Addis Ababa University
College of Natural Science

Knowledge Graph Construction based on Ontology from Source Code: the case of Python

Amanuel Tadiwos Agago

A Thesis Submitted to the Department of Computer Science in Partial Fulfillment for the Degree of Master of Science in Computer Science

March, 2021

Addis Ababa University

College of Natural Sciences

Amanuel Tadiwos Agago

Advisor: *Fekade Getahun (PhD)*

This is to certify that the thesis prepared by *Amanuel Tadiwos Agago*, titled: *Knowledge Graph Construction based on Ontology from Source Code: the case of Python* and submitted in partial fulfillment of the requirements for the Degree of Master of Science in Computer Science complies with the regulations of the University and meets the accepted standards with respect to originality and quality.

Signed by the Examining Committee:

	Name	Signature	Date
Advisor:	Fekade Getahun (PhD)	_____	_____
Examiner:	_____	_____	_____
Examiner:	_____	_____	_____

Abstract

Technology companies and online communities have been emerging tremendously and this resulted in release of millions of software. Source code is believed to hold necessarily important information about the software and business logic. Therefore, a semantically well linked and organized code data management system has been crucial issue in the field of software engineering. This study deals with an automatic method for constructing knowledge graph for python source code based on domain ontology. This allows software engineers in various fields such as online communities, open-source developers, knowledge management, expert systems, and semantic web to understand and process code semantically. A supervised Bi-LSTM (bi directional Long Short-Term Memory) network with CRF (Conditional Random Fields) on the top was used to extract candidate terms to be concepts/entities. The models were defined manually and trained automatically and simultaneously on a labeled data corpus. Using CRF on the top of BI-LSTM makes an optimized classification of terms in a given source code. Some features to be extracted from source code in addition to the default CRF features were defined and this helped the model to learn constraints for classification. Then Bi-LSTM model was adopted to extract relations (taxonomic and non-taxonomic). We have extracted relations among concepts both in term level and code level and the result was merged using max pooling.

Experiments on SNIPS-NLU library (python library for natural language processing) shows the relevance and feasibility of proposed approach. Evaluation was done in two ways, one using gold standard ontology developed by expert and the other by expert evaluation. The result of experiment shows this approach achieved average f-measure of 77.04 and average relevance of 81.275 based on expert evaluation. This result implies that recurrent neural networks are efficient and promising in entity and relation extraction from python and other related programming languages.

Keywords: knowledge graph, knowledge graph construction, ontology, ontology learning, semantic web, knowledge-base.

Dedication

To my dad, Tadiwos Agago and

To my mom, Aster Data

Acknowledgement

Thanks to almighty God my savior and provider for his love and provisions in my life. Next, I would like to express my sincere gratitude to my advisor Dr. Fekade Getahun for his guidance from problem identification and title selection to the completion of this thesis. The resources related to this work he provided me were a big help and I would also like to say thank you for the freedom he gave me.

Next to my advisor, I would like to thank Dr. Yaregal Assabie for his technical comments and encouragement. Finally, my heartfelt gratitude goes to my lovely family. Mom, dad and freye thank you for your love, care and effort to push me here. This accomplishment would not be possible without you and the rest of my family, friends (Bro Temu, Epha, Tsi, Kuni) and colleagues. Thank you.

Table of Contents

List of Figures

List of Tables

List of Algorithms

List of Acronyms and Abbreviations

Bi-LSTM – Bi-directional LSTM

CRF – Conditional Random Fields

HMM – Hidden Markov Model

KG- Knowledge graph

LSTM – Long Short-term memory

NLP – Natural Language processing

NER: Named Entity Recognition

NLP: Natural Language Processing

OBIE: Ontology Based Information Extraction

OL- Ontology Learning

RNN – Recurrent neural networks

RDF: Resource Description Framework

SNIPS-NLU- a python library for natural language understanding

Chapter One

Introduction

1.1 Background

Ontology is a formal explicit specification of shared conceptualization of a domain, where formal specialization refers to machine-readability with computational semantics, explicit refers to unambiguous terminology definitions, shared that it is commonly accepted understanding and conceptualization implying conceptual model of a domain [1]. Ontologies are views of the world that tend to evolve rapidly over time and between different applications. Currently, ontologies are often developed in a specific context with a specific goal in mind. However, it is ineffective and costly to build ontologies for each new purpose from scratch, which may cause a major barrier for large-scale use in knowledge markup in Semantic Web [5]. Different approaches were used for building ontology and most of them are manually [1] and some of them are (semi-) automatic [2].

Ontology learning (ontology extraction, ontology generation, or ontology acquisition) is the automatic or semi-automatic creation of ontologies, including extracting the corresponding domain's terms and the relationships between the concepts that these terms represent from a corpus of natural language text, and encoding them with an ontology language for easy retrieval. The process consists the following eight tasks [3].

- ✓ **Domain terminology extraction**: during the domain terminology extraction step, domain-specific terms are extracted, which are used in the following step (concept discovery) to derive concepts
- ✓ **Concept discovery**: in the concept discovery step, terms are grouped to meaning bearing units, which correspond to an abstraction of the world and therefore to concepts.
- ✓ **Concept hierarchy derivation**: in the concept hierarchy derivation step, the ontology learning (OL) system tries to arrange the extracted concepts in a taxonomic structure
- ✓ **Learning of non-taxonomic relations**: At the learning of non-taxonomic relations step, relationships are extracted, which do not express any sub- or super-assumption

- ✓ **Rule discovery**: here axioms (formal description of concepts) are generated for the extracted concepts
- ✓ **Ontology population**: at this step, the ontology is augmented with instances of concepts and properties
- ✓ **Concept hierarchy extension**: in this step, the OL system tries to extend the taxonomic structure of an existing ontology with further concepts
- ✓ **Frame and event detection**: in this step the OL system tries to extract complex relationships from text, e.g., who departed from where to what place and when

These tasks are not all necessarily applied in every ontology learning system. The last two tasks are most of the time optional. Hence ontology learning in this work is concerned with automatic creation of ontology from software source code which includes domain terminology extraction, concept discovery, taxonomic and non-taxonomic relations and rules or properties. The core component of any software is its source code. Any bugs that occur while testing or running the software occur within source code. So, source code ontology models the software source code in terms of the contained software objects, such as modules, packages, functions, namespaces, variables and database objects. Due to the large amount of source code available in a software, building an ontology for software source code manually is time consuming and intensive. Therefore, a need for automatic ontology learning is critical in the domain [1, 4, 5].

In addition, source code is a dynamic resource in software project. There is a need for the ontology to grow and update while the code changes. Knowledge graph [6] is a graphical, smart and living representation of knowledge which has a flexible structure: the ontology can be extended and revised as new data arrives. Knowledge graphs are also able to capture diverse meta-data annotations such as provenance or versioning information, which make them ideal for working with a dynamic dataset. Hence, it can be used to represent the evolving resource i.e., source code ontology.

1.2 Motivation

Semantic web is designed to make information available for both computers and human. Ontology is the conceptual backbone that provides meaning to data on the Semantic Web. By

using ontologies of software source code, the software developers can be supported in their interaction, bug resolution, testing and integrating the software. The manual ontology construction is ineffective and costly. Therefore, the main motivation to this work is to provide software developers and software engineers with a method to automatically generate a standard ontology from the code.

1.3 Statement of the Problem

Software developers who are engaged in developing open-source software tools face different types of challenges. When an open-source project becomes larger it becomes difficult for a core contributor to review each and every code request submitted. Very quickly this becomes a bottleneck for the entire project and slows the progress and growth of the software. Implementing peer review is the most common practice for fixing this bottleneck. This process requires other developers to understand the mission of the project and the quality to be achieved from all submissions [7]. Also, in open-source software, no one really knows who initiated, designed and created the product. This leads to wasted labor and reduced productivity. The other problem software developers experience is while testing and bug resolution. Thus, someone who has expertise in the area of the bug, but has never modified a file or submitted a patch, is likely to have the in-depth knowledge of the code required for fixing a bug. This was shown in [1] focusing on the online communities. In general, software developers need a better tool to help them in testing and maintaining their system, integrating plugins, reusing codes by peers.

Different researchers tried to implement ontology on software source code in order to overcome some of these problems. The work in [1] tried to show the need of semantic website based on source code ontology for online community. Their approach is aimed to help bug resolution for online software developers. Although they have developed a particular method to automatically extract and annotate Web information with respect to the ontologies, their work depends on manually created ontology of the domain. Building ontology for software source code manually is time taking and labor intensive. And also, their work focusses more on bugs and community ontologies. The task of understanding code was not given attention.

Genapathy and Sagayaraj [5] proposed to solve the problem of building ontology manually by making a method to automatically create ontology by extracting metadata from the Java source code. However, the ontology created in this approach can represent the metadata of source code not the source code itself. By generating ontology for source code, it will be effective to reuse, search and extract components of the software. And also, rather than using different APIs in extracting ontology for even a metadata it will be effective using machine learning for discovering relations and rules.

The work in K. Bontcheva and S. Marta [2] also proposed to automatically generating ontology from different software artifacts. They have presented an approach to learn ontology from multiple sources i.e., source code, user guide, and discussion forums. But their work lacks the necessary tasks in ontology learning which are discovering taxonomic relations, non-taxonomic relations, rules and properties of concepts. In addition, their approach fails to deal with the dynamic nature of source codes, which needs ontology versioning and evolution (i.e., Software source code is not a static resource and may change over time). This change often leaves the ontology in an undefined or inconsistent state. It is thus very important to have a method to preserve the code and its ontology.

F. Jiomekong Azanzi & C. Gaoussou [8] proposed a method based on HMM (Hidden Markov Models) to extract knowledge from the source code for ontology enrichment. Although their work can be as a good input for ontology learning which simplifies the first and most important task; domain term extraction, their work cannot be complete ontology learning because the terms extracted need to be mapped to concepts and relations and should be populated in to ontology automatically. The relations (terms) discovered are only hierarchical and programmer defined; missing the relations hidden in the source code and also this work lacks a good evaluation method.

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] and Section 2.5 of this study. Therefore, researchers have been attempting to develop an approach for learning ontology from source

code. The main important works are presented in the above Sections of this Chapter. However, there is still a gap in designing the architecture for ontology learning from source code and most of the researchers have experimented their work on Java source code. These was believed to be because of the dominance of Java language at the time of the studies but now programming languages like Python are appearing to take over the others specially in the field of Machine Learning and Artificial Intelligence.

In addition, evolution and versioning have been left as open problem through researches in source code ontology development. 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 an approach for knowledge graph construction based on ontology from python source code.

1.4 Objectives

General Objective

The general objective of this work is to construct knowledge graph based on ontology from python software source code.

Specific Objectives

The following specific objectives will help to accomplish the general objective stated above.

- ✓ Assess different techniques and approaches employed so far in ontology learning and knowledge graph construction;
- ✓ Understand the behavior of python source code and its structure;
- ✓ Select an appropriate approach for ontology learning that fits to the behavior of python source code;
- ✓ Design architecture for ontology learning and knowledge graph construction;
- ✓ Prepare corpus for ontology learning and knowledge graph construction;

- ✓ Develop a prototype that demonstrates the proposed solution; and
- ✓ Test and evaluate the capability of the system

1.5 Methods

Since this work is aimed at designing an effective solution for a given problem, design science [9] will be used as the main methodology for this work. Therefore, this research will be conducted based on the following high-level steps of design science research methodology.

Problem identification: in which problem will be identified and objectives for the solution will be defined. It includes activities like: identify problem, literature review, expert interviews, pre-evaluate relevance.

Solution design: this includes designing and developing the solution/system/artifact. It includes steps like: develop a system architecture, analyze and design the system, build the System.

Evaluation: the whole system developed will be observed and evaluated. In other words, Case study and results summarization will be conducted. An open-source project which have an ontology developed by domain experts or a suitable project for future manual development will be used to evaluate the proposed ontology learning method. This method of evaluation is called gold standard evaluation. And relevance of the proposed approach will be evaluated by domain experts based on a given criteria of ontology evaluation. Therefore, an experiment will be done on a project to be selected based on its relevance to the study. Open-source software will be a focus area as source of data for experimenting this work.

1.6 Scope and Limitations

This research study is limited to designing and developing an ontology learning method from software source code. The learning tasks including domain terminology extraction, concept extraction, creating concept hierarchy or learning taxonomic relations and non-taxonomic relations are under the scope of this work. In addition to this, the proposed work is supposed to be able to learn rules or properties of the concepts, concept hierarchy extension and frame and event detection.

1.7 Applications of Results

The main application area of the result would be any software development platform especially online problem-solving community. Ontologies learned from source code could be used as a backbone in developing a semantic site for software developers. It can be delivered as a better tool for exploring, guiding and manipulating software source code for developers. Since it is aimed to focus on the open-source projects knowing that open-source software has so many challenges, it would be applied in semantic web sites developed for open-source projects. An ontology-based access to code would enhance the process of integrating, testing and fix issues in their code or product. Even though the focus group of this work is the open-source software developers, an ontology generated from source code can help any software developers in manipulating their work.

1.8 Thesis Organization

The rest of this thesis is organized as follows. Chapter 2 presents literature related to ontology learning and knowledge graph construction as a foundation of this research work. Chapter 3 presents related research works in the area of text data, ontology learning from source code, and knowledge graph construction. The proposed architecture is presented in Chapter 4 by discussing each components of the architecture and summarizing the overall ontology learning workflow. The experimentation of the proposed solution is presented in Chapter 5 with evaluation of the ontology learner. Finally, Chapter 6 concludes the main contributions of this study and suggests an outline for the future works.

Chapter Two

Literature Review

2.1 Introduction

In order to make a deep look on the area of Ontology learning and knowledge graph construction, this chapter presents an overview of the two basic concepts and their definition. So, we will attempt to give an introduction to ontology, basic components of ontology and the advantages of using ontology over the traditional knowledge base in Section 2.2. Then an overview of ontology learning, basic steps or tasks in ontology learning, some learning techniques in hand and ontology learning evaluation will be discussed in Section 2.3. And section 2.4 will provide an overview of Knowledge graph, construction and some applications of knowledge graph. Finally, Section 2.5 is about software source code as data source for learning.

2.2 Ontology

As introduced in the first chapter, ontology is a formal explicit specification of shared conceptualization of a domain. Ontologies in computer science are usually regarded as formal representations of knowledge – often restricted to a particular domain. In our days, ontologies play a central role in knowledge integration and semantic representation. In addition to data integration, reasoning and querying scenarios, ontologies are also a means to document the structure of a particular domain, which helps to develop a common understanding of its concepts [16].

An ontology must be formal and, therefore, machine readable. This way ontologies can provide a common vocabulary between various applications. This knowledge representation structure usually consists of a set of frame-based classes organized hierarchically describing a domain. It provides the skeleton of the knowledge base. The knowledge base uses the represented concepts to describe a dynamic reality and it changes according to the changes in the reality [16].

2.2.1 Components of Ontology

An ontology is composed, on one hand, by concepts, taxonomic relationships (that define a concept hierarchy) and non-taxonomic relationships between them and, on the other hand, by concept instances and assertions about them [10]. More formally, an ontology can be defined as a tuple as shown in Equation 2.1:

$$O = (C, R, H^C, A) \quad (2.1)$$

- **Concepts:** C in the above tuple is the set of ontology concepts. The concepts represent the entities of the domain being modeled. They are designated by one or more natural language terms and are normally referenced inside the ontology by a unique identifier.
- **Properties:** are used to describe the characteristics of Individuals of a concept. They are composed of attributes properties and relations. A set of taxonomic relationships between the concepts given as H^C defining the concept hierarchy and R the set of non-taxonomic relationships.
- **Axioms**: A in the above tuple is a set of axioms, usually formalized into some logic language. These axioms specify additional constraints on the ontology and can be used in ontology consistency checking and for inferring new knowledge from the ontology through some inference mechanism.
- **Rules:** are a set of statements used to infer new knowledge.
- **Individuals**: are the instances of the concepts and relationships, e.g., the instances of the elements of C, H^C and R.

2.2.2 Advantages of Ontology

One of the main features of ontologies is that, by having the essential relationships between concepts built into them, they enable automated reasoning about data [11]. Such reasoning is easy to implement in semantic graph databases that use ontologies as their semantic schemata. What's more, ontologies function like a 'brain'. They 'work and reason' with concepts and relationships in ways that are close to the way humans perceive interlinked concepts.

