Dissertation on

A Machine Learning Approach for Screening of Resume using Semantic Analysis

A thesis submitted toward partial fulfillment
Of the requirements for the degree of

Master of Technology in IT (Courseware Engineering)

Submitted by
BITHI TALUKDER

EXAMINATION ROLL NO.: M4CWE22002
UNIVERSITY REGISTRATION NO.:154484

Under the guidance of
Mr. JOYDEEP MUKHERJEE

School of Education Technology
Jadavpur University

Course affiliated to
Faculty of Engineering and Technology
Jadavpur University
Kolkata-700032
India
2022

A Machine Learning Approach for Screening of Resumes using Semantic Analysis

Course affiliated to
M.tech.IT (Courseware Engineering)
Faculty of Engineering and Technology
Jadavpur University, Kolkata, India

CERTIFICATE OF APPROVAL

This foregoing thesis is hereby approved as a credible study of an engineering subject carried out and presented in a manner satisfactory to warranty its acceptance as a prerequisite to the degree for which it has been submitted. It is understood that by this approval the undersigned do not endorse or approve any statement made or opinion expressed or conclusion drawn therein but approve the thesis only for the purpose for which it has been submitted.

--

Committee of final examination **--**
For evaluation of Thesis

--

--

A Machine Learning Approach for Screening of Resumes using Semantic Analysis

DECLARATION OF ORIGINALITY AND COMPLIANCE OF ACADEMIC ETHICS

I hereby declare that this thesis contains a literature survey and original research work by the undersigned candidate, as part of his/her **Master of Technology in IT (Courseware Engineering)** studies.

All information in this document has been obtained and presented in accordance with academic rules and ethical conduct.

I also declare that, as required by this rule and conduct, I have fully cited and referenced all materials and results that are not original to this work.

NAME: BITHI TALUKDER

EXAMINATION ROLL NUMBER: M4CWE22002

THESIS TITLE: **A Machine Learning Approach for Screening of Resumes using Semantic Analysis**

SIGNATURE: DATE:

A Machine Learning Approach for Screening of Resumes using Semantic Analysis

Acknowledgment

I feel extremely glad in presenting this thesis at the School of Education Technology, Jadavpur University, Kolkata, in partial fulfillment of the requirements for the degree of Master in Technology in IT (Courseware Engineering).

The satisfaction and euphoria that accompany the successful completion of any task would be incomplete without the mention of the people who made it possible, whose constant guidance, supervision, and encouragement crowned my effort with success.

I convey my gratitude to our guide, Mr. Joydeep Mukherjee, for his timely help, encouragement, constructive suggestion, and many more innovative ideas in carrying out the thesis.

I am also grateful to Prof. Ranjan Parekh, Director of the School of Education Technology, for his support, encouragement, and t imely advice. I am really indebted to Dr. Matangini Chattapadhyay and Dr. Saswati Mukherjee for their constant support during the entire course of the research work. Their advice and support were highly inspirational and helpful.

I would like to take this opportunity to pay my regards and thanks to all of my classmates at M.Tech.IT (Courseware Engineering) motivated me to complete my research work successfully. I do wish to thank all of those who were associated with this research work.

Thanks & Regards,

Bithi Talukder
Class Roll No: 002030402002
Exam Roll No: M4CWE22002
Registration No:154484
Masters in IT (Courseware Engineering)
School of Education Technology
Jadavpur University,
Kolkata:700032

A Machine Learning Approach for Screening of Resumes using Semantic Analysis

Contents:

Chapter 1: INTRODUCTION

1.1 Executive summary:

In today's world, the population increases day by day, and with this increasing population, the need for a job is also increasing. So, it is important for a candidate to get the right job as well as for a recruiter to get the right candidate for the company. At this point, a Resume plays an important role.

A resume can make the bridge between the recruiter and the candidate. A resume describes one's skills, experience, accomplishments, personality, etc.

The issue of hiring people who will properly fit their vacant jobs will continue to be a challenge that any company or organization will encounter regularly. Because a good fit is dependent on a variety of circumstances, this is a task with many intricacies. These include both technical skills such as a fit between the requested and provided job experience, education, and talents, as well as many soft skills such as a match between the company's personality and the applicant employee's personality.

In the first generation of the hiring system, The HR team published the vacancy detains newspapers, television, etc. Candidates sent their resumes to the company. According to the requirement, a shortlist was made by the team. After that shortlisted candidates went through a further round of interviews. This whole process is very time-consuming and hectic.

In the second generation of the hiring system, with the rapid growth of the industry, some consultancy now comes into play. Candidate sends their resumes to these consultancies in some format. Then the consultancy searches for the right candidate for the right job according to the keywords. Actually, the consultancy acts here as a middleman. This is also time- consuming and not as convenient as candidates' need to upload their resumes in a particular format.

In the third generation of the hiring system, this proposed system can accept CVs in any format and analyze the resumes using machine learning algorithms, NLP, etc. The system gives the best fit according to the job description.

1.2 Literature Survey:

In [1] Author used Natural Language Processing (NLP), Section - based segmentation, and the Natural Language ToolKit (NLTK) to compare resumes and detect similarities be tween the resumes and rank the resumes according to their capabilities.

Fouad Nasser A Al Omran, Christoph Treude used Spacy, NLTK python library, Google's syntax net , and Stanford's CoreNLP suite and finds that only a 91 percent of tokens were comparable across all four libraries, with just 64 percent of tokens receiving the same part-by-speech tag in [2].

In [3] Author improved the accuracy of extracting information from the resume using a two - step verification process: text block identification and name recognition.

A Machine Learning Approach for Screening of Resumes using Semantic Analysis

Pradeep Kumar Roy, Sarabjeet Singh, and Chowdhary Rocky Bhatia used the KNN algorithm, based on the similarity index it recommends the resumes in [4]. About 79% accuracy is given wit h the help of the Linear SVM Classifier.

