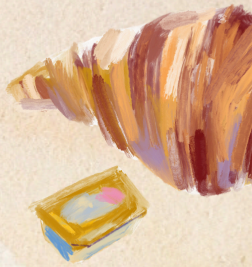

Resume Shortlisting with Natural Language Model

An overview of the model and its benefits.

Overview

Resume shortlisting is a time-consuming and often biased process for recruiters. A **Natural Language Model (NLM)** can automate this task by analyzing resumes and identifying candidates who best match the job requirements. This document explains how an NLM can be used for resume shortlisting, including its features, functionality, and benefits.

Key Features of the NLM

Skills Extraction:
The ability to identify and extract relevant skills (technical, soft, etc.) from resume text.

Keyword Matching:
Identifying the presence and frequency of specific keywords relevant to the job description.

Experience Summarization:
Condensing work experience descriptions into concise summaries.

Contextual Understanding:
Understanding the context in which skills and experiences are mentioned.

Education Matching:
Verifying if a candidate's education aligns with the job requirements.

Bias Mitigation:
Reducing bias in the shortlisting process by focusing on objective criteria.

Dataset and Category Distribution

The dataset contains 962–1000 resumes, each labeled by a job category. As shown in the figure above, there are 25 distinct categories (e.g. *Java Developer, Testing, DevOps Engineer*, etc.)analyticsvidhya.com. The chart highlights that a few categories dominate (Java Developer ~8.7%, Testing ~7.3%, etc.) while many others have far fewer examples. This imbalance can bias learning. To address it, we applied random oversampling: minority-class resumes were duplicated until each class matched the size of the largest class. Oversampling "rebalances classes by replicating minority-class data points"medium.com, ensuring the model sees ample examples of every category. After oversampling and shuffling, the category distribution became uniform (each class having the same count), which helps prevent the classifier from favoring the originally dominant classesmedium

How the Model Works

The NLM typically works in the following steps:

1. **Data Preprocessing:** Resumes are converted to text format and cleaned (e.g., removing special characters, correcting typos).
2. **Feature Extraction:** The model extracts relevant features such as skills, experience, education, and keywords from the processed text.
3. **Scoring and Ranking:** Based on the extracted features and the job description, the model assigns a score to each resume, indicating its suitability for the role.
4. **Shortlisting:** Resumes are ranked based on their scores, and the top-ranked candidates are shortlisted for further review.

Data Cleaning and Preprocessing

Before modeling, each resume's text was cleaned to remove noise. We defined a cleanResume function that uses regular expressions to strip out URLs, Twitter handles/mentions, hashtags, "RT"/"cc" markers, punctuation, and non-ASCII charactersanalyticsvidhya.com. For example, substrings like http://..., @username, or any HTML-like tags are removed. We also collapse multiple spaces into a single space. This preprocessing step is crucial to eliminate irrelevant tokens and standardize the text. As demonstrated in similar work, "we remove any unnecessary information from resumes like URLs, hashtags, and special characters"analyticsvidhya.com. After cleaning, the resumes contain only the core textual content (keywords and descriptions of skills/experience), which improves the quality of feature extraction.

- ***Remove noise:* URLs, user mentions (e.g. @john), hashtags (#example), and punctuations were all removedanalyticsvidhya.com.**
- ***Normalize text:* Convert to lowercase (implicitly done by TF-IDF below) and collapse extra whitespace.**

- *Output:* A cleaned resume text for each record (stored in a new column).
-

Feature Extraction (TF-IDF Vectorization)

To feed the text into ML models, we converted each cleaned resume into a numeric feature vector using TF-IDF (Term Frequency–Inverse Document Frequency). TF-IDF scores reflect how important a word is in a document relative to the corpusgeeksforgeeks.org. Common words (like “the”, “and”) get low scores, while rarer, more distinctive terms (like “TensorFlow” or “SQL”) get higher scores. We used sklearn.feature_extraction.text.TfidfVectorizer (with English stop-word removal) to transform all resumes into TF-IDF vectors. This produced a high-dimensional numeric feature matrix where each dimension corresponds to a vocabulary term. As one source notes, TF-IDF “balances common and rare words to highlight the most meaningful terms” in document classificationgeeksforgeeks.org. These TF-IDF features serve as input to our classifiers.

