve an accuracy of approximately 1.73.

目录

1. 问题描述与背景知识

1.1 因果推断与干预效果

1.1.1 因果推断

因果推断（Causal Inference）是一种研究因果关系的方法，旨在理解某个事件或变量对另一个事件或变量的影响。在统计学和机器学习领域，因果推断试图回答“什么导致了什么”这样的问题，而不仅仅是相关性的问题。

因果推断的目标是从观察到的数据中推断出因果关系，而不是仅仅描述变量之间的相关性。

1.1.2 干预

干预（Intervention）是指在研究中有目的地改变某个或某些变量，以观察这种改变对其他变量的影响。在因果推断中，干预是为了理解通过主动改变一个变量如何影响其他变量，并推断这种变化是否具有因果关系。通过实施干预，研究人员可以更好地理解事件之间的因果关系，而不仅仅是观察它们之间的相关性。

在本问题中，“营销活动”是对客户的一种干预，0 代表客户未被营销活动触达而 1 代表客户被营销活动触达，用 T 表示。通过对第 i 个样本计算：

$$P(Y_i = 1|X_i, T_i = 1) - P(Y_i = 1|X_i, T_i = 0)$$

来得到干预的提升值 lift。

1.2 机器学习

1.2.1 xgboost 模型

XGBoost（eXtreme Gradient Boosting）是一种强大而高效的机器学习算法，主要用于解决回归和分类问题。作为一种梯度提升框架，XGBoost 通过集成多个弱学习器（通常是决策树）来构建一个强大的模型。其核心特点包括梯度提升的迭代学习方式，正则化项以控制模型复杂性，能够评估特征重要性，支持处理缺失值，具备并行处理能力，以及灵活地适用于分类和回归任务。

1.2.2 voting 和 stacking 集成策略

Voting 和 Stacking 是机器学习中常用的集成学习策略，用于提高模型性能。

Voting（投票法）： 在投票法中，多个独立的模型被训练，然后它们的预测结果通过投票的方式进行集成。对于分类问题，通常有硬投票和软投票两种方式。硬投票是基于每个模型的绝对多数来决定最终的预测类别，而软投票是基于每个模型的概率估计，将它们的概率平均，然后选择概率最高的类别。对于回归问题，可以进行平均投票。Voting 可以包括不同类型的模型，例如决策树、支持向量机、逻辑回归等。通过集成多个模型，Voting 能够减小过拟合风险，提高整体性能。

Stacking（堆叠法）： 在堆叠法中，多个基础模型被训练，然后另一个模型（元模型）被训练来组合这些基础模型的预测结果。不同于简单的投票，堆叠法通过训练一个元模型，从而对基础模型的输出进行加权组合，以提高集成模型的性能。通常，训练集被划分为多个折，基础模型在其中一部分上进行训练，然后在其他部分上进行预测。这样得到的预测结果被用作元模型

的输入，从而形成最终的集成模型。Stacking 提供更高层次的模型组合，可以更灵活地捕捉数据中的复杂关系，但也需要更多的计算资源。

1.2.3 causalml 工具包

Causalml 是一个专注于因果推断的 Python 工具包，最初在 Uber 内部使用，后来在 github 上面开源，为研究人员提供了丰富的工具和算法，以解决处理因果效应估计、选择偏差、干预效果等任务。该工具包提供了多种因果效应估计方法，涵盖传统统计学方法和机器学习方法，使用户能够在观察性数据中估计干预效果。其中，重点解决选择偏差问题，通过匹配、加权等方式减轻或纠正选择偏差，提高因果推断的准确性。Causalml 支持基于倾向得分的匹配方法，有助于更好地匹配干预组和对照组。此外，工具包还提供了处理潜在卷入偏差的方法，以更好地控制因果效应的估计。强调结果的解释性，causalml 帮助用户更好地理解模型的输出和因果关系的解释。支持集成学习方法，例如通过使用 stacking 进行模型集成，提高因果推断的性能。

2. 数据查看

2.1 缺失值填充

首先查看原生的.csv 文件可以看出，问题所给的训练集和测试集有较多缺失值，同时非缺失部分的数值均是非负的。因此为了区分缺失值，将原有的缺失部分用“-1”进行填充，以期望模型能够学习到缺失位置是一个“与其他值不一样”的信息。尽管主流的缺失值填充有均值填充、中位数填充、众数填充等，但由于本次任务的数据特性，经过实验后，采用负值填充的效果明显优于其他方式。代码为：

```
import pandas as pd
import numpy as np
X = pd.read_csv('./data/train.csv')
Y = pd.read_csv('./data/B_test.csv')
X = X.replace([np.nan], -1)
Y = Y.replace([np.nan], -1)
X.to_csv('./data/-999_train.csv',index =False)
Y.to_csv('./data/-999B_test.csv',index =False)
```

2.2 倾向性得分可视化

填充完成后，打开并运行 see_data.ipynb 文件。关于 see_data.ipynb 的代码及其说明如下：

读取数据集，并且定义可视化函数 see_data:

```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
from causalml.propensity import GradientBoostedPropensityModel

# 准备数据集，将特征和标签分开
data = pd.read_csv('./data/-999_train.csv')  # 读取数据集


def see_data(data,col,width= 0.01):
    data = data.iloc[:,col]

    value_counts = {}
    for value in data:

                if value in value_counts:
                   value_counts[value] += 1
                else:
                   value_counts[value] = 1
    values = list(value_counts.keys())
    frequencies = list(value_counts.values())
    plt.bar(values, frequencies,width=width)
    plt.title(col)
    plt.xlabel("Value")
```

```
    plt.ylabel("Frequency")
    plt.show()
```

计算各个样本的倾向性得分，并且可视化，代码：

```
kwa = {
            "max_depth": 8,
            "learning_rate": 0.1,
            "n_estimators": 100,
            "objective": "binary:logistic",
            "nthread": -1,
            "colsample_bytree": 0.8,
            "random_state": 42,
        }
pro_s =  GradientBoostedPropensityModel (early_stop=False,
clip_bounds=(1e-3, 1 - 1e-3), **kwa)
data = data.iloc[:,1:]

X = data.drop(['y','treatment'],axis=1) # 特征列
y = data['y']  # 标签列
treatment = data['treatment']
pro_s.fit(X,treatment)
the_pro_score = pro_s.predict(X)
tps = pd.DataFrame(the_pro_score)
see_data(tps,0,width= 0.0001)
```

