CareerBuilder - A Job Recommendation System

Srinivas Manda
IIIT Bangalore, India
Srinivasa.Bhargava@iiitb.ac.in

Karanjit Saha
IIIT Bangalore, India
Karanjit.Saha@iiitb.ac.in

Monjoy Narayan Choudhury
IIIT Bangalore, India
Monjoy.Choudhury@iiitb.ac.in

Arya Kondawar
IIIT Bangalore, India
Arya.Kondawar@iiitb.ac.in

***Abstract*—In today's competitive job market, finding the right job can be a challenging task. To address this problem, we have designed and implemented a job recommendation system that helps users find their dream job by using relevant keywords obtained from their Curriculum vitae / Resume and from Job Postings available on websites like LinkedIn, Glassdoor, Internshala, etc. The system is designed to provide personalized job recommendations to users based on their skills, experience, and career goals.**

***Index Terms*—Job Recommendation System, Siamese Network, Word2Vec, Transformers, BERT, Content-based filtering, Web Scraping, Natural Language Processing, Self-supervised learning.**

I. Introduction

In recent years, the job market has become increasingly competitive, with a growing number of job seekers vying for limited job openings. In this scenario, finding the right job that matches a user's skills and career goals can be a daunting task. To address this problem, we have designed and implemented a job recommendation system that uses relevant keywords obtained from a user's Curriculum vitae / Resume and from Job Postings available on websites like LinkedIn, Glassdoor, Internshala, etc.

The proposed system uses state-of-the-art machine learning techniques such as Siamese network, Word2Vec, and transformers to identify the most relevant job postings for a user and recommend them accordingly. The system takes into account the user's skills, experience, and career goals, and provides personalized job recommendations in real-time.

To make the system accessible to a wide range of users, we have developed a full-stack web application that is easy to use and intuitive. The application allows users to upload their CV/Resume and receive personalized job recommendations based on their profile.

The proposed job recommendation system has several potential applications in the job market, including helping job seekers to find the right job that matches their skills and career goals. Furthermore, the system can help employers to identify suitable candidates for their job openings.

In this paper, we describe the design and implementation of the proposed job recommendation system. We also present experimental results that demonstrate the effectiveness of the system in providing relevant job recommendations to users.

II. Dataset Creation

A. Web Scraping the data from LinkedIn

To ensure that our job recommendation system provides up-to-date recommendations, we used web scraping techniques to collect job postings from LinkedIn. We used Selenium and Geckodriver to scrape job listings, and we assumed the location to be India, with job postings mostly related to the tech industry.

B. Datasets from Kaggle

To provide specialized recommendations, we needed a corpus of text that was specific to our application and demographic. We used the Kaggle dataset "Skill Set" [1] to obtain skill set data for our users. Additionally, we obtained a dataset of job postings based on the United States from the "USA Job Postings" [2] dataset, which we used for pre-training our models.

III. Preprocessing Steps

The Job Description we obtained would be unfiltered, hence we perform the following text-cleaning steps on it:

1) Removed HTML elements.
2) Removed "https://" text.
3) Removed # and @ symbols which are commonly used in job description text found on social media sites like LinkedIn.
4) Removed punctuations
5) Removed extra whitespaces
6) Removed any non-ASCII characters for smoother parsing and tokenization.

IV. Our Approaches

We divide our solution into 2 major black boxes:-

- Embedding Generation Block
- Ranking Network Block

The Embedding Generation Block works to create a good representation of the text which is extracted from the resume of the individuals as well the job description text. The discussion

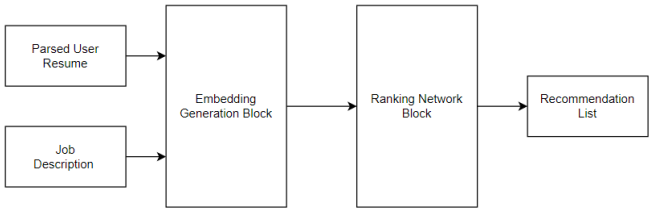

Fig. 1. Our Approach

on embedding generation block is discussed in the upcoming section.

Once the embeddings are generated they are passed through a ranking network which can be trained on a supervised learning task with labels on the affinity of a user towards a particular job post. The concept of affinity itself can lead to various use cases of the recommender system. Unluckily we didn't have any labeled dataset and labeling would have been a time-consuming task hence we suggest the networks based on some simple possible affinity metrics which are discussed in Section VI

V. Language Model for Word Embeddings Generation

We tried two language models to generate suitable word embeddings for our job recommendation system:

1) Word2Vec
2) BERT-based fine-tuned models.

