CareerBuilder

A Job Recommender System

Srinivas Manda - IMT2020001

Karanjit Saha -IMT2020003

Arya Kondawar- IMT2020088

Monjoy Narayan Choudhury- IMT2020502

Problem Statement

To design a job recommendation system for a user by using relevant keywords obtained from their Curriculum vitae / Resume and from Job Postings available on websites like LinkedIn, Glassdoor, Internshala etc.

This is designed to help the users to get their dream job. By using this recommendation system users can take key decisions that could shape their career.

Some aspects of the problem statement

- Since job postings are time-sensitive one cannot rely on static datasets from Kaggle to use as outputs for the recommender system.

- Since, there is no concept of rating a job posting, hence traditional methods like Collaborative Filtering(both user and item based) is not viable in this scenario.

- The job postings mostly comprise of text based modality, so our major focus should be on understanding the nature of the text data used.

Dataset Preparation

- The data used for our solution came from the following sources:-
  - Web Scraping:-
    - Since, job postings are time sensitive, we wanted our recommendations up to date.
    - Most pretrained approaches are trained on general corpus of text and is not specialised on application, demography/local information under which the data is collected.
    - We used Selenium and geckodriver to scrape job listings from LinkedIn.
      - We assumed the location to India for now and the job postings are mostly related to tech.

Dataset Preparation

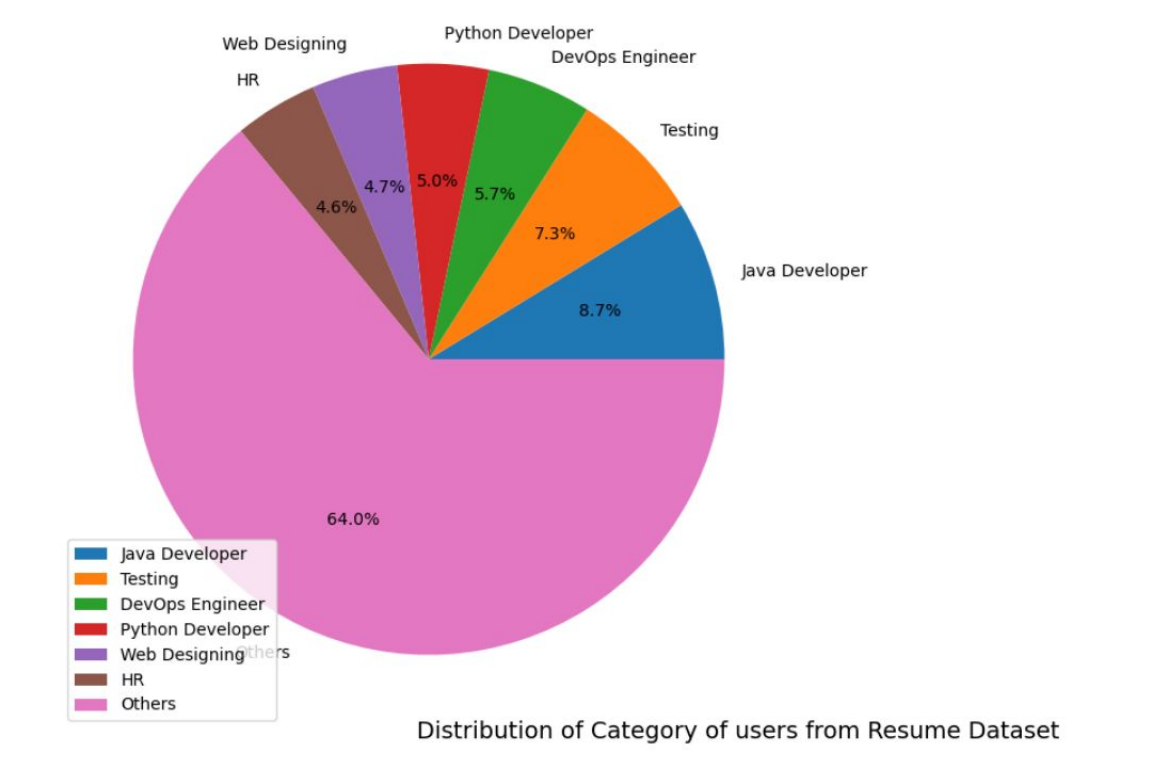
- For one of our approaches, we needed skill set data of the users, for this we used the Kaggle dataset (“Skill Set”).

