THE GEORGE WASHINGTON UNIVERSITY
WASHINGTON, DC

Natural Language Processing
Final Project
October 15, 2024
Amir Jafari
Due: *Check the Syllabus*

Name: Aravinda Vijayaram Kumar

GWID: G36084456

Introduction

When we first started working on this project, our goal was to build a system that could help align candidate resumes with job descriptions and give feedback on how to improve it. Over time, the scope expanded significantly, and we became involved in designing multiple features: analyzing similarity between resumes and job descriptions, generating resumes, providing visualization tools, ranking resumes based on their suitability for a given job, recommending relevant jobs to candidates, and evaluating grammar and formatting in resumes. The final product integrated all these components into a Streamlit application, creating a comprehensive platform where users can upload their resumes, paste job descriptions, and then explore a host of functionalities such as similarity scoring, skill gap analysis, resume generation, and other visualization techniques.

Initially, the project was simply about finding a good model for similarity and feedback with grammar checks and exploring how to compare a resume to a job description. But as I began adding more capabilities—like generating formatted resumes based on instructions and identifying missing skills in a candidate's profile—we realized that each component influenced and complemented the others. For instance, the similarity score and feedback from skill gap analysis could guide improvements to the user to add missing skills, help highlight what needed to be added or improved in the resume. This interplay eventually led us to develop a more holistic system rather than just a single model or script.

Description of My Individual Work

Most of my personal efforts revolved around the resume generation module and its integration with other features like skill gap analysis and visualization tools. We had decided to use the dataset from Hugging face with prompt instructions and resume text (https://huggingface.co/datasets/InferencePrince555/Resume-Dataset/viewer). The average resume length in the dataset was around 900 words with 52 different prompts related to 52 fields. The resume data for Software Engineer was highest with 5828 resumes followed by Systems Administrator, and lowest was BPO job with 22 resumes.

When I first tried to generate resumes from raw text using models like T5-small or T5-base, I faced numerous challenges. The generated resumes were often repetitive, truncated, and not very meaningful. The maximum token length of 512 for T5-small and T5-base was a significant limitation, causing the output to get cut off or lose coherence. Also, the model often produced repetitive phrases that were not formatted properly.

To address these issues, I switched over to BART-base. BART provided 1024 tokens for maximum length, which immediately gave the model more context to produce well-structured outputs. After switching to BART-base, I embarked on a series of experiments to improve the model's performance. I utilized the Hugging Face Trainer class for training, which provided a high-level API for fine-tuning the model on my dataset. Initially, I simply passed the tokenized dataset to the trainer with default parameters, but the results were only slightly better than T5. The generated resumes were still somewhat repetitive and lacked proper formatting.

```
model_name = "facebook/bart-base"
tokenizer = BartTokenizer.from_pretrained(model_name)
model = BartForConditionalGeneration.from_pretrained(model_name)

training_args = TrainingArguments(
    output_dir="./resume_generator_bart",
    evaluation_strategy="epoch",
    learning_rate=5e-5,
    per_device_train_batch_size=8,
    per_device_eval_batch_size=8,
    num_train_epochs=5,
    save_strategy="epoch",
    logging_steps=50,
    save_total_limit=2
)

trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=tokenized_train,
    eval_dataset=tokenized_test,
    tokenizer=tokenizer
)
```

With this initial setup, the model trained, but the resumes still had recurring issues. To tackle this, I started tuning parameters one after another. I tried increasing the number of epochs from 5 to 15, which allowed the model to see the data more times and better learn the mapping from instructions to resumes. I also experimented with a higher learning rate (e.g., 5e-4) and tried using mixed-precision training (fp16=True) to speed up training and potentially improve the model's convergence.

