CODE2VEC Based Cognitive Agent System to Retrieve Relevant Code Component from Repository

Meghna Talari[1], *Krishna Chythanya* N[2], and *C.R.K* Reddy[3]

[1]PG Scholar, Gokaraju Rangaraju Institute of Engineering and Technology, India
[2]Asst. Professor, Department of CSE, Gokaraju Rangaraju Institute of Engineering and Technology, India
[3]Professor, Department of CSE, MGIT-Hyd, India

Abstract: The cognitive agent system helps to retrieve most relevant code component by introducing latest techniques. In this paper the authors used latest approach of code embedding which undergoes code2vec tokenization model by tokenizing and converting the code components present in the dataset into a numeric representation to create a input for neural network environment and also implemented cosine similarity matching technique to acquire the relevancy and perform retrieval of code component.

1. Introduction

The growth of code component reusability had increased with most of the developers or end users majorly browse for the required code components in the internet, as it is providing many open source software code components. The user generally enters query in natural language and get plenty of results among which the relevancy of required code component is less, as it contains huge amount of noisy data than relevant data. To overcome this problem the authors introduced a cognitive agent system to retrieve most relevant code component from the repository. As per the work done to implement the concept the authors made use of a latest approach called Code2Vec. This is a neural embedding process of converting the code components into numerical representation called vectors. According to Piyush Arora etal[1], The conversion of code component to vectors can be done in three ways - one is general code as vectors in which the spaces or new lines and stop words are eliminated using tokenizer, another one is tokenization in which it provides the lexical scanner for the code components to convert into vectors, and last one is AST(abstract structure tree) in which the code components are separated depending on their relationship and represented in the form of a tree. In this work natural language processing is used to perform the retrieval. The code2vec is mainly used to predict the method names. With an idea of fetching the method or code snippet of a particular method, it was decided to implement Code2vec concept as it was proven as effective code embedding for predicting the methods. As per Hong jin kang[2], they proposed token embedding's by code2vec to represent the source code in three downstream tasks as code authorship identification, code clones detection and code comment generation but resulted with a thread of not generalizing to other tasks. By considering these two papers the authors decided to implement code2vec for retrieval process using tokenization. The cognitive agent system has implemented cosine similarity matching technique to fetch the most accurate code component from the repository - as per the Tim vor der Brick etal[4], they have calculated many similarity matching models and demonstrated cosine similarity as most accurate especially on large dataset. By analyzing piece of writings available, the authors have decided to implement a cognitive system which need to be user friendly, get accurate results and gives reusable code components.

The vectorization idea is implemented as it reduces the complexity and increases the quality of results. The authors want to develop a user friendly system hence, implemented using "tkinter" to create a user interface and the query which the user enters can be in any random combination to get accurate results. The user input is a combination of features of code components required.

2. Methodology

The author has introduced embedding technique called Code2Vec in which the code snippets from the dataset will be converted into vectors by using tokenization method.

For finding the similarity, the author has used cosine similarity technique which can give the best results.

The retrieval is performed to get relevant code snippet by combining both the techniques.

2.1 Steps followed

Step 1: User uploads the data set into the system

Meghna Talari: meghnatalari@gmail.com

Step 2: The data (code components) is internally pre-processed

Step 3: The pre-processed data is applied to tokenization method

Step 4: The system asks user to enter the query which is to be retrieved

Step 5: Internally the cosine similarity between the data set and query entered is calculated

If(result) = 1

"Query found"

If(result) = 0

"Query not found"

Step6 : The most relevant code component is retrieved from the database if the result is 0 or system asks to re-enter the query

Step7 : The relevancy with other files is also shown

Step 8 : User can view the graph of query entered and similarity present in the database.

3. Experimentation and Results

Procedure follows five modules 1) Uploading the dataset, 2) Pre-processing the dataset, 3) Code2Vec 4) Giving input and 5) Retrieving the code component.

The first step is to upload the dataset for pre- processing where each and every code component from the dataset is analyzed by the agent, the count of dataset is viewed and then it is applied on second module in which the whole dataset containing the code components is embedded by applying embedding technique then the role of user takes place by giving query as an input which is also converted as vector internally and finally perform comparison is done by calculating the cosine similarity between them.

The retrieval is performed based on cosine similarity score after which the acquired highest matching score document is retrieved.

In the below architecture the overview of the cognitive agent system developed is given. The Qv1 is query entered by the user and Cv1, Cv2,…Cvn are the dataset which are embedded by performing tokentization and v indicated the vector.

