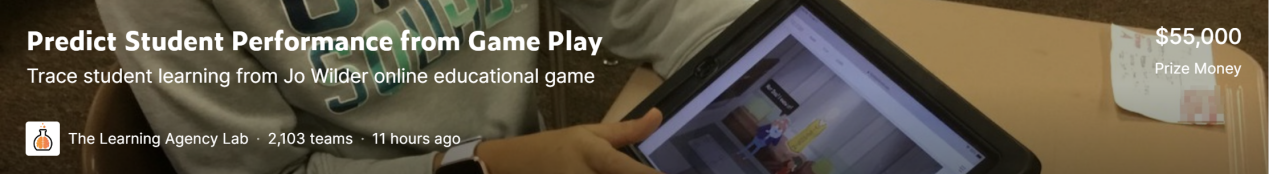
**案例实战：Predict Student Performance from Game Play**

本次比赛以kaggle平台2023年的Predict Student Performance from Game Play竞赛为例讲解数据挖掘预测的实战。包含赛题理解、数据探索、基础框架搭建、特征工程到模型融合以及一些trick应用等方面，直到在公榜获得较好名次，最终方案达到top2.5%银牌区。



1.1 赛题理解

1.1.1 赛题背景

基于游戏进行学习能让学校变得有趣，这种教育方法能让学生在游戏中学习，使其变得有趣和充满活力。尽管基于游戏的学习正在越来越多的教育环境中使用，但能用应用数据科学和学习分析原理来改进基于游戏学习的数据集仍然有限。

大多数基于游戏的学习平台没有充分利用知识追踪来支持个别学生。知识追踪方法是在在线学习环境和智能辅导系统的背景下开发和研究的。但教育游戏中对知识追踪的关注较少。

竞赛主办方 Field Day Lab 是威斯康星州教育研究中心的一个公共资助的研究实验室。他们为许多学科和年龄段设计游戏，将当代研究成果带给公众，利用游戏数据来了解人们如何学习。 Field Day Lab 对可访问性的承诺确保其所有游戏都是免费的并可供任何人使用。该实验室还与学习机构实验室等非营利组织合作，该实验室致力于开发基于学习的工具和项目的科学，以造福社会。

如果成功，参赛者将使游戏开发人员能够改进教育游戏，并通过分析工具进一步支持使用这些游戏的教育工作者。同样的，人们可能会看到对基于游戏的学习平台的更广泛支持。

1.1.2 赛题数据

本次比赛使用了Kaggle的时间序列API。测试数据将分组交付，不允许访问未来的数据。本次比赛的目的是利用在线教育游戏生成的时间序列数据来判断玩家是否会正确回答问题。共有三个问题检查点（level 4, level 12, and level 22），每个检查点都有许多问题。在每个检查点，您都可以访问该部分之前的所有测试数据。本次竞赛的数据集包含以下文件：

* tain.csv：训练集
* test.csv：测试集
* sample\_submission.csv：格式正确的示例提交文件
* train\_labels.csv：训练集中每段游戏时间(session)对应18个问题的correct值

1.1.3 赛题任务

本次比赛的目标是实时预测学生在游戏学习过程中的表现。您将开发一个在最大的开放游戏日志数据集上的模型。

1.1.4 评价指标

本次竞赛使用F1 score作为评价指标，用来计算参赛者提交结果的成绩，具体计算方式如下：

1.1.5 F&Q

Q：需要哪些文件完成建模？

A：您可以访问训练集和标签。每段游戏时间(session)有 18 个问题 ，您不会得到答案，但只会被告知特定游戏时间的用户是否正确回答了每个问题。

Q：我在预测什么？

A：对于每个 <session\_id>\_<question #>，您将预测正确的列，确定您是否相信此特定会话的用户将仅使用会话的先前信息正确回答此问题。timeseries API 按级别顺序向您呈现问题和数据 - 级别段 0-4、5-12 和 13-22 均按顺序提供，您将预测每个段对应问题的正确性。请注意，隐藏的测试集大致与训练集一样大； 预计它的运行时间将比提供的三个测试示例要长得多。

1.2 数据探索

本次比赛提供的数据文件达到了4.74GB，需要先理解每个字段的意义，在了解每个字段的意义后才能更好地进行特征组合及后续建模。

1.2.1 字段类别含义

train.csv与test.csv中的字段及含义

* session\_id：每个游戏段所对应的唯一ID，例如：20090312431273200
* index：每个session中一系列事件的索引：0、1、2…
* elapsed\_time：从这个游戏段开始到这个事件发生时所过去的时间（单位毫秒）
* event\_name：事件类型名称，例如：cutscene\_click
* name：事件名称，例如：basic
* level：事件发生的游戏级别（0-22）
* page：事件中笔记本的第几页（仅仅在笔记本相关事件中出现）
* room\_coor\_x：相对于游戏房间的点击x轴坐标（仅适用于点击事件）
* room\_coor\_y：相对于游戏房间的点击y轴坐标（仅适用于点击事件）
* screen\_coor\_x：参考玩家屏幕的点击x轴坐标（仅适用于点击事件）
* screen\_coor\_y：参考玩家屏幕的点击y轴坐标（仅适用于点击事件）
* hover\_duration：悬停发生的时间（以毫秒为单位）（仅适用于悬停事件）
* text：玩家在此事件中看到的文本，例如：What cha doing over there, Jo?
* fqid：事件的完全限定 ID，例如：intro
* room\_fqid：事件发生的房间完全ID，例如：tunic.historicalsociety.closet
* text\_fqid：文本的完全ID，例如tunic.historicalsociety.closet.intro
* fullscreen：玩家是否全屏玩游戏0/1
* hq：玩家是否开启高清模式玩游戏0/1
* music：玩家是否在游戏中打开声音0/1
* level\_group：该数据行属于哪一组级别以及哪组问题（0-4、5-12、13-22）

train\_labels.csv字段及含义

* session\_id：每个游戏段所对应的唯一ID的问题，例如：20090312431273200\_q1代表了序列为20090312431273200第一个问题的回答情况。
* correct：每个序列对应问题回答正确与否，即我们需要进行预测的值

