STACKCONNECT

Connecting individuals to their career interests

CMPT 733: PROGRAMMING FOR BIG DATA II

FINAL PROJECT REPORT

GROUP NAME: Datagrammers

GROUP MEMBERS:

Harikrishna Karthikeyan, Roshni Shaik,

Sameer Pasha, Saptarshi Dutta Gupta

TABLE OF CONTENTS

1. MOTIVATION AND BACKGROUND

Have you ever been in the market looking for jobs that match your skillset? Have you ever looked at what skills are needed to land your dream job? All of us have gone through this at least once in our lives, but every job hunt is different from the other and unique to the individual. It takes a considerable amount of time to look for relevant jobs that match our skills as well as look for the next big tech skill to bet our careers on. Our motivation behind doing this project was to save time for the job seekers and suggest useful technologies to upskill themselves with so they can land their dream job. Our project provides career suggestions as well as temporal and location-based trends in major technologies so as to facilitate the end-user in making an informed decision. We look at the user's StackOverflow activity, job postings scraped from LinkedIn and Indeed, and combine that with glassdoor reviews and salary insights to provide job recommendations using tag matching to the end-user. The data science tools and techniques that we used in the project are explained in the following sections of the report.

2. PROBLEM STATEMENT

For this project, we started by scraping the job postings from Indeed and LinkedIn job portals. We further made use of publicly available StackOverflow dataset that is hosted in BigQuery to get user information like the questions asked, answers provided and overall profile statistics like upvotes, downvotes, etc. We also consolidated the salary insights and company reviews from the Kaggle Glassdoor dataset that is available publicly. We extracted relevant tags from the questions and answers provided by the user from his StackOverflow profile and matched it against the key skills required from the description of the job posts that we scraped to suggest jobs to the end-user. We also present temporal trends in various technologies and immediate future projection in technology trends, so the user can know what to expect and work towards equipping himself with the right technologies. We have also implemented a semantic search strategy that improves upon existing search utilities in StackOverflow by taking into account the popularity and sentiment in the user answers. Finally, we predict tags for the question asked by the user so that it reaches the right audience.

Some of the challenges that we faced in the above steps were as follows:

- **Data Collection:** The data required to suggest jobs were not readily available. Web scraping techniques were used to scrape LinkedIn and Indeed websites. Accessing the StackOverflow and Glassdoor datasets was tricky as there was no standard schema defined for the various tables.

- **Data Integration:** Combining the data scraped into a uniform format to remove duplicate job listings proved to be difficult as the matching criteria used to eliminate duplicates were difficult to come up with.
- **Extracting** the relevant user tags and matching them to the skills extracted from the job descriptions proved to be very tricky and required careful analysis.
- **Evaluating** and improving the performance of the model to obtain relevant job recommendations.
- **Integrating** all the features of the project into a unified platform was challenging.

3. DATA SCIENCE PIPELINE

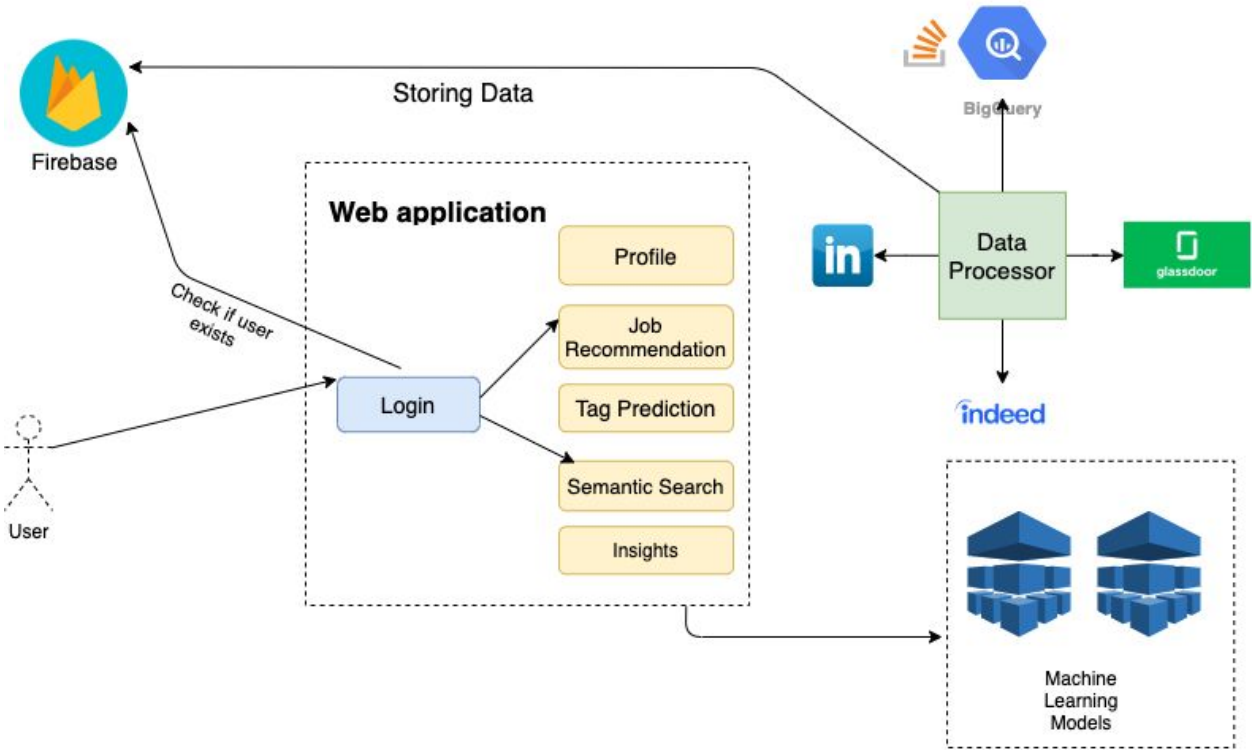

Fig 1. Data flow diagram

3.1 Data Collection

For the project requirements, we needed datasets containing the latest job postings from some of the major job portals, which were not readily available. We also needed a reliable way to access company reviews and salary insights about various positions and roles which were

difficult to gather. This made Data Collection one of the very important tasks for our project. To obtain these datasets, we crawled through thousands of job positions on LinkedIn and Indeed career portals. We used BeautifulSoup - a python library along with Requests - another python library, for downloading and processing HTML files. We had to crawl through each website in a different manner due to the unique HTML page structure used by the websites. Around 40k job postings and about 30k job postings were collected from Indeed and LinkedIn respectively. We also accessed the publicly available StackOverflow dataset hosted in BigQuery to gather user information and other things needed for our tag prediction task. Since the dataset was very large in size(about 200 GB), we had to write optimized queries to look for only the exact information that we needed. For getting company insights and salary reviews, we accessed the Glassdoor dataset which was made publicly available through Kaggle. It contained information about reviews for each company, the pros, and cons of working at the company and finally the salary offered for jobs offered at the company.

