Ultimate Resume Helper

EECS 549 Final Project Report

Hsiao Yi-Ting, Huang Hsuan Hsueh

1 Introduction

Building a resume is a tough job for people finding their first job because they usually need a lot of references to create a delicate one. Without a search engine, people have to manually collect others' resumes and classify useful information. To make the process more efficient, we built a search engine for resume experience search. Given the keywords and types of documents the user needs, it will retrieve relevant items from our database built by LinkedIn profiles data. Unlike traditional search engines that incorporate rule-based algorithms (e.g. TF-IDF), our model includes auto-correction, query expansion, and learning-to-rank techniques to improve our search performance. Our IR model has outperformed the traditional baselines by 13 %.

2 Data

2.1 Data Acquisition

We designed a web crawler tool based on selenium that can collect data from LinkedIn user profiles and store them as JSON files. The information we collected from each profile are the experience and education. Each document will be a bullet point in each experience. Company information, period, title, education...etc data will be our meta data from learning to rank model. An example was illustrated in figure 1. More details on how we preprocess data will be introduced in the later of this report.

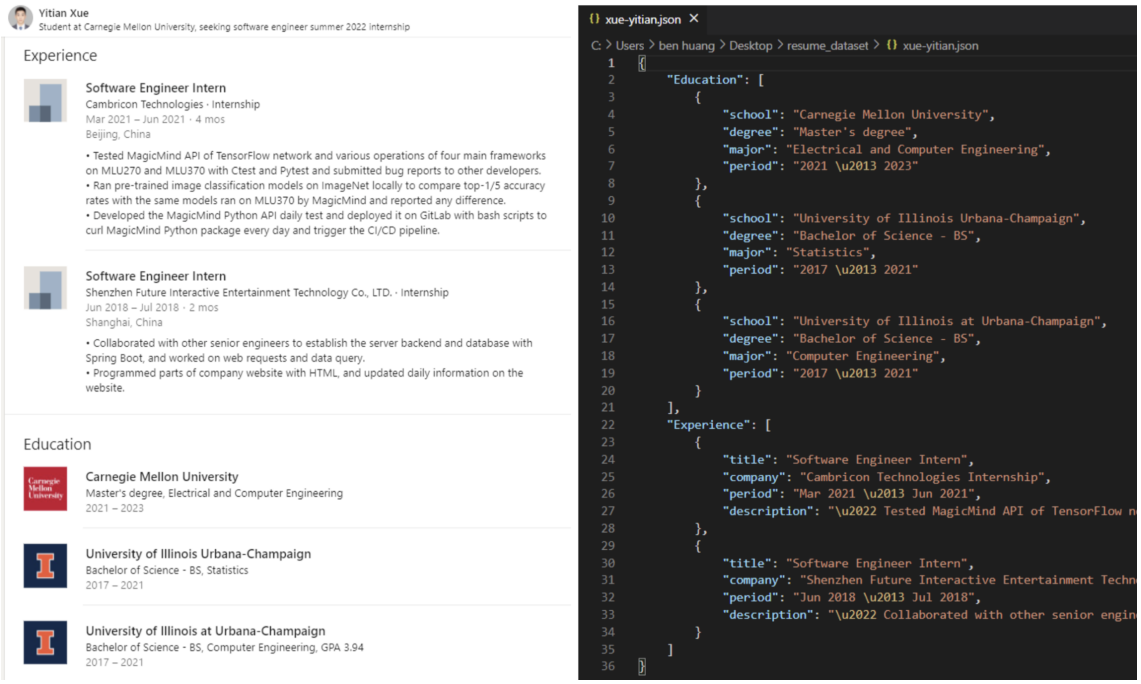

Figure 1: Raw Data from LinkedIn Profile and scraped data

2.2 Data Statistics and Annotations

We have 4928 annotated data in total, with 2464 labeled training data and testing data, and each consists of 7 queries and 352 documents. For the annotations (labeling) part of each query, there are four label scores available: -1, 0, 1, 2 . 2 is the most relevant document(with the highest relevance score), followed by 1 and 0. -1 is irrelevant documents. Apart from the relevance, we will also take into account of the quality of the document while labeling the documents. That is to say, if a document is highly relevant but poorly written, its score will bump down for one grade (i.e. score 2 will become score 1, or score 1 will become score 0). We used

two criteria to define the document quality". The first is to check if this document delivered a concrete result, and the second is to check if this document starts with an action verb. For example, "Spearhead the team to design an SVM classification model for a business problem and increase product net sale by 10 %" is a good document because it starts with an action verb (spearhead) and delivers a quantified result (increase net sale by 10 %)

