

Job Seeking Helper

Hongming Wang, Yehe Fang, Xiaoyu Chen

ECE 657A Group2

Abstract. Going through thousands of job openings and picking out the a list of favorite jobs can be time-consuming. To reduce this kind of tedious and laborious work, we learn the basic knowledge of building a recommendation system and choose this area as the topic of the project. In this project, we focus on the task of designing and implementing the job recommendation systems. In particular, we use two approaches called content-based filtering and user-based collaborative filtering to implement the job recommendation systems. The experimental results show the effectiveness of our approaches. We also compare the performance metrics and analyze the results of two approaches.

Keywords: Job recommendation system, Content-based filtering, Collaborative filtering

1 Introduction

All members of our group will graduate this year. And we plan to start professional careers to survive and thrive in the society. Therefore, we are looking for jobs to get started in the near future. However, the problem is that every time we browse the employment-related Web such as LinkedIn, WaterlooWorks and Monster, we feel overwhelmed because enormous amount of recruiting information are beyond our ability to digest. Indeed, with the explosion of information, there are a great many job postings on the various recruiting web sites nowadays. Most of them are not related to our professional backgrounds thus need to be filtered out.

What we are really looking for are the jobs that either we are interested in or match our qualifications perfectly. To get a list of such jobs, typically we have two ways to interact with the pool of jobs, namely search and recommendation.

Regarding the process of search, when the user does a search, the search engine is not actually running through the database in the real time. Instead, the search engine caches the job postings in advance, which is called indexing. So when the user inputs the keywords of jobs, the search engines will use its own ranking algorithm to rank the jobs, which is called ranking. Fig. 1 shows the diagram of how search engines work to make responses.

What if the user does not have a clear image of what kind of keywords should be included in the job postings? This happens a lot, sometimes we get tired of guessing the keywords of our dream jobs, or we just want to refresh our minds so that we won't stick a fixed type of jobs. We want the system to behave

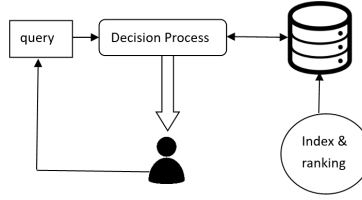


Fig. 1. Diagram of how the search engine works.

intelligently and serve as a helper to help us find the highly-matching jobs, which is the reason why we give the name “job seeking helper” to our project.

Fortunately, the recommendation system can do such assistance for us smartly. As Fig. 2 showed, based on the history of user activity, the recommendation system can access the database (similarly, this process can be done ahead of time) and generate the list of target jobs to return. This architecture can be used to give you a high-level overview of our project.

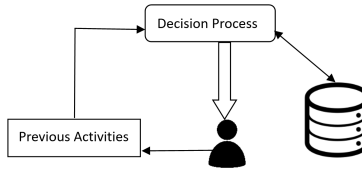


Fig. 2. Diagram of how the recommendation system works.

In [1], the researchers summarized the types of recommendation systems. The first one is editorial recommendation. It is the simplest type of recommendation system and relies on a lot of manual work on picking out the items to make recommendation. The second type performs some data aggregation to get global statistical metrics for recommendation, such as popularity-based recommendation. The third type can be tailored to individual tailors [6], which means it can make user-specific recommendation. We only focus on the third type. Moreover, we use two recommendation methods of the third type to implement our system, they are content-based recommendation and user-based collaborative filtering recommendation.

The rest of the paper is divided into six sections. In section 2, we review some papers to know about some basic algorithms of recommendation system and how to evaluate the system. Section 3 provides an overview of the project, including goals to achieve, methods to use and the dataset introduction. Section 4 shows the detail implementation of content-based recommendation. In section 5, we design and implement another method called collaborative filtering to make recommendation. In section 6, the experimental results of two methods are shown. We also analyzed the prons and cons for each method. Section 7

draws the conclusions of our project and points out some directions of future work.

