A logo for the northcar university

Description automatically generated

**Computational Linguistics And**

**Natural Language Processing**

Project Report On

Job Description and Resume Matching System

Department of Computer Science and Engineering

The NorthCap University, Gurugram

Name: Tanishka

Section: AIML-C (AI-6)

Roll No.: 21CSU491

**ACKNOWLEDGEMENT**

I extend my deepest gratitude to those whose support and guidance have been invaluable in the completion of this project.

I am especially grateful to **Dr Srishti Sharma**, whose expertise, encouragement, and constructive feedback guided this work from inception to completion. Her insights into Natural Language Processing and machine learning were crucial in navigating the challenges of model implementation and evaluation.

I am equally grateful to **The NorthCap University**, where access to a collaborative environment and advanced resources has fostered not only the technical achievements of this project but also the spirit of innovation that underpins it.

This project is as much a reflection of those who have supported and guided me as it is my own effort, and for that, I am sincerely thankful.

Tanishka

Date: 25 November 2024

**ABSTRACT**

Recruitment is a critical process for organizations, requiring efficient screening and evaluation of candidates' resumes to identify the most suitable individuals for a given job description. The traditional manual process of resume screening is time-consuming, error-prone, and often inconsistent. To address these challenges, this project presents an Automated Resume Matching System that leverages Natural Language Processing (NLP) techniques to rank resumes based on their relevance to a specified job description.

The system is designed to process resumes in multiple file formats, including PDF, DOCX, and TXT. It uses robust text extraction methods tailored for each file type to extract the relevant content. The extracted text is then transformed into numerical vectors using TF-IDF (Term Frequency-Inverse Document Frequency), which quantifies the importance of terms within the resumes relative to the job description. By employing cosine similarity, the system calculates the degree of alignment between the job description and each resume. This allows for an accurate assessment of relevance and ensures that only the most suitable resumes are shortlisted.

The output of the system is a ranked list of the top five resumes, sorted by their similarity scores to the job description. This significantly reduces the time and effort required for the initial stages of recruitment, providing recruiters with a concise and prioritized list of candidates to consider further. The project is implemented in Python and integrates libraries like PyPDF2, docx2txt, and scikit-learn for text processing and similarity computations.

This project not only automates the resume screening process but also enhances its accuracy and scalability. It builds upon existing research in document similarity and NLP applications while offering a practical tool for real-world use. Additionally, the system incorporates error handling for missing or unsupported file types, making it robust and user-friendly. The project concludes with suggestions for future improvements, including the integration of semantic analysis for better contextual understanding, support for additional file formats, and the development of a graphical user interface (GUI) for improved accessibility.

**CONTENTS**

|  |  |
| --- | --- |
| Topic | Page |
| Software Requirements | 5-6 |
| Introduction | 7 |
| Relevance of problem statement | 8-9 |
| Survey Of Domain | 10-13 |
| Project Implementation and My Contributions | 14-17 |
| Code | 18-19 |
| Result | 20 |
| Block Diagram | 21 |
| Conclusion | 22 |
| Github Link | <https://github.com/Tanishka5502/NLP-7th-Sem-Project> |

**SOFTWARE REQUIREMENTS**

**1. Operating System**

* Supported Platforms:
  + Windows 10 or later
  + macOS 10.15 (Catalina) or later
  + Linux distributions (Ubuntu 18.04 or later preferred)

**2. Programming Language**

* Python: Version 3.7 or higher

**3. Required Python Libraries**

The following Python libraries are required for the project:

* Text Extraction:
  + PyPDF2: For extracting text from PDF files.
  + docx2txt: For extracting text from DOCX files.
* Natural Language Processing:
  + scikit-learn: For TF-IDF vectorization and cosine similarity calculation.
* Utilities:
  + os: To handle file path operations and file validation.

**4. Development Environment**

* Recommended IDEs or Text Editors:
  + Visual Studio Code: With Python extension installed.
  + PyCharm: Community or Professional edition.
  + Jupyter Notebook: For modular testing of individual components.

**5. Dependency Management**

* pip: Python package installer to manage dependencies.
  + Command to install all required libraries:

pip install PyPDF2 docx2txt scikit-learn

**6. Computational Requirements**

* Processor: Intel i5 or equivalent processor or higher.
* RAM: Minimum 4 GB (8 GB or higher recommended for better performance with large datasets).
* Storage: At least 500 MB free disk space for scripts, libraries, and sample resumes.

