Auto Code Generation using Classified Source Code Archives
Kavya V J
Govt Engineering College, Sreekrishnapuram, Palakkad, Kerala, India
DOI: 10.23956/ijarcsse/V7I6/0139

Abstract—The Auto code generation system helps the programmers to write the code easily. The system accepts a text input as the user requirement and returns back the program code from the existing corpus. Thereby the programmers can save their time by not searching throughout the web for their requirement which might be new for them. Initially the code repositories are classified into various categories. This classification later helps in retrieving the code quickly. Once the user types the text query the category of the text is predicted. The category predicted is used to map into the code corpus which is kept already categorised. The text similarity is computed with the program code contents which are in the particular category. Thereby the code with highest similarity is returned to the user. Hence the system help improve the programmers productivity.

Keywords—Auto code generation, classification, machine learning, SVM, similarity.

I. INTRODUCTION
The auto code generation helps the programmers to utilise their time efficiently as they get it readily. Otherwise they will have to go through various sites for their requirement to be satisfied. Hence this improves their productivity and also enables to reuse the existing code. The user gives his requirement in the form of text and he gets the program generated. The code is displayed quickly and easily.

There are lots of program codes spread over the world wide web. It would be easier to handle if the source codes are categorised using various topics. The categorising of the source code can be done on the basis of the code comments, other text and keywords in the program.

The text extracted from the code is hence used for categorisation. There are various algorithms for text categorisation. Some of the popular text classification algorithms are Decision tree, Naive Bayes, Support Vector Machines, K Nearest neighbors, K-means, Neural networks. Three main approaches used for text classification are bag of keywords approach, statistical systems and rule based systems. The bag of keywords approach is the simplest. Various studies shows that Support Vector Machine outperforms other classification algorithms. The classification problem is studied widely in various areas like data mining, database and information retrieval communities. There will be a set of training records which are labeled with a class value.

The training data is used to create the classification model. The classification model relates the features in the record to one of the class labels. Later the training model is used to predict the class label of a test instance. Though it is possible to use continuous values as labels the problem assumes categorical values for the labels.

By classifying the source code collections automatically manual organisation of the source code repositories are avoided as they are also large and growing rapidly. Hence by this classification we can reuse the source codes efficiently.

There are various studies conducted in information retrieval for measuring similarity between documents and queries. In this work the query is the text provided by the user. The angle between the document vectors in the term vector space gives the similarity. That means the cosine angle between the documents determine the document similarity. In the information retrieval vector similarity is used for ranking where for a query, most similar documents are ranked in the order of similarity.

The auto code generation helps in reducing the development costs. It also improves the quality of software developed and also increases the agility of development process.

II. RELATED WORKS
There are works related to clone detection and removal. The clones are detected using abstract syntax tree in one of the works. The clones usually occur by code reuse, coding styles, accidents, etc. The source code is parsed and an AST is produced as the first step. To find clones three algorithms are applied. Sub-tree clones are detected first. Sequence
detection algorithm is performed then. Third algorithm looks for near-miss clones. The automation of removal of detected clones from the source is considered as the next step in their work. Other improvements like performance are also taken into account [17].

Clone removal works discuss the elimination of function and class clones from the industrial object oriented code thereby facilitating the reduction of code size and hence helpful for the maintenance. Restructuring is done for the removal of duplicate codes which is performed in two steps: searching for the clones and then replacing them by a single code entity. Ripples effects caused by the code removal need to be handled [15].

There are also works done for pattern mining in the source code. Frameworks are proposed to automatically infer system specific interface specifications from program source code.

The paper "Mining idioms from source code" presents the first method to mine code idioms from a collection of software project codes. An idiom is defined by the authors as the syntactic fragment that recurs across the projects that have a single semantic purpose. A system Haggis is presented for mining idioms. Advanced statistical natural language processing techniques like nonparametric Bayesian probabilistic tree substitution grammars are used by the system [4].

In the paper "Mining Programming Language Vocabularies from Source Code" a foundation is laid for applying corpus linguistic methods to programming language. Term word is defined in the programming language. Data collection tools and data storage schema were developed for Java programming language. Also it presented a linguistic analysis of an example corpus. They take the hypothesis that the use of programming language mimics the use of natural language. In their work they define words as the atomic units from which vocabularies can be composed. The work also extended to the data collection and storage implementations. They also validated their proposals with preliminary studies. Their results were based on a data collection of 10,947 java source files which contain 3,148,796 lines of source code. They make it more accessible for the novices by being able to detect the conventions in the programming languages with the help of automated tools. Also these helps in developing style checking tools which makes it more acceptable and useful for the developers [8].

Vipin et al. proposes a solution for the problem syntactic patterns querying in the underlying programming language rather than using specialised query. The solution is Abstract Syntax Tree (AST) structural similarity match. An AST is a tree representation of the abstract syntactic structure of source code written in a programming language. The query snippet is converted to AST and compared with subtrees of source files in repository. Relevance score is calculated for each of the source file. Two challenges were that there. One is there can be only a small subtree of a source file that will be matching the AST of the query snippet. Other challenge is there can be many subtrees of the source file that might match the query snippet. The challenges were addressed by considering the subtrees of the query AST with size above a certain threshold and computing the similarity between these subtrees and all the subtrees of a source AST. The paper gives a brief background about the Abstract Syntax Tree and about Tree Edit distance and approximation. Also an approach is proposed to scale the algorithm for the problem to code repositories which are large based on numerical vector approximation of trees and locality sensitive hash functions [3].