- We also had found another dataset on job postings(based on USA), which we used for pretraining of our models(to be discussed later). For this we used the dataset (“USA Job Postings”).

Preprocessing Steps

- Text Cleaning:- The Job Description we obtained would be unfiltered, hence we perform the following text cleaning steps on it:
  - Removed HTML elements.
  - Removed “https://” elements.
  - Removed # and @ s which are commonly used in job description text found on social media sites(LinkedIn).
  - Remove Punctuations.
  - Remove Extra whitespace.
  - Removing any non ASCII character(for smoother parsing and tokenizer).

Exploratory Data Analysis

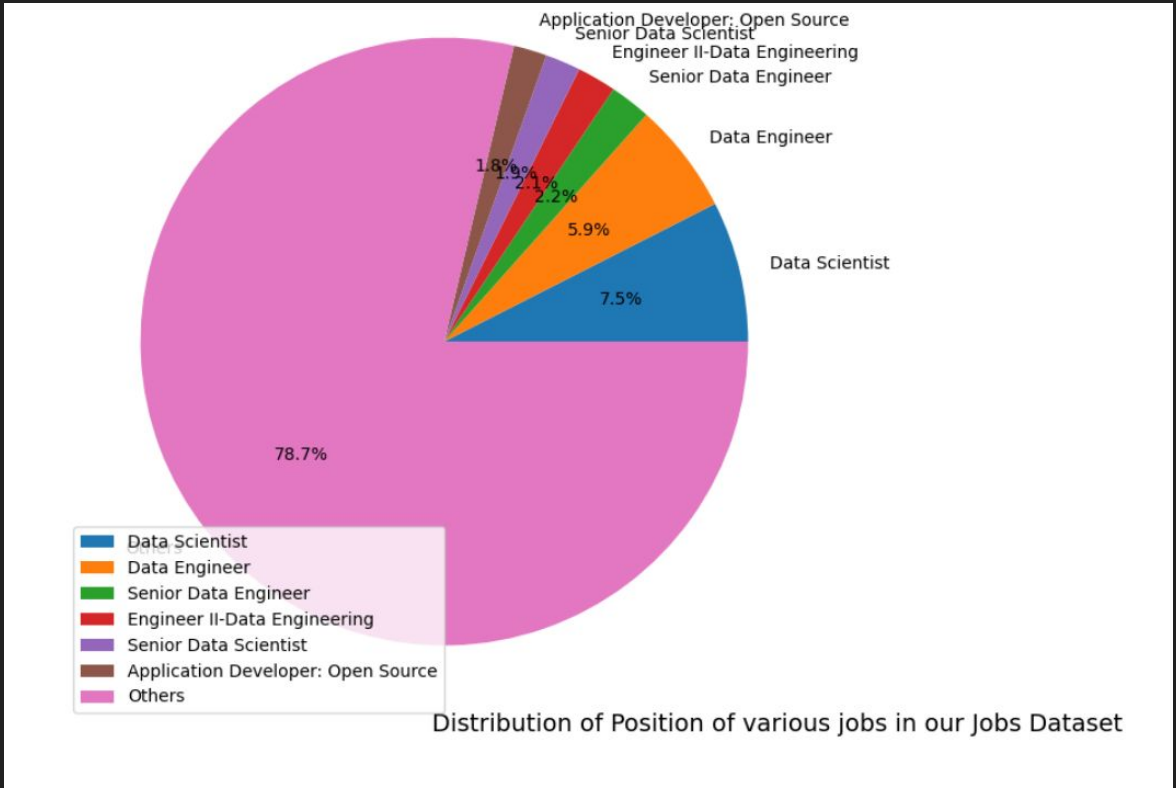


The pie chart shows the distribution of the category of users in our Resume dataset.

