INDUSTRIAL ORIENTED MINI PROJECT

Report

On

RESUME CATEGORIZER USING MACHINE LEARNING

Submitted in partial fulfilment of the requirements for the award of the degree of

BACHELOR OF TECHNOLOGY

In

COMPUTER SCIENCE AND BUSINESS SYSTEMS

By

Y.S.S.K Keerthija – 22261A3261

C.Shashank Reddy – 22261A3211

Under the guidance of

Dr. U.Chaitanya

Assistant Professor, Department of IT

DEPARTMENT OF INFORMATION TECHNOLOGY

MAHATMA GANDHI INSTITUTE OF TECHNOLOGY (AUTONOMOUS)

(Affiliated to JNTUH, Hyderabad; Eight UG Programs Accredited by NBA; Accredited by NAAC with 'A++' Grade)

Gandipet, Hyderabad, Telangana, Chaitanya Bharati (P.O), Ranga Reddy District, Hyderabad– 500075, Telangana

2024-2025

CERTIFICATE

This is to certify that the **Industrial Oriented Mini Project** entitled **Resume Categorizer Using Machine Learning** submitted by **Y.S.S.K Keerthija (22261A3261) and C.Shashank Reddy (22261A3211)** in partial fulfillment of the requirements for the Award of the Degree of Bachelor of Technology in Computer Science and Business Systems as specialization is a record of the bona fide work carried out under the supervision of **Dr. U.Chaitanya**, and this has not been submitted to any other University or Institute for the award of any degree or diploma.

Internal Supervisor:	**IOMP Supervisor:**
Dr. U. Chaitanya	**Dr. N. Sree Divya**
Assistant Professor	Assistant Professor
Dept. of IT	Dept. of IT

EXTERNAL EXAMINAR

Dr. D. Vijaya Lakshmi
Professor and HOD
Dept. of IT

DECLARATION

We hear by declare that the **Industrial Oriented Mini Project** entitled **Resume Categorizer Using Machine Learning** is an original and bonafide work carried out by us as a part of fulfilment of Bachelor of Technology in Computer Science and Business Systems, Mahatma Gandhi Institute of Technology, Hyderabad, under the guidance of **Dr.U. Chaitanya , Assistant Professor**, Department of IT, MGIT.
No part of the project work is copied from books /journals/ internet and wherever the portion is taken, the same has been duly referred in the text. The report is based on the project work done entirely by us and not copied from any other source.

Y.S.S.K Keerthija - 22261A3261
C.Shashank Reddy - 22261A3211

ACKNOWLEDGEMENT

The satisfaction that accompanies the successful completion of any task would be incomplete without introducing the people who made it possible and whose constant guidance and encouragement crowns all efforts with success. They have been a guiding light and source of inspiration towards the completion of the **Industrial Oriented Mini Project**.

We would like to express our sincere gratitude and indebtedness to our internal supervisor **Dr. U. Chaitanya , Assistant Professor**, Dept. of IT, who has supported us throughout our project with immense patience and expertise.

We are also thankful to our honourable Principal of MGIT **Prof. G. Chandramohan Reddy** and **Dr. D. Vijaya Lakshmi**, Professor and HOD, Department of IT, for providing excellent infrastructure and a conducive atmosphere for completing this **Industrial Oriented Mini Project** successfully.

We are also extremely thankful to our IOMP supervisor **Dr. N. Sreedivya**, Assistant Professor, Department of IT, senior faculty **Mrs. A.V.L Prasuna** Department of IT and **Ms. M. Varalakshmi** Department of IT for their valuable suggestions and guidance throughout the course of this project.

We convey our heartfelt thanks to the lab staff for allowing us to use the required equipment whenever needed.

Finally, we would like to take this opportunity to thank our families for their support all through the work. We sincerely acknowledge and thank all those who gave directly or indirectly their support for completion of this work.

Y.S.S.K Keerthija - 22261A3261
C.Shashank Reddy - 22261A3211

ABSTRACT

Automated resume categorization streamlines the recruitment process by leveraging machine learning and natural language processing techniques. Uploaded PDF resumes are processed to extract and clean textual content, enabling accurate feature extraction using the TF-IDF method. A pre-trained Logistic Regression classifier predicts the most relevant job category for each resume based on its content. The application, developed with Streamlit, offers a user-friendly interface for batch uploading and categorization. Each resume is sorted into a corresponding category folder, and years of experience are extracted for additional insights. Results are presented in a tabular format and can be exported as a CSV file for further analysis. The system handles multiple files efficiently and provides robust error management for unsupported or ambiguous inputs. Integration of machine learning models with a web interface demonstrates practical HR automation. Organizations benefit from reduced manual effort and improved consistency in resume screening. The solution is scalable and adaptable to various recruitment needs. Customizable category mappings allow the system to be tailored for different industries or job roles. The modular design supports easy updates to the classification model as new data becomes available. Security and privacy considerations are addressed by processing files locally without uploading sensitive information to external servers. Comprehensive test cases ensure reliability across a range of input scenarios, including edge cases and error conditions. The project exemplifies the effective application of data science to real-world human resources challenges, offering a foundation for further enhancements such as skill extraction or candidate ranking

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

1.1 MOTIVATION

Manual screening and categorization of resumes is a time-consuming and error-prone process for recruiters, especially when dealing with large volumes of applications. The increasing diversity in resume formats and job roles further complicates efficient candidate sorting. Automation in resume processing can significantly reduce the workload on human resources teams, allowing them to focus on more strategic tasks. Leveraging machine learning and natural language processing enables accurate and consistent classification of resumes, minimizing human bias and oversight. The demand for intelligent recruitment solutions is growing as organizations seek to improve hiring efficiency and candidate experience. By automating resume categorization, companies can accelerate the shortlisting process and ensure that suitable candidates are identified promptly. The integration of such technology also supports data-driven decision-making in talent acquisition. Streamlining resume management not only saves time but also enhances the overall quality of recruitment outcomes. The project aims to bridge the gap between traditional HR practices and modern technological advancements. Ultimately, the motivation lies in empowering organizations to build stronger teams through smarter, faster, and fairer recruitment processes.

1.2 PROBLEM STATEMENT

The problem statement for the "Resume Categorizer Application" project is to address the inefficiencies and time-consuming nature of manually sorting and categorizing large volumes of resumes received by HR departments, recruitment agencies, and job portals. The goal is to develop an automated system that can accurately categorize resumes into predefined job categories using machine learning, thereby streamlining the recruitment process, improving candidate-job matching, and providing valuable insights into application trends.

1.3 EXISTING SYSTEM

The existing resume screening process remains predominantly manual, with HR professionals reviewing each application individually through spreadsheets and email correspondence. While traditional Applicant Tracking Systems (ATS) introduce some level of automation, they are generally limited to basic keyword matching techniques for initial candidate filtering.

These conventional systems often lack sophisticated categorization capabilities and do not incorporate intelligent analysis of resume content. Consequently, resumes are frequently shortlisted based solely on keyword presence rather than on true contextual relevance. This can result in the rejection of qualified candidates whose resumes do not include the exact expected terms.

Manual screening is also time-intensive and susceptible to human error, making it inefficient—especially when managing a high volume of applications. Furthermore, basic ATS platforms provide minimal assistance in organizing or classifying resumes, placing an additional burden on HR teams to manually sort and interpret applicant data.

Because keyword-based systems are unable to grasp the underlying context or meaning within resumes, they tend to produce inaccurate results and weak candidate-role alignment. Overall, the current approach lacks efficiency, contextual intelligence, and effectiveness in identifying the most suitable candidates for specific roles.

