Software-specific Part-of-Speech Tagging: An Experimental Study on Stack Overflow

Deheng Ye, Zhenchang Xing, Jing Li, and Nachiket Kapre
School of Computer Engineering
Nanyang Technological University, Singapore
ye0014ng@e.ntu.edu.sg, {zcxing, jli030, nachiket}@ntu.edu.sg

ABSTRACT

Part-of-speech (POS) tagging performance degrades on out-of-domain data due to the lack of domain knowledge. Software engineering knowledge, embodied in textual documentations, bug reports and online forum discussions, is expressed in natural language, but is full of domain terms, software entities and software-specific informal languages. Such software texts call for software-specific POS tagging. In the software engineering community, there have been several attempts leveraging POS tagging technique to help solve software engineering tasks. However, little work is done for POS tagging on software natural language texts.

In this paper, we build a software-specific POS tagger, called S-POS, for processing the textual discussions on Stack Overflow. We target at Stack Overflow because it has become an important developer-generated knowledge repository for software engineering. We define a POS tagset that is suitable for describing software engineering knowledge, select corpus, develop a custom tokenizer, annotate data, design features for supervised model training, and demonstrate that the tagging accuracy of S-POS outperforms that of the Stanford POS Tagger when tagging software texts. Our work presents a feasible roadmap to build software-specific POS tagger for the socio-professional contents on Stack Overflow, and reveals challenges and opportunities for advanced software-specific information extraction.

CCS Concepts

•**Software and its engineering** → Software maintenance tools; •**Information systems** → Social networking sites;

Keywords

Mining software repositories; information extraction; natural language processing; part-of-speech tagging.

SAC 2016, April 04-08, 2016, Pisa, Italy

 ISBN 978-1-4503-3739-7/16/04…$15.00
DOI: http://dx.doi.org/10.1145/2851613.2851772

1. INTRODUCTION

Software engineering is a knowledge-intensive work. With the advent of Web 2.0, programming-specific Q&A sites, such as Stack Overflow, become a prominent platform where developers learn and share knowledge. In fact, through years of accumulation, Stack Overflow has grown into a tremendous repository of user-generated content that complements traditional technical documentations [21]. The questions, answers and comments in Stack Overflow are being used extensively by developers for knowledge acquisition [15].

Stack Overflow posts [1] are in the form of textual discussions. It would greatly benefit developers' information seeking process if we could have advanced information extraction techniques dedicated for exploiting the knowledge base of Stack Overflow, such as providing direct answers to common programming issues, named entity extraction in search queries, etc. To enable these advanced down-stream applications, we must first have a part-of-speech (POS) tagger tailored specifically for the emerging genre of Stack Overflow texts.

Part-of-speech (POS) tagging is a foundational step in modern natural language processing (NLP) pipelines. It is applicable to many NLP tasks including named entity recognition and information extraction [22]. However, when applying existing POS taggers on software artifacts that are also written in natural language, the tagging performance drops significantly. For example, the accuracy of the NLTK Tagger [2] decreases from 97% on regular English corpus to only 85% when tagging software bug reports, as studied in [25]. Therefore, we need a domain-specific POS tagger suitable for processing software-specific corpus. However, little work has been done for building a software-specific POS tagger for software engineering research.

In this paper, we perform an experimental study using the textual content of Stack Overflow posts to build a software-specific POS tagger. We are faced with two challenges.

First, the discussions among developers on Stack Overflow are both *technical* and *social*. Some example Stack Overflow posts are shown in Table 1. We highlight tokens that are tagged incorrectly by the State-of-the-art tagger (Stanford Tagger v3.5.2) in boldface. On the one hand, software-specific concepts, terminologies and entities are extensively

[1] In this paper, a Stack Overflow post refers to a question (body and title), or an answer, or a comment.
[2] http://www.nltk.org/

	Post Type	Post ID	Texts	State-of-the-art Tagger Mistakes
1	question title	8318233	JQuery **Availability** on Maven **Repositories**	1. Incorrectly tag 'Availability' and 'Repositories' as Proper Noun
2	question body	252703	What's the difference between the list methods **append()** and **extend()**?	1. Improper tokenization of APIs 2. Tag API names as Verb
3	answer	143400	at the danger getting a **down-vote python** is easier as **BASIC :-)**	1. Tag 'down-vote' as Verb 2. Tag 'python' as Common Noun 3. Tag 'BASIC' as Adjective 4. Emoticon tokenization
4	comment	271576	**@Supericy** Basically yes, but **equals** (or whatever method) has to check for **null** anyway.	1. Tag API 'equals' as Verb 2. Tag terminology 'null' as Adjective 3. At-mention recognition

Table 1: Examples of Stack Overflow Texts

referenced in the discussions, such as the words "JQuery", "Maven" and "BASIC" highlighted in Table 1. A word can have a different POS tag than normal in a software-specific context. For example, normally the POS of the word "equals" is verb, but here "equals" refers to a programming API, which is a software-specific proper noun. The API name "append()", which should be considered as one token, is split into 3 tokens, i.e., "append", "(" and ")", and "append" is tagged as a verb. On the other hand, as a social Q&A site, Stack Overflow texts are ungrammatical and noisy (typos occur regularly). Stack Overflow involves many social-media-style conversations, e.g., the usage of at-mentions, emoticons, and abbreviations.