In [5] An application Tracking System is used to take the resumes as input and ranked them based on two sections: Intra and Inter.

Natural Language Processing is used in [6] to extract information from resumes. The a lgorithm can evaluate applicants based on content- based suggestions that emplo y the Vector Space Model and similarity to match the resume requirements extracted from the resume with the job description requirements.

Momin Adnan et al [7] proposed a Linear SVM classifier - based approach for screening candidates for recruitment. To find the resumes that are the most similar to the one supplied, the model employed cosine similar it y, KNN to describe the task and make Recommendations based on content. The model only works with CSV format whether Most CVs are in.doc, pdf, and other formats.

A Machine Learning Approach for Screening of Resumes using Semantic Analysis

When creating a simulation with the "gensim" package, the text is compressed to the summary resulting in t he loss of crit ical information.

In [8] the author suggested an Artificial Intelligence - based Resume Sorting method. This system organizes all resumes according to the company's needs and forwards them to HR for additional review. The required résumé is chosen from a pool of candidates, with the rest being eliminated. All resumes are sorted according to the company's needs and sent to the appropriate HR department for further examination. The needed resume is chosen from a pool of applicants, with the other individuals being rejected.

Dr. K Sateesh et al [9] developed a technique that assists recruiters in quickly picking resumes based on job descriptions. It facilitates an easy and fast hiring process by automatically extracting the requirements.

Many people are applying for a single job. Job recruiters face a huge issue in selecting the most qualified profile/resume from a large pool of prospects. [10]

1.3 Objective:

The system's main goal is to elevate the present resume rating system to the next level by making it more adaptable for both parties.

- Those who have been hired.
- The client firm is the one that hires the candidates.

In the **first point** , Candidates that have re cently graduated and are looking for work. A large proportion of those individuals are so desperate that they are willing to work on any job, regardless of their skill set or competence. The main cause of unemployment is like a disease that is there in society; if a candidate does not find work after being passed out, for a year, society, including relatives, begins to blame him/her.

Despite this, the candidates are willing to work in any environment and in any position. As a result, they won't have to deal with t hose issues.

Where the system assists such applicants in being hired by a company or organization that values their abilit ies and skill sets.

In the **second** point, being an owner of a company, it would be natural that his/ her goal would be to build the best team in the world. It 's as if a C++ developer position opened up in my company. So, rather than hiring a Python developer and forcing him/ her to learn C++, I 'd rather hire a C++ developer. For both the candidate and the organization, this is a waste of t ime.

The model assists the organization in compiling a list of t he finest possible candidates based on the limits and requirements for that specific vacancy.

This t ype of strategy will aid the hiring industry in improving and being more efficient as the r ight person is hired for the r ight job. As a result, neither the client firm nor the hired candidate would have any regrets.

1.4 Problem Statement:

The issue is that the current approach is not very flexible or t ime-saving, because it is not assured that only qualified individuals would upload their resumes for a certain organization. As a result, the recruiter must sort through a large number of resumes. This method saves t ime by automatically rat ing resumes bas ed on job descriptions and finding out the best one according to the job description. Both companies and candidates will benefit from it.

Here, in this paper, the ranking of resumes is done according to the job description and the best ones will be filt er ed using machine learning techniques. This is a lit t le effort to learn how to rank the resumes according to respective job descriptions based on the existing systems and also include the future work.

The existing paper which is described in the previous chapter has some issues i. e. format of resumes is limited to CSV format which is overcome is the structure also semantic analysis is performed and with cosine similarity, Sorensen- Dice is applied to find out the best one.

Chapter 2: Methodology and Implementation

2.1 DataSet:

The database contains 65 resumes and 13 job descriptions in docs/pdf format. For privacy considerations, the dataset was filtered. There were other records that were incorrectly formatted and had to be discarded.

The dataset has 3 columns:

1. Name: It contains a unique name for each row.

2. Cleaned resumes: It contains the cleaned text of the resume after removing all punctuations and stopwords.

2. Skills: It contains the skills of the resumes.

The input resumes and job descriptions are written in English scripts as the preprocessor like stop words removal, OCR is strict ly limited to English Script.

2.2 2 Proposed Structure:

The proposed structure is divided into different stages. A block diagram of the structure is shown below in figure 1.

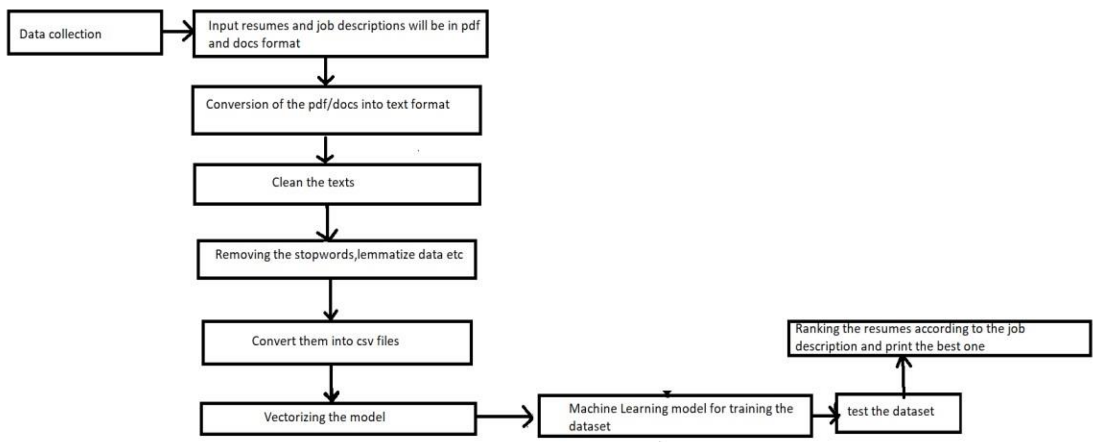

Fig 2.1: Proposed Structure

So, as per the block diagram, the complete work is divided into data processing: Cleaning, Removing Stop Words, lemmatizing data comes under it , creating a cave, Vectorizing the data, t raining the model, test ing the model, ranking the resumes, and printing the best one.

