Project Report

Goutham Baggu

ATS Resume Screening and Ranking System Using NLP in R

February 14,2026

—

Data & Analytics

INTRODUCTION

Recruitment processes today involve screening large volumes of resumes, making manual evaluation time-consuming and inefficient. This project develops an ATS (Applicant Tracking System) Resume Screening and Ranking System using NLP techniques in R, which automatically analyzes resumes by comparing them with a given job description. By applying text preprocessing, TF-IDF vectorization, and cosine similarity, the system calculates a match score and ranks candidates based on relevance. This approach demonstrates how natural language processing can be used to automate and enhance recruitment decision-making.

Project Objective

The primary objective of this project is to develop a core ATS (Applicant Tracking System) Resume Screening Engine that automates the evaluation of resumes against a given job description. The system aims to analyse unstructured textual data from both documents using Natural Language Processing (NLP) techniques and compute a similarity-based match score.

This project seeks to simulate how modern ATS platforms function by:

- Extracting textual content from resume documents
- Preprocessing and cleaning text for consistent analysis
- Converting textual data into numerical representations using TF-IDF
- Measuring document similarity through cosine similarity
- Generating a meaningful ATS match percentage
- Providing an interpretable match category (Strong, Moderate, Weak, or Poor)

The system is designed to offer a structured, objective, and automated method for assessing candidate suitability, thereby reducing manual effort and improving efficiency in the recruitment screening process.

The Process

Resume Text Extraction

The first step in the system involves extracting textual content from resume documents in PDF format. Using text parsing techniques in R, the unstructured resume data is converted into readable text format. This allows further analysis and preprocessing to be applied consistently across all candidate resumes.

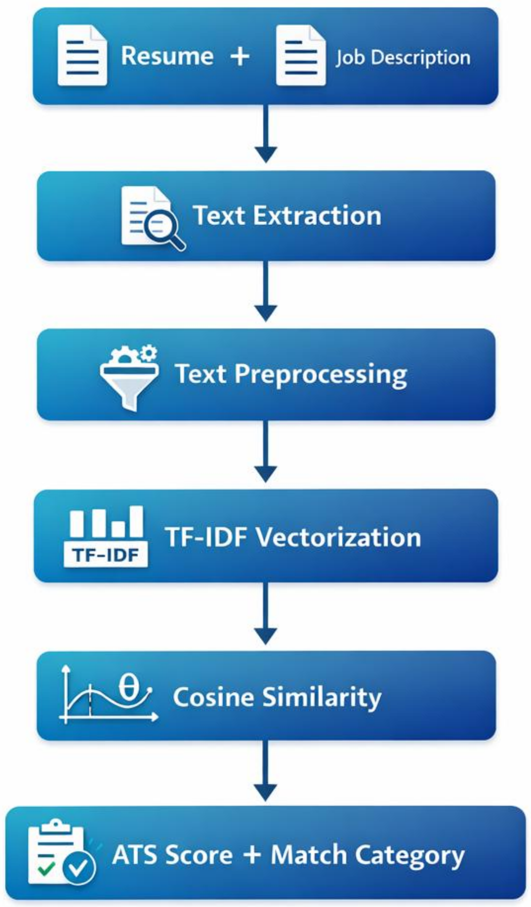
The extracted text serves as the foundational input for the ATS system. Each resume is stored along with its corresponding file name to maintain structured processing and evaluation.

Text Preprocessing and Feature Engineering

After extraction, both the job description and resumes undergo text preprocessing. This includes converting text to lowercase, removing special characters and stopwords, and normalizing the content for analysis. Cleaned text ensures consistent and meaningful feature representation.

Next, TF-IDF vectorization is applied to convert textual data into numerical format. Cosine similarity is then used to calculate the relevance score between each resume and the job description. Based on the computed similarity scores, candidates are ranked to identify the best match.

System Architecture



Implementation

Program Implementation in R

1. Input Handling Implementation

1.1 Implementation

```
# Required libraries
library(pdftools)
library(tm) # for stopwords() and removeWords()
library(text2vec) # used later for TF-IDF

#Take resume
cat("Select the path of the resume (only pdf)")
resume <- file.choose()
if(tools::file_ext(resume) == "pdf"){

  resume_txt <- pdf_text(resume)
  resume_txt <- paste(resume_txt, collapse = " ")

}else{

  stop("Given file is not pdf. Please Try again")

}

#Take JD
cat("Select the path of the JD")
jd <- file.choose()
if(tools::file_ext(jd) == "pdf"){
  jd_text <- pdf_text(jd)
  jd_text <- paste(jd_text,collapse = " ")

}else if(tools::file_ext(jd)=="txt"){

  jd_text <- readLines(jd,encoding = "UTF-8", warn = FALSE)
  jd_text <- paste(jd_text,collapse = " ")

}else{

  stop("Error can't go further")

}
```

1.2 Explanation:

In this stage, the system collects the Resume and Job Description (JD) files from the user using an interactive file selection mechanism. These two documents are the primary inputs required for calculating the ATS match score.

The file type of each document is validated using extension checking. This ensures that only supported formats are processed by the system.

- For PDF files, the pdf_text() function from the **pdftools** package is used to extract textual content.
- For text files (.txt), the readLines() function is used to read the content line by line.

After extraction, both PDF and TXT files return text as vectors. However, TF-IDF requires each document to be represented as a single consolidated string. Therefore, the paste() function with the collapse parameter is used to combine multiple text elements into one unified string suitable for further processing.

Error handling is implemented using the stop() function to immediately terminate execution if an unsupported file format is selected.