Here for better visualisation we have only shown the top 6 and combined the rest in “Others” section.

From the pie chart we can see that the Resume dataset for the users mostly comprises of users from the tech field.

Exploratory Data Analysis



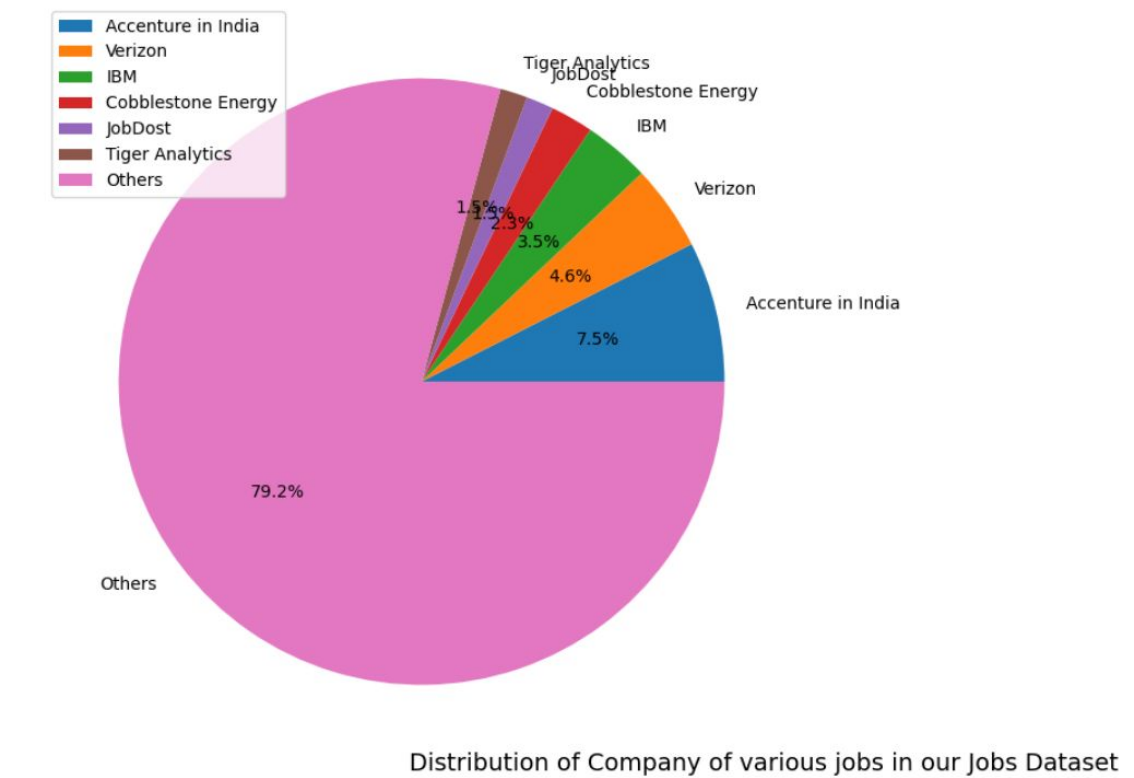
The pie chart shows the distribution of the various Positions in our Job dataset.

Here for better visualisation we have only shown the top 6 and combined the rest in “Others” section.

From the pie chart we can see that the Job dataset mostly comprises of Data engineer related jobs(since it was created in such a way).

We have similar dataset for other Positions as well but it cannot be implemented currently due to hardware constraints.

Exploratory Data Analysis



The pie chart shows the distribution of the various Companies in our Job dataset.

Here for better visualisation we have only shown the top 6 and combined the rest in “Others” section.