Machine learning techniques for t raining the model in a ranking problem are referred to as learning to rank. Many applications in Information Retrieval, Natural Language Processing, and Data

Mining benefit from this. The subject has been studied extensively, and great progress has been made. This brief study provides an overview of learning to rank, as well as an explanation of the fundamental issues, current methodologies, and future work in the field.

2.2.1 Data Processing:

A) PDF to text:

The goal of this project is to create an end- to- end tool that accepts a document and produces the desired outcome, in this example, the categorization and ranking of the resume according to the job description. Because the vast majority of resumes are provided in PDF/ DOCs format, we opted to include a preprocessing phase that converts PDF/DOCs to text using the well- known Optical Character Recognition. For this project, we used pytesseract in Python.

2.2.2 Data Cleaning: A) Private Information:

As the dataset contained real- world resumes, many personal details such as phone numbers, email addresses, and addresses were filtered (replaced by an 'x') for privacy considerations. This would introduce unneeded noise to the dataset and bring no value.

B) Tokenization:

Tokenization is the process of breaking down large portions of text into smaller pieces known as tokens. This is accomplished by deleting or isolating characters like whitespace and punctuation. Tokens are sentences that are divided into individual words after being tokenized out of paragraphs. We can get information like the number of words in a text, the frequency of a specific term in the text, and much more by tokenizing it by using Natural Language Toolkit [NLTK], the spaCy library, and other resources.

C) Stopwords and Punctuation:

As for punctuation and stop words did not appear to bring any value to the analysis, they were eliminated.

D) Lemmatization:

It is common to see a single English word employed in a variety of ways in distinct phrases according to the language's grammatical rules. -import, imported, and importing, for example, are all tenses of the same verb. As a result of this condition, all altered or derived variants of a word must be reduced to their central stem or base, so that these derivationally related terms with comparable meanings are not deemed dist inct. Different approaches like stemming and lemmatization to achieve the goal. ‘

The process of using a language dictionary to produce an accurate reduction of root words is known as lemmatization. Lemmatization is a more thorough process that employs language vocabulary and morphological examination of words to produce linguistically correct lemmas. This means that lemmatization makes use of context knowledge t o dist inguish between words with dist inct meanings based on parts of speech. This system employs the NLTK python package's Lemmatizer for the English language.

E) Parts of Speech (POS) tagging:

It is a method of assigning grammatical information to a word based on its context and relationship to other words in a phrase (Gelbukh, 2014). According to its usage in the sentence, the part-of-speech tag identifies whether the word is a noun, pronoun, verb, adjective, etc. This is more complicated than simply mapp ing a word to its matching part of speech tags. This is because, depending on the context in which a word is used, it may have a different part of speech.

Here, we take the Tags which are allowed by the user and then eliminate the rest of the words based o n their Part of Speech (POS) Tags.

2.2.3 Vectorizing text:

Most classifiers and learning a lgorithms require numerical feature vectors with a fixed size rather than raw text docs with variable length, they cannot handle the text documents in t heir original form. As a result, the text is changed to a more understandable representation during this step.

a) Bag of words model is a method we used for extracting features from the resumes in which t he presence (and often the frequency) of words is considered for e ach document, but the order in which they appear is ignored.

b) TF-IDF: Term Frequency – Inverse Document Frequency is abbreviated as TF-IDF. In-text mining, the TF-IDF weight is commonly used. For information retrieval and document search, the TF- IDF was developed. This weight is a numerical value that indicates how essential a term is in relation to a document in a corpus or collection. The importance of a word in a document rises in proportion to its frequency but is countered by the number of papers in which it appears. So, even if they may appear several times in a document, terms like this, and, whom, is, the, if, etc. rank low because they don't mean much to that paper. The TF- IDF value for a term in a document is calculated by multiplying two different metrics as shown in equation (1) below:

$$TF - IDF\ (t,\ d) = TF\ (t,\ d) * IDF\ (t,\ d)$$

The number of times a word appears in every document in the corpus is counted by Term Frequency. Because a term may appear more frequently in heavier papers than in lighter ones, the frequency must be adjusted. A normalized term frequency is calculated by dividing the number of times a term appears in a document by the total number of terms in that text. It can be expressed mathematically as: Mathematically it can be shown as:

$$TF\ (t,\ d) = freq\ (t,\ d)\ /\ \sum freq\ (ti\ ,\ d)$$

freq (t , d) = count of the instances of the term t in document d.

TF (t, d) = proportion of the count of term t in document d.

n =number of distinct terms in document d.

The importance of a word in a corpus of documents is determined by **Inverse Document Frequency** . The IDF value of terms that appear more frequently in t he set of papers is close to 0, whereas the IDF value of rare terms is significant. By dividing the total number of documents by the number of documents that contain a phrase, the logarithm is calculated. Mathematically, we can represent it as shown below in the equation

$$IDF(t) = log\ (\ N/\ count(t)\ )$$

N =the number of distinct documents in the corpus.

Count (t) = number of documents in the corpus in which the term t is present.

For each document, the vectors that represent the resume are created using Term Frequency and Inverse Document Frequency.

