Personalized Job Recommendation System

Kun Su, Taylor Yang, SungYin Yang, Tian Lan,
Department of Software Engineering
San Jose State University
San Jose, CA, USA
Email: kunsu.903@gmail.com, tyang0904@gmail.com, sungyin.yang@sjsu.edu, tian.lan@sjsu.edu

***Abstract*—Currently, we are not able to find an existing solution on where it can provide personalized job recommendations to job seekers precisely. In this paper, we are trying to address this problem with some data cleaning, keywords extraction, user profile creation, and some common recommendation technologies based on the existing research.**

***Index Terms*—Job Recommendation, User-User CF, Item-Item CF, Content-Based Recommendation, TF-IDF, NLTK**

I. Introduction

Nowadays, online recruiting websites, such as Dice, Indeed, and Linkedin, have become the main source for people to land their dream job. Among all, Linkedin has been a leader in the industry to connect job seekers, recruiters, and professionals with needs. However, sometimes job seekers might still receive mismatched job recommendations due to the limitations of the system. For instance, a new grad may still see a recommended job post that requires senior experience. Or, a Back-end engineer may receive a recommendation for a Front-end engineer job. With all these in mind, our project focuses on how to improve the recommendation result that will further satisfy both the company and customer's needs. We approach the above problems with a hyper model that uses a combination of content-based recommendation system, User-User Collective Filtering, and Item-Item Collective Filtering.

II. Related Work

Scholars found that current information retrieval techniques cannot satisfy the need for candidate-job matching requirements [1]. Most of the current job recommendation systems provide recommendations based on the skill set retrieval from both the job requirements and user resumes without scaling skill level or job level, which is the main reason for the mismatched recommendation. In order to provide a better recommendation to users, the system needs to fully understand the job requirements content and the user resume. A similar concept was mentioned by Krishnaram, Benjamin, and Ganesh from Linkedin during the ACM conference [2], where they introduced member and job level regression coefficients to improve the ability to capture user's personal preferences. Besides that, Collaborative filtering had also been proved stably for job recommending. Thomas and Philippe had build a hybrid ”MAJORE” model, which combine collaborative filtering and locally linear embedding technique to train high accuracy model. [3]

III. Data Set Overview

In this project, we use a dataset from career builder cite-career where there are 3.8 million user data, 11 million job descriptions, and 1.6 Million users applied history. The user data contains job-related basic information such as major, degree, year of experience, current employment state, etc. The job description dataset contains the job title, descriptions, and requirements. The overall architecture of the system will be similar to what Shiqiang described in his thesis [4]. Data cleaning is mostly done by using Regular expression which was inspired by Shahules [5].

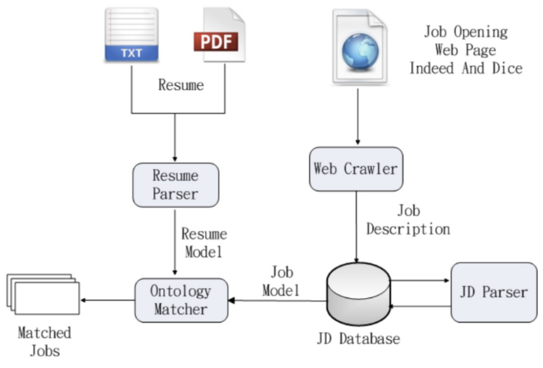


Fig. 1. Overall Architecture [4]

IV. Technical Approach

A. Content-Based Recommendation - Kun Su

In order to provide job recommendations to users. one of the key concepts is a content-based recommendation system that can suggest related jobs to the user based on the information from his/her profile or resume. The system recommends jobs in three steps: First, extract skillset from the job description and skill requirement which contains standardized job title and key skill information using NLTK [6]. Second, extract the skill set information from the user's resume. Lastly, use cosine similarity to calculate the similarity from the job pool and pick the top k recommendation to the user.