The annotation distribution of our training data was illustrated in figure 2. According to the histogram below, we can see that most documents are labeled as irrelevant(i.e. -1), followed by highly relevant(i.e. 2), relevant(i.e. 1), and somewhat relevant(i.e. 0).

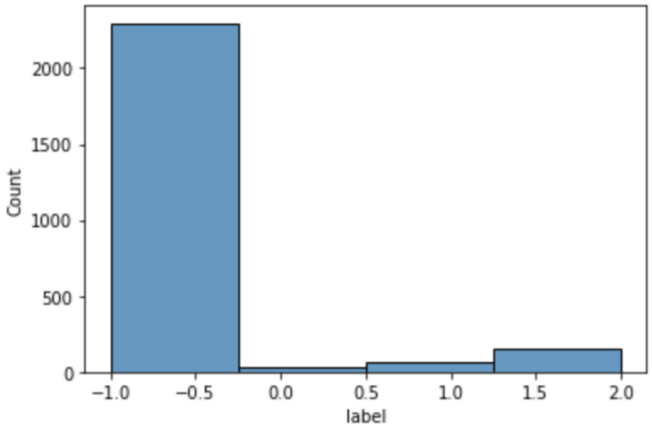

Figure 2: Relevance score distribution of our training data

3 Related Works

Given a large difference in content that needs to be ranked, personalized and recommended, each marketplace has a somewhat unique challenge. Search Engine for resume is a crucial part for Companies like LinkedIn and Indeed and they have put a lot of effort researching this topic. The search system in LinkedIn and natural language processing techniques for query suggestion, query auto completion, and documents ranking are introduced in [2]. Three key techniques, ranking functions, semantic matching features and query rewriting adopted by Yahoo search system are introduced in [5]. Also, some novel document embedding techniques for improving the document ranking performance are also introduced in [4] and [3]. Instead of only using the text of query to retrieve relevent documents, penalization is an important feature to improve the recommendation and the ranking system for Airbnb, and how it uses user data to achieve personalized searching results are introduced in [1].

4 Methodology

4.1 A short guide to the Ultimate Resume Helper

Ultimate Resume Helper(URH) aims to help a rookie job seeker to build his own resume by referencing others' resume descriptions that satisfied the keyword(query) he searched. For example, if I have skills regarding SQL but I don't know how to address this skill through my past experience, I can simply search "SQL" in URH, and choose a ranking algorithm I prefer. Shortly, URL will return results such as "analyzed clients' requirements, designed ER models and constructed corresponding databases using SQL server including 200k records; gathered comprehensive profiles of cars and customers by SQL queries". These results can therefore be a helpful reference for creating my own professional resume! A short demo of URH is illustrated in Figure 3.