3.2 Data Cleaning and Integration

Having collected all the data mentioned above, the next big task was cleaning the datasets and integrating them. Due to the inherent nature of websites, there was no uniform website HTML structure and no single solution works for all of them.

- **Indeed:** The job postings from the Indeed job portal were extracted using BeautifulSoup and Requests python modules. The main fields that were scraped include the Job title, Job Description, Company Name, Company Location, and Salary offered. Most postings did not have salary information and thus this field had a lot of blank values. To overcome this, we relied on salary insights from the Glassdoor dataset to provide an average estimate for the salary that can be expected for any given role.
- **LinkedIn:** LinkedIn website had a completely different website structure which required us to write a completely new crawler. We had to remove HTML markup tags from the scraped data. The fields we scraped include Job title, Job Description, Company Name, Company Location, and Salary offered. We again made use of salary insights provided by the Glassdoor dataset to provide expected salary information to the end-user for those listings that did not have this information.
- **Integrating scraped data:** The main task when it came to integrating the scraped values from LinkedIn and Indeed was to eliminate duplicate records. We found a lot of similar posts from the same companies on both websites. We used the values in the Job Title, Company Name and Company Location primarily to remove duplicate records. This helped us arrive at a central repository for resolving all job-related queries.
- **StackOverflow:** The StackOverflow dataset which is hosted on BigQuery was very large

in size(over 200GB). We had to optimize our search queries to only access the tables required for extracting the information we needed. We accessed data like the user profile information, the number of questions asked, the number of answers provided by the user. We use this information to extract relevant user tags which we used to match against job profiles later.

- **Glassdoor:** The data provided by Kaggle on Glassdoor was highly unclean and unformatted. The schema for the tables was not defined properly which made accessing the dataset very challenging. We made use of the Glassdoor dataset to gather reviews about the companies, the salary for the different roles that they offered.

Overall, we performed tasks like HTML markup tags removal, null values removal, stop words removal, case conversion to facilitate string matching, abbreviation transformation to a unified format, stemming and lemmatization of the words to achieve efficient Data Integration.

3.3 Exploratory Data Analysis(EDA)

In order to have a better understanding of the data collected and integrated for making the recommendation and tag prediction problems, we performed extensive exploratory data analysis on various features of each dataset. This step was highly significant in helping us understand the distribution of salary and company ratings across different roles and positions in the dataset and also in extracting the relevant tags from the user questions and answers. Performing EDA enabled us to know what features are important in integrating the glassdoor dataset effectively. EDA helped us arrive at the decision for using specific attributes from each dataset for use in both the major Data Science applications of our project, namely, career recommendations and tag predictions.

The following is the list of some of the important analysis that led us to the decisions we made:

- Temporal trend analysis on StackOverflow dataset which gives a chronological view of the evolution of technology over time.
- Location-based trends on the StackOverflow dataset which gives insights about the trending technology in the given area, and also the list of areas where a given technology is popular.
- WordCloud of the most common tags across different time periods to supplement the analysis of the temporal trends in technology.
- Bigram and Trigram visualizations of the Job Title and Job Descriptions that were used to extract the top skills required from each job posting.

3.4 Feature Engineering and Modelling

The unstructured text of the Job Title and Job descriptions were used to extract skills using the TF-IDF(term frequency-inverse document frequency) statistic to determine how important a given skill was in a corpus of job postings. For the tag prediction problem, we made use of the Word2Vec model for text embedding before feeding our input into an LSTM model trained to predict tags based on the user's activity on StackOverflow.

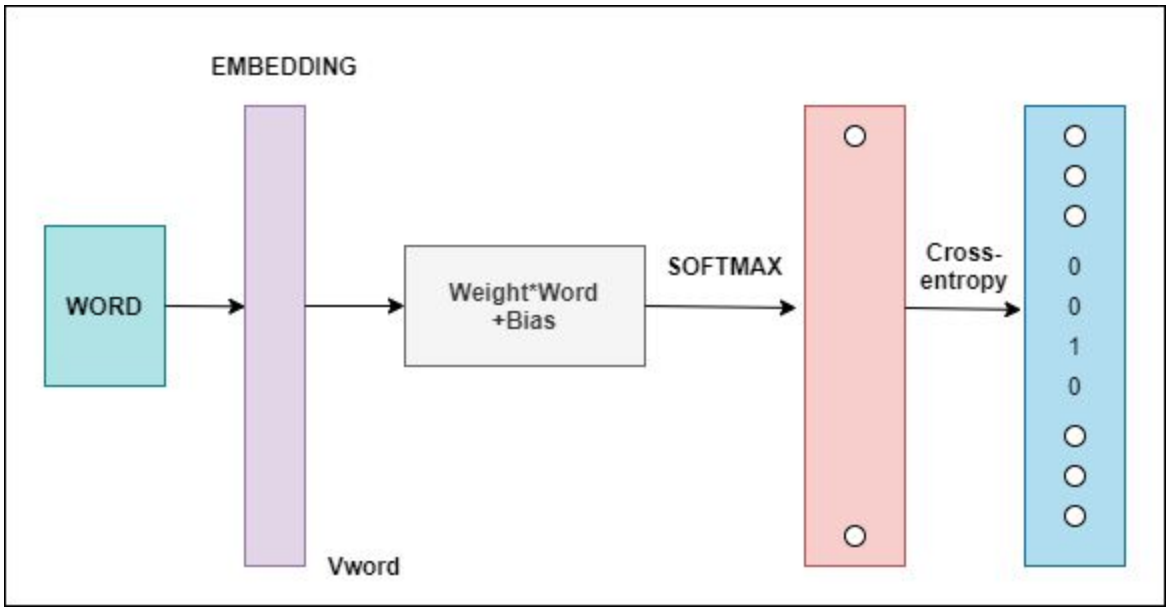

Fig 2. Working of the Word2Vec model

For the job recommendation system, we utilized CountVectorizer along with the Jaccard Similarity metric to match the user tags to the skills from job postings to provide relevant jobs to the user. We also implemented a semantic search strategy that takes into account the popularity of the user, sentiment of the answer's provided by the user along with the traditional cosine similarity metric to improve upon the existing search implementation provided by StackOverflow. We can clearly see the results the semantic search has made to the search query in the screenshots below.

