

# **SCREENING AND RANKING RESUME USING STACKED MODEL**

Submitted in partial fulfillment of the  
requirements for the award of  
Bachelor of engineering degree in Computer Science and Engineering

By

**ABIRAAMI S (Reg. No - 391100012)**

**ABINAYA R (Reg. No - 39110010)**



**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING  
SCHOOL OF COMPUTING**

## **SATHYABAMA**

**INSTITUTE OF SCIENCE AND TECHNOLOGY  
(DEEMED TO BE UNIVERSITY)**

**Accredited with Grade "A" by NAAC | 12B Status by UGC | Approved by AICTE  
JEPPIAAR NAGAR, RAJIV GANDHISALAI,  
CHENNAI - 600119**

**APRIL - 2023**



# **SATHYABAMA**

**INSTITUTE OF SCIENCE AND TECHNOLOGY  
(DEEMED TO BE UNIVERSITY)**

Accredited with Grade "A" by NAAC | 12B Status by UGC | Approved by AICTE

[www.sathyabama.ac.in](http://www.sathyabama.ac.in)

---

## **DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

### **BONAFIDE CERTIFICATE**

This is to certify that this Project Report is the bonafide work of **ABIRAAMI.S (39110012)** and **ABINAYA.R (39110010)** who carried out the Project Phase-2 entitled "**SCREENING AND RANKING RESUME USING STACKED MODEL**" under my supervision from January 2023 to April 2023.

**Internal Guide**

**Dr. S. PRINCE MARY, M.E., Ph.D.**

**Head of the Department**

**Dr. L. LAKSHMANAN, M.E., Ph.D.**

---

**Submitted for Viva voce Examination held on 20.04.2023**

**Internal Examiner**

**External Examiner**

## DECLARATION

ii

I, **ABIRAAMI.S (Reg. No: 39110012)**, hereby declare that the Project Phase-2 Report entitled “**SCREENING AND RANKING RESUME USING STACKED MODEL**” done by me under the guidance of **Dr. S. Prince Mary, M.E., Ph.D.** is submitted in partial fulfillment of the requirements for the award of Bachelor of Engineering degree in **Computer Science and Engineering**.



**DATE:20. 04.2023**

**PLACE: Chennai**

**ABIRAAMI.S**  
**SIGNATURE OF THE CANDIDATE**

## ACKNOWLEDGEMENT

I am pleased to acknowledge my sincere thanks to **Board of Management of SATHYABAMA** for their kind encouragement in doing this project and for completing it successfully. I am grateful to them.

I convey my thanks to **Dr. T. Sasikala M.E., Ph. D, Dean**, School of Computing, **Dr. L. Lakshmanan M.E., Ph.D.**, Head of the Department of Computer Science and Engineering for providing me necessary support and details at the right time during the progressive reviews.

I would like to express my sincere and deep sense of gratitude to my Project Guide **Dr. S. Prince Mary, M.E., Ph.D.** for her valuable guidance, suggestions and constant encouragement paved way for the successful completion of my phase-1 project work.

I wish to express my thanks to all Teaching and Non-teaching staff members of the **Department of Computer Science and Engineering** who were helpful in many ways for the completion of the project.

## **ABSTRACT**

The evolving technology is creating many chances of employment for many. Nowadays to apply for any job the most essential document is a resume. A resume tells a lot about the person's achievements and the skill sets in all walks of life. The person applying for the job highlights the strong points and skill sets required for the company. Multinational organizations receive thousands of emails from such people who send their resumes for them to apply for a certain post. Now the real challenge is to know which resume is to be sorted and shortlisted according to the constraints. One method is to manually check and sort the resume. Now, this method is the most time-consuming and also can lead to a lot of errors because of human interventions. Also, humans cannot keep on working continuously. Using Natural Language Processing and Machine Learning to rank the resumes according to the given constraint, this intelligent system ranks the resume of any format according to the given constraints or the following requirement provided by the client company. We will basically take the bulk of input resume from the client company and that client company will also provide the requirement and the constraints according to which the resume should be ranked by our system.

## TABLE OF CONTENTS

Chapter No.	TITLE	Page No.
	<b>ABSTRACT</b>	v
	<b>LIST OF FIGURES</b>	viii
	<b>LIST OF ABBREVIATIONS</b>	ix
1	<b>INTRODUCTION</b>	1
2	<b>LITERATURE SURVEY</b>	4
	2.1 Inferences from Literature Survey	7
	2.2 Open problems in Existing System	7
	2.3 Scope of the project	7
3	<b>REQUIREMENTS ANALYSIS</b>	8
	3.1 Feasibility Studies	8
	3.2 System Use case	10
	3.3 Software Requirement Specification	11
4	<b>DESCRIPTION OF PROPOSED SYSTEM</b>	12
	4.1 Selected Methodology or process model	12
	4.2 Architecture / Overall Design of Proposed System	13
	4.3 Description of Software for Implementation and Testing plan of the Proposed Model/System	14
	4.4 Project Management Plan	18
5	<b>IMPLEMENTATION DETAILS</b>	19
	5.1 Algorithms	19
	5.2 Existing System	24
	5.3 Proposed System	24
6	<b>RESULTS AND DISCUSSION</b>	27
7	<b>SUMMARY</b>	29
	7.1 Conclusion	29
	7.2 Future work	30
	<b>REFERENCES</b>	31

<b>APPENDIX</b>	<b>33</b>
<b>A SOURCE CODE</b>	<b>33</b>
<b>B SCREENSHOTS</b>	<b>46</b>
<b>C RESEARCH PAPER</b>	<b>49</b>

## LIST OF FIGURES

FIGURE NO.	FIGURE NAME	PAGE NO.
3.2	USECASE DIAGRAM	10
4.2	ARCHITECTURE DIAGRAM	14
4.4	PROJECT MANAGEMENT PLAN	18
6.1	KNEIGHBOURS CLASSIFIER	27
6.2	CONFUSION MATRIX	27
6.3	WORD CLOUD	28



# **CHAPTER 1**

## **INTRODUCTION**

Day by day there are so many new researches that are being carried out by so many organizations in many fields. Right from the IT sector to the Medical fields, there are so many researches as well as progress made in day to day life. These progress in these fields are creating new employment opportunities all over the world. Now we know that the hiring process across the world is the same. That is the candidate has to make the resume first and as most of the recruitment process is based on the candidates' resume. Now most of the organizations tell the candidates to send the resume via e-mails. Now after they receive the email the next job is to sort them according to the requirement. Now usually they sort these manually but this process is time consuming.

Also there may be so many errors due to that. The efficiency of such sorting is extremely low. This may result in giving a chance to a candidate whose credentials are not up to the mark for the organization or this may also make the manual sorting miss the candidate who is extremely good for the organization. Hence a system which is intelligent enough to sort all these documents without any error and within time is needed which is exactly our aim. Machine learning for recruiting is the application of machine learning, such as the learning or problem-solving that a computer can do, to the recruitment function. This new technology is designed to streamline or automate some part of the recruiting workflow, especially repetitive, high-volume tasks. For example, software that applies machine learning to resumes to auto-screen candidates or software that conducts sentiment analysis on job descriptions to identify potentially biased language. The current recruitment process is more tedious and time consuming which forces the candidates to fill all their skill and information manually. And HR team requires more man power to scrutinize the resumes of the candidates. So that motivated to build a solution that is more flexible and automated.

Recruitment is a 200-billion-dollar business. It deals with hiring the best-fit candidates having the relevant skills for a given job profile from an immensely large pool of candidates. If a company has any job opening for a position, scores of candidates mail their resumes to the company to apply for that opening. In the hiring process, the first task for any recruiter is to screen the resumes of all the job applicants. Any company having a job opening for a particular position will have their mail inboxes bombarded with thousands of emails from aspiring job applicants every single day. Selecting the prospective candidates for that job position from a large pool of candidates for any recruiter is very tedious. It is an extremely daunting task for the recruiters of a company to manually go through thousands of resumes and select the most appropriate candidates for the job. Out of those thousands of resumes submitted to the company for the given job posting, about 75% of them do not showcase the relevant skills that are required for the job profile.

Due to this, the recruiters quite often find it really arduous to narrow down the most appropriate candidates from a large applicant pool. In recent years, there have been more than 50,000 e-recruitment sites have been developed. The developers of these online recruitment sites have used various approaches to identify the prospective candidates for a given job profile of a company. Some of these, have managed to employ classification techniques that will classify the candidate resumes into various categories for every job posting given by every company. In these approaches, every candidate's resume is tried to match with every given job posting on the recruitment site. The aim of these recruitment sites is to throw up the results to the candidate to which they are the best fit. The techniques used by these sites have resulted in high accuracy and precision, but one of the major disadvantages is the factor of time complexity. If every candidate's resume is matched with every other job posting given on the online recruitment site, the time complexity for acquiring the results is very high.

The world of Artificial Intelligence [AI] and Machine Learning [ML] has grown significantly in recent times. The availability of large amounts of data

brought about by advancements in technology which has made the internet cheap and accessible to previously inaccessible regions of the world has contributed to a great increase in the performance of the ML models in recent times. Software companies around the world exploit the advances in ML to drive automation and increase their productivity in areas that relied mostly on manual human labor. The approach discussed in this project is by using machine learning to train the dataset for a particular type of job position. It is also proposed to use section-based segmentation for data extraction using Natural language Processing (NLP).