运行结果：

其中，横轴代表倾向性得分的取值，纵轴代表该取值所出现的次数。可

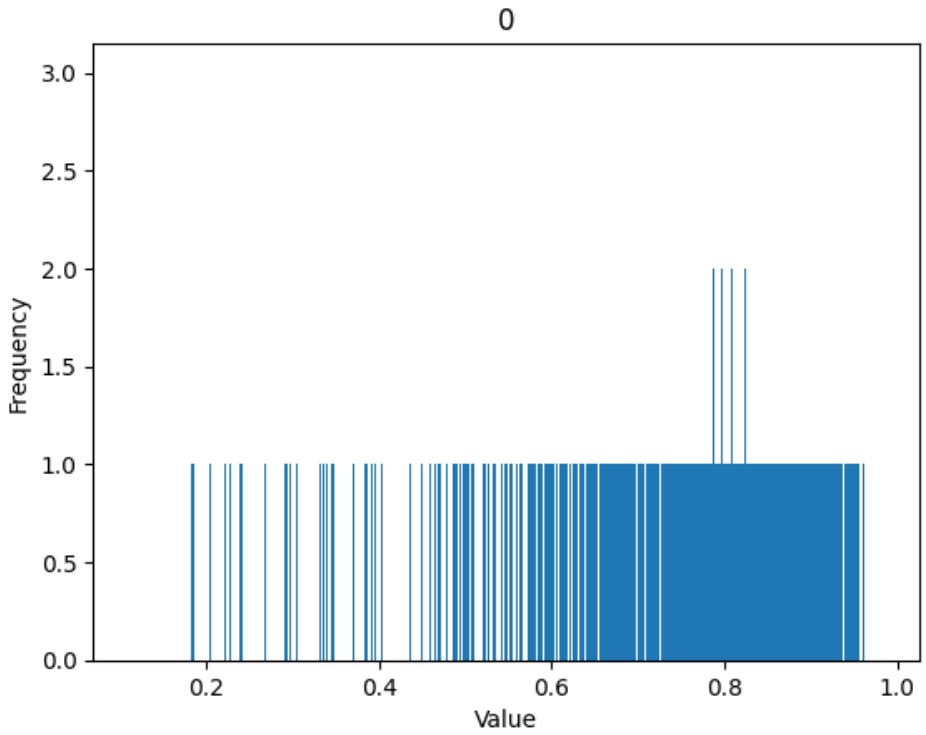


图 1：倾向性得分统计

见各个样本的倾向性得分主要集中在 0.6 到 0.9 之间。

2.3 特征可视化

对各个特征进行可视化，代码：

```
for i in range(0,11):
 see_data(data,i+1,0.01)
```

通过对 range 内的范围进行改变，可以查看各个特征维度的具体分布，以 x2 特征为例，其图像为：

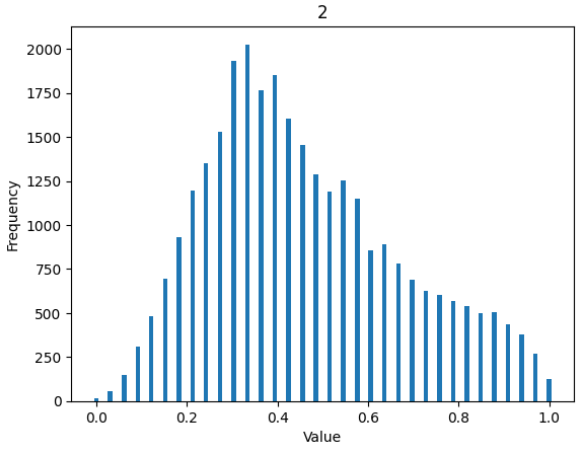


图 2：X2 特征统计

其中横轴为每个样本的该特征取值，纵轴为出现次数。可见 x2 特征主要分布在 0 到 1 之间，没有缺失值，并且 0.3 以上的值较多。

3. 数据重要性查看

在本章节中，将会在多种维度上对特征重要性进行查看，从而得到更全面的特征表达。打开并运行 feature_importance.ipynb 文件，其具体代码与解释如下：

3.1 导入必要的包并准备数据

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

```
from sklearn.ensemble import RandomForestRegressor,
GradientBoostingRegressor,RandomForestClassifier
from sklearn.tree import DecisionTreeRegressor
from xgboost import XGBRegressor
from lightgbm import LGBMRegressor

from causalml.inference.meta import BaseSRegressor, BaseTRegressor,
BaseXRegressor, BaseRRegressor
from causalml.inference.tree import UpliftTreeClassifier,
UpliftRandomForestClassifier


import shap
import matplotlib.pyplot as plt
import time
from sklearn.inspection import permutation_importance
from sklearn.model_selection import train_test_split
from sklearn.feature_selection import SelectFromModel

import os
data=pd.read_csv('./data/-999_train.csv').iloc[:,1:]
X=data.iloc[:,:161]
treatment=data.iloc[:,161]
y=data.iloc[:,162]
data= pd.read_csv('./data/-999_train.csv',header=None)
feature_names = data.iloc[0,1:162]
print(feature_names)
```

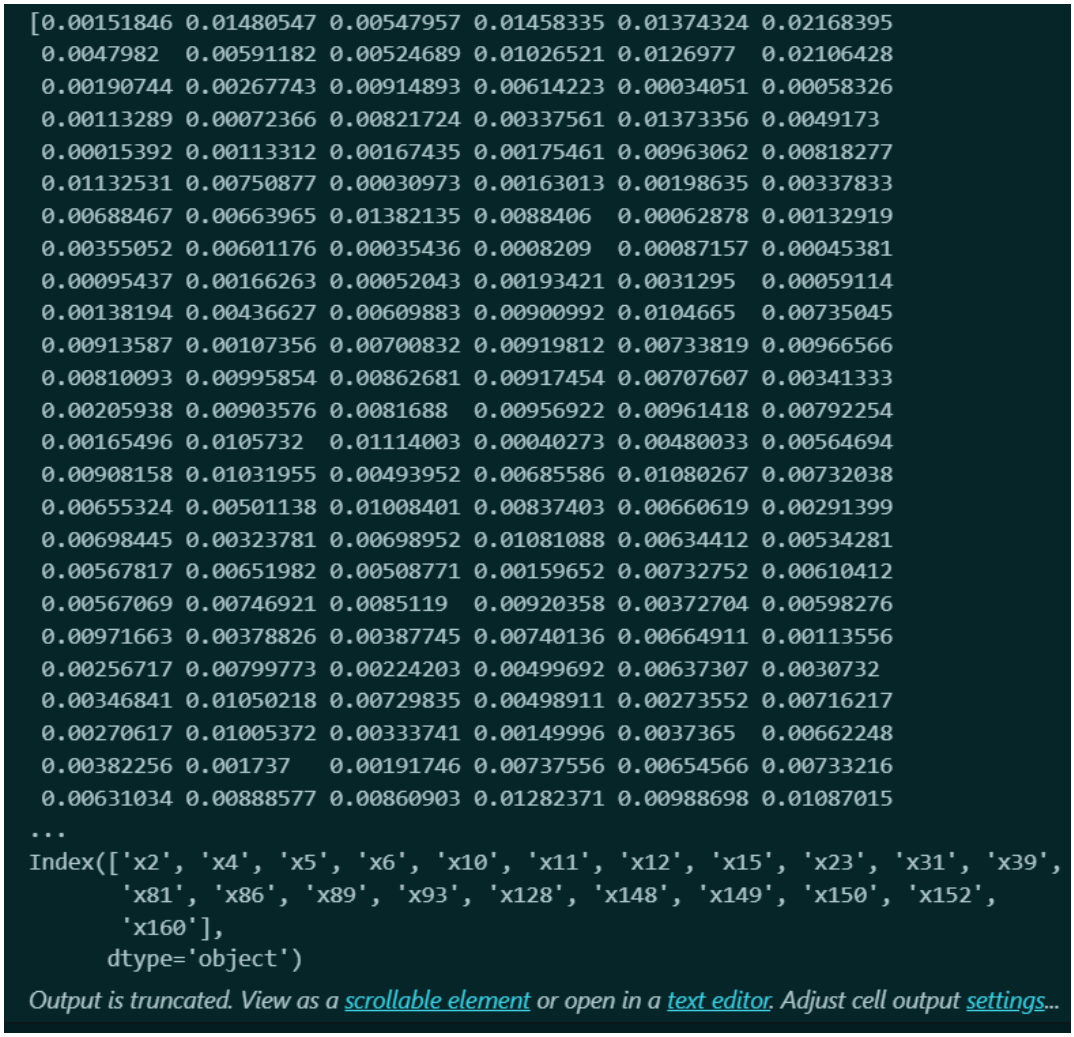
3.2 使用随机森林进行特征重要性查看

建立随机森林分类器，并且把特征重要性阈值设置为0.01，代码：

```
# 创建随机森林分类器
rf = RandomForestClassifier()
```

```
# 使用随机森林分类器拟合数据
rf.fit(X, y)

# 获取特征重要性
importance = rf.feature_importances_
print(importance)

# 创建选择器，并基于特征重要性进行特征选择
selector = SelectFromModel(rf, threshold=1e-02)  # 设置重要性阈值
selector.fit(X, y)

# 获取选择后的特征矩阵
X_selected = selector.transform(X)

# 获取选择后的特征列名
selected_features = X.columns[selector.get_support()]  # 使用columns
获取列名
print("选择后的特征矩阵:")
print(X_selected)
print("选择后的特征列名:")
print(selected_features)
```

