

Behaviour of Tokenizers in LAPPS

An assessment of three different tokenizers in LAPPS, Stanford Tokenizer, OpenNLP Tokenizer and GATE Tokenizer has been completed. Limitations for each tokenizer and the ways to correct their weaknesses are described in this report.

As an example, the following input text is used to illustrate the difference of the three tokenizers' performance. This input text is randomly taken from raw content of one pathology report.

```
1 MSH|^~\&| Cancer Reporting|BioReference Laboratories,
Inc^31D0652945^CLIA|||20170508114344||ORU^R01|201705081143440006|P|2.3.1\n\nPID|1||8262341434756^^^M
Laboratories, Inc&31D0652945&CLIA~999999999^^^SS^BioReference Laboratories,
Inc&31D0652945&CLIA||Test^Test^Test||18991230|F|||475 Market Str^^Elmwood
Park^NJ^07407|||||999999999\n\nORC|RE|||||||Bioreference^^999999999^^^CLIA|475 Market
Str^^Elmwood Park^NJ^07407|^^^^^800^29-5227|475 Market Str^^Elmwood
Park^NJ^07407\n\nOBR|1||826234143|10^GI Pathology
Report^L||18991230010101|||||999999999^PHYSICIANNNAME^UNKNOWN^UNKNOWN|^^^^^800^29-5227|||||F|
Pathology\n\nSpecimen is received in formalin, labeled as \"Rectum\" with the patient's name and consists of 4
pieces and fragments of soft pink tissue measuring from 0.3 x 0.2 x 0.1 up to 0.3 x 0.3 x 0.1 cm. All submitted (1
Block).\n\n\nFinal Diagnosis\n\nSuperficial fragments of colonic adenocarcinoma.\n\n\nNature of
Specimen\n\nColon-rectum biopsy (Sample 1)\n\n\nComments\n\nComment\n\n\nClinical
History\n\nClinicalInformation\n\n\n
```

This input text is passed into the three built-in tokenizers in LAPPS as well as the process built to compare the differences.

Difference between the three tokenizers

The tokenizer results of the three tokenizers on the above input is presented below. The differences in tokenization have been categorized into several categories. Examples of each category will be discussed using screenshots of the output in LAPPS.

- **Newline symbol '\n'**

The different tokenizers use different ways to deal with the newline symbol '\n'. All three tokenizers do not split the input string by the newline symbol '\n'. Stanford Tokenizer and GATE Tokenizer separates the tokens by backslash '\' and merge n into the next word. The OpenNLP Tokenizer keeps the whole newline symbol '\n' together but combined with the next token. An example of the results is shown in the following Figure1.

n	n	
n	n	
nFinal	nFinal	
Diagnosis	Diagnosis	
n	n	
		n\n\nFinal
nSuperficial	nSuperficial	Diagnosis\n\nSuperficial

Stanford Tokenizer GATE Tokenizer OpenNLP Tokenizer
Figure 1. Tokeniser behaviour for “\n” token for 3 tokenisers: Stanford, GATE, OpenNLP.

However, these two strategies are not the ideal ways to deal with this issue. This problem happens in most section headings and the beginning of paragraphs. Therefore, this issue has been corrected in the PostTokenizer tool added into LAPPS.

- **Other special symbol including ‘|’, ‘^’, ‘~’**

Different tokenizers deal with special symbols including ‘|’, ‘^’, ‘~’ differently. The results are shown in the following figure 2.

Market	Market	
Str	Str	
^	^	
^	^	
Elmwood	Elmwood	
Park	Park	
^	^	
NJ	NJ	
^	^	Market
07407	07407	Str^Elmwood
		Park^NJ^07407

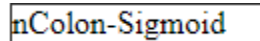
Stanford Tokenizer GATE Tokenizer OpenNLP Tokenizer
Figure 2. Tokenisation of special characters.

The Stanford Tokenizer and GATE Tokenizer split the tokens by these symbols including ‘|’, ‘^’, ‘~’ which should be the correct way of handling it. The OpenNLP Tokenizer does not recognize these symbols and combines them together with the nearby words which is the incorrect way of handling these strings.

Therefore, some manual rules needed to be used on OpenNLP tokenisation to correct this issue. This feature has been included in the PostTokenizer tool added into LAPPS for OpenNLP.

- **Hyphen ‘-’**

When there is hyphen in the main text of a pathology report, the correct way of handling it would be to separate the word before and after the hyphen into two tokens since they have different meanings in most cases. However, all these three built-in tokenizers do not split by hyphen and treat the whole combination as one token. The results from the three built-in tokenizers is shown in the screenshot below, Figure 3.



All three tokenizers

Figure 3. Tokenisation of hyphens by all three tokenisers.

This issue is corrected in the PostTokenizer tools added into LAPPS for all three tokenizers.