4. METHODOLOGY & EVALUATION

4.1 Tag Prediction

For predicting tags from a given user query, we first needed information about the questions

answered by the user. So, we fetched the title, question body, answer for the question, tags used for the question and votes for each answer. Once we fetched this data from Google Big Query, we stored this information in a pandas dataframe for easier processing.

In order to construct a corpus, we needed to group all the answers by concatenating them based on their common questions and tags. We also added the scores for each answer in order to get a collective score for an entire question.

In order to make the data obtained useful, we first needed to:

1. Convert raw text into tokens
2. Convert tokens to lowercase
3. Remove punctuations
4. Remove stop words

Numeric data was not removed as it would remove precious contextual information. We also do not perform stemming and lemmatization while preprocessing as we do not want to alter domain-specific terms used in our corpus and risk losing precious information.

A new feature called the ‘post_corpus’ is constructed by combining the title, question body, and all the answers. The title is prepended to the question body. Then, 2 features for sentiment namely ‘sentiment_polarity’ and ‘sentiment_subjectivity’ are added using the Textblob library.

The processed data is then stored as a CSV, this file is then read and used for further creating our model which can help in generating tags for a given user query.

Although we did filter out a lot of tags in the Data Preparation step, there still exist a lot of "stray" tags which may appear only once or twice as compared to thousands of occurrences for other tags. This increases the dimensions of the ground truth data, which is not desirable for our model. Thus we extract the top 500 tags based on their occurrences to reduce the dimensions of ground truth (Given that we had roughly ~140k data points, 500 tags seemed to be a good number to get good results).

Since we are dealing with multi-label data, i.e. there is more than one answer for each input, we decided to use the **MultiLabelBinarizer** from sklearn. For instance, If we have total tags as [python, Django, Flask, Nginx,gunicorn], our ground truth of [python, flask] would get converted to [1,0,1,0,0].

In order for our model to understand the raw text data, we needed to vectorize it. Bag of Words and TF-IDF are very common approaches for vectorizing. However, the sparse nature of the BOW and TFIDF would pose a problem. Thus, we decided to go for Word Embeddings, which

are dense vector representations and thus were perfect for our neural network.

The Stackoverflow data is very technical and uses a very specific vocabulary of words, it is not a good idea to use pre-trained WordEmbedding because they are trained on plain English text and would not be able to understand the relations between the words in our vocabulary. Thus we decided to train a WordEmbedding model from scratch. We trained a **Word2Vec** model on the tags.

In order to have a constant size for the input, we used a predefined size. Thus, any input sequence more than this size was truncated and padded by adding a special PAD token for sequences smaller than it. Finally, we constructed an embedding matrix that contains the vectors for all tokens in our vocabulary. This acts as a weights matrix for the Embedding Layer in our neural network.

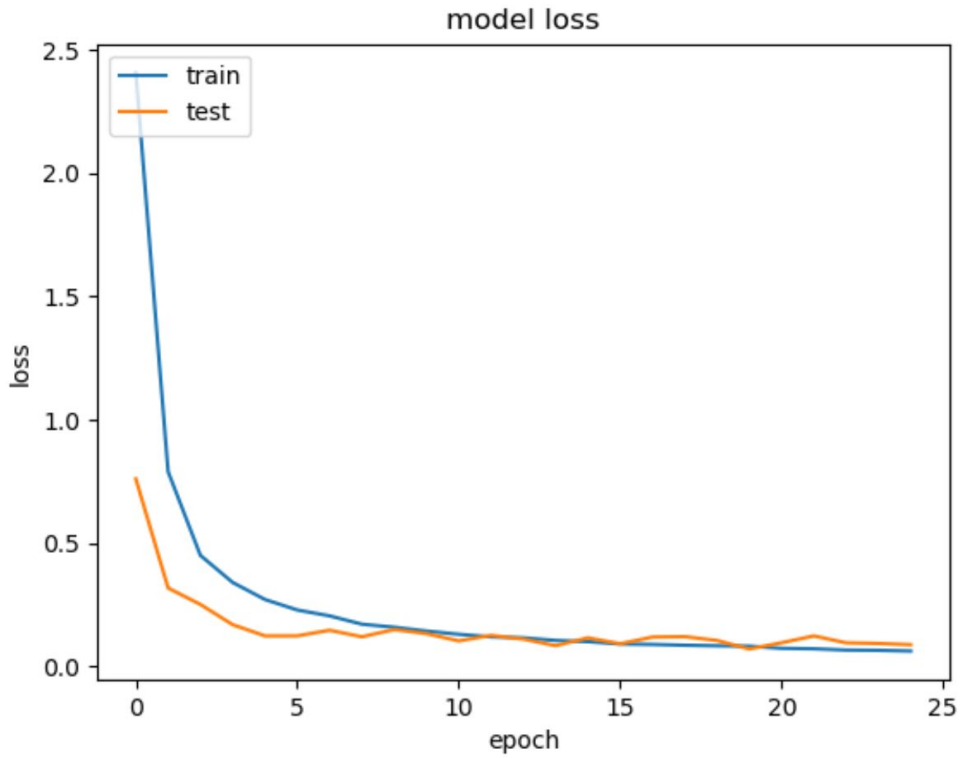

Fig 3. Model loss of the Tag prediction model for the training and testing data

Since we are dealing with a multilabel classification problem here, we cannot train the model using a binary cross-entropy loss. Thus we used log loss applied on each class separately and then computed it's mean based on the number of classes as per the formula given below.

