Stackoverflow VS-Code extension

Annanya Pratap Singh Chauhan **Arpit Gupta** **Aryan Agrawal** **Shivang Dalal**
Indian Institute of Technology Guwahati
Group No: 11
{annan170101008, arpit170101012, aryan170101013, shiva170123047}@iitg.ac.in

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 solutions, be it in depth solutions or hacky workarounds. The more legitimate 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 for the platform to understand the content of the user's search query, generate appropriate tags, and then return the most relevant result.

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 (1) has an archive of Stack Overflow content, containing for each post, its corresponding tags and 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 current iteration of the search model can be broken down into three sub-tasks:

- **Data Processing [2.1]:** 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 [2.2]:** Given the user query, predict the most relevant tags which will help in generating better search results.
- **Generating search results: [2.3]** Based on a user query and the generated tags, return an existing question that closely approximates the user's query.

Details of each sub-task are explained in the section [2] . Our current version of the trained search engine model inputs a search query and outputs the top matching questions.

For the final version, we wish to optimize the Search engine further and focus on developing the VSCode extension (2). The extension will require

a considerable effort since we have to integrate the python trained model into the extension as well as have an intuitive and simple UI.

2 Method

This report focuses on developing and training an NLP model for generating tags and performing stack overflow searches from a user's query. The following are the major steps for achieving the same:

2.1 Data Collection and Data 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 then 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 computing 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 dataset is stored in CSV format.

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.

2.2 Training Word Embeddings and Tag Predictor

Following data processing, the next step is to create a neural network model to predict tags based on a user's search query. To train the model, we had to represent the text in the form of vectors (word embeddings). Hence we trained a word embedding model to represent words in the form of vectors. This allows us to have fine-grained control over every aspect of our model, which using a pre-trained word embedding model, will not provide. We are hopeful that will translate into objectively better results as well.

The word embeddings are trained on a dataset,

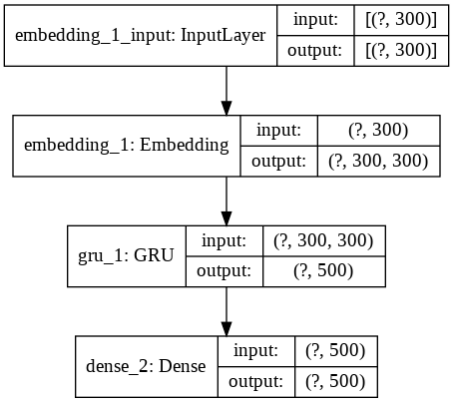

Figure 1: Tag predictor

which is a concatenation of Question Title, Question Body, and Answers. We used a window size of 7 and an output vector dimension of 300. Only the top 500 most popular tags are used for tag prediction since multiple tags are either misspellings or one-off tags, which will only add noise to our model. After training the word embedding model, we constructed a neural network tag predictor with the structure, as shown in the figure [1]. We trained the tag prediction model on the Question Titles and took the tags associated with it as the ground truth labels.

2.3 Generating search results

We used the word embedding model to generate the centroid (4) of our query and calculated its cosine similarity with the centroid of all the question titles present in our database. We then generated the tags utilizing the tag predictor and calculated its cosine similarity with all the question tags present in the database. Finally, we searched based on these generated metrics and ranked our search results.

3 Progress

We shall now describe the progress made in the various aspects of the project so far and the contributions made by the group members.

Training and Preprocessing: Data preprocessing involved combining answers and scores for questions, normalizing the text, and creating the corpus on which word embeddings had to be created. We created the word embeddings by training a word2vec model using gensim (3) on the given data. We also successfully trained a neural net to predict tags given a query.

Testing and Validating: We settled on using cosine similarity to find the dataset questions that

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

resemble the most to this query. The model is currently trained as a standalone application and will be integrated into an extension later. Currently, we are working on developing simple prototypes for familiarizing ourselves with the IDE extension development process.

Contributions: We regard everyone to have an equal contribution to this project. However, since requested, we broadly divided the work into pairs. Annanya and Aryan explored the various data sets available and worked on data processing and collection. Arpit and Aryan worked on training word embedding. Arpit and Shivang created the neural network structure and trained the model. Annanya and Shivang created the Search module, which uses tag prediction and document centroid measures to generate search results. Everyone contributed towards writing their part in the report.

4 Results

The results we have achieved so far, albeit incomplete, indicate the promising results we hope to achieve through this project.

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 0.0113% and a validation loss of 0.0114%. We aim to improve the model in further iterations by hyper-parameter tuning and experimenting with extra layers. Some examples of predicted tags for queries and ground truth tags can be found in table 1.

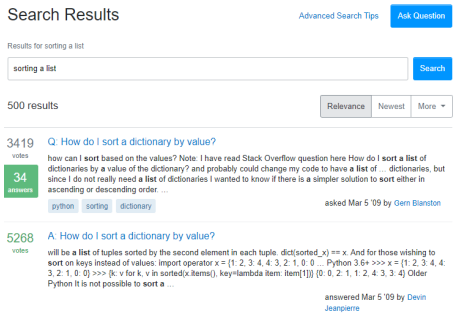

Figure 2: Stackoverflow results for query "sorting a list"

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

1. sorting a list Link
2. Sort 4 lists according sorting first list 4 Link
3. Appending and Sorting list working code trying Link
4. List sorting with bubble sort (list of list) Link
5. sorting a 2d list based on the number of elements in each list Link

If we compare these queries with the most relevant questions that Stack Overflow displays (Figure: 2) for the same query, we can say that our model performs better.

5 Conclusion

Since we have created a workable model, our focus will be on developing a VS-code extension, which will use the pre-trained models to give relevant results. We shall also work on further optimizing our model to achieve better overall results. Since the extension has to be written in javascript, we will look at TensorFlow-js for further work.

References

[1] Kaggle Stack Overflow Data (BigQuery Dataset) *https://www.kaggle.com/stackoverflow/stackoverflow*.

[2] IsaacSomething/stackoverflow-view-vscode *https://github.com/IsaacSomething/stackoverflow-view-vscode*

[3] Speech and Language Processing. Daniel Jurafsky & James H. Martin *https://web.stanford.edu/ jurafsky/slp3/6.pdf*

[4] Gensim: *https://radimrehurek.com/gensim/models/word2vec.html*