# Information Retrieval

## Search Engine Bandits

#### Sara Asaad – 207674318

#### Ward Zidani – 318531944

#### Safwan Halabi – 213784127

[GitHub Link](https://github.com/Safwan-Halabi/Information-Retrieval.git): - <https://github.com/Safwan-Halabi/Information-Retrieval.git>

## **1. Introduction:**

In recent years we’ve seen a steady decline in software related jobs especially those for students currently enrolled in a software engineering or computer science program and so we wanted to create a crawler that scrapes LinkedIn job listings to grant the user some foresight into the software market at the moment and provide them with tools that can potentially help them land a job or at least go over a list of jobs faster than how they would have if they worked alone.

The system we create leverages several powerful Python libraries such as requests, BeautifulSoup, RE, NLTK, Pandas and more to scrape, process and analyze LinkedIn data and export the results to several csv files with all the relevant information processed.

### **1.1 Queries**

Our crawler had 3 main queries (2 original ones we presented and a new one):

Query 1 – Responsible for finding the number of open positions for several keywords we provided, such keywords can be adjusted in the code, the goal of this query is to create a comparison between jobs requirements allowing the user to see the difference in demand between many skills/technologies and how the market looks at said skills/technologies in job candidates.

Query 2 – Responsible for scraping the data of various job listings from a given keyword, the scraped data specifically targets the link/URL of the listing and the requirements listed and saves them to a local csv file, this query also handles duplicates in input data such that if the word “Python” and “py” both show up in requirements the csv file will only include “Python”.

Query 3 – The newest query added as a suggestion for improvement after the presentation we gave, is responsible for taking the csv file from Query 2 as input and asking the user for skills they have/technologies they’re proficient in and filters the jobs and requirements found in the csv file and creates a “qualification\_percentage” metric which tells the user how qualified they are to have that specific job, the query also suggests various skills/technologies which the user lacks/should improve on to be more qualified for a specific job, this query also exports the results to a csv file under a different name.

**\*\*\* Queries 3 should be used in tandem with query 2 as it uses query 2’s output in order to process the data and rank the results. \*\*\***

## **2. Technologies and Runtime:**

Here are some interesting technologies used in the project:

1. **Requests library** – used to send HTTP requests in code.
2. **BeautifulSoup library** – used to process the HTML pages returned from the requests.
3. **Pandas library** – used to save results in a DataFrame to later be exported to a csv file.
4. **NLTK library** – used to process text, specifically to remove stop-words from the texts returned.
5. **RE library** – used to process text and look for specific patterns in the strings.

For the **first Query** the runtime is about **1.5 minutes.**

For the **second Query** the runtime for a single execution is about **1.5 ~ 2 minutes.**

For the **third Query** the runtime is less than 1 second, this query does not scrape the website but it uses the results of Query 2, the time it takes for this query to conclude is under a second in most cases.

The code is **heavily dependent** on the request **bypassing LinkedIn’s bot detection** and the time it took to send various requests in both queries.

The **processing time** is **negligible**, it only takes a **fraction of a second** to extract the text, process it, save it in a DataFrame and export it to a csv.

If provided with a fast internet connection and run closer to a LinkedIn’s server the time could **potentially go down to less than a minute.**

## **3. Inverted Index and TF-IDF:**

For the sake of completing this assignment on time we will solve all the 3 questions by executing the second query on the word “tensorflow” and saved the results to “job\_requirements.csv”.

Our query yielded 47 results.

Here are the 15 most common words in the search results:

('experience', 232), ('learning', 61), ('data', 60), ('years', 43), ('machine', 37), ('tensorflow', 32), ('pytorch', 31), ('frameworks', 29), ('models', 28), ('working', 26), ('cloud', 26), ('science', 25), ('deep', 23), ('3', 23), ('ml', 23)

Here is the inverted index for the 15 most common words for the first 20 pages only:

1. experience: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
2. learning: [0, 1, 2, 3, 7, 8, 9, 10, 11, 13, 14, 15, 16, 17]
3. data: [1, 2, 4, 6, 8, 9, 10, 11, 13, 15, 18, 19]
4. years: [1, 2, 5, 7, 8, 9, 10, 12, 13, 14, 18, 19]
5. machine: [1, 2, 7, 8, 10, 11, 13, 15]
6. tensorflow: [0, 1, 2, 4, 6, 9, 11, 13, 14, 15, 16, 17, 18]
7. pytorch: [0, 1, 2, 4, 6, 9, 11, 13, 14, 15, 16, 17, 18]
8. frameworks: [0, 1, 2, 4, 6, 8, 11, 13, 14, 15, 16, 17]
9. models: [1, 2, 7, 11, 12, 15, 17]
10. working: [1, 2, 4, 6, 7, 11, 15, 18]
11. cloud: [2, 7, 8, 11, 15, 18]
12. science: [1, 2, 5, 9, 11, 13, 15, 18]
13. deep: [0, 2, 3, 9, 11, 14, 15, 16, 17]
14. 3: [1, 2, 9, 11, 13, 14, 15, 18, 19]
15. ml: [4, 6, 7, 8, 9]

