Individual Project Report

-Harshavardana Reddy Kolan

Introduction

The task of matching resumes with job descriptions is a critical step in recruitment processes. By automating this task using Natural Language Processing (NLP), organizations can streamline candidate screening, saving both time and resources. This report details the development and refinement of a resume-job description similarity system, covering its initial design, challenges faced, iterative improvements, and the final implementation. The system's evolution reflects a commitment to accuracy, efficiency, and adaptability in real-world applications.

Similarity between resumes and job description

Initial Approach

When I started, I implemented a basic pipeline to extract and preprocess text from resumes and job descriptions in PDF format. The process included the following steps:

1. **Text Extraction:** I utilized the PyPDF2 library to extract raw text from PDF documents. This gave me a foundational way to work with unstructured data.
2. **Preprocessing:** I created a cleaning function that converted the text to lowercase and removed all non-alphabetic characters. This helped ensure the input text was standardized before further processing.
3. **Embedding Generation:** To represent textual information numerically, I leveraged a pretrained BERT model from the Hugging Face Transformers library. BERT embeddings provided contextualized representations of the job descriptions and resumes.
4. **Similarity Calculation:** Using cosine similarity, I computed the similarity score between job description embeddings and resume embeddings.

Although this initial approach worked, I quickly identified several limitations:

- **Performance:** Computing BERT embeddings for long texts was computationally expensive and time-consuming.
- **Scalability:** The model was designed to handle entire documents, treating resumes and job descriptions as monolithic texts. This ignored the nuanced importance of different sections, such as skills and qualifications.
- **Feature Flexibility:** My initial approach did not allow for weighing different sections (e.g., skills versus responsibilities) based on their importance for a job.

Transition to a Refined Solution

Recognizing these shortcomings, I set out to improve the system. My goal was to build a solution that was faster, more modular, and capable of delivering finer-grained similarity assessments.

1. **Enhanced Parsing and Section-Based Analysis:**
    - I introduced a parser to break resumes and job descriptions into structured fields, such as skills, responsibilities, and qualifications. This allowed me to treat these sections independently, addressing the issue of unequal importance across different resume/job description parts.
2. **Leveraging Sentence Transformers:**
    - Instead of the bulky BERT model, I adopted the all-MiniLM-L6-v2 model from the SentenceTransformers library. This model provided lightweight, highly efficient embeddings while retaining sufficient semantic richness. The switch significantly improved computational performance.
3. **Weighted Similarity Scoring:**
    - I designed a weighted similarity scoring mechanism to assign different importance to each field. For example, I prioritized skills (50%), responsibilities (40%), and qualifications (10%). This approach allowed the final score to better reflect job-specific priorities.
4. **Reusable and Modular Design:**
    - I used the parsing logic from a separate module, resume_job_description_parser, which converts resume and job description files into Json format, to enhance code modularity and reusability. This decoupling made the solution easier to extend and maintain.

Through this iterative process, I evolved from a rudimentary system to a sophisticated, scalable, and flexible solution for resume and job description matching. The final implementation is better suited for real-world applications, offering improved accuracy and efficiency. By addressing the initial limitations, I demonstrated the importance of continuous experimentation and refinement in achieving robust NLP solutions.

Figure 1:

Note: The above figure shows the interface of the above process in streamlit to dynamically adjust the weights.

Resume Ranking System Development

Initial Approach

1)Enhanced Features and XGBoost Integration

In this iteration, I introduced several enhancements:

1. Chunking Long Texts: Texts exceeding BERT's token limit were split into smaller chunks, and their embeddings were aggregated to retain all the information.
2. Feature Engineering: I added label-based relevance scores and used them to train an XGBoost model for ranking resumes.
3. Evaluation Metrics: The inclusion of metrics like NDCG (Normalized Discounted Cumulative Gain) allowed for better evaluation of ranking performance.

