Extraction of Skills and Benefits from Job Postings Descriptions

NI-MVI Semestral Work Report

Matus Botek, botekmat@fit.cvut.cz

31.12.2023

1 Assignment

The goal of this project is to develop a method for extracting relevant skills and benefits from job posting descriptions. Processing speed is very important so further optimizations should be applied to the initial solution to enhance it. Such improvements should be bench-marked. The last step is the labeling of a subset of job postings to validate my approach by calculating Precision, Recall, F1-score.

2 Data

2.1 Job Posting Dataset

I used Kaggle's Indeed Job Posting Dataset[1] which contains 30,000 web-scraped job descriptions and directly attached it to my Kaggle Notebook that I used for running my code. Firstly, I dropped job description duplicates. The texts are present in raw HTML format, so I removed HTML tags using BeautifulSoup4, split the descriptions into sentences using nltk library, further split the sentences by newlines (I wanted to preserve information partitioning from bullet points etc.) and removed some special characters after I found out it improved the similarity scores between sentences and ESCO skills by some fraction.

Each job description got cleaned and split into a list of sentences that then served as an input for the extraction method.

[1] https://www.kaggle.com/datasets/promptcloud/indeed-job-posting-dataset

2.2 ESCO skills

For the ontology, I used ESCO skills dataset[2] that contains over 13,000 skills. As preprocessing, following the mentioned paper, I detected and removed expressions enclosed by () brackets. This improved similarity scores for some entries. I also displayed the word count frequencies in Figure 1, this later helped me to determine the size of n-grams to use when extracting skills (similar to the mentioned paper), which will be mentioned later.

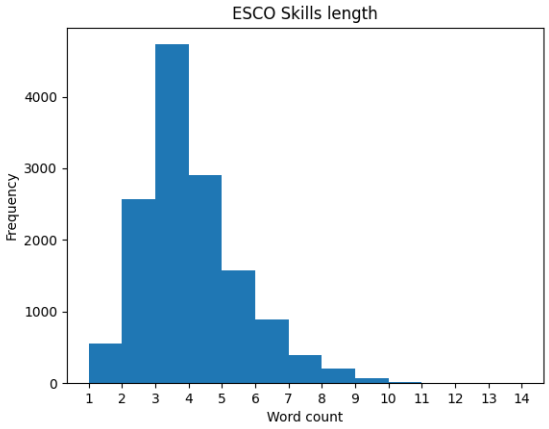

Figure 1: ESCO Skills word counts

2.3 Dataset labelling

My task was also to label a subset of data. For the labelling approach, I chose to use indexes of ESCO

[2] https://esco.ec.europa.eu/en/use-esco/download

skills as labels, because I planned to extract texts already mapped to ESCO skills based on similarity. For the baseline, I embedded whole job sentences and I did not want to mark the whole sentence as a skill after a successful match. This might mean that some skills that are present in the description might not be picked up, but the outputs will be more generalized over the dataset. I found out that the task was harder than I expected so I ended up manually labelling only 4 randomly selected job ads.

Another method I considered was span-level labeling using a tool for quicker and easier data labeling. This approach could help extract less-represented skills but wasn't attempted due to time constraints. I did not have time to try this approach so this could be a future improvement.

3 My approach

3.1 Embeddings and vector similarity

I chose to extract skills based on vector similarity between each embedded job sentence and ESCO skill embeddings. This approach seemed promising in a paper where they used the same strategy to extract skill spans from job descriptions, using different encoders and encoding techniques for the ontology and job sentences.[1] For my baseline in terms of both processing speed and metrics I chose to compute similarity between the full sentence embedding (CLS token) and all ESCO skill embeddings, take the maximum and if it was bigger than a given threshold, map the sentence to according ESCO skill and extract it.

The author of the weak supervision paper was also one of the authors of a SkillSpan[2] dataset which they created, annotated and then showcased it comparing multiple language models. One of the outputs was a domain-specific pre-trained BERT model called JobBERT[3] and they made it available on Huggingface. Papers show that domain-specific pre-training is an effective way to improve model performance on target task[3]. I manually compared embeddings created by BERT and JobBERT and I could see that the job sentences were finding better matches in ESCO skills using JobBERT. I decided to use it as my baseline model for embeddings.