B. User-User CF - Taylor

Collaborative filtering (CF) is one of the techniques used by recommender systems. ”The underlying assumption of the collaborative filtering approach is that if a person A has the same opinion as a person B on an issue, A is more likely to have B's opinion on a different issue than that of a randomly chosen person [7]." Take our data and problems for example. We have an app.csv file that records each user's application history. If user A applied for a job that means this user is very interested in this job. So if another user B applied for similar jobs, we can assume that B's opinion is very similar to A, and thus it is very likely that they are going to apply for similar jobs.

C. Item-Item CF - Sung-Yin Yang

Another way of doing Collaborative filtering is item-item collaborative filtering. It is a type of recommendation system that is based on the similarity between jobs. The experiment would be done in three steps: First, extract the keyword by TF-IDF from each job description and then build a fixed dimension vector for each job ID. Different sizes of data set should have different sizes of fixed dimensions for job ID. Second, the similarity based on the scalar product of each jobID's vector will be calculated. Lastly, the final matrix based on the formula would be built.

D. Item-Item Collaborative Filtering - Tim

Item-Item Collaborative filtering recommends items to a user based on the similar items that the user has rated or interested before. It is a great algorithm that typically performs better than user-user collaborative filtering since the user may have many different preferences, but items have only a limited number of attributes. The model of item-item collaborative filtering for our application is derived from the original model by removing the denominator on the right-hand side of the equation based on the special context in our applications. [8]

V. Experiments

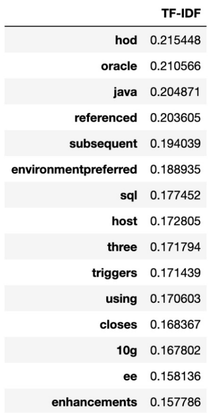
A. Data Clearing

Since the text sentence is clawed directly from the internet, it contains lots of metadata such as HTML tags, special characters, URLs, numbers, punctuation, and special encoding symbols. Those unrelated information have been removed by using Regular Expression. With exploratory data analysis, we also observed that some job requirements are blank or multiple job posts share the same job requirement content. In this case, we marked these cases and removed them from the training dataset as these data points would either provide insufficient or duplicate information.

B. Content-Based Recommendation - Kun Su

1) Standardized Job Title: The job titles have been clawed directly from the internet so they are not uniform and contain lots of noise. For example, "ft SAP software engineer" should be standardized as "software engineer". Here we use the "best ngrams" method, introduced by Estasney [**?**], to extract the standardized job title by dynamically calculating the score for the title and pick the best among them. Furthermore, we remove the low variance data where the count of the job title is less than 100 as the low number feature will bring in noise and impact the performance of the model. In the future, a finite state machine mentioned by Shiqiang [4] might be able to improve the performance and result in better standardized job titles.

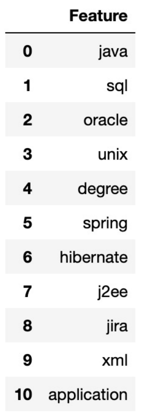
2) Extract Skillset From Job Requirements: By converting the job requirements into the TF-IDF vectors, we collect a set of important words that can be used to represent the requirements itself. However, in figure-2, we find that even though we are able to extract some technical terms from the requirements, we don't want to see terms like "three", "10g" and "Referenced" as part of our skillset.



	TF-IDF
hod	0.215448
oracle	0.210566
java	0.204871
referenced	0.203605
subsequent	0.194039
environmentpreferred	0.188935
sql	0.177452
host	0.172805
three	0.171794
triggers	0.171439
using	0.170603
closes	0.168367
10g	0.167802
ee	0.158136
enhancements	0.157786

Fig. 2. TF-IDF Skillset

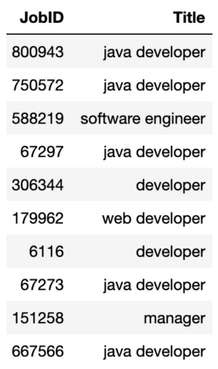
In order to extract the right term from the requirements, we apply NLTK to analyze the lexical categories for all target skills in the requirement [6]. After some experiments on analyzing the job requirements, we find that most of the target skills are categorized as "NN", "NNS", "NNP", "NNPS". Therefore, we only extract the skill set that is categorized as part of the NN families [9]. Figure-3 shows the sample of extracting skill set by using NLTK.



