AI RESUME ANALYZER USING NLP & TF-IDF

Automated Resume–Job Description Matching System

Developed By

Chinmaya Sajeevan

Project Domain

Natural Language Processing | Machine Learning | Recruitment Automation

Technologies Used

Python | NLTK | Scikit-learn | pdfplumber | Streamlit

Core Techniques

TF-IDF Vectorization | Cosine Similarity | Text Preprocessing

Year

2026

Project Overview

The AI Resume Analyzer is an NLP-based application that evaluates the similarity between a candidate's resume and a job description.

The system extracts text from PDF resumes, applies Natural Language Processing (NLP) techniques, converts text into numerical vectors using TF-IDF, and calculates similarity using cosine similarity.

The application is deployed using Streamlit for real-time resume evaluation.

Problem Statement

Recruiters often manually screen hundreds of resumes against job descriptions, which is:

- Time-consuming
- Subjective
- Inconsistent
- Prone to human bias

This project aims to build an automated resume screening system that:

- Quantifies resume-job match percentage
- Assists recruiters in preliminary screening
- Provides objective similarity scoring
- Improves hiring efficiency

System Architecture

Resume Text Extraction

PDF resumes are processed using:

- pdfplumber

The system extracts raw text page-by-page and combines it into a single string.

Text Preprocessing (NLP Pipeline)

Implemented using:

- NLTK

Steps performed:

- Unicode normalization
- Lowercasing
- Tokenization
- Stopword removal
- Lemmatization

Purpose:

- Clean the text
- Remove noise
- Convert words to base form
- Improve feature consistency

Feature Engineering – TF-IDF

Implemented using:

- scikit-learn

TF-IDF (Term Frequency – Inverse Document Frequency):

- Converts text into numerical vectors
- Assigns importance weights to words
- Uses unigrams and bigrams
- Limits features to 5000 for efficiency

Similarity Calculation

Cosine Similarity is used to measure similarity between:

- Resume vector
- Job Description vector

Score formula:

- Output range: 0–100%

- Higher score → Higher relevance

Technologies Used

- Python
- pdfplumber
- NLTK
- Scikit-learn
- NumPy
- Streamlit

Deployment

The application is deployed using:

- Streamlit

Features:

✓ Upload Resume (PDF)
✓ Paste Job Description
✓ Real-time match score
✓ Strong / Moderate / Low match interpretation

Run locally:

streamlit run app.py

Output Example

Match Score: 62.45%
⚡ Moderate Match

The system provides an interpretable similarity score that helps users understand resume-job alignment.

Skills Demonstrated

- Natural Language Processing

- Text Preprocessing
- Feature Engineering
- Vectorization Techniques
- Similarity Metrics
- Python Development
- Model Deployment

THE CODE SNIPPETS

utils.py

```python
# IMPORTS

import pdfplumber
import unicodedata
import nltk
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from nltk.tokenize import word_tokenize
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity

# NLTK SETUP (Run once if needed)
# Uncomment these only the first time you run
# nltk.download('punkt')
# nltk.download('stopwords')
# nltk.download('wordnet')

lemmatizer = WordNetLemmatizer()
stop_words = set(stopwords.words('english'))

# 1️⃣ Resume Text Extraction

def extract_text_from_pdf(pdf_file):
    text = ""
    with pdfplumber.open(pdf_file) as pdf:
        for page in pdf.pages:
            page_text = page.extract_text()
            if page_text:
                text += page_text + " "
    return text

# 2️⃣ Text Preprocessing
```

