NLP Semantic Analysis for Job Applications

ECS 170: Final Project

Group 16

Anish Ahuja, Joshua Sechrist, Aiperi Baktybekova, Aaron Bernstein, Joseph Lee, Nedaa Aljalab, Alexis Lopez, Malvina Singh, Zeeva Chaver

Overview

In the current competitive job market, it is challenging to tailor a resume to every job description. The motivation of this project is to develop a browser extension that analyzes and contrasts resumes with job postings using artificial intelligence (AI). The browser extension would provide feedback for resume optimization, such as similarity scores between a user's resume and a target job posting, highlighting missing lines, as well as strengths in the form of semantically matching lines. This project explores how pre-trained language models can used in real world applications of natural language processing (NLP) tasks, while addressing practical challenges of sourcing data, model bias and system integration. This project implements and evaluates various AI models like BERT (Bidirectional Encoder Representations from Transformers), Word2Vec, and BART (Bidirectional and Auto-Regressive Transformer) that capture semantic similarities between text, beyond the simple matching of keywords. The system uses baseline, non-AI methods for comparison purposes, which include text overlap and TF-IDF (Term Frequency-Inverse Document Frequency) algorithms.

Background

Word embeddings are a central technique in NLP, as vector representations of words that capture their semantic meaning. With these vectors, relationships between words can be inferred through measuring their vector similarity. Early approaches of word embedding, such as one-hot encoding, bag of words, and TF-IDF, represent words without the ability to capture their context in a sentence. This issue is addressed in modern NLP models that use word embeddings generated by neural networks, which can create vector representations that capture the contextual relationships between a word and its surrounding sentence. Models like Word2Vec learn word relationships from large document sets, while transformer models, like BERT and BART, generate token embeddings by considering the context of the words before and after it. This project applies and compares multiple vector representations of words, including static and dynamic embeddings generated by AI models, to evaluate how effectively they capture semantic overlap between texts. For the baseline portion of this project, non-AI text comparison methods that emphasize simplicity, interpretability, and efficiency were implemented. These techniques, text overlap and TF-IDF, rely on word frequency and occurrence rather than semantic understanding. Although these approaches do not capture contextual meaning or synonym relationships, they provide transparent similarity scores and keyword comparisons that establish a clear performance baseline for evaluating more advanced neural network models.

Methodology

Three different AI models were implemented to compute semantic similarity between a resume and a job description. With the goal of identifying overlaps between a candidate's experience and the requirements in a specific job listing, similarity scores were generated between the overall files, as well as between all line pairs. The outputs of the models were organized to present a candidate's strengths as line pairs between the resume and job description that had high similarity scores, with each model requiring a slightly different similarity threshold to gather valuable results. Weaknesses in a user's resume were presented as job description lines without resume matches. For each method, non-AI software for text processing and output formatting were implemented. Text processing included splitting the resume and job description texts into individual lines, split on whitespace, considering that resumes are often structured non-grammatically (without punctuation). Each model's output was returned in JSON format, requiring processing to sort through the job description and resume line pairs. Since each AI model generated embedding vectors for individual words (tokens) in a line, entire line embedding vectors were computed by taking the mean of the token embeddings in that line. Embedding vectors were computed for an entire text by taking the mean of all the line embeddings in that text. For each AI model, cosine similarity was calculated between files, as well as between all possible job description and resume line pairs. Cosine similarity is computed for two lines (or files) by taking the cosine of the angle between their embeddings (vectors). The range of cosine similarity is [-1,1], where 1 represents identical direction and semantic equality.

The first AI method that was implemented for similarity was the BERT model. BERT is a transformer-based language model developed by Google, which produces word embeddings that take into account a relatively deep level of sentence context. This context is learned by the self-attention mechanism, which weighs the importance of tokens in a sequence to understand their relatedness. This project utilized the pretrained "google-bert/bert-base-uncased" model from the HuggingFace Transformers library. For the BERT model, the most valuable resume and job description pairs were found above a threshold above 0.75.