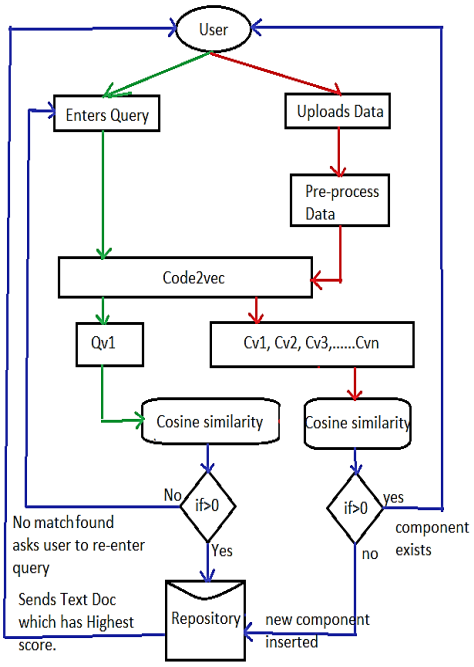

Fig. 1. Overview of cognitive agent system developed

3.1 Uploading Dataset

Authors have collected the dataset of many programming languages like Java, C language, Python, C# and C++. The process of uploading is done on user interest as the users can upload depending on their requirement to add components when they are not available in repository, if user wants any code snippet of any particular programming language then he can upload that directory to the system. For example, if user needs code snippet on python language then he can upload python code files. If user wants collection of code snippets on any language then he can upload a folder named train in which the programming components of the language are collected in one folder. In this experiment case we used a train folder that has more than 300 code components including simple code snippet to project functions.

3.2 Pre-processing Dataset

The raw dataset need to be uploaded in the cognitive system in which the raw data is analyzed by the agent to apply embedding. Each and every code component data from the directory is analyzed and displays the number of components present in the repository which the user has uploaded.

3.3 Code2Vec

This is the process of converting the code components into vectors by performing tokenization technique. Representation in the form of vectors is also known as neural embedding. The dataset uploaded by the user is analyzed and embedded by performing tokenization. In lexical analysis, this technique eliminates the stop words including extra spaces, new lines and breaking a stream

of code into words, symbols, phrases or other meaningful elements called tokens by importing NLTK corpus and displays the count of each vector in the dataset as shown in the following figure.2.

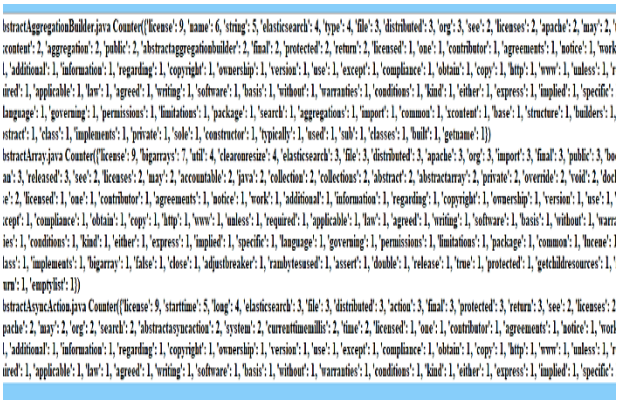

Fig.2 Code to Tokens

In the above figure, the tokens and the count of the tokens are displayed. The numerical representation can make the retrieval process more effective by giving qualitative results. Embedding can help the system to reduce the complexity of the data by representing in the form of vectors

3.4 Giving Input and Retrieval process

The input can be given in the form of features of more than one word, for example if user needs code snippet of "implementation of recursive bubble sort", rather than typing the whole sentence in natural language query the user can enter only features like "recursive bubble" or "bubble recursive sort" or "bubble sort recursive". Presence of agent helps the system to analyze the query and convert the query into vector internally.

To retrieve the component which satisfies the query, the agent performs cosine similarity measure in which the Euclidean Distance between the vectors of dataset with the vector value of user entered query is calculated. To compute cosine similarity, it considers the vector values in each document and vector values of the query. The result is acquired in the form of matrix and gives similarity score for it. The following image is the basic calculation of cosine similarity measure from [5].

$$Cos\theta = \frac{\vec{x}.\vec{y}}{||\vec{x}||\vec{y}||} = \frac{\sum_1^n x_i y_i}{\sqrt{\sum_1^n x_i^2}\sqrt{\sum_1^n by_i^2}} \qquad (1)$$

Where, $x^{\rightarrow}.y^{\rightarrow} = \sum_1^n x_i y_i = ax_1 y_1 + xy_2 + \cdots + x_n y_n$ is dot product of the two vectors (query vector, and code component vector).

4. Experiment results

Concentrating on the score acquired by cosine similarity measure, the maximum score is considered as most relevant one and retrieved from the repository in the form of text document to make it user friendly. The query entered by the user is compared with all the programs present in the repository by measuring its similarity, the programs which acquire match is displayed with its title name and predicted score as shown in the below figure.3

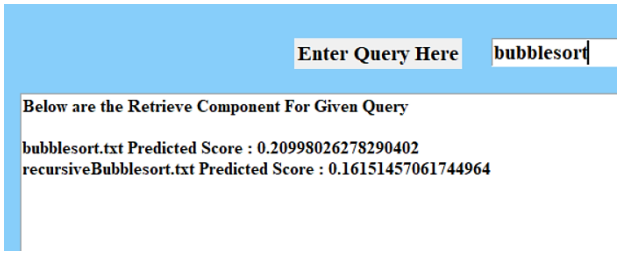

Fig.3 Predicted score

As per the above figure, program named bubblesort.txt is retrieved as it has near Euclidean Distance. Here the advantage is that we not need to go through the whole program of bubble sort, only the relevant code snippet of bubble sort function is retrieved as a text document as shown in the below figure.4.

