Job Recommendation And Filtering System

Supervised By

Dr. Mohamed Ahmed Wahby

TA . Shrouk Mansour

Implemented By

Moaz Gamal Zain Alabdeen Emam	**20190549**
Abdalrahman Mohamed Elsayed	**20190315**
Mostafa Mahmoud Hassan	**20190545**
Alaa Taher Abd El Naby	***20190740***
Mohamed Zaki Maher	**20190437**
Esraa Mosaad Zaky	***20190673***

Artificial Intelligence Department

Cairo University

2022-2023

Acknowledgment

After our work, we would like to thank both **Dr.Mohamed Ahmed Wahby** And **Eng/Shrouk Mansour** for allowing us to work on the graduation project under Their supervision.

They also supported from the beginning by presenting to that idea and the technology used in it, not leaving, and always directing us to reach the next step in research and correct our endeavor path during the project.

We thank Them for Their time with us to advise us and listen to the problems we face and try to find solutions to them.

And of course, we don't forget to thank our families, especially our parents for their unconditional support mentally, financially, encouragement, patience, and help over the years. We are forever indebted to our parents, who always kept us in their prayers and pushed us to success.

We also thank our friends and colleagues for their support and feedback.

Finally, for our faculty for providing a suitable environment that led us to represent the best image that computer science graduates of Cairo University are supposed to represent.

Summary

Many individuals may possess certain abilities that are applicable to various occupations, yet they may not be aware of this. For instance, a machine learning engineer may have skills that are relevant to data analysis or business intelligence, but they may not recognize that these skills can be used for other jobs. With the abundance of job opportunities available, it is essential to find a way to assist people in selecting the most suitable job for them. So, we built this system.

Job Recommender System is a project that aims to match the skills of a person with the job description and print the top matched jobs. This project is motivated by the fact that job seekers often find it difficult to find suitable jobs that match their skills and qualifications. The current job market is highly competitive, and it is becoming increasingly difficult for job seekers to find the right job. To address this problem, this project proposes a Job Recommender System which uses Machine Learning algorithms to analyze the skills of a person and match them with available jobs.

The Job Recommender System will use Natural Language Processing (NLP) techniques to extract relevant information from job descriptions, such as required qualifications, experience, and skills. This information will then be used to create a profile for each job seeker which will be compared against available jobs in order to identify suitable matches. The system will also use Collaborative Filtering algorithms to recommend jobs based on user preferences and past experiences.

Our system also will be used by the companies in case if they want to filtering CVS for specific job description and our system will recommend the best CVS

The proposed system will be implemented using Python programming language along with various libraries such as Scikit-learn, NLTK, Gensim, and models such as Doc2Vec which is an NLP tool for representing documents as a vector and is a generalizing of word2vec method, we use tf-idf also and it gave us good results.

Overall, this project aims to develop an efficient Job Recommender System which can accurately identify suitable matches between job seekers and available jobs in order to help them find their desired career paths quickly and easily

Table of Content

List of Table

List of Figures

Abbreviations list

1) **LSA**: latent semantic analysis
2) **Word2vec**: word to vector
3) **Doc2vec**: document to vector
4) **Tf**: term frequency
5) **Idf**: inverse document frequency
6) **CBOW**: continues bag of words
7) **SVD**: singular value decomposition
8) **LDA**: Latent Dirichlet allocation

Chapter 1 : Introduction

1.1 Overview

In our life, there are a lot of people that need to know which will be the most suitable jobs for them that they can They bring out the best in their skills.

And sometimes people want to know if they are qualified for a specific job or a specific field so we design our system to help these people as it recommends jobs based on your real skills and it recommends jobs with a specific percentage so you can know you are qualified for a specific job or you must work on yourself to gain more skills.

Our project is designed to focus on recommending the best matching jobs so the main area of our project is to help people to know which jobs will be suitable for them or another case to help people that want to work in a specific field and don't know if they are qualified to this field or not Another feature of our system is an option for companies if they want to filter CVs for specific job description.

1.2 Motivation

Our motivation is that sometimes a lot of people don't know the best jobs that are suitable for their skills or people want to work in a specific field and don't know if they are qualified for this field or not.

Another motivation for us is that there aren't a lot of applications in this field as only a few applications like LinkedIn, wuzzuf so we took advantage of this situation.

1.3 Problem definition

In our life sometimes we don’t know which jobs are the most suitable jobs for us or we want to work in a specific job but we don’t know if our skills will be satisfiable for this job or not so our project solves these problems.

1.4 Stakeholder

The stakeholders of the project include job seekers, and recruiters, For example, job seekers in the field of machine learning can benefit from the project's advanced machine learning and AI techniques to identify job opportunities that align with their skills and experience. By uploading their CV and receiving personalized job recommendations based on their expertise in machine learning, job seekers can find job openings that best fit their interests and strengths in this specific field. By addressing the specific needs of job seekers in this field, the project can contribute to the growth and success of the IT industry and create new opportunities for job seekers.

1.5 Main techniques:

The main project technologies include Natural Language Processing (NLP), machine learning algorithms, and web development frameworks. NLP techniques are used to extract and compare skills that we get from job descriptions and the skills that we get from CVs like cosine similarity, including the use of the TfidfVectorizer or doc2vec, word2vec models to convert text data into vector that the model uses to train on.

Machine learning algorithms are used for job matching and candidate evaluation, including cosine similarity to compare skills and experience between job descriptions and CVs.

For web development frameworks, we use flask in our project, flask helps in the area of integration between our machine and the website, to build a web-based interface that allows users to upload their CVs and receive job recommendations based on their skills and experience. Additionally, the system allows employers and recruiters to upload job descriptions and receive recommendations for the best matching CVs from a collection of uploaded CVs.

Overall, the project uses a combination of advanced NLP techniques, machine learning algorithms, and web development frameworks to provide a powerful tool for job seekers and employers to identify the most suitable job opportunities and candidates.

The use of these technologies demonstrates the potential of AI and machine learning to revolutionize the job market and improve the efficiency and effectiveness of job matching and candidate evaluation processes.

1.6 Main application

The main application is a web-based system that uses Flask, a lightweight web application framework, to enable users to upload their CVs and receive recommendations for the best matching jobs based on their skills and experience. The system is designed to help individuals recognize their transferable skills and find job opportunities that align with their interests and strengths.

In addition to providing job recommendations based on user CVs, the application also allows users to upload job descriptions and receive recommendations for the best matching CVs from a collection of uploaded CVs. This feature is designed to help employers and recruiters identify the most qualified candidates for their job openings.

When a user uploads their CV or a job description is uploaded, the system uses natural language processing and machine learning algorithms to extract relevant information about the skills and experience required for the job. This information is then matched against a database of job listings or uploaded CVs to identify the most suitable job opportunities or candidates.

The Flask application provides a user-friendly interface that allows users to easily upload their CVs or job descriptions and view the recommended job listings or CVs. The application also includes features such as filtering and sorting options to help users refine their search results.

Using Flask for deployment offers several advantages, including its flexibility, ease of use, and compatibility with a wide range of Python libraries and tools. However, developing the application using Flask also presented some challenges, such as ensuring the scalability and reliability of the system as the number of users and job listings grows.

Overall, the main application is a powerful tool for helping individuals identify job opportunities that best fit their skills and experience and for employers and recruiters to identify the most qualified candidates for their job openings. By using Flask and machine learning algorithms, the system provides a user-friendly and efficient way for users to find the most suitable job opportunities and candidates.

Chapter 2 : Literature Review

2.1 Existing system

Many employment businesses have developed methods for offering the job board 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 applies.

Due to the abundance of job boards, candidates typically choose the one that offers the best services to them, including CV writing, developing a job profile, and suggesting new positions to job seekers. Job seekers are looking for new possibilities that fit their talents more actively and persistently.

But it's difficult for companies that target these job seekers to determine the candidate's skill set, linked in is the most similar application to our idea as it recommends jobs based on your profile, activity, job notifications, and searches for positions,

but our project recommends jobs based on the user skills and it recommends jobs with a specific degree of matching like your skills matching this job with 98%, another option for services is filtering CVs for specific job descriptions as when a company shared a job description for a specific job a lot of people will upload their cv and it is so difficult to arrange an interview to all these people so our system recommend the best CVs based on the job description so it will be easy to make like 5 interviews instead of 500 interviews.

2.2 Term frequency-inverse document frequency (TF-IDF)

Computers are good with numbers, but not that much with textual data. One of the most widely used techniques to process textual data is TF-IDF

From our intuition, we think that the words which appear more often should have a greater weight in textual data analysis, but that's not always the case. Words such as "the", "will", and "you" — called stopwords — appear the most in a corpus of text, but are of very little significance. Instead, the words which are rare are the ones that actually help in distinguishing between the data, and carry more weight.

TF-IDF stands for "Term Frequency — Inverse Document Frequency". First, we will learn what this term means mathematically.

