

BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE, PILANI
Work Integrated Learning Programmes
Division

M.Tech.in Data Science and Engineering

Job recommendation System based on Applicant's preferences

DISSERTATION

Submitted in partial fulfillment of the requirements of the
MTech Data Science and Engineering Degree programme

By

Ritika Anand
2020SC04787

Under the supervision of

Rohini Sirsath

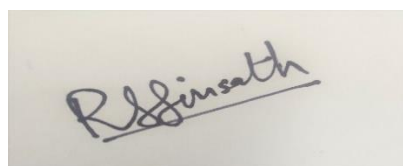
BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE
Pilani (Rajasthan) INDIA
March, 2023

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE, PILANI

CERTIFICATE

This is to certify that the Dissertation entitled Job Recommendation System based on Applicant's preferences.

and submitted by Ms. Ritika Anand IDNo.2020SC04787 in partial fulfillment of the requirements of DSECLZG628T Dissertation, embodies the work done by him/her under my supervision.

A photograph of a handwritten signature in blue ink on a light-colored surface. The signature appears to be 'R. Sirsath'.

Signature of the Supervisor
Place: Pune

Name: Rohini Sirsath
Designation: QA Analyst

Date: 14-03-2023

Acknowledgement

I am appreciative of the organization for giving the research topic. I want to thank my mentor Rohini Sirsath for her unwavering dedication to ensuring that the company has the best research facilities available. I am also appreciative to my company for giving me the opportunity to work on the use case and present it. They have given me the resources I needed and the sporadic assistance I needed, and I appreciate them for that. My deepest gratitude goes out to my colleague, who helped and inspired me throughout the process. My ability to recognize and address the project's issues was aided by their helpful criticism and input.

In addition, I want to express my gratitude to Prof. Ambati VKP of BITS Pilani for providing the direction and conducive working atmosphere that were essential to the successful completion of this research.

I want to sincerely thank my parents and my other family members for their unwavering affection, support, and inspiration during my academic career. My desire and drive to reach this goal came from their constant support. I want to thank everyone who helped finish this research project from the bottom of my heart once more.

BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE, PILANI
FIRST SEMESTER 2022-23

DSECLZG628T DISSERTATION

Dissertation Title : Job Recommendation System based on Applicant's preferences.

Name of Supervisor : Rohini Sirsath

Name of Student : Ritika Anand

ID No. of Student : 2020SC04787

Abstract

In emerging nations, especially India, a sizable chunk of the workforce is employed in the unorganized sector. The informal labor market is one of India's key economic sectors. The country's unemployment rate is rising because it is challenging for workers from different parts of the nation to contact businesses or entrepreneurs about jobs that are located outside of their state or place of residence. If this information is made regularly and methodically available to the employee, they can work constantly and profitably without taking breaks. With this indentation, they may avoid travelling ineffectively in pursuit of employment. Research was done to determine what workers and other stakeholders, such as contractors, needed.

In this project, we suggest creating a web portal with a system of job recommendations to connect unorganized workers with businesses and help them find the ideal position. The obstacles that unorganized workers face in finding informal work will be resolved by this idea, which will also connect unskilled workers across the country to jobs they can have without paying a commission, improving the employment rate. Both part-time and full-time positions are included here. College students looking for part-time employment will thus have the opportunity to apply for the position in accordance with their availability.

There has never been any prior contact between user data and job listing data in the data collected for our investigation. We have tackled the issue of cold start from both a User and Job standpoint using such a dataset. Also, employ content-based filtering, which was developed in support of natural language processing and cosine similarity, to analyses and quantify the similarity between the user preference and explicit elements of the job listing. This will help the system offer the top-n jobs to the user. The Recommender System can then be assessed using Cosine Similarity Score, Precision, and Recall. This web application will be installed on a local server. The Gradio web framework is used to create the presentation layer web application. Based on the available data, we would research the

current collaborative filtering techniques. In order to recommend the best model to the company, the output of the models would be compared and examined. For our use case study, we used algorithms like TF-IDF and Cosine Similarity employing cosine similarity, spaCy, and KNN.

Keywords: Recommendation System, Cosine Similarity, Machine Learning, Natural Language Processing Classification, Job Domain etc

List of Symbols & Abbreviations used

Tf-idf, TF-IDF - Term frequency- inverse document frequency
 CS - Cosine Similarity
 HF - Hybrid Model
 NLTK- Natural Language Toolkit
 CF, CB - Content Based Filtering
 ACF - Automated Collaborative Filtering
 NLP - Natural Language Processing
 Dtm - document-term matrix
 KNN - k-nearest neighbors algorithm

List of Figures

1.5.1 System Sequence Diagram -----	10
1.5.2 High Level System Architecture Diagram -----	11
3.3.1 Null column Bar plot -----	19
3.3.2.1 Scatter and Density Plot of Job Dataset -----	19
3.3.2.2 Scatter and Density Plot of User Dataset -----	20
3.3.3 Column Distribution Plot using subplot graph -----	20
4 Data Pre-processing Flow for User/Job Corpus -----	21
5 Process flow - Recommender system -----	25
5.1 Similarity Function Module -----	26
5.2.2 Sample representation of Count Vectorizer -----	28
5.2.3 Array representation of Nearest Neighbors after applying KNN -----	30
5.2.4 Sample Vector for the word "BANANA" -----	31
6.1.1 Similarity Score of Job Recommendation Model using TF-IDF and CS-----	32
6.1.2 Similarity Score of Job Recommendation Model using Count Vectorizer and CS-----	33
6.1.3 Similarity Score of Job Recommendation Model using TF-IDF and KNN -----	33
6.1.4 Similarity Score of Job Recommendation Model using TF-IDF and spaCy -----	34
7.2.1 Code snippet showcasing the input and output section for our Web application -----	36
7.2.2 Code snippet showcasing the parameters passed through the interface -----	36
7.2.3 UI Implementation -----	36

Table of contents

Cover page -----	2
Certificate -----	3
Acknowledgement -----	4
Abstract -----	5
List of Symbols and Abbreviations -----	6
List of Figures -----	6
Chapter 1	
Introduction -----	9
1.1 Recommendation System -----	9
1.2 Implement Recommendation System in Hiring process -----	9
1.3 Problem Statement -----	10
1.4 Objective of the project -----	10
1.5 System Design -----	10
1.5.1 System Sequence Diagram -----	10
1.5.2 High-Level System Architecture -----	11
1.6 Road Map for Dissertation -----	11
Chapter 2	
Literature Review Summary -----	13
2.1 Introduction -----	13
2.2 Types of Recommender System -----	13
2.3 Natural language processing -----	15
2.4 Inferences -----	16
Chapter 3	
Exploratory Data Analysis -----	17
3.1 Information about Datasets -----	17
3.1.1 Combined jobs dataset details -----	17
3.1.2 Job Views dataset details -----	17
3.1.3 Experience dataset details -----	17
3.1.4 Position of Interest Dataset details -----	18
3.2 Observations done on datasets -----	18
3.3 Data Visualization -----	18
3.3.1 Plotting null graph -----	19
3.3.2 Scatter and Density Plot -----	19
3.3.3 Column Distribution Plot -----	20
Chapter 4	
Data Pre-processing -----	21

4.1 Dropping Columns -----	21
4.2 Renaming column names -----	22
4.3 Data Imputation -----	22
4.4 Combining Columns -----	22
4.5 Lemmatization -----	22
4.6 Tokenization -----	23
4.7 Stop words -----	23
4.8 Lowercase -----	23
4.9 Fitting and transforming Data -----	23
4.10 Merging datasets -----	24
Chapter 5	
Modeling of Recommender System -----	25
5.1 Similarity Function Module -----	25
5.2 Model Implementation -----	27
5.2.1 Model I – Using tf-idf vectorizer and Cosine Similarity Measure -----	27
5.2.2 Model II - Using count vectorizer and Cosine Similarity Measure -----	28
5.2.3 Model III -Recommender System using KNN -----	29
5.2.4 Model IV - Recommender System using spaCy -----	30
Chapter 6	
Model Comparison and Analysis -----	32
6.1 Model Analysis -----	32
6.1.1 Model I - Job recommendation model using TF-IDF and Cosine Similarity----	32
6.1.2 Model II- Job recommendation model using Count Vectorizer and Cosine Similarity -----	33
6.1.3 Model III- Job recommendation model using TF-IDF and KNN -----	33
6.1.4 Model IV- Job recommendation model using TF-IDF and spaCy -----	34
6.2 Model Comparison -----	34
Chapter 7	
Application Deployment -----	35
7.1 Gradio -----	35
7.2 UI Implementation -----	35
Conclusion -----	37
Direction for Future Work -----	38
Bibliography/ References -----	39
Check list of items for Final report -----	42

Chapter 1

Introduction

Businesses afflicted by the pandemic disease progressively attempt to regain the impetus they lost as the entire planet begins to stand again. The time has come for firms to invest in their human resources in order to regain the momentum that was lost during this time. Many organizations urged their workers to work remotely when governments throughout the world asked enterprises to suspend operations in an effort to contain the pandemic. Contrarily, many other businesses began to cut their operational costs by letting go of workers who had permanent and contract positions. Those who lost their jobs as a result of the closure are waiting for their next chance. We humans naturally try to work past all obstacles to fulfil our mission in life. A person's daily job gives them a sense of purpose and they strive to get better at it, which leads them to leave their current job and hunt for another one; this is a cycle of hiring that never ends.

Many employment businesses have developed methods for offering the job board in order to serve the continuous cycle of the recruiting process from the viewpoint of the job candidate. Here, a job seeker searches for the positions that interest him and submits an application. Given the number of employment boards, in order to write a CV, create a job profile, or suggest new jobs to a job seeker, candidates often choose the tool that offers better services to them. Job seekers are looking for new possibilities that fit their talents more actively and persistently. Companies that target these job seekers, nevertheless, are having trouble determining their skill sets and making tailored employment recommendations.

1.1 Recommendation System

When we choose something online, these technologies assist us in weeding away comparable options. A recommender system is the idea of determining a user's preferences based on their online behavior, prior purchases, or system history. The demand for a recommender system has occasionally increased. The benefits of these technologies were initially used by the entertainment industries. Later recommender systems were used in online news and e-commerce, but relatively few organizations have tried using them in the employment process.

