International Journal for Multidisciplinary Research (IJFMR)
E-ISSN: 2582-2160 ● Website: www.ijfmr.com ● Email: editor@ijfmr.com

Domain Specific Named Entity Recognition Using Natural Language Processing(NLP) and Software Engineering in Production World

Bickey Kumar Shah[1], Prince Kumar[2]

[1]Site Reliability Engineer, Airtel Africa Digital Labs, Airtel Africa
[2]Software Engineer, Airtel Africa Digital Labs, Airtel Africa

Abstract
This research is concerned with domain-specific named entity recognition in which certain keywords are withdrawn from the phrase and sentences of the words or paragraph. The project in hand has been effective in different sectors like the insurance companies and hospitals for withdrawing the required keywords in a small amount of time. For instance, a medicine that belongs to a specific keyword is extracted easily and the same occurs with necessary keywords in the case of insurance companies. Advancements in deep neural networks have been established to build the named entity recognition model (NER Model). We have used Spacy and natural language processing for keyword extraction. The future scope of the project is to use LSTM for domain-specific keyword extraction and transfer learning using NLP. A large dataset is required to build the system for keyword extraction since dataset have always been the challenging part of such a system.

Keywords: NER, LSTM, NLP

1. INTRODUCTION

Domain-specific named entity recognition is related to ML and AI. It has played a significant role in several fields such as the medical industry, insurance companies, and many more where they classify the entity from a bundle of tasks and words. For instance, if anyone needs to find a medicine for fever, then it can be easily identified through this system of domain-specific named entity recognition. Social online communities like Stack Overflow and Quora have played a significant role in knowledge sharing and acquisition for software developers [10]. User-generated content on these websites has grown into an important information resource on the web which complements traditional technical documentation [2]. A basic task for reusing content in these websites is looking for discussions of a specific software entity (e.g., a book, an object, an API), to find good usage patterns, bug solutions, or alternatives. Existing methods treat software engineering social content as textual documents, and use vector space model, topic model, or neural network language model to index the content [3]. These methods have an important limitation, i.e., uniform importance assumption which mentions of software-specific entities in the content are treated in the same way as other regular textual content. This assumption may result in less desirable indexing of the content because traditional information retrieval concepts such as term frequency do not apply naively to recognize essential domain-specific entities [1]. The socio-technical nature of software

engineering social content which is available on social websites calls for innovative forms of information extraction, organization, and searching. A very beneficial goal would be to organize the information in a knowledge base consisting of different software-specific entities and relationships between entities. Such a knowledge base could be represented as a graph, also known as a knowledge graph. Search systems can exploit knowledge graphs not only for finding the content that discusses a particular software-specific entity but also for displaying additional facts and direct information about the central entity in a query. As the first step towards knowledge graphs and entity-centric search systems for the software engineering domain, we must be able to recognize mentions of software-specific entities in software engineering social content and classify them into predefined categories [4]. This task is referred to as named entity recognition or NER for short. NER has been comprehensively studied on formal text, informal text, and social content. Our goal is to recognize real-world objects in texts, such as a person, location, and organization. Some NER work from other domains exists for recognizing domain-specific entities, such as Biomedical NER. In contrast, our study is focused on designing domain-specific NER methods for software engineering social content, a new genre of socio-technical texts. Proposed solutions to NER fall into three categories: rule-based, machine learning-based, and hybrid methods. Existing studies show that machine learning-based methods usually outperform rule-based methods. However, for software engineering texts, existing methods are limited to dictionary look-up and rule-based methods based on code or text parsing. Furthermore, the entity category is limited to only the API. In this research, it is aimed to design and evaluate a machine learning-based method for general NER in software engineering social content. By general NER for software engineering, it would like to recognize not only APIs but also other categories of software-specific entities (such as programming languages, platforms, tools, libraries, frameworks, software standards) [3].

2. LITERATURE SURVEY

Innovation in the field of natural language processing and named entity recognition is significant and being carried out with high mathematical theory time and again. Every year the flows and NLP based algorithms are being optimized with different research carried out in the different domains and subdomains. In the year 2016 under an idiosyncratic domain about named entity recognition was developed by Sebastian Arnold.

3. METHODOLOGY

A. Data Preparation

Labeled Data Preparation: We have created our own dataset throughout the project. We have created data sets in the text and CSV formats. Preparing the dataset has always been a demanding part of the system. Many code elements in code snippets are defined by question-askers or answerers for illustration purposes, and these code elements do not refer to software-specific entities that other developers are concerned with. However, we keep small code elements embedded in the post texts that are surrounded with code and /code tags. These small code elements often refer to APIs, programming operators, and simple user-defined code elements for explanation purposes. Removing them from the texts will impair the sentence's completeness and meaning. Finally, we strip all other HTML tags from the post texts.