Challenges and Limitations:

- Complexity: While XGBoost improved ranking accuracy, it introduced additional dependencies and increased the complexity of the pipeline.
- Overfitting Risk: The model's performance heavily depended on the quality and quantity of labelled data, which wasn't always sufficient.

2) Improved Text Splitting with LangChain

To further enhance text preprocessing, I integrated LangChain's RecursiveCharacterTextSplitter to handle long resumes more effectively:

1. Resumes were split into smaller chunks with overlap, maintaining context across chunks.
2. Aggregated embeddings of these chunks provided more comprehensive representations.

Challenges and Limitations:

- While chunking improved the accuracy of embeddings, it added additional processing time.
- The overall pipeline still felt heavy and complex, especially for real-time or interactive use cases.

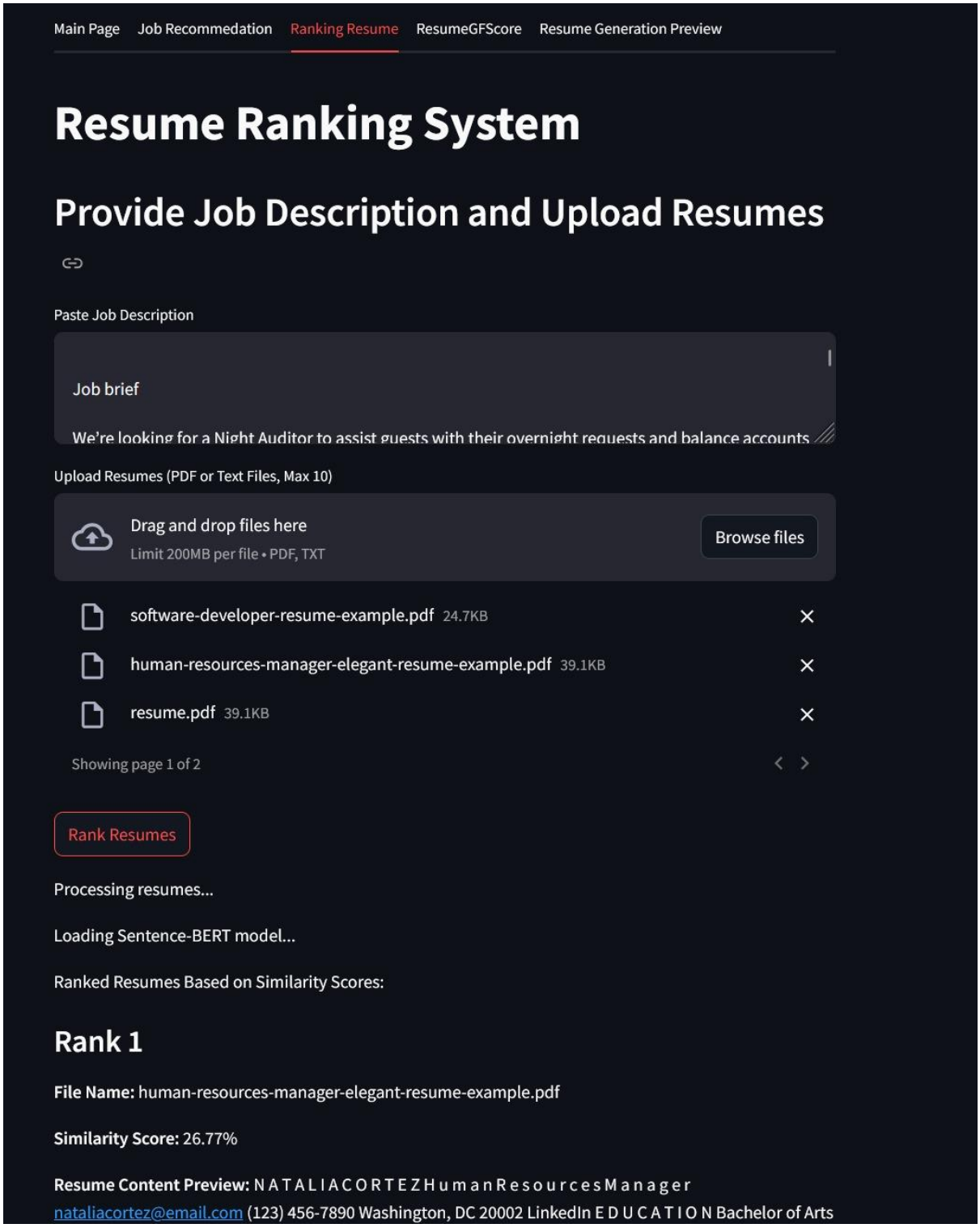
Final Implementation

Transition to Sentence-BERT

In the final version, I made significant changes to simplify and optimize the system:

1. Adopted Sentence-BERT:
    - Replaced BERT with Sentence-BERT (SBERT), which is specifically designed for generating sentence-level embeddings. SBERT significantly reduced computation time and memory usage while maintaining accuracy.
    - The embeddings could be directly compared using cosine similarity without the need for additional ranking models like XGBoost.
2. Refined Text Processing:
    - Simplified text extraction and cleaning functions for better usability.
    - Limited the maximum number of resumes to ensure system responsiveness.
3. Parallelization:
    - Leveraged concurrent processing using ThreadPoolExecutor to embed texts in parallel, significantly reducing runtime.

Figure 2:



Note: The above figure shows the interface of ranking resume with a given job description in streamlit.

Resume and Job Description Analysis using visualization:

1: Extracting Text from Documents

I began by extracting text from the provided resume and job description files.

- For the resume (PDF format), I used the PyPDF2 library to parse the document and extract text. This involved iterating over each page and concatenating the extracted text.
- For the job description (TXT format), I read the file directly using Python's built-in file-handling functionality.

This ensured that both documents were in a format suitable for further analysis.

2: Feature Extraction Using TF-IDF

To compare the two texts quantitatively, I implemented the **Term Frequency-Inverse Document Frequency (TF-IDF)** technique using TfidfVectorizer:

- I transformed the textual data from both documents into a numerical format that represents the importance of each word in the context of the respective document.
- I limited the features to the top 50 most significant terms for simplicity and interpretability.

This step provided a foundation for generating various visualizations.

Visualizations
1: Word Clouds

To visualize the most prominent terms in the resume and job description, I generated word clouds:

- Using the TF-IDF scores, I created word clouds for each document, where the size of each word corresponds to its importance.
- This visualization highlights key themes in both documents, making it easy to identify focus areas at a glance.

Significance: The word clouds provide a quick, intuitive way to compare the primary focus areas of the resume and job description. This helps identify alignment in terms of keywords.

Figure 3:

Note: The figure above shows the resume and job description word cloud to focus on important words.