In order to improve the time efficiency of the web application, the candidate's resume will only be matched to those job openings where they are interested in and have applied to which will, in turn, reduce the time complexity. Besides, the results of the resume matching of all the candidates who have applied for the job opening will be visible only to the recruiter of that particular company. This is done with the aim to aid the recruiters of any company from the long and tedious task of viewing and analyzing thousands of candidates' resumes. In this intelligent-based approach, they will be given the option to view the candidate's resume as well as they will get the results of the best candidates suitable for the required job position. Same disease, we use it as the score to rank the similarity among patients

## **CHAPTER 2**

### **LITERATURE SURVEY**

This problem statement has been extensively studied over the past 5 years by researchers and automotive companies in a bid to create a solution, and all their solutions vary from analyzing various patterns of distractive habits to analyzing resume.

Abdul Samad Ebrahim Yahiya [1] this resumed striving to reach a review of the tools and the necessary methodology to present a clear understanding of the association of NLP and DL for truly understanding in the training. Efficiency and execution both are improved in NLP by Part of speech tagging (POST), Morphological Analysis, Named Entity Recognition (NER), and Semantic Role Labeling (SRL), Syntactic Parsing, and Conference resolution. Artificial Neural Networks (ANN), Recurrent Neural Network (RNN), Convolution Neural Networks (CNN), dealings among Dense Vector (DV), Windows Approach (WA), and Multitask learning (MTL) as a characteristic of Deep Learning.

Anushka Sharma[2] , the research work in [7] focuses on extracting data from resumes and performing the required analysis on the data to convert it into useful information for the recruiters. Thus, the Resume Parser would help the recruiters to select the best relevant candidates in a minimal amount of time, consequently saving their time and effort.

Elizabeth George [3] suggested that, the proposed system, JARO accelerates the interview process towards an unbiased decision-making process by proposing a chatbot that would conduct interviews by analyzing the candidate's Curriculum Vitae (CV), based on which, it then prepares a set of questions to be asked to the candidate. The system will consist of features like resume analysis and automatic interview processes.

Garima Bhardwaj [4] The study work in [8] was conducted among 115 HR professionals at various IT sectors in Delhi/NCR region. A multiple regression method was used to test the hypothesis and confirmed a positive relationship between these two factors establishing about the increased use of AI at work results in better HR functional performance. However, AI has a significant relationship with innovativeness and also with the ease of use which reflects AI affects HR with innovations and ease of use.

Huaizheng Zhang [5], the author in [8] developed a method for automatic RQA. Since there is also no public dataset for model training and evaluation, we build a dataset for RQA by collecting around 10K resumes, which are provided by a private resume management company. By investigating the dataset, we identify some factors or features that could be useful to discriminate good resumes from bad ones, e.g., the consistency between different parts of a resume.

Muath Sabha [6] the research work of author in [5] presented a hybrid approach that employs conceptual-based classification of resumes and job postings and automatically ranks candidate resumes (that fall under each category) to their corresponding job offers. In this context, the exploit an integrated knowledge base for carrying out the classification task and experimentally demonstrate - using a real-world recruitment dataset- achieving promising precision results compared to conventional machine learning-based resume classification approaches

Mujtaba [7] presents an overview of fairness definitions, methods, and tools as they relate to recruitment and establishes ethical considerations in the use of machine learning in the hiring space. Considering Deep Learning (DL) method recognize artificial Neural Network (NN) to nonlinear process, NLP tools become increasingly accurate and efficient that begins a debacle. Multi-Layer Neural Network obtaining the importance of the NLP for its capability including standard speed and resolute output. Hierarchical designs of data operate recurring processing layers to learn and with this arrangement of DL methods manage several practices.

Pradeep Kumar Roy [8] suggested that an automated way of “Resume Classification and Matching” could really ease the tedious process of fair screening and shortlisting; it would certainly expedite the candidate selection and decision making process. This system could work with a large number of resumes for first classifying the right categories using different classifier, once classification has been done then as per the job description, top candidates could be ranked using Content-based Recommendation, using cosine similarity and by using k-NN to identify the CVs that are nearest to the provided job description.

Sanjay Revanna [9] proposed the top applicants might be rated using content-based suggestion, which uses cosine similarity to find the curriculum vitae that are the most comparable to the job description supplied and KNN algorithm is used to pick and rank Curriculum Vitaes (CV) based on job descriptions in huge quantities.

The work of Sharadadevi Kaganurmath [10] created an automated machine learning-based algorithm that recommends acceptable applicant resumes to HR based on the job description provided. The suggested methodology had two stages: first, it classified resumes into various groups. Second, it suggests resumes based on their resemblance to the job description. If an industry produces a high number of resumes, the proposed approach can be used to create an industry-specific model.

## **2.1 INFERENCES FROM THE LITERATURE SURVEY**

Resume Screening is a very important step in the hiring process. Even today, many companies are following the traditional resume screening process. Even though many efficient systems were proposed to perform resume screening operations, due to some problems; the systems are not efficient as expected. Resume Screening using Machine Learning approaches has some issues like a time consuming, not being user-friendly, etc. Resume Screening using Artificial Intelligence approaches has some issues with input/output format, single resume acceptance for a time (which leads to a lot of time to screen all the resumes), etc.

## **2.2 OPEN PROBLEMS IN EXISTING SYSTEM**

The systems has models that don't have any way to improve themselves over the time, the models will be trained only once. The models used Machine learning algorithms which have a tend to plateau in performance when run over a large dataset. The application can only be used effectively by the programmer. Each code block is interdependent; each change requires disrupting the entire code, which may disrupt the flow.

## **2.3 SCOPE OF THE PROJECT**

Resume screening is the process of quickly reviewing a resume while looking for keywords and characteristics. It saves recruiters time during the hiring process and leads to faster onboarding of new team members. Whether you screen resumes manually or with software, using the best techniques can help you identify the most suitable candidates. it is the process of quickly reviewing a resume while looking for keywords and characteristics. It saves recruiters time during the hiring process and leads to faster onboarding of new team members. Whether you screen resumes manually or with software, using the best techniques can help you identify the most suitable candidates.

## **CHAPTER 3**

### **REQUIREMENTS ANALYSIS**

Software requirement means requirement that is needed by software to increase quality of software product. These requirements are generally a type of expectation of user from software product that is important and need to be fulfilled by software. Analysis means to examine something in an organized and specific manner to know complete details about it. Therefore, Software requirement analysis simply means complete study, analyzing, describing software requirements so that requirements that are genuine and needed can be fulfilled to solve problem. There are several activities involved in analyzing Software requirements.

#### **3.1 FEASIBILITY STUDIES**

Recruitment is a 200-billion-dollar business. It deals with hiring the best-fit candidates having the relevant skills for a given job profile from an immensely large pool of candidates. If a company has any job opening for a position, scores of candidates mail their resumes to the company to apply for that opening. In the hiring process, the first task for any recruiter is to screen the resumes of all the job applicants. Any company having a job opening for a particular position will have their mail inboxes bombarded with thousands of emails from aspiring job applicants every single day. Selecting the prospective candidates for that job position from a large pool of candidates for any recruiter is very tedious. It is an extremely daunting task for the recruiters of a company to manually go through thousands of resumes and select the most appropriate candidates for the job. Out of those thousands of resumes submitted to the company for the given job posting, about 75% of them do not showcase the relevant skills that are required for the job profile.

Due to this, the recruiters quite often find it really arduous to narrow down the most appropriate candidates from a large applicant pool. In recent years, there have been more than 50,000 e-recruitment sites have been developed. The

