A Combined Method for Usage of NLP Libraries Towards Analyzing Software Documents

Xinyun Cheng, Xianglong Kong(✉), Li Liao, and Bixin Li

School of Computer Science and Engineering, Southeast University, Nanjing, China
{yukicheng,xlkong,lliao,bx.li}@seu.edu.cn

Abstract. Natural Language Processing (NLP) library is widely used while analyzing software documents. The numerous toolkits result in a problem on NLP library selection. The selection of NLP library in current work commonly misses some objective reasons, which may pose threats to validity. And it is also not clear that whether the existing guideline on selection still works for the latest versions. In this work, we propose a solution for NLP library selection when the effectiveness is unknown. We use the NLP libraries together in a combined method. Our combined method can utilize the strengths of different NLP libraries to obtain accurate results. The combination is conducted through two steps, i.e., document-level selection of NLP library and sentence-level overwriting. In document-level selection of primary library, the results are obtained from the library that has the highest overall accuracy. Through sentence-level overwriting, the possible fine-gained improvements from other libraries are extracted to overwrite the outputs of primary library. We evaluate the combined method with 4 widely used NLP libraries and 200 documents from 3 different sources. The results show that the combined method can generally outperform all the studied NLP libraries in terms of accuracy. The finding means that our combined method can be used instead of individual NLP library for more effective results.

Keywords: Natural Language Processing · NLP library selection · Software document

1 Introduction

Software documents are important sources of knowledge in an information system. The documents (e.g., requirement documents, design documents or logs) are commonly written in natural language by some developers or automatically created by generators for different purposes. To obtain the information contained in software documents, many researchers directly use state-of-the-art Natural Language Processing (NLP) libraries to carry out the tasks [11,21,26]. There are

Technical paper.

S. Dustdar et al. (Eds.): CAiSE 2020, LNCS 12127, pp. 515–529, 2020.
https://doi.org/10.1007/978-3-030-49435-3_32

several high-quality NLP libraries which are widely used in both industry and academia, such as NLTK[1], spaCy[2], Stanford CoreNLP[3], Google's SyntaxNet[4], OpenNLP[5] and so on. ALL the NLP libraries can provide API supports for the common NLP tasks (e.g., tokenization and part-of-speech tagging). The rich sources of candidates result in a new problem on the selection of suitable NLP library while analyzing documents. In most cases, the selection of NLP library is subjective [22,28,29,32]. And the results from different NLP libraries may also be different, the subjective selection may pose threats to validity [17].

To solve the problem, Omran and Treude propose an empirical study to give a guideline for choosing appropriate NLP libraries [17]. The guideline is expressed as the best choice of NLP libraries for a specific type of software documents. The effectiveness of NLP libraries is influenced by the NLP task and the source. For example, spaCy performs best on Stack Overflow text with nearly 90% of tokens tagged correctly in the experiments, while it is significantly outperformed by Google's SyntaxNet when parsing GitHub ReadMe text. However, the guideline cannot always help researchers to choose the best NLP library for a specific task. There are three reasons for the possible inaccuracy. First, there exist a great variety of software documents in real world [4,13,20], and it is too tedious to produce an universal guideline for all the types of software documents. Second, the NLP libraries are optimized continuously, the existing guideline may be inaccurate in some upcoming scenarios. Third, different types of software documents may be analyzed together in some cases [6,31], the individual selection of different NLP libraries on different documents and NLP tasks may weaken the automation.

Since it is impractical to summarize an universal and continuously updated guideline of NLP library selection, we turn to investigate a method to use them together. In this paper, we propose a combined method which can utilize the strengths of different NLP libraries to achieve more effective results. The combined method is expected to generate more accurate results than single NLP library. To obtain the combined results, we firstly choose the whole results from the NLP library which has the highest overlapping degree. The overlapping degree of a NLP library defined in our work is the average proportion of identical outputs (e.g., tokens or part-of-speech tags) among the results achieved by it and any other NLP library individually. The overlapped results have high possibility to be correct due to the high quality of the studied NLP libraries. When the primary NLP library is selected in the first step, we identify the result of each sentence. The results of low-overlap sentences within a document are overwritten with the fine-gained improvements from other NLP libraries.