Cosine Similarity:

A similarity metric is a stat ist ic for determining how similar two objects are. The metric of cosine similarit y determines how similar two documents are regardless of their size. When plotted on an N- dimensional space, it indicates the orientation of the documents, with each dimension depicting the object's attributes. It 's a symmetrical procedure, which means the results of computing item X's similarity to item Y's are the same. It can be expressed mathematically as follows:

$$cos(\theta) = \vec{a} \cdot \vec{b} / \|\vec{a}\| \|\vec{b}\| = \sum_{i=1}^{n} a_i b_i / \sqrt{\sum_{i=1}^{n} a_i^2} \sqrt{\sum_{i=1}^{n} b_i^2}$$

$\vec{\mathbf{a}} \cdot \vec{\mathbf{b}} = \sum_{1}^{n} a_i b_i = a_1 b_1 + a_2 b_2 + \ldots + a_n b_n$ is the dot product of the two vectors.

The cosine similarity of all pairs of items is calculated using this formula. It can then be used to rank resume materials in relation to a set of query words. Cosine similarity, on the other hand, only considers aspects that are connected to the text's words and hence produce less accurate conclusions. So, we will try to improve that further.

The last phase of the proposed system is to create a content - based recommendation engine t hat uses the extracted elements from phase one to suggest the best resumes for a given job description. For calculating the similarit y between the contents of the documents, the system uses concepts such as Vectorisation importance or weight assignment techniques such as TF - IDF, as well as similarity metrics such as cosine distanc e, Sorensen- Dice, and Overlap Coefficient.

2.2.4 Overlap Coefficient:

The overlap coefficient, which is related to the Jaccard measure and quantifies the overlap between two sets, is defined as the intersection size divided by the smaller of the two sets' sizes. The overlap coefficient for two sets X and Y is overlap coefficient:

$$(X,Y)=|XY|\min(|X|,|Y|)$$

X and Y are two sets.

|XY| means the number of elements in set X and Y together

min(|X|, |Y|) means the minimum element between the X and Y set.

2.2.5 Sorensen-Dice:

The Dice similarity coefficient is a statistical tool for comparing two sets of data. It is also known as the Srensen–Dice index or simply the Dice coefficient. This index is likely the most extensively used tool for validating

AI- based photo segmentation algorithms, but it is a much larger concept that can be applied to data sets for a variety of purposes, including NLP. The equation for this concept is:

$$2 * |X \cap Y| / (|X| + |Y|)$$

X and Y are two sets.

$|X|$ means the number of elements in set X

$|Y|$ means the number of elements in set Y

$\cap$ is used to represent the intersection of two sets, and means the elements that are common to both sets.

2.2.6 Semantic Analysis:

The method of extracting meaning from text is known as semantic analysis. Examining their grammatical structures and determining the relationships between specific words in a given context, enables computers to comprehend and interpret phrases, paragraphs, or entire works.

In semantic analysis, lexical semantics is crucial because it enables computers to comprehend the connections between lexical elements (words, phrasal verbs, etc.)

The model is trained with semantically enhanced machine learning algorithms with text samples.

2.2.7 Topic Modelling:

Topic modeling is an abstract modeling technique for identifying abstract 'themes' in document sets. The idea is to undertake unsupervised classification of a variety of articles, which will lead to the discovery of certain natural subject groupings. Here we use this on the resume.

Fig 2.2 Topic Modeling

2.2.8 Latent Dirichlet Allocation:

Latent Dirichlet allocation is one of the most used approaches to topic modelling. Each document has a wide range of terminology, and each topic may be associated with specific phrases. The LDA's purpose is to use the words in the text to determine which subjects it belongs to. The usage of similar terminology is assumed in texts with similar themes. This allows the documents to map the latent theme probability distribution to the topic probability distribution.

2.2.9 WordCloud:

In a word cloud, which is a data visualization tool for visualizing text data, the size of each word symbolizes its frequency or relevance. A word cloud can be used to highlight key textual information. Word clouds are widely used to evaluate data from social networking platforms.

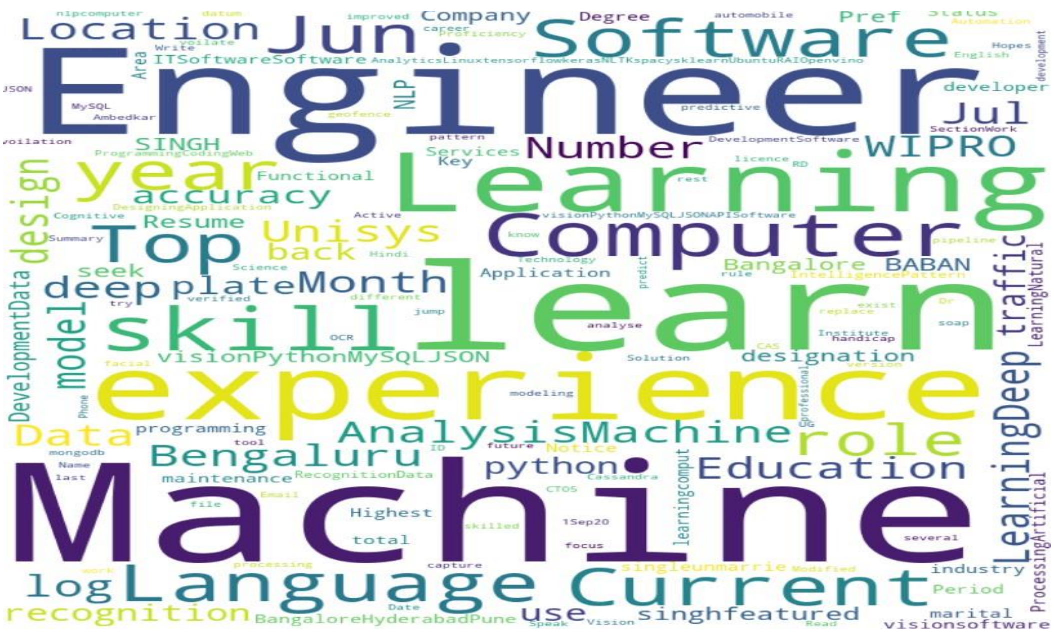

Fig 2.3 WordCloud

2.2.10 Matplotlib:

We use matplotlib to compare the acquired skill set to the job description and display the graph depending on the match score.

