ISSN No:-2456-2165

Knowledge Graph Creation based on Ontology from Source-Code: The Case of C#

Seid Mehammed, Sebahadin Nasir
Department of Computer Science, Woldia University

Abstract:- The software's core data and business logic are believed to be contained in the source code. Therefore, the necessity for a semantically soundly linked and structured code data management system is a major challenge in the field of software engineering. This paper investigates a domain ontology-based automatic knowledge graph creation method for C# source code. The semantic web, open-source developers, knowledge management, expert systems, and online communities are just a few of the fields where software engineers may now understand and analyze code in a semantic manner. By layering conditional random fields on top of a trained Bi-LSTM network, candidate terms for concepts or entities were extracted.The models were automatically trained on a labeled data corpus while also being manually defined. To improve the classification of terms in a particular source code, BI-LSTM and CRF are integrated. Other characteristics to be extracted from the source code were defined in addition to the basic CRF features, which helped the model understand the categorization constraints. Then, the Bi-LSTM model was utilized to extract relations (taxonomic and non-taxonomic). Max pooling has been used to integrate the links between concepts at the word and code levels.

Studies demonstrating the applicability and practicality of the proposed approach make use of the SNIPS-NLU library, a C# library for natural language processing. The evaluation process made use of both expert evaluation and the gold standard ontology that was established by experts. According to an expert analysis of the experiment's results, this approach generated an average f-measure and relevance of 77.04 and 81.275, respectively. By extracting elements and relations from C# and other programming languages that are similar, recurrent neural networks appear to be efficient and promising.

Keywords:- *Knowledge graph, ontology,, knowledge base.*

I. INTRODUCTION

An ontology is a formal explicit specification of shared conceptualization of a domain, where formal specialization denotes machine-readability with computational semantics, explicit denotes unambiguous terminology definitions, and shared denotes that it is generally accepted understanding and conceptualization, implying conceptual model of a domain. [1].

Ontology learning, which is also referred to as ontology extraction, ontology generation, or ontology acquisition, is the automatic or semi-automatic creation of ontologies that entails extracting the terms from a corpus of natural language text that correspond to the domain and the connections between the concepts those terms represent, then encoding them with an ontology language for easy retrieval.

Any piece of software must have its source code. The source code contains all errors that arise during software testing or execution. As a result, source code ontologies represent the program source code in terms of the software objects it contains, including modules, packages, functions, namespaces, variables, and database objects. Building an ontology for software source code manually takes a lot of time and effort because there is so much source code available in a software. Consequently, the domain is crucial for automatic ontology learning [1, 4, 5].

Additionally, a dynamic resource in software projects is the source code. The ontology must develop and be updated as the code is modified. The ontology of a knowledge graph [6] can be expanded and altered as new data is received. It is a visual, intelligent, and dynamic representation of knowledge. Knowledge graphs are the best tool for working with a dynamic dataset because they may capture a variety of meta-data annotations, such as provenance or versioning information. As a result, it can be used to represent the source code ontology, a changing resource.

II. RELATED WORK

A. Introduction

Ontologies have been present in research for decades, serving as a back bone for semantic web and semantic representation. However, interest in ontologies was lost, as machine learning became the hot research area taking focus of researchers. However, in the past decade, ontologies and semantic data came back into the spotlight. Realizing that knowledge management, data integration, data publishing, smart data access and analytics are impossible without the smart knowledge representation, different researchers made their intention on the area. In this Chapter, we will try to present some of the works in ontology learning starting from those dealing with text data to source code.

ISSN No:-2456-2165

B. Ontologylearning from text source

A large collection of methods for ontology learning from text have developed over recent years by proceedings of various workshops in semantic web area.

- Amharic Ontology Learner(AOL)

  AOL [32] is a learning system that converts plain text written in Amharic into a domain ontology automatically. The automatic idea extraction from an Amharic text document, relationship mining (among concepts), and formal representation of those concepts on an ontology are all being done for the first time in this research project. In order to create an ontology that can represent a given domain, the Amharic Ontology Learner takes a set of Amharic documents that are focused on that topic as input.

