基于 Q-learning 的论文推荐算法设计

摘 要

近年来，随着信息技术的飞速发展，互联网中的论文发表数目呈爆炸式增长。为了进行科学研究，学者们必须阅读他人的论文，从中吸取经验，并进一步发展。面对的众多的论文，如何有效的进行论文推荐也成为当前的研究热点。以往的论文推荐算法主要采用基于内容的方法、基于用户兴趣、协同过滤算法，以及在这些算法之上衍生出的各种混合算法。然而，这些算法存在着“数据稀疏”、“冷启动”、“忽视兴趣动态”等问题。受启发于电商平台的强化学习推荐算法，本文设计了一种基于强化学习中 Q-Learning 方法的论文推荐算法来以解决上述问题。该论文推荐算法可以根据用户对论文的点击，动态更新推荐权重，来实现更好的推荐效果。本文利用 CiteULike 数据集进行实验，对比了该系统与传统的三种论文推荐算法的推荐效果。结果表明，本文论文推荐算法效果要好于传统的论文推荐算法。

关键字：推荐算法、强化学习、Q-Learning、协同过滤、矩阵分解

Design of Paper Recommendation Algorithm

Based on Q - learning

Abstract

In recent years, with the rapid development of information technology, the number of papers published on the Internet has exploded. In order to conduct scientific research, scholars must learn from others by reading other people's papers and further develop them. Faced with many papers, how to effectively recommend papers has become a research hotspot. Previous paper recommendation algorithms mainly use content-based methods, user-based interests, collaborative filtering algorithms, and various hybrid algorithms derived from these algorithms. However, these algorithms have problems such as “data sparseness”, “cold start”, and “ignoring interest dynamics”. Inspired by the reinforcement learning recommendation algorithm of e-commerce platform, this paper designs a paper recommendation algorithm based on Q-Learning method in reinforcement learning to solve the above problems. The paper recommendation algorithm can dynamically update the recommendation weight according to the user's click on the paper to achieve a better recommendation effect. This paper uses the CiteULike dataset to conduct experiments and compares the recommended effects of the system with the traditional three paper recommendation algorithms. The results show that the proposed algorithm is better than the traditional paper recommendation algorithm.

Key words: recommendation algorithm, reinforcement learning, Q - Learning, collaborative filtering, matrix decomposition

目录

一、序言

（一）论文推荐算法的发展现状

当学者想要进行学术研究时，总会参考前人的相关文献，来帮助自己理解研究领域的背景和内容，同时启发自己的创新。然而，经研究显示，在信息爆炸的今天，网络上的学术论文的数量以 6%-8%的速率持续增长。随着越来越多的学术论文发表在互联网上，学者想要完全读完和理解这些相关学术论文是不太现实的。

最初，人们采用搜索引擎协助自己完成论文搜索，比如国内的百度、搜狗，或者国外的Google、Bing。这些搜索引擎可以通过关键字匹配完成论文匹配，在一定程度上满足了用户的需求。但由于这些搜索引擎是通用搜索引擎，搜索结果混杂着众多无关论文的信息。同时这些搜索引擎对于模糊的论文搜索结果不佳，所以难以帮助用户寻找感兴趣的新论文。

为了提供更良好的搜索结果，人们设计了专业的论文搜索引擎，帮助学者在海量的论文库中检索论文，比如国内的知网、万方数据库和国外的 CiteULike、dblp。相比于通用的搜索引擎，它们更加专业，提供的信息更加精准。然而，同通用搜索引擎一样，论文搜索引擎难以帮助用户找到感兴趣的新论文。当用户想要利用一个关键词搜索感兴趣的新论文时，返回的众多论文还是令读者难以挑选。

为了解决这个问题，人们想到了将各种推荐算法运用到论文推荐中。

最早的也是使用最多的推荐算法是基于内容的推荐算法(Content-based)。这种推荐算法的特点是关注于论文本身特点，可以寻找到内容上与用户过去感兴趣的论文的相似论文,然后推荐给用户(Lops et al., 2011)。实际上，现在的基于内容的推荐算法也开始考虑结合用户个人信息，论文发表时间等特征，对原有的单纯计算文本相似度的方法做出了改进。 目前，中国知网中的推荐功能使用的就是基于内容的推荐算法。

最近几年，新的论文推荐普遍使用的推荐算法是协同过滤算法（Collaborative filtering）。这种推荐算法不考虑论文本身的内容，而是将用户和论文关联起来。协同过滤算法又可以分为基于用户的推荐(User-based)、基于项目的推荐(Item-based)、还有基于模型的推荐（Model-based）。基于用户的推荐是向用户推荐与此用户兴趣相似的用户喜欢的内容。基于项目的则是向用户推荐与他历史行为相似的物品。 现在，结合机器学习的方法，人们创建了基于模型的协同过滤算法，可以预估用户对新论文的评分（黄立威 et al., 2018）。国外论文搜索网站 CiteULike 的推荐功能采用的正是协同过滤算法。

还有一种比较流行的论文推荐算法是基于图的（Graph-based）。这种论文推荐算法利用用户-论文矩阵建模为一个二部图(bipartite graph)。用节点来表示用户和论文，节点间的边表示用户对论文的评价或用户喜欢该论文。之后，基于图结构，通过分析计算后给出合理的推荐(谢玮 et al., 2016)。

一些论文推荐采用的算法是由以上几种推荐算法组合起来的，称为混合推荐算法。主要有四种组合方法(杨博&赵鹏飞., 2011)：

（1）多种方法单独进行，最后将结果融合；

（2）将基于内容的特征融合到协同过滤算法中；

（3）将协同过滤分离出的特征向量融合到基于内容的算法；

（4）将多种算法混合到新的框架中。

还有一些不常见的推荐算法，比如基于知识脉络的论文推荐算法。其核心是利用论文文本中关键词之间的同义关系、上下位关系构建成知识脉络(谭红叶 et al., 2016)。

根据 Joeran Beel(2015)等学者统计，在过去的 16 年间，有超过 200 篇论文是关于论文推荐算法的。其中超过一半的论文推荐算法采用了基于内容的算法（55%），采用协同过滤算法的有 18%，基于图论的算法约占 16%，其余的推荐算法占 11%。

（二）论文推荐算法比较

1、基于内容的推荐算法

基于内容的推荐算法的优势是(刘建国 et al.,2009)：

（1）具有用户独立性，针对每个用户单独构建用户的特征，不需要考虑其他用户的因素。

（2）算法透明度高，对推荐结果有直观的解释，便于分析推荐算法的过程。

（3）接受新论文更快，不需要其余用户对新论文评价，可以直接根据内容推荐。

然而，基于内容的推荐算法也有一些不足：

（1）特征提取受限制，难以充分提取论文的特征，因为处理的主要是非结构化文本数据。

（2）推荐单一总是受限于推荐与过去论文相似的论文，难以发现新的有趣的文章。

（3）对于新用户，因为数据不足，推荐准确率低(这种常被称为冷启动问题)。

2、协同过滤算法与基于图的算法

相比较于基于内容的推荐算法，协同过滤算法与基于图的算法优势在于(陈浩, 2018)：

（1）更易于实施，不需要了解文章的内容，就可以进行推荐。因此，它们不会受限制于特征的提取和转换难度。

（2）推荐更多元化，相比于基于内容的算法会推荐更多新颖的论文。

但它们也有自己的问题：

（1）数据稀疏性问题。当论文数目很多，而相对的每个用户读过的很少，这时推荐结果的召回率会很低。

（2）不关注论文内容，论文发表时间和发表刊物。而一些用户可能最关注的就是这些。对这些用户的推荐不如基于内容的推荐算法有效。

（3）也存在冷启动问题，对新用户难以做出准确推荐。

3、混合推荐算法

混合推荐算法融合了多种推荐算法的混合推荐算法能在一定程度上能部分解决上述推荐算法的部分缺陷，但还是存在很多问题。其中，最难以解决的问题是如何分配各种推荐算法的权重。往往，由于权重分配不当，在对于特定用户推荐时，推荐的准确度不一定比单一推荐算法更好。

（三）传统论文推荐算法存在的普遍问题

1、冷启动问题

冷启动问题分为两个方面——用户冷启动和内容冷启动。前者是指新的用户在初期特征和偏好模糊，同时用户浏览的论文记录很少，因此论文推荐算法准确性不足。后者指新的论文读者很少，所以难以推荐给用户。

2、无法针对用户兴趣的动态变化

对于传统论文推荐算法，当一个长期用户的研究方向或兴趣出现转变时，论文推荐算法很难及时做出反馈。特别是对于普通的论文读者（比如本科生），在不同时期有着不同种类的论文阅读需求。而传统论文推荐算法往往是基于过去所有的浏览记录做出推荐。

3、重视短期收益，轻视长期收益

传统论文推荐算法追求的是短期吸引用户的点击，而不能长期引导用户发现新的兴趣点。所以在长期，传统论文推荐算法的推荐能力远远不如短期。

从以上三点可以看出，目前论文推荐算法还无法很好的满足用户的需求。为此本文提出了新的论文推荐算法思路。

（四）基于强化学习的论文推荐算法

近年来，一些学者将强化学习技术应用到商品推荐算法中，比如淘宝与京东所使用的商品推荐系统。这些推荐系统（Agent）把用户视作环境（Environment），把商品搜索后的推荐问题视作典型的顺序决策问题。系统把每一次的商品推荐选择看作一次试错，把用户的反

