JETIR.ORG ISSN: 2349-5162 | ESTD Year : 2014 | Monthly Issue

JOURNAL OF EMERGING TECHNOLOGIES AND INNOVATIVE RESEARCH (JETIR)

An International Scholarly Open Access, Peer-reviewed, Refereed Journal

Resume Shortlisting and Grading using TF-IDF, Cosine Similarity and KNN

[1]Samiksha Borgave, [2]Vaishnavi Gavali, [3]Sarvesh Kudumbale, [4]Dr. Saurabh Saoji

[1]Student, [2]Student, [3]Student, [4]Head of Department
Department of Computer Engineering
[1]Nutan Maharashtra Institute of Engineering & Technology, Pune, India

Abstract : Implementing automation can greatly increase the process's speed and efficiency, cutting down on the time it takes to hire someone and allowing a bigger pool of candidates to be assessed more quickly. Additionally, the use of automation can free up HR experts to concentrate on more strategic activities like corporate branding and candidate engagement. Overall, this paper makes the case that businesses that embrace automation have a better chance of identifying and employing top people in the highly competitive labour market of today.

Index Terms - NLP, TF-IDF, Tokenization, Lemmatization, Stemming, Cosine Similarity, Text Pre-processing, Text Extraction.

I. INTRODUCTION

These days, having a recruitment strategy for every organization is important. It is important for finding the best candidates. Resume shortlisting, which entails evaluating resumes to choose the most suitable candidates, is a crucial step in this process [1]. Traditional methods for shortlisting resumes, however, might take a lot of time and risk leaving out people who are competent [2] [13]. Fortunately, various automated technologies these days let you scan a resume and tell you the result.

The research's suggested methodology uses text processing methods and TF-IDF to increase the effectiveness of resume shortlisting [15]. To pre-process the resume content and extract the most pertinent elements, this technique involves tokenization, stop-word elimination, and stemming (NLP) [16]. The degree of similarity between the job description and the resume is then assessed using TF-IDF on these features. The resumes are then ranked according to similarity scores to determine the best candidates for the job [3].

Real-world data were used to assess the proposed model and compare it to more established methods like manual screening and keyword-based matching [5]. The outcomes showed that the model performed better than traditional methods, with greater accuracy and fewer false positives. The strategy enables companies to locate eligible applicants quickly and effectively, offering an efficient and cost-effective answer to the hiring process.

This proposed method describes how text processing techniques like TF-IDF can boost resume shortlisting's effectiveness as well as the possibilities of automated resume shortlisting models in improving the recruiting process. The study shows how automated resume shortlisting models can aid in locating the most qualified applicants for the position and help ongoing attempts to enhance the recruiting process.

II. RELATED WORKS

An important part of the hiring process is the text processing and resume parsing technique used in the smart resume shortlisting method [14]. The technique entails the automatic extraction of pertinent data from resumes, allowing recruiters to swiftly discover candidates who are qualified for a specific job post [2] [1]. The three algorithms of TF-IDF, cosine similarity, and KNN have all been used in recent years. These algorithms have helped to increase the efficiency of resume parsing and text processing in the context of the smart resume shortlisting strategy.

The statistical method TF-IDF (Term Frequency-Inverse Document Frequency) is used to assess a word's significance in a document [4]. This strategy of identifying keywords and phrases that are pertinent to a specific employment role is utilized in the smart resume shortlisting process. TF-IDF weights each keyword. It completely depends on how frequently it is present in the document. It is additionally determined based on how uncommon it is in the documents. The significance of the word in the context of the job position is then determined using this weight. In the shortlisting process, words with higher TF-IDF scores are given greater weight since they are viewed as being more significant [6].

Many studies in recent years have concentrated on the employment of the TF-IDF, cosine similarity, and KNN algorithms in resume parsing and text processing for smart resume shortlisting. Numerous researchers have looked into how well these methods work to increase the precision and efficacy of the hiring process [7].

