Resume Classification

Soundarya Lahari Valipe
Department of Computer Science
University of Central Florida
Orlando, Florida, USA
soundaryalahari@Knights.ucf.edu

Venkata Ramana Rohith Neralla
Department of Computer Science
University of Central Florida Orlando,
Florida, USA
rohith.neralla@knights.ucf.edu

ABSTRACT

Resume classification is the automated process of categorizing or labelling resumes or CVs (Curriculum Vitae) into pre-defined categories or classes based on their content. It is a frequently used task in recruitment processes, especially when there is a large volume of resumes that needs to be efficiently reviewed and sorted. Automation of the process of classification of resumes can be a time-saving and efficient approach in the recruitment process. It enables recruiters to focus on in-depth evaluations of relevant resumes, while also aiding in the identification of qualified candidates for specific job roles or industries. This project intends to develop methods to classify and assign applicants' resumes to relevant class of domains of different job postings (e.g., Data Science, Web Designing, etcetera). The categorization of resumes is based on various techniques in Natural Language Processing (NLP) and Machine Learning (ML) such as K-Nearest Neighbors (KNN), Convolutional Neural Networks (CNN) and DistilBERT models. We train the data we gathered from Kaggle using these models and evaluate them using various metrics. The model with the best accuracy will be used to assign the input resume to a particular class of job postings. This utilization of Machine Learning and Natural Language Processing techniques for resume classification has the potential to greatly minimize the time and effort needed for manual resume screening, leading to enhanced overall efficiency in the recruitment process.

Keywords – Resume Classification, NLP, CNN, DistilBERT

1 Introduction

In today's competitive job market, an enormous number of resumes are received for each job opening in an organization. The process of manually reviewing a large volume of resumes can be time-consuming and labor-intensive, particularly for organizations that receive a high number of resumes for each job opening. Human recruiters may need to invest significant time and effort in thoroughly reviewing and assessing each resume, leading to potential delays in the overall recruitment process. When there is high influx of resumes, manual resume screening can pose challenges in terms of scalability leading to increased workload and reduced productivity for recruiters.

Manual resume screening is susceptible to human biases, such as unconscious bias, personal preferences, and subjective judgment, resulting in inconsistent and biased decision-making. The subjective nature of manual resume screening can result in inconsistency and variability in the evaluation process, as different recruiters may have different criteria and interpretations of resume attributes, which may not always align with the organization's predefined criteria and requirements. Lack of consistency and standardization along with possible human errors in manual resume screening can lead to variations in the assessment process, making it challenging to compare and prioritize resumes consistently and resulting in inconsistent hiring decisions such as overlooking qualified candidates.

Resume classification is an automated process that involves categorizing resumes into predefined classes or categories using machine learning and natural language processing (NLP) techniques. It aims at streamlining the resume screening process. Resume Classification serves as a valuable solution for organizations to process resumes quickly and efficiently during the initial screening stage of the recruitment process. By utilizing machine learning and natural language processing techniques, resume classification models can categorize resumes into predefined categories, based on specific criteria such as education, work experience, skills, and other attributes.

Resume classification enables organizations to efficiently handle large volumes of resumes. Resume classification, being an automated process, can quickly and accurately categorize resumes into predefined classes or categories, allowing recruiters to handle a large volume of resumes in a very efficient manner. Resume classification mitigates the impact of human biases on decision-making by relying on predefined rules and patterns learned from labeled data during the model training process. This minimizes the influence of human biases, promoting fairness and objectivity in the screening process. Resume Classification minimizes the risk of human errors by thoroughly and consistently evaluating all resumes based on predefined criteria, reducing the likelihood of missing qualified candidates due to human error.

Resume classification is a supervised machine learning task. It involves initial pre-processing and feature extraction from resume data, followed by training a classifier using machine learning and natural language processing (NLP) techniques. The trained classifier can then automatically categorize new resumes into predefined categories based on learned patterns and rules.

2 Problem Statement

The problem this project aims to solve is the challenge of sorting through a large volume of resumes to find suitable candidates for job postings. The objective is to develop a system that accepts resumes as input and uses NLP and ML techniques to categorize them into relevant job posting domains based on skills, such as Data Science or Web Designing.