Second, building a POS tagger for software-specific natural language corpus requires annotated software engineering corpus or empirical rules. However, while there are many well-annotated corpora for traditional POS taggers, e.g., the Penn Treebank (PTB) Project [16], there is no annotated corpus dedicated for software engineering research. Creating a software engineering corpus is inevitable for building a software-specific POS tagger, which requires substantial human efforts.

To address these challenges, we propose S-POS, an English POS tagger that is designed for processing the textual contents of Stack Overflow. Our key contributions are:

- We build a software-specific part-of-speech tagger. To achieve this, we develop a software-specific POS tagset. We customize a tokenizer targeting software artifacts. We select and annotate corpus from Stack Overflow posts. We adopt a machine learning based approach using maximum entropy Markov model (MEMM) [17], and develop features for model training.
- We evaluate our POS tagger against Stanford Tagger. We reduce the tagging error by 49.6% on POS tags that have mappings in the PTB tagset [16] when compared to pre-trained Stanford Tagger; and by 26.4% on all POS tags we used when compared to re-trained Stanford Tagger.
- We provide our annotated corpus and trained models to the software engineering community for validation and further research.

The paper is structured as follows. We review related work in Section 2. Section 3 presents the system working flow of our part-of-speech tagging tool. We describe our experiments in Section 4, followed by discussions in Section 5. We conclude our paper in Section 7.

2. RELATED WORK

2.1 POS Tagging in Software Engineering

In the software engineering community, there have been several research efforts that leverage POS tagging to help solve software engineering tasks, such as source code comprehension [11, 2, 5, 8], and software terminologies identification [7, 24].

Capobianco et al. [7] identify nouns using an existing POS tagger (TreeTagger [23]) to improve the performance of traceability recovery. Similarly, Shokripour et al. [24] determine words' POS from bug reports to create a link to source code files, also using an existing tagger (Annie Tagger [12]). In [2, 5, 8], researchers apply POS tagging technique to source code identifiers for better program comprehension. By comparison, we build a software-specific POS tagger dedicated for processing software-specific natural language in Stack Overflow that is both social and technical.

In [11], Gupta et al. build a rule-based POS tagger based on WordNet [19] and some common naming conventions for processing code identifiers. In contrast, our POS tagger processes developers' discussions on Stack Overflow, rather than source code. And we use state-of-the-art machine learning approach instead of traditional rule-based approach.

Tian et al. [25] compare the effectiveness of 7 popular POS taggers on bug reports. They show that the accuracies of these taggers on bug reports drop significantly. However, they only compare the performance of existing POS tagging tools when applied to software texts, while we build a working software-specific POS tagger.

2.2 Domain Adaption of POS Tagging

Our work also falls into the broad category of domain adaption of POS tagging. For this line of research, Gimpel et al. [9] customize a POS tagger for Twitter, a genre of social media texts. Owoputi et al. [20] use word clustering to further improve the tagging accuracy of [9]. Jiang et al. [13] propose a cascaded linear model for joint Chinese word segmentation and POS tagging. Lynn et al. [14] perform POS tagging for social texts written in Irish. Compare to these

work, we contribute a POS tagger designed for processing software-specific texts on community Q&A sites.

3. THE S-POS SYSTEM

3.1 Overview

To build a software-specific POS tagger handling Stack Overflow texts, the preliminaries involve:

- define a tagset that is suitable to describe the socio-professional discussions on Stack Overflow (Section 3.2).
- determine the POS of certain tokens based on the context of discussion and software domain knowledge (Section 3.4 and Section 3.5).

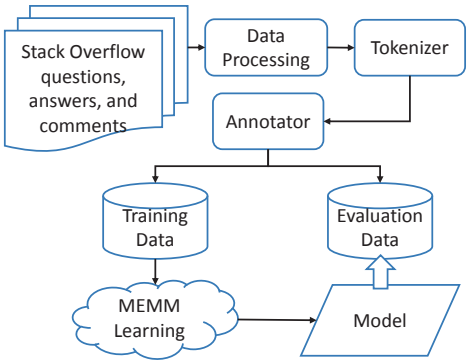

Figure 1: System Working Flow

In Figure 1, we show the system working flow of S-POS. We start with data selection from Stack Overflow, we preprocess the data and feed it into our custom tokenizer. Then, the tokenized data is distributed to annotators for tag labeling. After annotation, the data is randomly divided into two parts, one for model training, the other for model evaluation. We use a machine learning based approach for POS tagging, rather than relying on empirical rules, which have been used in POS tagging for source code [11]. Rule-based approach is possible for source code because legal code follows strict syntax. However, it is almost impossible to develop a robust rule set to build a POS tagger for Stack Overflow texts due to the discussions' social nature and informality. A word can have many variations in Stack Overflow texts. For example, we have seen the programming language "JavaScript" being written as "js", "JS", "javascript", "Javascript", or even a wrongly spelled "javasript" in Stack Overflow discussions.