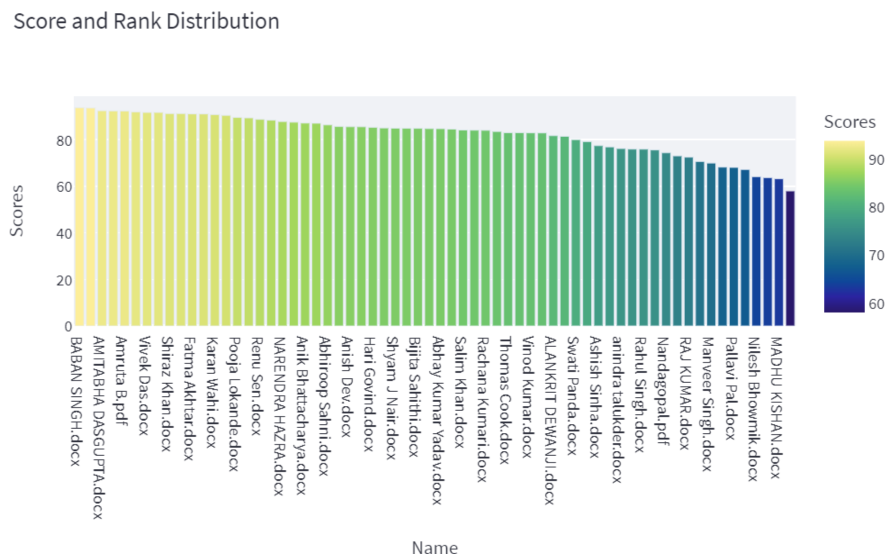

Fig 2.2 Ranking Resumes with Scores

2.2.11 Streamlit:

Streamlit is used to deploy the ranking model. Streamlit is an open-source mobile app framework built on Python. It allows us to easily create data science and machine learning web applications. It supports scikit- learn, Keras, PyTorch, SymPy (latex), NumPy, pandas, and Matplotlib, among others.

Chapter 3:

3.1 Results:

The model is now classified using KNN, Stochastic gradient descent, and Multinomial Naive Bayes. Out of these, Stochastic gradient descent gives the best result.

3.1.1 KNN:

For classification, the KNN algorithm is a supervised machine learning method. The KNN method predicts the values of new data points based on 'feature similarity,' which means that a value will be assigned to the new data point based on how closely it matc hes the points in the t raining set. At first, KNN is used as it is simple to implement, robust to noisy data, and effective if t raining data is large. It gives 94 % accuracy.

The most likely combinations of points are used for KNN classification. The Hamming, Minkowski, and Euclidean distances are all possible distance functions. In t his model, Euclidean distance formula is used.

The shortest distance between any two points, regardless of dimension, is known as the Euclidean distance. The distance between two points on a plane with the coordinates (x, y) and (a, b) is determined by the Euclidean distance formula, which is as follows:

$$\mathrm{dist}((x, y), (a, b)) = \sqrt{(x - a)^2 + (y - b)^2}$$

The input x is then assigned to the class with the highest probability once the distance has been calculated:

$$P(y=j|X=x) = 1/k \sum_{i \in A} I(y^{(i)}=j)$$

Once the distance has been computed, the input x is next assigned to the class with the highest probability:

Classification report for KNN Classifier():

	precision	recall	fi-score	support
0	1.00	1.00	1.00	3
1	1.00	0.66	1.00	3
2	0.80	0.65	0.89	5
3	1.00	1.00	0.87	9
4	1.00	0.82	0.88	6
5	0.83	1.00	0.91	5
6	1.00	0.66	0.85	9
7	1.00	0.81	1.00	7
8	1.00	0.91	0.74	11
9	1.00	1.00	0.77	9
10	1.00	1.00	1.00	8

Fig 3.1 KNN Accuracy

After that, the other models like Multinomial Naive Bayes and Stochastic gradient descent are used to compare the result. Amongst all of them Stochastic gives the best result as it gives 99% accuracy.

3.1.2 Multinomial Naïve Bayes:

The Multinomial Naive Bayes method i s a popular Bayesian learning methodology in Natural Language Processing (NLP). The programme uses the Bayes theorem to guess the label of a text, such as a piece of newspaper. It evaluates each tag's likelihood for a given sample and returns the tag with the highest chance. Here we use this model to classify the resumes. It gives 95% accuracy.

```
Classification report for classifier MultinomialNB():
              precision    recall  f1-score   support

           0       1.00      0.67      0.80         3
           1       0.75      1.00      0.86         3
           2       1.00      0.80      0.89         5
           3       1.00      1.00      1.00         9
           4       1.00      0.83      0.91         6
           5       1.00      1.00      1.00         5
           6       1.00      0.78      0.88         9
           7       1.00      1.00      1.00         7
           8       1.00      0.91      0.95        11
           9       1.00      0.67      0.80         9
          10       1.00      1.00      1.00         8
```

Fig 3.2 Multinomial Naive Bayes

3.1.3 SGDClassifier:

For fit t ing linear models, stochastic gradient descent is a simple and effective method. It 's especially useful when there are a lot of samples to go through. It has a variety of loss functions and penalties for classification.

Classification report for classifier SGDClassifier():

	precision	recall	f1-score	support
0	1.00	1.00	1.00	3
1	1.00	1.00	1.00	3
2	1.00	1.00	1.00	5
3	1.00	1.00	1.00	9
4	1.00	1.00	1.00	6
5	1.00	1.00	1.00	5
6	1.00	1.00	1.00	9
7	1.00	1.00	1.00	7
8	1.00	0.91	0.95	11
9	1.00	1.00	1.00	9
10	1.00	1.00	1.00	8

Fig 3.3 Stochastic Gradient Descent

Precision, F-score, and recall are also calculated here:

Precision:

Precision informs us how many of all t he occurrences that were expected to be in class Y were actually in class Y. For class Y, the precision is calculated as follows:

TP/ TP+FP

TP means the number of true positives for class Y

FP means the number of false positives for class Y

Recall:

The recall indicates how many instances of class Y. Y were successfully anticipated.