1.3.1 LIMITATIONS

- **Lack of Contextual Understanding:** Conventional systems depend heavily on simple keyword matching, lacking the ability to interpret the context or true relevance of resume content. This often results in inaccurate filtering and the exclusion of highly qualified candidates whose resumes do not contain exact keyword matches.

- **Manual and Time-Consuming Process:** Resume screening frequently involves manual review and organization by HR teams, making the process slow, resource-intensive, and vulnerable to human error—particularly when processing a high volume of applications.

- **Limited Automation and Categorization:** Basic ATS tools offer minimal automation and weak categorization features. They provide little to no support for intelligent resume analysis or structured organization, ultimately diminishing the overall efficiency and effectiveness of the screening process.

1.4 PROPOSED SYSTEM

The proposed solution is an intelligent and automated resume categorization application powered by machine learning. It features a user-friendly web interface that enables users to upload multiple PDF resumes with ease. Upon upload, the system performs text extraction and cleaning to remove irrelevant content and standardize the input data.

Leveraging a pre-trained TF-IDF vectorizer and classification model, the system analyzes the resume content to accurately determine the most appropriate job category for each candidate. In addition, it extracts key information such as the candidate's years of experience from the text.

Resumes are then automatically sorted into folders based on their predicted job categories, significantly improving organization and efficiency. The system presents the results—including the resume filename, predicted category, and years of experience—in a structured, easy-to-read table format. Users also have the option to download the complete categorization output as a CSV file for further analysis or documentation.

By eliminating the need for manual screening, reducing human error, and incorporating contextual understanding through machine learning, the proposed system delivers a smarter, faster, and more reliable approach to resume management and talent acquisition.

1.4.1 ADVANTAGES

- **Time Efficiency:** Automates the categorization process, significantly reducing the time required for manual resume screening.
- **Consistency:** Ensures uniform classification of resumes, minimizing human errors and subjective biases.
- **Scalability:** Capable of handling large volumes of resumes simultaneously without a drop in performance.
- **User-Friendly Interface:** Provides an intuitive web interface for easy uploading, categorization, and result retrieval.
- **Accurate Classification:** Utilizes machine learning techniques for reliable and precise categorization based on resume content.

- **Enhanced Organization:** Automatically sorts resumes into category-specific folders, simplifying file management for recruiters.
- **Actionable Insights:** Extracts additional information such as years of experience, aiding in better candidate evaluation.
- **Customizability:** Allows for easy updates to job categories and model retraining as organizational needs evolve.
- **Data Privacy:** Processes resumes locally, ensuring sensitive candidate information is not exposed to external servers.
- **Downloadable Results:** Offers results in a structured, downloadable CSV format for further analysis and record-keeping.

1.5 OBJECTIVES

- **Automated Resume Classification**
  Implemented intelligent automation to categorize resumes, significantly reducing the manual effort and time spent on screening.
  Enabled recruiters to focus on strategic hiring and candidate engagement rather than administrative tasks.
- **Improved Matching Accuracy**
  Increased precision in mapping candidates to relevant job categories, resulting in higher quality shortlists.
  Minimized skill-role mismatches and improved interview-to-offer conversion rates.
- **Strengthened Domain Generalization**
  Effectively categorized resumes across various industries, including IT, Finance, Healthcare, and Education.
  Overcame domain-specific limitations of traditional systems, enhancing adaptability across sectors.

1.6 HARDWARE AND SOFTWARE REQUIREMENTS

1.6.1 SOFTWARE REQUIREMENTS

- **Programming Language**: Python 3.8+ (NLTK, scikit-learn, pandas, numpy)
- **Backend**: Flask/FastAPI, RESTful APIs, ML models (TF-IDF, Logistic Regression)
- **Frontend**: Streamlit, HTML5, CSS3
- **Database**: Resumes Dataset from Kaggle
- **Development Environment**: Cursor, Jupyter Notebook
- **Operating System**: Windows 11

1.6.2 HARDWARE REQUIREMENTS

- **Processor**: Intel i3 (min), i5/i7 (recommended), multi-core (server)
- **GPU**: Integrated (basic), NVIDIA GPU (optional, CUDA support)
- **RAM**: 4GB (min), 8GB (recommended), 16GB+ (server)
- **Storage**: 256GB HDD (min), 512GB SSD (recommended), 1TB+ (server)

2. LITERATURE SURVEY

T.K. Das, A.K. Tripathy et al. explored various ML models for resume classification, comparing their effectiveness in automating screening. The study emphasizes reduced manual effort but lacks performance metrics and real-world validation, suggesting future work on deployment and benchmarking [1].

Irfan Ali et al. developed a resume classification system using NLP and ML techniques tailored to recruitment. It improved efficiency but was limited to specific domains, lacking generalization across diverse resumes. Further research is needed for broader adaptability [2].

S.K. Singh et al. proposed an ML-based system combining NLP for resume extraction and classification to support intelligent screening. While scalable and precise, the system needs domain-specific training and hasn’t been tested across job sectors, indicating limited generalizability [3].

M. Sharma, P. Gupta et al. introduced a BERT-based deep learning framework for resume classification to enhance semantic understanding. It improved accuracy but required high computational resources and supported only English, pointing to a need for multilingual, resource-efficient models [4].

R. Kumar and S. Iyer proposed a hybrid system combining rule-based filtering with ML classification to improve precision and customization. Though effective, the rule creation was time-consuming and lacked adaptability, suggesting future work on domain-independent rule generation [5].

Table 2.1 Literature Survey of Resume Categorizer

Reference	**Author & Year**	**Journal / Conference**	**Method / Approach**	**Merits**	**Limitations**	**Research Gap**
[1]	T.K. Das, A.K. Tripathy et al., 2022	Int. J. of Computer Apps Tech & Research, Vol. 9(9)	Compared ML models for resume classification	Comparative analysis of ML models	No performance metrics; not validated on large datasets	Needs real-world benchmarks and testing on large datasets
[2]	Irfan Ali et al., 2022	ResearchGate, Vol. 41	NLP + ML for resume classification	Improved automation in recruitment	Domain-limited; not tested for diverse resumes	Evaluate generalization across multiple domains
[3]	S.K. Singh et al., 2023	Revue d'Intelligence Artificielle, Vol. 37(3)	NLP-based data extraction + ML classification	Automated screening with improved accuracy	Domain-specific training; limited job sector testing	Cross-domain validation and generalization required
[4]	M. Sharma, P. Gupta et al., 2023	Int. J. of IT & Computer Science, Vol. 15(1)	BERT + deep learning for semantic matching	Context-aware; better semantic analysis	High computational cost; English-only	Explore multilingual, resource-efficient models
[5]	R. Kumar, S. Iyer, 2024	IEEE Access, Vol. 12	Hybrid rule-based + ML classification	Reduced false positives; flexible matching	Rule creation is manual and domain-specific	Develop adaptive, domain-independent rules

3. ANALYSIS AND DESIGN

The recruitment and talent acquisition landscape is increasingly challenged by the need to efficiently and accurately screen vast volumes of resumes to identify the most suitable candidates for specific roles. This project proposes the development of an **Intelligent Resume Categorization Application** that leverages Natural Language Processing (NLP), Machine Learning (ML) techniques, and automated document handling to address these challenges.

The system is designed to streamline the resume screening process by providing accurate, **context-aware categorization** of resumes. This allows Human Resource (HR) professionals to shift their focus from time-consuming manual sorting to high-value strategic decision-making.

At the heart of the application lies its capability to **extract, clean, and analyze textual data** from resumes in real time. By intelligently processing key details such as skills, work experience, and education, the system evaluates resumes based on meaningful content rather than relying solely on basic keyword matches. This context-driven, data-centric approach is vital in recruitment, where the relevance and depth of candidate information are critical to making informed hiring decisions.