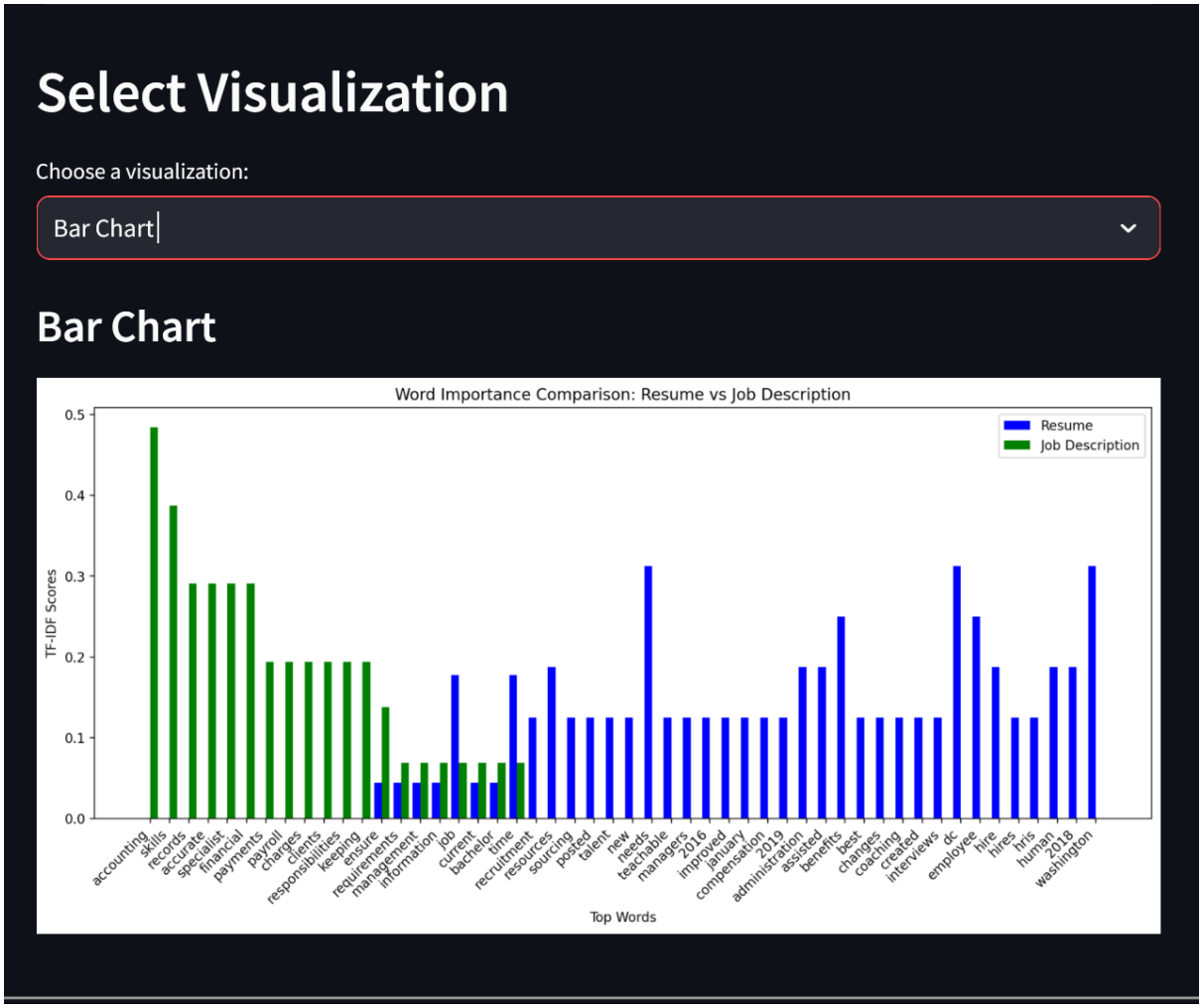
2: Bar Chart for TF-IDF Scores

To provide a detailed comparison of word importance, I generated a bar chart:

- Words were sorted based on their TF-IDF scores in the job description.
- The chart displayed the scores for each word in both the resume and job description.

Significance: The bar chart allows for a direct comparison of how well the candidate's resume reflects the specific needs outlined in the job description.

Figure 4:



Note: The above bar chart scores scores for each word in resume and job description.