**INTRODUCTION**

In today’s fast-paced world, organizations are inundated with resumes for every job opening they announce. The sheer volume of resumes received often makes it a daunting task for recruiters to manually screen and shortlist candidates. Traditional resume screening methods rely heavily on human effort, which is both time-consuming and prone to biases or errors. As businesses strive to streamline recruitment processes and reduce time-to-hire, automating the initial stages of resume evaluation has become a necessity.

The advent of Natural Language Processing (NLP) and Machine Learning (ML) has introduced innovative ways to handle and process large volumes of text data. Leveraging these technologies, automated resume screening systems can intelligently analyze resumes and match them against job descriptions. Such systems not only save significant time and effort but also enhance the accuracy and fairness of the recruitment process.

This project presents an Automated Resume Matching System that uses NLP techniques to compare the content of resumes with a given job description. The system employs TF-IDF (Term Frequency-Inverse Document Frequency) vectorization to represent textual data as numerical vectors and computes the relevance of each resume using cosine similarity. The output is a ranked list of resumes, making it easier for recruiters to identify the most suitable candidates for further evaluation.

The system is designed to process resumes in multiple file formats, including PDF, DOCX, and TXT, ensuring versatility and ease of use. It incorporates robust text extraction techniques tailored to each file type, making it capable of handling diverse formats commonly used by candidates. The modular design and scalability of the system make it an ideal solution for modern-day recruitment challenges.

**RELEVANCE OF PROBLEM STATEMENT**

The recruitment process is critical for organizations as hiring the right talent is fundamental to their success. However, several challenges persist in traditional resume screening:

1. Volume of Applications:

Job openings, especially in popular fields like technology and healthcare, often attract hundreds or even thousands of applications. Screening such a high volume manually is impractical and prone to delays.

1. Time Constraints:

Recruiters need to shortlist candidates quickly to maintain the pace of hiring, but manual review of resumes slows down the process significantly, affecting the organization’s overall productivity.

1. Inconsistencies and Bias:

Human screening is subject to biases, which can lead to unfair evaluations. Additionally, different reviewers may interpret resumes differently, resulting in inconsistencies in the selection process.

1. Varied Resume Formats:

Resumes come in diverse file formats (PDF, DOCX, TXT) and layouts. Extracting meaningful content from these files is often challenging and requires specialized tools.

1. Keyword Overreliance in ATS Systems:

Many existing Applicant Tracking Systems (ATS) rely on simple keyword matching. This approach lacks context, as it may overlook resumes with relevant experience described using synonyms or different phrasing.

Given these challenges, an automated system that can intelligently evaluate resumes for their relevance to job descriptions is crucial. By employing advanced NLP techniques, this project addresses these pain points, offering a scalable, efficient, and unbiased solution to resume screening.

The relevance of this project extends beyond individual recruiters to large organizations, staffing agencies, and even job portals. It ensures that the initial stages of recruitment are handled effectively, saving both time and resources. Furthermore, as organizations increasingly adopt data-driven approaches, this system aligns well with modern recruitment strategies by integrating advanced technologies into traditional workflows.

In summary, the Automated Resume Matching System tackles one of the most pressing challenges in recruitment today efficient and fair initial screening of resumes. Its design, grounded in NLP and machine learning principles, ensures scalability, accuracy, and ease of use, making it a valuable tool for organizations of all sizes.

**SURVEY OF DOMAIN**

The recruitment domain has undergone significant transformation over the years, with advancements in technology reshaping the way organizations attract, evaluate, and hire talent. Traditional methods of resume screening, which involve manual evaluation by recruiters, are increasingly being replaced by automated solutions powered by Artificial Intelligence (AI) and Natural Language Processing (NLP). Below is an overview of the domain, existing approaches, and related works that highlight the relevance and necessity of automated resume matching systems.

**1. Evolution of Resume Screening Methods**

* **Manual Screening**:
  + The conventional approach involves human recruiters reading through resumes and matching them to job descriptions.
  + Challenges include time inefficiency, inconsistency, and bias.
* **Keyword-Based Applicant Tracking Systems (ATS)**:
  + Early automation systems use simple keyword matching to filter resumes.
  + Limitations:
    - Lack of context or understanding of synonyms.
    - Overlooks relevant resumes with non-standard phrasing.
* **Modern AI-Based Systems**:
  + Utilize advanced algorithms such as TF-IDF, machine learning, and semantic analysis.
  + Focus on contextual understanding and scoring resumes based on relevance.