To ensure precise categorization, the application utilizes a **hybrid machine learning pipeline**, combining advanced text vectorization methods like **TF-IDF** with robust classification algorithms trained on diverse job categories. The model effectively detects subtle patterns within resume data, enabling accurate classification into roles such as *Java Developer*, *Data Scientist*, or *HR Manager*.

Furthermore, the system **automatically organizes resumes** into category-specific folders and provides a detailed tabular summary of results. Users can also download reports for further analysis, significantly enhancing the **efficiency, speed, and accuracy** of recruitment workflows.

In essence, this intelligent, end-to-end solution transforms traditional resume screening into a **smarter, faster, and more reliable process**, making it highly beneficial for modern hiring environments.

3.1 MODULES

PDF Processing Module

Extracts text content from the uploaded PDF resumes using the PdfReader library.

• Input: Uploaded PDF files.

• Output: Raw text extracted from the PDF resumes.

Text Cleaning Module

Preprocesses the extracted text by removing unwanted elements such as extra spaces, headers, and footers using regular expressions.

• Input: Raw extracted text from resumes.

• Output: Cleaned and normalized text ready for feature extraction.

Feature Transformation Module

Converts the cleaned resume text into numerical features using a TF-IDF vectorizer, preparing the data for machine learning models.

• Input: Cleaned resume text.

• Output: TF-IDF feature vectors representing each resume.

Prediction Module

Uses a pre-trained machine learning model to classify resumes into job categories based on the transformed features.

• Input: TF-IDF feature vectors.

• Output: Predicted job category IDs for each resume.

Category Mapping Module

Maps the predicted numerical category IDs to human-readable job category names for user-friendly display.

• Input: Predicted category IDs.

• Output: Job category names corresponding to each resume.

File Organization Module

Organizes and saves categorized resumes into folders corresponding to their predicted job categories for easy access and management.

• Input: Classified resumes with job category labels.

• Output: Categorized resume files saved in respective directories.

Result Compilation Module

Compiles all categorization results into a DataFrame to display results and enable users to download the classification summary.

• Input: Predicted categories and resume metadata.

• Output: DataFrame summarizing resume classifications, downloadable by the user.

3.2 ARCHITECTURE

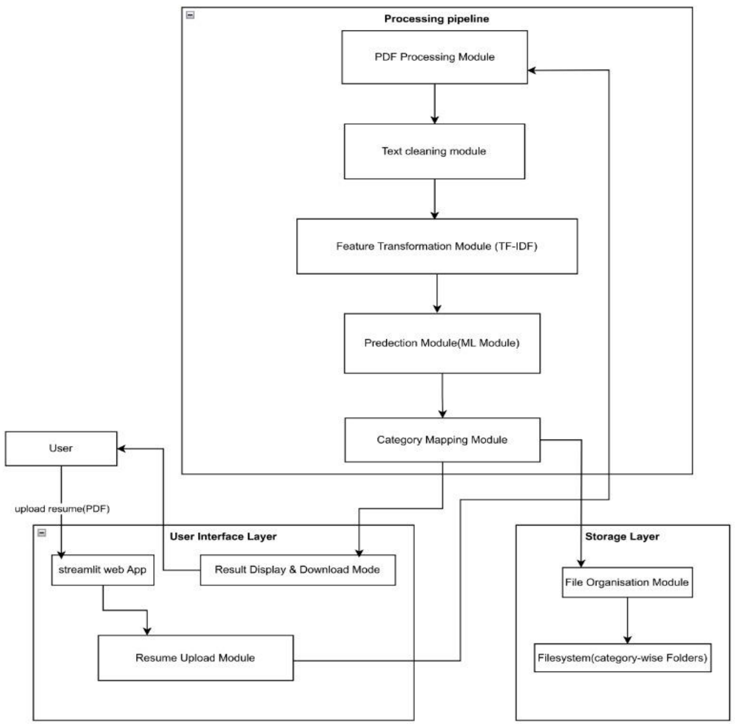


Fig. 3.2.1 Architecture of Resume Categorizer

The architecture of the system, as shown in **Fig. 3.2.1**, is designed to automate and streamline the process of resume screening using machine learning. The system consists of three main layers: the processing pipeline, the user interface layer, and the storage layer.

The process begins with the PDF Processing Module, which extracts raw text from uploaded resumes. This text is then passed to the Text Cleaning Module, where unwanted characters, formatting, and noise are removed to standardize the input. The cleaned text is further processed in the Feature Transformation Module using TF-IDF vectorization, which converts textual content into numerical features suitable for machine learning.

These features are input into the Prediction Module, where a pre-trained machine learning model categorizes the resumes into predefined job roles such as Data Analyst, Software Developer, or UI/UX Designer. The predicted outputs are mapped to their respective job labels using the Category Mapping Module.

Users interact with the system via a Streamlit-based web application. Through the Resume Upload Module, users can upload one or more resumes in PDF format. The application then displays categorized results in a tabular format using the Result Display & Download Module, along with an option to download the output as a CSV file.

Once categorized, resumes are automatically sorted and stored in job-specific directories by the File Organization Module, ensuring they are saved within the appropriate folders in the underlying filesystem.

This architecture ensures an end-to-end flow—from resume upload and processing to categorization and organized storage—while offering a clean and intuitive user experience.

3.3 UML DIAGRAMS

3.3.1 USE CASE DIAGRAMS

A use case diagram is a visual representation that depicts the interactions between various actors and a system, capturing the ways in which users or external entities interact with the system to achieve specific goals. It is an essential tool in system analysis and design, often used in software engineering and business analysis. In a use case diagram, actors are entities external to the system that interact with it, and use cases are specific functionalities or features provided by the system as seen in Fig. 3.3.1.1. These interactions are represented by lines connecting actors to use cases. The diagram helps to illustrate the scope and functionality of a system, providing a high-level view of how users or external entities will interact with it.

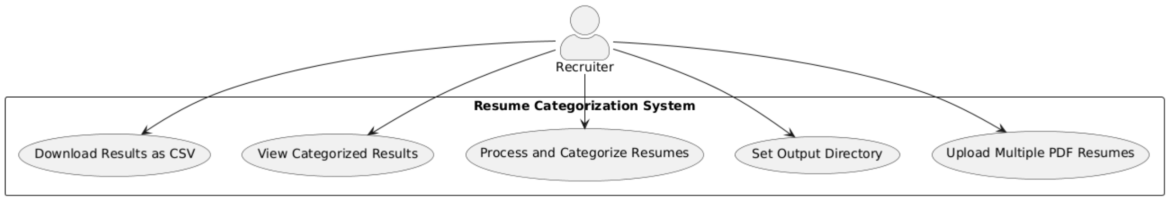

Recruiter
Resume Categorization System
Download Results as CSV
View Categorized Results
Process and Categorize Resumes
Set Output Directory
Upload Multiple PDF Resumes

Fig. 3.3.1.1 Use Case Diagram

Actors:

1. **Recruiter**: The end user who interacts with the system to upload resumes, initiate categorization, and access the categorized results.

Use Cases:

1. **Upload Resume Files**: The recruiter uploads one or more PDF resume files to the system for processing.
2. **Specify Output Directory**: The recruiter selects or specifies a folder where the categorized resumes will be saved.
3. **Start Categorization**: The recruiter initiates the categorization process, which includes parsing resumes, predicting job roles, and extracting key information.
4. **Organize Resumes by Category**: The system automatically organizes resumes into category-specific folders based on predicted job roles.
5. **Extract Years of Experience**: The system parses resume content to extract and display the candidate's total years of experience.
6. **View Categorized Results**: After processing, the recruiter can view the list of resumes along with their predicted job categories and extracted experience.
7. **Download Results as CSV**: The recruiter can download a CSV file that contains the processed results, including file names, categories, and experience data.

