机器学习大作业-实验报告

1 参赛的 ID

default13275126

2 分组情况

徐怡 SA23011089

胡科伟 SA23011123

杨飞航 SA23218094

古田辉 SA23011228

3 分工情况

徐怡：在任务中积极探索了多种赛题方案，实现了传统模型来解决预测问题，撰写调研和尝试部分的实验报告，并积极参与小组讨论。

胡科伟：承担了实验代码主体的编写任务，并积极参与小组讨论。

杨飞航：负责撰写实验报告，并积极参与小组讨论

古田辉：负责撰写实验报告，并积极参与小组讨论

4 赛题内容介绍

选题：电商用户购买行为预测。

互联网的出现和广泛普及为用户提供了大量信息，满足了科技时代用户对信息的需求。然而，在线信息的显著增加也带来了“信息过载”的问题。这使得用户在面对庞大的信息量时很难从中筛选出真正有用的内容，从而降低了信息利用的效率。为了帮助用户更快速地过滤出有价值的信息，需要利用真实的用户购买行为记录，并运用机器学习等技术建立电商用户购买行为的可靠预测模型。这些模型旨在预测用户的下一个行动，基于准确识别用户购买兴趣进行个性化商品推荐，最终提高电商平台产品的转化率。本题的目标是利用电子商务平台上真实的用户行为记录，采用机器学习技术构建可靠的电商用户购买行为预测模型，以预测用户下一次可能购买的商品。

5 数据分析

数据整理自一家中等化妆品在线商店公布的网上公开数据集，为该化妆品商

店真实的用户交易信息，数据集中每一行表示一个事件，所有的事件都与商品和用户相关，并且用户的点击行为之间是有时间顺序的。数据集中包含了商品和用户的多个属性，其中包括每个事件的 event_time、event_type、product_id、category_id、category_code、brand、price、user_id、user_session。

这些特征在预测用户下一个购买商品方面发挥着关键作用。首先，event_type 表示事件的类型，如查看、加入购物车或购买，为模型提供了有关用户互动程度和购买意图的重要线索。product_id 和 category_id 提供了对具体产品和产品类别的了解，允许模型学习用户对不同商品的偏好。category_code 进一步提供了产品类别的详细信息，有助于更细致地捕捉用户的兴趣。brand 反映了用户对特定品牌的偏好，对于预测下一个购买的品牌选择至关重要。price 可以影响用户的购买决策，因为不同价格范围内的商品可能对用户有不同的吸引力。user_id 使得模型能够个性化学习用户行为，根据过去的互动历史调整预测。最后，通过综合考虑这些特征，模型可以更全面地理解用户与商品之间的关系，从而提高预测下一个购买商品的准确性。其次，对于购买行为的预测任务而言，通常我们更关注用户的行为模式和趋势，而不是具体的时间戳信息。因此， 'event_time'、'user_session' 和 'timestamp' 这类上下文中可能被认为是无用的数据被我们丢弃。

6 复现说明

下面介绍复现我们的代码，可以采取的步骤。

6.1 程序运行环境

IDE	Visual Studio Code
运行环境	Linux CentOS 7.8
使用框架	LightGBM / Slides Window
语言	Python
Verison	3.8.0

6.2 安装依赖库

依赖库已经写入 requirement.txt 里了，直接运行下面的命令即可：

pip install -r requirement.txt

6.3 运行方式

先通过运行 preprocess.ipynb，得到删除重复行后的数据文件：cleand_trian.csv。不过我们提交的时候也加入了 cleand_train.csv，所以这一步可以省略。

我们一共尝试了五种方法：

1. Popularity model: python kaggle_main --method popularity
2. SVD: python kaggle_main --method SVD
3. Lightfm with default weight matrix: python lightfm_default_weight.py
4. Lightfm with user-defined weight matrix:

   python lightfm_userdefined_weight.py
5. Lightgbm with slide window: python slide_window.py

7 初步调研和尝试

参考网站：[Kaggle Recommender Systems in Python 101]

对于推荐问题，最经典的模型有：Popularity Model，基于内容的模型，协同推荐，混合推荐。

7.1 Popularity Model

运行方式：python kaggle_main --method popularity

根据训练集，总结出最受欢迎的商品，然后“相信群众的智慧”，把最受欢迎的商品推荐给用户。这个模型是一个很好的 baseline。

我们先是对不同的 event_type 定义了权重：

```
strength = {
 'view': 1.0,
 'cart': 2.0,
 'purchase': 4.0,
 'remove_from_cart': -1
 }
```