运行结果：



可见随机森林分类器找到的前 5 个重要特征为：‘x2’，’ x4’,’ x5’,’ x6’,’ x10’。

3.3 使用 RLearner 进行特征重要性查看

使用 causalml 内置的分类器进行训练并查看特征重要性，代码：

```
##########使用RLearner 自身特征重要性
##使用集成模型
def
make_models(nums,max_depth_list,n_estimators_list,learning_rate_list,
min_child_weight,spw ):
    models=[]
    for i in range(nums):
        models.append((str(i),xgb.XGBRegressor(max_depth=max_depth_li
st,learning_rate=0.06,gamma=0, min_child_weight =
min_child_weight ,reg_alpha=0, # noqa: E501
         n_estimators=n_estimators_list ,scale_pos_weight=spw)))
    return models

model1=make_models(3,200,200,0.06,spw=0.8,min_child_weight=0.5)
model2=make_models(3,260,125,0.06,spw=0,min_child_weight=4)
stacking_model1 = StackingRegressor(estimators=model1,cv=5, n_jobs=5)
stacking_model2 = StackingRegressor(estimators=model2,cv=5, n_jobs=5)


rlearner = BaseRLearner(learner=None,
        outcome_learner=stacking_model1,
        effect_learner=stacking_model2,
        ate_alpha=3,
        n_fold=5,
        random_state=42)
rlearner.fit(X=X_train, treatment=t_train, y=y_train)
r_tau = rlearner.predict(X=X_train)
model_tau_feature = RandomForestRegressor()
# specify model for model_tau_feature
importance =  rlearner.get_importance(X=X_train, tau=r_tau,
model_tau_feature=model_tau_feature,
                      normalize=True, method='auto',random_state=42)
pd.set_option('display.max_rows', 200)
print(importance)
```

```
{1: Feature_014    0.074473
Feature_093    0.052952
Feature_005    0.043234
Feature_011    0.030651
Feature_043    0.018099
Feature_001    0.016617
Feature_030    0.016576
Feature_080    0.014074
Feature_010    0.013067
Feature_088    0.012377
Feature_038    0.012146
Feature_031    0.011946
Feature_009    0.011924
Feature_151    0.011712
Feature_022    0.011663
Feature_127    0.011283
Feature_147    0.010669
Feature_003    0.010538
Feature_039    0.009913
Feature_068    0.009823
Feature_073    0.009632
Feature_055    0.009547
Feature_028    0.009509
Feature_133    0.009473
Feature_004    0.009427
...
Feature_044    0.000233
Feature_016    0.000217
Feature_081    0.000100
```

运行结果：

可见使用 Rlearner 得到的前五个重要特征为：’x14’,’x93’,’x5’,’x11’,’x43’。

3.4 使用 RLearner 进行 Shap 特征重要性查看

代码：

```
# Using SHAP
shap_rlearner = rlearner.get_shap_values(X=X, tau=rlearner_tau)

# 使用这些列名绘制 SHAP 图
rlearner.plot_shap_values(X=X, tau=rlearner_tau,max_display=161)
```

运行结果（截取一部分）：

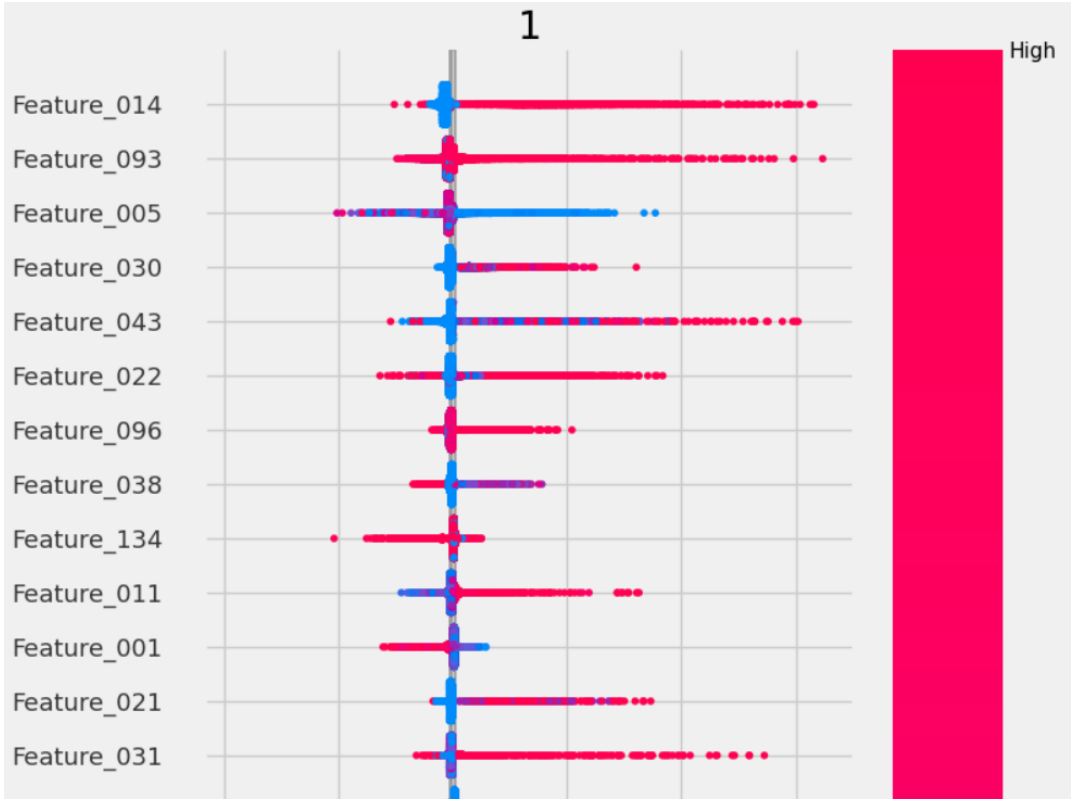


图 3：特征的 SHAP 值

其中左边是各个特征的编号，横轴代表对最终结果的影响是正还是负，而颜色从红色到蓝色代表特征值从高到低。在图中，每一个点代表一个样本。

最终得到在 Shap 层面对预测结果影响最大的五个特征为：’x14’,’x93’,’x12’,’x43’,’x31’。

此外，我们也可以挑选某个特征，来检测它是否对另一个特征有独立性。代码为：

```
# interaction_idx set to 'auto' (searches for feature with greatest
approximate interaction)
rlearner.plot_shap_dependence(treatment_group=1,
                              feature_idx=1,
                              X=X,
                              tau=rlearner_tau,
                              interaction_idx='auto')
```