The project consists of two main blocks that are sequentially connected: training and classification. In the training block, the system will be trained with a set of resumes using pre-processing steps such as removing stop words and lemmatization. We then train our models – K-Nearest Neighbor (KNN), Convolutional Neural Network (CNN) and DistilBERT. We evaluate these models using the appropriate metrics to select the best one. In the classification block, the best model will be used to match each applicant's resume to a relevant job posting domain. This system will save time and resources in the hiring process by automatically categorizing resumes based on skills, allowing recruiters to focus on the most suitable candidates for each job posting.

3 Related Work

In [1], the paper is titled 'Resume Screening Using Machine Learning and NLP: A Proposed System' and the dataset used is from Kaggle and they proposed a system to predict what kind of job is best suited to the resume under consideration using K-Nearest Neighbours model. The resume is passed to the resume parser which is a pipeline of NLP techniques that will extract useful information from the resume, and then the system will visit the person's LinkedIn and GitHub profile to scrape useful information from the website provides it to the Machine learning Model for the prediction. An accuracy of 79.8% was achieved. There is no mention of the train and test split percentage of the data in this paper.

[2] is a paper titled 'Resume Screening using NLP and LSTM' by S. Bharadwaj. et al and in their approach, Resume Screening is performed using NLP and LSTM. The system accepts candidate's resume as an input and provides the best suited job role to this resume as an output based on the data obtained Kaggle. The paper describes in detail about the NLP pipeline and the model used for training and predictions is LSTM. There is no mention of the accuracy percentage and there is no mention of the train and test split percentage of the data in this paper.

In [3], the paper is titled 'Resume Classification using various Machine Learning Algorithms' and it is by Riya Pal. et al. For this project, data was gathered from various sites which includes collection of datasets from various websites like kaggle.com, glassdoor.com and indeed.com. 70% of the data is being used for training data and the remaining 30% will be used for test data. In [3], classification of resumes was performed using Machine Learning Algorithms such as Naive Bayes, Random Forest, and SVM, which will aided in the extraction of skills and classification of test resumes under appropriate job profile classes. Accuracies of 70, 60 and 45 were obtained for the models Random Forest, SVM and Naïve Bayes, respectively.

4 Implementation

In this section, we are going to discuss about data preparation and modelling Techniques.

4.1 Data Loading

In our project, we utilized the "Resume Dataset" that is publicly available on Kaggle. The dataset consists of a CSV file named 'ResumeDataSet.csv' that contains 962 records of resumes along with their corresponding categories.

To use this dataset, we first loaded the CSV file into our code. We then proceeded to perform various preprocessing tasks such as cleaning the text data, removing stop words, and tokenizing the text into sequences of words, in order to prepare the data for training our machine learning models. Overall, this dataset proved to be a valuable resource for our project and helped us to achieve our project objectives.

4.2 Exploratory Data Analysis

As part of our data analysis, we performed exploratory data analysis in three steps. Firstly, we conducted an analysis of the distribution of resumes among different categories. Our dataset consisted of 25 different categories of resumes, and we aimed to understand the distribution of resumes within each category.

Secondly, we visualized the length distribution of resumes using the seaborn library's distplot() function. The plot displayed the length of words on the X-axis and density on the Y-axis. We observed that most resumes had a length of 2000 to 4000 words, while some resumes had a length of more than 8000 words. The mean number of words in the resumes was 3160. This visualization could assist in choosing an appropriate maximum length for padding while preparing the data for the model.

Lastly, we created a visual representation of the most frequently used words in each category of resumes. We used the WordCloud function from the wordcloud package to create a word cloud where the size of the word represented its frequency in the text. This visualization helped us gain insights into the language and vocabulary used in each category of resumes, which could aid in better understanding the data and improving the model..

4.3 Data Preprocessing

We converted all the text into lowercase and then removed punctuations, special characters, quoting text, progressive pronouns, and stop words as they do not carry much meaning on their own and may interfere with downstream analysis. Removing them will reduce noise in the data and can improve analysis

accuracy. We then performed lemmatization to reduce the words to the base form – lemma. This groups the variations of the same word together and reduces the number of unique words in the dataset.