4.2 Data Pre-processing

To pre-process our raw data, we first remove non-informative data in each resume experience. For example, if a LinkedIn profile experience block doesn't consist of any detailed description, it will be considered as an empty document and the data will be filtered out. Furthermore, if the document length is too short (less than 5 words,

```
Query: sql
IR model: DPH

Top10 relevent documents:
91    • Analyzed clients' requirements, designed ER models and constructed corresponding databases using SQL server including
200k records; gathered comprehensive profiles of cars and customers by SQL queries
96    • Accelerated data processing by 50% by automating data retrieval and extraction with SQL queries
152    • Wrote stored procedures, functions, and packages using PL/SQL, reducing project time by 25%
216    • Cooperated with engineers and PMs to design image recommendation feature; analyzed relationship between image usage
and ad performance using TB level data (SQL, C#)
243    •Designed an automated customer retention system with Python and SQL to complete weekly client status report for
strategy development. Increased customer retention rate by 15%.
176    – Use SQL / python to combine various source of financial data
347    – Implement front-end and back-end for various projects with multiple programming languages and platforms, including
ASP.NET, MVC, React.js, Node.js, JavaScript, C#, SQL, and Python
114    • Developed an automated system by designing APIs with Python, SQL, and Shell Script to facilitate other engineers'
model training processes, such as filtering, processing, and auto-labeling data from the internal shared database
55    ● As teaching assistant reviewed student's real-world application projects and presentation gave some
feedbacks/suggestions and trained student's R, python, SQL though TA sessions. Also, developed and maintained technique
questions bank.
189    The curriculum includes Linux, SQL, Java, Python, Javascript, Data Analysis, Machine Learning, Data Visualization. In
the final project, we leveraged the skills above and build an online platform to evaluate reasonable house prices of certain
area and provide analytic report and visualization.
```

Figure 3: Short Demo of Ultimate Resume Helper

in our case) it will not also be seen as a valid document and will be filtered out. After that, we removed stop words in the document. Finally, transform the JSON files to feature vectors for the machine learning model.

On the other hand, since we have a lot of metadata(for example, the education history of them) in our documents, we encode them into a string and separate them using a special character. For example, the writer of one resume might get his Bachelor's degree in National Chiao-Tung University, and get his Master's degree in the University of Michigan, then the education metadata for this document will be NationalChiaoTungUniverstiy$UniversityofMichigan.

After all the data are well prepared, it will be indexed using the library PyTerrier.

4.3 Query Expansion

Before our input data (i.e. query and documents) enters our ranking model, we will modify our query using the query expansion techniques. First, we will get the related terms of the original query using google search recommendations shown in figure 4. Specifically, we use beautifulsoup to implement the action of search, retrieving related terms, and scraping terms. Aftermath, we will then expand these terms behind the original query. A Short example of the original query and its related terms(expanded query) are shown in table 1. However, the weight of the original query and the expanded query are not of the same importance. Specifically, sometimes the related terms provided by google are not relevant. For example, the related term "background" doesn't really match the original query "teamwork". To solve this problem, we introduce a weight factor for our query expansion algorithm. That is, we will increase the importance of the original query by duplicating the terms of that. in the expanded query. For instance, if the original query is "apple" and the weight factor is 2, the expanded query will be "apple apple fruit banana red...." . The weight factor is a hyperparameter and we found this query expansion most effective when its value is 5. A brief example of query expansion is illustrated in table 1.

The reason why query expansion is required is that our documents are very short due to the nature of our search engine. Since each document usually only consists of one to two sentence(s), it is very likely that a document doesn't contain any query terms but is highly relevant.

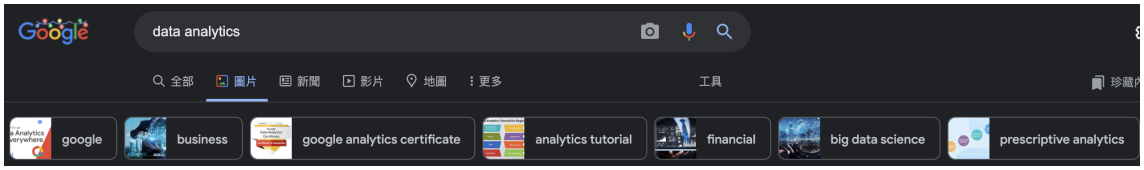

Figure 4: Retrieve related terms using Google Search

	original query	expanded query
1.	data analytics	tableau, machine learning, ml, analysis, data, analytics, prediction, regression
2.	system design latency	system design latency, load balancing, architecture, trading, logging system, vlsi
3.	teamwork	teamwork, collaboration, leadership, background, communication

Table 1: Example of Original Query and Related Terms retrieved from google search

4.4 Query Auto-Corrections

Right after preprocessing our documents, we will also preprocess the query by auto-correcting the mis-spelled query terms. This is necessary when the query provided by the user are often mi-spelled, which is a common situation for a search.A brief example is illustrated in table 2.