Our study has two main novel aspects. First, we conduct an empirical study to check whether the existing guideline fits to different versions of NLP libraries

[1] http://www.nltk.org/.
[2] https://spacy.io/.
[3] https://nlp.stanford.edu/nlp/.
[4] https://opensource.google.com/projects/syntaxnet.
[5] http://opennlp.apache.org/.

and different documents. The subjects include 4 publicly available NLP libraries and 200 documents from 3 different sources. The results confirm that the existing guideline should be updated with the evolution of NLP libraries and the change of given documents. Furthermore, each NLP library can generate its own exclusive correctness. This finding inspires us to extract correct results from different libraries to achieve a better effectiveness. The combined method for usage of NLP libraries brings the second novelty of our work. It contains two steps, i.e., document-level selection of NLP library and sentence-level overwriting. We evaluate the combined method according to a handmade benchmark in our study. The results show that our method can outperform any individual library in terms of accuracy on tokenization and part-of-speech tagging. The numeric promotion is around 2%. Due to the high quality of the studied NLP libraries, the improvements are meaningful in practice. And the majority of correct NLP results come from the outputs of primary NLP library. These findings mean that our combined method can be used instead of individual NLP library for more effective results in the analysis of software documents.

To sum up, this paper makes the following contributions.

- We conduct an empirical study to investigate the effectiveness of NLP libraries, and prove that the existing guideline should be updated with the evolution of libraries and each library has its own exclusive results.
- We propose a method to combine the strengths of different NLP libraries, and the combined method can outperform any individual NLP library in the experiments.

2 Background

In the section, we discuss some background knowledge of our method, i.e., NLP Library, NLP task and software document.

2.1 NLP Libraries and Tasks

There are a lot of NLP libraries in both academia and industry, and the publicly available libraries are very popular in the analysis of software documents. Among them, we introduce several widely used NLP libraries in this section. NLTK is a platform for building Python programs to work with human language data. It provides a suite of text processing libraries. SpaCy is another free open-source library for NLP which is called "Industrial-Strength Natural Language Processing in Python". Stanford CoreNLP is a set of open-source Java NLP tools developed by Stanford University. Google's SyntaxNet is a new open-source natural language parsing model which is developed by Google. Apache OpenNLP is a toolkit based on machine-learning for processing natural language text.

NLP tasks denote the various purposes of analyzing documents. We list 3 frequently used NLP tasks as follow. Tokenization is the process of dividing a

string into several parts [30]. The resulting tokens will be used in other tasks. When processing texts in English, tokenization of strings is impacted by the rules on processing punctuations. Stop words removal is designed to save storage space and improve search efficiency, some words or phrases will be filtered while processing natural language texts. These stop words or phrases are commonly insignificant. Part-of-speech is a basic grammatical property of a vocabulary. Part-of-speech tagging is the process of determining the grammatical category of each word in a given sentence and tagging it [5,18], the tags are usually used in syntactic analysis and information retrieval.

2.2 Software Documents

Documents in software life-cycle may contain both oral words and formal statements. The formal documents are usually generated by some approaches, and the human-written documents may have a low degree of formalization. According to the different degree of formalization, we introduce three types of software documents as below.

Feature requests are requirements of new features which are received by developers in order to modify the code and documents [26]. The documents of feature requests can be extracted from JIRA[6] (an issue and project tracking system). There is not a standard of the formalization of the language in the documents of feature requests. ReadMe Files are written by the developers to help other people understand the projects. Developers can author the files in different languages. ReadMe files are more formal than descriptions of feature requests. Java API documentation is quite different from the documents mentioned above. Java API documentation has their own structures and not as casual as the descriptions of feature requests. Moreover, Java API documentation contains many code elements which may obey the rules of grammar in daily expression. It is the official text provided by the community[7]. Therefore, it can be treated as formal software documents. In this work, we apply the four NLP libraries on three types of software documents to conduct tokenization and part-of-speech tagging.

3 Approach

It is hard to predict the accuracy of NLP libraries while analyzing software documents, especially for the people which are not professional at NLP. Since the subjective selection of NLP libraries may result in threats to the validity, and random selection may obtain unexpected NLP results, we want to propose a combined method to utilize the strengths of different NLP libraries. The overall framework of our combined method is illustrated in Fig. 1. The combined method mainly consists of two stages: document-level selection of NLP library and sentence-level overwriting. In this section, we present the details of our approach in sequence.

[6] https://www.atlassian.com/software/jira.
[7] https://docs.oracle.com/javase/7/docs/api.