**2. Existing Tools and Systems**

Several systems and tools have been developed in the domain of automated resume screening:

* **Commercial Applicant Tracking Systems (ATS)**:
  + Examples: Workday, Greenhouse, Lever, and Bullhorn.
  + These systems provide keyword filtering and basic matching features.
  + Often criticized for inaccuracies due to reliance on rigid keyword matching algorithms.
* **AI-Powered Platforms**:
  + Tools like HireVue and Pymetrics leverage AI for candidate assessment.
  + Some systems include personality analysis or interview simulations alongside resume matching.
* **Open-Source and Academic Research**:
  + Several academic projects propose NLP and machine learning-based approaches to resume screening.
  + Open-source tools focus on specific aspects, such as text extraction or similarity computation.

**3. Research in Text Similarity for Recruitment**

The use of NLP and machine learning for text similarity and relevance scoring has been widely explored in academia and industry:

* **TF-IDF (Term Frequency-Inverse Document Frequency)**:
  + Widely used for text representation in information retrieval systems.
  + Strengths:
    - Evaluates the importance of terms in a document relative to a corpus.
    - Efficient for structured and short texts like resumes.
  + Limitations:
    - Lacks deep contextual understanding of words.
* **Cosine Similarity**:
  + A standard metric for measuring textual similarity.
  + Well-suited for determining the alignment between job descriptions and resumes.
* **Word Embeddings and Semantic Analysis**:
  + Techniques like Word2Vec, GloVe, and BERT have been employed to capture deeper semantic meanings.
  + These methods are computationally intensive and may not be practical for all systems.

**4. Gaps in Existing Systems**

Despite advancements, there are limitations in current resume screening systems:

* **Overreliance on Keywords**:
  + Many systems fail to account for variations in phrasing or synonyms.
  + Candidates with creative formatting may be penalized.
* **Inability to Process Diverse File Formats**:
  + Some tools struggle with non-standard resume formats or lack robust text extraction capabilities.
* **Bias in Algorithmic Design**:
  + Machine learning models trained on biased datasets may inadvertently favor certain demographics.

**5. Relevance of This Project**

This project addresses key gaps in the domain:

1. **Support for Multiple File Formats**:
   * Handles PDF, DOCX, and TXT formats seamlessly, ensuring inclusivity.
2. **Contextual Matching**:
   * Uses TF-IDF and cosine similarity to match resumes based on relevance rather than simple keyword occurrences.
3. **Efficiency and Scalability**:
   * Automates resume screening to save time and scale across large applicant pools.
4. **Open-Source and Accessible**:
   * Designed to be cost-effective and adaptable for organizations of all sizes.

**PROJECT IMPLEMENTATION AND MY CONTRIBUTIONS**

**Project Implementation**

The project was designed and implemented as an automated resume matching system capable of ranking resumes based on their relevance to a given job description. The implementation involved multiple stages, each addressing key aspects of text extraction, data preprocessing, and similarity computation. Below is a detailed breakdown of the project implementation:

**1. File Handling and Text Extraction**

* **Objective**: Extract text content from resumes stored in various formats (PDF, DOCX, and TXT).
* **Implementation Steps**:
  + **PDF Files**:
    - Used PyPDF2 to read and extract textual content from PDF resumes. Each page of the PDF was iteratively processed.
  + **DOCX Files**:
    - Utilized the docx2txt library to extract text from Microsoft Word documents.
  + **TXT Files**:
    - Read text files directly using Python’s built-in file handling methods.
  + **Error Handling**:
    - Implemented checks to handle unsupported file types and missing files gracefully.

**2. Data Preprocessing**

* **Objective**: Prepare the extracted text for vectorization and similarity analysis.
* **Implementation Steps**:
  + Normalized text by removing unnecessary whitespace, punctuation, and special characters.
  + Ensured that all textual data (job description and resumes) were encoded in UTF-8 to prevent encoding issues.

**3. Text Vectorization**

* **Objective**: Represent textual data numerically for similarity computation.
* **Implementation**:
  + Used **TF-IDF (Term Frequency-Inverse Document Frequency)** vectorization from the scikit-learn library.
  + Combined the job description and all resumes into a single corpus for vectorization.
  + Generated feature vectors for the job description and each resume.