3.3.2 CLASS DIAGRAM

A class diagram is a visual representation that models the static structure of a system, showcasing the system's classes, their attributes, methods (operations), and the relationships between them as seen in Fig. 3.3.2.1. It is a key tool in object-oriented design and is commonly used in software engineering to define the blueprint of a system.

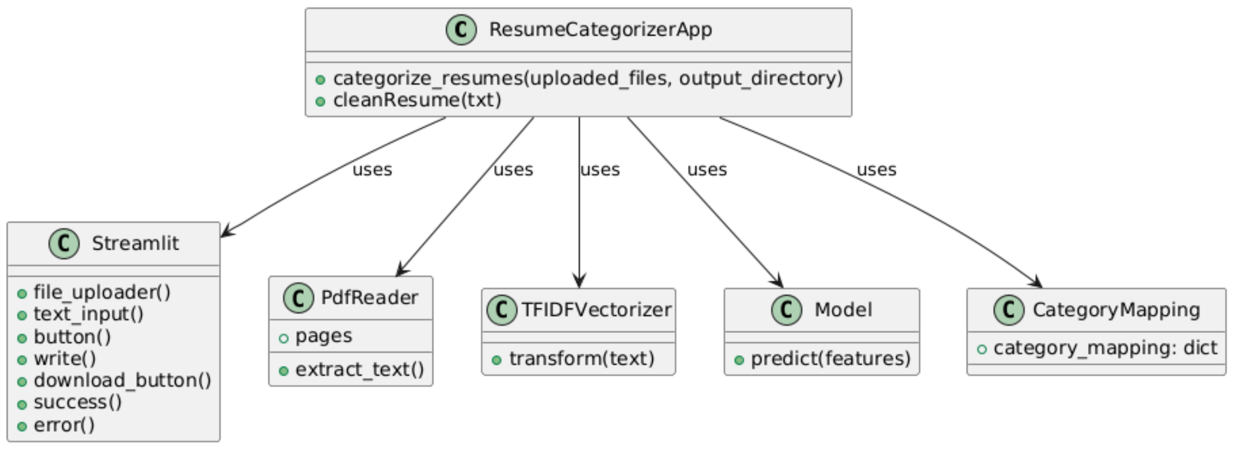


Fig. 3.3.2.1 Class Diagram

Relationships:

1. ResumeCategorizerApp → Streamlit
   - The ResumeCategorizerApp class uses Streamlit functions such as file_uploader(), button(), text_input(), write(), and others to build the interactive web interface.
   - Streamlit handles user input and provides feedback via success() and error() messages.
2. ResumeCategorizerApp → PdfReader
   - The application uses the PdfReader class to read PDF files and extract text using the extract_text() function.
   - This enables parsing of resume content from uploaded documents.
3. ResumeCategorizerApp → TFIDFVectorizer
   - The TFIDFVectorizer class transforms the cleaned resume text into a numerical feature matrix using the transform(text) method.
   - These transformed features are required for machine learning prediction.
4. ResumeCategorizerApp → Model
   - The Model class contains the predict(features) function which takes the vectorized text features and outputs job category predictions using the trained ML model.
5. ResumeCategorizerApp → CategoryMapping

    - This class contains a category_mapping dictionary used to map numerical model predictions to actual category labels (like Data Analyst, Software Developer, etc.).

System Flow:

1. User Interaction
    - A recruiter uploads resumes and selects an output directory via the Streamlit interface.
    - Streamlit functions facilitate UI components and interactions.
2. PDF Text Extraction
    - Uploaded PDF resumes are passed to the PdfReader, which extracts raw text from each document.
3. Text Cleaning and Feature Transformation
    - The raw text is cleaned by the cleanResume(txt) method in the ResumeCategorizerApp.
    - The cleaned text is transformed into numerical features using TFIDFVectorizer.
4. Model Prediction
    - These features are then fed into the Model's predict() function to generate category predictions.
5. Category Mapping and Output
    - The prediction output is matched to category labels using CategoryMapping.
    - Final results are shown on the Streamlit interface and can be downloaded.

3.3.3 ACTIVITY DIAGRAM FOR RESUME CATEGORIZER

An Activity Diagram is a type of behavioral diagram used in Unified Modeling Language (UML) to represent the flow of control or data through the system as seen in Fig. 3.3.3.1. It focuses on the flow of activities and actions, capturing the sequence of steps in a particular process or workflow. Activity diagrams are commonly used to model business processes, workflows, or any sequential activities in a system.

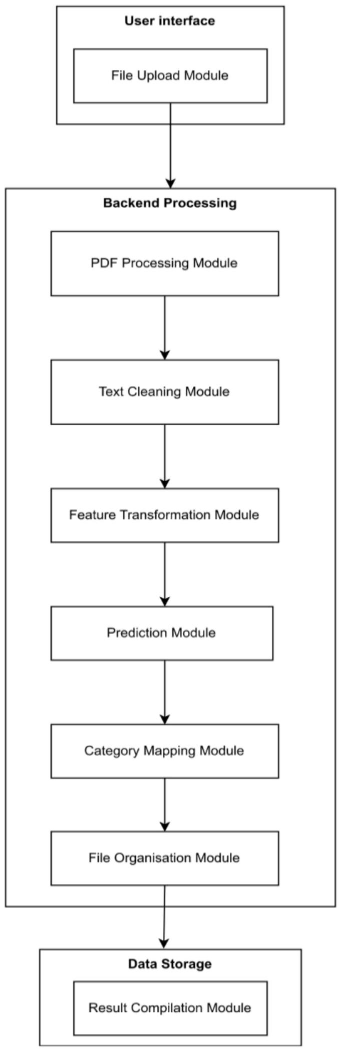


Fig. 3.3.3.1 Activity Diagram

Flow Explanation:

1. **Open Application**
   The user initiates the Resume Categorization system via the web-based interface.
2. **Upload Resume Files**
   The user uploads one or more resume files in PDF format through the File Upload Module located in the User Interface.
3. **PDF Processing Module**
   Once uploaded, the resumes are passed to the PDF Processing Module, which extracts raw textual content from the PDF files.

4. **Text Cleaning Module**
   The extracted text is then cleaned by removing irrelevant characters, formatting inconsistencies, and noise using the Text Cleaning Module.
5. **Feature Transformation Module**
   Cleaned resume text is transformed into machine-readable vectors using the TF-IDF vectorizer in the Feature Transformation Module. These vectors represent the frequency and importance of terms across resumes.
6. **Prediction Module**
   The transformed features are sent to the Prediction Module, where a trained machine learning model predicts the most relevant job category for each resume.
7. **Category Mapping Module**
   Predicted labels are then mapped to human-readable category names (e.g., "Data Analyst", "Software Developer") using a predefined dictionary in the Category Mapping Module.
8. **File Organization Module**
   Based on the predicted categories, resumes are systematically organized into category-specific folders by the File Organization Module.
9. **Result Compilation Module**
   Finally, categorized results are compiled and stored in the Result Compilation Module under the Data Storage layer. Users can access the categorized results, typically available as downloadable CSVs or organized folder structures.
10. **End of Workflow**
    The system completes the categorization task and notifies the user of successful completion. The user can then download results or restart the process for new resumes.