We discuss both techniques in the upcoming part.

A. Word2Vec

Word2Vec is a natural language processing (NLP) approach that was first published in 2013. To learn word connections from a vast corpus of text, the Word2Vec technique employs a neural network model. Once trained, a model of this type can find synonymous words or suggest extra words for an incomplete text. These are shallow, two-layer neural networks that have been taught to recreate word linguistic contexts. Word2Vec takes a large corpus of text as input and outputs a vector space with several hundred dimensions, with each unique word in the corpus allocated a matching vector in the space. For our implementation, we went ahead with the Gensim Word2Vec [3] and used the preprocessed text with stopwords removal and passed through TF-IDF to train on it.

However, Word2Vec being a shallow model might not work well for our task since job descriptions are typically long. Additionally, if we trained it on our specific dataset, Word2Vec's vocabulary would be limited to our dataset, and it may not be able to handle new words introduced to the model.

Therefore the need of sequence-based modeling is necessary to overcome these challenges. Naturally, RNN and LSTMs come into mind, but the problem with the two is the long training time due to a lack of parallelization. Transformers overcome this and considering many State-of-the-art models already use transformers we planned to switch towards a transformer-based architecture that can capture the meaning of words in the context of the sentence.

B. Transformer Based Architectures

We identified two transformer architectures that can be useful for our use case. They are as follows:

1) Job-skill-sentence-transformer-tsdae [5]
2) JobBERT [4]

Job-skill-sentence-transformer-tsdae: The Job-skill-sentence-transformer-tsdae is a fine-tuned Sentence Transformer model based on state-of-the-art unsupervised method based on pre-trained Transformers and Sequential Denoising Auto-Encoder (TSDAE). The training for sentence embeddings is done by adding a certain type of noise (e.g. deleting or swapping words) to input sentences, encoding the damaged sentences into fixed-sized vectors, and then reconstructing the vectors into the original input. This model was trained on AskUbuntu, StackExchange, SciDocs, and Twitter paraphraser datasets originally.

JobBERT: JobBERT is a much more typical BERT-based transformer that is trained in a self-supervised way from a bert-base-cased model on 3.2M sentences from job postings. The paper tries to prove that domain-adapted models significantly outperform their non-adapted counterparts.

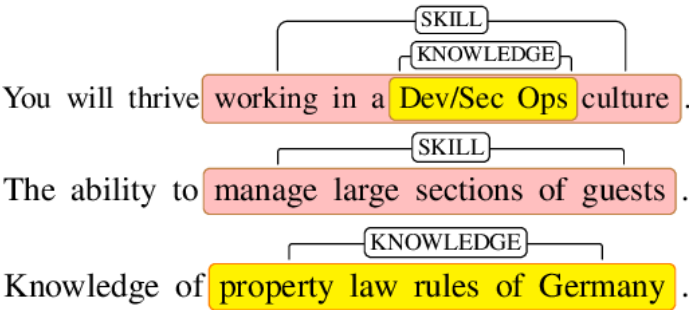

Fig. 2. Image Courtesy of [4]: Components of a job posting

We take some ideas from this paper, especially on capturing domain-specific information for our embeddings. However, just like any model these models too are trained on datasets that pertain to a specific demography and not the demography our recommender system is targetting. Hence, we decide to proceed toward a self-supervised learning paradigm to add the demographic information using our datasets to the transformer model.

C. Self-Supervised Learning for Improving Pretrained Models

Self-supervised learning uses labels generated from the data without any manual or weak label sources. The main idea behind self-supervised learning is to hide or modify part of the input and ask the model to recover the input or classify what has changed.

To improve our pre-trained models for job recommendation, we used the masked language modeling technique. We hid

15% of the words in the input by using a $\langle MASK \rangle$ token. The model tries to recover this and in the process ends up fine-tuning the encoder for a better representation of words. By doing this, we were able to learn domain-specific representations without the need for hand-labeled data. By leveraging the structure of the input data, we were able to train our models to capture the underlying patterns and relationships in the data.