1.2.2 统计特征

我们根据每个玩家所有序列中下一个elapsed\_time和上一个的差值得到elapsed\_time\_diff，代表该玩家在进行某个事件所花费的时间。在去除字段hq、music和music后可以发现 'event\_name', 'name', 'fqid', 'room\_fqid', 'text\_fqid'为分类字段，'page', 'room\_coor\_x', 'room\_coor\_y', 'screen\_coor\_x', 'screen\_coor\_y', 'hover\_duration', 'elapsed\_time\_diff'为数值字段。

根据ONELUX的开源代码，我们得到包含event\_name 、name 、fqid、room\_fqid、text\_fqid所有值的列表，MATSAMAN也给出了一个包含游戏文本中所有语气词的列表DIALOGS = ['that', 'this', 'it', 'you', 'find', 'found', 'Found', 'notebook', 'Wells','wells','help','need', 'Oh','Ooh','Jo', 'flag', 'can','and','is','the','to']。

基于比pandas占用内存更小的polars库，我们可以构造特征：

# 每个玩家完成游戏的index总和

pl.col("index").count().alias(f"session\_number\_{feature\_suffix}"),

# 玩家经历text文本中包含对话单词c的总数

\*[pl.col('index').filter(pl.col('text').str.contains(c)).count().alias(f'word\_{c}') for c in DIALOGS],

# 每个玩家所有数值字段的均值、标准差、最小值、最大值和中位数

\*[pl.col(c).mean().alias(f"{c}\_mean\_{feature\_suffix}") for c in NUMS],

\*[pl.col(c).std().alias(f"{c}\_std\_{feature\_suffix}") for c in NUMS],

\*[pl.col(c).min().alias(f"{c}\_min\_{feature\_suffix}") for c in NUMS],

\*[pl.col(c).max().alias(f"{c}\_max\_{feature\_suffix}") for c in NUMS],

\*[pl.col(c).median().alias(f"{c}\_median\_{feature\_suffix}") for c in NUMS],

# 玩家fqid、text\_fqid、room\_fqid、event\_name、name、level、level\_group中各个值的数量

\*[pl. col ("fqid"). Filter (pl.col ("fqid") == c). count(). alias (f"{c}\_fqid\_counts {feature\_suffix}") for c in fqid\_lists],

\*[pl. col ("text\_fqid"). Filter (pl.col ("text\_fqid") == c). count(). alias (f"{c}\_text\_fqid\_counts{feature\_suffix}") for c in text\_lists],

\*[pl.col("room\_fqid").filter(pl.col("room\_fqid") == c). count(). alias (f"{c}\_room\_fqid\_counts{feature\_suffix}") for c in room\_lists],

\*[pl.col("event\_name").filter(pl.col("event\_name") == c). count(). alias (f"{c}\_event\_name\_counts{feature\_suffix}") for c in event\_name\_feature],

\*[pl.col("name").filter(pl.col("name") == c).count(). alias (f"{c}\_name\_counts{feature\_suffix}") for c in name\_feature],

\*[pl.col("level").filter(pl.col("level") == c). count(). alias (f"{c}\_LEVEL\_count{feature\_suffix}") for c in LEVELS],

\*[pl.col("level\_group").filter(pl.col("level\_group") == c).count().alias(

f"{c}\_LEVEL\_group\_count{feature\_suffix}") for c in level\_groups],

判断一个玩家是否能够正确回答问题，我们认为其在各个情况下的反应时间十分重要，由此构建基于elapsed\_time\_diff的特征：

# text文本包含对话单词时elapsed\_time\_diff的均值、标准差、最大值、累计值和中位数

\*[pl.col("elapsed\_time\_diff").filter((pl.col('text').str.contains(c))).mean().alias(f'word\_mean\_{c}') for c in DIALOGS],

\*[pl.col("elapsed\_time\_diff").filter((pl.col('text').str.contains(c))).std().alias(f'word\_std\_{c}') for c in DIALOGS],

\*[pl.col("elapsed\_time\_diff").filter((pl.col('text').str.contains(c))).max().alias(f'word\_max\_{c}') for c in DIALOGS],

\*[pl.col("elapsed\_time\_diff").filter((pl.col('text').str.contains(c))).sum().alias(f'word\_sum\_{c}') for c in DIALOGS],

\*[pl.col("elapsed\_time\_diff").filter((pl.col('text').str.contains(c))).median().alias(f'word\_median\_{c}') for c in DIALOGS],

对于剩下的分类字段同样如此构造elapsed\_time\_diff的统计特征…

1.2.3 业务特征

在得到游戏网址<https://pbswisconsineducation.org/jowilder/play-the-game/>后我们进行游戏以此来更好地根据业务构造特征。在游戏第二幕的数据中，我们构建以下特征：

pl.col("elapsed\_time").filter((pl.col("text") == "Here's the log book.") | (pl.col("fqid") == 'logbook.page.bingo')) .apply(lambda s: s.max() - s.min()).alias("logbook\_bingo\_duration"),

pl.col("index").filter((pl.col("text") == "Here's the log book.") | (pl.col("fqid") == 'logbook.page.bingo')).apply(lambda s: s.max() - s.min()). alias ("logbook\_bingo\_indexCount"),

pl.col("elapsed\_time").filter( ((pl.col("event\_name") == 'navigate\_click') & (pl.col("fqid") == 'reader')) | (pl.col("fqid") == "reader.paper2.bingo")).apply(lambda s: s.max() - s.min()).alias("reader\_bingo\_duration"),

pl.col("index").filter(((pl.col("event\_name") == 'navigate\_click') & (pl.col("fqid") == 'reader')) | (pl.col("fqid") == "reader.paper2.bingo")).apply(lambda s: s.max() - s.min()).alias("reader\_bingo\_indexCount"),

pl.col("elapsed\_time").filter(((pl.col("event\_name") == 'navigate\_click') & (pl.col("fqid") == 'journals')) | (pl.col("fqid") == "journals.pic\_2.bingo")).apply(lambda s: s.max() - s.min()).alias("journals\_bingo\_duration"),

pl.col("index").filter(((pl.col("event\_name") == 'navigate\_click') & (pl.col("fqid") == 'journals')) | (pl.col("fqid") == "journals.pic\_2.bingo")).apply(lambda s: s.max() - s.min()).alias("journals\_bingo\_indexCount"),