query	after Auto-Correction
dataC learning model	data learning model
web Tsecurity	web security

Table 2: Example of Query Auto-correction

4.5 Learning to Rank

After all the preprocessing steps, we train a pointwise learning-to-rank model using random forest. Apart from the metadata and query-document information, we also create some features that help our model to determine the document quality. Specifically speaking, we used the nlp library Spacy to perform Part-of-Speech(POS) tagging and check if the first word of a document is an action verb. Also, we checked the numerical values in the document as one feature to help our model analyze whether this document delivered quantifiable impacts. A short exmaple is illustrated in table 3.

document	action verb	quantifiable impact
Designed an SVM for a business problem and increase product net sale by 10 %	True	True
Enfoi is a company that makes AI-driven decision using machine learning models	False	False

Table 3: example of two customized input features that help determine document quality

5 Evaluation and Results

5.1 Overview

	name	MAP	nDCG@5	nDCG@10
1.	TF-IDF	0.493749	0.768400	0.771243
2.	BM25	0.496354	0.768400	0.771243
3.	DPH	0.506164	0.806511	0.797535
4.	Random_Forest	0.485357	0.985868	0.940411

Table 4: Query Search Result: with Aucot-correction and query expansion

For evaluating our model result, we use both nDCG@5, nDCG@10, and MAP to analyze our performance. Table 4 is an overview of the result of our model with query expansion and auto-correction. According to our result, we can see that DPH and Random Forest outperformed our baselines BM25 and TF-IDF. More details will be discussed in the next section.

5.2 Auto-correction

For evaluating the result of our auto-correction model, we created a function to randomly replace a character in our original query into a random character. Then, we analyze how our model will perform with and without the auto-correction. According to our result, although there is a significant increase in nDCG@10 and MAP for the auto-correction model compared to the one without auto-correction. However, the result of nDCG@5 is lower for the auto-correction model. We will further discuss this result in the next section.

	name	MAP	nDCG@5	nDCG@10
1.	TF-IDF	0.365668	0.644799	0.616955
2.	BM25	0.365744	0.644799	0.616955
3.	DPH	0.382320	0.701326	0.650689
4.	Random_Forest	0.348325	0.698849	0.682299

Table 5: Query Search with auto-correction

	name	MAP	nDCG@5	nDCG@10
1.	TF-IDF	0.309748	0.682605	0.601278
2.	BM25	0.309748	0.682605	0.601278
3.	DPH	0.327497	0.740909	0.632336
4.	Random_Forest	0.298993	0.740909	0.668672

Table 6: Query Search without auto-correction

5.3 Query Expansion

For evaluating our query expansion model, we also test the performance of each model without query expansion. According to the result, we can see that there is a significant increase in model performance after incorporating query expansion. The performance of our model without query expansion is illustrated in table 7

	name	MAP	nDCG@5	nDCG@10
1.	TF-IDF	0.383435	0.718270	0.672158
2.	BM25	0.383491	0.718270	0.672158
3.	DPH	0.400839	0.765254	0.698722
4.	Random_Forest	0.366590	0.762776	0.730332

Table 7: Query Search without Query Expansion

6 Discussion

6.1 An overview of our model performance

6.2 Effects of Learning to Rank

Our query-expanded random forest algorithm outperformed all of our baselines by 20 %. However, unfortunately, our Random Forest algorithm doesn't always outperform our baseline TF-IDF and BM25. This is because Random Forest may overfit easily due to the nature of the algorithm. One way to solve the problem of overfitting is to use techniques such as early-stopping.

6.3 Effects of Auto-correction

According to our auto-correction experiment, we can see that both MAP and nDCG@10 increase significantly for each type of ranking model. In general, the performance will be increased by 5 %-15 %. However, when we look deep into the result of nDCG@5, the performance has decreased after the auto correction. This is because our auto-correction algorithm might sometimes make mistakes. For example, the auto-correction of the term "InfoQrmation retrieval nlp" is wrongly corrected to "Information retrieval nl".