馈作为奖励（Award）。在反复的试错中，推荐系统算法学习到最优的推荐排序，并最大化积累奖励。基于强化学习的商品推荐算法在冷启动的方面的性能一般要好于传统商品推荐算法，而当用户兴趣改变时能及时做出反应，同时会注重推荐的长期效应。

本文受此启发，考虑到可以将强化学习的思想应用在论文推荐算法上。

（五）本文的主要内容和创新点

1、主要内容

本文首先阐述了论文推荐算法的发展现状，分析了当前论文推荐存在的问题。针对传统论文推荐算法的问题，本文提出了基于强化学习的论文推荐算法设计新构思，以解决这些问题。之后，本文设计了对比实验，比对传统推荐算法和本文推荐算法之间的推荐效果差别。最后，本文分析总结了强化学习系统的优势与不足，并给出算法可改进的方向。

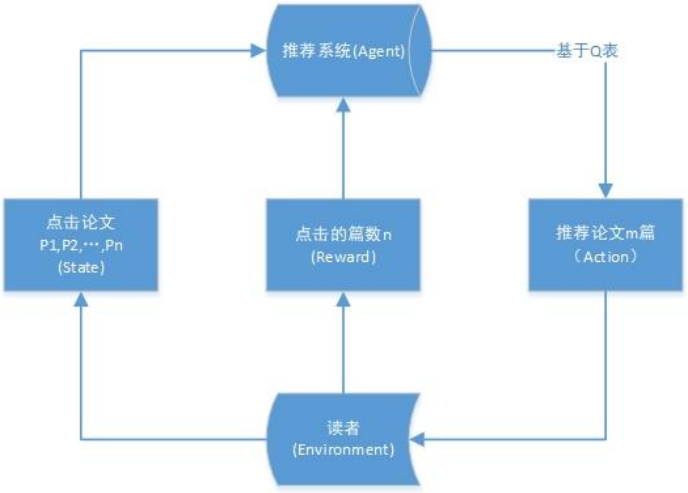

图 1：推荐算法概念图

上图是本文推荐算法的概念图。本文推荐算法是一个基于强化学习的动态推荐算法。首先，由推荐系统(Agent)根据读者阅读历史给出初次论文推荐，读者根据喜好选择推荐列表中感兴趣的文章。之后，读者的选择将反馈给推荐系统(Agent)，推荐系统(Agent)将根据这些反馈更新自己的推荐策略，再给出下一次推荐。以此循环往复，直到读者退出推荐系统。在一次次的推荐和反馈中，论文推荐算法不断优化自己，提高推荐准确率。

2、创新之处

本文设计的推荐算法相比于传统推荐算法的创新之处在于：

（1）实现动态推荐。当用户的兴趣发生改变，它可以及时捕捉，并快速改变自己的推荐策略。传统论文推荐算法过于依赖用户历史数据，短时间内难以察觉用户兴趣的转变。

（2）较少的冷启动问题。它所需要的用户历史数据不多，而是更加看重在推荐过程中用户之间的交互。在面对新用户时，虽然一开始会和传统论文推荐算法一样不太准确，但经过几轮训练后，理论上推荐准确率会快速提升。

（3）长期推荐效果更好。传统论文推荐算法追求的是短期吸引用户的点击，而不能长期探索用户、发现新的兴趣点。而本文论文推荐算法的推荐更关注长期收益，目标是能持续推荐用户感兴趣的论文。

二、相关理论概述

本章将概述与基于强化学习的论文推荐算法相关的理论知识。本文后面所构建的论文推荐算法正是基于这部分的内容。

（一）推荐算法相关理论

首先，将介绍本文论文推荐算法中使用到的推荐算法中的一些子算法，我们的推荐模型中应用到了这些算法。

1、基于内容的推荐算法——TF—IDF 方法

基于内容的推荐算法的核心是比较文本之间的相似度,然后选择与读者喜爱文章相似度高的文章推荐给读者。而为了衡量文本之间的相似度，必须要确定文本的特征，常用的一种方法是 TF-IDF 方法。

TF-IDF（term frequency - inverse document frequency）是基于内容的推荐算法中的一种重要方法，它以论文的关键字为特征，确定每个关键字的权重。TF-IDF 方法的核心思想是：在一方面，关键字 w 在论文 P 中出现的次数越多，表示 w 对于论文 P 来说越重要，越能用关键字 w 表示论文 P 的主要内容。在另一方面，关键字 w 在不同的论文中出现的越多，表示关键字 w 对区别论文差异的意义越小。TF-IDF 方法综合这两方面提出了特征设置的方法(Roberston, 2004)。设论文集中包含 N 篇论文，文档集合中包含关键字w_i的论文数为n_i，关键字w_i在论文P_i中出现的次数为f_{ij}，关键词w_i在论文P_i中的词频TF_{ij}定义为：

$$\mathrm{TF}_{ij} = \frac{f_{ij}}{max_y f_{yj}} ,$$

W_i在论文集中出现的逆频数IDF_i定义为：

$$\mathrm{IDF}_i = log\frac{N}{n_i} ,$$

由以上两式可以得到：

$$v_{ij} = \mathrm{TF-IDF}_{ij} = \mathrm{TF}_{ij} \cdot \mathrm{IDF}_i ,$$

有了 TF-IDF 值我们就可以轻松得出论文之间的相似度了。比如使用最常用的夹角余弦相似度算法：

$$\mathrm{sim}(P_x, P_y) = \cos(P_x, P_y) = \frac{\sum_i v_{ix} v_{iy}}{\sqrt{\sum_i v_{ix}^2}\sqrt{\sum_i v_{iy}^2}} .$$

2、协同过滤算法——MF 方法

如第一章所述，协同过滤算法主要分为三种——基于用户的、基于项目的、基于模型的。本文中主要使用到的是基于模型的协同过滤。而基于模型的协同过滤也包含很多种子算法，包括关联算法、聚类算法、分类算法、回归算法、矩阵分解，还有更复杂一点的神经网络，SVM 等等。本文用到的是矩阵分解算法(Matrix Factorization, MF），其核心思想如下：

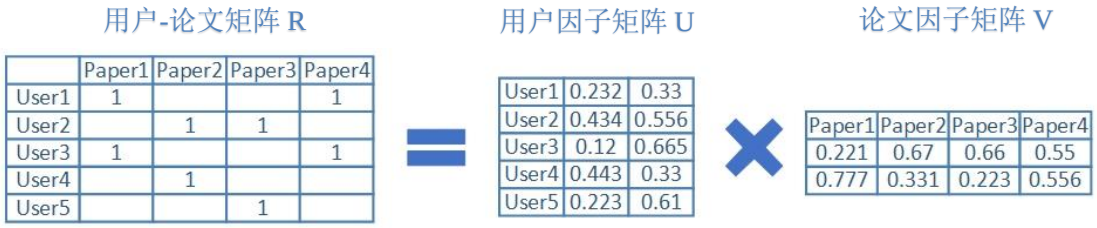

图 2：矩阵分解示意图

首先，可以由读者和论文创建出二维表（用户-论文表），如上图左边二维表所示。其中的 1 表示读者看过（或喜欢）对应论文。比如上如中，User1 喜欢 Paper1 和 Paper2。

矩阵分解的目标是把用户-论文表分解成用户因子矩阵和论文因子矩阵乘的形式，即：

$$\mathrm{R} = \mathrm{U}_{m\times k} \times \mathrm{V}_{k\times n} \ ,$$

此时可以把用户因子矩阵看作用户与 k 个特征的二维表，论文因子矩阵可以看作论文与 k 个特征的二维表。

由论文因子矩阵，我们可以进一步得到任意两篇论文之间的相似度。本文使用是夹角余弦相似度:

$$\mathrm{sim}(P_x, P_y) = \cos(P_x, P_y) = \frac{\sum_i V_{ix} V_{iy}}{\sqrt{\sum_i V_{ix}^2}\sqrt{\sum_i V_{iy}^2}} \ .$$

（二）强化学习相关理论

1、强化学习

强化学习（Reinforcement Learning）一般涉及一个代理(Agent)与环境（Environment）之间的交互。环境会根据代理的行动（Action）改变状态（State）并将奖励（Reward）反馈给代理。代理再根据状态和奖励做出新的行动，以此与环境不断交互。强化学习的目标是学会如何采取行动使奖励最大化(Sutton & Barto, 1999)。在本文中推荐系统是 Agent，读者是 Environment，Action 代表推荐论文，State & Reward 是从读者反馈中得到。

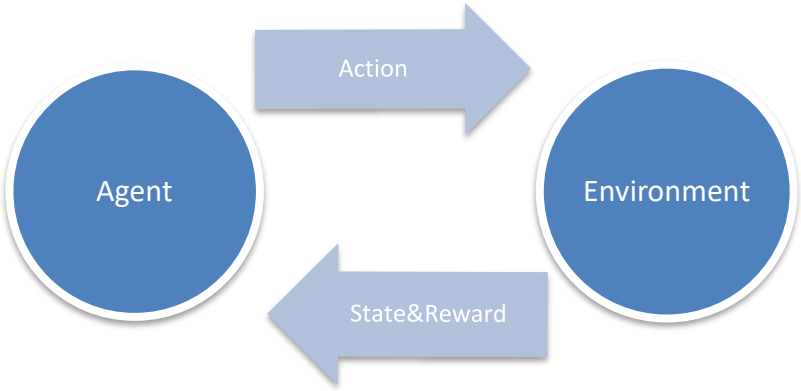

图 3：强化学习概念图

2、马尔可夫决策过程

马尔可夫决策过程（Markov Decision Process）是强化学习的核心理论，它是基于马尔可夫过程理论的随机动态系统的最优决策过程(Whittle & Puterman, 1994)。马尔可夫决策过程包含五个关键的元组(S, A, R, P, γ)。

