**W266 NLP with Deep Learning Final Project**

**Andrew Mamroth, Cathy Zhou, Yubo Zhang**

**Semantic Search for Programming Language - Python Edition**

## Motivation

In the past few decades, the growing availability of structured information on the Internet enables new emerging opportunities for better and more precise information access. Semantically oriented search, or semantic search, has gained considerable interest.

Traditionally, strict words search has dominated the searching industry. Even if it does exert significant power in bringing together knowledge and information at incredible speed, it still has its obvious flaws: strict words only. In scenario where semantic meaning is needed, strict search does not work, or not work as well.

For this final project, we attempt to create a model that is able to locate python functions based on their semantic meaning for the purpose of being used as tool for searching and querying reference .

## Introduction

For the final project, we are aiming at creating a flexible search function based on word meaning as opposed to exact keyword matches. By using word embeddings instead of simply using keywords, we will work on matching similar meaning words and thereby better capture the meaning of the search phrase.

One example of such search function is shown as follows:

Search term: “How to create a python function to merge two sorted arrays”

Search results:

###### a function to merge arrays ######

```
def merge(self, nums1, m, nums2, n):
    while m > 0 and n > 0:
        if nums1[m-1] >= nums2[n-1]:
            nums1[m+n-1] = nums1[m-1]
            m -= 1
        else:
```

```
            nums1[m+n-1] = nums2[n-1]
            n -= 1
    if n > 0:
        nums1[:n] = nums2[:n]
```

In this scenario we searched for “How to merge two sorted arrays”, and expect to return a python function which does exactly the same.

The dataset we will be using is from the open source platform GitHub. There are millions of open source projects/repositories being hosted on there which would be the perfect source of this project.

The method we will be using for this project is Word2Vec and GloVe. There has been previous work on this subject by many scholars and practitioners, for example: *GloVe: Global Vectors for Word Representation* from Jeffrey Pennington, Richard Socher, and Christopher D. Manning.

## Methodology

Data source:

The first thing we will want to do is to gather python code. There is an open dataset that Google hosts on BigQuery that has code from open source projects on Github. We were using bigquery to get the python files as a tabular dataset by executing SQL query in the bigquery console (Here is a link to the SQL Query)i. The raw data contains approximate 1.2 million distinct python code files.

The logic of query source data is as follows:

Query source: [bigquery-public-data.github_repos.files] and [bigquery-public-data.github_repos.contents]

Join logic:

1. All query content should be dated in or after year of 2017
2. All query content should end with ‘.py’ and contain syntax ‘def’ in order to filter for only python language repositories which contains function
3. All query should come from repositories which have been watched at least twice
4. All query content larger than 15,000 in size will be excluded

For the final dataset we collected 172,413 records of python file.

After gathering the data, we use two different models to get word embedding vectors. In both cases, we will use 100 dimensional vectors to improve processing speed without significant loss of search power. An important part of search is speed.

The overall methodology is essentially the same between the two models, the difference being how the word embeddings are generated. For the first model, we use existing pre-trained GloVe embeddings. For the second, we utilize the ast library in python to parse and extract the docstrings from the documents. We then tokenize the docstrings with the keras preprocessing library to build our vocabulary. From here we utilize the word2vec methodology outlined in the skip-gram model to generate our 100 dimensional word vectors.

Once the word embeddings are generated, we then generate an overall document embedding score for each document in the dataset. This is done by generating tf-idf value for each word in the vocabulary, then taking the average of each documents word embeddings weight by the tf-idf scores. This is done to weight out unimportant words such as “the” or “and” and to focus on words that have more relevance to what the code actually does. We then add the document embedding score to a lookup table associated with the code itself for ranking and search.

The search term provided by the user is processed almost the same as the document embedding score with a minor caveat that is, the tf-idf scores used for the search term will be the same as those used in the docstring vocabulary.

Once the search embedding score is generated, it’s distance is computed for each of the document embedding score and the values are ordered by short distance and documents for the top n shortest distances are returned.

The metric used for measuring the relevance of the returned code is precision at 10, where we count the number of relevant documents returned in the top 10 results. Since semantic meaning is a fuzzy metric, there is some amount of interpretation as to what constitutes a “relevant” document in terms of valuable returns and since we have to read each document individually to check for relevance, we chose to use only 10 for the sake of time. We use 6 different queries of varying complexity.

