Building Career-Skill Knowledge Graph and Job Recommendation

Ruijie Rao, Linkai Shao

1. Introduction & Goal

Employment is utterly important for graduating students like us. A great news is that, in the recent labor market, more and more employers are shifting from unidimensional Degree-based Hiring to a much more diverse talent acquisition mode: Skill-based Hiring (Harvard Business Review, 2022). This means that employers care about if the candidate's skillset matches the offered position: a candidate with skills like SQL, Spark and Python is much more competitive than someone with only Excel, Statistics and Mathematics for positions like Data Analyst. Fortunately, a lot of those required skills are able to be acquired using tons of online resources. Platforms like Coursera and Udemy have grown wildly in the past few years due to the pandemic, making self-learning very accessible. Now, the question for us is: which career should we pursue and what skills should we acquire to make us more competitive as candidates? Our Career-skill knowledge graph is designed to answer those 2 exact questions by enriching the knowledge on career positions and skills. Moreover, linking those two entities together enables recommendation algorithms to advice suiting positions for students according to their current skillsets. We aim to give any employee a better understanding of the current job market, while more importantly helping them to see their career path in a skill-based way.

2. Data Sources

We have 4 data sources: Lightcast API, Indeed, Glassdoor, and Kaggle Dataset. All entities are collected from the Lightcast API, which contains 55000+ titles and 31000+ skills. Then, other job information like salary, job description, related titles, skills categories, etc., are extracted from the websites and dataset.

- Titles: name, salary, related skills, related titles, job description, responsibility
- Skills: name, isSoftware, isLanguage, category, description, type

We use Selenium to crawl all data from Indeed and Glassdoor. Then, we apply a rule-based method to conduct entity resolution across different sources for titles and skills.

3. Technical Challenge

3.1 Crawling

Since the data from Lightcast API is very structured and widely covered, our goal is to use its data as an anchor and crawl the information on each career title. Since we found out that Glassdoor has a different coverage on titles than Indeed, we decided to crawl each title on both websites. Although this means the load will be doubled, we believe that it will greatly increase the data accuracy because they can validate each other. Other than the huge load, both websites require some kind of validation to show the information. For example, Indeed has an anti-crawl mechanism that shows an error message, and Glassdoor requires the user to first sign in to view any information. Thus, Selenium is our only option, which is way slower than simple requests and requires more RAM. In order to accelerate the process, ThreadPool from Concurrent library is used to construct 10 parallel drivers. The process cost us 100+ hours in total for crawling 55,797 titles twice. 93% of all titles have a valid annual salary record, while all of the rest have a valid hourly salary as substitute. All titles have an unique ID and name, but only 36.6% have an available description. However, after our observation, those titles with no valid description are not very well-known (with its random sample including: Meat Processor, Proposal Development

Consultant, and Technical Cryptologic Technician). Regarding their skillset record, most titles have around 10 and 20 skills recorded, and all titles include at least 1 skill. As a result, the validity of the data crawled is pretty good.

3.2 Job Recommendation System

We want to recommend jobs to users based on their acquired skills. For example, users who target a Data Scientist/Data Analyst position usually have skills like Python, data analysis, A/B Testing, data mining, etc. Then, for those people, who actually have these skills but don't exactly know where their skills will fit into future careers, our knowledge graph and query engine will recommend Data Scientist/Data Analyst to them.

The main idea for the recommender system is firstly we use all existing information in our KG to pretrain a nearest neighbor model. Then, we vectorize users' input skills to predict the jobs which have the most similar skill sets. During the data modeling procedure, we had several options in choosing the way to do skill vectorization and distance metrics as they will mainly affect our prediction results.

For the skill vectorization part, we tried one-hot encoding, bi-gram encoding, Word2Vec and FastText embedding. Initially, we used one-hot encoding and bigram, and both give accurate results. But we wanted to try more advanced techniques, like Word2Vec embedding, but it's unable to handle unknown or out-of-vocabulary (OOV) words. Then we did some research on some other embedding methods, like Facebook's FastText. The advantage of FastText is that it operates texts in a more granular way with n-grams. So we switched to FastText, but then we found a problem with it. For example, if user input contains "data visualization", then the n-gram model will partially focus on the word "data". And then the model will focus more on jobs whose skills contain many data related skills like "data mining", "data lake", etc., but less on other skills. So finally we decided to use one-hot encoding to represent a word vector which allows us to convert any skills combination to an input vector. Then we use bitwise and metric to determine the distance between two word vectors. And our output would generate jobs which have top 5 closest skill embeddings to the input.

Finally, to evaluate our model, we conducted a user research interview and received 95% accuracy in predicting users' target jobs.

4. Lesson Learned

This is an interesting project which gives us hands-on experience to construct a KG from real world data. Starting from crawling, we firstly found that Scrapy would cause some anti crawling errors on Indeed. Then we find an alternative way to use Selenium. Then during entity linking, we need to consider what conditions can link two titles/skills together. Among those conditions, data cleaning is also important as real world information is sometimes messy.

For the technical challenge part, we also learned and practiced many different concepts. To convert users' skills to vectors, we actually tried many different methods. Across those vectorization methods, we then tried many distance metrics for them. Overall, this is really a good experience for us to learn to corporate KG with data modeling to make job recommendations.

Vue is brand new to us, and we are not familiar with Javascript at all. However, the time limit in this project definitely pushed us to grasp them. After all, we are very glad that we learned them and believe those skills will help us in the future. Meanwhile, creating an application on Knowledge Graph also inspired us. At the beginning, we were frustrated on how to visualize the whole graph and how to make things interactive. However, soon we realized that although KG is a graph, its application can be totally irrelevant from graphs: it can be a query, table, menu or library as long as it holds all the knowledge from the KG and fully utilizes its power of linkage and relations on fulfilling the designed purpose.