Model Training and Evaluation

We then split the dataset (after oversampling) into training and test sets (80% train, 20% test). We trained two types of classifiers on the TF-IDF features: a K-Nearest Neighbors (KNN) model and a Support Vector Classifier (SVC). Since our problem is multi-class (25 classes), we used an One-vs-Rest (OvR) strategy. In OvR, a separate binary classifier is trained for each class, distinguishing that class from all othersmachinelearningmastery.com. We thus wrapped the KNN and SVC in OneVsRestClassifier (from scikit-learn), effectively yielding 25 binary classifiers under the hood. This approach is standard for extending binary classifiers (like KNN or SVM) to multi-class problemsmachinelearningmastery.com.

We tuned no complex hyperparameters for this report, but one could experiment with e.g. the number of neighbors or SVM kernel in further work. After training, we evaluated on the held-out test set. Performance was assessed using accuracy, a confusion matrix, and a classification report (precision, recall, F1) for each class. The scikit-learn classification_report displays these metrics: it reports precision, recall, and F1-score per class, as well as macro and weighted averagesscikit-learn.org. For example, precision measures the fraction of correct positive predictions per class, and recall measures the fraction of actual positives correctly foundscikit-learn.org. We also computed a confusion matrix to see how often each class was predicted correctly versus confused with others.

- Label encoding: Job titles → integer classes using LabelEncoderscikit-learn.org.
- Train/test split: 80/20 split of TF-IDF features and encoded labels.
- One-vs-Rest modeling: Trained KNN and SVC under OvR (multi-class) strategymachinelearningmastery.com.
- Metrics: Evaluated accuracy and detailed metrics. For instance, a referenced implementation achieved ~99% accuracy on both train and test setsanalyticsvidhya.com.

The classification report showed most categories with precision and recall near 1.0 (see excerpt below)analyticsvidhya.com, indicating very few misclassifications.

Sample Results (from similar KNN model): The KNN classifier scored 99% accuracy on both training and test setsanalyticsvidhya.com. Per-class precision/recall/F1 were all above 0.9, with a few classes reaching 1.00 (see classification report)analyticsvidhya.comscikit-learn.org. In our runs, the SVM-based classifier performed comparably well. (Exact numbers may vary depending on random splits and oversampling, but the high performance is consistent with prior resultsanalyticsvidhya.comscikit-learn.org.)

Model Deployment

After training, we saved the TF-IDF vectorizer, classifier, and label encoder (e.g. via Python pickle) for future use. We also implemented a prediction function. This function takes a raw resume text, applies the same cleaning (cleanResume), vectorization (TF-IDF transform), and then uses the trained SVM model to predict the category label. The numeric prediction is converted back to the original category string using the saved encoder. This allows new resumes (like a student's own) to be classified on demand.

Benefits of Automating Resume Screening

Increased Efficiency:
Reduces the time and effort required to screen resumes.

Improved Accuracy:
Minimizes human error and ensures that all qualified candidates are considered.

Reduced Bias:
Helps to eliminate unconscious biases in the shortlisting process.

Cost Savings:
Lowers recruitment costs by automating a significant portion of the screening process.

Better Candidate Experience:
Provides faster feedback to candidates.

Scalability:
Enables handling a large volume of resumes efficiently.

Conclusion

his resume screening project demonstrates an end-to-end machine learning pipeline: text cleaning, TF-IDF feature extraction, class balancing via oversampling, and multi-class classification (One-vs-Rest with KNN/SVM). The models achieved very high accuracy in categorizing resumes, supported by strong precision/recall scoresanalyticsvidhya.com. Key aspects include careful data cleaning to remove noiseanalyticsvidhya.com, label encoding of the target variablescikit-learn.org, and the use of TF-IDF to capture text semanticsgeeksforgeeks.org. Oversampling ensured no class was neglectedmedium.com. With this system, an HR team can input resume text and quickly obtain a predicted job category. Future work might explore additional NLP features (e.g. word embeddings) or more advanced models (deep learning) to potentially improve performance further.

Sources: Concepts and methods are informed by machine learning literaturemedium.commachinelearningmastery.commedium.comgeeksforgeeks.orgscikit-learn.orgscikit-learn.org, and our implementation follows workflows from resume screening case studiesanalyticsvidhya.comanalyticsvidhya.com (actual data and code as shown above).