doi：10.13413/j.cnki.jdxblxb.2020158

一种基于数据的GitHub项目个性化混合推荐方法

何锴琦[1]，马宇骁[2]，张　炎[3]，刘华虓[3]
（1. 吉林大学 研究生院，长春 130012；2. 美国东北大学 工程学院，美国 波士顿 02115；
3. 吉林大学 计算机科学与技术学院，长春 130012）

摘要：将两种传统基于内存的协同过滤方法相结合，提出一种基于数据的GitHub项目个性化混合推荐方法. 该方法不仅可动态地计算相似用户以保证推荐的个性化，且只用很小规模的相似用户便可得到与基于项目的方法相近的推荐质量；同时，该方法通过建立倒排表和利用K均值分类，在一定程度上解决了原方法在面对GitHub用户及项目数量级较大但交叉度较低的数据集时数据稀疏和冷启动问题. 通过与传统方法进行对比实验，验证了该方法的有效性和优越性.

关键词：数据分析；推荐系统；协同过滤技术；冷启动
中图分类号：TP311　　**文献标志码**：A　　**文章编号**：1671-5489(2020)06-0-08

A Data-Based Personalized Mixed Recommendation Method for GitHub Projects

HE Kaiqi[1], MA Yuxiao[2], ZHANG Yan[3], LIU Huaxiao[3]
(1. *School of Graduate, Jilin University, Changchun* 130012, *China*;
2. *College of Engineering, Northeastern University, Boston* 02115, *USA*;
3. *College of Computer Science and Technology, Jilin University, Changchun* 130012, *China*)

Abstract: We combined the traditional two memory-based collaborative-filtering methods and proposed a data-based personalized mixed recommendation method for GitHub projects. The method could not only calculate the similar users dynamically to ensure the personalized recommendation, but also obtain the recommendation quality comparable to the item-based method with only small scale of similar users. At the same time, the method solved the data sparsity and cold boot problems of the original method in the face of GitHub, a data set of users and projects of an order of magnitude but with low degree of crossover to some extent by establishing inverse table and using K-means classification. By comparing with the traditional method, we verified the effectiveness and superiority of the proposed method.

Keywords: data analysis; recommendation system; collaborative-filtering technology; cold boot

GitHub是目前知名的软件项目托管平台之一，在该平台上开发人员可通过账户共享自己创建

收稿日期：2020-06-08.

第一作者简介：何锴琦(1982—)，男，汉族，硕士，助理研究员，从事数据分析的研究，E-mail：hekaiqi@jlu.edu.cn. **通信作者简介**：刘华虓(1986—)，男，汉族，博士，副教授，博士生导师，从事软件工程的研究，E-mail：liuhuaxiao@jlu.edu.cn.

基金项目：吉林省自然科学基金(批准号：20190201193JC).

(Create)的项目、派生(Fork)他人的项目或关注(Watch)其他开发者的项目. 其提供了搜索功能,在不提供任何自动建议的情形下允许用户对平台中的项目进行手动检索. 传统的搜索引擎通常侧重于对检索内容的文本匹配,但对于开源软件代码库的用户,搜索合适的项目是一项困难的任务:一方面,用户通常很难用几个关键词准确地描述软件项目的特征,导致难以获得理想结果;此外,由于搜索结果较多,待选项目的质量通常参差不齐,只能依靠星级等有限评价体系;另一方面,由于不同项目的参与难度不同,会导致缺乏经验的新用户无从下手. 因此,如何利用数据分析方法,根据不同用户的专业方向和专业水平的差异,提供个性化项目推荐方法研究已成为目前的热门问题[1].

本文基于内存的协同过滤方法,提出一种基于数据的面向领域入门的GitHub项目个性化混合推荐方法,通过将基于用户和基于项目的协同过滤方法进行混合,并利用倒排表和K均值分类方法,在一定程度上解决了传统方法在面对GitHub用户及项目数量级较大但交叉度较低的数据集时数据稀疏和冷启动问题,并通过对比实验验证了本文方法的有效性.

