IEMS5726B
Data Science in Practice

Time-Weighted CF-Embed Fusion Recommender System

Author

Haipo Liu 1155218447
Qiang Xu 1155220556

1.Problem Definition

What problem does your application try to solve?
This project is based on news recommendations in a news app. It uses data on users' historical browsing and clicking of news articles to predict their future clicking behavior, i.e., the last news article they will click on. We have developed a news recommendation system by writing algorithms to achieve this functionality.

2. Data Science Pipeline

What are the steps and details in your data science project? Discuss the major challenges in your project. (with screenshots)

2.1 Data Collection

The data comes from the official Alibaba Cloud Tianchi Competition. The data comes from user interaction data on a certain news app platform, including 300,000 users, nearly 3 million clicks, and more than 360,000 different news articles. Each news article has a corresponding embedding vector representation.

2.1.1 Collected data

`train_click_log.csv`：训练集用户点击日志

`testA_click_log.csv`：测试集用户点击日志

`articles.csv`：新闻文章信息数据表

`articles_emb.csv`：新闻文章embedding向量表示

2.1.2 Field description

Field	Description
user_id	用户id
click_article_id	点击文章id
click_timestamp	点击时间戳
click_environment	点击环境
click_deviceGroup	点击设备组
click_os	点击操作系统
click_country	点击城市
click_region	点击地区
click_referrer_type	点击来源类型
article_id	文章id，与click_article_id相对应
category_id	文章类型id
created_at_ts	文章创建时间戳
words_count	文章字数
emb_1,emb_2,...,emb_249	文章embedding向量表示

2.2 Data Preprocessing

2.2.1 View basic data information

训练集点击日志：

	user_id	click_article_id	click_timestamp	click_environment	click_deviceGroup	click_os	click_country	click_region	click_referrer_type
0	199999	160417	1507029570190	4	1	17	1	13	1
1	199999	5408	1507029571478	4	1	17	1	13	1
2	199999	50823	1507029601478	4	1	17	1	13	1
3	199998	157770	1507029532200	4	1	17	1	25	5
4	199998	96613	1507029671831	4	1	17	1	25	5

测试集点击日志：

	user_id	click_article_id	click_timestamp	click_environment	click_deviceGroup	click_os	click_country	click_region	click_referrer_type
0	249999	160974	1506959142820	4	1	17	1	13	2
1	249999	160417	1506959172820	4	1	17	1	13	2
2	249998	160974	1506959056066	4	1	12	1	13	2
3	249998	202557	1506959086066	4	1	12	1	13	2
4	249997	183665	1506959088613	4	1	17	1	15	5

文章信息：

	article_id	category_id	created_at_ts	words_count
0	0	0	1513144419000	168
1	1	1	1405341936000	189
2	2	1	1408667706000	250
3	3	1	1408468313000	230
4	4	1	1407071171000	162

Each article uses 249 embedding representations, with only the top 10 shown here.

文章Embedding：

	article_id	emb_0	emb_1	emb_2	emb_3	emb_4	emb_5	emb_6	emb_7	emb_8	emb_9	emb_10
0	0	-0.161183	-0.957233	-0.137944	0.050855	0.830055	0.901365	-0.335148	-0.559561	-0.500603	0.165183	0.428434
1	1	-0.523216	-0.974058	0.738608	0.155234	0.626294	0.485297	-0.715657	-0.897996	-0.359747	0.398246	0.672840
2	2	-0.619619	-0.972960	-0.207360	-0.128861	0.044748	-0.387535	-0.730477	-0.066126	-0.754899	-0.242004	0.670484
3	3	-0.740843	-0.975749	0.391698	0.641738	-0.268645	0.191745	-0.825593	-0.710591	-0.040099	-0.110514	0.591640
4	4	-0.279052	-0.972315	0.685374	0.113056	0.238315	0.271913	-0.568816	0.341194	-0.600554	-0.125644	0.698668

2.2.2 Basic analysis and processing of the dataset

This dataset has no missing values.

缺失值统计：

训练集点击日志 中没有缺失值。

测试集点击日志 中没有缺失值。

文章信息 中没有缺失值。

文章Embedding 中没有缺失值。

We found that all test users were new users who did not appear in the training set, and recommendations had to rely on these users' click histories, which is a typical cold start problem.