1.2 Implement Recommendation System in Hiring process

We have the chance to add a recommender system to the employment cycle because we are familiar with it. From both the employer and job seeker perspectives, we can implement a recommender system. We can construct a recommender system that might gather user preferences, such as user skills and location, from the viewpoint of job seekers. We are focusing on employment opportunities in the information and technology sector. These IT-related positions call for knowledge of programming languages, databases, frameworks, and user preferences across many platforms. The author was personally motivated to put into place a recommender system for a job seeker throughout my job

search. The user-submitted jobs would be available on all employment boards. Based on the terms we put in the search box, all job boards would show us the jobs the user had searched for. Is it not preferable to have a system in place that suggests employment based on user likes and expertise. In my dissertation, I'll be putting into practice a recommender system that uses filtering methods and NLP to suggest the best jobs based on user profiles.

1.3 Problem Statement

To create an efficient recommender system that will recommend jobs to the applicants in their position of interest, job description, company and location by taking information from the applicants through basic User Interface.

1.4 Objective of the project

Gather information from the applicants and check for which positions of interest, company, job description and location are of more interest to the job seekers. Help organization to suggest multiple ways of shortlisting jobs. Improve user experience by implementing User Interface. Combine NLP with classification techniques of machine learning in order to determine the best fit model with high accuracy.

1.5 System Design

In this chapter, we can showcase the System Sequence Diagram which explains the flow of our Recommendation System and will also explain about the system architectural design implemented in our project.

1.5.1 System Sequence Diagram

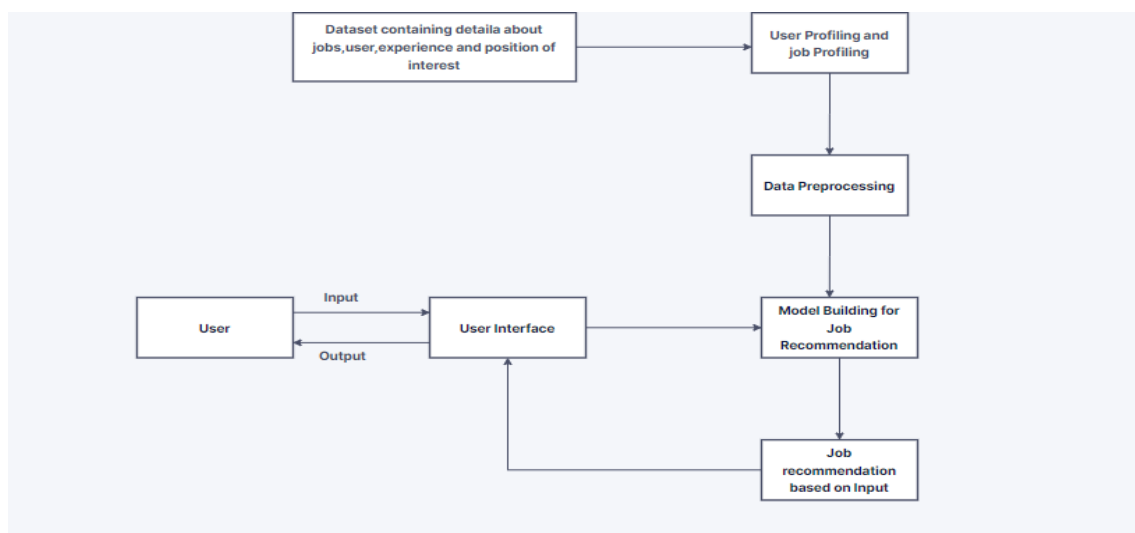


Fig 1.5.1 System Sequence Diagram

1.5.2 High-Level System Architecture

The high-level architecture diagram Figure 1.5.2 displays the system's overall layout.

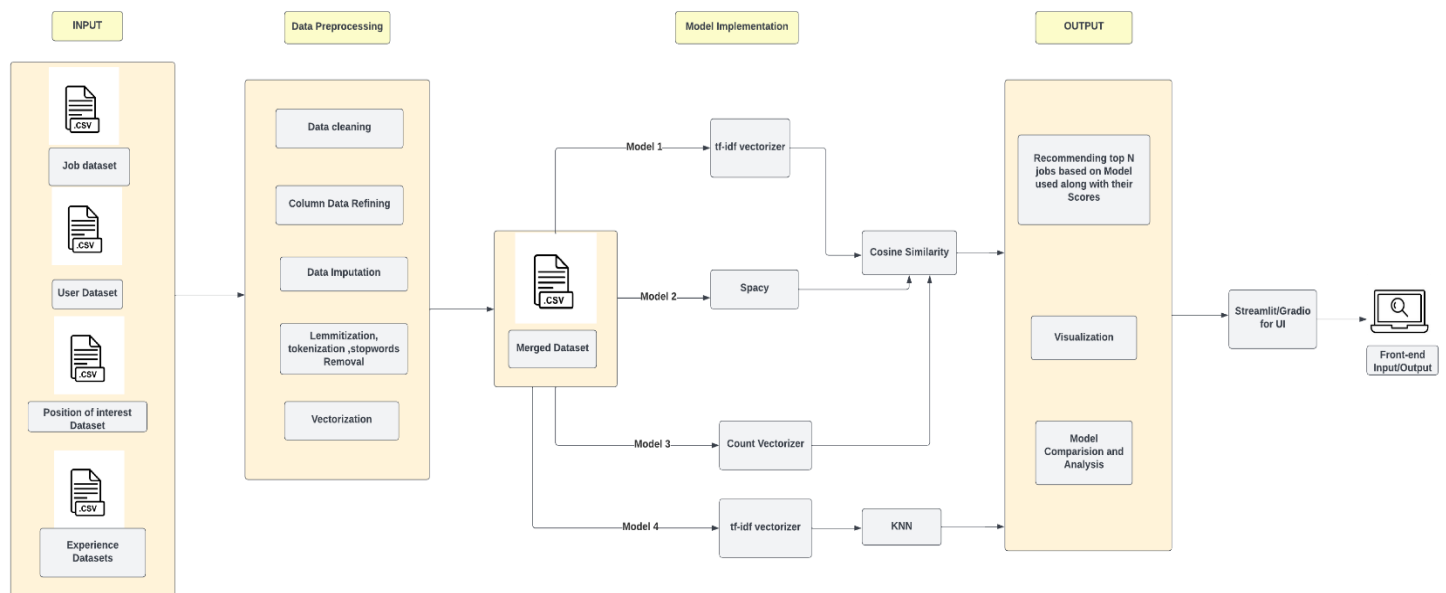


Fig 1.5.2 High Level System Architecture Diagram

1.6 Road Map for Dissertation

This dissertation is divided into numerous chapters, each of which is described below-

Chapter One- This chapter provides an overview of the subject, background information, and the author's inspiration for writing this dissertation. Also, the study question and purpose for this dissertation were briefly covered.

Chapter Two- A literature survey on recommender systems, natural language processing, and some similarity measures that have been used in recommender system research is provided in this chapter.

Chapter Three- In this Chapter, we elaborate on the Datasets explaining about the required columns showcasing the data types, identifying null and garbage values. Also includes the Data visualization part i.e plotting bar, distribution, scatter and density plot.

Chapter Four- In this chapter, we discuss on the Data Preprocessing part which mainly covers the Use of different processing methods to obtain clean data for our implementation.

Chapter Five- This Chapter will discuss on the models which we are implementing and the use of different

methods used.

Chapter Six- This Chapter will discuss on the model comparison and analysis of the models implemented. We will compare models using similarity scores to get the most optimized model.

Chapter Seven- This Chapter will discuss on the representation of the web application that we will be implemented through UI framework.

Chapter 2

Literature Review Summary

2.1 Introduction

Every firm is incorporating the recommender system. By recommending new products based on user interests, the company hopes to enhance user interaction and income. With their methods to integrate a recommender system into their current environment, many recommending system has become a dominant force in the entertainment industry. Yet, there hasn't been much research done on hiring practices from the viewpoint of job seekers. It is vital to evaluate pertinent work in the field and the technology before beginning any research.

2.2 Types of Recommender System

Recommendation System is the system that analyses user choice history and provides them with several service options based on the necessity, as was previously discussed. The mid-1990s saw the emergence of recommender systems as a separate field of study (Ricci et al., 2011). The demand for recommender systems has skyrocketed in recent years. There are four categories in the recommendation algorithm: content-based filtering, collaborative filtering, rule-based techniques, and hybrid approaches.

Collaborative Filtering (CF)

Collaborative filtering is a method that uses human ratings given to items by users to compare ratings from other users who have given comparable ratings to those items. In order to group users who share a common interest, the memory-based nearest neighbor method is the key operation performed here. There will be significant lag when creating recommendations as the volume of data gradually increases. Although content-based filtering techniques have an advantage over collaborative filtering, due to the nature of the hiring process, a job cannot be rated by the user and cannot be compared using a similarity matrix.

Content-based filtering (CBF)

The most subjective and descriptive kind of filtering are content-based filters (CBF). As it makes use of an item's expressly stated property, content-based filtering is often referred to as an attribute-based recommender system. It is a method for solving a machine learning or information retrieval challenge. In content-based filtering, it is assumed that users favor items with comparable features. The user is given recommendations for products using content-based filtering that have qualities in common with the item in which the user has previously expressed interest. According to previous recommendation system made, a problem of this filtering method is that it has a propensity to over-specialize when recommending an item to a user profile because user profiles are based on a characteristic of the prior item chosen by the user.

Due to the nature of the job domain, the tendency to over-specialize in proposing the same item would not be a concern in the job domain recommender system. Nevertheless, jobs published on job boards in the job domain are only available for a short period of time. The user's preferences tend to alter depending on a variety of circumstances in domains like entertainment, but in the employment domain, the user prefers to hunt for a position where he can apply his prior talents.

As a user's preferences change, or if the user decides to modify their job domain by upgrading their new skills and the job domain if they so want, new job recommendations may be provided. Another scenario of new recommendation is when new jobs are listed in the database; system would identify the properties of the job listed, such as job domain and skills required for the job and matches with the users with a high similarity score.

Rule-based Filtering (RBF)

These filtering methods rely on decision rules, such as automatic or human decision rules, which are tweaked to produce a suggestion for the user profile. The E-commerce sector currently use a rule-based filtering technique to suggest an item based on a user's purchasing history, demographic location, and other characteristics that can be used to profile a user. The fact that the user supplies the system with information is a downside of rule-based filtering. These inputs are used to describe a user profile or to reflect a user's preferences as they are expressed by the user. As a result, the data is subject to bias. Recommendations often reach saturation and stagnate as user profiles get older.

Hybrid filtering (HF)

As the term suggests, it incorporates several strategies to enhance the effectiveness of recommendations. The advice that was previously mentioned that each technique has its limitations and advantages. This strategy is desired in order to obtain a better recommendation and get beyond the difficulties presented by earlier methods. The cold-start problem affects all learning/model-based strategies in one way or another. A new user or new item handling is the issue at hand. The CF, CBF, and RBF have a number of drawbacks that could be improved by adopting hybrid filtering approaches.

Many have conducted surveys that found many hybrid filtering approaches that may be applied by combining CF, CBF, and RBF.

1. **Weighted:** To produce a more useful recommendation, the similarity scores from the various recommendation components are mathematically connected.
2. **Mixed:** A single recommendation is offered after recommendations from several recommending approaches have been combined.
3. **Changing:** picking one of the suggested components based on the situations in which it works best.

4. Feature Combination: Several knowledge sources' attributes are combined and sent to a recommendation system.
5. Feature Augmentation: One recommendation technique computes a set of user or object attributes that are then used as input for the following recommendation technique. To get on recommendation, two or more recommendation techniques are serialized.
6. Cascade: Systems that provide recommendations are given strict precedence, with the lower priority systems breaking ties amongst the higher priority ones in the scoring. Here, the recommendation of another Recommendation System approach is refined.

Some researchers had made an effort to create a recommendation system. Many have carried out one such implementation. A hybrid Recommendation System, for Job Finding search engine had been developed by them. In their employment recommendation system, they had put in place a personalised case retrieval module and an automatic collaborative filtering module.

The ACF module used data on user behaviour, such as read time and page activity during the user's time on the system, to profile the user. For comparable grouping users versus target users, clustering methods like the Jaccard index and other similarity measures were used. Their second module, PCR, determines whether the user's query and system jobs are similar. Using several similarity metrics, the module determines how closely a target user's query and jobs from the job case base are related. Problems with sparsity and scalability have plagued this system.

2.3 Natural language processing

The only type of data we deal with in Recommendation system for the hiring domain is text data. An individual's abilities and user experience are detailed in their user profile. On the other hand, the job listing includes details such as the job title and the qualifications needed to perform the task. All of this content is composed entirely of text. By comparing the similarities between the job title and work description of the listed position, we may use natural language processing to gauge how similar two jobs are. Text summarization, video tagging, and keyword search are only a few examples of industrial applications where determining text similarity is a crucial task.

In these circumstances, we use a method known as word embedding. This represents a significant advancement in distributional semantics. As this simply needs a significant volume of unlabeled word data. These terms have semantic representations as a vector. In other words, semantically comparable terms will remain close to one another in the semantic space. We can use the most well-known method called word2vec, a vector space model, to recover phrases that are based on the similarity between two terms, and then we can use cosine similarity to assess that similarity, this approach can also be used to determine how similar the sentences are.

Word embedding is created using a group-related paradigm, and these area group of NLP feature learning and language modelling algorithms where words are mapped to actual values in the vector. Usually, word2vec uses a huge set of words known as. Using a corpus as an input build a vector space

with hundreds of dimensions. Once a vector space model has been created, we can utilize a similarity measuring approach to gauge how closely a word compares to another. We can use similarity metrics like Cosine similarity and Jaccard similarity to detect similarity in vector space.

2.4 Inferences

The CF methodology cannot be taken into consideration based on all the research procedures and techniques covered in this chapter because it does not meet the objectives of the study. The dataset we won't be able to develop a rating matrix that calls for CF method if the user does not own the rating data for a specific job. I've opted to use content-based filtering instead.

We have built user profiles using a variety of user data factors and recommended jobs to those profiles with the highest cosine similarity scores. Also, when calculating the similarity scores between the user profile and the job, I have given job skills a higher priority than the user's job domain.

Chapter 3

Exploratory Data Analysis

3.1 Information about Datasets

For this project, we have used 4 datasets from open-source site Kaggle.

Combined jobs dataset – it contains information about the jobs.

Job views dataset – contains information about the users.

Experience dataset – gives details about the applicants Position name.

Position of interest dataset - contains information about the applicant's position of interest

3.1.1 Combined jobs dataset details

This dataset contains details about jobs. Total number of rows in dataset – 84090. The dataset contains following required columns of interest:

- Job.ID – Unique Id for Jobs
- Title – Title usually contains the position of the job and the respective company name.
- Position – Job position
- Company – The company where there is vacancy for that particular job
- City – The city where the company is present
- Employment type – The column contains the choice of employment type the applicant wants i.e. part-time or full-time.
- Job Description – It describes the role of the job having brief description.

For the columns City, Company, Employment type and Job description contains NULL values as those data can be entered by the employer.

3.1.2 Job Views dataset details

The dataset contains applicant details. Total number of rows in dataset – 9997

Following are the dataset columns/ features of our interest:

- Applicant.ID - Unique id for each applicant
- Job.ID - Specifies the unique Job Id.
- Position– Contains Job Position of respective user
- Company- The company where the applicant is working.
- City- The city where the company resides

For the column Company contain NULL values since the data entered depends on the user.

3.1.3 Experience dataset details

The dataset contains details about the applicant's id and for which position they are working. Total number of rows in dataset - 8654

Following are the dataset columns/ features of our interest:

- Applicant ID – Unique if for user. Also present in User dataset.
- Position Name- Unique book number of each book. Same as in Book dataset.

For the column Position Name Null values are present.

3.1.4 Position of Interest Dataset details

The dataset contains details about the applicant's position of interest. Total number of rows in dataset - 6561.

Following are the dataset columns/ features of our interest:

- Applicant ID – Unique if for user. Also, present Job Views dataset.
- Position of Interest – It contains for which position the users are most searching for.

For the column Position of Interest Null values are present.

3.2 Observations done on datasets

Data type for each dataset:

Column name	Data type
Job.ID	Integer
Title	String
Position	String
Company	String
City	String
Job description	String
Applicant id	Integer
Position Name	String
Position Of Interest	String

NULL values: Combined jobs dataset contains Null values in column in Company, City, employment type and job description. Job Views dataset has column Company with Null values. Experience dataset has column Position Name with Null values. Position of Interest has column Position of interest with Null Values.

Duplicate values: There are no duplicate values in any of the datasets used.

Garbage values in features- There are some unnecessary characters in Position column in combined job dataset.

3.3 Data Visualization

We have tried to implemented two types of the plotting for Visualizing Datasets.

3.3.1 Plotting null graph:

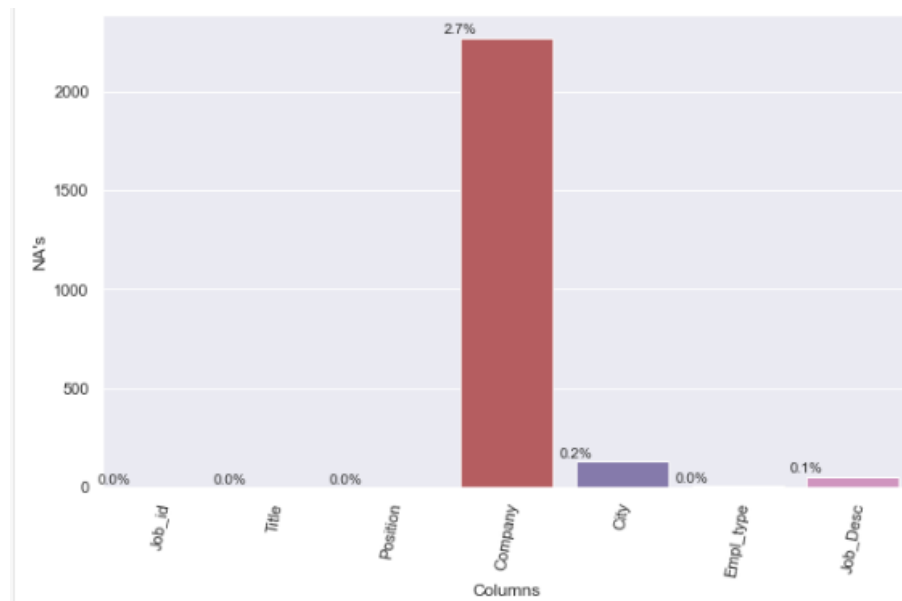


Fig 3.3.1 Null column Bar plot

3.3.2 Scatter and Density Plot

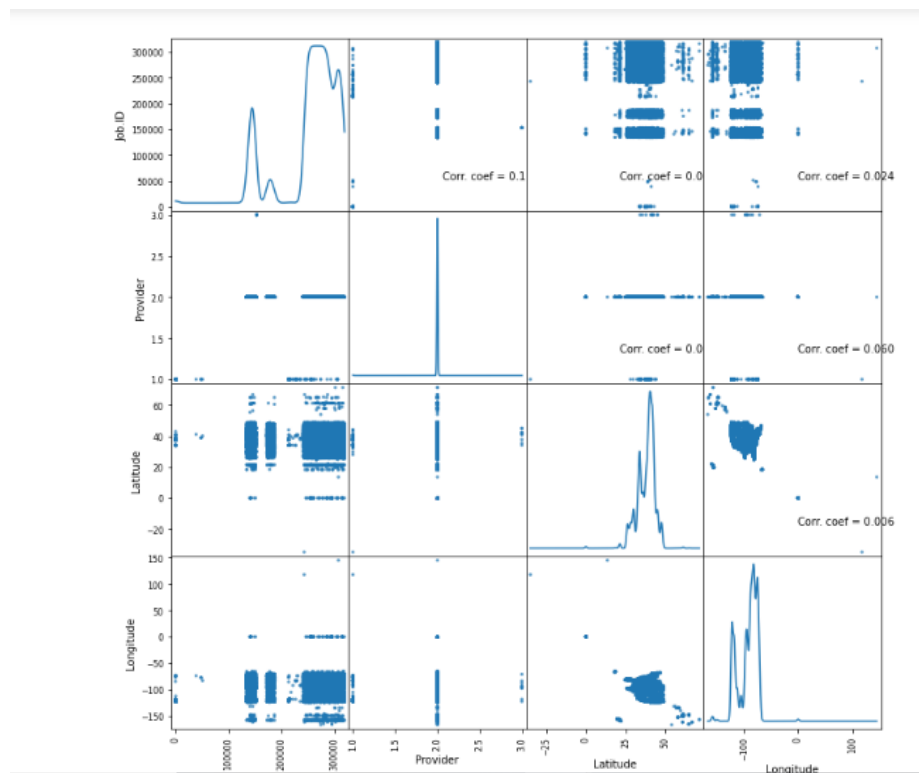


Fig 3.3.2.1 Scatter and Density Plot of Job Dataset

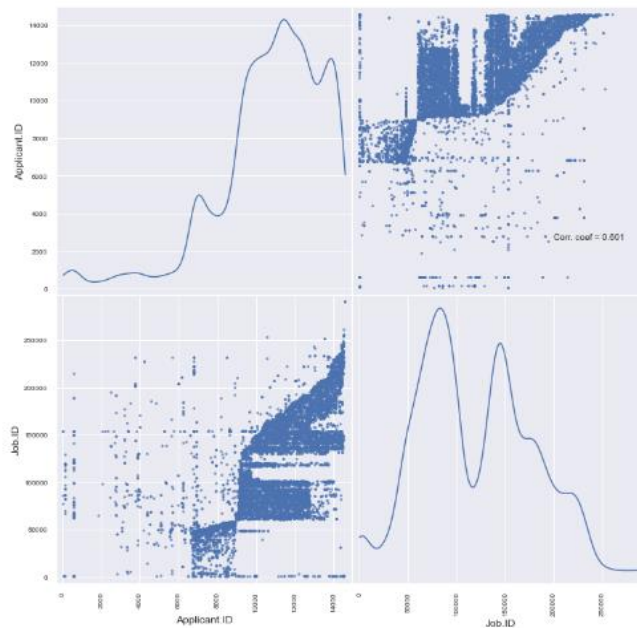


Fig 3.3.2.2 Scatter and Density Plot of User Dataset

3.3.3 Column Distribution Plot

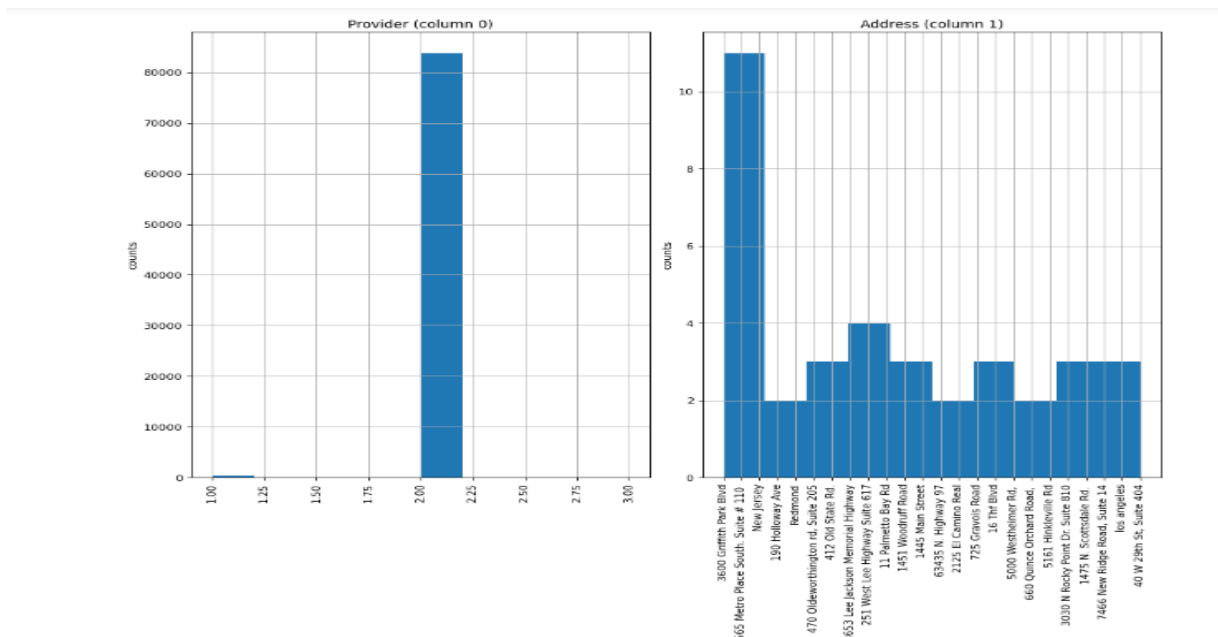


Fig 3.3.3 Column Distribution Plot using subplot graph

Chapter 4

Data Preprocessing

In this chapter, we will focus on Data Preprocessing which is the most important aspect of Data Arrangement. All the User, Job views, experience and position of interest datasets contains null values, irrelevant columns, column names containing keywords etc. In data preprocessing, our goal is to create a user and job data corpuses for easy search of words while searching jobs. In this application, data preprocessing we are dropping irrelevant columns, renaming column names having keywords, filling Nan columns through Imputation, combining related columns, applying WordNetLemmatizer, Tokenization, Stop words, and converting to lowercase, Fitting and transforming vectors for combined column and Merging Datasets

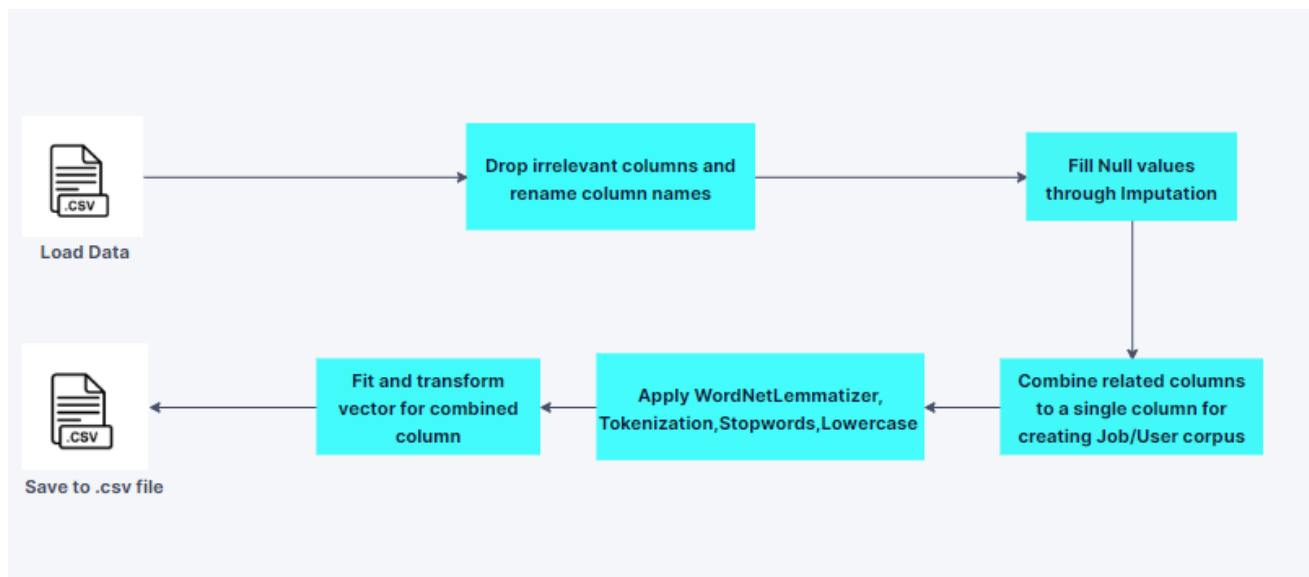


Fig 4- Data Preprocessing Flow for User/Job Corpus

Following are the steps used in Data Preprocessing

4.1 Dropping Columns

The datasets contain columns that are irrelevant to the research and have lot of missing values. The columns in our dataset like -

'Provider','Status','Slug','State.Name','State.Code','Address','Latitude','Longitude','Industry','Requirements','Salary','Listing.Start','Listing.End','Created.At','Updated.At','View.Start','View.End','View.Duration','Employer.Name','Can.Contact.Employer' doesn't have much relevance in the application. So, by omitting those columns from the datasets it allows us to concentrate on the remaining columns. We use **.drop()** method by Pandas to drop the columns.

4.2 Renaming column names

Certain column names present in the dataset are not depicted correctly like Position.Name, Employment.type ,Job.Description ,Position.Of.Interest . These column names have some special keywords like .Name, .type, .description etc which are used in Python for different functionalities and using those as column names might create confusion in extracting columns during various operation. So, those are renamed.

Example- [Employment.type]→[Empl_type].

4.3 Data Imputation

Dataset used in our context contains a lot of missing values on our required columns. So, we need to handle those by filling those values with the most appropriate values possible. We therefore use different imputation techniques for replacing categorical column like Position_Of_Interest , Position_Name , Job_desc by the use of Mode. This strategy means the “Most frequent”. It replaces missing values using the most frequent data used in the column. For some columns like City we are replacing nan with their Company headquarters location by grouping. And some columns like Position , we have used fillna(" "). It manages and let the user replace NaN values with some value of their own.

4.4 Combining Columns

Here we are using different datasets specifically designed for a particular purpose. From two datasets we are creating corpuses like job corpus and user corpus. So for creating the job corpus we are combining the required columns like position, company, city, emp_type and position and job_desc . And for User corpus we are combining the required columns like position, company, city. These combined columns helps in easy implementation of various data cleaning techniques like, Tokenization etc.

4.5 Lemmatization

For the English language, WordNet in NLTK offers a sizable, openly accessible lexical database with the goal of establishing organised semantic associations between words. It is one of the earliest and most widely used lemmatizers and also has lemmatization capabilities.

A WordNetLemmatizer () instance must be created, and the Lemmatize () function must be called on a single word in order to lemmatize it. Lemmatization is required in our datasets to transform each word with a distinct part of speech (POS) to its noun form in order to provide the thorough retrieval and extensive indexing needed for job searching.

Example-
[joining] →[join]

[assisting] →[assist]

[driving]→[drive]

4.6 Tokenization

Tokenization transforms unusable data strings into raw data. Tokenization breaks down phrases and paragraphs into more manageable chunks so that meaning may be ascribed quickly. Tokenization aids in determining the text's meaning by examining the word order. Here word tokenization is done as the combined column created have large string of words and each word give different meaning at different context . So having words as tokens help in identifying its true meaning while creating clean corpus.

Example-

["Part Time Service Agent Car Detailer Sarasota Bradenton"]

With tokenization

["Part" "Time" "Service" "Agent" "Car" "Detailer" "Sarasota" "Bradenton"]

4.7 Stop words

Getting rid of stop words entails getting rid of the terms that recur often across the corpus's documents. The algorithm just scans or selects those two features since it doesn't need words like "is" or "are," which don't award a score or points, for comparison. Pronouns and articles are typically categorised as stop words. These terms are not particularly useful because they have no meaning in some NLP tasks like information retrieval and categorization. Stop words are used as a result to get rid of all the unnecessary and meaningless terms.

Example:

[Intern in Quality Improvement of Department for Patient Satisfaction]

Without Stop words

[Intern Quality Improvement Department Patient Satisfaction]

4.8 Lowercase

We have tried to use lowercase word for the corpuses created. If there are no uppercase characters in the given string, it returns the original string and help in easy retrieval of words on searching and indexing. The .lower () method takes no arguments and returns the lowercased strings from the given string by converting each uppercase character to lowercase.

4.9 Fitting and transforming Data

We are transforming the collection of unprocessed documents into a matrix of TF-IDF characteristics using the tf-idf vectorizer. It is utilised to retrieve information. The word frequency-inverse documents is the complete version of the term "tf-idf". It is used to search through a corpus of words for the

pertinent or significant words in a document. All things are done utilising numerical statistics. Prior to calling the fit transform method on our test corpus, we must first create the class. The corpus is vectorized by the function to produce a sparse matrix.

The matrix's present shape, as determined by applying to the Job Corpus, is shown below.

```
In [40]: #initializing tfidf vectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
tfidf_vectorizer = TfidfVectorizer()

tfidf_jobid = tfidf_vectorizer.fit_transform((df_combinedJobs['combinedCols'])) #fitting and transforming the vector
tfidf_jobid

Out[40]: <84090x51497 sparse matrix of type '<class 'numpy.float64'>'
        with 8493157 stored elements in Compressed Sparse Row format>
```

4.10 Merging datasets

Datasets are merged to filter out the most optimized columns to be used for searching and recommending. Here we are merging Job and experience dataset are merged by using Applicant.ID. Then Position of Interest dataset are merged with dataset created from above using Applicant.ID. Then we are combining all the columns containing important columns in our interest i.e Applicant_ID, Position_Company_City(combined single column), Position Name, Position of Interest and text column which contains the job profile related information.

```
In [81]: df_jobs_exp_poi["textColumns"] = df_jobs_exp_poi["positionCompanyCity"].map(str) + df_jobs_exp_poi["Position_Name"] + " " + df_j
df_jobs_exp_poi.head()
```

```
Out[81]:
```

	Applicant.ID	positionCompanyCity	Position_Name	Position_Of_Interest	textColumns
0	2		volunteer writer uloop blog		volunteer writer uloop blog
1	3		market intern server prep cook		market intern server prep cook
2	6		project assistant		project assistant
3	8		deli clerk server cashier food prep order taker		deli clerk server cashier food prep order tak...
4	11		cashier		cashier

After the Preprocessing steps, we obtain the field combined jobs having all the texts used for searching jobs using Applicant ID.

```
In [109]: df_selectJob= df_allUsersExpInterest[['applicant_id','combinedJobs']]
df_selectJob.head()
```

```
Out[109]:
```

	applicant_id	combinedJobs
0	2	volunteer writer uloop blog
1	3	market intern server prep cook
2	6	project assistant
3	8	deli clerk server cashier food prep order tak...
4	11	cashier

Chapter 5

Modeling of Recommender System

In the modelling stage of the study, we made the decision to move forward and develop a recommender system utilizing a collaborative-based recommender system based on our literature review. Applying this supposition to the hiring industry, a job seeker would look for a new position that required talents related to his current title and skill set. In this kind of filtering, a user profile vector and an item profile vector are expected to be used to contain all user and object-related data. The score of the job that is relevant to the user's preferences is then determined using similarity measures like the cosine similarity measure.

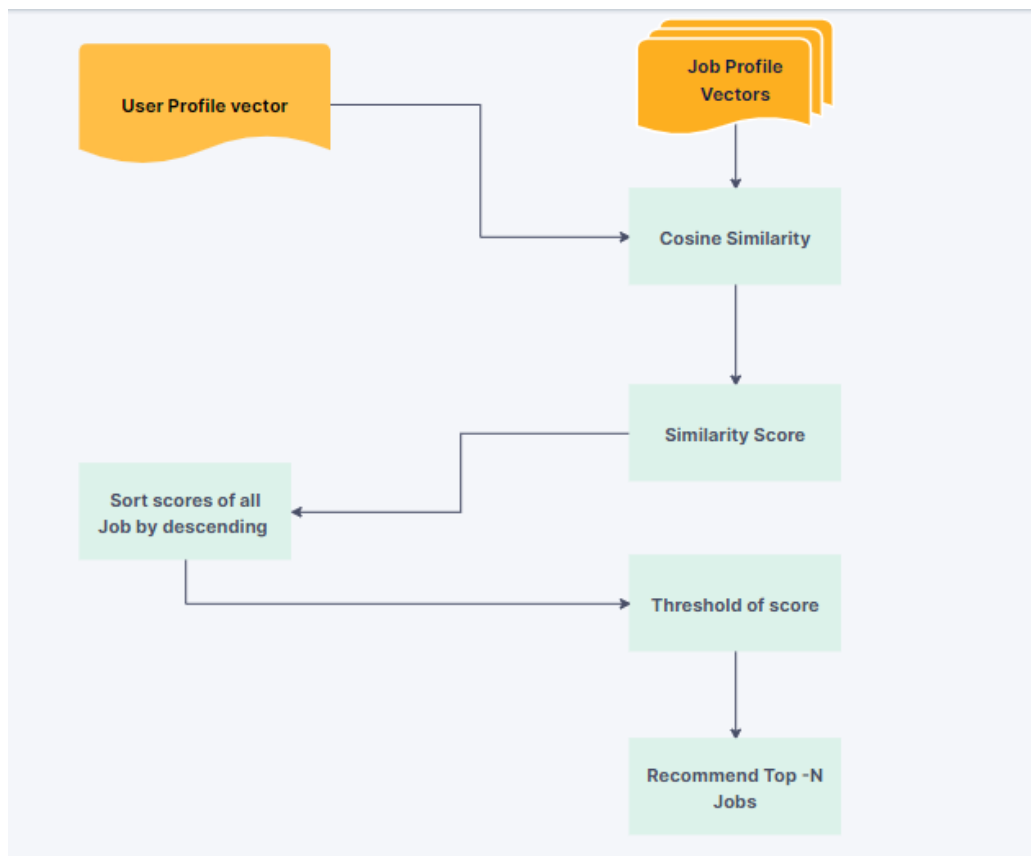


Fig 5 - Process flow - Recommender system

5.1 Similarity Function Module

The generated combined dataset constitutes the Similarity Function Module. During this procedure, the machine chooses cosine similarity with tf-idf vectorizer. The combined dataset is used in the Cosine Similarity function, which also lemmatizes and tokenizes the data and removes stop words (which waalso used to analyse the job description).

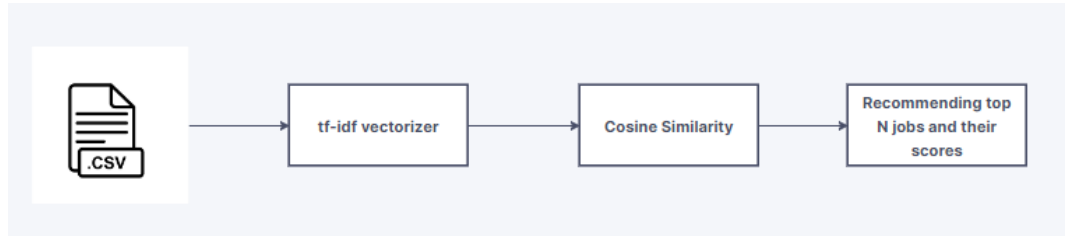


Fig 5.1 - Similarity Function Module

TF-IDF, or term frequency-inverse document frequency, should be used. For the processed datasets, Tf-idf was used. The raw text already has a matrix of values, which is what we are given. The tf-idf values a term more highly the more frequently it appears in a document. It is balanced by the number of documents in the corpus that contain the word to take into consideration that a few terms appear more frequently overall.

$$\text{tf-idf} = \text{tf} \times \text{idf}$$

$$\text{idf}(t) = \log \frac{n+1}{\text{df}(d,t)+1} + 1$$

The cosine similarity formula can be utilised after getting a matrix value system. Scores show how well the job description and the user preferences match. The formula is the sum of the vectors' dot products divided by the sum of the vector products divided by the vector products of lengths. We can alter the scores' default display order, which is decreasing. so that the user may easily choose the top jobs and concentrate on them. To compare the similarity between the text entered by the applicant, we employ cosine similarity.

A measure called cosine similarity is used to assess how similar two papers are, regardless of their sizes. The angle that two vectors make when they are projected into a multidimensional space is quantified mathematically as cosine similarity. I'm talking about two arrays of two vectors, each of which has the word counts of two distinct publications. How is the cosine similarity measure of similarity different from the number of shared terms? When plotted on a multi-dimensional space, each dimension representing a word in the document, the cosine similarity captures the orientation (the angle) but not the amplitude of the texts.

$$\text{Cos}\theta = \frac{\vec{a} \cdot \vec{b}}{\|\vec{a}\| \|\vec{b}\|} = \frac{\sum_1^n a_i b_i}{\sqrt{\sum_1^n a_i^2} \sqrt{\sum_1^n b_i^2}}$$

where, $\vec{a} \cdot \vec{b} = \sum_1^n a_i b_i = a_1 b_1 + a_2 b_2 + \dots + a_n b_n$ is the dot product of the two vectors.

5.2 Model Implementation

We are not utilizing further matrix decomposition methods, such as SVD, or correlation coefficient-based methods, such as Pearson's R correlation because this application has more textual data and there are no ratings provided for any jobs. Hence, content-based filtering will only show us how to suggest products to customers based solely on those products' own attributes.

As part of this project, we are developing four different types of recommender Systems-

1. Recommender system using with TF-IDF
2. Recommender system with CountVectorizer
3. Recommender system with KNN
4. Recommender system with Spacy

5.2.1 Model I – Using tf-idf vectorizer and Cosine Similarity Measure

For Model I, we are utilising the tf-idf vectorizer and Cosine Similarity Measure to build the model and identify the top N job recommendations by using user preferences as input and providing output that suggests the best suited occupations based on the scores. The user's profile vector for that specific Applicant ID is loaded when the user submits his Applicant ID. The next step is to enter a for-loop to compute the cosine similarity measure between each job profile vector and the selected user profile vector. At the end of each loop, the similarity score is calculated and recorded in the list type. after calculating the cosine similarity between a user skill and job skill vector.

$$Job\ Score(U, J) = (0.6 * \sum_{i=1}^n Sim\ Dist_{skill}(U, J_i)) + (0.4 * \sum_{i=1}^n Sim\ Dist_{domain}(U, J_i)) \quad (4.1)$$

Where,

n = Total number of Jobs

U = Profile Vector of User

J = Profile Vector of Job

$Sim\ Dist(U, J_i)$ = Cosine distance equation as shown in equation 4.2

$$Similarity\ distance(A, B) = 1 - \cos(\theta) = 1 - \frac{\sum_{i=1}^n A_i * B_i}{\sqrt{\sum_{i=1}^n A_i^2} * \sqrt{\sum_{i=1}^n B_i^2}} \quad (4.2)$$

The list that results from the For-last Loop's step is sorted by job similarity value in descending order and contains all of the job similarity scores for a given Applicant ID. After obtaining the Job Similarity Score, we can recommend the User using the Jobs Top-N technique, i.e. recommend Jobs with the highest scores.

