</> Code Search

Answering Python Questions with Source Code

Erica Chen

Winnie Lee

Yu-Cheng Lin

Advisor: David Bamman

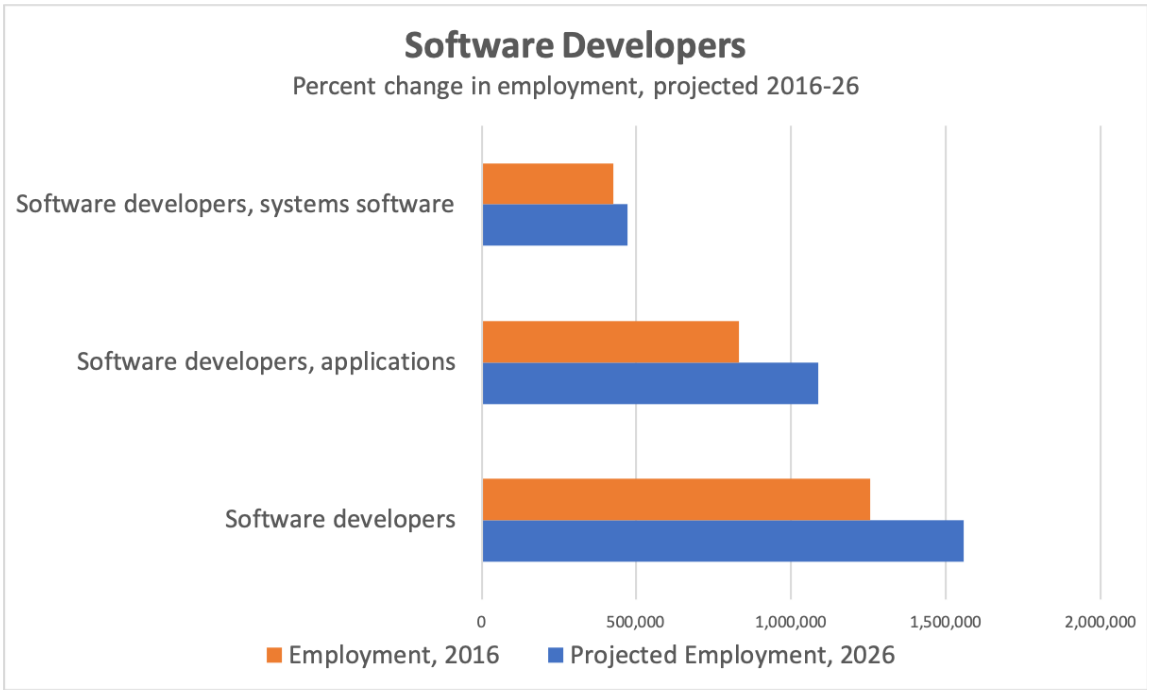
MIMS 19 Capstone Project

Berkeley SCHOOL OF INFORMATION

Table_of_Contents

The_Problem

In the past few decades, the needs for software development have been dramatically increasing.



(Data Source: United State Department of Labor)