	Feature
0	java
1	sql
2	oracle
3	unix
4	degree
5	spring
6	hibernate
7	j2ee
8	jira
9	xml
10	application

Fig. 3. NLTK Skillset

3) Result and Future Improvement: However, the original dataset does not contain any resume data, so we decide to use our own resume as input to test the content-based recommendation system. A collection of skill sets that are directly extracted from our resume has been inputted as part of the user profile. After calculating the cosine similarity among all the jobs, the system will pick the top 10 most related jobs to the user. Figure-4 shows the result for input skillset as Java Back-End Engineer. In the future, further filtering and input parameters can be added to further increase the accuracy of the system, such as the scale of skill level or job level we mentioned before, and personal preferences.



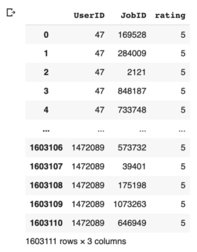
JobID	Title
800943	java developer
750572	java developer
588219	software engineer
67297	java developer
306344	developer
179962	web developer
6116	developer
67273	java developer
151258	manager
667566	java developer

Fig. 4. Top 10 Recommendation Result

C. User-User CF - Taylor

1) Creating user rating profile: Before training and filling in the missing entries from the user-job matrix, we need to create the user-job matrix by extracting data from our current data and creating a user rating profile. Then we will use spark.ml to train and generate the result.

2) Learn from Application history: First is building the user rating profile. By read and rearrange the data format from app.csv, we can build a matrix where the columns are user, jobID, and ratings. The rating score is assigned based on the definition that how likely a user will apply for the job. The maximum rating is 5. Since users already applied for these jobs, thus we need to use the maximum rating to assign to these users to jobID rows. The result will look like the following figure.

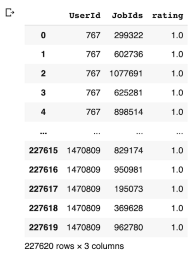


	UserID	JobID	rating
0	47	169528	5
1	47	284009	5
2	47	2121	5
3	47	848187	5
4	47	733748	5
...	...	...	...
1603106	1472089	573732	5
1603107	1472089	39401	5
1603108	1472089	175198	5
1603109	1472089	1073263	5
1603110	1472089	646949	5

1603111 rows × 3 columns

Fig. 5. User rating profile from application history

3) Learn from popularity history: Since the job data is very big compare to the application history. Around 11 million job descriptions we collected and around 1.6 million of application history. Especially the 1.6 million are repeatable job application data. For example, some users have applied for more than 10 jobs where some users did not apply at all. To deal with this cold star issue and such a sparse matrix. We assigned each user with a score 1 based on the top 10 popular jobs from their same state. The idea is that, compare to all the other jobs, the most popular jobs from that city are usually from well-known big companies. Besides, based on the demographic factor, people from the same city may have a similar background. So we can assign a small score for each user and assume that these popular jobs, even though may not be related to their background, have some effect on them and they are likely to apply for these jobs compare to all other jobs. After data processing, the top 10 most popular jobs for specific users and their ratings are as follows.