1 基于内存的协同过滤方法

协同过滤方法是一组用于推荐系统设计和实现的方法,具有实现简单且推荐结果准确的特征. Tapestry[2]作为第一个手动协作过滤系统,允许用户通过参考其他用户的操作查询信息域中的项目. 文献[3]提出了自动协作过滤系统GroupLens,其能自动定位具有相关性的信息,并聚合为目标用户提供建议. 用户只需执行可观察到的操作,系统将收集到的信息与其他用户的相关操作结合,提供个性化的结果. 协同过滤技术中最常见的是基于内存的技术,基于内存的技术使用已有的评价数据计算用户或项目之间的相似度[4],根据计算出的相似度值做出推荐,其又可分为基于用户的方法和基于项目的方法两种.

基于用户的协同过滤算法(user-based collaborative filtering)是目前应用较多的算法. 该算法通过计算目标用户与邻居用户之间的相似性,根据邻居用户对某个资源的评分,预测目标用户的评分,预测评分最高的若干项目作为对用户的推荐呈现给目标用户. 其核心思想是计算用户间的相似性,寻找与用户相似度最高的邻居. 而基于项目的协同过滤算法(item-based collaborative filtering)则关注目标用户评分的项目与其邻居项目的相似性,将预测评分高的若干项目作为对用户的推荐呈现给目标用户.

1.1 基于用户的协同过滤方法

对于目标用户u,基于用户的协同过滤方法首先在用户群中找到与用户u存在相似性的用户,然后通过对其感兴趣的项目进行筛选,结合他们对某一项目r的评分预测用户u对该项目的评分,从而得到用户u对未知项目的评分排序列表,进而达到推荐目的. 算法步骤如下.

1) 将获取的数据集进行预处理,为不同操作赋予不同的分数权重,然后利用处理好的数据集构建用户-项目矩阵. 结果表示为一个$m\times n$的用户评价矩阵$\boldsymbol{M}$,其中m是用户数,n是项目数,$M[i,j]$表示第i个用户对第j个项目的操作;若用户未对某个项目进行过操作,则记为0分. 表1为用户-项目矩阵的一个示例.

表1 用户-项目矩阵示例

Table 1 Example of user-item matrix

用户	r_1	r_2	…	r_{500}	r_{501}	…
u_1	0	1	…	5	3	…
u_2	3	5	…	0	0	…
⋮	⋮	⋮	⋮	⋮	⋮	⋮
u_{500}	1	1	…	1	0	…
u_{501}	3	0	…	3	1	…
⋮	⋮	⋮	⋮	⋮	⋮	⋮

2) 得到用户-项目矩阵后,要完成对目标用户最近邻居的查找. 通过计算目标用户与其他用户之间的相似度,得到与目标用户最近的邻居集. 假设$R(u)$为用户u进行过操作的项目集合,$R(v)$为用户v进行过操作的项目集合,如果将用户-项目矩阵中的每行记录视为一个向量,则在推荐问题中计算u

与 v 的相似度常用 Jaccard 相似度算法[5]和余弦相似度算法[6]，计算公式分别为

$$S_{u,v}=\frac{|R(u)\cap R(v)|}{|R(u)\cup R(v)|},\tag{1}$$

$$S_{u,v}=\frac{|R(u)\cap R(v)|}{\sqrt{|R(u)|\times|R(v)|}},\tag{2}$$

Jaccard 相似度算法用于计算两个集合之间的相似程度，但集合中的元素取值是二元的，只能是 0 或 1；而余弦相似度算法在这方面则没有限制.

3）先从已计算完相似度的用户集中筛选出与目标用户 u 最相似的 k 个用户，用集合 $S(u,k)$表示，其中 k 是预设的相似用户窗口尺寸. 然后提取 S 中用户的所有评分项目，并删除目标用户 u 进行过评分的项目，将保留下的项目视为候选项目，对每个候选项目 r，用户对其预测评分(感兴趣程度)为

$$P(u,r)=\sum_{S(u,k)}S_{u,v}\times RT_{v,r},\tag{3}$$

其中 $RT_{v,r}$表示用户 v 对项目 r 的评分. 最后根据预测的目标用户评分结果进行排序，取前 N 个作为最终的推荐结果.