3.3.4 SEQUENCE DIAGRAM

A sequence diagram illustrates the flow of interactions between actors and system components over time as seen in Fig. 3.3.4.1, emphasizing the order in which messages are exchanged to achieve specific functionalities. Actors represent external entities that interact with the system, while lifelines depict the system components involved in the process. Messages are shown as arrows, indicating the flow of information or actions between these elements. By providing a step-by-step view of workflows, sequence diagrams help in understanding and designing the dynamic behaviour of a system.

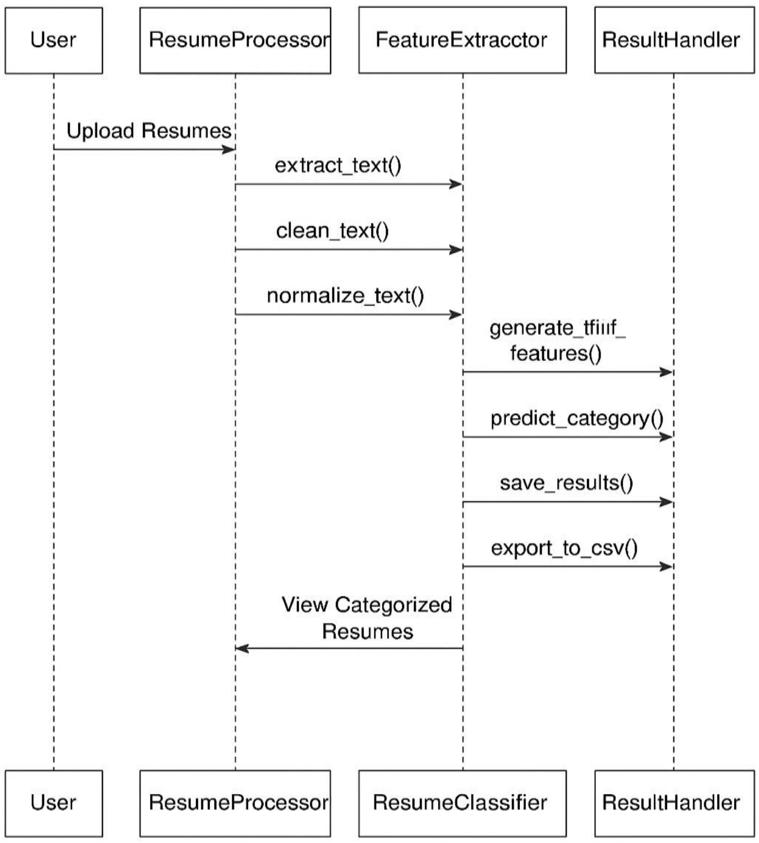


Fig. 3.3.4.1 Sequence Diagram

Key Interactions and Relationships

User and ResumeProcessor

- **Resume Upload:** The user initiates the interaction by uploading one or more resumes through the interface, triggering the resume processing workflow.
- **Text Extraction:** The ResumeProcessor component begins by calling extract_text() to retrieve text data from the uploaded resume PDFs.
- **Text Cleaning:** Next, the component uses clean_text() to remove unwanted formatting, special characters, and irrelevant content.

- **Normalization:** It applies normalize_text() to ensure consistent formatting, such as lowercasing and punctuation removal.

ResumeProcessor and FeatureExtractor

- **Feature Generation:** Once the text is normalized, generate_tfidf_features() is called by the FeatureExtractor to convert the textual content into TF-IDF feature vectors for machine learning processing.

FeatureExtractor and ResumeClassifier

- **Category Prediction:** The transformed features are passed to the ResumeClassifier, which uses a trained machine learning model to run predict_category() and determine the appropriate job category for each resume.

ResumeClassifier and ResultHandler

- **Result Saving:** The predicted results are sent to the ResultHandler, which executes save_results() to store the categorized data.
- **CSV Export:** The handler further calls export_to_csv() to generate a downloadable CSV file containing categorized resume data.

System and User

- **Display of Results:** Finally, the user views the categorized results returned by the system in a clean, organized format, completing the resume categorization process.

3.3.5 COMPONENT DIAGRAM

A Component Diagram is a type of structural diagram used in software engineering to represent the components of a system and how they interact or depend on each other. It shows how the components (which could be software modules, subsystems, or other significant parts) are organized and connected within a system. In this diagram, each component encapsulates a set of related functionalities and interfaces as shown in Fig. 3.3.5.1

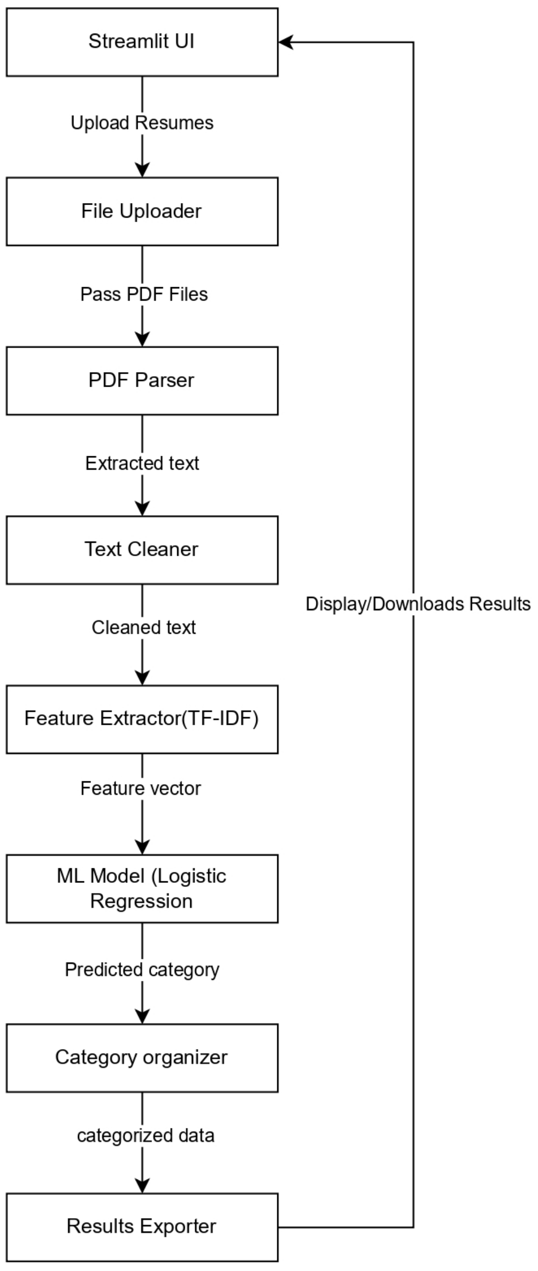


Fig. 3.3.5.1 Component Diagram

- **Streamlit UI**
  Description: This is the front-end user interface built with Streamlit. It allows users to upload resume PDFs for processing and view or download categorized results after processing.

- **File Uploader**
  Description: This module handles the input of resumes. It accepts and forwards the uploaded PDF files to the next component (PDF Parser) for text extraction.
- **PDF Parser**
  Description: This module is responsible for parsing the content from the uploaded PDF resumes. It extracts raw text data from each resume for further cleaning and processing.
- **Text Cleaner**
  Description: This component performs preprocessing on the extracted text. It removes special characters, normalizes case, eliminates stop words, and prepares the text for feature extraction.
- **Feature Extractor (TF-IDF)**
  Description: This module converts the cleaned text into numerical representations using TF-IDF (Term Frequency–Inverse Document Frequency). These feature vectors are crucial for training and prediction using machine learning models.
- **ML Model (Logistic Regression)**
  Description: A trained machine learning model—specifically, Logistic Regression—is used to classify resumes into relevant categories based on the feature vectors.
- **Category Organizer**
  Description: This module maps the model's predicted categories to human-readable labels. It ensures that each resume is associated with the correct job domain (e.g., Data Analyst, Software Engineer).
- **Results Exporter**
  Description: This final module compiles the categorized resumes and allows users to download or view the output in a CSV or structured format directly from the UI.

