Job postings skills classification

Filip Kašpar
ČVUT - FIT
kaspafil@fit.cvut.cz

January 8, 2025

1 Introduction

The goal of this semester project was to extract skills from job postings. First step was to train a model capable of classifying raw skill data into predefined normalized skill categories. The dataset contains over 1.5 million entries of raw skills paired with their normalized equivalents. The objective is to enhance the accuracy of skill extraction from job postings, making it easier to summarize the requirements from those postings.

This was also my first AI-related course and project, so I spent considerable time researching every aspect of the technologies used, starting from the fundamentals. As a result, I didn't have much time to explore other potential solutions.

2 Input data

Two datasets were used for this task. The first dataset contained the raw and normalized skills provided by the class tutor. The first dataset contained over 1.5 million pairs of entries, which was quite substantial. It was shared in the following format:

raw skill	normalized skill
.Net Framework 3.5 and above	Microsoft .NET
.NET application architecture	Microsoft .NET
3D graphics for mobile devices	3D Graphics
...	...

Table 1: Few records from the dataset

Since this dataset was preprocessed from the beginning, extra preprocessing was unnecessary.

The second dataset was the job posting dataset. The dataset contained information about **job_title** and **job_description**. The dataset also included information about the location where was the job post published, but that wasn't important for this task. The job descriptions were in HTML format so some preprocessing had to be done.

The second dataset can be found on kaggle: Indeed Job Posting Dataset

3 Methods

For the first part of the project, I implemented the LSTM RNN and Encoder-only Transformer models, using both the basic-english and bert-base-uncased tokenizers. I chose these models because they are commonly recommended in various papers and blogs for NLP tasks, which also influenced my decision to use the BERT tokenizer. The basic-english tokenizer was included to observe how the models perform with different tokenization methods.

In addition, an optimizer and scheduler were used in the project. While I didn't experiment much with them, I focused more on understanding their purpose. Adam was chosen as the optimizer due to its widespread use and effectiveness. For the scheduler, I used the CosineAnnealingLR scheduler.

In the second part, one of the models was used to extract skills from the job postings. The job posting has been split into multiple sentences. Each of these sentences has been evaluated using the model and the output was a skill predicted by the model for each sentence. If the model has little confidence in the predicted skill, the skill is disregarded. The threshold for the confidence can be modified in the code; the default value is **70%**.

The output of the second part is a list of skills for a given job posting sorted by frequency, with the most frequent skills appearing first, from left to right. The program is in **extract_skills.py** file.

4 Results

First, the LSTM model was implemented using the basic-english tokenizer. After that, I spent some time experimenting with different hyperparameters. This initial model achieved approximately **87%** accuracy and a **0.72** loss on the testing data.

The Transformer model, also using the basic-english tokenizer, performed slightly worse, achieving around **84%** accuracy with a loss of **0.77**.

Things got more interesting after implementing the BERT tokenizer. With the BERT tokenizer, the LSTM model's performance improved slightly, reaching about **88%** accuracy and a loss of

0.7. While this isn't a dramatic improvement over the basic-english tokenizer, the Transformer model showed more notable gains. Its accuracy rose to around **86.5%**, and its loss dropped significantly to **0.58**, which is much lower than the LSTM's loss, which generally struggled to go below **0.7**.

Details about the hyper-parameters used for the LSTM RNN and the transformer model can be found in folder **trained_models/** in files **bert_lstm_nodrop_stats.txt** and **bert_transform_nodrop_stats.txt** respectively.

The accuracy and loss for both models on training data with the BERT tokenizer are illustrated here:

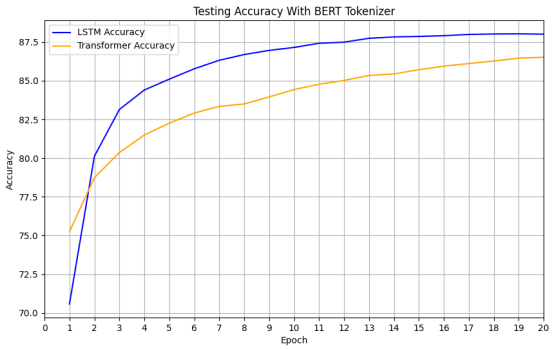

Figure 1: Models accuracy

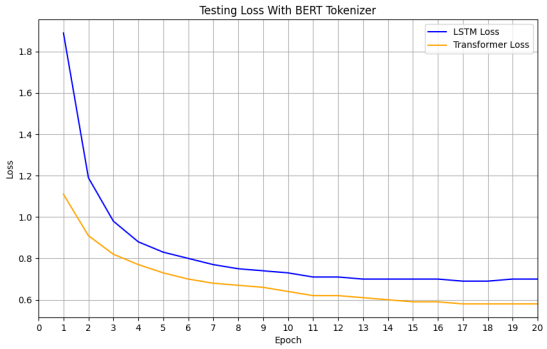

Figure 2: Models loss

As shown in 1 and 2, both models perform reasonably well across different criteria.

In terms of precision, recall, and F1 score, the model with LSTM and basic-english tokenizer achieved **0.8787** in precision, **0.8798** recall, and **0.8759** F1 score. That is pretty decent for a skill classification model. The metrics can be seen in **evaluate.py** file.

For the second part of the project; getting the skills from the job posting, I tested a bunch of the job postings from the dataset. I tested those job postings manually. First I read the job posting and then I ran it through the model and checked if the outputted skills make sense considering the meaning of the job posting.

Since I evaluated this part manually I don't have statistics for it, but with the threshold for accepting a predicted skill being **70%**, the model performed very well. In every job posting I tested, the model got the main skills right, and very rarely included a wrong skill.

The final results for the job postings I manually checked are accessible in **job_postings_skills.txt** file.

5 Conclusion

I think achieving **0.8759** F1 score and **88%** accuracy is quite decent for the skill classification model. Therefore I think the first part was pretty successful. The second part of this project, the extraction of skills from job postings yielded pretty impressive results as well. When I manually checked it, all of the main skills of the job postings were correctly recognized. Therefore I would say that this solution for extracting skills from job postings works pretty well.

That said, there's still plenty of room for improvement. For instance, different transformer models could be explored, and techniques like grid search could be used to find better hyperparameters for the model. Experimenting with various optimizers and learning rate schedulers could also yield improvements. Additionally, trying out different tokenizers and embedders might help.

References

[1] Mohammad Ahmed. Technical skill assessment using machine learning and artificial intelligence algorithm. *International Journal of Engineering Research and*, V8, 12 2019.

[2] Ditria Luke. Github pytorch tutorials. 2024. `https://github.com/LukeDitria/pytorch_tutorials`.

[3] Meng Zhao, Faizan Javed, Ferosh Jacob, and Matt McNair. Skill: A system for skill identification and normalization. *Proceedings of the AAAI Conference on Artificial Intelligence*, 29:4012–4017, 01 2015.