1.2　基于项目的协同过滤方法

与基于用户的协同过滤方法相反，基于项目的协同过滤方法[7]通过计算项目之间的相似度找到最相似的项目，直接执行协同过滤. 该算法步骤如下：

1）对于每个项目，算法通过使用余弦相似度算法计算相似度，得到 k 个与其最相似的项目，k 也称为邻居数；

2）该算法从 k 个最相似的项目中删除待推荐用户已评分(进行过操作)的项目集合 R，即删除用户已经创建、派生或关注的所有存储库，将结果保存在集合 U 中；

3）计算集合 U 和集合 R 中每个项目之间的相似度，即计算用户已评分的项目与其 k 个最相似项目之间的相似度，结果是一个按相似度递减排序的项目列表.

2　混合推荐方法设计

通过对两种基于内存的协同过滤方法比较可见：基于用户的协同过滤方法具有易实现、准确率高、移植性强等特点，但随着用户量的增加，推荐过程的计算量呈线性增加，对于千万级的用户和项目，存在严重的可扩展性问题；而基于项目的协同过滤方法则可通过建立以项目为基础的推荐模型，解决高计算成本的问题，且在多数情形下推荐结果甚至优于传统的基于用户的协同过滤方法. 但这两种方法也存在数据稀疏、高维度、冷启动等问题；此外，在推荐结果上，基于用户的方法偏向于推荐流行度高的项目，而基于项目的方法则偏向于推荐更具个性化的项目.

基于上述分析，本文将两种基于内存的协同过滤方法相结合，提出一种混合推荐方法，不仅可动态计算相似用户以保证推荐的个性化，而且只用很小规模的相似用户便可得到与基于项目方法相近似的推荐质量. 本文混合推荐方法如下：

1）利用基于用户协同过滤方法中计算用户相似度的方法，计算目标用户 u 与其他所有用户的相似性，确定 u 的邻居用户. 可采用两种方法：一是预先确定相似度阈值，与目标用户相似度大于该阈值的用户作为其邻居；二是预先确定邻居数 k，选择相似度最高的前 k 个用户作为邻居. 本文使用第二种方法确定目标用户的邻居.

2）根据上述确定的邻居用户，利用基于项目协同过滤的方法进行预测评分及项目推荐，即通过计算项目之间的相似度，找出与目标用户及其邻居已评分项目中的相似项目，并根据相似度顺序进行 Top-N 推荐.

通过对两种推荐方法的结合，使推荐结果的流行度与个性化兼备，且降低了推荐成本，提高了推荐效率.

2.1　数据稀疏性问题

在协作过滤技术中，通常需要构建用户-项目矩阵，其中每个条目都是未知值或用户对项目的评

分. 本文在构建用户-项目矩阵中的用户评分建模时考虑用户对项目的 Create,Fork 和 Watch 三个操作，并根据操作中所隐含的用户与项目的关联程度，为三种操作指定评分值 5,3 和 1，即如果开发人员是该代码库的创建者，则其评分值为 5；如果开发人员派生了该代码库，则其评分值为 3；如果开发人员关注了该代码库，则其评分值为 1，否则为 0(未知评分).

定义 1 设 $U=\{u_1,u_2,\cdots,u_n\}$ 是所有用户的集合，$R=\{R_1,R_2,\cdots,R_n\}$ 是所有项目的集合，则用户-项目矩阵(开发人员-代码库矩阵)为 $\boldsymbol{M}_{n\times m}$，其中，

$$M_{i,j}=\begin{cases}5, & u_i \text{ 创建(Create)} r_j,\\ 3, & u_i \text{ 派生(Fork)} r_j,\\ 1, & u_i \text{ 关注(Watch)} r_j,\\ 0, & \text{其他}.\end{cases}$$