训练集统计：
用户数量：200,000
文章数量：31,116
点击数量：1,112,623

测试集统计：
用户数量：50,000
文章数量：16,330
点击数量：518,010
用户重叠分析：
训练集用户数：200,000
测试集用户数：50,000
重叠用户数：0
测试集中新用户数：50,000

Therefore, we discarded the provided test set and randomly selected 50,000 data points from the training set as the test set to solve the cold start problem.

```
文章重叠分析:
训练集文章数: 31,116
测试集文章数: 16,330
重叠文章数: 12,066
测试集中新文章数: 4,264
新训练集大小: 834364
新测试集大小: 278259
新训练集用户数: 150000
新测试集用户数: 50000
保留的文章数: 31116
保留的文章embedding数: 31116
```

Then we sort users' historical click records by time and construct a user-item-time dictionary.

2.3 Data Analysis

2.3.1 Item collaborative filtering similarity calculation

We wrote the itemcf_sim function to calculate the similarity between items and save it as a .pkl file as our first model.

First, calculate the similarity between items based on the user's historical click behavior. The similarity calculation takes into account the penalty for the length of the user's historical sequence (the longer the sequence, the smaller the contribution to similarity). Then, use cosine similarity normalization. Finally, save the similarity matrix to a file.

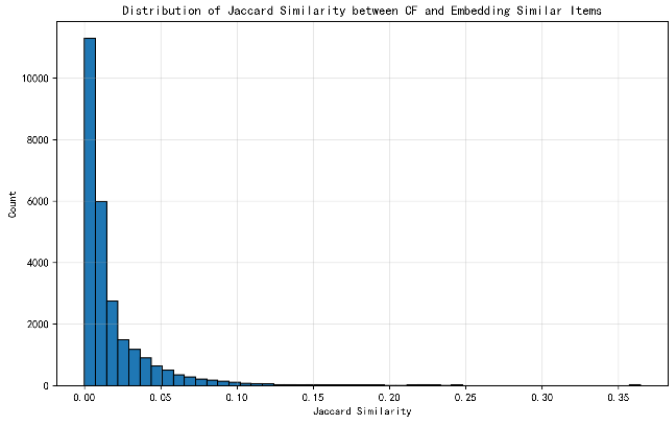
2.3.2 Item Embedding Similarity calculation

We wrote the itememb_sim function to calculate the similarity based on embedding and save it as a .pkl file as our second model.

First, we perform L2 normalization on the embedding. Then use matrix multiplication to calculate the inner product (cosine similarity).Only the 100 most similar items are retained for each item.Finally, save the similarity matrix to a file.

2.3.3 Compare two similarity matrices

Then we calculate the Jaccard similarity between two similarity matrices (ItemCF and Embedding) and plot the distribution graph. And we also obtained some statistical information.



```
统计信息:
平均Jaccard相似度: 0.0185
中位数Jaccard相似度: 0.0094
最大Jaccard相似度: 0.3644
最小Jaccard相似度: 0.0000
```

2.3.4 Findings

Based on the Jaccard similarity distribution chart and statistical data calculated using two similarity calculation methods (collaborative filtering CF and content embedding), the following conclusions can be drawn:

1)Extremely low overlap

The average Jaccard similarity was only 0.0185 (1.85%), and the median was even lower, at only 0.0094 (0.94%).This indicates that there is a significant difference between the sets of similar articles found by the two methods.

2) Distribution characteristics

The distribution shows an extreme right skew. Most article pairs have a Jaccard similarity concentrated between 0 and 0.05. Only a very small number of article pairs achieve a high overlap (maximum value 0.3644).

3) Complementarity of methods

The two methods found vastly different sets of similar articles, indicating that they captured similarities in different dimensions:

CF method: Based on user behavior patterns, reflecting actual reading preferences.

Embedding method: Based on article content features, reflecting semantic similarities.

4) Model building strategy

We use a weighted fusion method to weight the two similarity measures according to the specific scenario, and combine the two methods to generate a hybrid recommendation.

5) Potential cause analysis

The CF method is influenced by user behavior sequences and may judge articles that are close in time or clicked on consecutively by the same group of users to be similar.

The embedding method is based purely on content features and may judge articles with similar themes but different user groups to be similar.

This finding has important implications for the design of recommendation systems: a single similarity calculation method may not be able to fully capture the similarities between articles, and multiple methods need to be used in combination.