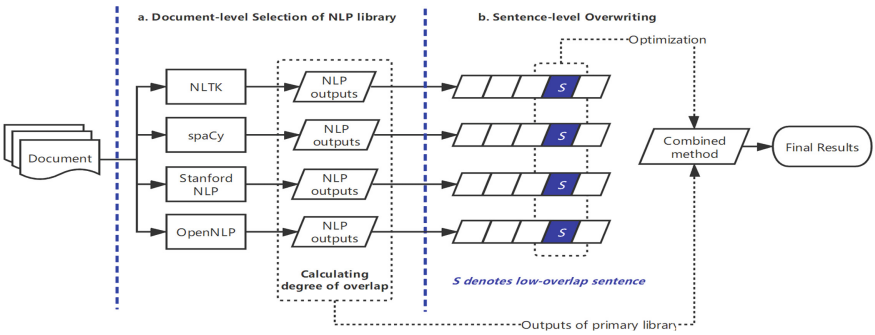

Fig. 1. Overall framework of the combined method

3.1 Document-Level Selection of NLP Library

For the given documents, we firstly apply NLTK, spaCy, Stanford CoreNLP and OpenNLP to produce four sets of results independently. Then we calculate the degree of overlap for each studied NLP library. The NLP library with highest degree of overlap is selected as primary library, its outputs are transfered to the next step as primary outputs. Document-level overlapping degree of a NLP library defined in our experiment is the average proportion of identical outputs (e.g., tokens or part-of-speech tags) among the results achieved by it and any other NLP library individually. The degree of overlap between NLP library i and j is calculated as

$$o_{i,j} = \frac{|\{Outputs_i\} \bigcap \{Outputs_j\}|}{|\{Outputs_i\}| + |\{Outputs_j\}|} \cdot 2 \tag{1}$$

where $o_{i,j}$ denotes the degree of overlap between NLP library i and j, $\{Outputs_i\}$ and $\{Outputs_j\}$ are the sets of all the annotations generated by NLP library i and j, respectively.

The document-level overlapping degree of library i is calculated as

$$O_i = \frac{\sum_{j=1}^{k-1} o_{i,j}}{k-1} \tag{2}$$

where O_i denotes the document-level overlapping degree of library i, and k is the total number of studied NLP libraries.

We show the process of calculating document-level overlapping degree through a case. Figure 2 shows an example of document from Java API Documentation. The document contains 14 sentences, and we list the 2 selected sentences in the figure. For the whole document, NLTK, spaCy, Stanford CoreNLP and OpenNLP identify 244, 246, 242 and 246 tokens, respectively. Table 1 presents the results of document-level overlapping degrees on part-of-speech tagging. In the table, the first column presents the studied NLP libraries, and overlapping degrees of the paired NLP libraries are shown in column 2 to 5. The last column

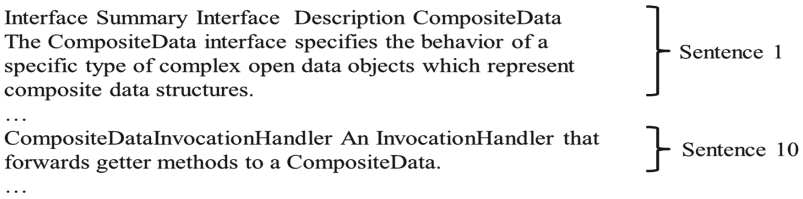

Fig. 2. Document from JavaDoc of javax.management.openmbean

presents the document-level overlapping degree. The data within parentheses in row 2 to 5 denotes the number of identical tags for each pair of NLP libraries. In this case, NLTK gets the highest document-level overlapping degree, so we select NLTK as primary library.

Table 1. Document-level overlapping degree of NLP libraries on POS tagging

Document	NLTK (244)	spaCy (246)	CoreNLP (242)	OpenNLP (246)	Overlapping degree
NLTK	–	0.89 (218)	0.84 (203)	0.87 (213)	**0.87**
spaCy	0.89 (218)	–	0.82 (199)	0.87 (214)	0.86
CoreNLP	0.84 (203)	0.82 (199)	–	0.81 (197)	0.82
OpenNLP	0.87 (213)	0.87 (214)	0.81 (197)	–	0.85

The reason for choosing a primary library is shown as follows. The studied NLP libraries are all widely used and well maintained by professional teams. So the expected effectiveness of these NLP libraries is high enough to ensure that the identical overlapping results in their outputs are highly possible to be correct. The selected NLP library in this step is supposed to have the best overall effectiveness. We also record the results of each sentence separately, which will be analyzed in the next step.

3.2 Sentence-Level Overwriting

After document-level selection, we obtain a primary NLP library which has the best overall performance. Then we try to investigate the fine-gained improvements on sentences. Due to the complex features of software documents, the primary NLP library may not perform better than other NLP libraries on every sentence within the document.