$$-\frac{1}{m}\sum_{i=1}^{m}\sum_{j=1}^{k}\left(y_j^{(i)}\log\hat{y}_j^{(i)}+\left(1-y_j^{(i)}\right)\log\left(1-\hat{y}_j^{(i)}\right)\right)$$

We used an LSTM layer right after the Embedding layer because of the proficiency of LSTMs in working with sequential and text data. Subsequent Dense layers were added along with Dropout regularization in order to build a robust model. Below are the precision and recall values of some of the top 10 from the dataset:

	precision	recall	f1-score
python	0.80	0.82	0.81
c#	0.89	0.75	0.81
android	0.75	0.73	0.74
java	0.54	0.48	0.51
iPhone	0.56	0.58	0.57
c++	0.72	0.81	0.76
PHP	0.88	0.83	0.85

Fig 4. Precision, Recall, and F1- score values for top 7 tags

4.2 Semantic Search

In our semantic search implementation, based on the user query, we returned an existing question that most closely resembles the user's query. In order to calculate the embeddings for an entire sentence, we needed to take the average for the embeddings for each valid token. Since the number of titles we had was fixed, we saved the sentence embeddings for all titles in a CSV file to save computation time on future runs.

So the way we actually calculated the most similar results is by comparing how far each result is from the query in terms of distance. This can only be done if both the query and the results are in a shared vector space. Fortunately, that is exactly what our word embeddings achieve - They create each sentence as a vector in the embedding space, which made it easier for us to distinguish them.

After we had those vectors, we assigned a Similarity measure as a metric that measures the

closeness of two vectors. We decided to assign a custom similarity measure. It is defined as follows:

Final Similarity Measure(q, t) = Cosine distance(q, t) + 0.1*Normalized Vote Score(t) + 0.4*Sentiment(t)

1. It considers the cosine distance as a base measure.
2. It takes into account the popularity of the post based on the votes it has received by users at StackOverflow.
3. It also takes into account the overall sentiment of the responses that people have made. A positive sentiment entails that the answers were helpful and thus is a good post.

4.3 Job Recommendation

For the job recommendation system, we had first obtained the list of the top 10 tags from the list of questions answered by the user by running a query joining the questions and answers tables based on the user and questions id on the StackOverflow dataset. Second, we had taken each of the job descriptions and job titles from the LinkedIn and Indeed datasets and preprocessed them to make our corpus. The preprocessing steps include converting each word to lower case, removing punctuations, stemming and lemmatization. Subsequently, we had made use of the following three methods to find the most predominant keywords from the titles and descriptions:

1. **CountVectorizer**: This converts a collection of text documents into a matrix of token counts. The CountVectorizer can specify the lower and upper boundary of the range of n-values for different word n-grams to be extracted.
2. **Term frequency-inverse document frequency (TFIDF):** TFIDF shows how important a word is to the entire document or corpus. In applications like text mining and information retrieval, it behaves like a weighting factor.
3. **Jaccard Similarity:** The Jaccard coefficient measures similarity between finite sample sets, and is defined as the size of the intersection divided by the size of the union of the sample sets and is given by the formulae: **$J(X, Y) = |X \cap Y| / |X \cup Y|$**

In our model, we ran the CountVectorizer to count the number of times a token shows up and compute its weight. An n-gram analysis of our job description showed that the unigram and bigram models are best suited for the CountVectorizer. For example, in the figure below the bi-gram analysis of the job description had been shown for our sample data.

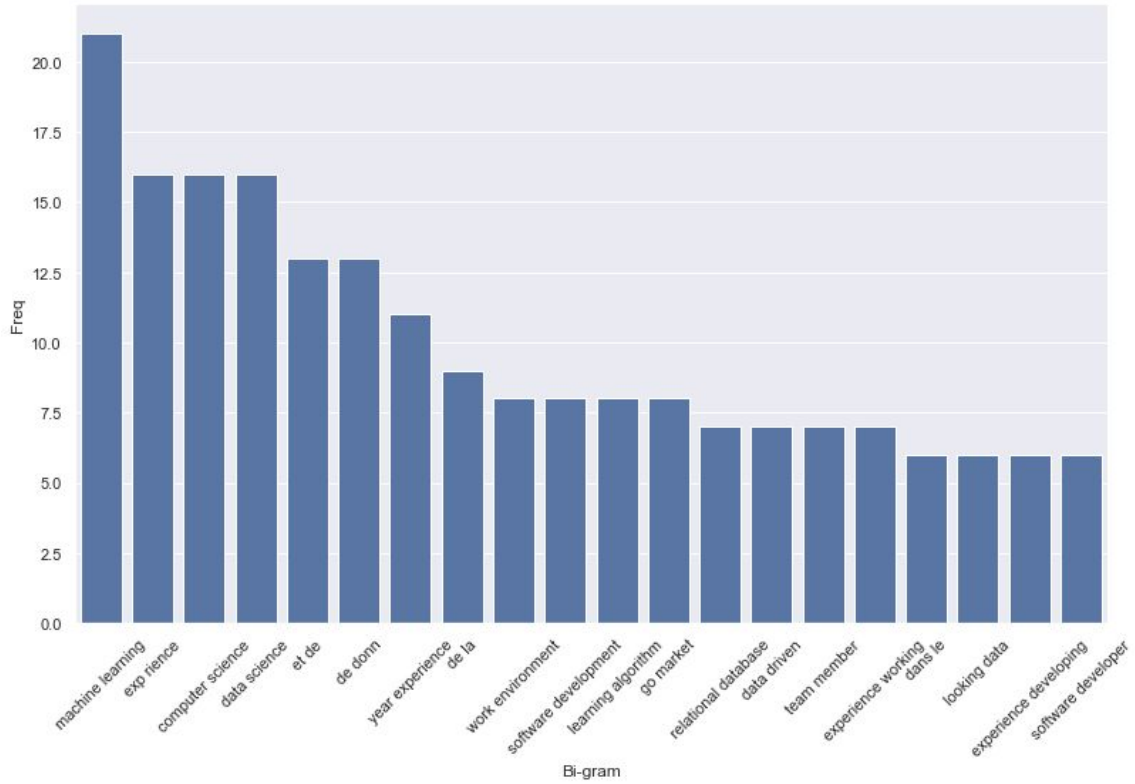

Fig 5. Generated bigrams for the job description corpus

We then used these weights to compute TFIDF to extract the top skills for a particular job post. Once we had the set of relevant tags which shows us the technologies the user is proficient and experienced in, along with the set of keywords from the job title and description, we ran the Jaccard similarity between these two sets to obtain a score of the top matching jobs suited to the user.