5.2.2 Model II - Using count vectorizer and Cosine Similarity Measure

Using user preferences as input and showing output that suggests the best suited tasks based on the scores, Model II uses count vectorizer and CS Measure to create the model and determine the top N job recommendations. The "count vectorizer" function, which gets the data ready for the vector representation, fits and transforms the data.

The "count vectorizer" function, when applied to text data, produces a matrix containing the word counts for each word. The aggregated columns are fit transformed using CountVectorizer(). Next, we use "cosine similarity" to identify similarities. The cosine angle between two vectors in a multi-dimensional space is measured in this dynamic method of determining similarity, In this method, it doesn't matter how big the documents are. The papers' cosine angles may be similar despite their Euclidean distance being far apart.

A collection of text documents is transformed into a vector of term/token counts using Scikit-CountVectorizer. Also, it makes it possible to pre-process text data before creating the vector representation. It is a very flexible feature representation module for text because of this functionality.

CountVectorizer converts a collection of text documents to a matrix of token counts: the occurrences of tokens in each document. This implementation produces a sparse representation of the counts.

```
vectorizer = CountVectorizer(analyzer='word', ngram_range=(1, 1))
vectorized = vectorizer.fit_transform(corpus)
pd.DataFrame(vectorized.toarray(),
             index=['sentence '+str(i)
                   for i in range(1, 1+len(corpus))],
             columns=vectorizer.get_feature_names())
```

	1000	be	for	great	greatest	is	it	lasagna	life	love	loved	of	the	thing	times	to
sentence 1	0	0	0	0	1	1	0	0	1	1	0	1	1	1	0	0
sentence 2	0	1	0	2	0	1	1	0	0	1	1	0	0	0	0	1
sentence 3	0	0	0	0	1	1	0	0	0	1	0	0	1	1	0	0
sentence 4	1	0	1	0	0	0	0	1	0	1	0	0	0	0	1	0

Fig 5.2.2 – Sample representation of Count Vectorizer

We have a lot of distinct terms in the text; therefore, we constructed a lot of different columns, each one representing a distinct word in the matrix. The word count is shown in the row. Due to the fact that the terms "assisting" and "joining" were repeated twice, we counted them as 2 and 1 respectively. By default, the Count vectorizer utilises word-level tokenization and lowercases the text.

The full vocabulary's length and an integer count of how many times each word appeared in the text are

both included in an encoded vector that is returned. In order to look at and better understand what is happening, you can translate the sparse vectors provided by a call to `transform()` back to NumPy arrays by using the `toarray()` function. When this prints, the array version of the encoded sparse vector only displays the one instance of the vocabulary word and completely ignores the other word that is not in the vocabulary. The machine learning algorithm can then use the encoded vectors directly. After sorting the assortment of outputs and their resulting scores, the top Suggested jobs were found. With the Job ID and Applicant ID, we can filter the jobs.

5.2.3 Model III -Recommender System using KNN

The documents with a specific format module automatically remove the control characters that might have a negative impact on the result of classification. The main module in the framework is the KNN & TF-IDF module. The module contains the main methods for classification and determination of the document weight value. The results of the classification forward to the last module for the measurement and presentation statistical indicators. The last module allows displaying the classification results and their simple statistical analysis. As previously mentioned, the algorithm is based on the machine learning. Preprocessing and document preparation is followed by the learning phase. The algorithm determines the basic documents which will be comparing with each new document. Algorithm checks where a document is categorized by only looking at the training documents that are most similar to it

TF-IDF refers to Term Frequency-Inverse Document Frequency. TF is simply the frequency a word appears in a document. IDF is the inverse of the document frequency in the whole corpus of documents. The idea behind the TF-IDF is to dampen the effect of high-frequency words in determining the importance of an item (document). KNN refers to K Nearest Neighbor and is a simple algorithm that, in our case, classifies data based on a similarity measure. In other words, it selects the “nearest” matches for our user input.

We used the default tokenizer in the Tfidf Vectorizer to tokenize the features, but if we wanted to extract features from a large text description, we would likely use a custom tokenizer. We then instantiate the vectorizer. The features are then vectorized into a document-term matrix (dtm). The dtm is a matrix with each feature in a column and a “count” in each row. The count is the tf-idf value.

A Nearest Neighbors model is then instantiated and the dtm is fit on the model. This is our recommendation model used to select the top 10 recommendations. The variable is then vectorized and run through the model. The results are returned in a tuple of arrays. In order to obtain the values, the 2nd tuple is accessed, then the 1st array then the 1st value — results [0][0][1:].

```
NNs[0][0][1:]
```

```
array([0.71988065, 0.72524925, 0.81034882, 0.84257062, 0.86541138,  
       0.90300462, 0.96929812, 0.98940985, 1.          , 1.          ])
```

Fig 5.2.3: Array representation of Nearest Neighbors after applying KNN

With this model, we are using keywords or job titles for similarity scores from the jobs. Having `n_neighbors` value 11, we are able to fetch top 10 results showcasing the most optimized scores.

5.2.4 Model IV - Recommender System using spaCy

Making computers capable of comprehending human language is the aim of the natural language processing (NLP) branch of artificial intelligence. NLP includes the study of, measurement of, comprehension of, and extrapolation of meaning from natural languages. Python's open-source NLP package, spaCy, was created in Cython. SpaCy was designed to make it simple to build information extraction or general-purpose natural language processing systems. There are different spaCy models available for various languages. We are using the `en-core-web-sm` as the default model for the English language.

The numerous tasks utilised in NLP projects, such as tokenization, lemmatization, part-of-speech (POS) tagging, entity recognition, dependency parsing, sentence recognition, word-to-vector conversions, and other text cleaning and normalisation methods, are all handled by spaCy in one place. We will read the same text from a file and utilise spaCy to break down a given input string. Here we are creating a Doc object and then processing our data. A lexical token is represented by a series of Token objects called Doc objects. Each Token object contains details on a specific textual component, usually a single word. By invoking the Language object and passing it the input string as a parameter, we can create a Doc object. A Doc object is created using the text.

Word embedding is the method to translate the words or phrases from the corpus to vectors of real number as shown in figure 3.2 . It's used to language modeling and feature learning methods as-well. The corpus here is taken from the spaCy's one of the largest word model i.e., `en-core-web-sm`. Word2vec word embedding algorithm takes large corpus `en-core-web-sm` of text as input and produces a word vector space, which is in several hundred dimensions. Below is the figure 5.4 which vector output for the word "BANANA" taken from spaCy.

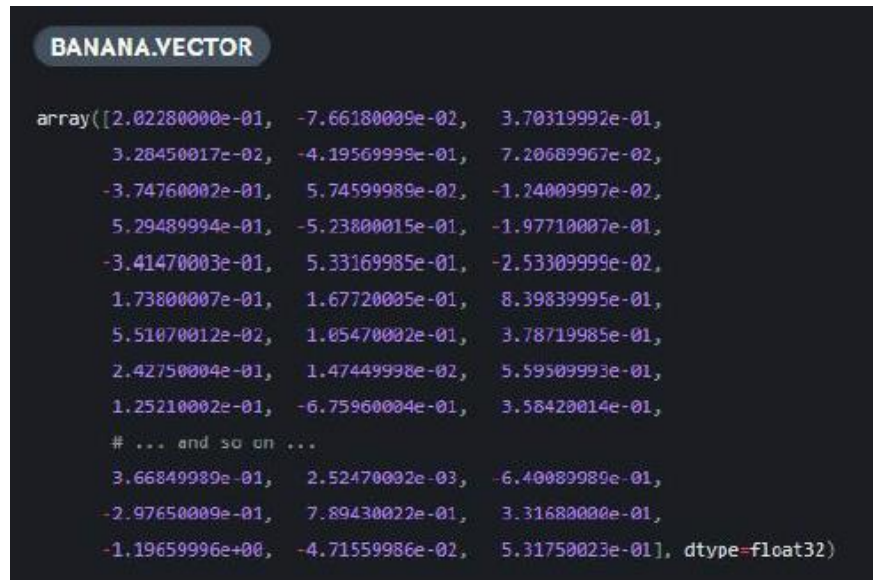


Fig. 5.2.4 Sample Vector for the word "BANANA"

source: <https://spacy.io/usage/vectors-similarity>

The job description and other necessary input fields that a job seeker will search for are extracted from processed text data using spaCy in the current study. It is also used to determine whether two user profiles are identical. We may compare two word vectors using the similarity function in the spaCy package to determine how similar two words are to one another. The word vector used in this similarity function is a multidimensional representation of the word. With the word embedding technique, these word vectors were produced.

```

def calculateSimWithSpaCy(nlp, df, user_text, n=6):
# Calculate similarity using spaCy
    list_sim = []
    doc1 = nlp("select_user" + user_text + "")
    for i in df.index:
        try:
            doc2 = list_docs[i][0]
            score = doc1.similarity(doc2)
            list_sim.append((doc1, doc2, list_docs[i][1], score))
        except:
            continue
    return list_sim

```

To determine how similar the vectors are, the aforementioned function is used. If an applicant looks for a certain job by providing a description, it recommends positions that other candidates have previously searched for that are comparable.

Chapter 6

Model Comparison and Analysis

To assess and choose the best optimised recommendation model for the job recommendation system, we used 4 alternative models in this case. As we are using text data to develop the recommendation system, there is no established testing matrix for calculating the accuracy score. Instead, we use TF-IDF, spacy, count vectorizer, and cosine similarity. Our recommendations scores has been manually verified.

6.1 Model Analysis

Upon the implementation of the model using cosine similarity and KNN as the evaluating matrix, we could obtain the top 10 results displaying the ApplicantId , JobId ,corresponding combined text field(title) which contains the data of the jobs the user have searched. The scores(score) represent the scores how the job corpus related to same job type that are interlinked according to the Applicant's choices.