The second AI method that was implemented for similarity was the Word2Vec model, specifically the pretrained "word2vec-google-news-300" model from the Gensim library. Word2Vec is a shallow, two-layer neural network model that produces word embeddings where semantically similar words have similar vectors. Word2Vec does not capture sentence context like a transformer model, instead representing words that appear in similar linguistic contexts as similarly valued vectors in a multi-dimensional vector space. This particular model contains 3 million word vectors trained on 100 billion tokens and is well trained on resume and job description language, as workplace terminology is well-represented in news corpora. Based on tutorials for this model, text preprocessing came in the form of a Gensim function "simple_preprocess". Valuable line pairs were returned above a similarity threshold of 0.55.

The third AI method that was implemented for similarity was the BART (Bidirectional and Auto-Regressive Transformers) model, specifically the "facebook/bart-large" model from the HuggingFace Transformers library. The goal of including this model was to implement similarity analysis like BERT, using a model that could generate replacement sentences in later implementations according to similarity pairs. BART is a sequence-to-sequence model combining denoising autoencoder pretraining with a transformer architecture. For

text preprocessing, texts from the resume and job description were converted to lowercase and split into lines. Meaningful line pairs were returned above a similarity threshold of 0.9.

The baseline implementations for this project include text searching and comparison algorithms to estimate the similarity between documents. One implementation was a simple text overlap, which involves a few steps. First, the algorithm strips down words like articles, prepositions, and conjunctions, as well as punctuation to create a clean document. Then, the algorithm searches for words that are present in both documents to compute an integer result of matched words. Finally, the matched word count is divided by the total number of words in the shorter of the two provided documents to result in a similarity score. The TF-IDF algorithm measures the frequency of each term in a document, then multiplying it by the rarity of the term across all input documents, as words that are rarer across a document space imply importance and specificity. Words that are present multiple times are accounted for as a single object and word rarity is accounted for. However, TF-IDF still has its disadvantages; it cannot handle synonyms and has no ability to treat specific groups of words as one concept, which are necessary features needed for our semantic analysis application.

Another experiment in this project was generating replacement sentences using a large language model (LLM) for a user's resume based on a target job description. The intended output was producing rewritten resume lines according to highly similar sentence pairs between texts to help the user adopt keywords and job description specific phrasing for resume optimization purposes. To attempt this, the high similarity line pairs returned by BERT ("bert-base-uncased") and BART ("facebook/bart-large") were passed to BART for generation. Language generation was first attempted using the "facebook/bart-large" BART variant, which simply echoed the input prompt, likely due to not being fine-tuned for paraphrasing tasks and an input prompt that was too long. Next, the "eugenesiow/bart-paraphrase" BART variant from HuggingFace was used, a variant fine-tuned for semantic paraphrasing.

A Firefox browser extension was built to house the models, which reads data from Indeed.com. A user can upload their resume as a text file or copy and paste it into a text box. Then, for any job listing on Indeed, the extension can scrape the description and make an API call to a locally hosted Flask web server. Alternatively, a user can paste a job description into a text box for a job found outside of Indeed, allowing users to utilize different job listing sites. JavaScript/HTML were used for the frontend and a Flask server in Python was set up that calls the different algorithms and can be accessed via an API call. For testing purposes, we ran the server locally rather than hosting it on a third-party. If this project were to be launched as a product, it would be hosted externally and more endpoint logic would be added to verify calls are not being made from third-party sources. Refer to Figure 1 and 2 for examples of the browser extension usage on Indeed.com with a Software Engineer resume.

Results

Among the three AI models implemented, BERT and BART generated similar scores, which were higher than the scores from Word2Vec, reflecting the different architecture of the models' relative embeddings. Both BART and BERT dynamically create transformer-based

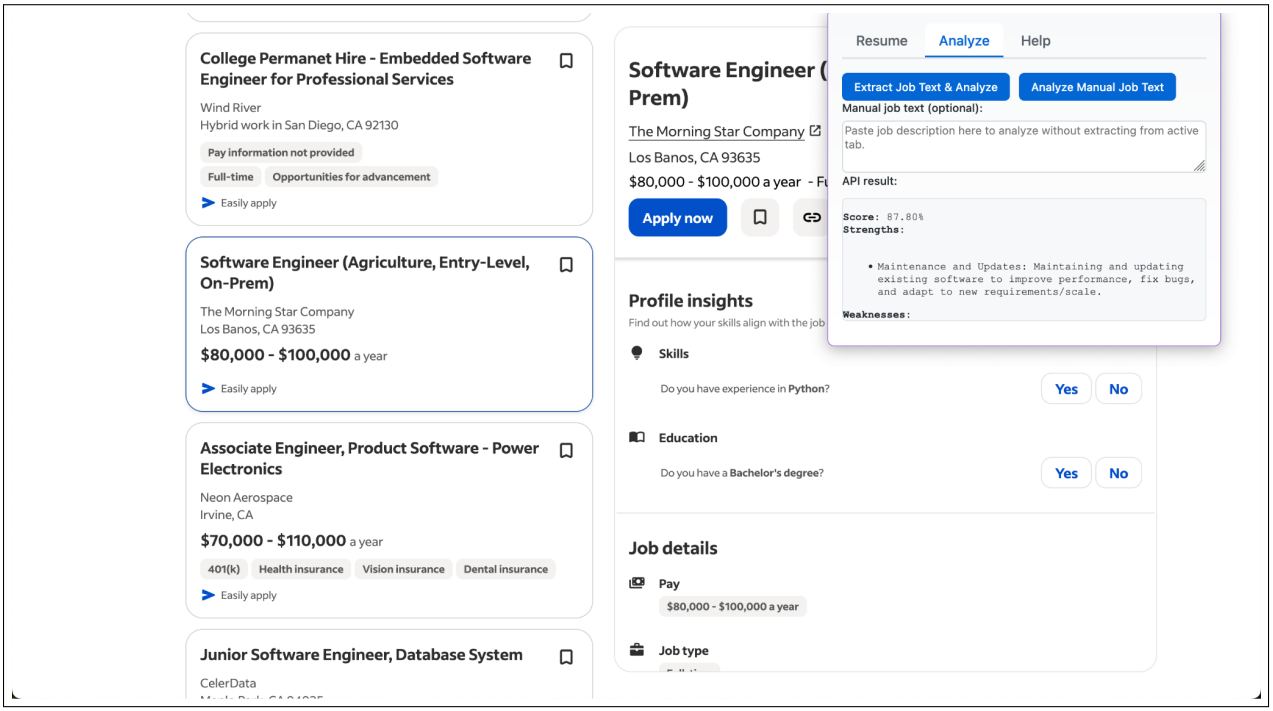

Figure 1: Browser extension with a Software Engineer position.

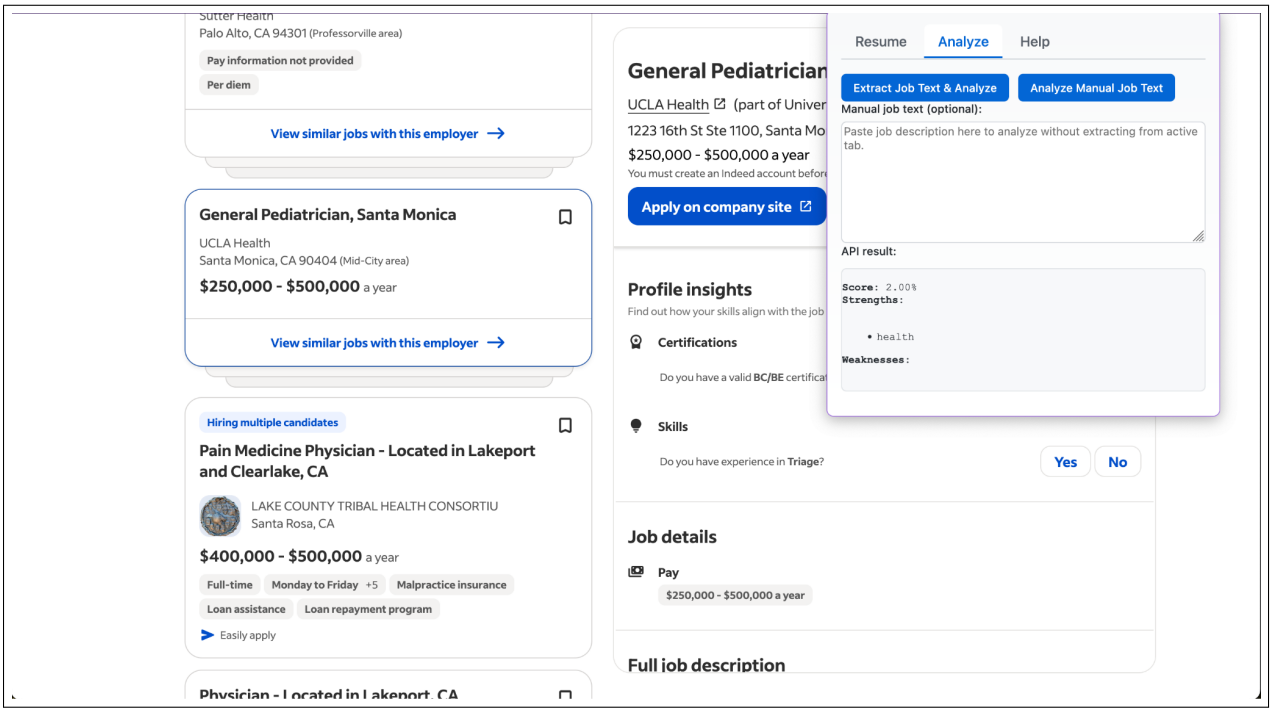

Figure 2: Browser extension with a healthcare (General Pediatrician) position.

embeddings for each new sentence read, which capture a token's context in a sentence using bidirectional attention mechanisms (taking into account the words both before and after a token). Word2Vec generates static word embeddings that have already been computed during pretraining, where every word has one fixed vector. Since Word2Vec doesn't utilize the context of the word's sentence, it inherently captures less context than a dynamic transformer model. Generally, even when a resume and job description line pair matched in meaning, Word2Vec assigned a lower similarity score due to exact vocabulary differences. Each model required different optimal threshold values for sentence similarity scores, increasing in the order of Word2Vec, BERT, and then BART. Since Word2Vec's static word embeddings led to lower cosine similarity scores, a much lower similarity score threshold was required in order to capture meaningful line matches between the texts. Refer to Figure 3 for a comparison of the semantic similarity scores, computed by each AI model and TF-IDF, between a Software Engineer resume and three different job postings, Construction Worker, Civil Engineer, and Software Engineer.

During testing, one challenge was that shorter sentences generated higher similarity scores across all models, compared to longer sentences expressing similar content. This was due to the construction of sentence embeddings by averaging the embeddings of individual words. Shorter lines tend to describe one simpler concept, which produces a sentence embedding that reflects one main idea. Longer sentences tend to contain multiple ideas, which produces a sentence embedding that reflects these several semantic components. This is common in resume writing practices where candidates combine ideas in one line to save page space and to connect their responsibilities with outcomes. Thus, when two longer lines overlapped on some subtopics, the greater noise in their word embeddings reduced their similarity score. Conversely, a short line that matched only one of the ideas in a longer sentence would often receive a higher similarity score because its embedding was more concentrated. Additionally, shorter sentences tend to contain fewer filler words or verbs, which reduces noise during the averaging process. Another challenge we observed was that some AI models would consume too many resources occasionally, and cause the device to freeze. While we were able to test all of our models at different times, future endeavors would focus on trying to offload computation to a third-party server.

For the baseline methods, the text overlap and TF-IDF methods produced comparable outputs, with differences that reflected the improvements gained when moving from a simple word-matching algorithm to a weighted text-processing algorithm. The text overlap method produced slightly higher similarity scores than TF-IDF. This was likely due to repeated words being counted multiple times in the text overlap method's output, which inflated the similarity score between the texts. For example, in some test files the text overlap method identified several occurrences of the terms "python" and "postgresql", giving them a disproportionate weight in similarity calculation. Additionally, the text overlap method didn't distinguish between technical keywords and filler or conversational terms. For example, in some test files this method highlighted words from the job description that have no relevance to the qualifications and do not meaningfully influence similarity, such as "we're" and "nice", as terms missing from the user's resume. TF-IDF addressed these issues, not counting repeated words as distinct matches and identified words relevant to the job as missing from the resume.

In terms of language generation, the chosen workflow and model did not perform well

for this task. This was likely due to how BERT and BART produced word embeddings that led to higher cosine similarity scores for short job description phrases, as discussed above, even with minimal semantic overlap. This caused many resume lines to be matched with shorter, generic job description phrases, which performed poorly in the BART paraphrasing model. In testing, this model often generated short job description lines rather than actual paraphrasing that combined related job description and resume lines. These outcomes reveal that successful language generation for this task requires stricter constraints on the computation of word embeddings, perhaps with a penalty on line length, to mitigate shorter lines having strong vector representations that contribute to disproportionately high cosine similarity scores. Additionally, giving a model more context for this task, by fine-tuning an LLM specifically for resume and job description writing, would improve the generated language, instead of relying on generic paraphrasing instructions.

Discussion

Comparing the baseline algorithms with AI-based models demonstrated that simpler methods such as text overlap and TF-IDF fail to capture deeper semantic relationships between resume content and job requirements. Models like BERT and BART proved to be more effective in identifying meaningful semantic matches and revealing strong and weak points in resumes that traditional methods were unable to reliably detect. These findings demonstrated the advantage of transformer based models for capturing semantic relationships that simpler baseline techniques fail to detect.

For testing, each model was used against three different job postings per industry, ranging from extremely low alignment to high alignment with a Software Engineer resume, with the average similarity plotted in a bar graph (Figure 3). AI-based models frequently assigned high similarity scores across industries. This is likely due to a baseline similarity between the nature of all job descriptions with a resume, as all contain career-oriented terminology. In contrast, baseline models tended to be more brittle when it came to similarity between a job description and a resume, searching for exact keyword matches, which created more pessimistic similarity scores. However, all models were able to understand some degree of alignment between a Software Engineer position with a Software Engineer resume, as all models consistently assigned their highest score to the third category.

Conclusion

This project successfully developed a functional browser extension that analyzes resume-job description alignment using both non-AI text-matching algorithms and advanced AI models. By comparing baseline methods like TF-IDF with transformer-based models such as BERT and BART, AI-driven approaches outperform simpler techniques in capturing semantic similarity and providing actionable feedback to job applicants. The system identifies strengths and weaknesses in resumes, offering users concrete insights to optimize their applications for specific roles. While challenges emerged around sentence length sensitivity and language generation, this project ultimately validated the practical value of integrating NLP models into real-world career development tools.