In addition to the reasoning feature, ontologies provide a more coherent and easy navigation as users move from one concept to another in the ontology structure. Another valuable feature is that ontologies are easy to extend as relationships and concept matching are easy to add to existing ontologies. As a result, this model evolves with the growth of data without impacting dependent processes and systems if something goes wrong or needs to be changed. Ontologies also provide the means to represent any data formats, including unstructured, semi-structured or structured data, enabling smoother data integration, easier concept and text mining, and data-driven analytics.

The main advantages of using ontology [16] over the general knowledge bases are:

- ✓ Formal or machine-readable representation, no natural language descriptions inside;
- ✓ Full and explicitly described vocabulary of the domain, together with explicit semantics of the concepts;
- ✓ Full model of the domain, including redundant for the task relations between the concepts;
- ✓ A consensus knowledge, or common understanding of the domain;
- ✓ Developed specially for being shared and reused;
- ✓ Web-based.

2.3 Ontology Learning

Madche and Staab [21] introduced the term "ontology learning" for a newly emerging field of research aiming at nothing less than the automatic generation of ontologies. When we see different works on the area, they were based on ripple down rules, word sense clustering, and information extraction.

Ontology learning [15] refers to the extraction of ontological knowledge from unstructured, semi-structured or fully structured data sources, and building an ontology from them with little or no human intervention. It has grown from the early methods focusing on the extraction of concepts from structured data to the complex learning algorithms to extract taxonomic non-taxonomic and rules from the semi or unstructured source of data. Although their methods do

not automate every task in ontology learning, different researchers have been trying to explore and deal with ontology learning.

Ontology learning have been researched over several possible sources of knowledge such as unstructured, semi-structured and structured sources [11]. Unstructured sources contain data which has no predefined organization. Textual resources (Web pages, manuals, discussion forum posting, specification, analysis and conception document, source code comments) and multimedia contents (videos, photos, audio files) are all unstructured sources. Structured data sources contain data described by a schema. The advantage of using structured data sources is that they contain directly accessible knowledge. Knowledge bases, database schema and ontologies are some examples of structured data sources. And, Semi-structured data sources contain data having a structure that already reflect part of the semantic interdependencies which makes schema extraction simple. Knowledge sources like XML (Extensible Markup Language), Source Code, UML meta models and Entity-relation diagrams are categorized as semi-structured.

From the Ontology engineering point of view, there are several methodologies for constructing ontologies from scratch. The main tasks in ontology learning are discussed in the next section.

2.3.1 Ontology Learning Process

As mentioned by several authors [4,10,13,25], the main steps and knowledge acquisition techniques employed for building domain ontologies are: Extraction of terms that represent domain concepts, construction of an initial taxonomy of concepts using is-a relations, identification and labelling of non-taxonomic relations, ontology population by the detection of instances for the discovered concepts, treating semantic ambiguity (mainly polysemy and synonymy) in order to improve the quality of the results and evaluation of the obtained results.

The general layer cake [12] of domain ontology learning is shown in the figure below. Each step in the general layer cake is discussed briefly in the following section.

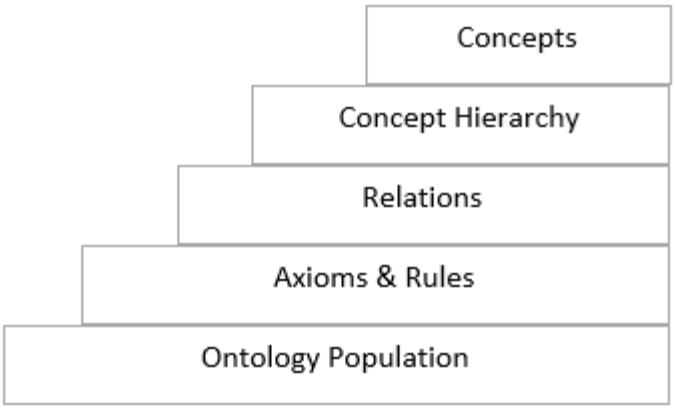

Figure 2.1: *Ontology Learning Process (The Layer cake)*

1) Discovering concepts and taxonomic relationships.

The first step in construction of ontology is to extract initial base elements for a given domain or application. This knowledge, then, can be used in the next most important and difficult learning steps. Extracting concepts and their taxonomy is the common point of start of many learning methodologies.

There exist many approaches for performing this task. As Paul Buitlar et al. [10] mentioned in their work, term extraction from a text data implies more or less advanced levels of linguistic processing. But for some other semi-structured data, different techniques may be needed to extract terms and identify concepts representing a cluster of data in a domain. Bontcheva et al. tried to retrieve concepts from multiple sources which are structured, semi-structured and unstructured sets of data. For example, the special code tokenizer, English Morphological analyzer, and phrase extractor based on TF.IDF (term frequency/inverted document frequency) were used to extract terms from source code. More recently, Azanzi & Gaoussou [11] introduced a method based on HMM (Hidden Markov Model) to automate concept identification from software source code.

The other important task of the learning process is the creation of an initial basic taxonomy of terms that will relate with is-a relationships the concepts that are representative for the searched domain. Moreover, if individualities (concretely, named entities) for a specific

concept are found, they will be considered as instances of the corresponding classes. The use of HMM [11] can again be mentioned as method of this task.

2) Non-taxonomic relations

Many ontology learning techniques have been developed, but most of these approaches focus on the extraction of concepts and taxonomic relationships, and often ignore the importance of other relations between concepts. Even though taxonomic knowledge is certainly of utmost importance, major efforts must be given to the definition of non-taxonomic conceptual relationships.

In general, two tasks need to be performed [13]. First, we have to detect which concepts are related. Second, we have to figure out how these concepts are related; thus, a name for the relation has to be found. This is specified by a verb in most cases. In fact, the role of the verb as a central connecting element between concepts is undeniable. Verbs specify the interaction between the participants of some action or event by expressing relations between them. In parallel, it can be argued, from an ontology engineering point of view, that verbs express a relation between two classes that specify the domain and range of some action or event.

3) Rule discovery

The extraction of rules [11] is most probably the least addressed researched area in ontology learning. Further, the recent challenge has strongly increased the awareness of the problem of deriving lexical entailment rules and lead many researchers to address the problem, so that a plethora of approaches to tackle the problem of learning ontological rules from text corpora can be expected in the near future. The problem was first introduced in 2001 by Lina et al [14]. and the authors proposed an unsupervised learning method for extracting rules from textual corpus. Some other methods using an ontology of rules have also been a solution for this. Azanzi & Gaoussou [11] in 2017 used a method based on HMM (Hidden Markov Model) to extract candidate words and phrases of rules from the software source code.

4) Ontology population

Ontology population commonly refers to the extraction of instances of ontological concepts from text. From the philosophical point of view, the distinguishing between a specialization of a certain concept (subclass) or a particular individual (instance)can represent a matter of discussion. In general, one has to define specifically which the instances –real world entities- in a particular ontology are (e.g. persons, organizations, events, etc.). In any case, there is a wide agreement in considering named entities as instances [16].

2.3.2 Ontology Learning Techniques

Ontology learning have been done by using different techniques depending on the behavior of data. As discussed in the first sections the types of data can be structured, semi-structured or unstructured. So, the technique we use for the learning may depend on the type of data we use. Shamsfard and Barforoush [28] proposed a classification of learning techniques by considering statistics, symbolic and multi-strategies each to be discussed below.

A. Statistic Based Techniques

Statistical analysis is the science of collecting, exploring and presenting large amounts of data to discover underlaying patterns and trends. Statistical analysis for ontology learning is performed on data gathered from input, to build a statistical model [29]. Several statistical methods for extracting ontological knowledge have been identified in the literature [30]. In this section, we present some static based methods.

1) Co-occurrence and collocation analysis

In linguistics, co-occurrence refers to an above-chance frequency of occurrence of two terms from a text corpus alongside each other in a certain order. Corpus linguistics and its statistical analysis reveal patterns of co-occurrence within a language and enable to work out typical collocations for its lexical items.

Co-occurrence and collocation are related and can be defined as the occurrence of some words in the same sentence, paragraph or document. This occurrence hints a potential direct relation between these words.

2) Clustering:

Clustering [19] defined as the process of grouping a set of physical or abstract objects into classes of similar objects. A cluster is a collection of data objects that are similar to one another within the same cluster and are dissimilar to the objects in other clusters.

A cluster of data objects as shown in [15] can be treated collectively as one group and so may be considered as a form of data compression. Although classification is an effective means for distinguishing groups or classes of objects, it requires the often-costly collection and labeling of a large set of training tuples or patterns, which the classifier uses to model each group. It is often more desirable to proceed in the reverse direction: First partition the set of data into groups based on data similarity (e.g., using clustering), and then assign labels to the relatively small number of groups. Additional advantages of such a clustering-based process are that it is adaptable to changes and helps single out useful features that distinguish different groups.

Clustering tools [16] assign groups of records to the same cluster if they have something in common, making it easier to discover meaningful patterns from the dataset. Clustering often serves as a starting point for some supervised DM techniques or modeling.

In ontology learning, clustering is generally used to create groups of similar words which can be regarded as representing concepts, and further order these clusters hierarchically. This technique is generally used for concepts learning by considering clusters of related terms as concepts, and learning of taxonomy relations by ordering these groups hierarchically [11].

3) Hidden Markov Model (HMM)

A Hidden Markov Model is a generative statistical model used to generate data sequences according to probability distributions [7]. It is generally used for pattern recognition, automatic voice processing, automatic natural language processing, character recognition and musical genres classification. Maedche and Staabused the n-gram technique based on HMMs to process documents on the morphological level before supplying them to terms extraction tools [6]. Azanzi & Gaoussou [11] used this method to extract knowledge from software

source code of EPICAM a tuberculosis surveillance system. They have defined different HMM structure to extract concepts, relations, rules and axioms.

4) Conditional Random Fields (CRF)

Conditional Random Fields (CRFs) first introduced in [17] are undirected graphical models that encode a conditional probability distribution using a given set of features. CRFs are a class of statistical modeling methods often applied in pattern recognition and machine learning, where they are used for structure prediction. Whereas an ordinary classifier predicts a label for a single sample without regard to "neighboring" samples, a CRF can take context into account.

The reason why CRFs are more effective than HMMs is that CRFs use the conditional probability property instead of the independence assumption mainly used in HMMs. CRFs also avoid label bias problems and avoid the weaknesses of other Markov models derived from MEMMs and graphic models. CRFs show better performance than MEMMs and HMMs in bioinformatics, computational linguistics, and voice recognition. CRFs are also used for the prediction and analysis of labels for data in natural language writing. Features can be chosen randomly, and they are to be normalized to obtain solution.

B. Symbolic Based Techniques

Symbolic method is the term for the collection of all methods in research that are based on high-level symbolic (human-readable) representations of problems, logic and search. The approach is based on the assumption that many aspects of intelligence can be achieved by the manipulation of symbols. A symbolic method can be rules based, linguistic based or pattern based [11].

1) Rule based

A rule-based model is usually represented as a set of rules, where each rule consisted of a condition testing and an action execution. It can be based on logical and association rules discussed below.

Logical rules: Logic-based learning is a particular area of Artificial Intelligence defined at the intersection between knowledge representation and machine learning, which involves the automated construction of logic-based programs from examples and existing domain knowledge.

Logic based learning methods may discover new knowledge by deduction (deduce new knowledge from existing ones) or induction (synthesize new knowledge from experience) and represent these knowledges by propositions. Inductive logic programming can for example used to learn new concepts from extensional data [28]. Pivk et al. present an approach for automatically generation of F-Logic frames out of tables which support the automatic population of ontologies from tables [28].

Association rules: Association rule mining is a rule-based learning method for discovering interesting relations between variables in large set of data. it is intended to identify strong rules discovered in database using some measures of interestingness.

The aims of association rules are to find correlations between items in a dataset. This technique is generally used to lean relations between concepts [22], [25] and can be used to recognize a taxonomy relation or to discover gaps in conceptual definitions [26].

2) Linguistic based

Linguistic approaches (syntactic analysis, morpho-syntactic analysis, lexicon-syntactic pattern parsing semantic processing and text understanding) are used to derive knowledge from natural language text. This technique can be used to derive an intentional description of concepts in the form of natural language description [11].

3) Pattern Based / Template Driven

The field of pattern-based learning [18] or pattern recognition is concerned with the automatic discovery of regularities in data through the use of computer algorithms and with the use of these regularities to take actions such as classifying the data into different categories. This approach allows to search for predefined keywords, template or patterns. A large class of

entity extraction tasks can be accomplished by the use of carefully constructed regular expressions (regex). But robust extraction requires the use of complex expressions. This effort is generally reduced by regular expression learning.

C. Deep learning

Deep learning methods [19] have also been used in ontology learning in different researches. This method is concerned with using deep learning neural networks like Convolutional Neural Network (CNN) and Recurrent Neural Network (RNN) for ontology extraction. The methods have mostly been used for NER (Named Entity recognition) or domain terminology extraction step of ontology learning.

Bi-Directional LSTM

A Bidirectional LSTM [20], or Bi-LSTM, is a sequence processing model that consists of two LSTMs: one taking the input in a forward direction, and the other in a backwards direction. Bi-LSTMs effectively increase the amount of information available to the network, improving the context available to the algorithm. Figure 4.5 shows our model for relation extraction. By running information forward and backward Bi-LSTM helps to find relations like this from code elements. For example, a method "my-function" and its return type can be related as: returns (my-function, List).

Given a sentence/code file X = (x_1, x_2, ..., x_T), the words are projected into a sequence of word vectors, denoted by (e_1, e_2, ..., e_T) where T is the number of words in a code. These word vectors are put to the recurrent layer step by step. For each step t, the network accepts the word vector e_t and the output at the previous step h_{t-1}^{fw} as the input, and produces the current output h_t^{fw} by a linear transform followed by a non-linear activation function of forward LSTM, is given by Equation 1.

$$h_t^{fw} = \tanh(W_{fwet} + U_{fw}\, h_{t-1}^{fw} + b_{fw}) \quad (2.2)$$

where $h^{fw} \in R^M$ is the output of the RNN at the t^{-th} step, which can be regarded as local segment-level features produced by the word segment (x_1, ..., x_t). Note that M is the dimension of the feature vector, and $W_{fw} \in R^{M \times D}$, $U_{fw} \in R^{M \times M}$, bfw $\in R^{M \times 1}$ are the model parameters.

Bi-LSTM adopted from the work of [20] prediction at step t is obtained by simply concatenating the output of the forward RNN and the backward RNN, formulated as shown in Equation 2.

$$h_t = h_t^{fw} + h_t^{bw} \tag{2.3}$$

where $h_t^{bw} \in R^M$ is the output of the backward RNN, which possesses the same dimension as h_t^{fw} defined by Equation 3:

$$h_t^{bw} = \tanh(W_{bwet} + U_{bw}\, h_{t-1}^{bw} + b_{bw}) \tag{2.4}$$

Where the model parameters are the same as forward RNN and they are trained simultaneously.