  They used TF-IDF based method for single word concept extraction and C-Value for multi-word concepts. Then agglomerative hierarchical clustering and verbal expressions were used for the extraction of taxonomic and non-taxonomic relations respectively. The researchers used Parts of Speech Tagged multi-domain news document of WALTA information center. They have achieved 70%, 70.20% and 51.7% of precision in concept, taxonomic relations and non-taxonomic relations extraction respectively.

  However, these ontology-learning approaches are inadequate to deal with ontology learning from source code. This is because of the following challenges in learning domain ontology from source code as mentioned in Bontcheva et al. [2]. Each programming language and software project tends to have naming conventions and these need to be considered. The second problem is that the ontology learning methods need to distinguish between terms specific for the programming language being used and the application-specific terms. And many of the extracted terms can refer to the same concept. Hence, researches have been conducted in providing best method for building ontology from software source code.

C. Ontology Learning from code

- Java Ontologies

  An efficient method for automatic generation or extraction of ontology from software source code has been a critical issue. Genapathy and Sagayaraj [5] proposed a method to automatically create ontology by extracting metadata from the Java source code. They used Qdox to extract metadata from the source code. The metadata is an information about the package, classes, methods and interfaces in the file. After extracting the metadata, the frame work stores the meta-data in to OWL using Jena API. Then the entire project folder stored in the HDFS, is linked to the method signature in the OWL ontology for retrieval purpose. By generating ontology for source code, it will be effective to make software source code available for semantic webs, reuse code, and extract components of the software. Rather than using different APIs in extracting ontology even for metadata it will be effective using machine learning to discover relations and rules.

  K. Bontcheva and S. Marta [2] also developed ontology learning (extraction or generation) from software artifacts which is from multisource including source code. Simple terms are extracted from the source code and then used as a starting point for identifying compound terms in the user documentation. They combined terms from source code and other sources to learn concepts or domain terminologies. They have experimented with term extraction from 536 Java source files in GATE Version 3.1. They found only 218 terms have frequency more than 1 out 576 total terms extracted. Then these terms were combined with terms from forum posts producing 153 multiword terms. Totally they found 286 frequent terms out of 719. Evaluated by a domain expert this resulted in precision of 73.4%. They have achieved only two tasks of ontology learning namely term extraction and concept identification. Their work fails to create concept hierarchy or taxonomic relations and non-taxonomic relations among concepts. It also fails to deal with the dynamic nature of source codes, which needs ontology versioning and evolution.

  F. JiomekongAzanzi& C. Gaoussou [8] proposed a method to extract knowledge from the source code for ontology enrichment. Their method is based on Hidden Markov Models (HMMs). This work was aimed to extract terms from source code which can be annotated as different components of ontology automatically. From the source code, they have extracted 808, 55111, 3522, 263 candidate terms to be concepts, properties, axioms and rules respectively. The relational terms they have extracted are only hierarchical which are programmer defined. Even though they have presented the knowledge extracted is complete, it was by comparing the number of terms they have extracted with those extracted from database and meta model; this shows their work lacks a good evaluation method. This work can be as a good input for ontology learning which simplifies the first and important step namely domain term extraction.

  As we can see from [1, 2, 5, 8], ontology learning from software source code is not fully covered through researches. Different frameworks and architectures were proposed for ontology learning from textual data while learning from code has no general architecture. Therefore, this work is aimed to develop an ontology learning method from software source code in which the generated ontology can fully represent the source code in a given project, and also to automate all the tasks in ontology learning using deep learning neural networks. Knowledge graph construction for code based on the domain ontology is also to be addressed by this research work.

D. Knowledge Graph Creation

Knowledge graph has been the hottest research area since Google released its first knowledge graph as part of the search engine in 2012 [23].