在游戏第三幕的数据中，我们构建以下特征：

pl.col("elapsed\_time").filter( ((pl.col("event\_name") == 'navigate\_click') & (pl.col("fqid") == 'reader\_flag')) | (pl.col("fqid") == "tunic.library.microfiche.reader\_flag.paper2.bingo")).apply(lambda s: s.max() - s.min() if s.len() > 0 else 0).alias("reader\_flag\_duration"),

pl.col("index").filter( ((pl.col("event\_name") == 'navigate\_click') & (pl.col("fqid") == 'reader\_flag')) | (pl.col("fqid") == "tunic.library.microfiche.reader\_flag.paper2.bingo")).apply(lambda s: s.max() - s.min() if s.len() > 0 else 0).alias("reader\_flag\_indexCount"),

pl.col("elapsed\_time").filter(((pl.col("event\_name") == 'navigate\_click') & (pl.col("fqid") == 'journals\_flag')) | (pl.col("fqid") == "journals\_flag.pic\_0.bingo")).apply(lambda s: s.max() - s.min() if s.len() > 0 else 0).alias("journalsFlag\_bingo\_duration"),

pl.col("index").filter( ((pl.col("event\_name") == 'navigate\_click') & (pl.col("fqid") == 'journals\_flag')) | (pl.col("fqid") == "journals\_flag.pic\_0.bingo")).apply(lambda s: s.max() - s.min() if s.len() > 0 else 0).alias("journalsFlag\_bingo\_indexCount"),

1.3 基础框架

1.3.1 基础特征筛选

由于最后提交时测试数据按照每个玩家level\_group分段，我们在训练时就按照level\_group划分数据集并进行特征构建。

删除空值大于90%的字段：

null1 = df1.isnull().sum().sort\_values(ascending=False) / len(df1)

null2 = df2.isnull().sum().sort\_values(ascending=False) / len(df1)

null3 = df3.isnull().sum().sort\_values(ascending=False) / len(df1)

drop1 = list(null1[null1>0.9].index)

drop2 = list(null2[null2>0.9].index)

drop3 = list(null3[null3>0.9].index)

print(len(drop1), len(drop2), len(drop3))

for col **in** df1.columns:

if df1[col].nunique()==1:

print(col)

drop1.append(col)

print("\*\*\*\*\*\*\*\*\*df1 DONE\*\*\*\*\*\*\*\*\*")

for col **in** df2.columns:

if df2[col].nunique()==1:

print(col)

drop2.append(col)

print("\*\*\*\*\*\*\*\*\*df2 DONE\*\*\*\*\*\*\*\*\*")

for col **in** df3.columns:

if df3[col].nunique()==1:

print(col)

drop3.append(col)

print("\*\*\*\*\*\*\*\*\*df3 DONE\*\*\*\*\*\*\*\*\*")

# 将index设置为session\_id

df1 = df1.set\_index('session\_id')

df2 = df2.set\_index('session\_id')

df3 = df3.set\_index('session\_id')

# 得到每个group\_level对应的特征

FEATURES1 = [c for c **in** df1.columns if c **not** **in** drop1+['level\_group']]

FEATURES2 = [c for c **in** df2.columns if c **not** **in** drop2+['level\_group']]

FEATURES3 = [c for c **in** df3.columns if c **not** **in** drop3+['level\_group']]

1.3.2 模型训练

对于每个玩家需要预测的18个问题，我们均使用五折训练，一共需要训练18\*5个模型，其中level\_group为0-4对应问题1、2、3，level\_group为5-12对应问题4-13，level\_group为13-22对应问题14-18。本次比赛选取的模型均为树模型，包括LightGBM、XGBoost和CatBoost。

1.3.2.1 LightGBM模型训练

Lgb超参数我们设置为：

lgb\_params = {

'boosting\_type': 'gbdt',

'objective': 'binary',

'metric': 'binary\_logloss',

'learning\_rate': 0.05,

'alpha': 8,

'max\_depth': 4,

'subsample': 0.8,

'colsample\_bytree': 0.5,

'random\_state': 42

}

以下是lgb的训练代码：

pred\_lgb = np.zeros((df1.shape[0], 18))

n\_splits = 5

kf = KFold(n\_splits=n\_splits)

for q **in** range(1, 19):

if q <= 3:

grp = '0-4'

df = df1

FEATURES = FEATURES1

elif q <= 13:

grp = '5-12'

df = df2

FEATURES = FEATURES2

elif q <= 22:

grp = '13-22'

df = df3

FEATURES = FEATURES3

lgb\_params['n\_estimators'] = estimators\_lgb[q - 1]

for fold, (train\_idx, val\_idx) **in** enumerate(kf.split(df)):

df\_train = df.iloc[train\_idx] *#.reset\_index(drop=True)*

train\_users = df\_train.index.values

train\_y = targets[targets['session'].isin(list(train\_users))].loc[targets.q == q].set\_index('session')

df\_val = df.iloc[val\_idx] *#.reset\_index(drop=True)*

val\_users = df\_val.index.values

val\_y = targets[targets['session'].isin(list(val\_users))].loc[targets.q == q].set\_index('session')

clf = LGBMClassifier(\*\*lgb\_params)

clf.fit(df\_train[FEATURES].astype('float32'), train\_y['correct'], verbose=0)

clf.booster\_.save\_model(f'LGBM\_question**{**q**}**\_fold**{**fold**}**.lgb')

存储模型到列表中：

models\_list = [[Booster(model\_file = f"LGBM\_question**{**q**}**\_fold**{**fold**}**.lgb") for fold **in** range(5)] for q **in** range(1, 19)]

在得到模型后，对测试集进行预测和提交：

limits = {'0-4':(1,4), '5-12':(4,14), '13-22':(14,19)}

count = 0

for (test, sample\_submission) **in** iter\_test:

session\_id = test.session\_id.values[0]

grp = test.level\_group.values[0]

a, b = limits[grp]

preds = []

*# ------------------- level 0-4 --------------------------------*

if a == 1:

FEATURES = FEATURES1

test = (pl.from\_pandas(test)

.drop(["fullscreen", "hq", "music"])

.with\_columns(columns))

test = feature\_engineer\_pl(test, grp, use\_extra=True, feature\_suffix='')

test = time\_feature(test)

test = test[FEATURES]

*# ------------------- level 5-12 -------------------------------*

elif a == 4:

FEATURES = FEATURES2

test = (pl.from\_pandas(test)

.drop(["fullscreen", "hq", "music"])

.with\_columns(columns))

test = feature\_engineer\_pl(test, grp, use\_extra=True, feature\_suffix='')

test = time\_feature(test)

test = test[FEATURES]

*# ------------------- level 13-22 ------------------------------*

elif a == 14:

FEATURES = FEATURES3

test = (pl.from\_pandas(test)

.drop(["fullscreen", "hq", "music"])

.with\_columns(columns))

test = feature\_engineer\_pl(test, grp, use\_extra=True, feature\_suffix='')

test = time\_feature(test)

test = test[FEATURES]

for q **in** range(a,b):

fold = 0

thresh = 0.63

model\_0 = models\_list[q-1][fold]

model\_1 = models\_list[q-1][fold+1]

model\_2 = models\_list[q-1][fold+2]

model\_3 = models\_list[q-1][fold+3]

model\_4 = models\_list[q-1][fold+4]

pred\_0 = model\_0.predict(test[FEATURES].astype(np.float32))

pred\_1 = model\_1.predict(test[FEATURES].astype(np.float32))

pred\_2 = model\_2.predict(test[FEATURES].astype(np.float32))

pred\_3 = model\_3.predict(test[FEATURES].astype(np.float32))

pred\_4 = model\_4.predict(test[FEATURES].astype(np.float32))

pred = (pred\_0 + pred\_1 + pred\_2 + pred\_3 + pred\_4) / 5

preds.append(int(pred > thresh))

sample\_submission["correct"] = preds

env.predict(sample\_submission)

1.3.2.2 XGB模型训练

导入必要的库：

import pandas as pd, numpy as np, gc

from sklearn.model\_selection import KFold, GroupKFold

from xgboost import XGBClassifier

from sklearn.metrics import f1\_score

使用GroupKFold进行分组抽样，同时构建模型字典存储xgb模型

gkf = GroupKFold(n\_splits=5)

oof = pd.DataFrame(data=np.zeros((len(ALL\_USERS),18)), index=ALL\_USERS)

models = {}

进行模型训练：

for q **in** range(1, 19):

print(f"question**{**q**}**")

if q <= 3:

grp = '0-4'

df = df1

FEATURES = FEATURES1

elif q <= 13:

grp = '5-12'

df = df2

FEATURES = FEATURES2

elif q <= 22:

grp = '13-22'

df = df3

FEATURES = FEATURES3

for i, (train\_index, test\_index) **in** enumerate(gkf.split(X=df, groups=df.index)):

print('Fold:',i+1)

xgb\_params = {

'objective' : 'binary:logistic',

'eval\_metric':'logloss',

'learning\_rate': 0.05,

'max\_depth': 4,

'n\_estimators': 1000,

'early\_stopping\_rounds': 50,

'tree\_method':'hist',

'subsample':0.8,

'colsample\_bytree': 0.4}

train\_x = df.iloc[train\_index] *#.reset\_index(drop=True)*

train\_users = train\_x.index.values

train\_y = targets[targets['session'].isin(list(train\_users))].loc[targets.q == q].set\_index('session')

valid\_x = df.iloc[test\_index] *#.reset\_index(drop=True)*

valid\_users = valid\_x.index.values

valid\_y = targets[targets['session'].isin(list(valid\_users))].loc[targets.q == q].set\_index('session')

clf = XGBClassifier(\*\*xgb\_params)

clf.fit(train\_x[FEATURES].astype('float32'), train\_y['correct'], eval\_set=[ (valid\_x[FEATURES].astype('float32'), valid\_y['correct']) ], verbose=0)

clf.save\_model(f'XGB\_question**{**q**}**\_fold**{**i**}**.xgb')

print(f'**{**q**}**(**{**clf.best\_ntree\_limit**}**), ',end='')

models[f'**{**grp**}**\_**{**i**}**\_**{**q**}**'] = clf

oof.loc[valid\_users, q-1] = clf.predict\_proba(valid\_x[FEATURES].astype('float32'))[:,1]

1.3.2.3 Cat训练

导入库：

from catboost import CatBoostClassifier, Pool

模型训练：

for q **in** range(1, 19):

print(f"question**{**q**}**")

if q <= 3:

grp = '0-4'

df = df1

FEATURES = FEATURES1

elif q <= 13:

grp = '5-12'

df = df2

FEATURES = FEATURES2

elif q <= 22:

grp = '13-22'

df = df3

FEATURES = FEATURES3

for i, (train\_index, test\_index) **in** enumerate(gkf.split(X=df, groups=df.index)):

print('Fold:',i+1)

cat\_params = {

'iterations': 1000,

'early\_stopping\_rounds': 90,

'depth': 5,

'learning\_rate': 0.02,

'loss\_function': "Logloss",

'random\_seed': 222222,

'metric\_period': 1,

'subsample': 0.8,

'colsample\_bylevel': 0.4,

'verbose': 0,

'l2\_leaf\_reg': 20,

}

train\_x = df.iloc[train\_index] *#.reset\_index(drop=True)*

train\_users = train\_x.index.values

train\_y = targets[targets['session'].isin(list(train\_users))].loc[targets.q == q].set\_index('session')

valid\_x = df.iloc[test\_index] *#.reset\_index(drop=True)*

valid\_users = valid\_x.index.values

valid\_y = targets[targets['session'].isin(list(valid\_users))].loc[targets.q == q].set\_index('session')

clf = CatBoostClassifier(\*\*cat\_params)

clf.fit(train\_x[FEATURES].astype('float32'), train\_y['correct'],

eval\_set=[ (valid\_x[FEATURES].astype('float32'), valid\_y['correct']) ],

verbose=0)

clf.save\_model(f'Cat\_question**{**q**}**\_fold**{**i**}**.cbm')

models[f'**{**grp**}**\_**{**i**}**\_**{**q**}**'] = clf

oof.loc[valid\_users, q-1] = clf.predict\_proba(valid\_x[FEATURES].astype('float32'))[:,1]