2 Literature Review

2.1 Typical recommender algorithms

The recommendation algorithm decides the performance of a recommender system. The use of efficient and accurate recommendation algorithm is important for a system that will provide good and useful recommendation to its individual users.

In recent years, the most common used recommender algorithms are Content-based filtering (CB), Collaborative filtering (CF) and Hybrid filtering (HF). The CF can be divided into Model-based CF and Memory-based CF. Fig. 3 [3] shows the anatomy of the typical recommender algorithms.

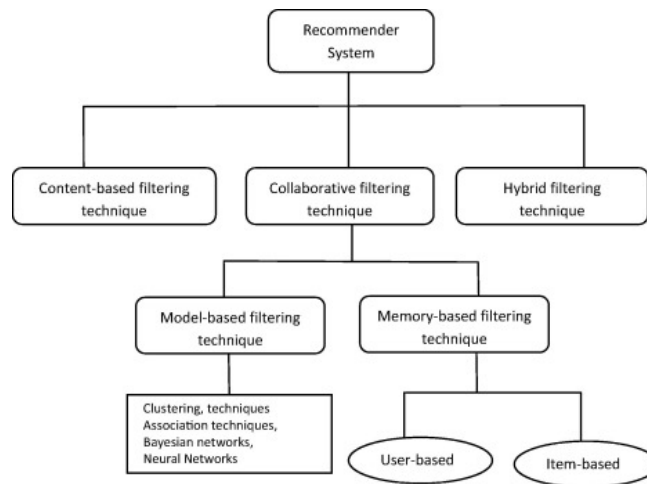


Fig. 3. Anatomy of different recommendation filtering algorithms.

2.2 Evaluation metrics for recommendation algorithms

To evaluate a recommender system, we can use different types of measurements, which can be accuracy or coverage. The type of metrics used depends on the type of recommender algorithms used. Accuracy is the proportion of the correct recommendations out of total possible recommendations, while coverage measures the proportion of objects in the search space that the system can provide [4]. Metrics for measuring the accuracy of recommendation filtering systems are divided into statistical and decision support accuracy metrics.

Statistical accuracy metrics: Evaluate accuracy of a filtering technique by comparing the numerical recommendation scores against the actual user ratings for the user-item pairs in the test dataset. Mean absolute error (MAE), root mean square error (RMSE) and correlation are commonly used as statistical accuracy measures [4]. MAE is the most popular and commonly used; it is a measure of deviation of recommendation from their true user-specific values. The lower the MAE, the more accurately the recommendation engine predicts user ratings.

Decision support accuracy metrics: evaluate how effective a prediction engine is at helping a user select high-quality items from the set of all items. In [5], researchers summarized that the most commonly used decision support accuracy metrics are reversal rate, weighted errors, receiver operating characteristics (ROC) sensitivity, precision recall curve (PRC), precision, recall and F-measure.

3 Project Description

In this section, we briefly clarify the task and goals of our project, methods to use and the dataset to deal with.

3.1 Project task

Our task is to implement a job recommendation system. This system will return a list of user-specific recommended jobs for the user who has the record of job applications in the dataset. Furthermore, the recommendation results are generated according to the information of applied jobs in the user's history or other similar users' applications.

3.2 Used Methods

We use two methods to implement the recommendation system, they are content-based recommendation and user-based collaborative filtering recommendation.

Content-based recommendation compares the similarity between the user profile and the content of a job to determine the strength of recommendation, while user-based collaborative filtering looks for users who share the similar application history patterns with the current user to make user-specific predictions.

3.3 Introduction to the dataset

The dataset we use is from a data challenge on *Kaggle* called "job recommendation challenge" [2]. The files we use including `users.tsv`, `jobs.tsv` and `apps.tsv`.