After training, I focused on generation parameters at inference time. Using top_k and top_p sampling controls the diversity of the generated text, while adjusting temperature influences the randomness. Setting a repetition penalty helped reduce repetitive lines. For example:

```
    outputs = model.generate(
        inputs["input_ids"],
```

```
        max_length=512,
        top_k=50,
        top_p=0.95,
        temperature=5.0,         # Higher temperature for more diverse output
        repetition_penalty=3.0, # Penalizing repetition
        num_beams=4,             # Beam search for more coherent results
        no_repeat_ngram_size=3, # Preventing repetition certain n-grams
        early_stopping=True)

```

By iterating over these parameters, I could gradually improve the coherence and structure of the generated resumes. However, the underlying dataset contained raw text cluttered with HTML tags, invalid spans, and excessive whitespace. To address this, I added a preprocessing step that cleaned and restructured the input data before training. I used regex and custom functions to remove unwanted tags and extract meaningful sections (like “Professional Summary,” “Work History,” and “Skills”) from the raw text. I identified these section headers by scanning the entire dataset and selecting those that occurred at least three times, ensuring I focused on consistent and relevant headers.

At some point, I realized that the model was generating resumes that had better structure but still sometimes repeated lines or produced less informative content. To mitigate this, I implemented a final postprocessing step. After generation, I would parse the output to remove duplicate lines and reformat sections. By adding bullet points and standardizing capitalization, I achieved more human-readable results. On average, after these improvements, BLEU scores for generated resumes ranged from about 0.4 to 0.7 when tested on random samples of 10 to 100 resumes from the test dataset. While these scores may not be perfect, they marked a significant improvement from the initial attempts that produced nearly unusable text.

Skill Gap Analysis

Initially, I tried a very straightforward similarity approach to identify skill gaps, using simple cosine similarity between TF-IDF or embedding vectors of the resume and the job description. However, I soon realized that a more semantic approach was needed. The cosine similarity approach was too literal and did not capture nuanced relationships between words. For example, “Python development experience” in the job description might not match exactly with “skilled in Python for web applications” in the resume due to differing phrasing, even though semantically they mean something similar.

To improve this, I integrated spaCy’s semantic similarity. SpaCy’s pre-trained word embeddings allowed me to compute similarity scores between entire phrases, making it possible to detect missing skills or experience more accurately. I would first extract the job requirements and the candidate’s listed skills, experience, and qualifications, then use spaCy’s nlp model to compute similarity. If the similarity fell below a certain threshold, I considered that skill or experience missing from the resume. From these extracted data I constructed a feedback that was then displayed to the user on the UI.

Finally, I integrated these modules—resume generation, skill gap analysis, along with other small changes in visualization—into a single Streamlit interface. Users can upload their resume PDFs and paste job descriptions, and the app displays similarity scores, identifies skill gaps, and generates plots like word clouds, Venn diagrams, and heatmaps to help users understand the data distribution and how their resume compares to the job posting.

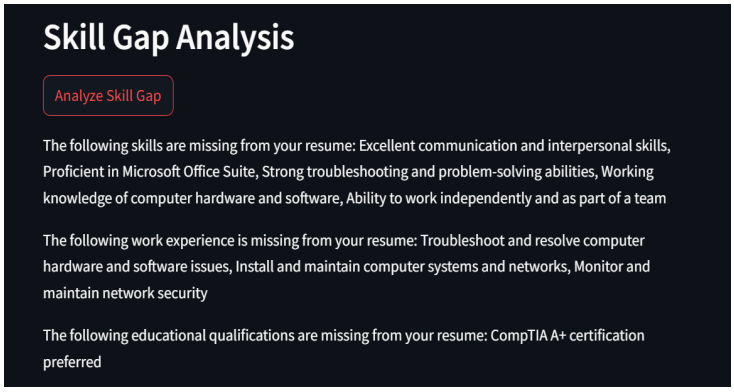
In the Streamlit app, I combined the functions and provided an interactive UI. For instance, the user can drag and drop a resume, adjust weights for similarity scoring, and click a button to analyze skill gaps. They can also open another tab to generate a resume based on a prompt or request a ranked list of resumes from a folder for a specific job post. The grammar and formatting check was integrated as well, giving users a chat-like interface to review suggestions. On the Streamlit app side I worked on the basic structure of it, adding some changes in visualization and a few other small changes while other features were integrated by my teammates.

