Part of Speech Tagging Java Method Names

Wyatt Olney, Emily Hill, Chris Thurber, Bezalem Lemma
Drew University
Madison, NJ 07940
Email: {wolney, emhill}@drew.edu

***Abstract*—Numerous software engineering tools for evolution and comprehension, including code search, comment generation, and analyzing bug reports, make use of part-of-speech (POS) information. However, many POS taggers are developed for, and trained on, natural language. In this paper, we investigate the accuracy of 9 POS taggers on over 200 source code identifiers taken from method names in open source Java programs. The set of taggers includes traditional POS taggers for English as well as some tuned to source code identifiers. Our results indicate that taggers tailored for source code are significantly more effective.**

I. Introduction

Numerous software engineering tools for evolution and comprehension make use of part-of-speech (POS) information, including code search [1], [11], [15], requirements traceability [5], comment generation [13], [16], program comprehension [4], [8], and analyzing bug reports [18]. However, there has not yet been a comparative study to evaluate the accuracy of existing tools on POS tagging source code identifiers, despite research which has indicated the existence of significant differences between source code language and natural language [17].

Previous work has evaluated the accuracy of some POS taggers on artifacts related to software, such as bug reports [18]. Furthermore, several taggers have been evaluated on the corpus of source code artifacts in the past, using 5 POS tags: noun, verb, adjective, adverb and closed list [9]. However, this experiment was limited to a small number of taggers which had been designed to label source code artifacts, and did not conclusively show that these taggers outperformed traditional POS tagging tools.

In this paper, we investigate the accuracy of 9 POS taggers on source code identifiers taken from method names in open source Java programs. The set of taggers includes traditional POS taggers for English as well as some tuned to source code identifiers. Our results indicate that taggers tailored for source code are significantly more effective.

II. Experiment Design

To evaluate the ability of taggers to label the parts of speech of Java method names, we first selected which taggers would be evaluated. Then, to ensure that our results were reliable, we created a unified mapping for comparing taggers which used different tagsets. Finally, we created a gold set which reflected the corpus of Java method names and compared tagger outputs to the manual tags that were given to each method name in our gold set.

A. POS Taggers

For the purposes of evaluation, taggers were selected that have high reported accuracy on a natural language corpus, have been used in prior applications of source code POS tagging, or were designed specifically for use on source code artifact. A working implementation of the tagger also needed to be readily available. For instance, the implementation for one source code-based tagger was unavailable in time for its results to be included in this study [3]. For this paper, our evaluation focused on the following taggers:

1) **Stanford Tagger:** the most commonly used tagger for English, this tagger has been applied to software artifacts [4], [18].
2) **Stanford+I:** the Stanford tagger has been used to parse source code identifiers by pre-pending an “I” to the method name [15].
3) **GATE’s Twitter Tagger:** designed to work with sparse and noisy data [7], which source code identifiers exhibit (e.g., frequent abbreviations, loose grammatical rules).
4) **RASP:** designed to robustly handle parsing errors and incomplete sentences, like those found in identifiers.
5) **Apache’s OpenNLP:** an open source POS tagger that uses the maximum entropy approach.
6) **SpaCy:** a relatively accurate tagger (91.8% [6]) written in python.
7) **University of Pennsylvania’s LTAG-Spinal Tagger:** a highly accurate tagger (97.33% [14]).
8) **SWUM:** a tagger specifically developed for source code identifiers based on possible tags of a word and its position in an identifier [10].
9) **POSSE:** Like SWUM, POSSE uses all possible POS tags for a word and the frequency of a word occurring in a particular context to identify POS tags in program identifiers [12], [9]. However, POSSE uses finer-grained rules and frequency information when determining POS information. POSSE boasts accuracy of 94% for Java identifiers [9], significantly outperforming two other state-of-the-art taggers [2], [8].

B. Tagset Unification

The taggers in this study do not all use the same *tagset*, or the same tag to represent part of speech information such as nouns or verbs. Before the comparison between different taggers could be performed, it was necessary to create a unified mapping of the 3 different tagsets used: Penn Treebank,

TABLE I: Mapping part-of-speech tags across tagsets

Description	Gold set tag	Penn Treebank	CLAWS7	SWUM/POSSE
Noun	N	NN, NNP	NN, NN, NNA, NNB, NNL1, NPM1, ND1, APPGE	N, NI, NM
Plural Noun	NP	NNS, NNPS	NN2, NNL2, NNO2, NNT2, NP2, NPD2, NPM2	NP, NPL
Pronoun	PN	PRP, PRP$, WP$	PN, PN1, PNQO, PNQS, PNQV, PNX1, PPGE, PPH1, PPHO1, PPHO2, PPHS1, PPHS2, PPIO1, PPIO2, PPIS1, PPIS2, PPX1, PPX2, PPY	PN
Determiner	DT	DT, PDT, WDT	AT, AT1, BCL, DA, DA1, DA2, DAR, DAT, DB, DB2, DD, DD1, DD2, DDQ, DDQGE, DDQV, EX	DT
Adjective	ADJ	JJ, JJR, JJS	JJ, JJR, JJT, JK	ADJ
Preposition	P	IN, TO	IF, II, IO, IW	P
Past Tense Verb	PP	VBD, VBN	VHD, VHN, VVD, VVN	PP
Verb	V	MD, VB, VBP	VB0, VD0, VH0, VM, VV0	V, VING, VB, VI
3rd Person Present Verb	V3	VBG, VBZ	VHG, VVG, VVZ	V3
Adverb	VM	RB, RBR, RBS, WRB	RL, RR	VM
Verb Particle	VPR	RP	RP	VPR
Conjunction	CJ	CC	CC, CCB, CS, CSA, CSN, CST, CSW	CJ
Digit	D	CD	MC, MC1, MC2, MCGE, MCMC, MD, MF	D
Unknown	UN	EX, FW, LS, POS, SYM, UH	FO, FU, FW, XX, ZZ1	UN, ABV

CLAWS7, and SWUM. Because the SWUM tagset is the smallest, it was necessary to map each tag from Penn Treebank and CLAWS7 to this set. Table I shows the full mapping.

C. Gold Set Creation

The experiment was conducted on two different gold sets of method names. The first is a gold set of 194 Java methods randomly selected from open source Java programs, first used to evaluate POSSE [12]. After initially analyzing the results, it was determined that POSSE was also trained on this set. To avoid threats to validity, a supplemental gold set of 53 methods was randomly selected from 3 open source Java programs used in prior evaluation [10].

Because the accuracy of the experiment relies on the gold set, the tagging of each identifier phrase was evaluated by 3 paper authors until a consensus was reached as to the proper tagging for both gold sets. The original source code context was investigated as necessary. Any identifiers containing ambiguous words to tag were discarded. None of the 3 authors involved had any knowledge of how the taggers in the study worked. The gold sets can be found at http://goo.gl/FG26JN.

D. Methodology & Measures

Each method name was first split into their constituent words by hand in the course of creating the gold set. Each method name was treated from then on as an independent phrase. Each tagger was run on a list of split method names, using default tagger configurations. Then, each tagged method name was compared to the gold set's tagging for the method.

Two separate measures were used to determine the accuracy of each tagger. The first was *per-word accuracy* of the individual phrases. This was a percentage of each phrase that a tagger correctly labelled. A second measure of accuracy was *per-identifier* accuracy. This was a binary measure for each phrase, if a tagger got the entire phrase correct.

E. Threats to Validity

There are a number of potential threats to validity in this study. The creation of the gold set is a challenging aspect of the work, with POS tags being extremely difficult to determine in some cases. We attempted to mitigate this threat as much as possible by having multiple humans look at the most challenging identifiers, and using any available source code context information.

The second threat to validity is in the alignment of POS tags. It is possible that the slightly different tagsets are not capturing precisely the same information, and so conflating these tags to a subset could result in spurious errors. We have tried to mitigate this threat to the extent possible by having multiple authors verify and cross-reference the mapping, and by making it publicly available.

Finally, the method names were all from open source Java programs. These results may not generalize to field names, class names, or identifiers from other languages.

III. Results

The per-identifier accuracy for each phrase can be seen in Figure 1. Each line represents an identifier in the gold set. A blue vertical bar indicates a correct tagging, while an orange bar indicates an error. The identifiers are sorted by the number of correct taggings, increasing from right to left. In addition to showing the results for every identifier, this figure highlights the overlap of each tagger on different identifiers. For example, SWUM and POSSE have the least orange (i.e., incorrect) identifiers. Furthermore, a combination of taggers might yield even higher accuracy, as there were very few identifiers (22) that *no* tagger correctly tagged.

A. Mean Accuracy

Figures 2a–2d present boxplots of the mean accuracy for both gold sets. In each box plot, the thick horizontal line denotes the median, the + denotes the mean, and the box indicates the interquartile range (i.e., the difference between the first and third quartiles). The whiskers extend to the maximum and minimum values of the range, with outliers marked ◦. Each boxplot is sorted by increasing mean.

Figures 2a and 2b indicate the average per-word accuracy of each tagger for POSSE's gold set and the supplemental gold

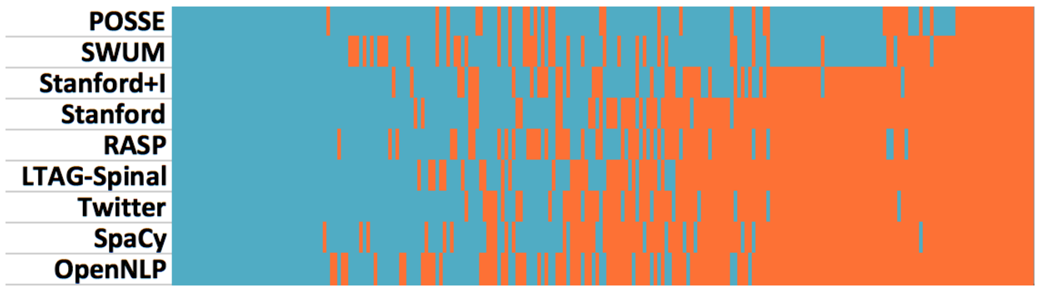
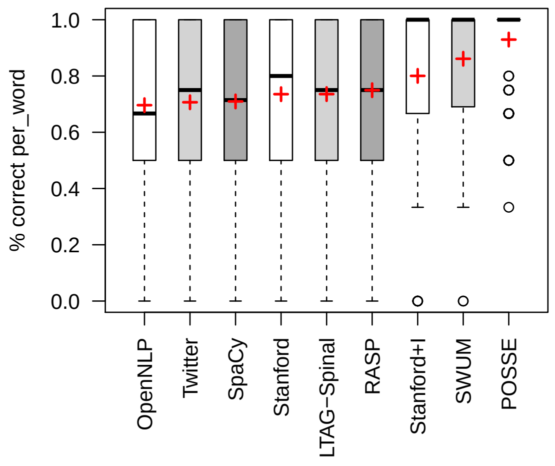
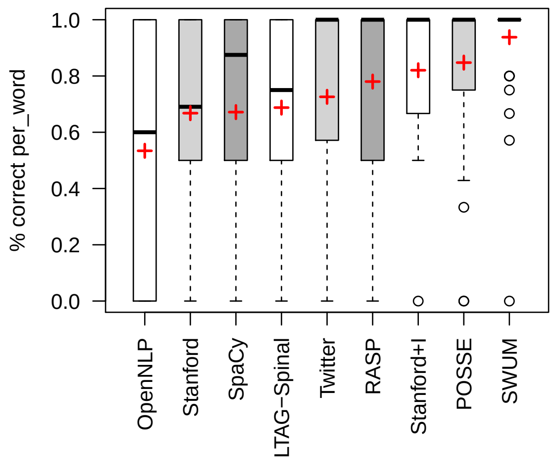


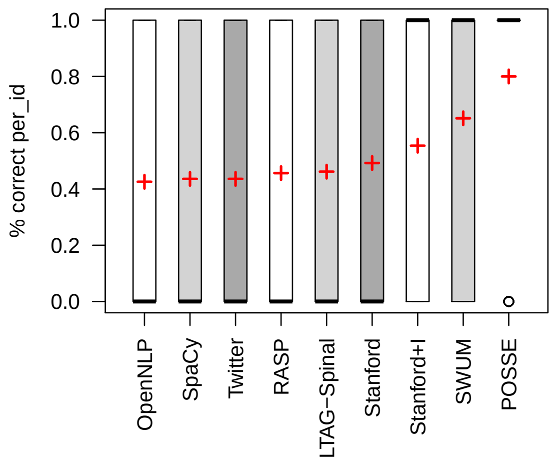
Fig. 1: Color-coded per-identifier accuracy by tagger. Blue lines are correctly tagged identifiers, orange are not.



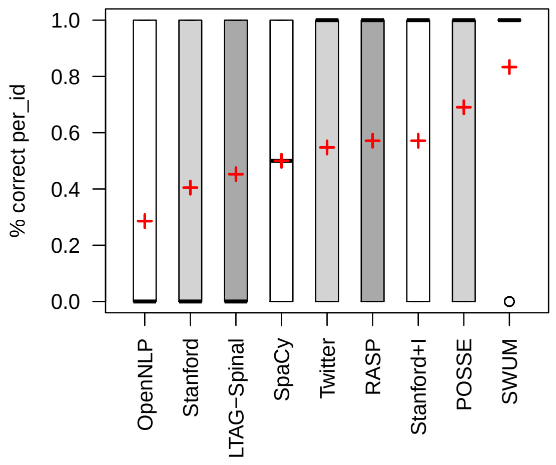
(a) Per-word accuracy on the POSSE gold set



(b) Per-word accuracy on the supplemental gold set



(c) Per-identifier accuracy on the POSSE gold set



(d) Per-identifier accuracy on the supplemental gold set

Fig. 2: Accuracy of taggers on both gold sets.

set, respectively, considering each word independently. In both cases, SWUM and POSSE were the most accurate, followed closely by Stanford+I. RASP and SpaCy have comparably good accuracy on both sets, but are not the most accurate. In contrast, the OpenNLP, LTAG-Spinal, and Twitter taggers perform poorly. For example, on the method named `fireEvent`, only SWUM, POSSE and Stanford+I correctly labeled both words in the method. This was due to most taggers recognizing "fire" as a noun, not as a verb, as would be expected, given that this is a method name. By prepending "I", the Stanford Tagger was able to recognize this as a verb.

When applying a Tukey range separation test, both POSSE and SWUM outperform every tagger to a statistically significant degree ($\alpha = .05$), except for Stanford+I. Although POSSE significantly outperforms Stanford+I, SWUM does not. RASP did perform significantly better than Open-NLP, but not more than any of the other taggers. There is no statistically significant difference between the remaining taggers.

Figures 2c and 2d indicate the average per-identifier accuracy of each tagger on whole identifier phrases for POSSE's gold set and the supplemental gold set, respectively. This measure considers each identifier phrase, rather than an individual word. For example, on the supplemental gold set, POSSE has 100% accuracy for 63% of the identifiers in the set, whereas OpenNLP only has 100% accuracy for all the words in 26% of the identifiers. From the per-identifier perspective, POSSE and Stanford+I are the most consistently accurate.

B. Results By POS Tag Mistake

To help further explain the results, Table II shows the frequency of tagging mistakes for each tagger in the study. Across all taggers, the most common mistake was differentiating between nouns and verbs, which occurred over 300 times. For example, the Stanford tagger, the Twitter tagger, and Open-NLP labelled "show" in `showInputDialog` as a noun, while the remaining taggers, including Stanford+I, all correctly labeled "show" as a verb. As another example, SWUM mis-tagged "start" in `startSocket` as a noun rather than a verb. This is likely because so many words that are traditionally nouns in English are predominantly used as verbs in method names.

Another common mistake is swapping verbs and adjectives. For example, LTAG-Spinal mis-tagged "front" in `moveToFrontAction` as a verb rather than an adjective. This is likely because "to" is more frequently a preposition in method names than an indication of an infinitive verb, as with English text.

It is interesting to note that both SWUM and POSSE have fewer verb-noun mistakes than verb-adjective mistakes as compared to the traditional English taggers. Given the prevalence of verbs in method names as compared to English text, it makes sense that these taggers would take greater care, and thus have higher accuracy, on verbs.

In Table II, it is interesting to notice that the second most common mistake is mistaking nouns for plural nouns. While this distinction may seem trivial, it may be sufficiently important for applications of POS tagging in source code.

Furthermore, despite their overall higher accuracy on method names, POSSE and SWUM perform noticeably worse than traditional POS taggers when distinguishing nouns from adjectives, with POSSE and SWUM mislabeling adjectives as nouns 22 and 30 times respectively, with the worst performing tagger for this task making only 9 such mistakes. This may be because adjectives are fairly rare within source code, so POSSE and SWUM, having been trained on source code, may be less likely to label a given word as an adjective. This suggests both a limitation in POSSE and SWUM, and that potentially by using a combination of taggers, it may be possible to improve accuracy.

IV. Conclusion

In this paper, we investigate the accuracy of 9 POS taggers on over 200 source code identifiers taken from method names in open source Java programs. We observe that taggers specifically designed for source code, POSSE and SWUM, were significantly more accurate than most other taggers. In future, it may be possible to improve traditional POS taggers by mimicking natural language structure within identifiers, as illustrated by the success of Stanford+I over the default Stanford tagger, or to create a hybrid approach that combines multiple taggers for more accurate results.

V. Future Work

Although this paper presents an early analysis of applications of POS tools to method names, these results may not generalize to other source code artifacts, and further study would be needed to test if our results generalize to method names from languages other than Java, as only Java method names were evaluated. Additionally, as was shown by prepending "I" to the method names before running the Stanford tagger, simple modifications may improve the accuracy of some taggers, and with further study, the accuracy of traditional POS taggers can be improved.

The limitations of POS taggers developed for source code, such as the aforementioned difficulty with adjectives, may be possible to mitigate in the future with further research. Finally, further research could be conducted to evaluate the accuracy of POS tags when multiple taggers are used in conjunction with each other, but such an application is beyond the scope of this paper.

References

[1] Surafel Lemma Abebe, Anita Alicante, Anna Corazza, and Paolo Tonella, *Supporting concept location through identifier parsing and ontology extraction*, J. Syst. Softw. **86** (2013), no. 11, 2919–2938.

[2] Surafel Lemma Abebe and Paolo Tonella, *Natural language parsing of program element names for concept extraction*, Proceedings of the 2010 IEEE 18th Int'l Conf. on Program Comprehension, 2010.

[3] Reem S. AlSuhaibani, Christian D. Newman, Michael L. Collard, and Jonathan I. Maletic, *Heuristic-based part-of-speech tagging of source code identifiers and comments*, IEEE 5th Workshop on Mining Unstructured Data, 2015.

[4] Dave Binkley, Matthew Hearn, and Dawn Lawrie, *Improving identifier informativeness using part of speech information*, Proceedings of the 8th Working Conference on Mining Software Repositories, 2011.

Error → Expected	POSSE	Stanford	Stanford+I	Twitter	RASP	OpenNLP	SpaCy	SWUM	LTAG-Spinal	Total
N → V	16	38	2	56	17	50	40	13	45	277
NP → N	0	30	29	28	25	30	31	0	33	206
V → N	1	21	36	13	26	12	10	20	9	148
PP → V	0	20	12	2	9	23	21	0	17	104
N → ADJ	22	7	4	9	4	8	6	30	7	97
ADJ → N	2	10	8	10	20	11	21	0	7	89
V3 → V	0	11	11	12	10	10	11	0	10	75
ADJ → V	1	6	3	7	5	13	10	0	9	54
UN → N	0	5	5	3	11	0	2	2	0	28
PP → ADJ	1	2	3	4	5	3	3	0	3	24
V3 → N	0	2	2	4	7	2	1	0	1	19
N → V3	4	1	1	2	1	1	1	5	2	18
N → PP	2	1	2	0	0	0	2	7	0	14
P → N	0	3	2	1	0	3	1	0	3	13
V → ADJ	0	0	2	0	3	0	0	6	0	11
P → V	1	1	1	1	1	1	1	1	1	9
VM → N	0	1	0	3	1	3	0	0	1	9
PP → N	0	1	1	1	2	1	2	0	1	9
N → P	5	0	0	0	0	0	0	3	0	8
N → NP	4	0	0	0	0	0	0	4	0	8
DT → N	0	0	0	0	0	1	2	1	2	6
V → PP	4	0	0	0	0	0	0	0	0	4
VM → ADJ	0	1	1	0	2	0	0	0	0	4
UN → V	0	0	0	4	0	0	0	0	0	4
PR → N	0	0	0	1	0	1	1	0	1	4
P → VM	0	1	0	1	1	0	0	0	0	3
VPR → VM	0	0	1	0	0	0	1	0	1	3
VPR → N	0	0	1	0	1	1	0	0	0	3
NP → V3	0	0	1	0	0	1	1	0	0	3
D → N	0	0	0	1	0	1	0	0	1	3
N → VM	1	0	0	0	0	0	0	1	0	2
VM → V	0	0	1	0	1	0	0	0	0	2
CJ → P	0	0	0	0	2	0	0	0	0	2
DT → ADJ	0	0	0	0	0	1	0	1	0	2
NP → V	0	0	0	0	0	1	0	0	1	2
NP → PP	0	0	0	0	0	2	0	0	0	2
UN → ADJ	0	0	0	0	0	0	0	2	0	2
N → DT	1	0	0	0	0	0	0	0	0	1
N → D	1	0	0	0	0	0	0	0	0	1
NP → ADJ	0	0	0	0	0	0	1	0	0	1
Total	66	162	129	163	154	180	169	96	155	

TABLE II: Errors by tagger on both gold sets

[5] G. Capobianco, A. De Lucia, R. Oliveto, A. Panichella, and S. Panichella, *On the role of the nouns in ir-based traceability recovery*, IEEE 17th International Conference on Program Comprehension, 2009 (ICPC '09), 2009.

[6] Jinho D Choi, Joel Tetreault, and Amanda Stent, *It depends: Dependency parser comparison using a web-based evaluation tool*, Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics, 2015.

[7] Leon Derczynski, Alan Ritter, Sam Clark, and Kalina Bontcheva, *Twitter part-of-speech tagging for all: Overcoming sparse and noisy data.*, Proceedings of Recent Advances in Natural Language Processing, 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*, 18th Int'l Conf. on Program Comprehension, 2010.

[9] S. Gupta, S. Malik, L. Pollock, and K. Vijay-Shanker, *Part-of-speech tagging of program identifiers for improved text-based software engineering tools*, IEEE 21st International Conference on Program Comprehension, 2013.

[10] Emily Hill, *A model of software word usage and its use in searching source code*, Ph.D. thesis, University of Delaware, 2010.

[11] Emily Hill, Lori Pollock, and K. Vijay-Shanker, *Automatically capturing source code context of nl-queries for software maintenance and reuse*, Proceedings of the 31st International Conference on Software Engineering, 2009.

[12] Sana Malik, *Parsing java method names for improved software analysis*, Master's thesis, University of Delaware, 2011.

[13] Yusuke Oda, Hiroyuki Fudaba, Graham Neubig, Hideaki Hata, Sakriani Sakti, Tomoki Toda, and Satoshi Nakamura, *Learning to generate pseudo-code from source code using statistical machine translation*, 30th IEEE/ACM International Conference on Automated Software Engineering, 2015.

[14] Libin Shen, *Statistical ltag parsing*, Ph.D. thesis, University of Pennsylvania, 2006.

[15] David Shepherd, Zachary P. Fry, Emily Hill, Lori Pollock, and K. Vijay-Shanker, *Using natural language program analysis to locate and understand action-oriented concerns*, Proceedings of the 6th International Conference on Aspect-Oriented Software Development, 2007.

[16] Giriprasad Sridhara, Emily Hill, Divya Muppaneni, Lori Pollock, and K. Vijay-Shanker, *Towards automatically generating summary comments for java methods*, Proceedings of the IEEE/ACM International Conference on Automated Software Engineering (2010).

[17] Giriprasad Sridhara, Vibha Singhal Sinha, and Senthil Mani, *Naturalness of natural language artifacts in software*, Proceedings of the 8th India Software Engineering Conference, 2015.

[18] Yuan Tian and David Lo, *A comparative study on the effectiveness of part-of-speech tagging techniques on bug reports*, SANER ERA, 2015.