4.4 Data Splitting

We split the dataset consisting of 962 records into train data and test data such that 10% of the data is assigned for testing and 90% of the data is assigned for training. We used the "stratify" parameter which makes sure that the distribution of categories is the same in both the test set and the training set. This will ensure that each category is present in both sets. To make sure the same split is produced every time we run the code, the "random_state" parameter is used.

4.5 Tokenization and Vectorization

Vectorization and tokenization are essential steps in natural language processing (NLP) and are performed to convert textual data into numerical format. During the implementation of the KNN model, we utilized the word_tokenizer() function from the nltk library to tokenize the text data. This function breaks down the text data into individual words or tokens, allowing us to process the data more effectively. To convert the tokenized text data into numerical form, we used the Bag of Words technique. This technique involves generating a vocabulary of words from the input text data, and then creating a matrix with each column representing a word from the vocabulary and each row representing a document. The frequency of each word in each document is then counted and placed in the corresponding cell of the matrix. To generate this matrix, we passed the tokenized text data into the CountVectorizer() function. This function creates a vocabulary of words from the input text and generates a matrix that represents each document's frequency of words. This matrix is then used as an input to train the KNN model.

In implementing the CNN model, we utilized the Tokenizer class from Keras to tokenize the textual data into sequences of integers. This class converts the text into sequences of integers, where each integer represents a specific word from the vocabulary. This allows the model to understand the textual data in a numerical format. To create word embeddings, we utilized the GloVe word embeddings. These embeddings are pre-trained word vectors that associate each word in the vocabulary with a high-dimensional vector. GloVe word embeddings can capture semantic relationships between words based on their co-occurrence in large text corpora. The GloVe word embeddings generate an embedding matrix that represents each word in the vocabulary with a high-dimensional vector. This embedding matrix is then used as the weight matrix for the Embedding layer of the CNN model. The Embedding layer maps the input sequences to a higher-dimensional space and enables the model to learn meaningful representations of the input data. Before feeding the tokenized sequences to the model, the sequences undergo Padding to ensure all input sequences have the same length. Padding involves adding zeros to the end of sequences that are shorter than the maximum sequence length, ensuring that all input sequences have the same length, which is required for training the model. This ensures that all the data fed to the model is of the same length and that the model can learn from the full dataset.

In the context of our DistilBERT model implementation, tokenization is achieved using the AutoTokenizer module from the Transformers library, which creates tokens from the input text data. These tokens are then mapped to their corresponding IDs from the model's vocabulary to generate input sequences. This process involves adding special tokens, padding the sequences, and truncating them to a specified maximum length. After tokenization, the resulting sequences are converted to input IDs and attention masks, which are necessary for DistilBERT's self-attention mechanism to learn the dependencies between the input tokens. Finally, vectorization is done by feeding the tokenized and preprocessed sequences to the DistilBERT model, which generates contextualized embeddings for each token. These embeddings capture the meaning and context of the token in the sentence and are used to perform various NLP tasks such as text classification, named entity recognition, and question-answering.

4.6 Model Building

We built three models for the classification task– KNN, DistilBERT and CNN.

4.6.1 K-Nearest Neighbor (KNN)

K-Nearest Neighbors (KNN) is a machine learning algorithm that has been widely used in both classification and regression problems. It is a simple and easy-to-understand algorithm that operates by storing all the available data points with their corresponding labels. When presented with a new data point, the algorithm determines the K closest data points in the feature space and uses their majority vote or average value to predict the new point's class or regression value.

The KNN algorithm is non-parametric, meaning it does not assume any distribution of the data. Moreover, it is an instance-based algorithm, meaning it stores the data for later use in prediction rather than learning a model from the data explicitly. KNN is popular for low-dimensional data sets, but it becomes computationally expensive for high-dimensional or large data sets as it requires calculating distances between the new point and all the stored data points. KNN is versatile, as it can be used for both classification and regression tasks. In our case, we use KNN for classification. The value of 'K' refers to the number of nearest neighbors that the algorithm considers. We chose an empirical value of K equal to 7, where the seven closest neighbors of a test instance are considered in the classification.

We implemented the KNN model using the 'KNeighborsClassifier()' function from the Scikit-learn module.

By setting the value of 'n' to 7, we specify the number of neighbors that the algorithm should consider in the classification process. This value of K gave us a good result, and it strikes a balance between reducing noise and capturing detailed patterns.

