Job Matching – A Neural Network Approach

Hymavathi Gummudala, Shreyas Aswar

May 3, 2023

1 Introduction

In today's competitive job market, if you are a job seekers you will only get a job if your profile matches the job description closely and if you're a recruiter you will want to hire someone who fits perfectly to the job description. Both 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. 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 different 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 project includes various models to be trained using a dataset of job descriptions and resumes, and the results are evaluated based on various metrics such as accuracy, precision, recall, and F1 score. The project's outcome is Match or Not a Match.

2 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.

In [1], the author used cosine similarity to match the resumes and job description, while it is a good metric, it might get affected based on the number of occurrences of a particular skill in a resume and job description. One might get low similarity score if the resume is not having the particular word with the same frequency as the job description. While in [2], matching is done just based on the skills extracted from both resumes and job description. 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.

3 Data Description

For this particular problem, in order to train neural networks models we require huge dataset which contains Resumes, Job Description and the label if they are matching or not. After excessive research we were unable to find a dataset containing both resumes and the relevant job descriptions. However, we were able to find a huge dataset of alone resumes GitHub repository with 29,783 rows. For these resumes we needed relevant job descriptions, and we decided to leverage transfer learning for this part. In order to generate relevant job descriptions using some text generating pretrained model we require it to fine-tune. We decided to gather data for fine tuning pre-trained models using [3] 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. Using gpt 3.5 we generated around 4000 Job Descriptions exactly relevant to the respective resumes provided in the prompt to the gpt 3.5. This 4000 resumes and job descriptions were used to fine-tune various pre-trained models. We experimented with BART-LARGE, BART-BASE, and GPT2 text generation models for our task with the help of Hugging Face platform.

Among these models, BART-LARGE performed exceptionally well and gave better results compared to others. We found that GPT2's generated job descriptions were more innovative however BART-LARGE generated more apt job descriptions. 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.

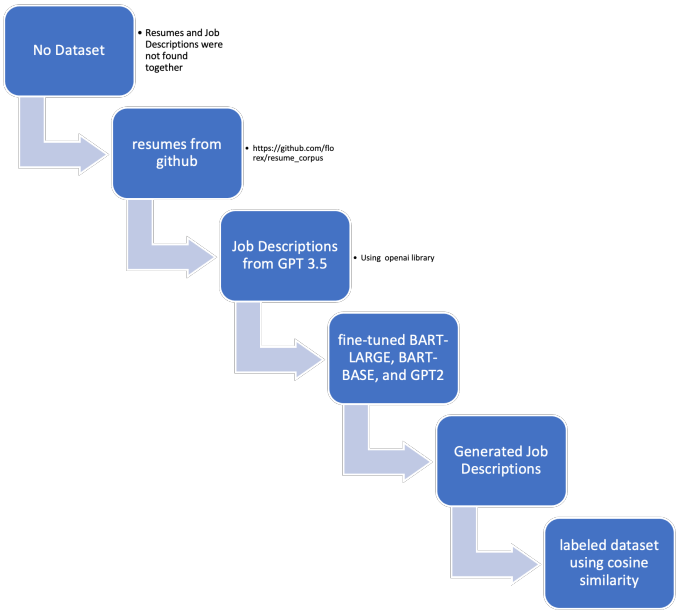

Figure 1: .Data Set Creation Procedure

3.1 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.

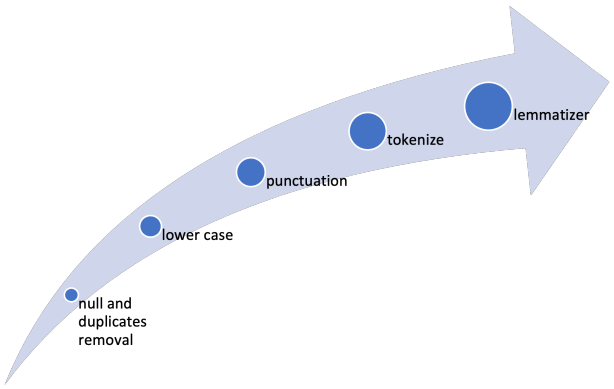

Figure 2: .Pre Processing Steps

4 Methodology

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 [2] 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 fine-tune the models, we needed a small amount of data which could contain exactly relevant job descriptions for their respective resumes. To accomplish this we used GPT 3.5 to generate one-seventh of the job descriptions we needed. We first removed the stopwords from the resumes and job descriptions before feeding it to the pre-trained models in order to fine-tune them, however this resulted in job descriptions being generated without stopwords. To tackle this we required to include stopwords and then train the pre-trained models on the fine-tuning data we collected from gpt3.5. 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 performing data preprocessing on the resumes and job descriptions, we proceeded to calculate similarity scores using different methods, such as cosine similarity, Jaccard similarity, and Euclidean distance. Out of these methods, cosine similarity emerged as the most effective approach for producing meaningful and distinguishable scores.

One of the main reasons for choosing cosine similarity over other methods is that it is well-suited to handling high-dimensional data, such as text documents, which is often the case with resumes and job descriptions. Cosine similarity measures the cosine of the angle between two vectors, which reflects the similarity of their orientations regardless of their magnitudes.