Finally, we found relevant jobs by computing the Jaccard similarity between tags and skills set to find the Jaccard score. Higher the Jaccard score, greater is the particular job posting suitable for the user. Below is a sample dataframe of the job postings along with the Jaccard similarity.

	jobTitle	jobLocation	companyName	employmentType	topSkills	jaccard
0	Android Developer	Montreal, CA	mindGeek	Full-Time	[android, java, communication, swift]	0.500
1	Senior Mobile Developer	Vancouver, BC	Fortinet	Full-Time	[ios, nodeJs, communication, Python]	0.125
2	Data Engineer	Vancouver, CA	Scribd	Full-Time	[python, spark, real-time, communication]	0.000
3	Data Analyst	Toronto, CA	PressReader	Internship	[excel, statistics, detail-oriented, python]	0.000

Fig 6. Different Jaccard similarity values for different job postings for a particular user

5. DATA PRODUCT

StreamLit is an open-source application development framework for creating data science applications using pure python. We decided to use this framework to build the application which will let the user access the entire suite of features.

HOMEPAGE

The user starts off at this page, where he/she can get some basic information about the application and how to go about using it.

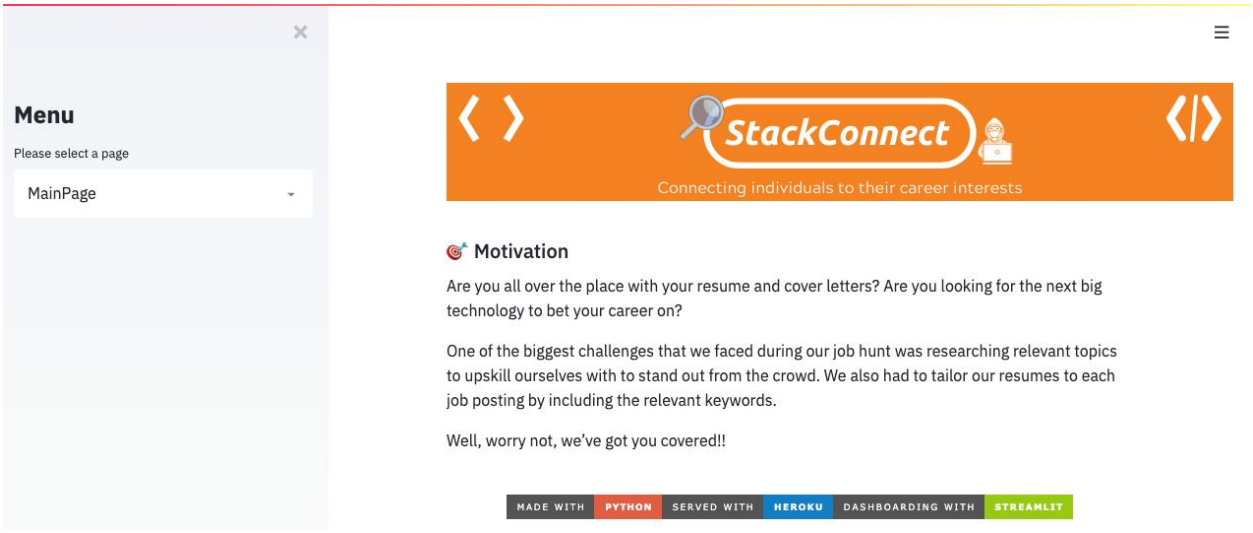

Fig 7. Screenshot of web application displaying the homepage

LOGIN

On this page, the user can log in to his/her account. For now, we authenticate the users based on the StackOverflow users table. We maintain a table in firebase which has all the user's information. If the user exists, then we add the user details to the active_users table on firebase, which lets the user access other features of the app. Access is provided to the other features only when the user has successfully logged in.

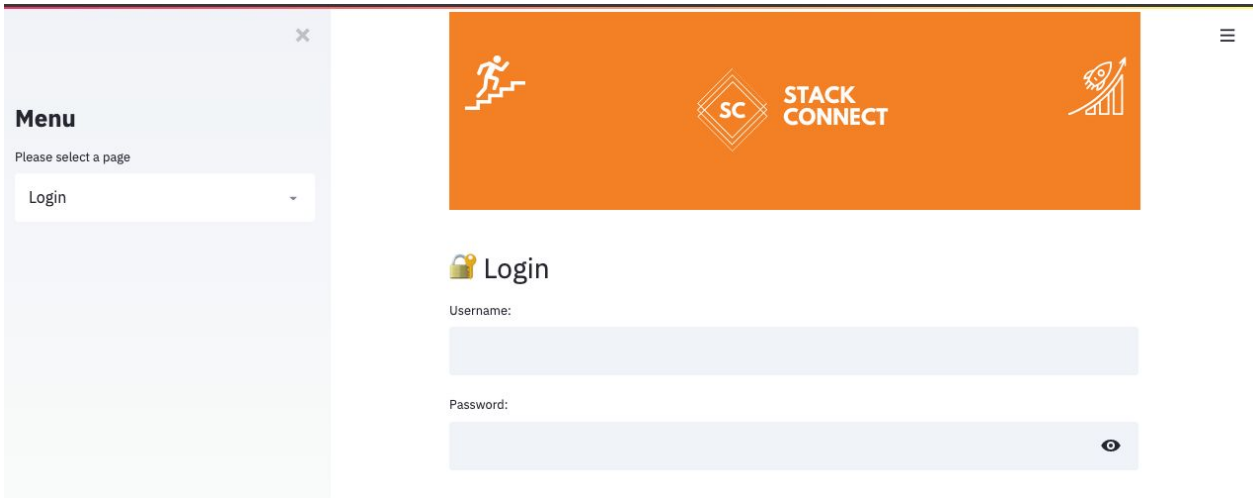

Fig 8. Screenshot of web application displaying the login page

PROFILE

The profile tab lets the user view his basic information, like the number of upvotes, downvotes, place of origin, etc.