（1）S 表示表示一组可能的状态（State）；

（2）A 表示一组可能采取的行动（Action）；

（3）$P_a(s, s')$表示 t 时刻由状态 s，采取行动 a，转移到s'的概率，如下等式所示：

$$P_a(s, s') = Probility(s_{t+1} = s' \mid s_t = s, a_t = a) \ ;$$

（4）$R_a(s, s')$表示 t 时刻由状态 s，采取行动 a，转移到s'所得到的奖励;

（5）$\gamma \in [0, 1]$是折现系数，代表未来奖励与现在奖励之间的重要差别。

马尔可夫决策过程可以概括为以下几步：

（1）t = 0 时，初始化环境的状态$s_0 \sim p(s_0)$;

（2）代理 Agent 选择行动a_t;

（3）环境根据行动a_t，给出奖励$r_t \sim R(.|s_t, a_t)$，和下一个状态$s_{t+1} \sim P(.|s_t, a_t)$;

（4）代理 Agent 收到r_t, s_{t+1};

（5）t = t+1，转到（2）

如果定义π 为 Agent 由 State 选择 Action 的策略，那么强化学习的目的就是找到策略π^*使得累计奖励的现值最大，即：

$$min_\pi \sum_{t>0} \gamma^t r_t \ .$$

3、Q – Learning

Q - Learning 是本文强化学习论文推荐算法的核心算法(Tsitsiklis, 1994)。它是强化学习算法中基于价值（Value-Based）的算法。Q - value 即为 Q（s, a）就是在某一时刻的 s 状态下(s∈S)，采取行动 a (a∈A)动作，根据策略π能够获得奖励 r 的累计现值期望，可以用公式表示为：

$$\mathrm{Q}^{\pi}(\mathrm{s},\mathrm{a}) = \mathrm{E}[\sum_{t>0}\gamma^{t}r_{t}\,|s_{0}=s,a_{0}=a,\pi]\ ,$$

所以，最优策略π^{*}条件下的 Q – value 公式可以表示为：

$$\mathrm{Q}^{*}(\mathrm{s},\mathrm{a}) = \max_{\pi}\mathrm{E}[\sum_{t>0}\gamma^{t}r_{t}\,|s_{0}=s,a_{0}=a,\pi]\ ,$$

同时可以验证 Q – value 公式服从贝尔曼等式（Bellman equation）：

$$\mathrm{Q}^{*}(\mathrm{s},\mathrm{a}) = \mathrm{E}_{s'\sim\varepsilon}\left[r+\gamma\max_{a'}\mathrm{Q}^{*}(s',\mathrm{a}')\,|s,a\right]\ ,$$

所以 Q – Leaning 算法的**主要思想**就是将 State 与 Action 构建成一张 Q - table 来存储 Q - value，环境会根据代理 Agent 的行动 a 反馈相应的奖励 r 和状态 s，而代理 Agent 会根据 s，a，r 不断利用：

$$Q_{i+1}(\mathrm{s},\mathrm{a}) = \mathrm{E}_{s'\sim\varepsilon}\left[r+\gamma\max_{a'}\mathrm{Q}_{i}(\mathrm{s}',\mathrm{a}')\,|s,a\right]\ .$$

来更新 Q – table，然后根据 Q - value 来选取能够获得最大的奖励的行动 Action，以此循环。

（三）推荐算法评估指标

评估一个论文推荐算法的好坏一般常用的指标有平均误差（MAE）、准确率（Precision）、召回率（Recall）、综合评价指标（F-Measure）等。

1、平均误差 MAE

MAE(Mean Absolute Error)通过计算所有真是评分和预测评分之间的差异来确定推荐是否准确。MAE 越小，推荐的效果越好。假设真实的读者对论文集合的评价向量为（v_1，v_2,…，v_i），而预测的（w_1，w_2,…，w_i），那么 MAE 的公式为：

$$\mathrm{MAE} = \frac{\sum_{i=1}^{n}|w_i - v_i|}{n}\ .$$

2、准确率和召回率

准确率(Precision)和召回率(Recall)是最常见的统计指标。在本文中，召回率被定义为推荐论文中用户喜欢的论文数占读者所有喜欢的论文数的百分比;准确率被定义为推荐论文中读者喜欢的论文数占所推荐的所有论文数的百分比。假设 TP 表示推荐论文中读者喜欢的论文数，FN 是没有推荐给读者但读者喜欢的论文数量，FP 表示推荐论文中读者不喜欢的论文数。准确率和召回率大小都介于 0 到 1，越大表示推荐效果越好。准确率和召回率可以如下表示：

$$\text{Precison} = \frac{TP}{TP+FP} \ ,$$

$$\text{Recall} = \frac{TP}{TP+FN} \ .$$

3、综合评价指标

准确率和召回率有时会出现矛盾，这是可以采用 F-Measure 方法综合考虑两者。F-Measure 是召回率和准确率的加权调和平均，当 a = 1 时，就是最常见的 F1。本文就是采用最简单的 F1 来确定推荐效果。F - Measure 越大表示推荐效果越好。

$$\text{F} - \text{Measure} = \frac{(a^2+1)\cdot precision\cdot recall}{a^2(precision+recall)} \ .$$

三、论文推荐算法的设计

（一）论文推荐算法总体设计

1、总体框架

本文论文推荐算法的主体框架如下图所示：

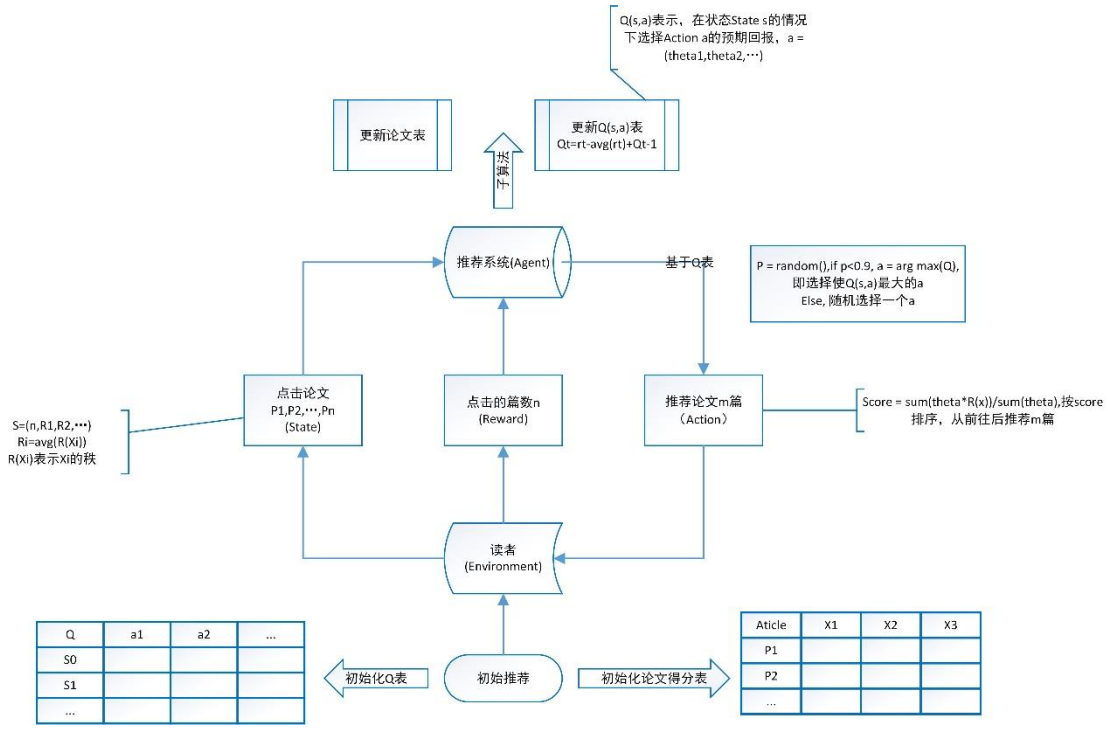

图 4：算法主题框架图

可见，算法核心在于一个推荐系统（Agent）与读者（Environment）之间交互的循环。该算法可以大致分为三个部分，初始推荐部分、读者部分、推荐系统部分

首先在初始推荐部分，算法完成对 Q 表（Q – table）和论文得分表的初始化。Q 表要初始化其中的 Q（s, a）的值（s 表示返回状态，a 表示推荐策略）。论文得分表通过计算论文在各个特征（x1，x2，x3…）上的得分得到。之后，参考论文得分表完成初次推荐。读者部分主要任务是从推荐的论文中选出的自己喜爱的论文，这些论文的篇数作为奖励反馈给推荐系统，同时这些论文的特征作为状态也传入推荐系统。最关键的是推荐系统部分，它的任务则是根据输入的状态和奖励完成对用户论文列表以及 Q 表的更新，再利用 Q 表完成下一次推荐。