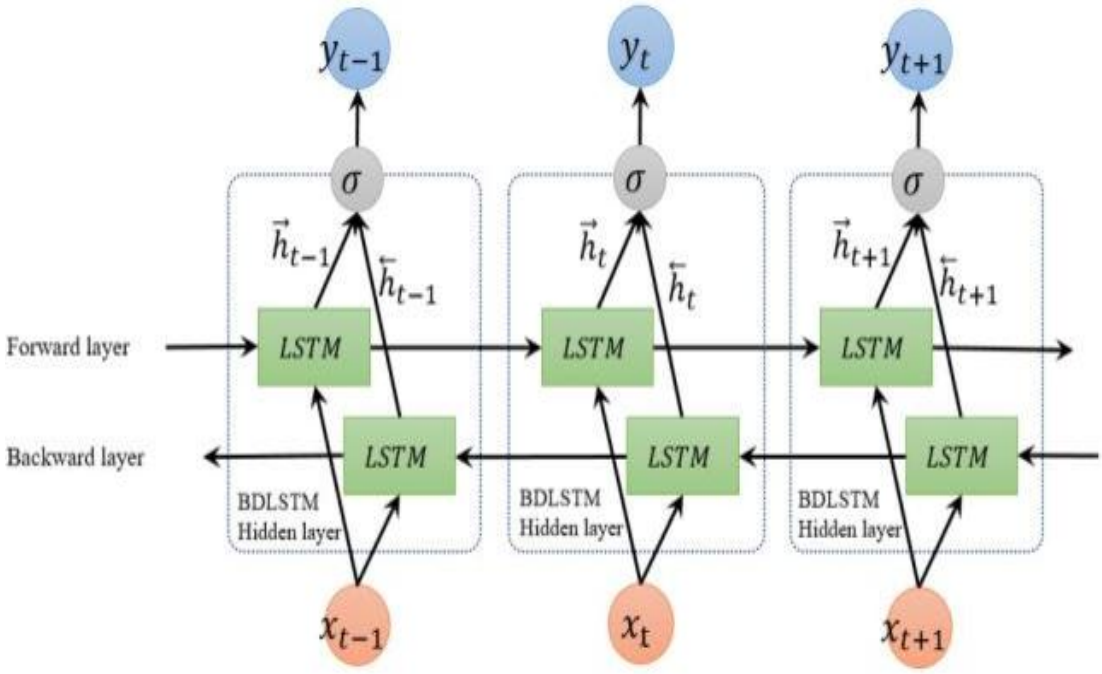

Figure 2.2: *Bi-LSTM model*

D. Multi Strategy learning

Multi Strategy learning (MSL) is concerned with developing learning methods and systems that integrate different inferential or representational strategies in solving a given task of learning. In ontology learning, MSL leverage the strengths of the above techniques to extract many types of ontological knowledge from different types of resources. Maedche and Staab [10] present for example, the use of clustering for concepts learning and association rules learning to learn relations between these concepts [16]. The other important example of multi strategy learning is a combination of statistical CRF with RNN network Bi-LSTM for ontology learning. This approach has been shown to effective in sequence labeling by combining the power of both methods.

2.3.3 Ontology Learning Evaluation.

In order to prove the quality of the results obtained by the ontology learning process, an evaluation phase is mandatory. A properly evaluated structure will not guarantee the absence of problems, but it will make its use more reliable.

The evaluation of automatically obtained ontologies is recognized to be an open problem. Many of the researchers on the area of ontology learning have been challenged on the evaluation of their ontology. This may be because of the lack of evaluation tools and methods. Even though efforts have been made on this area, the results are available on the methodological rather than the experimental side. For example, Azanzi & Gaoussou [11] tried to evaluate their method by comparing the number of concepts, relations and axioms they have obtained with their corresponding number in the meta model ontoEPICAM. However, Gomez-Perez and et al [24] presented the methodology for evaluating ontologies. One of the criteria they have mentioned is completeness. An ontology is said to be complete if it contains all the concepts, relations and rules and their respective definitions of the modelled domain. Evaluation can be done either by human experts manually or by comparing the resulting ontology to the ground truth or gold standards.

2.4 Knowledge Graph

The term knowledge graph has been used frequently in research and business in recent decades, in close association with Semantic Web technologies, linked data, large-scale data analytics and cloud computing. A knowledge graph (KG) is a graph-structured knowledgebase that stores knowledge in the form of the relation between entities. A knowledge graph gets and integrates data into an ontology and applies a reasoner to derive new knowledge [21].

A Knowledge Graph is a structured data (in the form of a specific data structure), which is normalized (consisting of small units, such as vertices and edges) and connected together (defined by the – possibly distant – connections between objects) [21].

The underlying basis of a knowledge graph is the ontology [22], which specifies the semantics of the data. An ontology is typically based on logical formalisms which support some form of inference: allowing implicit information to be derived from explicitly asserted data. Some of the information inferred can be otherwise hard to discover.

Knowledge graph, in mathematical sense, allow for the application of various graph-computing techniques and algorithms such as, shortest path computations, or network analysis, which add additional intelligence over the stored data [21].

knowledge graphs have a flexible structure [23]: the ontology can be extended and revised when new data arrives. This makes it efficient to store and manipulate data in a knowledge graph if you have use cases where regular updates and data growth are important, particularly when data is arriving from diverse, heterogeneous sources. A knowledge graph can support a continuously running data pipeline that keeps adding new knowledge to the graph, changing it as new information arrives.

Knowledge graphs are able to capture different meta-data annotations such as versioning information. This make them ideal for working with a dynamic dataset [23]. There is an increasing need to account for the provenance of data and include it so that the knowledge can be assessed by its consumers in terms of credibility and trustworthiness. A knowledge graph can answer what it knows, and also how and why it knows it.

The knowledge graphs are characterized by properties that distinguish them from the other knowledge management paradigms [24]:

- ✓ **Normalization**: Information is decomposed into small units of information, interpreted as edges of some form of graph.
- ✓ **Connectivity**: Knowledge is represented by the relationships between these units.
- ✓ **Context**: Data is enriched with contextual information to record aspects such as temporal validity, provenance, trustworthiness, or other side conditions and details.

Moreover, knowledge graphs are typically characterized as follows [21]:

- ✓ **Explicit** (created purposefully with an intended meaning)
- ✓ **Declarative** (meaningful in itself, independent of a particular implementation or algorithm)
- ✓ **Annotated** (enriched with contextual information to record additional details and meta-data)
- ✓ **Non-hierarchical** (more than just a tree-structure)
- ✓ **Large** (millions rather than hundreds of elements)

2.4.1 Knowledge graph construction

The basic unit for the knowledge graph is composed of "Entity−Relation−Entity" triples as shown in Equation 2.5 [6].

$$G = (E, R, S) \tag{2.5}$$

E is the set of entity in a knowledge base, different entity types represented by |E| which is shown in Equation 2.6, R is the set of relation in a knowledge base, different relationship types represented by |R| as shown in Equation 2.7 and S represents the set of triples in a knowledge base as shown in Equation 2.8.

$$E = \{e_1, e_2, \ldots e_n\}. \tag{2.6}$$

$$R = \{r_1, r_2, \ldots r_n\}. \tag{2.7}$$

$$S \subseteq E*R*E \tag{2.8}$$

Where e1, e2,….en are entities in a knowledge base, r1, r2,…rn are relations in a knowledge base and E,R,E represents entity-relation-entity triples in a given knowledge base.

Therefore, construction of knowledge graph from any unstructured or semi-structured data can be achieved by the following three steps:

I. Entities extraction

This step is also known as named entities recognition which refers to automatic extraction of entities from structured (relational databases, XML), semi-structured (Source code) and unstructured (text, documents, images) data sources. NER has been evolved from rule-based approaches to statistic machine learnings [25].

- ***Rule-based approach:*** this approach is by manually constructing some set of rules and mining strings/terms from the data source based on that rule. Later, researches have been conducted on automatic construction of rules using machine learning.
- ***Machine learning-based approaches:*** these approaches include tasks like model selection, model training and feature extraction. Some of the machine learning methods of entity extraction available are: heuristic algorithm, conditional random field (CRF) and

clustering algorithm. However, it is difficult to obtain higher learning accuracy with these methods. Therefore, recent research works provided an efficient learning accuracy by using Convolutional Neural Networks (CNN). The deep learning methods using RNN can achieve extraction of relation as well.

II. Relation extraction

Once entities are extracted through information extraction, the next step is mining relations for edge construction. Relation extraction in early years depended on rule-based methods as entity extraction. Lately researches have been conducted on relation extraction and we classified the methods proposed through researches based on the supervision [24] as follows:

- ✓ **Supervised learning methods:** Supervised learning methods apply the idea of classification on human-annotated data. The extraction model is trained on manually labeled training data corpus for relation extraction and Once trained, these methods can recognize entities through a matching process and extract specific relations. Supervised learning for relation extraction can be divided into two main categories: feature vector-based methods and kernel-based methods.
- ✓ **Semi-supervised methods:** these methods apply two additional steps compared to the above supervised approaches. The first is to pre-design a few relation types. The second is to incorporate appropriate entity pairs as seeds into training sets. These methods alleviate dependence on huge amounts of labels.
- ✓ **Domain-independent methods:** Domain-independent methods relax the need for domain specifications, which means these methods are easy to extend and can even be applied to multiple domains. Some researchers have incorporated external knowledge bases, such as Wikipedia, to complement their methods [26, 27]. The work of [28] also proposed a framework for open information extraction and an extraction model named Text Runner. These methods assume that each pair of entities has a known relation and uses context information to construct feature representations for the entities.
- ✓ **Distant-supervised methods:** Distant-supervised methods [29, 30] generate large amounts of training data automatically by aligning unstructured text with a knowledge base. The work of [30] attempted to incorporate distant supervision into text processing to

automatically generate training samples by aligning corpus and text so as to extract a feature training classifier. They proposed a sentence-level model that selects valid instances and makes full use of the supervision information from a knowledge base.

- ✓ **Deep learning methods:** Deep learning methods have proven to be exceptionally powerful in NLP (natural language processing) and graph identification, which has inspired their use for solving relation extraction problems. The architectures of deep networks come in many flavors. There are RNNs (recurrent neural networks [30], CNNs (convolutional neural networks) [31], combined CNNs and RNNs [32], [33], and LSTMs (long short-term memories) [34].

III. Coreference Resolution

One named entity could correspond to multiple entity concepts or multiple terms corresponding to the same entity [21]. For instance, "President Trump" and "Donald John Trump" are the same person, so these two entity references should be merged before they are connected to an entity in the knowledge base. To solve this problem entity disambiguation is needed. Entity linking methods can be used to solve ambiguous entity problem. Most solutions proposed for coreference resolution have been based on machine learning method. Some assume it as classification problem and use decision trees while others assume it as clustering and train classifier for resolution.

2.4.2 Applications of Knowledge Graph

Some of the potential applications include: semantic search, automated fraud detection, intelligent chat bots, advanced drug discovery, dynamic risk analysis, content-based recommendation engines and knowledge management systems.

GRAKN.AI [23] is a database in the form of a knowledge graph that uses machine reasoning to simplify data processing challenges for AI applications. It stores data flexibly and in a way that allows machines to understand the meaning of information in the complete context of their relationships.

In the 1980s, researchers from the University of Groningen and the University of Twente in the Netherlands initially introduced the term knowledge graph to formally describe their knowledge-based system that integrates knowledge from different sources for representing natural language [21]. The authors proposed KGs with a limited set of relations and focus on qualitative modeling including human interaction, which clearly contrasts with the idea of KGs that has been widely discussed in recent years.

In 2012, Google introduced the Knowledge Graph as a semantic enhancement of Google's search function that does not match strings, but enables searching for "things", in other words, real-world objects [8].

To mention some other implementations, DBPedia, YAGO (Yet Another Great Ontology), Freebase, Wikidata, Yahoo's semantic search assistant tool Spark, Google's Knowledge Vault, Microsoft's Satori and Facebook's entity graph [16]. The applications differ in their characteristics, such as architecture, operational purpose, and technology used, which makes it difficult to find a consensus and to create a definition of knowledge graph. The lowest common denominator of the listed open-source applications is their use of Linked Data, whereas hardly any proven information is available about Satori and the entity graph.

2.5 Source code as Data for Learning

Source code is generally written according to good programming practices, including naming conventions. These practices tell programmers how to name variables, organize and present their source codes, and so on. They contain several types of files: files describing data, files processing data, user interface files and configuration files [11].

Description files: These files contain entities, rules between these entities allowing the management of the correspondence between them in a data source. They are good candidates for concepts, attributes, axioms and even rules extraction.

Files containing data processing: Located between user interfaces files and data description file, this part of source code contains data processing. They are consisting of code for user

access control, validations and data fetching. These files are good candidates for axioms and rules extraction.

Configuration files: Allow developers to specify certain information such as the type and path of a data source, different languages used by users, and so on. In this article, we do not consider this kind of files.

User interfaces: Unlike the first three file types, these files contain the words of a human-readable vocabulary that can be found in a dictionary. The User interfaces are composed of files to describe the information that will be presented to users for viewing or for recording data. They are therefore good candidates for concepts and their definition, properties, axioms and rules extraction. These files usually contain comments that can be useful when enriching an ontology.

Chapter Three

Related Work

3.1 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. But 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.

3.2 Ontology Learning from text data

A large collection of methods for ontology learning from text have developed over recent years by proceedings of various workshops in semantic web area. From the various works, we will focus on the most relevant ones for our research.

3.2.1 OntoLT

OntoLT in the work of P. Buitelaar et al. [31] is a plug-in for the widely used Protégé ontology development tool. OntoLT supports the interactive extraction and/or extension classes and relations of ontologies from a collection of text. A number of predefined mapping rules are used to map the annotated terms into ontology classes and slots. For this purpose, the plugin defines a number of linguistic and/or semantic patterns over the XML-based annotation format that will automatically extract class and slot candidates. A user can add new mapping rule either manually or by using machine learning.

OntoLT approach, provides a precondition language for mapping rules and this allows the selection of entities from annotated document. The approach works by first annotating the document semantically. They have done an experiment on English-German corpus of medical texts in the neurology domain which have 9000 abstracts and about 1 million tokens of each language and used chi-square method to extract linguistically relevant entities.

3.2.2 Text2Onto

Text2Onto [4] is a framework for ontology learning from textual resources. This approach tries to address the limitations of other learning methods by introducing a Probabilistic Ontology Model (POM) which combines different algorithms. Text2Onto combines machine learning approaches with basic linguistic processing such as tokenization or lemmatizing and shallow parsing.

3.2.3 AOL (Amharic Ontology Learner)

AOL [32] is a learning system that automatically generates domain ontology from Amharic plain text. This is the first research work conducted in the field of automatic concept extraction, relationship mining (among concepts) and formal representation of them on Ontology from Amharic text document. Amharic Ontology Learner works by accepting a set of Amharic documents focusing on a specific domain as input and generates an ontology that shows relevant and related concepts that can describe the domain.

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.

3.3 Ontology Learning from source code

3.3.1 DHRUV

Dhruv is a semantic web for OpenACS, a toolkit for open-source software community developed by Ankolekar et al. [1] . They have shown that knowledge about a software project is often spread in several different information sources such as source code, discussion messages, bug descriptions, documentation and manuals. Dhruv is based on an ontology that describes project structure. Ontologies were developed from code, bugs, interactions and community discussions. The system uses ontologies determine the location and context of given software objects and related bug reports.

Dhruv provides some important information about the bug; which is the project it belongs to, a person, and forums. However, their work depends on manually created ontology of domain, which is aimed to help bug resolution for online problem-solving community. Building ontologies manually is extremely labor-intensive and time-consuming. Developing an ontology manually from a software source code is not only time taking but depends on the level of understanding of the developer about the code. Also, this work lacks well presentation of experiment and evaluation.

3.3.2 Java Ontologies

An efficient method for automatic generation or extraction of ontology from software source code has been a critical issue. Hence, many researchers tried to work on the area to bring it to spot. 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. Even though their work has no grounded evaluation, using their framework, they have produced an OWL file which can be successfully loaded to both protégé and Altova semantics. However, the ontology created in this approach can represent the metadata of source code not the project. 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 multi-word 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. Jiomekong Azanzi & 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. They have done experiments on EPICAM, a tuberculosis surveillance system developed in Java. 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.

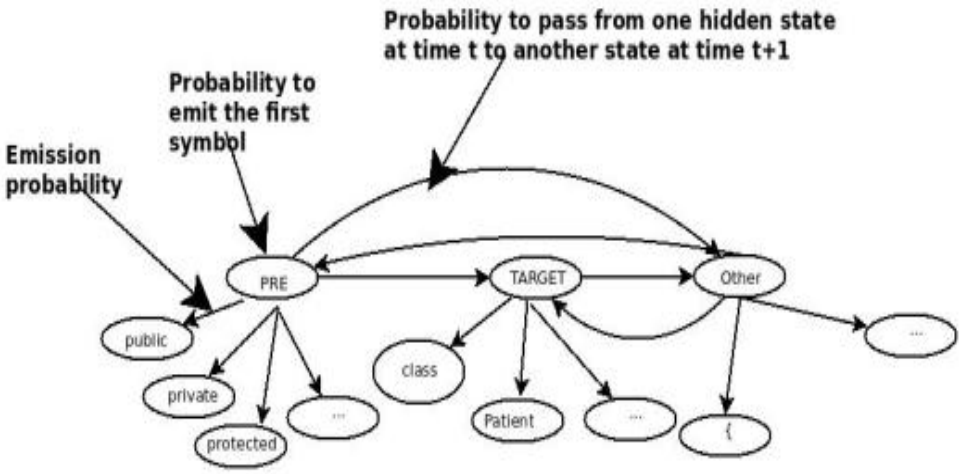

Figure 3.1: HMM example labeling the Java code elements

While very important, this work cannot be modified as it is for another programming language specially for python. This is b/c the PRE class label they have used for extraction of some keywords mostly access modifiers are not in python language. In python, access modifiers are defined using only by prefixing the variable name with underscore double and single. Therefore, a well-defined learning method which extracts features from given variables is needed for python code.