Martinez-Rodriguez et al. in [33] proposed Open-IE based approach for knowledge graph construction from text. This method is based on a combination of Natural Language Processing (NLP) and Information Extraction (IE) operations in order to transform an input text into RDF triples. This paper presents basic techniques and methods used in knowledge graph construction. They classified the process into EEL (entity extraction and linking) and REL (relation extraction and linking) plus property identification. By integrating different approaches of entity and relation extraction, the researchers tried to fill the gap in KG (Knowledge Graph) construction. By implementing in Java, they have achieved a precision of 0.81 in entity extraction and 0.63 in relation extraction.

E. Summary

Different approaches have been developed for extracting ontology from textual and source code data. While very important for understanding the methods and techniques in the field of ontology, methods used for ontology learning from text data such as Natural Language processing and Linguistic based approaches are not applicable when it comes to source code because of the nature of code as presented in the work of [2]. Therefore, researchers have been attempting to develop an approach for learning ontology from source code.

After extracting ontology from a given domain, there is a need for integrating a new arriving data into the ontology and this was done by re-extracting the ontology. Extracting ontology becomes a redundant work then. Therefore, we need a flexible and graphical representation of the ontology where the newly arriving data can simply be integrated into the knowledge base. Hence, this study is mainly to design a general approach for knowledge graph construction based on ontology from C# source code.

To the best of researcher's knowledge, this is the first work to explore the advantage of using knowledge graph for code analysis and representation. In this paper we will extract RDF triples from C# source code and visualize the knowledge graph using forced graph.

III. ONTOLOGY LEARNER AND KNOWLEDGE GRAPH CREATION

A. Introduction

To handle their code as data in the semantic web, software engineers require a standard method to extract ontology and build knowledge graph from their source code. Thus, by detailing various components and pertinent approaches, this part describes the suggested basic architecture for automatic ontology extraction from source code and knowledge network generation.

B. System Architecture

The source code ontology learner creates an ontology that contains the pertinent concepts of the domain and their relationships by accepting source code files as input. The knowledge graph is then created using the data that was taken from the structured ontology. The general architecture of our suggested method is depicted in Figure 1. The three primary stages of our approach's overall architecture are code preprocessing, learning domain ontologies, and building code knowledge graphs. The sections that follow cover each component in detail.

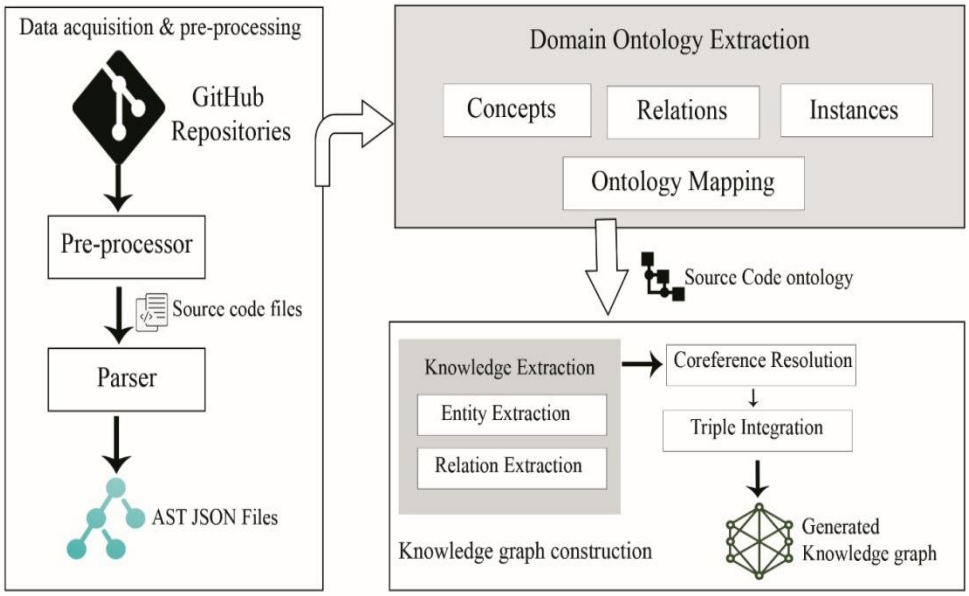

Fig. 1: General Architecture of Proposed System

- Data preprocessing and parser

ISSN No:-2456-2165

Data was gathered using a C# open source Snips-NLU code that was downloaded from GitHub. There are numerous more files in software projects that are not part of the source code but do include information on the project or code, such as documentations, readme files, and debates. Consequently, in this phase, markdown (.md) files, YAML files, text files, and copies from other projects are all eliminated and just source code files are provided for the following step. This is because ontology learning from text data sources is a topic of extensive investigation. The following parts describe language detection and code validation, the subsequent preprocessing stages. Data was gathered using a C# open source Snips-NLU code that was downloaded from GitHub. There are numerous more files in software projects that are not part of the source code but do include information on the project or code, such as documentations, readme files, and debates. Consequently, in this phase, markdown (.md) files, YAML files, text files, and copies from other projects are all eliminated and just source code files are provided for the following step. This is because ontology learning from text data sources is a topic of extensive investigation. The following parts describe language detection and code validation, the subsequent preprocessing stages.

- **Code validation**
  Prior to using the parser to extract the candidate phrases, this sub-task is equally crucial. This is due to the fact that using the pre-defined parsers in this work requires a valid and syntactically accurate code. The grammar of any programming language's source code is checked using a straightforward technique that is tailored to each programming language. Giving the parser just legitimate code files will assist to reduce errors during the parsing stage. While others do not, some computer languages provide parser libraries that do. These libraries can detect errors in a code during parsing.

- **Code Parser**
  Parsing a code into its abstract syntax tree (AST) and expressing it in a Json format comes before extracting keywords or words that are potential candidates for concepts. Similar to syntax trees used by linguists for human languages, an abstract syntax tree represents each syntactical component of a programming language. We are unable to tokenize source code using linguistic filtering, natural language processing tools, or other techniques, as we do with human language.

  AST focuses on the rules rather than elements like braces or semicolons that terminate statements in some languages. Since AST module provide only the relevant words for analyzing the code, we don't need any method to get rid of those bulky irrelevant punctuations. This makes it better than defining regular expressions for getting rid of these kind of code elements.
  - HTML / XML open/close tags
  - Open/close braces {/} in programming languages
  - Open/close parentheses in arithmetical expressions
  - To parse these types of patterns, we need something more powerful like parser (AST).

Hence, AST library is used to parse the code file into a tree of nodes for different programming languages in this work. Each node of the tree stands for a statement occurring in the code. Sample AST parser libraries for some programming languages. languages which do not have libraries for parsing can be parsed using user defined codes by using built in AST parsing libraries (like AST for C#).

- Domain Ontology Extraction
  Domain ontology extraction as mentioned in above chapters is automatic extraction of the elements of ontology for a specific domain. Hence, this module is concerned about extracting all the elements of ontology from source code files. This module includes task like term extraction, concept learning and relation extraction. The relation can be classified as taxonomic and non-taxonomic among entities. Methods and techniques, we have used for each sub tasks in ontology learning are discussed in the following sections.

IV. TERM EXTRACTION

A crucial stage in ontology learning is term extraction, which will be dealt with in this lesson. To mine semantically significant aspects of a domain, such as entities, is the process of term extraction, or information extraction (e.g., concepts and instances). This process is managed using NER (Named entity recognition), which is based on linguistic or statistical methodologies, for natural languages like English. In order to extract instances of concepts or entities from source code, NER approaches must take into account the characteristics of natural language.

Therefore, this module involves the identification of proper terms in code, and the classification of these terms into a set of predefined classes of code elements define in the work of [21] like class definition, function definition, expression, library, identifier, argument, attribute, etc. In many studies, it has been depicted that CRF on the top of Bi-LSTM is a successful method for extracting entities from unstructured and structured data sources. Therefore, we use Bi-LSTM + CRF in this research for term extraction.

Since the programmer writes code sequentially, we consider labeling terms in a source code file as sequence labeling. Programmer types the statements step by step sequentially, which indicates that given observations or input labels are interdependent. For example, let's take the statement *Result = x+y/2* in C# language: in this statement

ISSN No:-2456-2165

the value of variable result depends on the calculated result from the right-side operation. If the previous observation is a term class, it is obvious in many languages that the next observation or term is the name of class. If the previous observed label is def it's obvious in C# that the next observation will be the name of function.

A. Model

For this work, a supervised Bi-LSTM + CRF model is used to extract concepts from a given corpus. The model was trained with pre-labeled corpus. After parsing code in to ast, training corpus was prepared by labeling the terms with corresponding classes. Basic class labels are based on the general ontology classes in [21] adopted to include C# object classes. Our dataset is prepared in the format of Kaggleentity_annotated dataset [33] which has column Code-Index, word. The model consists of four layers namely, word embedding layer, Bi-LSTM Layer, CRF layer and output layer as shown in Figure 2.

B. Embedding layer

Many ML models including Keras Bi-LSTM require that the input data be integer encoded, so that each word is represented by a unique integer. This data preparation step is performed using the Tokenizer API also provided with Keras. It is basically a dictionary lookup that takes integers as input and returns the associated vectors. It takes three parameters:

- Input_dim: Size of the vocabulary in the text data
- Output_dim: Dimensionality of the embedding
- Input_length: Length of input sequence

We used Keras embedding layer for word embedding in our case.

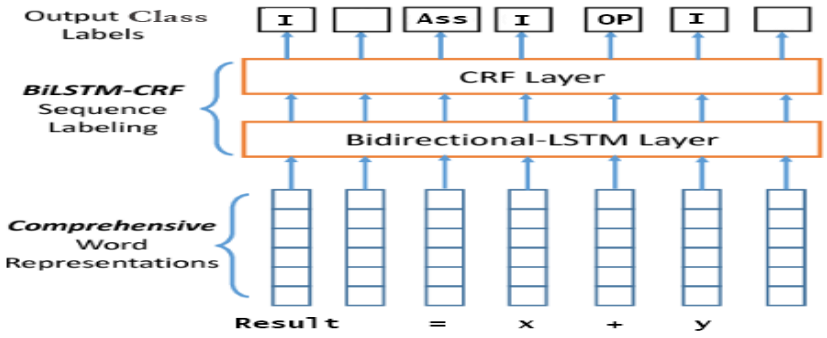

Fig. 2: Bi-LSTM+CRF *Model*

C. BI-LSTM Layer

The second component of our model is BI-LSTM and it is used to produce vector representation for our words. It takes each word in a sentence (code file in our case) as an input and produce a vector representation of each word in both directions (i.e.; forward and backward) where, forward direction access past information and backward direction access future. The bi-directional network is needed because the future words are not fully utilized when predicting label in the middle of a code. Since we are considering code file as a sentence, we need long term memory in both directions depending on the behavior of programming language.

The outputs of BI-LSTM are scores of each label for a given word/term.

D. CRF Layer

CRF layer is an optimization on top of BI-LSTM layer. CRF layer adds constraints to the predicted label during model training and this constraint could be used for validating the label.

The overall workflow of term extraction is shown in Figure 3.4. Our model is a combination of embedding, bi-lstm, and crf layers and it finally classifies terms in to their corresponding class.