`users.tsv` contains information about the users. Each user is assigned a unique ID, which is stored in the column called "UserID". `jobs.tsv` contains information about job postings. Each job posting has its unique JobID, title, description, requirements, location information (state and city), start date and end date. `apps.tsv` contains information about applications made by users. Each application has recorded UserID, JobID and application date.

4 Content-based Recommendation

The general idea of content-based recommendation is to recommend jobs to a user \mathbf{x} which is similar to the jobs he or she applied previously. We notice that `jobs.tsv` contains a number of textual columns, such as title, description and requirements. We can capture the raw text of these fields and do some proper preprocessing to build job profiles and user profiles.

The next question is how can we tell that job j_A is similar to job j_B ? As we discussed before, we need to compare their title, description, requirements. And here we also want to address the importance of the job location. Generally speaking, users tends to apply jobs within a certain area to stay connected with family and friends. And most of the people do not like to spend long time commuting or moving in and out frequently. Therefore, taking into account of the job location can improve the recall of the recommendation system. We summarize the input and output in equation (1):

$$\text{Feature} = f(g(\text{Title}) + g(\text{Description}) + g(\text{Requirements}) + g(\text{State}) + g(\text{City})), \quad (1)$$

where *Feature* is a new job feature which combines several relevant job attributes and will be used for the model training later. $f(\cdot)$ denotes the rule of generating the *Feature*. $g(\cdot)$ denotes a sequence of preprocessing operations on the raw text.

4.1 Text preprocessing

In order to better understand the techniques we use in the phase of text preprocessing, let us take an example from the dataset. As showed in the figure below, in `jobs.tsv`, we take a look at the job j_1 whose `JobID = 1`, the raw text are not readable at all, and we can't apply any recommendation algorithm directly.

JobID	WindowID	Title	Description	Requirements	City	State
1		1 Security Engineer/Technical Lead	<p>Security Clearance Required: Top Secret</p> <p>Job Number: TMR-447</p> <p>Location of Job: Washington, DC</p> <p>TMR, Inc. is an Equal Employment Opportunity Company</p> <p>For more job opportunities with TMR, visit our website at http://www.tmrhq.com/</p> <p>Send Resumes to HR@tmrhq.com</p> <p>SUMMARY:</p> <p>Leads the customer's overall Cyber Security strategy, formalizes service offerings consisted with ITIL best practices, and provides design and architecture support</p> <p>Provide security design / architecture support for OIP's Security Division (ITSD)</p> <p>Leads the SECOPS team in the day to day OIP Security Operations support</p> <p>Provides direction when needed in a security incident or technical issues</p> <p>Works in concert with network operations on design / integration for best security posture</p> <p>Supports business development functions including Capture Management, Proposal Development and responses, and other initiatives to include conferences, trade shows, webinars, developing white papers and the like</p> <p>Identifies resources and mentors in-house talent to ensure TMR remains responsive to growing initiatives and contracts with qualified</p>	<p>SKILL SET:</p> <p>Network Security tools</p> <p>Webdefend Web Application Firewall (WAF), Cisco Routers, Fortigate 3800 Firewall series, Palo Alto 4000 firewall series, Cisco ASA 55x Firewall Platform, Cisco FWSM, SourceFire Defense Center, SourceFire IP Sensor Platform, BlueCoat SG Appliance, F5 BIG-IP (reverse proxy)</p> <p>Web Application tools:</p> <p>AppDetective, Fortify SCA, HP WebInspect, and the like</p> <p>Tenable Security Center, McAfee Foundstone scanner, Cain and Able, L0phtcrack - Password Cracker, Nessus Vulnerability Scanner, NMAP - Nmap, Port Scanner, and other scanning and vulnerability mapping tools</p> <p>DESIRABLE SKILLS:</p> <p>CISSP and/or related Certifications</p> <p>EDUCATION AND YEARS OF EXPERIENCE:</p> <p>BS Computer Science or related discipline; minimum of 8 years in IT Security; minimum 4 years in Senior/Lead position</p> <p>https://www.tmrhq.com/jobapplicationstep1.aspx</p>	Washington	DC

Fig. 4. j_1 's attributes in raw text.

