Job Recommendation System for WaterlooWorks

Siyi Ji, Junheng Wang, Dong Xu

University of Waterloo

1 Introduction

More than 20,000 undergraduate students at the University of Waterloo undergo job search process every year as required by the co-op program using WaterlooWorks, the internal job-searching system. The website displays an enormous volume of jobs for students to apply, while the number of application for each student is limited. Students have to glance through all the job descriptions in order to find the jobs they want. This process is tedious and inefficient under the time limit. Currently, WaterlooWorks allows students to search keywords appeared in job title descriptions and location as well as filter jobs by the target programs. However, these features are not sufficient in eliminating the number of applications displayed to job seekers. In addition, students have personal preferences that may be hard to describe in any keyword.

In this work, we implement a job recommendation system based on job seekers' past job applications and similar user's applications. The aim is to improve the job searching process through a personalized system to help WaterlooWorks users find the ideal job postings efficiently.

2 Previous Work

2.1 Recommendation Algorithms

Various algorithms have been developed to implement a recommendation system.
A. Content-based Filtering
In a content-based filtering recommendation system, the content of items is compared with the a profile of users' preferences. The content of an item is typically represented by a set of keywords extracted from a document. A user profile is represented by the same set of keywords and is generated by analyzing the content of items this user likes. This is determined by either explicit feedback through asking users to evaluate items on a scale, or implicit feedback by observing users' action. Items are ranked by how similar they are to the user profile and the top N items are recommended.
B. Collaborative Filtering (CF)
A collaborative filtering system recommends items based on similarity measures between users and/or items. There are two basic approaches in Collaborative Filtering, user-based CF and item-based CF.

- User-based CF: find all other users with similar past behaviors and use their ratings on other items to predict what the current user likes
- Item-based CF: Instead of looking for the similar users, item-based CF finds all the similar items to the ones that the current user likes. The similarities between jobs are based on the applications received. Two jobs are similar when the same person apply to both of them.

One of the major weakness of Collaborative Filtering is the cold start problem, scalability and sparsity .

2.2 Methods of Similarity Calculation

A. Cosine Similarity
Cosine similarity calculates the cosine of the angle between two vectors. It generates a metric indicating how related are two attributes by looking at the direction and not magnitude. It is efficient to evaluate especially for sparse vectors.

$$sim(\mathbf{A},\mathbf{B}) = \cos\theta = \frac{\mathbf{A}\cdot\mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|}$$

B. Jaccard Similarity
Jaccard similarity measures similarity between finite sample sets. It is defined as the cardinality of the intersection divided by the cardinality of the union of the sample sets.

$$\mathbf{J}(\mathbf{A},\mathbf{B}) = \frac{\mid \mathbf{A}\cap\mathbf{B} \mid}{\mid \mathbf{A}\cup\mathbf{B} \mid}$$

C. Pearson Correlation Coefficient
Pearson correlation measures the extent to which two variables are linearly related. It is defined as the covariance of two variables divided by the product of their standard deviation. It has a value between -1 and 1 where -1 is negative linear correlation, 1 is positive linear correlation and a correlation of 0 indicates that there is no linear relation between the two variables.

3 Empirical Evaluation

3.1 Dataset

The dataset we used is extracted from Kaggle's Job Recommendation Challenge sponsored by CarearBuilder. The entire dataset contains 389,709 users, job postings, 1,048,576 job applications that users made from April 1, 2012 to July 1, 2012. The dataset is split in 7 groups, each group representing a 13-day period. Each user as well as each job posting is randomly assigned to exactly one of these 7 groups. The data has the following structure:

1. *Job postings:*
   Each job posting is composed of a unique JobID, a WindowId indicating which of the 7 groups the job posting is assigned to and information of the job posting, including the location and a job description.
2. *Users:*
   Each user contains a unique UserID, a WindowID indicating which of the 7 groups the user is assigned to. The remaining columns contain demographic and professional information about the user, such as Country, Degree Type and Work Experience.
3. *Applications made by users to job postings:*
   Each application contains aJobID, UserID, WindowID defined as above.

Since the dataset is too large, it takes a significant amount of time to train our models. We only used Group 1 data to train and make predictions.