其中 feature_idx 的不同取值可以查看不同维度特征与其 Shap 值的联系。以特征 x1 为例，运行结果为：

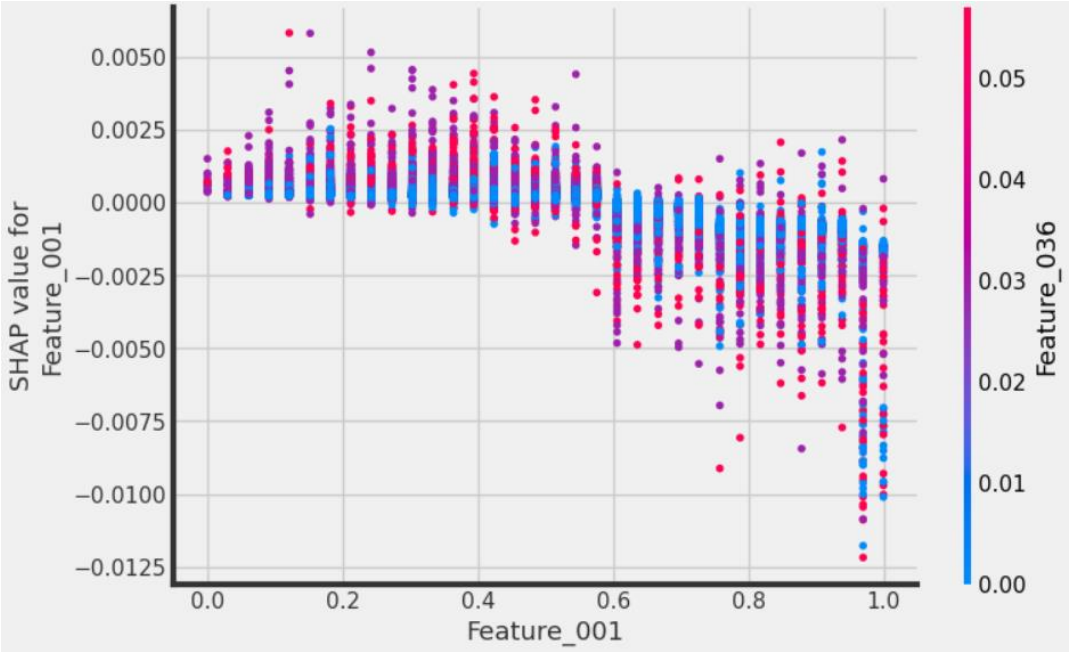


图 4：X1 特征独立性分析

其中横轴为 x1 的取值，纵轴为每个样本的标签预测值，而颜色代表特征 x30 的取值。由图中可以看出，x1 取值的变化对于 x36 取值较小的样本来说，影响不大并且几乎在 0 附近；而对于 x30 取值较大的样本来说，x1 的取值则很可能使得最终预测结果偏离 0 点并且倾向于得到负数。

3.5 使用 Filter 方法进行特征重要性查看

代码：

```
###############挑选出感兴趣的列
from causalml.metrics import *
from causalml.feature_selection.filters import FilterSelect
filter_method = FilterSelect()
data = pd.read_csv('./data/-999_train.csv')  # 读取数据集
data = data.iloc[:,1:]

X = data.drop(['y','treatment'],axis=1) # 特征列
y = data['y']  # 标签列
treatment = data['treatment']  # 处理组标识
feature_names = X.head(0)
# F Filter with order 1
method = 'ED'
f_imp = filter_method.get_importance(data, feature_names, 'y', method,
                                     experiment_group_column="treatment",c
ontrol_group=0,
                                     treatment_group=1,
                                     n_bins=5,
                                     null_impute='median'
                                     ,order=2)


print(f_imp.iloc[0:10,1])

linear_importance = f_imp.iloc[:,1]
```

其中 method =‘ED’代表衡量各个特征的欧几里得距离（Euclidean distance），order=2 表示查看各个特征与结果的一次相关性和二次相关性。

运行后得到前五个特征重要性为：’x94’,’x112’,’x39’,’x97’,’x100’。

4. 特征工程

4.1 各个方法特征重要性一览

表 1：各维度特征重要性

衡量标准	第 1 重要特征	第 2 重要特征	第 3 重要特征	第 4 重要特征	第 5 重要特征
随机森林分类器	X2	X4	X5	X6	X10
RLearner	X14	X93	X5	X11	X43

Shap 值	X14	X93	X12	X43	X31
欧式距离二次型	X94	X112	X39	X97	X100

4.2 特征生成

选取上一节中，各个维度的前三个重要特征来生成新特征，也即是：x2, x14, x94, x4, x93, x112, x5, x12, x39。打开并运行 deep_fs.ipynb，其具体代码与说明如下：

4.2.1 导入必要的包，并且选择特征序号

```
import pandas as pd
import numpy as np
train = pd.read_csv('./data/-999_train.csv')
#train = train.drop('y')

selected_cols = ['id','x2', 'x14',
'x94','x4','x93','x112','x5','x12','x39']
  # 选择感兴趣的列
```

```
# 筛选出感兴趣的列形成新的特征矩阵
train = train[selected_cols]
```

4.2.2 对训练集进行特征生成

代码：

```
import featuretools as ft
es = ft.EntitySet(id='single_dataframe')  # 用id标识实体集

es.add_dataframe(  dataframe_name = 'train_data',
            dataframe=train
            )

trans_primitives=['add_numeric',
ft.primitives.SubtractNumeric(commutative=False),'multiply_numeric']

#ft.list_primitives()

feature_matrix, feature_names = ft.dfs(entityset=es,
    target_dataframe_name = 'train_data',
     max_depth=1,
     verbose=1,
     trans_primitives=trans_primitives
)
feature_matrix.to_csv('./data/train_feature.csv',index= False)
```

运行结果：

```
c:\Users\14251\anaconda3\envs\test001\lib\site-p
  "Woodwork may not support Python 3.7 in next r
c:\Users\14251\anaconda3\envs\test001\lib\site-p
  FutureWarning,
c:\Users\14251\anaconda3\envs\test001\lib\site-p
  "Using first column as index. "
Built 153 features
Elapsed: 00:00 | Progress:   0%|
c:\Users\14251\anaconda3\envs\test001\lib\site-p
  data[k] = com.apply_if_callable(v, data)
Elapsed: 00:00 | Progress: 100%|
```

可见，构建了 153 个新特征。

4.2.3 对测试集进行特征生成

代码：

```
test = pd.read_csv('./data/-999B_test.csv')
test = test[selected_cols]
es_test = ft.EntitySet(id='test')  # 用id标识实体集
# 增加一个数据框，命名为iris
es_test.add_dataframe(  dataframe_name = 'test_data',
            dataframe=test
            )

feature_matrix_test, feature_names = ft.dfs(entityset=es_test,
    target_dataframe_name = 'test_data',
     max_depth=1,
     verbose=1,
     trans_primitives=trans_primitives
)
feature_matrix_test.to_csv('./data/Btest_feature.csv',index = False)
```

同样构建出 153 个新特征。

4.2.4 将原数据和生成的特征数据进行拼接

代码：

```
X = pd.read_csv('./data/-999_train.csv')
Y = pd.read_csv('./data/-999B_test.csv')
XF = pd.read_csv('./data/train_feature.csv')
YF = pd.read_csv('./data/Btest_feature.csv')
selected_cols = ['x2', 'x14', 'x94','x4','x93','x112','x5','x12','x39']
XF = XF.drop(selected_cols,axis=1)
YF = YF.drop(selected_cols,axis=1)
print(XF.shape,YF.shape)
final_train =  pd.concat([X,XF], axis=1)
final_test =  pd.concat([Y, YF], axis=1)
print(final_train.shape,final_test.shape)
final_train.to_csv('./data/final_train.csv',index=False)
final_test.to_csv('./data/Bfinal_test.csv',index=False)
```

此处代码把原数据和生成的特征数据进行了横向拼接，并且分别保存为：

final_train.csv：训练集　Bfinal_test.csv：测试集

至此，对数据的预处理已经完成，接下来将建立因果推断模型进行训练和预测。

5. 机器学习建模

5.1 导入必要的包

```
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
from sklearn.model_selection import train_test_split
from xgboost import XGBRegressor
from causalml.inference.meta import XGBTRegressor
from causalml.inference.meta import BaseSLearner, BaseTLearner,
BaseRLearner, BaseXLearner
from causalml.inference.tree import UpliftRandomForestClassifier
from causalml.dataset import *
from causalml.metrics import *
import sklearn
from sklearn.ensemble import
StackingRegressor,VotingRegressor,VotingClassifier
from sklearn import svm
import xgboost as xgb
import random
from sklearn.linear_model import LinearRegression, LogisticRegression
from sklearn.feature_selection import SelectFromModel
from causalml.propensity import GradientBoostedPropensityModel
from causalml.feature_selection.filters import FilterSelect
```