The first step is to remove html tags, like '`<p>`' and '`</p>`'. To remove all characters that are enclosed in matching pairs of angle brackets, we use regular expression to match those tags and replace them with spaces.

The next step is to remove all other new line characters such as '\n', '\r', and a common character entity ' ', which is called non-breaking space.

The third step is to convert the string to lowercase. Imagine if not do so, the word “Security” with a capital letter ‘S’ will be identified as a different word rather than the original word “security”, because Python is case sensitive. After that, we can tokenize the string to get a list of words. Because we usually call each word a token in the field of natural language processing.

Then we filter out the stop words. Stop words are some of the most common and short function words, such as ‘the’ ‘is’ ‘at’ ‘which’ ‘in’ and so on. They contribute nothing towards the meaning of a job, so just simply get rid of them. Are we done at this step? Should we consider doing stemming? Our consideration is that although it reduces inflected words to their root forms, it could make the text harder to understand and interpret. It is really a trade-off between analyzability and readability. Doing lemmatization would be a better choice, but it increases the time of processing heavily, therefore it is also deprecated in practice.

In the final step, we gather processed text of all relevant fields, and concatenate them in a specific rule to get the feature for each job, see equation (2):

$$f(\cdot) = g(\text{Title}) * w_{\text{title}} + g(\text{Description}) + g(\text{Requirements}) + g(\text{State}) * w_{\text{state}} + g(\text{City}) * w_{\text{city}}, \quad (2)$$

where $w_{\text{title}} = (1 + \text{length}/100)$, $w_{\text{state}} = (1 + \text{length}/250)$, $w_{\text{city}} = (1 + \text{length}/150)$ and $\text{length} = \text{len}(g(\text{Description})) + \text{len}(g(\text{Requirements}))$. This rule shows how many times the text of a certain field is repeated, the more times the more important. You can see according to our setting, the job title is the most important field, and then the city and the state.

4.2 Data preprocessing

Data preprocessing is much easier than text preprocessing. Since many job postings do not have complete information, or just unavailable to apply, so few users applied those jobs. We only retain the jobs that are applied more than once.

To avoid the cold start problem, we only focus on the users who applied more than 5 jobs in the training time. Otherwise it is hard to provide reliable recommendation.

Regarding the splitting of training and testing data, one way is to use cross-validation, it guarantees the generalization of the data. And we go further to use a more robust way. It splits data according to a reference date. This way can ensure the behaviors of testing always happen after the training time. According to our implementation, we set ‘4/10/2012’ as the reference date. Applications before are used as the training data, applications after are used as the testing data.

4.3 Model training

The goal of this phase is to create a job profile for each job using the 'Feature' attribute generated before. A job's profile contains a set of important words with values showing how important the corresponding word is in the job's feature. From a mathematical angle, the so-called job profile is actually a unit vector.

An algorithm called TF-IDF (short for term frequency-inverse document frequency) is used to determine the importance of a word in a job's feature. The term frequency TF_{ij} for word i in a job's Feature (document) j is just the number of times the word i appears in Feature (document) j divided by the maximum number of times that same word appears in any job's Feature, which is described as $TF_{ij} = \frac{f_{ij}}{\max_k f_{kj}}$.

Intuitively, term frequency means the more a word appears in a document, the more important it is to that document. But for some common words in this task, such as "resume" and "skill", how can we weaken those words and address the words like "security" which are able to truly contribute to the feature of a job? The answer is to introduce the concept of inverse document frequency IDF. Let n_i denotes the number of documents that mention the word i , and let N be the total number of documents in the training set, we have $IDF_i = \log \frac{N}{n_i}$. Combine TF_{ij} and IDF_i together, we can calculate the score of TF-IDF using $w_{ij} = TF_{ij} \times IDF_i$. Therefore, the job profile can be rewritten as

Job Profile = words with high TF-IDF scores + corresponding TF-IDF scores.

