Job Search Engine / Intro to DS project

Juuso Saavalainen, Kozheen Taher Esa, Lauri Kauppinen

October 2023

1 Project overview

We started the project by brainstorming ideas. We ended up choosing the idea of making it easier to search for jobs that are suitable for students and fresh graduates. Members of the group had found it challenging to find relevant entry-level jobs from platforms such as LinkedIn. Typical job posts on such platforms contain too many or stringent expectations to be realistic for people looking for their first professional role. This doesn't necessarily mean that the job listings are bad but for someone without any previous work experience job posts with 3-10 years of experience required can be really daunting especially when those listings are labelled as "Entry-level roles".

In the beginning, we were thinking that we could do more than that and also pull data from sources like Glassdoor to get estimates for salaries and job satisfaction rates in the hiring organizations to better rank available positions. However, as the we started working on the project and got feedback from the teaching assistant, we decided to focus only on ranking jobs based on (i) how realistic those are for someone looking for an entry-level role and (ii) relevance of the job with respect to some keywords provided by the user. Main reason for the decision was to keep the scope of the project in line with course requirements and guidance.

To summarize, we aimed to build a tool that solves the following situation:

- User: an aspiring professional looking for entry-level jobs matching some keywords about the job
- Input: keywords describing what kind of a job is being searched (e.g. "data scientist", "python backend engineer")
- Output: list of job posts that are relevant with respect to the keywords and are actually entry-level

From a technical point of view, the project can be divided into the following parts:

1. **Scraper**: getting raw data about available jobs given some keywords

2. **Ranker**: ranking jobs based on how well they suit someone looking for an entry-level job and how relevant they are with respect to the keywords

3. **User interface**: an application that gets keywords from user and presents the most suitable entry-level jobs

We ended up building a scraper that fetches jobs from LinkedIn for given keywords. The scraper is explained in more detail in Section 2.

We tried out multiple approaches to figuring out which job posts are relevant for entry-level applicants. Based on a subjective evaluation of which approach produces the most relevant results, we ended up using zero shot classification with a pre-trained natural language inference (NLI) model (more details here). The other approaches were based on manually labeling a training set of job posts to entry/not categories and using machine learning (ML) algorithms to predict categories of unseen job posts. The rankers are described in Section 3.

We built a simple web app ("app 1") using Streamlit to showcase a "minimum viable product". The web app asks user to provide keywords (e.g. "data engineer") based on which LinkedIn is queried for all relevant jobs. Then, the jobs that don't require previous experience according to the NLI model are shown to user. In case no jobs are found from LinkedIn or all jobs are evaluated to require previous experience, the user is told about it explicitly. Section 4 contains more details about the web app.

Due to legal uncertainties related to LinkedIn scraping, we built another web app ("app 2") that fetches previously scraped job posts from cache based on user input instead of scraping LinkedIn at runtime, and filters entry-level posts using the NLI model just like "app 1". "App 2" can be tested here.

The source code can be accessed in GitHub.

2 Scraper

The scraper was implemented in two parts. The first part of the scraper uses Beautiful Soup (a Python Library) to fetch jobs from the same API that LinkedIn's public website uses to achieve the same goal (example). We hard-coded the query to scrape a maximum of 5 pages of results and only jobs in Helsinki region, but the query could easily be extended to cover more areas and more results. The variable of the query is the string of keywords provided by the user. The effective output of the query is a list of (LinkedIn) job post IDs, which are user in the second part of the scraper to fetch details about the jobs.

The second part of the scraper fetches relevant information for each job post ID retrieved by the first part of the scraper from the same API that LinkedIn's public website uses to achieve the same goal (example). The second part is also implemented with Beautiful Soup. Relevant information contains:

- Name of the company: e.g. "Nokia"
- Job title: e.g. "Data Engineer"

- Description: long text explaining what the job is about and what is expected from a candidate. This is the core information used in determining whether the job is an entry-level one or not
- URL: link to the job post so that we can offer it to the user

We used the same scraper to both (i) get jobs at runtime in "app 1" and (ii) fetch a database of job posts used in "app 2" and in training ML-based classifiers. The latter was accomplished by running the scaper with a set of keywords generated by the group with the intention to get a comprehensive set of data and software related jobs (e.g. "python", "developer", "backend", "frontend"). User can search for jobs in "app 2" using the same set of keywords.

3 Ranker/labeler

3.1 Training and developing own model

Our first attempt was to use a ML classifier to label job posts that are entry level friendly. The first task was to label the data since raw job posts from LinkedIn are not classified in a reliable manner (which is exactly the problem solved in the project). We tried different automatic labeling methods and unsupervised approaches with no success. Hence, we ended up labeling around 300 job descriptions manually to get some training data to work with. During the labeling, we also dropped some rows containing unwanted entries. The binary labels were 1 ("realistic for entry level applicants") and 0 ("unrealistic"). The output of labelling was a training set of 161 "zeros" and 97 "ones" making a total of 258 entries.