Next, we describe our software-specific tagset, and then discuss each step in the working flow in detail.

3.2 Software-Specific Tagset

Table 2 shows the tagset we use to tag the POS of tokens in Stack Overflow texts. Compared to the default PTB tagset [3], this tagset is simpler for annotators to learn, while is also informative. Our tagset is a domain adaption of the tagset used in [9]. Most of the tags in Table 2 are groupings of PTB tags [16]. The tag mappings between PTB tagset and our tagset are summarized in the second column of Table 2. For example, the PTB tags describing a common noun include 'NN' and 'NNS', which are mapped to our tag 'N'. We choose to use compound tags ('Z', 'S', 'L', 'M', 'Y') [9], as we find there are many nonstandard spellings in Stack Overflow similar to Twitter. We also compile a set of Stack Overflow-specific tags, including URLs, programming operators, code elements, emoticons and at-mentions.

Tag	Description + TagSet Mapping to PTB Tags	Examples
O	pronoun (PRP, WP)	you it me
N	common noun (NN, NNS)	website outputs
^	proper noun (NNP, NNPS)	android c# ajax
V	Verb (V*, MD)	may is run ran
A	adjective (J*)	nice nicer nicest
R	adverb (R*, WRB)	when never safely
!	interjection (UH)	YEA haha ok
D	determiner (WDT, DT, WP$, PRP$)	a the which its
P	subordinating conj. pre(post)position (IN,TO)	while to for
T	verb particle (RP)	out off up Up
X	existential *there* (EX), predeterminers (PDT)	there both
&	coordinating conj. (CC)	and but
$	numeral (CD)	five 5 1.2
,	punctuation (. , : ( ))	' , ! ( ) : ; . ?
G	misc, abbr., garbage, incorrect tokenizations (FW, POS, SYM, LS)	i.e., %, (]
Z	^ + possessive	java's android's
S	**O**/**N** + possessive	book's applet's
L	**O**/**N** + verbal	what's he's I'm
M	^ + verbal	Java'll
Y	**X** + verbal	there's all's
=	programming operators	< != ==
C	self-defined code element	var myClass foo
@	at-mention	@user123
U	URL references	https://github.com
E	emoticon	:) :-)

Table 2: Software-specific POS Tagset. In the second column, the tags in parentheses are the POS tags in the PTB tagset.

Generally, a proper noun refers to a unique entity. For example, the name of a person and the name of an organization are defined as proper nouns. In the world of software engineering, the names of many software entities are unique. An API in software engineering is analogical to a person in life. A programming platform that hosts the execution of different APIs is like an organization where each of its components works collaboratively. Therefore, in this paper, we define that programming languages (e.g., C, JavaScript), API names (including the name of package,

[3]Full list of PTB tags: https://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html

class, interface, method), platform names (e.g., Android), library names (e.g., JUnit), framework names (e.g., Spring), data format names (e.g., JSON), protocols (e.g., HTTP), as proper nouns, because they are all unique entities in software engineering context. We consider domain terminologies, programming operations and concepts as common nouns. For example, in the phrase "java plugin", the programming language "java" is a proper noun, while the domain term "plugin" is considered as a common noun.

3.3 Dataset Preparation

This paper aims to implement a software-specific POS tagger using supervised learning. We use questions that tagged with the <java> tag and the answers and comments of these questions for an experimental study. Specifically, we randomly select 525 Stack Overflow posts (26,144 tokens after data preprocessing and tokenization) that are tagged with <java> from Stack Overflow official data dump released on March 16th, 2015. Note that only Stack Overflow questions have explicit tags. To obtain the Stack Overflow tags for an answer or a comment, we retrieve the question it links to and check the tags of the question. Stack Overflow posts are usually tagged with more than one tag. The tag <java> can cooccur with other tags, e.g., <javascript>, <android>, etc. Therefore, these <java>-tagged posts cover diverse knowledge that are not restricted to purely Java.

In the official data dump of Stack Overflow, standalone code snippets are surrounded with HTML tags <pre> and </pre>. We remove these code snippets, because the tokens of standalone code snippets do not have much POS information. But we keep those small code elements embedded in the discussion texts that are surrounded with <code> tags. These small code elements mainly consist of APIs, programming operators and simple self-defined variables for explanation purpose. Removing them will impair the sentence's semantic meaning. Finally, we strip all other HTML tags from the discussion texts.

