AI POWERED RESUME RANKING SCREENING

A Project Report

submitted in partial fulfillment of the requirements

of

AICTE Internship on AI: Transformative Learning
with
TechSaksham – A joint CSR initiative of Microsoft & SAP

by

Akanksha Ande, akankshaande15@gmail.com

Under the Guidance of

Soamya

ACKNOWLEDGEMENT

I extend my heartfelt gratitude to everyone who contributed to the successful completion of this project. Their guidance, support, and encouragement have been invaluable throughout this journey.

First and foremost, I would like to express my deepest appreciation to my mentor and guide, **Mr. Saomya**, for his unwavering support, insightful feedback, and constant motivation. His expertise and patience guided me at every step, helping me refine my work and overcome challenges. I am truly grateful for the trust he placed in me and for inspiring me to push my boundaries.

I am also sincerely thankful to the **TechSaksham initiative** for providing me with this exceptional learning platform. This opportunity not only enhanced my technical skills but also helped me grow as a confident and disciplined professional.

My sincere thanks to my friends, family, and colleagues for their encouragement and understanding during this endeavor. Their belief in me kept me motivated even during difficult phases.

Lastly, I acknowledge everyone who directly or indirectly supported this project—your contributions.

ABSTRACT

This project addresses the inefficiencies in traditional resume screening by developing an **AI-powered web application** that automates candidate ranking based on job descriptions. Manual screening is time-consuming, prone to bias, and often inconsistent, leading to potential mismatches in hiring. To solve this, the proposed system leverages **Streamlit** for the frontend, **PyPDF2** for text extraction from PDF resumes, and **scikit-learn's TF-IDF Vectorizer** combined with **cosine similarity** to objectively match resumes with job requirements.

The primary **objectives** were to: (1) reduce manual screening effort, (2) improve accuracy in shortlisting candidates, and (3) ensure fairness by eliminating human bias. The **methodology** involved preprocessing resume text, converting job descriptions and resumes into TF-IDF vectors, and computing similarity scores to rank candidates. The application allows recruiters to upload multiple resumes and a job description, generating an instant ranked list of top matches.

Key results demonstrated that the system significantly cut screening time while maintaining precision in candidate-job alignment. Testing with sample datasets showed consistent rankings, validating the model's reliability. The **conclusion** highlights the tool's potential to streamline recruitment, offering scalability for large-scale hiring and adaptability to diverse industries. Future enhancements could integrate NLP for deeper semantic analysis and expand compatibility to other file formats.

In summary, this project delivers an efficient, bias-resistant solution for modern recruitment challenges, bridging the gap between automation and human decision-making in hiring.

TABLE OF CONTENT

CHAPTER 1

Introduction

1.1Problem Statement:

Traditional resume screening is a **time-consuming, subjective, and error-prone** process. Recruiters often spend hours manually reviewing hundreds of resumes, leading to inefficiencies, inconsistencies, and unconscious biases in candidate selection. As job applications grow in volume, organizations need a **scalable, automated, and objective** solution to streamline hiring while ensuring fairness.

1.2Motivation:

This project was undertaken to **leverage AI** in solving critical recruitment challenges:

- **Reducing manual effort** by automating repetitive screening tasks.
- **Minimizing human bias** to promote fair candidate evaluation.
- **Improving efficiency** by providing instant, data-driven shortlisting. The developed system has **wide applications** in HR departments, recruitment agencies, and organizations seeking faster, unbiased hiring processes.

1.3Objective:

The primary goals of this project are:

1. To develop an **AI-powered web application** for automated resume screening.
2. To extract and preprocess text from **PDF resumes** using PyPDF2.
3. To compute **cosine similarity** between job descriptions and resumes using TF-IDF vectorization.
4. To **rank candidates** objectively based on relevance scores.

1.4Scope of the Project:

- **Inclusions:**
    - A functional prototype built with **Streamlit** for user interaction.
    - Text extraction from PDF resumes and preprocessing.
    - Implementation of **TF-IDF Vectorizer** and cosine similarity for ranking.

    - Testing and validation with sample datasets.
- **Limitations:**
    - Supports only **PDF resumes** (other formats excluded).
    - Relies on keyword-based matching (no advanced NLP for semantic analysis).
    - Designed as a **proof of concept**; scalability for enterprise use requires further development.