	UserId	JobIds	rating
0	767	299322	1.0
1	767	602736	1.0
2	767	1077691	1.0
3	767	625281	1.0
4	767	898514	1.0
...	...	...	...
227615	1470809	829174	1.0
227616	1470809	950981	1.0
227617	1470809	195073	1.0
227618	1470809	369628	1.0
227619	1470809	962780	1.0

227620 rows × 3 columns

Fig. 6. Updated user rating profile

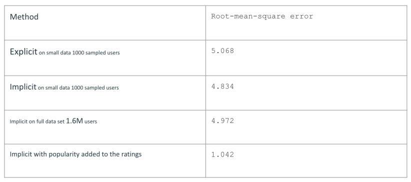
4) Building the model from different approaches: After generating the user rating to job table, we need to start to train our data. How the recommend-er system collects the ratings can make a difference for the result. For recommend-er systems, there are usually two ways to collect and analyze the matrix. Explicit and implicit way.

5) Explicit approach: The common approach to matrix factorization treats the entries as explicit preferences given by the users. For example, Netflix movies ask users to give out an exact rating from 1 to 5 to present their preferences in the movies. This method requires additional efforts to the user and usually is hard to get. [1] For our data we don't have an explicit rating that each user gave to different jobs so that the result of recommending jobs based on this method may not as accurate the other approach. We can also get this conclusion by looking at the final Result and evaluation data.

6) Implicit approach: In order to improve and more accurately reflect the data we received. Instead of trying to model the matrix of ratings directly, we are going to use the Implicit approach from Spark API. By their definition, "this approach treats the data as numbers representing the strength in observations of user actions (such as the number of clicks, or the cumulative duration someone spent viewing a movie). Those numbers are then related to the level of confidence in observed user preferences, rather than explicit ratings given to items. The model then tries to find latent factors that can be used to predict the expected preference of a user for an item." [10] As a result, we can also see that the RMSE(root-mean-square error) is lower than the explicit approach.

7) Result and evaluation: As we can see from the figure below. This User to user collaborative filtering method can handle both the 1.6 million application data or the sampled

1000 data. The RMSE (root-mean-square error) is about the same. Also as mentioned above, the implicit approach with added popularity information have the lowest error and thus performs the best out of other approaches.



Method	Root-mean-square error
Explicit on small data 1000 sampled users	5.068
Implicit on small data 1000 sampled users	4.834
Implicit on full data set 1.6M users	4.972
Implicit with popularity added to the ratings	1.042

Fig. 7. Result for all RMSE evaluation

D. Item-Item Collaborative Filtering Recommendation - Sung-Yin Yang

1) Job ID Vector Building: After data cleaning, the job description still include some unrelated data. In order to build a job ID vector, we will need to use only a portion of the job description data set to build the vector. For example, some columns in job descriptions such as zip-code and post-time are useless in building vector. After reduce the unwanted columns, we will then perform TF-IDF to select the top important words in the entire document. Different size of data set would require different sizes of job vector dimensions. For example, in this experiment, I used 10 thousand users and 60 thousand of jobs. And each job vector dimension is 7000. Then, build each job ID's vector based on these fixed dimensions. For instance, a job ID contains a list of vectors such as [security: 0.425, design:0.147, …]. Security is the column Name and 0.425 will be the corresponding value.

2) Formula: Below is the formula that will be used in this part.

$$sim_c(x, y) \propto \sum_{x'} sim(x, x')\mathcal{M}_{x',y}$$

Fig. 8. formula

Similarity C represent the final similarity that would be found in the Matrix; x is the job index; y is the user index; sim(x,x') is the scalar product of the two job ID vector between job ID x and x'; M is the interaction Matrix.

3) Similarity and Matrix Building: The similarity that we use in our formula is the scalar product instead of cosine similarity. The reason is that doing cosine similarity will require additional computation of each job ID, which would require another n square computation. The Interaction Matrix M is built by using the applied history of each user. The applied history includes the job that each user applied for. If user ID X has applied job ID y, then the corresponding matrix M(x,y) will be marked as 1; otherwise, it will be initialized as zero.