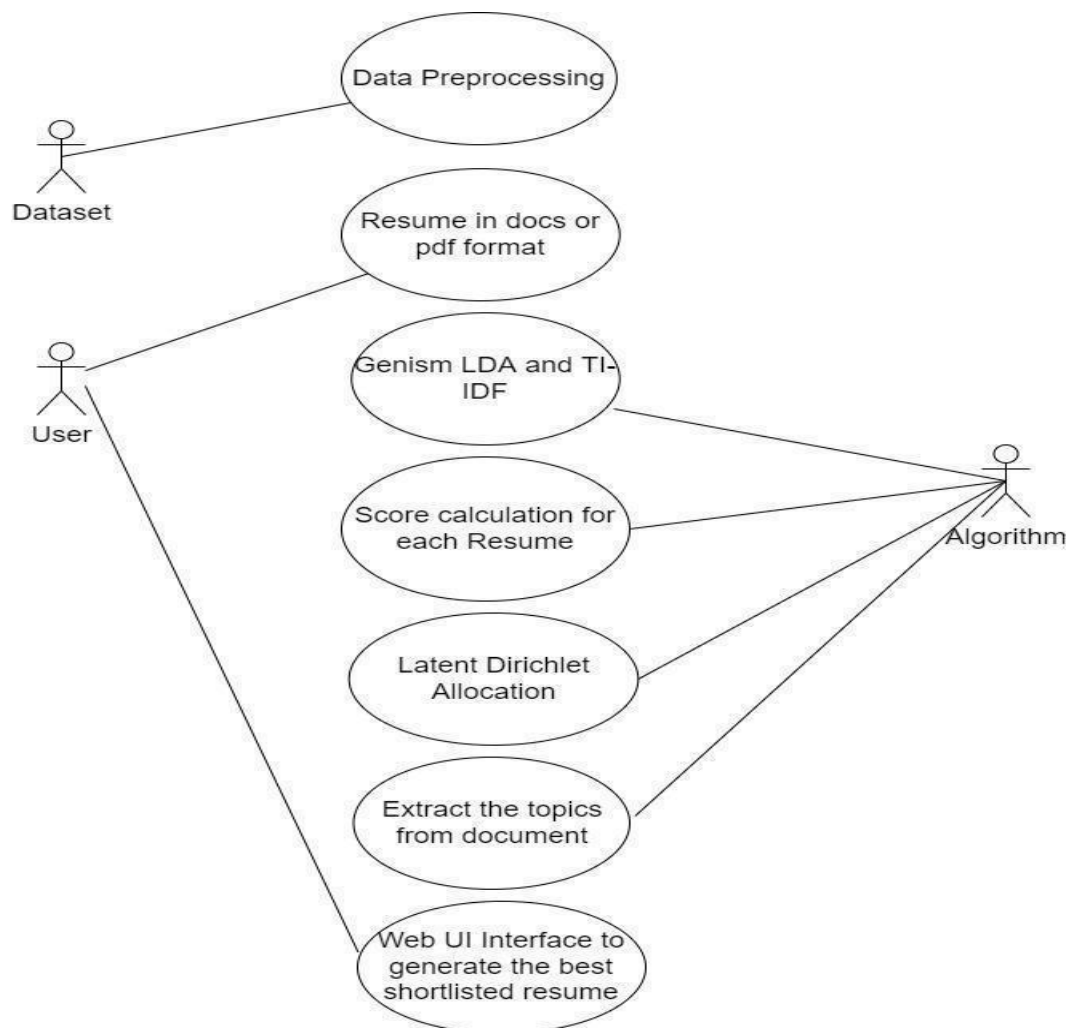


developers of these online recruitment sites have used various approaches to identify the prospective candidates for a given job profile of a company. Some of these, have managed to employ classification techniques that will classify the candidate resumes into various categories for every job posting given by every company. In these approaches, every candidate's resume is tried to match with every given job posting on the recruitment site. The aim of these recruitment sites is to throw up the results to the candidate to which they are the best fit. The techniques used by these sites have resulted in high accuracy and precision, but one of the major disadvantages is the factor of time complexity. If every candidate's resume is matched with every other job posting given on the online recruitment site, the time complexity for acquiring the results is very high.

The world of Artificial Intelligence [AI] and Machine Learning [ML] has grown significantly in recent times. The availability of large amounts of data brought about by advancements in technology which has made the internet cheap and accessible to previously inaccessible regions of the world has contributed to a great increase in the performance of the ML models in recent times. Software companies around the world exploit the advances in ML to drive automation and increase their productivity in areas that relied mostly on manual human labor. The approach discussed in this project is by using machine learning to train the dataset for a particular type of job position. It is also proposed to use section-based segmentation for data extraction using Natural language Processing (NLP). In order to improve the time efficiency of the web application, the candidate's resume will only be matched to those job openings where they are interested in and have applied to which will, in turn, reduce the time complexity. Besides, the results of the resume matching of all the candidates who have applied for the job opening will be visible only to the recruiter of that particular company. This is done with the aim to aid the recruiters of any company from the long and tedious task of viewing and analyzing thousands of candidates' resumes. In this intelligent-based approach, they will be given the option to view the candidate's resume as well as they will get the results of the best candidates suitable for the required job position. Same disease, we use it as the score to rank the similarity among patients.

### 3.2 SYSTEM USE CASE

This Use Case Diagram is a graphic depiction of the interactions among the elements of Resume Builder. It represents the methodology used in system analysis to identify, clarify, and organize system requirements of Resume Builder. The main actors of Resume Builder in this Use Case Diagram are: Dataset, User, Algorithm who perform the different type of use cases such as Manage Resume, Manage Skills, Manage Job, Manage Formats, Manage Resume Types, Manage Users and Full Resume Builder Operations. Major elements of the UML use case diagram of Resume Builder are shown on the Fig 3.2.



**Fig 3.2 – Use-Case Diagram**

### **3.3 SOFTWARE REQUIREMENTS SPECIFICATION DOCUMENT**

#### **3.3.1 HARDWARE REQUIREMENTS**

Processor	: Pentium Dual Core 2.00GHZ
Hard disk	: 120 GB
RAM	: 2GB (minimum)
Keyboard	: 110 keys enhanced

#### **3.3.2 SOFTWARE REQUIREMENTS**

Operating system	: Windows7 (with service pack 1), 8, 8.1 and 10
IDE	: Anaconda
Language	: Python

## **CHAPTER 4**

### **DESCRIPTION OF PROPOSED SYSTEM**

This system is a resume ranking software that uses natural language processing (NLP) and machine learning. Following resume screening, the software rates prospects in real-time depending on the recruiter's job needs. The web application aims to order the resumes, by intelligently reading job descriptions as input and comparing the resumes which fall into the category of given Job Descriptions. It provides a ranking after filtering and recommends the better resume for a given textual job description. In order to match and rate candidates in real-time, the software employs natural language processing.

This work takes a different approach as it focuses mainly on the content of the resumes where the extraction of skills are performed and related parameters to match candidates with the job descriptions

#### **4.1 SELECTED METHODOLOGY OR PROCESS MODEL**

##### **4.1.1 NATURAL LANGUAGE PROCESSING (NLP)**

Resume collection is being performed and folder Structure creation is being done. Data Cleaning such as removing clutter and unnecessary punctuation would be taken off, Feature Engineering would be performed for enhancement This includes removing stop words, punctuation, and stemming. This process will construct a graph with sentences as the vertices. Importing necessary libraries is performed. This library creates a summary of the supplied information within the word limit. With the help of Natural language processing techniques, the application which extracts the names of people, places, and other entities from text, the main goal being to get a reduced version of it that retains the most important information. It is followed by Exploratory Data Analysis. Exploratory data analysis (EDA) is often a necessary task in uncovering hidden patterns, detecting outliers, and identifying important variables and any anomalies in data.

#### **4.1.2 TF-IDF**

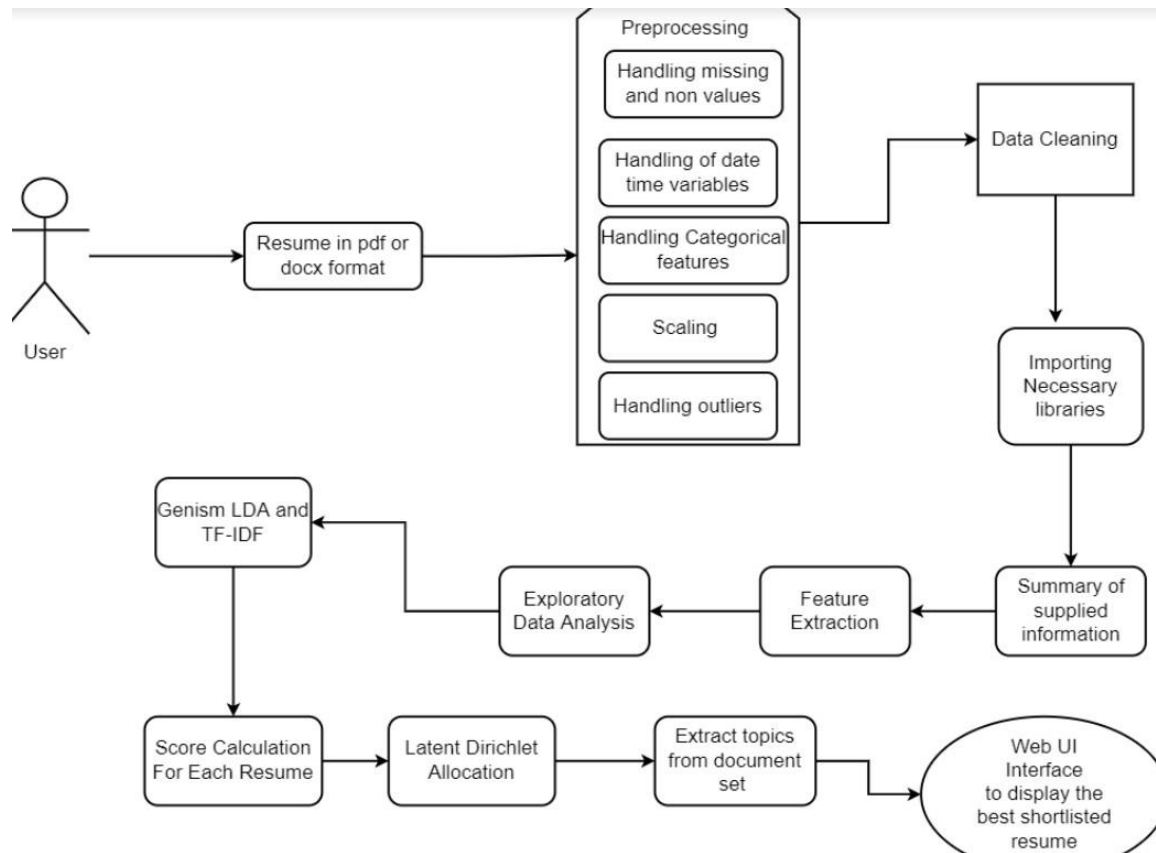
At this stage, a dynamic Script for the Tf-Idf approach is written. Term frequency-inverse document frequency, is a numerical statistic that is intended to reflect how important a word is to a document in a collection. TF-IDF is word frequency scores that aim to emphasize phrases that are more interesting, e.g., common in a text but not across texts, without delving into the arithmetic. The TF-IDF Vectorizer tokenizes texts, learns vocabulary, inverts frequency weightings, and allows encoding new ones. It provides information on a word frequency in the documents. Higher the TF- IDF score of a term which is computed using the above equations represents more relevance in a document.