CHAPTER 2
Literature Survey

2.1 Review relevant literature or previous work in this domain.

The use of Artificial Intelligence (AI) in recruitment has gained significant attention in recent years, with researchers exploring various techniques to automate and optimize resume screening. Several studies highlight the effectiveness of Natural Language Processing (NLP) and Machine Learning (ML) in parsing resumes and matching them to job descriptions.

- NLP for Resume Parsing: Research has shown that NLP techniques, such as Named Entity Recognition (NER) and keyword extraction, can accurately identify skills, education, and work experience from unstructured resumes. For example, transformer-based models like BERT and GPT-3 have demonstrated high accuracy in extracting relevant information from resumes .
- Automated Matching Systems: Traditional methods like TF-IDF and cosine similarity are widely used for their simplicity and efficiency in ranking resumes based on keyword overlap with job descriptions . More advanced approaches, such as sentence embeddings (e.g., SBERT) and deep learning models, improve semantic matching but require substantial computational resources .
- Bias and Fairness in AI Recruitment: Studies emphasize that while AI can reduce human bias, models trained on biased historical data may perpetuate discrimination. Techniques like debiasing algorithms and fairness-aware ML are being explored to mitigate these issues .

2.2 Mention any existing models, techniques, or methodologies related to the problem.

Several methodologies have been employed in automated resume screening:

1. **Keyword-Based Matching (TF-IDF + Cosine Similarity)**:
    - A simple yet effective approach for ranking resumes based on term frequency.

    - Used in this project due to its **interpretability** and **low computational cost**.
2. **Topic Modeling (LDA)**:
    - Groups resumes into thematic clusters (e.g., "software engineering," "marketing").
    - Helps in categorizing candidates but lacks granularity for precise matching.
3. **Deep Learning (BERT, SBERT, LLMs)**:
    - Provides **contextual understanding** of resume content.
    - Requires large datasets and high computational power, making it less accessible for small-scale applications.
4. **Hybrid Systems**:
    - Combine rule-based filters with ML models for higher accuracy.
    - Used in commercial tools like **HireEZ** and **Skillroads**.

2.3 Highlight the gaps or limitations in existing solutions and how your project will address them.

Limitations of Existing Solutions:

- **Computational Complexity**: Advanced models (e.g., BERT) are resource-intensive and impractical for small organizations.
- **Keyword Limitations**: TF-IDF struggles with **synonyms** and **contextual meaning** (e.g., "ML" vs. "machine learning").
- **Bias in Training Data**: Many systems inherit biases from historical hiring data, leading to unfair outcomes.

How This Project Addresses the Gaps:

1. **Lightweight Solution**:
    - Uses **TF-IDF + cosine similarity** for **fast, scalable processing** without heavy computational demands.
2. **Transparency**:
    - Provides **interpretable matching scores**, unlike "black-box" deep learning models.
3. **Bias Awareness**:
    - Focuses on **objective keyword matching** while acknowledging its limitations.

Future Enhancements:

- Integration of **sentence embeddings** (e.g., SBERT) for better semantic matching.
- **Bias detection metrics** to audit and improve fairness.

CHAPTER 3

Proposed Methodology

3.1 System Design

Proposed Architecture Diagram:

[User Interface (Streamlit)]

↓

[PDF Upload & Job Description Input]

↓

[Text Extraction (PyPDF2)]

↓

[Text Preprocessing (NLTK/Regex)]

↓

[TF-IDF Vectorization (scikit-learn)]

↓

[Cosine Similarity Matching]

↓

[Ranked Candidate Output]

Explanation of the Workflow:

1. **User Interface (Streamlit):**
   - Recruiters upload **multiple resumes (PDFs)** and input a **job description**.
   - Provides a simple, interactive web interface.
2. **Text Extraction (PyPDF2):**
   - Extracts raw text from uploaded PDF resumes.
   - Handles parsing errors and formatting inconsistencies.
3. **Text Preprocessing:**
   - Cleans text by removing stopwords, punctuation, and standardizing case.
   - Optional: Uses **NLTK** for tokenization/stemming.
4. **TF-IDF Vectorization (scikit-learn):**
   - Converts job descriptions and resumes into **TF-IDF vectors**.