3.2 Collaborative Filtering

The traditional CF use three different similarity measurements mentioned in Section 2 to determine how similar are two users/jobs. For example, in user-based CF, for each user in the dataset, compute a weight indicating the similarity measure to the current user by looking at their past applications. Predictions are made based on all other users' applications in the dataset. The procedure is presented below:

```
Input: current user j, applied jobs
Output: a prediction for every unapplied jobs
for $job_i$ in unapplied jobs do
    for $user_j$ in users do
        Weight = $similarity$(current user, $user_j$)
    end for
    Prediction = $normalize$($sum$(weights on $job_i$ for all users))
end for
```

The prediction is a number between 0 and 1, where 1 indicates that the current user will apply the job and 0 indicates that the user will not apply.
The performance of is evaluated based on the mean square error on the results for all three similarity measurements on both users and items.

In our case, we adjust the algorithm by making prediction based on k most similar users instead of considering all other users' behaviour in the dataset. This should give us better results because users that are not similar to the current user have no effect on the recommendation results. The procedure is presented below:

```
Input: current user j, applied jobs
Output: a prediction for every unapplied jobs
for $job_i$ in unapplied jobs do
    for $user_j$ in users do
        Weight = $similarity$(current user, $user_j$)
    end for
    Prediction = $normalize$($sum$(weights on $job_i$ for top-k-users))
end for
```

3.3 Experiments on Collaborative Filtering

We first run experiments on k from 1 to 100 to find k which gives the best result.

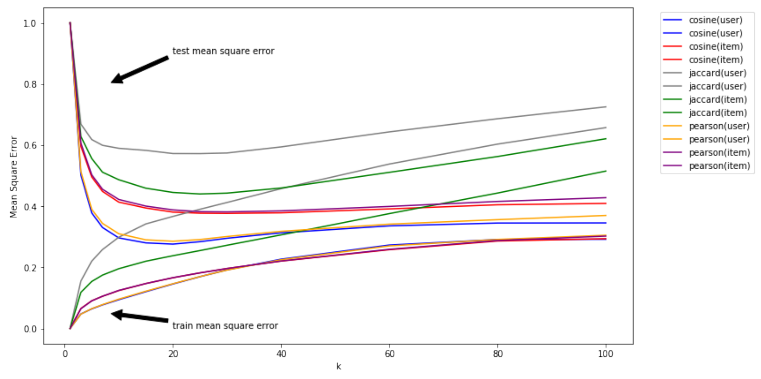

Fig. 1. Mean Square Error with k from 1 to 100

Fig. 1 shows the Mean Square Error(MSE) as k increases from 1 to 100. Both cosine similarity and Pearson correlation converge when k is greater than 40. MSE of Jaccard, however, keeps increasing as k increases. In order to do further experiments, we set the value of k to 40 and repeat our experiment.

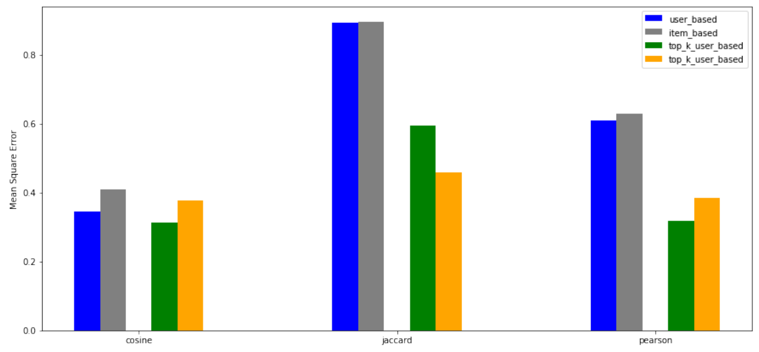

Fig. 2. Mean Square Error for item-based CF and user-based CF

Fig. 2 shows that both user-based and item-based algorithm reached higher accuracy when considering only top k users during recommendation. Second, user-based CF overall performs better than item-based CF. All two algorithms which use cosine similarity have the highest accuracy while algorithms that use jaccard have the lowest. The reason is that jaccard similarity is measured based on the number of intersections. The sparsity of our dataset is very low, most of the user profile vectors contain over 95% 0s. Even if two users may have very different applications, their jaccard similarity will be over 95%. As a result, all the weights are between 0.95 and 1, which causes the inaccuracy.

