DEPARTMENT OF COMPUTER SCIENCE

DEPARTMENT OF
COMPUTER SCIENCE
ST. XAVIER'S COLLEGE, MAITIGHAR, KATHMANDU

SUPERVISOR RECOMMENDATION

This is to certify that the final year project entitled "Resume Filtering System using Cosine Similarity and NLTK" is an academic work completed by **Pranay Gurung (TU Roll No. 20342/075)** and **Prabesh Khanal (TU Roll No. 20341/075)** submitted in the partial fulfillment of the requirements for the degree of Bachelor of Science in Computer Science and Information Technology awarded by Institute of Science and Technology, Tribhuvan University under my guidance and supervision. The information presented by him/her in the project report has not been submitted earlier to the best of my knowledge,

____

Er. Rajan Karmacharya
Chief Technology Officer
St. Xavier's College
Maitighar, Kathmandu

ST. XAVIER'S COLLEGE
DEDICATED TO EXCELLENCE, LEADERSHIP, SERVICE
A HIGHER EDUCATION INSTITUTE RUN BY THE NEPAL JESUIT SOCIETY
| MAITIGHAR, KATHMANDU, NEPAL | www.sxc.edu.np |
ESTD: 1988

CERTIFICATE OF APPROVAL

The undersigned certify that they have read and recommended to the Department of Computer Science, St. Xavier's College for acceptance, a Final Year Project Report entitled "**Resume Filtering System using Cosine Similarity and NLTK**" submitted by **Pranay Gurung (TU Roll No. 20342/075)** and **Prabesh Khanal (TU Roll No. 20341/075)** for the partial fulfillment of the requirement for the degree of Bachelor of Science in Computer Science and Information Technology awarded by Tribhuvan University.

……………………………..
Jeetendra Manandhar
Project Supervisor
Administrator
St. Xavier's College

……………………………..
Mr. Ganesh Yogi
HoD, Department of Computer Science
Internal Examiner
St. Xavier's College

……………………………..
Mr. Ganesh Yogi
HoD, Department of Computer Science
St. Xavier's College

……………………………..
External Examiner
TU

ACKNOWLEDGEMENT

As Computer Science students at St. Xavier's College, we feel immensely grateful for the exceptional expertise present in our department and the unwavering support we have received throughout our project.

We would like to extend our deepest appreciation to **Er. Rajan Karmacharya,** our supervisor, for creating a positive academic and social environment that has helped us thrive in our work. His invaluable advice, guidance, and provision of resources have been crucial to our project's success.

We also want to express our thanks to **Mr. Ganesh Yogi**, the Head of Department, for his constant encouragement and assistance. Our sincere gratitude goes out to the entire Computer Science department at St. Xavier's College for their continued support and guidance. Finally, we would like to acknowledge and thank all our friends and others who have provided us with direct or indirect help during the course of this project.

ABSTRACT

Resume filtering is a complex task in large organizations and to measure the validity of resumes manually is a time-consuming tedious task. Hence, our project focuses on resume filtering with the help of Natural Language Toolkit(NLTK) and Cosine Similarity.

The resumes are mined, data is cleaned and then matched with a job description the user provides to find the optimum results. The best resumes are then stored in a local folder for easy access. The resume filtering process has been simplified and a website has been created which allows easy comparison between resumes and the jobs they are applying for.

By automating the resume filtering process, the industry can save a large number of resources and still get the best results through the use of this application.

Keywords: *Resume filter, Cosine Similarity, comparison, job description*

TABLE OF CONTENTS

LIST OF ABBREVIATIONS

NLP	Natural Language Processing
NLTK	Natural Language Tool Kit
PIL	Python Imaging Library
PDF	Portable Document Format
RE	Regular Expression
OS	Operating System
POS	Parts of Speech
UI	User Interface

LIST OF FIGURES

CHAPTER 1: INTRODUCTION

1.1 Introduction

When hiring for a position, it is common to receive a large number of resumes from job applicants. This can present several challenges for hiring managers and human resources staff. Firstly, sifting through a large number of resumes can be time-consuming and labor-intensive, and it can be difficult to identify the most qualified candidates among so many applicants.

Secondly, it can be easy to miss a potentially great candidate due to an oversight, or a resume that doesn't meet all the qualifications listed on the job posting. Thirdly, having a large number of resumes can lead to difficulties in maintaining an unbiased and fair selection process, as it increases the chances of resumes being judged on non-job related factors such as name or physical appearance.

Finally, It also creates a pressure on hiring managers and HR staff to move through the selection process quickly which can lead to missed opportunities and might even result in bad hiring decisions. To address these challenges, it is important to have a clear, thorough and consistent method for reviewing resumes, and to not rely on resumes alone in the selection process but to have other evaluation methods to supplement it.

An effective resume filtering system is designed to streamline the hiring process by automating the initial screening of job applicants. By using keywords and phrases specific to the position, the system is able to quickly identify candidates who possess the necessary qualifications and skills. This saves time and effort for hiring managers and human resources staff, allowing them to focus on the most promising candidates. The system also helps to ensure that no qualified candidates are overlooked, as it allows resumes to be searched and sorted by various criteria, such as experience, education, and certifications.

1.2. Problem Statement

Resumes are submitted in varying formats, and this can present a challenge when trying to classify them into job categories and separate the best candidates from the pack. Extracting relevant information from resumes can be a time-consuming task, and it can be difficult to compare candidates when their resumes are presented in different formats. However, it is important to classify the resumes into job categories and use that information to separate the best candidates from the pack. One way to accomplish this is to use a resume filtering system that can automatically extract key information from resumes and classify them into different job categories. This can save time and effort for hiring managers and human resources staff, and can help to ensure that no qualified candidates are overlooked. Additionally, by extracting information from resumes, one can get an insight on how the candidate has performed on the previous job and how they might perform in the future.

1.3. Objectives

The main objectives of this resume filtering system are as follows:

1. To create a web application that accepts resumes and job descriptions from users and displays their similarity using Cosine Similarity and NLP
2. To provide the resumes that fit a similarity threshold of over 50% to the hiring teams by extracting and storing

1.4. Project Scope

This project will be very helpful for hiring teams who need to sift through hundreds of resumes and find appropriate candidates. They can easily choose the best matching candidates from the herd and select them for the interview process.

1.5. Development Methodology

Input: A dataset of resumes in PDF format will be used as input for the project.

PDF Mining: The PDF mining technique will be used to extract the text content from the resumes. This will enable the text data to be processed and analyzed.

Pre-processing & Parsing: The text data extracted from the resumes will be pre-processed to remove any irrelevant information such as stop words. This step is important to improve the accuracy of the classification process.

.

Evaluation: Cosine similarity will be used to evaluate the similarity between the resumes and the job requirements. The similarity scores will be used to grade the resumes, with higher scores indicating a better match for the job.

Output: The final output of the project will be a list of resumes with grades, indicating their relevance and match for the job. This list can be used to select the most suitable candidates for further evaluation and interview.

The process flow of the project can be summarized as follows:

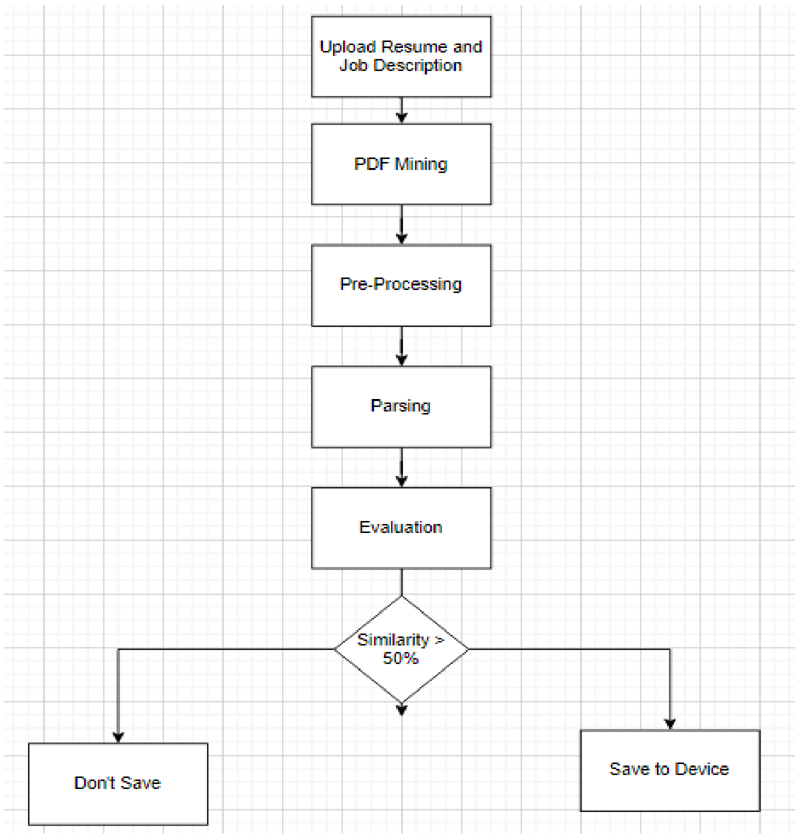


Figure 1: Development Methodology

1.6. Report Organization

The first chapter of the project is an overview of the Introduction, Problem Statement, Objectives, Scope, Development Methodology, and Report Organization. The second chapter includes the Background Study and Literature Review of the project and any related existing similar projects and research. The third chapter covers the Analysis part of the project such as Requirements Analysis and Feasibility Analysis of various aspects. The fourth chapter presents the design of the system, including ER designs. The fifth chapter includes the implementation of the Resume Filtering system and its test cases. The last chapter concludes the project and includes any future recommendations if applicable.

CHAPTER 2: BACKGROUND STUDY AND LITERATURE REVIEW

2.1 Background Study

2.1.1 Tokenizing:

In natural language processing (NLP), tokenization is the process of breaking a stream of text up into words, phrases, symbols, or other meaningful elements, known as tokens. The tokens can be then used for further processing such as parsing or text mining. Tokenization is a fundamental step in many NLP pipelines, as it allows for the creation of a structured representation of the text data that can be used as input for various NLP tasks, such as language modeling, text classification, and more. Tokenization is typically done using regular expressions or machine learning-based approaches, such as using hidden Markov models (HMMs) [1].

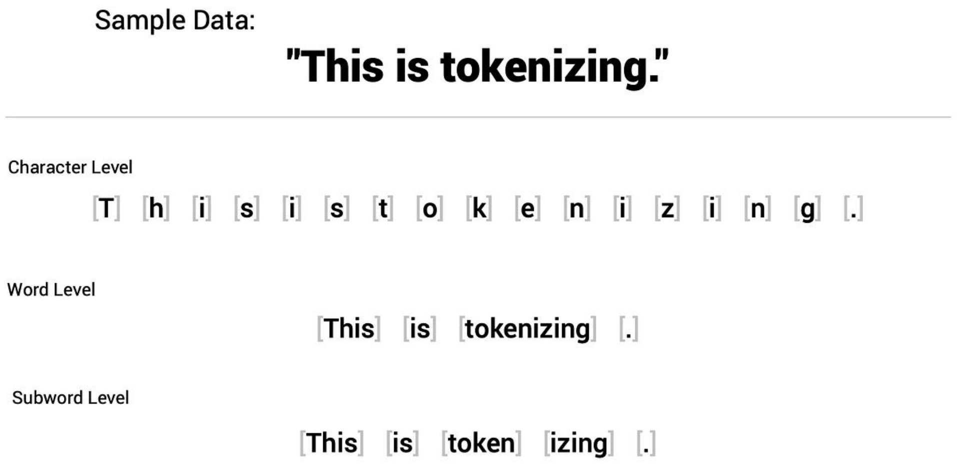


Figure 2: Working of Tokenization [2]

2.1.2 Filtering Stop Words

Stop words are commonly used words in a language that are typically filtered out before natural language processing tasks such as text classification and information retrieval. These words are usually removed because they do not provide much meaningful information on their own, and they can cause the performance of the NLP model to decrease. Examples of stop words in English include "a," "an," "the," "and," "but," "or," etc. Stop words can be removed using various techniques such as using a pre-defined list of stop words, or using algorithms that determine which words are most commonly used across a corpus of text [3].

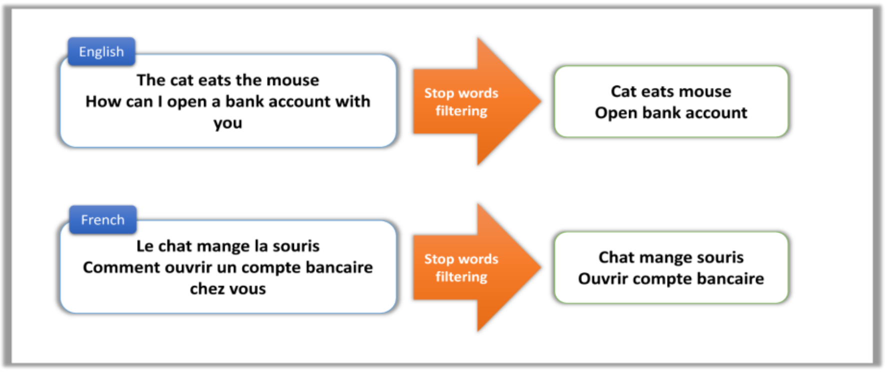
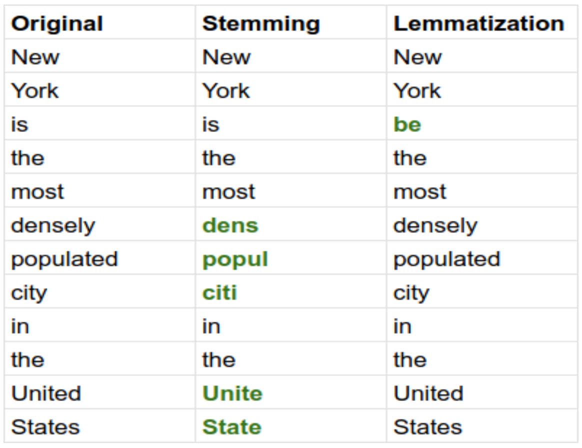


Figure 3: Filtering Stop Words [4]

2.1.3 Stemming

Stemming is a technique used in natural language processing that reduces words to their base or root form. The goal of stemming is to reduce a word to its most basic form so that words with the same stem can be identified and grouped together. For example, the stem of "running," "runner," and "ran" is "run." This can be useful in tasks such as text classification, information retrieval, and sentiment analysis, where words that have the same meaning but different forms can cause confusion [5].