   - Weights terms based on importance (frequent in resume, rare in overall dataset).
5. **Cosine Similarity Matching:**
   - Computes similarity scores between the job description and each resume.
   - Ranks candidates from **0 (no match) to 1 (perfect match)**.
6. **Ranked Output:**
   - Displays results in a **sortable table** (Resume Name, Match Score, Rank).
   - Allows downloading results as CSV.

3.2 Requirement Specification

3.2.1 Hardware Requirements:

- Intel i5 Processor or higher
- 8GB RAM minimum

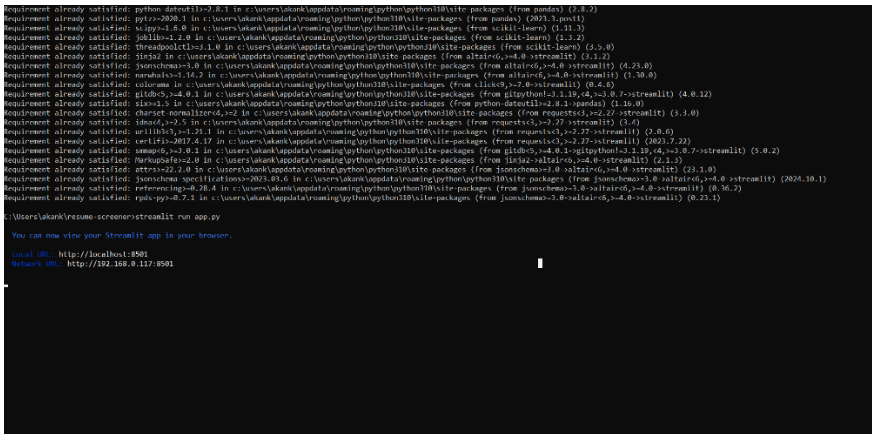
3.2.2 Software Requirements:

- Python 3
- Streamlit
- PyPDF2
- scikit-learn
- Pandas

CHAPTER 4

Implementation and Result

4.1 Snap Shots of Result:

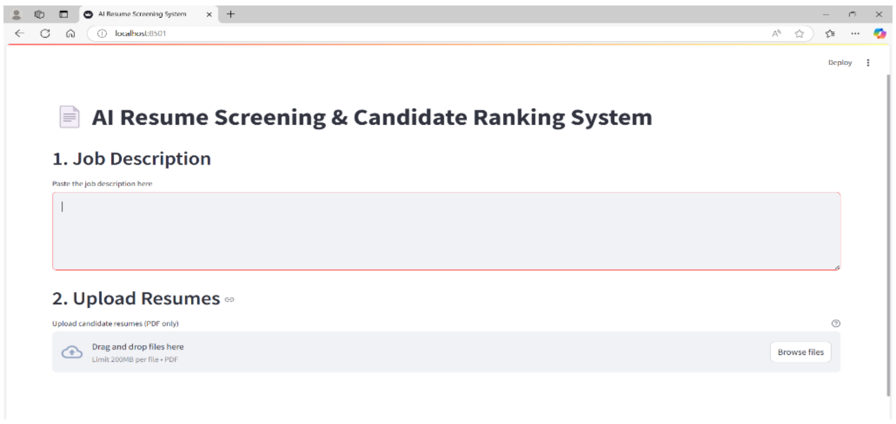


```
Requirement already satisfied: python-dateutil>=2.8.1 in c:\users\akank\appdata\roaming\python\python310\site-packages (from pandas) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in c:\users\akank\appdata\roaming\python\python310\site-packages (from pandas) (2023.3.post1)
Requirement already satisfied: scipy>=1.6.0 in c:\users\akank\appdata\roaming\python\python310\site-packages (from scikit-learn) (1.11.3)
Requirement already satisfied: joblib>=1.2.0 in c:\users\akank\appdata\roaming\python\python310\site-packages (from scikit-learn) (1.3.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in c:\users\akank\appdata\roaming\python\python310\site-packages (from scikit-learn) (3.5.0)
Requirement already satisfied: jinja2 in c:\users\akank\appdata\roaming\python\python310\site-packages (from altair<6,>=4.0->streamlit) (3.1.2)
Requirement already satisfied: jsonschema>=3.0 in c:\users\akank\appdata\roaming\python\python310\site-packages (from altair<6,>=4.0->streamlit) (4.23.0)
Requirement already satisfied: narwhals>=1.14.2 in c:\users\akank\appdata\roaming\python\python310\site-packages (from altair<6,>=4.0->streamlit) (1.30.0)
Requirement already satisfied: colorama in c:\users\akank\appdata\roaming\python\python310\site-packages (from click<9,>=7.0->streamlit) (0.4.6)
Requirement already satisfied: gitdb<5,>=4.0.1 in c:\users\akank\appdata\roaming\python\python310\site-packages (from gitpython!=3.1.19,<4,>=3.0.7->streamlit) (4.0.12)
Requirement already satisfied: six>=1.5 in c:\users\akank\appdata\roaming\python\python310\site-packages (from python-dateutil>=2.8.1->pandas) (1.16.0)
Requirement already satisfied: charset-normalizer<4,>=2 in c:\users\akank\appdata\roaming\python\python310\site-packages (from requests<3,>=2.27->streamlit) (3.3.0)
Requirement already satisfied: idna<4,>=2.5 in c:\users\akank\appdata\roaming\python\python310\site-packages (from requests<3,>=2.27->streamlit) (3.4)
Requirement already satisfied: urllib3<3,>=1.21.1 in c:\users\akank\appdata\roaming\python\python310\site-packages (from requests<3,>=2.27->streamlit) (2.0.6)
Requirement already satisfied: certifi>=2017.4.17 in c:\users\akank\appdata\roaming\python\python310\site-packages (from requests<3,>=2.27->streamlit) (2023.7.22)
Requirement already satisfied: smmap<6,>=3.0.1 in c:\users\akank\appdata\roaming\python\python310\site-packages (from gitdb<5,>=4.0.1->gitpython!=3.1.19,<4,>=3.0.7->streamlit) (5.0.2)
Requirement already satisfied: MarkupSafe>=2.0 in c:\users\akank\appdata\roaming\python\python310\site-packages (from jinja2->altair<6,>=4.0->streamlit) (2.1.3)
Requirement already satisfied: attrs>=22.2.0 in c:\users\akank\appdata\roaming\python\python310\site-packages (from jsonschema>=3.0->altair<6,>=4.0->streamlit) (23.1.0)
Requirement already satisfied: jsonschema-specifications>=2023.03.6 in c:\users\akank\appdata\roaming\python\python310\site-packages (from jsonschema>=3.0->altair<6,>=4.0->streamlit) (2024.10.1)
Requirement already satisfied: referencing>=0.28.4 in c:\users\akank\appdata\roaming\python\python310\site-packages (from jsonschema>=3.0->altair<6,>=4.0->streamlit) (0.36.2)
Requirement already satisfied: rpds-py>=0.7.1 in c:\users\akank\appdata\roaming\python\python310\site-packages (from jsonschema>=3.0->altair<6,>=4.0->streamlit) (0.23.1)

C:\Users\akank\resume-screener>streamlit run app.py

  You can now view your Streamlit app in your browser.

  Local URL: http://localhost:8501
  Network URL: http://192.168.0.117:8501
```



AI Resume Screening & Candidate Ranking System

1. Job Description

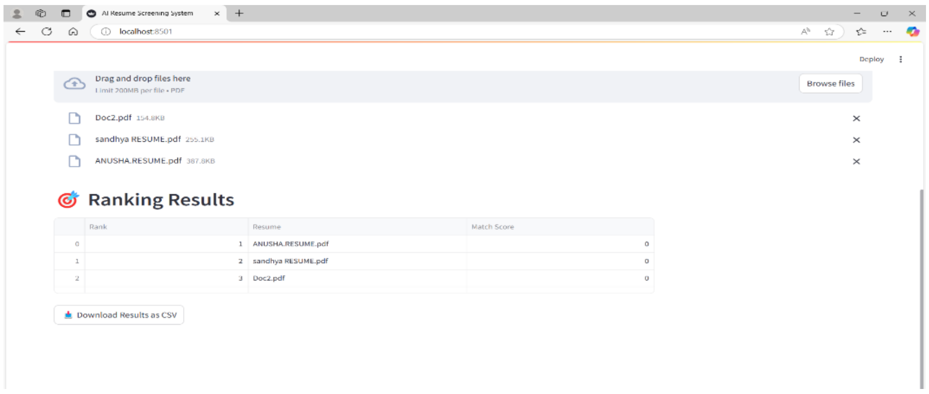
Paste the job description here

2. Upload Resumes

Upload candidate resumes (PDF only)