4) Evaluation method and Cutoff: The apply history data set will be split into a training data set and testing data set in 80/20. After the model was trained, we can then compute the accuracy and precision from the matrix using the test data set. Since our final matrix will be full with value between 0 to 1, we need to decide a cutoff to classify the job is recommended by the model or not.

Based on the paper from Thomas and Phillipe [3], their TF-IDF model can achieve accuracy for around 36 percent. We can easily achieve the same level of accuracy by lowering the cutoff. However, considering that adjusting a lower cutoff, the false positive rate could be high. The corresponding precision rate could be very low. Thus, we need to decide a good cutoff to evaluate our training model.

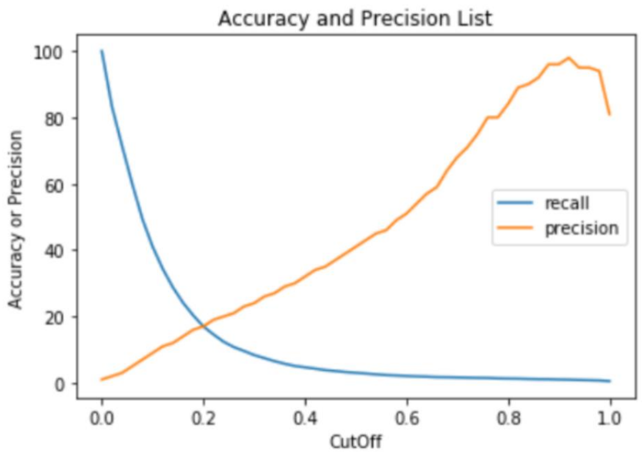


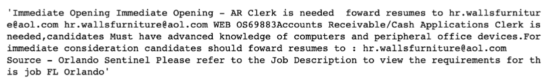
Fig. 9. CutOff and Accuracy/Precision

5) Result and Cutoff Observation: In figure 8, one can easily observe that as the cutoff increase the accuracy becomes lower. Conversely, as the cutoff increase the precision becomes higher. Thus, to balance the accuracy and precision, we will choose the final cutOff at around 0.2 where the precision/accuracy score is 20 percent.

Currently, for this experiment, only ten thousand users and 24 thousand users apply history were utilized, which are only 3 percent of our original data set. Half of the data set had been tried to run on the same algorithm. However, the result was not achievable because the computation requires enormous RAM and time. Why the computation are so costly? It is because as the number of job ID and user ID increased, the computation growth time will grow in a cubical way. Even though the adjacent list is applied to save the space resources, the computation could still easily run out of time. Thus, SVD matrix computation is a required work for the future experiment.

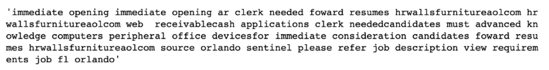
E. Item-Item Collaborative Filtering Version 2 - Tim

1) Data Preprocessing: Since the size of the original data is too large for us to analyze, data sampling is the first thing we did to reduce the size of the data for processing. The first 1000 users and their related applications and job titles and contents were extracted from the original dataset. After that, each attribute of the job entity was extracted and concatenated into a long text string which contains all the information about the job. Since this job text string contains some meaningless information, data cleaning is the next step to carry out. The data cleaning includes removing punctuation, stop words, and any word with digit letter combination. After that, all the letters in the text string are converted to lowercase so that lowercase and uppercase of the same word will not make it different since they have the same meaning. Figure 9 shows a sample job text string before data cleaning is executed.



```
'Immediate Opening Immediate Opening - AR Clerk is needed  foward resumes to hr.wallsfurnitur
e@aol.com hr.wallsfurniture@aol.com WEB OS69883Accounts Receivable/Cash Applications Clerk is
needed,candidates Must have advanced knowledge of computers and peripheral office devices.For
immediate consideration candidates should foward resumes to : hr.wallsfurniture@aol.com
Source - Orlando Sentinel Please refer to the Job Description to view the requirements for th
is job FL Orlando'
```

Fig. 10. Sample job text string before data cleaning



```
'immediate opening immediate opening ar clerk needed foward resumes hrwallsfurnitureaolcom hr
wallsfurnitureaolcom web  receivablecash applications clerk neededcandidates must advanced kn
owledge computers peripheral office devicesfor immediate consideration candidates foward resu
mes hrwallsfurnitureaolcom source orlando sentinel please refer job description view requirem
ents job fl orlando'
```

Fig. 11. Sample job text string after data cleaning

Figure 1 and 2 shows a sample text string before and after data cleaning respectively, and we can observe that many useless information is filtered out in Figure 2.