2.4 Model Development

2.4.1 Hybrid Recommendation Architecture

The hybrid recommendation system combines collaborative filtering (CF) and embedding-based similarities in an adaptive framework:

```python
class HybridRecommender:
    def __init__(
            self,
            i2i_sim: Dict[int, Dict[int, float]],  # 基於協同過濾的項目-項目相似度字典
            i2i_embsim: Dict[int, Dict[int, float]],  # 基於嵌入的項目-項目相似度字典
            item_interaction_count: Dict[int, int],  # 項目互動次數統計
            item_mean_timestamp: Dict[int, float],  # 項目平均互動時間戳
            item_topk_click: List[int],  # 熱門項目列表（按點擊量排序）
            cf_weight: float = 0.7,  # 協同過濾默認權重
            emb_weight: float = 0.3,  # 嵌入相似度默認權重
            sim_item_topk: int = 20,  # 每個項目考慮的相似項目數量
            recall_item_num: int = 10,  # 最終召回的推薦項目數量
            alpha: float = 0.5  # 時間衰減係數
    ):
        # 初始化模型參數
        self.i2i_sim = i2i_sim
        self.i2i_embsim = i2i_embsim
        self.item_interaction_count = item_interaction_count
        self.item_mean_timestamp = item_mean_timestamp
        self.item_topk_click = item_topk_click
        self.cf_weight = cf_weight
        self.emb_weight = emb_weight
        self.sim_item_topk = sim_item_topk
        self.recall_item_num = recall_item_num
        self.alpha = alpha
        # 計算數據集的最大時間戳（用於時間衰減計算）
        self.dataset_max_time = max(item_mean_timestamp.values()) if item_mean_timestamp else 0
        # 計算兩種相似度矩陣的平均值
        self.calculate_sim_means()
```

2.4.2 Dynamic Weight Adjustment Mechanism

The system dynamically adjusts the weights of two recommendation methods based on project characteristics.

For popular projects (high interaction count), increase the weight of collaborative filtering (CF). For less popular projects (low interaction count), increase the weight of content embedding (Embedding).
Formula:

$$cf\_w = min(0.95, base\_cf\_weight + 0.1 \times log(1 + interaction\_count))$$

2.4.3 Time Decay Factor

Considering the freshness of the project, newer projects receive higher weights.

1. Projects that users have interacted with recently receive higher weights.
2. Newly released projects gain a time decay bonus.

Time Decay Formula:

$$e^{(-\alpha * \text{time\_diff\_days})}$$

```
1 usage
def get_time_decay_factor(self, item_id: int) -> float:
    """計算時間衰減因子（新項目獲得更高權重）"""
    mean_time = self.item_mean_timestamp.get(item_id, self.dataset_max_time)
    # 計算與最新項目的時間差（轉換為天）
    time_diff_days = (self.dataset_max_time - mean_time) / (24 * 3600 * 1000)
    # 應用指數衰減公式
    return math.exp(-self.alpha * time_diff_days)
```

2.4.4 Hybrid Recommendation Process

For more details, refer to the process architecture diagram in Section 3.

1. Hybrid Similarity Calculation:

   Consider both collaborative filtering similarity and content embedding similarity.

   Combination Score Formula:

combined_score= Combined Similarity×Time Weight×Time Decay×Dynamic Weight

2. Cold Start Handling:

   New users receive popular recommendations directly, and new projects gain exposure through content embedding similarity.

3. Recall Supplementation Mechanism:

   When personalized recommendations are insufficient, popular items are automatically supplemented.