直观上，purchase 表示用户对这个商品非常喜欢，所以它的权重应该大于 view 和 cart；而如果用户把某个商品移出了购物车，表示他对这个商品不再感兴趣，因此我们把 remove_from_cart 的权重设成了负数。

然后，我们新增了一列来表示每个事件的权重：

```
full_df['strength'] = full_df['event_type'].apply(lambda x: strength[x])
```

为了避免冷启动（如果一个用户的数据过少，就不足以说明这个用户的喜好），我们把数据少于 5 条的用户从训练集中剔除掉：

```
selected_df = full_df.groupby(['user_id', 'product_id'])
selected_df = selected_df.size().groupby('user_id').size()
    selected_df = selected_df[selected_df >=
5].reset_index()[['user_id']]
```

然后对同个用户、和同个商品的访问，我们对权重进行求和：

```
def smooth_user_preference(x):
    return math.log(max(1, 1+x), 2)
sum_df = selected_df.groupby(['user_id',
'product_id'])['strength'].sum()
    sum_df =
sum_df.apply(smooth_user_preference).reset_index()
```

接着，我们对同个商品在不同用户那里的权重进行求和、再排序，就得到了从广受欢迎的商品到不受欢迎的商品的列表：

```
popular_df = sum_df.groupby('product_id')['strength'].sum()
popular_df =
popular_df.sort_values(ascending=False).reset_index()
```

最后，我们从用户没有访问过的商品中采样 100 个商品、加上用户访问过的商品，选出其中最受欢迎的商品，推荐给用户：

```
recs_df = popular_df
interacted = get_interacted(user_id)
non_interacted = get_not_interacted_sample(user_id)
# combining the interacted items with the 100 random items
filter = non_interacted.union(interacted)
# filtering only recommendations that are either the
interacted item or from a random sample of 100 non-interacted
items
recs_df =
recs_df[recs_df['product_id'].isin(filter)]
recs_df = recs_df['product_id'].values
pred = recs_df[0]
```

（以上代码为了展示，做了一定的简化，但大致思想和我们的源代码是一样的）

意料之中的，这个模型效果非常的差，只有 0.0054 分：

因为这种推荐方式只基于数据全局的规律，没有考虑到单个用户自己的偏好。

7.2 基于内容的模型

学习商品的特征，根据用户访问过的商品的特征，推荐相似的商品。

由于在 Kaggle 的教程里，这种方法不如接下来要介绍的协同推荐，所以我们没有采用。

7.3 协同推荐

运行方式：python kaggle_main --method SVD

可以基于用户（又叫近邻协同），也可以基于物品。基于用户的推荐流程大致如下：

1. 根据用户的行为，对商品打分
2. 比较不同用户的相似度（如根据 Manhattan distance 或 Euler distance）
3. 找到和用户 i 最相似的一些用户（相似度大于某个 threshold）
4. 把相似用户的喜好推荐给用户 i

基于物品的类似，也是比较物品的相似度，找到和物品相似的物品，然后进行推荐。

Kaggle 的教程中用 SVD 分解来实现。

先根据用户和商品的打分，构造一个稀疏矩阵（因为训练集里，一个用户并没有访问每个商品），每一行表示一个用户，每一列表示一个商品，第 i 行第 j 列就表示用户 i 对商品 j 的打分。

```
pivot_df = selected_df.pivot(
    index='user_id',
     columns='product_id',
     values='strength').fillna(0)
pivot_matrix = pivot_df.values
```

```
pivot_sparse_matrix = csr_matrix(pivot_matrix)
```

然后，对这个稀疏矩阵进行 SVD 分解，并正则化：

```
NUMBER_OF_FACTORS_MF = 15
U, sigma, Vt = svds(pivot_sparse_matrix,
k=NUMBER_OF_FACTORS_MF)
sigma = np.diag(sigma)
dot1 = np.dot(U, sigma)
user_ratings = np.dot(dot1, Vt)
user_ratings = (user_ratings-user_ratings.min()) \
            / (user_ratings.max()-user_ratings.min())
```

