Stackoverflow VS-Code Extension
Final Project Report

Annanya Pratap Singh Chauhan **Arpit Gupta** **Aryan Agrawal** **Shivang Dalal**
Group No: 11
{annan170101008, arpit170101012, aryan170101013, shiva170123047}@iitg.ac.in
Project Github Link

Abstract

The sheer number and complexity of the programming languages available today have ensured that almost every programmer is familiar with the Stack Overflow platform. It provides one of the largest Q & A platforms for programmers, covering fundamental concepts to niche errors for every programming language. Users post their queries, and their peers try to give resolutions, be it in-depth solutions or hacky workarounds. The more reliable an answer, the higher number of votes it gets.

Given its popularity, it is safe to say that there is a plethora of data on the platform. However, this enormous amount of information also makes searching for a precise answer a problematic task. Our aim through this project is to simplify this process by creating an NLP based search engine for searching queries on Stack Overflow. Furthermore, to reduce the hassle, we propose a Stack Overflow search engine extension for a frequently used IDE (VS Code). The extension will streamline the querying process and enable programmers to search queries while editing the code.

1 Introduction

The current searching method in Stack-Overflow uses a tag-based search algorithm; however, it is only used if we provide the tags beforehand; otherwise, it defaults to a simple text-based search. It does not pose an issue to veteran programmers and other experienced professionals since they know the appropriate keywords for a precise search. However, for a novice, this poses a significant concern. For instance, if he/she wishes to learn *'how to make a server'* using Python, it is quite unlikely that he/she would use the terms *'Django'* or *'Flask'* in the query. Therefore, an intelligent search engine is necessary to efficiently navigate this vast data set and provide accurate search results.

Our objective is to develop a platform that can understand the user's search query content and generate appropriate tags using natural language processing. An advanced search with the tags using the StackOverflow API can be done, and the most relevant results can be displayed directly in the IDE to simplify the querying process.

The Stack Overflow dataset consists of 17 million question-answer examples, thus choosing an optimal subset of the dataset for our analysis is essential due to processing limitations. Google Big-Query dataset (3) has an archive of Stack Overflow content, containing for each post, its corresponding tags, answers, and also the number of votes and badges. This is constantly updated to mirror the Stack Overflow dataset on the internet archive. We exploit the SQL query based extraction method of Google Big-Query Dataset to extract the required subset of the dataset. Our neural network-based search algorithm can be divided into three sub tasks:

- **Data Processing [2]:** Collecting a subset of the Stack Overflow dataset and processing it to generate the corpus for training the prediction models.
- **Training Word Embedding and Tag Predictor [3.1]:** Given the user's query, predict the most relevant tags which will help in generating better search results.
- **Generating search results: [3.3]** Based on a user query and the generated tags, return an existing question that closely approximates the user's query.

The training model has been written in Python and the VS-Code Search Engine extension in JavaScript. Therefore, for running the prediction model on a query, a server has been hosted using

Flask. For each query, the extension sends the request to the server, which returns the question ids of the most similar questions. These ids are then used to fetch the question's body and answers, which are displayed in *HTML* format by the Search Engine extension.

In the following section, we shall describe in detail the dataset used along with the preprocessing done. The Neural architecture designed will be explained in the section after that. Next, the IDE extension's development process will be outlined along with its integration with the search model. We shall then qualitatively analyze our results and finally conclude our report along with the future scope of the project.

2 Data Collection and Pre-Processing

We start by collecting data from Stack-Overflow Question-Answer Dataset, which is publically available as a Google Big Query Dataset. The dataset is queried to create data points of the format {*Question Title, Question Body, Answer, Number of Upvotes, Tags*}. Each data point in our corpus corresponds to a unique question-answer pair. As a proof of concept and to reduce the computational complexity, we have currently only considered data points related to Python programming language. Along the same lines, we have also limited the number of data points to 500000 to save on the model training time. The considered dataset can be extended to include multiple programming topics in the future, given enough computation power, without significant changes to the search model.

The created dataset can have multiple entries corresponding to the same question since multiple answers are allowed for each question. Hence, in the next step, we grouped data points according to the question, merged their answers, and summed their upvotes. Following this, we pre-processed the text by removing *punctuations*, *HTML tags*, *inline code*, *STOP words*, and converting all letters to lowercase. We also normalized the number of upvotes.

