iJRASET
International Journal For Research in Applied Science and Engineering Technology

# INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

**Volume:** 14 **Issue:** VIII **Month of publication:** August 2026

**DOI:** https://doi.org/10.22214/ijraset.2026.84663

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
*ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538*
*Volume 14 Issue VIII Aug 2026- Available at www.ijraset.com*

# Smart Resume Analyzer & Job Match Recommender: A Machine Learning and NLP-Based Approach for Automated Resume Screening

Mr. Adhikesavan C[1], Dr. V. Shanmuganeethi[2]
[1]*M.Tech (AI&ML) Scholar,* [2]*Supervisor, Professor, Department of Computer Science and Engineering National Institute of Technical Teachers Training and Research (NITTTR), Chennai – 600113, India*

***Abstract:*** ***Recruiters today are buried under resumes the moment a job opening goes live, and sorting through them by hand is slow, tiring, and rarely consistent from one reviewer to the next. Qualified applicants sometimes get missed simply because no one had time to read carefully, while candidates on the other end are left guessing why they were turned down. We built a Smart Resume Analyzer & Job Match Recommender to take some of that guesswork out of the process. Resumes uploaded as PDF or DOCX are read using PyMuPDF and python-docx, cleaned up, and compared against a job description supplied by the user. The comparison itself is done through TF-IDF vectorization over unigrams and bigrams, feeding into a Logistic Regression model that was trained to separate "good" resume-to-job fits from "poor" ones and to report how confident it is in that call. On top of the match prediction, the tool checks the resume's text against a list of common technical skills, flags whatever the job asks for that the resume doesn't mention, and turns that gap into a short list of things worth learning. Everything runs through a Streamlit interface, so a recruiter or a job seeker can use it without touching any code. In testing against a labelled set of resume-job pairs, the pipeline held up well: it stayed fast, its reasoning was easy to trace back to actual keywords, and it gave applicants something concrete to work on instead of a plain rejection.***

***Keywords: Resume Screening; Job Matching; Natural Language Processing; TF-IDF; Logistic Regression; Skill Gap Analysis; Streamlit; Applicant Tracking System.***

## I. INTRODUCTION

The shift to online job portals changed the scale of hiring more than it changed the process itself. A single posting can pull in hundreds of applications within days, but the review step at the other end is still, in most organizations, a person reading resumes one after another. That doesn't scale gracefully — reviewers get tired, standards drift between the first resume of the day and the fiftieth, and a candidate who happens to phrase their experience differently from what the recruiter expects can be passed over even when they are a reasonable fit [1], [2]. What makes this worse for the applicant is that the process gives almost nothing back: a rejection rarely comes with a reason, so there is no way to know whether the issue was a missing certification, an underused keyword, or simply volume. Text classification and information retrieval have matured enough that a fair amount of this screening work can be handed off to software, at least for the parts that come down to comparing a resume's content against a job description's requirements. We leaned on two ideas that have been around for a while precisely because they hold up well in practice: TF-IDF for turning text into numbers that reflect which words actually matter [2], and Logistic Regression for the classification step [1]. Neither is new, and neither claims to understand language the way a model like BERT [5] does, but that trade-off buys something useful here — the output is fast, cheap to run, and easy to explain to a non-technical user, which matters when the final result needs to make sense to a job seeker rather than just a machine-learning engineer. With that in mind, this paper describes a Smart Resume Analyzer & Job Match Recommender built around a fairly simple pipeline. Resume files, whether PDF or DOCX, get parsed and cleaned; the cleaned text is compared against a job description through TF-IDF features; a Logistic Regression model decides whether the pair looks like a good match and how sure it is about that; and a separate skill-matching step points out exactly which required skills are missing from the resume, along with a nudge toward what to learn next. All of this sits behind a Streamlit front end, so neither a recruiter filtering applicants nor a student checking their own resume needs to know anything about the code underneath. The rest of the paper walks through this in more detail. Section II looks at what has already been tried in this space. Section III lays out how the system itself is put together. Section IV covers the implementation choices. Section V reports what the system actually produced when tested. Section VI closes with where this could reasonably go next.