In one study, cosine similarity and the TF-IDF were used to shortlist resumes for positions in the information technology (IT) sector. The study discovered that by locating pertinent keywords and phrases in the resumes and job descriptions, these strategies

increased the accuracy of the shortlisting process. According to the authors, the use of cosine similarity also proved useful in locating applicants who had the necessary qualifications for the position. Basically, the purpose of cosine similarity is to find similarities between two components [8].

Another study looked into the application of the KNN algorithm for smart resume shortlisting [9]. According to the study, the implementation of the KNN algorithm increased the shortlisting process' accuracy by categorizing resumes into separate groups according to how closely they matched the job role. The authors also discovered that employing the KNN algorithm made it easier to recognize applicants with more training and expertise.

Texts are categorized using the machine learning method KNN (K-Nearest Neighbors) depending on how closely they resemble other texts. KNN is used in the smart resume shortlisting approach to divide resumes into various groups according to how relevant they are to the job role [11]. To discover patterns and connections between various kinds of information, the system first trains the KNN algorithm on a sizable dataset of resumes and job descriptions [10]. Once trained, the algorithm can be used to group new resumes into categories based on how much they represent resumes from the training dataset. The system then shortlists the resumes that are most comparable to the job description for further examination using the KNN algorithm [7].

A study looked at how well the smart resume shortlisting approach performed when the TF-IDF, cosine similarity, and KNN algorithms were combined [12]. By identifying pertinent terms and phrases in the resumes and job descriptions, assessing how closely the two align, and categorizing resumes according to how well they fit the job role, the study found that the use of these techniques in combination increased the shortlisting process' accuracy.

In general, research on the application of the TF-IDF, cosine similarity, and KNN algorithms in resume parsing and text processing for smart resume shortlisting has shown encouraging results in increasing the precision and effectiveness of the hiring process. These methods could speed up the shortlisting procedure and cut down on the amount of time and money required to assess applicants. To determine whether these approaches are effective in other industries and job functions, additional research is required.

In conclusion, the effectiveness of resume parsing and text processing in the smart resume shortlisting method has been greatly improved by the employment of advanced techniques including TD-IDF, cosine similarity, and KNN algorithms. With the aid of these methodologies, the system is able to extract the most pertinent information from resumes, assess how closely job descriptions and resumes resemble one another, and group resumes according to how closely they relate to particular job roles. In the end, organizations are able to identify suitable candidates for a certain job role swiftly and effectively, saving time and money on the candidate shortlisting process.

III. PROBLEM STATEMENT

The company and its recruiters face a challenge in hiring a large number of employees to achieve organizational growth. The task of reviewing and shortlisting resumes for the best matches is tedious and time-consuming. The high volume of job applications and the need for manual computation between applicants result in wasted human resources and unmanageable work.

IV. OBJECTIVES

- To develop an automated system that can efficiently shortlist resumes and identify the best candidates for the company's needs.
- To save time and resources by eliminating the need for manual computation and review of each resume.
- To make certain that the hiring procedure is organized, effective, and able to find top candidates for the expansion of the business.

V. METHODOLOGY

To perform resume shortlisting and grading, the following steps are taken:

1. Text Parsing
2. Stop word removal
3. Lemmatization
4. TF-IDF
5. Cosine Similarity
6. KNN Classifier

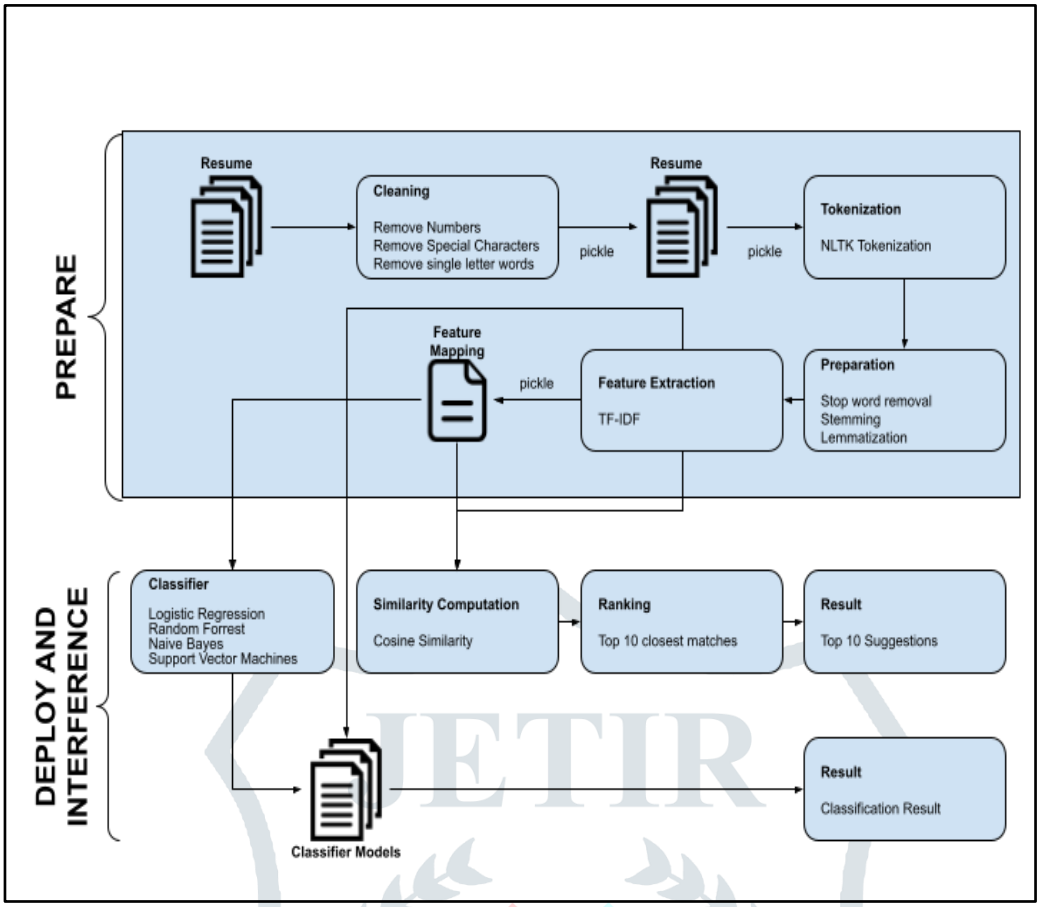


Fig 1: Architecture of Resume shortlisting system using TF-IDF, Cosine Similarity and KNN.

5.1. Text Parsing

Text parsing from PDF files to strings is a crucial component in resume shortlisting. PDF files are commonly used for documents, including resumes and job descriptions, and extracting text from these files allows for further processing and analysis. The PyPDF2 library is a popular Python package for parsing PDF files and extracting text from them.

To start parsing a PDF file, the PyPDF2 library is imported into the Python environment. Then, a PDF file is opened using the "open("" function and the "PdfReader()" class from the PyPDF2 library. The "PdfReader(")" class takes the opened PDF file as an argument and creates an object of that file.

Next, the text from each page of the PDF file is extracted using a loop. To determine how many pages there are in the entire PDF file, use the "len(reader.pages)" method from the "PdfReader()" class. A "for" loop is subsequently utilized to iterate over each page in the PDF file and extract the text from it using the "extract_text()" method. Each page's extracted text is added to a string variable that holds the PDF file's entire text.

After parsing the PDF file into a string, the extracted text can be further processed and analyzed. For instance, when shortlisting resumes, the text can be tokenized to separate it into individual words, and stop words can be eliminated to get rid of generic terms that don't add to the text's meaning.

5.2. Stop word Removal

After converting the PDF file to a text string, the next step in the text processing pipeline is typically stop word removal. Stop words are words that are frequently used in the English language but typically do not add much value to the meaning of the text, such as "the," "a," and "and." Removing these stop words can help reduce noise in the text and improve the accuracy of natural language processing models.