对 user_ratings 排序，推荐排名第一的商品给用户：

```
# convert the reconstructed matrix back to a Pandas dataframe
cf_preds_df = pd.DataFrame(user_ratings,
columns=pivot_df.columns).transpose()
# Get and sort the user's predictions
sorted = cf_preds_df[user_id].sort_values(ascending=False)
sorted = sorted.reset_index().rename(columns={user_id:
'recStrength'})
# Recommend the highest predicted rating movies that the user
hasn't seen yet.
recs_df = sorted.sort_values('recStrength', ascending=False)
rec_values = recs_df['product_id'].values
    pred = rec_values[0]
```

SVD 这种方法的打分是 0.0179 分：

比 popularity model 好了很多，但分还是很低。

7.4 混合推荐

运行方式：python lightfm_default_weight.py

或 python lightfm_userdefined_weight.py

把协同推荐和基于内容的推荐结合起来，常见的混合方法有：加权混合、切换混合、分区混合、分层混合。

我们用了一个可以混合推荐的开源框架，叫做 lightfm

调用 lightfm 的 Dataset()，我们可以得到对用户、商品的映射下标（因为最终得到的推荐矩阵是用映射后的下标排序的）：

```
from lightfm.data import Dataset
# Create user, item and feature mappings: (user id map, user
feature map, item id map, item feature map)
dataset = Dataset() # helper function
dataset.fit(train_users, # creates mappings between userIDs
and row indices for LightFM
                train_items)
# We want the user and item mappings (we'll use feature
mappings later on)
user_mappings = dataset.mapping()[0]
item_mappings = dataset.mapping()[2]
```

官网上的教程是，调用 lightfm 的库直接得到对商品的打分（lightfm_default_weight.py）：

```
# Create an interactions matrix for each user, item and the
weight
train_interactions, train_weights =
dataset.build_interactions(train[['user_id', 'product_id',
'weight']].values)
```

但是这种方法并不好（我们提交结果得到的打分是 0 分）：

所以我们还是采用了在 popularity model 时使用的手动构造打分策略（lightfm_userdefined_weight.py，和前面的一样，就不赘述了）。

然后调用 lightfm 的模型进行预测：

```
model = LightFM(no_components=10,  # the dimensionality of the
feature latent embeddings
    learning_schedule='adagrad', # type of optimiser to use
    loss='warp', # loss type
    learning_rate=0.05) # set the initial learning rate
```

```
model.fit(train_interactions, # our training data
              epochs = 20,
              verbose=True)
# Create all user and item matrix to get predictions for it
n_users, n_items = train_interactions.shape
# Force lightFM to create predictions for all users and all
items
scoring_user_ids = np.concatenate([np.full((n_items, ), i) for
i in range(n_users)]) # repeat user ID for number of prods
scoring_item_ids = np.concatenate([np.arange(n_items) for i in
range(n_users)]) # repeat entire range of item IDs x number of
user
scores = model.predict(user_ids = scoring_user_ids,
        item_ids = scoring_item_ids)
scores = scores.reshape(-1, n_items) # get 1 row per user
recommendations = pd.DataFrame(scores)
```

最后根据用户和商品映射的下标，得到预测的商品 id：

```
# Find the maximum value in each row
max_scores = recommendations.max(axis=1)
max_column = recommendations.idxmax(axis=1)
user_ids = test['user_id'].unique()
preds = []
for uid in user_ids:
    user_mapped_id = user_mappings[uid]
    top_mapped_id = max_column[user_mapped_id]
    top_item_id = inv_item_mappings[top_mapped_id]
    preds.append(top_item_id)
```

不过最后的得分很低，才 0.00358，甚至不如 popularity model：

最后我们经过调研，使用了 lightgbm 库，详见第 7 节。

8 模型设计

为解决电商用户购买行为预测的挑战，我们选择了 lightgbm 库，并采用了一

种创新的滑动窗口方法来建立模型。初始时，我们将窗口大小设置为 1，即使用上一个事件的各类特征进行决策树分类，以商品编号作为分类标签。这个滑动窗口的设计旨在充分利用时间序列数据，捕捉用户行为的动态变化，从而提高模型的预测性能。

在优化方面，我们采取了两个关键步骤。首先，我们尝试去除时间点相近的重复数据，以减少冗余信息的干扰，从而更准确地反映用户行为。其次，我们进行了滑动窗口大小的扩大，旨在更全面地考虑历史特征，使模型更具普适性。

这一模型设计的方法使我们能够更好地理解用户购买行为的演变过程，并在预测下一个可能购买的商品时提供更准确的结果。通过灵活运用时间序列信息和优化策略，我们期待该模型在电商领域的实际应用中表现出色。