1.3.3 模型本地cv

根据train\_labels.csv将其session\_id分列得到每个玩家及其对应问题的正确性：

targets = pd.read\_csv('/kaggle/input/predict-student-performance-from-game-play/train\_labels.csv')

targets['session'] = targets.session\_id.apply(lambda x: int(x.split('\_')[0]) )

targets['q'] = targets.session\_id.apply(lambda x: int(x.split('\_')[-1][1:]) )

构建真实正确率df：

true = oof.copy()

for k **in** range(18):

*# GET TRUE LABELS*

tmp = targets.loc[targets.q == k+1].set\_index('session').loc[ALL\_USERS]

true[k] = tmp.correct.values

寻找最佳阈值：

scores = []; thresholds = []

best\_score = 0; best\_threshold = 0

for threshold **in** np.arange(0.4,0.81,0.01):

print(f'**{**threshold**:**.02f**}**, ',end='')

preds = (oof.values.reshape((-1))>threshold).astype('int')

m = f1\_score(true.values.reshape((-1)), preds, average='macro')

scores.append(m)

thresholds.append(threshold)

if m>best\_score:

best\_score = m

best\_threshold = threshold

绘制最佳阈值图像及本地cv分数

import matplotlib.pyplot as plt

*# PLOT THRESHOLD VS. F1\_SCORE*

plt.figure(figsize=(20,5))

plt.plot(thresholds,scores,'-o',color='blue')

plt.scatter([best\_threshold], [best\_score], color='blue', s=300, alpha=1)

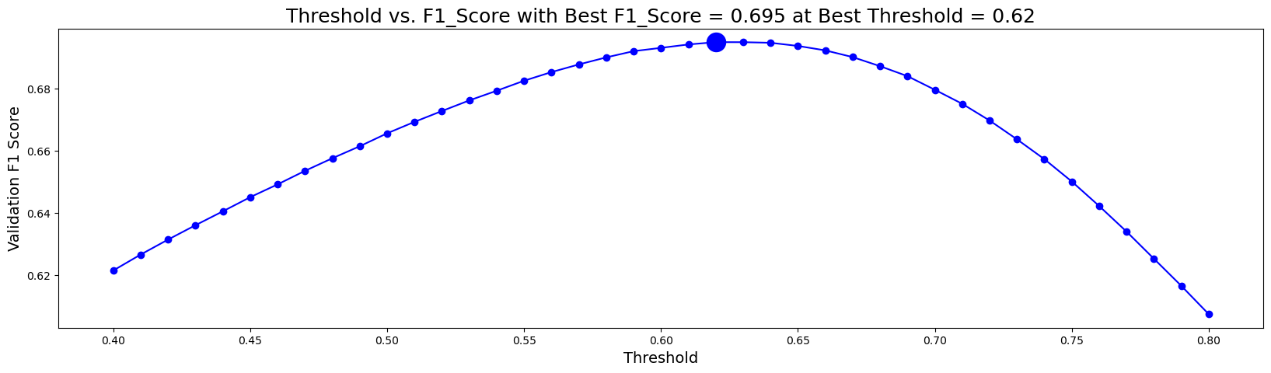
plt.xlabel('Threshold',size=14)

plt.ylabel('Validation F1 Score',size=14)

plt.title(f'Threshold vs. F1\_Score with Best F1\_Score = **{**best\_score**:**.3f**}** at Best Threshold = **{**best\_threshold**:**.3**}**',size=18)

plt.show()

其中基础xgb模型为：



可以基础xgb的最佳阈值为0.62，本地cv为0.695，Public Score为0.695，Private Score为0.696，其中Private Score为本次竞赛最后所看的分数。

获取每个问题的f1分数：

for k **in** range(18):

m = f1\_score(true[k].values, (oof[k].values>best\_threshold).astype('int'), average='macro')

print(f'Q**{**k**}**: F1 =',m)

m = f1\_score(true.values.reshape((-1)), (oof.values.reshape((-1))>best\_threshold).astype('int'), average='macro')

print('==> Overall F1 =',m)

1.3.4 基础模型分数

|  |  |  |  |
| --- | --- | --- | --- |
|  | CV | Public Score | Private Score |
| LGB | 0.693 | 0.693 | 0.694 |
| XGB | 0.695 | 0.695 | 0.696 |
| Cat | 0.696 | 0.697 | 0.695 |