To locate the potential sentences, we calculate the overlapping degree of the primary NLP library for each sentence by treating it as an individual document. We apply the same formula as the previous step on each sentence. Once the sentence-level overlapping degree of a particular sentence is less than the document-level overlapping degree of the document it belongs to, we mark it as low-overlap sentence. The outputs of primary NLP library on low-overlap

Table 2. Sentence-level overlapping degree of NLP libraries on POS tagging

Sentence 10	NLTK (11)	spaCy (11)	CoreNLP (11)	OpenNLP (11)	Overlapping degree
NLTK	–	0.64 (7)	0.73 (8)	0.64 (7)	0.67
spaCy	0.64 (7)	–	0.82 (9)	0.64 (7)	0.70
CoreNLP	0.73 (8)	0.82 (9)	–	0.73 (8)	0.76
OpenNLP	0.64 (7)	0.64 (7)	0.73 (8)	–	0.67

sentences are highly possible to have great space for improvements. For the low-overlap sentence, we overwrite its primary outputs with another set of results from the library which has highest overlapping degree on it.

Table 2 presents the overlapping degree of sentence 10 in this above case. We can find that the primary library NLTK obtains 0.67°, which is smaller than the document-level degree (i.e., 0.87). And Stanford CoreNLP gets the highest overlapping degree on it, so we overwrite the primary outputs on sentence 10 with the annotations generated by Stanford CoreNLP. In this case, the tag of token *getter* changes from adjective to noun.

Algorithm 1: Combined usage of NLP libraries

Input : Given Document D, candidate NLP libraries $\{NLP\}$
Output: Combined NLP results

$InitialResultsSet\{InitialR\} \leftarrow$ Analyze(NLP, D);
for *each* $NLP_i \in \{NLP\}$ **do**
  for *each* $(R_i, R_j) \in InitialR$ **do**
    Document-level overlapping degree $O(d_{ij}) \leftarrow$ Compare((R_i, R_j) ;
  end
  $O(d_i) \leftarrow \sum_{j=1}^{|\{NLP\}|-1} O(d_{ij})$ / $(|\{NLP\}| - 1)$;
end
$NLP_{primary} \leftarrow \{NLP_i, \max(O(d_i))\}$;
for *each* $d \in \{D\}$ **do**
  for *each* $s \in \{d\}$ **do**
    for *each* $NLP_k \in \{\{NLP\}\text{-}NLP_{primary}\}$ **do**
      $O(s_{primary,k}) \leftarrow$ Compare($(R_{primary}, R_k)$;
    end
    $O(s_{primary}) \leftarrow \sum_{k=1}^{|\{NLP\}|-1} O(s_{primary,k})$ / $(|\{NLP\}| - 1)$;
    if $O(s_{primary}) < O(d)$ **then**
      $\{S_{improve}\} \leftarrow (s, R_s)$;
    end
  end
end
$FinalResults \leftarrow$ Combine($R_{primary}$, $\{S_{improve}\}$);
return $FinalResults$

The NLP library may have some exclusive correctness, which usually occur on some controversial spots. If we directly use sentence-level overlapping degree to combine the results, the exclusive results will be totally overwritten. In our method, we can keep parts of the exclusive correctness, which are extracted when other NLP libraries cannot generate the same wrong results. Algorithm 1 shows the details of our combined method. $O()$ represents the calculation of overlapping

degree, and $R()$ denotes the related results of the subject. The document-level selection of NLP library is implemented in line 2 to line 8, $NLP_{primary}$ denotes the primary NLP library. The sentence-level overwriting is implemented in line 9 to line 19.

4 Experiment

We propose a combined method for usage of NLP libraries to obtain more accurate NLP results than individual NLP library. The outputs generated by our method may include NLP annotations from several different NLP libraries. We discuss the effectiveness of our method in this section.

4.1 Research Questions

Our experiments investigate the following research questions.

- **RQ1:** Does the existing guideline for NLP library selection still work for the latest version of NLP libraries?
- **RQ2:** How effective is the combined method compared with individual NLP library?

To answer RQ1, we evaluate the guideline according to the comparison of NLP libraries on different documents in our study. RQ2 evaluates the effectiveness of our method compared with individual NLP library on tokenization and part-of-speech tagging. We also want to discuss the contributions of the two sources of information in our final results.

4.2 Subjects

To answer the above two research questions, we conduct experiments with four state-of-the-art NLP libraries and several documents from three different sources.

Selection of NLP Libraries. In the literature, there are a wide variety of NLP libraries used in different studies. Since our method focuses on analysis of software documents, we select several publicly available NLP libraries according to empirical results in existing work [17]. In their work, NLTK achieves the best overall results on tokenization, and spaCy performs better than other NLP libraries on part-of-speech tagging. They also make a systematic literature review of 1,350 conference papers in the area of software engineering [17]. Their results show that Stanford CoreNLP is the most frequently used NLP library. Furthermore, we add a new NLP library in our experiments, i.e., Apache OpenNLP. Apache OpenNLP is another widely used NLP library and it is proved to have a good performance on text chunking and other NLP tasks [2]. Finally, we select NLTK (version 3.4), spaCy (version 2.0.18), Stanford CoreNLP (version 3.9.2) and OpenNLP (version 1.9.1) as NLP libraries in our experiments. These NLP libraries are used as either individual NLP library or a source of outputs in

the combined method. The releasing time between our studied NLP libraries and the existing work [17] is more than 2 years, which may result in different performance.