2、算法主流程

基于 Q – Learning 的推荐算法主要有一下的几个步骤：

（1）根据读者的阅读记录计算待推荐论文库中所有论文的得分 Score 并排序，同时初始化 Q 表；

（2）推荐前 m 篇论文给读者；

（3）读者选择是否结束推荐，结束推荐则算法结束，否则进入（4）；

（4）读者从推荐的论文选择 n 篇喜爱的文章，结果反馈给推荐系统；

（5）推荐系统根据读者反馈更新 Q 表，将读者点击的文章加入读者阅读记录。

（6）推荐系统根据 Q 表结合用户阅读记录得到新的论文得分排序，转到（2）

因此推荐算法的流程图如下所示：

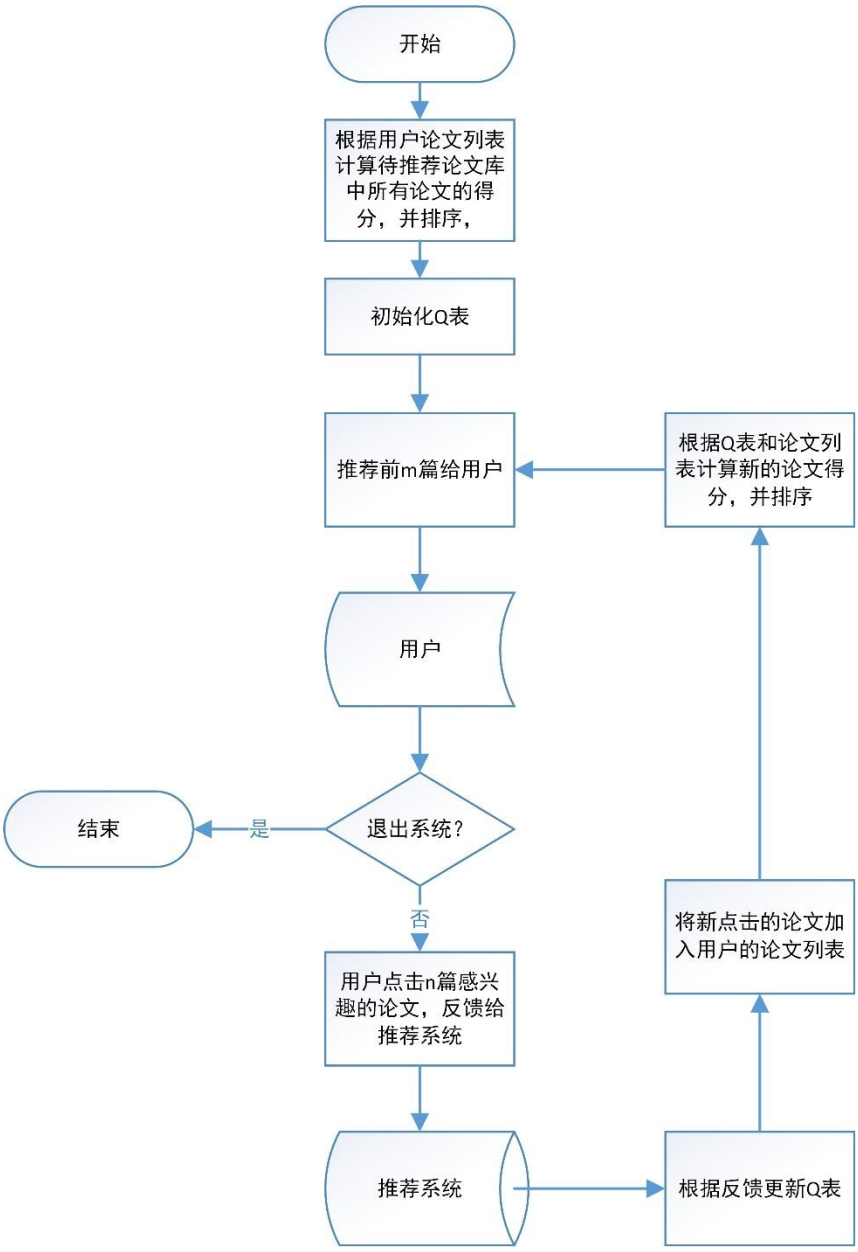

图 5：算法流程图

（二）初始推荐部分

1、提取论文特征

在初始推荐部分，要做的第一步是提取论文的特征。论文的特征要能体现论文的特点，拥有相似论文特征的论文，一般是相似的论文。在本算法中，我们可以选用的论文特征主要有两种——基于内容的论文特征和基于协同过滤的论文特征。

基于内容的论文特征也有两种——数值型和文本型。一般，论文的数值型特征有：出版时间、出版刊物的影响因子、字数等。这些数值型特征是结构化的数据，所以几乎不需要经过处理就可以直接使用。对于论文P_i，将它的数值型特征记录为D_{i1}，D_{i2}，…，D_{ij}

论文的文本型特征主要有：论文题目、论文关键字、论文摘要、论文正文。这些非结构化的文本数据必须处理转化成结构化的数据以量化论文的特征，以通过计算相似度进行论文推荐。对于论文的这些文本特征，本文采用的 TF-IDF 方法（详见二、（一）、1）将他们转化为结构化数据。以处理论文题目（title）为例，最终论文库$P_{all}\{P_1, P_2, \dots, P_n\}$可以转化为n × m 的矩阵$V_{title} = \{v_{ik} | i \in (0,n), k \in (0,m)\}$, 其中$v_{ik}$表示论文$P_i$在关键词$w_k$上的 TF-IDF 值。同理，对于关键字、摘要也可以得到对应的矩阵V_{key}和V_{abs}。

基于协同过滤的论文特征是利用读者和论文二维表计算得出的。在本文中基于协同过滤的论文特征使用矩阵分解法提取出的（详见二、（一）、2）。通过矩阵分解法，最终我们得到矩阵论文因子矩阵$V_{k\times n}$，转置后$V_{n\times k}$即为所需的论文特征矩阵。

表 1 论文特征分类表

主类型	子类型	特点	例子
基于内容	数值型	结构性化数据	出版时间、出版刊物的影响因子
基于内容	文本型	非结构化数据	论文题目、论文关键字、论文摘要
基于协同过滤		二维矩阵	用户-论文表得到

2、初始化论文得分表

假设我们现在要给读者 U 推荐论文。首先，我们要从 U 的历史阅读记录P_{his}中提取最近阅读的 h 篇论文组成论文集合$P_u\{P_{u1}, P_{u2}, \dots, P_{uh}\}$，$P_u \subset P_{all}$，且$P_u$论文编号（下标）与$P_{all}$对应。如：$P_u$中$\{P_1, P_3, P_5\}$对应$P_{all}$中$\{P_1, P_3, P_5\}$。

然后计算P_u中各论文与P_{all}中各论文在每个特征之间的相似度。对于数值型的特征，直接以特征相减的绝对值作为距离，取导数然后再标准化后的绝对值作为相似度。以论文P_i, P_j在D_1特征上的相似度为例：

$$sim(P_i, P_j|D_1) = \left|\frac{1/|D_i - D_j| - avg}{std}\right|,\quad \text{avg} = \frac{\sum_{j=1}^{n} 1/|D_i - D_j|}{n},\quad \text{std} = \sqrt{\frac{(1/|D_i - D_j| - avg)^2}{n}}\ ,$$

对于文本型特征和基于协同过滤的特征，我们则是采用余弦相似度，以P_x, P_y在题目特征上的相似度为例 ：

$$\text{sim}\left(P_x, P_y|title\right) = \cos\left(P_x, P_y\right) = \frac{\sum_i v_{ix} v_{iy}}{\sqrt{\sum_i v_{ix}^2}\sqrt{\sum_i v_{iy}^2}},\ V_{title} = \{v_{ik}|i \in (0,n), k \in (0,m)\}\ ,$$

对于P_{all}中的论文P_i，对每个特征计算P_i与P_u中 h 篇论文的相似度的平均值，结果即为论文P_i在特征 C 上的得分，用公式表示为：

$$\text{Score}(P_i|C) = \frac{\sum_{j=1}^{h} \text{sim}(P_j, P_i|c)}{\text{h}},\ \text{C} \in \{\text{D}, \dots, \text{title}, \dots\}\ ,$$

最后由论文编号 index 和所有的$\text{Score}(P_i|C)$，可以初始化论文得分表：

$$L_{ps}(\text{index}, \text{Score}(P_i|C_1), \text{Score}(P_i|C_2), \dots, \text{Score}(P_i|C_c))\ .$$

3、初始化 Q 表

初始推荐部分中，我们还要完成Q表的初始化。Q表是一个二维表，行变量为$\{s_0, s_1, \dots s_a\}$代表 a 种不同的论文返回状态，

例如：s_0表示返回表为空（P_{sel} = Null）。

列变量为$\{a_0, a_1, \dots a_b\}$代表 b 种不同的推荐策略。在本算法中，s 由返回论文的各特征得分确定。而 a 被定义为各特征的权重分配策略。在计算论文总得分时，需要赋予每个特征不同权重φ_c：

$$\text{Score}(P_i) = \sum_c \varphi_c \text{Score}(P_i|C_c)\ ,$$

例如：$a_0 = (1,1,\dots,1)$, 那么$\varphi_1 = 1, \varphi_2 = 1, \dots, \varphi_c = 1$。

Q - value 即为 Q（s, a）就是在某一时刻的 s 状态下(s∈S)，采取行动 a (a∈A)动作，能够获得奖励 r 的累计现值期望。Q 表有三种初始化方法：

（1）将所有值设置为 0，即$\text{Q}(s_i, a_j) = 0, \forall s_i, \forall a_j$;

（2）设置为服从 U(0,1)的随机数，即$\text{Q}(s_i, a_j) = Random, \forall s_i, \forall a_j$;