3.4 Customized Tokenization

Our tokenizer is designed to handle texts with software-specific entities. We do not split contractions or possessives in compound tags shown in Table 2. Wo do not split valid programming operators. We use regular expressions to match valid URLs, at-mentions, and emoticons. We consider separate parentheses, i.e., '(' and ')', as punctuation. But parentheses, as well as dot, #, and $, that appear in an API are considered as part of the API itself. In Table 3, we show an example of S-POS's tokenization results. Line 1 is the input sentence. Line 2 is the tokenization result of Stanford Tagger. Line 3 is the tokenization done by our S-POS tagger. As we can see, our tagger keeps compound words as they are, and is able to recognize the Java API *Thread.sleep()* as a whole.

3.5 Annotation

After tokenization, the data is ready for annotation. The annotation process involves 3 stages and is performed by 9 annotators who are all from computer science background with 4+ years of programming experience. To reduce the amount of human efforts, we first pre-tag all tokens with Stanford Tagger using its *english-bidirectional-distsim* model. Then, we convert the PTB tags tagged by Stanford Tagger to our tags based on the mapping rules in Table 2. In this case, annotators only have to correct the mistakes made by Stanford Tagger, instead of annotating every token from scratch. Before annotation, we give all annotators a 1-hour tutorial on how to annotate the tokens to reach a consensus.

Input Sentence	What's␣the␣equivalent␣of␣java's␣Thread.sleep()␣in␣js?
Stanford Tagger	What␣'s the␣equivalent␣of␣java␣'s␣Thread␣.␣sleep␣(␣)␣in␣js␣?
Our S-POS	**What's**␣the␣equivalent␣of␣**java's**␣**Thread.sleep()**␣in␣js␣?

Table 3: An Example of Our Tokenzation

In Stage 1, each annotator is assigned with some Stack Overflow posts. We provide them a Web interface for annotation. 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 tagged using our tagset. After this stage, we use the feedbacks from our annotators to improve the tokenization, refine our tagset, and clear up the annotated data. In Stage 2, we let annotators cross validate the data, i.e., the same set of tokens from Stage 1 will be examined by a different annotator in Stage 2. In Stage 3, a final sweep to all the annotated data is made to improve the consistency of the tagging. We finally select 50 Stack Overflow posts (876 tokens) from the prepared dataset randomly and re-annotate them from scratch. The purpose is to estimate the inter agreement of the annotations among different annotators. We compare these 876 re-annotated tokens with their annotation results in Stage 2. Only 49 tokens are tagged differently. This results in an inter agreement rate of 94.4%.

3.6 Supervised Learning

3.6.1 Model

Our tagging model is a linear maximum entropy Markov model (MEMM) [17]. We use it due to its efficient training capability and suitability for small labeled training set [20]. We adopt the implementation of MEMM in [20]. When training a MEMM, the search method for optimization used is OWL-QN [3].

Given a sequence of observations (tokens) $O_1, ..., O_n$, we label them with tags (from our tagset) $T_1, ..., T_n$, so as to maximize the conditional probability $P(T_1, ..., T_n|O_1, ..., O_n)$. This probability is factored into Markov transition probabilities in MEMM:

$$P(T_1, ..., T_n|O_1, ..., O_n) = \prod_{i=1}^{n} P(T_i|T_{i-1}, O_i) \quad (1)$$

For a certain i, the probability is modeled as a maximum entropy classifier [4]:

$$P(T_i|T_{i-1}, O_i) \propto \exp(\sum_a \lambda_a f_a(O_i, T_i)) \quad (2)$$

where $f_a(O_i, T_i)$ are features for model training, consisting of transition features and observation features. We use tran-

sition features for every two tags. We extract observation features from the current token and its context in our annotated corpus.

3.6.2 Feature Design

We extract various observation features from our annotated corpus. We also add external resources as training features.

Orthographic features. We use regular expressions to detect URLs, at-mentions and emoticons. Word shapes are used as features. Specifically, the training model considers if a token has capitalizations, whether a token contains digits, slashes, hyphens, parentheses, etc. We use word prefixes and suffixes of character length up to 10 as features.

Lexical and contextual features. We consider every token in the training set as a feature. To utilize the context information, a window size [-1, 1] is used to add the previous and the next one token as features. We also tried setting the window size as [-2, 2] or [-3, 3], but found it not helpful for performance improvement.

Traditional tag dictionary. We add features for all tags in our tagset that each word occurs with in the corpora used in the PTB project [16]. This is the same treatment as Twitter-specific POS tagging in [9]. We use this traditional tag dictionary as soft constraints during model training.

Stack Overflow tag dictionary. Stack Overflow's official data dump provides all tags that are used to label Stack Overflow posts. Note this tag is not the POS tag. It is Stack Overflow's categorization of its own posts, such as <java>, <javascript>, <spring>, etc. Each of these Stack Overflow tags has its own POS. For example, the Stack Overflow tag <java> is a proper noun, while <debugging> is a common noun. We collect Stack Overflow tags that coexist with the tag <java>, manually identify tags that are proper nouns, and put them into a dictionary. Note Stack Overflow tags are all written in lower-case. We revert these proper noun tags into their formal written forms. For example, "java" should be written as "Java" in formal cases, and "javascript" should be "JavaScript". We add this Stack Overflow tag dictionary into MEMM training as hard constraints.