The final dataset is stored in CSV format and with this dataset generated, we now move onto the neural search model that will utilize this dataset.

3 Search Mechanism

This section focuses on developing and training an NLP model using the above dataset to generate tags and perform stack overflow searches from a user's query. The following are the significant steps taken for achieving the same:

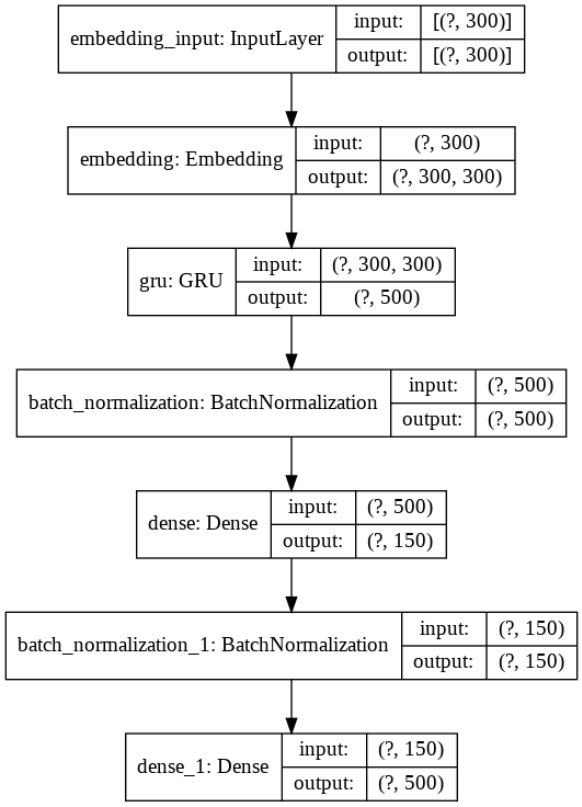

Figure 1: Architecture of tag prediction model

3.1 Word Embedding Model

When a user enters a search query, we need to convert the textual information into a meaningful vector representation. We have created a word embedding model to convert the input words into vector embeddings. Using self-trained vector embeddings allows us to have fine-grained control over every aspect of our search model, which using a standard word embedding model like *GLoVe* (7), will not provide.

The word embeddings are created by training a word2vec model using gensim (2) on a dataset, which is a concatenation of *Question Title, Question Body*, and *Answers*. We used a window size of 7 and an output dimension of 300 for our word vectors. The word embedding model converts the input word into vectors, which are then fed into our tag predictor model.

3.2 Tag Predictor Model

The next step is to pump the vectorized user query to the tag predictor model. This model predicts the closest Stackoverflow tags for a given search query.

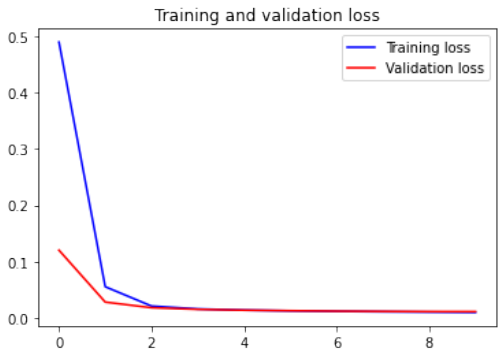

Figure 2: Training Tag Predictor Model

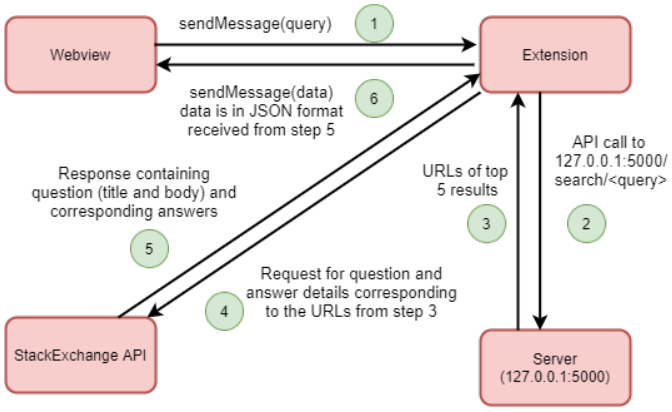

Figure 3: Flow chart of Extension

We constructed a neural network tag predictor with the structure, as shown in figure 1. We trained the tag prediction model on the Question Titles and took the tags associated with it as the ground truth labels.