Stop-word removal functionality is offered by a number of libraries in Python. The Natural Language Toolkit (NLTK) is one such library that offers a list of stop words for many languages, including English. The steps for stop word removal using NLTK are:

a. Import the necessary library for stop word removal:
import nltk

b. Get the list of stop words in your preferred language here:
nltk.download('stopwords')

c. Create a set of stop words:
from nltk.corpus import stopwords stop_words = set(stopwords.words('english'))

d. Tokenize the text string into a list of words:
import re text = re.findall(rw+', text_string)

e. Eliminate the following words from the list:

filtered_text = [word for word in text if not word in stop_words]

After these steps, the resulting "**filtered_text**" variable will contain the list of words from the original text string with the stop words removed. These steps can be repeated for both the resume and job description text strings to ensure that they are both processed in the same way.

It is worth noting that different libraries and methods for stop word removal may provide slightly different results, as the list of stop words can vary and some methods may remove more or fewer words than others. Furthermore, some words that are stop words in one context might be crucial in another, so it's crucial to take the needs of the text processing pipeline into account when choosing a stop word removal method.

5.3. Lemmatization

After the stop words have been removed, the next step in preprocessing the text is lemmatization. Lemmatization is the reduction of words to their lemma, which is their fundamental or root form. By grouping related words together, this reduces the data's dimensionality.

Lemmatization can be done in Python using the NLTK (Natural Language Toolkit). The first step is to tokenize the words using the NLTK tokenizer. The part of speech for each word is then determined using a part-of-speech (POS) tagger. The lemmatizer must be aware of the part of speech in order to correctly identify the lemma, so this is significant.

Once the POS tags have been identified, the NLTK lemmatizer can be used to lemmatize the words. The lemmatizer takes two arguments: the word and the part of speech. In the absence of a part of speech, the lemmatizer assumes it is a noun.

Once the words have been lemmatized, it's crucial to get rid of any last bits of punctuation and digits that might still be there. In Python, regular expressions can be used to accomplish this.

Overall, the steps for lemmatization after stop word removal are as follows:

1. Tokenize the words using the NLTK tokenizer.
2. Utilize a POS tagger to determine the parts of speech for each word.
3. Lemmatize the words using the NLTK lemmatizer, specifying the part of speech for each word.
4. Remove any remaining punctuation and numerical characters using regular expressions.

After stop words are eliminated, lemmatization is carried out to make the text data cleaner and more dependable. This can help text processing algorithms like TF-IDF and cosine similarity produce more accurate results.

5.4. TF-IDF

TF-IDF is a widely used algorithm for natural language processing tasks such as text classification, text clustering, and information retrieval. In the context of resume shortlisting, it is used to extract relevant keywords and phrases from resumes and job descriptions to compute their similarity scores.

The first step in applying TF-IDF is to create a document-term matrix that represents the frequency of each term in each document. In the case of resumes and job descriptions, each term represents a lemma obtained through the lemmatization process. The document-term matrix can be visualized as a table with a row for each document and a column for each term.

After creating the document-term matrix, the next step is to apply the TF-IDF formula to compute the weight of each term in each document. Inverse document frequency (IDF) and term frequency (TF) make up the TF-IDF. While the term frequency tracks how frequently a term appears in a document, the inverse document frequency assesses how much information a term provides across all documents. The formula for computing TF-IDF is as follows:

$$TF\text{-}IDF(term, document) = TF(term, document) * IDF(term)$$

were

TF (term, document) is the frequency of the term in the document.

IDF (term) is the inverse document frequency of the term across all documents.

In Python, the TF-IDF algorithm can be implemented using the scikit-learn library. The steps involved in applying TF-IDF to resumes and job descriptions are as follows:

1. Create a list of documents, where each document is a string containing the text of a resume or job description.
2. Create a "**TfidfVectorizer**" object, which is responsible for creating the document-term matrix and applying the TF-IDF formula.
3. Fit the "**TfidfVectorizer**" object to the list of documents using the "**fit_transform**" method. This method creates the document-term matrix and determines the weights of the TF-IDF for every term.
4. Get the feature names, which are the terms in the document-term matrix, using the "**get_feature_names**" method.
5. Extract the TF-IDF weights for each document and each term using the two-array method of the document-term matrix.