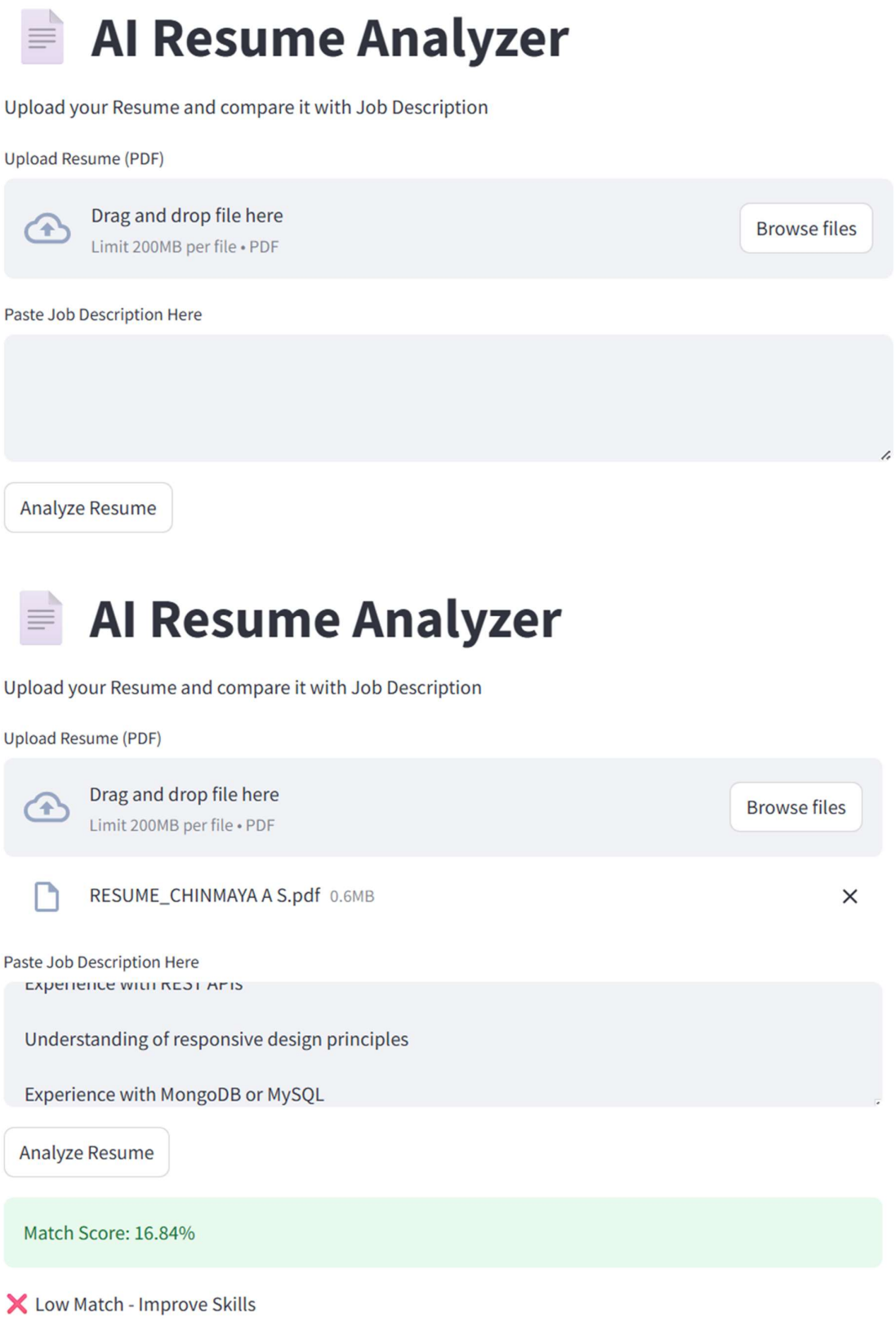
```python
def preprocess_text(text):
    # Normalize unicode
    text = unicodedata.normalize('NFKD', text)
    text = text.encode('ascii', 'ignore').decode('utf-8')
    text = text.lower()

    # Tokenization
    tokens = word_tokenize(text)

    cleaned_tokens = []
    for word in tokens:
        if word.isalpha() and word not in stop_words:
            word = lemmatizer.lemmatize(word)
            cleaned_tokens.append(word)

    return " ".join(cleaned_tokens)

# 3️⃣ TF-IDF + Cosine Similarity

def calculate_similarity(resume_text, jd_text):

    vectorizer = TfidfVectorizer(
        ngram_range=(1, 2),
        max_features=5000,
        sublinear_tf=True
    )

    # Fit on resume
    resume_vector = vectorizer.fit_transform([resume_text])
    jd_vector = vectorizer.transform([jd_text])

    similarity = cosine_similarity(resume_vector, jd_vector)

    score = similarity[0][0] * 100

    return round(score, 2)
```