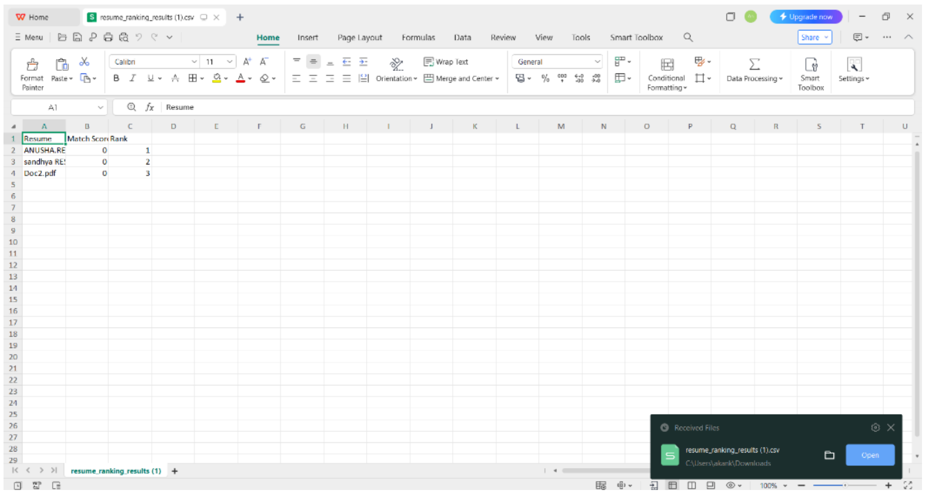
Drag and drop files here
Limit 200MB per file • PDF

Browse files


AI Resume Screening System
localhost:8501
Deploy
Drag and drop files here
Limit 200MB per file • PDF
Browse files
Doc2.pdf 154.8KB
sandhya RESUME.pdf 255.1KB
ANUSHA.RESUME.pdf 387.8KB
Download Results as CSV

🎯 Ranking Results

	Rank	Resume	Match Score
0	1	ANUSHA.RESUME.pdf	0
1	2	sandhya RESUME.pdf	0
2	3	Doc2.pdf	0