2) Item-Item Collaborative Filtering Recommendation Algorithm: Item-Item Cooperative filtering is a very classic recommendation algorithm that mainly recommend the item to users based on the similar items that they have rated before. The item in our context is a job, so we should first know how to compare the similarity of each job entity. Since each job entity contains many words and it is very inefficient and unnecessary to compare all of the words, TF-IDF was used to extract the most representative word respect to the complete job text string dataset. Each job is further reduced to a series of TF-IDF words. All the unique TF-IDF words are used to represent the column of the TF-IDF matrix, and each row is the ID of each job. If the specific job entity has the TF-IDF words in it, it gets a number 1 otherwise gets 0. The resulting matrix is very sparse since most of the job entity only contains a very small set of the TF-IDF words.

Cosine similarity is used to calculate the similarity between each job entity in the TF-IDF matrix. After calculating the cosine similarity of each job in the utility matrix, we generate a similarity matrix that contains the similarity value of all the job pairs.

3) Model Explanation: The standard item-item collaborative filtering mode is showed in Figure 11 [8]

$$pref\,(U_i, Item_j) = \frac{\sum_{l=1}^{k} sim\,(Item_l, Item_j) * p(U_i, Item_l)}{\sum_{l}^{k} sim\,(Item_l, Item_j)}$$

Fig. 12. Item-Item Collaborative Filtering model

Since the users do not provide any rating for the jobs, and we only know whether the user applied for the job or not. We use 1 represents that the user applied for the job and 0 represents the user who does not apply for the job. Therefore, p (U, Item) is a binary term. If we keep the denominator of the right-hand side of the equation, the preference value is either 0 or 1 which does not show how much the user likes the job. Therefore, the denominator is dropped and only the numerator is kept. The derived version of the model is shown in Figure 12. [8]

$$pref_o(U_i, Item_j) = \sum_{l=1}^{x} sim(Item_l, Item_j) * p(U_i, Item_l)$$

Fig. 13. Derived Item-Item Collaborative Filtering model

4) Result Evaluation and Explanation: The preference value of the user for a specific job is calculated based on equation 2. The threshold of the preference value for the candidate set of recommend jobs for a user is set to 0.3, which means that the jobs recommended to the user by the algorithm has a preference value higher than 0.3. The performance of the recommendation algorithm is evaluated by precision and recall.

$$Precision = \frac{Truth\ Positive}{Truth\ Positve + False\ Positive}$$

$$Recall = \frac{Truth\ Positive}{Truth\ Positive + False\ Negative}$$

Fig. 14. Derived Item-Item Collaborative Filtering model

The precision value is about 0.03 and the recall value is 0.28. The precision value is very low, but the recall value looks reasonable. Many potential reasons make the precision value very low. The first one is that only a small sample size of the dataset is used to build the model because it takes a very long time to process the whole dataset, so the model does not perform very well. Finding similar jobs for a specific job is the most time-consuming part of this model since the dimension of the similarity matrix is extremely large. In the future, the dimension of the similarity matrix should be reduced by applying certain dimensionality reduction techniques like PCA to speed up the computation of finding similar jobs for any specific jobs, so a much larger size data can be processed in a reasonable time with limited computation resources. On the other hand, the threshold for the preference may not be the best value for this model. More threshold of the preference value for this model should be set to find the best one that gives us the highest precision by drawing the receiver operating characteristic (ROC) curve. If all of this work is completed , the recommendation precision of this model should be significantly improved.