I computed the JobBERT ESCO skill embeddings and stored them in a DataFrame to be accessed later so they don't have to be computed every time. I embedded the isolated skills without any further context.

My code was run in a Kaggle Notebook. This way I could utilize available GPU, 4 CPUs and recently increased 29GB of RAM.

3.2 Performance optimization

When running model inference for computing the embeddings, I used Kaggle's GPU P100 and did the other operations (vector similarity) on CPU. When I tried moving all the ESCO embeddings from the DataFrame to GPU the overhead was too big.

My initial method consisted of embedding each sentence in it's whole and then run a function that computed the similarity for each pair of *(sentence embedding, skill embedding)*, take the maximum value and if it was over a threshold, count it towards extracted skills after mapping the sentence to corresponding ESCO skill. At first I computed the vector similarity iteratively, but realized this was the biggest bottleneck. I optimized it in two steps, at first I computed the similarities between one sentence and all ESCO skills in one step by utilizing tensor matrix multiplication. The second improvement was to first compute all sentence embeddings, store them as a matrix and calculate similarities for all sentences and ESCO skills at once. I measured the time improvement on a randomly selected job posting which contained 37 sentences after pre-processing. Table 1 contains the measured times in seconds. The iterative method was very naive as I did not utilize the properties of tensors. In all other experiments I used the last, fastest implementation.

Similarity method	**Time (s)**
iterative	165.603
sentence x ESCO matrix	2.0319
sentence matrix x ESCO matrix	0.4503

Table 1: Performance times

[3] `https://huggingface.co/jjzha/jobbert-base-cased`

3.3 JobBERT and ESCOXLM-R for skill-extraction

In papers[2][4] they also published their models as ready-to-be-used demos on Huggingface. I was already using a version of JobBERT but only for embeddings, the ESCOXLM-R model, based on XLM-R is from a recent paper by the same author focusing on multilingual tasks in job domain, utilizing domain-adaptive pretraining on ESCO. Both of the models have their demos available[4][5] and use transformers pipelines to run the inference. The models perform a token classification task and output possible skills and knowledge. I regarded skills and knowledge both as skills and modified the code so that the outputs are mapped to ESCO skills as to be consistent with my baseline.

4 Experiments and Results

4.1 JobBERT embeddings vs token classification

At first I epxerimentid with different models for embeddings, but without domain-specific pre-training, the mapped ESCO skills did not make sence. I decided to work more only on **JobBERT embeddings** and **JobBERT model for token classification** as the ESCOXLM-R seemed as if it had some problems with the outputs, the extracted skills seemed to be misformatted, even though the demo worked fine on Huggingface. Before I labelled the data, I manually selected a couple of specific job ads and monitored the outputs of both models. JobBERT for token classification worked better from the beginning as it was able to export more skills compared to one skill per sentence when using JobBERT embeddings. Both models extracted relevant skills, but the first one was worse in terms of skill relevance.

I experimented with the threshold and it seemed that the best setting was using 0.8 for both models. For the token classification, I used two thresholds, first to filter out extracted skill spans, the second to filter out ESCO skills based on similarities.

As I said earlier, I labeled only 4 job descriptions as it proved to be quite a tedious task. At first a did a manual run, where I went sentence by sentence and tried to find the most important skills in each sentence by querying ESCO skills by sub-strings. After that I checked outputs of both of my models to see which similarities were most important and if the ESCO skills seemed relevant, I added them. Sometimes there were multiple similarly-formulated skill in ESCO, so when I labelled data with for example "communication skills" but the models found "communicating with customers" with high similarity, I changed the label. By this approach it is apparent that some skills that I found myself without the models will be very hard to extract so I did not expect very high metric scores. I was more focused on the difference between my two models.

I ran both models on the 4 annotated job postings and removed duplicate skill labels as they sometimes repeated throughout the description. This process produced 36 labels in total. Table 2 shows the metrics on the labelled subset. The first model performed very poorly and I expect that to be because I only embedded the whole sentences.

Model	Precision	Recall	F1-score
JobBert emb	0.1538	0.0541	0.0800
JobBert cls	0.3167	0.5135	0.3918

Table 2: Precision, Recall and F1-score metrics

The token classification model was significantly better than the first one in all of the metrics, but still the results were not satisfactory. I attribute this to the fact that I used ESCO indexes as labels, which in turn caused some correctly extracted skill spans to be mapped onto irrelevant ESCO skills and also to the fact that the ontology does not represent all values in enough detail.