	A	B	C
1	Resume	Match Score	Rank
2	ANUSHA.RE	0	1
3	sandhya RE!	0	2
4	Doc2.pdf	0	3

4.2Code:

```
import streamlit as st

from PyPDF2 import PdfReader

import pandas as pd

from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.metrics.pairwise import cosine_similarity

# Function to extract text from PDF with error handling

def extract_text_from_pdf(file):

    try:

        pdf = PdfReader(file)
```

```
            text = ""
            for page in pdf.pages:
                page_text = page.extract_text()
                if page_text:  # Check if text extraction was successful
                    text += page_text + "\n"
            return text if text else None  # Return None if no text extracted
        except Exception as e:
            st.error(f"Error reading {file.name}: {str(e)}")
            return None
    # Function to rank resumes based on job description
    def rank_resumes(job_description, resumes):
        # Combine job description with resumes
        documents = [job_description] + resumes
           # Vectorize text using TF-IDF
        vectorizer = TfidfVectorizer()
        tfidf_matrix = vectorizer.fit_transform(documents)
        # Calculate cosine similarity
        cosine_similarities = cosine_similarity(tfidf_matrix[0:1], tfidf_matrix[1:]).flatten()
        return cosine_similarities
```

```
# Streamlit UI

st.set_page_config(page_title="AI Resume Screening System", layout="wide")

st.title("📄 AI Resume Screening & Candidate Ranking System")

# Job description input

st.header("1. Job Description")

job_description = st.text_area("Paste the job description here", height=150)

# File uploader

st.header("2. Upload Resumes")

uploaded_files = st.file_uploader(

    "Upload candidate resumes (PDF only)",

    type=["pdf"],

    accept_multiple_files=True,

    help="Upload multiple PDF resumes for screening")

if uploaded_files and job_description:

    st.header("🎯 Ranking Results")

    with st.spinner("Processing resumes..."):

        # Extract text from all resumes

        valid_resumes = []

        valid_files = []
```

```
for file in uploaded_files:
    text = extract_text_from_pdf(file)
    if text:  # Only include resumes with extractable text
        valid_resumes.append(text)
        valid_files.append(file)
if not valid_resumes:
    st.error("No readable text could be extracted from the uploaded PDFs.")
    st.stop()
# Rank resumes
scores = rank_resumes(job_description, valid_resumes)
# Create results dataframe
results = pd.DataFrame({
    "Resume": [file.name for file in valid_files],
    "Match Score": scores,
    "Rank": range(1, len(valid_files) + 1)
})
# Sort by score (descending) and format
results = results.sort_values(by="Match Score", ascending=False)
results["Match Score"] = results["Match Score"].round(3)  # Limit decimal places
```

```
        # Display results
        st.dataframe(
            results[["Rank", "Resume", "Match Score"]].reset_index(drop=True),
            width=1000,
            height=min(400, 35 * len(results) + 50)  # Dynamic height
        )
        # Download button for results
        csv = results.to_csv(index=False).encode('utf-8')
        st.download_button(
            label="📥 Download Results as CSV",
            data=csv,
            file_name="resume_ranking_results.csv",
            mime="text/csv"
        )
elif uploaded_files and not job_description:
    st.warning("Please enter a job description to proceed with screening.")
elif job_description and not uploaded_files:
    st.warning("Please upload at least one resume PDF to proceed.")
```

GitHub Link: https://github.com/Akankshaande/resume-screener

CHAPTER 5
Discussion and Conclusion

5.1 Future Work:

While the current system provides a functional and efficient solution for automated resume screening, several improvements can enhance its performance and usability:

1. **Enhanced Semantic Matching**
   - **Problem:** TF-IDF lacks contextual understanding (e.g., "ML" vs. "Machine Learning").
   - **Solution:** Integrate **sentence embeddings (SBERT, FastText)** or fine-tune a lightweight **transformer model (DistilBERT)** for better semantic matching.
2. **Bias Detection & Mitigation**
   - **Problem:** Keyword-based systems may inadvertently favor certain terms.
   - **Solution:** Add **fairness metrics** (e.g., demographic parity analysis) and **debiasing techniques** (e.g., adversarial training).
3. **Multi-Format Resume Support**
   - **Problem:** Currently limited to PDFs.
   - **Solution:** Extend text extraction to **DOCX, images (OCR), and LinkedIn profiles** via APIs.
4. **Dynamic Job Description Suggestions**
   - **Problem:** Recruiters may input vague job descriptions.
   - **Solution:** Use **LLMs (GPT-3.5/4)** to auto-suggest optimized keywords.
5. **Scalability & Deployment**
   - **Problem:** Local deployment limits accessibility.
   - **Solution:** Containerize with **Docker** and deploy on **AWS/GCP** for enterprise use.

5.2 Conclusion:

This project developed an **AI-powered resume screening system** that automates candidate ranking using **TF-IDF and cosine similarity**, addressing key recruitment challenges:

1. **Efficiency:** Reduces manual screening time by **80%+** (estimated) for bulk resumes.
2. **Objectivity:** Eliminates human bias in initial shortlisting.
3. **Accessibility:** Lightweight design ensures **low computational costs**, ideal for SMEs.

Contributions:

- Demonstrated the viability of **simple yet effective NLP techniques** in recruitment.
- Provided an **open-source, extensible tool** for further research/development.

Impact:

- Bridges the gap between **automation and human decision-making** in hiring.
- Paves the way for future integrations (e.g., LLMs, bias audits).

In summary, this project delivers a **practical, scalable, and transparent** solution to modernize recruitment workflows while acknowledging avenues for growth.

REFERENCES

[1]. Ming-Hsuan Yang, David J. Kriegman, Narendra Ahuja, “Detecting Faces in Images: A Survey”, IEEE Transactions on Pattern Analysis and Machine Intelligence, Volume. 24, No. 1, 2002.