4. EVALUATION

In this section, we compare our S-POS tagger against Stanford Tagger v3.5.2. Our experiments are designed to demonstrate the need of a software-specific POS tagger, and to test the efficacy of our feature set for POS tagging given a small-sized annotated corpus.

4.1 Experimental Setup

The total number of tokens in our annotated corpus is 26,144 (525 Stack Overflow posts). In Table 4, we give the details of this corpus. As we can see, the most used part-of-speeches in Stack Overflow texts are verb and common noun, which accounts for 18.2% and 16.5% of the total number of tokens, respectively. The proportion of proper nouns used in Stack Overflow is 6.9%, which is higher than that of another genre of social texts – Twitter (the proportion is 6.4% according to [9]). This can be explained by the fact that Stack Overflow discussions are centered around programming issues of software entities.

We randomly divide our annotated corpus into a training set with 265 posts (13,196 tokens), and an evaluation set (260 posts, 12,948 tokens), which is further divided into a development set (100 posts, 5,535 tokens) and a test set (160 posts, 7,413 tokens). The training set is used to train a MEMM model. The development set is used for tuning regularization parameters during MEMM training. The test set is used for testing the accuracy of the trained model.

We first compare S-POS with pre-trained Stanford Tagger, then we compare it with re-trained Stanford Tagger. To compute the tagging accuracy of S-POS or Stanford Tagger, we let it tag the unannotated version of the evaluation corpus, and compare the tagging results with the annotated corpus. For all the experiments comparing re-trained Stanford Tagger and S-POS, the training data and evaluation data used by both taggers are always the same.

4.2 Comparisons

4.2.1 Compare against Pre-trained Stanford Tagger

Pre-trained Stanford Tagger uses PTB tagset, while S-POS uses the tagset in Table 2. So, we cannot compare the two taggers directly. However, our tags and PTB tags have mappings to each other, except those compound tags and Stack Overflow-specific tags, as shown in Table 2.

To ensure unbiased comparison, we only compare the tagging accuracy of those tags that have direct mappings between PTB tagset and our tagset [4]. Such comparison is similar to the POS tagging comparison in the study of named entity recognition in tweets [22]. For Stanford Tagger, we use its pre-trained *english-bidirectional-distsim* model to tag the unannotated version of our corpus and then translate the resulted PTB tags into the tags in our tagset.

In Table 5, we show the comparison results between S-POS and pre-trained Stanford Tagger. 10,622 tokens in the evaluation set have found direct mappings to PTB tags. As we can see, for the test set, the tagging accuracy of our S-POS is 94.1% over these mapped tags, which improves pre-trained Stanford Tagger by 5.8 percentage points. The error reduction, computed as $(0.941 - 0.883)/(1 - 0.883)$ is as high as 49.6%. A similar level of error reduction (48%) is obtained for the development set.

We further look into the tagging accuracy of individual tags. In Figure 2, we show the most common errors made by the pre-trained Stanford Tagger in the evaluation set (development test and test set), and compare S-POS's tagging error rate against that of the Stanford Tagger. We see that one of the most significant error reductions occurs for the case of proper nouns. Pre-trained Stanford Tagger performs poorly in recognizing software-specific proper nouns, its error rate for proper nouns is 46%. In contrast, S-POS reports an error rate of 12% when tagging proper nouns. However, 12% error rate is still higher than S-POS's performance over all

[4]Mapped tag list includes: O N ^ V A R ! D P T X & $. For fair comparison, we choose not to map G (garbage), because the PTB tags FW, POS, SYM, LS cover only a subset of G. We do not map punctuations, because Stanford Tagger escapes some punctuation characters like parentheses.

Tag	O	N	^	V	A	R	D	P	&	$	,	G	T X !	Comp. Tags[1]	SO Tags[2]
Count	1601	4326	1809	4764	1209	1340	2809	3173	660	175	3208	229	199	343	299
%	6.1%	16.5%	6.9%	18.2%	4.6%	5.1%	10.7%	12.1%	2.5%	0.7%	12.3%	0.9%	0.8%	1.3%	1.1%

[1]Compound tags [9]: Z S L M Y [2]Stack Overflow-specific tags: = C @ U E

Table 4: Proportions of Different Tags in the Annotated Corpus

System	Accuracy		Error Reduction	
	Devel.	Test	Devel.	Test
Stanford Tagger	87.5%	88.3%	-	
Our S-POS	93.5%	**94.1%**	48.0%	**49.6%**

Table 5: Comparing Tagging Accuracy with Pre-trained Stanford Tagger. The number of tokens that have direct mappings in the PTB tagset is 10,622.

mapped tags (7.6% error rate), which implies that there is still space to enhance the identification of proper nouns. A potential future work is to incorporate more domain knowledge, such as adding tag dictionaries of official Oracle Java APIs, Android official APIs into the feature set, to help with proper noun identification. We observe similar magnitudes of error reductions for common noun (N) and verb (V). The tagging performances over them are both improved from around 10% error rate to slightly more than 5%. Other non-trivial error reductions include the cases of interjections (!) and predeterminers (X), but the number of tokens with these two tags only accounts for a very small proportion of the tokens in total, as we can see from Table 4.