We notice that the sparsity has a huge impact on the performance of algorithms using jaccard similarity. We repeat our experiment to see how sparsity may affect algorithms using cosine similarity by applying three different filters to increase the sparsity of data as below:

1. Only consider the users who applied more than 10 jobs and the jobs that have more than 20 applicants. It gives sparsity of 1.03.
2. Only consider the users who applied more than 15 jobs and the jobs that have more than 25 applicants. It gives sparsity of 2.27.
3. Only consider the users who applied more than 20 jobs and the jobs that have more than 30 applicants. It gives sparsity of 5.88.

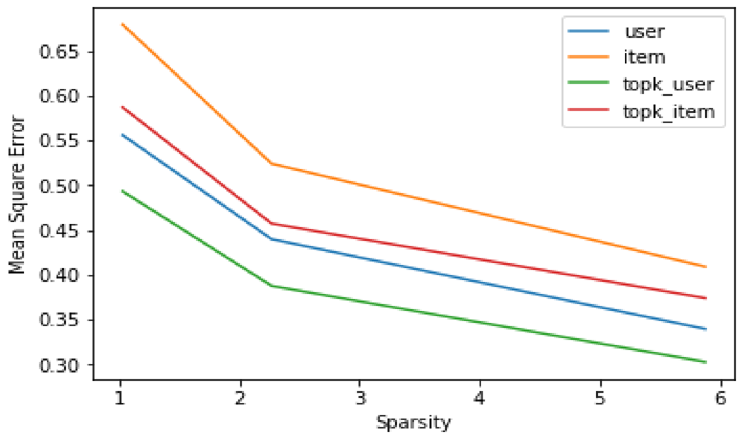

Fig. 3. Mean Square Error after applying filter

The result of Fig. 3 indicates that MSE decreases as the sparsity increases. More dense the dataset is, more accurate the recommendations will be.

CF suffers from poor scalability. Even if we are only training on the first group of dataset, it takes hours to train the recommender. The reason is that the complexity of CF is $O(mn + p^2)$ where is m is the number of unique users, n is the number if unique jobs, and p=m in user based algo $p = n$ in item-based algorithm. As the number of users or jobs increases, the running time increases quadratically.

3.4 Content-based Filtering

Training and evaluating a content-based filtering recommender has two steps. First, we compute the similarity measure among all job postings. Second, we recommend jobs to test users based on jobs they applied in the past and evaluate the results by computing the accuracy. The procedure is presented below:

```
for $job_i$ in jobs do
    Create a TF-IDF matrix of unigrams, bigrams, and trigrams
end for
for $(job_i, job_j)$ in jobs do
    $similarity = cosine\_similairity(job_i, job_j)$
end for
Sort similarity
for $user_i$ in users do
    randomly select some $user_i$ applied and recommend based on highest total similarity measures
    Compute accuracy by (number of jobs recommended and applied)/(number of jobs applied)
end for
Compute overall accuracy by averaging accuracy for each user
```

3.5 Experiments on Content-based filtering

We repeat our experiments for two different setups. In the first experiment, we train using different number of jobs (from 1,000 to 10,000), try (number of recommendations)/(number of future applications) from 10 up to 100. Every recommendation is based on one job per user. We choose only 1 job here to make it simple for recommendation and fair under comparison. In the second experiment, we train 10,000 jobs in total, fix (number of recommendations)/(number of future applications) to 20. Recommendations are based on up to 9 jobs per user.

The results of Experiment 1 shows that accuracy becomes lower as we train using more

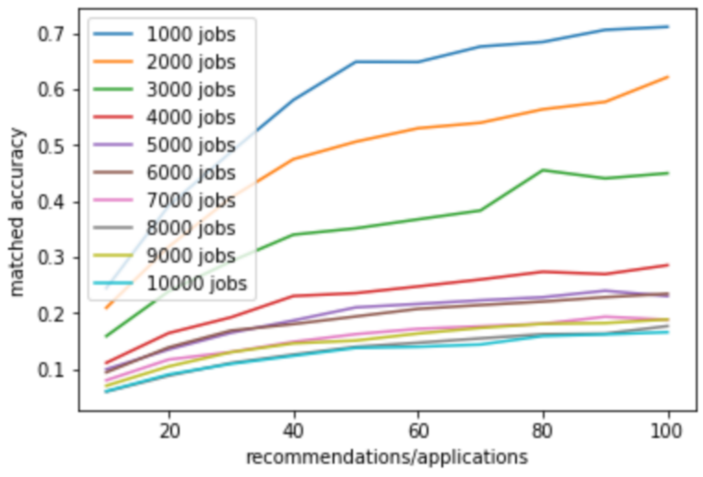

Fig. 4. Accuracy for content-based filtering Experiment 1

jobs. This is because the possibility to make a correct guess within a larger set of jobs is

lower.
The more job applications known, the better recommendation can be made (with diminish-