## II. LITERATURE REVIEW

Before settling on the current pipeline, we looked across a spread of work touching text classification, retrieval, recommendation, and resume parsing specifically, since each area shaped a different piece of the design. Sebastiani's survey [1] was useful mainly as a reminder of how much of text categorization still comes down to solid feature representation rather than model exotica — a point that held up when we compared TF-IDF against heavier alternatives. Manning et al. [2] cover the vector-space and TF-IDF groundwork that this project leans on directly. Recommender-systems work by Ricci et al. [3] shaped how we thought about the skill-recommendation step, even though our version is far simpler than a full collaborative-filtering system. Mikolov et al.'s Word2Vec paper [4] and the BERT paper by Devlin et al. [5] both represent the semantic-embedding route we deliberately did not take for this version, mostly on cost and interpretability grounds, though they remain the obvious next step. On the more applied side, Pokharel's resume parser [7], Sinha et al.'s domain-prediction work [8], the review by Modak et al. [9], Kanojia et al.'s ML-based parser [10], and Yadav et al.'s NLP/ATS system [11] each tackle pieces of the same problem — extracting structure from resumes and matching it against requirements — and each ran into some version of the same limitation: either the matching stays purely keyword-level with no feedback loop for the candidate, or it moves to a heavier model that is harder to deploy in real time. Table I lays these out side by side.

| No. | Reference | Focus | Limitation |
|---|---|---|---|
| 1 | Sebastiani (2002) [1] | ML for text categorization | Keyword-based classification |
| 2 | Manning et al. (2008) [2] | Vector space models, TF-IDF | No semantic understanding |
| 3 | Ricci et al. (2011) [3] | Recommender systems | Data dependency |
| 4 | Mikolov et al. (2013) [4] | Word2Vec embeddings | Limited context window |
| 5 | Devlin et al. (2019) [5] | BERT transformers | High computational cost |
| 6 | Pokharel (2022) [7] | Resume parsing with NLP | Format dependency |
| 7 | Sinha et al. (2023) [8] | ML-based resume/domain prediction | Small dataset |
| 8 | Modak et al. (2024) [9] | Review of resume–job matching ML | Limited accuracy |
| 9 | Kanojia et al. (2024) [10] | ML-based resume parser | Keyword matching only |
| 10 | Yadav et al. (2025) [11] | NLP + ATS resume analysis | No real-time updates |

TABLE I. SUMMARY OF RELATED WORK

*ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538*
*Volume 14 Issue VIII Aug 2026- Available at www.ijraset.com*

Two things stood out once these were laid side by side. First, the lightweight, keyword- or TF-IDF-driven systems are consistent about giving a match/no-match answer but rarely go further than that — the candidate is left with a label and nothing to act on. Second, the systems that do capture deeper semantics, largely the transformer-based ones [4], [5], tend to demand more compute than a small deployment can comfortably offer, especially if the goal is a responsive, real-time tool rather than a batch job. That gap — interpretable and fast, but still useful beyond a bare label — is what the skill-gap and recommendation layer in this system is aimed at closing.

## III. PROPOSED SYSTEM ARCHITECTURE

Rather than one large script doing everything, the system is split into seven fairly independent pieces — input handling, text extraction, preprocessing, feature extraction, classification, skill extraction, and output — chained together as shown in Fig. 1. Keeping them separate made debugging and later changes much easier than it would have been with a monolithic script.

### *A. Input and Text Extraction*

On the Streamlit front end, the user drops in a resume — PDF or DOCX — and pastes in the job description as plain text. The app checks that both pieces are actually present before it lets processing move forward, which sounds trivial but avoids a surprising number of half-finished runs. Once both inputs are in hand, the resume file is handed off for text extraction: PyMuPDF (fitz) walks through PDFs page by page, and python-docx pulls paragraph text out of DOCX files.