4.6.2 DistilBERT

DistilBERT is a type of pre-trained model in Natural Language Processing (NLP) that is based on the Transformer architecture. It is a smaller, faster, and lighter version of the pre-trained BERT (Bidirectional Encoder Representations from Transformers) model that was designed to reduce the complexity of the original model while maintaining its accuracy. DistilBERT has been developed and optimized by Hugging Face, a leading company in the development of state-of-the-art natural language processing (NLP) tools and libraries.

The architecture of DistilBERT includes a stack of several identical layers of self-attention and feedforward neural networks. These layers help the model to learn the relationships between words in a text by processing it in both directions. In addition to that, DistilBERT also includes a masking mechanism that masks some of the input tokens to help the model generalize better.

The model is trained using an enormous text corpus using an unsupervised approach, meaning that it does not require labeled data. During the training process, DistilBERT learns to predict missing words from a sentence given the surrounding words. This task is known as masked language modeling. Overall, the DistilBERT model is a powerful tool for solving various NLP tasks, including classification, question answering, and text generation, among others. Its smaller size and faster training time make it ideal for tasks with limited computational resources or large datasets.

The architecture of DistilBERT model we implemented can be broken down into two main components: the embedding layer and the transformer layer:

1. The embedding layer is responsible for converting the input text into a set of embeddings that can be processed by the transformer layer. In this implementation, the 'AutoTokenizer' from the Hugging Face Transformers library is used to tokenize the input text and convert it into a set of input embeddings. Due to the limited hardware resources and the limit set by DistilBERT model on the number of tokens, we set the number of tokens for this model to be 512. The tokenizer adds special tokens to the input, such as '[CLS]' and '[SEP]', and pads or truncates the input so that all inputs have the same length. The resulting input embeddings are passed to the transformer layer.

2. The transformer layer consists of a stack of multi-head self-attention mechanisms and pointwise feed-forward layers. These layers are used to model the interactions between the input tokens and produce a set of output embeddings that precisely captures the input text's meaning. In this implementation, the 'TFDistilBertForSequenceClassification' class from the Hugging Face Transformers library is used to implement the transformer layer. This class is a pre-trained DistilBERT model that has been fine-tuned for sequence classification tasks.

The output of the transformer layer is a set of final hidden states, one for each input token. The 'Flatten' layer is used to flatten these hidden states into a one-dimensional tensor that can be passed through a series of fully connected layers. In this implementation, there are four fully connected layers, with 1024, 512, 256, and 128 units, respectively. A batch normalization layer follows each layer that is fully connected and a dropout layer, which are used to regularize the network and prevent overfitting. Finally, the output layer is a fully connected layer with 25 units and a softmax activation function, which produces a probability distribution over the 25 possible output classes.

The Adam optimizer is used to train the model, with a learning rate of 5e-5, an epsilon value of 2e-8, and a decay rate of 0.01. The sparse categorical cross-entropy loss function is used, and the performance of the model is evaluated using the balanced accuracy metric. The training is stopped early if the validation balanced accuracy does not improve for 250 epochs, and the best model is saved based on its performance on the validation set.

4.6.3 Convolution Neural Network.

Convolution Neural Networks (CNN) are a type of neural network commonly used for image recognition and classification tasks. They are designed to mimic the way the human brain processes visual information by analyzing small portions of an image, called filters, and combining them to form a complete representation of the image.

A typical CNN consists of several layers, including a convolutional layer, a pooling layer, and a fully connected layer. The convolutional layer applies a set of filters to the input, producing a set of feature maps that capture different aspects of the input. The pooling layer down samples the feature maps, reducing their size and making the network more computationally efficient. Finally, the fully connected layer takes the output of the previous layers and produces a classification output.

Although CNNs are primarily used in Computer Vision, the reason they are used in NLP is that they can learn useful local features from the text, which can be important for capturing relationships between words. In contrast, LSTM layers work well with sequential data where the order of the input matters. However, resumes are not sequential in nature, and there is no specific order in which the words are arranged. Therefore, CNNs are more appropriate for this task.