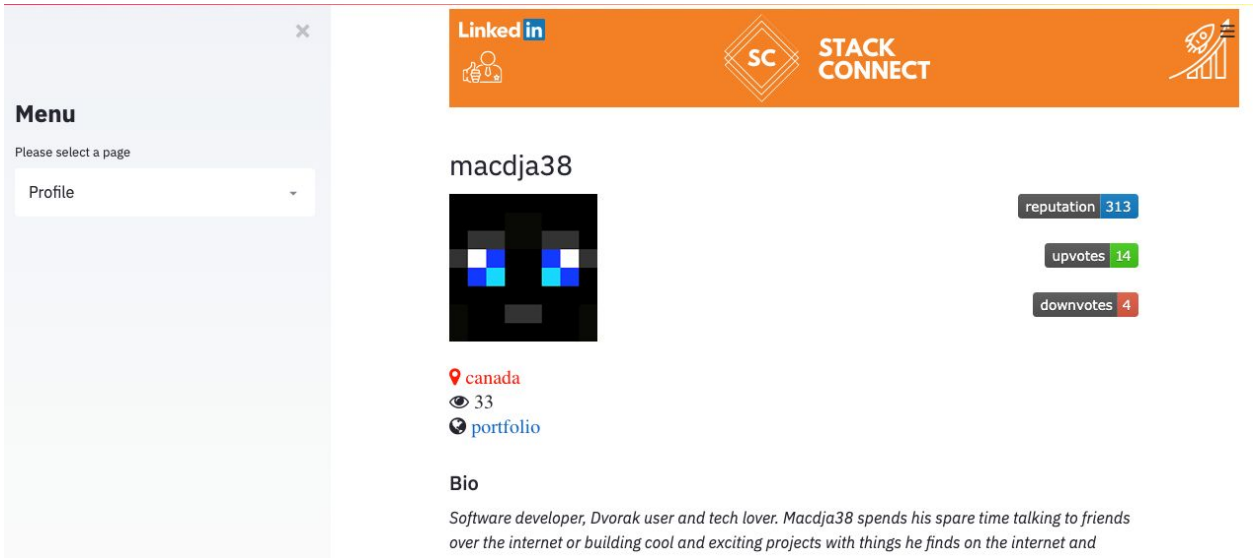

Fig 9. Screenshot of web application displaying the user profile information page

TAG PREDICTION

The user can enter the title and the question description as shown and our model will return the recommended tags. With the right tags, we can ensure that the user's question reaches the

right audience and helps them get a quick and better response.

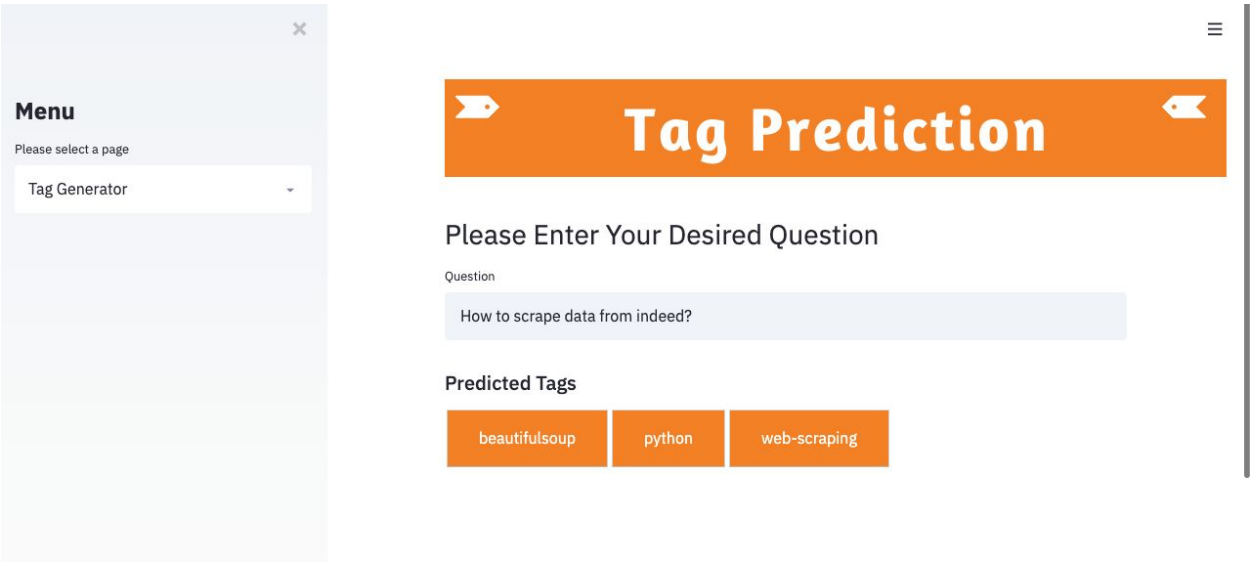

Fig 10. Screenshot of web application displaying the tag prediction module

JOB RECOMMENDATION

The user can access his job recommendations based on the questions answered by him on Stackoverflow. The user also gets reviews of the job collected from the glassdoor dataset along with salary insights.

	Job Title	Company	Location	Salary Range	Pros	Cons
0	Software Developer	Fortinet	Vancouver, BC	$91,432 - $121,334	Joining Fortinet was the best decision i ever took in my life	not very good work/life balance
1	Full Stack Developer	CITI	Mississauga, ON	$NA - $NA	Dynamic organization with great opportunities to learn and grow	Diversity in culture can sometime be an obstacle while working as a team

Fig 11. Screenshot of web application displaying the job recommendations

SEMANTIC SEARCH

The user can get similar articles from the StackOverflow based on the question that the user has posted using improved semantic search.