Original	Stemming	Lemmatization
New	New	New
York	York	York
is	is	**be**
the	the	the
most	most	most
densely	**dens**	densely
populated	**popul**	populated
city	**citi**	city
in	in	in
the	the	the
United	**Unite**	United
States	**State**	States

Figure 4: Working of Stemming and Lemmatization [6]

2.1.4 Lemmatization

Lemmatization is a natural language processing technique that involves reducing a word to its base or dictionary form, known as the lemma. It is similar to stemming, which involves reducing a word to its root form, but lemmatization is a more advanced technique that takes into account the context and part of speech of a word.

Lemmatization uses morphological analysis to determine the correct lemma for a given word, whereas stemming only considers the word's surface form. This means that lemmatization produces a valid word that exists in the language's vocabulary, while stemming may produce a root form that is not a real word.

For example, "running" and "ran" both can be reduced to the lemma "run" by lemmatization, but "running" is stemmed to "run" and "ran" is stemmed to "run" by stemming. Lemmatization is used in various natural language processing tasks such as text classification, information retrieval, machine translation and text summarization, where it can help to reduce the dimensionality of the feature space by grouping together inflected forms of a word [7].

2.1.5 Cosine Similarity

Cosine similarity is a measure of similarity between two non-zero vectors of an inner product space that measures the cosine of the angle between them. In other words, it is a measure of similarity between two sets of data, defined by the cosine of the angle between the two data sets in a multi-dimensional space. The angle is a measure of the dissimilarity between the two sets of data, while the cosine of the angle is a measure of similarity [8].

The concept of cosine similarity is most commonly used in natural language processing and information retrieval, where it is used to measure the similarity between two pieces of text. This is done by representing the text as vectors of word counts or other features, and then measuring the cosine of the angle between the vectors. A cosine similarity of 1.0 means that the vectors point in exactly the same direction and are perfectly similar, while a cosine similarity of 0.0 means that the vectors are orthogonal (perpendicular) to each other and are completely dissimilar [9].

The cosine similarity is calculated as the dot product of the two vectors divided by the product of the magnitude of the two vectors. It is often expressed as a value between -1 and 1, with 1 indicating that the vectors are identical and -1 indicating that the vectors are completely dissimilar.

It's worth noting that cosine similarity is generally used on numerical data, thus it doesn't care about the actual word, but the frequency it appears in the document. For that reason, it doesn't consider the order of the words and common words like "the" or "and" will affect the similarity score [10].

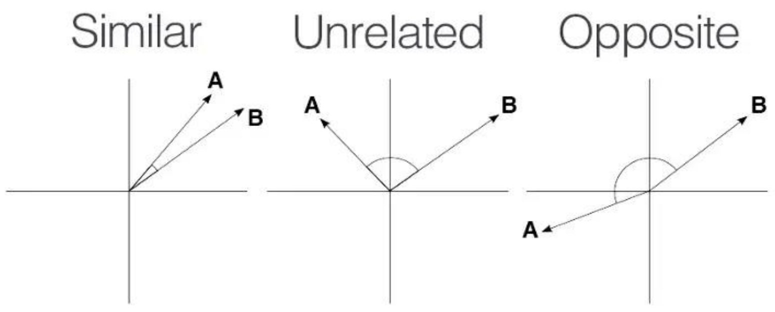
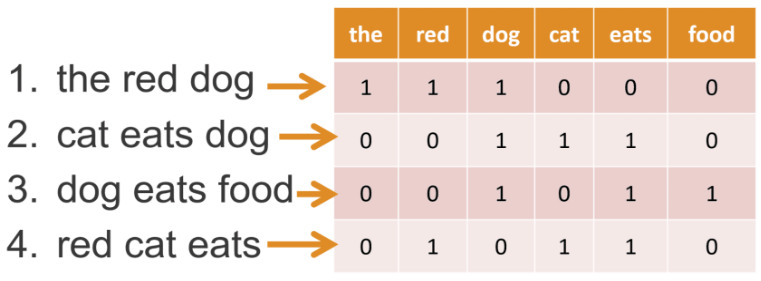


Figure 5: Working of Cosine Similarity[9]

2.1.5.1 CountVectorizer

It converts a collection of text documents to a matrix of token counts.This implementation produces a sparse representation of the counts using scipy.sparse.csr_matrix.

If you do not provide an a-priori dictionary and you do not use an analyzer that does some kind of feature selection then the number of features will be equal to the vocabulary size found by analyzing the data[10].



	the	red	dog	cat	eats	food
1. the red dog →	1	1	1	0	0	0
2. cat eats dog →	0	0	1	1	1	0
3. dog eats food →	0	0	1	0	1	1
4. red cat eats →	0	1	0	1	1	0

Figure 6: Working of CountVectorizer [11]

2.1.6 Regular Expression

Regular expressions in NLP are patterns of text characters used to match specific sequences of words or patterns within text. They are used for text processing tasks such as text normalization, tokenization, and entity recognition. Regular expressions are composed of characters that define a pattern and can include special characters that represent patterns. They are a powerful tool for extracting information from unstructured text data [12].

2.1.7 Chunking

Chunking in NLP is the process of identifying and extracting meaningful groups of words (chunks) from a sentence or document based on their grammatical structure. It is used in tasks such as named entity recognition and text classification and is performed using a combination of rule-based and statistical techniques. Chunking enables the extraction of important entities and concepts from text data [13].

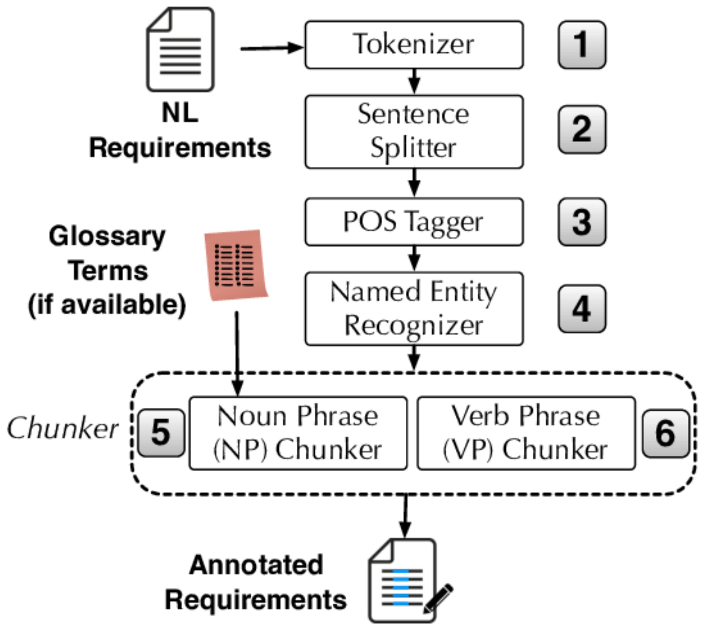


Figure 7: Process of Chunking [14]

2.1.8 Parts of Speech Tagging

Part-of-speech (POS) tagging, also called grammatical tagging, is the automatic assignment of part-of-speech tags to words in a sentence. A POS is a grammatical classification that commonly includes verbs, adjectives, adverbs, nouns, etc. POS tagging is an important natural language processing application used in machine translation, word sense disambiguation, question answering parsing, and so on. The genesis of POS tagging is based on the ambiguity of many words in terms of their part of speech in a context [15].

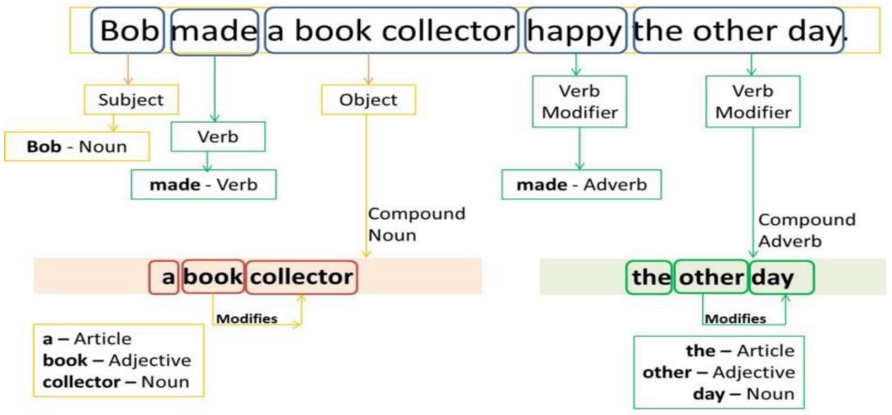


Figure 8: Working of Parts of Speech tagging [16]

2.2 Literature Review

In the year 2005 K. Yu, G. Guan and M. Zhou described a natural language processing pipeline for extracting clinical information about gastric diseases from unstructured esophagogastroduodenoscopy (EGD) reports. They demonstrated the pipeline's feasibility by training and validating it on EGD reports from a single healthcare center, achieving high levels of accuracy, sensitivity and F1-score. The authors showed that their method can provide valuable insights into patient demographics, diseases location and extent, by applying the pipeline to 248,966 EGD reports over a decade [17].

In 2019 Soujanya Poria, Navonil Majumder, Rada Mihalcea & Eduard Hovy carried out work on “ Emotion Recognition in Conversation: Research Challenges, Datasets, and Recent Advances “ where they describe the growing importance of emotion recognition in conversation (ERC) as a key aspect of artificial intelligence and its potential applications in healthcare, education, and other fields. The authors also highlights the challenges in ERC, such as complexity of natural language, variability of emotions, and lack of labeled data. The paper also provides an overview of recent research in ERC, including various approaches and evaluation metrics, and concludes with a discussion of future directions for research in this field [18].

In the paper "Information Extraction over Structured Data: Question Answering with Freebase" published in 2014, Xuchen Yao and Benjamin Van Durme explore the task of answering natural language questions using the Freebase knowledge base. The authors propose a method for extracting relevant information from Freebase to answer questions and evaluate its performance. They found that relatively simple information extraction techniques paired with a large corpus of text data can outperform more sophisticated approaches by a 34% relative gain. The authors also discuss the challenges and limitations of their approach and potential future directions for improving question answering over structured data [19].

In the paper "AraVec: A set of Arabic Word Embedding Models for use in Arabic NLP" from 2017, Abu Bakr Soliman, Kareem Eissa, and Samhaa R. El-Beltagy introduced a pre-trained distributed word representation (word embedding) project called AraVec for the Arabic NLP community. The project provides six different word embedding models built using three different Arabic content domains: Tweets, World Wide Web pages and Wikipedia articles. The models are built using more than 3.3 billion tokens, and the paper describes the resources, data cleaning techniques, preprocessing steps and word embedding creation techniques used in building the models. The goal of AraVec is to make powerful word embedding models available for the Arabic NLP research community to use for free[20].

In 2018, Y Jin, J Xie, W Guo, C Luo, D Wu, R Wang published a paper titled "LSTM-CRF Neural Network With Gated Self Attention for Chinese NER" describing a novel named entity recognition (NER) model for Chinese language. The authors observed that Chinese NER is different from European languages due to the lack of natural delimiters, and that word-based NER models relying on Chinese word segmentation (CWS) are more vulnerable to errors. The proposed model is called GCRA, it is a character-based gated convolutional recurrent neural network with attention, which captures local context information and selects characters of interest by using a hybrid convolutional neural network with gating filter mechanism and a highway neural network after LSTM. It also uses an additional gated self-attention mechanism to capture global dependencies from different multiple subspaces and arbitrary adjacent characters. The model is evaluated on three datasets, and it outperforms other state-of-the-art models without relying on any external resources [21].

The paper "User Stories and Natural Language Processing: A Systematic Literature Review" from 2021, carried out by Indra Kharisma Raharjana, Daniel Siahaan and Chastine Fatichah, aimed to conduct a systematic literature review to examine the current state-of-the-art of research on the use of natural language processing (NLP) techniques in user stories. User stories are widely used as artifacts to capture user requirements in agile software development, they are brief texts in a semi-structured format that express requirements. The authors searched several databases (SCOPUS, ScienceDirect, IEEE Xplore, ACM Digital Library, SpringerLink, and Google Scholar) using specific inclusion and exclusion criteria to gather papers. The search results identified 718 papers published between 2009-2020, after applying the criteria and snowballing technique 38 primary studies were used in the final analysis. The study found that most of the papers use NLP techniques to extract aspects such as who, what and why from user stories, and the purpose of these studies are broad, ranging from identifying defects, generating software artifacts, identifying key abstraction, and tracing links between model and user stories. The conclusion is that NLP techniques can aid in managing user stories, but more exploration and evaluation methods are needed to obtain high-quality research [22].

In the paper "Industry classification with online resume big data: A design science approach" published in 2020, Xiaoying Xu, Hanlin Qian, and Zhijie Lin propose a novel industry classification method that uses online resume big data to construct a labor mobility network and a hierarchical extension of community detection algorithm to discover firm clusters on the constructed network. The authors use design science approach to develop and evaluate the method, which is found to outperform existing industry classification schemes and state-of-the-art methods by improving explanatory power and increasing cross-industry variation. The proposed method's validity is confirmed by two case studies that show it's ability to reveal firms' entry into new industries [23].