The complexity of the searches were varied due to the nature of documents in the corpus. Many of the python programs in the dataset were meant for a very specific task and because of this it made the results more dependant on the documents available than the efficacy of the search methodology.

## Results

For the first model, we use existing pre-trained GloVe embeddings.  We used the 6 different queries of varying complexity. And scores were manually calculated based on P@10 or "Precision at 10", which corresponds to the number of relevant results on the first search results page that are correct results.

Table 1. Six different queries and corresponding “precision at 10” scores using GloVe embeddings.

| Search | Score |
|---|---|
| ‘function that calculates distance’ | 1/10 |
| 'merge two lists' | 5/10 |
| 'determine if a Sudoku is valid' | 0/10 |
| 'unique binary search tree' | 1/10 |
| 'voice recognition function' | 1/10 |
| 'LSTM model for semantic search' | 0/10 |

For the second word2vec methodology, the P@10 scores are listed below:

Table 2. Six different queries and corresponding “precision at 10” scores using word2vec embeddings.

| Search | Score |
|---|---|
| ‘function that calculates distance’ | 1/10 |
| 'merge two lists' | 7/10 |
| 'determine if a Sudoku is valid' | 0/10 |
| 'unique binary search tree' | 0/10 |
| 'voice recognition function' | 0/10 |
| 'LSTM model for semantic search' | 0/10 |

As we can see from Table 1 and 2, both GloVe and word2vec embeddings did not score very well on all the six queries except on 'merge two lists' search. GloVe is an unsupervised learning algorithm for obtaining vector representations for words. The model basically count how frequently a word appears in a context. Word2vec models are shallow, two-layer neural networks that are trained to reconstruct linguistic contexts of words, Word2vec models try to capture co-occurrence one window at a time. Both of these embeddings methods are based on the nature of the corpus, and in our case, they did not perform well on this python dataset.

## Conclusion

While this method of search is efficient and returns some useful results, it doesn’t appear to be more effective than a simple keyword search model. This seems to stem from the fact that the tf-idf scores do not work very well for this type of language.

As a secondary test, we ran a keyword search over the docstrings to try and replicate the results of some of the test queries. The obvious test was to search for “sudoku” as in query 3 as that is by far the most relevant word to the search. The first document that comes up from a simple word search yields exactly the result we were looking for in the test query but our model did not produce this result.

People's language for documenting code varies widely and is very inconsistent. Additionally, proper sentence formation and syntax are not present and therefore words that commonly get filtered out by having low tf-idf become more highly representative of some documents. For future work we believe it would be more effective to use the readme files present with the code. Either way, there exists at least some bridge between the language present in the documentation strings and that of code as almost all of the queries returned at least some useful results.

Github link:

**https://github.com/yrzhou0120berk/w266_final**

## References


[1] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. GloVe: Global Vectors for Word Representation.

[2] Karpathy, A., & Fei-Fei, L. (2015). Deep visual-semantic alignments for generating image descriptions. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 3128-3137).

[3] Klusch, M., Kapahnke, P., Schulte, S., Lecue, F., & Bernstein, A. (2016). Semantic web service search: A brief survey. KI-Künstliche Intelligenz, 30(2), 139-147.

[4] Bengio, Y., Ducharme, R., Vincent, P., & Jauvin, C. (2003). A neural probabilistic language model. Journal of machine learning research, 3(Feb), 1137-1155.

[5] Jindal, V., Bawa, S., & Batra, S. (2014). A review of ranking approaches for semantic search on web. Information Processing & Management, 50(2), 416-425.

[6] UREN, V., LEI, Y., LOPEZ, V., LIU, H., MOTTA, E., & GIORDANINO, M. (2007). The usability of semantic search tools: A review. The Knowledge Engineering Review, 22(4), 361-377. doi:10.1017/S0269888907001233

[7] Berners-Lee, T., Hendler, J., & Lassila, O. (2001). The semantic web. Scientific american, 284(5), 34-43.

[8] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. ICLR Workshop, 2013