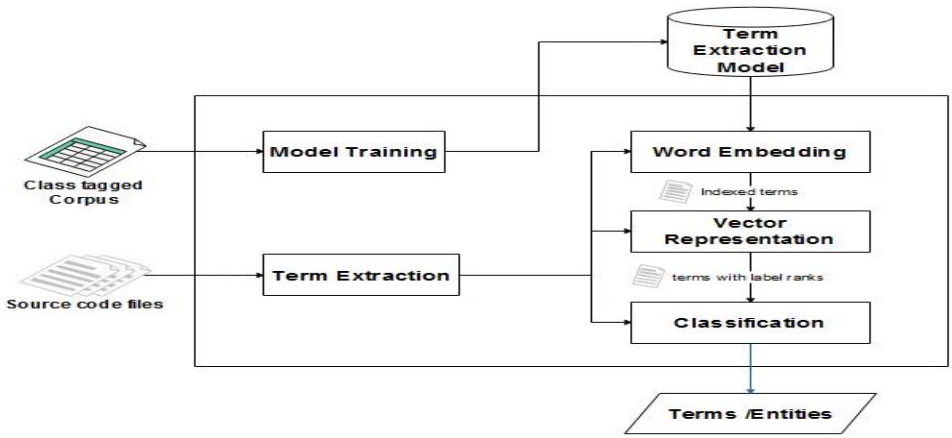

Fig. 3: Overall workflow of term extraction process

ISSN No:-2456-2165

V. EXTRACTION OF CONCEPTS

The purpose of this module is to identify concepts of a given source code data. The main step in ontology learning is extracting concepts from a given corpus and many researches have been conducted on this area.

Our concept extraction process is based on the class tags provided as the general ontology structure in the work of [21] and TF-IDF. These class labels are used to group entities according to their classes. By making AST parser to label tokens as type, value and children labels, we can use the type class elements of a code to label terms. This makes term-concept mapping simple.

VI. RELATION EXTRACTION

The next task in ontology takes us to taxonomic and non-taxonomic relation extraction. It is a very important task in ontology construction. Taxonomic relations mostly indicate "is-a", "sub-class-of", "has-type" and other relations among the concepts. If two classes 'A' and 'B' are candidate terms to be concepts and class "B" inherits "A", then, one can define a taxonomic relation between the classes 'B' and 'A'. Andboth classes are sub classes of the entity "class" in the ontology.

This module is responsible for automatically extracting relations. We choose Bi-LSTM model as this type of Recurrent Neural Network has proven itself to be well suited to tasks where remembering long-term dependencies is crucial. It is very important to keep track of the past terms and structures to detect long-distance relation patterns in a code. Figure 4 shows workflow of relation extraction process.

We adopted Bi-LSTM in [14] to extract both taxonomic and non-taxonomic relations in our research. We have implemented this module first by taking list of concepts and converting it into indexed word vector by using Keras embedding layer. Then Bi-LSTM is used to extract relations based on word/term level features and max-pooling is used to merge the sentence level relations with word level relations.

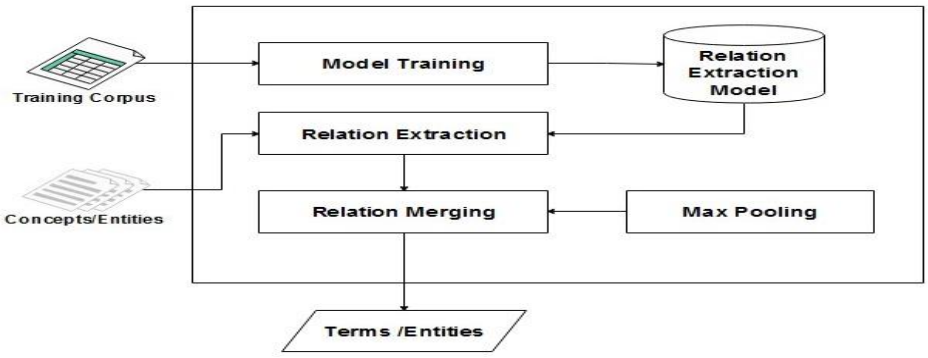

Fig. 4: Relation Extraction workflow

A. *Knowledge graph storage, Access and visualization*

We used SPARQL, an RDF query language—that is, a semantic query language for databases—able to retrieve and manipulate data stored in Resource Description Framework (RDF) format to access a knowledge graph. VOWL plugin for protégé is used to display, analyze, and visualize

B. *RDF graphs*

RDF graphs are handled using rdflib (a C# library used to manipulate rdf files). It has facilities to copy subgraphs from one graph to another, making it possible to assemble local graphs that contain facts relevant to a particular decision, work on them intimately, and then store results in a permanent triple store.

VII. EXPERIMENTS

A. *Introduction*

In this Chapter, we will present the experiments conducted on Snips NLU platform developed in C#. Our algorithms have been coded in C# and we extracted the candidate terms to be concepts and relations. In the following sections, we will first present Snips NLU platform, the experiments of the approach on Snips NLU source code, the results and evaluation of knowledge extracted from the source code of this platform and, finally, we will present the results and evaluation of our approach.