数据稀疏是指构建的用户-项目矩阵多数项均为 0，导致当数据量达到一定规模时，推荐过程将占用大量的内存且运行效率较低. 下面用一个示例直观地描述该问题. 假设目前共有 4 个用户(A,B,C,D)和 5 个项目(a,b,c,d,e)，用户与项目的关系如图 1 所示. 若参照图 1 的结构运用相似度公式计算每个用户对之间的相似度，如 $S_{A,B}=1/\sqrt{6}$，$S_{A,D}=1/3$ 等，该方法的时间复杂度为 $O(|U|\times|U|)$，当达到一定数量级时，将非常耗时且效率较低. 此外，用户-项目矩阵通常是一个稀疏矩阵，一个用户能接触到的项目有限，从而导致在多数情形下 $R(u)\cap R(v)=\varnothing$，其中 $R(u)$ 为用户 u 进行过操作的项目集合，$R(v)$ 为用户 v 进行过操作的项目集合. 因此，本文算法先计算 $R(u)\cap R(v)\neq\varnothing$ 的用户，再用余弦相似度算法进行相似计算. 将用户-项目矩阵进行倒排，建立项目-用户倒排表，如图 2 所示.

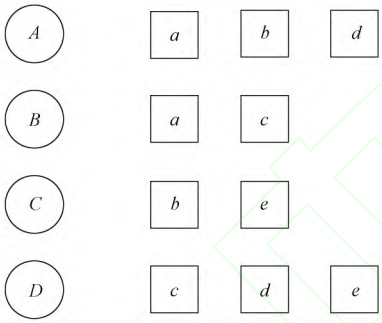

图 1 用户与项目的关系

Fig. 1 Relationship between users and items

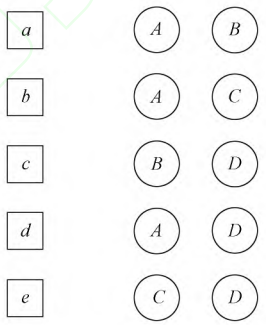

图 2 项目-用户倒排表

Fig. 2 Inverse table of item-user

设相似度矩阵为 $C[u][v]$，令 $C[u][v]=R(u)\cap R(v)$，即在倒排表中假设用户 u 和用户 v 同时属于倒排表中 K 个项目对应的用户列表，则有 $C[u][v]=K$. 如图 1 所示，只有在项目 a 对应的用户列表中同时出现了用户 A 和用户 B，则在矩阵中将 $C[A][B]$ 赋值为 1. 综合图 2 的项目-用户倒排表，相似度矩阵赋值如图 3 所示. 用余弦相似度算法对相似度矩阵进行优化处理，可得如图 4 所示的相似度矩阵，从而降低了数据的稀疏性.

2.2 冷启动问题

当期望以用户或项目的历史行为作为研究对象时，不可避免地会遇到冷启动问题，即当用户或项目作为新数据进入数据库时，其历史行为集为空集，使得研究方法无法应用. 在推荐问题上，冷启动问题表现为：对新用户推荐结果的合理性和有效性无法得到保障，以及新项目无法被推荐给用户. 本文对前者进行一定程度的缓解，将分类结果作为项目的一个属性留存，在系统对领域入门的用户进行推荐时，将参考该属性对其做出推荐，步骤如下：

1) 将项目的文本信息矢量化，将其转换为高维度向量，同时，利用词频-逆文件(TF-IDF)频率方法，在矢量化过程中计算各词语的 TF-IDF 值，以便后续分析与处理；

2) 按照提前设置好的类别数 K，将 K 均值聚类算法应用到步骤 1)中的矢量结果中，不断运行直至聚类结束；

	A	B	C	D
A	0	1	1	1
B	1	0	0	1
C	1	0	0	1
D	1	1	1	0

图 3　相似度矩阵赋值

Fig. 3　Similarity matrix assignment

	A	B	C	D
A	0	$\frac{1}{\sqrt{3\times2}}$	$\frac{1}{\sqrt{3\times2}}$	$\frac{1}{\sqrt{3\times3}}$
B	$\frac{1}{\sqrt{3\times2}}$	0	0	$\frac{1}{\sqrt{3\times2}}$
C	$\frac{1}{\sqrt{3\times2}}$	0	0	$\frac{1}{\sqrt{3\times2}}$
D	$\frac{1}{\sqrt{3\times3}}$	$\frac{1}{\sqrt{3\times2}}$	$\frac{1}{\sqrt{3\times2}}$	0

图 4　余弦相似度计算后的相似度矩阵

Fig. 4　Similarity matrix after cosine similarity calculation

3）输出每个项目的所属类别结果，并输出每个类的关键词信息.

