Job Matching – A Neural Network Approach

Hymavathi Gummudala, Shreyas Aswar

1. Introduction

In today's competitive job market, job seekers and recruiters face the daunting task of matching resumes to job descriptions. This task requires a significant amount of time and effort, which can often result in manual errors and mismatches between job descriptions and resumes. Applying to jobs that do not match the job description can lower the chances of getting hired, and recruiters need to go through thousands of resumes to select the most qualified candidate.

To overcome these challenges, traditional methods such as keyword matching and semantic matching are used. However, these methods may not be efficient enough to handle the complexity of the task.

In this project, we propose a new approach to solve the problem of resumes and job descriptions matching using a neural network approach. Our project aims to use natural language processing (NLP) and Deep learning algorithms to automate the process of matching resumes to job descriptions. The goal is to reduce the time and effort required to match resumes to job descriptions and improve the chances of finding the perfect candidate for the job.

The proposed algorithm will be trained using a dataset of job descriptions and resumes, and the results will be evaluated based on various metrics such as accuracy, precision, recall, and F1 score. The project's outcome will be a novel solution to the problem of resumes and job descriptions matching, which can benefit job seekers and recruiters alike by streamlining the hiring process and increasing the efficiency of the job market.

Our Problem is a Binary Classification but NLP Text Classification Problem.

1. Problem Description:

The process of matching resumes to job descriptions is a critical step in the hiring process for both job seekers and recruiters. However, this task can be a time-consuming and challenging process. Job seekers need to tailor their resumes to match the job description to increase their chances of getting hired. On the other hand, recruiters need to go through thousands of resumes to find the most qualified candidate for the job.

Traditional methods like keyword matching and semantic matching are used for this problem. However, these methods have their limitations, and they may not be efficient enough to handle the complexity of the task. Keyword matching involves matching the keywords in the job description to the resume. While semantic matching uses natural language processing techniques to match the meaning of the job description and the resume.

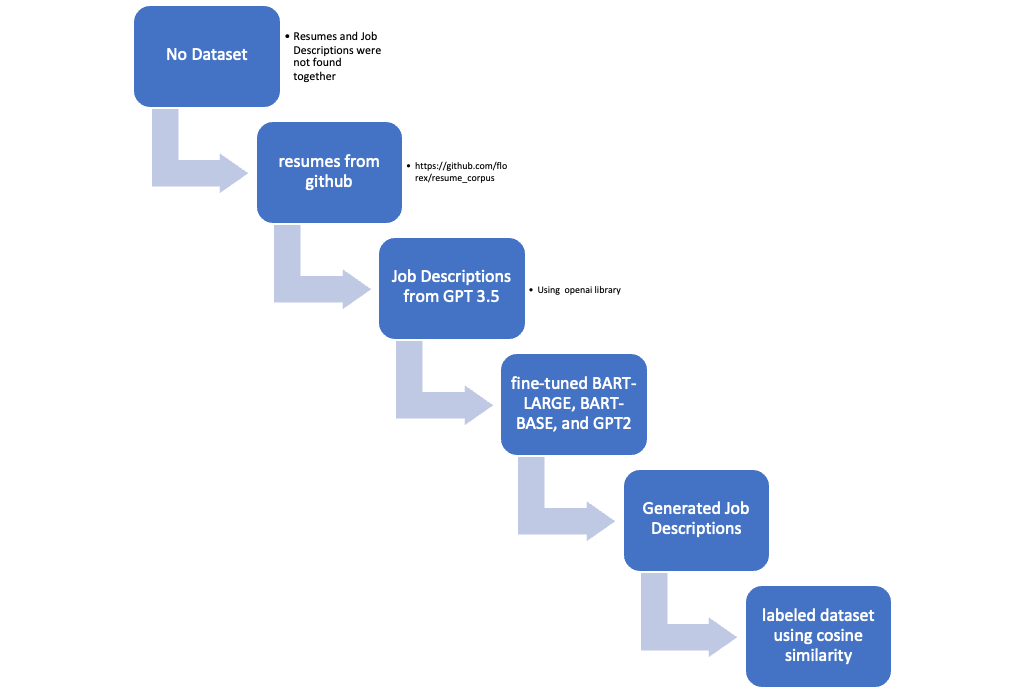
These methods may result in mismatches between job descriptions and resumes due to the use of different keywords or phrasing. Moreover, manual errors can occur by skimming and scanning the job description, leading to an inefficient hiring process.

To overcome these challenges, we propose a neural network approach that uses natural language processing (NLP) techniques to match resumes to job descriptions. The proposed algorithm will be trained using a dataset of job descriptions and resumes. It will take into account various factors such as skills, experience, and education to determine the best match between a job description and a resume.