Additionally, by stacking multiple convolutional and pooling layers, the model can learn higher-level features that are more abstract and complex, which can be helpful for tasks like sentiment analysis where the meaning of the text is more nuanced. CNN models are computationally efficient and require less training time compared to LSTM models. In addition, the CNN model used in this project has fewer trainable parameters than an LSTM model, which makes it less prone to overfitting.

The CNN model was implemented using the Keras functional API, which is a flexible and powerful way to create complex neural network architectures. The Keras API allows for easy customization and modification of the network architecture, making it easier to experiment with different models. The architecture of the CNN model built is as follows:

1. Input Layer: The first layer of the CNN model is the input layer, which receives the preprocessed resumes in the form of tokens. The number of tokens is 3000, which means that each resume is represented as a vector of 3000 integers.

2. Embedding Layer: The input layer is followed by the embedding layer, which maps each token to its corresponding embedding vector. The embeddings are created using pre-trained GloVe embeddings, which have been trained on a large corpus of text and have been shown to be effective in many natural language processing tasks.

3. Convolutional Layers: Two 1D convolutional layers are then added to the network, with 128 filters each and a kernel size of 3. The ReLU activation function is used in both layers, which helps to introduce non-linearity into the model and improve its performance.

4. Max Pooling Layer: The next layer is the max pooling layer, which is used to reduce the spatial size of the output feature map from the convolutional layers. This layer has a pool size of 2, which means that it selects the maximum value from every two adjacent values in the feature map.

5. Dropout Layers: o prevent overfitting, two dropout layers with a dropout rate of 0.5 are added to the network after the global max pooling layer. Dropout layers randomly drop out neurons during training, which helps to prevent the model from overfitting the training data.

6. Flatten Layer: The flatten layer is then added to the network, which converts the output from the previous layer into a one-dimensional array.

7. Dense Layers: Two dense layers follow the flatten layer, each with 128 neurons and a ReLU activation function. These layers introduce more non-linearity into the model and help to further improve its performance.

8. Output Layer: The final output layer has the ‘softmax’ activation function, which outputs a probability distribution for the 25 categories of resumes. The ‘softmax’ function ensures that the output probabilities sum to 1, making it easy to interpret the model's predictions.

Finally, the model is compiled using the ‘Cross Entropy’ loss function and ‘Adam’ optimizer. The ‘Cross Entropy’ loss function is commonly used in classification problems and is used to measure how well the model can predict the correct category for each resume. The ‘Adam’ optimizer is a popular optimization algorithm that is used to update the model parameters during training.

4.7 Model Evaluation

We Evaluation metrics are used to measure the performance of machine learning models. In our project, we have used three different models, namely CNN, DistilBERT, and KNN, and evaluated them using four commonly used evaluation metrics: Accuracy, Recall, Precision, and F1 score.

Accuracy: Accuracy is a common evaluation metric used to measure the overall performance of a model. It is calculated by dividing the number of correctly classified samples by the total number of samples in the dataset. In other words, accuracy measures how often the model correctly predicts the correct class of a given input.

Recall: Recall is another important evaluation metric that is often used in NLP tasks. It measures the proportion of actual positive cases that are correctly identified by the model. In other words, recall measures the ability of the model to identify all positive cases in the dataset.

Precision: Precision is another commonly used evaluation metric that measures the proportion of true positive cases out of all positive predictions made by the model. In other words, precision measures how often the model correctly predicts the positive class when it predicts it.

F1 score: F1 score is a metric that combines both precision and recall into a single value. It is the harmonic mean of precision and recall and is often used as a more balanced measure of performance in cases where precision and recall have different priorities. In other words, F1 score provides a trade-off between precision and recall.

Overall, these four-evaluation metrics are commonly used in NLP tasks to measure the performance of models. By evaluating our CNN, DistilBERT, and KNN models using these metrics, we can gain insight into the strengths and weaknesses of each model and make informed decisions about which model is best for our specific NLP task.