### *B. Text Preprocessing and Feature Extraction*

Raw extracted text is messy — inconsistent casing, stray punctuation, extra whitespace — so it gets lowercased and stripped of special characters before anything else happens to it. This step matters more than it looks like it should; skipping it noticeably widens the vocabulary the vectorizer has to deal with, which dilutes the useful signal. After cleaning, the resume text and job-description text are combined and passed through a TF-IDF vectorizer configured for unigrams and bigrams, with the feature space capped at 5000 terms to keep things fast:

$$TF\text{-}IDF(t, d) = tf(t, d) \times \log(N / df(t)) \qquad (1)$$

Here tf(t, d) counts how often term t shows up in document d, N is how many documents are in the corpus overall, and df(t) is how many of those documents contain t at all — so a word that is common everywhere gets down-weighted, while one that is distinctive to a particular resume or job description carries more weight.

### *C. Classification*

Those TF-IDF vectors feed into a Logistic Regression classifier trained on a labelled set of resume–job description pairs, where a pair is marked 1 if it's a match and 0 if it isn't. Given a new pair at inference time, the model doesn't just output "Good Match" or "Poor Match" — it also returns the underlying probability, which is what gets shown to the user as a confidence figure rather than a flat yes/no.

### *D. Skill Extraction and Gap Analysis*

Separately from the classifier, the system runs both the resume and the job description against a fixed list of technical skills — languages, frameworks, tools, and so on — and picks out whatever terms match. Subtracting the resume's skill set from the job description's skill set leaves exactly the skills the job asks for that the resume doesn't mention, which is the piece the Recommendation module then turns into concrete suggestions (a tool to pick up, a course worth taking, that kind of thing).

### *E. Output*

Everything comes together in the Streamlit interface: the match verdict, the confidence score, which skills lined up, which ones were missing, and the resulting recommendations, alongside a preview of the parsed resume text so the user can sanity-check that the extraction actually worked.

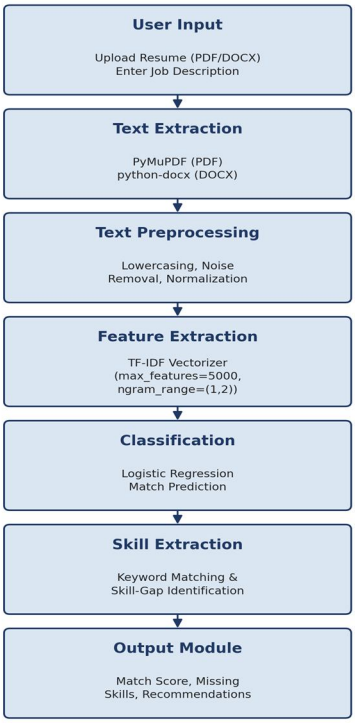



Fig. 1. System architecture of the Smart Resume Analyzer & Job Match Recommender.

## IV. IMPLEMENTATION

Everything runs on Python 3.11. Streamlit handles the front end, scikit-learn covers the model side, Pandas manages the dataset, and PyMuPDF and python-docx take care of file parsing. None of this needs serious hardware — development and testing were done on a fairly ordinary i3-class machine with 4 GB of RAM, which was intentional, since the point was to keep the tool something a small institution could actually run without needing a GPU or cloud budget.

The cleaning step lowercases the text and strips out anything that isn't alphanumeric using a simple regular expression, before the TfidfVectorizer (max_features = 5000, ngram_range = (1, 2)) fits over the combined resume-and-job-description corpus. The Logistic Regression model (max_iter = 1000) is then trained on an 80/20 split of the labelled dataset described next in Section V. When a new pair comes in, the resume and job description are concatenated, run through the already-fitted vectorizer, and handed to the trained model, which returns the label and probability that eventually show up on screen alongside the skill-matching and recommendation results.