5.2 准备数据

```
# 准备数据集，将特征和标签分开
data = pd.read_csv('./data/final_train.csv')  # 读取数据集
```

```
data = data.iloc[:,1:]

X = data.drop(['y','treatment'],axis=1) # 特征列
y = data['y']  # 标签列
treatment = data['treatment']  # 处理组标识
feature_names = X.head(0)

print('特征、标签、干预的大小为: ',X.shape,y.shape,treatment.shape)
# 将数据集分为训练集和测试集
X_train, X_test, y_train, y_test, t_train, t_test = train_test_split(X,
y, treatment, test_size=0.3,random_state=42)
```

5.3 进行双钳位特征选取

```
###############挑选出感兴趣的列
filter_method = FilterSelect()
# F Filter with order 2
method = 'ED'
f_imp = filter_method.get_importance(data, feature_names, 'y', method,
                                     experiment_group_column="treatment",c
ontrol_group=0,
                                     treatment_group=1,
                                     n_bins=5,
                                     null_impute='median'
                                     ,order=2)


print(f_imp.iloc[0:10,1])

linear_importance = f_imp.iloc[:,1]
#########选取两个钳位对个特征进行选取
ton_n1 = 125
top_n2 = 175
X_train = X_train[linear_importance[ton_n1:top_n2+1]]
X_test = X_test[linear_importance[ton_n1:top_n2+1]]
```

注意到，这里先用欧几里得二次型对 381 个特征进行重要性排序。考虑到如果全部选取靠前的特征，则数据的二次函数性质过于明显；如果选取靠后的特征，则数据的独立性较高，难以正确学习。因此本文创新型地使用了双钳位特征选取，使用 top_n1 = 250 和 top_n2 = 290 来选取排序为 250 到 290 的 41 个特征进行训练。实测用该数据训练的效果要优于选取前 41 个特征和后 41 个特征数据。

5.4 对训练集和验证集计算倾向性得分

```
##########计算倾向性得分
kwa = {
            "max_depth": 8,
            "learning_rate": 0.1,
            "n_estimators": 100,
            "objective": "binary:logistic",
            "nthread": -1,
            "colsample_bytree": 0.8,
            "random_state": 42,
            # "scale_pos_weight":0.2
        }
pro_s =  GradientBoostedPropensityModel (early_stop=False,
clip_bounds=(1e-3, 1 - 1e-3), **kwa)

pro_score_train=pro_s.fit_predict(X_train,t_train)
pro_score_test=pro_s.fit_predict(X_test,t_test)
```

5.5 搭建模型

```
##尝试使用集成模型
def
make_models(nums,max_depth_list,n_estimators_list,learning_rate_list,
min_child_weight,spw ):
    models=[]
    for i in range(nums):
        models.append((str(i),xgb.XGBRegressor(max_depth=max_depth_li
st,learning_rate=0.06,gamma=0, min_child_weight =
min_child_weight ,reg_alpha=0, # noqa: E501
         n_estimators=n_estimators_list ,scale_pos_weight=spw)))
    return models

model1=make_models(3,200,200,0.06,spw=0.8,min_child_weight=0.5)
model2=make_models(3,260,125,0.06,spw=0,min_child_weight=4)
stacking_model1 = StackingRegressor(estimators=model1,cv=5, n_jobs=5)
stacking_model2 = StackingRegressor(estimators=model2,cv=5, n_jobs=5)
voting_model1 = VotingRegressor(estimators=model1)
voting_model2 = VotingRegressor(estimators=model2)
```

这里分别使用 voting 和 stacking 集成机器学习的方法搭建了四个模型：

voting_model1, voting_model2, stacking_model1, stacking_model2。

5.6 将搭建的集成模型导入 Rlearner 框架进行训练

```
###Try R Learner  此处需要调参


learner_r = BaseRLearner(learner=None,
        outcome_learner=stacking_model1,
        effect_learner=stacking_model2,
        ate_alpha=3,
```

```
        n_fold=5,
        random_state=None)
learner_r.fit(X = X_train,
treatment=t_train,y=y_train,p=pro_score_train)
```

6. 预测结果可视化

6.1 模型训练完成后，进行验证集的预测

```
cate_r = learner_r.predict(X=X_test,p=pro_score_test)
print(type(cate_r))
print(cate_r.shape)
```

6.2 对预测效果进行可视化

代码：

```
t_test = np.array(t_test).reshape(9000,1)
y_test = np.array(y_test).reshape(9000,1)
data_r = np.concatenate((y_test, t_test,-cate_r), axis=1)
data_r = pd.DataFrame(data_r)
data_r.columns = ['y','w','model']
# 计算前 t 个样本的累计增益
get_cumgain(data_r,normalize=False, random_seed=42)
# 可视化前 t 个样本的累计增益，即 Uplift Curve
```

```
plot_gain(data_r, normalize=True, random_seed=42, n=300, figsize=(8,
8))
auuc_score(df=data_r, normalize=True, tmle=False)

get_cumlift(data_r,random_seed=42)

plot_lift(data_r,  random_seed=42, n=100, figsize=(8, 8))
```

运行结果：

```
model     1.169091
Random    0.498263
dtype: float64
```

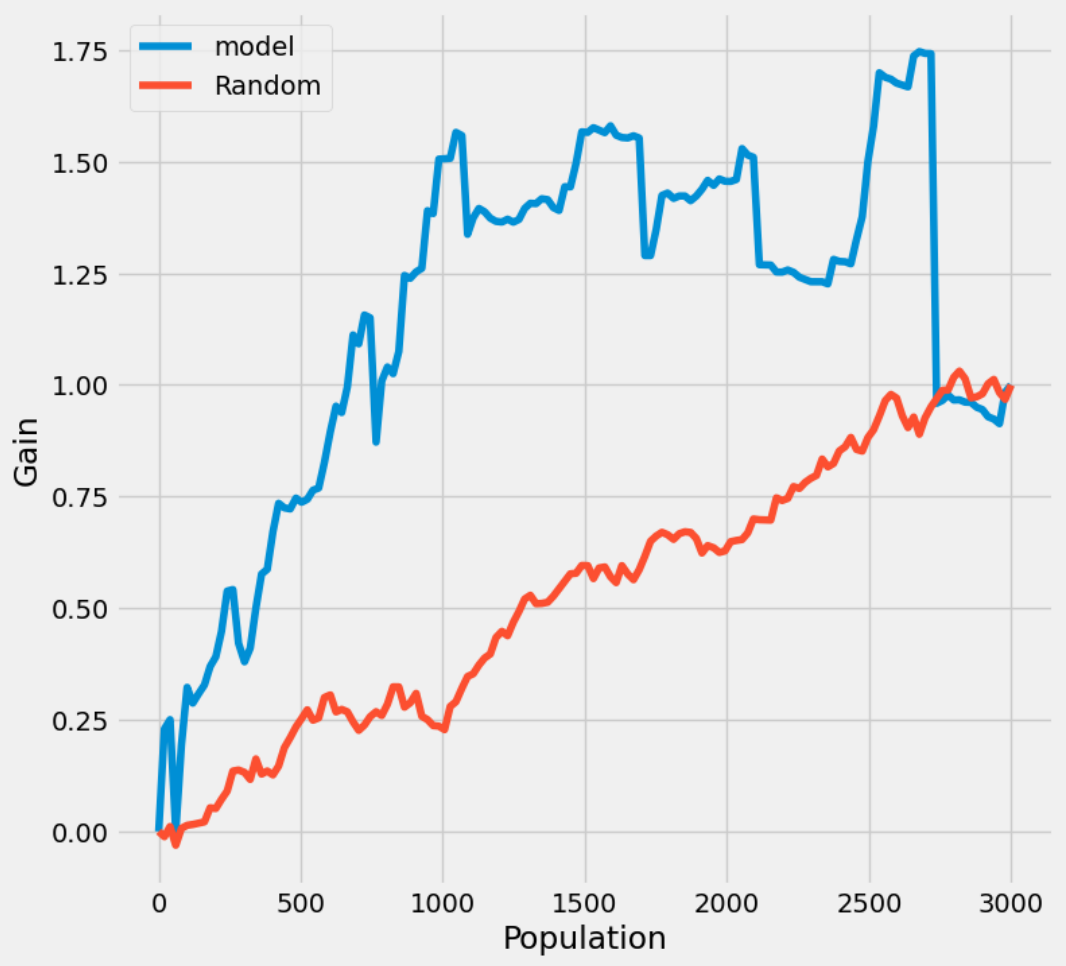


图 5：模型效果验证

如图所示，在进行随机猜测实验的分数 random 为 0.5 左右时，我们搭建的模型的分数 model 可达到 1.69 ，相较随机猜测的分数提升了 238%

6.3 对预测样本 lift 进行可视化

代码：

```
get_cumlift(data_r,random_seed=42)

plot_lift(data_r,  random_seed=42, n=100, figsize=(8, 8))
```