We specify the capacity of the set of the most important words to keep to be 5000. We also allow this set to contain not only unigrams (single words) but also bigrams (two-word phrases) such as "sales manager". The trained model we get is a sparse matrix with the shape of (46739, 5000), where 46739 is the number of jobs in the dataset.

After we get all the job profiles in hand, we can then create the profile for every user. A user profile is also a unit vector. We sum up all the job profiles that the user \mathbf{x} had applied in the training time, and then average the sum to get \mathbf{x} 's user profile x_p . To make a prediction, we need to calculate the cosine similarity between the user profile x_p and a job profile \mathbf{j} . Higher values indicates higher probability that user \mathbf{x} will be interested in that job.

Let us look at an example of the content-based recommendation results. A user whose UserID is 8189 had applied three jobs in the period of training, which are "Online Marketing Manager" (JobID = 406104), "Canon Insights Program - Social Media Internship" (JobID = 758462), and "Event Coordinator - Sales & Marketing" (JobID = 472338). A list of five jobs with the highest strength of recommendation (cosine similarity) are showed in Fig. 5, we can see the recommendation results are similar to the jobs he or she had applied.

5 User-based Collaborative Filtering Recommendation

In the previous sections we talked about the content-based recommendation, which predicts user's applications using jobs similar to that user's previous choice

	recStrength	JobID	Title
0	0.642109	563167	Administrative Assistant/ Research Assistant
1	0.632758	262678	Executive Assistant/ Marketing Assistant
2	0.611017	862963	Recent College Graduate/Administrative Assistant
3	0.611017	563172	Recent College Graduate/ Administrative Assistant
4	0.593569	562659	Administrative Assistant/ Client Services

Fig. 5. Example of content-based recommendation results.

based on the relevant fields of jobs. It did not use the information of users to provide recommendations. In this part, we will build a user-based recommendation system that uses the profile of the given user as well as its similar users to recommend jobs. The basic idea is people who agreed in the past are likely to agree again. The basic idea is people who agreed in the past are likely to agree

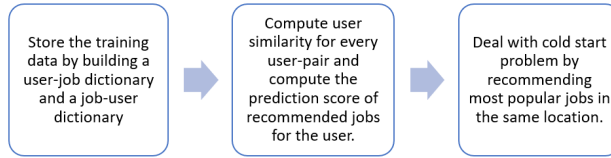


Fig. 6. Experimental setup of collaborative filtering.

again. To predict a user's opinion for an job, use the opinion of similar users. The procedures of the system will be given below.

5.1 Implementations details

First it takes the training data as the input. The task is to predict the applications made by the user based on the training data, which is the same as used in the content-based recommendation, so the training data includes users with more than 5 applications. Then we compute the user similarity using Jaccard index. There are many similarity measures that can do the works. Such as Euclidean Distance, Manhattan Distance, Cosine Similarity, etc. The intuition of using Jaccard index is that the similarity between users is decided by looking at their overlap in opinions. The Jaccard similarity index between user u and user v is $J(u, v)$ is shown in equation (3).

$$J(u, v) = \frac{|u \cap v|}{|u \cup v|} \quad (3)$$

Generally, we need to make a user similarity matrix [7] and do some computations using that matrix in order to do recommendation. However, the user application table is a sparse matrix see Fig. 7.










		job 1	job 2	job 3	job 4	job m
							
user 1		1	1	0	0		0
user 2		0	1	1	0	1
user 3		0	0	1	1		0
....				
user n		0	0	1	1	0

Fig. 7. Example of user application matrix.