9 实验结果分析

运行方式：python slide_window.py

由于我们的代码中采用了滑动窗口技术，通过构建时间窗口内的交互序列，为机器学习模型提供训练数据。同时，LightGBM 分类器则被用来进行预测，以便确定用户在未来时间步中可能会选择的产品。因此我们主要从这两个方面来分析实验的结果。

9.1 去重数据集、滑动窗口为 1

我们首先去重了数据集，并将滑动窗口置为 1。得到的结果为 0.125 左右。

```
# slide_len 窗口长度
slide_len = 5
for uid in tqdm(user_ids):
    user_data = raw_train[raw_train['user_id'] == uid].copy(deep=True)
    # if user_data.shape[0] < 2:
    if user_data.shape[0] < slide_len + 1 or user_data.shape[0] > 300:    # 0.12544803
        # 小于两条或者大于300条的，直接忽略
        continue
```

我们发现如果采用对用户信息不足的数据进行截取的策略，模型的结果会由

0.125 提升到 0.127 左右。

9.2 原数据集(未去重）滑动窗口为 5

相对于后续模型来说，该模型分数相对较低，这个结果可能因为在数据未经过去重处理的情况下得到的，因此我们尝试对数据集进行去重。

9.3 去重数据集，滑动窗口为 5

添加去重数据集行为，观察模型效果为 0.179。

9.4 去重数据集，滑动窗口为 7

增大滑动窗口大小，观察模型效果为 0.188。

9.5 去重数据集，滑动窗口为 10

最终，我们添加了去重数据集的行为，同时将滑动窗口的大小增大到 10。达到了目前的模型效果 0.205。

10 源代码

见源代码附录文件

11 各模块功能

程序的主要目标是根据用户过去的交互历史来预测用户未来可能与哪个产品进行交互。为了实现这一目标，代码采用了滑动窗口技术，通过构建时间窗口内的交互序列，为机器学习模型提供了训练数据。而 LightGBM 分类器则被用来进行预测，以便确定用户在未来时间步中可能会选择的产品。

在这个推荐系统的背景下，代码的主要任务是利用用户的历史行为模式来预测他们的未来偏好。滑动窗口技术有助于捕捉时间序列性质，以便更好地理解用户行为的演变趋势，从而更准确地进行预测。而 LightGBM 分类器则是一种高效的机器学习算法，可用于处理具有大量特征和大规模数据集的分类问题，适合用于这种任务。通过结合这两种方法，我们希望建立稳健的电商用户购买行为预测模型，预测用户下一个可能会购买的商品。

11.1 函数依赖与数据加载

代码首先导入了必要的 Python 库，如 time、numpy、pandas 和 tqdm。还从 sklearn.preprocessing 和 lightgbm 库中导入特定函数和类。加载了三个 CSV 文件：cleaned_train.csv、test.csv 和 submit_example.csv 到 Pandas DataFrames 中，分别命名为 raw_train、raw_test 和 submit_df。

11.2 数据预处理

我们先是观察到，train.csv 里有很多行都是重复数据。于是我们在 preprocess.ipynb 里，判断前后两行的 product_id 和 event_type 是不是一样的，如果一样的话，就删除重复的行。

然后，我们在分析数据缺失值后对数据进行了一定的处理。

1. raw_train 和 raw_test 中的'category_code'和'brand'列中的缺失值用字符串'<unkown>'填充。

2. 'event_time'列被转换为日期时间格式，并创建一个新的'timestamp'列以存储每个事件的 UNIX 时间戳。
3. 使用 Label Encoding 对分类特征'category_code'和'brand'进行编码。
4. 'event_type'列也使用 Label Encoding 进行编码。
5. 从 raw_train 和 raw_test 中删除列'event_time'、'user_session'和'timestamp'。

11.3 数据排序

首先，将 raw_train 和 raw_test DataFrames 按'user_id'排序，然后按'timestamp'排序。同时，将 raw_train 和 raw_test 连接成一个名为 df 的单一 DataFrame。

11.4 滑动窗口训练数据生成

在程序设计的过程当中，我们选取了滑动窗口技术。通过构建时间窗口内的交互序列，为机器学习模型生成训练数据。

对于 raw_train 中的每个唯一用户，一个长度由 slide_len 定义的窗口在用户的交互历史上滑动。为窗口内的每个时间步创建特征，包括'productX'和'eventX'，其中 X 代表时间步。目标变量'y'则表示用户将在将来的时间步与之交互的产品。