**The TF-IDF calculation is found in “TF-IDF.csv” file, as it was too big to fit in this document.**

## **4. Hubs, Authorities and PageRank:**

**Query 1**: The pages returned are mainly **48 Independent pages**, we simply send 48 requests that aren’t connected at all, therefore we don’t have any **hubs** or **authorities**.

**Query 2**: The pages returned are mainly **1 Hub page** which has 60 links to the first 60 job listings.

**Query 3:** Does not scrape therefore it doesn’t have any **Hub** or **Authority** pages

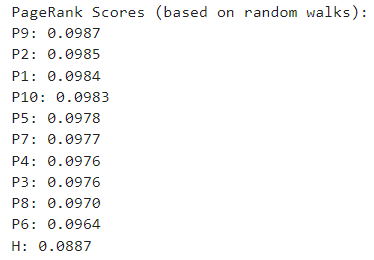
Each job listing is a stand-alone page and isn’t associated with the rest of the job listings – These are the **Authority pages** of which there are **60** in total.

Given that our system has 1 **Hub** and several **Authorities** (which no connection between authorities) we’ll have to use a different variation of the PageRank formula – one that incorporates random walks:

Each Authority page has a 10% chance to go to a different page altogether.

Here are the results we got:

Run 1:



Run 2:

A screenshot of a computer code

Description automatically generated

Run 3:

A screenshot of a computer

Description automatically generated

We see that a common trend we’re having is the **Hub** page having the lowest PageRank – which is logical given that **no pages have explicit links going into** the **Hub** page; moreover we see that the order of the Authority pages doesn’t matter and is constantly changing – which can be explained by the fact that each **Authority** page has **exactly 1 explicit link going into it** and they all **have a 10% chance to go out and branch into another random node** in the graph – meaning that the order of the nodes is reliant on chance alone.

We’ll take the average of all the rankings of runs 1-3, we then get the following (final) ranking:

A number of numbers on a white background

Description automatically generated

## **5. Precision and Recall:**

Out of the first 10 results only **9 are actually relevant – all but document 4.**

User 1 looked at the first 10 results and found that the following documents are **relevant: [ 1, 2, 3, 5, 8, 9, 10 ]** while the following are **irrelevant: [ 4, 6, 7 ].**

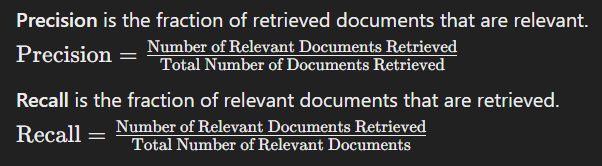
User 2 looked at the first 10 results and found that the following documents are **relevant: [ 1, 2, 6, 7, 9, 10 ]** while the following are **irrelevant: [ 3, 4, 5, 8 ].**

In order to suggest a more accurate query at the end we will look at the unique keywords that we can use to identify the documents.

**Unique keywords (for each job listing):**

|  |  |
| --- | --- |
| Job Listing | Unique Keywords |
| P1 | Deep learning |
| P2 | retention, predictive models, data visualization, Hadoop, Spark, SaaS, NLP, sci-kit learn |
| P3 | Forecasting, SQL, gradient boosting, MLOPs, reinforcement, Apache Airflow |
| P4 | Biomedical, deep learning |
| P5 | Research, publishing, paper, feature engineering, media |
| P6 | Formatting |
| P7 | Research, publishing, paper, feature engineering, media |
| P8 | ML, generative, AWS, GCP, RAG |
| P9 | pipelines, distributed programming, AWS |
| P10 | AWS, Big Data, CNN, RNN, LSTM, Transformer, sci-kit learn, Multimodal, transfer |

We will use the following formulas to calculate Precision and Recall:



**Results:**

**User 1:**

* Precision = 7 / 10 = 0.7.
* Recall = 7 / 9 =~ 0.77.