Since we only need to know the what jobs does the user apply (positions of the value 1 in the matrix), we can simplify the calculation by using two dictionaries. In detail, the system iterates over all applications, and create two dictionaries for user-job and job-user key-value pairs. In this case, we don't need to do complex matrix multiplications. Everytime we need to recommend jobs to a user, the system computes weighted sum of different similar users rating on job i. The prediction score $P_{u,i}$ is given by:

$$P_{u,i} = \frac{\sum_{v=1}^n (r_{v,i} * s_{u,v})}{\sum_{v=1}^n (s_{u,v})}, \quad (4)$$

where $s_{u,v}$ is the Jaccard similarity index between user u and user v , and $r_{v,i}$ is the rating of user v on job i . After we get prediction scores of all recommended jobs, we sort the jobs by the prediction score in descending order and choose the top N jobs as the final recommendation. The system may encounter a cold start problem that there are no similar users for the job seeker, which results in no jobs recommended at all. This is possible if the job seeker only applied unpopular jobs previously that no other users applied. In the last step the system deals with this problem by recommending top N popular jobs (applied most times by all other users) in the same location where the job seeker lives.

6 Experimental Results and Analysis

In this section, we evaluated two approaches of recommendation system by measuring Top-N recall metrics. This evaluation method evaluates the accuracy of the top N recommendations returned to the user, comparing to the jobs that the user has applied during the period of test. The details of this method are given below:

```

for each user do
  for each job  $j$  the user had applied in test set do
    Randomly sample 100 jobs that the user had not applied;
    Combine the current job  $j$  with the sample list;
    Let the recommendation algorithm generates a ranked list for this
    combined job set;
    Get the ranking of  $j$  and compute Top-N recall metrics;
  end
end
Aggregate metrics and get the global metrics for the recommendation algorithm;
end

```

Algorithm 1: Evaluation method used in this section.

We visualize the experimental data to see how the performance metrics of two approaches change as the N increases, as is showed in Fig. 8. Each data point in the figure denotes a metric of Recall@ N , which evaluates whether the applied jobs in the test set is among the top N jobs in the ranked list of 101 ($100 + 1$) jobs with different recommendation strength. We choose N to be [3, 5, 10, 15, 20, 25] to see how the two recommendation algorithms perform with different acceptable ranges of ranking.

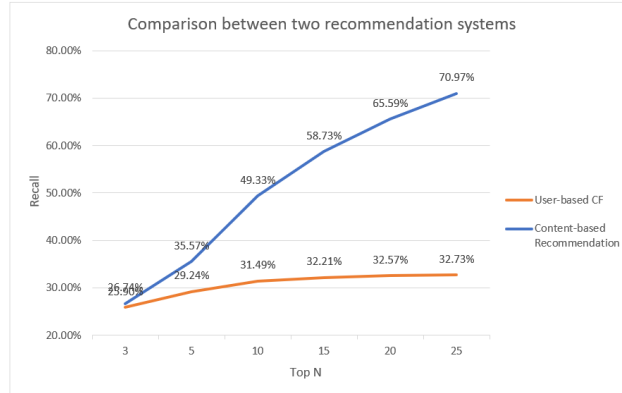


Fig. 8. Curves of Top-N recall metrics of two recommendation approaches.

We notice when N is relatively small, the metrics of two algorithms are close. Recall@3 is 25.90% for user-based CF, and 26.74% for content-based recommen-

dation, which means there is a rate of one quarter that the test applications are ranked top 3 out of 100 random jobs. As N increases, the performance differences of two methods become significant. We can see the performance of content-based recommendation is much better than user-based collaborative filtering, which demonstrates the validity of content-based recommendation.

As we can see, the performance of content-based recommendation does pretty well in the experiment. It reaches 0.3557 on Recall@5, which means that about 36% of applied jobs in the test set are ranked by this algorithm among the top-5 jobs out of a list of 100 random jobs. Moreover, as the parameter N increases, the corresponding metrics grow rapidly, Recall@10 is nearly 50%.