6.1.1 Model I - Job recommendation model using TF-IDF and Cosine Similarity

We can see that Model I has a similarity score ranging from 0.48 to 0.74 in the screenshot below. As a result of Applicant Id 326's job search, we can recommend positions that are similar to Java Developers.

	ApplicantID	JobID	title	score
0	326	303112	Java Developer @ TransHire	0.749478
1	326	294684	Java Developer @ Kavaliro	0.740886
2	326	269922	Entry Level Java Developer / Jr. Java Develop...	0.737007
3	326	141831	Lead Java/J2EE Developer - Contract to Hire @ ...	0.671667
4	326	270171	Senior Java Developer - Contract to Hire - Gre...	0.645037
5	326	305264	Sr. Java Developer @ Paladin Consulting Inc	0.625532
6	326	309945	Java Software Engineer @ iTech Solutions, Inc.	0.592291
7	326	245753	Java Administrator @ ConsultNet	0.530231
8	326	146640	Jr. Java Developer @ Paladin Consulting Inc	0.510534
9	326	150882	Java Consultant - Mobile Apps Development @ Co...	0.486789

Fig 6.1.1- Similarity Score of Job Recommendation Model using TF-IDF and Cosine Similarity

6.1.2 Model II- Job recommendation model using Count Vectorizer and Cosine Similarity

The screenshot below shows that Model II has a similarity score ranging from 0.32 to 0.63. We can find recommendations for jobs that are comparable to Java Developers based on Applicant Id 326's search.

	ApplicantID	JobID	title	score
0	326	303112	Java Developer @ TransHire	0.635001
1	326	294684	Java Developer @ Kavaliro	0.600245
2	326	269922	Entry Level Java Developer / Jr. Java Develop...	0.571726
3	326	141831	Lead Java/J2EE Developer - Contract to Hire @ ...	0.496907
4	326	270171	Senior Java Developer - Contract to Hire - Gre...	0.481757
5	326	309945	Java Software Engineer @ iTech Solutions, Inc.	0.454673
6	326	305264	Sr. Java Developer @ Paladin Consulting Inc	0.406017
7	326	245753	Java Administrator @ ConsultNet	0.378968
8	326	150882	Java Consultant - Mobile Apps Development @ Co...	0.363216
9	326	146640	Jr. Java Developer @ Paladin Consulting Inc	0.323381

Fig 6.1.2- Similarity Score of Job Recommendation Model using Count Vectorizer and Cosine Similarity

6.1.3 Model III- Job recommendation model using TF-IDF and KNN

The screenshot below shows that Model III displays a similarity score between 0.71 and 1. According to Applicant Id 326's search, we can recommend jobs that are related to Java Developers.

	ApplicantID	JobID	title	score
0	326	294684	Java Developer @ Kavaliro	0.719881
1	326	269922	Entry Level Java Developer / Jr. Java Develop...	0.725249
2	326	141831	Lead Java/J2EE Developer - Contract to Hire @ ...	0.810349
3	326	270171	Senior Java Developer - Contract to Hire - Gre...	0.842571
4	326	305264	Sr. Java Developer @ Paladin Consulting Inc	0.865411
5	326	309945	Java Software Engineer @ iTech Solutions, Inc.	0.903005
6	326	245753	Java Administrator @ ConsultNet	0.969298
7	326	146640	Jr. Java Developer @ Paladin Consulting Inc	0.98941
8	326	253863	Resident Assistant	1
9	326	246058	Dietary Aide	1

Fig 6.1.3- Similarity Score of Job Recommendation Model using TF-IDF and KNN

6.1.4 Model IV- Job recommendation model using TF-IDF and spaCy

The screenshot below shows that Model IV has a similarity score ranging from 0.61 to 0.69. We are able to receive recommendations for positions as "Java developers" here, but in the majority of cases, we also have "Drupal developer" available.

	ApplicantID	JobID	title	score
0	326	250216	Microstrategy Developer @ Kavaliro	0.698317
1	326	294489	Magento Developer (ONSITE) @ Creative Circle	0.654219
2	326	257251	Front End Developer @ ConsultNet	0.63
3	326	294684	Java Developer @ Kavaliro	0.622253
4	326	257437	Drupal Developer-offsite @ Creative Circle	0.617237
5	326	316365	Jr. Ruby on Rails Developer @ ConsultNet	0.617211
6	326	257439	Drupal Developer-offsite @ Creative Circle	0.612062
7	326	257438	Drupal Developer-offsite @ Creative Circle	0.611012
8	326	257440	Drupal Developer-offsite @ Creative Circle	0.610952
9	326	303112	Java Developer @ TransHire	0.610007

Fig 6.1.4- Similarity Score of Job Recommendation Model using TF-IDF and spaCy

6.2 Model Comparison

Four models were compared using four distinct methodologies, and we found that Model III, a recommender system using TF-IDF vectorizer and KNN, exhibits the most optimal output. The top 10 Recommendation Approach similarity scores for Applicant ID 326, the Job ID with values 245753 and 146640, are very near to 1. As a result, we can say that the "Role of Java Developer" is the position that candidates with the same interests as Applicant ID 326 are most likely to be recommended for.

Model IV, a recommender system using TF-IDF and spaCy, yields the most incorrect results of all since it frequently suggests Drupal developers (Fig. 6.1.4) above Java developers.

The second-most optimal outcome is displayed by Model II, a recommender system that uses a TF-IDF vectorizer and cosine similarity. A candidate for a job recommendation can also be seen as having a similarity score of at least 0.60 and receiving the title of Java Developer in most circumstances.

Chapter 7

Application Deployment

In this phase, we interact with the final part of recommender system in this final stage i.e full website deployment using Gradio toolkit. The presentation layer, which is the last part of the design, is where we send requests for recommendations and receive the top 10 suggestions for a given request. This study employs a full deployment design, with the first part serving as our data layer and housing all of the information pertaining to user profiles and item profiles. The presentation layer communicates with the second part to process a user request for a recommendation.

7.1 Gradio

Gradio is a python open-source toolkit that was introduced in February 2019 for the creation of simple user interfaces that can be customised for any deep learning or machine learning model. Just a few lines of code can be used to quickly create a Gradio user interface. Any Python function can be used with Gradio to provide a straightforward user interface. Everything from a straightforward tax calculator to a deep learning model might be that function.

There are three parameters in Gradio:

1. fn: a function that handles the user interface's primary task
2. inputs: the input component type
3. outputs: the output component type

It complies with the most fundamental criterion for a user interface designed for a deep learning model, which is the support for numerous inputs and outputs. Gradio may also be employed to do comparison analyses of various models. Even for novice coders, Gradio offers a more straightforward and user-friendly user interface. Gradio's expanded scope includes more customizations that may be made as well as an improved user interface.

7.2 UI Implementation

We are using the three parameters in Gradio for the implementation of the Web Application.

Input: The input parameter will contain the required fields; the applicant will like to search for job recommendation. In our case, we are using 4 main fields which are Position of Interest, Job Description, Location and Company Name as inputs.

Output: The output will show the recommended jobs based on the search text of the Applicant in the input section

fn : This is the function that is called in order to retrieve the jobs that are suggested depending on the criteria the applicant has specified in the input section.

```
In [161]: #create input and output objects
#input object

inputPOS = gr.inputs.Textbox(placeholder="Enter Position Of Interest",label="Position of Interest")
inputJobDescription= gr.inputs.Textbox(placeholder="Enter Job Description",label="Job Description")
inputCompany = gr.inputs.Textbox(placeholder="Enter Company",label="Company")
inputLocation = gr.inputs.Textbox(placeholder="Enter Location",label="Location")
input = gr.inputs.Textbox(placeholder="Enter Position Of Interest")

#output object
output = gr.inputs.Textbox(placeholder="Recommendations",label="Jobs Recommended")
```

Fig 7.2.1 – Code snippet showcasing the input and output section for our Web application.

```
In [162]: interface=gr.Interface(fn= gradioUI,
                                inputs=[inputPOS, inputCompany,inputLocation,inputJobDescription],
                                outputs=[output])
```

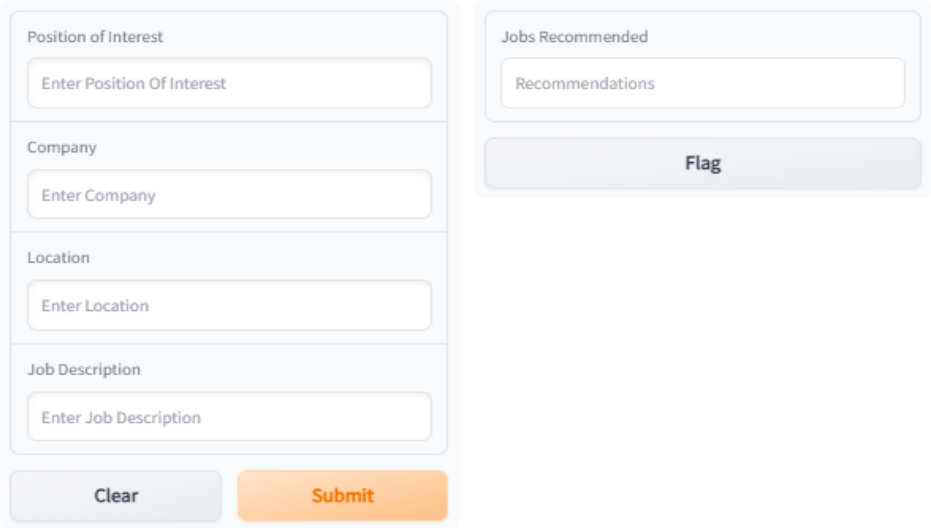
Fig 7.2.2 – Code snippet showcasing the parameters passed through the interface.

The gradio interface must be launched after the input, output, and function parameters have been passed. Upon the interface's launch, a link to a local URL appears, and we can also see the Jupyter Notebook's input and output area, as seen in the below image.

```
In [163]: interface.launch()

Running on local URL: http://127.0.0.1:7863/

To create a public link, set `share=True` in `launch()`.
```



```
Out[163]: (<gradio.routes.App at 0x19101b0e130>, 'http://127.0.0.1:7863/', None)
```

Fig 7.2.3 – UI Implementation

Conclusion

The collaborative-based Filtering based method was developed on the basis of this study, a variety of research techniques, and after the implementation of algorithms for better performance and overall factors. Of course, in addition to collaborative-based algorithms, there needs to be a lot of progress and the use of hybrid algorithms. The sparsity of the user profile is kept under control, and various techniques for filling the user preference matrix can be used, in order to further optimize the recommendation system and integrate the system for better performance. The suggestions based on cosine similarity appear to produce the greatest outcomes in this particular situation out of the collaborative-based recommender systems that we constructed in this post. Ultimately, we've seen that even a very basic collaborative-based recommender may deliver respectable outcomes.