运行结果：

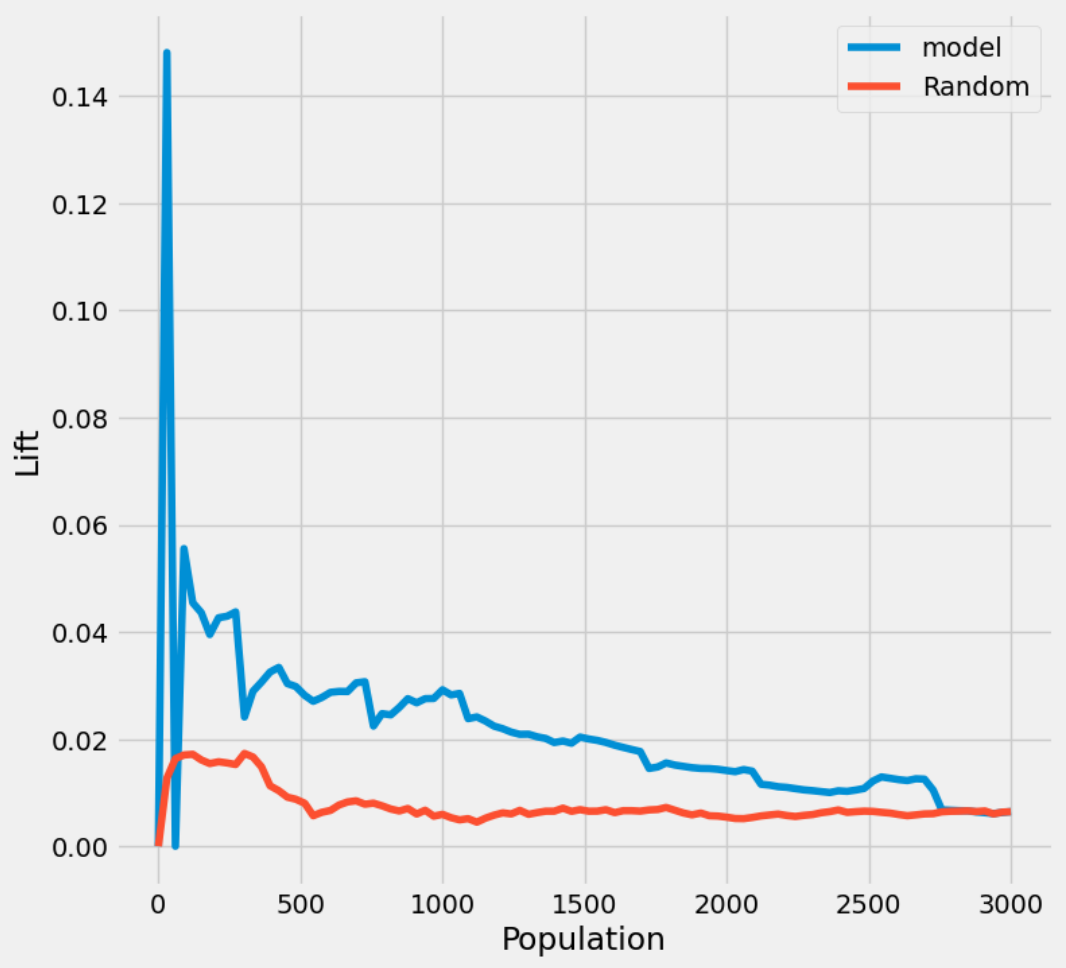


图 6：各样本 LIFT 绝对值统计

可见，在随机实验 random 的 lift 值在 0 附近的同时，我们搭建的模型效果可以有效地加大 lift 值的准确度。

6.4 对测试集 B 进行预测并保存

```
sub_data = pd.read_csv('./data/Bfinal_test.csv')  # 读取数据集
sub_feature = sub_data.iloc[:,1:].drop('treatment',axis=1)
sub_t = sub_data['treatment']



sub_feature = sub_feature[linear_importance[ton_n1:top_n2+1]]

sub_score=pro_s.fit_predict(sub_feature,sub_t)
tau_test = learner_r.predict(X=sub_feature,p=sub_score)


from sklearn.preprocessing import MinMaxScaler
for i in range(10000):
    if float(tau_test[i]) <= -1:
        print(tau_test[i])
        tau_test[i] = -1
    if float(tau_test[i]) > 1:
        print(tau_test[i])
        tau_test[i]=1

# scaler = MinMaxScaler()
# lift = scaler.fit_transform(cate_r.reshape(-1, 1))
lift = tau_test


sub = pd.DataFrame({'id':sub_data.iloc[:,0],'lift':lift.squeeze()})
sub.to_csv('sub.csv',index=False)
```

至此，我们完成了模型的搭建与训练，以及对模型效果的验证；也完成了对测试集的读取和预测，并且把预测结果保存到了’sub.csv’文件之中。

7. 附录

7.1 环境说明

本次建模使用语言是 python 3.7.1，使用的文件格式为后缀名为.ipynb 的 notebook 形式，使用的编辑器为 vscode。（也可以使用 jupyter notebook 或者 jupyterlab 或者 pycharm 专业版来打开。）

7.2 各代码文件运行说明

7.2.1 数据放置路径

在各个代码文件的同一路径新建一个’data’文件夹，并将将训练数据（名称为 train.csv）和测试数据（名称为 B_test.csv）放入其中。

7.2.2 各代码运行顺序

表 2：各代码文件运行顺序

运行顺序	代码文件名称	代码具体功能
1	1_fill_in.ipynb	填充缺失值
2	2_see_data.ipynb	查看数据分布
3	3_feature_importance.ipynb	查看特征重要性
4	4_deep_fs.ipynb	生成新特征
5	5_conbine_data.ipynb	将原数据和生成的新特征进行拼接
6	6_try_Rlearner.ipynb	搭建模型，训练并且将预测结果可视化

7.3 各代码文件源码

我们把相应代码文件放在了’code’文件夹中，建议去那里直接查看。在本报告中查看会比较长。

7.3.1 fill_in.ipynb

```
import pandas as pd
import numpy as np
X = pd.read_csv('./data/train.csv')
Y = pd.read_csv('./data/B_test.csv')
X = X.replace([np.nan], -1)
Y = Y.replace([np.nan], -1)
X.to_csv('./data/-999_train.csv',index =False)
Y.to_csv('./data/-999B_test.csv',index =False)
```