4.2 JobBERT embeddings - ngrams

In the end, I also tried splitting the sentences into all possible n-grams of size 1-4 (as mentioned in the pa-

[4] https://huggingface.co/spaces/jjzha/skill_extraction_demo

[5] https://huggingface.co/spaces/jjzha/multilingual_skill_extraction

per[1]) and use two approaches. The first took only the maximum similarity match for all sentence n-grams, the other accepted all above a certain threshold. For each n-gram, I computed the final embedding as a weighted sum of n-gram embedding and whole sentence embedding. The threshold for accepting an n-gram was 0.85, lower values resulted in worse Precision. I only tried a couple of manual options for the weights and chose to use 0.8 and 0.2 for the n-gram and sentence. Both approaches improved the Recall and F1-score metrics slightly, but more experiments could have been made to find the optimal setup. The first approach chose mostly 1-word n-grams as they had the highest similarity scores, this could have been further modified to take the size of span into account, as bigger spans might lead to more relevant skills. Table 3 contains results for the two mentioned n-gram approaches.

Model	Precision	Recall	F1-score
n-gram max	0.1562	0.2703	0.1980
n-gram th	0.1136	0.4054	0.1775

Table 3: N-grams: Precision, Recall and F1-score metrics

Both of them performed worse than the token classification model. N-gram max stands for taking only the maximum for each sentence, n-gram th takes all n-grams with similarity above the threshold. This approach was much slower due to the huge number of embedding computations. It ran on the labelled sample for around 60 seconds.

4.3 Job Benefits

I was trying to find a suitable datasource that could be used as an ontology for job benefits, but the best dataset I found was a Kaggle dataset[6] which contained artificially-made job postings. I extracted all unique values from benefits column and ended up with 35 values. I felt like this would be a too small ontology, but I did not have time to label my data and try using it.

[6] `https://www.kaggle.com/datasets/ravindrasinghrana/job-description-dataset`

5 Conclusion and Further work

Overall, the results achieved were not very promising, but I saw that the token classification model was better at skill extraction. Changing from whole sentences to n-grams for the embeddings approach also improved the metrics. The results might have been different if more data was labelled or if I chose a different labelling strategy. My experiments could be used as a base for more work in this area, utilizing more advanced approaches.

For further work, more embedding computations could have been tried as the paper about weak supervision extraction[1] mentioned some more advanced methods that significantly improved the metrics. Also, more models apart from the domain-adapted JobBERT could have been experimented with, fine-tuned on some other job domain task, for example using one of labelled job postings datasets like Sayfullina or Skillspan to obtain better embeddings.

References

1. ZHANG, Mike; JENSEN, Kristian Nørgaard; GOOT, Rob van der; PLANK, Barbara. *Skill Extraction from Job Postings using Weak Supervision*. 2022. Available from arXiv: `2209.08071 [cs.CL]`.
2. ZHANG, Mike; JENSEN, Kristian Nørgaard; SONNIKS, Sif Dam; PLANK, Barbara. *SkillSpan: Hard and Soft Skill Extraction from English Job Postings*. 2022. Available from arXiv: `2204.12811 [cs.CL]`.
3. GURURANGAN, Suchin; MARASOVIĆ, Ana; SWAYAMDIPTA, Swabha; LO, Kyle; BELTAGY, Iz; DOWNEY, Doug; SMITH, Noah A. Don't Stop Pretraining: Adapt Language Models to Domains and Tasks. In: JURAFSKY, Dan; CHAI, Joyce; SCHLUTER, Natalie; TETREAULT, Joel (eds.). *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Online: Association for Computational Linguistics, 2020, pp. 8342–8360.

Available from DOI: 10.18653/v1/2020.acl-main.740.

4. ZHANG, Mike; GOOT, Rob van der; PLANK, Barbara. ESCOXLM-R: Multilingual Taxonomy-driven Pre-training for the Job Market Domain. In: ROGERS, Anna; BOYD-GRABER, Jordan; OKAZAKI, Naoaki (eds.). *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Toronto, Canada: Association for Computational Linguistics, 2023, pp. 11871–11890. Available from DOI: 10.18653/v1/2023.acl-long.662.