PROJECT REPORT
ON

“LinkedIn Job Recommendations”

Submitted by:

Datta Prasad G Joshi **A20328560**
Kaushik Vaghasiya **A20336343**

CHAPTER 1

Introduction

Development of a recommendation system using LinkedIn data. The system recommends jobs to a person based on his/her LinkedIn profile, also recommends suitable candidates to the recruiter for a job post.

1.1 Problem Definition

LinkedIn is the world's largest professional social networking site. But this fails to recommend jobs, which we have to do manually. Also there is no concept of recommending candidates to the recruiter.

1.2 Proposed Solution/Hypothesis

Extraction of the user profile data like “Headline”, “Location”, “Latest Experience” and “Summary” and storing in a user document. Extraction of job description from current available jobs on LinkedIn and storing in jobs data document.

Matching these documents using cosine similarity and finally recommending suitable jobs to the candidates and also recommending suitable candidates for a job post based on Job Description.

1.3 Research questions from this Analysis

By following this Hypothesis we are trying to answer the following questions:

- How to retrieve LinkedIn data?
- How does Web Scraping works?
- How different document similarity methods work on real world data?

CHAPTER 2

Data

Two types of data are collected:

1. User Profile Data.
2. Job Description Data.

2.1 User Profile Data

This represents data of LinkedIn user profile. Following are the fields we are extracting from LinkedIn user profile:

- “User Name”
- “User Headline”
- “User Location”
- “User Latest Experience”
- “User Summary”

We tried to extract user information using **LinkedIn API**. However LinkedIn provides very limited access and one can only get the user information of single user like user name, number of connections etc., which fails to achieve our target.

Hence we started to work on automation script, which takes LinkedIn user name & password as input. Once the login is successful using **selenium** we automate **firefox** browser to visit random user profiles. Once user profile is visited by the browser, we are using **BeautifulSoup** to parse the html page and get all user data using **xpath**.

The required data once extracted are written into documents like each user’s individual document.

2.2 Job Description Data

This represents data of current available jobs posted in LinkedIn. Following are the fields we are extracting from LinkedIn jobs:

- “Job Title”
- “Company Name”
- “Location”
- “Job Description”

These fields are extracted by following the similar approach as we extracted the User profile data i.e. by **Web Scraping** the data.

The required data once extracted are written into documents like each job’s individual document.

Totally **1000 jobs** and **1000 user profile** data was extracted.

CHAPTER 3

Methods

There are two files one for retrieving the data & the other is to search for the recommendation.

The methods that are implemented for data extraction are as follows:

- **getPeopleLinks():** given HTML page as input returns page links of user profile.
- **getUserID():** for given user profile URL returns user id
- **getJobLinks():** given HTML page as input get links of job posting.
- **getJobID():** for given job posting URL return job id.
- **createJobDetailFile():** takes job details as input and creates a file and writes all details into it.

- **createUserDetailFile():** takes all user details as input and creates a document and writes all details into it.
- **getPeopleData():** for given browser and count of total people needs to be retrieved, returns people profile data.
- **getJobData():** for given browser and count of total jobs needs to be retrieved, returns posted job data.

The methods that are implemented for recommendation are as follows:

- **read_documents():** takes a directory name as input, and reads all the files to string, from directory and outputs text files.
- **tokenize():** takes a text file as input, converts the contents of file into list of lowercase words.
- **stem():** takes tokens as input, stems the english words using Snowball Stemmer and outputs stem words.
- **count_doc_frequencies():** takes documents as input and returns dictionary mapping terms to document frequency.
- **create_tfidf_index():** takes document & document frequency of each term as input, creates index in which each postings list contains a list of doc_id, tf-idf weight pairs.
- **compute_doc_lengths():** takes tf-idf index weight list as input and returns a dictionary mapping document ids to its length.
- **query_to_vector():** converts query term to dictionary with frequency.
- **search_by_cosine():** query vector, index and doc length, returns a sorted list of doc_id, score pairs, where the score is the cosine similarity between the query_vector and the document.
- **print_job_info():** takes all document ids as input & returns top 5 job results.

CHAPTER 4

Experiments

We started with the idea of “Phrase search” and “Boolean search” on documents. If users latest experience is as follows:

- Data Architect
- Greater Chicago Area
- Core Competencies

We used "Data Architect" as query and searched through each job documents and listed it. Though it was very efficient in finding exact match jobs, it was of no use since there was no criteria to sort good matches from thousands of jobs listed for Data Architect, also there were some missing expertise matches like if user has worked in NoSQL data, but job asks for Object oriented data processes, then job asks for Object oriented data processes with same job title.

We also worked briefly on **Jaccard similarity** between user document and job document. However it was really a bad idea. It was matching many unrelated jobs for user profile, which was expected. Since it only matches if term is present in the document or not. And since we were using job description, user summary and description of experience as well, hence there was very strong similarity between many of the documents.

Finally we started working on **tf-idf** and **cosine similarity** to counter the matching of conjunctions and very common terms. Now we were able to solve the problem of way more unrelated jobs like jaccard similarity. However this approach does not return as accurate results as phrase/boolean search, but still it solves the problem of skills matching.

One of the thing we learned through these experiments in this project is more the data, better the results. We started with few data and were getting really good outputs but it failed immediately once we increased the data size. And once we increased the data size, though getting output and verifying them was tough, but we were sure that it'll work on real world scenarios.

CHAPTER 5

Related Works

In many social networking sites or any movies or video sites there are recommendations, but for jobs we couldn’t find any, so this is kind of new problem that we are trying to solve.

CHAPTER 6

Conclusion & Future Works

6.1 Conclusion

To conclude, from this project we will come to know that Cosine similarity works better in terms of identifying better matched jobs for the user than phrase search and jaccard similarity. Although phrase search is efficient in finding the exact job match but as our purpose was to match skill sets, hence Cosine similarity worked better in this project. Also we can tell that this approach works effectively when there is huge amount of data.

6.2 Future Works

As mentioned before Cosine similarity on document similarity worked better than others but not efficient enough like phrase search. Few of the ideas, we would like to implement in near future are as follows:

- Rank similar documents by cosine similarity and filter them using phrase search.
- Maintain a list of words used for professional skills (like HTML5, SQL, Java etc.) and give them high weightage while comparing.