Term Frequency (tf): gives us the frequency of the word in each document in the corpus. It is the ratio of number of times the word appears in a document compared to the total number of words in that document. It increases as the number of occurrences of that word within the document increases. Each document has its own tf.

$$tf_{i,j} = \frac{n_{i,j}}{\sum_k n_{i,j}}$$

Inverse Document Frequency (idf): used to calculate the weight of rare words across all documents in the corpus. The words that occur rarely in the corpus have a high IDF score. It is given by the equation below.

$$idf(w) = log(\frac{N}{df_t})$$

Combining these two we come up with the TF-IDF score (w) for a word in a document in the corpus. It is the product of tf and idf:

$$w_{i,j} = tf_{i,j} \times \log\left(\frac{N}{df_i}\right)$$

$tf_{i,j}$ = number of occurrences of i in j
df_i = number of documents containing i
N = total number of documents

Table 1: Example Solution on TF-IDF

Word	TF		IDF	TF*IDF	
	A	B		A	B
The	1/7	1/7	log(2/2) = 0	0	0
Car	1/7	0	log(2/1) = 0.3	0.043	0
Truck	0	1/7	log(2/1) = 0.3	0	0.043
Is	1/7	1/7	log(2/2) = 0	0	0
Driven	1/7	1/7	log(2/2) = 0	0	0
On	1/7	1/7	log(2/2) = 0	0	0
The	1/7	1/7	log(2/2) = 0	0	0
Road	1/7	0	log(2/1) = 0.3	0.043	0
Highway	0	1/7	log(2/1) = 0.3	0	0.043

From the above table, we can see that TF-IDF of common words was zero, which shows they are not significant. On the other hand, the TF-IDF of “car”, “truck”, “road”, and “highway” are non-zero. These words have more significance.

2.3 Word to vector

2.3.1 Word2Vec Architecture

The ability of Word2Vec to assemble vectors of related words gives its efficiency. Word2Vec can provide accurate predictions about a word's meaning based on its usage in the text when given a big enough dataset. With other words in the corpus, these estimations produce word connections. Words like "King" and "Queen," for instance, sound quite similar to one another.

You can obtain a good approximation of word similarity by performing algebraic operations on word embeddings. For instance, the 2-dimensional embedding vector of "king" plus the 2-dimensional embedding vectors of "man" and "woman" produced a vector that is extremely similar to the 2-dimensional embedding vector of "queen." Please take note that the values below were selected at random.

```
King    -    Man    +    Woman    =    Queen
[5,3]   -    [2,1]  +    [3, 2]   =    [6,4]
```

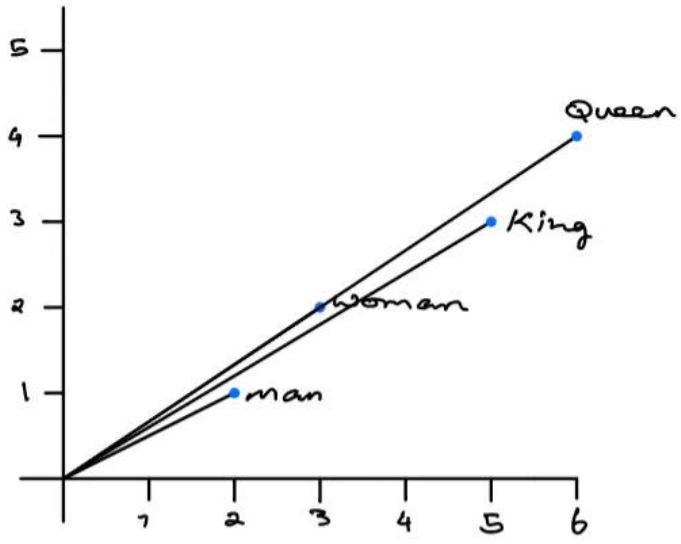


Figure 2.3.1 Word2vec Example

The success of word2vec may be attributed to two fundamental designs. the CBOW and skip-gram designs. The skip-gram and CBOW differ significantly; while the skip-gram is more sophisticated than the CBOW, it is also the most useful.

2.3.2 CBOW (Continuous Bag of Words)

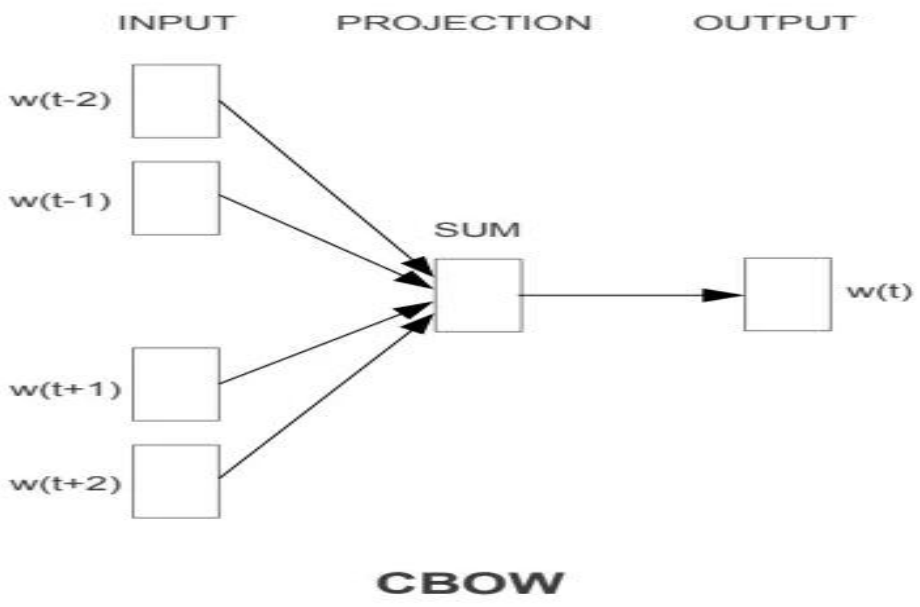


Figure 2.3.1 Continuous Bag of Words's Architecture

This design resembles a feed-forward neural network very much. Using this model architecture, a target word is effectively predicted from a list of surrounding terms. The idea behind this approach is pretty straightforward: given the sentence "Have a great day," we will select "a" as our target word and ["have," "great," and "day"] as our context terms. To attempt to predict the target word, this model will use distributed representations of the context words.

2.3.3 Skip-Gram

The skip-gram model is a straightforward neural network with a single hidden layer that has been trained to forecast the likelihood that a particular word will appear when an input word does. It seems sense to think of the skip-gram model as the CBOW model's antithesis. This architecture attempts to properly anticipate the words that come before and after the current word using it as an input. In essence, the goal of this model is to discover and forecast the words that surround the given input word. A vast variety of word vectors results in better prediction quality, but it also adds to the computational cost, according to trials evaluating the model's accuracy. Visual representations of the procedure are shown below.

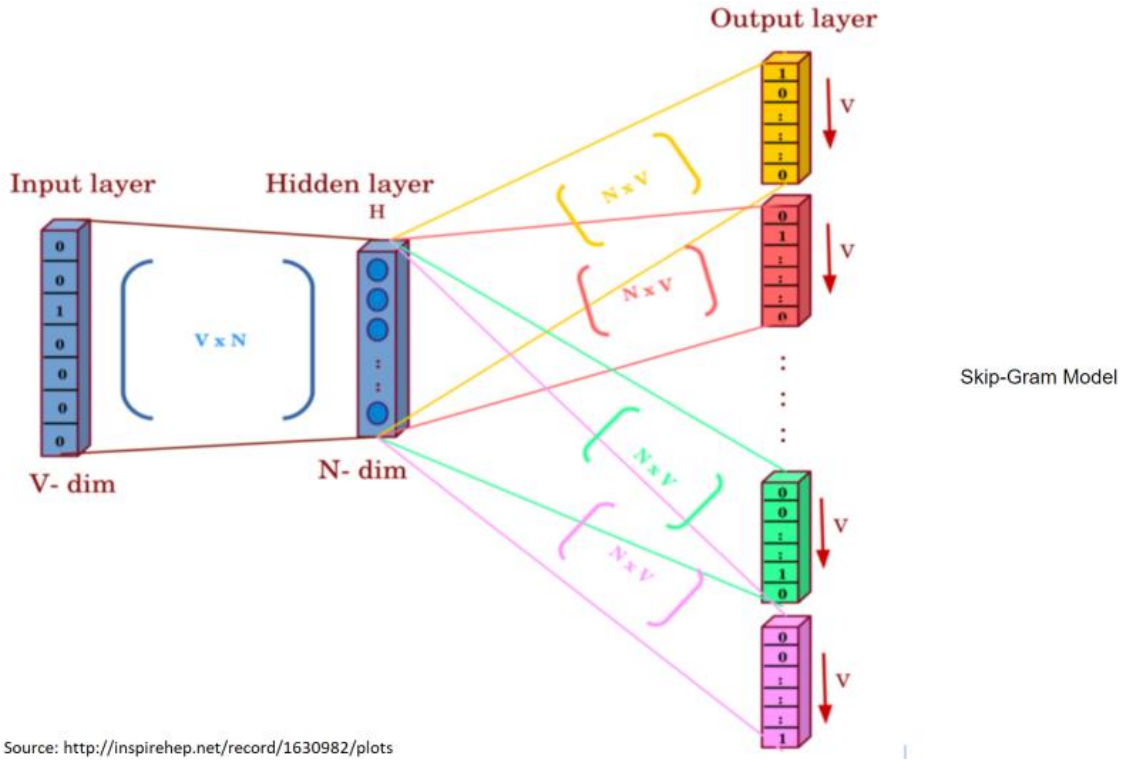


Figure 2.3.2 The Skip-gram Architecture

2.4 Doc2vec (Document to vector)

It should be simpler to comprehend how doc2vec functions after you have a better idea of what word2vec is. No matter how long a document is, doc2vec's objective is to provide a numeric representation of it. Nevertheless, unlike words, documents do not have logical structures as words do, thus a different approach must be devised.