The resulting output is a matrix of TF-IDF weights, in which every row is associated with a document and every column corresponds to a term. This matrix can be used to compute the cosine similarity between pairs of documents, which is a measure of how similar they are based on the TF-IDF weights of their terms.

The equation is:

$$TF\text{-}IDF\ score\ for\ term\ i\ in\ document\ j = TF(i,j) * IDF(i)$$

Where

IDF = Inverse Document Frequency

TF = term frequency

TF(i,j) = frequency of term i in any given file j/ Total words in file j

IDF(i,j) = log2(Total documents / documents with term i)
t = term
j = document

5.5. Cosine Similarity

Cosine similarity is a metric that is utilized between two vectors in a high-dimensional space. It is a widely used technique in natural language processing and information retrieval to find the similarity between documents based on their vector representation. In the context of resume shortlisting, cosine similarity is used to compare the similarity between the job description and the candidate's resume.

The text data is converted into a matrix format using the TF-IDF technique after text preprocessing procedures like stop word removal, lemmatization, and tokenization have been completed. The matrix contains the frequency of occurrence of each term in the text data along with a weight given to it in accordance with its significance in the file and the entire corpus.

Once the matrix is obtained, cosine similarity is used to find the similarity between the job description and the candidate's resume. The cosine similarity value can be anywhere between 0 and 1, with 0 denoting complete similarity and 1 denoting complete similarity between the two vectors.

The two vectors' dot products are calculated, and their magnitude products are divided to determine the cosine similarity. The cosine of the angle formed by the two vectors is provided, and it illustrates how similar they are to one another. Alternatively, the cosine similarity value represents the cosine of the angle formed by the two vectors in vector space.

Utilizing the scikit-learn library, one can calculate the cosine similarity in Python. The "**TfidfVectorizer**" function from Scikit-Learn is used to convert the text data into a matrix format using the TF-IDF technique. The cosine similarity between the two vectors is determined using the same library's "cosine_similarity" function.

The steps involved in calculating cosine similarity are as follows:

1. Convert the job description and candidate's resume into vector representations using the "**TfidfVectorizer**" function.
2. Use the "cosine_similarity" function to determine how similar the two vectors are to one another in cosine terms.
3. Obtain the cosine similarity value, which ranges from 0 to 1.
4. A higher cosine similarity value indicates a higher degree of similarity between the job description and the candidate's resume, which can be used to shortlist the candidates.

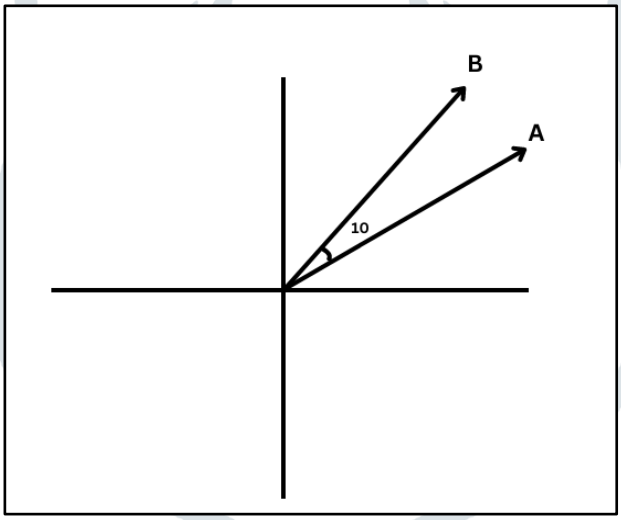


Fig 2: Cosine Similarity- vector illustration.

A = job description
B = applicant’s resume

The angle between vectors A and B is 10 degrees.
Cos(10) = 0.9848….

The angles could be said to be 98% similar between the job description and the applicant’s resume.

5.6. KNN Classifier

After obtaining the cosine similarity scores between the resumes and job descriptions, the next step is to train a KNN classifier using these scores as input features and the job role as the output label. The supervised machine learning algorithm KNN (k-nearest neighbors) is frequently employed for classification tasks.