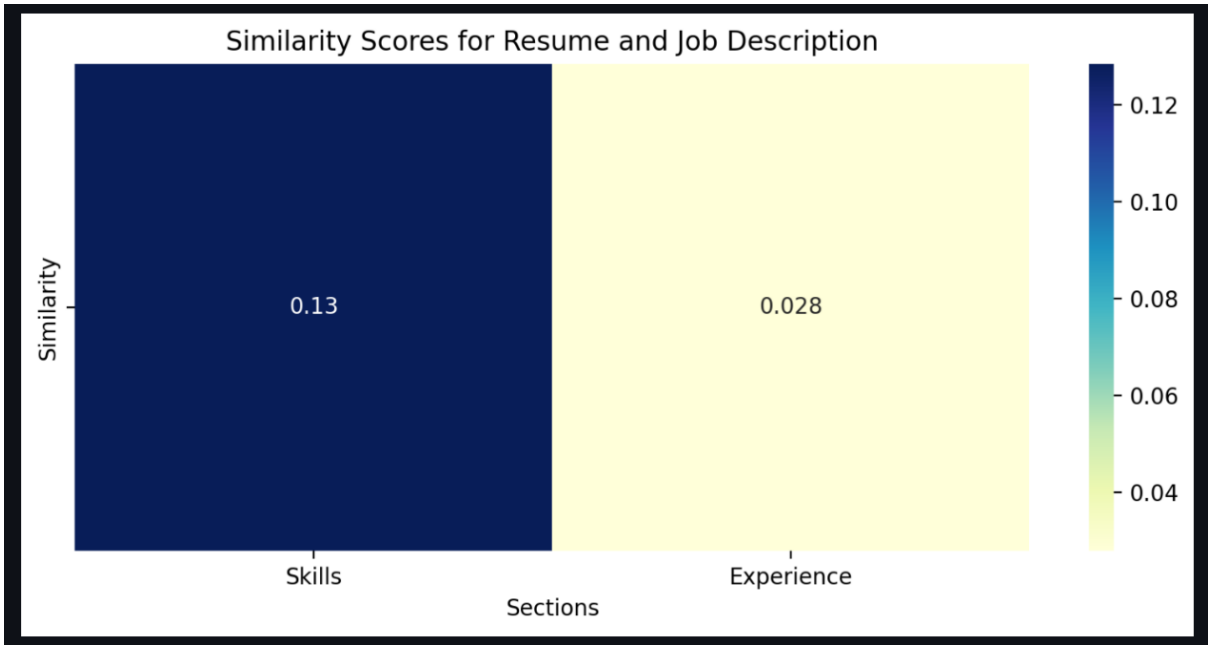
3: Heatmap for Similarity Scores

Using cosine similarity, I calculated similarity scores for key sections:

- Skills and experiences were extracted from the parsed documents and compared.
- The results were visualized as a heatmap, showing the similarity scores for these sections.

Significance: The heatmap quantitatively highlights the level of alignment between the resume and job description, enabling focused improvements in specific areas.

Figure 5:



Note: The above figure shows similarities scores for specific areas from the job description and resume.