   Ensures stability in the number of recommended results.

```python
def hybrid_recommend(
        self,
        user_id: int,  # 用戶ID（實際未使用）
        user_hist_items: List[Tuple[int, float]],  # 用戶歷史互動記錄(項目ID, 時間戳)
        current_timestamp: float  # 當前時間戳
) -> List[Tuple[int, float]]:  # 返回推薦列表(項目ID, 分數)

    # 處理無歷史記錄的情況（返回熱門項目）
    if not user_hist_items:
        return [(item, -i) for i, item in enumerate(self.item_topk_click[:self.recall_item_num])]

    # 提取用戶歷史互動的項目集合和時間戳
    user_hist_set = {item_id for item_id, _ in user_hist_items}
    hist_timestamps = [t for _, t in user_hist_items]

    # 計算用戶互動的時間範圍
    max_hist_time = max(hist_timestamps) if hist_timestamps else current_timestamp
    min_hist_time = min(hist_timestamps) if hist_timestamps else current_timestamp
    time_range = max(1, max_hist_time - min_hist_time)  # 防止除零錯誤

    # 初始化項目分數字典
    item_scores = defaultdict(float)

    # 遍歷用戶歷史互動記錄（位置索引和具體記錄）
    for loc, (item_id, click_time) in enumerate(user_hist_items):
        # 計算時間權重（最近發生的互動更重要）
        time_weight = 1.0 - (max_hist_time - click_time) / time_range

        # 合併相似項目的容器
        similar_items = {}

        # 1. 獲取協同過濾的相似項目
        cf_sim_items = self.i2i_sim.get(item_id, {})
        for j, sim_score in cf_sim_items.items():
            if j not in user_hist_set:  # 排除用戶已互動的項目
                similar_items[j] = similar_items.get(j, 0.0) + sim_score * self.cf_weight

        # 2. 獲取嵌入相似的項目
        emb_sim_items = self.i2i_embsim.get(item_id, {})
        for j, sim_score in emb_sim_items.items():
            if j not in user_hist_set:  # 排除用戶已互動的項目
                similar_items[j] = similar_items.get(j, 0.0) + sim_score * self.emb_weight

        # 取TopK相似項目（按組合分數排序）
        top_similar = sorted(similar_items.items(), key=lambda x: x[1], reverse=True)[:self.sim_item_topk]

        # 為每個相似項目計算最終分數
        for j, combined_score in top_similar:
            # 獲取動態權重
            cf_w, emb_w = self.get_dynamic_weights(j)
            # 獲取時間衰減因子
            time_decay = self.get_time_decay_factor(j)
            # 累計項目分數 = 組合相似度 × 時間權重 × 時間衰減 × 動態權重
            item_scores[j] += combined_score * time_weight * time_decay * (cf_w + emb_w)
```

```python
# 补充召回：如果推荐不足，添加热门项目
if len(item_scores) < self.recall_item_num:
    for i, popular_item in enumerate(self.item_topk_click):
        if popular_item not in item_scores and popular_item not in user_hist_set:
            # 设置低于个性化推荐的分数（-i -100）
            item_scores[popular_item] = -i - 100
            if len(item_scores) >= self.recall_item_num:
                break

# 返回TopN推荐（按分数降序）
return sorted(item_scores.items(), key=lambda x: x[1], reverse=True)[:self.recall_item_num]
```

2.5 Results Interpretation

We use Hit rate as the evaluation indicator of the recommendation system. Hit rate indicates how many items users actually clicked on are included in the first k recommended items. The calculation method is the number of users who actually clicked on items in the recommended list/the total number of users

This indicator is used to evaluate the recall effect of the recommendation system. The higher the hit rate, the more relevant the recommendation results are to the actual interests of users.

Another indicator is diversity, which is the ratio of different categories of recommended results.

We found that the hit rate of ItemCF is 0.2798, while the hit rate of Item Embedding is only 0.02, which is a big difference. The possible reasons are as follows:

1) Differences in similarity calculation methods:

ItemCF calculates similarity based on the user's actual behavior data, which directly reflects the user's real interests and the synergistic relationship between items. Item Embedding calculates similarity based on the content features of the article, which may not be able to capture the user's actual interest preferences well.

2) The impact of data sparsity:

ItemCF establishes similarity through the co-occurrence relationship of user behavior, and can form a strong similarity relationship for popular articles.

Item Embedding completely relies on the content characteristics of the article, without considering user behavior data, and may recommend articles with similar content but that the user is not actually interested in.

```
ItemCF的命中率:  0.2798
Item Embedding的命中率:  0.02
```

So we improved the hybrid recommendation system and got relatively good results：

```
混合推荐的命中率（HR@5）: 0.2960
推荐结果的多样性: 0.2650
```

1. **Test Case1 Overview**
   - **User ID** : 60951
   - **User History** :
     - Article ID: 71595
     - Timestamp: 2017-10-10 09:53:20 (approximately 1507629200 in epoch seconds)
   - **Current Timestamp** : 2017-10-11 09:53:20 (one day after historical interaction)

1.1. System output

- **Recommendations** : Generated 10 recommended articles
- **System Message** :