本文改进方法的思想：当用户在 GitHub 中拥有一定量对项目的操作历史后，推荐系统可根据行为历史对用户作出推荐；但当用户行为历史不足时，如领域入门的开发者，期望通过对其仅有的行为历史得到一些项目信息，并为其推荐拥有相似信息的项目，使推荐结果尽可能合理、有效，从而在一定程度上缓解协同过滤方法的冷启动问题.

本文选取的项目信息是项目文本信息. 在 GitHub 中项目具有多种文本信息，如名称、简介、代码、ReadMe 文件等，从中选取最能快速区分项目间差异的项目名称、项目简介，并遵循用户友好的原则，同时考虑项目的编程语言种类因素. 本文用 K 均值聚类算法对项目文本信息进行分析，确定其之间的相似关系. 在将文本信息矢量化的同时，直接利用词频-逆文件频率方法[8]对矩阵进行优化.

3　实　验

3.1　数据收集

为方便研究人员对 GitHub 进行研究，平台开放了 GitHub rest API，用以提供 GitHub 的用户及项目数据. 但 GitHub 的数据量非常庞大，且 5 000 个/h 请求的 API 速率限制使完整下载数据几乎是不可能的. 为鼓励研究者对 GitHub 数据进行更深入地挖掘，Gousios 等[9-11]创建了 GHTorrent，本质上是 GitHub rest API 所提供数据的一个可伸缩、可查询的离线镜像.

本文选择最新的完整数据包，该数据包中包含了目前 GitHub 各类数据，包括用户、项目、提交等. 然后将整个数据包导入到 MySql 数据库系统中，并对其进行如下预处理：

1）数据集中共有 121 725 743 个项目，将其均匀分为 50 档，每档随机抽取 100 个项目，共得到 5 000 个项目；

2）从 2019-01-01—2019-05-31 的相关数据中截取 5 000 个项目，对这些项目最近 5 个月的数据进行研究；

3）从数据集中提取出所有对 5 000 个项目进行过创建(Create)、派生(Fork)或关注(Watch)操作的用户；

4）对项目和用户进行真实性和存在性的筛选，以确保所有项目和用户是真实的且未注销或删除；

5）以用户的操作量作为筛选条件，以 10 作为筛选阈值，删除对项目操作数低于 10 次的用户；

6）最后将用户和项目的行为关系进行格式化，使其能被推荐系统成功读取.

经过上述预处理，本文共获得 10 742 名 GitHub 用户面向 4 879 个 GitHub 项目的创建、派生和关注操作数据共计 195 850 条. 此外，本文还从 GHTorrent 数据集中获取到实验所用项目的文本信息，包括项目名称、项目简介和编程语言等，并对其进行如下预处理：

1）分别得到 4 879 个候选项目的名称、简介和所用编程语言(占最大比例)，并将三者在文本层面上进行连接；

2）删除文本中的标点符号、特殊字符及乱码等干扰信息；

3) 将项目编号与对应文本进行格式化，使其能被推荐系统成功读取.

3.2 参数设置

下面通过控制变量对比性实验的方法对改进方法的参数进行优化. 首先对混合推荐方法的邻居用户数、邻居项目数和推荐项目数分别进行对比实验. 选取推荐项目数为5，邻居项目数为15，取邻居用户数10～20，观察精确率和召回率指标的变化，结果列于表2. 由表2可见，指标随邻居用户数的增多而提高，为了兼顾推荐效率、覆盖率和流行度等其他变量，本文将邻居用户数设为15. 选取推荐项目数为5，邻居用户数为15，取邻居项目数10～20，观察其各项指标变化，结果列于表3. 观察结果与邻居用户数类似，因此同理将邻居项目数设为15. 选取邻居用户数为15，邻居项目数为15，取推荐项目1～11，观察其各项指标的变化，结果列于表4. 由表4可见，随着推荐项目数的增加，精确率迅速下降，而召回率迅速提升，因此本文取均值，将推荐项目数设为5.