7.3.2 see_data.ipynb

```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
from causalml.propensity import GradientBoostedPropensityModel

# 准备数据集，将特征和标签分开
data = pd.read_csv('./data/-999_train.csv')  # 读取数据集


def see_data(data,col,width= 0.01):
    data = data.iloc[:,col]

    value_counts = {}
    for value in data:

                if value in value_counts:
```

```
                value_counts[value] += 1
            else:
                value_counts[value] = 1
    values = list(value_counts.keys())
    frequencies = list(value_counts.values())
    plt.bar(values, frequencies,width=width)
    plt.title(col)
    plt.xlabel("Value")
    plt.ylabel("Frequency")
    plt.show()



kwa = {
            "max_depth": 8,
            "learning_rate": 0.1,
            "n_estimators": 100,
            "objective": "binary:logistic",
            "nthread": -1,
            "colsample_bytree": 0.8,
            "random_state": 42,
        }
pro_s =  GradientBoostedPropensityModel (early_stop=False,
clip_bounds=(1e-3, 1 - 1e-3), **kwa)
data = data.iloc[:,1:]

X = data.drop(['y','treatment'],axis=1) # 特征列
y = data['y']  # 标签列
treatment = data['treatment']
pro_s.fit(X,treatment)
the_pro_score = pro_s.predict(X)

tps = pd.DataFrame(the_pro_score)
see_data(tps,0,width= 0.0001)
for i in range(0,11):
 see_data(data,i+1,0.01)
```

7.3.3 feature_importance.ipynb

```
import pandas as pd
```

```
import numpy as np
from matplotlib import pyplot as plt
from sklearn.model_selection import train_test_split
from xgboost import XGBRegressor
from causalml.inference.meta import XGBTRegressor
from causalml.inference.meta import BaseSLearner, BaseTLearner,
BaseRLearner, BaseXLearner
from causalml.inference.tree import UpliftRandomForestClassifier
from causalml.dataset import *
from causalml.metrics import *
import sklearn
from sklearn.ensemble import
StackingRegressor,VotingRegressor,VotingClassifier,RandomForestRegres
sor
from sklearn import svm
import xgboost as xgb
import random
from sklearn.linear_model import LinearRegression, LogisticRegression
from sklearn.feature_selection import SelectFromModel
from causalml.propensity import GradientBoostedPropensityModel
from causalml.feature_selection.filters import FilterSelect
np.random.seed(42)
# 准备数据集，将特征和标签分开
data = pd.read_csv('./data/-999_train.csv')  # 读取数据集
data = data.iloc[:,1:]

X = data.drop(['y','treatment'],axis=1) # 特征列
y = data['y']  # 标签列
treatment = data['treatment']  # 处理组标识
feature_names = X.head(0)

print('特征、标签、干预的大小为: ',X.shape,y.shape,treatment.shape)
# 将数据集分为训练集和测试集
X_train, X_test, y_train, y_test, t_train, t_test = train_test_split(X,
y, treatment, test_size=0.1,random_state=42)
############使用随机森林特征重要性
# 创建随机森林分类器
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier()
# 使用随机森林分类器拟合数据
rf.fit(X_train, y_train)
# 获取特征重要性
importance = rf.feature_importances_
print(importance)
```

```
# 创建选择器，并基于特征重要性进行特征选择
selector = SelectFromModel(rf, threshold=1e-02)  # 设置重要性阈值
selector.fit(X_train, y_train)

# 获取选择后的特征矩阵
X_selected = selector.transform(X)

# 获取选择后的特征列名
selected_features = X.columns[selector.get_support()]  # 使用columns
获取列名
print("选择后的特征矩阵:")
print(X_selected)
print("选择后的特征列名:")
print(selected_features)

##########使用RLearner 自身特征重要性
##使用集成模型
def
make_models(nums,max_depth_list,n_estimators_list,learning_rate_list,
min_child_weight,spw ):
    models=[]
    for i in range(nums):
        models.append((str(i),xgb.XGBRegressor(max_depth=max_depth_li
st,learning_rate=0.06,gamma=0, min_child_weight =
min_child_weight ,reg_alpha=0, # noqa: E501
         n_estimators=n_estimators_list ,scale_pos_weight=spw)))
    return models

model1=make_models(3,200,200,0.06,spw=0.8,min_child_weight=0.5)
model2=make_models(3,260,125,0.06,spw=0,min_child_weight=4)
stacking_model1 = StackingRegressor(estimators=model1,cv=5, n_jobs=5)
stacking_model2 = StackingRegressor(estimators=model2,cv=5, n_jobs=5)


rlearner = BaseRLearner(learner=None,
        outcome_learner=stacking_model1,
        effect_learner=stacking_model2,
        ate_alpha=3,
        n_fold=5,
        random_state=42)



rlearner.fit(X=X_train, treatment=t_train, y=y_train)
```

```
r_tau = rlearner.predict(X=X_train)
model_tau_feature = RandomForestRegressor()
# specify model for model_tau_feature
importance =  rlearner.get_importance(X=X_train, tau=r_tau,
model_tau_feature=model_tau_feature,
                      normalize=True, method='auto',random_state=42)
pd.set_option('display.max_rows', 200)
print(importance)
# Using the feature_importances_ method in the base learner
(LGBMRegressor() in this example)
rlearner.plot_importance(X=X_train, tau=r_tau, normalize=True,
method='auto')
###########使用SHAP 指标获得特征重要性
shap_rlearner = rlearner.get_shap_values(X=X_train, tau=r_tau)

# 使用这些列名绘制 SHAP 图
rlearner.plot_shap_values(X=X_train, tau=r_tau,max_display=161)


# interaction_idx set to 'auto' (searches for feature with greatest
approximate interaction)
rlearner.plot_shap_dependence(treatment_group=1,
                              feature_idx=1,
                              X=X_train,
                              tau=r_tau,
                              interaction_idx='auto')
###########使用Filter 方式获得特征重要性
from causalml.metrics import *
from causalml.feature_selection.filters import FilterSelect
filter_method = FilterSelect()
data = pd.read_csv('./data/-999_train.csv')  # 读取数据集
data = data.iloc[:,1:]

X = data.drop(['y','treatment'],axis=1) # 特征列
y = data['y']  # 标签列
treatment = data['treatment']  # 处理组标识
feature_names = X.head(0)
# F Filter with order 1
method = 'ED'
f_imp = filter_method.get_importance(data, feature_names, 'y', method,
                                     experiment_group_column="treatment",c
ontrol_group=0,
                                     treatment_group=1,
                                     n_bins=5
```

```
                                ,order=2)


print(f_imp.iloc[0:10,1])

linear_importance = f_imp.iloc[:,1]
```

7.3.4 deep_fs.ipynb

```
import pandas as pd
import numpy as np
train = pd.read_csv('./data/-999_train.csv')
#train = train.drop('y')

selected_cols = ['id','x2', 'x14',
'x94','x4','x93','x112','x5','x12','x39']
  # 选择感兴趣的列

# 筛选出感兴趣的列形成新的特征矩阵
train = train[selected_cols]
import featuretools as ft
es = ft.EntitySet(id='single_dataframe')  # 用id标识实体集

es.add_dataframe(  dataframe_name = 'train_data',
            dataframe=train
            )

trans_primitives=['add_numeric',
ft.primitives.SubtractNumeric(commutative=False),'multiply_numeric']

#ft.list_primitives()

feature_matrix, feature_names = ft.dfs(entityset=es,
    target_dataframe_name = 'train_data',
     max_depth=1,
     verbose=1,
     trans_primitives=trans_primitives
)
feature_matrix.to_csv('./data/train_feature.csv',index= False)
```

```
test = pd.read_csv('./data/-999B_test.csv')
test = test[selected_cols]
es_test = ft.EntitySet(id='test')  # 用id标识实体集
# 增加一个数据框，命名为iris
es_test.add_dataframe(  dataframe_name = 'test_data',
            dataframe=test
            )

feature_matrix_test, feature_names = ft.dfs(entityset=es_test,
    target_dataframe_name = 'test_data',
     max_depth=1,
     verbose=1,
     trans_primitives=trans_primitives
)
feature_matrix_test.to_csv('./data/Btest_feature.csv',index = False)
X = pd.read_csv('./data/-999_train.csv')
Y = pd.read_csv('./data/-999B_test.csv')
XF = pd.read_csv('./data/train_feature.csv')
YF = pd.read_csv('./data/Btest_feature.csv')
selected_cols = ['x2', 'x14', 'x94','x4','x93','x112','x5','x12','x39']
XF = XF.drop(selected_cols,axis=1)
YF = YF.drop(selected_cols,axis=1)
print(XF.shape,YF.shape)
final_train =  pd.concat([X,XF], axis=1)
final_test =  pd.concat([Y, YF], axis=1)
print(final_train.shape,final_test.shape)
final_train.to_csv('./data/final_train.csv',index=False)
final_test.to_csv('./data/Bfinal_test.csv',index=False)
```