（3）根据其他读者的 Q 的平均值平均来设置，$\text{Q}(s_i, a_j) = \frac{\sum_u \text{Q}_u(s_i, a_j)}{n_u}, \forall s_i, \forall a_j$，其中 u 表示不同读者。

4、初始推荐

在初次推荐时我们赋予每个特征相同的权重（即$\varphi_c \equiv 1$），利用论文得分表中各特征加权可以得到每篇论文的最终得分。P_i的得分为：

$$\mathrm{Score}(P_i) = \sum_c \varphi_c \mathrm{Score}(P_i|C_c)，\ \varphi_c \equiv 1\ .$$

之后，我们按 Score 降序排列论文，从中剔除用户已经阅读过的文章，最后得到一个推荐排序表。选取其中的前 m 篇组成推荐列表P_{rec}推荐给读者。

（三）读者部分

1、真实情况

真实推荐情况中，读者部分是三个部分中机制最简单的部分。读者根据推荐列表P_{rec}，从中选择自己喜欢的论文，组成选择列表P_{sel}。P_{sel}将被发送给推荐系统，经过推荐系统处理后，将会有新的P_{rec}发送回读者部分。一直循环，直到读者选择退出推荐算法。

2、模拟情况

有时候我们可以先用读者阅读记录数据进行模拟实验来训练得到更好的初始 Q 表。在模拟实验中，读者部分在接受推荐列表P_{rec}后会和读者喜爱的论文列表P_{like}进行比对,找到P_{rec}与P_{like}的交集即为选择列表P_{sel}。P_{sel}将被发送给推荐系统，经过推荐系统处理后，将会有新的P_{rec}发送回读者部分。一直循环，直到读者选择退出推荐算法。

（四）推荐系统部分

1、更新 Q 表

在接收到读者反馈的P_{sel}后，推荐系统开始更新 Q 表。更新步骤主要有三步：

（1）确定要更新的 Q(s, a)。其中 a 就是上次推荐所用的推荐策略（一组$\{\varphi_c\}$）。s 是根据上次反馈回来的P'_{sel}确定的；

（2）计算奖励 r。在本算法中，奖励就是本次P_{sel}中论文篇数与选择论文篇数期望E_{sel}的差值（E_{sel}可以通过多次实验取平均获得），公式为：

$$\mathrm{r} = \mathrm{N}(P_{sel}) - E_{sel}\ ，$$

（3）根据公式更新 Q(s, a)。更新公式如下：

$$Q_{i+1}(\mathrm{s},\mathrm{a}) = r + \gamma \mathrm{Q}_i(\mathrm{s},\mathrm{a})，\ \gamma \in (0,1)\ ，$$

2、更新论文得分表

在接收到读者反馈的P_{sel}后，经过三步更新论文得分表

（1）推荐系统将P_{sel}中的论文加入用户阅读记录P_{rec}，即：

$$P_{rec}^{*} = P_{rec} \cup P_{sel} \ ；$$

（2）从新的用户阅读记录P_{rec}^{*}中提取最近阅读的 h 篇，组成论文集合P_u；

（3）根据P_u计算新的论文得分表L_{ps}，具体计算过程同本章第三节（三、（二）、2）

3、再次推荐

在得到新的 Q 表和论文得分表之后可以开始新一轮的推荐：

（1）确定推荐策略 a。先生成一个随机数 rand，如果 rand<ε则随机选择一个 a 作为下次推荐的策略；否则：

$$\mathrm{a} = \max_{a} Q(s,a),$$其中 s 是由P_{sel}得到的；

（2）a 确定以后就可以计算论文的总体得分，根据 a 中$\{\varphi_c\}$：

$$\mathrm{Score}(P_i) = \sum_c \varphi_c \mathrm{Score}(P_i|C_c) \ ；$$

（3）之后，我们按 Score 降序排列论文，从中剔除用户已经阅读过的文章，最后得到一个推荐排序表。选取其中的前 m 篇组成推荐列表P_{rec}再次推荐给读者。

四、推荐算法对比试验

（一）实验数据集

本文对比试验采用的是由 CiteULike 论文搜索网站(http://www.citeulike.org)提供的两个相关联的数据集(http://www.citeulike.org/faq/data.adp)。可以通过括号中的网站申请下载这两个数据集。

其中，Paper 数据集记录着 16980 篇论文的信息，包括论文标题、作者和摘要。数据集中有少量的乱码，在清理时做了替换处理。这个数据集主要是为了提取论文的文本型特征数据截图如下。

doc.id	title	citeulike.ic	raw.title	raw.abstract
1	the metab	42	The metal	To elucidate the organizational and evolutionary principles of the
2	reverse en	43	Reverse Er	Advanced technologies and biology have extremely different phys
3	exploring	44	Exploring	The study of networks pervades all of science, from neurobiology
4	comparati	46	Comparat	Comprehensive protein protein interaction maps promise to revea
5	navigation	47	Navigatio	The small-world phenomenon — the principle that most of us are
6	random g	48	Random g	Recent work on the structure of social networks and the internet
7	artificial g	49	Artificial g	Motivation: Large-scale gene expression profiling generates data
8	the segme	50	The segm	All insects possess homologous segments, but segment specificati
9	the evolut	52	The evolu	A long-standing challenge to evolutionary theory has been wheth
10	early langu	60	Early langu	Infants learn language with remarkable speed, but how they do it
11	organizati	61	Organizat	Recent research has revealed general principles in the structural
12	motifs in k	62	Motifs in k	Complex brains have evolved a highly efficient network architectu

图 6：Paper 数据集

User-info 数据集包括自 2004 年到 2010 年的 5551 个用户和他们的阅读记录。数据集中总共有 204986 对用户-论文记录，数据截图如下。

user.id	doc.id	rating
1	496	1
1	1632	1
1	2318	1
1	2527	1
1	2847	1
1	2932	1
1	3172	1
1	3298	1
1	3333	1
1	3405	1
1	3592	1

图 7：User-info 数据集

（二）Q 表设置

1、状态 State 设置

我们需要设置 Q 表中状态 State（s）。它代表着论文的反馈状态。在本实验中一共有 3 个特征：题目、摘要、协同过滤特征。用最简单的方法，我们可以设置A_3^3+1 即 7 种状态 s。根据返回的P_{sel}，参照论文得分表l_{ps}，经过求和可以得到各特征的总分，计算公式为：

$$\mathrm{Sc}_j = \sum_i \mathrm{Score}(P_i|C_j)$$

根据 Sc 的大小可以如下定义各状态 s：

表 2：状态 S 设置表

$\boldsymbol{S}$	Sc 排序
$\boldsymbol{S_0}$	$P_{sel} = null$
$\boldsymbol{S_1}$	$\mathrm{Sc}_1 \leq \mathrm{Sc}_2 \leq \mathrm{Sc}_3$
$\boldsymbol{S_2}$	$\mathrm{Sc}_1 \leq \mathrm{Sc}_3 < \mathrm{Sc}_2$
$\boldsymbol{S_3}$	$\mathrm{Sc}_2 < \mathrm{Sc}_1 \leq \mathrm{Sc}_3$
$\boldsymbol{S_4}$	$\mathrm{Sc}_2 \leq \mathrm{Sc}_3 < \mathrm{Sc}_1$
$\boldsymbol{S_5}$	$\mathrm{Sc}_3 < \mathrm{Sc}_2 \leq \mathrm{Sc}_1$
$\boldsymbol{S_6}$	$\mathrm{Sc}_3 \leq \mathrm{Sc}_1 < \mathrm{Sc}_2$

2、策略 Action 设置

我们还需要设置 Q 表中状态策略 Action（a）。实际上 a 代表着不同的推荐特征的权重。不同权重意味着不同的推荐策略。简单起见，本文将 a 设置如下表所示：

表 3：策略 A 设置表

A	φ_1	φ_2	φ_3
$\boldsymbol{a_0}$	1	1	1
$\boldsymbol{a_1}$	1	2	3
$\boldsymbol{a_2}$	1	3	2
$\boldsymbol{a_3}$	2	1	3
$\boldsymbol{a_4}$	2	3	1
$\boldsymbol{a_5}$	3	1	2
$\boldsymbol{a_6}$	3	2	1

3、Q-表初始设置

本实验中采用的是随机初始化设置，Q - value 设置为服从 U(0,1)的随机数，即：

$$\mathrm{Q}(s_i, a_j) = Random, \forall s_i, \forall a_j \ .$$

（三）实验过程

在实验中，因为计算机能力的限制，所以只能从 5551 个读者中采用随机抽样的方法选择 10%的用户作为实验对象。同时对于每个读者，实验选取其阅读记录的前 10 篇作为读者的阅读历史P_{his}，读者其余的历史记录作为读者潜在论文喜好。

1、实验组

在实验组中，我们利用本文第三章的算法，完成模拟推荐过程。其中读者部分采用在模拟情况下的算法来确定读者是如何从推荐列表中自己喜爱的文章。对于每个读者，每次推荐给读者的文章数为 20，重复推荐 50 次，每次读者选择后记录选择的论文数目，计算并输出平均误差、准确率、召回率和综合指标。

2、对照组

我们设置了四个对照组，分别基于内容推荐、协同过滤和混合推荐：

表 4 实验组与对照组对比

组别	推荐算法	备注	图中简称
对照组 1	基于内容	利用题目相似度	CB - title
对照组 2	基于内容	利用摘要相似度	CB - abstract
对照组 3	协同过滤	利用矩阵分解法	CF
对照组 4	混合推荐	利用结果融合法	Hybrid
实验组	强化学习	利用 Q - Learning	Q - Learning