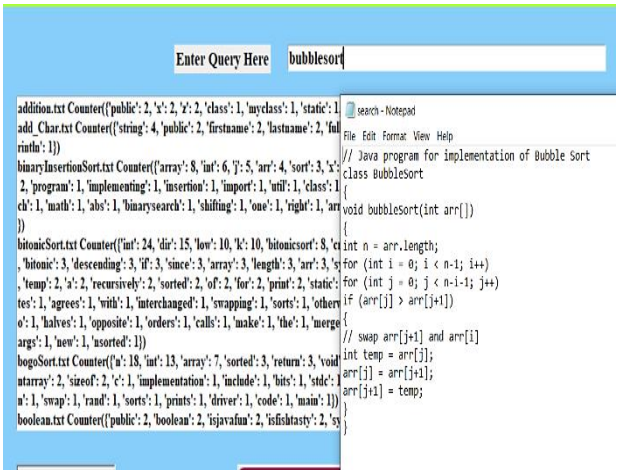

Fig.4 Retrieved code snippet

The representation of the results has displayed in the below graph of total code components in the repository and the components which found similarity.

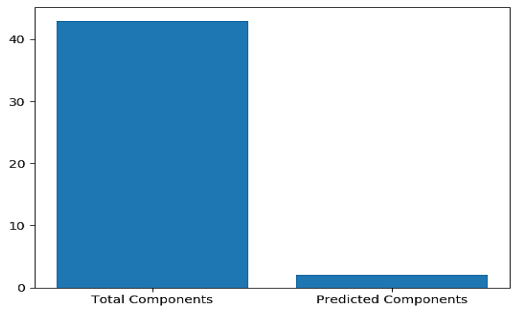

Fig. 5 Graph of total components Vs Predicted components

Table.1 Confusion matrix

Total=359 components; 20 queries.	Classified as Existing	Classified as Non Existing
Existing Components(12)	12 (TP)	1(FN)
Non Existing Components(4)	3(FP)	4(TN)

By considering the above Table 1 - Accuracy, Precision, Recall are calculated.

$$\text{Accuracy} = \frac{TruePositive+FalseNegetive}{TotalNo.ofSamples}$$

$$= 16/16 => 100\%$$

$$\text{Precision} = \frac{TruePositive}{TruePositive+FalsePositive}$$

$$= 12/15 = 0.8 => 80\%$$

$$\text{Recall} = \frac{TruePositive}{TruePositive+FalseNegative}$$

$$12/13 = 0.923016923 => 92.3$$

Our experiment results on 350 component repository where 20 user queries were used shows the values as given in Confusion Matrix table 1. above where in we considered queries for 12 existing components and deliberately given 4 queries for non existing components. True Positive(TP),True Negative(TN), False Positive(FP), False Negative(FN) values found are recorded as in matrix above. We could get an Accuracy of 100% with our model and a Precision of 80%.

5. Conclusion

To increase the reuse of software component, software component repositories and to improve the relevancy of the retrieval process - the authors have used Code2Vec concept in which the dataset is embedded by implementing tokenization technique which converts the whole code components from the dataset into vectors. The cognitive system attained success in retrieving the most relevant code snippet by comparing their cosine similarity measure and made it user friendly by abetting in the form of text document.

6. Future Work

As an extension authors would like to work with both Code2Vec using tokenization technique or AST(abstract syntax tree) and Word2vec using skip gram technique or CBOW(continuous bag of words) concepts giving more deep attention on neural network, to achieve better results. More models can be tried to improve the Precision and Recall of the system.

References

1. DavidAzcona, Piyush Arora, I-Han Hsiao, Alan Semeaton, *"user2code2vec:Embeddings forbProfiling Students Based on Distributional Representations of Source Code",* In The **9th** International Learning Analytics & Knowledge Conference (LAK19), Mar(2019), Tempe, AZ, USA.ACM,NewYork,NY, USA,10pages. https://doi.org/10.1145/3303772.3303813
2. Hong Jin Kang, Tegawende F. Bissyande, David Lo, *"Assessing the Generalizability of code2vec Token Embeddings",***34th** IEEE/ACM International Conference on Automated Software Engineering (ASE), 11-15 Nov (2019),https://ieeexplore.ieee.org/abstract/document/8952475
3. Bart Theeten, Frederik Vandeputte, TomVan Cutsem, *"Import2vec Learning Embeddings for Software Libraries",*Proceedings of the **16th** International Conference on Mining Software Repositories, May (2019) , https://www.researchgate.net/publication/332300538_Import2vec_-_Learning_Embeddings_for_Software_Libraries
4. TimvorderBr¨uck, Mare pouly, *"Text Similarity Estimation Basedon Word Embeddings and Matrix Norms for Targeted Marketing*", Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume **1**, june (2019), https://www.aclweb.org/anthology/N19-1181
5. https://www.machinelearningplus.com/nlp/cosine-similarity/