AI CV Filtering System – ATS

Overview

This project aims to create an AI-powered Applicant Tracking System (ATS) to automate the process of screening, ranking, and recommending CVs. It is divided into multiple stages, each focusing on distinct functionalities and leveraging AI for efficiency.

Main Features to be Implemented.

Stage 1: CV Submission

- **Functionality**:
  - Users should be able to upload CVs through a user-friendly interface.
  - Supported formats: PDF, DOC
  - CVs will be stored in a centralized database.
- **Implementation**:
  1. **Frontend**:
     - Use a file upload widget (e.g., Dropzone.js or Material-UI upload components for React).
     - Allow bulk upload for efficiency.
  2. **Backend**:
     - Use REST API endpoints (e.g.FastAPI) to handle file uploads.
     - Validate the file type and store extracted text in a database.
  3. **Database**:
     - Store structured information (e.g., name, contact, skills, job titles) in a database (MongoDB).

Stage 2: Dashboard with Analytical Insights

- **Functionality**:
  - An interactive dashboard showing:
    - **Number of CVs**.
    - **Job categories present**.
    - **Number of candidates for each job category**.
    - **Bar charts and pie charts** for visual representation.
- **Implementation**:
  1. **Frontend**:
     - Use React.js or Angular for the UI.
     - Integrate libraries like Chart.js or D3.js for dynamic graphs.
  2. **Backend**:
     - Create API endpoints to retrieve:
       - Total CV count.
       - Count of job categories.
       - Candidates per category.
  3. **Database**:
     - Ensure job categories and candidate counts are stored and updated automatically upon new CV submissions.

Stage 3: Candidate Ranking

- **Functionality**:
  - Users will input a prompt with job requirements (e.g., skills, experience).
  - The system ranks CVs from best to worst based on relevance to the prompt.
  - Users can view the ranked CVs, click to review, and select candidates for hiring.

- **Implementation**:
    1. **Text Matching**:
        - Use **TF-IDF** to compare CVs against the job description:
            - Convert text into feature vectors.
            - Compute cosine similarity between the prompt and CVs.
        - Alternatively, use **BERT** or **similar pre-trained transformers** to encode the job description and CVs into embeddings, then compute similarity.
    2. **Frontend**:
        - Create a form for users to input job requirements.
        - Display ranked CVs in a table or list format with clickable links to view full CV details.
    3. **Backend**:
        - Process the prompt and compare it with all CVs.
        - Return a ranked list of CVs based on similarity scores.

Stage 4: AI-Powered Recommendations

- **Functionality**:
    - Use AI to recommend CVs that match the job role based on learning patterns from previously selected candidates.
    - The system will suggest “hidden gems” from the database.
- **Implementation**:
    1. **Feature Extraction**:
        - Extract features from CVs (e.g., skills, education, experience) using **spaCy** or **Hugging Face Transformers**.
    2. **Training a Recommendation Model**:

        - Use a classification model (e.g., Logistic Regression, Decision Trees) or an ensemble model to predict the likelihood of selection based on historical data.
        - Alternatively, use clustering (e.g., K-means) to group CVs and suggest candidates from underexplored clusters.
3. **Evaluation**:
        - Implement cross-validation and performance metrics (e.g., accuracy, precision) to ensure the model's reliability.

Steps to Implement the Solution

1. Text Extraction from CVs

- **PDFs**:
    - Use **PyPDF2** or **PDFPlumber** for text extraction.
    - Handle different layouts and embedded images carefully.
- **Enhancements**:
    - Normalize and preprocess the extracted text:
        - Convert to lowercase.
        - Remove special characters, stopwords, and irrelevant sections (e.g., headers, footers).
        - Use regular expressions to extract structured data (e.g., contact info, education, skills).

2. Natural Language Processing (NLP) for Keyword Matching

- **Tools**:
    - **spaCy**: Tokenization, named entity recognition (NER), and part-of-speech tagging.
    - **TF-IDF**: Use TfidfVectorizer from **scikit-learn** for keyword weighting.

- **Pre-trained Transformers**:
  - Use models like **BERT** from Hugging Face for semantic similarity.
  - Fine-tune if needed to improve domain-specific understanding.
- **Enhancements**:
  - Implement **phrase matching** to identify multi-word skills (e.g., "machine learning").
  - Use **Word2Vec** or **GloVe embeddings** if transformers are too resource-intensive.

3. Ranking and Matching Candidates

- **Methods**:
  - Compute cosine similarity between job descriptions and CV vectors (TF-IDF or embeddings).
  - Rank CVs based on similarity scores.
- **Alternatives**:
  - Implement a scoring system that considers weighted criteria (e.g., skills: 40%, experience: 30%, education: 30%).

4. Candidate Recommendation

- **Approach**:
  - Train a supervised learning model using historical selection data:
    - Inputs: CV features (skills, experience, education).
    - Output: Binary label (selected/not selected).
  - Use the model to predict and recommend CVs from unselected candidates.
- **Enhancements**:
  - Use an active learning framework to iteratively improve recommendations based on user feedback.

Additional Suggestions

1. **UI/UX**:
    - Include filters (e.g., by experience, education) to narrow down candidates.
    - Implement a search bar for specific skills or keywords.
2. **Scalability**:
    - Use a cloud database (e.g., Firebase, AWS RDS) for larger datasets.
    - Employ distributed computing (e.g., Apache Spark) for faster processing if the database grows significantly.
3. **Security**:
    - Encrypt sensitive candidate data during storage and transmission.
    - Use access control to limit data visibility.
4. **Evaluation Metrics**:
    - Monitor system performance (e.g., time taken for ranking).
    - Collect user feedback on recommendations and refine the model periodically.