7.3.5 conbine_data.ipynb

```
import pandas as pd
import numpy as np
X = pd.read_csv('./data/train.csv')
Y = pd.read_csv('./data/B_test.csv')
X = X.replace([np.nan], -1)
Y = Y.replace([np.nan], -1)
X.to_csv('./data/-999_train.csv',index =False)
Y.to_csv('./data/-999B_test.csv',index =False)
```

```
print(X.shape)
print(Y.shape)
XF = pd.read_csv('./data/train_feature.csv')
YF = pd.read_csv('./data/Btest_feature.csv')
selected_cols = ['x2', 'x14', 'x94','x4','x93','x112','x5','x12','x39']

XF = XF.drop(selected_cols,axis=1)
YF = YF.drop(selected_cols,axis=1)
print(XF.shape,YF.shape)
final_train =  pd.concat([X,XF], axis=1)
final_test =  pd.concat([Y, YF], axis=1)
print(final_train.shape,final_test.shape)
final_train.to_csv('./data/final_train.csv',index=False)
final_test.to_csv('./data/Bfinal_test.csv',index=False)
```

7.3.6 try_Rlearner.ipynb

```
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
from sklearn.model_selection import train_test_split
from xgboost import XGBRegressor
from causalml.inference.meta import XGBTRegressor
from causalml.inference.meta import BaseSLearner, BaseTLearner,
BaseRLearner, BaseXLearner
from causalml.inference.tree import UpliftRandomForestClassifier
from causalml.dataset import *
from causalml.metrics import *
import sklearn
from sklearn.ensemble import
StackingRegressor,VotingRegressor,VotingClassifier
from sklearn import svm
import xgboost as xgb
import random
from sklearn.linear_model import LinearRegression, LogisticRegression
from sklearn.feature_selection import SelectFromModel
from causalml.propensity import GradientBoostedPropensityModel
from causalml.feature_selection.filters import FilterSelect
np.random.seed(42)
# 准备数据集，将特征和标签分开
```

```
data = pd.read_csv('./data/final_train.csv')  # 读取数据集
data = data.iloc[:,1:]

X = data.drop(['y','treatment'],axis=1) # 特征列
y = data['y']  # 标签列
treatment = data['treatment']  # 处理组标识
feature_names = X.head(0)

print('特征、标签、干预的大小为：',X.shape,y.shape,treatment.shape)
# 将数据集分为训练集和测试集
X_train, X_test, y_train, y_test, t_train, t_test = train_test_split(X,
y, treatment, test_size=0.3,random_state=42)
################挑选出感兴趣的列
filter_method = FilterSelect()
# F Filter with order 2
method = 'ED'
f_imp = filter_method.get_importance(data, feature_names, 'y', method,
                                     experiment_group_column="treatment",c
ontrol_group=0,
                                     treatment_group=1,
                                     n_bins=5,
                                     null_impute='median'
                                     ,order=2)


print(f_imp.iloc[0:10,1])

linear_importance = f_imp.iloc[:,1]
##########选取两个钳位对个特征进行选取
ton_n1 = 125
top_n2 = 175
X_train = X_train[linear_importance[ton_n1:top_n2+1]]
X_test = X_test[linear_importance[ton_n1:top_n2+1]]
print(X_train.shape)
##########计算倾向性得分
kwa = {
            "max_depth": 8,
            "learning_rate": 0.1,
            "n_estimators": 100,
            "objective": "binary:logistic",
            "nthread": -1,
            "colsample_bytree": 0.8,
            "random_state": 42,
```

```
        }
pro_s =  GradientBoostedPropensityModel (early_stop=False,
clip_bounds=(1e-3, 1 - 1e-3), **kwa)

pro_score_train=pro_s.fit_predict(X_train,t_train)
pro_score_test=pro_s.fit_predict(X_test,t_test)


##尝试使用集成模型
def
make_models(nums,max_depth_list,n_estimators_list,learning_rate_list,
min_child_weight,spw ):
    models=[]
    for i in range(nums):
        models.append((str(i),xgb.XGBRegressor(max_depth=max_depth_li
st,learning_rate=0.06,gamma=0, min_child_weight =
min_child_weight ,reg_alpha=0, # noqa: E501
         n_estimators=n_estimators_list ,scale_pos_weight=spw)))
    return models

model1=make_models(3,200,200,0.06,spw=0.8,min_child_weight=0.5)
model2=make_models(3,260,125,0.06,spw=0,min_child_weight=4)
stacking_model1 = StackingRegressor(estimators=model1,cv=5, n_jobs=5)
stacking_model2 = StackingRegressor(estimators=model2,cv=5, n_jobs=5)
voting_model1 = VotingRegressor(estimators=model1)
voting_model2 = VotingRegressor(estimators=model2)

###Try R Learner  此处需要调参


learner_r = BaseRLearner(learner=None,
        outcome_learner=stacking_model1,
        effect_learner=stacking_model2,
        ate_alpha=3,
        n_fold=5,
        random_state=42)
learner_r.fit(X = X_train,
treatment=t_train,y=y_train,p=pro_score_train)

cate_r = learner_r.predict(X=X_test,p=pro_score_test)
print(cate_r.shape)


t_test = np.array(t_test).reshape(9000,1)
```

```
y_test = np.array(y_test).reshape(9000,1)
data_r = np.concatenate((y_test, t_test,-cate_r), axis=1)
data_r = pd.DataFrame(data_r)
data_r.columns = ['y','w','model']
# 计算前 t 个样本的累计增益
get_cumgain(data_r,normalize=False, random_seed=42)
# 可视化前 t 个样本的累计增益，即 Uplift Curve
plot_gain(data_r, normalize=True, random_seed=42, n=300, figsize=(8,
8))
auuc_score(df=data_r, normalize=True, tmle=False)

get_cumlift(data_r,random_seed=42)

plot_lift(data_r,  random_seed=42, n=100, figsize=(8, 8))

sub_data = pd.read_csv('./data/Bfinal_test.csv')  # 读取数据集
sub_feature = sub_data.iloc[:,1:].drop('treatment',axis=1)
sub_t = sub_data['treatment']



sub_feature = sub_feature[linear_importance[ton_n1:top_n2+1]]

sub_score=pro_s.fit_predict(sub_feature,sub_t)
tau_test = learner_r.predict(X=sub_feature,p=sub_score)


from sklearn.preprocessing import MinMaxScaler
for i in range(10000):
    if float(tau_test[i]) <= -1:
        print(tau_test[i])
        tau_test[i] = -1
    if float(tau_test[i]) > 1:
        print(tau_test[i])
        tau_test[i]=1

# scaler = MinMaxScaler()
# lift = scaler.fit_transform(cate_r.reshape(-1, 1))
lift = tau_test

sub = pd.DataFrame({'id':sub_data.iloc[:,0],'lift':lift.squeeze()})
sub.to_csv('sub.csv',index=False)
```

经世济民

敦敦以求

西南财经大学

SOUTHWESTERN UNIVERSITY OF FINANCE AND ECONOMICS

校址：四川成都温江柳台大道555号

电话：028-87092032　传真：028-87092632

邮编：611130

网址：http://www.swufe.edu.cn

Address：Liulin Campus (Main Campus): 555, Liutai Avenue,

Wenjiang District, Chengdu, Sichuan, P. R. China

Tel：86-28-87092032　Fax：86-28-87092632　Postcode：611130