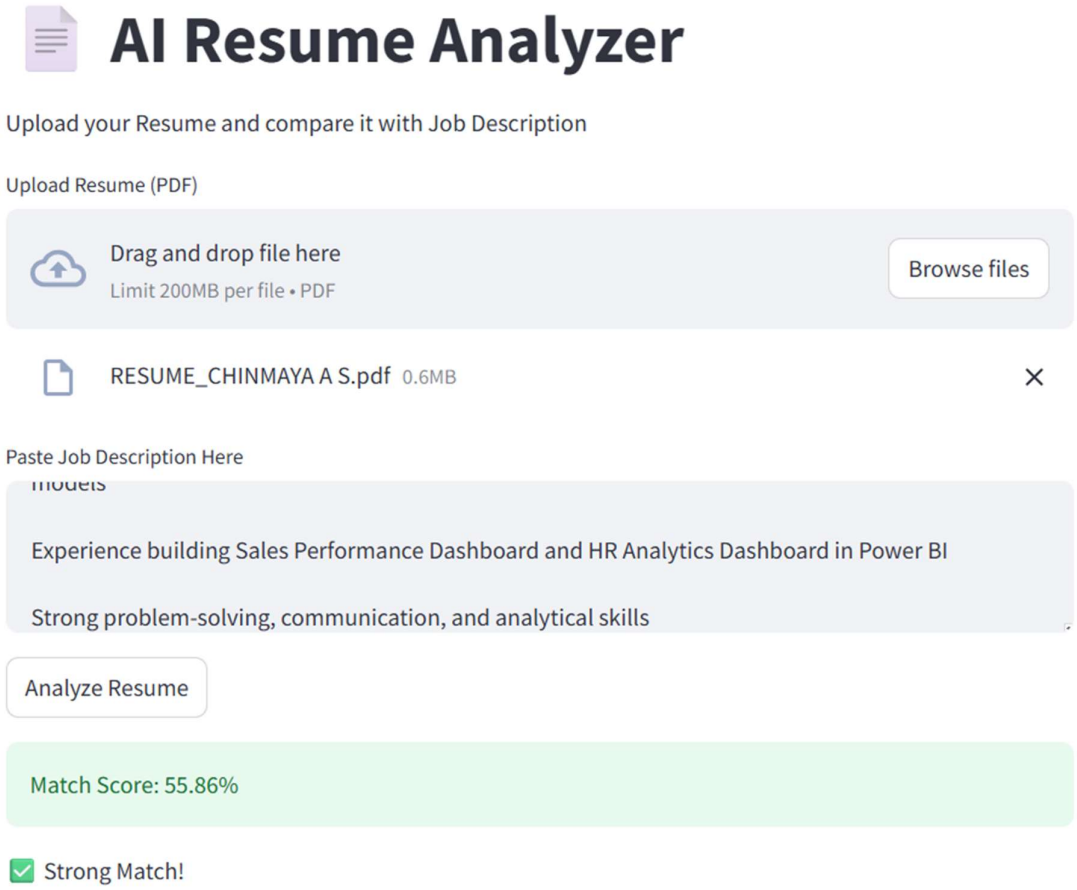
app.py

```python
# IMPORTS

import streamlit as st
from utils import extract_text_from_pdf, preprocess_text, calculate_similarity

# APP TITLE

st.title("📄 AI Resume Analyzer")
st.write("Upload your Resume and compare it with Job Description")
```

```python
# FILE UPLOAD

uploaded_file = st.file_uploader("Upload Resume (PDF)", type=["pdf"])

# JOB DESCRIPTION INPUT

jd_text = st.text_area("Paste Job Description Here")

# ANALYZE BUTTON

if st.button("Analyze Resume"):

    if uploaded_file is not None and jd_text != "":

        # Extract resume text
        resume_text = extract_text_from_pdf(uploaded_file)

        # Preprocess both texts
        resume_text = preprocess_text(resume_text)
        jd_text = preprocess_text(jd_text)

        # Calculate similarity
        score = calculate_similarity(resume_text, jd_text)

        st.success(f"Match Score: {score}%")

        # Score Interpretation (Realistic Scale)
        if score > 45:
            st.write("✅ Strong Match!")
        elif score > 25:
            st.write("⚡ Moderate Match")
        else:
            st.write("❌ Low Match - Improve Skills")

    else:
        st.warning("Please upload resume and enter job description.")
```

THE OUPUT


AI Resume Analyzer
Upload your Resume and compare it with Job Description
Upload Resume (PDF)
Drag and drop file here
Limit 200MB per file • PDF
Browse files
Paste Job Description Here
Analyze Resume
AI Resume Analyzer
Upload your Resume and compare it with Job Description
Upload Resume (PDF)
Drag and drop file here
Limit 200MB per file • PDF
Browse files
RESUME_CHINMAYA A S.pdf 0.6MB
Paste Job Description Here
Understanding of responsive design principles
Experience with MongoDB or MySQL
Analyze Resume
Match Score: 16.84%
Low Match - Improve Skills



Conclusion

The AI Resume Analyzer successfully demonstrates the practical application of Natural Language Processing (NLP) techniques in automated recruitment screening. By integrating PDF text extraction, structured text preprocessing, TF-IDF vectorization, and cosine similarity, the system effectively quantifies the relevance between a resume and a job description.

The solution provides an objective and scalable approach to resume evaluation, reducing manual effort and improving screening efficiency. The deployment using Streamlit enables real-time interaction, making the system user-friendly and practically usable.

This project highlights strong understanding of NLP pipelines, feature engineering, and similarity metrics while showcasing the ability to build and deploy an end-to-end intelligent application. Future improvements may include contextual embeddings such as BERT, skill gap analysis, and multi-resume ranking for enterprise-level recruitment systems.