3.2.1 Model Comparison

For building the tag-predictor model, we considered two possibilities of the RNN(Recurrent Neural Network) Architecture, namely LSTM (Long Short-Term Memory) and GRU (Gated Recurrent Unit). The LSTM based model gave an average cosine similarity of 0.604 on the test dataset, while the GRU model gave an average cosine similarity of 0.778. Since we were getting better results on the GRU architecture, we chose GRU as our RNN layer of the Tag Predictor model.

3.2.2 Testing and Validation

We have trained both LSTM and GRU models for ten epochs, each with a batch size of 1024. Processed data is split into 80% training data and 20% testing data. The training data was further split into 10% validation data and 90% training. The training and validation loss for the GRU based tag predictor model is illustrated in Figure 2. The overall training was done on the Google Colab GPU with a training time of about 20 minutes for each model.

Thus, the tag predictor and word embedding model allow us to take as input a user question string and generate appropriate tags, which help us querying the Stack overflow database.

3.3 Generating search results

The search model uses the word embedding model to generate the centroid (8) of our query and calculate its cosine similarity with the centroid of all the question titles present in our database. It then generates the tags utilizing the tag predictor and calculates its cosine similarity with all the question tags present in the database. We then add these cosine similarity metrics with the question score to generate the final indexing parameter. Question score symbolizes the normalized number of upvotes. Finally, we search the Stack overflow dataset based on the indexing parameter and rank the search results. The IDE extension uses this ranked list and the corresponding data to display the search results to the user.

3.4 Implementaion Details

The Searching algorithm is written in python. For the tokenization, we have used the tokenizer model of the open-source library, *spaCy* (5). We have used *'stopword'* corpus of *nltk* (4) for removing the stopwords from the tokenized words list. And we utilised a library called $Beautiful Soup$ (1) to remove the code fragments and HTML tags from the question's body. To generate the word embedding, *word to vec* model of *gensim* (2) library has been used. For the tag-predictor model development and training we have used *Tensorflow* and *Keras* libraries.

4 Integration with an IDE extension

After discussing the searching model in the previous section, we now outline how the IDE extension has been created. This section also discusses how the neural model is integrated with the extension since the neural model has been created using Python, whereas the VS Code extension has to be made using JavaScript.