**User 2:**

* Precision = 6 / 10 = 0.6.
* Recall = 6 / 9 =~ 0.66.

**We see that user 1 is more interested in:**

retention, predictive models, data visualization, Hadoop, Spark, SaaS, NLP, sci-kit learn, Forecasting, SQL, gradient boosting, MLOPs, reinforcement, Apache Airflow, ML, generative, GCP, RAG, pipelines, distributed programming, AWS, Big Data, CNN, RNN, LSTM, Transformer, sci-kit learn, Multimodal, transfer.

**We see that user 2 is more interested in:**

retention, predictive models, data visualization, Hadoop, Spark, SaaS, NLP, sci-kit learn, Formatting, pipelines, distributed programming, AWS, Big Data, CNN, RNN, LSTM, Transformer, sci-kit learn, Multimodal, transfer.

## **6. HTML Pages:**

All the HTML pages can be found on our GitHub in the “docs” folder:

<https://github.com/Safwan-Halabi/Information-Retrieval.git>

## **7. Improvements and Reflection**

|  |  |
| --- | --- |
| Suggested Improvement | Is this idea feasible?  Explain how it is or why it is not. |
| In my opinion, the results returned can be presented to the user in a better way. | In our opinion the csv format we presented is ideal for the result visualization. |
| The presentation of the results. | In our opinion the csv format we presented is ideal for the result visualization. |
| As they mentioned, they also wanted to improve specific search for the user for employer and job. | We implemented this idea by adding the 3rd query. |
| Maybe add more filtering types to the queries. | It is feasible to add a flag in the function that scrapes the data and extracts the relevant part to make the function behave differently thus introducing more filters. |
| Improving the appearance for the user. | This idea is good on paper but bad in practice, we did not build an entire application rather we built the functionality behind it. |
| Complete the last query. | Implemented :) |
| 1.5 minutes to run the queries seems a bit long; maybe it is worth checking if it can be improved by optimizing algorithms or fetching less data. | As presented in the HTML pages and in this document, most of the time isn’t processing time or scraping time its time taken to bypass LinkedIn’s bot detection. |
| Preparing the crawler will be a good step forward. | No idea what this comment meant.  No comment... |
| Continue implementation. | Implemented :) |
| Shorten runtime. | We would love to shorten the runtime, however as stated earlier, the runtime is mostly for bypassing LinkedIn’s bot detection. |
| A more comprehensive retrieval plan, considering the presentation of the data to the user. | We entertained the idea of adding more data to the csv for each job listing, however we found that the link and requirements are currently the best metrics to keep in mind. |
| User interface improvement: Suggest a user-friendly interface that allows users to refine their query and add more precise parameters for the desired result. | This idea is good on paper but bad in practice, we did not build an entire application rather we built the functionality behind it, as for the query improvement/customization we can add that using a more complex way of processing the “relevant” and “non-relevant” job listings according to the user. (A.K.A a questionnaire of sorts) |
| Queries can be more accurate. | No comment... |
| Complete the third query. | Implemented :) |
| They didn't manage to finish the code, so maybe finish the code to improve retrieval. | Implemented :) |
| Can search times be improved by retrieving some of the open jobs first and then filtering them in parts with the second query, instead of running the first one and then the second query? | Not exactly, theoretically the search time can be cut down by using parallel requests but that increases the likelihood that the bot detection system stop us from scraping the website. |
| Parallel programming can be used to make the code run faster. | Not exactly, theoretically the search time can be cut down by using parallel requests but that increases the likelihood that the bot detection system stop us from scraping the website. |
| Response time. | As presented in the HTML pages and in this document, most of the time isn’t processing time or scraping time its time taken to bypass LinkedIn’s bot detection – it cannot be cut down by normal means. |

**Challenges:**

Most of the challenges we faced in this course were in the development phase of the crawler, given that LinkedIn is a giant company with thousands of engineers behind it, we found it particularly difficult to scrape the website successfully without sacrificing something in return, case in point – the runtime of the crawlers.

We should also mention that the first 2 weeks were also hard on us given that we changed ideas almost everyday, until we sat down with Dr. Naomi and she gave us a concrete idea and the greenlight to start the planning and development of the crawler.

That aside this course has been fun, interesting and practical – one would describe it as “קל ומעניין”, we thank Dr. Naomi Unkelos-Shpigel for this course and for the help she provided during the semester, we wish you the best.

**- Sincerely, Search Engine Bandits (Ward Zidani, Safwan Halabi and Sara Asaad).**