同时，我们排除具有少于 slide_len + 1 交互或超过 300 个交互的用户，将结果的训练数据存储在名为 train_df 的 DataFrame 中，并保存到名为'slide_window.csv'的 CSV 文件中。

11.5 特征和数据类型调整

为了确保 train_df 中的'productX'和'eventX'列的正确数据类型，我们进行了数据结构的调整，从 train_df 中删除了'user_id'列。

11.6 生成测试数据

测试数据通过选择 raw_test 中每个用户的最后一个交互来生成。类似于训练数据，为测试数据创建窗口内每个时间步的特征。

11.7 模型训练和预测

我们为 train_df 中的每个用户基于其来自 train_df 的训练数据训练了一个 LightGBM 分类器。通过使用测试数据的最后一组特征进行预测，并将预测的结果，即预测的产品 ID 附加到'preds'列表中。

11.8 创建提交文件

我们将'preds'中的预测产品 ID 添加到 submit_df DataFrame 的'product_id'列中。结果的 DataFrame 保存为名为'baseline.csv'的 CSV 文件。

12 外部库

LightGBM (Light Gradient Boosting Machine)是一种高效、快速、可扩展的梯度提升框架，特别设计用于解决分类、回归和排名等机器学习任务。它的设计目标是处理大规模数据集和高维特征，以提供卓越的性能和准确性。

常用的机器学习算法，例如神经网络等算法，都可以以 mini-batch 的方式训练，训练数据的大小不会受到内存限制。而 GBDT 在每一次迭代的时候，都需要遍历整个训练数据多次。如果把整个训练数据装进内存则会限制训练数据的大小；如果不装进内存，反复地读写训练数据又会消耗非常大的时间。尤其面对工业级海量的数据，普通的 GBDT 算法是不能满足其需求的。

LightGBM 提出的主要原因就是为了解决 GBDT 在海量数据遇到的问题，让 GBDT 可以更好更快地用于工业实践。

12.1 LightGBM 的基本原理

12.1.1 直方图优化算法

LightGBM 的核心优化之一是其直方图算法。该算法的基本思想是将连续的浮点特征值离散化为整数，并构建一个宽度固定的直方图。在数据遍历过程中，利用离散化后的值作为索引在直方图中累积统计量，从而在一次数据遍历后获得所需的统计量，并在直方图的基础上寻找最优分割点。这种方法的优点包括更小的内存占用（由于只需存储离散化后的较小位数整型值）和更低的计算成本（减少了计算分裂增益的次数）。

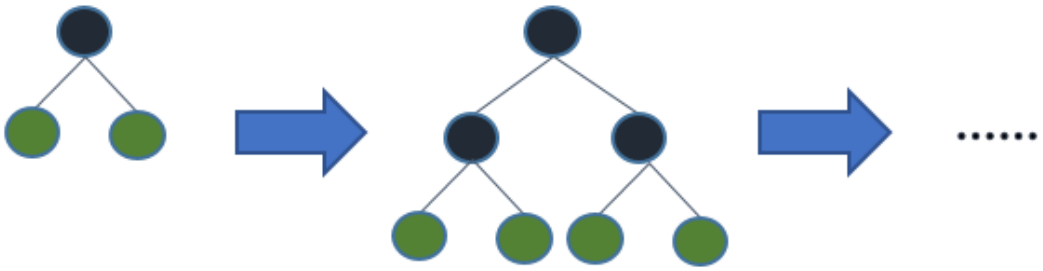
12.1.2 直方图做差加速

LightGBM 还引入了直方图做差加速的方法。在这个优化中，一个子节点的

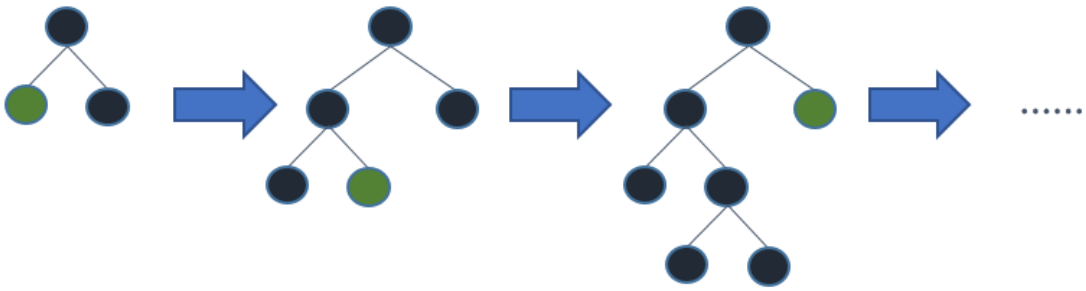
直方图可以通过其父节点的直方图与其兄弟节点的直方图做差得到。这种方法减少了必须遍历的数据量，提升了效率。