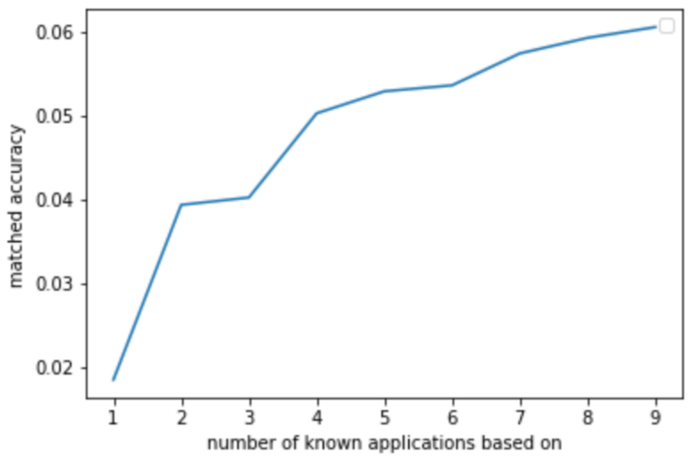

Fig. 5. Accuracy for content-based filtering Experiment 2

ing improvement). That is due to the limitation of being personalized and lacks variation. The knowledge of a users interest can only be calculated from his previous applications. So the more instance we get the better we know about the users interest. The diminishing effect is because we are less likely to learn about a new aspect of users interest with growing instances. The new instances are more likely to fall into a known aspect.

4 Conclusion

On the basis of Section 3, we evaluated two different approaches, Collaborative Filtering and Content-based Filtering to build a job recommendation system. CF uses behaviour of other users and items such as other users' past application to make recommendations. The performance of CF improves if the users submit more applications. This also implies that CF suffers from cold start, when it does not have adequate data of either the user or the item to make accurate predictions. Another potential problem is scalability. As the number of users and items grows, the running time increases quadratically. However, it outperforms Content-based Filtering when the content in items is difficult to analyze.

In content-based filtering systems, recommendations are made based on a user profile that contains information on the past applications. It does not need to learn the behavior of other users as it is not relevant to make recommendations to the current user. As a result, it takes significantly less amount of time to train the model compared to CF. Also, if the user preferences change, CBF can quickly learn the change and accommodate its recommendations. However, the accuracy of CBF depend on the expressiveness of item data.

Further research can be conducted on Hybrid filtering, where different recommendation techniques are combined to provide more accurate and efficient recommendations. For example, we can build a recommendation system using content-based filtering approach in a collaborative filtering model and vise versa. Using multiple techniques can solve problems in both CF and CBF while still preserving their advantages.

References

1. H. Li, F. Cai and Z. Liao, "Content-Based Filtering Recommendation Algorithm Using HMM," 2012 Fourth International Conference on Computational and Information Sciences, Chongqing, 2012, pp. 275-277.
2. Y. Zhang, C. Yang and Z. Niu, "A Research of Job Recommendation System Based on Collaborative Filtering," 2014 Seventh International Symposium on Computational Intelligence and Design, Hangzhou, 2014, pp. 533-538.
3. Cristianini, N., Shawe-Taylor, J.: An Introduction to Support Vector Machines. Cambridge University Press, Cambridge (2000)
4. Meteran, Robin van, and Maarten van Someren. Using Content-Based Filtering for Recommendation. Using Content-Based Filtering for Recommendation, users.ics.forth.gr/ potamias/mlnia/paper_6.pdf
5. Paparrizos, I., Cambazoglu, B., Gionis, A. (2011). Machine Learned Job Recommendation (Unpublished master's thesis).
6. CareerBuilder. Job Recommendation Challenge.