5 Final Results

The following figures show our results upon initial execution of CNN, KNN and DistilBERT respectively.

```
Accuracy: 0.9896907216494846
Precision: 0.9914089347079038
Recall: 0.9896907216494846
F1 score: 0.9896907216494846
```

Figure 1: Result – Evaluation Metrics for CNN

```
Accuracy: 0.9072164948453608
Precision: 0.9373424971363116
Recall: 0.9072164948453608
F1 score: 0.9072164948453608
```

Figure 2: Result – Evaluation Metrics for KNN

```
Accuracy: 0.6701030927835051
Precision: 0.9224349533627885
Recall: 0.6701030927835051
F1 score: 0.6701030927835051
```

Figure 3: Result – Evaluation Metrics for DistilBERT

MODEL	Accuracy	Precision	Recall	F1 Score
CNN	0.989	0.99	0.989	0.989
KNN	0.907	0.937	0.907	0.907
DistilBERT	0.67	0.92	0.67	0.67

Table 1: Results – Evaluation Metrics Comparison

6 Discussion

In this project, we have implemented three models, K-Nearest Neighbors, Convolutional Neural Networks and DistilBERT models for the purpose of classification of resumes. We achieved the highest accuracy of 98.9 percent for the CNN model and the lowest accuracy of 67 percent has been obtained for DistilBERT model among the three models. As a part of future work, we intend on achieving an even higher accuracy for the CNN model by fine tuning the parameters. We also plan on improving the accuracy of DistilBERT by making modifications to the model architecture and by performing hyperparameter tuning. We are also intending to incorporate more models which can outperform the performance of our currently implemented models and find larger datasets which can facilitate data-driven insights and improve the performance further.

Both of us worked on both code and report. Soundarya Lahari Valipe concentrated on data pre-processing and model building of K-Nearest Neighbors and Convolutional Neural Networks. Venkata Ramana Rohith Neralla concentrated on Data Analysis, Tokenization, BOW and GloVe word embeddings and model building of DistilBERT.

7 Conclusion

In summary, the main objective of this project was to explore the performance of three distinct models, namely K-Nearest Neighbors (KNN), Convolutional Neural Network (CNN), and DistilBERT for resume classification, where DistilBERT is our most recent implementation.

The results of the study revealed that all three models showed promising performance in resume classification, but with varying levels of accuracy, precision, recall, and F1 score. The KNN model, which is a simple and interpretable model, performed reasonably well with an accuracy of 90.7 percent. The CNN model, which is a deep learning model specialized in image and sequence data, showed a higher and the best accuracy and better performance an accuracy of 98.9 percent. The DistilBERT model, which is a pre-trained transformer-based language model, demonstrated the lowest performance, achieving an accuracy of 67 percent.

In conclusion, the outcomes of this project contribute to the literature on resume classification and offer insights into the performance of three different models - K-Nearest Neighbors (KNN), Convolutional Neural Network (CNN), and DistilBERT - in this task. Further research and experimentation can be conducted to explore alternative models or techniques, and to address the limitations and challenges associated with resume classification in real-world scenarios. The results of this study have the potential to provide valuable insights to recruitment agencies and other stakeholders involved in the recruitment process, enabling them to enhance the efficiency and effectiveness of resume screening and selection procedures.

REFERENCES

[1] Kinge, B. et al. (2022) Resume screening using machine learning and NLP: A proposed system, International Journal of Scientific Research in Computer Science, Engineering and Information Technology. Available at: https://ijsrcseit.com/CSEIT228240 (Accessed: April 12, 2023).

[2] S. Bharadwaj, R. Varun, P. S. Aditya, M. Nikhil and G. C. Babu, "Resume Screening using NLP and LSTM," 2022 International Conference on Inventive Computation Technologies (ICICT), Nepal, 2022, pp. 238-241, doi: 10.1109/ICICT54344.2022.9850889.

[3] Riya Pal, Shahrukh Shaikh, Swaraj Satpute and Sumedha Bhagwat, "Resume Classification using various Machine Learning Algorithms", ITM Web Conf. Volume 44, 2022, International Conference on Automation, Computing and Communication 2022 (ICACC-2022), doi:https://doi.org/10.1051/itmconf/20224403011. Available at https://www.itm-conferences.org/articles/itmconf/pdf/2022/04/itmconf_icacc2022_03011.pdf

[3] Gaurav Dutta, Resume Dataset, Version 1, Retrieved from https://www.kaggle.com/datasets/gauravduttakiit/resume-dataset

[4] Jeffrey Pennington, Richard Socher, Christopher D. Manning, GloVe: Global Vectors for Word Representation, Retrieved from https://nlp.stanford.edu/projects/glove/