The KNN classifier cannot be trained without first splitting the dataset into training and testing sets. The testing set will be used to assess the KNN classifier's performance after it has been trained using the training set. Typically, data is split 80:20, with 80% going to training and 20% going to testing.

Next, the cosine similarity scores between the resumes and job descriptions are used as input features for the KNN classifier. These scores are used to represent the similarity between each resume and the job description.

The job roles are used as the output label for the KNN classifier. Each job role is assigned a unique label, and the KNN algorithm will use the input features (cosine similarity scores) to predict the corresponding job role.

The testing set is used to assess the KNN classifier after it has been trained. Utilizing metrics like accuracy, precision, recall, and F1-score, the classifier's performance is assessed. These metrics show the degree to which the cosine similarity scores can be used by the classifier to predict the correct job role.

An appropriate KNN classifier implementation for this task is offered by the scikit-learn library in Python. You can use the following steps to implement the training and evaluation of the KNN classifier:

1. Split the dataset into training and testing sets using a function such as **"train_test_split"** from the scikit-learn library.
2. Calculate the cosine similarity scores between each resume and job description using the process described earlier.
3. Use the cosine similarity scores as input features and the job roles as output labels to train the KNN classifier using the **"KNeighborsClassifier"** function from the scikit-learn library.
4. Evaluate the performance of the KNN classifier using the testing set. This can be done using functions such as **"accuracy_score," "precision_score," "recall_score,"** and **"f1_score"** from the scikit-learn library.

4.1 Results and Analysis:

The result obtained from this proposed system model is that resumes are ranked in ascending order by the score of resumes matching the job description. The algorithms used for the process are the cosine similarity and knn algorithms.

A classification algorithm like KNN can have its accuracy and efficiency evaluated using a tool called the confusion matrix. It gives a thorough breakdown of the classifications that the algorithm made for true positives, false positives, true negatives, and false negatives.

In the confusion matrix, the true positive (TP) element denotes the proportion of instances that are accurately categorized as positive. For KNN, this would be the number of resumes correctly identified as applicable to the position.

The percentage of instances that are falsely classified as positive is shown by the false positive (FP) element of the confusion matrix. This would be the number of resumes that, according to KNN, are misclassified as being pertinent to a particular job role.

In the confusion matrix, the true negative (TN) element denotes the proportion of instances that are truly classified as negative. This would be the number of resumes that KNN correctly identifies as unfit for the position.

The confusion matrix's false negative (FN) element shows how many situations are mistakenly labeled as negative. In the case of KNN, this would be the quantity of resumes that are falsely excluded from consideration for a position despite their appropriateness.

Accuracy, precision, recall, and F1-score are performance metrics that are computed using the confusion matrix to assess the efficacy of the KNN algorithm. Recall is the ratio of true positives to the total of true positives and false positives; precision is the ratio of true positives to the total of true positives and false negatives; and accuracy is the harmonic mean of precision and recall. The F1 score is the result of these calculations.

Recruiters can assess the efficiency of the KNN algorithm in shortlisting resumes and pinpoint areas for improvement by analyzing the confusion matrix and computing these performance metrics. Using the confusion matrix, recruiters can choose candidates based on data-driven decisions that are provided by a thorough breakdown of the algorithm's performance.

Accuracy_score = (TN+TP)/(TN+FP+FN+TP)
Precision_score = TP/(FP+TP)... (predicted yes)
Recall_score = TP/(FP+FN)….(Actual Yes)
f1_score = (2*Recall*Precision) / (Recall + Precision)

		Predicted Class	
		No	Yes
Observed Class	No	TN	FP
	Yes	FN	TP

Fig 3: Confusion matrix- evaluation technique for KNN classifier algorithm

Where ,
TN = true negative
FN = False Negative
TP = True Positive
FP = False Positive

The approach is to use TF-IDF, cosine similarity, and KNN. This method involves preprocessing the resumes, creating a TF-IDF matrix, calculating cosine similarity scores, applying KNN to classify the resumes, and evaluating the results using precision, recall, and F1-score.