Since we just need the most basic information about a user collaborative-based recommenders clearly have the advantage of not having the cold-start issue. As a result, we draw the conclusion that a job recommendation system that analyses the job description and suggests a job based on the abilities and preferences of the user is a valuable model for a recommendation system that helps job searchers find open positions. As a result, out of all the threshold and filtering strategies, we decided to construct the recommender system using collaborative-based filtering, which achieves a similarity score that is near to 1, making it the most effective recommendation model in our situation. Thus, we can conclude that the optimal model for recommending jobs in this case can be thought of as Model III, i.e Recommender System using KNN and the use of Nearest neighbor search (NNS) can be identified as the most optimized solution for finding the point in a given set that is closest (or most similar) to a given point.

Direction for Future Work

Based on the results of the current study, the recommendation system uses cosine similarity as a measure of similarity along with word embedding from the TF-IDF as a collaborative-based filter. It is feasible to produce a better recommendation by developing a corpus relevant to the IT skills, terminology, job domain, and industry jargon because the corpus gives general information about the word and terms nearby. When analysing implicit text data in the job description, the recommendation may be improved by employing such a corpus that is specific to the hiring domain. It can be put into many categories. Research needs to be done on the data that has previously interacted in the hiring domain because the recommendation system is now operating on data that does not interact.

This would enable us to continuously recommend new jobs based on how the user's tastes evolve. LinkedIn offers a recommender system in the employment space, but it's from the perspective of the recruiter rather than the job seeker. Similar to this, we might carry out research utilising LinkedIn data to suggest positions using collaborative-based filtering. It is not a good idea to propose a job just because a user liked it because circumstances vary by domain; rather, the recommendation must be taken into consideration if the user's profile fits the criterion.

As for the future work, we aim to build a more efficient model and find out the best evaluating matrix for the Job recommendation system. Here the classic framework for displaying the recommendation results is in the form a single sorted list which is very restrictive and highlighting of multiple criteria is not present like we are not evaluating on the basis of job seeker qualifications or prior experience in other companies. So, conducting more study based on collaborative-based filtering ensemble with other filtering technique in hiring domain in the perspective of a job seekers. In order to provide job searchers with a more reliable recommendation list, we can also construct taxonomies for the job domain. To address the current flaw in our proposed model, we can additionally expand our effort by integrating the resume of new users.

Bibliography/ References

1. Wenxing Hong; Siting Zheng; Huan Wang, 2013 8th International Conference on Computer Science & Education, 2013, IEEE, "Dynamic user profile-based job recommender system".
2. S.R. Rimitha; Veda Samhitha Abburi; Annem Kiranmai; Marimuthu C; K. Chandrasekaran 2019 Fifteenth International Conference on Information Processing (ICINPRO), 2019, IEEE, "Improving Job Recommendation Using User Profiles and Ontological Models"
3. Vijay Yadav; Ujjwal Gewali; Suman Khatri; Shree Ram Rauniyar; Aman Shakya, 2019 Artificial Intelligence for Transforming Business and Society (AITB), 2019, IEEE, "Smart Job Recruitment Automation: Linking Business and Academic Communities".
4. Kevin Appadoo; "Job Recommendation using Machine Learning and Recommendation Engine"
5. Muhammad Bilaal Soonnoo; Zahra Mungloo-Dilmohamud, 2020 IEEE Asia-Pacific Conference on Computer Science and Data Engineering (CSDE), 2021, IEEE.
6. Amber Nigam; Aakash Roy; Hartaran, "Employment Recommendation through Job Selection Progression"
7. Pazzani M J, Billsus D. Content-based recommendation systems [M]//The adaptive web. Springer Berlin Heidelberg, 2007: 325-341.
8. Schafer J B, Frankowski D, Herlocker J, et al. Collaborative filtering recommender systems [M]//The adaptive web. Springer Berlin Heidelberg, 2007: 291-324.
9. Sarwar B, Karypis G, Konstan J, et al. Item-based collaborative filtering recommendation algorithms [C]//Proceedings of the 10th international conference on World Wide Web. ACM, 2001: 285-295.
10. Gayatri Khanvilkar; Deepali Vora, 2019 International Conference on Nascent Technologies in Engineering (ICNTE), 2019, IEEE, "Smart Recommendation System Based on Product Reviews Using Random Forest",
11. Khushbu Jalan; Kiran Gawande, 2017 International Conference on Energy, Communication, Data Analytics and Soft Computing (ICECDS), 2017, IEEE, "Context-aware hotel recommendation system based on hybrid approach to mitigate cold-start-problem",
12. Al-Otaibi, S.T. and Ykhlef, M. (2012) Job recommendation systems for enhancing recruitment process in: Proceedings of the International Conference on Information and Knowledge Engineering (IKE) p. 1

13. Barrón-Cedeno, A., Eiselt, A. and Rosso, P. (2009) Monolingual text similarity measures: A comparison of models over wikipedia articles revisions ICON 2009, pp. 29–38
14. Barzilay, R. and Elhadad, N. (2003) Sentence alignment for monolingual comparable corpora in: Proceedings of the 2003 conference on Empirical methods in natural language processing pp. 25–32 Association for Computational Linguistics
15. Burke, R. (2007) Hybrid web recommender systems in: The adaptive web pp. 377–408 Springer
16. Dhameliya, J. and Desai, N. (2019) Job recommender systems: A survey in: 2019 Innovations in Power and Advanced Computing Technologies (i-PACT) vol. 1 pp. 1–5 IEEE
17. Herlocker, J.L., Konstan, J.A., Borchers, A. and Riedl, J. (1999) An algorithmic framework for performing collaborative filtering in: 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 1999 pp. 230–237 Association for Computing Machinery, Inc
18. Herremans, D. and Chuan, C.H. (2017) Modeling musical context with word2vec arXiv preprint arXiv:1706.09088
19. Hu, R. and Pu, P. (2011) Enhancing collaborative filtering systems with personality information in: Proceedings of the fifth ACM conference on Recommender systems pp. 197–204
20. Jain, H. and Kakkar, M. (2019) Job recommendation system based on machine learning and data mining techniques using restful api and android ide in: 2019 9th International Conference on Cloud Computing, Data Science & Engineering (Confluence) pp. 416–421 IEEE References 52
21. Jijkoun, V., de Rijke, M. et al. (2005) Recognizing textual entailment using lexical similarity in: Proceedings of the PASCAL Challenges Workshop on Recognising Textual Entailment pp. 73–76 Citeseer
22. Kakarla, S. (2019) Natural language processing: Nltk vs spacy
23. Kenter, T. and De Rijke, M. (2015) Short text similarity with word embeddings in: Proceedings of the 24th ACM international on conference on information and knowledge management pp. 1411–1420
24. Luk, K. (2019) Introduction to two approaches of content-based recommendation system McAlone, N. (2016) Why netflix thinks its personalised recommendation engine is worth \$1 billion per year
25. Mihalcea, R., Corley, C., Strapparava, C. et al. (2006) Corpus-based and knowledge-based measures of text semantic similarity in: Aaai vol. 6 pp. 775–780

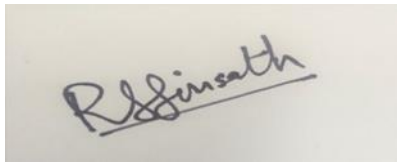
26. Mikolov, T., Chen, K., Corrado, G. and Dean, J. (2013) Efficient estimation of word representations in vector space arXiv preprint arXiv:1301.3781
27. Mobasher, B. (2007) Data mining for web personalization in: *The adaptive web* pp. 90–135 Springer
28. Rafter, R., Bradley, K. and Smyth, B. (2000) Personalised retrieval for online recruitment services in: *The BCS/IRSG 22nd Annual Colloquium on Information Retrieval (IRSG 2000)*, Cambridge, UK, 5-7 April, 2000 Recsys (2012) Recommender systems-how they works and their impacts: Content-based filtering
29. Ricci, F., Rokach, L. and Shapira, B. (2011) Introduction to recommender systems handbook in: *Recommender systems handbook* pp. 1–35 Springer
30. Rong, X. (2014) word2vec parameter learning explained arXiv preprint arXiv:1411.2738 Shearer, C. (2000) The crisp-dm model: the new blueprint for data mining *Journal of data warehousing* 5(4), pp. 13–22
31. Shrestha, P. (2011) Corpus-based methods for short text similarity
32. Slamet, C., Andrian, R., Maylawati, D.S., Darmalaksana, W., Ramdhani, M. et al. (2018) Web scraping and naïve bayes classification for job search engine in: *IOP Conference Series: Materials Science and Engineering* vol. 288 p. 012038 IOP Publishing
33. Sternitzke, C. and Bergmann, I. (2009) Similarity measures for document mapping:A comparative study on level of an individual scientist *Scientometrics* 78(1), pp. 113–130

Check list of items for Final report

- | | | |
|----|--|---|
| a) | Is the Cover page in proper format? | Y |
| b) | Is the Title page in proper format? | Y |
| c) | Is the Certificate from the Supervisor in proper format? Has it been signed? | Y |
| d) | Is Abstract included in the Report? Is it properly written? | Y |
| e) | Does the Table of Contents page include chapter page numbers? | Y |
| f) | Does the Report contain a summary of the literature survey? | Y |
| | i. Are the Pages numbered properly? | Y |
| | ii. Are the Figures numbered properly? | Y |
| | iii. Are the Tables numbered properly? | Y |
| | iv. Are the Captions for the Figures and Tables proper? | Y |
| | v. Are the Appendices numbered? | Y |
| g) | Does the Report have Conclusion / Recommendations of the work? | Y |
| h) | Are References/Bibliography given in the Report? | Y |
| i) | Have the References been cited in the Report? | Y |
| j) | Is the citation of References / Bibliography in proper format? | Y |

Note: A copy of this checklist should be included as the last page of the Final report. This checklist, duly completed and signed by the student, should also be verified and signed by the supervisor. Supervisors are requested to ensure that the students have prepared their reports properly.

Verified and Signed by Supervisor



Signed by Student