The objective of this project is to develop an efficient and accurate job matching algorithm that can help job seekers and recruiters alike. The project aims to reduce the time and effort required to match resumes to job descriptions and improve the chances of finding the perfect candidate for the job. The project's outcome will be a novel solution to the problem of resumes and job descriptions matching, which can benefit job seekers and recruiters alike by streamlining the hiring process and increasing the efficiency of the job market.

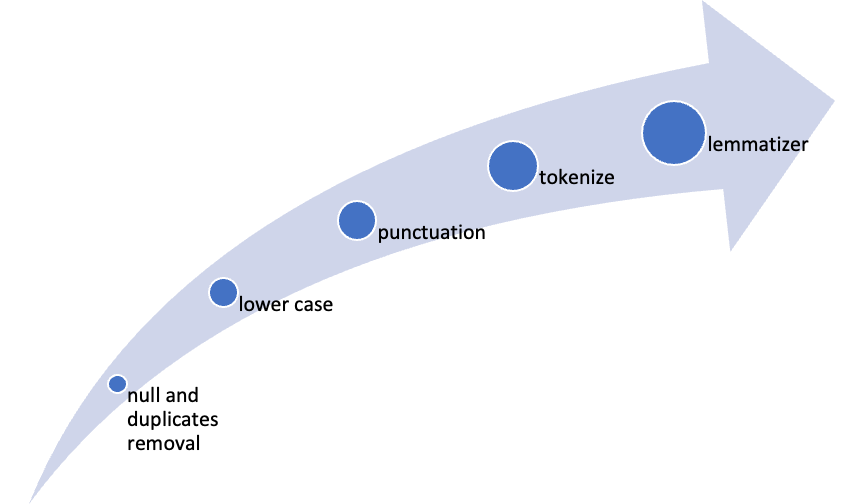
3. Data Description

We were unable to find a dataset containing both resumes and job descriptions, so we downloaded a resume dataset from a GitHub repository <https://github.com/florex/resume_corpus> with 29,783 rows and decided to generate job descriptions using GPT 3.5 with the OpenAI library. To fine-tune the pre-trained models for job description generation, we required a set of data with resumes and job descriptions. Therefore, we generated some job description samples using the resumes we had. We experimented with BART-LARGE, BART-BASE, and GPT2 text generation models for our task and took the transformer data from the Hugging Face platform <https://huggingface.co/models>. Among these models, BART-LARGE performed exceptionally well and gave better results compared to others. To help the neural network model differentiate between matches and non-matches, we added some negative sampling data along with positive sampled data. We labeled our dataset using cosine similarity, with cosine similarity greater than 0.8 labeled as a match and others labeled as not a match.



Data Cleaning:

After obtaining our dataset, we removed null and duplicate values. Since our dataset contained text data, we preprocessed it using various functions such as null and duplicate removal, lower case, punctuation removal, tokenizing and lemmatizing. Initially, we removed stop words and numbers from the text data. However, we realized that removing numbers could lead to losing important information such as experiences and skills with specific names like HTML5, CSS3, D3. Similarly, removing stop words could result in losing context. Therefore, we decided to not remove stop words and numbers to retain the context and not miss any crucial information.



4 Methodology

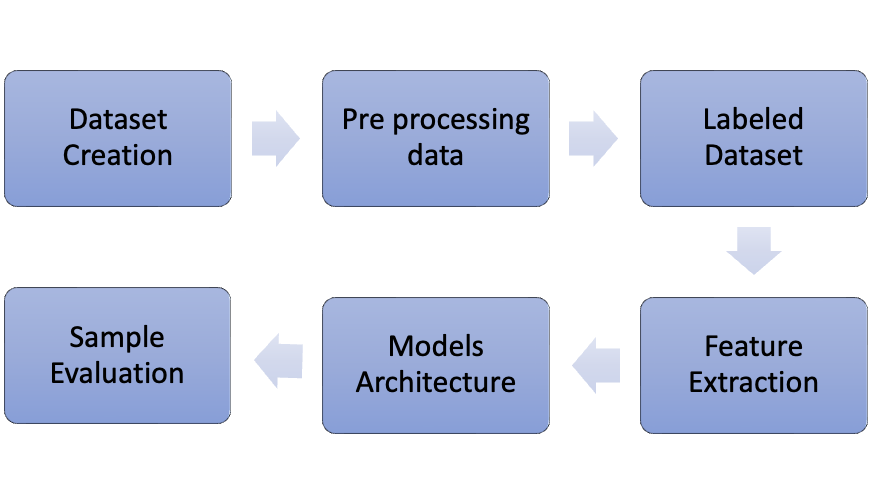
After we created our dataset. We have preprocessed the data by passing the columns of the data frame we were having 21000 resumes and Job descriptions after filtering everything.