- Evaluation
  Ontology learning techniques are evaluated in two ways; automatically by using gold standard and manually by experts. We used both evaluation by experts and gold standard method to evaluate our ontology learning techniques. The gold standard ontology was developed from Snips-NLU project and validated by experts. We will use this ontology for evaluating our proposed approach in the following paragraphs.

- Expert Evaluation
  We requested two experts in the area of semantic web and ontology to check the ontology extracted carefully before evaluating. In order to resolve miss understanding during evaluation we have prepared a description the description of basic criteria of evaluating ontology. From the expert's evaluation, we have got the following result shown in Table 1.

Experiments	Relevant %	Irrelevant %
Candidate terms	77	23
Concepts	89.52	10.48
Taxonomic relations	82.58	17.42
Non-Taxonomic relations	76	24

Table 1: Expert evaluation results

The above evaluation result shows that our approach is efficient in all the tasks based on subjective expert evaluation. The system has average relevance of 81.275% from expert evaluation.

- Gold standard evaluation

  The other method we have used is gold standard evaluation which is done by using pre developed ontology of the project based on evaluation metrics, standard recall (R), precision (P) and F-measure (F).

$$\text{Precision} = \frac{TP}{TP+FF}$$

$$\text{Recall} = \frac{TP}{TP+FN}$$

$$\textit{F-measure} = 2\times\left(\frac{Precision \times Recall}{Precision + Recall}\right)$$

TP (True Positive): a positive instance that is also predicted to be positive.

FP (False Positive): a negative instance that is predicted to be positive.

FN (False Negative): a positive instance that is predicted to be negative.

Based on the above metrics, Figure 4.3 summarizes the evaluation result of the four phases in our approach.

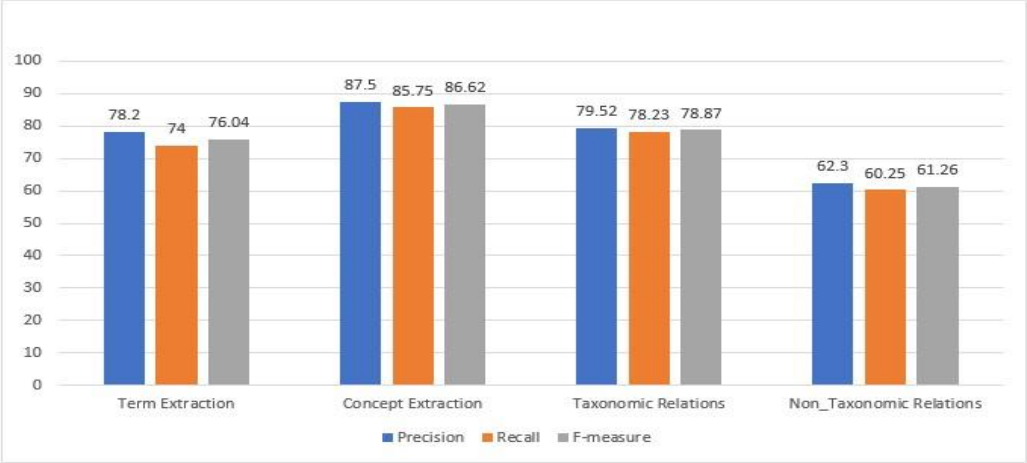

Fig. 5: Ontology learner performance measurement

From this evaluation results, we can conclude that the use of bi-directional RNN network (BiLSTM and CRF) for extraction of ontology is effective and promising. Our approach has yielded average f-measure of 77.04 as shown in Figure 5. Our extracted ontology consists all the elements in the gold standard ontology and many new elements were found.