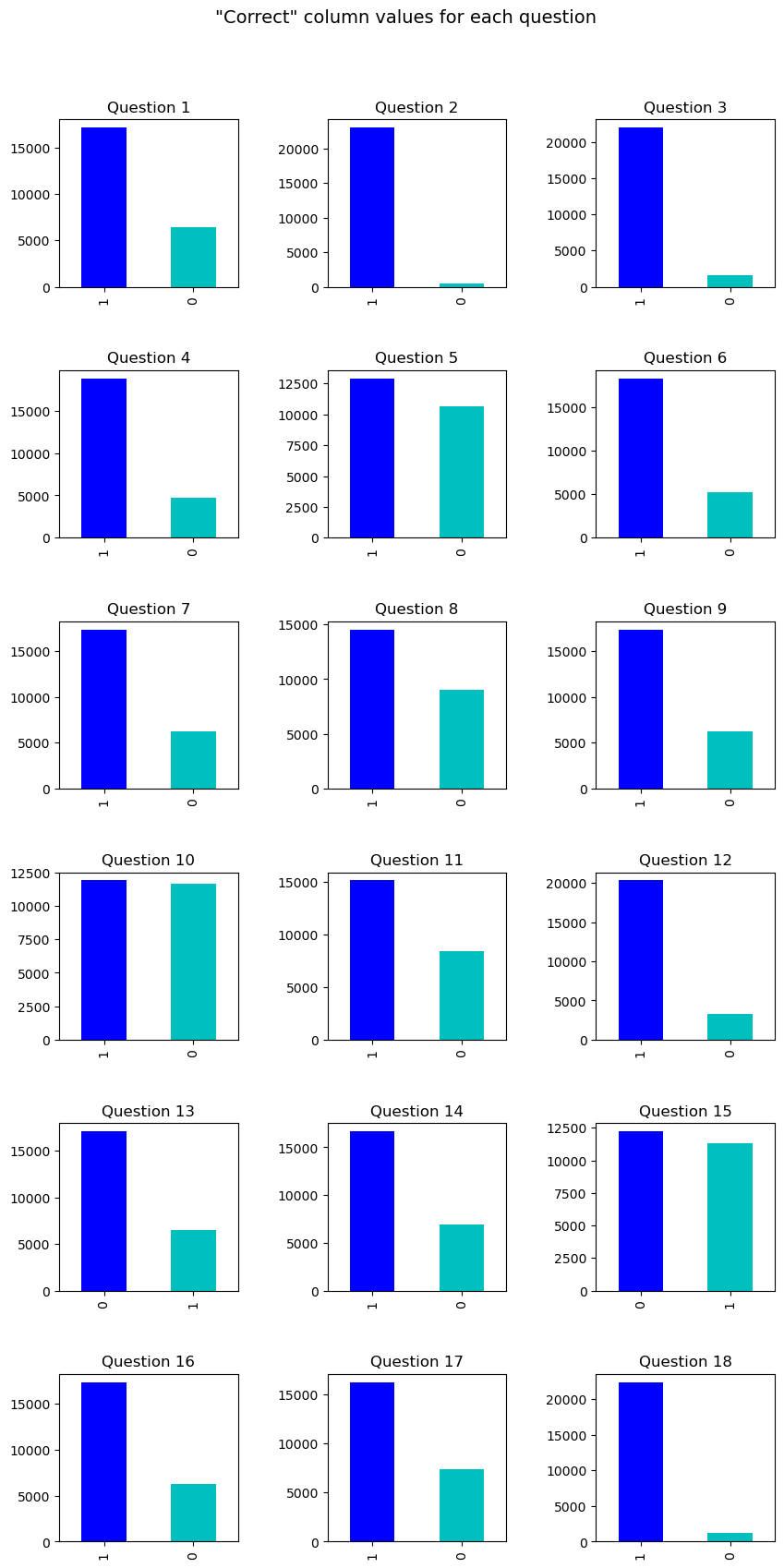
从基础单模来看，XGB在私榜上获得了最高的分数，Cat在本地交叉验证和公榜获得了较高的分数。

1.4 分数提升

在开源代码的特征上，我们使用了三种树模型建立了基础框架，在后续的比赛过程中，我们会在这三个框架上不断增加新的技巧以及特征构建，以此来获得更高的分数。

1.4.1 问题准确率分布

参考GUSTHEMA的代码，我们可以发现问题正确率的分布规律，容易发现问题2和问题18中学生基本能够回答出所有问题，而使用基本模型对这两个问题进行预测得到的f1分数往往是这18个问题中最低的。我们使用trick，直接预测问题2和问题18的correct为1，可以提升最后的分数。



除了直接预测问题2和18的correct为1以外，我们还可以使用一个较低的阈值来保留问题为0的可能性，比如：

mask = sample\_submission.session\_id.str.contains(f'q**{**q**}**')

sample\_submission.loc[mask,'correct'] = int( avg\_p > 0.1)

1.4.2 数据集合并

我们注意到在提交时，由于我们获得的测试集是一个个level\_group得到的，无法在预测前几个问题时获得后面level\_group的数据集，但我们能够通过合并之前得到level\_group的测试集来获得更多的数据，提高后续问题准确率。在模型训练时候，我们将原数据集按照三个level\_group进行拆分后分别进行训练，但如果我们对于第二个level\_group:5-12，我们将其与第一个level\_group进行合并，然后将最后一个level\_group:13-22与之前两个合并。这样在训练时候，后面的问题可以使用更多的数据进行训练，能够提高f1 score，其中训练集的划分为:

df1 = df.filter(pl.col("level\_group")=='0-4')

df2 = df.filter((pl.col("level\_group")=='5-12') | (pl.col("level\_group")=='0-4'))

df3 = df

在提交时，我们需要对得到的数据集进行合并来预测后续问题，我们通过计数器counter来记录当前获得测试集是哪个level\_group的，如果是第一个，则用temp\_1来存储获得的测试集，当测试集为后续level\_group时，将其合并起来，具体代码如下：

limits = {'0-4':(1,4), '5-12':(4,14), '13-22':(14,19)}

counter = 0

for (test, sample\_submission) **in** iter\_test:

test = test.sort\_values(by = 'index')

session\_id = test.session\_id.values[0]

grp = test.level\_group.values[0]

a,b = limits[grp]

if counter % 3 == 0:

print(test.level\_group.values[0], 'FEATURES1')

temp\_1 = test

FEATURES = FEATURES1

test = (pl.from\_pandas(temp\_1)

.sort(["session\_id", "elapsed\_time"])

.drop(["fullscreen", "hq", "music"])

.with\_columns(columns))

test = feature\_engineer\_04(test)

test = test[FEATURES]

elif counter % 3 == 1:

print(test.level\_group.values[0], 'FEATURES2')

temp\_2 = pd.concat([temp\_1, test])

FEATURES = FEATURES2

test = (pl.from\_pandas(temp\_2)

.sort(["session\_id", "elapsed\_time"])

.drop(["fullscreen", "hq", "music"])

.with\_columns(columns))

test = feature\_engineer\_512(test)

test = test[FEATURES]

elif counter % 3 == 2:

print(test.level\_group.values[0], 'FEATURES3')

temp\_3 = pd.concat([temp\_2, test])

FEATURES = FEATURES3

test = (pl.from\_pandas(temp\_3)

.sort(["session\_id", "elapsed\_time"])

.drop(["fullscreen", "hq", "music"])