Selection of Software Documents. We select several software documents from feature requests in Jira, ReadMe files on GitHub and Java API documentation randomly to evaluate the effectiveness of our method. For feature requests, we select 160 descriptions from 8 Apache projects[8], i.e., AXIF2, CXF, Hadoop HDFS, Hadoop Common, Hadoop MapReduce, Hbase, Maven and Struts2. These projects are widely used in research of software engineering [26,32], and we select 20 feature requests from each project. For ReadMe files, we randomly select 20 projects on Github[9] and remove Github markdown and code snippets in the selected documents. For Java API documentations, 20 APIs of Java7 are randomly selected. The total number of tokens from the selected software documents is around 40,000.

Selection of NLP Tasks. We select two NLP tasks in our experiments, i.e., tokenization and part-of-speech tagging. There are two reasons for this selection. One reason is that these NLP tasks are widely used in the literatures [3,9,11, 14,23,25]. Once researchers want to analyze software documents, tokenization and part-of-speech tagging are highly possible to be applied. The other reason is that both the two tasks can produce individual outputs, which can help us to conduct a quantitative assessment easily.

4.3 Experimental Steps

For each NLP task, we perform the following steps:

- To control the size of inputs from different sources, the selected 200 documents (i.e., 160 descriptions of feature requests, 20 ReadMe files and 20 API documents) are repartitioned into 100 new documents. There are about 400 tokens in each document. All the documents are used to analyze the overlapping degree of the studied techniques, and help to evaluate the existing guideline on NLP library selection.
- The four NLP libraries are applied to analyze the preprocessed documents for tokenization and part-of-speech tagging, respectively.
- To obtain the benchmark of correct NLP results, 20 students in our group conduct manual tokenization and part-of-speech tagging. Due to the limitation of human labor, we only collect 50 documents (i.e., about 10,000 tokens in total) to build the benchmark. We also perform a cross-validation to identify the controversial spots. The threshold value of controversy is set as 0.5 overlapping degree, i.e., more than 10 students generate different annotations on it. We obtain 82 controversial results in total, and the final results

[8] https://projects.apache.org/.
[9] https://github.com/.

are determined through our discussion. For the other parts of studied documents, the handmade benchmarks are generated by results with the highest degree of overlap.
– We apply the two-steps method on the collected 50 documents, and discuss the effectiveness of our combined method and the four NLP libraries based on the handmade benchmark generated in the above step.

4.4 Experimental Results

RQ1 Evaluation of the Existing Guideline
To evaluate the accuracy of tokenization and part-of-speech tagging for the studied NLP libraries, we compare the generated results with manual annotations. Table 3 and Table 4 show the results of comparison on tokenization and part-of-speech tagging, respectively. In the tables, column 1 lists all the studied libraries, column 2 presents the number of identical results from the outputs of NLP libraries and manual benchmark, column 3 presents the total number of results generated by NLP libraries. Column 4 presents the accuracy of each studied technique, i.e., the ratio of identical results to all the results. Columns 2 to 4 present the results on feature request, columns 5 to 7 present the results on ReadMe file, and columns 8 to 10 present the results on JavaDoc. We list the results of four individual NLP libraries at rows 3 to 6. And the last two rows in the tables present the results of our combined method which will be discussed in RQ2. From the data in Table 3 and Table 4, we have the following observations.

Table 3. Comparison on tokens (NLP libraries vs. Manual benchmark)

	Feature request			ReadMe file			JavaDoc		
	Identical	All	ACC	Identical	All	ACC	Identical	All	ACC
NLTK	2484	2545	**98%**	3154	3238	**98%**	4209	4256	**99%**
spaCy	2453	2580	96%	3113	3335	95%	4188	4348	97%
Stanford CoreNLP	2478	2529	**98%**	3170	3243	**98%**	4222	4245	**99%**
OpenNLP	2382	2535	94%	3080	3218	96%	4105	4221	97%
Combined method	2496	2543	99%	3171	3215	99%	4219	4243	99%
Overall improvement			**1%**			**1%**			**0%**

First, Stanford CoreNLP and NLTK perform best in terms of accuracy of tokenization on software documents. This finding is different with the existing guideline [17]. In that work, NLTK performs best on all kinds of software documents. Stanford CoreNLP performs worse than NLTK. Second, OpenNLP is the most effective NLP library on part-of-speech tagging in our experiments. While in the existing work [17], spaCy outperforms other NLP libraries on part-of-speech tagging except on Readme files. Based on the findings, we can infer that the guideline of NLP library selection should be updated with the change of releasing version and given documents.