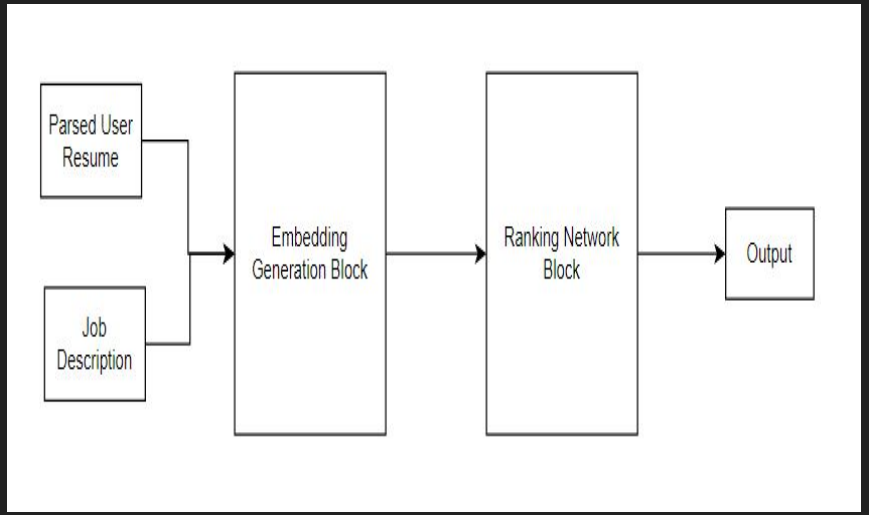
From the pie chart we can see that the Job dataset is not heavily dominated by a single company.

Our Approach

We divide our solution into 2 major black boxes:-

1. Embedding Generation Block

2. Ranking Network Block



Embedding Generation

We have tried using the following language models to generate suitable word embeddings:-

- Word2Vec
- Fine-tuned BERT based transformer model
  - job-skill-sentence-transformer-tsdae
  - jjzha/jobbert-base-cased

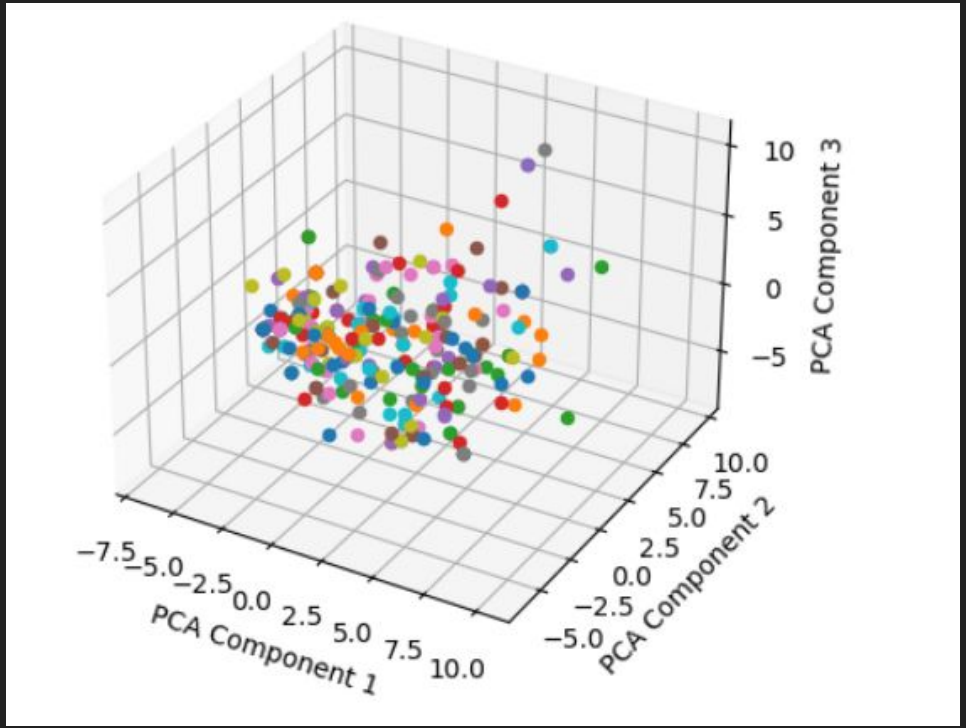


Fig: This is a 3D representation of the word embeddings from the Description column of our job dataset (restricted to only 50 words for clarity) in our dataset.