.with\_columns(columns))

test = feature\_engineer\_1322(test)

test = test[FEATURES]

for q **in** range(a,b):

if q == 2:

mask = sample\_submission.session\_id.str.contains(f'q**{**q**}**')

sample\_submission.loc[mask,'correct'] = 1

elif q == 18:

mask = sample\_submission.session\_id.str.contains(f'q**{**q**}**')

sample\_submission.loc[mask,'correct'] = 1

else:

total\_p = 0

print(f'question**{**q**}**', list\_contains\_all\_elements(test.columns, top500\_features\_list[q-1]))

temp\_test = test[top500\_features\_list[q-1]]

for i **in** range(5):

clf = models[f'**{**grp**}**\_**{**i**}**\_**{**q**}**']

p = clf.predict\_proba(temp\_test.astype('float32'))[0,1]

total\_p = total\_p + p

avg\_p = total\_p / 5

mask = sample\_submission.session\_id.str.contains(f'q**{**q**}**')

sample\_submission.loc[mask,'correct'] = int( avg\_p > best\_score)

env.predict(sample\_submission)

counter += 1

在使用了“额外”的数据集对模型进行训练和提交后，可以发现除了第一个level\_group以外，后续的其他level\_group对应问题的分数均有了提高。公榜的分数均提升了至少0.001。

1.4.3 改变抽样方式

开源代码通常使用的抽样方式为分组抽样：

gkf = GroupKFold(n\_splits=5)

然而我们本身就需要对这些问题进行分组预测，这样的抽样方式并就没有什么意义，我们将原来的分组抽样改成基于训练样本correct分布的分层抽样，代码如下：

skf = StratifiedKFold(n\_splits=5, shuffle=False)

我们使用的是五折分层抽样，并没有打乱顺序，如果需要打乱顺序可以将shuffle为False改为True同时加入参数random\_state=42（其中数字随意）。使用五折分层抽样意味着每次抽样会将correct为“0”和“1”的数据各留出1/5作为验证集，这五次抽样的验证集的并集是整个训练集。

通过改变抽样方式，我们能够略微提升模型分数。

1.4.4 特征筛选

特征筛选的目的是减少特征空间的维度，提高模型的性能、减少过拟合的风险，并增加对数据的理解和解释能力。

通过选择与预测目标相关性较高的特征，可以提高机器学习模型的预测性能。不相关或冗余的特征可能会引入噪声或干扰，导致模型性能下降。特征筛选可以剔除这些无关或冗余的特征，使模型更专注于重要的特征，从而提高预测准确性。

过多的特征可能导致模型过于复杂，容易出现过拟合的问题。过拟合指的是模型过度拟合训练数据，无法很好地泛化到新数据。通过特征筛选，可以减少特征的数量，降低模型的复杂性，从而减少过拟合的风险。

通过减少特征的数量，可以降低机器学习模型的计算资源需求和训练时间开销。在大规模数据集和复杂模型的情况下，特征筛选可以显著提高计算效率，加快模型训练和推断的速度。

我们在lgb的框架上筛选重要性为前500的特征：

dfs = []

top500\_features\_list = []

for q **in** range(1, 19):

*# USE THIS TRAIN DATA WITH THESE QUESTIONS*

print(f"question**{**q**}**")

if q <= 3:

grp = '0-4'

df = df1

n = 400

FEATURES = FEATURES1

elif q <= 13:

grp = '5-12'

df = df2

n = 500

FEATURES = FEATURES2

elif q <= 22:

grp = '13-22'

df = df3

n = 700

FEATURES = FEATURES3

temp = targets.loc[targets['question'] == q]

*# temp = temp.set\_index('session')*

dfs.append(temp)

lgb\_params = {

'boosting\_type': 'gbdt',

'objective': 'binary',

'metric': 'binary\_logloss',

'learning\_rate': 0.05,

'alpha': 8,

'max\_depth': 4,

'subsample': 0.8,

'colsample\_bytree': 0.5,

'random\_state': 42

}

lgb\_params['n\_estimators'] = estimators\_lgb[q - 1]

question\_feature\_importances = np.zeros(len(FEATURES))

for i, (train\_index, test\_index) **in** enumerate(sss.split(dfs[q-1], y=dfs[q-1]['correct'])):

print('Fold:',i+1)

train\_users = dfs[q-1].iloc[train\_index].session.values

train\_x = df[df.index.isin(list(train\_users))]

train\_y = targets[targets['session'].isin(list(train\_users))].loc[targets.question == q].set\_index('session')

train\_x = train\_x.sort\_index()

train\_y = train\_y.sort\_index()

valid\_users = dfs[q-1].iloc[test\_index].session.values

valid\_x = df[df.index.isin(list(valid\_users))]

valid\_y = targets[targets['session'].isin(list(valid\_users))].loc[targets.question == q].set\_index('session')

valid\_x = valid\_x.sort\_index()

valid\_y = valid\_y.sort\_index()

*# TRAIN MODEL*

clf = LGBMClassifier(\*\*lgb\_params, importance\_type='gain')

clf.fit(train\_x[FEATURES].astype('float32'), train\_y['correct'],

eval\_set=[ (valid\_x[FEATURES].astype('float32'), valid\_y['correct']) ],

verbose=0)

*# clf.booster\_.save\_model(f'LGB\_question{q}\_fold{i}.lgb')*

print(f'Model saved for question **{**q**}** fold **{**i**}** with iterations = **{**estimators\_lgb[q-1]**}**')

*# SAVE MODEL, PREDICT VALID OOF*

models[f'**{**grp**}**\_**{**i**}**\_**{**q**}**'] = clf

oof.loc[valid\_users, q-1] = clf.predict\_proba(valid\_x[FEATURES].astype('float32'))[:,1]

importances = clf.feature\_importances\_

question\_feature\_importances += importances

sorted\_indices = np.argsort(question\_feature\_importances)[::-1]