III. AUTO CODE GENERATION

The source code contains a lot of text which can be used for natural language processing. Thereby utilising this we can perform the classification of source code archives. When the source code archives are classified it can be handled easily. Here we utilise this in order to later fetch the appropriate source code when the user requests. The user types a text like file upload and the system fetches the source code related to that query. Figure 1 shows the flow of code categorisation. Figure 2 shows the auto code generation system design.
A. Implementation
Various steps implemented in the system are as follows:
- Corpus collection
- Creating training and test datasets
- Extracting feature vectors for machine learning
- Training a model to perform classification
- Predicting the category
- Code generation

B. Corpus collection
Source codes are collected from various source code repositories across the web like GitHub. The classification of the program codes are done based on the text extracted from the source codes. Text include the comments, other statements like those in print statements and other in line documentations in the source codes. These text documents created from the source codes are grouped and later procedures are performed. The text documents are preprocessed. Preprocessing involves tasks like elimination of stop words and word stemming. Removal of stopwords gets rid of irrelevant and noisy data.

C. Creation of training and test datasets
The text documents created from the source code are initially split into training data and test data. The training data and test data are categorised.

D. Extracting feature vectors for machine learning
The text content is converted to numerical feature vectors in order to later perform machine learning. Bag of words representation is used. Each feature corresponds to the word in the training corpus for most “bag of words” representation. Apart from the removal of infrequent and infrequent words and stopwords, features are made more statistically independent. Stemming is performed. Stemming maps the words to its root form. For example “learner”, “learning”, “learned” are mapped to the stem word “learn”.

Other kind of representation is ”phrase based representation” where the phrases are used as features thereby preserving some of the information left out in bag of words. But the problem with using this representation is the increase in the number of features.

There is also another kind of representation called hypernym representations. The bag of words also does not consider the semantic relationship between words. The WordNet can be used for this purpose. WordNet contains information about synonymy and hypernymy.

In case of bag of words representation a fixed integer id is assigned to each word in the training dataset. Number of occurrences of each word in each document are also stored in matrix. Bag of words are typically sparse datasets.

E. Training a model to perform classification
Once we get the features the classifier is trained to predict the category of new instances. Various classifiers are trained and checked. Found the accuracy more for SVM classifier. Therefore linear Support Vector classification is used.
It supports dense and sparse input. One vs rest scheme is the scheme followed to support multi-class support. Linear function is given as $<x, x>$. The figure shows the comparison of various classifiers with their scores, training time and testing time.

**F. Predicting the category**

When new document category has to be predicted, the features are extracted. The method predict($X$), provided by scikit-learn package helps in predicting the class labels for the samples in $X$.

**G. Code Generation**

Atom IDE is used for the demonstration of code generation. The user types the requirement for the code as a text and trigger for obtaining the source code. The text input given by the user or the programmer is then passed to the model used for classification which predicts the category. The category of the text is then used to select the collection of program codes that belong to the category in the code corpus. Then the text similarity is checked with those text documents which had already been extracted from the program codes. The most similar text document’s program code is returned to the user.

### IV. RESULTS

The correctness of the auto code generation system is evaluated on the basis of how much accurately the classification of program codes happened and the usefulness of the code returned by the system to the user. An accuracy of 78.5% is obtained for predicting category of a new instance. Table 1 shows the results.

<table>
<thead>
<tr>
<th>Category</th>
<th>Precision</th>
<th>Recall</th>
<th>F1-score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Area</td>
<td>0.75</td>
<td>1.00</td>
<td>0.86</td>
</tr>
<tr>
<td>Directory</td>
<td>1.00</td>
<td>0.75</td>
<td>0.86</td>
</tr>
<tr>
<td>File</td>
<td>0.85</td>
<td>0.65</td>
<td>0.73</td>
</tr>
<tr>
<td>Quadratic</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>String</td>
<td>0.67</td>
<td>0.86</td>
<td>0.75</td>
</tr>
<tr>
<td>Add</td>
<td>0.50</td>
<td>1.00</td>
<td>0.67</td>
</tr>
<tr>
<td>Random</td>
<td>1.00</td>
<td>0.50</td>
<td>0.67</td>
</tr>
<tr>
<td>Search</td>
<td>0.80</td>
<td>0.67</td>
<td>0.73</td>
</tr>
<tr>
<td>Set</td>
<td>0.80</td>
<td>1.00</td>
<td>0.89</td>
</tr>
<tr>
<td>Variable</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>avg/total</td>
<td>0.81</td>
<td>0.79</td>
<td>0.78</td>
</tr>
</tbody>
</table>

### V. CONCLUSION AND FUTURE WORK

Here an auto code generation system is proposed and implemented which helps the programmers to write the code in a productive way. They need not have to search for the code across the web. Currently the system depends on the classified program codes for the fast retrieval of appropriate program code. The classification is based on the in line documentations and other keywords. Also the system has taken only a small code corpus. The system works only for python language. It can be extended to other languages.

**REFERENCES**


[17] Ewan; Loper Edward Bird; Steven; Klein, Natural language processing with python, 2009.