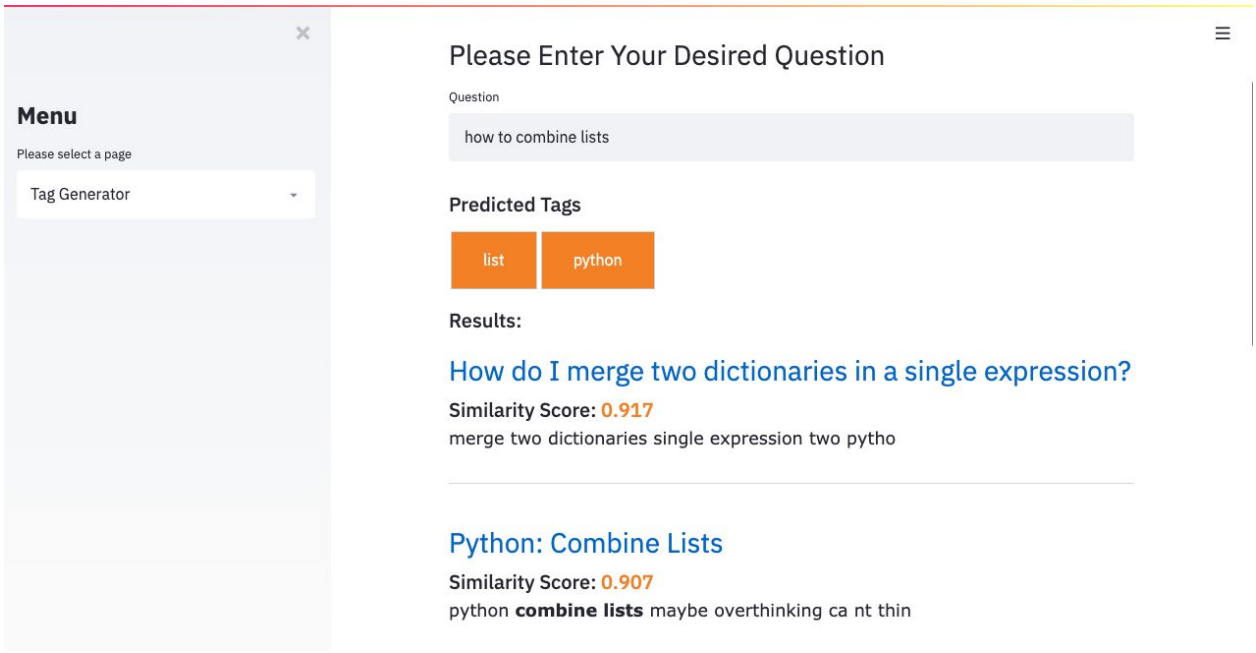

Fig 12. Screenshot of web application displaying the matching results with improved semantic search

INSIGHTS

WORDCLOUD

The word cloud helps users visualize the top technologies in any given year. In a word cloud, the popularity is denoted by the font size - Higher font size, higher the popularity of the word. This gives the user an idea about the technologies that have always been dominating the market over the years.

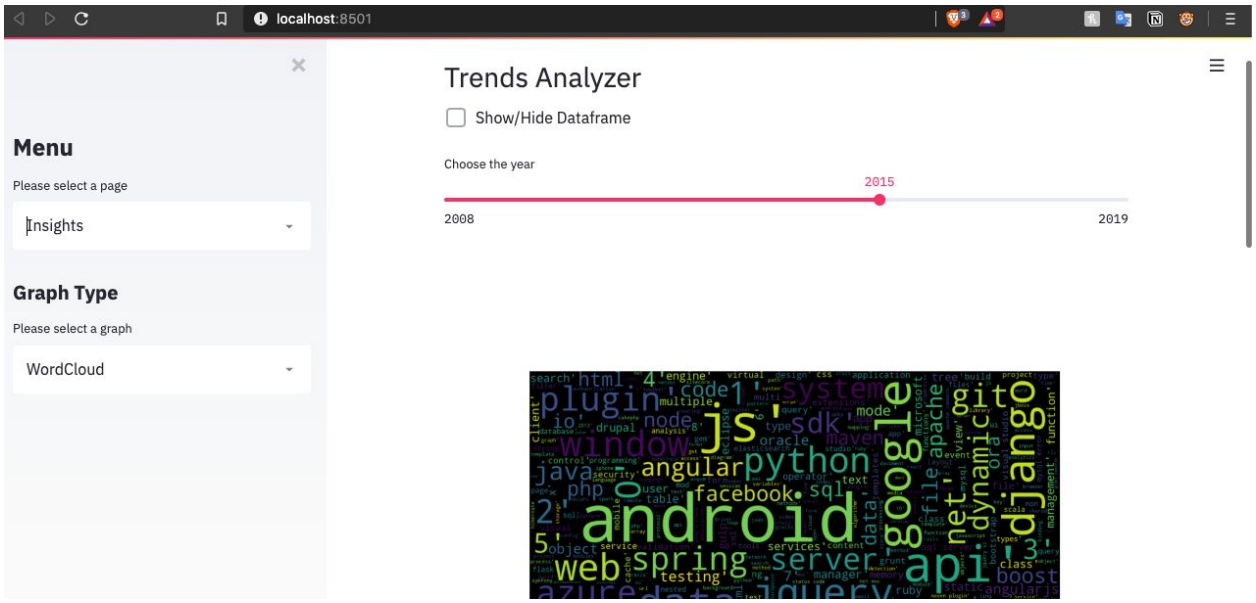

Fig 13. Screenshot of web application displaying the word cloud of the most commonly used tags

LINE CHART

The line chart helps the user visualize how the popularity of any given programming language/technology has changed over time. The user can choose the technologies to compare and visualize how those technologies have changed over time.