TP/ TP+FN

TP means the number of true positives for class Y.

FN means the number of false negatives for class Y.

F-score:

The F1- score takes the harmonic mean of a classifier's precision and recall to create a single stat ist ic. It 's mostly used to compare the results of two different classifiers. Assume that classifier A has a higher recall and precision than classifier B. Th e F1- scores for both classifiers can be used to identify which delivers superior results in this scenario. A classification model's F1 - score is calculated as follows:

$$2(P*R)/P+R$$

P means precision

R means recall of the classification model

Support:

The amount of real instances of the class in the given dataset is known as support. The requirement for strat ified sampling or rebalancing may be indicated by unbalanced support in the t raining data, which may point to structural flaws in the classifier's reported scores.

3.2 Difference between the existing approach and the proposed approach:

Existing Paper	Proposed Paper
Only CSV format of the resumes is accepted	All formats of the resumes are accepted
Dataset of 20 resumes	Dataset of 100 Resumes
Semantic Analysis is not used	Semantic Analysis is used.
KNN classifier is applied and it gives 78 % accuracy	KNN classifier is applied and it gives 94 % accuracy

To improve the model's accuracy, we have also checked the model with the Multinomial Naïve Bayes classifier and stochastic gradient descent classifier. MNB gives 95% accuracy and SGDClassifier gives 99% accuracy. So SGDClassifier seems the best fit for the model.

Comparison graph:

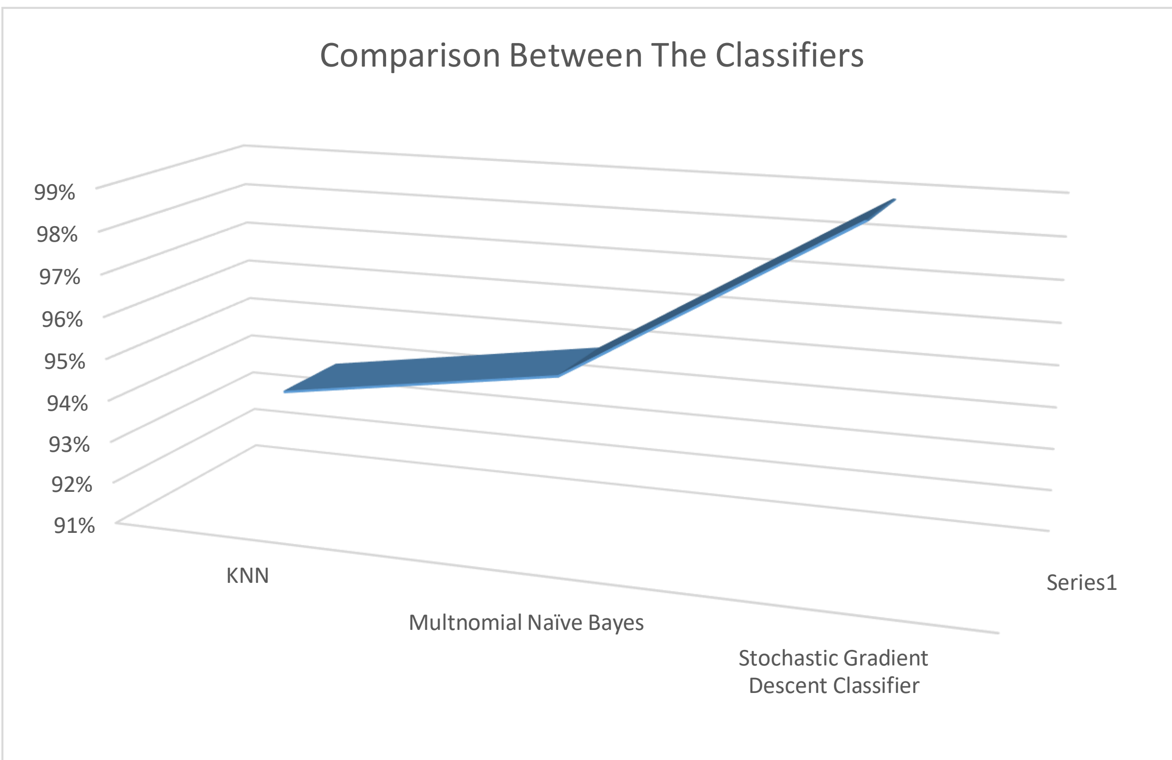

Fig 3.4 Comparison graph

Chapter 4:

4.1 Conclusion :

In this study, A resume ranking system using machine learning is introduced that streamlines the e- recruitment process by removing the numerous issues that recruiters encountered when they relied on human shortlisting of candidates for a particular job post. The system functions on two levels. First, it emplo ys tokenization and other perimeters to pull pertinent data from the resumes ' varied and unstructured formats. It produces a condensed version of each resume that only contains the information necessary for the selection process.

On the other hand, the approach of this project allows the ranking of applications by ut iliz ing similarity matrices. So the best fit for the respective job description can get easily.

4.2 Future Scope:

The model is mainly focused on IT companies. The proposed system also can be experimented on datasets of other sectors like health, school, etc.

A feedback feature can be added where the recruiter can also send feedback to the respective candidate.

Using Deep Learning Models like Long-Short Term Memory, Recurrent Neural Network, Convolutional Neural Network, and others to enable the semantic translation of terms found in resumes and job descriptions could improve the model's performance even further. Similar to this, it would be intriguing to train a neural language model on the resumes of applicants who were chosen for their soft skills, which are frequently not included in job descriptions but are important to the firm.