- **Forward slash ‘/’**

In terms of forward slash ‘/’, the GATE Tokenizer performs differently compared to Stanford Tokenizer and OpenNLP Tokenizer. The GATE Tokenizer splits the strings before and after forward slash into different tokens. Stanford and OpenNLP Tokenizers keep strings before and after as a whole token. The difference in the result is shown in the following screenshot.

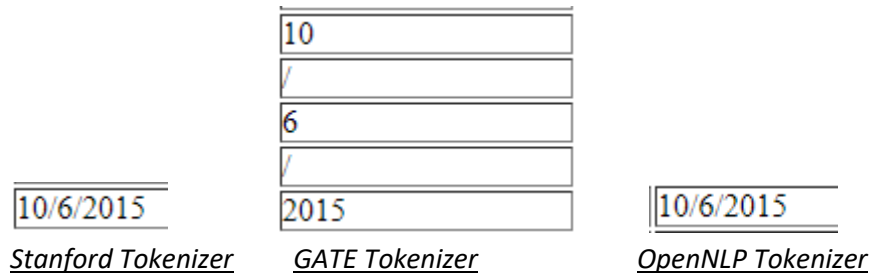


Figure 4. Tokenisation of forward slash.

In order to extract the correct information, it would be better to separate the strings before and after the forward slash into different tokens. Therefore, some manual rules have been added into the PostTokenizer tool for Stanford and OpenNLP Tokenizer.

This is a problematic case and would be better treated as an option to be set at run-time as it is easy to identify examples where in some circumstances the whole entity might well be better kept intact, such as dates.

- **Numbers together with letters**

The GATE Tokenizer performs differently in the cases where there are letters placed together with several numbers. The GATE Tokenizer tokenizes the numbers and letters separately. However, the Stanford and OpenNLP Tokenizers keep them together as a single token if there are no other delimiters amongst them. Examples are shown in Figure 5. The GATE tokeniser needs to be adjusted to prevent this subdivision of alphanumerics.

	Inc	
	^	
Inc	31	
^	D	
31D0652945	0652945	
^	^	
CLIA	CLIA	Inc^31D0652945^CLIA
<u>Stanford Tokenizer</u>	<u>GATE Tokenizer</u>	<u>OpenNLP Tokenizer</u>

Figure 5. Tokenisation of alphanumeric strings.

- **Digits**

The GATE Tokenizer performs differently in the case of digits (e.g 0.1, 0.2) compared to Stanford Tokenizer and OpenNLP Tokenizers. The GATE Tokenizer splits the number before and after the decimal point into two tokens. Stanford and OpenNLP Tokenizers keep the digit string as a single token. Examples are shown in Figure 6.

	0	
	.	
	3	
	X	
	0	
	.	
0.3	2	0.3
X	X	X
0.2	0	0.2
X	.	X
0.1	1	0.1
cm	cm	cm
<u>Stanford Tokenizer</u>	<u>GATE Tokenizer</u>	<u>OpenNLP Tokenizer</u>

Figure 6. Tokenisation of decimal numbers.

Ideally, the digit should be tokenized together as a single token. Therefore, some manual rules need to be applied after GATE Tokenization in order to keep the decimal number as a single token.

PostTokenizer Tool installed in LAPPS

In order to correct the tokenization errors in the built-in Tokenizers mentioned above, a PostTokenizer tool has been implemented to merge the relevant tokens into a single token as well as split relevant

token into several tokens which is the more common occurrence in medical texts. The workflow integrating this PostTokenizer tool is shown as follows:

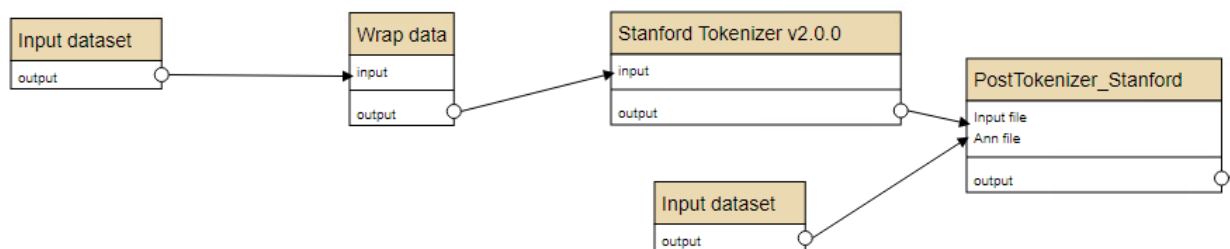


Figure 12. Process flow for the Stanford pipeline incorporating Post-tokenisation tools.

The rules and performance of the PostTokenizer tool are illustrated below in categories consistent as stated above

- **Newline symbol '\n'**

Since all three tokenizers do not recognize the newline symbol '\n', the text is split by the newline symbol in our PostTokenizer tool after using the built-in tokenizers. More specifically, the text is split into different lines explicitly. In this way, the built-in tokenizers treat texts in different lines differently which would solve the issue. The resulted tokenization is shown in the screenshot below.