**4. Similarity Computation**

* **Objective**: Calculate the relevance of each resume to the job description.
* **Implementation**:
  + Applied **cosine similarity** to compute the degree of alignment between the job description vector and each resume vector.
  + Sorted the resumes based on their similarity scores in descending order.

**5. Result Ranking**

* **Objective**: Output a ranked list of the top 5 resumes.
* **Implementation**:
  + Indexed the similarity scores and linked them to their respective resume files.
  + Displayed the top 5 resumes along with their similarity scores for easy interpretation.

**6. Modular Design**

* Designed the system with reusable functions for file handling, text extraction, vectorization, and similarity calculation. This ensures scalability and ease of maintenance.

**My Contributions**

The successful implementation of this project required active involvement in every phase, from design to testing. Below are my specific contributions:

**1. System Design and Architecture**

* Conceptualized the overall workflow, including input processing, text extraction, similarity computation, and result ranking.
* Designed a modular structure to ensure reusability and scalability.

**2. File Handling and Text Extraction**

* Developed robust functions to extract text from PDF, DOCX, and TXT files.
* Researched and integrated appropriate libraries (PyPDF2 and docx2txt) for reliable text extraction.

**3. Preprocessing and Normalization**

* Implemented text cleaning methods to prepare extracted content for vectorization.
* Ensured compatibility with diverse resume formats and handled exceptions for unsupported file types.

**4. Vectorization and Similarity Analysis**

* Utilized the scikit-learn library to implement TF-IDF vectorization and cosine similarity.
* Fine-tuned the vectorization parameters for improved relevance scoring.

**5. Testing and Validation**

* Conducted extensive testing using sample resumes and job descriptions to validate system performance.
* Debugged and resolved issues related to file compatibility and encoding errors.

**6. Documentation**

* Documented the code thoroughly, including detailed comments and usage instructions.
* Prepared user instructions for running the system effectively.

**7. Future Enhancements**

* Suggested potential improvements, such as incorporating semantic analysis using word embeddings (e.g., Word2Vec or BERT) and adding a graphical user interface (GUI) for non-technical users.

**CODE**

!pip install docx2txt

!pip install PyPDF2

import os

import docx2txt

import PyPDF2

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.metrics.pairwise import cosine\_similarity

def extract\_text\_from\_pdf(file\_path):

    """Extract text from a PDF file."""

    text = ""

    with open(file\_path, 'rb') as file:

        reader = PyPDF2.PdfReader(file)

        for page in reader.pages:

            text += page.extract\_text()

    return text

def extract\_text\_from\_docx(file\_path):

    """Extract text from a DOCX file."""

    return docx2txt.process(file\_path)

def extract\_text\_from\_txt(file\_path):

    """Extract text from a TXT file."""

    with open(file\_path, 'r', encoding='utf-8') as file:

        return file.read()

def extract\_text(file\_path):

    """Determine the file type and extract text accordingly."""

    if file\_path.endswith('.pdf'):

        return extract\_text\_from\_pdf(file\_path)

    elif file\_path.endswith('.docx'):

        return extract\_text\_from\_docx(file\_path)

    elif file\_path.endswith('.txt'):

        return extract\_text\_from\_txt(file\_path)

    else:

        print(f"Unsupported file type for {file\_path}. Skipping.")

        return ""

def match\_resumes(job\_description, resume\_files):

    """Match resumes against a job description and return ranked results."""

    resumes = [extract\_text(resume) for resume in resume\_files]

    if not resumes:

        print("No valid resumes were processed.")

        return []

    # Vectorize the job description and resumes

    vectorizer = TfidfVectorizer().fit\_transform([job\_description] + resumes)

    vectors = vectorizer.toarray()

    # Calculate cosine similarities

    job\_vector = vectors[0]

    resume\_vectors = vectors[1:]

    similarities = cosine\_similarity([job\_vector], resume\_vectors)[0]

    # Rank resumes by similarity

    ranked\_indices = similarities.argsort()[::-1][:5]  # Top 5 matches

    ranked\_resumes = [(resume\_files[i], round(similarities[i], 2)) for i in ranked\_indices]

    return ranked\_resumes

def main():