Results

The final system functioned as a multi-faceted tool. The resume generation quality, while not perfect, became substantially better than when I started. I relied on BLEU scores to check the quality of the generated resumes. Due to computational and time constraints, it was not feasible to compute BLEU scores for all 3,300 resumes in the test set. Instead, I sampled subsets of the test set ranging from 10 to 100 randomly selected resume and calculated BLEU scores on these smaller batches. This sampling approach allowed me to rapidly iterate and assess the impact of various preprocessing steps, hyperparameter changes, and model adjustments without incurring excessively long evaluation times.

Initially, the BLEU scores were quite low, with many generated resumes containing repetitive or truncated content. However, after switching to BART-base, introducing better preprocessing and postprocessing techniques, and experimenting with different generation parameters such as top_k, top_p, temperature, and repetition_penalty, the BLEU scores improved notably. On these sampled subsets of the test data, BLEU scores consistently ranged from about 0.4 to 0.7. While these scores do not represent the entire distribution of test resumes and may not be definitive indicators of real-world performance, they serve as meaningful checkpoints. They confirmed that the changes I made especially preprocessing, section extraction, data cleaning, and hyperparameter tuning resulted in more coherent, better-structured resumes that were closer in content and form to the ground-truth references in the dataset. More importantly, from a qualitative perspective, the generated resumes were now readable, properly segmented, and included relevant sections.

The skill gap analysis provided meaningful insights. For instance, if the job asked for administration and communication experience, and the candidate's resume only mentioned certain general skills, the system would highlight that the candidate was missing experience with these frameworks. Similarly, if a certain degree level was required, the system checked the educational background to confirm its presence. Below is one such example attached -



The visualization options also helped users understand the data more intuitively. A word cloud of the job description could emphasize key terms, and a Venn diagram could quickly show what skill sets overlapped. The resume ranking module allowed a recruiter to upload multiple candidate resumes and see which ones were more closely aligned with the job description. Finally, the job recommendation system analyzed the candidate's resume and proposed suitable job listings.

All of these results hinged on careful preprocessing, hyperparameter tuning, and iterative testing. The project taught me that text generation models benefit immensely from clean, well-structured data. It also underscored the importance of trying different approaches for model training, evaluation, and balancing the dataset.

Summary and Conclusions

Throughout this journey, I learned that resume generation is not just about picking a good model; it's about providing that model with well-preprocessed data, controlling its generation parameters, and iteratively refining the pipeline. Initially, I struggled with truncated and repetitive outputs, but by switching models, adjusting hyperparameters, and introducing preprocessing steps, I managed to produce coherent, sectioned resumes that made sense to the end-user.

The skill gap analysis and similarity scoring provided valuable feedback loops. Users can now understand what they're missing and improve their resumes accordingly. Adding visualization tools made the results easier to digest, and integrating grammar and formatting checks ensured that the final output was not only contextually relevant but also stylistically polished.

In the future, I could further refine the model by adding more domain-specific training data, experimenting with newer model architectures, or introducing feedback loops, more robust pre processing. Additionally, expanding the dataset to include a wider variety of industries and roles would improve the model's versatility and accuracy across different job sectors. Another area for improvement is the integration of real-time user feedback into the training loop. By allowing users to provide ratings or corrections to generated resumes, the system can continuously learn and adapt, leading to progressively better performance. Implementing active learning techniques

where the model identifies uncertain or low-confidence areas and seeks user input can also enhance the quality and reliability of the generated resumes.

In summary, this project has not only demonstrated the feasibility of automating resume generation and analysis but also highlighted the importance of data quality, model selection, and iterative optimization in developing effective NLP applications. The insights gained from this work lay a strong foundation for my future enhancements, promising even more better work in the future.

Percentage of Code Sourced Online

Total lines – 1004

Borrowed from external – 120

Modified -70 Added- 40

Percentage = (120-70/120+40) *100 = 31.25%

References

1. https://github.com/611noorsaeed/Resume-Screening-App/tree/main
2. https://github.com/lil-Ribhav-Bhatt1012/Resume-Matchers/tree/master
3. https://github.com/OmkarPathak/pyresparser
4. https://github.com/srbhr/Resume-Matcher
5. https://www.analyticsvidhya.com/blog/2021/06/resume-screening-with-natural-language-processing-in-python/
6. https://kartikmadan11.medium.com/building-a-job-description-to-resume-matcher-using-natural-language-processing-5a4f5
7. 181cfe4