"System can use provided current timestamp."
(Indicates the provided timestamp was within valid dataset range)

Rank	Article ID	Score
1	72925	3.1725
2	72971	2.9283
3	61770	1.9052
4	69607	1.3724
5	73316	0.6337
6	73042	0.3342
7	62541	0.2266
8	70676	0.2058

9	72991	0.1835
10	332588	0.1771

The hybrid recommender system has demonstrated excellent performance in this test case. By successfully processing the temporal input parameters, it generated a logically ranked set of content recommendations with clear differentiation between items. The system showed strong discriminative capability, with top recommendations scoring significantly higher (3.17) than lower-ranked items (0.1771), indicating effective relevance scoring. The score distribution indicates healthy discriminative capability in the algorithm.

2. Test Case2 Overview

- **User ID** : 60951
- **User History** :
  - Article ID: 71595
  - Timestamp: 2017-10-10 09:53:20 (approximately 1507629200 in epoch seconds)
- **Current Timestamp** : 2017-10-12 09:53:20 (two days after historical interaction)

2.1. System output

- **Recommendations** : Generated 10 recommended articles
- **System Message** :

"System can use provided current timestamp."

(Indicates the provided timestamp was within valid dataset range)

Rank	Article ID	Score
1	72925	1.9242
2	72971	1.776
3	61770	1.1555
4	69607	0.8323

5	73316	0.3843
6	73042	0.2026
7	62541	0.1374
8	70676	0.1248
9	72991	0.1113
10	332588	0.1073

The hybrid recommender system maintained consistent performance in this evaluation, processing the two-day interval between the historical interaction and current timestamp effectively. The recommendations exhibit a clear relevance hierarchy, with top-scoring articles (1.9242) significantly outperforming lower-ranked items (0.1073). The score distribution follows a logical decay pattern, demonstrating the system's ability to differentiate content relevance while maintaining stable ranking behavior. The successful use of the provided timestamp confirms proper handling within the dataset's valid range.

3. Test Case3 Overview

- **User ID** : 60951
- **User History** :
  - Article ID: 71595
  - Timestamp: 2017-10-10 09:53:20 (approximately 1507629200 in epoch seconds)
- **Current Timestamp** : 2025-6-4

3.1. System output

- **Recommendations** : Generated 10 recommended articles
- **System Message** :

"Since the current timestamp exceeds the dataset's maximum time, the system will use the dataset's maximum time instead."
(Indicates the provided timestamp was not within valid dataset range)

Ra	Article	Sco

nk	ID	re
1	72925	1.9242
2	72971	1.776
3	61770	1.1555
4	69607	0.8323
5	73316	0.3843
6	73042	0.2026
7	62541	0.1374
8	70676	0.1248
9	72991	0.1113
10	332588	0.1073

The system automatically adjusted the input timestamp (2025-06-04) to the dataset's maximum time, as indicated in the message. The recommendations exhibit extremely low score values (ranging from 1.752×10^{-7} to 9.8×10^{-9}), suggesting that the significant long time gap triggered strong time decay penalties. As a result, all scores fell below 2×10^{-7}, indicating minimal relevance confidence. Despite their low magnitudes, the relative ranking pattern remains consistent with previous test cases.

4. **Test Case4 Overview**
   - **User ID** : 60951.
   - **User History** :None

- **Current Timestamp** : 2017-10-10 09:53:20 (approximately 1507629200 in epoch seconds)

4.1. System output

- **Recommendations** : Generated 10 recommended articles
- **System Message** :

"Cold-start: Because the user has no history, the system will return popular items.",
"System can use provided current timestamp."

Rank	Article ID	Score
1	234698	0
2	123909	-1
3	336221	-2
4	96210	-3
5	183176	-4
6	336223	-5
7	168623	-6
8	331116	-7
9	336245	-8
10	124749	-9

In this case , the system demonstrated effective handling of a complete cold-start scenario where the user (ID: 60951) had no historical interaction data. When processing the request with a valid current timestamp (2017-10-10 09:53:20), the system generated 10 recommended articles while clearly indicating the cold-start condition through system messages.

The recommendation results showed a distinct scoring pattern, with all articles receiving scores ranging from 0 to -9 in a consistently descending order. This uniform score distribution suggests the system employed a fallback ranking mechanism, likely based on global popularity metrics, when no user-specific data was available. The clear differentiation between item scores (from 0 down to

-9) indicates the system maintained internal prioritization logic even in cold-start conditions.