    # Hardcoded paths to resume files

    resume\_files = [

        "/content/Data Scientist.pdf",

        "/content/Finantial Analist.pdf",

        "/content/Healthcare.txt",

        "/content/Software engineer.pdf",

        "/content/Teacher.pdf",

        "/content/advocate.txt",

        "/content/backend developer.pdf",

        "/content/banking.txt",

        "/content/data engineer.pdf",

        "/content/designer.pdf",

        "/content/dev oops engineer.pdf",

        "/content/Engineer.pdf",

        "/content/marketing specialist.pdf",

        "/content/project manager.pdf",

        "/content/software developer.pdf"

    ]

    # Check if all files exist

    valid\_files = [file for file in resume\_files if os.path.exists(file)]

    if len(valid\_files) != len(resume\_files):

        print("Some files were not found and will be skipped.")

    # Input job description

    job\_description = input("Enter the job description: ")

    # Match resumes

    ranked\_resumes = match\_resumes(job\_description, valid\_files)

    # Display results

    print("\nTop 5 Matching Resumes:")

    for rank, (filename, score) in enumerate(ranked\_resumes, start=1):

        print(f"{rank}. {filename} - Similarity Score: {score}")

**RESULT**

if \_\_name\_\_ == "\_\_main\_\_":

    main()

Enter the job description: Job Description: We are looking for a highly skilled Software Engineer to join our development team. The ideal candidate will have experience in Python, Java, or C++, along with a strong understanding of software development life cycles. You will collaborate with cross-functional teams to design, develop, and maintain scalable software solutions. Key Responsibilities: Develop high-quality software that meets technical and functional requirements. Write clean, maintainable, and efficient code. Troubleshoot, debug, and resolve software issues. Collaborate with team members on architectural designs and technical decisions. Skills Required: Proficiency in programming languages such as Python, Java, or C++. Experience with frameworks like Django, React, or Spring Boot. Knowledge of databases (SQL, NoSQL). Familiarity with cloud platforms like AWS or Azure.

Top 5 Matching Resumes:

1. /content/Software engineer.pdf - Similarity Score: 0.29

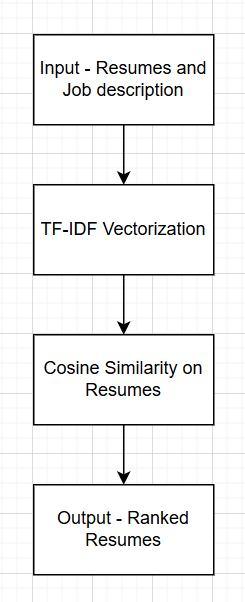
2. /content/Engineer.pdf - Similarity Score: 0.22

3. /content/designer.pdf - Similarity Score: 0.17

4. /content/Teacher.pdf - Similarity Score: 0.15

5. /content/banking.txt - Similarity Score: 0.15

**BLOCK DIAGRAM**



**CONCLUSION**

The Automated Resume Matching System successfully demonstrates the potential of Natural Language Processing (NLP) and Machine Learning (ML) techniques in addressing the challenges of modern recruitment processes. By automating the initial screening of resumes, the system provides a scalable, efficient, and unbiased solution for organizations to identify the most relevant candidates for a job.

Key accomplishments of the project include:

* Versatile Text Extraction: The system supports multiple file formats, including PDF, DOCX, and TXT, ensuring wide applicability in real-world scenarios.
* Accurate Matching: Using TF-IDF vectorization and cosine similarity, the system evaluates resumes based on their contextual relevance to the job description, overcoming the limitations of traditional keyword-based methods.
* Efficiency: Automating the resume screening process saves significant time and effort for recruiters, allowing them to focus on higher-value tasks such as interviews and assessments.
* Cost-Effectiveness: As an open-source and lightweight solution, this project is accessible to organizations of all sizes, including small and medium-sized enterprises.

While the current implementation addresses many challenges in the recruitment process, there is room for improvement. Future enhancements could include incorporating advanced semantic analysis using embeddings like Word2Vec or BERT, developing a graphical user interface (GUI) for non-technical users, and integrating the system with existing applicant tracking systems (ATS).

In conclusion, this project showcases a practical application of NLP and machine learning to streamline and improve recruitment workflows. It has the potential to make hiring processes faster, fairer, and more reliable, meeting the demands of today’s competitive talent landscape.