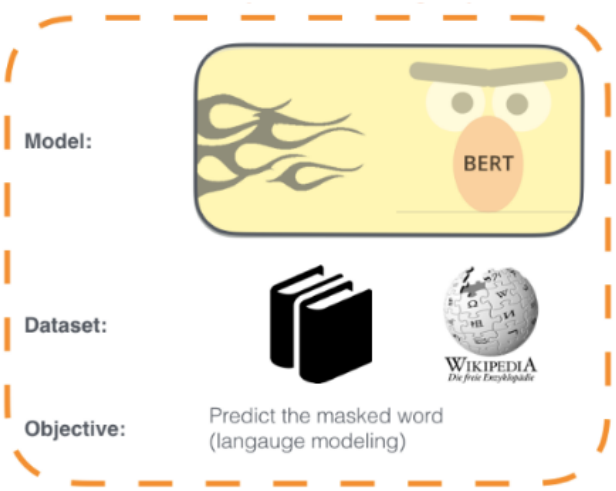

Fig. 3. Courtesy of [6]: The idea of Self-Supervised Learning

VI. Resume Parsing and Keyword Extraction

To obtain the important sections from the resume like skills and projects. We needed a parser that took a pdf file in and gave out all the required sections of the resume. This was done using the following sequence of steps:

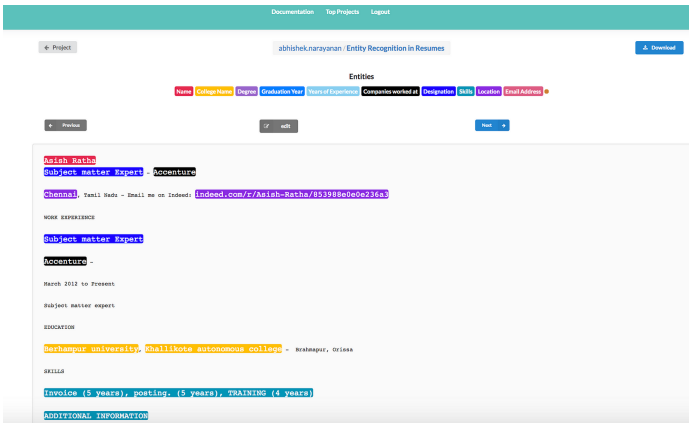

Fig. 4. Courtesy of [8]: Named Entity Recognition on Resume

1) Breaking the document into sentences, lines followed by tokens.
2) Applying a POS tag on it.
3) Extracting the information on this preprocessed text:
   a) Features like email, phone number can be simply identified using regular expressions. For eg: an email will have xx@xyz.com.
   b) To obtain the sklls section our task converts into a named entity recognition task. There is also a similar brute force way to just find a substring "Skills".

In the implementation, all these come pre-built in a package called Pyresparser [7].

VII. Ranking Network Block

Once the user embedding(from skills section of the resume) and the job embeddings are obtained, they are cached on the server to increase the speed of inference. For ranking the embeddings with the user embedding, we employ 2 approaches for defining the ranking network.