2) Unlabeled Data Preparation: As mentioned in Section II, we use unlabeled Stack Overflow data to compensate for the small-sized human-labeled data. We randomly select a huge-sized data consisting of more than 7 million Stack Overflow posts from 1.8 million Stack Overflow discussion threads tagged with

the 8 most frequently used tags (java, c#, javascript, php, python, html, android, and jquery). Again, the number of the posts selected for a particular Stack Overflow tag is proportional to the tag's usage frequency. We refer to this dataset as unlabeled data, as it will be fed into unsupervised word clustering to learn word representations (i.e., word bitstrings). The word representations will in turn be used as features for training a CRF model. Data pre-processing steps on these huge-sized unlabeled Stack Overflow texts are the same as the above steps on the labeled data.

3) External Knowledge Resources: By authentic, it means that every phrase in the gazetteer should be an entity. We can use gazetteers as features for training a CRF model. While there are many gazetteers publicly available for common person names, locations, organizations, products, temporal expressions, there are no gazetteers that can help to recognize software-specific entities in software engineering texts. We contribute a set of software-specific gazetteers, including a comprehensive list of programming languages, a list of platforms, a variety of API names covering popular programming languages, a list of community-recognized software tools, libraries and frameworks, and software standards. For programming languages, we derive notable languages in existence from Wikipedia list 1. For platforms, we obtain the gazetteer from several Wikipedia lists, including computing platforms. 2. List of operating systems 3. List of instruction sets 4. List of mobile platforms 5. We crawl the API names as defined in Table I from the official websites of the studied programming languages (Java, JavaScript, PHP, C#, Python, HTML), platform (Android), and library (jQuery).

B. Customized Tokenization

Tokenizers designed for general English texts cannot properly handle software engineering social content which is both social and technical. We develop a domain-specific tokenizer for handling texts with software-specific entities. The tokenizer uses regular expressions to match valid URLs, at-mentions, and emoticons (e.g., :), :))). The tokenizer does not split the name of a software entity, e.g., the name of an API. It does not split valid programming operators, such as "==" and "!=". It considers separate parentheses, i.e., '(' and ')', as punctuations. However, parentheses, as well as dot, #, and $, that appear in an API are considered as part of the API itself. In Table III, we show an example of the S-NER's tokenization results. Line 1 is the input sentence. Line 2 is the tokenization done by Stanford Tokenizer, which is designed for general English texts. Line 3 is tokenized by S-NER. As we can see, S-NER is able to tokenize the Java API Thread. sleep() as a whole, while the tokenizer for general texts splits the API name into 5 tokens [4].

C. Human Entity Annotation

For annotation, we use Brat, a web-based annotation tool. We adopt the widely used BIO representation of text chunks [9]. In our context, BIO means the Begin, Inside, and Outside of an entity. Take the 5-token sentence "Apache ant is a tool" as an example. The correct annotation is "B-Fram I-Fram O O O", where "Fram" is the annotation tag as shown in Table I, and B and I indicate the Begin and Inside of the text chunk. This means that the phrase "apache ant" refers to a framework, while the last three words are not entities. The annotation process involves 3 stages and is performed by 9 annotators who are all from computer science backgrounds with 5+ years of programming experience. Before annotation, we give all annotators a 1-hour tutorial regarding the tool usage, annotation methods, and entity categories [10]. We provide some annotation examples for the annotators to practice. The purpose is to let them reach a consensus on what kinds of entities to annotate and how [7]. In Stage 1, each annotator is assigned with some Stack Overflow posts. During this manual annotation process, we ask them to report to us when there are tokenization errors or deficiencies, and when certain tokens are hard to be labeled using the

software-specific entity categories we develop. After this stage, we use the feedback from our annotators to improve the tokenization, refine our software entity categories, and clean up the annotated data. In Stage 2, we let annotators cross-validate the data, i.e., the same set of tokens from Stage 1 is examined by a different annotator in Stage 2. In Stage 3, a final sweep to all the annotated data is made by the first, third, and forth author of this paper to improve the consistency of our annotation.

D. Unsupervised Word Clustering

To alleviate the problem of out-of-vocabulary (OOV) and lexical word variations, we rely on unsupervised word clustering to group together words that are distributionally similar. Specifically, we apply Brown Clustering on the unlabeled Stack Overflow posts (see Section IV-A2)[8]. Brown Clustering assigns words that appear in similar contexts into the same cluster. Words in the cluster are represented as a bitstring. We use Liang's implementation of Brown Clustering 6. We configure the number of clusters to 1000 and we only cluster words that appear no less than 10 times. It takes 15 hours to finish the word clustering of the unlabeled dataset on a 4-core Intel i5-4570 processor. Table IV lists some resulting word clusters and their corresponding strings. We can see that word clusters can represent semantically similar words in Stack Overflow posts [5].

4. RESULT ANALYSIS

This research is designed to demonstrate the need for a machine learning-based software-specific NER system and to test the efficiency of the software-specific feature set that is developed, given a small-sized annotated software engineering corpus.

5. CONCLUSION AND FUTURE WORK

The research problem of NER is formatted completely in the direction of software engineering rather than Machine Learning and Deep Learning Problem Statement. To design a software-specific NER system, It is Shown that one must first understands the unique characteristics of domain-specific texts that bring unique design challenges. Then, based on the understanding of these design challenges, It can be shown how to combine state-of-the-art supervised and unsupervised machine learning and NLP techniques to design an effective software-specific NER solution, which can reduce the demand for labeled data, meanwhile maintain the generality and robustness of the NER system. It builds S-NER, a semi-supervised machine learning method for NER in software engineering social content and demonstrate that S-NER significantly outperforms a well-designed rule-based NER system when applied on Stack Overflow posts. In the process of building this NER system, there is contribution of an inventory of software-specific entity categories, a corpus of labeled Stack Overflow posts, a software-specific tokenizer, a collection of software-specific gazetteers, unsupervised word clusters, and a rich and effective set of features for NER in software engineering texts. There is also an annotated dataset and trained CRF models 12 for community validation and further research. The method presented in this paper can be extended to more software engineering texts. It has a continuous effort to extract software-specific entities from different types of software engineering texts (e.g., API documentations, bug reports, Tweets), and to develop entity-centric search systems for the software engineering domain.

6. ACKNOWLEDGEMENT

I am thankful and indebted to the central library of Delhi Technological University for providing the access

to the required study materials and team at Airtel Africa Digital Labs for their support wherever and whenever required.

REFERENCES:

1. S. Arnold, F. A. Gers, T. Kilias, and A. Löser, "Robust Named Entity Recognition in Idiosyncratic Domains," 2016, [Online]. Available: http://arxiv.org/abs/1608.06757.
2. J. Shang, L. Liu, X. Gu, X. Ren, T. Ren, and J. Han, "Learning named entity tagger using domain-specific dictionary," Proc. 2018 Conf. Empir. Methods Nat. Lang. Process. EMNLP 2018, pp. 2054–2064, 2020, doi: 10.18653/v1/d18-1230.
3. D. Ye, Z. Xing, C. Y. Foo, Z. Q. Ang, J. Li, and N. Kapre, "Software-Specific Named Entity Recognition in Software Engineering Social Content," pp. 90–101, 2016, doi: 10.1109/saner.2016.10.
4. S. R. Kundeti, J. Vijayananda, S. Mujjiga, and M. Kalyan, "Clinical named entity recognition: Challenges and opportunities," Proc. - 2016 IEEE Int. Conf. Big Data, Big Data 2016, pp. 1937–1945, 2016, doi: 10.1109/BigData.2016.7840814.
5. C. Janarish Saju and A. S. Shaja, "A survey on efficient extraction of named entities from new domains using big data analytics," Proc. - 2017 2nd Int. Conf. Recent Trends Challenges Comput. Model. ICRTCCM 2017, pp. 170–175, 2017, doi: 10.1109/ICRTCCM.2017.34.
6. S. Ghannay et al., "End-To-End Named Entity and Semantic Concept Extraction from Speech," 2018 IEEE Spok. Lang. Technol. Work. SLT 2018 - Proc., pp. 692–699, 2019, doi: 10.1109/SLT.2018.8639513.
7. V. Kedia, A. Bhatia, S. R. Regmi, S. Dugar, K. Jha, and B. K. Shah, "Time Efficient IOS Application For CardioVascular Disease Prediction Using Machine Learning," no. Iccmc, pp. 869–874, 2021.
8. C. Sun and Z. Yang, "Transfer Learning in Biomedical Named Entity Recognition: An Evaluation of BERT in the PharmaCoNER task," pp. 100–104, 2019, doi: 10.18653/v1/d19-5715.
9. E. Alfonseca and S. Manandhar, "An Unsupervised Method for General Named Entity Recognition and Automated Concept Discovery," Proc. 1st Int. Conf. Gen. WordNet Mysore India, vol. 69, no. 6, pp. 1–9, 2002, doi: 2682189.
10. S. Tomori, T. Ninomiya, and S. Mori, "Domain specific named entity recognition referring to the real world by deep neural networks," 54th Annu. Meet. Assoc. Comput. Linguist. ACL 2016 - Short Pap., pp. 236–242, 2016, doi: 10.18653/v1/p16-2039.