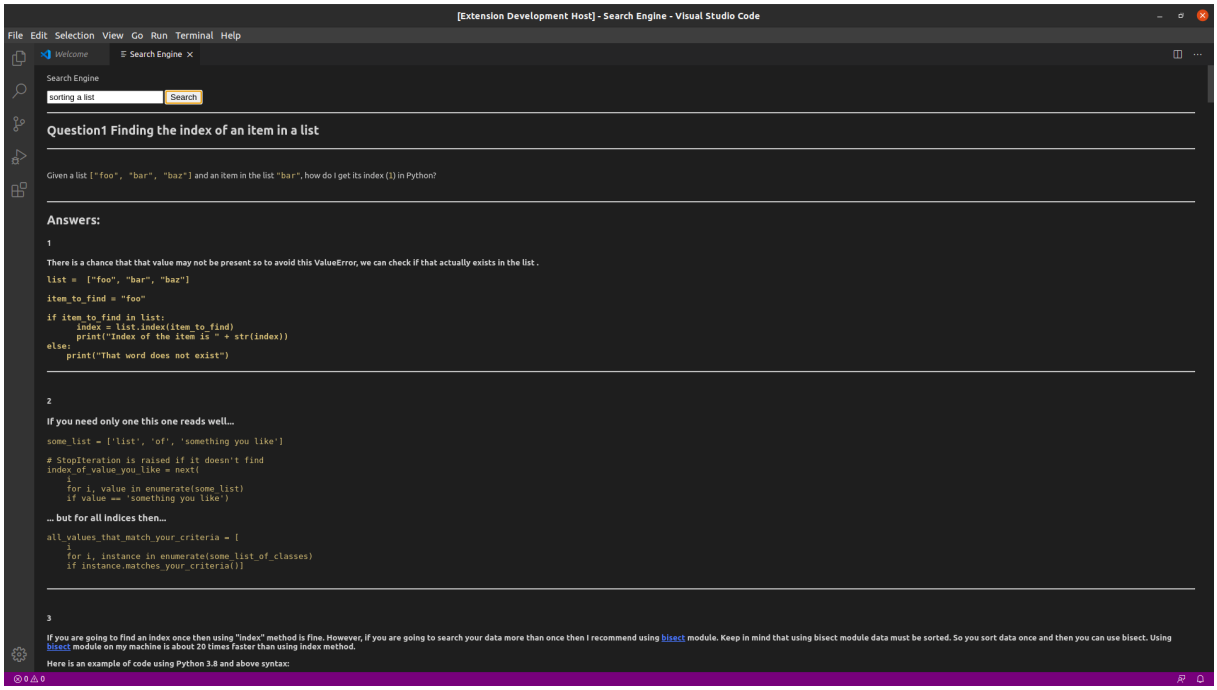

Figure 4: VS Code extension with the query ”sorting a list”

4.1 Developing a VS Code extension

VS-code extension works on the runtime environment of node.js. The Extension package has three main files, namely *package.json* which contains the information about the modules used in the extension, assisting in an easy installation of the dependencies on any system. *extension.js* contains the main executable code for the extension. It contains an activation function that is the first function to be run, each time the extension is started; it contains a register command which binds the main function to be run for search query engine. The third component of the extension package is *app.html* file, which displays the output in the required *HTML* format.

4.2 Integration and Deployment

Since the VS-Code extension does not support a python environment, we have hosted a flask server that runs the prediction model. The server has two API routes. The *127.0.0.1:5000/* route loads the word embedding model, tag predictor model, and title embeddings used to generate search results. After loading the models, the extension makes API calls to *127.0.0.1:5000/search/<query>* route to get the most similar results corresponding to the query. The result obtained from the server contains only the URL of questions. To get the complete information about the question like its title, body and all corresponding answers, the extension makes API requests to Stack OverFlow API(6). The returned responses are then processed and passed to webview for display. The complete integration and deployment cycle is explained in figure 3.

5 Results

This section shall describe the results obtained from different components of the search model.

Word embeddings: A few examples for most similar words corresponding to the given word are(in decreasing order of similarity):

- *dataframe*: column, dataframes, pandas
- *matplotlib*: pyplot, pylabs, imshow
- *list*: lists, tuples, dictionary, sublist
- *scipy*: curvefit, linalg, statsmodel

Tag Prediction:The neural net for tag prediction obtained a loss of 0.0104% and a validation loss of 0.0118%. Some examples of predicted tags for queries and ground truth tags can be found in table 1.

Search Results: As an example of our final prediction using cosine similarity, we take a query *”sorting a list”* and feed it to our model. The following are the results as ranked by our model:

1. Finding the index of an item in a list Link

Query	Predicted tags	Ground truth
file directory test1001jpg	'file', 'path', 'python', 'python-2.7', 'python-3.x'	'pandas', 'python'
convert mysql query date range django query	'django', 'mysql', 'postgresql', 'python', 'sql'	'django', 'mysql', 'python'
setting django virtualenv windows7	'apache', 'django', 'python', 'python-2.7', 'virtualenv'	'django', 'path', 'python', 'virtualenv'
remove first 3 characters field database	'database', 'django', 'python', 'python-2.7', 'python-3.x'	'django', 'python'

Table 1: Predicted and ground truth tags for various queries

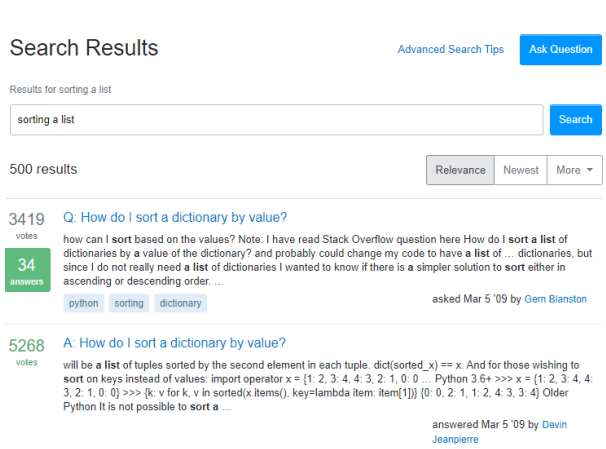

Figure 5: Stackoverflow results for query "sorting a list"