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
          Linear-1               [-1, 1, 128]          98,432
          Linear-2               [-1, 1, 128]          16,512
          Linear-3                 [-1, 1, 1]             129
          Linear-4               [-1, 1, 128]          98,432
          Linear-5               [-1, 1, 128]          16,512
          Linear-6                 [-1, 1, 1]             129
================================================================
Total params: 230,146
Trainable params: 230,146
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 2.25
Forward/backward pass size (MB): 0.00
Params size (MB): 0.88
Estimated Total Size (MB): 3.13
```

Fig. 5. Network Architecture Summary

1) Naive Approach: Here we simply compute the cosine similarity between the two types of embeddings. This is much more geometric-based and is simple to perform on an unlabelled dataset. Once the values are obtained we organize them in decreasing order of similarity and present it to the user.
2) Siamese Network: This is a feed-forward network that attempts to learn the affinity of the embeddings. The concept of affinity itself is subjective in nature based on the target of the recommender system. Due to the lack of labeled data, we were unable to validate much on our network. We tried to show the working of the network using the resume dataset and jobs web scrapped to predict the cosine similarity. Due to the simplicity of the target function, the network was easily able to fit this function.

VIII. Full Stack Application as a Proof of Concept

To show the viability of the application we created a full-stack application using the following tech stack:

1) NextJS as the frontend
2) Flask Server with Pytorch 2.0 hosted on an Nvidia GTX 1660Ti Machine. Our final model in this application

comprises of the "job-skill-sentence-transformer-tsdae as the embedding generator" for users whose details are obtained using pyresparser and the ranking network is a simple cosine similarity computation.
3) Firebase for user authentication and cloud storage.

IX. Performance

1) For the Siamese Network we have the following statistics when trained on cosine similarity as ground truth values:
Testing Error (MAE) = $(8.8 * 10^{-5})$ on Train-test ratio = 0.7:0.3 with L1-norm as the loss metric.
2) Runtime: In the full stack application, the click-to-result display time is about 3-5 seconds. This includes the time to process the request get the inference, rank them, and send the JSON output which is rendered by the front-end. The test is run on a list of about 1000 jobs and user details.

X. Some Drawbacks Of our approach

Like every other system, our system too is not flawless by design but has some downsides. We address those problems in this section.

A. Inheriting the problems of a Transformer

The model is limited by the number of tokens it can process for an instance (512). This restricts the size of the input job description sentence to be limited to a size < 512 (CLS, SEQ, etc .. tokens are also a part of the tokens extracted). To fix this we performed a mean pooling with a sliding window which helped us tackle the problem upfront. However, the flexibility that an RNN or LSTM would have provided is sacrificed for the parallelisability which a transformer gives.

Other than this transformers are hardware intensive especially RAM and GPU RAM intensive. Due to the lack of proper resources, we aren't able to leverage all the data we collected for fine-tuning.

B. Lack of labeled data and consequently an evaluation metric

In the case of job recommendation, there is no explicit rating defined and hence, we cannot associate an affinity score with the user embeddings (obtained from the resume dataset) and the job embeddings which would have helped us for supervised training. We tried to use cosine similarity as the baseline to address this. However, that led to the Siamese network fitting the criteria in a low number of epochs. We cannot prove our hypothesis that it can do well for actual data also or would it need more hyperparameter tuning?

C. Problems with Cosine Similarity as criteria

Cosine similarity is limited in its ability to capture some features of meaning: Cosine similarity is a measure of vector similarity, however, it does not capture all dimensions of meaning. Two phrases using different terms to represent the same notion, for example, may have a lower cosine similarity score than two sentences using identical words but with distinct meanings. This can be understood with the example: [9]

1) same meaning: "I try to avoid the company of gamblers" and "We avoided the ball"
2) different meaning: "You must carry your camping gear" and "Sound carries well over water".

However, a cosine similarity metric would not capture the similarity in the 1st one and would rate a less value even if they mean the same thing as the words are dissimilar. The opposite can be seen in the second example. That is why the need for explicit feedback makes it easy to avoid such assumptions of using cosine similarity.

XI. Future Work

The following can be the course of action for this project to take in the future to make a better recommender system.

1) Working towards a combined resume text and Job preference dataset where a user rates the job postings they see with some regression value. This can allow us to personalize better and leverage the ranking network which we suggested.
2) Access to APIs from Job Posting websites that would allow us to fetch the latest job data when a query is made by the user and prevent viewing stale information.
3) Addressing the issue of scalability. For now, we do a user comparison with all the job embeddings we have. This is not scalable if the number of embeddings is very large. We need to find some way to incorporate the comparison for large data. Along with this, we need to find a way to update our models automatically so that it improves over time with more data it sees.
4) Collect user history with proper consent which would allow us to create some sort of collaborative filtering-based paradigm which can improve the quality of recommendations.

Links

All the work done in the project can be found in the public repository on GitHub

References

[1] https://www.kaggle.com/datasets/gauravduttakiit/resume-dataset
[2] https://www.kaggle.com/datasets/jsrshivam/job-recommendation-case-study?select=jobs.tsv
[3] https://radimrehurek.com/gensim/models/word2vec.html
[4] Zhang, M., Jensen, K.N., Sonniks, S.D. and Plank, B., 2022. Skillspan: Hard and soft skill extraction from English job postings. arXiv preprint arXiv:2204.12811.
[5] Wang, K., Reimers, N. and Gurevych, I., 2021. Tsdae: Using transformer-based sequential denoising auto-encoder for unsupervised sentence embedding learning. arXiv preprint arXiv:2104.06979.
[6] Jay Almar's Visual guide to BERT: https://jalammar.github.io/illustrated-bert/
[7] Pyresparser: https://github.com/OmkarPathak/pyresparser
[8] https://github.com/DataTurks-Engg/Entity-Recognition-In-Resumes-SpaCy
[9] Zhou, K., Ethayarajh, K., Card, D. and Jurafsky, D., 2022. Problems with Cosine as a Measure of Embedding Similarity for High Frequency Words. arXiv preprint arXiv:2205.05092.