在每个对照组实验时，分别计算推荐数为 10、50、100、200、500 时，读者从推荐列表中选择的论文数。之后通过这些论文数来计算并输出每个算法的平均误差、准确率、召回率和 F - Measure。实现代码见附录二。

（四）实验结果与分析

1、评估指标对比

四个评估指标的结果绘制为折线图后如下所示：（具体的数据见附录一）

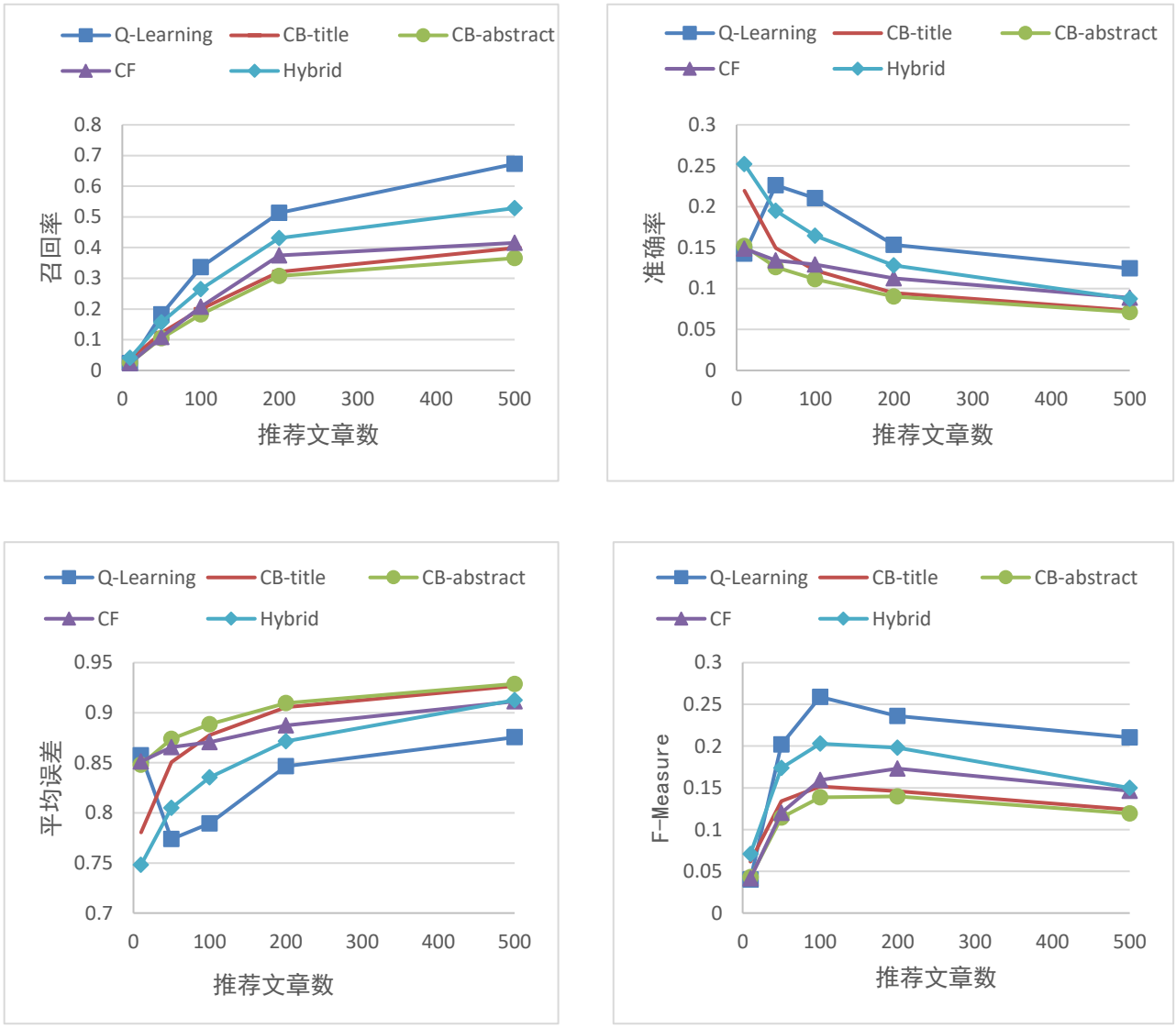

图 8：评估指标对比图

首先，通过召回率对比，我们可以发现：当推荐数比较小时，各推荐算法的推荐效果极为接近。而当推荐数逐渐增加时，本文设计的推荐算法（Q－Learning）召回率增长最快，当推荐数达到 500 时，召回率达到接近 70%，这远远大于其他的推荐算法。

其次，综合来看准确率和平均误差率的对比图。虽然在一开始的时候，本文推荐算法处于劣势，但它能很快的改变推荐策略，在推荐数达到 40 的时候，推荐效果超过其他的推荐算法。随着推荐数的增大，本文推荐算法一直有着高准确率和低误差率。因为本文是模拟实

验，用户的兴趣集是有限的，所以准确率会随着推荐数目的增加，最终无限趋紧于 0；同样误差率会无限趋紧于 1。

最后，观察综合指标 F-Measure。在低推荐数时，各种算法效果相近。但本文推荐算法和混推荐算法在推荐数低时 F-Measure 增长率很高，所以很快推荐效果就好于其他算法。而之后混合推荐算法下降的比本文推荐算法快。最终，随着推荐数进一步增大本文推荐算法要稳定优于其他推荐算法。

通过这些指标的，我们可以发现，在推荐数比较低的时候，本文算法的推荐效果略弱于传统推荐算法，而当推荐次数增多后，本文算法的推荐准确率大幅上升，而其他传统算法准确率一直处于下降状态。当推荐数超过 50 后，本算法的各种指标要明显好于对照组的四个算法。所以我们从中可以得到本论文算法的优势在于：

（1）部分解决了冷启动问题。算法解决冷启动问题的方向是更加看重在推荐过程中用户之间的交互。在面对新用户时，虽然一开始会和传统论文推荐算法一样不太准确，但经过几轮训练后，理论上推荐准确率会快速提升。

（2）算法注重长期收益。算法会不断更新自己的推荐策略。传统论文推荐算法追求的是短期吸引用户的点击，而不能长期探索用户发现新的兴趣点。而本文论文推荐算法的推荐更关注长期收益，目标是能持续推荐用户感兴趣的论文。

2、具体实例分析

为了更好的分析本文推荐算法和传统推荐算法之间的区别，我们随机抽取了一名用户，具体计算出其出前十次的推荐结果，推荐结果汇总如下：

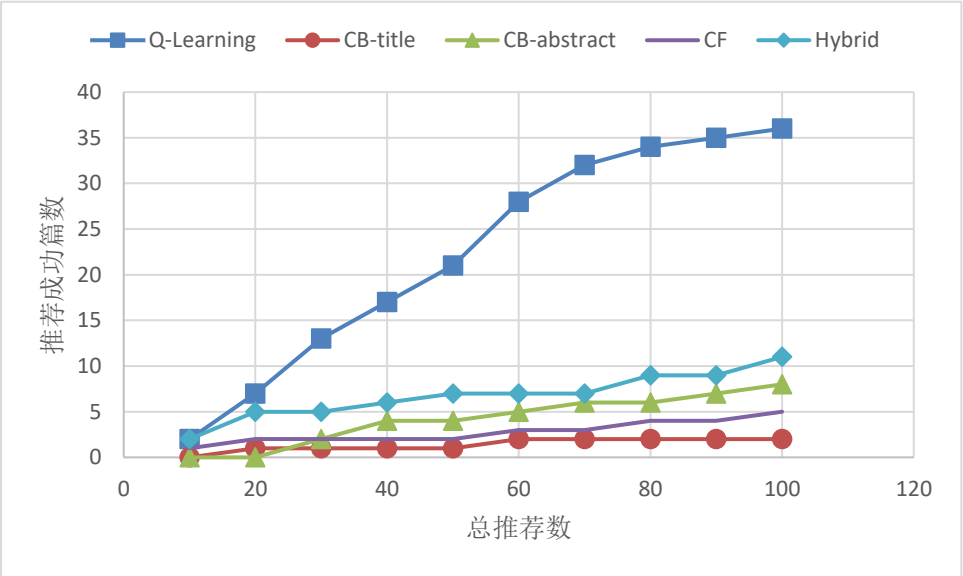

图 9：推荐结果

可见，本文推荐算法的推荐效果明显要好于好于其他推荐算法的效果，这也符合上一小节的结论。现在为了探索本文推荐算法与传统推荐算法的差异，我们具体来看一下本文推荐算法和其他推荐算法在两轮中推荐的论文。

表 5 第 1 轮推荐按结果

推荐算法	推荐论文编号	
Q - Learning	7932	2145
CB-title		
CB-abstract		
CF	4531	
Hybrid	7932	2145

在第一轮推荐中，本文推荐算法(Q-Learning)和混合推荐算法(Hybrid)的推荐结果完全一样，查询 Paper 数据集发现这两篇论文（7932，2145）都是关于 RNA 的。

表 5 第 2 轮推荐按结果

推荐算法	推荐论文编号				
Q - Learning	1317	5485	9231	1296	1749
CB-title	7932				
CB-abstract					
CF	2289				
Hybrid	2305	5249	6997		

在第二轮推荐的时候，本文推荐算法(Q-Learning)和其他的推荐算法有了较大差别。其中 Q－Learning 推荐算法推荐的文章主题发生了转变，以 RNA 为基础扩展到其他生物学领域如蛋白质、进化。而其他的推荐算法推荐的还是 RNA 主题。

由这个推荐的转折，我们可以总结出本文推荐算法的优势所在——在动态推荐过程中，可以快速的拓展主题。本文推荐算法可以探索到更多读者感兴趣的主题，不局限于在同一个主题的推荐。

五、总结

近年来，随着信息技术的飞速发展，互联网中的论文发表数目呈爆炸式增长。为了进行科学研究，学者们必须通过阅读他人的论文，从中吸取经验，并进一步发展。面对的众多的论文，如何有效的进行论文推荐变得十分关键。本文设计的基于 Q－Learning 的算法，将强化学习的思想引入论文推荐，经过实验论证，这种算法可以有效提高推荐效果。

（一）算法特点总结

本文推荐算法是一个基于强化学习的动态推荐算法。首先，由 Agent 根据读者阅读历史给出初次论文推荐，读者根据喜好选择推荐列表中感兴趣的文章。之后，读者的选择将反馈给 Agent，Agent 将根据这些反馈更新自己的推荐策略，再给出下一次推荐。以此循环往复，

直到读者退出推荐系统。在一次次的推荐和反馈中，论文推荐算法不断优化自己，提高推荐准确率。本文算法优势在于：

（1）实现了动态推荐。当用户的兴趣发生改变，它可以及时捕捉，并快速改变自己的推荐策略。传统论文推荐算法过于依赖用户历史数据，短时间内难以察觉用户兴趣的转变。

（2）较少的冷启动问题。它所需要的用户历史数据不多，而是更加看重在推荐过程中用户之间的交互。在面对新用户时，虽然一开始会和传统论文推荐算法一样不太准确，但经过几轮训练后，理论上推荐准确率会快速提升。

（3）注重长期收益。传统论文推荐算法追求的是短期吸引用户的点击，而不能长期探索用户发现新的兴趣点。而本文论文推荐算法的推荐更关注长期收益，目标是能持续推荐用户感兴趣的论文。

（二）不足和展望

本文设计的算法还存在着一些不足:

（1）本文在提取论文文本特征时，所有的关键字都是作为单词，没有联系到短语和段落的含义。这样提取的特征可能不太准确。可能可以采用更新的 NLP 技术来优化这个步骤。

（2）Q 表的 Action 是离散的，即推荐策略是有限的。而实际上推荐策略中的参数是连续的。因此，本算法可能无法找到最优的推荐策略，只能寻找到近似的策略进行推荐。或许可以采用强化学习中的 Policy Gradients 方法来确定最优的参数。

（3）没有联系用户的个性化信息。目前很多推荐算法关注用户的个性化信息，比如性格、喜好、职业等等。或许本算法中可以利用这种类似信息来更好的完成 Q 表初始化。但囿于实验数据的限制，算法中并没有使用这种方法初始化 Q 表。

六、参考文献

[1]周昊. 基于 CiteULike 的论文推荐系统设计与实现[D].复旦大学,2012.

[2]杨博,赵鹏飞.推荐算法综述[J].山西大学学报(自然科学版),2011,34(03):337-350.

[3]谭红叶,要一璐,梁颖红.基于知识脉络的科技论文推荐[J].山东大学学报(理学版),2016,51(05):94-101.

[4]李冉,林泓. 基于频繁主题集偏好的学术论文推荐算法[J]. 计算机应用研究,2019,(09):1-6.

[5]黄立威,江碧涛,吕守业,刘艳博,李德毅.基于深度学习的推荐系统研究综述[J].计算机学报,2018,41(07):1619-1647.

[6]谢玮,沈一,马永征.基于图计算的论文审稿自动推荐系统[J].计算机应用研究,2016,33(03):798-801.

[7]杨博,赵鹏飞.推荐算法综述[J].山西大学学报(自然科学版),2011,34(03):337-350.

[8]曹占伟. 基于 Spark 的推荐算法研究[D].西南交通大学,2018.

[9]陈金鹏. 基于多兴趣的学术论文推荐研究[D].内蒙古大学,2017.

[10]周泉. 基于论文社区的文章推荐及评价[D].上海交通大学,2015.

[11]陈克寒,韩盼盼,吴健.基于用户聚类的异构社交网络推荐算法[J].计算机学报,2013,36(02):349-359.

[12]李建国,毛承洁,刘晓,梁茹.学术信息服务平台的研究与设计[J].华南师范大学学报(自然科学版),2012,44(03):51-54.

[13]Lops P, de Gemmis M, Semeraro G. Content-based recommender systems: state of the art and trends. In: Recommender systems handbook. Springer, 2011. : 73–105.

[14]Tiroshi A, Berkovsky S, Kaafar M A, et al. Graph-Based Recommendations: Make the Most Out of Social Data[C]. international conference on user modeling, adaptation, and personalization, 2014: 447-458.

[15]Beel J, Gipp B, Langer S, et al. Research-paper recommender systems: a literature survey[J]. International Journal on Digital Libraries[J], 2016, 17(4): 305-338.

[16]Robertson S E. Understanding inverse document frequency: on theoretical arguments for IDF[J]. Journal of Documentation, 2004, 60(5): 503-520.

[17]Sutton R S, Barto A G. Reinforcement Learning: An Introduction[C]. neural information processing systems, 1999.

[18]Tsitsiklis J N. Asynchronous Stochastic Approximation and Q-Learning[J]. Machine Learning, 1994, 16(3): 185-202.

[19]Whittle P, Puterman M L. Markov Decision Processes[J]. Journal of The Royal Statistical Society Series A-statistics in Society, 1994, 158(3).

七、附录

附录一（表格）

表 1: 评估指标对比表

指标	算法	10	50	100	200	500
召回率	Q-Learning	0.02343	0.182286	0.336324	0.513027	0.672649
	CB-title	0.035799	0.121855	0.199422	0.320907	0.398844
	CB-abstract	0.025083	0.104456	0.182842	0.307996	0.365684
	CF	0.024107	0.10844	0.207709	0.374309	0.415419
	Hybrid	0.041201	0.1569	0.264276	0.431194	0.528551
准确率	Q-Learning	0.142683	0.22622	0.210427	0.153339	0.12455
	CB-title	0.219512	0.149268	0.122256	0.094599	0.07333
	CB-abstract	0.151829	0.126098	0.111524	0.090418	0.071223
	CF	0.14878	0.134512	0.129329	0.112573	0.08876
	Hybrid	0.251829	0.195	0.164634	0.128426	0.08742
平均误差	Q-Learning	0.857317	0.77378	0.789573	0.846661	0.87545
	CB-title	0.780488	0.850732	0.877744	0.905401	0.92667
	CB-abstract	0.848171	0.873902	0.888476	0.909582	0.928777
	CF	0.85122	0.865488	0.870671	0.887427	0.91124
	Hybrid	0.748171	0.805	0.835366	0.871574	0.91258
F-Measure	Q-Learning	0.040251	0.20189	0.258881	0.236108	0.210182
	CB-title	0.061558	0.134176	0.151584	0.146123	0.123883
	CB-abstract	0.043053	0.114261	0.138544	0.139796	0.119225
	CF	0.041491	0.120077	0.159405	0.173089	0.146268
	Hybrid	0.070817	0.173887	0.202881	0.197908	0.150026

附录二（程序）

1、主程序 main_file.py

```
import csv
import time
import agent as ag
import pandas as pd
import numpy as np
import other as ot
from intro_paper import con_sim_matrix
from user_in import dis_cf_matrix, user_paper, find_suit_user
from article_list import art_array

Q = np.random.random((7, 7))    # Q 表初始化
#  给 action 表赋值
```

```
action = [[1, 1, 1], [1, 2, 3],
                [1, 3, 2], [2, 1, 3],
                [2, 3, 1], [3, 1, 2],
                [3, 2, 1]]

if __name__ == "__main__":
        time_start = time.time()
        filename = r'C:\Users\严书航\Desktop\data.csv'
        data = []
        with open(filename, "r", encoding='utf-8', errors='ignore') as f:
                csv_reader = csv.reader(f)
                header = next(csv_reader)
                for row in csv_reader:
                        data.append(row)

        data = np.array(data)
        data1 = data[:, 1]   #  标题数据
        data2 = data[:, 4]   #  摘要数据
        data3 = pd.read_csv("user-info.csv", delimiter=',')
        title_mat = con_sim_matrix(data1)
        abs_mat = con_sim_matrix(data2)
        user_mat, item_mat = dis_cf_matrix(data3)

        time_end = time.time()
        print("<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<")
        print('time cost: ', time_end - time_start, 's')

        user = find_suit_user(200, data3)   #  挑选样本用户
        print(len(user))

        number = 20   #  每次推荐文章数
        turn = 25   #  推荐次数
        recall_list = np.zeros((1, turn))
        precision_list = np.zeros((1, turn))
        out = pd.DataFrame()

        for x in user:
                docs_all = user_paper(x, data3)       #  提取读者所有文章
                doc_history = list(docs_all[:number])   #  选择作为历史数据的部分
                len_doc_all = len(docs_all)
                paperlist = art_array(title_mat, abs_mat, item_mat, doc_history)   #  根据用户历史计算得到的各论文
得分

                doc_r = ag.recomender(a=0, action=action, docs=paperlist, history=doc_history)   #  用平均分方法得
到第一次推荐
                a = 0   # action 初始值
                count = 0   #  推荐成功篇数计数
                recall_user = []   #  初始化该用户召回率
                precision_user = []   #  初始化该用户准确率
                doc_except = list(doc_history)

                for i in range(turn):
                        doc_s = ag.user(doc_r, docs_all)     #  用户选择论文
                        #print(doc_s)
                        state = ag.state(doc_s, paperlist)     #  选择的论文状态
                        reply = len(doc_s)     #  选择的论文数
                        count += reply
                        recall = count/len_doc_all
                        precision = count/((i+1)*number)
                        recall_user.append(recall)
                        precision_user.append(precision)

                        Q = ag.renewQ(state, a, reply, Q)     #  更新 Q 表
```

```
                if len(doc_s) != 0:
                    doc_history = doc_history + doc_s  # 更新用户喜爱的论文

                doc_except = doc_except + list(doc_r)  # 不推荐的文章集合

                paperlist = art_array(title_mat, abs_mat, item_mat, doc_history[-number:])  # 根据用户历史计算得到的各论文得分
                a = ag.agent(state, Q)
                doc_r = ag.recomender(a=a, action=action, docs=paperlist, history=doc_except, m=number)

            recall_list = recall_list + np.array(recall_user)
            precision_list = precision_list + np.array(precision_user)
            # print(recall_list)
            time_end = time.time()
            print("<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<")
            print('time cost: ', time_end - time_start, 's')

        recall_list = recall_list[0] / len(user)
        precision_list = precision_list[0] / len(user)
        out['recall'] = recall_list
        out['pre'] = precision_list
        print(Q)

        # 参照实验
        colum = ['x1', 'x2', 'x3', 'mix']
        for z in colum:
            recall_list = np.zeros((1, turn))
            precision_list = np.zeros((1, turn))
            for x in user:
                docs_all = user_paper(x, data3)  # 提取读者所有文章
                doc_history = list(docs_all[:number])  # 选择作为历史数据的部分
                len_doc_all = len(docs_all)
                paperlist = art_array(title_mat, abs_mat, item_mat, doc_history)  # 根据用户历史计算得到的各论文得分
                recall_user = []  # 初始化该用户召回率
                precision_user = []  # 初始化该用户准确率

                for j in range(turn):
                    num = number * (j + 1)
                    doc_rr = ot.recomender(paperlist, z, doc_history, num)
                    sn = ot.select_num(doc_rr, docs_all)
                    recall = sn / len_doc_all
                    precision = sn / num
                    recall_user.append(recall)
                    precision_user.append(precision)

                recall_list = recall_list + np.array(recall_user)
                precision_list = precision_list + np.array(precision_user)

            recall_list = recall_list[0] / len(user)
            precision_list = precision_list[0] / len(user)
            out['recall_{}'.format(z)] = recall_list
            out['pre_{}'.format(z)] = precision_list


out.to_csv('pre_recall.csv')  # 输出结果
```

2、文本特征导入包 intro_paper.py

```
import numpy as np
import pandas as pd
```

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import linear_kernel


#计算相似度
def con_sim_matrix(data):
    #tf-idf 矩阵
    tfidf = TfidfVectorizer()
    tfidf_matrix = tfidf.fit_transform(data)
    #print(tfidf_matrix.toarray())
    #相似度矩阵
    cosine_sim = linear_kernel(tfidf_matrix, tfidf_matrix)
    return cosine_sim


#截取出输入的 doc_id 中论文的相似度列表
def con_sim(matrix, doc_id):
    li = []
    for x in doc_id:
        li.append(matrix[x-1])
    li2 = [np.mean(x) for x in np.transpose(li)]#计算平均的相似度
    #将自身相似度设置为 0，避免推荐自己
    for x in doc_id:
        li2[x-1] = 0
    return li2


#p 排序并标注秩
def sort_mat(sim, ascend=False):
    dd = pd.DataFrame({"x": sim, "y": range(len(sim))})
    dd = dd.sort_values(by="x", ascending=ascend)
    dd['z'] = range(len(sim))
    dd = dd.sort_values(by="y")
    return dd["z"]
```

3、论文-用户特征导入包 user_in.py

```
import pandas as pd
import numpy as np
from scipy.spatial.distance import pdist
from scipy.spatial.distance import squareform
from intro_paper import    sort_mat


# 运用矩阵分解的方法得到用户相似矩阵和论文相似矩阵
def dis_cf_matrix(data):
    row = np.array(data['user.id'])
    col = np.array(data['doc.id'])
    mat = np.zeros((np.max(row), np.max(col)))
    for i in range(len(row)):
        mat[row[i]-1, col[i]-1] = 1

    r = 4
    u, s, v = np.linalg.svd(mat)
    u, s, v = u[:, : r], s[:r], v[:r]
    sk = np.diag(np.sqrt(s))    # r*r
    uk = u @ sk    # m*r
    vk = sk @ v    # r*n

    dis_uk = pdist(uk, metric='euclidean')
    dis_vk = pdist(vk.T, metric='euclidean')
    dis_uk = squareform(dis_uk)
    dis_vk = squareform(dis_vk)
    return dis_uk, dis_vk
```

```
# 根据给出论文（读者）矩阵算出所有论文（读者）距离
def con_dis(dis_mat, doc_id):
    li = []
    for x in doc_id:
        li.append(dis_mat[x-1])
    li2 = [np.mean(x) for x in np.transpose(li)]  # 计算平均的距离
    # 将自身距离设置为 1，避免推荐自己
    for x in doc_id:
        li2[x-1] = 1
        return li2


# 通过用户名返回论文记录
def user_paper(user_id, data):
    user_data = data.loc[data['user.id'] == user_id]
    return user_data['doc.id']
def find_suit_user(num, data):
    user = []
    user_num = np.max(data['user.id'])
    for x in range(user_num):
        length = len(user_paper(x+1, data))
        if length > num:
            user.append(x+1)

    return user
```

4、论文得分表包 article_list.py

```
import pandas as pd
import numpy as np
from intro_paper import con_sim_matrix, con_sim, sort_mat
from user_in import dis_cf_matrix, con_dis
# 输入各种相关性矩阵，和要比对的文章，输出每篇文章的得分
def art_array(mat1, mat2, mat3, docs):
    x1 = con_sim(mat1, docs)
    x1 = sort_mat(x1)
    x2 = con_sim(mat2, docs)
    x2 = sort_mat(x2)
    x3 = con_dis(mat3, docs)
    x3 = sort_mat(x3, ascend=True)
    paper_list = pd.DataFrame({'id': range(len(x1)), 'x1': x1, 'x2': x2, 'x3': x3})
    return paper_list
```

5、推荐系统包 agent.py

```
import numpy as np
# 用户从推荐的论文中挑选喜欢的
def user(docs_r, docs_u):
    docs_s = []
    docs_u = list(docs_u)
    for id in docs_r:
        if id in docs_u:
            docs_s.append(id)
    # print(len(docs_s))
    return docs_s
# 通过选择的论文确定状态
def state(docs_s, paper_list):
    if docs_s:
        id = [x-1 for x in docs_s]
        score = [np.array(paper_list[x:x+1])[0][1:4] for x in id]
        ss = np.sum(score, axis=0)
```

```
            if ss[2] <= ss[1] & ss[1] <= ss[1]:
                s = 1
            elif ss[0] <= ss[2] & ss[2] <= ss[1]:
                s = 2
            elif ss[1] <= ss[0] & ss[0] <= ss[2]:
                s = 3
            elif ss[1] <= ss[2] & ss[2] <= ss[0]:
                s = 4
            elif ss[0] <= ss[1] & ss[1] <= ss[1]:
                s = 5
            else:
                s = 6
        else:
            s = 0
        return s
#  更新 Q 表
def renewQ(s, a, r, Q):
        Q[s, a] += r - 0.5
        return Q
#根据状态和 Q 表给出 action
def agent(s, Q, e=0.2):
        x = np.random.random()
        if x > e:
            a = np.argmax(Q[s])
        else:
            a = np.random.randint(0, 7)
        return a
#  根据 action 推荐
def recomender(a, action, docs, history, m=10):
        score = np.dot(docs.iloc[:, 1:], action[a])
        docs['score'] = score
        docsn = docs.sort_values(by='score')
        docs_r = docsn["id"]
        for x in history:
            docs_r = docs_r.drop(x-1)
        docs_r = [x+1 for x in docs_r[0:m]]
        return docs_r
```

6、其他推荐算法包 other.py

```
import numpy as np


def recomender(paperlist, colum, history, num):
        if colum == 'mix':
            score = np.dot(paperlist.iloc[:, 1:4], [1, 1, 1])
        else:
            score = paperlist['{}'.format(colum)]

        paperlist['score'] = score
        docsn = paperlist.sort_values(by='score')
        docs_r = docsn["id"]

        for x in history:
            docs_r = docs_r.drop(x - 1)
        docs_r = [x + 1 for x in docs_r[0:num]]
        return docs_r


def select_num(docs_r, docs_u):
        count = 0
        docs_u = list(docs_u)
        for id in docs_r:
            if id in docs_u:
```

```
                count = count + 1
        return count


def select_paper(docs_r, docs_u):
        paper = []
        docs_u = list(docs_u)
        for id in docs_r:
                if id in docs_u:
                        paper.append(id)
        return paper


def re_se(paperlist, colum, history, num, docs_u):
        docs_r = recomender(paperlist, colum, history, num)
        count = select_num(docs_r, docs_u)
        return count
```