We tried to use the Naive Bayes classifier from scikit-learn. It produced some predictions, but the correctness of the predictions was only slightly better than random guesses so we tuned it more. We noticed the imbalance of labels so we tried to use Synthetic Minority Over-sampling Technique (SMOTE) to balance the labels. With that, we combined some hyperparameter tuning with scikit-learn's GridSearchCV and tried Gradient boosting and Randomforest classifiers. With this setup, we reached 70-80% cross-validation accuracy, which was still deemed to be too low for this specific task. On top of that these values only tell us so much about the ability of the model to predict these complex patterns in real unseen changing data with a small fixed dataset. So we could see the job listings with clear unrealistic requirements to get thought of as realistic. Our main mission was to find a way to filter these out so this was not really enough.

3.2 Zero shot method

In our project, one of the most challenging aspects we encountered was how to effectively measure "entry-levelness." During Week 4 of the course, we were introduced to pre-trained Large Language Models (LLMs) and their potential applications. We discovered that these pre-trained models could be leveraged

for a wide range of tasks, including classification. One particularly intriguing approach we explored was zero-shot learning. Zero-shot learning became our choice because it addressed the complexities of detecting intricate patterns within textual data, given our constraints in time and data availability. The key advantage of zero-shot learning is that it eliminates the need for labeled data and extensive training, making it a valuable technique for our project. Initially, we attempted to employ OpenAI's API with the gpt-3.5-turbo model. However, we were dissatisfied with the accuracy of this model in this spesific context. Subsequently, we delved into the models available on Hugging Face, where we came across the (MoritzLaurer/mDeBERTa-v3-base-xnli-multilingual-nli-2mil7) model. We regret not having more time to thoroughly evaluate and compare the performance of various models and labeling techniques. Nevertheless, with the MoritzLaurer/mDeBERTa-v3-base-xnli-multilingual-nli-2mil7 model, we achieved impressive results. With the model, we found out that using "requires job experience" and "does not require previous experience" as labels yielded really good results. We integrated the model directly into our application, allowing us to input job descriptions and obtain output labels enriched with the model's calculated probabilities based on the input text. Our journey in zero-shot classification led us from initial experiments with the OpenAI model to the discovery of a more effective solution on Hugging Face. This approach has enabled us to successfully identify job postings that explicitly require prior work experience and seamlessly integrate the model into our application for real-time analysis.

4 User interface

We developed our app using on click scraping which unfortunately could not be deployed. LinkedIn lines their policy clearly here and in robots.txt. To avoid any violations in their policies we deployed our demo app using static data instead of scraping it on click. We used Streamlit to easily deploy the demo application which can be visited (here). The hosted site uses predefined names as search parameters instead of typing keywords. The data and names used in this site come from a baseline search/crawl that you can find (here). Alternatively, the user can clone the repository and run the app with the scraper at their own risk. The hosted version of the application is stored in file called (stappdemo.py) and the version with real-time scraping (not hosted) (stapp.py)

5 Conclusion

Overall, we are rather satisfied with the outcome of the project. We did what we aimed to do (after reducing the scope from the initial idea) and learned about web scraping, text data cleaning and wrangling, text-based ML, LLMs and web app development.

In hindsight, we could've done the following things differently:

- Project scope: we could've spent a bit more time and effort in narrowing down the project of the scope at the start. The original vision was too ambitious, and it's easier in may ways to start working on a simple idea and expand it if time allows instead of doing the opposite.
- Framework for evaluating rankers: our approach to comparing different rankers was heuristic and not based on a pre-defined set of metrics or evaluation criteria. We assume that in "real-life" machine learning projects it's essential to spend more time in defining what needs to be maximized/minimized since the outcome of a project will reflect its goals.
- Given more time and a clearer initial scope, we could have developed the app to cater to job seekers at various experience levels, not just those entering the job market. Different levels of job seekers might have unique requirements, and the app could have been adapted to provide personalized insights and recommendations for each group.
- Data and Privacy: We could have taken a more proactive approach in addressing data and privacy concerns associated with scraping LinkedIn, even though the data is publicly available. Early engagement with these issues would have allowed us to establish a more robust framework for handling user data responsibly and in compliance with privacy regulations.

At least The following things could be done next to improve the app further and help out a broader user base:

- Expand the app to any geographies defined by the user
- Expand the job scope from software and data related ones
- Expand job seniority labeling to e.g. mid-level and senior jobs
- Move from binary labeling to scoring jobs based on relevance
- Show more data about the jobs in the UI instead of just the job post URL
- Sort out the legal uncertainties regarding LinkedIn scraping
- Scrape more job post platforms than just LinkedIn