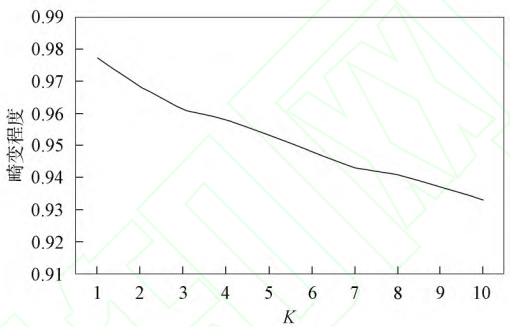

图5 K值对数据畸变程度的影响

Fig. 5 Influence of K values on degree of data distortion

下面对缓解冷启动问题时K均值聚类算法的K值进行优化. 将K值取1～10，并分别计算每个K值对应的畸变程度，即每个类心与其内部数据成员位置距离平方和的平均值，若类内成员彼此间越紧凑则类的畸变程度越小，反之，若类内成员彼此间越分散则类的畸变程度越大，实验结果如图5所示. 由图5可见，畸变程度随聚类数K的增大而降低，但若K值过小，聚类则失去了实际意义，无法为用户推荐个性化强的项目，因此本文取$K=6$，期望能兼顾项目的相似性与独特性.

表2 邻居用户数对精确率和召回率的影响

Table 2 Influence of neighbor user number on precision and recall

邻居用户数	10	11	12	13	14	15	16	17	18	19	20
精确率	0.106 2	0.106 9	0.107 3	0.107 7	0.108 0	0.108 9	0.109 2	0.109 9	0.110 1	0.110 9	0.111 4
召回率	0.102 2	0.103 0	0.103 9	0.104 2	0.104 9	0.105 5	0.106 2	0.107 0	0.107 3	0.107 9	0.108 3

表3 邻居项目数对精确率和召回率的影响

Table 3 Influence of neighbor item number on precision and recall

邻居项目数	10	11	12	13	14	15	16	17	18	19	20
精确率	0.106 7	0.107 2	0.107 7	0.107 9	0.108 0	0.108 3	0.109 2	0.110 0	0.110 4	0.110 6	0.111 3
召回率	0.101 4	0.101 9	0.102 4	0.102 6	0.102 6	0.102 9	0.103 8	0.104 6	0.104 9	0.105 1	0.105 7

表4 推荐项目数对精确率和召回率的影响

Table 4 Influence of recommendation number on precision and recall

推荐项目数	1	2	3	4	5	6	7	8	9	10	11
精确率	0.170 7	0.145 9	0.127 8	0.113 5	0.104 1	0.096 9	0.091 2	0.086 4	0.082 1	0.078 0	0.073 6
召回率	0.032 4	0.055 3	0.072 8	0.086 1	0.098 7	0.110 3	0.121 1	0.131 1	0.140 1	0.147 9	0.154 4

3.3 实验结果及评价

将上述参数应用到改进后的推荐方法中，并将改进方法与传统的两种基于内存的协同过滤方法进行对比，以验证本文方法的可行性. 本文对基于改进后的推荐系统进行了实际操作，针对某个真实用户进行推荐结果分析，使实验结果更直观.

首先将改进方法与传统方法应用到经过预处理的GitHub数据集上进行评估，实验结果如图6所示. 由图6可见：改进方法在精确率和召回率指标上均高于两种传统方法，改进方法的覆盖率和流行度指标均介于二者之间；改进方法解决了数据稀疏性问题，并在一定程度上缓解了冷启动问题，提高了整体推荐性能.