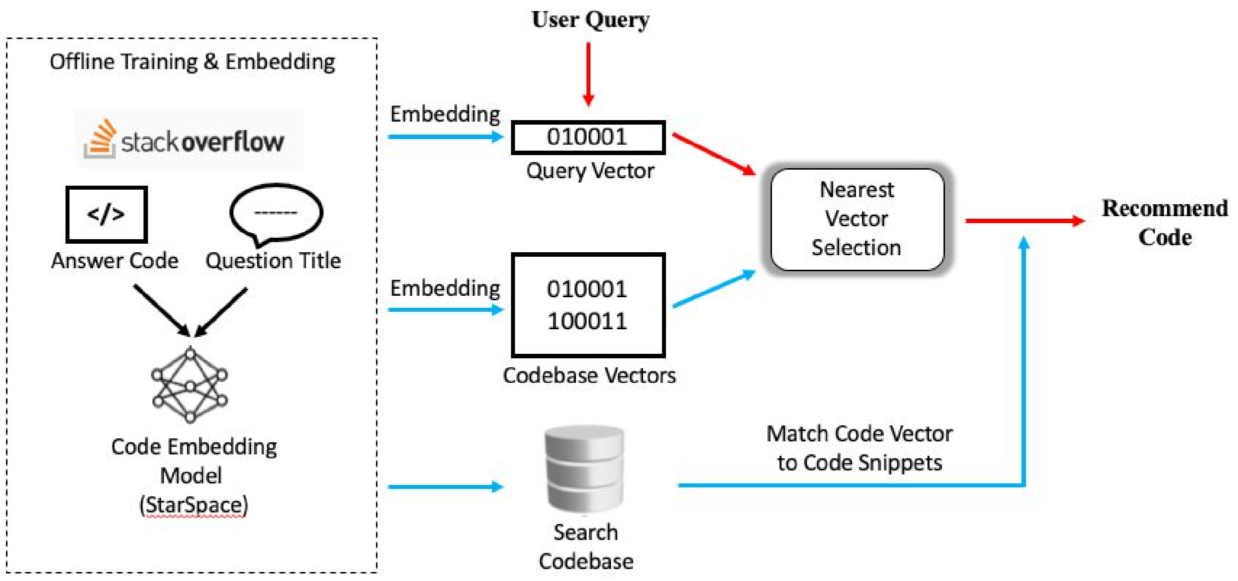
As writing code from scratch is time-consuming and easy to make mistakes, developers usually reuse existing solutions from large-scale codebases to implement a program efficiently. Many code search approaches have been proposed to help developers. However, they are mostly based on modeling source code in Java.

Python is a fast growing programming language especially in the area of machine learning and data analysis. In north America, the usage of Python even surpass Java on GitHub. Thus, we hope to contribute to a powerful productivity Python code search tool for developers. The tool would help them quickly find examples of code snippets related to their intent expressed with natural language queries.

Our_Solution

A search engine that answers users' Python questions with 10 code snippets. We compute potential answers with our natural language model on the cloud in real time.

Below is our overall process from offline code embedding training to the real-time search engine.



1. Data Collection

To select code search queries for training, testing and evaluation, we extract queries and correspond code snippet from Python programming questions in Stack Overflow. The tool we use here are StackExchange Data Explorer and Pandas to filter the data.

The StackOverflow questions should satisfy the following conditions:

- Data Between **2015-01-01 to 2019-04-21**
- The question is a Python programming task, including any versions of Python
- The question score to that question is **higher than or equal to 0**
- The question does have a best answer
- The best answer to that question has **exactly one def function** answer
- The question is **not** a 'TypeError' or a 'why' question, but a 'how to' question

We extract 50,000 data at first, and after data cleaning and filtering with above conditions, we select 40 data for testing, with 5037 data for training the word-code embedding and validation.

2. Word Embedding Model

StarSpace is a neural embedding model that could be used on learning entities embeddings such as word, sentence and document which are described by a bag of words or n-grams. The speciality is that StarSpace model could compare entities of different kinds based on measuring similarity between the entities. We applied this model in our experiment because of this advantage to deal with different kinds of entity embedding query in natural language and code snippets.

In our model development, we apply StarSpace supervised learning mode for training code-word embedding. StarSpace takes input files of the following format. Each line will be one input example, in the simplest case the input has k words, and each labels 1 to r is a single word (There is no need no have the same length of words and labels):

```
word_1 word_2 ... word_k <tab> label_1 label_2 ... label_r
```

In our training process, our training data should look like this:

```
how to write a function for numpy matrix in python    def f ( i , j , A ) : return A [ i , j ] * ( 1 . / np.sum ( A [ i , ] ) ) - ( 1 . / np.sum ( A [ : ,j ] ) )
```

We split the data to 4062 for training, 508 for validation and 508 for testing. After tuning the hyperparameters in 10 rounds, we got a final word-code embedding with 300 dimensions and 5409 vocab size. The parameters are listed below:

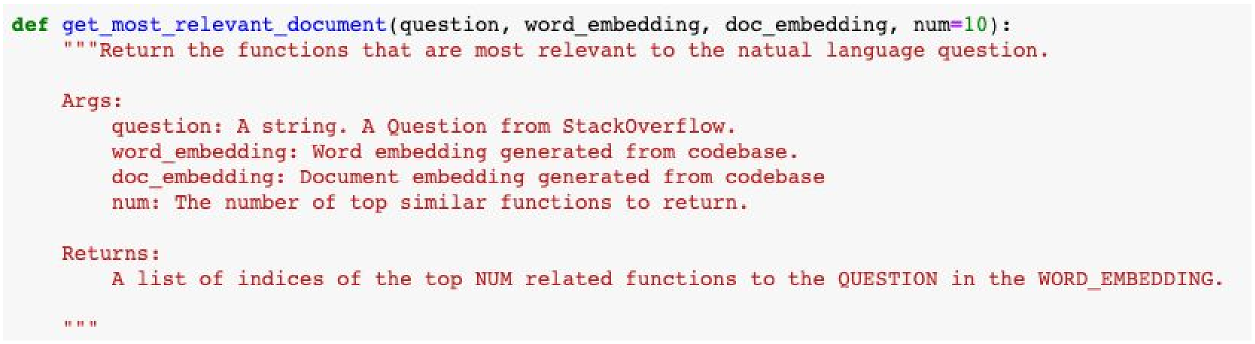
```
 -minCount        minimal number of word occurrences [2]

 -minCountLabel   minimal number of label occurrences [2]

 -lr              learning rate [0.05]

 -dim             size of embedding vectors [300]

 -epoch           number of epochs [25]

 -negSearchLimit  number of negatives sampled [50]
```

3. Build Document Representation

We then convert the code snippets in our base to document representations (code vectors) based on the embedding. First, we will split the code snippets to word level and check if this word exist in our word embedding. If so, we would convert it to the vector representation. Finally, we sum up all the vectors and calculate the average as the document representation.

In addition, for the user query in real time calculation, we will use the same method to convert it to query vector.

4. Get Most Relevant Document



```
def get_most_relevant_document(question, word_embedding, doc_embedding, num=10):
    """Return the functions that are most relevant to the natual language question.

    Args:
        question: A string. A Question from StackOverflow.
        word_embedding: Word embedding generated from codebase.
        doc_embedding: Document embedding generated from codebase
        num: The number of top similar functions to return.

    Returns:
        A list of indices of the top NUM related functions to the QUESTION in the WORD_EMBEDDING.

    """
```

Prototype_Development

1. Web application architecture
    a. Python flask

        Main web framework of the web application. Flask interacts with frontend users by caching users' queries, passing the inputs to AWS for generating matching functions and invoking the results, rendering the colorful pages with neat layouts.

    b. Bootstrap and Prism

        Frameworks that make the pages responsive and the search results, which are Python codes, printed with different colors as developers' IDEs do.

2. Amazon Web Services
    a. EC2

        The main environment that we used to train our model, evaluate the results, and all other high-volumed computation.

    b. EBS

        Where we created volumes / storage linked to EC2 for our large amount of data and output.

    c. Elastic Beanstalk

        The service for us deploying our web application, interacting with EC2 and S3 storage.

    d. S3

        Main storage of our embedding values for generating recommended results.

    e. Lambda

        Event-driven, serverless computing service for our application, listening the event from Flask and returning the suggested functions as results for users.

Next Steps

1. Enhance Code Search Quality
    a. Add more data source

        Currently, our codebase only includes the python code from StackOverFlow. We may extract more python code data and natural language comments from Github.

    b. Improve search ranking

        Users usually desire see their expected answer rank as top as possible. We may try to improve search result ranking by applying learning to rank methods and training from manually label data.

    c. Incorporating with more users' labeling

        We may try to get user labeling by giving the score option along with each search result. By collecting this kind of feedback, we could boost the model accuracy and generate a better search result.

2. Collecting user feedback

    We will keep the web application deployed as the AWS Educate provides free tier usage. Regarding the feedback, we will try to know how users think of the results, the expected or better answers (given some of the users know what exact functions work for their queries). Not limited to the search results, we will improve the app when advices are given including in, for example, user interface.

Acknowledgement

This project was supported by School of Information, UC Berkeley. We extremely thank our advisor David Bamman who provided insight and expertise that greatly assisted this project.

We are grateful to William Yang, hackMD CEO and MIMS alumni for inspiring us to take on this challenge and for being very resourceful in sending useful material our way.

Finally, we’d like to thank all the members of the ISchool community for all their support in usability testing.