3.4 METHODOLOGY

1. Data Acquisition

- **Source:** Users upload resumes in PDF format via a Streamlit-based interface.
- **Purpose:** These resumes serve as input data for text extraction, categorization, and analysis.

2. Data Preprocessing

- **Text Extraction:** Raw text is extracted from PDFs using the pypdf library.
- **Text Cleaning:** Special characters, extra whitespace, and irrelevant patterns are removed using regular expressions.
- **Normalization:** Text is lowercased, non-ASCII characters are filtered, and stop words are optionally removed for consistency.

3. Feature Engineering

- **TF-IDF Vectorization:** Cleaned text is transformed into numerical vectors using a pre-trained **TF-IDF** model.
- **Why TF-IDF:** This method captures important, context-specific words across resumes, improving classification accuracy.

4. Label Transformation

- **Label Encoding:** Job categories are mapped to numeric values using LabelEncoder, enabling the model to understand category distinctions during training and prediction.

5. Modeling & Prediction

- **Model Used:** A **pre-trained Logistic Regression classifier** is loaded from a serialized .pkl file.
- **Prediction:** The model outputs the most likely job category for each resume based on the TF-IDF features.
- **Why Logistic Regression:** Chosen for its speed, efficiency, and effectiveness in handling high-dimensional text data.

6. Additional Information Extraction

- **Years of Experience:** A custom regex-based parser extracts and computes the total experience (in years/months) mentioned in resumes.

7. Result Handling & Output

- **Categorization:** Resumes are organized into folders based on predicted job categories.
- **Summary Table:** A table is generated with columns for resume filename, predicted category, and extracted experience.
- **Export Options:** Users can download the summary in **CSV format** for offline use.

4. CODE AND IMPLEMENTATION

4.1 CODE

app.py

```
import os
import pandas as pd
import pickle
from pypdf import PdfReader
import re
import streamlit as st

# Load models
word_vector = pickle.load(open("tfidf.pkl", "rb"))
model = pickle.load(open("model.pkl", "rb"))

def cleanResume(txt):
    cleanText = re.sub('http\S+\s', ' ', txt)
    cleanText = re.sub('RT|cc', ' ', cleanText)
    cleanText = re.sub('#\S+\s', ' ', cleanText)
    cleanText = re.sub('@\S+', '  ', cleanText)
    cleanText = re.sub('[%s]' % re.escape("""!"#$%&'()*+,-./:;<=>?@[\]^_`{|}~"""), ' ',
cleanText)
    cleanText = re.sub(r'[^\x00-\x7f]', ' ', cleanText)
    cleanText = re.sub('\s+', ' ', cleanText)
    return cleanText

def extract_years_of_experience(text):
    # Look for patterns like "5 years", "3+ years", "2 yrs", "24 months"
    years = re.findall(r'(\d+)\s*(?:\+?\s*years?|yrs?)', text, re.IGNORECASE)
    months = re.findall(r'(\d+)\s*months?', text, re.IGNORECASE)
    total_years = sum(int(y) for y in years)
    total_months = sum(int(m) for m in months)
    total_years += total_months // 12
    return total_years

category_mapping = {
    15: "Java Developer",
    23: "Testing",
    8: "DevOps Engineer",
    20: "Python Developer",
    24: "Web Designing",
    12: "HR",
    13: "Hadoop",
    3: "Blockchain",
    10: "ETL Developer",
    18: "Operations Manager",
    6: "Data Science",
    22: "Sales",
    16: "Mechanical Engineer",
    1: "Arts",
    7: "Database",
    11: "Electrical Engineering",
    14: "Health and fitness",
    19: "PMO",
    4: "Business Analyst",
    9: "DotNet Developer",
    2: "Automation Testing",
    17: "Network Security Engineer",
    21: "SAP Developer",
    5: "Civil Engineer",
    0: "Advocate",
}

def categorize_resumes(uploaded_files, output_directory):
    if not os.path.exists(output_directory):
        os.makedirs(output_directory)

    results = []

    for uploaded_file in uploaded_files:
        if uploaded_file.name.endswith('.pdf'):  # Change the extension as needed
            reader = PdfReader(uploaded_file)
            page = reader.pages[0]
            text = page.extract_text()
            cleaned_resume = cleanResume(text)
            yoe = extract_years_of_experience(text)

            input_features = word_vector.transform([cleaned_resume])
            prediction_id = model.predict(input_features)[0]
            category_name = category_mapping.get(prediction_id, "Unknown")

            category_folder = os.path.join(output_directory, category_name)

            if not os.path.exists(category_folder):
                os.makedirs(category_folder)

            target_path = os.path.join(category_folder, uploaded_file.name)
            with open(target_path, "wb") as f:
                f.write(uploaded_file.getbuffer())

            results.append({'filename': uploaded_file.name, 'category': category_name, 'yoe':
yoe})

    results_df = pd.DataFrame(results)
    return results_df

st.title("Resume Categorizer Application")
st.subheader("With Python & Machine Learning")

uploaded_files = st.file_uploader("Choose PDF files", type="pdf",
accept_multiple_files=True)
output_directory = st.text_input("Output Directory", "categorized_resumes")

if st.button("Categorize Resumes"):
    if uploaded_files and output_directory:
        results_df = categorize_resumes(uploaded_files, output_directory)
        st.write(results_df)
        results_csv = results_df.to_csv(index=False).encode('utf-8')
        st.download_button(
            label="Download results as CSV",
            data=results_csv,
            file_name='categorized_resumes.csv',
            mime='text/csv',
        )
        st.success("Resumes categorization and processing completed.")
    else:
        st.error("Please upload files and specify the output directory.")
```

4.2 IMPLEMENTATION

```
ResumeCategorizer
│
├── app.py                  # Main Streamlit application
├── requirements.txt           # Python dependencies
│
├── models/
│   ├── tfidf.pkl             # Trained TF-IDF vectorizer
│   └── model.pkl                # Trained classification model
│
├── data/
│   └── categorized_resumes/       # Output folders for categorized resumes
│
├── src/
│   ├── resume_parser.py         # Functions for extracting and cleaning resume text
│   ├── categorizer.py          # Logic for prediction and categorization
│   └── utils.py              # Utility functions (e.g., experience extraction)
│
├── static/
│   ├── css/
│   │   └── style.css            # Custom styles (if using Streamlit components)
│   └── img/
│       ├── logo.png
│       └── category_icons/       # Icons for job categories
│
├── templates/                # (Optional, if using custom HTML with Streamlit)
│   └── custom.html
│
└── README.md                  # Project overview and setup instructions
```

Install Required Packages

- Open your terminal and run:
- pip install streamlit pandas scikit-learn pypdf

Prepare Models

- Place your trained tfidf.pkl and model.pkl files inside the models/ folder.
- To train models, use your training scripts and save them with pickle.

Run the Application

- Navigate to your project folder in terminal: cd ResumeCategorizer
- Launch the Streamlit app: streamlit run app.py

Using the App

- Upload PDF resumes via the interface.
- Set or accept the default output directory for categorized resumes.
- Click **"Categorize Resumes"** to process files.
- View results in a table and download them as CSV.
- Check your categorized resumes saved in folders by job category under data/categorized_resumes/.