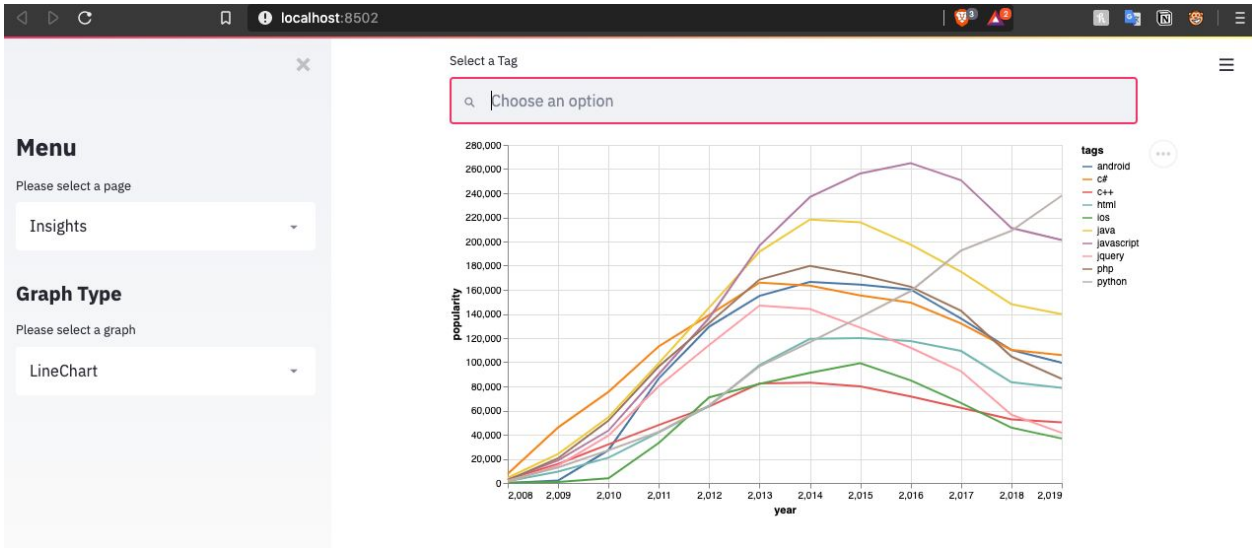

Fig 14. Screenshot of web application displaying the line chart of temporal trends in different technologies

MAPS

We present location-based information about the commonly used tags from people belonging to a particular region. This helps the end-user learn more about technologies around where he lives or know more about locations, to seek jobs based on his passion.

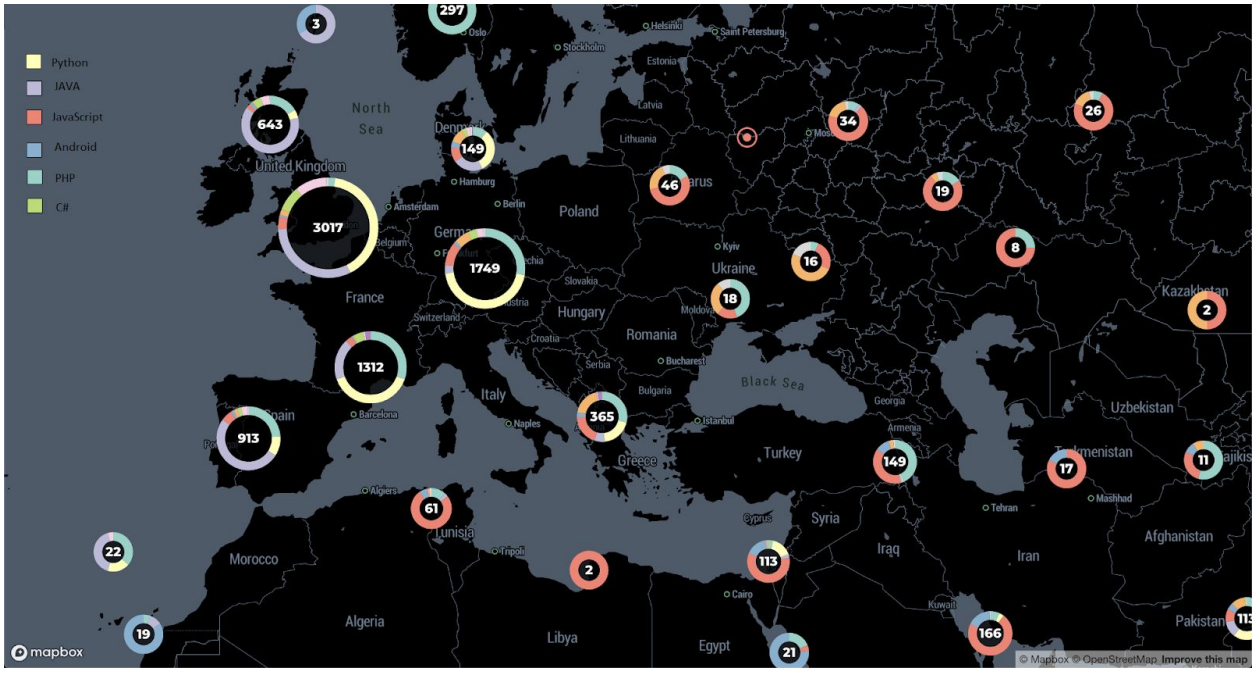

Fig 15. Screenshot of web application displaying the location based trends

6. LESSONS LEARNT

- Learned to crawl and scrape dynamic web pages in real-time using BeautifulSoup and Requests python libraries.
- Understanding the data requirements of the project and collecting the data required to create the recommendation system.
- We learned the importance of Exploratory Data Analysis in understanding the data.
- Explored the StreamLit framework to create an interactive data science web application.
- We learned that certain pre-trained models are highly specific to their application and do not generalize well across applications. We learned to train our own word embedding model.
- We also learned how to communicate our findings and project results to both technical as well as non-technical people through verbal, written and video communication.

7. SUMMARY

The main aim of our project was to provide career recommendations to the users based on their StackOverflow activity. We also committed to presenting temporal trends in technology along with future projections for technological trends. Finally, we aimed to perform a tag prediction based on the question asked by the user. The above-mentioned aims of the project were successfully accomplished by exploiting various Data Science tools and techniques and the recommendations and analyses were provided to the user. Additionally, we also implemented a semantic search technique that takes into account the popularity of the user, the sentiment of the answers and cosine similarity to improve search results.

8. REFERENCES

[1] https://www.kaggle.com/stackoverflow/stackoverflow

[2] https://www.kaggle.com/miljan/predicting-tags-for-stackoverflow

[3]https://www.freecodecamp.org/news/how-to-extract-keywords-from-text-with-tf-idf-and-pythons-scikit-learn-b2a0f3d7e667/

[4]https://towardsdatascience.com/improving-the-stack-overflow-search-algorithm-using-semantic-search-and-nlp-a23e20091d4c