6.4 Effects of Query Expansion

According to our result, our query expansion algorithm significantly increases the performance of our original model and baseline by 5 % - 15 %. As mentioned previously, this is reasonable because our document lengths are short, so it's important to find synonyms for our query. For example, the query term "teamwork" might not find good results because this term usually won't exist in a resume whereas similar words like "collaboration", "corporation", "lead" are often found in the documents.

6.5 Effects of different features

We'd tested different features for our learning to the rank model including metadata and testing the quality of the document. We realized that not all metadata is useful for training machine learning models. For example, while the metadata "title" is useful, "education" "period", and "company" are not helpful for ranking. The result is quite reasonable because this information is not relevant to the content of the document.

7 Conclusion

In this Ultimate Resume Helper project, we built a search engine for resume experience search. Specifically, given the keywords and types of documents the user needs, Ultimate Resume Helper will retrieve relevant items from our database built by LinkedIn profiles data. Our model includes auto-correction, query expansion, and learning-to-rank techniques and has outperformed the traditional baselines by 28 %. In the future, we would like to add on user features to our model for personalized personalized search. You can check our code on github https://github.com/5410tiffany/SI650-Final-Project.

8 Other Things We Tried

8.1 Spell-checker using Word2vec

After realizing our auto-correction package are not so correct, we plan to develop our correction using word2vec. Also, our query auto-correction doesn't solve the problem if there is a typo in our document.

Basically, we made a version that only works word by word(not n-gram). First, we use the package gensim to transform all the text into a word2vec model. This will transform word corpus into a vector space so we can use math to analyze our corpus. Then, we find the Levenshtein distance of each query term and each word in the document. If the distance between the two is larger than a threshold, we will be seen as the same word. For example, "yeess" (0.9566314220428467) and "yes" (0.9314475059509277) has a very close distance, so they will be seen as two identical word, and therefore fixed the term mismatch due to typo. However, the outcome doens't significantly greater than the original auto-correction package, so we abandoned this part.

9 What You Would Have Done Differently or Next

9.1 Personalized Search

Our data contains a lot of metadata. Originally, we plan to use these data for personalized search. For example, before a user conducts a search, he or she is able to add on his personal information such as his job preference, education...etc. It will then be one of our input features for our learning to rank IR model. Then, ideally, our model will be able to display documents in which the authors of these documents have similar personal information. We believe this function is useful for our search engine because when writing a resume, we usually would like to find references or samples from which the author has a similar background to us.

9.2 Document Expansion

Due to the nature of our document, our documents can be very short. As such, it is very likely that one document doesn't contain any query terms but is highly relevant to that query. To address this problem, we used query expansion and it seems to be quite effective. However, we can also expand the document using similar techniques.

9.3 Determine the whole Roadmap Earlier

One thing I think we could've done differently is to determine what we are going to do and what libraries/packages we need earlier. Specifically, in our project update, we use Pyserini to index our documents. However, we changed our whole indexing framework into PyTerrier (which takes quite a lot of time!) because a part of our model requires learning to rank algorithms. Therefore, it's better to think thoroughly about what we are going to do for our model and choose the right package at first so that we can reduce redundant work.

References

[1] Mihajlo Grbovic and Haibin Cheng. Real-time personalization using embeddings for search ranking at airbnb. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 311–320, 2018.

[2] Weiwei Guo, Xiaowei Liu, Sida Wang, Michaeel Kazi, Zhiwei Wang, Zhoutong Fu, Jun Jia, Liang Zhang, Huiji Gao, and Bo Long. Deep natural language processing for linkedin search. *arXiv preprint arXiv:2108.13300*, 2021.

[3] Omar Khattab and Matei Zaharia. Colbert: Efficient and effective passage search via contextualized late interaction over bert. In *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval*, pages 39–48, 2020.

[4] Sean MacAvaney, Andrew Yates, Arman Cohan, and Nazli Goharian. Cedr: Contextualized embeddings for document ranking. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 1101–1104, 2019.

[5] Dawei Yin, Yuening Hu, Jiliang Tang, Tim Daly, Mianwei Zhou, Hua Ouyang, Jianhui Chen, Changsung Kang, Hongbo Deng, Chikashi Nobata, et al. Ranking relevance in yahoo search. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 323–332, 2016.