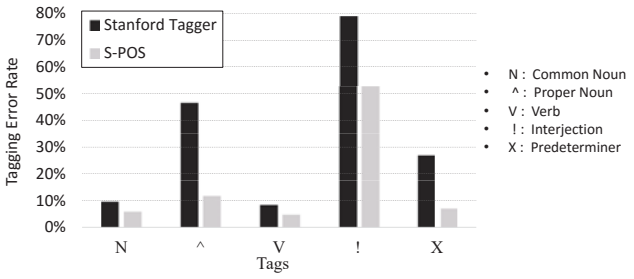

Figure 2: Tagging Error Rate for Individual Tag Classes. We list out the most common tagging errors S-POS reduced when compared to pre-trained Stanford Tagger.

4.2.2 Compare against Re-trained Stanford Tagger

We re-train Stanford Tagger using our training set. Stanford Tagger provides a set of features and optimization options for users to configure. Similar to [14, 9], the features for re-training Stanford Tagger include: word shapes in a [-3, 3] window (*wordshapes(-3,3)*), words in a [-2, 2] window (*bidirectional5words*), prefix (*prefix(3)*), suffix (*suffix(3)*), prefix-suffix pairs (*prefixsuffix(3)*), and a word shape dictionary (*naacl2003unknowns*). For Stanford Tagger, the search method for optimization used during model training is quasi-Newton, because of the unavailability of OWL-QN in the released package [1]. This is different from our OWL-QN setting as mentioned in Section 3.6.1.

System	Accuracy		Error Reduction	
	Devel.	Test	Devel.	Test
Stanford Tagger	90.6%	90.9%	-	
Our S-POS	93.1%	**93.3%**	26.6%	**26.4%**

Table 6: Comparing Tagging Accuracy with Re-trained Stanford Tagger

In Table 6, we compare the S-POS's tagging accuracy to re-trained Stanford Tagger over the tagset we defined in Table 2. For the test set, the tagging accuracy of S-POS is 93.3%, which outperforms that of the re-trained Stanford Tagger (90.9%). The error reductions for the test set and for the development set are 26.4% and 26.6%, respectively.

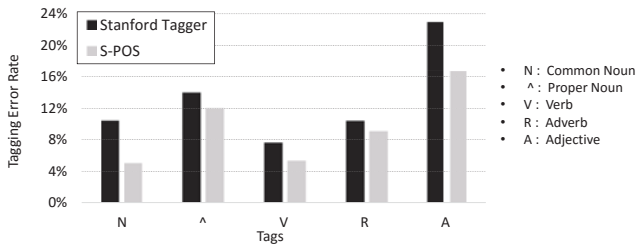

Figure 3: Tagging Error Rate for Individual Tag Classes. We list out the most common tagging errors S-POS reduced when compared to re-trained Stanford Tagger.

The tagging error rates of re-trained Stanford Tagger and S-POS on individual tag classes are shown in Figure 3. We see that the most common incorrect tag classes made by re-trained Stanford Tagger are different from those shown in Figure 2. We observe S-POS improves the tagging accuracies of adjective (A) and adverb (R) over Stanford Tagger. For the tagging of common noun (N), re-trained Stanford Tagger delivers a similar performance to its pre-trained counterpart (both report around 10% error rate). We can also see that re-trained Stanford Tagger improves its tagging accuracies on proper noun (^) and verb (V), but is still less accurate compared to S-POS.

4.2.3 Feature Ablation

We also perform independent feature ablation experiments to study the effect of individual feature(s) on tagging accuracy. In Table 7, we ablate one kind of feature(s) at one time from our full feature set and test the resulting accuracy. We find that adding traditional tag dictionary and Stack Overflow tag dictionary can boost tagging accuracy by 1.6 and 0.7 percentage points, respectively. Prefixes and suffixes features are the most important features in tagging Stack Overflow texts, without which the tagging accuracy

drops to 91.3%. Contextual features (next word and previous word) turn out to have less impact on the accuracy. Overall, we see that the absence of one particular feature does not impair the tagging accuracy significantly. S-POS's tagging performance is contributed by the combined action of all its features.