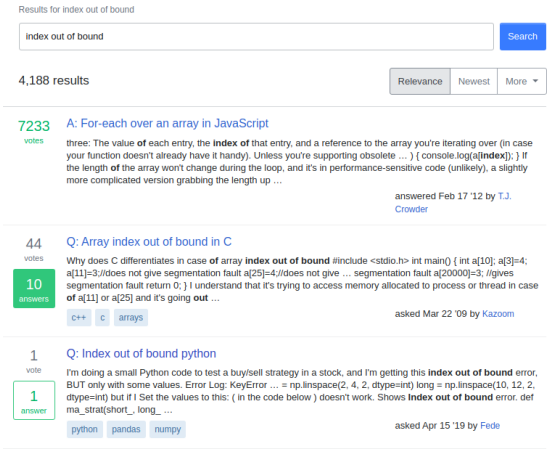

Figure 6: Stackoverflow results for query "Index out of bound"

2. List sorting with bubble sort (list of list) Link
3. Sorting a list into another list Link
4. Python List Sorting Link
5. Sorting a 2D list alphabetically? Link

If we compare these queries with the most relevant questions that Stack Overflow displays (Figure: 5) for the same query, we can qualitatively say that our model performs better.

On querying *"Index out of bound"* we obtain the following results:

1. List index out of bound Link
2. Finding the index of an item in a list Link
3. IndexError: index out of bound (Pandas) Link
4. Error with index. index 0 is out of bounds for axis 0 with size 0 Link
5. Index of numpy.ndarray Link

The results we obtained are similar to what Stackoverflow displays for the same query (Figure: 6)

6 Conclusion

Through this project, we have developed a Visual Studio Code extension that streamlines the process of searching queries on Stack Overflow. We firmly believe that this tool provides utility to all programmers, be it a veteran or a beginner. Being programmers ourselves, we hope to incorporate this extension into our programming habits and also provide this tool to our peers.

The NLP based Search mechanism developed gives qualitatively better results than standard Stack Overflow search. Multiple examples have been provided to illustrate this. We trained and compared the results of multiple models and finally chose GRU as the ideal neural model. Although the extension uses a pre-trained model to search, the search results always return the relevant results using the Stack Overflow API.

7 Future Scope

The latency and processing time of the extension can be optimized in the future. A possible alternative to achieve this is to hierarchically compare

the target word's cosine similarity with the word embeddings database. Doing so will reduce the number of comparisons needed to match the target word. Another possible optimization would be to search only in the stored dataset for the final query results rather than calling the Stack Overflow API. However, this makes the search outdated compared to the website and has thus been discarded by us.

The model has been trained only for the questions related to Python programming language; this can be extended to make the extension agnostic to the programming language. Lastly, we can extend the project to search using code snippets as well. More precisely, take code snippets from the editor and search for posts on Stack Overflow containing similar code snippets.

References

[1] Beautifulsoup. *https://pypi.org/project/beautifulsoup4*.

[2] Gensim: Unsupervised topic modeling and natural language processing. *https://radimrehurek.com/gensim/models/word2vec.html*.

[3] Kaggle stack overflow big query dataset. *https://www.kaggle.com/stackoverflow/stackoverflow*.

[4] Natural language toolkit. *https://www.nltk.org/*.

[5] Spacy natural language processing model. *https://spacy.io/api/tokenizer*.

[6] Stackexchange api. *https://api.stackexchange.com/*.

[7] JEFFREY PENNINGTON, RICHARD SOCHER, C. D. M. Glove: Global vectors for word representation. *EMNLP* (2014).

[8] JURAFSKY, D., AND MARTIN, J. H. Speech and language processing. *https://web.stanford.edu/ jurafsky/slp3/6.pdf*.