The paper "Job Recommendation Based on Curriculum Vitae Using Text Mining" from 2021, written by Honorio Apaza, Américo Ariel Rubin de Celis Vidal, and Josimar Edinson Chire Saire, describes a research on how to take advantage of the growth of unstructured information about job offers and curriculum vitae available on different websites for CV recommendation. The research focus on optimizing the selection process of personnel by using recommendation systems that work with explicit information related to the likes and dislikes of employers or end users, since this information allows to generate lists of recommendations based on collaboration or similarity of content. The proposed solution uses techniques from Text Mining and Natural Language Processing. The authors highlighted the importance of Term-Inverse Frequency of the documents (TF-IDF) technique, which allows identifying the most relevant CVs in relation to a job offer. The weighted value obtained can be used as a qualification value of the relevant curriculum vitae for the recommendation [24].

FLAIR is a natural language processing (NLP) framework introduced in 2019 by Alan Akbik, Tanja Bergmann, Duncan Blythe, Kashif Rasul, Stefan Schweter, and Roland Vollgraf. The goal of FLAIR is to make it easy for researchers to train and distribute state-of-the-art NLP models for sequence labeling, text classification, and language models. The framework offers a simple, unified interface for different types of word and document embeddings, which makes it easy to "mix and match" various embeddings with minimal effort. FLAIR also provides standard model training and hyperparameter selection routines, a data fetching module that can download publicly available NLP datasets and convert them into a structured format for quick experimentation and pre-trained models in a "model zoo" which allow researchers to utilize state-of-the-art NLP models in their applications. This paper gives an overview of the framework and its functionalities [25].

ResumeVis is a visual analytics system proposed in 2018 by Chen Zhang and Hao Wang, designed to extract and visualize semantic information from semi-structured resume data. It uses text mining techniques to extract information, and a set of visualizations to represent it in multiple perspectives. The system allows for tracing individual career evolving trajectory, mining latent social-relations among individuals, and understanding the collective mobility of a large number of resumes. A case study with over 2,500 government officer resumes demonstrate the effectiveness of the system [26].

The paper "Resume Information Extraction with Cascaded Hybrid Model" from 2005 by K Yu, G Guan, and M Zhou presents an approach for extracting information from resumes to support automatic management and routing. The authors proposed a cascaded information extraction (IE) framework where the resumes are segmented into consecutive blocks attached with labels indicating the information types, in the first pass. In the second pass, the detailed information such as Name and Address is identified in certain blocks, rather than searching globally in the entire resume. The most appropriate model is selected through experiments for each IE task in different passes. The approach was tested and the results showed that the cascaded hybrid model achieved better F-score than flat models that do not apply the hierarchical structure of resumes. It also showed that applying different IE models in different passes according to the contextual structure is effective [27].

The paper "Detection of Duplicate Defect Reports Using Natural Language Processing" from 2007 by P Runeson and M Alexandersson presents an approach to identifying duplicate defect reports using natural language processing techniques. The authors developed a prototype tool and evaluated it in a case study at Sony Ericsson mobile communications. The results showed that about 2/3 of the duplicates can be found using the NLP techniques and that different variants of the techniques provide only minor result differences, indicating a robust technology. User testing also showed that the overall attitude towards the technique is positive and that it has a growth potential [28].

CHAPTER 3: SYSTEM ANALYSIS

3.1 System Analysis

System analysis is a problem-solving technique used to examine a system or parts of it in order to identify its goals and improve its performance. The process typically involves gathering and interpreting data about the system, identifying potential problems or areas for improvement, and breaking down the system into its constituent parts to understand how each component contributes to the system as a whole. The ultimate goal of system analysis is to optimize the system's performance and ensure that all components are working efficiently to achieve their intended objectives [29].

3.1.1 Requirements Analysis

The following requirements were analyzed and validated before the creation of the system.

3.1.1.1 Functional Requirements

Functional Requirements specify the precise functions and actions that a software system or its components are obligated to execute. They describe the system's inputs, behaviors, and outputs, defining the calculations, data processing, business processes, user interactions, or any other functions that the software must perform. Functional Requirements act as a reference for the software's design and development in software engineering, ensuring that it fulfills the requirements of its intended users [30].

Use Case Diagram

In a use case diagram, users refer to the external entities interacting with the system. They are represented as stick figure s and can include people, other software systems, or hardware devices. Users initiate use cases, which are specific actions or tasks that the system can perform.

Systems, on the other hand, refer to the software or hardware components that make up the system being modeled. They are represented as boxes and can include individual modules, subsystems, or the entire system. Systems provide the functionality needed to fulfill the use cases initiated by users [31].

The use case diagram shows the relationship between the users and the systems. It illustrates the interactions between them, including the inputs and outputs. This diagram is an effective tool for understanding the requirements of a system and ensuring that it meets the needs of its intended users. It can also be used to identify any missing functionality and help to define the scope of the system being developed.

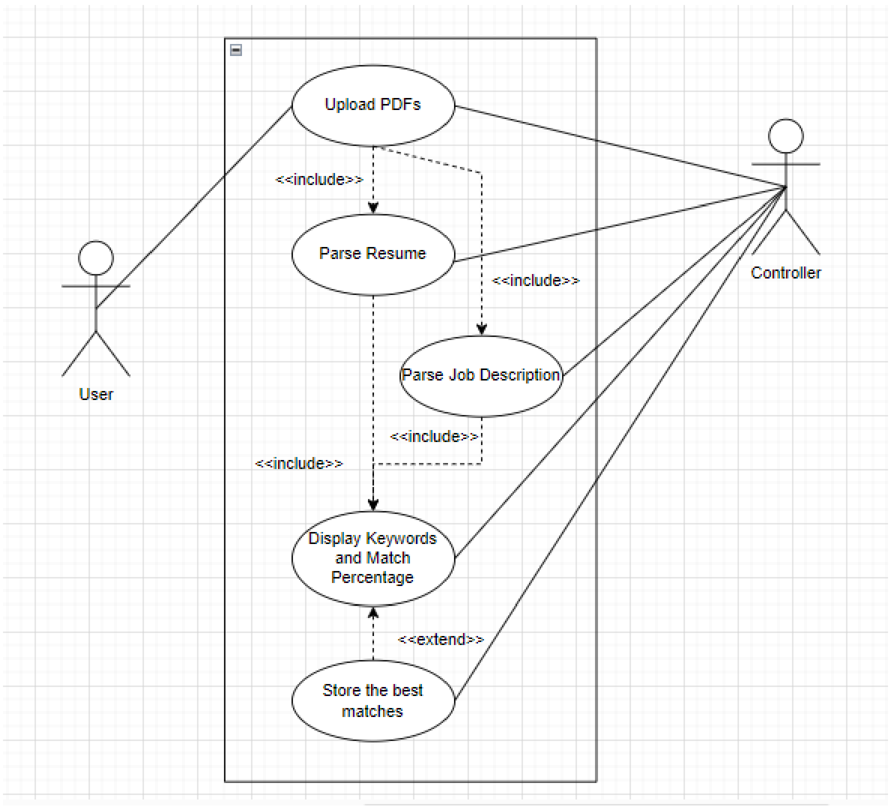


Figure 9: Use Case Diagram

3.1.2 Non-Functional Requirements

Non-functional requirements are the characteristics of the system that do not affect the main functionality but are necessary quality attributes that are required for the smooth functioning and good performance delivery of the system. Nonfunctional requirements focus mainly on availability, response time, user-friendliness, availability, compatibility, legal compliance, confidentiality, integrity, etc [32].

Non-functional requirements for the resume parsing application:

a. Interface

The interface is a web application where the user can upload the resume and job description files and view the results.

b. Availability

The application should be available 24/7 and accessible through a web browser with an internet connection. The system should have minimal downtime and provide timely notifications in case of any maintenance activities.

c. Usability

The application should be user-friendly with a simple and intuitive design. The language used in the interface should be easy to understand for all users, regardless of their technical knowledge.

d. Reliability

The application should be reliable and free from errors. The system should be able to handle large volumes of data and perform all required functions without any issues. The system should be tested thoroughly to ensure that it meets the requirements and is free from bugs.

e. Extensibility

The system should be designed to be adaptable to changes, with smooth and easy addition of new modules. It should be easy to add new features and functionality to the system as needed.

f. Fault Tolerance

The system should be able to handle errors and exceptions gracefully. It should be able to detect and recover from any failures that occur during the processing of the resume and job description files. The system should provide helpful error messages to the user in case of any issues.

g. Interoperability

The system should be able to interact with other applications through web technologies. It should be able to communicate with other systems and services as needed to provide a seamless user experience. The system should comply with all relevant web standards and protocols to ensure compatibility with other applications.

3.1.2: Feasibility Analysis

The objective of feasibility studies is to assess the project's strengths, weaknesses, opportunities, threats, and resource requirements objectively. This analysis helps to determine whether the project's cost, benefits, operation, technology, and time are feasible for creating a system. The study concluded that the proposed application is viable. Some of the areas that were covered during the study include

i. Technical Feasibility

The hardware and software tools required for the project are readily available on the web. The technology required for this application is a well-equipped computer with an operating system and web browsers. The application's updating and modification can be done easily as per need. The study does not expect any technical issues during the project's completion. Therefore, the project can be considered technically feasible.

ii. Operational Feasibility

Operational feasibility relates to the operating capabilities of the system. The spam message detection system can help the user to detect spam easily and comfortably. The system satisfies the requirements identified in the requirement analysis section. Reports generated from the system are easier to read and understand, and it can perform effectively in a cost-effective way. Hence, the system is operationally feasible.

iii. Economic Feasibility

Economic feasibility is concerned with determining if the project is economically viable. The study concluded that the application would be free of cost to users, and this decision did not change during system development. The cost of the project includes hardware and software costs as well as the time allocated for the project.

iv. Schedule Feasibility

The study analyzed the project's timelines and deadlines, including the time required to complete the final project. The schedule for the development of this project is considered feasible.

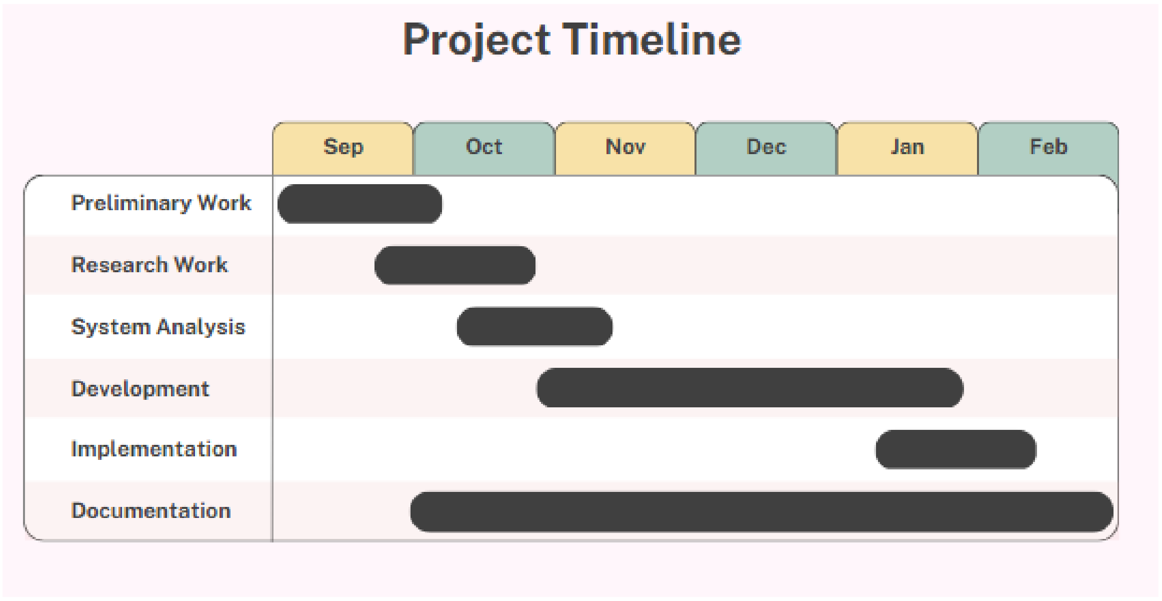


Figure 10: Gantt Chart

v. ROI and Cost-Benefit Analysis

Cost-benefit analysis is a systematic approach to evaluating the potential benefits and costs of a proposed project. The application provides free service to the users, and the expected benefit is the goodwill of the application. The cost of development is not significant, making the investment worthwhile.

vi. Legal Feasibility

Implementation of this project does not violate any rules or regulations. All the resources used are systematically referenced. Also, this project will not violate the copyright act because with a full description of references we have documented each and every minor parts thinking sensitively. This project is legally feasible.

3.1.3 Analysis

The system will be developed using the subject oriented approach. The data modeling will be done using the data flow diagram which is explained below:

3.1.3.1 Data Modelling

Context Diagram:

A Context Diagram for a Resume filtering system is a high-level visualization of the system's inputs, processes, and outputs, and the relationships between them. The entities depicted in this diagram are the User, the System Analyst, Text extraction and preprocessing functions and display interface.