Features	Accuracy
S-POS full-feature	**93.3%**
- without traditional tag dict.	91.7%
- without Stack Overflow tag dict.	92.6%
- without word shapes	92.2%
- without prefixes and suffixes	91.3%
- without contextual features	93.1%

Table 7: The Effects of Individual Feature(s)

4.2.4 *Varying Training Data Size*

We have shown that S-POS outperforms Stanford Tagger when tagging Stack Overflow texts. The amount of annotated tokens we used for model training in previous experiments is fixed. In this section, we want to further understand whether the tagging accuracy of S-POS could exceeds that of Standford Tagger using a different amount of training data.

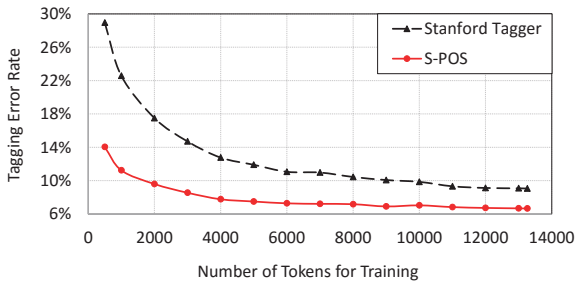

Figure 4: Tagging Accuracy as a Function of Training Data Size

In Figure 4, we vary the size of the training data and show the corresponding tagging error rate for both Stanford Tagger and S-POS. The results indicate that, generally it holds that tagging error rate decreases monotonically with the increase of training data. Our S-POS tagger consistently outperforms Stanford Tagger under different training data sizes. The smaller the training data size, the larger accuracy improvements S-POS delivers compared to Stanford Tagger. When the training data size grows up to more than 10,000 tokens, tagging error rate becomes relatively stable for both S-POS and Stanford Tagger.

5. DISCUSSION

5.1 Utilizing Unlabeled Data

Generally, increasing the amount of training data can boost tagging accuracy as we have shown in Section 4.2.4. Although we have limited annotated Stack Overflow data, we have access to enormous amount of unlabeled Stack Overflow texts. In this case, we can utilize unlabeled data through semi-supervised learning. Two popular approaches are bootstrapping [10] and word clustering [6].

Owoputi et al. [20] and Ritter et al. [22] use word clustering technique on unlabeled data and show word clustering can improve POS tagging performance. We have performed hierarchical word clustering on unlabeled Stack Overflow texts. We clustered all the Stack Overflow posts that are tagged as <java> (more than 1 million posts and more than 200 million tokens). We used the resulting word vectors as training features for MEMM. However, using word clustering impairs the tagging accuracy in our system (the accuracy decreases sharply to only 86%).

We find that word cluster can group lexical variations of words. However, it also introduces user-generated garbage and improperly tokenized tokens into the same cluster with normal words. For example, the token "i" (lower case) has been put into the same cluster as these tokens: "helloi", "buti", "function.i", "file_i", "code.i", etc. As a result, "i" is often incorrectly tagged as garbage (G), common noun (N), even preposition (P). We leave it to future work to identify ways to overcome this word clustering issue. We will also try bootstrapping method[10] using annotated data as seeds. We will compare bootstrapping with word clustering to see which approach could better assist the MEMM model when we use them over the same amount of the unlabeled data.

5.2 Towards Advanced Software-specific Information Extraction

POS tagging is a fundamental step in the pipeline of modern information extraction systems. As we have shown in Section 4.2.1, there are still needs to improve the identification of proper nouns in Stack Overflow. These proper nouns represent unique software entities, such as API names, frameworks, programming languages, etc. One potential downstream application that can be built upon our S-POS tagging tool is a software-specific named entity recognition and lining system. We could use POS of words as a feature for the model training of named entity recognition. An software-specific entity recognition and linking system would allow researchers to understand in depth the correlations between various software entities, thereby enabling more advanced software knowledge extraction and retrieval (semantic search [18]) centered around software entities.

6. CONCLUSIONS

In this paper, we presented S-POS, a software-specific part-of-speech tagging tool. We showed the challenges in building a software-specific POS tagger. We performed an experimental study using developer-generated texts on Stack Overflow to build such a tagger. We demonstrated that the accuracy of S-POS outperforms that of Stanford Tagger when tagging Stack Overflow texts. The work flow presented in this paper can be easily extended to cover more software engineering data. Our work could benefit future research on advanced software engineering knowledge extraction.

We are on a continuous effort to annotate more software data. We release our annotated corpus and trained machine learning models (available at https://drive.google.com/open?id=0ByoLWPpAxGVFX3JrWFFDSXBSNDQ) to the soft-

ware engineering community for further research.

7. ACKNOWLEDGMENTS

This work was partially supported by MOE AcRF Tier-1 grant M4011165.020 and MOE Scholarships.

8. REFERENCES

[1] Stanford tagger: Why does it crash when i try to optimize with search=owlqn? http://nlp.stanford.edu/software/pos-tagger-faq.shtml#owlqn.

[2] S. L. Abebe and P. Tonella. Natural language parsing of program element names for concept extraction. In *Program Comprehension (ICPC), 2010 IEEE 18th International Conference on*, pages 156–159. IEEE, 2010.