In our experiments, We will employ the doc2vec models in the following ways: A set of documents are needed for training. Word vectors (W) are generated for each word, and document vectors (D) are generated for each document. The model also trains the weights of a SoftMax hidden layer. During the inference phase, a new document may be presented. and all weights are fixed to determine the document vector.

2.4.1 Distributed Memory Version of Paragraph Vector

Mikilov and Le's idea was straightforward; they added another vector (Paragraph ID below) to the word2vec model as follows:

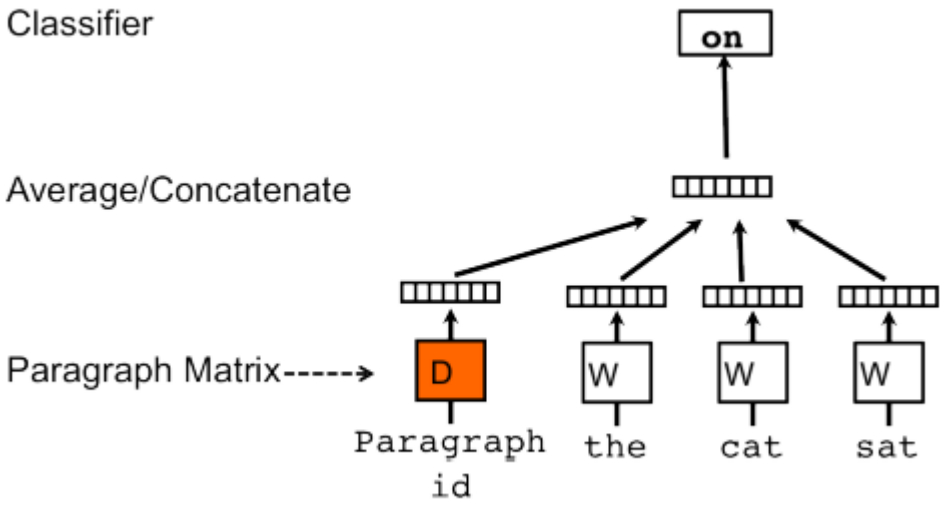


Figure 0.3.1 The distributed memory model of Paragraph Vector for an input sentence.

"Distributed Memory Version of Paragraph Vector" is the name of the above model (PV-DM).In order to capture the document's subjects, The paragraph vector aims to represent the concept of a document, whereas the word vectors express the concept of a word.

2.4.2 Distributed bag of words

Since we don't use any local context for the prediction task, the paragraph vector can be even more easily understood. We can reach the "Distributed Bag of Words" model as follows:

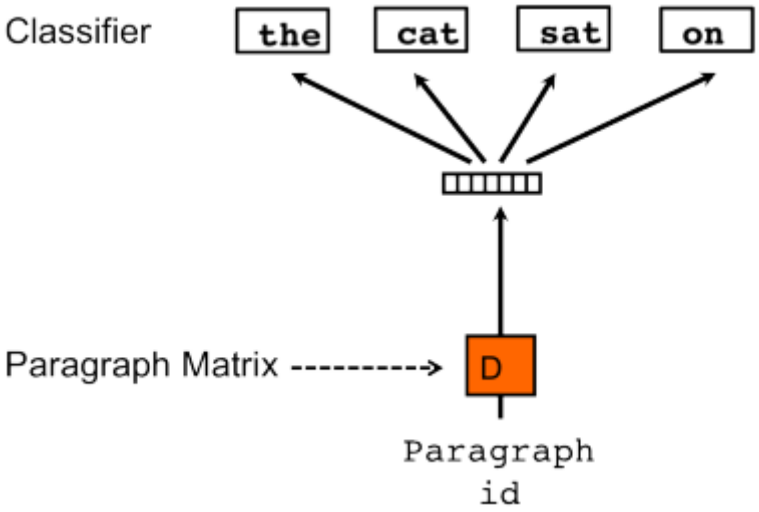


Figure 2.4.2 The distributed bag of words model of Paragraph Vector..

Backpropagation is utilized to fine-tune the paragraph vectors because the classifier and word vector parameters are not required at inference time. Given how effective the dispersed bag of words model is this algorithm is actually faster (as opposed to word2vec) and consumes less memory, since there is no need to save the word vectors.

2.5 Fast text

Fast Text is another word embedding method that is an extension of the word2vec model. Instead of learning vectors for words directly, Fast Text represents each word as an n-gram of characters. So, for example, take the word "*artificial*" with n=3. The fast text representation of this word is [*ar, art, rti, tif, ifi, fic, ici, ial, al]*, where the angular brackets indicate the beginning and end of the word.

As a result, the meaning of shorter words is captured and prefixes and suffixes are understood by the embeddings. A skip-gram model is trained to learn the embeddings after the word has been represented using character n-grams. Due to the lack of consideration for the underlying structure of the word, this model is regarded as a bag of words model with a sliding window over a word. The sequence of the n-grams doesn't matter as long as the letters are inside this window.

fast Text works well with rare words. So, even if a word wasn't observed during training, its embeddings may still be obtained by splitting it up into n-grams.

- Words that are not in the model dictionary have no vector representation in Word2vec or Glove, respectively. This is a significant benefit of this approach.

2.6 Topic Modeling and Latent Dirichlet Allocation (LDA)

We'll examine a technique called latent Dirichlet allocation (LDA) to comprehend how topic modelling functions. It is common practice to utilize this method for topic modelling in a range of applications. It is simple to deploy because it has decent implementations in coding languages like Python and Java.

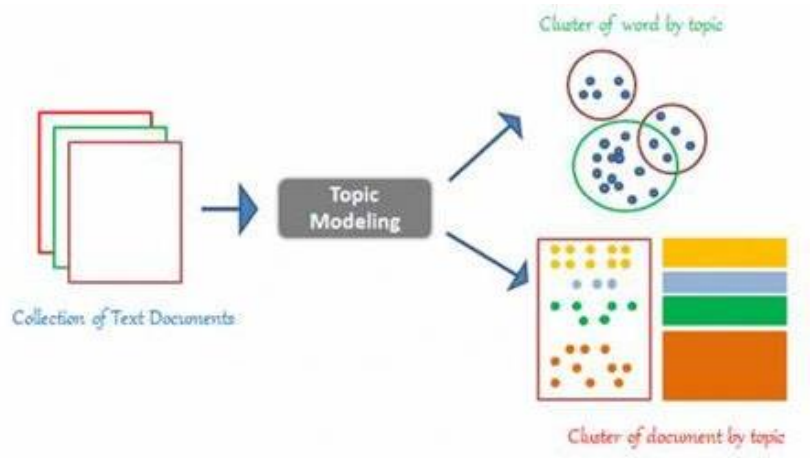


Figure 2.4 LDA topic modeling.

Latent topics in a collection of text documents are found using LDA topic modeling. It accomplishes this by assuming potential topics from the words used in the documents. To accomplish this, it makes use of Dirichlet distributions and a generative probabilistic model.

The first thing to keep in mind with LDA is that we must choose the K topic number in advance.

By selecting K, we are indicating that we assume K topics can adequately characterize the collection of documents under analysis. In our example, we'll use k = 9 because of the nature of the job.

2.6.1 The LDA algorithm

The topic mix, or distribution of topics for each document, will be generated by LDA for each document. All documents contain the same K topics but in varying amounts (mixes). The LDA algorithm goes through an iterative process as follows after selecting a value for K: It employs a four-step iterative process that starts with initializing topic assignments for each word in the documents, moves on to updating topic assignments for individual words based on co-occurrence probabilities with other words and topics as well as Dirichlet variability, repeats the process for all words in all documents, and ends with iterating.