VI. Conclusion and Future Work

In this paper, we conduct four different approaches, for example Content Based modeling, User-User Collaborative

Filtering modeling and Item-Item Collaborative Filtering modeling with two different angles to solve the job recommending problem. We also evaluated the performance on results from different models and discussed its future potential.

While using the ”best ngram” method from EEstasney [**?**] in the content-based recommendation system, the job title standardization process might lead to an information loss, which caused by removing too many vocabularies during the procedure. However, we are able to improve the accuracy and integrity of the job title by using deep learning models, such as LSTM or a self-created finite state machine [4]. By using the NLTK alone, there are still some unrelated skills being extracted from the job requirement. Thus, we use a combination of TF-IDF and NLTK to remove the low-performance words in the TF-IDF process and retrieve a more accurate skillset list. In the future study, we intend to bring in more filtering in the system, such as user degree, school, location, tending skillset, and user preferences, to create a more comprehensive model that would perform a better match of job and candidate profile.

In user-user collaborative filtering, two different approaches are used to collect and analyze the matrix. The explicit approach treats the entries as explicit preference give by the user. The implicit approach treats the data as number representing the strength in observations of user actions. The implicit approach with added popularity information has lowered the RMSE than the explicit approach and thus perform the best out of other approaches.

In item-item Collaborative filtering, TF-IDF is used to extract key words from job description. Then, the job vector will be build based on the TF-IDF value. The result of the final matrix of item-item collaborative filtering will be a matrix full of value between 0 to 1. Different cutoffs(threshold) between 0 to 1 for the candidate set of recommend job could affect the precision and recall largely.

In the future, we would like to cooperate with job finding companies which could provide us anonymous resume and the corresponding apply job data. The more data we have, the better potentiality for us to explore and evaluate our methods. Furthermore, in order to reduce the compute time, math model and dimensional reduction technique such as SVD and PCA should be applied to train larger data set in a reasonable time.

VII. Task Distribution

1) Kun Su:

- Initiated and lead the project
- Data Cleaning
- Content-Based Recommendation System
- Initiated, managed and code reviewed GitHub repository
- Initiated, managed final report/PPT

2) Taylor Yang:

- Research and analysis the papers
- User rating profile set up
- User-User Collaborative Filtering
- Final report and PPT - User-User / abstract

3) Sung-Yin Yang:

- Data set finding
- TF-IDF model setup
- Item-Item Collaborative Filtering
- Final report - Item-Item CF / Conclusion

4) Tian Lan:

- Data sampling
- Item-Item Collaborative Filtering
- Final report - Item-Item CF

References

[1] Lionel Ngoupeyou Tondji. *Web Recommender System for Job Seeking and Recruiting*. PhD thesis, 02 2018.

[2] Ganesh Venkataraman Krishnaram Kenthapadi, Benjamin Le. Personalized job recommendation system at linkedin: Practical challenges and lessons learned. 08 2017.

[3] Philippe Caillou Thomas Schmitt and Michele Sebag. Matching jobs and resumes: a deep collaborative filtering task.

[4] Tracy Hammond Shiqiang Guo, Folami Alamudun. Expert systems with applications. *RésuMatcher: A personalized résumé-job matching system*, 60:169–182, 30 October 2016.

[5] shahules. Basic eda,cleaning and glove.

[6] Steven Bird. Categorizing and tagging words.

[7] Wikipedia. Collaborative filtering.

[8] Y. Zhang, C. Yang, and Z. Niu. A research of job recommendation system based on collaborative filtering. In *2014 Seventh International Symposium on Computational Intelligence and Design*, volume 1, pages 533–538, 2014.

[9] Mark Liberman. Alphabetical list of part-of-speech tags used in the penn treebank project.

[10] Unknown. Collaborative filtering.