Chapter 5:

References:

[1] S. Amin, N. Jayakar, S. Sunny, P. Babu, M. Kiruthika and A. Gurjar, "Web Application for Screening Resume," 2019 International Conference on Nascent Technologies in Engineering (ICNTE), Navi Mumbai, India, 2019, pp. 1-7, doi: 10.1109/ICNTE44896.2019.8945869.

[2] Fouad Nasser A Al Omran, Christoph Treude," Choosing an NLP Library for Analyzing Software Documentation: A Systematic Literature Review and a Series of Experiments" in 2017 IEEE/ACM 14th International Conference on Mining Software Repositories.

[3] Jie Chen, Chunxia Zhang, and Zhendong Niu "A Two-Step Resume Information Extraction Algorithm", Volume 2018.

[4] Pradeep Kumar Roy Sarabjeet Singh Chowdhary RockyBhatia, "A Machine Learning approach for automation of Resume Recommendation system", International Conference on Computational Intelligence and Data Science (ICCIDS 2019).

[5] Jagadish P, Abhishek V, Anant Shukla, Anuj V, Prasanth Kumar Reddy K, "Resume Screening and Ranking with spaCy" in Turkish Online Journal of Qualitative Inquiry (TOJQI)Volume 12, Issue 7, July 2021: 9116-9123.

[6] Chirag Daryania, Gurneet Singh Chhabrab, Harsh Patel (2020) , Indrajeet Kaur Chhabrad, Ruchi Patele , "AN AUTOMATED RESUME SCREENING SYSTEM USING NATURAL LANGUAGE PROCESSING AND SIMILARITY" in Intelligent Computing and Industry Design (ICID) 2(2) (2020) 99-103.

[7] Momin Adnan, Gunduka Rakesh, Juneja Afza, Rakesh Narsayya Godavari, Gunduka and Zainul Abideen Mohd Sadiq Naseem., "Resume Ranking using NLP and Machine Learning", (2016b). Institutional Repository of the Anjuman-I-Islam's Kalsekar Technical Campus. https://core.ac.uk/display/55305289

[8] V. V. Dixit , Trisha Patel , Nidhi Deshpande , Kamini Sonawane, "Resume Sorting using Artificial Intelligence". (2019). International Journalof Research in Engineering, Science and Management Volume-2, Issue-4.

[9] Dr.K.Satheesh, A.Jahnavi, L Aishwarya, K.Ayesha, G Bhanu Shekhar, K.Hanisha, "Resume Ranking based on Job Description using SpaCy NER model". (2020). International Research Journal of Engineering and Technology.

[10] Breaugh, J.A., 2009. The use of biodata for employee selection: Past research and future directions. Human Resource Management Review 19, 219–231.

[11] Zhang, L.,Fei, W. ,Wang ,L.,2015.Pj matching model of knowledge workers.Procedia Computer Science 60,1128–1137.

[12] Roy, P.K., Singh, J.P., Baabdullah, A.M., Kizgin, H., Rana, N.P., 2018a. Identifying reputation collectors in community question answering (cqa) sites: Exploring the dark side of social media. International Journal of Information Management 42, 25–35

data. In SIGMOD Conference, pages 1005–1010, 2009.

Appendix:

Code Snippets:

The source code can be verified using the Github link: https://github. com/b7626

FileReader

```
from operator import index

from pandas. _config. config import options

import Cleaner

import textract as tx

import pandas as pd

import os

import tf_idf


resume_dir = "Data/Resumes/"

job_desc_dir = "Data/JobDesc/"

resume_names = os. listdir( resume_dir)

job_description_names = os. listdir( job_desc_dir)


document = []
```

```
def read_resumes(list_of_resumes, resume_directory):
    placeholder = []
    for res in list_of_resumes:
        temp = []
        temp.append(res)
        text = tx.process(resume_directory+res, encoding='ascii')
        text = str(text, 'utf-8')
        temp.append(text)
        placeholder.append(temp)
    return placeholder

document = read_resumes(resume_names, resume_dir)

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

```
        document[i].append(sentence)
    return document


Doc = get_cleaned_words(document)


Database = pd.DataFrame(document, columns=[
                        "Name", "Context", "Cleaned", "Selective",
"Selective_Reduced", "TF_Based"])


Database.to_csv("Resume_Data.csv", index=False)


# Database.to_json("Resume_Data.json", index=False)


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

```
        placeholder.append(temp)
    return placeholder

job_document = read_jobdescriptions(job_description_names,
job_desc_dir)

Jd = get_cleaned_words(job_document)

jd_database = pd.DataFrame(Jd, columns=[
                         "Name", "Context", "Cleaned", "Selective",
"Selective_Reduced", "TF_Based"])

jd_database.to_csv("Job_Data.csv", index=False)

Resumes = pd.read_csv('edit.csv')
Jobs = pd.read_csv('Job_Data.csv')
```

Cleaning:

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

```
from nltk.corpus import stopwords
# Define english stopwords
stop_words = stopwords.words('english')


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



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



def tokenize(text):
```

```
    # Removes any useless punctuations from the text
    text = re.sub(r'[^\w\s]', '', text)
    return word_tokenize(text)



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

# internal fuction, useless right now.


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

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

LabelEncoding:

```
from sklearn.preprocessing import LabelEncoder

var_mod = ['Category']
le = LabelEncoder()
for i in var_mod:
   Resumes [i] = le.fit_transform(Resumes[i])
```

Features Extraction:

```
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from scipy.sparse import hstack

requiredText = resumeDataSet['cleaned_resume'].values
requiredTarget = resumeDataSet['Category'].values

word_vectorizer = TfidfVectorizer(
    sublinear_tf=True,
    stop_words='english',
    max_features=1500)
word_vectorizer.fit(requiredText)
WordFeatures = word_vectorizer.transform(requiredText)

print ("Feature completed...... ")
```

```
X_train, X_test, y_train, y_test =
train_test_split( WordFeatures, requiredTarget, rando m_state=0,
test_size=0. 4)
print( X_train. shape)
print( X_test. shape)
```

Classifiers:

KNN:

```
clf = OneVsRest Classifier( KNeighbors Classifier())
clf. fit( X_train, y_train)
prediction = clf. predict( X_test)
print('Accuracy of KNeighbors Classifier on training set:
{:. 2f}'. format( clf. score( X_train, y_train)))

print('Accuracy of KNeighbors Classifier on test set:
{:. 2f}'. format( clf. score( X_test, y_test)))

print("\n Classification report for classifier %s:\n%s\n" % (clf,
metrics. classification_report( y_test, prediction)))
```

Multinomial Naive Bayes:

```
from sklearn.naive_bayes import MultinomialNB

nb = MultinomialNB()

nb.fit(X_train, y_train)
y_pred_class = nb.predict(X_test)

from sklearn import metrics
metrics.accuracy_score(y_test, y_pred_class)

print("\n Classification report for classifier %s:\n%s\n" % (nb,
metrics.classification_report(y_test,y_pred_class)))
```

Stochastic gradient descent:

```
from sklearn.linear_model import SGDClassifier
clf= SGDClassifier()

clf.fit(X_train,y_train)
```

```
y_pred_class = clf.predict(X_test)
metrics.accuracy_score(y_test, y_pred_class)
print('Accuracy of SGD Classifier on training set:
{:.2f}'.format(clf.score(X_train, y_train)))
print('Accuracy of SGD Classifier on test set:
{:.2f}'.format(clf.score(X_test, y_test)))

print("\n Classification report for classifier %s:\n%s\n" % (clf,
metrics.classification_report(y_test, y_pred_class)))
```

TF-IDF:

```
from sklearn.feature_extraction.text import TfidfVectorizer
def do_tfidf(token):
    tfidf = TfidfVectorizer(max_df=0.05, min_df=0.002)
    words = tfidf.fit_transform(token)
    sentence = " ".join(tfidf.get_feature_names())
    return sentence
```

```
def get_list_of_words(document):
    Document = []

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

    return Document

document = get_list_of_words(Resumes['Cleaned_data'])

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


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

LDA:

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

    return sent_topics_df
```

Topic Modelling:

```
cols = [color for name, color in
mcolors.TABLEAU_COLORS.items()]
cloud = WordCloud(background_color='white',
                  width=2500,
                  height=1800,
                  max_words=10,
                  colormap='tab10 ',
                  collocations=False,
                  color_func=lambda *args, **kwargs: cols[i],
                  prefer_horizontal=1.0)

topics = lda_model.show_topics(formatted=False)

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

for i, ax in enumerate(axes.flatten()):
    fig.add_subplot(ax)
    topic_words = dict(topics[i][1])
    cloud.generate_from_frequencies(topic_words,
max_font_size=300)
```

```
    plt.gca().imshow(cloud)
    plt.gca().set_title('Field ' + str(i), fontdict=dict(size=16))
    plt.gca().axis('off')
plt.subplots_adjust(wspace=0, hspace=0)
plt.axis('off')
plt.margins(x=0, y=0)
plt.tight_layout()
st.pyplot(plt)

st.markdown("---")



import textdistance as td


def match(resume, job_des):
    j = td.jaccard.similarity(resume, job_des)
    s = td.sorensen_dice.similarity(resume, job_des)
    c = td.cosine.similarity(resume, job_des)
    o = td.overlap.normalized_similarity(resume, job_des)
    total = (j+s+c+o)/4
    # total = (s+o)/2
```

```
return total*100
```

Ranking:

CALCUATION OF SCORE:

```
def calculate_scores(resumes, job_description):
    scores = []
    for x in range(resumes.shape[0]):
        score = Similar.match(
            resumes['skills'][x], job_description['skills'][index])
        scores.append(score)
    return scores
Resumes['Scores'] = calculate_scores(Resumes, Jobs)


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

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

SCORE TABLE PLOT :

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

fig1.update_layout(title="Top Ranked Resumes", width=700,
height=1100)
st.write( fig1)
st.markdown("---")
```

```
fig2 = px.bar(Ranked_resumes,
          x=Ranked_resumes['Name'], y=Ranked_resumes['Scores'],
color='Scores',
          color_continuous_scale='haline', title="Score and Rank
Distribution")
# fig.update_layout(width=700, height=700)
st.write(fig2)
st.markdown("---")
Print best match:
option_2 = st.selectbox("Show the Best Matching Resumes?",
options=[
  'YES', 'NO'])
if option_2 == 'YES':
  indx = st.slider("Which resume to display ?:",
               1, Ranked_resumes.shape[0], 1)
```

```
    st.write("Displaying Resume with Rank: ", indx)

    st.markdown("---")

    st.markdown("## **Resume** ")

    value = Ranked_resumes.iloc[indx-1, 2]

    st.markdown("#### The Word Cloud For the Resume")

    wordcloud = WordCloud(width=800, height=800,

                          background_color='white',

                          colormap='viridis', collocations=False,

                          min_font_size=10).generate(value)

    plt.figure(figsize=(7, 7), facecolor=None)

    plt.imshow(wordcloud)

    plt.axis("off")

    plt.tight_layout(pad=0)

    st.pyplot(plt)



    st.write("With a Match Score of :", Ranked_resumes.iloc[indx-1,
6])
```

```
fig = go.Figure(data=[go.Table(
    header=dict( values=["Resume"],
                fill_color='#f0a500',
                align='center', font=dict(color='white', size=16)),
    cells=dict(values=[str(value)],
               fill_color='#f4f4f4',
               align='left'))])


fig.update_layout(width=800, height=1200)
st.write( fig)
# st.text(df_sorted.iloc[ indx-1, 1])
st.markdown("---")
```