-
Final
Diagnosis
Superficial
fragments
of
colonic
adenocarcinoma
-

Figure 8. All three tokenizers + PostTokenizer

After inserting the PostTokenizer tool, all three tokenizers remove the newline symbol and tokenize the section headings and start of paragraphs correctly into separate tokens. An extra token representing newline is added between heading and paragraph.

- **Other special symbol including '|', '^', '~'**

As reported above, OpenNLP Tokenizer does not recognize these special symbols '|', '^', '~'. The implementation of a post-tokenizer called PostTokenizerOpenNLP includes splitting strings by newline symbols. The strings are split before and after these special symbols into different lines so that the OpenNLP Tokenizer will treat texts on different lines as separate tokens. The result

for the OpenNLP Tokenizer with the PostTokenizerOpenNLP tool is the same as the result of built-in Stanford and GATE Tokenizer shown below.

Market
Str
^
^
Elmwood
Park
^
NJ
^
07407

Figure 9. OpenNLP Tokenizer + PostTokenizerOpenNLP

- **Hyphen ‘-’**

All three tokenizers do not split by hyphen as stated above. A rule added into the PostTokenizer tool is to split the text before and after a hyphen into different tokens. This rule is added to the PostTokenizer for all three workflows. The result of adding this tool onto all three tokenizers is shown in the screenshot below.

Colon
-
rectum

Figure 10. All three tokenizers + PostTokenizer

After adding the post-tokenizer, ‘Colon’ and ‘rectum’ are tokenized separately which should be the correct way of handling this issue. This rule is not applied to normally hyphenated words

- **Forward slash ‘/’**

The Stanford and OpenNLP Tokenizers do not split by forward slash so rules were added into the post-tokenizer tool to correct this issue. The post-tokenizer tool splits text before and after the forward slash into different tokens. The resulting tokenization for the Stanford and OpenNLP Tokenizers with the post-tokenizer added onto the process is subsequently the same as the built-in GATE Tokenizer. The result is shown in the screenshot below.

10
/
6
/
2015

Figure 11. Stanford/OpenNLP Tokenizer + PostTokenizer

The tokenization is consistent for all three tokenizers so that elements before and after forward slash are tokenized separately. A function to optionally select this method has NOT been implemented.

- **Numbers together with letters**

The post-tokenizer detects if there are consecutive tokens where a number is followed by upper case initial word followed by another number. If there exist such tokens, they are merged into a single token with token type recorded as alphanumeric. The result of this tool is shown as follows.

Inc
^
31D0652945
^
CLIA

Figure 13. Concatenation of components of an alphanumeric string.

After the post-tokenizer, such alphanumeric string is tokenized as a whole token.

- **Digits**

The post-tokenizer detects if there are three consecutive tokens with the features 'number' + '.' + 'number'. If there exist such consecutive tokens, they are recognized as a digit. Therefore, in the post-tokenizer, these three tokens are replaced with a single token whose token type is recorded to be number. The result looks like the following.

0.3
X
0.2
X
0.1
cm

Figure 14. GATE Tokenizer + PostTokenizer for decimal numbers.

This result is the same as the built-in Stanford Tokenizer and OpenNLP Tokenizer.

- **Adding the semantic tag to each token from the annotation file (.ann)**

In the PostTokenizer tool for each Tokenizer, an extraction, from the annotation file (.ann file), is the position for each semantic tag's annotation and then corresponding semantic tag is assigned as a feature to the respective each token.

After extracting the start and end position for each semantic tag, a search of that text span to find the relevant tokens for the respective semantic tag and then attaches another feature called 'semanticTag' onto the tokens. For each semantic tag, the first token is assigned a tag

with the tag 'B-' concatenated to the tag name, the following tokens with the tag 'I-' concatenated to the tag name.

The added feature for each PostTokenizer is in the similar format to Figures 15, 16 and 17. Testing has shown that this added feature, 'semanticTag', does not affect the subsequent processes including sentence splitter, POS tagger and chunker.

```
{
  "id": "tok395",
  "start": 1244,
  "end": 1249,
  "@type": "http://vocab.lappsgrid.org/Token",
  "label": "Token",
  "features": {
    "word": "Final",
    "semanticTag": "B-FINAL DIAGNOSIS HEADING "
  }
}
```

Figure 15. PostTokenizer Stanford

```
{
  "id": "473",
  "start": 1244,
  "end": 1249,
  "@type": "http://vocab.lappsgrid.org/Token",
  "label": "Token",
  "features": {
    "length": "5",
    "orth": "upperInitial",
    "word": "Final",
    "semanticTag": "B-FINAL DIAGNOSIS HEADING ",
    "tokenType": "word"
  }
}
```

Figure 16. PostTokenizer GATE

```
{
  "id": "tok_397",
  "start": 1244,
  "end": 1249,
  "@type": "http://vocab.lappsgrid.org/Token",
  "features": {
    "word": "Final",
    "semanticTag": "B-FINAL DIAGNOSIS HEADING "
  }
}
```

Figure 17. PostTokenizer OpenNLP