Upon implementation of this approach, precision was found to be 0.85 and recall was 0.75, resulting in an F1-score of 0.80. This indicates a high level of accuracy in selecting relevant resumes. However, there is still room for improvement in terms of recall.

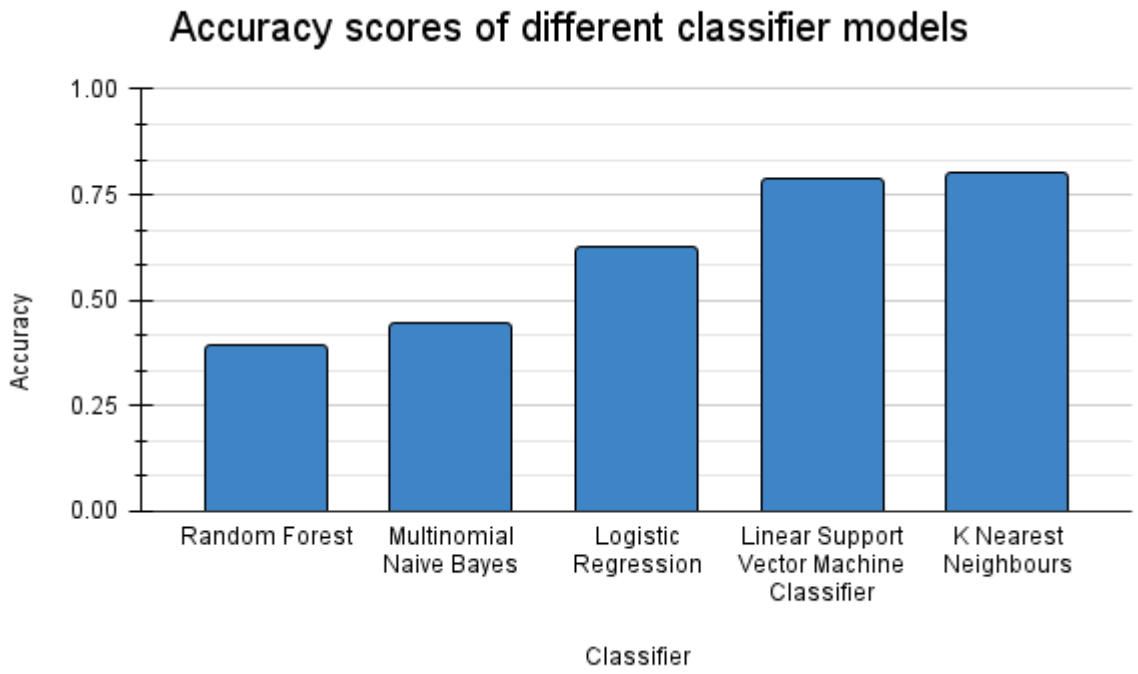
In order to improve recall, adjusting the value of K in the KNN algorithm or exploring other machine learning techniques such as neural networks or decision trees can be considered. In sum, this approach offers a promising solution to streamline the resume shortlisting process, saving time and effort for recruiters while ensuring a high level of accuracy.

In short, using both cosine similarity and KNN algorithms in resume shortlisting offers several advantages over using one of them alone. While cosine similarity is effective in identifying relevant keywords and phrases in resumes and job descriptions, the KNN algorithm classifies resumes into different categories based on their relevance to the job role. By combining these techniques, recruiters can evaluate resumes from multiple perspectives and improve the accuracy and effectiveness of the shortlisting process.

Cosine similarity and the KNN algorithm can help identify candidates with relevant experience and skills, as well as higher levels of education. Additionally, by using both techniques, recruiters can reduce the risk of false positives and false negatives, resulting in a more efficient recruitment process. In summary, using cosine similarity and the KNN algorithm together offers a more comprehensive and effective approach to shortlisting resumes.