Besides the great performance, the biggest pro of content-recommendation approach is that you do not need data about other users in order to make recommendations to a specific user. Another good thing about content-based recommendation is that it is able to recommend new and unpopular jobs. When a new job is posted online, we do not need any applications to build the job profile. Because the job profile depends entirely on the features of the job. Thirdly, it can provide an explanation to a user for why a certain job was recommended. With regard to cons, the most serious problem is that it is difficult to find a proper feature to create the job profile. In our implementation, we design a rule of how to concatenate all relevant fields to get the Feature for the job profile. But sometimes it just does not work so well.

On the other hand, the performance of user-based recommendation is not good enough. There is a prediction recall upper bound for user-based recommendation when we increase N ($N \leq 25$), which is around 33%. It means even N is chosen to be 25, only a rate of 33% of the test applications can be in the top quarter according to the recommendation strength given by the algorithm of user-based recommendation. One reason to explain this poor performance is because user community for this dataset is not connected closely, we only get a 33% prediction accuracy from the similar users. If a job is not popular and only a few users apply, it would not be recommended. Another reason is that this dataset has too many jobs. In contrast, there are not enough users. The user-based collaborative filtering does not have a good performance in a relatively large item (job) space. In contrast, the content-based recommendation predicts similar jobs based on relevant fields of jobs that the user previously applied, and is ensured not be affected by the size of the dataset.

7 Conclusion

In this paper, we design and implement two approaches of job recommendation systems. They are called content-based recommendation and user-based recommendation respectively. Content-based recommendation suggests jobs that have similar content information to the profiles of users. User-based recommendation finds similar users who share the taste of jobs with the target user and recommends jobs based on what the similar users had applied. Then we evaluate these two approaches on a real-world dataset. The experimental metrics

show that content-based recommendation have a great on job prediction, while the performance of user-based recommendation is limited. Detailed analysis and comparison is addressed to explain the results.

In the future, we plan to implement item-based CF recommendation to break through the limit of performance of user-based CF recommendation. Since in most the of cases, items (jobs) are inherently simpler than the users, while the tastes of users can be varied and changing from time to time, which makes the prediction harder. Moreover, to optimize the content-based recommendation, we plan to take advantage of users' professional backgrounds and employment history to provide more accurate recommendation.

References

1. Gigimol, S., and Sincy John. "A Survey on Different Types of Recommendation Systems." *Engineering and Science* 1.4 (2016): 111-113.
2. Kaggle Inc. (2013, June). Job recommendation challenge. Retrieved from <https://www.kaggle.com/c/job-recommendation>.
3. Isinkaye, F. O., Folajimi, Y. O., & Ojokoh, B. A. (2015). Recommendation systems: Principles, methods and evaluation. *Egyptian Informatics Journal*, 16(3), 261-273.
4. Sarwar, B., Karypis, G., Konstan, J., & Riedl, J. (2001, April). Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th international conference on World Wide Web* (pp. 285-295). ACM.
5. Sarwar, B. M., Konstan, J. A., Borchers, A., Herlocker, J., Miller, B., & Riedl, J. (1998, November). Using filtering agents to improve prediction quality in the grouplens research collaborative filtering system. In *Proceedings of the 1998 ACM conference on Computer supported cooperative work* (pp. 345-354). ACM.
6. Yao, G., & Cai, L. *User-Based and Item-Based Collaborative Filtering Recommendation Algorithms Design*. University of California, San Diego.
7. Saranya, K. G., Sadasivam, G. S., & Chandralekha, M. (2016). Performance comparison of different similarity measures for collaborative filtering technique. *Indian Journal of Science and Technology*, 9(29).
8. Jain, S., Grover, A., Thakur, P. S., & Choudhary, S. K. (2015, May). Trends, problems and solutions of recommender system. In *Computing, Communication & Automation (ICCCA), 2015 International Conference on* (pp. 955-958). IEEE.
9. Gabriel M. (2017, Oct). Recommender Systems in Python 101. Retrieved from <https://www.kaggle.com/gspmoreira/recommender-systems-in-python-101>.