Table 4. Comparison on POS tags (NLP libraries vs. Manual benchmark)

	Feature request			ReadMe file			JavaDoc		
	Identical	All	ACC	Identical	All	ACC	Identical	All	ACC
NLTK	2165	2545	85%	2807	3238	87%	3701	4256	**87%**
spaCy	2241	2580	**88%**	2851	3335	87%	3732	4348	**87%**
Stanford CoreNLP	2123	2529	84%	2717	3243	84%	3549	4245	84%
OpenNLP	2197	2535	87%	2842	3218	**89%**	3665	4221	**87%**
Combined method	2308	2528	91%	2931	3227	91%	3824	4252	90%
Overall Improvement			**3%**			**2%**			**3%**

Finding 1 *The guideline of NLP library selection should be updated with the change of releases and given documents.*

RQ2 Effectiveness of the Combined Method

Table 5 presents the data of overlapping degree. In the tables, column 1 lists all the six pairs of NLP libraries, while columns 2 to 4 and columns 6 to 8 present the percentages of identical tokens/tags to the average number of tokens/tags generated by the two NLP libraries. Column 5 and column 9 present the average percentages of identical tokens/tags for each pair of NLP libraries. The performance of studied NLP libraries is close to each other on both tokenization and part-of-speech tagging.

Comparing the results in Table 4 and Table 5, we find that the accuracy of part-of-speech tagging is higher than the overlapping degree of the paired NLP libraries in most cases. The promotion on accuracy of all the four NLP libraries indicates that every library has its own exclusive correctness on part-of-speech tagging. This finding inspires us to combine the exclusive correctness of each NLP library to achieve an improved effectiveness. The only one exception occurs on the overlapping degree of NLTK and OpenNLP on part-of-speech tags. They obtain 88% overlapped results, that is higher than the accuracy (i.e., 87%) in Table 4. We check the exception manually, and find that it is caused by the identical wrong results generated by the two libraries.

Table 5. Degree of overlap on tokens and part-of-speech tags

	Tokens				Part-of-speech tags			
	Feature request	ReadMe	JavaDoc	**Avg**	Feature Request	ReadMe	JavaDoc	**Avg**
NLTK vs. spaCy	92%	95%	97%	95%	79%	84%	85%	83%
NLTK vs. CoreNLP	95%	88%	99%	94%	75%	71%	82%	76%
NLTK vs. OpenNLP	93%	94%	98%	95%	82%	84%	88%	85%
spaCy vs. CoreNLP	89%	94%	97%	94%	74%	79%	83%	79%
spaCy vs. OpenNLP	89%	94%	96%	93%	80%	85%	86%	84%
CoreN vs. OpenNLP	90%	87%	98%	92%	77%	75%	84%	79%

Based on the degree of overlap in Table 5, we apply the combined method to generate results through document-level selection and sentence-level overwriting. The final results on tokenization and part-of-speech tagging are showed in last two rows in Table 3 and Table 4. According to the comparison of results, the combined method outperforms the studied libraries in most cases of tokenization and part-of-speech tagging. The exceptions occur in tokenization of Java API documentations by NLTK and Stanford CoreNLP. They already obtains a really high accuracy of tokenization, so the improvements from other NLP libraries cannot help too much in this case. Our combined method can achieve slight improvements on tokenization and part-of-speech tagging on the basis of state-of-the-art NLP libraries. Since the initial results of the NLP libraries already have high quality, the 2% promotion is small but meaningful. The promotion means that our combined method is better than the best selection of independent NLP library. We can use the combined method instead of any individual NLP library in the analysis of software documents.

Finding 2 *The combined method can generally outperform the individual NLP libraries in terms of effectiveness. The promotion on accuracy is around 2%.*

To evaluate the contribution of two steps in our combined method, i.e., document-level selection of NLP library and sentence-level overwriting, we analyze the proportion of each source in the final correct results. In our experiments, 1% of correct tokens and 4% of correct part-of-speech tags come from the improvements in sentence-level overwriting, and the other correct results are generated by the primary NLP library in document-level selection. Document-level selection plays a more important role than sentence-level overwriting in terms of contribution to correct results. However, the improvements of our combined method on individual NLP library mainly come from the sentence-level overwriting. The studied NLP libraries are all high-qualified toolkits with their own exclusive correctness. This is the main reason that we do not simply treat each sentence as an individual document and combined the results with highest sentence-level overlapping degree into the results of a document. The final outputs should mainly inherit the results of a primary NLP library. In this way, we can keep some exclusive correctness in the final results.