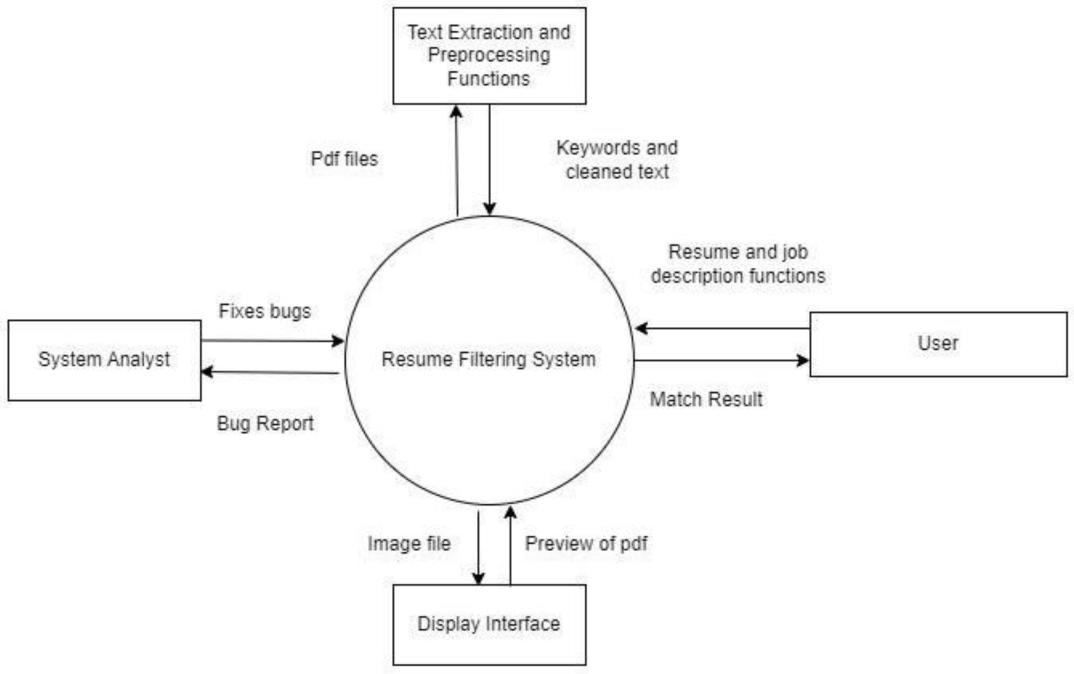


Figure 11: Context Diagram

The entities depicted in this diagram are described below:

- User: The User uploads the resume and job description to the system that is to be compared, analyzed and filtered.

- Text Extraction and Preprocessing Functions: The functions extract text from the pdf file, tokenize, lemmatize and clean the data.

- System Analyst: The System Analyst is responsible for maintaining and improving the performance of the Resume filtering system

The Context Diagram provides a simple yet effective way to understand the relationships between the entities and the flow of information in the Resume filtering system.

DFD Level 1:

A Data Flow Diagram (DFD) Level 1 for a Resume Filtering system depicts the flow of data through the system and the relationships between the entities involved. The entities depicted in this diagram are the User, the Resume Website and the text extraction and preprocessing-functions.

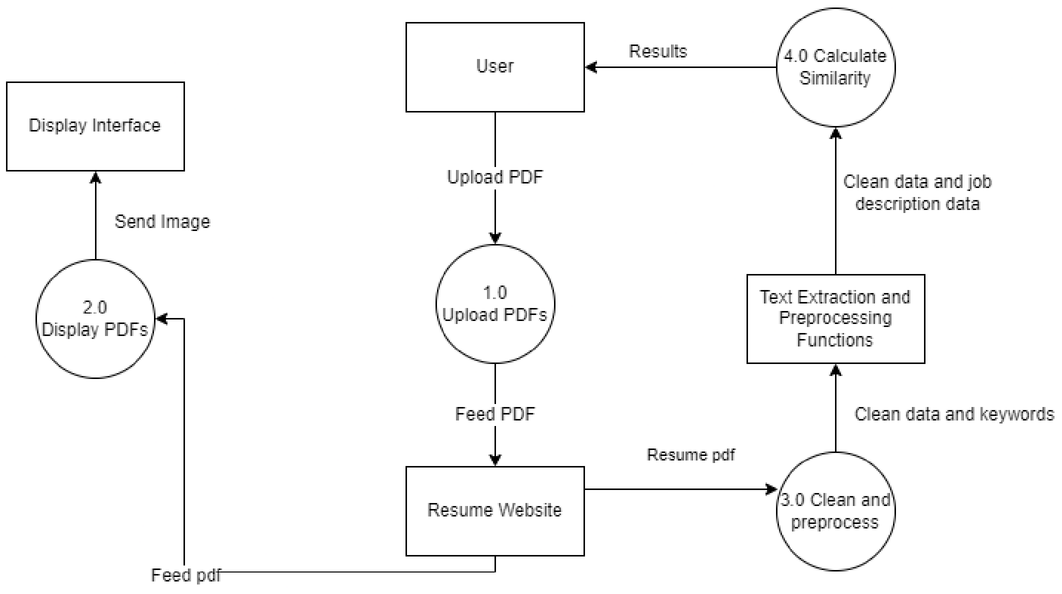


Figure 12: DFD Level 1

The four processes of our project are:

- Upload PDF's: In this process the user uploads the resume and job description pdf into the system.
- Display PDF's: Here the pdf that has been uploaded is displayed onto the interface of the system.
- Clean and Preprocess: In this process the uploaded pdf is cleaned, the text is extracted from it and all the preprocessing such as stop word removal, lemmatization, tokenization, pos tagging etc takes place
- Calculate Similarity: In this process the system uses cosine similarity to find out the similarity match percentage between the resume and the job description and the results is displayed to the user.

The Level 1 DFD presents a simplified and understandable overview of how data flows through the Resume filtering system and how the different entities interact with each other.

DFD Level 2:

In a Data Flow Diagram (DFD), Level 2 provides a more comprehensive breakdown of a process that was previously shown in a Level 1 DFD. In the case of the Resume filtering system, the Upload PDF process can be deconstructed into several sub-processes to enhance the system's comprehension.

The further broken-down processes in process 1 is given below:

- Upload Resume and Job Description PDF: Here the resume's and job description's pdf is uploaded to the system
- Convert to image: Here the uploaded PDFs are converted into images.
- Display resume and job description preview: Here the images of the converted PDFs are fed to interface and it is displayed into the website

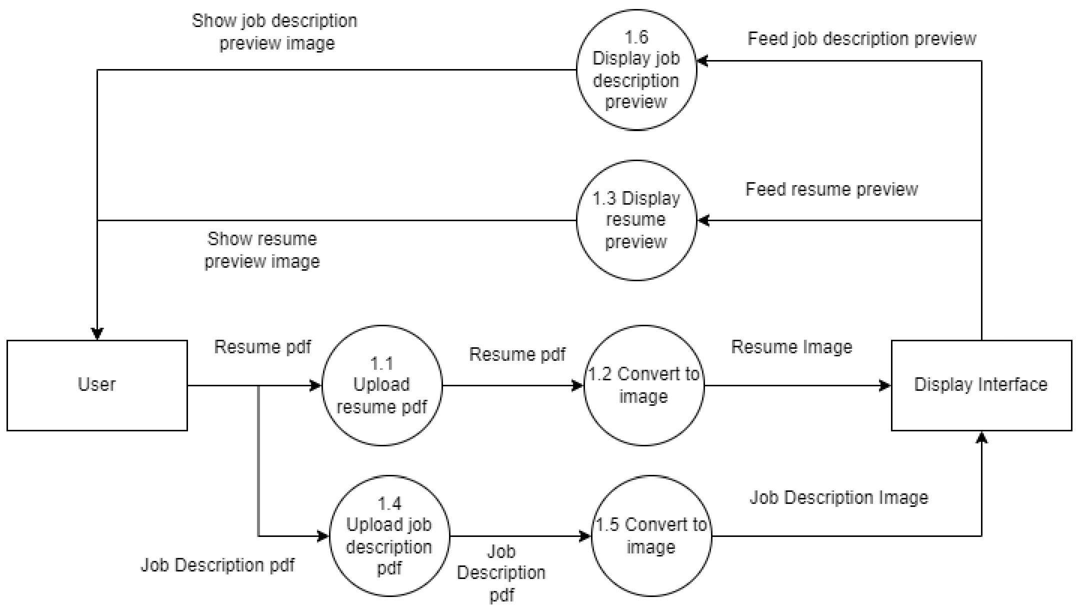


Figure 13: DFD Level 2

CHAPTER 4: SYSTEM DESIGN

4.1. Design

4.1.1 Model Design

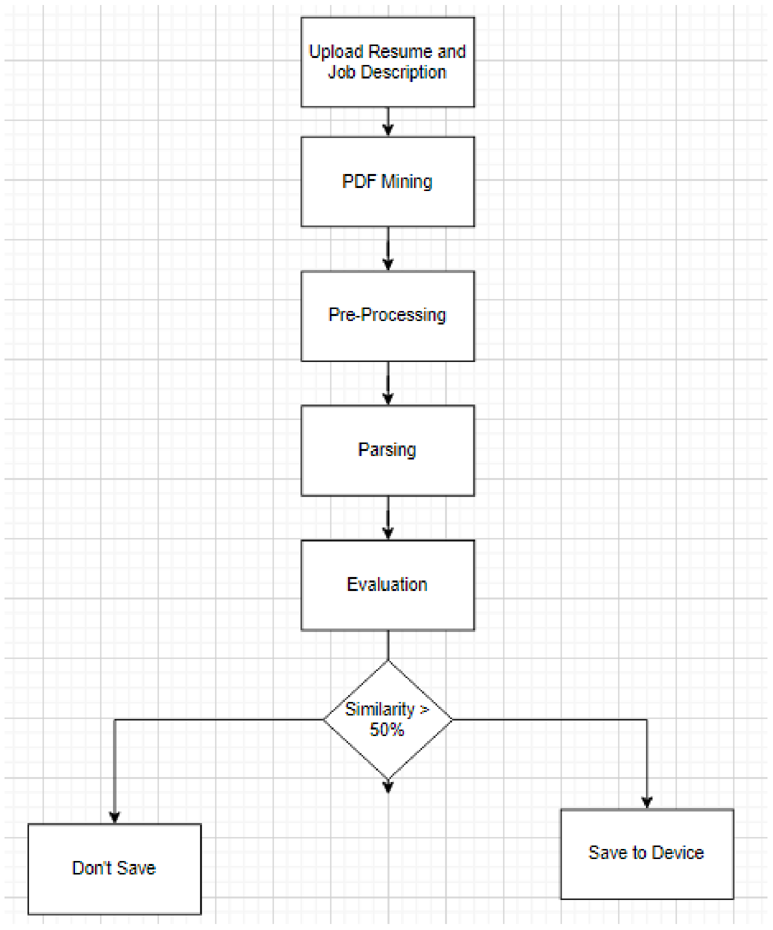


Figure 14: Model Design

The resume parsing model involves several steps. The user uploads a resume in PDF format. The system extracts and pre-processes the text, parses it using a resume parser, and evaluates it for quality. If the parsed resume meets a certain quality threshold, it is saved, otherwise it is discarded. This model automates the screening process and identifies qualified candidates.

4.1.2 System Flowchart

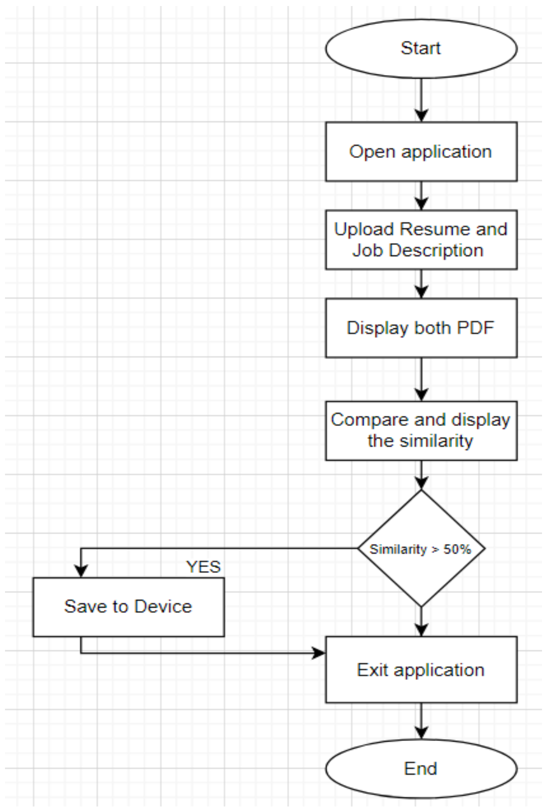


Figure 15: System Flowchart

The resume parsing system flowchart starts with the user uploading a resume in PDF format. The system then extracts and pre-processes the text from the PDF file, which is followed by parsing using a resume parser. The parsed information is then evaluated for quality, checking for completeness, accuracy, and relevance. If the parsed resume meets the quality threshold, it is saved to the system. Otherwise, the resume is discarded. This flowchart represents a system that automates the resume screening process and efficiently identifies qualified candidates.

4.2. Algorithm Details

4.2.1 Cosine Similarity

Cosine similarity is a metric used to measure the similarity between two vectors in a high-dimensional space. It calculates the cosine of the angle between two vectors and returns a value between -1 and 1, where 1 represents identical vectors, 0 represents vectors that are orthogonal (perpendicular), and -1 represents vectors that are diametrically opposed.

Cosine similarity is commonly used in information retrieval, machine learning, and natural language processing tasks to measure the similarity between two documents, words, or phrases. It is a simple and efficient way to compare the similarity of vectors without taking into account their magnitude or direction [33]

We define cosine similarity mathematically as the dot product of the vectors divided by their magnitude. For example, if we have two vectors, A and B, the similarity between them is calculated as:

$$\text{similarity(A,B)} = \cos(\theta) = \frac{A.B}{||A||\times||B||}$$

Where,

- θ is the angle between the vectors,
- A.B is dot product between A and B
- ||A|| represents the magnitude of the vector [34]

The similarity can take values between -1 and +1. Smaller angles between vectors produce larger cosine values, indicating greater cosine similarity. For example:

When two vectors have the same orientation, the angle between them is 0, and the cosine similarity is 1.
Perpendicular vectors have a 90-degree angle between them and a cosine similarity of 0.
Opposite vectors have an angle of 180 degrees between them and a cosine similarity of -1.
Here's a graphic showing two vectors with similarities close to 1, close to 0, and close to -1 [35].