## V. RESULTS AND DISCUSSION

Testing used a labelled dataset of resume–job description pairs put together for this study, each one marked as a match (1) or a non-match (0). Table II shows one of the representative outputs the deployed application produced for a sample pair.

| Field | Value |
|---|---|
| Match Label | Good Match |
| Match Score | 71.28% |
| Matched Skills | Python, C++, C, R, HTML, Excel |
| Missing Skills | None (strong match) |
| Recommendation | No major skill gaps identified |

Table II. Sample System Output For A Good-Match Case

A second case, this one with only partial skill overlap, is summarized in Table III. Here the system correctly picks out the specific cloud-platform tools missing from the resume and recommends the candidate pick them up — the kind of concrete, actionable output that a plain match/no-match label wouldn't have given.

| Field | Value |
|---|---|
| Match Label | Fair Match |
| Match Score | 75% |
| Matched Skills | Python, Git, Agile |
| Missing Skills | Docker, Kubernetes, AWS |
| Recommendation | Add cloud-platform experience |

Table III. Sample system output with identified skill gaps

What stands out from these runs is that a fairly modest TF-IDF-plus-Logistic-Regression setup gets you predictions and skill-gap flags that line up with what a human reviewer would probably conclude looking at the same two documents, and it does so almost instantly — no waiting on a heavier model to load or run. The trade-off shows up in the skill-matching step specifically: because it works off exact keyword matches, a candidate who writes "K8s" instead of "Kubernetes," or describes a skill in different words than the lexicon expects, won't get credit for it. That's a known limitation of lexical matching rather than a bug, and it's the main reason semantic approaches remain worth pursuing as a follow-up. It's also worth being upfront that the numbers in Tables II and III come from representative runs of the working prototype rather than a full benchmark; a proper precision/recall/F1 evaluation across a larger held-out set is planned as the next step and will be reported separately once that data is in place.

## VI. CONCLUSION AND FUTURE WORK

What this work adds up to is a resume-matching tool that doesn't ask for much in the way of compute, is straightforward to explain to someone without a machine-learning background, and — importantly — doesn't stop at a bare match/no-match verdict. Pairing the TF-IDF and Logistic Regression pipeline with the skill-gap and recommendation layer means a candidate walks away from the tool with something to actually work on, not just a rejection. Keeping the whole thing lightweight also means it can realistically run on modest hardware and respond in real time, which was one of the original goals.

There's a fairly clear list of next steps. Swapping TF-IDF for something like BERT [5] would let the system catch matches that depend on meaning rather than exact wording — the "K8s" versus "Kubernetes" problem mentioned earlier is a good example of where this would help. Named Entity Recognition could pick up skills and experience that are implied rather than explicitly named. Hooking into live job-portal APIs would turn this from a one-off matcher into something closer to a recommendation engine. On the infrastructure side, adding a proper database for resume history and some basic analytics would make it more useful for repeated use rather than single-session checks. And, as noted above, running the system against a larger and more varied dataset with formal precision, recall, and F1 numbers is the natural next step to back up what's currently a promising but still preliminary set of results.