Finding 3 *The majority of correct NLP results come from the outputs of primary NLP library. The meaningful improvements of our combined method on individual NLP library come from sentence-level overwriting.*

5 Related Work

5.1 Analyzing Software Artifacts with NLP Libraries

The NLP-related researchers have proposed many meaningful works for the analysis of documents [7,10,12,15,24]. These jobs are mostly designed to obtain accurate results of tokenization and part-of-speech tagging. Some works are already considered in the widely used NLP libraries [1,8]. However, for the researchers who are not professional at NLP, the common solution for NLP tasks is using state-of-the-art NLP libraries. To solve the problem on selection of NLP libraries, we propose a combined method to utilize the strengths of different NLP libraries.

5.2 Empirical Study on NLP Libraries

There are also several works which focus on the empirical comparison of current NLP libraries. Tian and Lo [27] conduct a comparison of the effectiveness of part-of-speech tagging on bug reports. In their work, Stanford CoreNLP performs better than other toolkits. To investigate the impacts of NLP toolkits on analyzing different texts, Pinto et al. [19] compared the results of tokenization and another three NLP tasks achieved by four NLP libraries. Olney et al. [16] investigated the accuracy of 9 part-of-speech taggers on more than 200 source code identifiers Omran et al. [17] compared the results of tokenization and part-of-speech tags achieved by four NLP libraries. Compared to their work, we select three types of software documents, along with four publicly available NLP libraries in this paper. We aim to check the effectiveness of the existing guideline, rather than the comparison of current NLP libraries. According to our results, each NLP library has its own exclusive results. This finding inspires us to investigate a combined method.

6 Conclusion

This paper evaluates the existing guideline on NLP library selection with 4 publicly available NLP libraries and 200 documents. The results report that the guideline should be updated continuously and each library has its own exclusive results. Based on these findings, we turn to investigate a combined method to utilize the strengths of different libraries. The evaluation confirms that our combined method can outperform any individual NLP library in the experiments. In future, we will conduct the study on more NLP tasks with additional NLP libraries and different software documents to improve the combined method.

Acknowledgments. This work is supported in part by the National Key R&D Program of China under Grant 2018YFB1003900, in part by National Natural Science Foundation of China under Grant 61402103, Grant 61572126 and Grant 61872078, and in part by Open Research Fund of Key Laboratory of Safety-Critical Software Fund (Nanjing University of Aeronautics and Astronautics), under Grant NJ2019006.

References

1. Abebe, S.L., Tonella, P.: Natural language parsing of program element names for concept extraction. In: Proceedings of the 18th IEEE International Conference on Program Comprehension, pp. 156–159 (2010)
2. Arora, C., Sabetzadeh, M., Goknil, A., Briand, L C., Zimmer, F.: Change impact analysis for natural language requirements: an NLP approach. In: Proceedings of the 23rd IEEE International Requirements Engineering Conference, pp. 6–15 (2015)
3. Asaduzzaman, M., Roy, C.K., Monir, S., Schneider, K.A.: Exploring API method parameter recommendations. In: Proceedings of IEEE International Conference on Software Maintenance and Evolution, pp. 271–280 (2015)
4. Swathi, B.P., Anju, R.: Reformulation of natural language queries on source code base using NLP techniques. Int. J. Adv. Comput. Technol. **8**(2), 3047–3052 (2019)
5. Brill, E.: A simple rule-based part of speech tagger. In: Proceedings of the 3rd Applied Natural Language Processing Conference, pp. 152–155 (1992)
6. Cao, Y., Zou, Y., Luo, Y., Xie, B., Zhao, J.: Toward accurate link between code and software documentation. Sci. China Inf. Sci. **61**(5), 1–15 (2018). https://doi.org/10.1007/s11432-017-9402-3
7. Capobianco, G., Lucia, A.D., Oliveto, R., Panichella, A., Panichella, S.: Improving IR-based traceability recovery via noun-based indexing of software artifacts. J. Softw.: Evol. Process **25**(7), 743–762 (2013)
8. Gimpel, K., et al.: 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, pp. 42–47 (2011)
9. Gupta, R., Pal, S., Kanade, A., Shevade, S.K.: DeepFix: fixing common C language errors by deep learning. In: Proceedings of the 31st AAAI Conference on Artificial Intelligence, pp. 1345–1351 (2017)
10. Gupta, S., Malik, S., Pollock, L.L., Vijay-Shanker, K.: Part-of-speech tagging of program identifiers for improved text-based software engineering tools. In: Proceedings of the 21st IEEE International Conference on Program Comprehension, pp. 3–12 (2013)
11. Hu, X., Li, G., Xia, X., Lo, D., Jin, Z.: Deep code comment generation. In: Proceedings of the 26th Conference on Program Comprehension, pp. 200–210 (2018)
12. Jiang, W., Huang, L., Liu, Q., Lü, Y.: A cascaded linear model for joint Chinese word segmentation and part-of-speech tagging. In: Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics, pp. 897–904 (2008)
13. Khamis, N., Witte, R., Rilling, J.: Automatic quality assessment of source code comments: the JavadocMiner. In: Hopfe, C.J., Rezgui, Y., Métais, E., Preece, A., Li, H. (eds.) NLDB 2010. LNCS, vol. 6177, pp. 68–79. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-13881-2_7
14. Kim, K., et al.: FaCoY: a code-to-code search engine. In: Proceedings of the 40th International Conference on Software Engineering, pp. 946–957 (2018)
15. Lynn, T., Scannell, K.P., Maguire, E.: Minority language Twitter: part-of-speech tagging and analysis of Irish tweets. In: Proceedings of the Workshop on Noisy User-generated Text, pp. 1–8 (2015)
16. Olney, W., Hill, E., Thurber, C., Lemma, B.: Part of speech tagging java method names. In: Proceedings of IEEE International Conference on Software Maintenance and Evolution, pp. 483–487 (2016)