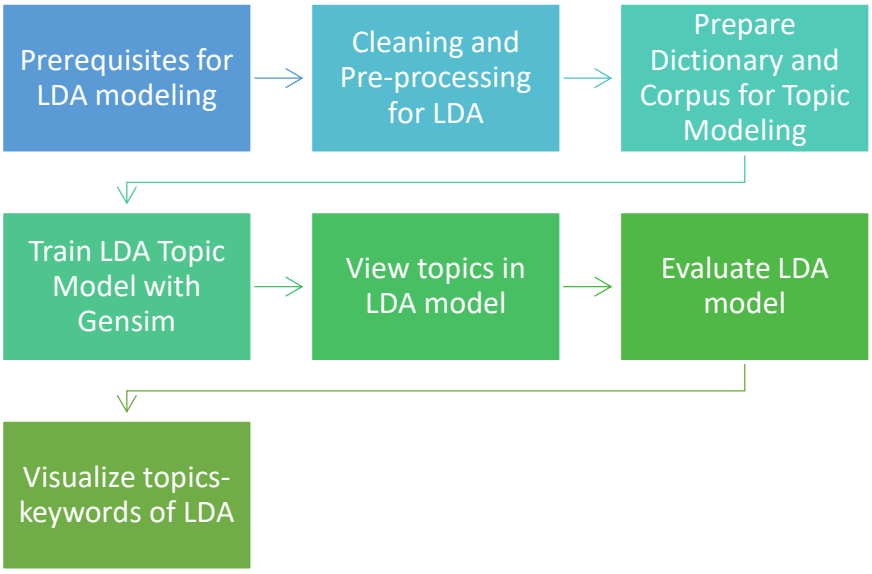


Figure 2.0.5 LDA Steps

2.7 Latent Semantic Analysis (LSA)

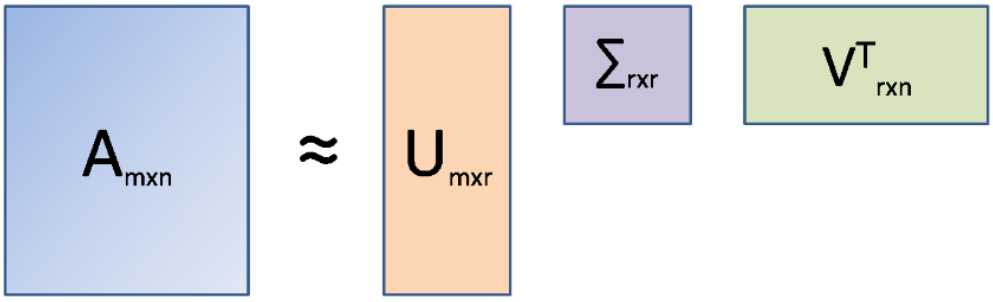
Latent Semantic Analysis (LSA) involves creating structured data from a collection of unstructured texts. Before getting into the concept of LSA, let us have a quick intuitive understanding of the concept. When we write anything like text, the words are not chosen randomly from a vocabulary.Rather, we think about a theme (or topic) and then chose words such that we can express our thoughts to others in a more meaningful way. This theme or topic is usually considered as a latent dimension.

It is latent because we can't see the dimension explicitly. Rather, we understand it only after going through the text. This means that most of the words are semantically linked to other words to express a theme. So, if words are occurring in a collection of documents with varying frequencies, it should indicate how different people try to express themselves using different words and different topics or themes.

In other words, word frequencies in different documents play a key role in extracting the latent topics. LSA tries to extract the dimensions using a machine learning algorithm called Singular Value Decomposition or SVD.

2.7.1 Singular Value Decomposition (SVD)

Singular Value Decomposition or SVD is essentially a matrix factorization technique. In this method, any matrix can be decomposed into three parts as shown below.



Here, A is the document-term matrix (documents in the rows(m), unique words in the columns(n), and frequencies at the intersections of documents and words). It is to be kept in mind that in LSA, the original document-term matrix is approximated by way of multiplying three other matrices, i.e., U, $\sum$ and V^T. Here, r is the number of aspects or topics. Once we fix r ($r<<n$) and run SVD, the outcome that comes out is called Truncated SVD and LSA is essentially a truncated SVD only.

SVD is used in such situations because, unlike PCA, SVD does not require a correlation matrix or a covariance matrix to decompose. In that sense, SVD is free from any normality assumption of data (covariance calculation assumes a normal distribution of data). The U matrix is the document-aspect matrix, V is the word-aspect matrix, and $\sum$ is the diagonal matrix of the singular values. Similar to PCA, SVD also combines columns of the original matrix linearly to arrive at the U matrix. To arrive at the V matrix, SVD combines the rows of the original matrix linearly. Thus, from a sparse document-term matrix, it is possible to get a dense document-aspect matrix that can be used for either document clustering or document classification using available ML tools. The V matrix, on the other hand, is the word embedding matrix (i.e. each and every word is expressed by r floating-point numbers) and this matrix can be used in other sequential modeling tasks. However, for such tasks, Word2Vec and Glove vectors are available which are more popular.

We test using network engineer cv

Table 2: Result Before LSA

Job title	Cosine similarity
Network Engineer Data Security	**0.445669**
Network Engineer	**0.443848**
Network Engineer	**0.440356**
lead Network Engineer Senior Network Engineer	**0.429953**
Senior Principal DevOps Engineer Sysops	**0.413897**

It gave us the right job title but the score was small

Table 3: Result After LSA

Job title	Cosine similarity
Network Engineer	**0.899950**
Network Security Senior Analyst	**0.899539**
Network Engineer	**0.892271**
Network Infrastructure Engineer Data Centre	**0.885371**
Network Infrastructure Engineer Data Centre	**0.883382**

The matching score is doubled so LSA was very good to improve our result

2.8 measuring similarity

The cosine similarity is advantageous because even if the two similar documents are far apart by the Euclidean distance (due to the size of the document), chances are they may still be oriented closer together. The smaller the angle, higher the cosine similarity.

2.8.1 Euclidean distance

The Euclidean distance between q↑ and d2↑ is large even though the distribution of terms in the query q↑ and the distribution of terms in the document d2↑ are very similar.

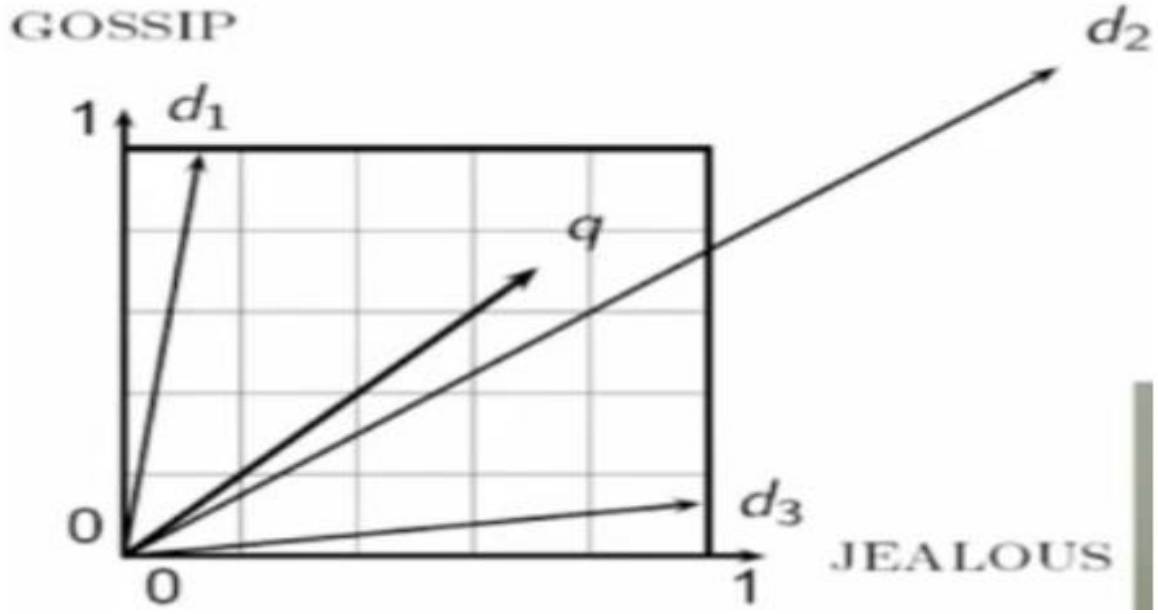


Figure 0.6 Euclidean distance

Thought experiment: take a document d and append it to itself. Call this document d'. Semantically d and d' have the same content

The Euclidean distance between the two documents can be quite large, the angel between the two documents is 0, corresponding to maximal similarity.

Key idea: Rank documents according to angel with query.

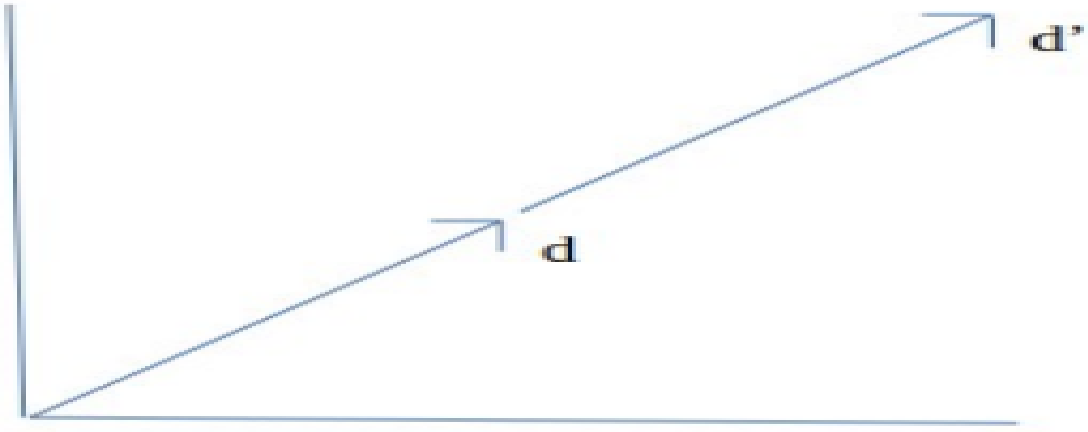


Figure 0.8 Example of Disadvantages of Euclidean distance.