VIII. CONCLUSION

This study's objective was to suggest a method for learning ontologies from source code that combined statistical CRF and Bi-LSTM recurrent neural networks. Additionally, this study set out to propose a framework for building knowledge graphs based on ontologies. In order to comprehend the approaches, procedures, and tools used in the creation of knowledge graphs and ontology learning, we have read a variety of scholarly works in the area. The lack of a clear and effective framework for ontology learning from source code and the gap in the semantic management of code as data served as the inspiration for this study.The majority of similar studies relied on Java source code, and Cflexibility #'s was overlooked in the semantic web. While significant, some related research focus on the generic ontological structure for object-oriented languages, while others depend on the nature of the programming language. So, based on the statistical and deep learning methods combined, we suggested a method for learning ontologies and building knowledge graphs.

The suggested method was tested using the C# programming language and the SnipsNLU natural language understanding package. A manually created CRF model was used to extract concepts and entities from the system, and a Bi-LSTM network was used to extract properties and relationships. The proposed approach showed 77.04% of f-measure during evaluation using the gold standard ontology. This indicates that the proposed approach is promising and can be refined and used to extract additional items from C# code. We employed expert evaluation to

determine the applicability of our extracted ontology elements because the gold standard evaluation is based on a manually produced ontology. As a result, our method had an average relevance of 81.275%.

REFERENCES

[1.] K. &. S. M. Bontcheva, " Learning ontologies from software artifacts: Exploring and combining multiple sources.," 2006.
[2.] Atzeni, Mattia&Atzori, Maurizio, "Codeontology: RDF-ization of source code," 2017.
[3.] Singhal, Amit, "Introducing the Knowledge Graph: Things, Not Strings," May 16, 2012.
[4.] https://www.Opensource.com, "12 challenges for open source projects".
[5.] JiomekongAzanzi, Fidel &Camara, Gaoussou., "Knowledge Extraction from Source Code Based on Hidden Markov Model: Application to EPICAM.," *10.1109/AICCSA.2017.99.*, pp. 1478-1485, 2017.
[6.] Antonio Moreno Ribas (URV) and Ulises Cortés, "Domain Ontology Learning from the Web," *Tarragona*, ,2007..
[7.] P. Cimiano, "Ontology Learning and Population from Text: Algorithms,Evaluation and Applications," *Secaucus, NJ, USA: Springer-VerlagNewYork*, 2006.
[8.] P. Buitelaar, P. Cimiano, and B. Magnini., " Ontology learning from text: Methods, applications and evaluation.," in *IOS Press*, 2005.
[9.] L. Zhou, "Ontology learning: state of the art and open issues," *in Springer Science+Business Media*, 2007, .
[10.] T. O. Ayodele, "Types of Machine Learning Algorithms," *New Advances in Machine Learning,*,2010.
[11.] S. S. S. a. S. Ben-David, Understanding Machine Learning:From Theory to Algorithms, Cambridge : Cambridge University Press, 2014.
[12.] Charles Sutton and Andrew McCallum, ""An Introduction to Conditional Random Fields"," *Foundations and Trends® in Machine Learning:*,vol. 4, pp. 267-373.
[13.] Witten, I. H. (Ian H.), Data mining: practical machine learning tools and techniques, 2010. [19] Li Deng & Yang Liu, "Deep Learning in Natural Language Processing," ISBN 978-981-105209-5, Singaphore, 2018.
[14.] D. Z. &. D. Wang, "Relation Classification via Recurrent Neural Network," *Tsinghua National Lab for Information Science and Technology*, 2015.
[15.] H. Paulheim., "Knowledge Graph Refinement: A Surveyof Approaches and Evaluation Methods.," *Semantic WebJournal*, vol. 1, p. 20, 2016..
[16.] , DuanHong, LiuYao&QinZhiguang., "Knowledge Graph Construction
[17.] Techniques[J]," *Journal of Computer Research and Development*, vol. 3, pp. 582-600, 2016.
[18.] K. Woldemariyam, "Generic Semantic Annotation Framework with Integrated," Addis Ababa University, Addis Ababa, 2017.
[19.] C. Quirk and H. Poon,, "Distant supervision for relation extraction," arXiv preprint arXiv:1609.04873, 2016.
[20.] BirhanuMengiste and FekadeGetahun, "Amharic Ontology Learner," Addis Ababa University, Addis Ababa, 2013.