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. It is also aimed to learn the relations which are non-hierarchical and not defined by the programmer automatically. The automatic handling of evolution and versioning of the generated ontology is also the aim of this research to address. Knowledge graph construction for code based on the domain ontology is also to be addressed by this research work.

3.4 Knowledge Graph Construction

Knowledge graph has been the hottest research area since Google released its first knowledge graph as part of the search engine in 2012 [23]. Different attempts have been made in realization of this graph knowledge base which is very powerful tool for knowledge representation and inference [22, 21, 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. They experimented their method on some IT news from different online news sites. 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.

Jinhua et al. in [34] proposed Knowledge graph based on domain ontology and natural language processing technology for Chinese intangible cultural heritage. In this aper, the researchers presented that knowledge graph construction based on ontology is efficient than the others from unstructured text sources. They captured Intangible cultural heritage (ICH) data and extracted information using some NLP techniques. They achieved better result by using Att-BLSTM (attention-based LSTM) and replacing LSTM with GRU. Character-level attention-based Bi-GRU model and the sentence-level attention-based BiGRU model were all adopted to extract the Chinese language entity relation of ICH. They first built domain ontology for 24 solar terms in Chinese cultural heritage and then mapped it into knowledge graph. ID-CNNs model combined with CRF algorithm was trained to extract the relevant entities from corpus. Then BiGRU model was used to extract relation. By using this method, the researchers have extracted about 1500 entities and relations from large amounts of the 24 solar terms text data.

3.5 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] and Section 2.5 of this study. Therefore, researchers have been attempting to develop an approach for learning ontology from source code. The main important works are presented in the above Sections of this Chapter. However, there is still a gap in designing the general architecture for ontology learning from source code and most of the researchers have experimented their work on Java source code. These was believed to be because of the dominance of Java language at the time of the studies but now programming languages like Python are appearing to take over the others specially in the field of Machine Learning and Artificial Intelligence.

In addition, evolution and versioning have been left as open problem through researches in source code ontology development. 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 python 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 python source code and visualize the knowledge graph using forced graph.

Chapter Four

Ontology Learner and Knowledge Graph Construction

4.1 Introduction

Software engineers need a general way to extract ontology and construct knowledge graph from their source code in order to manage their code as data in semantic web. Thus this Chapter presents the proposed general architecture for automatic extraction of ontology from source code and knowledge graph construction by describing different components along with relevant techniques and proposed algorithms.

4.2 System Architecture

The source code ontology learner works by taking source code files as input and generates an ontology which consists the relevant concepts of the domain and their relationships. Then the information extracted from structured ontology is used to construct knowledge graph. Figure 4.1 shows the general architecture of our proposed approach. The general architecture of our approach has three main stages namely, code preprocessing, domain ontology learning and code knowledge graph construction. Each of the components are discussed in the following sections of this chapter.

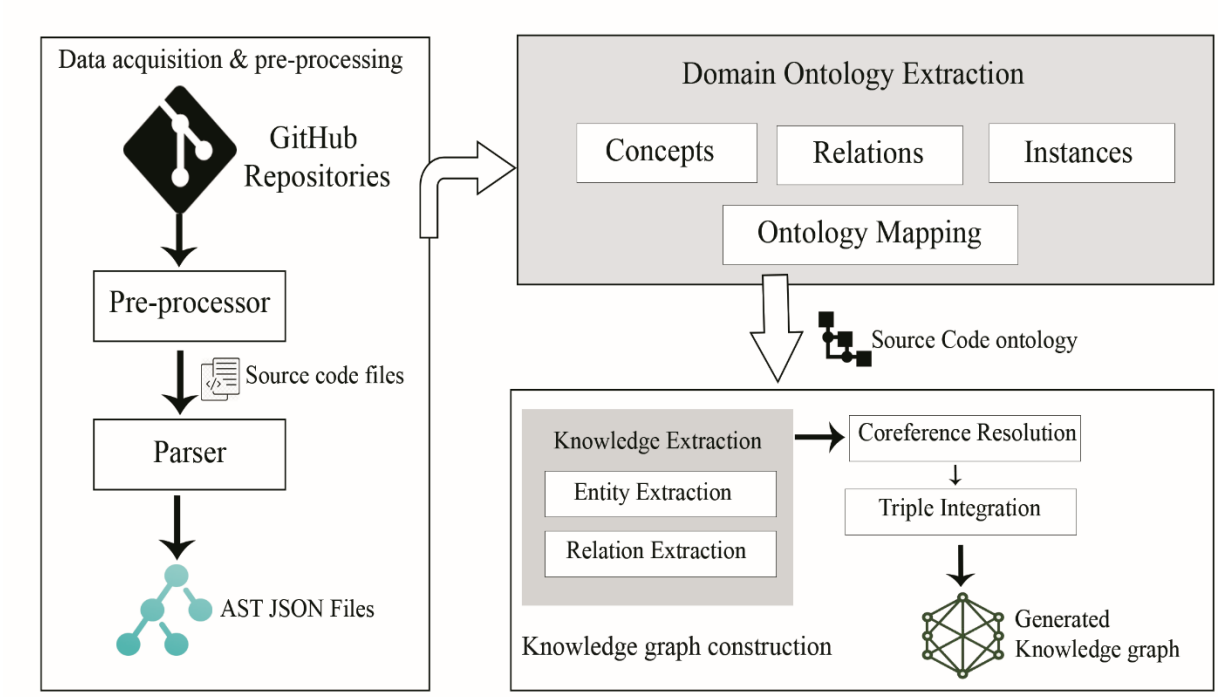

Figure 4.1: *General Architecture of Proposed System*

4.2.1 Data preprocessing and parser

Data collection was done by downloading an open source Snips-NLU project written in python from GitHub. The project is open source and licensed for validity by the NLP community. The purpose of preprocessing is to put data in a suitable form for the tools and methods to be used in the next steps. Software projects include a lot of additional files which are not part of the source code but provide some information about the project/code such as, documentations, readmes, and discussions. Therefore, in this step these types of files i.e., markdown (.md) files, YAML files, text files and copies from other projects are removed and only source code files are given to the next step. This is because there are many researches dealing with ontology learning from text data sources. The next steps in preprocessing are language detection and code validation and they are discussed in the following sections. Algorithm 4.1. shows preprocessing along with its sub tasks.

Code validation

This sub-task is also very important before extracting the candidate terms using the parser. This is because we need a valid and syntactically correct code in order to parse it using the pre-defined parsers in this work. A simple algorithm designed depending on the programming language is used to validate source code syntax of any programming language. Only valid code files should be given to parser and this helps to reduce errors during parsing step. Some programming languages have parser libraries which report error in a code during parsing and while others do not. For example, ply library in python reports error during parsing if the code is not syntactically correct.

Code Parser

Before extracting terms or words which are candidates to be concepts is parsing a code into its abstract syntax tree (AST) and representing it in a Json format. An abstract syntax tree represents all of the syntactical elements of a programming language, similar to syntax trees that linguists use for human languages. Unlike the human language, we cannot use natural language processing libraries, linguistic filtering or other methods to tokenize source code.

AST focuses on the rules rather than elements like braces or semicolons that terminate statements in some languages. These punctuations occur highly in code and its time and effort taking to deal with them. 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. Regular expressions are great for finding or validating many types of simple patterns, for example phone numbers, email addresses, and URLs. However, they fall short when applied to patterns that can have a recursive structure, such as:

- 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). The tree is hierarchical, with the elements of programming statements broken down into their parts. For example, a tree for a conditional statement has the rules for variables hanging down from the required operator.

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 are presented in Table 4.2. 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 python).

Algorithm 4.1: *Pre-processor pseudocode*

```
Input:
     Dir: Software project directory
Output:
     ASTJ – ast in json format
Begin
   1.  For file in Dir
   2.  For file in Dir
   3.    Isvalid = Code_validation(file)
   4.    If Isvalid = True
   5.       Comment_remover(file)
   6.       ASTJ = Parser(file)
   7.  Return ASTJ
   8.  Else
   9.    Return Code file is not correct syntactically
       End
```

4.2.2 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.

1. Term Extraction

This module is to handle term extraction, a very important step in ontology learning. Term extraction or information extraction in other word is the process of mining semantically relevant elements of a domain such as entities (e.g., concepts and instances). For natural languages like English, this process is handled using NER (Named entity recognition) based on linguistic or statistical techniques. NER techniques depend on the features of the natural language and these techniques are not sufficient for extracting instances of concepts or entities from source code.

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. See Annex C for code tags used in this work. Figure 4.5 shows the overall flow of term extraction process using Bi-LSTM and CRF models. 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. The model we used for term extraction is shown in Figure 4.3 and the algorithm used for term extraction is shown in Algorithm 4.2.

Since code is written sequentially by the programmer, 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 python language: in this statement the value of variable result depends on the calculated result from the right-side operation. The other example to show this dependency is class definition. 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 python that the next observation will be the name of function. The architecture of model is discussed in the following sections.

	code_index	word	label	tag	t
0	0	abstractclassmethod	CIASS	C	
1	0	ClassPropertyDescriptor	CIASS	C	
2	0	classproperty	Method	MTD	
3	0	__init__	Method	MTD	
4	0	__get__	Method	MTD	
5	0	__set__	Method	MTD	
6	0	setter	Method	MTD	
7	0	ClassPropertyDescriptor	Func_Call	FC	
8	0	type	Func_Call	FC	
9	0	isinstance	Func_Call	FC	

Figure 4.2: *Sample class labeled Data corpus*

➢ 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 python object classes. Our dataset is prepared in the format of Kaggle entity_annotated dataset [33] which has column Code-Index, word, and Tag as shown in Figure 4.4. The model consists of four layers namely, word embedding layer, Bi-LSTM Layer, CRF layer and output layer as shown in Figure 4.3. The details of model layers are discussed in the following sections.

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 embeddings
- ✓ Input_length: Length of input sequence

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

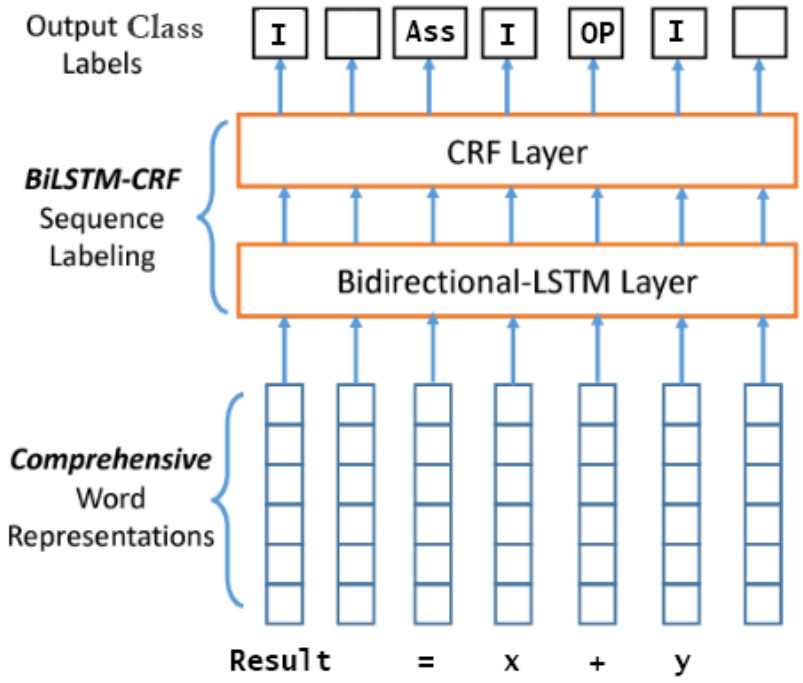

Figure 4.3: Bi-LSTM+CRF *Model*

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. This vector representation of all the label scores predicted by BI-LSTM will be given to CRF layer and in CRF the label with high square will be voted as the best answer.

CRF Layer

CRF layer is an optimization on top of BI-LSTM layer. It can be used to efficiently predict the current tag based on the past attributed tags. 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 4.5. our model is a combination of embedding, bi-lstm, and crf layers and it finally classifies terms in to their corresponding class.

Algorithm 4.2: *Term Extraction Pseudocode*

```
Input:
     C: Class Tagged Corpus,
     CF: Code files corpus
     Embedder: Embedding model
     Bi-LSTM: Bi-LSTM Model
     CRF: CRF model for tag prediction

Output: LT: List of Terms

Begin:
1.   Code_tokens = []
2.   Candidate_Concepts = []
3.   Labeled_Concepts = []
4.   Sentences = []
5.   Code_tokens = Parser(CF)
6.   Word_index = Word2Index(Embedder, N_Words, D)
7.   Tag_index = tag2index(Embedder, N_Tags, D )
8.   Term_Vector = ExtractTerms(Word_index, Tag_index, Bi-
     LSTM)
9.   For each index in Term_Vector
10.     If index ≠ 0 and index ≠ "nan"
        # index 0 indicates unknown token
11.     Candidate_Terms.add(index)
```

```
12.     End If
13.  End For

14.  For each term in Candidate_Terms
15.     Labeled_Terms.add(Predict_Label(term, CRF))

16.  TL = Normalize (Labeled_Concepts) “”” normalize Terms by
     using keras normalization method “””
17.  Return TL
     End
```

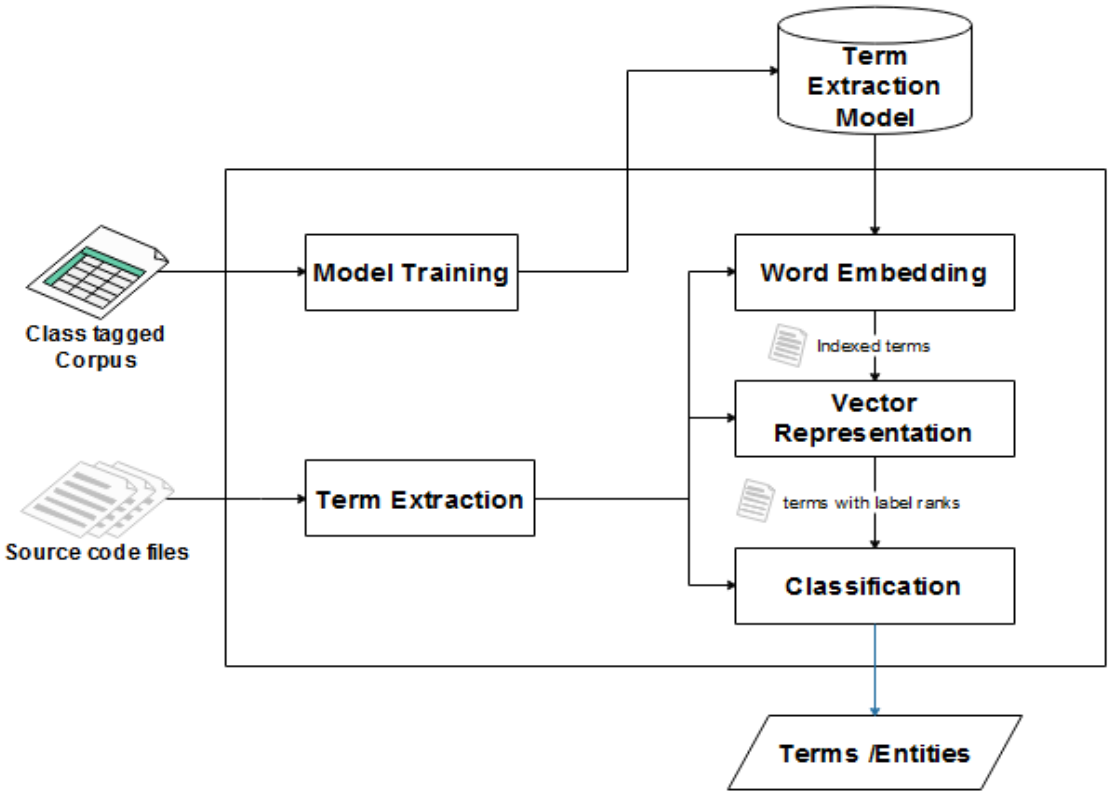

Figure 4.4: *Overall workflow of term extraction process*

2. 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. As mentioned in Chapter Two there are different approaches

which can be used to extract these elements. However, many of those studies are focusing on text data sources.

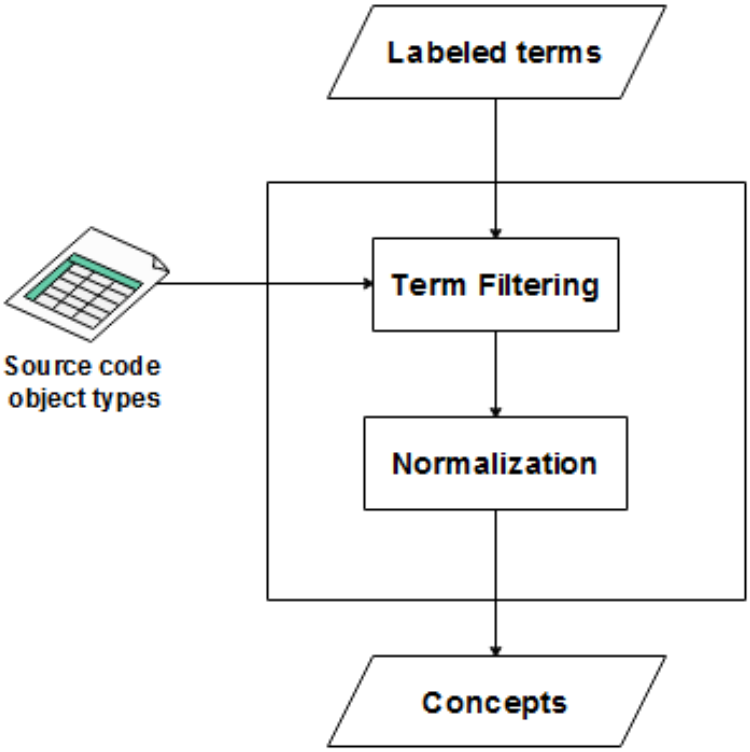

Figure 4.5: *Concept extraction flowchart*

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.

3. 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’. And both 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’s very important to keep track of the past terms and structures to detect long-distance relation patterns in a code. Figure 4.7 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 as shown in Algorithm 4.4 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.

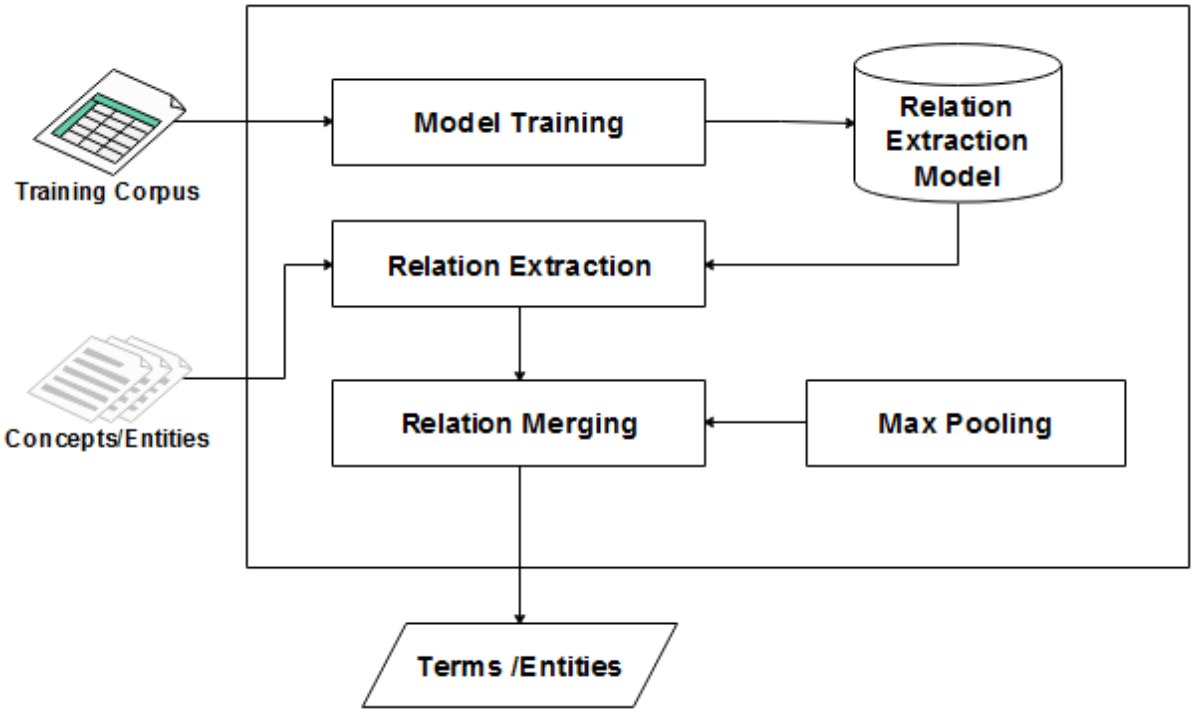

Figure 4.6: Relation Extraction workflow

Algorithm 4.3: *Relation Extraction Pseudocode*

```
Input:
     LC: List of Concepts
     R: Set of relation classes
     NC, D: No of words, Dimension
     Embedder: Embedder layer model
     Bi-LSTM: Bi-LSTM model /pre-trained
_____________________________________________________________
Output:
     Concepts with Relations
Begin:
1.      Cindex = Embedder (LC, NC, D)
2.      #Index each concept by using embedder layer
3.      For ci,cj in Cindex
4.         If index ≠ 0 and index ≠ "O"
             # index 0 indicates unknown token
5.           PR = Predict_Relation (ci, cj, Bi-LSTM)
6.         Else
7.            Add Concept to No Relation Class
8.         End If
9.      End For

10.     For each ci,cj in LC
11.        Taxonomic_Relations = Taxonomy (ci, cj, PR)
12.        Other_Relations = NonTaxonimic (ci, cj, PR)
13.     Return Taxonomic_Relations, Other_Relations
14.     End
```

4.2.3 Knowledge graph storage, retrieval 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 RDF graphs.

RDF graphs are handled using rdflib (a python 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.

Chapter Five

Experiments

5.1 Introduction

In this Chapter, we will present the experiments conducted on Snips NLU platform developed in Python. Our algorithms have been coded in Python 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.

5.2 Experimental environment settings

Execution of the proposed system was affected by several factors such as, hardware capability, Internet speed and size of dataset. Thus, experimental results and execution time would be improved using high performance hardware and parallel execution for the ontology learning tasks. Table 5.1 shows list of packages, dependencies and tools used in this experimentation.

Table 5.1: *List of tool, packages and dependencies*

Tool/package/dependency	**Version**	**Description**
Anaconda (Spyder)	3.1.2	Spyder stands for Scientific Python Development Environment included in Anaconda distribution.
Django	1.10.5	Django is a web application framework written in python used for creating user interface.

Keras	1.0.6	A Neural Network library written in Python, which is designed to be minimalistic and straight forward yet extensive that built on top of Theano and TensorFlow.
Owlready	0.3.0	Owlready is an ontology package used for domain ontology manipulation. Read and write ontology elements.
TensorFlow	2.4.1	A Python library for fast numerical computing created and released by Google. It is a foundation library that can be used to create Deep Learning models directly or by using wrapper libraries that simplify the process built on top of TensorFlow
Guesslang	2.0.1	Guesslang detects the programming language of a given source code. It supports 30 programming languages and detects the correct programming language with more than 90% accuracy.

5.3 Snips-NLU

Snips NLU is an open-source project which is a Natural Language Understanding python library that allows to parse sentences written in natural language, and extract structured information. Behind every chatbot and voice assistant lies a common piece of technology: Natural Language Understanding (NLU). Anytime a user interacts with an AI using natural language, their words need to be translated into a machine-readable description of what they

meant. The NLU engine first detects what the intention of the user is (intent), then extracts the parameters (called slots) of the query. The developer can then use this to determine the appropriate action or response. Sample code from Snips-NLU is shown in appendix A

Snips NLU library extracts the entities and slots and resolves them. It supports about 10 languages worldwide. Since our interest to build an ontology is for python language, we believe Snips NLU project is the right source of python source code.

5.4 Data collection and Preprocessing

The dataset used for this experimentation is collected from Snips-NLU project. This data is split into training and testing corpus. Snips NLU project is composed of different types of files such as markdown file, YAML files documentations, Json files, data sources and others. Data preprocessing involves removing files which are not python source file. The other purpose of data preprocessing is to remove comments and other irrelevant parts of the source code files. After removal of other type of files and comments, 125 python(.py) file were given to term extractor and other modules.

- **Code Parser**

AST module in python provides all the relevant words which are candidate to be concepts. For example, to extract all the terms which are class definitions in the code, if we use regular expression, we must declare some expression like something starting with the word class and followed by opening brace. But this is simplified with ast module; it can be accessed by using just the module ast.ClassDef walking through the nodes of AST. In addition to this, the tree structure of AST makes it easier to extract the taxonomic relationships among entities. For example: Python abstract syntax tree structure for the statement (x = y + 3) is shown in Figure 5.1.

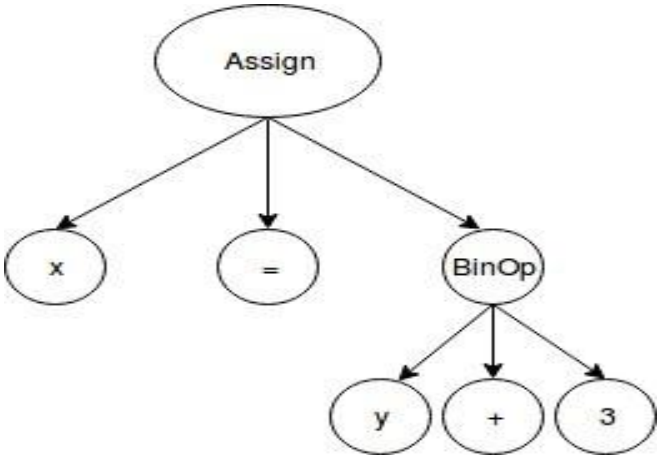

Figure 5.1: *Sample AST tree of python statement*

Where Assign and BinOp is AST node type. Then by using the ast nodes and some node functions we can access all the elements of the tree.

After parsing the code into AST, all the nodes in the tree are visited written into a json file with their corresponding types. The algorithm we have used gives out a json file corpus which has mainly three main nodes namely, type, children and value. Sample terms extracted from SNIPS-NLU corpus in json format is shown in annex B. Totally, 114,604 terms were extracted from SNIPS-NLU project. After removal of irrelevant strings, only 110,465 were found to be relevant and given to the ontology learning model. This dataset was split into training and test set based on the ratio of [80:20].

The extracted terms are labeled as class definitions, function definitions, function calls, Expressions, Binary Operations, Variables, Imports, and so on as discussed in following section.

5.5 Term Extraction

The purpose of this module is to extract candidate terms from source code. Candidate terms to be entities are extracted from all python code files automatically using pre-trained Bi-LSTM+CRF model. The overall model training and optimization process is presented below.

Model Training

The Bi-LSTM + CRF model was defined manually and trained automatically on the training data which is labeled based on the general types of code elements. The model was trained by iterating in 10 epochs by using fit model. Figure 5.2 shows the accuracy results of model training in the first three epochs. This was done to improve accuracy of the model. From the training epoch results we can see that the accuracy of model improved from 0.96007 to 0.98532. But by making the training epochs to 20 we can achieve a perfect model which is of accuracy 0.99017. the models after training are saved as hdf5 files.

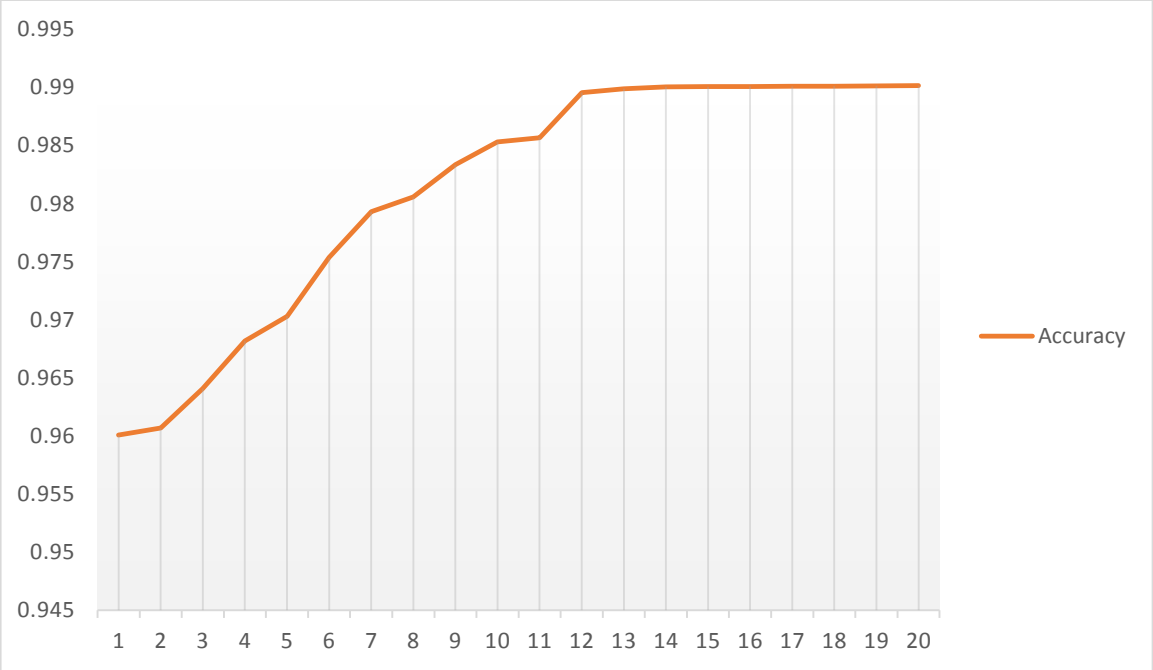

Figure 5.2: *Model training accuracy results for 20 epochs*

5.6 Concept Extraction

This section presents experimentation of ontology learning step, which is concepts and relation extraction. We used code element labels discussed in section 4.2 to group terms/entities in to concepts to experiment concept extraction. TF-IDF was used to enrich concept list with additional terms which represent the project. Sample code for concept

extraction is shown in Appendix F. Table 5.3 shows some of the concepts extracted from the corpus and some concept hierarchy is shown in Figure 5.2.

Table 5.2: *List of some concepts and properties extracted*

S. No	***Concept***	***Label***	***Elements***
1.	C1	Class	(SnipsNLUEngine)
2.	C2	Method	(redirect_stdout)
3.	C3	Method-call	(factory_name)
4.	C4	variable	(Offsets)
5.	C5	Method	(builtin_entity_match)

5.7 Relation Extraction

The next experimentation is on extraction of relations both taxonomic and non-taxonomic. Supervised Bi-LSTM model was used extract relations. This experimentation was done in two steps. First, detection of the existence of relation between two entities and then identifying the type of relation. Table 5.3 shows some of the taxonomic relations extracted from Snips-NLU corpus.