12.1.3 带深度限制的 Leaf-wise 算法

与传统的 Level-wise 决策树生长策略不同，LightGBM 采用了带深度限制的 Leaf-wise 算法。



这种策略每次选择当前所有叶子中分裂增益最大的一个进行分裂，从而在同样的分裂次数下降低更多的误差，提高模型精度。为防止过拟合，LightGBM 在 Leaf-wise 策略上增加了最大深度限制。



12.1.4 单边梯度采样（GOSS）

GOSS，即单边梯度采样，是从样本量减少的角度出发的一种优化。它通过保留梯度较大的样本并排除大部分梯度小的样本来计算信息增益，实现了在减少

数据量和保持精度之间的平衡。为了不改变数据分布，GOSS 在较小梯度样本中随机选择一部分，通过调整其权重来平衡样本的总体分布。

Algorithm 2: Gradient-based One-Side Sampling

Input: I: training data, d: iterations
Input: a: sampling ratio of large gradient data
Input: b: sampling ratio of small gradient data
Input: $loss$: loss function, L: weak learner
models ← { }, fact ← $\frac{1-a}{b}$
topN ← a × len(I) , randN ← b × len(I)
for $i = 1$ ***to*** d **do**
  preds ← models.predict(I)
  g ← $loss$(I, preds), w ← {1,1,...}
  sorted ← GetSortedIndices(abs(g))
  topSet ← sorted[1:topN]
  randSet ← RandomPick(sorted[topN:len(I)], randN)
  usedSet ← topSet + randSet
  w[randSet] × = fact ▷ Assign weight $fact$ to the small gradient data.
  newModel ← L(I[usedSet], − g[usedSet], w[usedSet])
  models.append(newModel)

12.1.5 互斥特征捆绑（EFB）

互斥特征捆绑（EFB）是 LightGBM 用于处理高维稀疏数据的另一种优化。通过捆绑互斥（或近似互斥）的特征，EFB 能够在不丢失信息的前提下降低特征的维度。这种方法将构建直方图的时间复杂度从基于特征的数量降低为基于捆绑包的数量，从而提高了效率。

12.2 LGBMClassifier 类的使用

LGBMClassifier 是 LightGBM 库中用于分类任务的主要类之一。它实现了梯度提升树算法。LGBMClassifier 具有许多参数和选项，可用于控制模型的行为和性能。在代码中，我们创建了一个 LGBMClassifier 对象并自定义模型的行为，例如设置树的深度、学习率和多样性策略等。

```
# 训练
clf = lgb.LGBMClassifier(**{'seed': int(time.time())})
```

总的来说，LightGBM 是一种非常强大的机器学习工具，特别适用于大规模数据集和复杂特征的分类和回归问题。它以其高效性、性能和可扩展性而著称，在我们的代码中，LightGBM 用于构建和训练一个梯度提升树分类器。通过与滑动窗口技术的结合，预测以解决产品预测任务，希望建立稳健的电商用户购买行为预测模型，准确地预测用户未来的商品购买行为。

13 参考文献

[1] Ke G, Meng Q, Finley T, et al. Lightgbm: A highly efficient gradient boosting decision tree[J]. Advances in neural information processing systems, 2017, 30.

[2] Chen C, Zhang Q, Ma Q, et al. LightGBM-PPI: Predicting protein-protein interactions through LightGBM with multi-information fusion[J]. Chemometrics and Intelligent Laboratory Systems, 2019, 191: 54-64.

[3] Ju Y, Sun G, Chen Q, et al. A model combining convolutional neural network and LightGBM algorithm for ultra-short-term wind power forecasting[J]. Ieee Access, 2019, 7: 28309-28318.

[4] https://blog.csdn.net/weixin_41929524/article/details/80739112

[5] https://cloud.tencent.com/developer/article/1758058