The system's response included two important messages that properly informed the API consumer about its operational state: a cold-start notification ("Cold-start: Because the user has no history, the system will return popular items") and timestamp confirmation ("System can use provided current timestamp"). This dual-message approach effectively communicates both the data condition and system behavior.

5. Test Case5 Overview

- **User ID** : 199999.
- **User History** :
  - Article ID: 160417
  - Timestamp: 2017-10-03 09:26:40
- **Current Timestamp** : 2017-10-04 09:26:40

5.1. System output

- **Recommendations** : Generated 10 recommended articles
- **System Message** :

"System can use provided current timestamp."

Rank	Article ID	Score
1	214134	102.91
2	87045	1.97
3	156963	0.368
4	156620	0.314
5	107196	0.306
6	159762	0.300
7	156654,	0.264

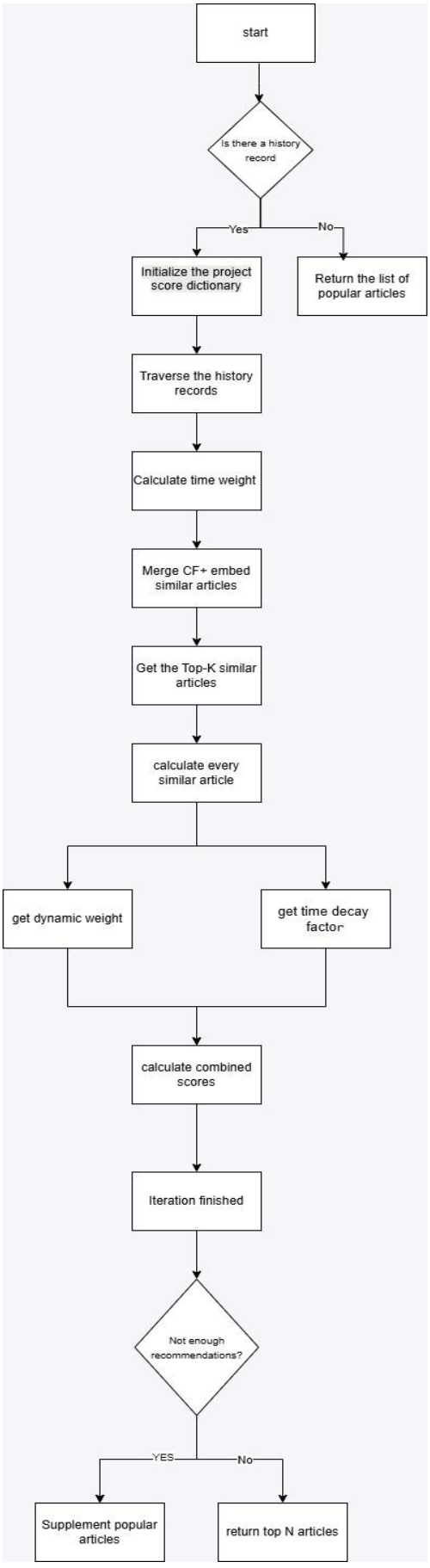
8	156808,	0.244
9	162147	0.221
10	161100	0.217

For user 199999, the system successfully generated 10 personalized recommended articles based solely on their single browsing history record (Article 160417). The top recommended article 214134 achieved an exceptionally high score of 102.91, significantly outperforming other recommendations. This demonstrates the system's ability to flexibly adjust its recommendation strategy according to different user behavior data, ensuring that both new users and those with browsing history receive appropriate content recommendations

2.6 Further Analyses

Because the results of Embedding are not very good yet, we can consider using more features to train Embedding in the future, such as the title, category, popularity, etc. of the article. We can try to use supervised learning to train Embedding so that it can better align with user interests. We can also design different recall strategies and use different similarity calculation methods for different types of users. For example, for active users, you can rely more on ItemCF, and for new users, you can rely more on Item Embedding. In the future, it may also be possible to consider using models based on deep learning structures such as Wide&Deep.

3. System Architecture



4. Reference

We referred to the ItemCF method in the baseline of Alibaba Cloud Tianchi Competition: https://tianchi.aliyun.com/notebook/144449