Table 5.3: *Sample taxonomic relations extracted*

Relation label	***(Domain, Range)***
Has_actual_argument	(builtin_entity_match, entity)
Is_a	(redirect_stdout, Method)
Has_type	**(**redirect_stdout, String)
Is_type_of	(public, Modifier)
Has_modifier	**(**redirect_stdout, protected)
Has_ identifier	**(**SnipsNLUEngine, term_list)
Has_method	(_InvalidCustomEntityFilter, redirect_stdout)

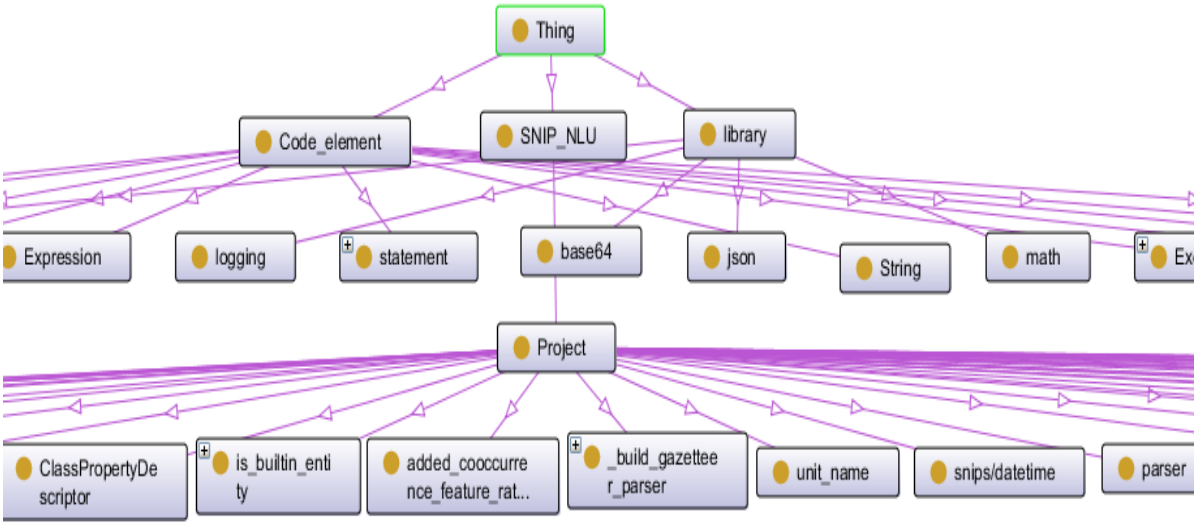

Figure 5.3: *Sample Snips-NLU ontology*

5.8 OWL translation.

After extracting all the necessary elements of ontology, it was written into a machine-readable language which is OWL in our case. We used protégé for manipulating our ontology. Figure 5.3 shows screen shot of some concepts and properties visualized using protégé. We used Owlready2 to write the concepts/entities into owl language automatically by using the sample code shown Annex F. Owlready2 is a module for manipulating OWL 2.0 ontologies in Python. It can load, modify, save ontologies, and it supports reasoning via HermiT (included). Owlready allows a transparent access to OWL ontologies.

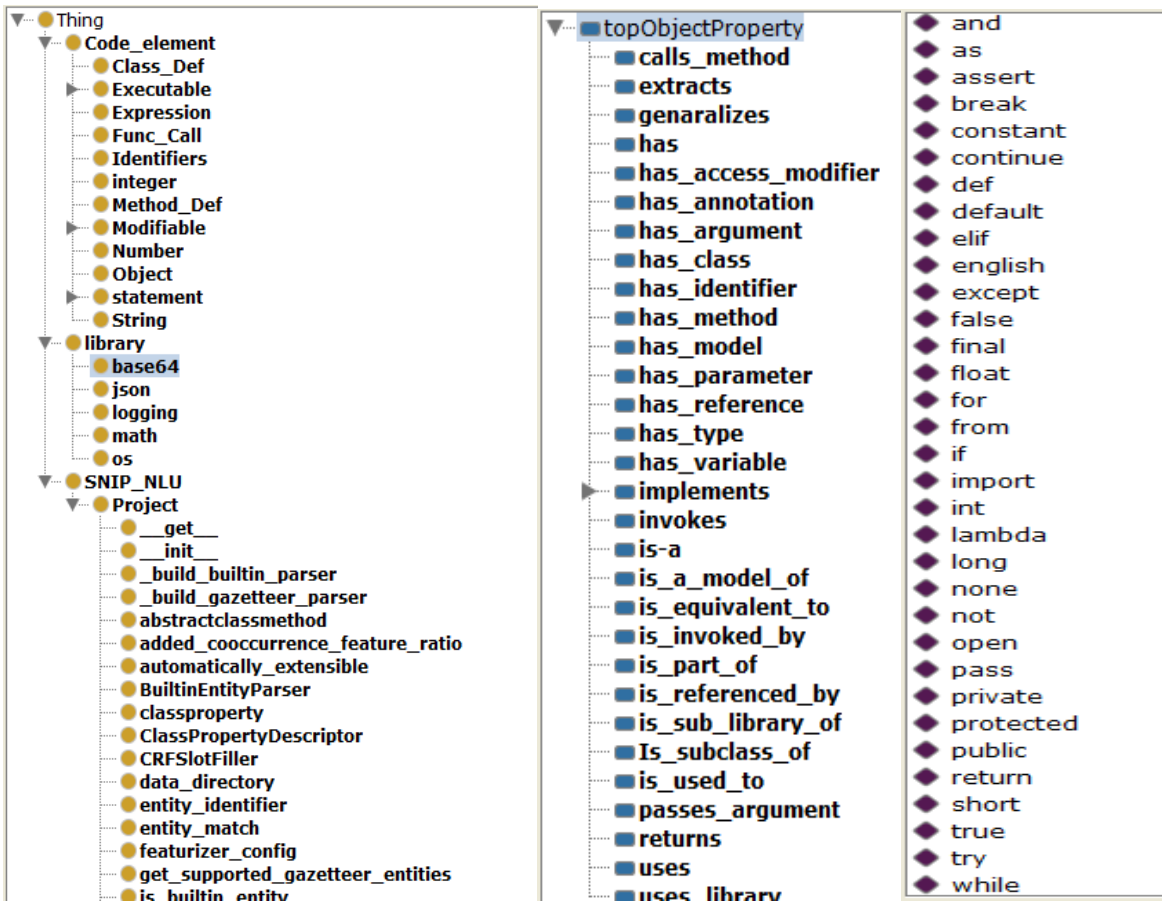

Figure 5.4: *Screen capture of snips-nlu ontology in protege.*

5.9 Construction of code knowledge graph.

With the methods used in the above experimentations, we obtained approximately 4,660 entities and 2,115 relationships from Snip-NLU code data. After entity disambiguation and coreference resolution, the valid knowledge graph of the project was obtained. There were a number of errors among the obtained knowledge because of unsuitable rules or algorithm precision issues. The knowledge validation was executed by experts and knowledge engineers to correct these errors. The massive entity-relationship triples were input to knowledge base. Knowledge inference was executed to look up new links among the entities. Finally, project knowledge base was constructed, and the knowledge graph was developed based on the above methods.

5.10 Storage and Visualization of Code Knowledge graph

Finally, the visual presentation of our KG was developed using Force-Directed Graph. Entity-relationship-entity triples of the project was presented using the VOWL plugin for protégé to visualize KG, as shown in Figure 5.4. The screenshot shows only the connection between project and code elements. Each node represents the entities of Snips-NLU project extracted. Connection between nodes represents relationship between entities.

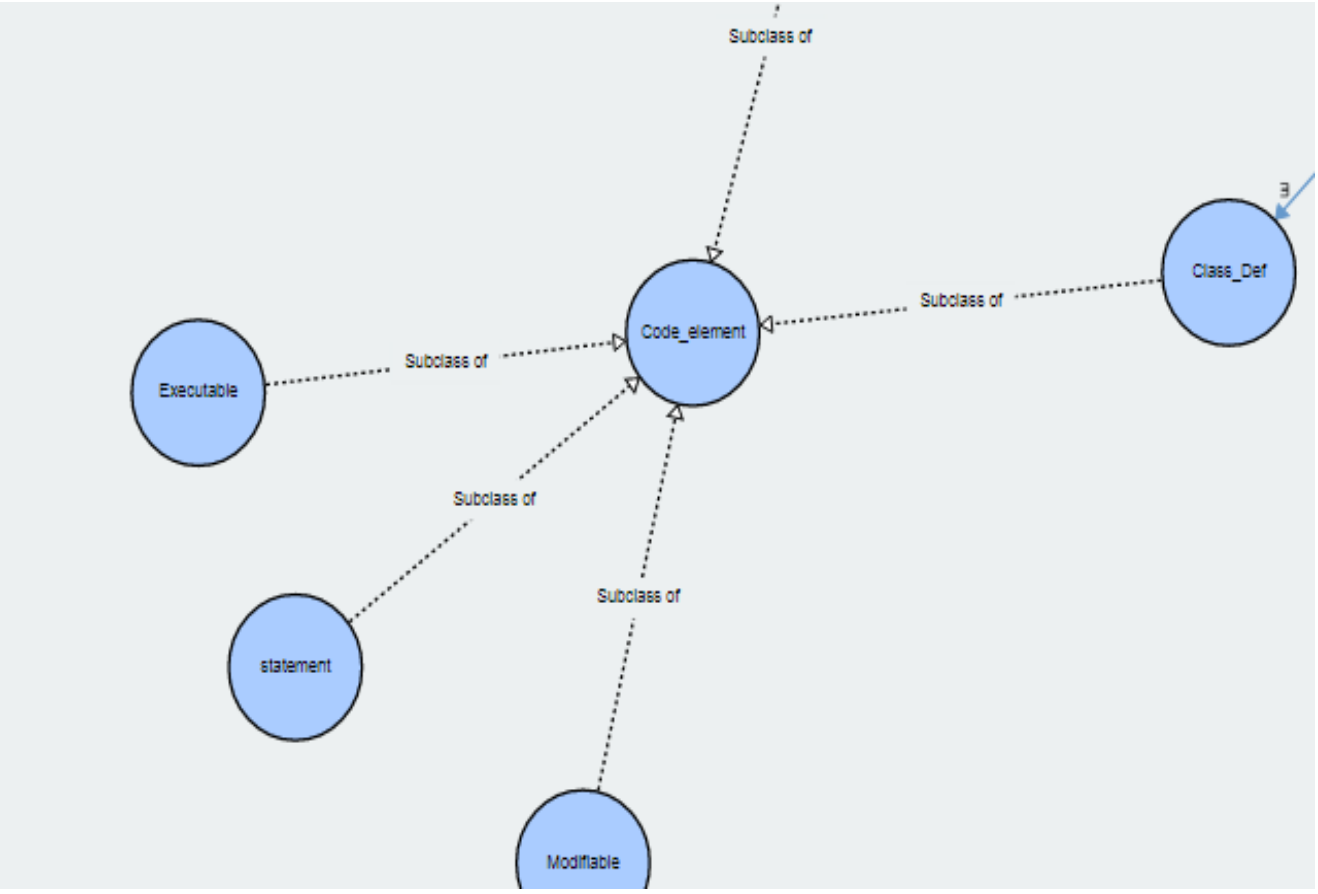

Figure 5.5: *Sample screenshot from knowledge graph*

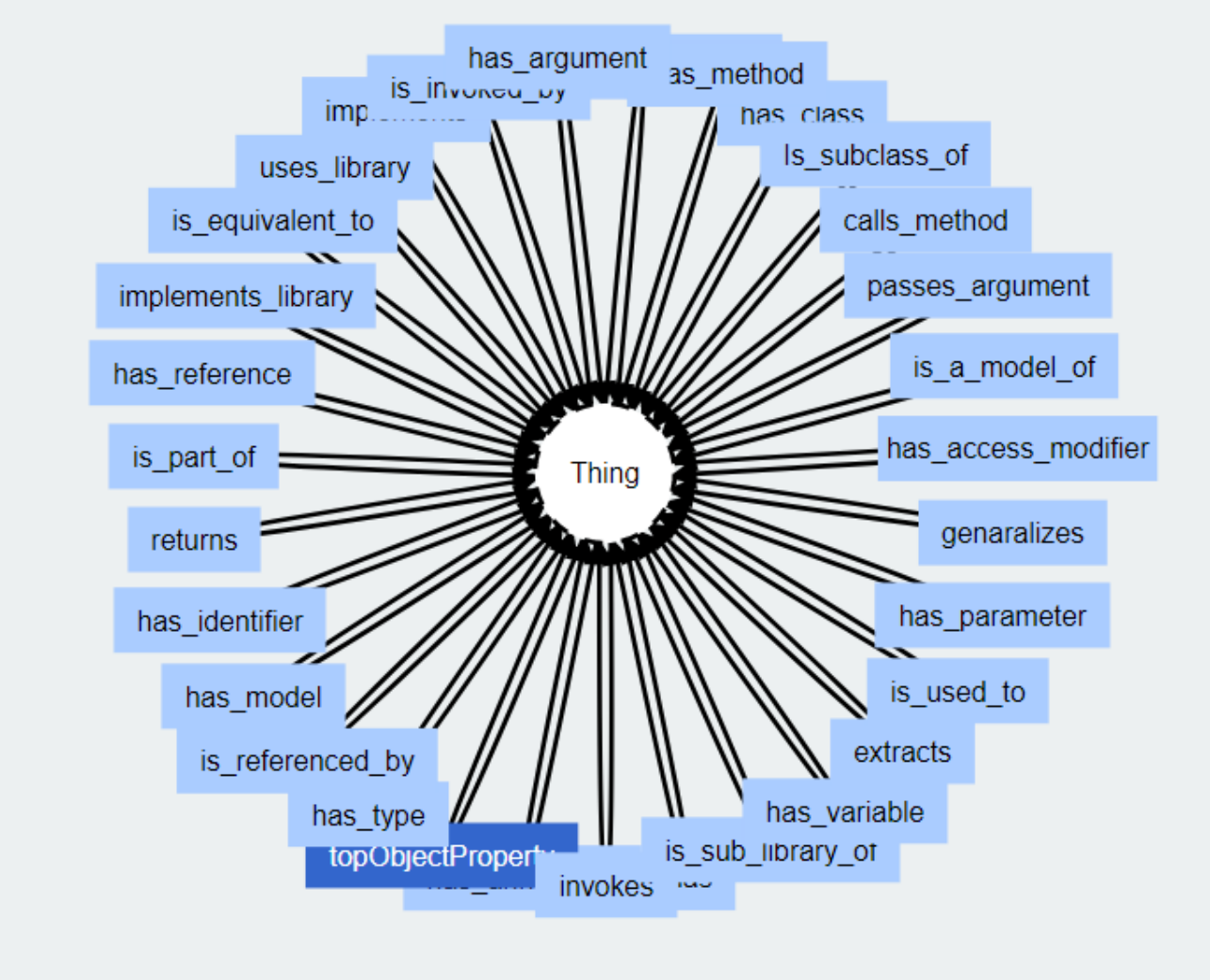

Figure 5.6: *Graph representing relations in the ontology*

We used rdflib, python library to store and retrieve our KG. RDFLIB is a python API for RDF that contains parsing and serializing for rdf/xml, ntriples, turtle and other formats is used with its graph interface that has support for SPARQL queries and update statements.

5.11 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 in our questioner as shown in Appendix G. From the expert's evaluation, we have got the following result shown in Table 5.5.

Table 5.4: *Expert evaluation results*

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

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). The standard recall and precision measures are calculated as follows and f-measure is calculated based on the values of precision and recall:

$$\text{Precision} = \frac{TP}{TP+FF} \qquad \text{Recall} = \frac{TP}{TP+FN}$$

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

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 5.3 summarizes the evaluation result of the four phases in our approach.

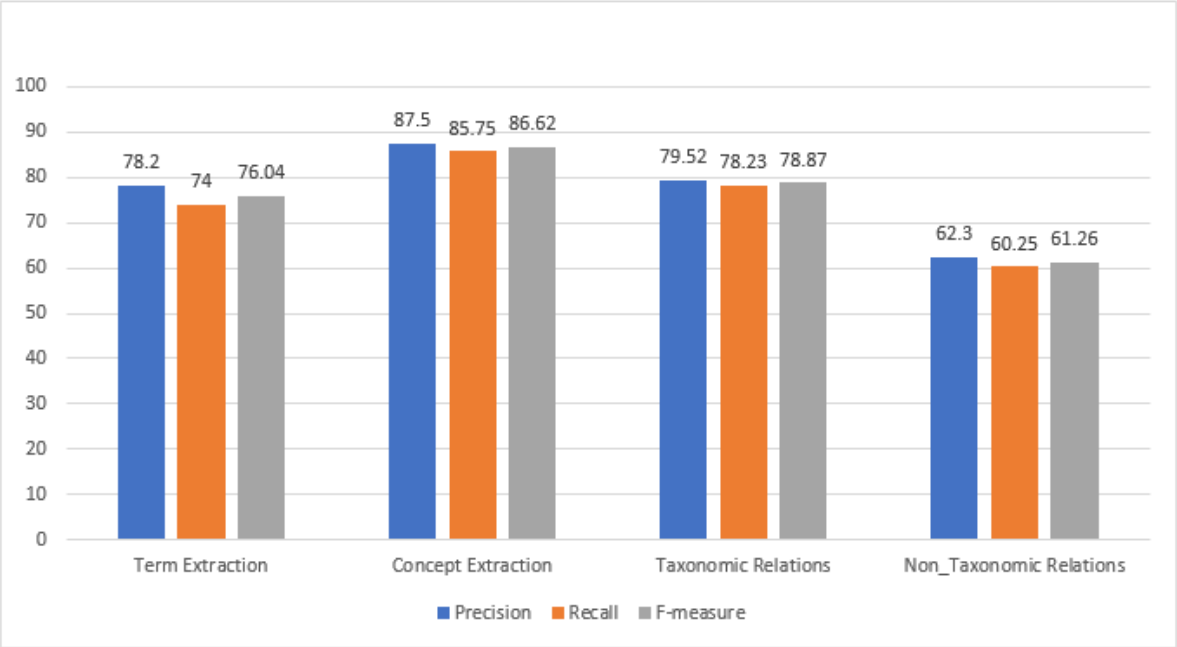

Figure 5.7: *Ontology learner performance measurement*

From this evaluation results, we can conclude that the use of bi-directional RNN network (Bi-LSTM and CRF) for extraction of ontology is effective and promising. Our approach has yielded average f-measure of 77.04 as shown in Figure 5.3. Our extracted ontology consists all the elements in the gold standard ontology and many new elements were found.