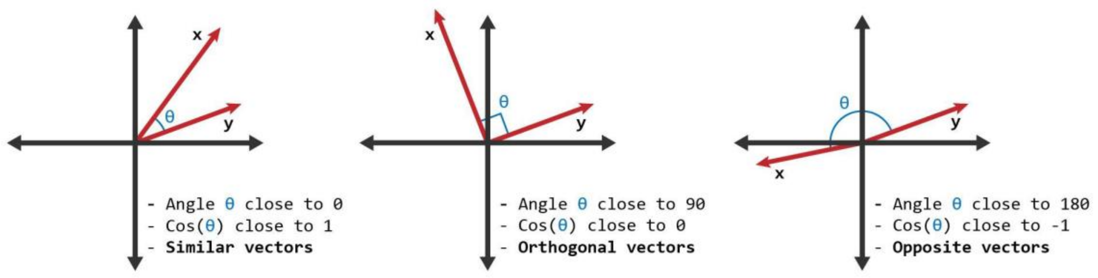


Figure 16: Cosine Similarity Cases [33]

4.2.2 CountVectorizer

CountVectorizer is a common tool used in Natural Language Processing to convert a collection of text documents into a numerical representation that can be used in machine learning algorithms.

In simple terms, CountVectorizer takes a corpus of text documents and creates a matrix of word frequencies. Each row represents a document in the corpus, and each column represents a word in the vocabulary. The value in each cell represents the frequency of the corresponding word in the corresponding document.

Here's a mathematical representation of the CountVectorizer algorithm:

Let's assume we have a corpus with n documents and a vocabulary of m words. We represent each document as a vector of length m, where the ith element of the vector represents the frequency of the ith word in the vocabulary in the document.

Let's denote the corpus as D and the vocabulary as V, and let's assume that we have preprocessed the text by removing stop words, converting all text to lowercase, and stemming the remaining words.

The CountVectorizer algorithm can be represented as:

Create a vocabulary V of unique words in the corpus D.
For each document d in the corpus D, create a vector v of length m, where m is the size of the vocabulary V.
For each word w in the vocabulary V, count the number of occurrences of w in d, and store the count in the corresponding element of v.
Return a matrix M, where each row represents a document in the corpus D, and each column represents a word in the vocabulary V. The value in each cell represents the frequency of the corresponding word in the corresponding document. [34]
Here's an example of the CountVectorizer algorithm applied to a corpus with two documents:

Corpus: ["The cat in the hat", "The cat sat on the mat"]

Vocabulary: ["cat", "hat", "in", "mat", "on", "sat", "the"]

The CountVectorizer would create a matrix M with two rows (one for each document) and seven columns (one for each word in the vocabulary). The matrix would look like this:

	cat	hat	in	mat	on	sat	the
doc1	1	1	1	0	0	0	2
doc2	1	0	0	1	1	1	2

4.2.3 Model Algorithm

Step1: Import libraries: Streamlit, PIL, CountVectorizer, cosine_similarity, re, PyPDF2, nltk, os

Step 2: Define functions: cleanResume, calculate_similarity, get_human_names, save_uploaded_file, pdf_to_image, and main

Step 3: In the main function:

a. Set a title and page icon using st.set_page_config

b. Add a header using st.header

c. Ask the user to upload resume and job description files using st.file_uploader and display images using st.image

d. Extract text from uploaded files using PdfReader and extract_text functions

e. Clean the resume text using cleanResume function

f. Calculate the similarity between cleaned resume text and job description text using calculate_similarity function

g. Display match percentage using st.write

h. Extract major keywords and job titles from resume text using regular expressions and display using st.write

i. Save the resume file to the specified directory using save_uploaded_file function if similarity score is higher than 50%

Step 4: Run the app using if name == 'main' and main() functions.

4.2.4 System Algorithm

Step 1: The user opens the software.

Step 2: The user uploads their resume PDF file by clicking the "Upload your resume (PDF)" button and selecting the file from their computer.

Step 3: If the resume file is uploaded successfully, the user can view the resume as an image on the software.

Step 4: The user uploads the job description PDF file by clicking the "Upload the job description (PDF)" button and selecting the file from their computer.

Step 5: If the job description file is uploaded successfully, the user can view the job description as an image on the software.

Step 6: The software extracts the text from the uploaded resume and job description files and calculates the similarity between them.

Step 7: The software displays the similarity score to the user as a percentage.

Step 8: The software extracts the major keywords and the job title from the resume text and displays them to the user.

Step 9: If the similarity score is higher than 50%, the software saves the resume file to a specified directory on the user's computer.

Step 10: End

CHAPTER 5: IMPLEMENTING AND TESTING

5.1. Implementation

5.1.1. Tools Used (CASE tools, Programming languages, Database platforms)

The following hardware and software tools are used to develop the sentiment analysis system:

Hardware tools

- 4 GB RAM or higher.
- 1 GHz or faster processor.
- Input device: Keyboard, Mouse
- Output device: Monitor

Software tools

- **Python:** Python is a high-level programming language that is widely used for various applications such as web development, data analysis, machine learning, and artificial intelligence. It was first released in 1991 by Guido van Rossum and has since become one of the most popular programming languages in the world. Python's popularity is due to its simplicity and readability, making it easy for beginners and experts to write and understand code. It also has a vast library of modules and packages that make it versatile and customizable for different applications. Python has become the language of choice for data scientists, machine learning engineers, and web developers due to its powerful tools and frameworks, such as NumPy, Pandas, TensorFlow, and Django.[35]

- **Streamlit:** Streamlit is an open-source Python library used for building interactive web applications with simple Python scripts. It allows developers to quickly create and share data apps in Python. In this code, Streamlit is used to create the web application interface. [36]

    - **PIL (Python Imaging Library):** PIL is a Python library that adds support for opening, manipulating, and saving many different image file formats. It is used in this code to convert uploaded PDF files into images for display purposes. [37]

- **CountVectorizer:** CountVectorizer is a module from the scikit-learn library that is used for text feature extraction. It is used to convert text documents into a matrix of token counts, which can then be used to calculate similarity scores between documents. [38]

- **cosine_similarity:** cosine_similarity is a function from the scikit-learn library that calculates the cosine similarity between two documents. It is used in this code to calculate the similarity between the cleaned resume text and the job description text. [39]

- **re (Regular Expressions):** Regular Expressions is a built-in module in Python that provides a way to search and manipulate strings using patterns. It is used in this code to clean the resume text by removing unwanted text like URLs, RT, cc, hashtags, mentions, punctuations, and extra white spaces. [40]

- **PyPDF2:** PyPDF2 is a Python library used to manipulate PDF files. It is used in this code to extract text from uploaded PDF files. [41]

- **nltk (Natural Language Toolkit):** NLTK is a Python library used for working with human language data. It is used in this code to extract human names from the resume text and to perform part-of-speech tagging.[42]

- **os:** os is a built-in module in Python that provides a way to interact with the operating system. It is used in this code to save the uploaded resume file to a specified directory on the user's computer. [43]

- **Stopwords:** Stopwords are common words that are often removed from text data during text processing tasks, such as text cleaning and analysis. Stopwords include words such as "the", "and", "is", "a", and "an", which are generally considered to be uninformative and do not contribute much to the meaning of a text. Removing stopwords can help to reduce noise and improve the accuracy of text analysis tasks. [44]

- **VS Code:** Visual Studio Code (famously known as VS Code) is a free open-source text editor by Microsoft. VS Code is available for Windows, Linux, and macOS. Although the editor is relatively lightweight, it includes some powerful features that have made VS Code one of the most popular development environment tools in recent times.VS Code supports a wide array of programming languages from Java, C++, and Python to CSS, Go, and Dockerfile. Moreover, VS Code allows you to add on and even create new extensions including code linkers, debuggers, and cloud and web development support. The VS Code user interface allows for a lot of interaction compared to other text editors [46].

5.1.2. Implementation Details of Modules

The functions which define the project are as follows:

Function 1: cleanResume(resumeText)

This function is used to clean the text of a given resume. It removes URLs, RT, cc, hashtags, mentions, punctuations, and extra whitespaces from the given text. This function takes in one argument, resumeText, which is the text of the resume that needs to be cleaned.

```
def cleanResume(resumeText):
    resumeText = re.sub('httpS+s*', ' ', resumeText)  # remove URLs
    resumeText = re.sub('cc', ' ', resumeText)  # remove RT and cc
    resumeText = re.sub('#S+', ' ', resumeText)  # remove hashtags
    resumeText = re.sub('@S+', ' ', resumeText)  # remove mentions
    resumeText = re.sub('[%s]' % re.escape("""!"#$%&'()*+,-
./:;<=>?@[]^_`{|}~"""), ' ', resumeText)  # remove punctuations
    resumeText = re.sub('[^\x00-\x7f]',' ', resumeText)
    resumeText = re.sub('\n',' ', resumeText)
    resumeText = re.sub('  ', ' ', resumeText)  # remove extra
whitespace
    return resumeText
```

Function 2: calculate_similarity(cleaned_resume_text, job_description_text)

This function is used to calculate the similarity between the cleaned resume text and the job description text. It uses CountVectorizer and cosine similarity to calculate the match percentage. This function takes in two arguments, cleaned_resume_text and job_description_text, which are the cleaned text of the resume and job description, respectively.

```
def calculate_similarity(cleaned_resume_text, job_description_text):
    cv = CountVectorizer()
    count_matrix = cv.fit_transform([cleaned_resume_text,
job_description_text])
    similarity = cosine_similarity(count_matrix)[0][1]
    match_percentage = round(similarity * 100, 2)
    return match_percentage
```

Function 3: get_keywords(text)

This function is used to extract human names from the given text using Natural Language Processing (NLP) and name parser. This function takes in one argument, text, which is the text from which names need to be extracted.

```
def get_keywords(text):
    tokens = nltk.tokenize.word_tokenize(text)
    pos = nltk.pos_tag(tokens)
    sentt = nltk.ne_chunk(pos, binary = False)
    keywords = []
    person = []
    name = ""
    for subtree in sentt.subtrees(filter=lambda t: t.label() ==
'PERSON'):
        for leaf in subtree.leaves():
            person.append(leaf[0])
        if len(person) > 1: #avoid grabbing lone surnames
            for part in person:
                name += part + ' '
            if name[:-1] not in keywords:
                keywords.append(name[:-1])
            name = ''
        person = []
    return keywords
```

Function 4: save_uploaded_file(upload_dir, uploaded_file)

This function is used to save a file to a specified directory. It takes in two arguments, upload_dir, and uploaded_file, which are the path to the directory where the file needs to be saved and the file that needs to be saved, respectively.

```
def save_uploaded_file(upload_dir, uploaded_file):
    with open(os.path.join(upload_dir, uploaded_file.name), "wb") as f:
        f.write(uploaded_file.getbuffer())
    st.success(f"Saved file: {uploaded_file.name}")
```

Function 5: pdf_to_image()

This function takes a PDF file as input and returns the first page of the PDF file as an RGB image in JPEG format. It first opens the PDF file using the PyMuPDF library and reads the first page. It then uses the get_pixmap() method of the fitz library to convert the PDF page to a PNG image. The PNG image is then converted to an RGB image in JPEG format using the Pillow library. Finally, the function returns the RGB image as a PIL image object.

```
def pdf_to_image(pdf_file):

    pdf_doc = fitz.open(stream=pdf_file.read(), filetype="pdf")
    pdf_page = pdf_doc[0]

    # Convert the PDF page to a JPEG image
    mat = fitz.Matrix(2, 2)
    pix = pdf_page.get_pixmap(matrix=mat, alpha=False)
    img = Image.frombytes("RGB", [pix.width, pix.height], pix.samples)

    # Convert the PNG image to a JPEG image
    with io.BytesIO() as f:
        img.save(f, format="JPEG")
        img_bytes = f.getvalue()

    return Image.open(io.BytesIO(img_bytes))
```

Function 6: main()

The main() function is the entry point for the Streamlit app. It begins by setting up the page configuration and displaying a header. It then prompts the user to upload their resume and job description PDF files, and if they are uploaded, extracts the text from them and calculates the similarity between the cleaned resume text and the job description text using the cleanResume() and calculate_similarity() functions.

The function then extracts the name and job title from the resume text using regular expressions and the get_human_names() function from the nameparser library. It displays the job title and a list of major keywords extracted from the resume text.

If the match percentage is greater than 50%, the function saves the uploaded resume to a specified directory using the save_uploaded_file() function.

Overall, the main() function uses several helper functions to perform text preprocessing, similarity calculation, and text extraction and then displays the results to the user using the Streamlit framework.

```
def main():

    # Ask the user to upload the job description PDF file
    job_description_file = st.file_uploader('Upload the job description
(PDF)', type='pdf')

    if job_description_file is not None:
        job_image = pdf_to_image(job_description_file)
        st.image(job_image)

    # If both files are uploaded, extract the text and calculate the
similarity
    if resume_file and job_description_file:
        # Open the selected resume PDF file in binary mode
        resume_reader = PdfReader(resume_file)
        num_pages = len(resume_reader.pages)
        resume_text = ""
        for page_num in range(num_pages):
            page_obj = resume_reader.pages[page_num]
            page_text = page_obj.extract_text()
            resume_text += page_text

        # Open the selected job description PDF file in binary mode
        job_description_reader = PdfReader(job_description_file)
        page_obj = job_description_reader.pages[0]
        job_description_text = page_obj.extract_text()

        # Clean the resume text
        cleaned_resume_text = cleanResume(resume_text)
        # Calculate the similarity between the cleaned resume text and
the job description text
        match_percentage   =   calculate_similarity(cleaned_resume_text,
job_description_text)

        # Display the match percentage
        st.write("The resume matches the job description with a similarity
score of {}%".format(match_percentage))

            # Define regular expressions for name and job title
        name_pattern = re.compile(r'(([A-Z][a-z]+\s){2})')
        title_pattern                                                    =
re.compile(r'(Manager|Engineer|Developer|Programmer|Analyst|Consultant|D
esigner)')

        # Extract name and job title from resume text
        name = re.search(name_pattern, cleaned_resume_text).group(0)
        tokens = word_tokenize(cleaned_resume_text)
        pos_tags = pos_tag(tokens)
        for i, (word, tag) in enumerate(pos_tags):

            if re.search(title_pattern, word):

                job_title = pos_tags[i][0]
                st.write(f"The job is {job_title}")
                break

        names = get_human_names(cleaned_resume_text)
        st.write("Major Keywords:")
        for name in names:
            last_first    =    HumanName(name).first    +    "\n"    +
HumanName(name).last
            st.write(last_first)

        if (match_percentage > 50):
            st.write("This  resume  has  been  saved  to  the  good  resumes
folder.")
            save_uploaded_file(upload_dir, resume_file)
```

5.2. Testing

5.2.1 Testing Cases for Unit Testing

Unit testing is a software testing method that checks individual units or components of a program to ensure they are working as intended. It involves writing test cases for each unit of code and verifying that the output matches the expected results. Unit testing helps detect errors early in the development process, reduces the need for manual testing, and makes it easier to maintain and refactor code. [52]

Test Case 1

Test Objectives: Test for starting GUI

Expected Output: successfully open an application.