Word2Vec

- Since we have job descriptions which are very large in length, hence a model like Word2Vec might not be able to work well for the task as compared to transformers, since it is shallower in depth. Hence we have put more focus on using transformers and related architectures to improve the embeddings
- Another problem with Word2Vec is that if it is trained on our dataset then it's vocabulary will be based on our dataset. Hence if a new word is introduced to the model it would not be able to handle it.
- We can address the problem mentioned above by using a pre trained word2Vec model. But then again we will lose on the specificity of our task.
- It captures only the meaning of the word but does not take into consideration other words in the sentence, which may play an important role in determining the meaning of that word.

Skipping LSTMs and RNNs

- Since RNNs suffer from the problem of vanishing gradients, which are resolved in LSTMs. Hence we skip RNNs and directly jump to LSTMs.

- Due to LSTMs autoregressive nature, it is slow and if we use it we will not be able to take advantage of our GPUs.

- Transformers, in general perform better than LSTMs, and are highly parallelizable. Hence, we are going to skip LSTMs and jump directly to transformers, to get the most optimum solution for our task.

Transformer based models used in embedding generation

Job-skill-sentence-transformer-tsdae

- A fine-tuned SentenceTransformer model on jobs and skills descriptions using the Transformer-based and Sequential Denoising AutoEncoder training method which is used mainly in tasks where you lack labelled data.
- 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

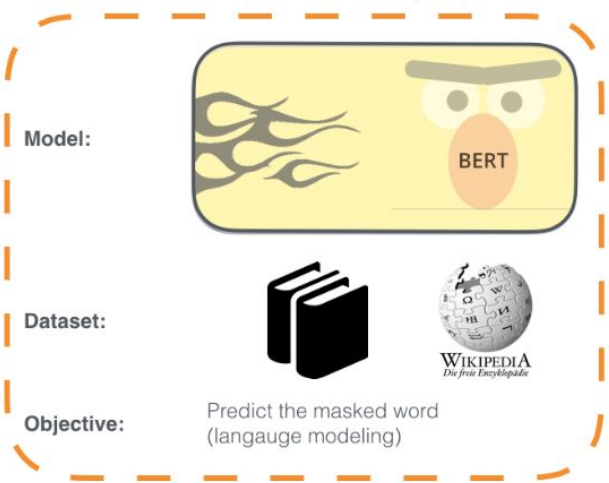
jjzha/jobbert-base-cased

- This model is continuously pre-trained from a bert-base-cased checkpoint on ~3.2M sentences from job postings.

Since both of these models are trained on job descriptions and related text from different regions. We want our model to understand job description from India well, hence we feel a pretraining step using the datasets we have is necessary.

ref: SKILLSPAN: Hard and Soft Skill Extraction from English Job Postings
TSDAE: Using Transformer-based Sequential Denoising Auto-Encoder for Unsupervised Sentence Embedding Learning

- Self-supervised learning (informal definition): supervise using labels generated from the data without any manual or weak label sources
- The main idea here is to hide or modify part of the input. Ask model to recover input or classify what changed.
- We hide 15% of the words by using a <MASK> token. This is exactly similar to how BERT trains the encoder!
- This helps us to learn domain specific representations without the cost of hand labelling data.