Upon calculation of Cosine Similarity, we have made two labels saying that whosoever cosine similarity is less than 0.8 is Not a match and greater than 0.8 assigned to Match.

Coming to feature extraction Word Embedding, TF-IDF, Wor2vec methods were used.

We have created our own Word2Vec model. We earlier planned to take google news or glove word2vec models, but they have content which is not relevant to our problem. So, we have decided to create our own Word2vec model.

Next, coming to Model Architecture we have tried and ran different Models like CNN, RNN, LSTM, BILSTM.

  
In our search for a dataset that combines job descriptions and resumes, we were unable to find one that suited our needs. However, we did come across separate datasets for resumes and job descriptions that we downloaded from a GitHub repository. We obtained 29,000 resumes and 29,000 job designation text files, but we needed a way to generate job descriptions for each resume. After some research, we determined that fine-tuning pre-trained models could accomplish this task.

To pre-tune the models, we needed a small amount of combined data, so we used GPT 3.5 to generate one-seventh of the job descriptions we needed. We worked with three pre-trained models - BART-Large, BART-Base, and OpenAI-GPT - and found that BART-Large yielded the most fruitful results due to its extensive training.

Before labeling the dataset, we knew that preprocessing was crucial for accurate similarity scores. We removed null and missing values, duplicate entries, and punctuation and new line characters, and converted everything to lower case. We also performed lemmatization, but we decided against removing stop words and numbers. Removing stop words could obscure the context of the data, and numbers in resumes are crucial for indicating experience and certain alphanumeric skills.

After preprocessing the data, we calculated similarity scores between resumes and job descriptions using various methods such as cosine similarity, Jaccard similarity, and Euclidean distance. Of these, cosine similarity was most effective at differentiating and producing good scores.

We then created a labeled dataset with outputs of Match and Not a Match, where cosine similarity < 0.8 was labeled as Not a Match and cosine similarity > 0.8 was labeled as Match. With our dataset in hand, we wanted to use a neural network approach to job matching instead of existing methods like Keyword Matching and Semantic Matching.

We decided to use RNN, CNN, BILSTM, and LSTM models for our text data problem. Given that our problem was binary classification with Match or Not a Match as output, we expected RNN, BILSTM, or LSTM to perform well.

In our feature extraction process, we used Word Embedding, TF-IDF, and Word2vec methods. While we initially considered using pre-trained models such as Google News or GloVe for Word2vec, we found that the content within these models was not directly relevant to our specific problem. Therefore, we made the decision to create our own Word2vec model instead.

By creating our own Word2vec model, we were able to train it on our specific dataset, resulting in more relevant and accurate word embeddings. This allowed us to extract more meaningful features from the text data and ultimately improve the accuracy of our models. Overall, our decision to create our own Word2vec model proved to be a valuable step in our feature extraction process.

We ran the models with multiple layers and assessed the results, then we hypertuned parameters such as learning rate and added dropout and L2 regularizations. We also created our own word2vec model and experimented with different architectures.

The results of the different models are listed in the Results Section. We have included the classification report of the two best-performing models.

Results:

We have run different models with different layers and dropouts, Regularization Included.

Please find the accurate results before and after the presentation. We have included the word2vec model created by us instead of using google news and Glove.

Accuracy has been increased from 84 to 96

|  |  |  |  |
| --- | --- | --- | --- |
| Models | Layers | Hyper Tuning Parameters | Results - Accuracy |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Also attaching the Classification Report:

Limitation and Difficulties:

1. we faced an issue with API calls per minute

2.OPENAI API supports only 4000 approx. tokens for a single request - so we had to limit the job description length to 500 tokens

3. For Labeling the dataset we tried with cosine , Jaccard, Euclidean and Manhattan. We have gone with Cosine as it was more suitable for our project.

References:

https://github.com/florex/resume\_corpus

https://www.kaggle.com/code/akashkotal/resume-screening-with-nlp

https://www.kaggle.com/datasets/gauravduttakiit/resume-dataset

https://www.kaggle.com/discussions/general/202112

https://huggingface.co/facebook/bart-large

https://chat.openai.com/

<https://datastock.shop/download-indeed-job-resume-dataset/>

Appendix:

Contribution of Hymavathi

Contribution of Shreyas