- **So, we use cosine similarity**

2.8.3 Cosine similarity

A commonly used approach to match similar documents is based on counting the maximum number of common words between the documents. but this approach has an inherent flaw, that is, as the size of the document increases, the number of common words tend to increase even if the documents talk about different topics. The cosine similarity helps overcome this fundamental flaw in the 'count-the-common-words' or Euclidean distance approach.

Cosine similarity is a metric used to determine how similar the documents are irrespective of their size.

Mathematically, Cosine similarity measures the cosine of the angle between two vectors projected in a multi-dimensional space.

In this context, the two vectors I am talking about are arrays containing the word counts of two documents.

When plotted on a multi-dimensional space, where each dimension corresponds to a word in the document, the cosine similarity captures the orientation (the angle) of the documents and not the magnitude. If you want the magnitude, compute the Euclidean distance instead.

The cosine similarity is advantageous because even if the two similar documents are far apart by the Euclidean distance because of the size (like, the word 'cricket' appeared 50 times in one document and 10 times in

another) they could still have a smaller angle between them. Smaller the angle, higher the similarity.

Cosine Similarity Example

Let's suppose you have 3 documents based on a couple of star cricket players – Sachin Tendulkar and Dhoni.Two of the documents (A) and (B) are from the Wikipedia pages on the respective players and the third document (C) is a smaller snippet from Dhoni's Wikipedia page.

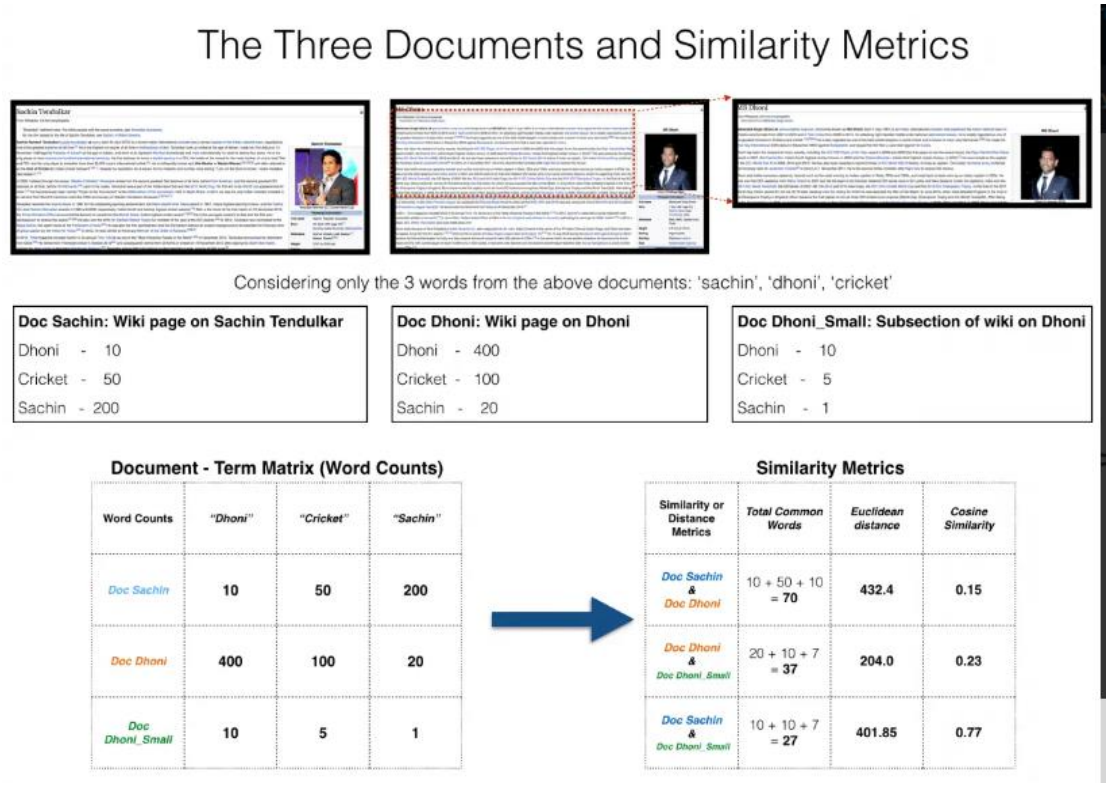


Figure 2.8.3Example of Cosine Similarity.

Our objective is to quantitatively estimate the similarity between the documents.

For ease of understanding, let's consider only the top 3 common words between the documents: 'Dhoni', 'Sachin' and 'Cricket'.

You would expect doc b and doc c, that is the two documents on Dhoni would have a higher similarity over doc a and doc b, because, doc c is essentially a snippet from Doc B itself.

However, if we go by the number of common words, the two larger documents will have the most common words and therefore will be judged as most similar, which is exactly what we want to avoid.

The results would be more congruent when we use the cosine similarity score to assess the similarity.

Let's project the documents in a 3-dimensional space, where each dimension is a frequency count of either: 'Sachin', 'Dhoni' or 'Cricket'. When plotted on this space, the 3 documents would appear something like this.

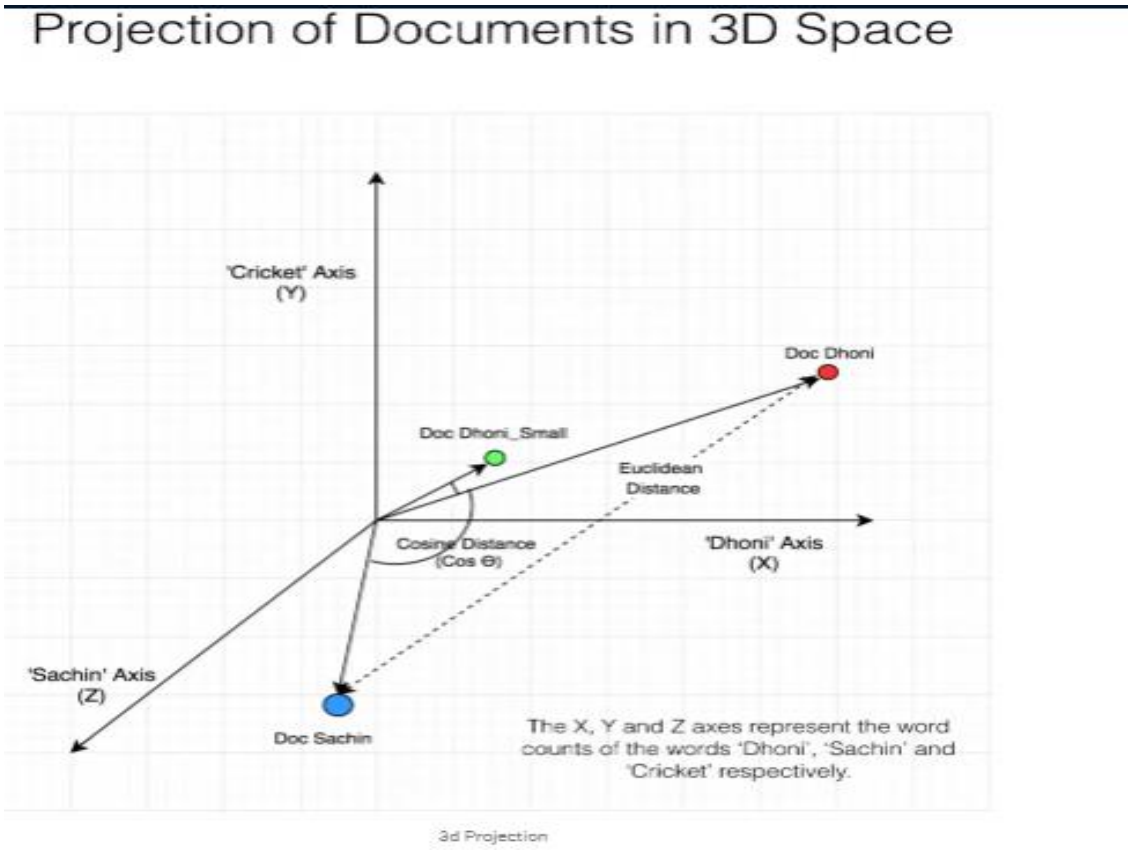


Figure 2.8.3 Projection of Documents in 3D Space.

As you can see, Doc Dhoni_small and the main Doc Dhoni are oriented closer together in 3-D space, even though they are far apart by magnitude.

It turns out, the closer the documents are by angle, the higher is the Cosine Similarity (Cos theta).