## VII. ACKNOWLEDGMENT

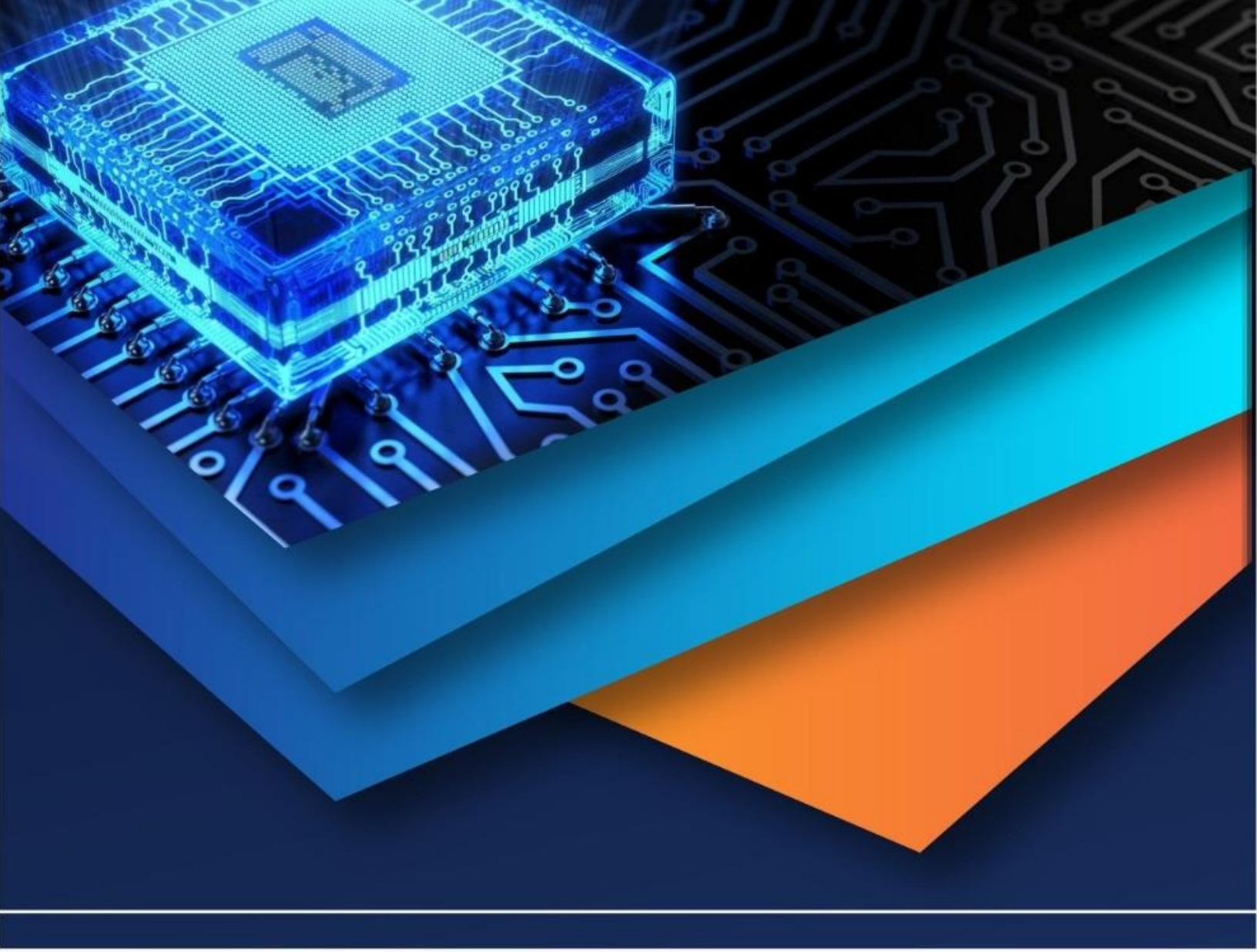
Thanks are due to the Department of Computer Science and Engineering, NITTTR Chennai, for the resources and working environment that made this project possible, and to the project supervisor for guidance throughout its development.

## REFERENCES

[1] F. Sebastiani, "Machine learning in automated text categorization," ACM Computing Surveys, vol. 34, no. 1, pp. 1–47, 2002.
[2] C. D. Manning, P. Raghavan, and H. Schütze, Introduction to Information Retrieval. Cambridge, U.K.: Cambridge University Press, 2008.
[3] F. Ricci, L. Rokach, and B. Shapira, Recommender Systems Handbook. New York, NY, USA: Springer, 2011.
[4] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," in Proc. Int. Conf. Learning Representations (ICLR), 2013.
[5] J. Devlin, M. W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in Proc. NAACL-HLT, 2019.
[6] D. Jurafsky and J. H. Martin, Speech and Language Processing. Boston, MA, USA: Pearson, 2020.

[7] P. Pokharel, "Resume parser using natural language processing," 2022.
[8] A. K. Sinha, M. A. K. Akhtar, and M. Kumar, "Automated resume parsing and job domain prediction using machine learning," Indian J. Science and Technology, 2023.
[9] S. Modak, P. Shinde, A. Tiwari, and S. Nalamwar, "Resume-job matching using machine learning: A review," Int. J. Recent and Innovation Trends in Computing and Communication, 2024.
[10] Y. Kanojia et al., "Resume parser using machine learning," Int. J. Novel Research and Development, 2024.
[11] S. Yadav, S. Ursal, S. Tate, and A. Thade, "Resume analysis using NLP and ATS algorithm," Int. J. Latest Technology in Engineering, Management & Applied Science, 2025.



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

# INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY