Resume-to-Job Matcher

Leveraging NLP and ML for Automated Candidate Selection

By –

Suvankar Das
suvankar_das@outlook.com

Introduction:

This is a report on a Resume-to-Job Matcher developed by me, **Suvankar Das**, as per the instructions mailed to us by **Capital Placement**, for the role of **AI/ML Engineering Intern**. The system deployed by me utilizes Natural Language Processing and Machine Learning techniques to automate the process of selecting the best candidate for the role.

This report provides a comprehensive overview of my approach to building the project. We will discuss the various steps involved, including data collection and preprocessing, embeddings, and similarity calculations, and ranking and selection of the top candidates. Additionally, we will dive into the challenges encountered during development and the solutions I implemented to overcome them.

In the end, we will look into the results/output and discuss the insights I got from this project.

Codes:

Original source code repository link: https://github.com/imSuvankar/Resume-to-Job-Matcher

Instructions:

Please make sure to follow these steps before processing:

- Step 1: Make sure to have all the resume PDFs in one single folder and upload that on Google Drive as a "ZIP" file before processing.
- Step 2: Download/use the "main.ipynb" file from the GitHub repo, and run it on Google Colab to avoid complications.
- Step 3: After mounting GDrive please copy the ZIP file path from Colab runtime storage and paste it into "resume_folder" variable.

Resume Data Extraction:

In today's recruitment landscape, using data-driven methods to understand resumes is incredibly important. This project carefully pulled information from a large collection of resumes saved in PDF format. The goal was to make the hiring process smarter and more efficient.

Approach:

My approach to resume data extraction was driven by a systematic and methodical framework. I used PyPDF2, a versatile library, to navigate through PDF files seamlessly. This library helped to extract textual content from each PDF resume. Subsequently, I focused on the extraction of critical data elements including:

- **Category:** Identifying the category or domain to which the candidate's profile belonged. This categorization serves as a vital initial step in candidate-job matching.
- **Skills:** Here I've used Part of Speech (PoS) tagging to efficiently identify and extract important skills possessed by candidates. This technique categorizes words based on their grammatical roles, aiding in accurate and scalable data extraction. PoS tagging helped overcome challenges like disorganized text and image-based content in PDFs. This approach contributed to creating comprehensive candidate profiles and enhancing the job-matching process.
- **Education:** Extracting the candidate's educational qualifications, or degrees, as educational background is a key determinant in evaluating a candidate's eligibility.

Challenges and Solutions:

- **Skill extraction challenge:** In the data extraction process I used two different methods. For educational degrees which are typically standard and easier to predict, I've used a predefined list which helped efficiently identify educational qualifications. However when it came to skills which can vary widely and are hard to predict in advance, I used a different approach, Noun keywords and Part-of-Speech (PoS) tagging. This method allowed the system to be more flexible and capture a wider range of candidate skills ensuring a comprehensive analysis.
- **Handling empty resumes and null data:** During the resume data extraction process I've seen some PDFs lacked structured formatting or contained images, making text extraction hard. To tackle this, I used exception-handling mechanisms to identify and bypass problematic PDFs. This approach ensured the robustness and reliability of the data extraction pipeline.

Job Description Data Extraction:

As directed in the problem statement, I've used 10 job descriptions from the dataset to start with. This is not a static order, we can always modify this range as per our need.

Approach:

While I could have used a similar method to find the required skills and education, the dataset had a valuable shortcut. It had a "model_response" column that contained organized insights about the "Required Skills" and "Educational Requirements" for each job description. This made the extraction process work faster with accuracy. This highlights how understanding the dataset is crucial for a developer for efficient data extraction leading to accurate results.

The approach to JD data extraction was designed to extract critical details essential for making accurate job-candidate pairings. Using the versatile "datasets" library I retrieved essential information from each job listing:

- **Company Name:** Extracted the name of the hiring company to offer a complete context.
- **Position Title:** Identifying the job's title which represents its core responsibilities.
- **Required Skills:** Using the "model_response" data I got the skills required which helps to create a comprehensive skill profile for each job.
- **Educational Requirements:** Like skills, the "model_response" data also helps to extract the educational requirements for each job role.

Challenges and Solutions:

- **Handling empty fields/requirements:** Data extraction sometimes faced challenges due to the inconsistent presence of educational qualifications and skill requirements in job descriptions. To ensure data consistency and prevent errors I marked the missing requirements as "N/A." This approach enhanced data analysis and overall dataset quality.
- **Dealing with JSON data:** Parsing data from the "model_response" field which was stored in JSON format created problems initially. To address this there is the eval() function to extract the data. Alternatively, Python's JSON module could also have been used for this purpose.

Matchmaking and Ranking Top Resumes:

With both the resume and job description data successfully extracted, we're now in the final phase of the project: matchmaking and cosine similarity analysis. This step aimed to rank top candidates for each listed job by assessing the compatibility of their skills, qualifications, and the job's requirements. Here I've used Natural Language Processing (NLP) techniques and cosine similarity as a metric to quantify the match.

Approach:

I've used embeddings to represent textual data (e.g., skills and education) in a numerical format. This conversion allowed us to measure the semantic similarity between resumes and job descriptions effectively. Here's how I accomplished it:

- **Embeddings Extraction:** To represent textual data in a format suitable for cosine similarity measurement, I've used the DistilBERT model. This model converted both the job descriptions and resumes into numerical embeddings.
- **Job position and candidate category matching:** I used TfidfVectorizer to get similarity in job positions and candidate category (job role) into this analysis.
- **Cosine Similarity Calculation:** The cosine similarity metric measured the similarity between these embeddings. Specifically, for both skills and educational requirements I calculated cosine similarity scores from scikit-learn library. And after adding category matching results this provided a comprehensive assessment of compatibility.

Challenges and Solutions:

- **Identifying problematic resume / job description:** Throughout the project, there are occasional challenges when dealing with unstructured or empty PDFs and job descriptions. To ensure the integrity of the analysis and maintain data quality, I used exception handling and identified those problematic PDFs and job descriptions.
- **Monitoring Progress with TQDM:** To enhance the transparency of data extraction and analysis processes, the system also had TQDM, a Python library for creating progress bars. This helped the project with visualizing progress, improved user experience, and debugging assistance.

Result:

This is the final output I got after the complete analysis. This result shows the top 5 (changeable as per the requirements) resume filenames according to the cosine score for each listed job. I've kept the cosine scores as it was, feel free to modify (add weightage, or convert into percentage) as per your need.

Also notice, for each cycle, it checks if there is any unusual PDF, if found it shows their filenames so that you can take the necessary actions (e.g., mark or delete those PDFs from database).

Processing time may vary based on the size of the dataset (resume) or number of jobs listed.

Conclusion and Insights:

The journey into data-driven recruitment has illuminated key takeaways for optimizing the hiring process:

- **Data-driven efficiency:** Implement data-driven methodologies to streamline recruitment.
- **Dataset understanding:** Familiarity with dataset structures enhances data extraction.
- **Flexible skill extraction:** Employ flexible techniques for skill extraction.
- **Managing missing data:** Marking missing data as "N/A" improves dataset quality.
- **Exception handling:** Robust exception handling ensures data extraction reliability.
- **Progress visualization:** Usage of modules like TQDM could help in understanding the progress.
- **Identifying unusual files:** Marking (or deleting) these files can reduce the possibility of error.

By adopting these insights, organizations can elevate their hiring process, reduce manual effort, and make informed decisions efficiently.