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

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

As you include more words from the document, it's harder to visualize a higher dimensional space. But you can directly compute the cosine similarity using this math formula. Enough with the theory

Chapter 3 : Methodology

3.1 Process flow

In this study, a natural language processing tool, provided by a Python software package called NLTK , will be used to analyze explicit content properties, such as Job title and Job description. Unlike previous recommender systems, content-based filtering considers both the user and the item's content. Based on the needs, content-based filtering enables us to examine the data using techniques like text processing or semantic information. Also, the recommender system will be transparent because the user's recommendation may be justified based on the item's content. In order to provide a better understanding of the many components and how they interact, we have included a flow chart of our proposed system's overall system architecture.

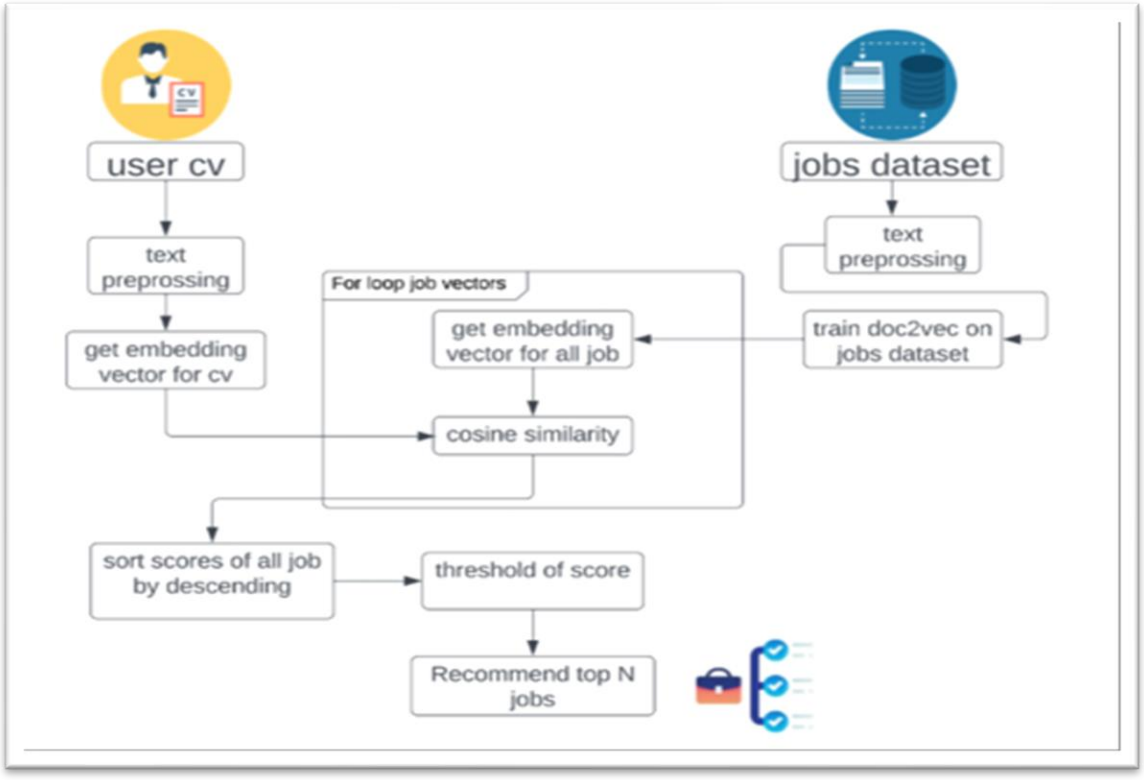


Figure 0.1 Process flow - Recommender system

- This figure shows our system methodology as the user firstly upload his cv then apply preprocessing on its text to extract user skills then convert text to embedding vectors

- Our system took job descriptions then apply preprocessing on its text to extract the skills that required for this job then convert text to embedding vectors
- In the third step our system measures the similarity between user skills and the required skills for jobs using cosine similarity
- Then sort all the jobs based on the degree of similarity and recommend the top matching jobs.

3.2 Data Source

For this experiment, we utilized two separate datasets from the job domain. The first dataset, "Job Boards Listings For IT Jobs" from Kaggle, is a simple dataset that includes job information such as job titles and corresponding job descriptions. The job titles examined are software engineer, project manager, data analyst, web developer, and data scientist. To expand our data, we scraped 2,975 job postings from LinkedIn using the Octoparse_8 application for the following job titles: DevOps Engineer, Testing, Python Developer, Java Developer, Data Science, Web Designing, Database, Business Analyst, Sales, Blockchain, Hadoop, and HR. We are combining the job data that we have gathered by scraping online sources with the job dataset that we obtained from Kaggle. Additionally, we utilize the "Resume Dataset" from Kaggle to obtain users' CVs and check for differences across various CVs. This helps us to ensure that our job matching algorithm can accurately match candidates with job postings based on their skills and experience. It also allows us to identify any potential biases in the algorithm and make adjustments to ensure fairness and equality.

Figure 3.2 Resume Dataset from Kaggle.

jobboard - jobboard.csv (42.47 MB)

Detail | Compact | Column

title	description
Software Engineer	Apply now ...
Software Engineer	Apply now ...
Software Engineer	Apply now ...
Software Engineer	Apply now ...
Software Engineer	Apply now ...

Figure 3.3 Job Boards Dataset from Kaggle.

	Job_title	Job_link	Job_description	Keyword
0	DevOps Engineer	https://eg.linkedin.com/jobs/view/devops-engin...	We looking DevOps Engineer join team develop s...	scrab_ddata
1	DevOps Engineer	https://eg.linkedin.com/jobs/view/devops-engin...	Who We Are We owned funded one Egypt renowned ...	scrab_ddata
2	DevOps Engineer	https://eg.linkedin.com/jobs/view/devops-engin...	Wind IS Company Oracle Partner Egypt KSA UAE h...	scrab_ddata
3	Cloud Infrastructure Engineer	https://eg.linkedin.com/jobs/view/cloud-infras...	Who We Are We owned funded one Egypt renowned ...	scrab_ddata
4	DevOps Engineer	https://eg.linkedin.com/jobs/view/devops-engin...	About Axis We believe financial services acces...	scrab_ddata
...	...	...	...	...
16951	Real Estate Development Executive	NaN	Manage strategy execution innovation delivery ...	google_job_skills
16952	Protective Services Specialist Executive Prote...	NaN	Plan implement large scale security operations...	google_job_skills
16953	Campus Security Manager	NaN	Manage daily operations security vendors provi...	google_job_skills
16954	Facilities Manager	NaN	Collaborate functional teams plan manage deliv...	google_job_skills
16955	Physical Security Manager	NaN	Partner closely offer guidance consultation Fa...	google_job_skills

16956 rows × 7 columns

Figure 0.4 ALL job data

3.3 System input and output

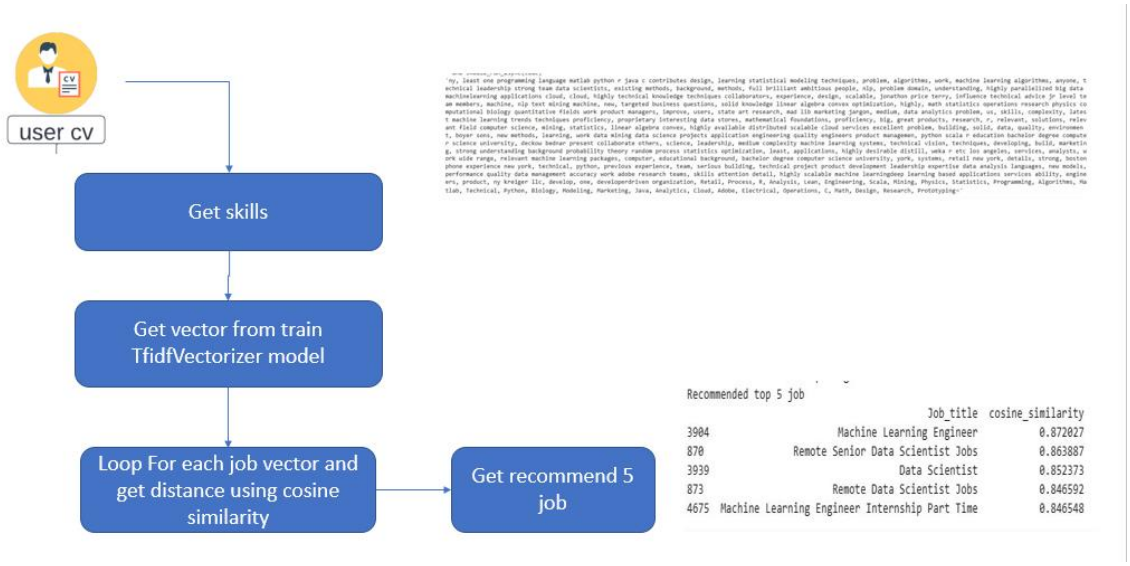


Figure 0.5 Input and output of the system.