下面针对实验数据集中的 8088 号用户“Ajeo”从推荐系统中获取的推荐结果进行分析，其 GitHub 主页如图 7 所示. “Ajeo”在本文实验数据集中共有 14 条评分数据，其中包括派生了自然语言处理课程“oxford-cs-deepnlp-2017/lectures”，关注了数据科学手册“jakevdp/PythonDataScienceHandbook”、著名的自然语言处理工具“google-research/bert”和深度学习论文“terryum/awesome-deep-learning-papers”等；本文推荐系统根据他的项目操作历史，为他个性化推荐了 5 个项目，结果列于表 5.

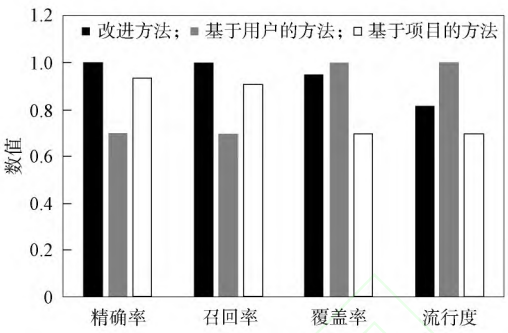

图 6　不同推荐方法在 4 个评价指标上的对比

Fig. 6　Comparison of different recommended methods on four evaluation indexes

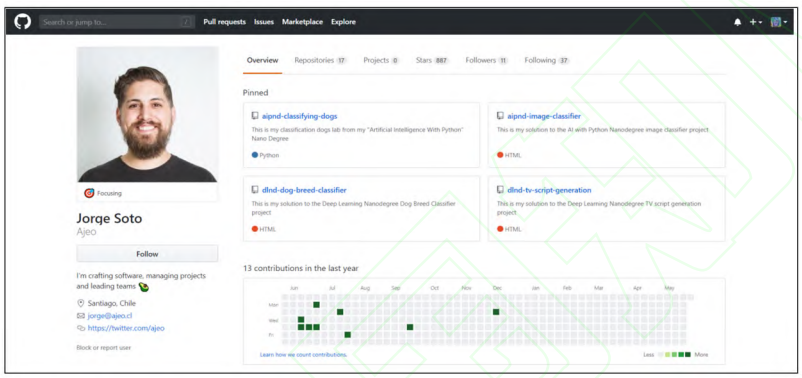

图 7　“Ajeo” GitHub 主页

Fig. 7　GitHub homepage of “Ajeo”

表 5　用户“Ajeo”获得的推荐结果

Table 5　Recommended results for user “Ajeo”

排序	项目所有者/项目名称	项目简介
1	LisaDziuba/Awesome-Design-Tools	设计工具插件
2	fastai/fastai	深度学习库
3	sebastianruder/NLP-Progress	自然语言处理工具
4	pytorch/pytorch	Python 神经网络
5	ageron/handson-ml	机器学习笔记

由表 5 可见，在 Top-5 推荐列表中，排在第一位的是一个设计工具的插件集，这是计算机从业者常用的工具，具有较高的流行度. 可以合理推测，与其相似的开发者可能很多人都在使用这个工具集. 而剩下的 4 个项目都与他在 GitHub 上的项目操作历史紧密相关，涉及到机器学习、自然语言处理等内容. 通过对用户“Ajeo”所得推荐结果的分析，可认为本文推荐系统具有较合理、有效的推荐能力.

下面在数据集中新加入一名用户“ILOVEJLU”，并令其派生一个 Android UI 库“wasabeef/awesome-android-ui”，然后使用推荐系统为其提供推荐结果，得到推荐结果列于表 6.

表 6　用户“ILOVEJLU”获得的推荐结果

Table 6　Recommended results for user “ILOVEJLU”

排序	项目所有者/项目名称	项目简介
1	Blankj/AndroidUtilCode	Android 工具集
2	Semantic-Org/Semantic-UI	UI 组件框架
3	nostra13/Android-Universal-Image-Loader	Android 图像库
4	bayandin/awesome-awesomeness	“Awesome”系列盘点
5	Trinea/android-open-project	Android 开源项目集

由表 6 可见，系统提供的推荐结果在文本信息层面与“ILOVEJLU”的项目操作历史相似度很高，

因而内容上也具有很强的相关性，对于新用户或领域入门的开发者具有一定的指导意义.