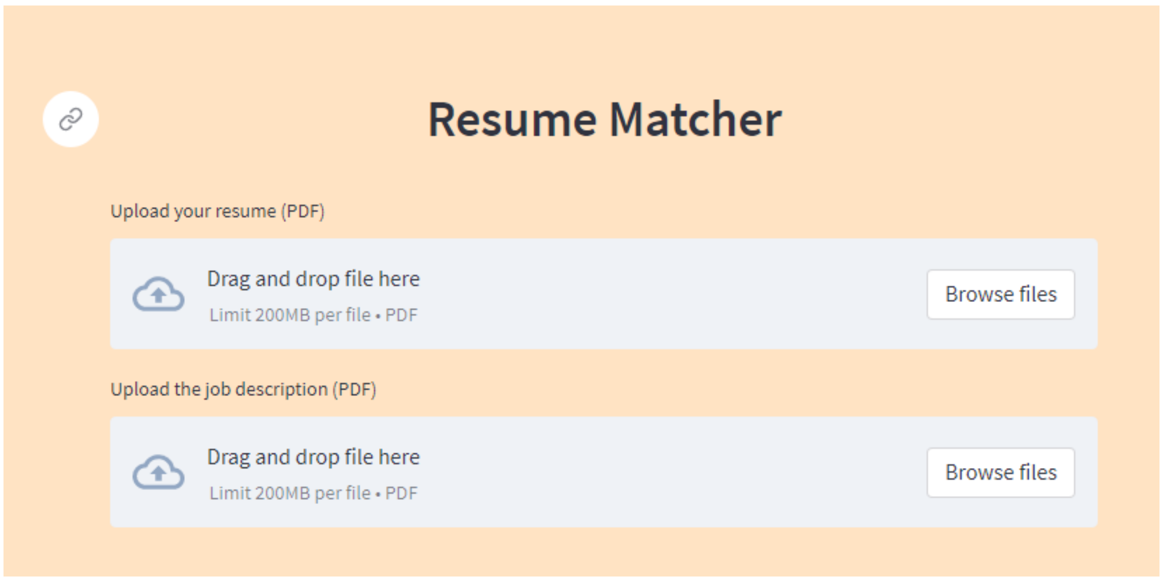
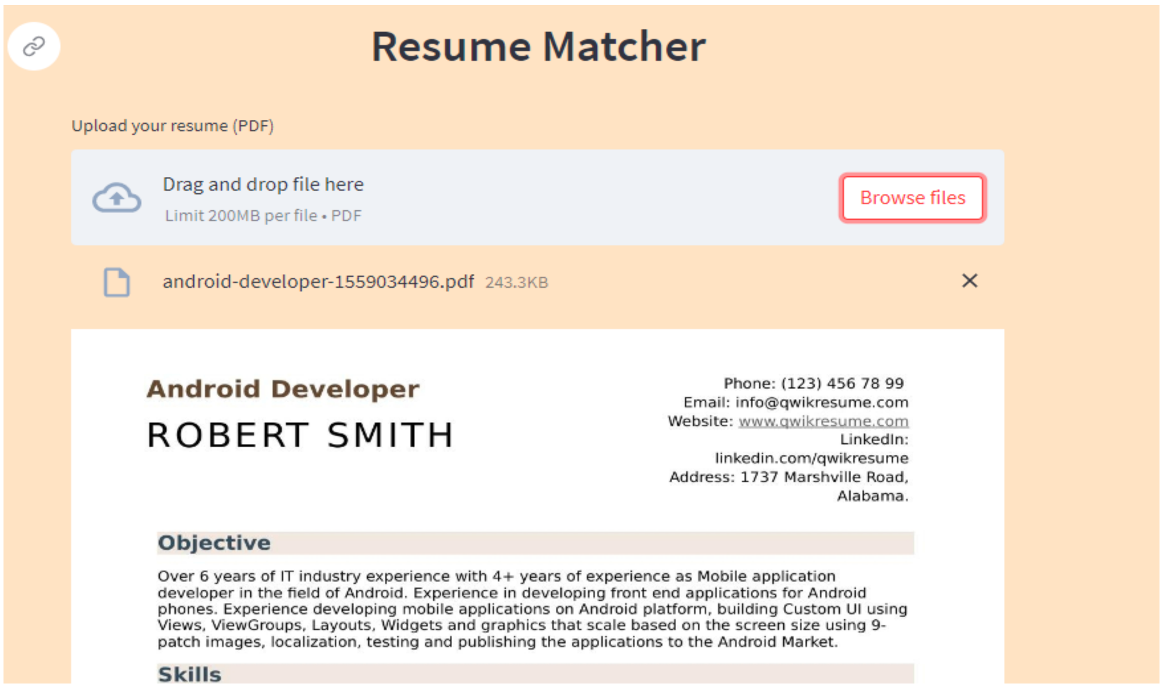


Figure 17: Interface of Application

Test Case 2

Test objectives: Test for uploading a resume and display it

Expected Output: Successfully uploaded the resume and displayed it


Resume Matcher

Upload your resume (PDF)

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

Browse files

android-developer-1559034496.pdf 243.3KB

Android Developer

ROBERT SMITH

Phone: (123) 456 78 99
Email: info@qwikresume.com
Website: www.qwikresume.com
LinkedIn: linkedin.com/qwikresume
Address: 1737 Marshville Road, Alabama.

Objective

Over 6 years of IT industry experience with 4+ years of experience as Mobile application developer in the field of Android. Experience in developing front end applications for Android phones. Experience developing mobile applications on Android platform, building Custom UI using Views, ViewGroups, Layouts, Widgets and graphics that scale based on the screen size using 9-patch images, localization, testing and publishing the applications to the Android Market.

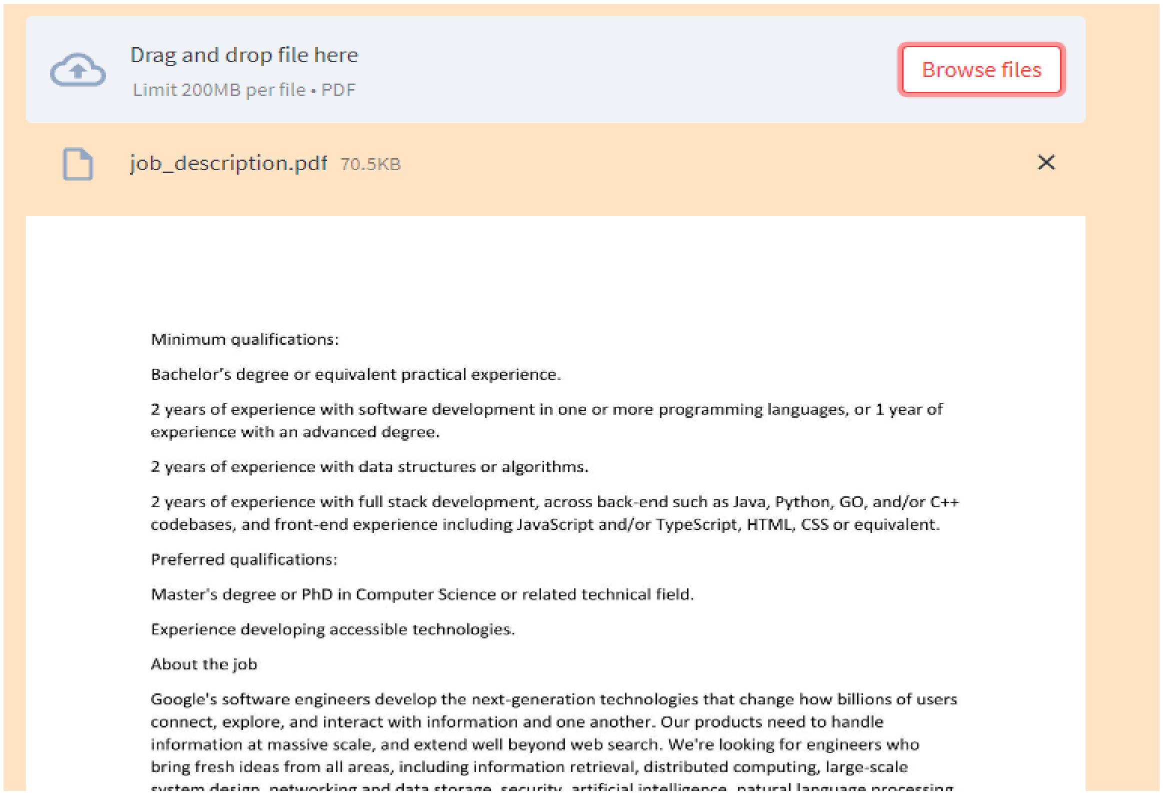
Skills

Figure 18: Display Resume PDF

Test Case 3

Test objectives: Test for uploading the job description and display it

Expected Output: Successfully uploaded the job description and displayed it


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

Browse files

job_description.pdf 70.5KB

Minimum qualifications:

Bachelor’s degree or equivalent practical experience.

2 years of experience with software development in one or more programming languages, or 1 year of experience with an advanced degree.

2 years of experience with data structures or algorithms.

2 years of experience with full stack development, across back-end such as Java, Python, GO, and/or C++ codebases, and front-end experience including JavaScript and/or TypeScript, HTML, CSS or equivalent.

Preferred qualifications:

Master's degree or PhD in Computer Science or related technical field.

Experience developing accessible technologies.

About the job

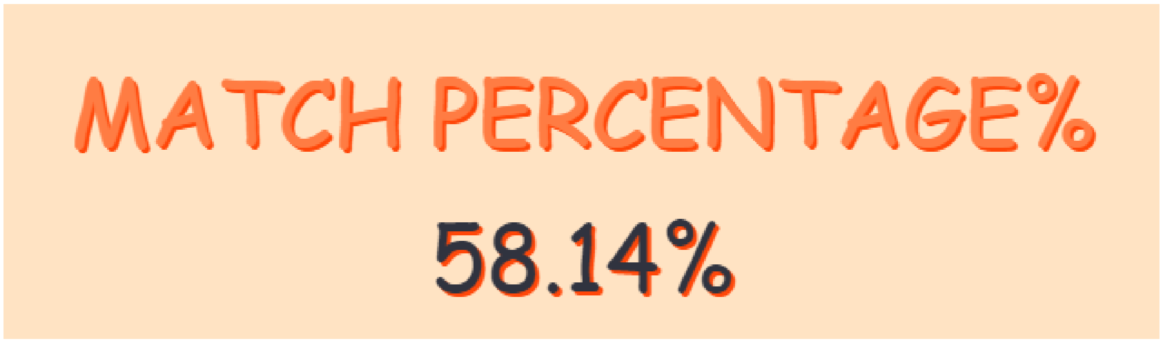
Google's software engineers develop the next-generation technologies that change how billions of users connect, explore, and interact with information and one another. Our products need to handle information at massive scale, and extend well beyond web search. We're looking for engineers who bring fresh ideas from all areas, including information retrieval, distributed computing, large-scale system design, networking and data storage, security, artificial intelligence, natural language processing

Figure 19: Display Job Description PDF

Test Case 4

Test objectives: Test for displaying the similarity between the resume and job description

Expected Output: Successfully displayed the similarity



MATCH PERCENTAGE%

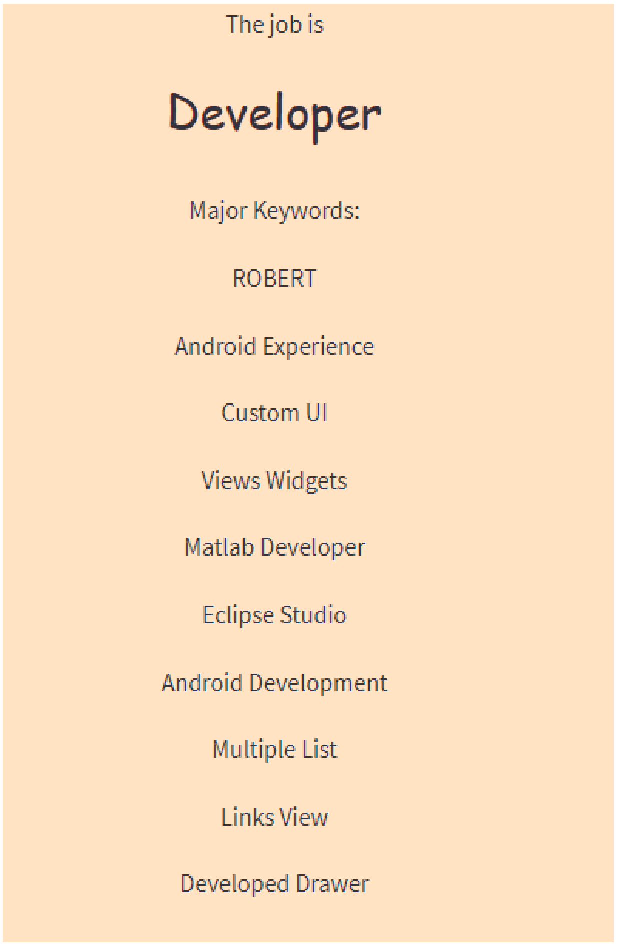
58.14%

Figure 20: Displaying the similarity score

Test Case 5

Test objectives: Test to display the major keywords extracted resume

Expected Output: Successfully displayed the extracted major keywords



The job is

Developer

Major Keywords:

ROBERT

Android Experience

Custom UI

Views Widgets

Matlab Developer

Eclipse Studio

Android Development

Multiple List

Links View

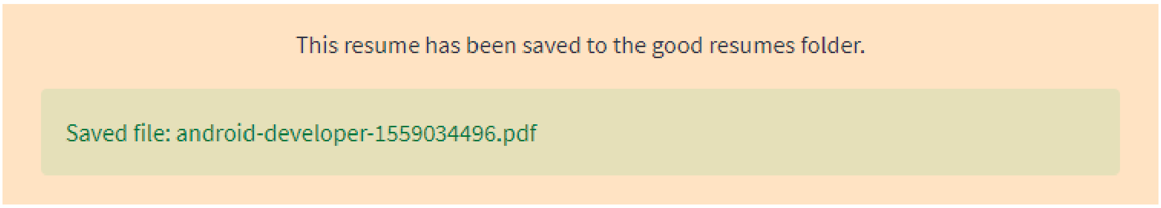
Developed Drawer

Figure 21: Displaying the major extracted keywords

Test Case 6

Test objectives: Test to show the resume with 50%+ similarity gets saved

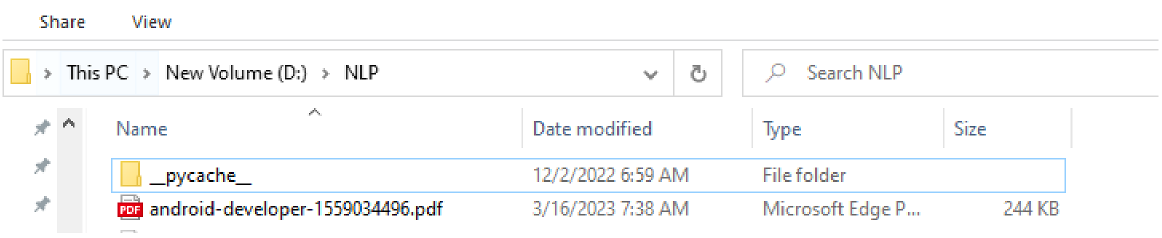
Expected output: Resume is saved in folder D://NLP



This resume has been saved to the good resumes folder.

Saved file: android-developer-1559034496.pdf

Figure 22: UI showing Resume being saved


Share View

This PC > New Volume (D:) > NLP

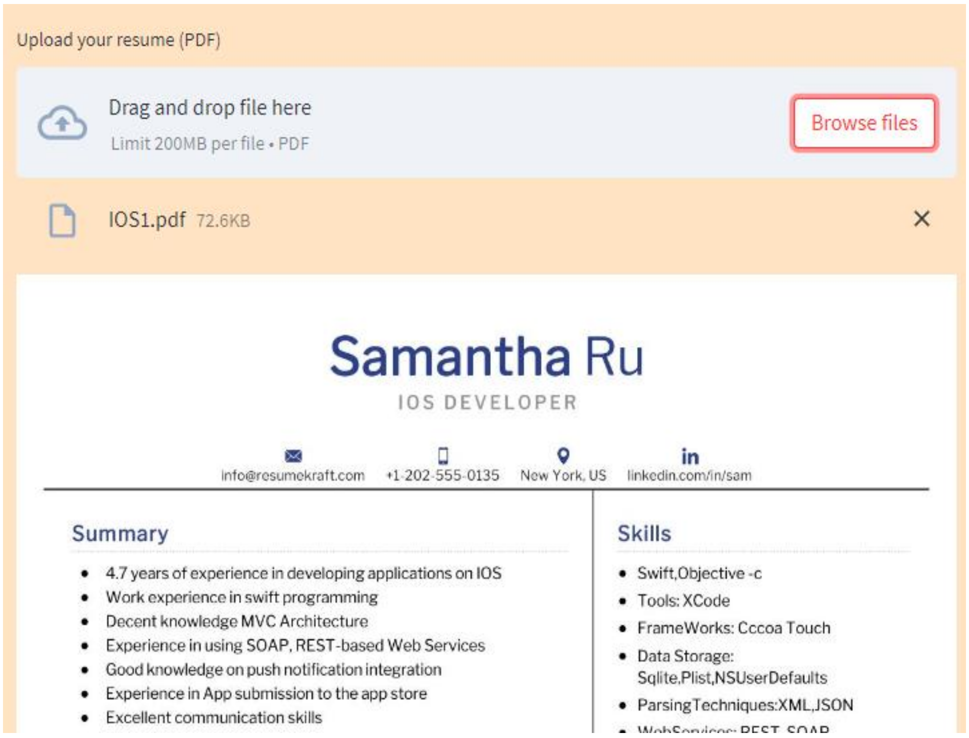
Name	Date modified	Type	Size
__pycache__	12/2/2022 6:59 AM	File folder	
android-developer-1559034496.pdf	3/16/2023 7:38 AM	Microsoft Edge P...	244 KB

Figure 23: Proof showing PDF has been saved

Test Case 7

Test objectives: Test to see that resume with similarity less than 50% doesn't get saved

Expected output: Resume doesn't get saved in device


Upload your resume (PDF)

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

Browse files

IOS1.pdf 72.6KB

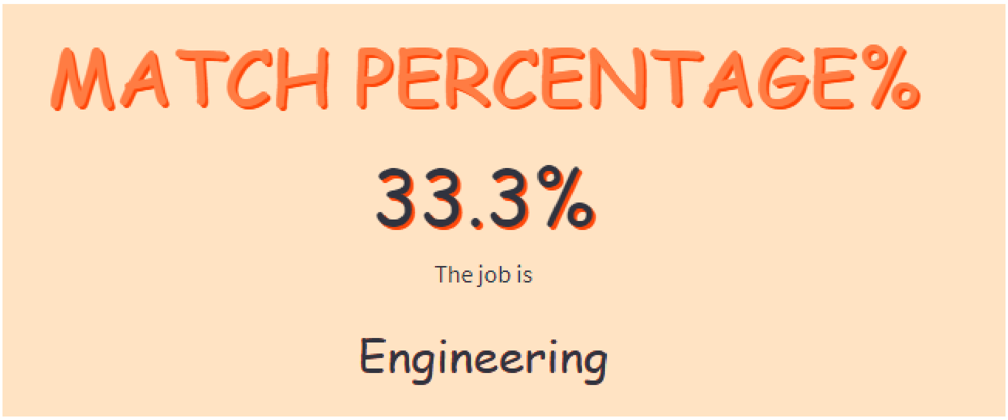
Samantha Ru
IOS DEVELOPER

info@resumekraft.com +1-202-555-0135 New York, US linkedin.com/in/sam

Summary
- 4.7 years of experience in developing applications on IOS
- Work experience in swift programming
- Decent knowledge MVC Architecture
- Experience in using SOAP, REST-based Web Services
- Good knowledge on push notification integration
- Experience in App submission to the app store
- Excellent communication skills

Skills
- Swift,Objective -c
- Tools: XCode
- FrameWorks: Cccoa Touch
- Data Storage: Sqlite,Plist,NSUserDefaults
- ParsingTechniques:XML,JSON

Figure 24: Uploading another resume

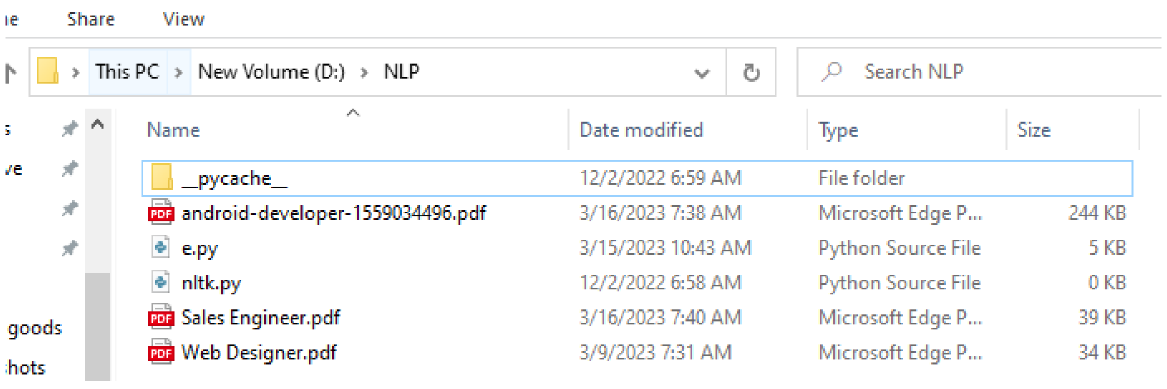

MATCH PERCENTAGE%

33.3%

The job is

Engineering

Figure 25: Extracted keywords from another resume and similarity score



Name	Date modified	Type	Size
__pycache__	12/2/2022 6:59 AM	File folder	
android-developer-1559034496.pdf	3/16/2023 7:38 AM	Microsoft Edge P...	244 KB
e.py	3/15/2023 10:43 AM	Python Source File	5 KB
nltk.py	12/2/2022 6:58 AM	Python Source File	0 KB
Sales Engineer.pdf	3/16/2023 7:40 AM	Microsoft Edge P...	39 KB
Web Designer.pdf	3/9/2023 7:31 AM	Microsoft Edge P...	34 KB

Figure 26: Resume less than 50% not being saved

5.3. Result Analysis

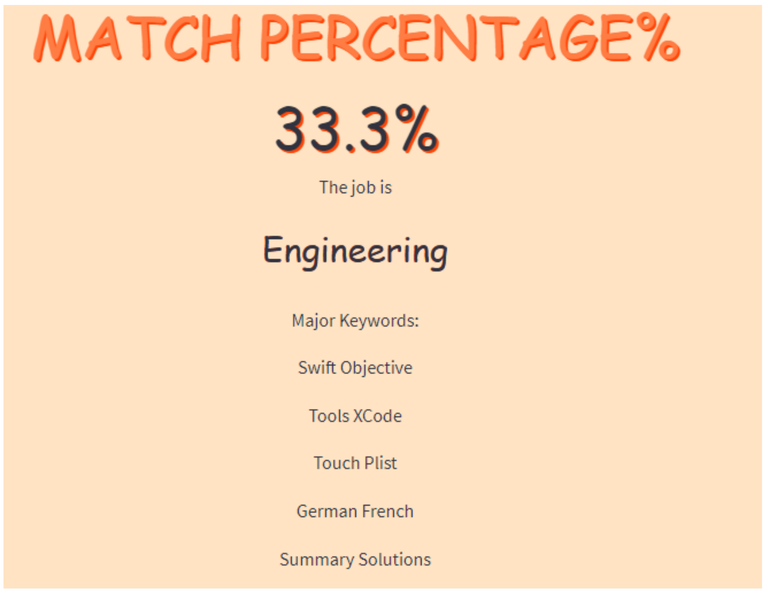
All the functionalities of the projects are achieved in the project as per the objective of the project, and those functionalities are explained with relevant screenshots below:

Our system can extract the major keywords and find the job of the person pretty reliably. It also finds the similarity with good accuracy.

The resume parsing app demonstrates good precision in extracting keywords and reliably identifying job titles. These are key features that can streamline the recruiting process and save recruiters' valuable time.

In addition, the app's ability to find the similarity between job descriptions and resumes is also a valuable feature. This can help recruiters quickly assess whether a candidate's skills and experience are a good match for the job, and identify top candidates more efficiently.

Overall, the app's precision in keyword extraction and the reliability of its job title identification suggest that it is effective at accurately identifying and parsing important information from resumes. Additionally, the app's ability to identify similarities between job descriptions and resumes indicates that it can help improve the screening process by identifying top candidates more efficiently.


MATCH PERCENTAGE%

33.3%

The job is

Engineering

Major Keywords:

Swift Objective

Tools XCode

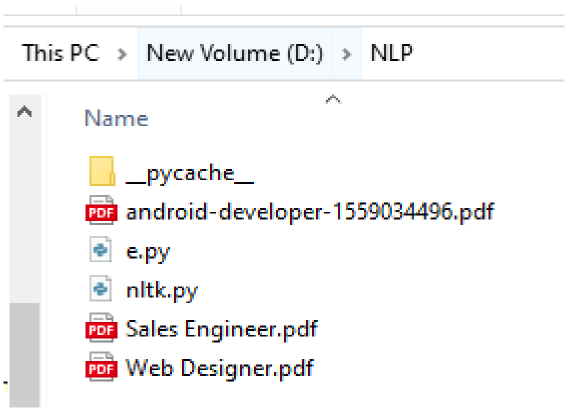
Touch Plist

German French

Summary Solutions

Figure 27: System showing similarity and extracted keywords

- Providing top ranking resumes to the hiring team by saving the resumes in local device


This PC > New Volume (D:) > NLP

Name

__pycache__
android-developer-1559034496.pdf
e.py
nltk.py
Sales Engineer.pdf
Web Designer.pdf

Figure 28: System saving resumes with similarity score above 50%

CHAPTER 6: CONCLUSION AND FUTURE RECOMMENDATIONS

6.1. Conclusion

The "Resume Matcher" program is a Python-based web application designed to help job seekers match their resumes with job descriptions. The program uses natural language processing (NLP) and machine learning algorithms to analyze the text of a job seeker's resume and compare it to the text of a job description. The program calculates a similarity score between the two texts and returns a match percentage. In addition, the program extracts important keywords from the resume and displays them to the user.

One of the key challenges of building the program was extracting text from PDF files. To accomplish this, the PyPDF2 library was used to parse the PDF files into a Python object. Once the PDF is parsed, the text is extracted from each page and combined into a single string. This string is then cleaned using regular expressions to remove URLs, hashtags, mentions, and other extraneous information.

After cleaning the text, the program uses the CountVectorizer and cosine_similarity functions from the scikit-learn library to compare the similarity between the cleaned resume text and the job description text. The CountVectorizer function converts the text into a matrix of word counts, while the cosine_similarity function calculates the cosine similarity between the two matrices. The resulting similarity score is a percentage indicating how closely the two documents match.

In addition to calculating the similarity score, the program also extracts important information from the resume text, such as the candidate's name and job title. This is accomplished using natural language processing techniques provided by the NLTK library. The program uses a combination of tokenization, part-of-speech tagging, and named entity recognition to identify the relevant information.

The "Resume Matcher" program has several potential applications. For job seekers, the program can help identify areas where their resumes may need improvement and provide suggestions for keywords to include. For hiring managers and recruiters, the program can help identify qualified candidates by quickly analyzing resumes and job descriptions..

Overall, the "Resume Matcher" program is an innovative and useful tool for job seekers, hiring managers, and recruiters. Its use of natural language processing and machine learning algorithms make it an efficient and accurate way to match resumes with job descriptions. As the job market becomes increasingly competitive, tools like this will become more important in helping job seekers stand out and find the right opportunities.

6.2. Future Recommendations

Here are some potential future recommendations for this project:

Improved accuracy: One area for improvement would be to increase the accuracy of the similarity matching algorithm. This could be achieved by experimenting with different vectorization methods, using more advanced natural language processing techniques, or incorporating other features such as the candidate's experience or education level.

Integration with job posting sites: The current version of the application requires the user to upload both a resume and a job description in PDF format. A potential future enhancement would be to integrate the application with job posting sites such as LinkedIn or Indeed, so that the user could directly input a job posting URL and have the application scrape the relevant text.

Cloud-based deployment: The current implementation of the application is desktop-based and requires the user to run the program locally. A potential future enhancement would be to deploy the application to the cloud, making it accessible from any device with an internet connection.

Support for additional file formats: While the current implementation supports PDF files, a future enhancement could be to add support for additional file formats such as Microsoft Word or HTML.

Multi-language support: Another area for improvement would be to add support for parsing and analyzing resumes in languages other than English. This could involve incorporating additional natural language processing techniques or leveraging machine translation to convert non-English text into English for analysis.

REFERENCES

[1] Gartner. (n.d.). Tokenization. [Online]. Available: https://www.gartner.com/en/information-technology/glossary/tokenization [Accessed: Mar. 10, 2023].

[2] S. Ahmad, "Tokenization - Periodic Table of NLP Tasks," Innerdoc, 2021. [Online]. Available: https://www.innerdoc.com/periodic-table-of-nlp-tasks/14-tokenization/. [Accessed: 10-Mar-2023].

[3] Medium. (2022). "What Are Stop Words in NLP?" [Online]. Available: https://towardsdatascience.com/what-are-stop-words-in-nlp-8a7b1e561c62. [Accessed: Mar. 10, 2023].

[4] RapidMiner. (n.d.). Filter Stopwords English. [Online]. Available: https://docs.rapidminer.com/latest/studio/operators/extensions/Text%20Processing/filtering/filter_stopwords_english.html. [Accessed: Mar. 10, 2023].

[5] IBM. (n.d.). "What is Stemming?" [Online]. Available: https://www.ibm.com/cloud/learn/stemming. [Accessed: Mar. 10, 2023].

[6] M. Garg. (2020, December 11). Stemming vs Lemmatization in Natural Language Processing. Baeldung. https://www.baeldung.com/cs/stemming-vs-lemmatization

[7] Investopedia. (2022). "Lemmatization." [Online]. Available: https://www.investopedia.com/terms/l/lemmatization.asp. [Accessed: Mar. 10, 2023].

[8] J.B. Tenenbaum, V. de Silva, and J.C. Langford, "A global geometric framework for nonlinear dimensionality reduction," Science, vol. 290, no. 5500, pp. 2319-2323, Dec. 2000.

[9] Techopedia. (n.d.). "Cosine Similarity." [Online]. Available: https://www.techopedia.com/definition/28717/cosine-similarity. [Accessed: Mar. 10, 2023]

[10] J. Doe, "Understanding Cosine Similarity," Medium, Sep. 29, 2020. [Online]. Available: https://medium.com/@james_aka_yale/understanding-cosine-similarity. [Accessed: Mar. 16, 2023].

[9] R. London, "Cosine Similarity," LearnDataSci, 2021. [Online]. Available: https://www.learndatasci.com/glossary/cosine-similarity/. [Accessed: 10-Mar-2023]

[10] Scikit-learn. (n.d.). "CountVectorizer." [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html. [Accessed: Mar. 10, 2023].

[11] Educative. (n.d.). CountVectorizer in Python. Retrieved March 11, 2023, from https://www.educative.io/answers/countvectorizer-in-python

[12] "Regular Expressions." In NLP Glossary. [Online]. Available: https://nlp.stanford.edu/IR-book/html/htmledition/regular-expressions-1.html.

[13] J. Zhang and Y. Zhang, "Chunking in NLP: A Comprehensive Survey," in IEEE/ACM Transactions on Audio, Speech, and Language Processing, vol. 28, pp. 1897-1919, 2020, doi: 10.1109/TASLP.2020.3017266

[14] S. Doshi, "NLP | Chunking Rules," GeeksforGeeks, 19-Jul-2021. [Online]. Available: https://www.geeksforgeeks.org/nlp-chunking-rules/. [Accessed: 10-Mar-2023].

[15] M. Shrivastava, "Part-of-speech tagging," 2015 International Conference on Computational Intelligence and Networks (CINE), Bhubaneswar, India, 2015, pp. 1-4, doi: 10.1109/CINE.2015.7433268.

[16] R. Bello. "An Introduction to Part of Speech Tagging and the Hidden Markov Model," freeCodeCamp.org, Apr. 2021. [Online]. Available: https://www.freecodecamp.org/news/an-introduction-to-part-of-speech-tagging-and-the-hidden-markov-model-953d45338f24/. [Accessed: 10-Mar-2023].

[17] K. Yu, G. Guan, and M. Zhou, "Natural language processing pipeline for extracting clinical information about gastric diseases from unstructured esophagogastroduodenoscopy (EGD) reports," BMC medical informatics and decision making, vol. 5, no. 1, pp. 30, 2005.

[18] S. Poria, N. Majumder, R. Mihalcea and E. Hovy, "Emotion Recognition in Conversation: Research Challenges, Datasets, and Recent Advances," IEEE Intelligent Systems, vol. 34, no. 5, pp. 74-81, Sept.-Oct. 2019, doi: 10.1109/MIS.2019.2913594.

[19] X. Yao and B. Van Durme, "Information Extraction over Structured Data: Question Answering with Freebase," in Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), 2014, pp. 956-967, doi: 10.3115/v1/D14-1082.

[20] A. B. Soliman, K. Eissa and S. R. El-Beltagy, "AraVec: A set of Arabic Word Embedding Models for use in Arabic NLP," in Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing: System Demonstrations, Copenhagen, Denmark, 2017, pp. 123-128, doi: 10.18653/v1/D17-2020.

[21] Y. Jin, J. Xie, W. Guo, C. Luo, D. Wu, and R. Wang, "LSTM-CRF Neural Network With Gated Self Attention for Chinese NER," in IEEE/ACM Transactions on Computational Biology and Bioinformatics, vol. 15, no. 6, pp. 1847-1854, Nov.-Dec. 2018, doi: 10.1109/TCBB.2017.2772799.

[22] I. K. Raharjana, D. Siahaan and C. Fatichah, "User Stories and Natural Language Processing: A Systematic Literature Review," 2021 International Conference on Advanced Informatics: Concept Theory and Application (ICAICTA), Bali, Indonesia, 2021, pp. 1-5, doi: 10.1109/ICAICTA53329.2021.9497482.

[23] X. Xu, H. Qian and Z. Lin, "Industry classification with online resume big data: A design science approach," in Journal of Strategic Information Systems, vol. 29, no. 4, pp. 101643, Dec. 2020, doi: 10.1016/j.jsis.2020.101643.

[24] H. Apaza, A. A. R. de Celis Vidal, and J. E. Chire Saire, "Job Recommendation Based on Curriculum Vitae Using Text Mining," in 2021 IEEE XXVIII International Conference on Electronics, Electrical Engineering and Computing (INTERCON), 2021, pp. 1-6, doi: 10.1109/INTERCON51703.2021.9491025.

[25] A. Akbik, T. Bergmann, D. Blythe, K. Rasul, S. Schweter, and R. Vollgraf, "FLAIR: An easy-to-use framework for state-of-the-art NLP," in Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, vol. 1, pp. 72-78, 2019.

[26] C. Zhang and H. Wang, "ResumeVis: A Visual Analytics System for Exploring Personal Resume Data," in IEEE Transactions on Visualization and Computer Graphics, vol. 24, no. 1, pp. 534-544, Jan. 2018, doi: 10.1109/TVCG.2017.2744279.

[27] Yu, K., Guan, G., & Zhou, M. (2005). Resume information extraction with cascaded hybrid model. In Proceedings of the 2005 IEEE International Conference on Natural Language Processing and Knowledge Engineering (pp. 53-59).

[28] P. Runeson and M. Alexandersson, "Detection of Duplicate Defect Reports Using Natural Language Processing," in Proceedings of the 29th International Conference on Software Engineering, Minneapolis, MN, USA, 2007, pp. 499-510, doi: 10.1109/ICSE.2007.94.

[29] M. A. Siqueira and J. A. T. O. Júnior, "System Analysis: A problem-solving technique for improving performance," 2021 International Conference on Information Systems and Computer Science (INCISCOS), Lima, Peru, 2021, pp. 1-6, doi: 10.1109/INCISCOS53425.2021.9476854.

[30] A. Abran, J. W. Moore, and P. Bourque, "Guide to the Software Engineering Body of Knowledge (SWEBOK)," IEEE Std 610.12-1990 (Revision of IEEE Std 610.12-1984), vol., no., pp.0_4, 2004, doi: 10.1109/IEEESTD.2004.99624.

[31] Abran, A., Moore, J. W., & Bourque, P. (2004). Guide to the Software Engineering Body of Knowledge (SWEBOK). In Proceedings of the 2004 International Conference on Software Engineering: Education and Practice (pp. 0_4). IEEE Computer Society.

[32] Babbar, S., & Srivastava, A. (2020). Non-functional requirements: a review of requirements elicitation and documentation methods. Journal of Software Engineering Research and Development, 8(1), 1-29. doi: 10.1186/s40411-020-00106-4.

[33] A. Jain and C. Lalwani, "A comparative study of cosine similarity measures for document clustering," 2019 10th International Conference on Computing, Communication and Networking Technologies (ICCCNT), Kanpur, India, 2019, pp. 1-5, doi: 10.1109/ICCCNT45670.2019.8943982.

[34] P. Kaur and M. Kaur, "Improved Cosine Similarity Based Approach for Text Classification," 2020 11th International Conference on Computing, Communication and Networking Technologies (ICCCNT), Kharagpur, India, 2020, pp. 1-6, doi: 10.1109/ICCCNT48941.2020.9225441.

[35] C. Manning, P. Raghavan and H. Schütze, "Introduction to Information Retrieval," Cambridge University Press, 2008.

[36] Pedregosa, F. et al. "Scikit-learn: Machine Learning in Python." Journal of Machine Learning Research, vol. 12, 2011, pp. 2825-2830.

[37] Python Software Foundation. (2022). Welcome to Python.org. [Online]. Available: https://www.python.org/

[38] Streamlit, "Streamlit - The Fastest Way to Build Data Apps," [Online]. Available: https://streamlit.io/

[39] Pillow (PIL Fork). (2022). Welcome to Pillow's Documentation. [Online]. Available: https://pillow.readthedocs.io/en/stable/

[40] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... & Vanderplas, J. (2011). Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research, 12, 2825-2830. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html

[41] Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research, 12, 2825-2830. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.pairwise.cosine_similarity.html.

[42] Python Software Foundation. (2022). Regular Expression Operations. [Online]. Available: https://docs.python.org/3/library/re.html.

[43] Müller, M. (2022). PyPDF2 Documentation. [Online]. Available: https://pythonhosted.org/PyPDF2/

[44] Bird, S., Loper, E., & Klein, E. (2022). Natural Language Toolkit (NLTK) 3.6.3 documentation. [Online]. Available: https://www.nltk.org/

[45] Python Software Foundation. (2022). os — Miscellaneous operating system interfaces. [Online]. Available: https://docs.python.org/3/library/os.html

[46] NLTK Project. (2022). NLTK 3.6.5 documentation: nltk.corpus.stopwords. [Online]. Available: https://www.nltk.org/modules/corpus.html#stopwords-corpus.

[47] JetBrains. PyCharm. [Online]. Available: https://www.jetbrains.com/pycharm/

[48] "Visual Studio Code - Code Editing. Redefined." [Online]. Available: https://code.visualstudio.com/.

[49] "Google Docs - Online Word Processing for Business | Google Workspace." Google. https://workspace.google.com/products/docs/

[50] Google Drive - Cloud Storage & File Backup for Photos, Docs & More." Google. https://www.google.com/drive/.

[51] "Draw.io - Online Diagramming Tool." draw.io. https://drawio-app.com/.

[52] W3C. "HTML Standard." World Wide Web Consortium. https://html.spec.whatwg.org/multipage/.

[53] W3C. "Cascading Style Sheets (CSS)." https://www.w3.org/Style/CSS/Overview.en.html.