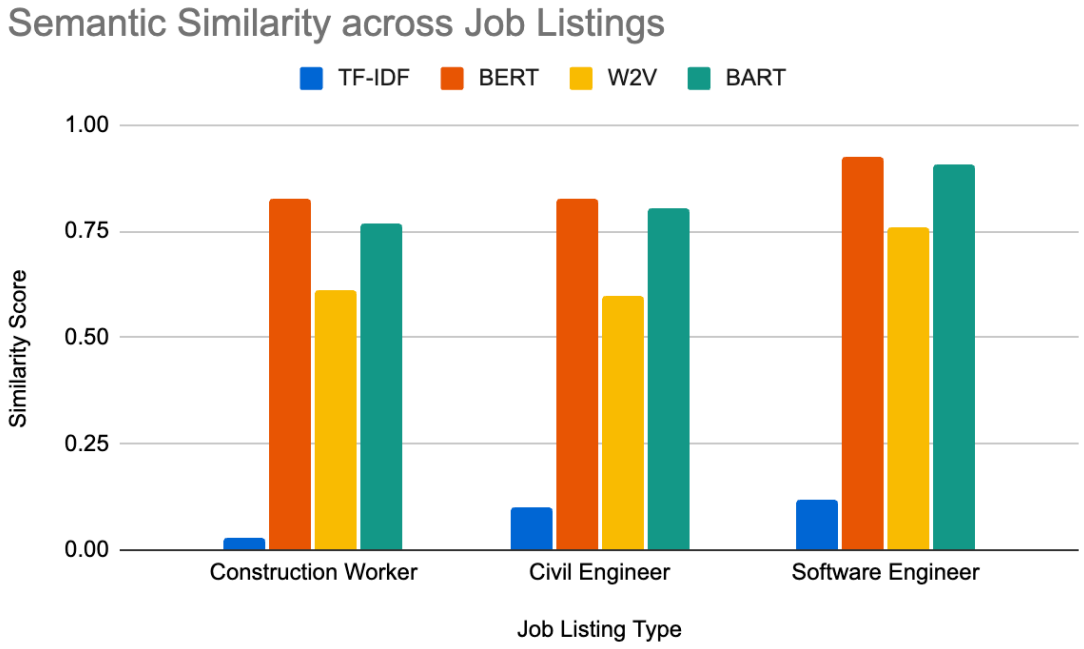

Figure 3: Comparing similarity scores across job listings.

Contribution

Zeeva Chaver: I implemented the similarity score calculations and line-pair matching using our AI methods, including BERT and Word2Vec, adapted the team's existing BART similarity code, and attempted language generation using different variants of BART.
Anish Ahuja: I developed the browser extension with functionality to take in resumes and job descriptions and call a backend server I made to communicate with our models, as well as created a few scripts for similarity scoring, primarily for BART.
Joseph Lee: I led the baseline research and implemented the core baseline algorithm.
Nedaa Aljalab: I assisted with baseline keyword search, TF–IDF vectorization, and cosine similarity analysis.
Joshua Sechrist: I contributed to baseline research, adapted baseline algorithms to the frontend application, and documented results and methodology for baseline models.
Aaron Bernstein: I contributed to frontend design and helped with tab functionality.
Aiperi Baktybekova: I did testing on my end, but was not able to make improvements to code or important changes.
Alexis Lopez: I assisted with the non-AI methods for carrying out our objective and comparing them with our AI approach.
Malvina Singh: I tested BART locally as an alternative to API-based LLMs for language generation, ran prompt experiments for resume rewriting, and reviewed generated outputs for quality and feasibility.

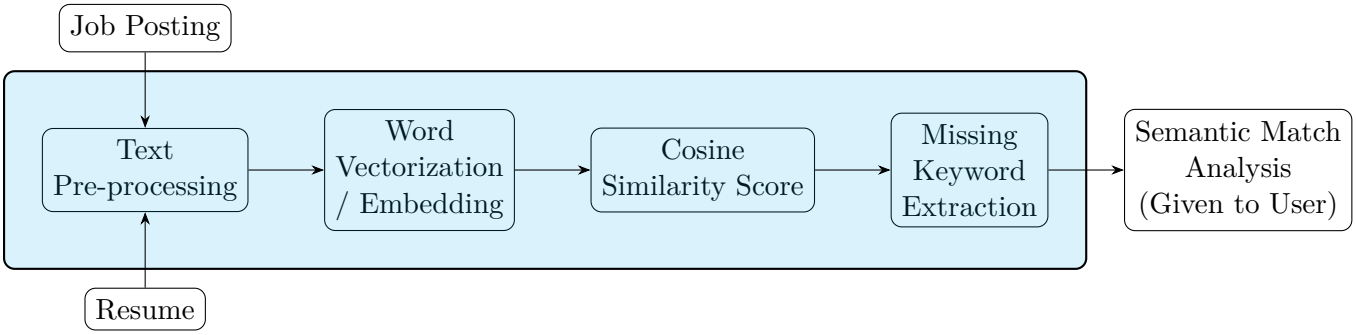

Figure 4: System diagram.