综上所述，针对GitHub平台现有项目推荐功能的缺陷，本文提出了一种基于数据的GitHub项目个性化混合推荐方法. 首先，对现有协同过滤方法进行了改进和优化，通过建立倒排表的方式大幅度降低了基于用户的协同过滤方法在计算用户相似度时的计算成本，并提高了运行效率，克服了协同过滤方法存在的数据稀疏性问题；其次，通过利用基于用户的协同过滤方法进行用户相似度计算，并利用基于项目的协同过滤方法进行项目推荐的方式，将两种基于内存的协同过滤方法相结合成为混合推荐方法，优化了推荐结果，兼顾了推荐结果的流行度与个性化，并保持了协同过滤推荐方法的优越性；再次，基于项目文本信息进行聚类，使评分历史不足的用户能利用仅有的项目历史获得较合理、有效的推荐，在一定程度上缓解了协同过滤方法存在的冷启动问题；最后对该推荐方法进行实证研究，将其应用到从GHTorrent获取的真实GitHub数据上，并通过实验对实验结果进行评估. 实验结果表明，在常见的几个评估推荐系统的评价指标上，本文方法与两种传统基于内存的协同过滤方法相比，都有较优越的表现，证实了该方法在面向领域入门的GitHub项目推荐问题上的可行性.

参 考 文 献

[1] SUN X B, XU W Y, XIA X, et al. Personalized Project Recommendation on GitHub [J]. Science China Information Sciences, 2018, 61(5): 1-14.

[2] GOLDBERG D, NICHOLS D A, OKI B M, et al. Using Collaborative Filtering to Weave an Information Tapestry [J]. Communications of the ACM, 1992, 35(12): 61-70.

[3] RESNICK P. GroupLens: An Open Architecture for Collaborative Filtering of Netnews [C]//Proceedings of the 1994 ACM Conference on Computer Supported Cooperative Work. New York: ACM, 1994: 175-186.

[4] ZHANG L X, ZOU Y Z, XIE B, et al. Recommending Relevant Projects via User Behaviour: An Exploratory Study on GitHub [C]//1st International Workshop on Crowd-Based Software Development Methods and Technologies (CrowdSoft 2014). New York: ACM, 2014: 25-30.

[5] 张晓琳，付英姿，褚培肖. 杰卡德相似系数在推荐系统中的应用 [J]. 计算机技术与发展，2015，25(4)：164-167. (ZHANG X L, FU Y Z, CHU P X. Application of Jaccard Similarity Coefficient in Recommder System [J]. Computer Technology and Development, 2015, 25(4): 164-167.)

[6] LIPKUS A H. A Proof of the Triangle Inequality for the Tanimoto Distance [J]. Journal of Mathematical Chemistry, 1999, 26(1): 263-265.

[7] SARWAR B. Item-Based Collaborative Filtering Recommendation Algorithms [C]//Proceedings International World Wide Web Conference. New York: ACM, 2001: 285-295.

[8] GERARD S, CHRISTOPHER B. Term-Weighting Approaches in Automatic Text Retrieval [J]. Information Processing and Management, 1988, 24(5): 513-523.

[9] GOUSIOS G, SPINELLIS D. GHTorrent: Github's Data from a Firehose [C]//9th International Working Conference on Mining Software Repositories. Piscataway, NJ: IEEE, 2012: 12-21.

[10] GOUSIOS G. The GHTorent Dataset and Tool Suite [C]//10th International Working Conference on Mining Software Repositories. Piscataway, NJ: IEEE, 2013: 233-236.

[11] GOUSIOS G, VASILESCU B, SEREBRENIK A, et al. Lean GHTorrent: GitHub Data on Demand [C]//11th International Working Conference on Mining Software Repositories. Piscataway, NJ: IEEE, 2014: 384-387.

(责任编辑：韩 啸)