#### **4.1.3 LATENT DIRICHLET ALLOCATION (LDA)**

A tool and technique for Topic Modeling, Latent Dirichlet Allocation (LDA) classifies or categorizes the text into a document and the words per topic, these are modeled based on the Dirichlet distributions and processes. Latent Dirichlet Allocation has been used in the application for the following functions- Discovering the hidden themes in the data. Classifying the data into the discovered themes. Using the classification to organize/summarize/search the documents. The application, then deals with the calculation of the score for a candidate's resume according to the job posting they have applied for. According to the score each candidate's resume receives, a rank list will be made with the candidate receiving a higher score placed higher as compared to the candidate receiving a lower score.

### **4.2 ARCHITECTURE OF PROPOSED SYSTEM**

This diagram is nothing but a simple description of all the entities that have been incorporated into the system. The diagram represents the relations between each of them and involves a sequence of decision-making processes and steps. You can simply call it a visual or the whole process and its implementation. All functional correspondences are explained in this Fig 4.2.



**Fig 4.2 – Architecture Diagram**

## 4.3 DESCRIPTION OF SOFTWARE FOR IMPLEMENTATION AND TESTING PLAN OF THE PROPOSED MODEL

### 4.3.1 PLATFORM SPECIFICATION– ANACONDA

Anaconda is an open-source package manager for Python and R. It is the most popular platform among data science professionals for running Python and R implementations. There are over 300 libraries in data science, so having a robust distribution system for them is a must for any professional in this field. Anaconda simplifies package deployment and management. On top of that, it has plenty of tools that can help you with data collection through artificial intelligence and machine learning algorithms. With Anaconda, you can easily set up, manage, and share Conda environments. Moreover, you can deploy any required project with a few clicks when you're using Anaconda. There are many advantages to using Anaconda and the following are the most prominent ones among them: Anaconda

is free and open-source. This means you can use it without spending any money. In the data science sector, Anaconda is an industry staple. It is open-source too, which has made it widely popular. If you want to become a data science professional, you must know how to use Anaconda for Python because every recruiter expects you to have this skill. It is a must-have for data science.

It has more than 1500 Python and R data science packages, so you don't face any compatibility issues while collaborating with others. For example, suppose your colleague sends you a project which requires packages called A and B but you only have package A. Without having package B, you wouldn't be able to run the project. Anaconda mitigates the chances of such errors. You can easily collaborate on projects without worrying about any compatibility issues. It gives you a seamless environment that simplifies deploying projects. You can deploy any project with just a few clicks and commands while managing the rest. Anaconda has a thriving community of data scientists and machine learning professionals who use it regularly. If you encounter an issue, chances are, the community has already answered the same. On the other hand, you can also ask people in the community about the issues you face there, it's a very helpful community ready to help new learners. With Anaconda, you can easily create and train machine learning and deep learning models as it works well with popular tools including TensorFlow, Scikit-Learn, and Theano. You can create visualizations by using Bokeh, Holoviews, Matplotlib, and Datashader while using Anaconda.

- **How to Use Anaconda for Python**

Now that we have discussed all the basics in our Python Anaconda tutorial, let's discuss some fundamental commands you can use to start using this package manager. Listing All Environments To begin using Anaconda, you'd need to see how many Conda environments are present in your machine.

`conda env list` - It will list all the available Conda environments in your machine. Creating a New Environment You can create a new Conda environment by going to the required directory and use this command:

`conda create -n <your_environment_name>`

You can replace `<your_environment_name>` with the name of your environment. After entering this command, conda will ask you if you want to proceed to which you should reply with `y: proceed ([y])/n)?`

On the other hand, if you want to create an environment with a particular version of Python, you should use the following command:

```
conda create -n <your_environment_name> python=3.6
```

Similarly, if you want to create an environment with a particular package, you can use the following command:

```
conda create -n <your_environment_name> pack_name
```

Here, you can replace `pack_name` with the name of the package you want to use.

If you have a `.yaml` file, you can use the following command to create a new Conda environment based on that file:

```
conda env create -n <your_environment_name> -f <file_name>.yaml
```

We have also discussed how you can export an existing Conda environment to a `.yaml` file later in this article.

- **Activating an Environment**

You can activate a Conda environment by using the following command:

```
conda activate <environment_name>
```

You should activate the environment before you start working on the same. Also, replace the term `<environment_name>` with the environment name you want to activate. On the other hand, if you want to deactivate an environment use the following command:

```
conda deactivate
```

- **Installing Packages in an Environment**

Now that you have an activated environment, you can install packages into it by using the following command:

```
conda install <pack_name>
```

Replace the term `<pack_name>` with the name of the package you want to install in your Conda environment while using this command.



- **Updating Packages in an Environment**

If you want to update the packages present in a particular Conda environment, you should use the following command:

```
conda update
```

The above command will update all the packages present in the environment. However, if you want to update a package to a certain version, you will need to use the following command:

```
conda install <package_name>=<version>
```

- **Exporting an Environment Configuration**

Suppose you want to share your project with someone else (colleague, friend, etc.). While you can share the directory on Github, it would have many Python packages, making the transfer process very challenging. Instead of that, you can create an environment configuration .yml file and share it with that person. Now, they can create an environment like your one by using the .yml file.

For exporting the environment to the .yml file, you'll first have to activate the same and run the following command:

```
conda env export ><file_name>.yml
```

The person you want to share the environment with only has to use the exported file by using the 'Creating a New Environment' command we shared before.

- **Removing a Package from an Environment**

If you want to uninstall a package from a specific Conda environment, use the following command:

```
conda remove -n <env_name><package_name>
```

On the other hand, if you want to uninstall a package from an activated environment, you'd have to use the following command:

```
conda remove <package_name>
```

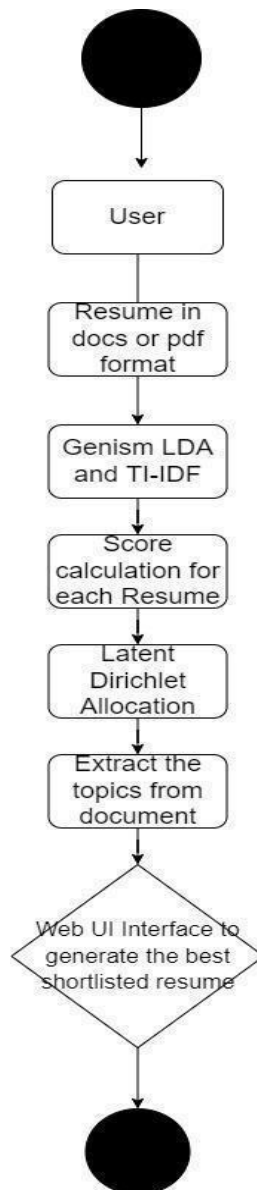
- **Deleting an Environment**

Sometimes, you don't need to add a new environment but remove one. In such cases, you must know how to delete a Conda environment, which you can do so by using the following command:

```
conda env remove --name <env_name>
```

The above command would delete the Conda environment right away.

## 4.4 PROJECT MANAGEMENT PLAN



**Fig 4.4 – Project management plan**

## CHAPTER 5

### IMPLEMENTATION DETAILS

#### 5.1 ALGORITHMS

We applied the following algorithms to solve this situation:

- natural language processing
- term frequency-inverse document frequency (tf-idf)
- latent dirichlet allocation (lda)

##### 5.1.1 NATURAL LANGUAGE PROCESSING

Natural language processing (NLP) is a field of artificial intelligence in which computers analyze, understand, and derive meaning from human language in a smart and useful way. By utilizing NLP, developers can organize and structure knowledge to perform tasks such as automatic summarization, translation, named entity recognition, relationship extraction, sentiment analysis, speech recognition, and topic segmentation. Resume collection is being performed and folder Structure creation is being done. Data Cleaning such as removing cluster and unnecessary punctuation would be taken off. Importing necessary libraries is performed. This library creates a summary of the supplied information within the word limit. With the help of Natural language processing techniques, the application which extracts the names of people, places, and other entities from text, the main goal being to get a reduced version of it that retains the most important information. It is followed by Exploratory Data Analysis. NLP is used to analyze text, allowing machines to understand how humans speak. This human-computer interaction enables real-world applications like automatic text summarization, sentiment analysis, topic extraction, named entity recognition, parts-of-speech tagging, relationship extraction, stemming, and more. NLP is commonly used for text mining, machine translation, and automated question answering.

A decade ago, the job-seeking and recruiting process were way different from what it is right now. With the constant and exponential advancement in technology, the talent acquisition process has changed by leaps and bounds. Earlier, if an applicant wishes to apply for a job, he needs to drop a resume at the

doorstep of the office in the hope of getting called for an interview. But now the scenario is totally different. The immense increase of internet connectivity has enabled change in the recruitment process of companies. With the various job boards and e-recruitment portals, both the recruiters and candidates got some benefits and convenience but there are a few challenges as well. Earlier, the companies who received 10–50 resumes on a daily basis, now are flooded with thousands of resumes. It's practically impossible and cost-ineffective to go through these many resumes. Also, there is no standard resume format that the candidate uses. People with different backgrounds, industries and job profiles have their own way of presenting details on their resume which is a daunting task for recruiters. To overcome this, companies and websites started seeking the candidates' details through online forms but they also fail to do justice to the process as the forms are very generic in nature. All these challenges have been substantially achieved by the Application Tracking System aka ATS which is based on Natural Language Processing.