The system's inputs consist of two datasets from the job domain, including job titles and corresponding job descriptions, along with CVs from the "Resume Dataset" from Kaggle. The system also takes into account any preprocessing techniques applied to the data before training the tfidf model. The output of the system is a job-matching algorithm that accurately matches candidates with job postings based on their skills and experience.

Steps of how TF-IDF works in a job recommendation system:

1. Corpus creation: The first step is to create a corpus of job descriptions and candidate resumes. This corpus is used as the basis for calculating the TF-IDF scores.

2. Tokenization: The next step is to tokenize the text, which involves breaking it up into individual words or terms. This process helps to standardize the data and make it easier to analyze.

3. Term frequency calculation: The term frequency (TF) of each term is calculated by counting the number of times it appears in a particular job description or resume.

4. Inverse document frequency calculation: The inverse document frequency (IDF) of each term is calculated by dividing the total number of documents in the corpus by the number of documents that contain the term. This helps to identify terms that are relatively rare and therefore more important or distinctive.

5. TF-IDF calculation: The TF-IDF score for each term is calculated by multiplying the TF and IDF values. This produces a score that reflects both the frequency of the term within a particular document and its relative importance within the corpus as a whole.

6. Similarity calculation: Once the TF-IDF scores have been calculated for each term in a job description or resume, the similarity between the two documents can be calculated using a similarity metric such as cosine similarity. This measures the degree of overlap between the terms used in the two documents and can be used to identify the most relevant job recommendations for a particular candidate.

3.4 Data preprocessing

Effective job descriptions are critical for attracting qualified candidates and making informed hiring decisions. This involves several tasks, including removing irrelevant information and refining the language used to make the description clean and structured. The data preprocessing function we created is a useful tool that can facilitate this process by removing email addresses, links, punctuation, digits, and stop words from the job description data. These pre-processing steps help to eliminate noise and irrelevant information, which can enhance the accuracy and effectiveness of any subsequent analyses. Overall, ensuring that job descriptions are

well-structured and clear can help to attract the right candidates and streamline the hiring process.

3.5 Skills extraction

Extracting skills from a resume is an essential step in the process of job matching and candidate evaluation. There are various methods to extract skills from a resume, and one of them is using pre-trained models and libraries like spaCy and NLTK. The first method uses spaCy to extract noun chunks and organization names from the resume text, and NLTK to extract keywords and phrases. These two extracted lists are then combined into a single list of skills. The second method involves opening the resume text as a Word document and using the ResumeParser library to extract skills from it. This method saves the resume as a Word document, and then extracts the skills from the saved document using ResumeParser's built-in skills extractor. Both methods have their advantages and disadvantages, but when used together, they can provide a comprehensive list of skills that can be used for job matching and candidate evaluation.

3.6 Train Model

After we tried different models, we found that tfidf is a powerful technique for identifying the most relevant job recommendations for a candidate based on the similarity between their skills and experience and the requirements of a particular job description, **based on the results that we will see in the section of results and analysis**.

Chapter 4 : Deployment Phase

4.1 About the website

The website consists of six pages, each serving a specific purpose:

1. Start Page: This is the landing page of the website.
2. Recommendation Page: Users can input their preferences to receive job recommendations.
3. Recommendation Results: Displays the recommended job listings based on user preferences.
4. Filtering Page: Users can filter CVS based on job description.
5. Filtering Results: Displays the filtered CVs and its emails and mobile numbers.
6. Team Page: Provides information about the project team

4.2 Technologies Used

- HTML
- CSS
- JavaScript
- Bootstrap
- Flask

4.3 Usage

To use the website, follow these steps:

- Open the website in a web browser.
- Navigate through the different pages using the navigation menu.
- Input preferences on the Recommendation Page or apply filters on the Filtering Page.
- View the corresponding results on the Recommendation Results or Filtering Results page.

4.4 Installation

To install and run the project locally, follow these steps:

Clone the project repository.

Install the required dependencies using the provided requirements.txt file.

Run the Flask development server.

Open the website in a web browser by accessing the specified localhost address.

4.5 The project in details

We have two tasks: one for the job seekers (job recommendation system) and the other for the job recruitments (filtering system).

4.5.1 commonalities between the two tasks

- We created a database to store our data and the extracted skills. We then established a connection with the database to retrieve the data for use in the app.py file.
- We defined a function called "data_preprocessing_text()" that preprocesses the given text using various techniques. Here's a breakdown of the steps involved:
    - Lowercasing the text
    - Removing tabulation and punctuation
    - Removing digits
    - Removing non-ASCII characters
    - Removing stop words
    - Lemmatizing the tokens

  The function takes a single argument, "text," which is the input text to be preprocessed. It returns the preprocessed text as a string.

- We defined two functions for extracting skills from text, such as resumes:
    - ✓ "extract_skills_1(resume_text)": This function takes the resume text as input. It utilizes the spaCy library to extract noun chunks and named entities (organizations) from the resume text. The NLTK library is used to tokenize the resume text and extract keywords and phrases. Skills are extracted from the collected noun chunks, named

entities, keywords, and phrases. The function returns the extracted skills as a list.

- ✓ "extract_skills_2(text)": This function takes a text as input, typically a resume or document. It creates a new Word document using the python-docx library and saves the given text in it. The ResumeParser class from the ResumeParser library is used to extract data from the document. Skills are extracted from the parsed data, and the function returns them as a list.

Both functions aim to extract skills from the given text using different techniques and libraries, such as Word processing and named entity extraction.

4.5.2 job seekers task (job recommendation system)

we performed three key actions:

1. Recommending relevant jobs

- ❖ We performed TF-IDF vectorization on the 'skills' column of the 'job_data' DataFrame using the TfidfVectorizer class from scikit-learn. The 'stop_words' parameter was set to 'english' to exclude common English stop words. The resulting TF-IDF matrix was stored in the 'tfidf_matrix' variable.
- ❖ We defined a function called "get_top_n_tfidf(cv_text, n)" that retrieves the top n job recommendations based on the cosine similarity between a given CV text and the job descriptions' TF-IDF matrix.
  - First, we performed Latent Semantic Analysis (LSA) using the TruncatedSVD class from scikit-learn. The TF-IDF matrix was transformed into a lower-dimensional representation stored in the 'lsa_matrix' variable.
  - Next, we transformed the CV text using the same TF-IDF vectorizer and the 'vectorizer2' object.
  - Then, we computed the cosine similarity between the transformed CV vector and the LSA matrix using the cosine_similarity function from scikit-learn.
  - Afterward, we created a DataFrame (df) containing the job titles and cosine similarity scores.
  - Next, we retrieved the indices of the top n job recommendations based on the cosine similarity scores.

- Then, we filtered the DataFrame to select the top n job recommendations and their corresponding cosine similarity scores.
- Finally, we returned the resulting DataFrame containing the top job recommendations.

2. Searching for existing jobs

• We defined a function called "get_similar_jobs_for_job(job_title, num_results=5)" that retrieves a list of similar job titles based on a given job title.

- First, we created a search query by appending the string "jobs" to the given job title.
- Then, we performed a search using the search function (assumed to be imported from a relevant library) with the provided query. The tld parameter specifies the top-level domain to use, num specifies the number of search results to retrieve, stop specifies the total number of search results to stop at, and pause specifies the time delay between consecutive searches.
- Next, we initialized an empty list (similar_jobs) to store the similar job titles.
- We then iterated through the search results and checked if each result contains the word "job" (case-insensitive). If it did, we added the result to the list of similar jobs.
- Afterward, we removed duplicates from the list of similar jobs using the list(set(similar_jobs)) idiom.
- Subsequently, we sorted the list of similar jobs in ascending order based on job title.
- Finally, we returned the sub-list of similar job titles, limited to num_results (defaulting to 5) elements.

3. Listing the missing skills on the user's resume

• We performed the following steps:

i. Created a dictionary (normalized_skill_dict) that maps normalized skill forms to the original skill names. The normalization process involved converting skill names to lowercase and removing punctuation.

ii. Defined a function called "get_skills_for_cv(cv_skills, job_skills)" that matches CV skills to job skills based on their normalized forms and returns the matching CV skills and recommended CV skills. The cv_skills parameter is a list of CV skills, and the job_skills parameter is a list of job skills.

The function proceeded as follows:

- Initialized lists for matching CV skills and recommended CV skills.
- Iterated through the CV skills and matched each skill to its normalized form in the normalized_skill_dict.
- Within the matching CV skills, found the most similar skill among the matching skills based on a similarity score computed using the fuzz.token_sort_ratio function.
- Added the best matching skill to the list of matching CV skills if the similarity score exceeded a threshold of 60 (you can adjust this threshold as needed).
- Printed the lists of CV skills, job skills, matching CV skills, and recommended CV skills.
- Returned a tuple containing the matching CV skills and recommended CV skills, both converted to sets to remove duplicates.

4.5.3 job recruitment (filtering system)

- After taking the job description, CVs, and the number of needed CVs from the user, we recommend the best CVs along with their emails and mobile numbers. Here's what we did:

- We loaded our Word2Vec model from a pickle file named "word2vecmodel.pkl" using the pickle module. The loaded model is assigned to the variable "my_model".