5. TESTING

5.1 INTRODUCTION TO TESTING

Testing is a critical phase in the software development lifecycle, aimed at ensuring that the application functions correctly, efficiently, and reliably under various scenarios. It helps verify that the system meets its intended requirements and delivers a consistent user experience, even when faced with unexpected or edge-case inputs. Through systematic testing, potential bugs and inconsistencies can be identified and resolved early, reducing the risk of failures in production and improving overall software quality.

In this project, **Resume Categorization Application**, testing played a vital role in validating the accuracy and robustness of the resume parsing, categorization, and user interface modules. The objective was to ensure that every component—from file upload and text extraction to machine learning-based categorization and result export—operated seamlessly and delivered reliable outcomes for end users.

Test cases were designed to cover the core functionalities and workflow of the application, including:

- Successful upload and processing of multiple PDF resumes
- Accurate extraction and cleaning of resume text
- Correct prediction and categorization of resumes into job roles
- Proper saving of categorized resumes into respective folders
- Display and download of results in tabular and CSV formats
- Handling of invalid or corrupted files gracefully
- User interface responsiveness and error messaging

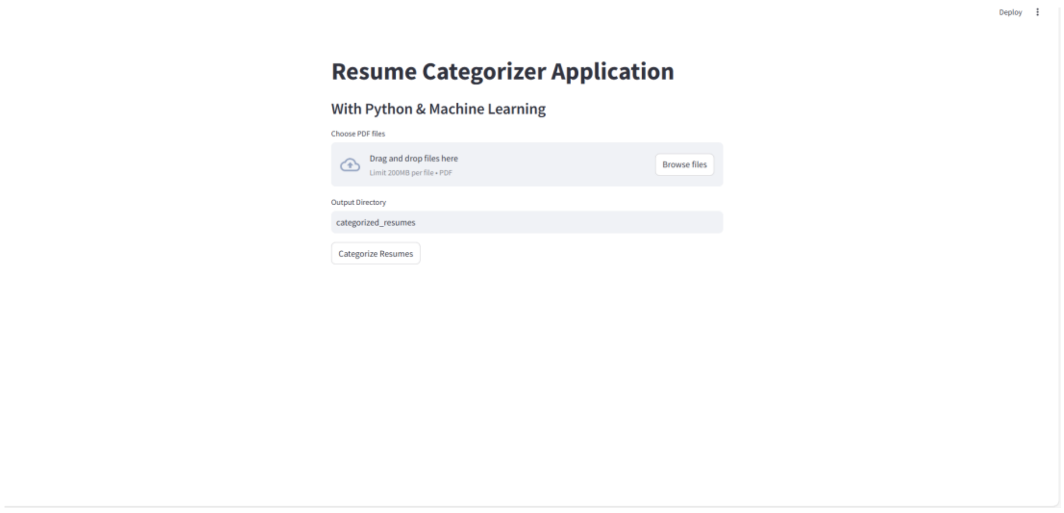
By thoroughly testing these aspects, the system's **reliability**, **usability**, and **effectiveness** in automating resume screening were ensured.

5.2 TEST CASES:

Table 5.1 Test Cases of Resume Categorizer

Test Case ID	**Test Case Name**	**Test Description**	**Expected Output**	**Actual Output**	**Remarks**
TC_01	File Upload	Upload valid PDF resumes	Files accepted and listed	Files listed	Success
TC_02	Invalid File Upload	Upload non-PDF or corrupted file	Error shown, file rejected	Error shown	Success
TC_03	Resume Parsing	Extract text from PDFs	Text extracted and cleaned	Text cleaned	Success
TC_04	Experience Extraction	Extract years of experience	Years correctly identified	Experience extracted	Success
TC_05	Categorization	Predict job category	Correct category assigned	Category correct	Success
TC_06	Folder Organization	Save in category folders	Files saved correctly	Saved properly	Success
TC_07	Results Table	Show results in app	Correct data displayed	Table shown	Success
TC_08	CSV Download	Download results as CSV	Accurate CSV downloaded	CSV correct	Success
TC_09	Output Directory	Set custom output path	Files saved to custom path	Saved in custom dir	Success
TC_10	Input Validation	Try without input or directory	Error prompts for input	Error shown	Success

6. RESULTS


Deploy

Resume Categorizer Application

With Python & Machine Learning

Choose PDF files

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

Browse files

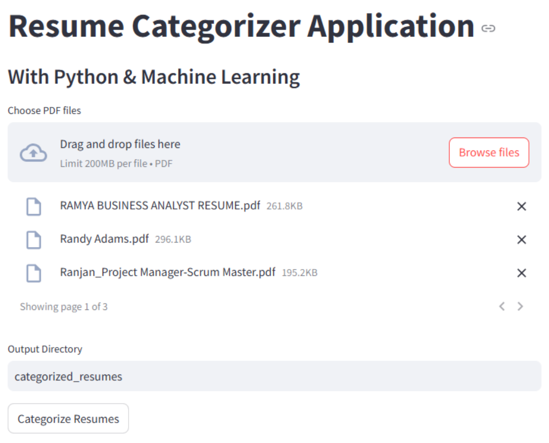
Output Directory

categorized_resumes

Categorize Resumes

Fig. 6.1 Initial page

Fig. 6.1 The initial home page of the Resume Categorizer Application appears when accessed through the local host. It introduces the purpose of the app, which is to automatically classify resumes using machine learning . Users can upload PDF resumes and specify the output directory for storing categorized files. Upon clicking the "Categorize Resumes" button, the system processes and organizes the resumes based on predicted job categories and yoe.


Resume Categorizer Application

With Python & Machine Learning

Choose PDF files

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

Browse files

RAMYA BUSINESS ANALYST RESUME.pdf 261.8KB

Randy Adams.pdf 296.1KB

Ranjan_Project Manager-Scrum Master.pdf 195.2KB

Showing page 1 of 3

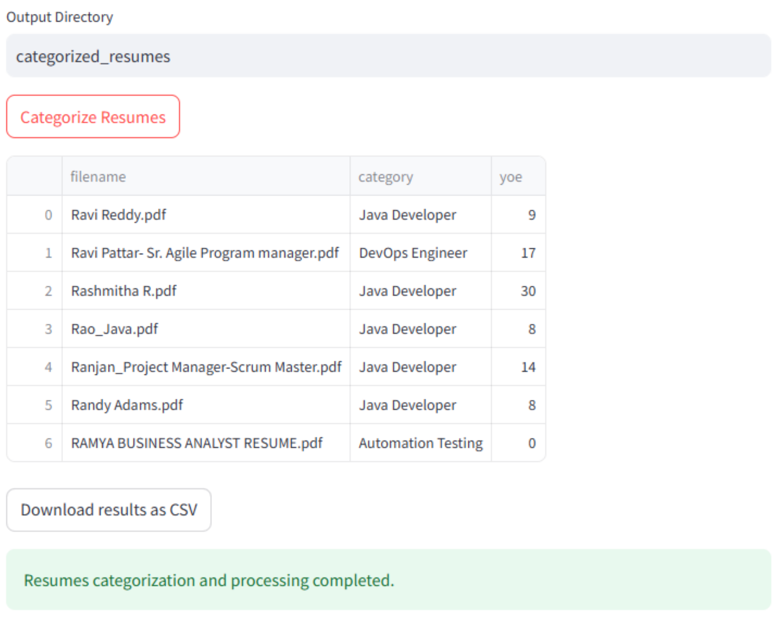
Output Directory

categorized_resumes

Categorize Resumes

Fig. 6.2 Input page

Fig. 6.2 This page shows the file upload interface of the Resume Categorizer Application. Users can drag and drop or browse to select multiple PDF resumes for classification. The selected files are listed with their names and sizes, allowing users to review or remove them before processing. The specified output directory is displayed below, and clicking "Categorize Resumes" triggers the machine learning model to classify and store the resumes accordingly.