[3] G. Andrew and J. Gao. Scalable training of l 1-regularized log-linear models. In *Proceedings of the 24th international conference on Machine learning*, pages 33–40. ACM, 2007.

[4] A. L. Berger, V. J. D. Pietra, and S. A. D. Pietra. A maximum entropy approach to natural language processing. *Computational linguistics*, 22(1):39–71, 1996.

[5] D. Binkley, M. Hearn, and D. Lawrie. Improving identifier informativeness using part of speech information. In *Proceedings of the 8th Working Conference on Mining Software Repositories*, pages 203–206. ACM, 2011.

[6] P. F. Brown, P. V. Desouza, R. L. Mercer, V. J. D. Pietra, and J. C. Lai. Class-based n-gram models of natural language. *Computational linguistics*, 18(4):467–479, 1992.

[7] G. Capobianco, A. D. Lucia, R. Oliveto, A. Panichella, and S. Panichella. Improving ir-based traceability recovery via noun-based indexing of software artifacts. *Journal of Software: Evolution and Process*, 25(7):743–762, 2013.

[8] J.-R. Falleri, M. Huchard, M. Lafourcade, C. Nebut, V. Prince, and M. Dao. Automatic extraction of a wordnet-like identifier network from software. In *Program Comprehension (ICPC), 2010 IEEE 18th International Conference on*, pages 4–13. IEEE, 2010.

[9] K. Gimpel, N. Schneider, B. O'Connor, D. Das, D. Mills, J. Eisenstein, M. Heilman, D. Yogatama, J. Flanigan, and N. A. Smith. Part-of-speech tagging for twitter: Annotation, features, and experiments. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 42–47, 2011.

[10] S. Goldman and Y. Zhou. Enhancing supervised learning with unlabeled data. In *ICML*, pages 327–334. Citeseer, 2000.

[11] S. Gupta, S. Malik, L. Pollock, and K. Vijay-Shanker. Part-of-speech tagging of program identifiers for improved text-based software engineering tools. In *Program Comprehension (ICPC), 2013 IEEE 21st International Conference on*, pages 3–12. IEEE, 2013.

[12] M. Hepple. Independence and commitment: Assumptions for rapid training and execution of rule-based pos taggers. In *Proceedings of the 38th Annual Meeting on Association for Computational Linguistics*, pages 278–277. Association for Computational Linguistics, 2000.

[13] W. Jiang, L. Huang, Q. Liu, and Y. Lü. A cascaded linear model for joint chinese word segmentation and part-of-speech tagging. In *In Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics*, 2008.

[14] T. Lynn, K. Scannell, and E. Maguire. Minority language twitter: Part-of-speech tagging and analysis of irish tweets. *ACL-IJCNLP 2015*, page 1, 2015.

[15] L. Mamykina, B. Manoim, M. Mittal, G. Hripcsak, and B. Hartmann. Design lessons from the fastest q&a site in the west. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 2857–2866. ACM, 2011.

[16] M. P. Marcus, M. A. Marcinkiewicz, and B. Santorini. Building a large annotated corpus of english: The penn treebank. *Computational linguistics*, 19(2):313–330, 1993.

[17] A. McCallum, D. Freitag, and F. C. Pereira. Maximum entropy markov models for information extraction and segmentation. In *ICML*, volume 17, pages 591–598, 2000.

[18] E. Meij, K. Balog, and D. Odijk. Entity linking and retrieval for semantic search. In *WSDM*, pages 683–684, 2014.

[19] G. A. Miller. Wordnet: a lexical database for english. *Communications of the ACM*, 38(11):39–41, 1995.

[20] O. Owoputi, B. O'Connor, C. Dyer, K. Gimpel, N. Schneider, and N. A. Smith. Improved part-of-speech tagging for online conversational text with word clusters. Association for Computational Linguistics, 2013.

[21] C. Parnin, C. Treude, L. Grammel, and M.-A. Storey. Crowd documentation: Exploring the coverage and the dynamics of api discussions on stack overflow. *Georgia Institute of Technology, Tech. Rep*, 2012.

[22] A. Ritter, S. Clark, O. Etzioni, et al. Named entity recognition in tweets: an experimental study. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 1524–1534. Association for Computational Linguistics, 2011.

[23] H. Schmid. Improvements in part-of-speech tagging with an application to german. In *In Proceedings of the ACL SIGDAT-Workshop*. Citeseer, 1995.

[24] R. Shokripour, J. Anvik, Z. M. Kasirun, and S. Zamani. Why so complicated? simple term filtering and weighting for location-based bug report assignment recommendation. In *Proceedings of the 10th Working Conference on Mining Software Repositories*, pages 2–11. IEEE Press, 2013.

[25] Y. Tian and D. Lo. A comparative study on the effectiveness of part-of-speech tagging techniques on bug reports. In *Software Analysis, Evolution and Reengineering (SANER), 2015 IEEE 22nd International Conference on*, pages 570–574, 2015.