Chapter Six

Conclusion and future works

6.1 Conclusion

The aim of this research was to propose an ontology learning method from source code using a combination of statistical CRF and Bi-LSTM recurrent neural network. In addition to this, proposing a framework for construction of knowledge graph based on the ontology was also the aim of this study. We have reviewed different literatures in the field in order to understand the techniques, methods, and tools in ontology learning and knowledge graph construction. Many related works along with their architecture, methods and contributions were presented in this paper. This research was motivated by looking at the gap in semantic management of code as data and lack of clear and efficient framework for ontology learning from source code. Most of the related works relayed on Java source code and python with its flexible nature was left unrecognized in the semantic web. some related works while important, depend the nature of the programming language and others deal with the general ontology structure for object-oriented languages. Hence, we proposed an ontology learning and knowledge graph construction method based on the statistic and deep learning method combined together.

The proposed solution was experimented by using python programming language on Snips-NLU library of Natural Language Understanding. The system was experimented by using manually defined CRF model for concept/entity extraction and Bi-LSTM network for property/relation extraction. Evaluation was done using the gold standard ontology and the proposed approach exhibited 77.04% of f-measure and this shows that the proposed approach is promising and can be improved and adopted for extracting other elements from python code. Since gold standard evaluation is based on manually developed ontology, we used expert evaluation to check the relevance of our extracted ontology elements. From this our approach has achieved average relevance of 81.275%.

Source code is a valuable resource and needs to be represented in semantic knowledge representation for a number of reasons: First, source code contains information about the

business rules used to perform operations. Policies and procedures of a company do not always hold the information related to operations in company but source code does.

Second, source code contains information about where and how the results of operation are stored in a company.

Hence, this study has made the following contributions in semantic web and code data management.

- General architecture for ontology learning and knowledge graph construction from python source code.
- Defining model parameters to fit the structure of python source code.
- Defining code related features to be mined from code files for model training.
- Exploring the necessity of knowledge graph for python source code as data and providing the first code knowledge graph which can serve as a foundation for any code analysis researches.

6.2 Future work

Based on the experimental results and limitations in resources we found the following works to be considered in the future.

- Since this study was experimented on python language, adapting the proposed solution for the other object-oriented languages should be experimented and checked for relevance.
- Using the large datasets provided for machine learning on code such as py150 dataset, for training and testing the learning models. This was impossible in this study b/c of lack of the hardware resources.

References

[1] A. Ankolekar, K. Sycara, J. Herbsleb, and R. Kraut., "Supporting Online Problem".

[2] K. &. S. M. Bontcheva, " Learning ontologies from software artifacts: Exploring and combining multiple sources.," 2006.

[3] P. Velardi, "Wikipedia, the free encyclopedia," Wikimedia Foundation, Inc, 9 September 2018. [Online]. Available: https://en.wikipedia.org/wiki/Ontology_learning. [Accessed 10 October 2018].

[4] Atzeni, Mattia & Atzori, Maurizio, "Codeontology: RDF-ization of source code," 2017.

[5] G. G. a. S. Sagayaraj, "To Generate the Ontology from Java Source Code by OWL Creation," *(IJACSA) International Journal of Advanced Computer Science and Applications,,* vol. Vol. 2, February 2011.

[6] Singhal, Amit, "Introducing the Knowledge Graph: Things, Not Strings," May 16, 2012.

[7] https://www.Opensource.com, "12 challenges for open source projects".

[8] Jiomekong Azanzi, 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.

[9] Ken Peffers, Tuure Tuunanen, Charles E. Gengler, Matti Rossi, Wendy Hui, Ville Virtanen, Johanna Bragge, "THE DESIGN SCIENCE RESEARCH PROCESS:A MODEL FOR

PRODUCING AND PRESENTING INFORMATION SYSTEMS RESEARCH," in *CA.*, 2006.

[10] Antonio Moreno Ribas (URV) and Ulises Cortés, "Domain Ontology Learning from the Web," *Tarragona, ,* 2007..

[11] P. Cimiano, "Ontology Learning and Population from Text: Algorithms,Evaluation and Applications," *Secaucus, NJ, USA: Springer-Verlag NewYork,* 2006.

[12] P. Buitelaar, P. Cimiano, and B. Magnini., " Ontology learning from text: Methods, applications and evaluation.," in *IOS Press*, 2005.

[13] L. Zhou, "Ontology learning: state of the art and open issues," *in Springer Science+Business Media,* 2007, .

[14] L. Zhou, "Ontology learning: state of the art and open issues," in *Springer Science+Business Media, LLC 2007,* 24 March 2007.

[15] T. O. Ayodele, "Types of Machine Learning Algorithms," *New Advances in Machine Learning,,* 2010.

[16] S. S. S. a. S. Ben-David, Understanding Machine Learning:From Theory to Algorithms, Cambridge : Cambridge University Press, 2014.

[17] Charles Sutton and Andrew McCallum, ""An Introduction to Conditional Random Fields"," *Foundations and Trends® in Machine Learning:,* vol. 4, pp. 267-373.

[18] 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 978-981-10-5209-5, Singaphore, 2018.

[20] D. Z. &. D. Wang, "Relation Classification via Recurrent Neural Network," *Tsinghua National Lab for Information Science and Technology,* 2015.

[21] H. Paulheim., "Knowledge Graph Refinement: A Surveyof Approaches and Evaluation Methods.," *Semantic WebJournal,* vol. 1, p. 20, 2016..

[22] Dou, Jinhua & Jingyan, Qin & Jin, Zanxia & Li, Zhuang, "Knowledge graph based on domain ontology and natural language processing technology for Chinese intangible cultural heritage.," *Journal of Visual Languages & Computing,* vol. 48, pp. 19-28, 2017.

[23] Lisa Ehrlinger and Wolfram Wob, "Towards a Definition of Knowledge Graphs, Institute for Application Oriented Knowledge Processing," *SEMANTICS ,* 2016..

[24] LiuQiao, LiYang, DuanHong, LiuYao & QinZhiguang., "Knowledge Graph Construction Techniques[J]," *Journal of Computer Research and Development,* vol. 3, pp. 582-600, 2016.

[25] Xindong Wu∗†, Jia Wu‡, Xiaoyi Fu∗, Jiachen Li∗, Peng Zhou†§, and Xu Jiang∗, "Automatic Knowledge Graph Construction: A Report on the 2019 ICDM/ICBK Contest," Mininglamp Academy of Sciences, Mininglamp Technology, , Beijing 10084, China, 2019.

[26] K. Woldemariyam, "Generic Semantic Annotation Framework with Integrated," Addis Ababa University, Addis Ababa, 2017.

[27] F. Wu and D. S. Weld,, "“Open information extraction using wikipedia,”," *in Proc. ACL,* p. 118–127, 2010.

[28] M. Banko, M. J. Cafarella, S. Soderland, M. Broadhead, and O. Etzioni,, "Open information extraction from the web," *Ijcai*, vol. 7, p. 2670–2676, 2007.

[29] M. Surdeanu, J. Tibshirani, R. Nallapati, and C. D., "Manning, “Multiinstance multi-label learning for relation extraction,”," in *EMNLPCoNLL*, 455–465, 2012.

[30] C. Quirk and H. Poon,, "Distant supervision for relation extraction," arXiv preprint arXiv:1609.04873, 2016.

[31] P. Buitelaar, D. Olejnik, and M. Sintek., "A Protege Plug-in for Ontology Extraction from TextBased on Linguistic Analysis.," in *In Proceedings of the 1st European Semantic Web Symposium*, 2004.

[32] Birhanu Mengiste and Fekade Getahun, "Amharic Ontology Learner," Addis Ababa University, Addis Ababa, 2013.

[33] A. Walia, "Annotated Corpus for Named Entity Recognition," kaggle, 02 02 2019. [Online]. Available: https://www.kaggle.com/abhinavwalia95/entity-annotated-corpus. [Accessed 05 01 2021].

[34] M. Grobelnik and D. Mladenic., " Knowledge Discovery for Ontology Construction.," in *In J. Davies, R. Studer, and P. Warren, editors, Semantic Web Technologies.* , 2006.

[35] M. Sabou., " Building Web Service Ontologies.," in *PhD thesis*, 2006.

[36] R. J. Brachman, P. Devanbu, P. G. Selfridge, D. Belanger, and Y. Chen., "Toward a software information system," in *AT&T Techn ical Journal*, 1990.

[37] T. R. Gruber, "A translation approach to portable ontology specification," in *Knowledge Acquisition*, 1993.

[38] C. S. a. A. McCallum, "An Introduction to Conditional Random Fields," in *Foundations and Trends® in Machine Learning:*, 2012.

[39] s. Community, "src-d/sourced-ce: source{d} Community Edition (CE) -," GitHub, [Online]. Available: https://github.com/src-d/sourced-ce. [Accessed 04 09 2020].

[40] P. Documentation, "ast — Abstract Syntax Trees," python.org, [Online]. Available: https://docs.python.org/3/library/ast.html#module-ast. [Accessed 15 02 202].

[41] Martinez-Rodriguez, J. L., Lopez-Arevalo, I., & Rios-Alvarado, "OpenIE-based approach for Knowledge Graph construction from text," *Expert Systems with Applications*, vol. 113, p. 339–355, 2015.

[42] A. Wroblewska, T. Podsiady-Marczykowska, R. Bembenik, G. Protaziuk, and H. Rybinski, "Methods and tools for ontology building, learning and integration application in the synat project," *Semantic web journal*, Vols. 390, , 2012.

[43] M. Shamsfard and A. AbdollahzadehBarforoush, "The state of the artin ontology learning:A framework for comparison," *Knowl. Eng.*, vol. 18, p. 293–316, 2003.

[44] A. G.-P. ´. erez, "Evaluation of ontologies," *International Journal of intelligent systems*, vol. 16, p. 391–409, 2001.

[45] Alexander Maedche and Steffen Staab, "Discovering conceptual relations from text.," in *W. Horn, editor, Proceedings of the 14th European Conference on Artificial Intellignece*, 2000..

[46] Brewster, C., Alani, H, Dasmahapatra, S and Wilks, Y., "Data-driven Ontology Evaluation.," in *In Proceedings of the 4th International Conference on Language Resources and Evaluation.*, 2004..

[47] A. Maedche and S. Staab, "Semi-automatic engineering of ontologies from text," in *Proceedings of the 12th Internal Conference on Software and Knowledge Engineering.* , Chicago, USA,, 2000..

[48] A. Gomez-P ´ erez, "Evaluation of ontologies," *International Journal of intelligent systems*, vol. 16, p. 391–409, 2001.

[49] Yogita Rani and Dr. Harish Rohil, "A Study of Hierarchical Clustering Algorithm.," *International Journal of Information and Computation Technology.* , vol. 3, no. ISSN 0974-2239, pp. 1225-1232, 2013.

[50] Zhanfang Zhao, Sung-Kook Han & In-Mi So, "Architecture of Knowledge Graph Construction Techniques," Hebei GEO University, City Iksan, JeonBuk, 54538, Korea, February 4, 2018.

[51] Khadilkar V, Kantarcioglu M, Thuraisingham B, et al. JenaHBase:, "A distributed, scalable and efficient RDF triple," in */Proceedings of the 2012th International Conference*, CEURWS.org, , 2012.

[52] Cudr-Mauroux P, Enchev I, Fundatureanu S, et al., "NoSQL databases for RDF: an empirical evaluation[C]," in *International Semantic Web Conference.*, Berlin, Heidelberg, 2013.

Annexes

Annex A: Sample code file from Snips-NLU library

```
from __future__ import unicode_literals

from builtins import object

from snips_nlu.resources import get_stems

def stem(string, language, resources):
    from snips_nlu_utils import normalize

    normalized_string = normalize(string)
    tokens = tokenize_light(normalized_string, language)
    stemmed_tokens = [_stem(token, resources) for token in tokens]
    return " ".join(stemmed_tokens)

def stem_token(token, resources):
    from snips_nlu_utils import normalize

    if token.stemmed_value:
        return token.stemmed_value
    if not token.normalized_value:
        token.normalized_value = normalize(token.value)
    token.stemmed_value = _stem(token.normalized_value, resources)
    return token.stemmed_value

def _stem(string, resources):
    return get_stems(resources).get(string, string)

class Token(object):

    def __init__(self, value, start, end, normalized_value=None,
                 stemmed_value=None):
        self.value = value
        self.start = start
        self.end = end
        self.normalized_value = normalized_value
        self.stemmed_value = stemmed_value

    def __eq__(self, other):
        if not isinstance(other, type(self)):
            return False
        return (self.value == other.value
                and self.start == other.start and self.end == other.end)

   def tokenize(string, language):

    from snips_nlu_utils import tokenize as _tokenize

    tokens = [Token(value=token["value"],
                    start=token["char_range"]["start"],
                    end=token["char_range"]["end"])
              for token in _tokenize(string, language)]
    return tokens
```

Annex B: Sample parser result in json format


```
[{"type":"Module","children":[1,3,5,7,9,47,95,126,143,244,289]},{"type":"I
mportFrom","value":"__future__","children":[2]},{"type":"alias","value":"u
nicode_literals"},{"type":"ImportFrom","value":"builtins","children":[4]},
{"type":"alias","value":"object"},{"type":"ImportFrom","value":"snips_nlu.
resources","children":[6]},{"type":"alias","value":"get_stems"},{"type":"E
xpr","children":[8]},{"type":"FunctionDef","value":"stem","children":[10,1
6,46]},{"type":"arguments","children":[11,15]},{"type":"args","children":[
12,13,14]},{"type":"arg"},{"type":"arg"},{"type":"arg"},{"type":"defaults"
},{"type":"body","children":[17,19,24,30,40]},{"type":"ImportFrom","value"
:"snips_nlu_utils","children":[18]},{"type":"alias","value":"normalize"},{
"type":"Assign","children":[20,21]},{"type":"NameStore","value":"normalize
d_string"},{"type":"Call","children":[22,23]},{"type":"NameLoad","value":"
normalize"},{"type":"NameLoad","value":"string"},{"type":"Assign","childre
n":[25,26]},{"type":"NameStore","value":"tokens"},{"type":"Call","children
":[27,28,29]},{"type":"NameLoad","value":"tokenize_light"},{"type":"NameLo
ad","value":"normalized_string"},{"type":"NameLoad","value":"language"},{"
type":"Assign","children":[31,32]},{"type":"NameStore","value":"stemmed_to
kens"},{"type":"ListComp","children":[33,37]},{"type":"Call","children":[3
4,35,36]},{"type":"NameLoad","value":"_stem"},{"type":"NameLoad","value":"
token"},{"type":"NameLoad","value":"resources"},{"type":"comprehension","c
hildren":[38,39]},{"type":"NameStore","value":"token"},{"type":"NameLoad",
"value":"tokens"},{"type":"Return","children":[41]},{"type":"Call","childr
en":[42,45]},{"type":"AttributeLoad","children":[43,44]},{"type":"Str","va
lue":" "},{"type":"attr","value":"join"},{"type":"NameLoad","value":"stemm
ed_tokens"},{"type":"decorator_list"},{"type":"FunctionDef","value":"stem_
token","children":[48,53,94]},{"type":"arguments","children":[49,52]},{"ty
pe":"args","children":[50,51]},{"type":"arg"},{"type":"arg"},{"type":"defa
ults"},{"type":"body","children":[54,56,65,80,90]},{"type":"ImportFrom","v
alue":"snips_nlu_utils","children":[55]},{"type":"alias","value":"normaliz
e"},{"type":"If","children":[57,60]},{"type":"AttributeLoad","children":[5
8,59]},{"type":"NameLoad","value":"token"},{"type":"attr","value":"stemmed
_value"},{"type":"body","children":[61]},{"type":"Return","children":[62]}
,{"type":"AttributeLoad","children":[63,64]},{"type":"NameLoad","value":"t
oken"},{"type":"attr","value":"stemmed_value"},{"type":"If","children":[66
,70]},{"type":"UnaryOpNot","children":[67]},{"type":"AttributeLoad","child
ren":[68,69]},{"type":"NameLoad","value":"token"},{"type":"attr","value":"
normalized_value"},{"type":"body","children":[71]},{"type":"Assign","child
ren":[72,75]},{"type":"AttributeStore","children":[73,74]},{"type":"NameLo
ad","value":"token"},{"type":"attr","value":"normalized_value"},{"type":"C
all","children":[76,77]},{"type":"NameLoad","value":"normalize"},{"type":"
AttributeLoad","children":[78,79]},{"type":"NameLoad","value":"token"},{"t
ype":"attr","value":"value"},{"type":"Assign","children":[81,84]},{"type":
"AttributeStore","children":[82,83]},{"type":"NameLoad","value":"token"},{
"type":"attr","value":"stemmed_value"},{"type":"Call","children":[85,86,89
]},{"type":"NameLoad","value":"_stem"},{"type":"AttributeLoad","children":
[87,88]},{"type":"NameLoad","value":"token"},{"type":"attr","value":"norma
lized_value"},{"type":"NameLoad","value":"resources"},{"type":"Return","ch
ildren":[91]},{"type":"AttributeLoad","children":[92,93]}]
```

Annex C: Sample terms labeled based on their classes.

	code_index	word	label	tag
0	0	abstractclassmethod	CIASS	C
1	0	ClassPropertyDescriptor	CIASS	C
2	0	classproperty	Method	MTD
3	0	__init__	Method	MTD
4	0	__get__	Method	MTD
5	0	__set__	Method	MTD
6	0	setter	Method	MTD
7	0	ClassPropertyDescriptor	Func_Call	FC
8	0	type	Func_Call	FC
9	0	isinstance	Func_Call	FC
10	0	classmethod	Func_Call	FC
11	0	type	Func_Call	FC
12	0	AttributeError	Func_Call	FC
13	0	isinstance	Func_Call	FC
14	0	classmethod	Func_Call	FC
15	0	super	Func_Call	FC
16	0	can't set attribute	String	STR
17	1	json	Library	LIB
18	1	shutil	Library	LIB
19	1	BuiltinEntityParser	CIASS	C
20	1	_build_builtin_parser	Method	MTD
21	1	_build_gazetteer_parser	Method	MTD
22	1	is_builtin_entity	Method	MTD
23	1	is_gazetteer_entity	Method	MTD
24	1	is_grammar_entity	Method	MTD
25	1	find_gazetteer_entity_data_path	Method	MTD
26	1	_get_gazetteer_entity_configurations	Method	MTD
27	1	_get_caching_key	Method	MTD
28	1	dict	Func_Call	FC
29	1	__init__	Method	MTD

Annex D – Sample terms after indexing.


```
{' Feature weights:': 0, ' to_dict': 1, ' unresolved_slot': 2, ' capitaliz
e': 3, ' deepcopy': 4, ' BuiltinEntityParser': 5, ' Language of the builti
n entities': 6, ' pretty_print': 7, ' The language resources were successf
ully downloaded': 8, ' _get_feature_weight': 9, ' stem': 10, ' cls': 11, '
link_resources': 12, ' O': 13, ' generate_utterance': 14, " Can't update k
ey '%s'": 15, ' download': 16, ' slot_filler.json': 17, ' _get_entity_plac
eholders': 18, ' get_package_path': 19, ' intent': 20, ' viewvalues': 21,
' entity_parsers': 22, ' Language of the builtin entity': 23, ' %s %s: %s'
: 24, ' range': 25, ' B-': 26, ' /': 27, ' Either a dataset or a language
must be provided in order to build a BuiltinEntityParser': 28, " Creating
a shortcut link for '%s' didn't work": 29, ' is_gazetteer_entity': 30, ' d
eterministic_intent_parser': 31, ' intent_classification_result': 32, ' da
ta_directory': 33, ' _get_gazetteer_entity_configurations': 34, ' IntentNo
tFoundError': 35, ' /latest': 36, " Gazetteer entity '%s' is not supported
in language '%s'": 37, ' log_inference_weights': 38, '420': 39, ' _downloa
d_language_builtin_entities': 40, ' Download compatible language or gazett
eer entity resources': 41, ' get_builtin_entity_shortname': 42, ' str': 43
, ' stop_words_whitelist': 44, ' _create_custom_entity_parser_configuratio
n': 45, ' _get_range_shift': 46, ' log_result': 47, " Invalid 'type' value
in YAML file '%s': '%s'": 48, ' Latest entity resources version': 49, ' _b
uild_builtin_parser': 50, ' resource_path': 51, ' placeholder_fn': 52, ' l
og_weights': 53, ' parsers_metadata': 54, ' _download': 55, ' ClassPropert
yDescriptor': 56, ' utf-8': 57, ' fit': 58, ' Transition weights to next t
ag:': 59, ' next': 60, " You can now use the '%s' builtin entity": 61, ' r
e': 62, ' CustomEntityParser': 63, ' _add_missing_entities': 64, ' _uttera
nce_to_pattern': 65, ' dict_setitem': 66, ' gazetteer_entity_parser': 67,
' Dataset': 68, ' sort_key_fn': 69, ' type_error': 70, ' get_contexts_iter
ator': 71, ' - %s': 72, " Tried to initialize LimitedSizedDict with more v
alue than permitted with 'limit_size'": 73, ' merge_usages': 74, ' fitted'
: 75, ' _cleanup': 76, ' Transition weights:': 77, ' Unknown tagging schem
e %s': 78, ' store_true': 79, ' start_of_bio_slot': 80, ' get_default_sep'
: 81, ' CRFSlotFiller': 82, ' _download_and_link': 83, ' filename': 84, '
DeterministicIntentParser': 85, ' get_all_grammar_entities': 86, ' compute
_features': 87, ' KeyError': 88, ' _get_incoming_weights': 89, ' download_
all_languages': 90, ' check_resources_alias': 91, ' sys': 92, ' is_builtin
_entity': 93, ' _load_dataset_parts': 94, ' io': 95, ' int': 96, ' Missing
deterministic intent parser metadata file: %s': 97, ' %%%s%%': 98, " 'size
_limit' must be passed as a keyword argument": 99, ' _ensure_safe': 100, '
_get_crf_model': 101, ' tokenize_light': 102, ' size_limit': 103, ' Limite
dSizeDict': 104, ' sorted': 105, ' entity_name': 106, '2': 107, ' dict': 1
08, ' DatasetFormatError': 109, ' parser_usage': 110, ' *': 111, ' Token':
112, ' snips_nlu.result': 113, ' base64': 114, ' Fitting deterministic int
ent parser': 115, ' _get_entity_values': 116, ' slot_names_to_entities': 1
17, ' Linking successful': 118, ' get_all_gazetteer_entities': 119, ' vali
date_type': 120, ' get_sequence_probability': 121, ' Features not seen at
train time:%s': 122, ' extra_pip_args': 123, ' group_names_to_slot_names':
124, ' extraction_result': 125, ' stopwords_fraction must be in': 126, ' u
tf8': 127, ' top_n argument must be greater or equal to 1': 128, ' super':
129, ' threshold': 130, ' end_of_io_slot': 131, ' get_all_builtin_entities
': 132, ' tempfile': 133, ' %s': 134, ' values': 135, ' ValueError': 136,
'0.001': 137, ' %s --> %s': 138, ' version': 139, ' get_slots': 140, ' pos
itive_tagging': 141, ' Missing slot filler model file: %s': 142,}
```

Annex E: Sample ontology in owl format

```
<!DOCTYPE rdf:RDF [
   <!ENTITY owl "http://www.w3.org/2002/07/owl#" >
   <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#" >
   <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#" >
   <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#" >
]>
<rdf:RDF xmlns="http://www.semanticweb.org/user/ontologies/2021/1/Snip-NLU#"
    xml:base="http://www.semanticweb.org/user/ontologies/2021/1/Snip-NLU"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
<owl:Ontology rdf:about="http://www.semanticweb.org/user/ontologies/2021/1/untitled-ontology-9"/>
<owl:Class rdf:about="http://www.semanticweb.org/user/ontologies/2021/1/untitled-ontology-9#CRFSlotFiller">
      <rdfs:subClassOf rdf:resource="http://www.semanticweb.org/user/ontologies/2021/1/untitled-ontology-9#Project"/>
      <rdfs:subClassOf>
         <owl:Restriction>
            <owl:onProperty rdf:resource="http://www.semanticweb.org/user/ontologies/2021/1/untitled-ontology-9#is-a"/>
            <owl:someValuesFrom rdf:resource="http://www.semanticweb.org/user/ontologies/2021/1/untitled-ontology-9#Class_Def"/>
         </owl:Restriction>
      </rdfs:subClassOf>
   </owl:Class>
   <!-- http://www.semanticweb.org/user/ontologies/2021/1/untitled-ontology-9#Class -->
   <owl:Class rdf:about="http://www.semanticweb.org/user/ontologies/2021/1/untitled-ontology-9#Class">
      <rdfs:subClassOf rdf:resource="http://www.semanticweb.org/user/ontologies/2021/1/untitled-ontology-9#Modifiable"/>
   </owl:Class>
```

<!-- http://www.semanticweb.org/user/ontologies/2021/1/untitled-ontology-9#has -->

<owl:ObjectProperty rdf:about="http://www.semanticweb.org/user/ontologies/2021/1/untitled-ontology-9#has"/>

<!-- http://www.semanticweb.org/user/ontologies/2021/1/untitled-ontology-9#has_access_modifier -->

<owl:ObjectProperty rdf:about="http://www.semanticweb.org/user/ontologies/2021/1/untitled-ontology-9#has_access_modifier"/>
<owl:Class rdf:about="http://www.semanticweb.org/user/ontologies/2021/1/untitled-ontology-9#_build_builtin_parser">
<rdfs:subClassOf rdf:resource="http://www.semanticweb.org/user/ontologies/2021/1/untitled-ontology-9#Project"/>
<rdfs:subClassOf>
<owl:Restriction>
<owl:onProperty rdf:resource="http://www.semanticweb.org/user/ontologies/2021/1/untitled-ontology-9#is-a"/>
<owl:someValuesFrom rdf:resource="http://www.semanticweb.org/user/ontologies/2021/1/untitled-ontology-9#Method_Def"/>
</owl:Restriction>
</rdfs:subClassOf>
</owl:Class>
<!-- http://www.semanticweb.org/user/ontologies/2021/1/untitled-ontology-9#_build_gazetteer_parser -->

<owl:Class rdf:about="http://www.semanticweb.org/user/ontologies/2021/1/untitled-ontology-9#_build_gazetteer_parser">
<rdfs:subClassOf rdf:resource="http://www.semanticweb.org/user/ontologies/2021/1/untitled-ontology-9#Project"/>
<rdfs:subClassOf>
<owl:Restriction>
<owl:onProperty rdf:resource="http://www.semanticweb.org/user/ontologies/2021/1/untitled-ontology-9#is-a"/>
<owl:someValuesFrom rdf:resource="http://www.semanticweb.org/user/ontologies/2021/1/untitled-ontology-9#Method_Def"/>
</owl:Restriction>
</rdfs:subClassOf>
</owl:Class>

Annex F: Sample knowledge graph in rdf format


```
<?xml version="1.0" encoding="UTF-8"?>

<rdf:RDF

   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"

   xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"

>

  <rdf:Description
rdf:about="http://example.org/SnipsNLUProject/http://id.loc.gov/authorities/names/no99007700
">

    <rdfs:range xml:lang="en"> Number</rdfs:range>

    <rdfs:label xml:lang="en">is_a</rdfs:label>

    <rdfs:domain xml:lang="en">1</rdfs:domain>

  </rdf:Description>

  <rdf:Description
rdf:about="http://example.org/SnipsNLUProject/http://id.loc.gov/authorities/names/no99007555
">

    <rdfs:domain xml:lang="en"> isinstance</rdfs:domain>

    <rdfs:label xml:lang="en">has_type</rdfs:label>

    <rdfs:range xml:lang="en"> Func_Call</rdfs:range>

  </rdf:Description>

  <rdf:Description
rdf:about="http://example.org/SnipsNLUProject/http://id.loc.gov/authorities/names/no99007546
">

    <rdfs:domain xml:lang="en"> __get__</rdfs:domain>

    <rdfs:label xml:lang="en">is_a</rdfs:label>

    <rdfs:range xml:lang="en"> Method</rdfs:range>
```

Annex G: Sample codes

- **Term indexing**

```
from keras.models import Model, Input
from keras.layers import LSTM, Embedding, Dense, TimeDistributed, Dropout, Bidirectional
import keras as k
from keras_contrib.layers import CRF

words = list(set(data["word"].values))
n_words = len(words)

input = Input(shape=(672,))
word_embedding_size = 150

# Embedding Layer
model = Embedding(input_dim=n_words, output_dim=word_embedding_size, input_length=672)(input)
```

- **Models used for Concept Extraction**

```
from keras.models import Model, Input
from keras.layers import LSTM, Embedding, Dense, TimeDistributed, Dropout, Bidirectional
import keras as k
from keras_contrib.layers import CRF
model = Bidirectional(LSTM(units=word_embedding_size,
                           return_sequences=True,
                           dropout=0.5,
                           recurrent_dropout=0.5,
                           kernel_initializer=k.initializers.he_normal()))(model)
model = LSTM(units=word_embedding_size * 2,
             return_sequences=True,
             dropout=0.5,
             recurrent_dropout=0.5,
             kernel_initializer=k.initializers.he_normal())(model)

# TimeDistributed Layer
model = TimeDistributed(Dense(n_tags, activation="relu"))(model)

# CRF Layer
crf = CRF(n_tags)

out = crf(model)  # output
model = Model(input, out)
```

Annex G: Ontology Learner Evaluation Questionnaire for domain experts

Addis Ababa University

College of Natural Science

Department of Computer Science

Ontology learning Method Evaluation Questionary

For Domain Experts

Dear participant,

Code ontology learning system is designed to automate the extraction of knowledge from source code by implementing some deep learning and statistical machine learning method. It also deals with the construction of knowledge graph for code based on the learnt ontology. We have experimented our approach on python code dataset and this needs a domain expert validation for evaluating the relevance of proposed approach.

Therefore, we kindly request you to fill the forms below based on your experience and give us any kind of decision or comments based on your evaluation. Evaluation is based on the following criteria's:

- **Accuracy** is a criterion that states if the definitions, descriptions of classes, properties, and individuals in an ontology are correct.
- **Completeness** measures if the domain of interest is appropriately covered in this ontology.
- **Conciseness** is the criteria that states if the ontology includes irrelevant elements with regards to the domain to be covered.
- **Adaptability** measures how far the ontology anticipates its uses. An ontology should offer the conceptual foundation for a range of anticipated tasks.
- **Clarity** measures how effectively the ontology communicates the intended meaning of the defined terms. Definitions should be objective and independent of the context.

- **Computational efficiency** measures the ability of the used tools to work with the ontology, in particular the speed that reasoners need to fulfil the required tasks.
- **Consistency** describes that the ontology does not include or allow for any contradictions.

Thank you!

I. Term Extraction

No of terms extracted	Relevant	Irrelevant

II. Concept Extraction

No of Concepts extracted	Relevant	Irrelevant

III. Taxonomic Relation Extraction

No of Relations extracted	Relevant	Irrelevant

IV. Non- Taxonomic Relation Extraction

No of Relations extracted	Relevant	Irrelevant

V. Knowledge graph

	Extracted	Relevant	Irrelevant
No of subjects			
No of predicates			
No of objects			

Please leave us your Comments and suggestions on the ontology extracted

_____

_____

_____

_____

Thank you for your participation!

Declaration Sheet

I, the undersigned, declare that this thesis is my original work and has not been presented for a degree in any other university, and that all source of materials used for the thesis have been duly acknowledged.

Declared by:

Name: _____Amanuel Tadiwos_____

Signature: _____

Date: _____

Confirmed by advisor:

Name: _____Fekade Getahun (PhD)_____

Signature: _____

Date _____