Output Directory

categorized_resumes

Categorize Resumes

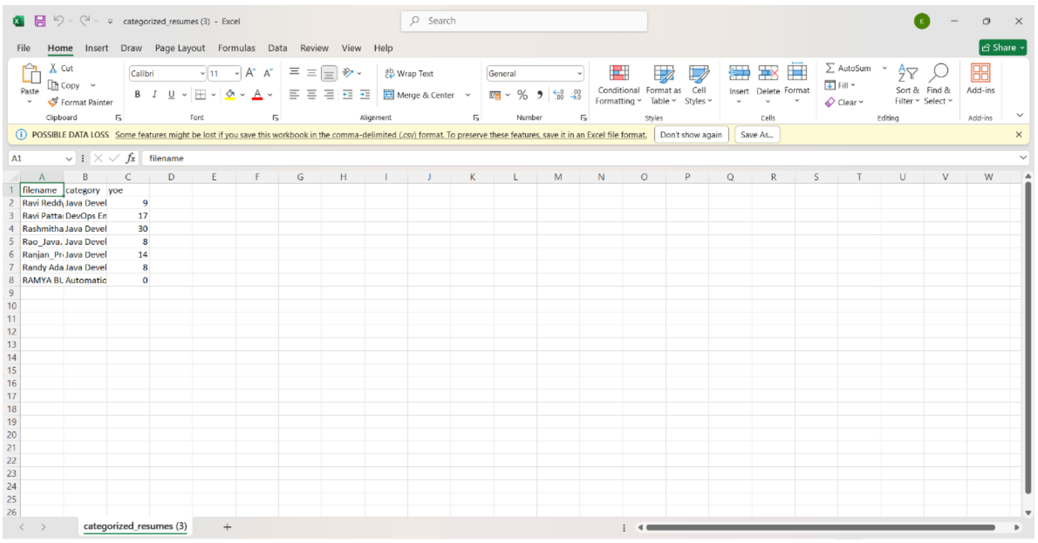
	filename	category	yoe
0	Ravi Reddy.pdf	Java Developer	9
1	Ravi Pattar- Sr. Agile Program manager.pdf	DevOps Engineer	17
2	Rashmitha R.pdf	Java Developer	30
3	Rao_Java.pdf	Java Developer	8
4	Ranjan_Project Manager-Scrum Master.pdf	Java Developer	14
5	Randy Adams.pdf	Java Developer	8
6	RAMYA BUSINESS ANALYST RESUME.pdf	Automation Testing	0

Download results as CSV

Resumes categorization and processing completed.

Fig. 6.3 Categorization page

Fig. 6.3. This page displays the final output after resumes have been processed and categorized. It presents a summary table showing each uploaded file's name, predicted job category, and extracted years of experience (yoe). A "Download results as CSV" button allows users to export the categorized data for external use. A green confirmation message indicates that the categorization process has been successfully completed.



POSSIBLE DATA LOSS Some features might be lost if you save this workbook in the comma-delimited (.csv) format. To preserve these features, save it in an Excel file format.

filename	category	yoe
Ravi Reddy	Java Devel	9
Ravi Pattai	DevOps En	17
Rashmitha	Java Devel	30
Rao_Java.	Java Devel	8
Ranjan_Pr	Java Devel	14
Randy Ada	Java Devel	8
RAMYA BL	Automatic	0

Fig. 6.4 Exported CSV File

Fig. 6.4 This page shows the exported CSV file opened in Excel, which contains the final results of the resume categorization process. Each row lists a candidate's resume filename, the predicted job category, and their years of experience (yoe). This structured format allows for easy analysis, sharing, and integration into other HR systems or dashboards. It is the downloadable output generated by the application for external usage.

7. CONCLUSION AND FUTURE ENHANCEMENTS

7.1 CONCLUSION

The Resume Categorization Application demonstrates the effective integration of machine learning and natural language processing to automate a critical aspect of the recruitment process. By leveraging TF-IDF feature extraction and a Logistic Regression classifier, the system achieves accurate and consistent categorization of resumes into relevant job roles.

The user-friendly Streamlit interface simplifies the workflow for recruiters, enabling efficient handling of multiple resumes and easy access to categorized results. Automation not only reduces manual effort and processing time but also minimizes human errors and biases.

The application's modular design allows for future enhancements, such as the inclusion of additional features or support for more file formats. Local processing ensures data privacy and security for sensitive candidate information. Comprehensive error handling and result export options further enhance usability.

Overall, the project provides a scalable and practical solution for modern human resource management, paving the way for smarter and more efficient hiring practices.

7.2 FUTURE ENHANCEMENTS

- **Support for More File Formats:** Extend the system to process resumes in formats such as DOCX, TXT, and RTF in addition to PDF.
- **Advanced Skill Extraction:** Integrate named entity recognition (NER) or deep learning models to extract specific skills, certifications, and education details from resumes.
- **Candidate Ranking:** Implement algorithms to rank candidates based on relevance to specific job descriptions or required skills.
- **Integration with Job Portals:** Enable direct integration with popular job portals or HR management systems for seamless resume import and export.
- **User Authentication:** Add user login and role-based access control to enhance security and support multi-user environments.

- **Interactive Analytics Dashboard:** Provide visual analytics and insights, such as category distribution, experience levels, and skill trends.
- **Multilingual Support:** Incorporate language detection and translation to handle resumes written in different languages.
- **Cloud Deployment:** Deploy the application on cloud platforms to enable remote access, scalability, and collaboration among HR teams.
- **Feedback Loop for Model Improvement:** Allow recruiters to provide feedback on categorization results, enabling continuous model retraining and improvement.
- **Automated Email Notifications:** Send automated notifications to candidates or recruiters based on categorization outcomes or next steps.

REFERENCES

[1] T. K. Das and A. K. Tripathy, "Exploring machine learning models for resume classification: A comparative study," *International Journal of Computer Applications Technology and Research*, vol. 9, no. 9, 2022.

[2] I. Ali, A. Shaikh, and M. Hussain, "Automated resume classification using NLP and ML techniques," *ResearchGate*, vol. 41, ISSN: 2413-7219, 2022.

[3] S. K. Singh, R. Jain, and P. Mehta, "NLP-based intelligent resume screening and job recommendation," *Revue d'Intelligence Artificielle*, vol. 37, no. 3, 2023.

[4] M. Sharma, P. Gupta, and N. Verma, "Context-aware resume-job description matching using deep learning and BERT embeddings," *International Journal of Information Technology and Computer Science*, vol. 15, no. 1, 2023.

[5] R. Kumar and S. Iyer, "Hybrid approach to resume parsing using rule-based and ML techniques for improved job matching," *IEEE Access*, vol. 12, 202

[6] **Resume Screening using Machine Learning.**
Retrieved from: https://medium.com/@jindalsaurabh13/resume-screening-using-machine-learning-aafdb7bdacfe

[7] **Scikit-learn: Machine Learning in Python.**
Retrieved from: https://scikit-learn.org/stable/

[8] **NLP Concepts – A Guide to Natural Language Processing.**
Retrieved from: https://machinelearningmastery.com/natural-language-processing/

[9] **What is Resume Parsing and How It Works – Naukri RMS.**
Retrieved from: https://www.naukrirms.com/blog/resume-parsing/

[10] **LinkedIn Talent Blog – How AI is Transforming Recruitment.**
Retrieved from: https://www.linkedin.com/business/talent/blog