2. Text Preprocessing Module

2.1 Implementation

```
clean_text <- function(text){
  text <- tolower(text)
  text <- gsub("[^a-z]"," ",text)
  text <- removeWords(text, stopwords("english"))
  text <- stripWhitespace(text)

  return(text)
}
```

```
resume_txt <- clean_text(resume_txt)
jd_text <- clean_text(jd_text)
```

2.2 Explanation:

A reusable function named clean_text() was developed to standardize and preprocess textual data before feature extraction.

The preprocessing pipeline performs the following steps:

- Converts all text to lowercase to ensure uniform comparison
- Removes special characters and numbers using regular expressions
- Eliminates common stopwords (such as *the*, *is*, *and*) to reduce noise
- Normalizes spacing by removing extra white spaces

This preprocessing stage ensures that both the Resume and Job Description are transformed into clean and standardized textual representations, which improves the accuracy and effectiveness of the TF-IDF vectorization process.

3. TF-IDF Vectorization and Feature Engineering

3.1 Implementation

```
# Combine documents (Important: same vocabulary space)
documents <- c(jd_text, resume_txt)

# Tokenization
it <- itoken(documents, progressbar = FALSE)

# Create vocabulary
vocab <- create_vocabulary(it)

# Vectorizer
vectorizer <- vocab_vectorizer(vocab)

# Create Document-Term Matrix
dtm <- create_dtm(it, vectorizer)

# Apply TF-IDF transformation
tfidf_transformer <- TfIdf$new()
tfidf_matrix <- fit_transform(dtm, tfidf_transformer)
```

3.2 Explanation

Why Combine Documents?

TF-IDF works on a **collection of documents**, not on a single document.

By combining:

- Job Description
- Resume

we ensure both are represented within the **same vocabulary space**.

This guarantees that both vectors have identical feature dimensions.

Tokenization (itoken())

Tokenization splits text into individual words (tokens).

Instead of manually splitting using strsplit(), we use itoken() from **text2vec**, which efficiently prepares documents for vectorization.

Vocabulary Creation (create_vocabulary())

This function scans both documents and creates a list of unique words.

Example vocabulary:

- python
- sql
- docker
- algorithm

This becomes the foundation for the document-term representation.

Document-Term Matrix (create_dtm())

The Document-Term Matrix (DTM):

- Rows → Documents
- Columns → Unique words
- Values → Frequency of words

4. Cosine Similarity Computation and ATS Score Generation

4.1 Implementation

```
tfidf <- TfIdf$new()
dtm_tfidf <- fit_transform(dtm, tfidf)

# Cosine Similarity
cosine_sim <- sim2(
  dtm_tfidf[1, , drop = FALSE],
  dtm_tfidf[2, , drop = FALSE],
  method = "cosine"
)

score <- round(as.numeric(cosine_sim) * 100, 2)
```

4.2 Explanation

TF-IDF Transformation

Here we convert the raw Document-Term Matrix into a **TF-IDF weighted matrix**.
This step:

- Reduces weight of common words
- Increases importance of rare but meaningful terms
- Helps emphasize technical skills

This makes similarity measurement more meaningful.

Cosine Similarity

Cosine similarity measures the angle between two vectors.

Formula:

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}},$$

Where:

- A → JD vector
- B → Resume vector

If:

- Result ≈ 1 → Very similar
- Result ≈ 0 → Completely different

5. ATS Score Interpretation and Classification

5.1 Implementation

```
score <- round(as.numeric(cosine_sim) * 100, 2)

if(score >= 80){
  category <- "Good Match"
}else if(score >= 60){
  category <- "Moderate Match"
}else if(score >= 40){
  category <- "Weak Match"
}else{
  category <- "Poor Match"
}
```

5.2 Score Conversion

The cosine similarity value ranges between **0 and 1**.

To make it more interpretable, the value is converted into a percentage by multiplying by 100.

Example:
- 0.82 → 82%
- 0.45 → 45%

This percentage represents how closely the resume matches the job description.

5.2 Match Category Classification

To provide meaningful interpretation, the ATS score is categorized into levels:

Score Range	Category
80–100%	Strong Match
60–79%	Moderate Match
40–59%	Weak Match
Below 40%	Poor Match

This classification helps recruiters or candidates understand alignment strength beyond just a numeric score.

6. Resume Improvement Suggestion Mechanism

6.1 Implementation

```
terms <- colnames(dtm_tfidf)

# JD vector (row 1)
jd_vector <- as.numeric(dtm_tfidf[1, ])

# Resume vector (row 2)
resume_vector <- as.numeric(dtm_tfidf[2, ])

# Words present in JD but not in resume
```

```
missing_terms <- terms[jd_vector > 0 & resume_vector == 0]

# Show top 10 missing keywords
top_missing <- head(missing_terms, 10)

cat("\nSuggested Keywords to Improve Your Resume:\n")
print(top_missing)
cat("\n=============================\n")
cat("      ATS EVALUATION RESULT\n")
cat("=============================\n")
cat("Match Category:", category, "\n")
cat("Match Score:", score, "%\n")
cat("=============================\n")
cat("Suggested Key words :",top_missing)
```

6.2 Explanation

To enhance the practical usefulness of the system, a keyword-based suggestion module is implemented.

The system:

1. Extracts vocabulary terms from the TF-IDF matrix
2. Identifies terms present in the Job Description
3. Checks which of those terms are absent in the Resume
4. Suggests the missing keywords

Example:
If JD includes:

- Docker
- Microservices
- CI/CD

And the resume does not contain them, the system suggests adding those skills.
This makes the ATS system actionable rather than purely evaluative.

7. Final Output Generation

The system generates structured output in the format:

```
==============================
   ATS RESULT
==============================
Match Category: Moderate Match
Match Score: 72.35 %
==============================

Suggested Keywords:
- Docker
- REST API
- Microservices
```

This provides:

- Clear evaluation
- Professional formatting
- Actionable improvement feedback