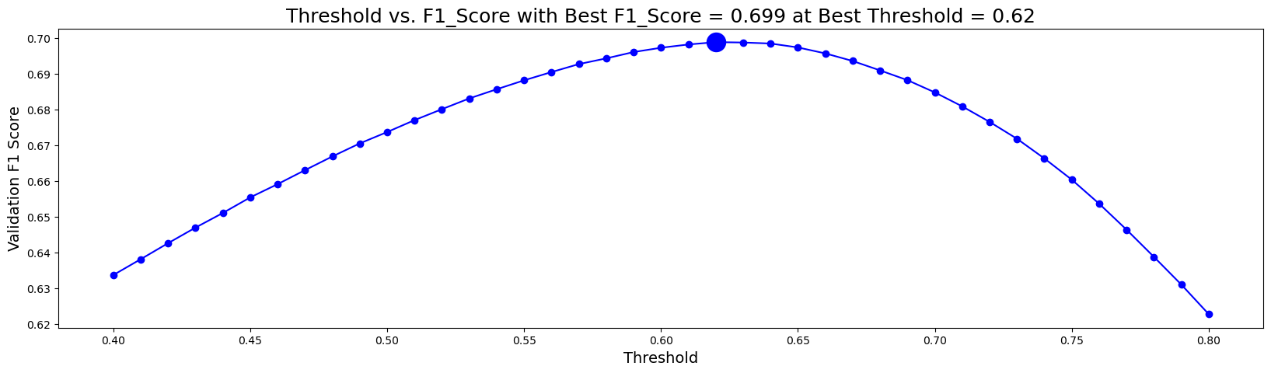
top500\_features = [FEATURES[i] for i **in** sorted\_indices[:n]]

top500\_features\_list.append(top500\_features)

top500\_features\_list包含了每个问题的前500重要性的特征，后续重新训练时候读入特征，得到lgb模型：

clf.fit(train\_x[top500\_features\_list[q-1]].astype('float32'), train\_y['correct'], eval\_set=[ (valid\_x[top500\_features\_list[q-1]].astype('float32'), valid\_y['correct']) ], verbose=0)

我们在lgb模型的框架下得到：



可以看到本地交叉验证cv分数达到0.699，而公榜分数为0.698，私榜分数达到了0.701。由于该模型的公榜分数较低，被我们放弃。

1.4.5 XGB+GPU

虽然比赛规定不能使用GPU和Internet我们可以使用GPU进行训练，在得到模型后infer得到submission，其中XGB只需要修改一行代码即可：

xgb\_params = {

'objective' : 'binary:logistic',

'tree\_method': 'gpu\_hist',

'eval\_metric':'logloss',

'learning\_rate': 0.05,

'max\_depth': 4,

'n\_estimators': 1000,

'early\_stopping\_rounds': 50,

'subsample':0.8,

'colsample\_bytree': 0.4}

将tree\_method从原来的hist改为gpu\_hist即可。

其中cv分数为0.696，公榜分数为0.699，私榜分数为0.698.

1.4.6 特征工程

在进行大量特征实验后，加入了很多分位数特征与elapsed\_time\_diff更细精度的特征：

\*[pl.col("elapsed\_time\_diff").filter(pl.col('text').str.contains(c)).quantile(0.3).alias(f'quantile0.3&word\_max\_**{**c**}**') for c **in** DIALOGS if f'quantile0.3&word\_max\_**{**c**}**' **in** all\_fea],

\*[pl.col("elapsed\_time\_diff").filter(pl.col('text').str.contains(c)).quantile(0.5).alias(f'quantile0.5word\_max\_**{**c**}**') for c **in** DIALOGS if f'quantile0.5word\_max\_**{**c**}**' **in** all\_fea],

\*[pl.col("elapsed\_time\_diff").filter(pl.col('text').str.contains(c)).quantile(0.65).alias(f'quantile0.65word\_max\_**{**c**}**') for c **in** DIALOGS if f'quantile0.65word\_max\_**{**c**}**' **in** all\_fea],

\*[pl.col("elapsed\_time\_diff").filter(pl.col('text').str.contains(c)).quantile(0.8).alias(f'quantile0.8word\_max\_**{**c**}**') for c **in** DIALOGS if f'quantile0.8word\_max\_**{**c**}**' **in** all\_fea],

…

\*[pl.col("elapsed\_time\_diff").filter((pl.col("event\_name")==e)&(pl.col("name")==n)&(pl.col("room\_fqid")==r)).sum().alias(f"etd&name**{**n**}**&eventname**{**e**}**&room**{**r**}**") for e **in** event\_name\_feature for n **in** name\_feature for r **in** room\_lists if f"etd&name**{**n**}**&eventname**{**e**}**&room**{**r**}**" **in** all\_fea],

\*[pl.col("elapsed\_time\_diff").filter((pl.col("event\_name")==e)&(pl.col("name")==n)&(pl.col("level")==l)).sum().alias(f"etd&name**{**n**}**&eventname**{**e**}**&level**{**l**}**") for e **in** event\_name\_feature for n **in** name\_feature for l **in** LEVELS if f"etd&name**{**n**}**&eventname**{**e**}**&level**{**l**}**" **in** all\_fea],

\*[pl.col("elapsed\_time\_diff").filter((pl.col("room\_fqid")==r)&(pl.col('text').str.contains(d))).sum().alias(f"etd&room**{**r**}**&diag**{**d**}**") for d **in** DIALOGS for r **in** room\_lists if f"etd&room**{**r**}**&diag**{**d**}**" **in** all\_fea],

\*[pl.col("elapsed\_time\_diff").filter((pl.col("text\_fqid")==t)&(pl.col('text').str.contains(d))).sum().alias(f"etd&text**{**t**}**&diag**{**d**}**") for d **in** DIALOGS for t **in** text\_lists if f"etd&text**{**t**}**&diag**{**d**}**" **in** all\_fea],

XGB需要大量的特征进行尝试，所有特征在筛选之前加起来大约1万多个，在polars库和GPU的使用下，时间还算可控。我们最终得到本地CV分数为0.7，公榜分数为0.7，私榜分数为0.698.

1.4.7 模型融合

我们舍去了耗费时间很长的Cat以及cv和公榜分数相差较大的lgb，也由于时间有限，我们融合的模型均为XGB+GPU模型。

融合的三个模型的特征工程有所差别，特征选取数量有所区别。我们将分数最好的单模给予权重0.7，剩下两个按照分数设置权重为0.2和0.1。同时将问题2、18和12的correct直接设置为1，问题13的预测correct直接设置为0.

最终得到公榜分数为0.702，私榜分数为0.7.这个融合方案也是最后拿到银牌方案。