- We defined a function called "**get_job_vector(job_description)"** that calculates the average vector representation of a job description. The "job_description" parameter is a string representing the job description. The function proceeds as follows:

    i. Tokenizes the job description into individual words.

    ii. Retrieves the vector representation of each word in the job description from "my_model.wv".

    iii. Calculates the average vector of all words in the job description using np.mean.

    iv. Returns the average vector representation as a numpy array.

- We defined a function called "get_best_cvs(all_CVs_vectors, job_description, num_of_CVs)" that returns the indices and similarity scores of the best matching CVs for a given job description. The "all_CVs_vectors" parameter is a list of CV vectors, the "job_description" parameter is a string representing the job description, and the "num_of_CVs" parameter is an integer specifying the number of best matching CVs to return. The function proceeds as follows:

    i. Calls the "**get_job_vector**" function to obtain the vector representation of the job description.
    Calculates the cosine similarity between the query vector and all other CV vectors in "all_CVs_vectors".

    ii. Sorts the similarities in descending order and retrieves the indices of the top "num_of_CVs" best matching CVs using np.argsort.

iii. Retrieves the similarity scores corresponding to the top "num_of_CVs" best matching CVs.

iv. Returns a tuple containing the top-k indices and top-k similarities.

- We defined a function called "**filter(CVs, job_description, num_of_CVs)"** that filters the CVs based on the best matching CVs for a given job description. The function proceeds as follows:

  i. Creates a DataFrame "df" from the input CVs list.

  ii. Calculates the CV vectors for all CVs in "df" by calling "get_job_vector" for each CV.

  iii. Calls "get_best_cvs" with the CV vectors, job description, and "num_of_CVs" to obtain the top matching CV indices and similarities.

  iv. Returns a tuple containing the top matching CV indices and similarities.

- **To extract the email and mobile number from the CV**

  - The "**extract_email(text)"** function extracts email addresses from the given text. The "text" parameter is the CV text. The function proceeds as follows:

    i. Defines an email address regex pattern to match valid email addresses.

    ii. Uses the re.findall() function to find all email addresses in the text based on the regex pattern.

    iii. Joins the extracted email addresses into a string, separated by commas.

iv. Returns the string containing the extracted email addresses.

- The "**extract_mobile_number(text)**" function extracts a mobile number from the given text. The "text" parameter is the CV text. The function proceeds as follows:

  i. Defines a phone number regex pattern to match valid phone numbers.

  ii. Uses the re.findall() function to find all phone numbers in the text based on the regex pattern.

  iii. Joins the extracted phone number components into a single string.

  iv. Checks if the extracted number has a length greater than 10 (to handle cases where additional characters are matched).

  v. Returns the extracted mobile number.

Chapter 5 : Results And Analysis

We will test on machine learning engineer cv and compare between the results

5.1 Word to vector(word2vec)

Table 4: Result of word2vec

Job title	Cosine similarity
Reference Data Operations Analyst Index Instru...	0.4898211
Data Modeling Segmentation Senior Supervisor	0.491261
Senior Claims Data Analyst	0.481294
Senior Claims Data Analyst	0.478027
C Core Systems Engineer Innovative Algo Trading	0.462209

5.2 Doc2vec

Table 5: Result of doc2vec

Job title	Cosine similarity
Data Scientist Graduate Role	0.475688
Remote Data Scientist Jobs	0.469568
Remote Backend Python Developer Jobs	0.457357
Research Internship Deep Learning	0.429679
Bioinformatics Data Analyst	0.422618

5.3 LDA

Table 6: Result of LDA

score	topic
0.49358832836151123	0.020*"Design" + 0.019*"software" + 0.019*"Engineering" + 0.018*"Technical" + 0.017*"Cloud"
0.0960303470492363	0.023*"data" + 0.020*"Analysis" + 0.013*"Analytical" + 0.012*"Sql" + 0.012*"business"
0.002589344745501876	0.024*"apply" + 0.021*"project"+0.019*"manager" + 0.017*"Engineering" + 0.017*"Recruitment"
0.002587845316156745	8.842*"Marketing" + 0.038*"Sales" + 0.021*"sales"+ 0.019*"BA BS" + 0.019*"Strategy"
0.002587404102087021	0.042*"Employment Agency" + 0.018*"We" + 0.018*"Employment Business" + 0.014*"reasonable adjustments" + 0

Table 7: Result of Coherence and perplexity

Title	score
Coherence Score (Higher is better)	0.560862044222963
Perplexity Score (Lower is better)	-7.1587906117098585

5.4 TFIDF

Table 8: Result of TFIDF

Job title	Cosine similarity
Machine Learning Engineer	0.872027
Remote Senior Data Scientist Jobs	0.863887
Data Scientist	0.852373
Remote Data Scientist Jobs	0.846592
machine Learning Engineer Internship Part Time	0.846548

Table 9: Results comparison

Model Name	job title	Similarity score
Word2vec	Good	Not bad
Doc2vec	Good	Not bad
TFIDF	Excellent	Excellent

From that table we conclude that TFIDF is the best model with best results as it always gives high weights for rare words in the document

Chapter 6 : Conclusions And Recommendations

After evaluating several machine learning models for job matching and candidate evaluation, the best result was obtained using the TF-IDF technique combined with LSA for dimensionality reduction. This approach was found to be effective because TF-IDF assigns higher weights to rare words, such as job-specific skills, and LSA reduces dimensionality so the words with least importance will not be considered This led to significantly higher cosine similarity scores and more accurate job matching results.

6.1 future work

We face some challenges when we design our system like when we scrapped job descriptions from linked in there were a lot of jobs with different name but has the same content like senior data analyst and graduated data analyst, in the content they are the same but the job titles are not matching so in the future we will work to solve this problem with techniques like categorization

Furthermore, we suggest that future research in this area should focus on expanding the dataset used for training the model to include more diverse job descriptions and CVs from different industries and regions. This would help to improve the generalizability of the model and ensure that it can be effectively applied to a wider range of job matching scenarios.

Overall, By building on the insights and recommendations provided by this project, future projects can continue to push the boundaries of what is possible and create new opportunities for job seekers and employers alike.

References

1. Azizi, S. (2021, April 28). Job recommendation system using machine learning and natural language processing. https://esource.dbs.ie/handle/10788/4254
2. Enhanced DSSM (deep semantic structure modelling) technique for job recommendation. (n.d.). ScienceDirect. https://www.sciencedirect.com/journal/journal-of-king-saud-university-computer-and-information-sciences
3. Fahad, A. H. (2022, January 6). Doc2Vec — Computing Similarity between Documents - Red Buffer. Medium. https://medium.com/red-buffer/doc2vec-computing-similarity-between-the-documents-47daf6c828cd
4. G. (2022, February 4). Topic Modeling with LDA Explained: Applications and How It Works. HDS. https://highdemandskills.com/topic-modeling-intuitive/
5. GeeksforGeeks. (2020, November 26). FastText Working and Implementation. https://www.geeksforgeeks.org/fasttext-working-and-implementation/
6. Gensim: topic modelling for humans. (n.d.). https://radimrehurek.com/gensim/auto_examples/tutorials/run_doc2vec_lee.html
7. Job Boards Listings For IT Jobs. (2023, January 19). Kaggle. https://www.kaggle.com/datasets/yuanmozhu/job-boards-listings-for-it-jobs
8. Lam, P. (2021, December 16). Building a Job Recommender via NLP and Machine Learning. Medium. https://towardsdatascience.com/building-a-job-recommender-for-non-technical-business-roles-via-nlp-and-machine-learning-626c4039931e

9. Le, Q. V., & Mikolov, T. (2014). Distributed Representations of Sentences and Documents. ArXiv: Computation and Language. https://arxiv.org/pdf/1405.4053.pdf

10. Li, S. (2018, June 20). Topic Modeling and Latent Dirichlet Allocation (LDA) in Python. Medium. https://towardsdatascience.com/topic-modeling-and-latent-dirichlet-allocation-in-python-9bf156893c24

11. Resume Dataset. (2021, February 24). Kaggle. https://www.kaggle.com/datasets/gauravduttakiit/resume-dataset

12. Shperber, G. (2019, November 5). A gentle introduction to Doc2Vec - Wisio. Medium. https://medium.com/wisio/a-gentle-introduction-to-doc2vec-db3e8c0cce5e

13. V. (2023, January 29). Word2Vec Explained - Towards Data Science. Medium. https://towardsdatascience.com/word2vec-explained-49c52b4ccb71

14. Prabhakaran, S. (2022b). Cosine Similarity – Understanding the math and how it works (with python codes). *Machine Learning Plus*. https://www.machinelearningplus.com/nlp/cosine-similarity/

15. Building a Job Recommender via NLP and Machine Learning | by Preston Lam | Towards Data Science

16. freeCodeCamp.org. (2019). How to process textual data using TF-IDF in Python. *freeCodeCamp.org*. https://www.freecodecamp.org/news/how-to-process-textual-data-using-tf-idf-in-python-cd2bbc0a94a3/