CS229 Notes
Jay Almar's visual guide to BERT

Resume Parser

We use a python package pyresparser to parse the resume in pdf format and it works as follows:

- Breaking the document into sentences, lines and tokens and applying a POS tags.
- Information extraction
  - Features like email address and phone number can be identified using regular expressions(an email will always have @ symbol).
  - To capture skills, we apply named-entity recognition techniques. This can also be done in a simplified way, by looking for the caption “skills” but the package does it with other NLP techniques (NER).

Downstream Task: the recommendation step

Once we obtain the user embedding(from skills section of the resume) and the job embeddings(we maintain them locally to speed up the inference process). We try 2 approaches for defining the ranking network.

- **Naive Approach:** We simply compute cosine similarity between the user embedding and all the job embeddings. We would then rank them on the basis of decreasing value of cosine similarity.
- **Siamese Network:** It is a similarity learning feed forward network.

Siamese Network architecture

Siamese architecture has both the inputs passed through a similar internal network and then asses the outputs based on some metric (L1 distance in this case).

The model faces no issues in learning a simple cosine similarity.

```
----------------------------------------------------------------
        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
----------------------------------------------------------------
```

Performance

- Siamese Network:
  - Testing Error = (8.8*10^-5)
  - Total Samples = 10,000
  - Ground Truths put as cosine similarity value
  - Train-test ratio = 0.7:0.3
- Run Time:
  - In application backend: about 3-5 seconds on flask server on a local machine
    - Will improve with a better server CPU, GPU.

Challenges

1) **The Problem with a transformer based model**:- 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).
   a) **Potential Fix** :- Sliding Window with mean pooling, use of CLS token.
2) **Lack of Labelled data**:- 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.
   a) **Potential Fix:- We initially planned to use number of people who applied for the posting as some kind of score.**
      i) **However, this would still not address whether the given user in question liked the posting or not(in the dataset).**
      ii) **While implementing(web scraping) due to security and privacy measures we were unable to extract this feature.**
      iii) **Also the values after certain range (200) the numbers are truncated to 200.**

Challenges

- **Lack of resources :-** Even in the pre training step we were unable to use the large job listings datasets, due to lack of enough system RAM as well as GPUs.

- **Coming up with a relevant evaluation metric:-** Due to the lack of ground truths, to compare user and job embeddings. We are unable to generate satisfactory statistics to support our hypothesis.
  - For the Siamese network, we tried to create a dataset between user and job embeddings, with ground - truth value equal to the cosine similarity between them.
  - However, the network was easily able to learn this similarity metric, which lead to negligible testing loss ($8.8 * 10^{-5}$ on a dataset of about 10000 rows).
  - In our current setting a user survey would have helped us validating our model but time didn't permit us to do so.

Application Development - Tech stack and final model

We present a Full Stack Web Application as a proof of work. Our tech stack comprises of :

- Frontend made using NextJS.
- Machine learning model deployed on Flask server, which also serves as backend.
    - Our final model in this application, comprises of the “job-skill-sentence-transformer-tsdae as the embedding generator” for user whose details are obtained using pyresparser and the ranking network is a simple cosine similarity computation.
- Firebase Database for user authentication and storage of resume data.

Novelty

- Web scraped data used to create a relevant representational model of our system.
- Self-supervised learning used to make embeddings more relevant in the common context, that is to understand and represent job embeddings for job description in India better.
- Full stack application where a user can register (using their Gmail ID) and pass their resume to get job recommendations.
- Personalised recommendations to users supported by using their resume data as well as local job embeddings.
- It handles newer jobs (cold start problem).

Further Improvement Aspect

- Exploring other similarity metrics for automated labelling.
- Obtaining legitimate API keys to get implicit parameters which may help us to come with an affinity score .
- Expanding the domain of the project to other professions.
- Address issues on scalability of our current solution:
    - For now we are comparing one user with all postings. Is there a way to efficiently handle this in a large dataset setting (Clustering?).
    - To come up with an real time updation of dataset to address the staleness of job postings.
- To leverage user histories to come up with some form of Collaborative Filtering models to improve recommendations.

THANK YOU