Another advantage of cosine similarity is that it is able to capture the semantic meaning of the text by taking into account the relative positions of the words in the document. This makes it more effective in identifying subtle similarities between documents, which can be crucial when matching resumes and job descriptions. In contrast, Jaccard similarity is based solely on the presence or absence of words and does not take into account their ordering or frequency, which can result in less accurate similarity scores.

Overall, our experience shows that cosine similarity is a reliable and effective method for matching resumes and job descriptions, particularly when dealing with large and complex data sets.

For the negative sampling, we took random resumes from one domain and matched them randomly

with job descriptions form different domain. The random matching allowed us to have great number of negative samples in the data set.

We then labeled our dataset with labels as Match and Not a Match, where cosine similarity less than 0.8 was labeled as Not a Match and cosine similarity greater than 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 plotted the similarity score distribution using Seaborn and Matplotlib we can see the dip is the clear distinction between Match and Not a Match.

Hence we came up with 0.8 to keep as Distinction.

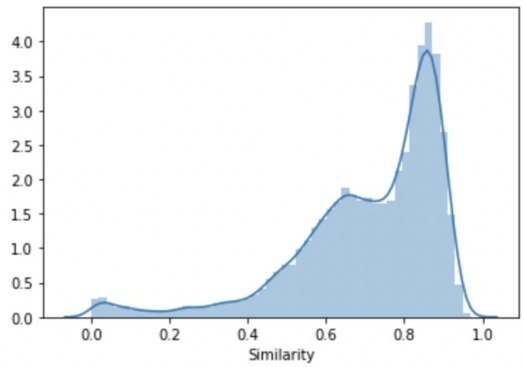

Figure 3: Plot to find the peak of Similarity

In addition to calculating similarity scores using various similarity metrics, we also experimented with different feature extraction techniques to further improve the accuracy of our resume-job description matching system. These techniques included widely-used tools like CountVectorizer, TF-IDF Vectorizer, WordEmbedding, as well as a custom feature extraction method that we developed ourselves.

CountVectorizer is a simple method that generates a matrix of word counts for each document, while TF-IDF Vectorizer calculates the term frequency-inverse document frequency (TF-IDF) values for each word, which reflects its importance in the document relative to the entire corpus. WordEmbedding is a more advanced technique that captures the semantic meaning of the words by representing them as dense vectors in a high-dimensional space. Word2Vec is a popular WordEmbedding method that learns word vectors based on the context in which they appear.

We created our own word2vec model and used it for the embedding layers.

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

5 Results

We conducted various experiments by implementing different models with distinct layers and incorporating dropouts and regularization techniques. Additionally, we utilized our own word2vec model instead of relying on pre-trained models such as Google News or Glove.

As a result of these efforts, we were able to significantly enhance the accuracy of our models from 84 to 98. This improvement was observed both before and after we presented our findings.

Layers are written in the format of input+ hidden+ output and due to time constraint we were not able to run for more number of epochs.

Model	Layers	Hyper-Tuned_Parameters	Testing_Accuracy	Training_Accuracy
LSTM	1+3+1	Dropout,Learning rate	96.40	91.65
BILSTM	1+3+1	Dropout,Learningrate	98.92	98.96
RNN	1+2+1	Learning rate, dropout	81.94	81.72
CNN	2+2+1	Learning rate, dropout	74.29	74.21

5.1 Sample Evaluation

We have evaluated with the best performing model and checked the Job Description and resume Matching output.

21658	Data Analyst	hymavathi gummudala indianapolis in 8129742574...	primary responsibility responsible management ...	0.564531	Not a Match
21659	Data Analyst	hymavathi gummudala indianapolis in 8129742574...	job description position data science associat...	0.847708	Match

Figure 4: sample evaluation

5.2 Classification Report of the biLSTM Model

```
print(classification_report(y_val, y_pred))
```

	precision	recall	f1-score	support
0	0.99	0.98	0.99	1915
1	0.98	1.00	0.99	2054
accuracy			0.99	3969
macro avg	0.99	0.99	0.99	3969
weighted avg	0.99	0.99	0.99	3969

Figure 5: Classification report of the best performing model

6 Limitations 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.

7 References

1. https://github.com/binoydutt/Resume-Job-Description-Matching
2. https://github.com/florex/resume_corpus
3. https://chat.openai.com/
4. https://www.kaggle.com/code/akashkotal/resume-screening-with-nlp
5. https://www.kaggle.com/datasets/gauravduttakiit/resume-dataset
6. https://www.kaggle.com/discussions/general/202112
7. https://huggingface.co/facebook/bart-large
8. https://datastock.shop/download-indeed-job-resume-dataset/

8 Appendix 1- Contributions from each team member

Task	Teammember
Idea selection	Hymavathi
Data set Resume Collection	Shreyas
Pre trained Models - BART	Shreyas
Pre trained Models - Openai	Hymavathi
Job Description Generation	Shreyas
Cosine Similarity	Hymavathi
Pre processing the data	Hymavathi
Models- BILSTM, LSTM	Shreyas
Models- CNN,RNN	Hymavathi
Classification Report	Shreyas

Figure 6: Contributions

This project can be reproduced using the python notebooks uploaded on the following Github page.
Project Github Link: https://github.com/shreyasaswar/job-matching