Natural Language Processing divides the content mentioned in the resume into small grammatical parts that is known as parsing. This allows the system to extract the relevant information from the resume and create a summary of it irrespective of the order and format of the resume. After creating the summary, the system ranks the candidates by mapping the similarity between the content mentioned in their resume and in the job description. So, this is how the NLP is used in resume screening. Let us walk through the technical depth of the process. Talking about the Resume Screening using NLP, there are two major phases that the system works in. The first is Information Extraction and another one is Candidate Recommendation on the basis of Content. Let us understand these things.

- **INFORMATION EXTRACTION**

As we know, there is no standard format for resumes so inconsistencies and irrelevant data are inevitable. The objective of information extraction is to extract relevant information or keywords from the unstructured data in the resume without any human intervention. Using techniques like Tokenisation, Stemming, POS Tagging etc shown In Fig 4.4 the system can obtain relevant job-related

information from the uploaded candidates. The summary of each resume is created in a JSON format that can be easily used for further processing tasks in the next phase of this resume screening system.

- **TOKENISATION**

The process of identifying terms or words that form up a character sequence is called tokenization. This involved dividing the big text into smaller parts called tokens; the characters such as whitespace and punctuation are removed in this process. Through Tokenization, we can get information like the number of words in a text, frequency of a particular word in the text etc. Tokenization is an essential step for further text processing such as removal of stop words, lemmatization and stemming.

- **STEMMING AND LEMMATIZATION**

In the English language, a single word has multiple forms in multiple sentences according to its grammatical rules. The task here is to reduce all the derived or altered forms into their root word. We can overcome this by stemming and lemmatization but their approaches are different from each other. Stemming is the mechanism of reducing inflected or derived words to their word root, or stem. This involves the removal of derivational affixes. On the basis of ruled based algorithms, the words often undergo over-steaming and under-streaming. Whereas, lemmatization is the process of using a language dictionary to perform an accurate reduction to root words. Unlike Stemming that merely cuts off tokens by easy pattern matching, lemmatization is an additional careful approach that uses language vocabulary and morphological analysis of words to present lingually correct lemmas. This implies lemmatization utilises the data of context and so will completely differentiate between words that have different meanings as per parts of speech.

- **PARTS OF SPEECH (POS) TAGGING**

POS is a process in which on the basis of a word's context in a sentence, its grammatical information is assigned. The POS tag specifies whether the word is a noun, pronoun, verb, adjective, etc. as per its usage within the sentence. It's crucial to assign these tags to understand the right meaning of a sentence and for

building data graphs for named entity recognition. For example: In the sentence “I am building a website”, here the building is a verb, but in the sentence “I live in the tallest building of Paris”, here the building is a noun.

### **5.1.2 TERM FREQUENCY-INVERSE DOCUMENT FREQUENCY (TF-IDF)**

It is a numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus. It is often used as a weighting factor in searches of information retrieval, text mining, and user modeling. The tf-idf value increases proportionally to the number of times a word appears in the document and is offset by the number of documents in the corpus that contain the word, which helps to adjust for the fact that some words appear more frequently in general. tf-idf is one of the most popular term-weighting schemes today.

TF-IDF (term frequency-inverse document frequency) is a statistical measure that evaluates how relevant a word is to a document in a collection of documents. This is done by multiplying two metrics: how many times a word appears in a document, and the inverse document frequency of the word across a set of documents. It has many uses, most importantly in automated text analysis, and is very useful for scoring words in machine learning. TF-IDF was invented for document search and information retrieval. It works by increasing proportionally to the number of times a word appears in a document but is offset by the number of documents that contain the word. So, words that are common in every document, such as this, what, and if, rank low even though they may appear many times since they don't mean much to that document in particular. However, if the word Bug appears many times in a document, while not appearing many times in others, it probably means that it's very relevant. For example, if what we're doing is trying to find out which topics some NPS responses belong to, the word Bug would probably end up being tied to the topic Reliability, since most responses containing that word would be about that topic.

TF-IDF for a word in a document is calculated by multiplying two different metrics: The term frequency of a word in a document. There are several ways of calculating this frequency, with the simplest being a raw count of instances a word

appears in a document. Then, there are ways to adjust the frequency, by the length of a document, or by the raw frequency of the most frequent word in a document. The inverse document frequency of the word across a set of documents. This means, how common or rare a word is in the entire document set. The closer it is to 0, the more common a word is. This metric can be calculated by taking the total number of documents, dividing it by the number of documents that contain a word, and calculating the logarithm. So, if the word is very common and appears in many documents, this number will approach 0. Otherwise, it will approach 1. Multiplying these two numbers results in the TF-IDF score of a word in a document. The higher the score, the more relevant that word is in that particular document.

Machine learning with natural language is faced with one major hurdle – its algorithms usually deal with numbers, and natural language is, well, text. So, we need to transform that text into numbers, otherwise known as text vectorization. It's a fundamental step in the process of machine learning for analyzing data, and different vectorization algorithms will drastically affect end results, so you need to choose one that will deliver the results you're hoping for. Once you've transformed words into numbers, in a way that's machine learning algorithms can understand, the TF-IDF score can be fed to algorithms such as Naive Bayes and Support Vector Machines, greatly improving the results of more basic methods like word counts. This works simply by putting, a word vector represents a document as a list of numbers, with one for each possible word of the corpus. Vectorizing a document is taking the text and creating one of these vectors, and the numbers of the vectors somehow represent the content of the text. TF-IDF enables us to gives us a way to associate each word in a document with a number that represents how relevant each word is in that document. Then, documents with similar, relevant words will have similar vectors, which is what we are looking for in a machine learning algorithm.

At this stage, a dynamic Script for the Tf-Idf approach is written. Term frequency-inverse document frequency, is a numerical statistic that is intended to reflect how important a word is to a document in a collection. The TF-IDF Victimizer tokenizes texts, learns vocabulary, inverts frequency weightings, and

allows encoding new ones. It provides information on a word frequency in the documents. Higher the TF- IDF score of a term which is computed using the above equations represents more relevance in a document.

### **5.1.3 LATENT DIRICHLET ALLOCATION (LDA)**

A tool and technique for Topic Modeling, Latent Dirichlet Allocation (LDA) classifies or categorizes the text into a document and the words per topic, these are modeled based on the Dirichlet distributions and processes. Latent Dirichlet Allocation has been used in the application for the following functions:

- Discovering the hidden themes in the data.
- Classifying the data into the discovered themes.
- Using the classification to organize/summarize/search the documents.

The application then deals with the calculation of the score for a candidate's resume according to the job posting they have applied for. According to the score each candidate's resume receives, a rank list will be made with the candidate receiving a higher score placed higher as compared to the candidate receiving a lower score. By displaying a resume list in order of relevance to the position, the technique ranks CVs according to their match with the job description, making it easy for recruiters. Customized options for job description in our web application are shown. Slider options are present in our app for a better user experience. The web application would also have a Bar Chart that shows the stats. The Resume Screening System replaces ineffective manual screening, ensuring that no candidate is overlooked.

## **5.2 EXISTING SYSTEM**

The existing system is a traditional Machine Learning based system. It gives lower rates of accuracy. It has lower efficiency. It gives lower inaccurate results. This system may lead to loss of human potential.



### 5.3 PROPOSED SYSTEM

Our system is resume ranking software that uses natural language processing (NLP) and machine learning. This AI-powered resume screening program goes beyond keywords to contextually screen resumes. Following resume screening, the software rates prospects in real-time depending on the recruiter's job needs. The web application aims to order the resumes, by intelligently reading job descriptions as input and comparing the resumes which fall into the category of given Job Descriptions. It provides a ranking after filtering and recommends the better resume for a given textual job description.

In order to match and rate candidates in real-time, the software employs natural language processing. Unlike generic processes, this app utilizes Mong for string matching, Cosine Similarity, Overlapping coefficient Natural Language. Our work takes a different approach as it focuses mainly on the content of the resumes where we perform the extraction of skills and related parameters to match candidates with the job descriptions. The interactive web application will allow the job applicants to submit their resumes and apply for job postings they may still be interested in. The resumes submitted by the candidates are then compared with the job profile requirement posted by the company recruiter by using techniques like machine learning and Natural Language Processing (NLP).