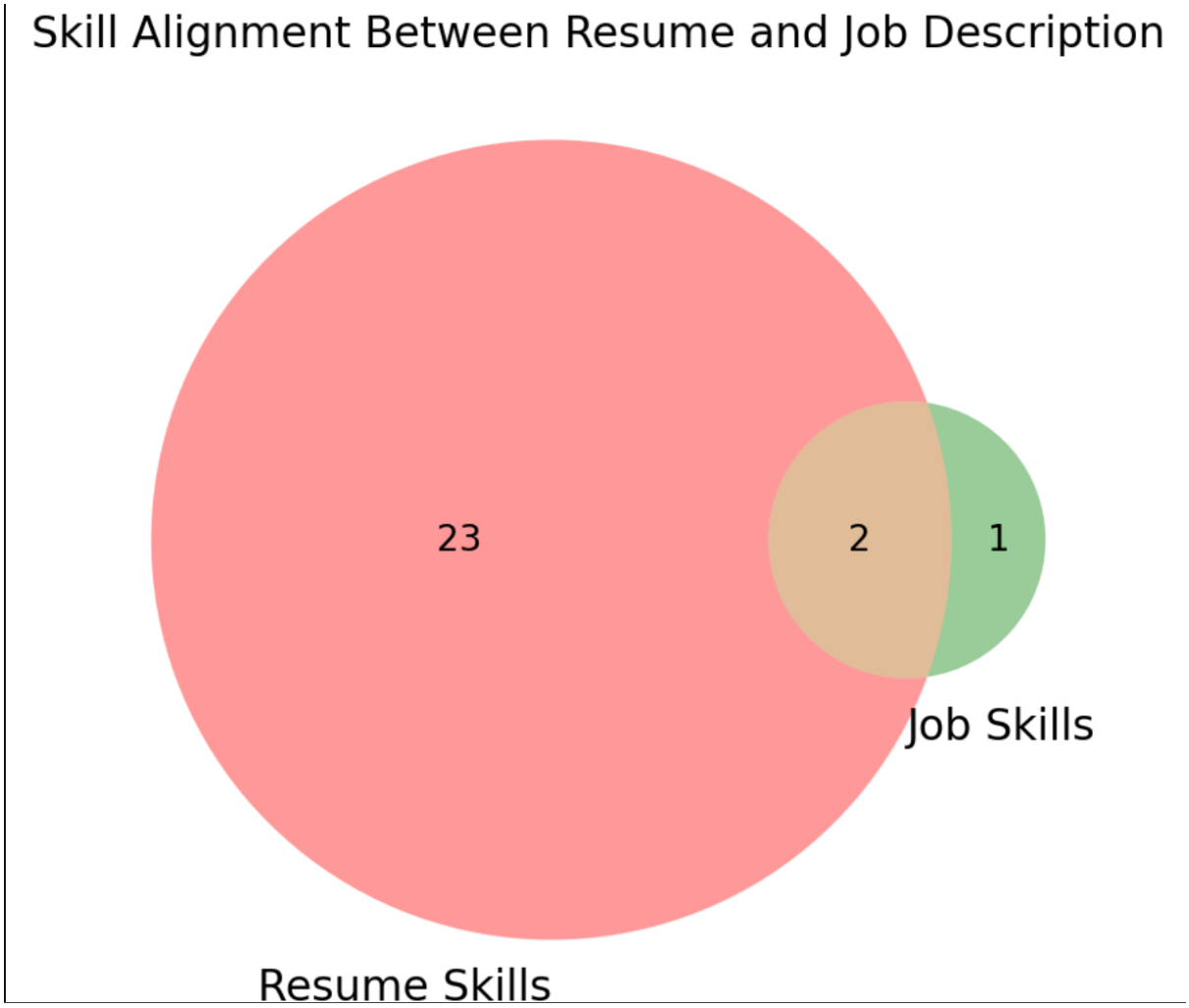
4: Venn Diagram for Skills

I generated a **Venn diagram** to visualize skill overlap:

- Skills from the resume and job description were extracted and matched using string similarity techniques.
- The diagram displayed unique and common skills between the two.

Significance: This visualization provides a clear representation of the skill alignment, pinpointing missing skills the candidate needs to acquire or highlight.

Figure 6:



Note: The above figure shows skills overlap between resume and job description in a venn diagram.

Conclusion

This project demonstrated a systematic approach to developing an effective resume-job description similarity system. Starting with a basic design, iterative refinements addressed computational inefficiencies, scalability, and usability challenges. The final implementation, leveraging Sentence-BERT, provides a scalable, modular, and accurate solution for real-world recruitment. Visualizations further enhance its usability, offering actionable insights for recruiters and candidates.

References

1. **PyPDF2 Library Documentation**: Text extraction capabilities from PDF documents.
2. **Hugging Face Transformers Library**: Pretrained models for text embedding.
3. **SentenceTransformers Documentation**: Lightweight sentence-level embeddings for semantic analysis.

4. **LangChain API**: Tools for advanced text chunking and processing.
5. **XGBoost Documentation**: Gradient boosting framework for classification and ranking tasks.
6. **TF-IDF in Python**: Techniques for term frequency analysis.