17. Al Omran, F.N.A., Treude, C.: Choosing an NLP library for analyzing software documentation: a systematic literature review and a series of experiments. In: Proceedings of the 14th International Conference on Mining Software Repositories, pp. 187–197 (2017)
18. Petrov, S., Das, D., Mcdonald, R.: A universal part-of-speech tagset. Comput. Sci. **1**(3), 2089–2096 (2011)
19. Pinto, A.M., Oliveira, H.G., Alves, A.O.: Comparing the performance of different NLP toolkits in formal and social media text. In: Proceedings of the 5th Symposium on Languages, Applications and Technologies, pp. 3:1–3:16 (2016)
20. Reinhartz-Berger, I., Kemelman, M.: Extracting core requirements for software product lines. Require. Eng. **25**(1), 47–65 (2019). https://doi.org/10.1007/s00766-018-0307-0
21. Reiss, S.P.: Semantics-based code search. In: Proceedings of the 31st International Conference on Software Engineering, pp. 243–253 (2009)
22. Rodriguez, C., Zamanirad, S., Nouri, R., Darabal, K., Benatallah, B., Al-Banna, M.: Security vulnerability information service with natural language query support. In: Advanced Information Systems Engineering, pp. 497–512 (2019)
23. Santos, A.L., Prendi, G., Sousa, H., Ribeiro, R.: Stepwise API usage assistance using n-gram language models. J. Syst. Softw. **131**, 461–474 (2017)
24. Shokripour, R., Anvik, J., Kasirun, Z.M., Zamani, S.: 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, pp. 2–11 (2013)
25. Thung, F., Oentaryo, R.J., Lo, D., Tian, Y.: WebAPIRec: recommending web apis to software projects via personalized ranking. IEEE Trans. Emerg. Top. Comput. Intell. **1**(3), 145–156 (2017)
26. Thung, F., Wang, S., Lo, D., Lawall, J.L.: Automatic recommendation of API methods from feature requests. In: Proceedings of the 28th IEEE/ACM International Conference on Automated Software Engineering, pp. 290–300 (2013)
27. Tian, Y., Lo, D.: A comparative study on the effectiveness of part-of-speech tagging techniques on bug reports. In: Proceedings of the 22nd IEEE International Conference on Software Analysis, Evolution, and Reengineering, pp. 570–574 (2015)
28. Tripathy, A., Rath, S.K.: Application of natural language processing in object oriented software development. In: Proceedings of International Conference on Recent Trends in Information Technology (2014)
29. van der Aa, H., Di Ciccio, C., Leopold, H., Reijers, H.A.: Extracting declarative process models from natural language. In: Giorgini, P., Weber, B. (eds.) CAiSE 2019. LNCS, vol. 11483, pp. 365–382. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-21290-2_23
30. Webster, J.J., Kit, C.: Tokenization as the initial phase in NLP. In: Proceedings of the 14th International Conference on Computational Linguistics, pp. 1106–1110 (1992)
31. Witte, R., Sateli, B., Khamis, N., Rilling, J.: Intelligent software development environments: integrating natural language processing with the eclipse platform. In: Butz, C., Lingras, P. (eds.) AI 2011. LNCS (LNAI), vol. 6657, pp. 408–419. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-21043-3_49
32. Xu, C., Sun, X., Li, B., Lu, X., Guo, H.: MULAPI: improving API method recommendation with API usage location. J. Syst. Softw. **142**, 195–205 (2018)