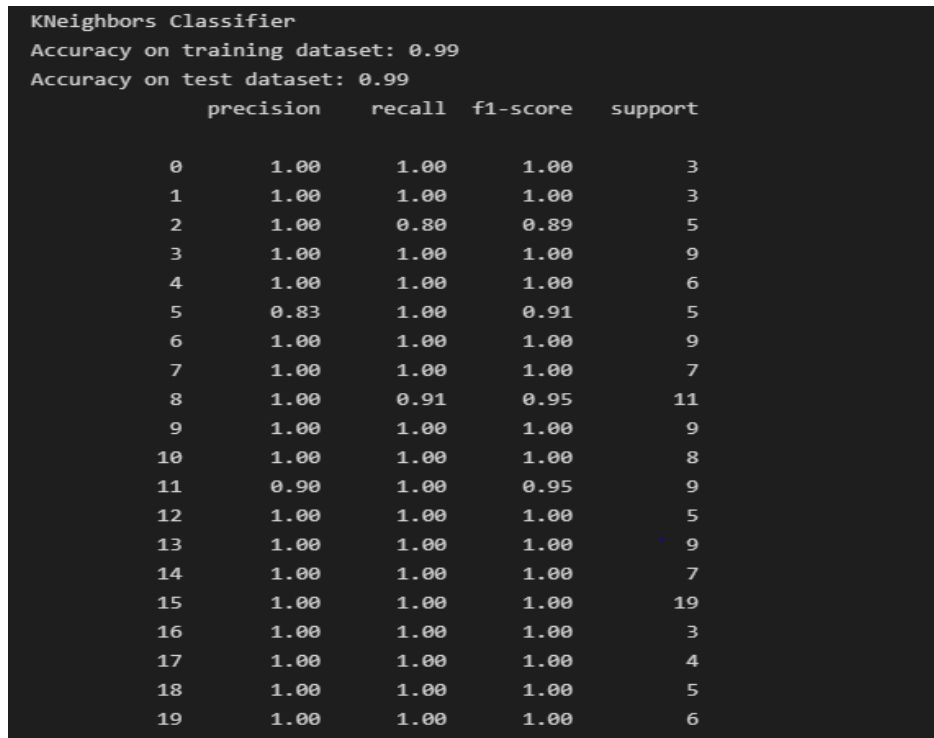
Scores can then be given to the resumes and they can be ranked from highest match to lowest match. This ranking is made visible only to the company recruiter who is interested to select the best candidates from a large pool of candidates. This is done with the aim to aid the recruiters of any company from the long and tedious task of viewing and analyzing thousands of candidate resumes. The calculated ranking scores can then be utilized to determine best-fitting candidates for that particular job opening. Since the dynamic model leverages NLP, it gives the output instantly. While going through all these pipelines, It will score each resume and give out accurate output with higher efficiency, precision, and accuracy. It works the following way.

Resumes from the Data set are parsed to remove white spaces, numbers, stop words like and, or, etc. TF-IDF vectorization is then applied to convert the words in the resumes to vectors. The text in the job description is also converted to vectors using the TF-IDF vectorizer. Cosine distance is computed to measure the similarity between the resume and the job description provided and Then Mong String algorithm is applied to identify the resumes which are closely matching with the JD provided by the recruiters. The main contributions of our work can be summarized as follows: Use of our framework is not limited to a single field of application and is useful for many more real-world applications. Providing a ranking-based approach after filtering Framework to highlight skills of a resume

## CHAPTER 6

### RESULTS AND DISCUSSION

In this document, we developed an automated resume screening system that simplifies the e-recruitment process by reducing the numerous issues that recruiters faced when they relied on human shortlisting of candidates for a given job posting. The model was generated with high accuracy using the Kneighbors approach.



```
KNeighbors Classifier
Accuracy on training dataset: 0.99
Accuracy on test dataset: 0.99
```

	precision	recall	f1-score	support
0	1.00	1.00	1.00	3
1	1.00	1.00	1.00	3
2	1.00	0.80	0.89	5
3	1.00	1.00	1.00	9
4	1.00	1.00	1.00	6
5	0.83	1.00	0.91	5
6	1.00	1.00	1.00	9
7	1.00	1.00	1.00	7
8	1.00	0.91	0.95	11
9	1.00	1.00	1.00	9
10	1.00	1.00	1.00	8
11	0.90	1.00	0.95	9
12	1.00	1.00	1.00	5
13	1.00	1.00	1.00	9
14	1.00	1.00	1.00	7
15	1.00	1.00	1.00	19
16	1.00	1.00	1.00	3
17	1.00	1.00	1.00	4
18	1.00	1.00	1.00	5
19	1.00	1.00	1.00	6

***Fig.6.1 Kneighbors classified is constructed***

Using Kneighbors algorithm, we have got 98% of accuracy which is the perfect accuracy which is shown in Fig 6.1. On first glance, Linear SVC and XGBoost provided the highest accuracy and a favorable confusion matrix. Both were able to correctly forecast the Role for a specific job description given to the system.



## **CHAPTER 7**

### **SUMMARY**

#### **7.1 CONCLUSION**

Our algorithm was successfully able to screen and shortlist the best candidates with the help of NLP. Highly accurate results were obtained by using Latent Dirichlet Allocation to display the best-shortlisted resume on Web Ui. The web application was successfully able to order the resumes, by intelligently reading job descriptions as input and comparing the resumes which fall into the category of given Job Descriptions. The organization receives a large number of applications for each employment opening. Finding the right candidate's application from a sea of resumes is a time-consuming endeavor for any company these days. The classification of a candidate's resume is a laborious, time-consuming, and resource-intensive process. To address this problem, we created an automated machine learning-based algorithm that recommends acceptable applicant resumes to HR based on the job description provided. The suggested methodology had two stages: first, it classified resumes into various groups. Second, it suggests resumes based on their resemblance to the job description. If an industry produces a high number of resumes, the proposed approach can be used to create an Industry-specific model.

By engaging domain experts such as HR professionals, a more accurate model may be built, and HR professionals' feedback can be used to iteratively enhance the model. The results from the model are encouraging. The resume classifier application is successful in automating the manual task of project allocation to the new recruits of the organization based on the interests, work experience, and expertise mentioned by the candidate in the profile. The Resume Screening System replaces ineffective manual screening, ensuring that no candidate is overlooked. The need for efficient and effective resume screening is at the heart of every excellent recruitment strategy. The system will be able to accept or reject a job applicant based on two factors: the company's requirements must match the skills listed in the applicant's resume, and the test evaluation will

be based on the applicant's skills, ensuring that the resumes uploaded by the applicant are genuine and the applicant is truly knowledgeable about the skills. Using NLP(Natural Language Processing) and ML(Machine Learning) to rank the resumes according to the given constraint, this intelligent system ranks the resume of any format according to the given constraints or the following requirement provided by the client company. We will basically take the bulk of input resume from the client company and that client company will also provide the requirement and the constraints according to which the resume should be ranked by our system. Besides the information provided by the resume, we are going to read the candidate's social profiles (like LinkedIn, GitHub, etc.) which will give us more genuine information about that candidate. The application automates the task of project allocation, thereby eliminating the tedious and redundant affair of opening and analyzing the resumes manually by the HR team of the organization.

## **7.2 FUTURE WORK**

The application can be extended further to other domains like Telecom, Healthcare, E-commerce, and public sector jobs. We also wish to put into effect and present a smart evaluation in the consistent database to survey with the present models. Efforts can be made to explore whether it is possible to identify in advance what lists might be ranked worse than the current baseline and to investigate whether there is another way to transform word embedding's into document embedding's, such as using doc2vec, that could have a greater impact on LTR results. Another area we would want to focus on is ranking the resumes of people in different languages. Be it Dutch, German, French, or Hindi, we want to explore further ways to analyze and operate our model on major languages of the world.

## REFERENCES

- [1] Abeer Zaroor; Mohammed Maree; Muath Sabha," A Hybrid Approach to Conceptual Classification and Ranking of Resumes and Their Corresponding Job Post", Smart Innovation, Systems and Technologies book series (SIST, volume 72), 2017
- [2] Anushka Sharma; Smiti Singhal; Dhara Ajudia," Intelligent Recruitment System Using NLP", International Conference on Artificial Intelligence and Machine Vision (AIMV), 2021
- [3] Fahad, SK Ahmed, and Abdul Samad Ebrahim Yahya. "Inflectional review of deep learning on natural language processing." 2018 International Conference on Smart Computing and Electronic Enterprise (ICSEE). IEEE, 2018
- [4] Garima Bhardwaj; S. Vikram Singh; Vinay Kumar," An Empirical Study of Artificial Intelligence and its Impact on Human Resource Functions", International Conference on Computation, Automation and Knowledge Management (ICCAKM), 2020
- [5] Jitendra Purohit; Aditya Bagwe; Rishabh Mehta; Ojaswini Mangaonkar; Elizabeth George, "Natural Language Processing based Jaro-The Interviewing Chatbot", 2019 3rd International Conference on Computing Methodologies and Communication (ICCMC), August 2019
- [6] Mujtaba, Dena F., and Nihar R. Mahapatra. "Ethical Considerations in AI-Based Recruitment." 2019 IEEE International Symposium on Technology and Society (ISTAS). IEEE, 2019.
- [7] Pradeep Kumar Roy; Sarabjeet Singh Chowdhary; Rocky Bhatia," A Machine Learning approach for automation of Resume Recommendation system", Procedia Computer Science Volume 167, Pages 2318-2327, 2020

[8] Rajath V; Riza Tanaz Fareed; Sharadadevi Kaganurmath," Resume Classification And Ranking Using KNN And Cosine Similarity",international JOURNAL OF ENGINEERING RESEARCH & TECHNOLOGY, Volume 10, Issue 08, AUGUST 2021

[9] Tejaswini K; Umadevi V; Shashank M Kadiwal; Sanjay Revanna," Design and Development of Machine Learning based Resume Ranking System", Global Transitions Proceedings, October 2021

[10] Yong Luo; Huaizheng Zhang; Yongjie Wang; Yonggang Wen; Xinwen Zhang," ResumeNet: A Learning-Based Framework for Automatic Resume Quality Assessment", 2018 IEEE International Conference on Data Mining (ICDM), December 2018



## APPENDIX

### A. SOURCE CODE

```
import matplotlib.colors as mcolors
import gensim
import gensim.corpora as corpora
from operator import index
from wordcloud import WordCloud
from pandas._config.config import options
import pandas as pd
import streamlit as st
import plotly.express as px
import plotly.graph_objects as go
import matplotlib.pyplot as plt
import Similar
from PIL import Image
import time

image = Image.open('Images//logo.png')
st.image(image, use_column_width=True)
st.title("Resume Screening")
# Reading the CSV files prepared by the fileReader.py
Resumes = pd.read_csv('Resume_Data.csv')
Jobs = pd.read_csv('Job_Data.csv')
#####JOB DESCRIPTION CODE #####
# Checking for Multiple Job Descriptions
# If more than one Job Descriptions are available, it asks user to select one as
well.
if len(Jobs['Name']) <= 1:
    st.write(
        "There is only 1 Job Description present. It will be used to create scores.")
else:
    st.write("There are ", len(Jobs['Name']),
```

"Job Descriptions available. Please select one.")