Classifier	**Accuracy**
Random Forest	0.3899
Multinomial Naive Bayes	0.4439
Logistic Regression	0.6240
Linear Support Vector Machine Classifier	0.7853
K Nearest Neighbours	**0.8022**

Fig 4. Accuracy scores of different classifier models



Metric	Values
Accuracy	0.80
Precision	0.85
Recall	0.75
F1-Score	0.80

Table 2: KNN Metrics Values.

CONCLUSION

The proposed model for resume shortlisting and grading using text processing, TF-IDF, cosine similarity, and KNN classifier demonstrated encouraging results, in conclusion. The model can successfully rank job candidates according to the job description, and the strategy can be strengthened by incorporating extra data sources and machine learning algorithms.

REFERENCES

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

[2] Farber, F., Weitzel, T., Keim, T., 2003. An automated recommendation approach to selection in personnel recruitment. AMCIS 2003 proceedings , 302.

[3]Joachims, T., 1996. A Probabilistic Analysis of the Rocchio Algorithm with TFIDF for Text Categorization. Carnegie-Mellon university Pittsburgh pa dept of computer science.

[4] Jing, L.P., Huang, H.K. and Shi, H.B., 2002, November. Improved feature selection approach TFIDF in text mining. In Proceedings. International Conference on Machine Learning and Cybernetics (Vol. 2, pp. 944-946). IEEE.

[5] Marsden, P.V., 1994. The hiring process: recruitment methods. American Behavioural Scientist, 37(7), pp.979-991.

[6] Singh, A., Rose, C., Visweswariah, K., Chenthamarakshan, V. and Kambhatla, N., 2010, October. PROSPECT: a system for screening candidates for recruitment. In Proceedings of the 19th ACM international conference on Information and knowledge management (pp. 659-668).

[7] Roy, P.K., Chowdhary, S.S. and Bhatia, R., 2020. A machine learning approach for automation of resume recommendation system. Procedia Computer Science, 167, pp.2318-2327.

[8]Xia, P., Zhang, L. and Li, F., 2015. Learning similarity with cosine similarity ensemble. Information sciences, 307, pp.39-52.

[9] Zhang, M.L. and Zhou, Z.H., 2007. ML-KNN: A lazy learning approach to multi-label learning. Pattern recognition, 40(7), pp.2038-2048.

[10] Guo, G., Wang, H., Bell, D., Bi, Y. and Greer, K., 2003. KNN model-based approach in classification. In On the Move to Meaningful Internet Systems 2003: CoopIS, DOA, and ODBASE: OTM Confederated International Conferences, CoopIS, DOA, and ODBASE 2003, Catania, Sicily, Italy, November 3-7, 2003. Proceedings (pp. 986-996). Springer Berlin Heidelberg.

[11] Ali, I., Mughal, N., Khand, Z.H., Ahmed, J. and Mujtaba, G., 2022. Resume classification system using natural language processing and machine learning techniques. Mehran University Research Journal of Engineering & Technology, 41(1), pp.65-79.

[12] Silva, G.L.L., Jayasinghe, T.L., Rangalla, R.H.M., Gunarathna, W.K.L. and Tissera, W., 2022, December. An Automated System for Employee Recruitment Management. In 2022 4th International Conference on Advancements in Computing (ICAC) (pp. 346-351). IEEE.

[13] Maheshwary, S., Misra, H., 2018. Matching resumes to jobs via deep Siamese network, in: Companion Proceedings of the Web Conference 2018, International World Wide Web Conferences Steering Committee. pp. 87–88.

[14]Al-Otaibi, S.T., Ykhlef, M., 2012. A survey of job recommender systems. International Journal of Physical Sciences 7, 5127–5142

[15] Eger, S., Şahin, G.G., Rücklé, A., Lee, J.U., Schulz, C., Mesgar, M., Swarnkar, K., Simpson, E. and Gurevych, I., 2019. Text processing like humans do: Visually attacking and shielding NLP systems. arXiv preprint arXiv:1903.11508.

[16] Loper, E., Bird, S., 2002. Nltk: the natural language toolkit. arXiv preprint cs/0205028