# Asking to Print the Job Description Names

if len(Jobs['Name']) > 1:

```
    option_yn = st.selectbox(
        "Should we need to Show the Job Description Names?", options=['YES',
'NO'])
    if option_yn == 'YES':
        index = [a for a in range(len(Jobs['Name']))]
        fig = go.Figure(data=[go.Table(header=dict(values=["Job No.", "Job Desc.
Name"], line_color='darkslategray',
                                fill_color='lightskyblue'),
                                cells=dict(values=[index, Jobs['Name']],
                                line_color='darkslategray',
                                fill_color='cyan'))
                                ])
```

```
        fig.update_layout(width=700, height=400)
```

```
        st.write(fig)
```

# Asking to chose the Job Description

```
index = st.slider("Slide to select te JD : ", 0,
    len(Jobs['Name'])-1, 1)
```

```
option_yn = st.selectbox("Do you want Job Description to be displayed ?",
options=['YES', 'NO'])
```

```
if option_yn == 'YES':
```

```
    st.markdown("---")
```

```
    st.markdown("### Job Description :")
```

```
    fig = go.Figure(data=[go.Table(
        header=dict(values=["Job Description"],
            fill_color='#f0a500',
            align='center', font=dict(color='white', size=16)),
        cells=dict(values=[Jobs['Context'][index]],
            fill_color='#f4f4f4',
```

```

        align='left'))))
fig.update_layout(width=800, height=500)
st.write(fig)
st.markdown("---")

##### SCORE CALCULATION #####
@st.cache()
def calculate_scores(resumes, job_description):
    scores = []
    for x in range(resumes.shape[0]):
        score = Similar.match(
            resumes['TF_Based'][x], job_description['TF_Based'][index])
        scores.append(score)
    return scores

Resumes['Scores'] = calculate_scores(Resumes, Jobs)
Ranked_resumes = Resumes.sort_values(
    by=['Scores'], ascending=False).reset_index(drop=True)

Ranked_resumes['Rank'] = pd.DataFrame(
    [i for i in range(1, len(Ranked_resumes['Scores'])+1)])

##### SCORE TABLE PLOT #####
fig1 = go.Figure(data=[go.Table(
    header=dict(values=["Rank", "Name", "Scores"],
        fill_color='#00416d',
        align='center', font=dict(color='white', size=16)),
    cells=dict(values=[Ranked_resumes.Rank, Ranked_resumes.Name,
Ranked_resumes.Scores],
        fill_color='#d6e0f0',
        align='left'))))

fig1.update_layout(title="Top Ranked Resumes", width=700, height=1100)

```

```

st.write(fig1)

st.markdown("---")

fig2 = px.bar(Ranked_resumes,
              x=Ranked_resumes['Name'],          y=Ranked_resumes['Scores'],
              color='Scores',
              color_continuous_scale='haline', title="Score and Rank Distribution")
# fig.update_layout(width=700, height=700)
st.write(fig2)
st.markdown("---")

##### TF-IDF Code #####
@st.cache()
def get_list_of_words(document):
    Document = []

    for a in document:
        raw = a.split(" ")
        Document.append(raw)

    return Document
document = get_list_of_words(Resumes['Cleaned'])

id2word = corpora.Dictionary(document)
corpus = [id2word.doc2bow(text) for text in document]

lda_model = gensim.models.ldamodel.LdaModel(corpus=corpus,
                                              id2word=id2word, num_topics=6, random_state=100,
                                              update_every=3, chunksize=100, passes=50,
                                              alpha='auto', per_word_topics=True)

##### LDA CODE #####
@st.cache # Trying to improve performance by reducing the rerun computations

```

```

def format_topics_sentences(ldamodel, corpus):
    sent_topics_df = []
    for i, row_list in enumerate(ldamodel[corpus]):
        row = row_list[0] if ldamodel.per_word_topics else row_list
        row = sorted(row, key=lambda x: (x[1]), reverse=True)
        for j, (topic_num, prop_topic) in enumerate(row):
            if j == 0:
                wp = ldamodel.show_topic(topic_num)
                topic_keywords = ", ".join([word for word, prop in wp])
                sent_topics_df.append(
                    [i, int(topic_num), round(prop_topic, 4)*100, topic_keywords])
            else:
                break

    return sent_topics_df

```

```

##### Topic Word Cloud Code #####
# st.sidebar.button('Hit Me')
st.markdown("## Topics and Topic Related Keywords ")
st.markdown(
    """This Wordcloud representation shows the Topic Number and the Top
    Keywords that constitute a Topic.
    This further is used to cluster the resumes.    """)

```

```

cols = [color for name, color in mcolors.TABLEAU_COLORS.items()]

```

```

cloud = WordCloud(background_color='white',
                  width=2500,
                  height=1800,
                  max_words=10,
                  colormap='tab10',
                  collocations=False,
                  color_func=lambda *args, **kwargs: cols[i],

```

```

        prefer_horizontal=1.0)

topics = lda_model.show_topics(formatted=False)

fig, axes = plt.subplots(2, 3, figsize=(10, 10), sharex=True, sharey=True)

for i, ax in enumerate(axes.flatten()):
    fig.add_subplot(ax)
    topic_words = dict(topics[i][1])
    cloud.generate_from_frequencies(topic_words, max_font_size=300)
    plt.gca().imshow(cloud)
    plt.gca().set_title('Topic ' + str(i), fontdict=dict(size=16))
    plt.gca().axis('off')


plt.subplots_adjust(wspace=0, hspace=0)
plt.axis('off')
plt.margins(x=0, y=0)
plt.tight_layout()
st.pyplot(plt)

st.markdown("---")

### SETTING UP THE DATAFRAME FOR SUNBURST-GRAPH #####

df_topic_sents_keywords = format_topics_sentences(
    ldamodel=lda_model, corpus=corpus)
df_some = pd.DataFrame(df_topic_sents_keywords, columns=[
    'Document No', 'Dominant Topic', 'Topic % Contribution',
    'Keywords'])
df_some['Names'] = Resumes['Name']

df = df_some

```

```

st.markdown("## Topic Modeling with Resume using LDA ")
st.markdown(
    "Using LDA to divide the topics into a number of usefull topics and creating a
    Cluster of matching topic resumes. ")
fig3 = px.sunburst(df, path=['Dominant Topic', 'Names'], values='Topic %
    Contribution',
                    color='Dominant Topic', color_continuous_scale='viridis', width=800,
    height=800, title="Topic Distribution Graph")
st.write(fig3)

```

#### ##### RESUME PRINTING

```

option_2 = st.selectbox("Do you want to see the Best Resumes?", options=[
    'YES', 'NO'])
if option_2 == 'YES':
    indx = st.slider("Slide, Which resume to display ?:",
                     1, Ranked_resumes.shape[0], 1)

    st.write("Displaying Resume with Rank: ", indx)
    st.markdown("---")
    st.markdown("## **Resume** ")
    value = Ranked_resumes.iloc[indx-1, 2]
    st.markdown("#### The Word Cloud For Resume")
    wordcloud = WordCloud(width=800, height=800,
                           background_color='white',
                           colormap='viridis', collocations=False,
                           min_font_size=10).generate(value)
    plt.figure(figsize=(7, 7), facecolor=None)
    plt.imshow(wordcloud)
    plt.axis("off")
    plt.tight_layout(pad=0)
    st.pyplot(plt)

```

```

st.write("With a Match Score of :", Ranked_resumes.iloc[indx-1, 6])
fig = go.Figure(data=[go.Table(
    header=dict(values=["Resume"],
        fill_color='#f0a500',
        align='center', font=dict(color='white', size=16)),
    cells=dict(values=[str(value)],
        fill_color='#f4f4f4',
        align='left'))])

fig.update_layout(width=800, height=1200)
st.write(fig)
st.markdown("---")
import spacy
import Distill
try:
    nlp = spacy.load('en_core_web_sm')

except ImportError:
    print("Spacy's English Language Modules aren't present \n Install them by doing
\n python -m spacy download en_core_web_sm")

def _base_clean(text):
    """
    Takes in text read by the parser file and then does the text cleaning.
    """
    text = Distill.tokenize(text)
    text = Distill.remove_stopwords(text)
    text = Distill.remove_tags(text)
    text = Distill.lemmatize(text)
    return text

def _reduce_redundancy(text):
    """

```



Takes in text that has been cleaned by the `_base_clean` and uses `set` to reduce the repeating words

giving only a single word that is needed.

```
"""
```

```
return list(set(text))
```

```
def _get_target_words(text):
```

```
"""
```

Takes in text and uses Spacy Tags on it, to extract the relevant Noun, Proper Noun words that contain words related to tech and JD.

```
"""
```

```
target = []
```

```
sent = " ".join(text)
```

```
doc = nlp(sent)
```

```
for token in doc:
```

```
    if token.tag_ in ['NN', 'NNP']:
```

```
        target.append(token.text)
```

```
return target
```

```
def Cleaner(text):
```

```
    sentence = []
```

```
    sentence_cleaned = _base_clean(text)
```

```
    sentence.append(sentence_cleaned)
```

```
    sentence_reduced = _reduce_redundancy(sentence_cleaned)
```

```
    sentence.append(sentence_reduced)
```

```
    sentence_targetted = _get_target_words(sentence_reduced)
```

```
    sentence.append(sentence_targetted)
```

```
    return sentence
```

```
import nltk
```

```
import spacy
```

```
import re
```

```
from nltk.tokenize import word_tokenize, sent_tokenize
```

```
from nltk.corpus import stopwords
```

```

# Define english stopwords
stop_words = stopwords.words('english')
# load the spacy module and create a nlp object
# This need the spacy en module to be present on the system.
nlp = spacy.load('en_core_web_sm')
# proces to remove stopwords form a file, takes an optional_word list
# for the words that are not present in the stop words but the user wants them
deleted.

```

```

def remove_stopwords(text, stopwords=stop_words, optional_params=False,
optional_words=[]):
    if optional_params:
        stopwords.append([a for a in optional_words])
    return [word for word in text if word not in stopwords]

```

```

def tokenize(text):
    # Removes any useless punctuations from the text
    text = re.sub(r'^\w\s', '', text)
    return word_tokenize(text)

```

```

def lemmatize(text):
    # the input to this function is a list
    str_text = nlp(" ".join(text))
    lemmatized_text = []
    for word in str_text:
        lemmatized_text.append(word.lemma_)
    return lemmatized_text

```

```

# internal fuction, useless right now.

```

```

def _to_string(List):
    # the input parameter must be a list
    string = " "

```

```

return string.join(List)

def remove_tags(text, postags=['PROPN', 'NOUN', 'ADJ', 'VERB', 'ADV']):
    """
    Takes in Tags which are allowed by the user and then eliminates the rest of the
    words
    based on their Part of Speech (POS) Tags.
    """
    filtered = []
    str_text = nlp(" ".join(text))
    for token in str_text:
        if token.pos_ in postags:
            filtered.append(token.text)
    return filtered

from operator import index
from pandas._config.config import options
import Cleaner
import textextract as tx
import pandas as pd
import os
import tf_idf

resume_dir = "Data/Resumes/"
job_desc_dir = "Data/JobDesc/"
resume_names = os.listdir(resume_dir)
job_description_names = os.listdir(job_desc_dir)

document = []

def read_resumes(list_of_resumes, resume_directory):
    placeholder = []
    for res in list_of_resumes:
        temp = []
        temp.append(res)

```

```

        text = tx.process(resume_directory+res, encoding='ascii')
        text = str(text, 'utf-8')
        temp.append(text)
        placeholder.append(temp)
    return placeholder

document = read_resumes(resume_names, resume_dir)

def get_cleaned_words(document):
    for i in range(len(document)):
        raw = Cleaner.Cleaner(document[i][1])
        document[i].append(" ".join(raw[0]))
        document[i].append(" ".join(raw[1]))
        document[i].append(" ".join(raw[2]))
        sentence = tf_idf.do_tfidf(document[i][3].split(" "))
        document[i].append(sentence)
    return document

Doc = get_cleaned_words(document)
Database = pd.DataFrame(document, columns=[
    "Name", "Context", "Cleaned", "Selective", "Selective_Reduced",
    "TF_Based"])
Database.to_csv("Resume_Data.csv", index=False)

def read_jobdescriptions(job_description_names, job_desc_dir):
    placeholder = []
    for tes in job_description_names:
        temp = []
        temp.append(tes)
        text = tx.process(job_desc_dir+tes, encoding='ascii')
        text = str(text, 'utf-8')
        temp.append(text)
        placeholder.append(temp)
    return placeholder

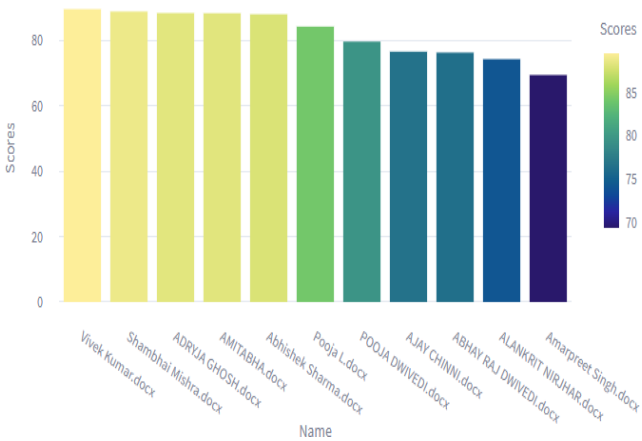
```

```
job_document = read_jobdescriptions(job_description_names, job_desc_dir)

Jd = get_cleaned_words(job_document)
jd_database = pd.DataFrame(Jd, columns=[
    "Name", "Context", "Cleaned", "Selective", "Selective_Reduced",
    "TF_Based"])
jd_database.to_csv("Job_Data.csv", index=False)
```

## B. SCREENSHOTS

### Score and Rank Distribution



## Score and rand distribution



Data visualization of word that were used more repeatedly in the resume collected

```

KNeighbors Classifier
Accuracy on training dataset: 0.99
Accuracy on test dataset: 0.99

```

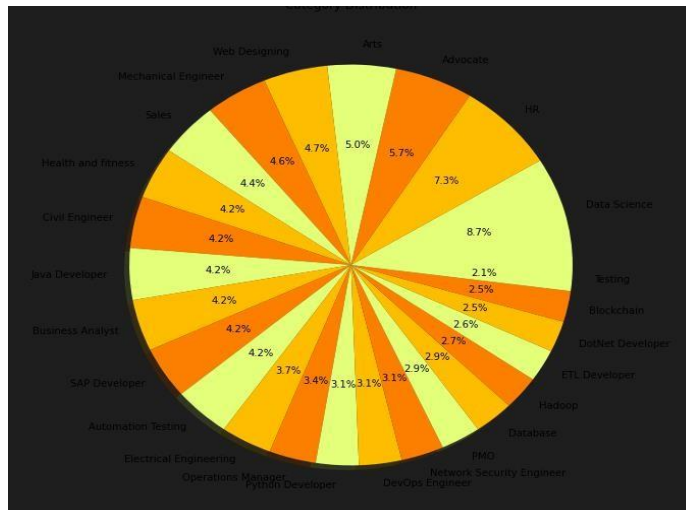
	precision	recall	f1-score	support
0	1.00	1.00	1.00	3
1	1.00	1.00	1.00	3
2	1.00	0.80	0.89	5
3	1.00	1.00	1.00	9
4	1.00	1.00	1.00	6
5	0.83	1.00	0.91	5
6	1.00	1.00	1.00	9
7	1.00	1.00	1.00	7
8	1.00	0.91	0.95	11
9	1.00	1.00	1.00	9
10	1.00	1.00	1.00	8
11	0.90	1.00	0.95	9
12	1.00	1.00	1.00	5
13	1.00	1.00	1.00	9
14	1.00	1.00	1.00	7
15	1.00	1.00	1.00	19
16	1.00	1.00	1.00	3
17	1.00	1.00	1.00	4
18	1.00	1.00	1.00	5
19	1.00	1.00	1.00	6
...				
accuracy			0.99	193
macro avg	0.99	0.99	0.99	193
weighted avg	0.99	0.99	0.99	193

We have build and got the perfect accuracy

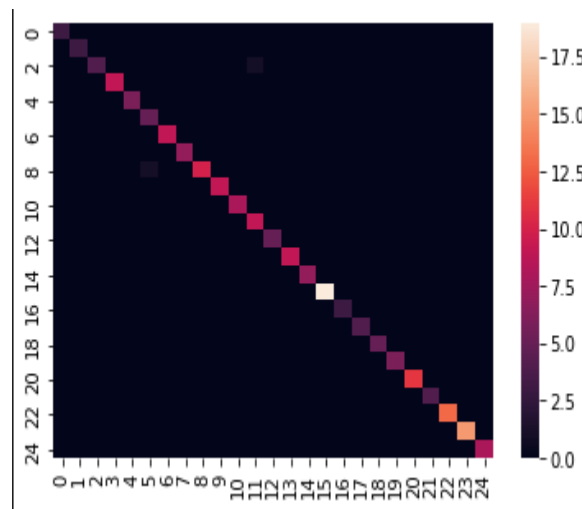


## Resume Screening

We have build an web application code with the algorithm which has given best performance.It has an accuracy of 98.3%.



Pie chart is plotted



Plotting the confusion matrix