Online Forum Data Processing

Edward Sujono
School of Computer Science and Engineering
Singapore, 639798
ed0003no@e.ntu.edu.sg

Gede Bagus Bayu Pentium
School of Computer Science and Engineering
Singapore, 639798
pentium001@e.ntu.edu.sg

Kelvin Chandra
School of Computer Science and Engineering
Singapore, 639798
kchandra001@e.ntu.edu.sg

Kenfrand Gosti
School of Computer Science and Engineering
Singapore, 639798
kgosti001@e.ntu.edu.sg

Steve Alexander Prasetya
School of Computer Science and Engineering
Singapore, 639798
stev0023@e.ntu.edu.sg

ABSTRACT

This experiment consists of 3 main steps which are tokenization, pos-tagging, and further analysis. Firstly, tokenizer is used to divide sentences into several tokens. Next, pos-tagger will annotate each of the tokens based on its tags. To process information from online forums, we need to develop a specific tokenizer and pos-tagger to handle irregular tokens in the sentences. Additionally, irregular token needs to be defined and annotated manually for training data. Lastly, further analysis will be done to develop the real-world application by using the tokenizer and pos-tagger. The applications developed include negative sentence analyzer, semantic sentence analyzer and exception handling sentence analyzer. The objective of developing those applications is to observe behaviours of machine learning algorithms together with the pos-tagger and tokenizer in processing natural language. The machine learning algorithms used are Recursive Neural Network (RNN), Support Vector Machine (SVM), Naive Bayes. Those machine learning algorithms need a base comparison to assess the performance so that regex is used.

CCS CONCEPTS

• **Computing methodologies** → Natural language processing; *Information extraction* • **Theory of computation** → Machine learning theory

KEYWORDS

Natural language processing, online forum, stack overflow, data processing, semantic analysis, POS tagging, stemming, tokenization, Conditional Random Field, regex, Naïve Bayes, Recurrent Neural Network, Support Vector Machine.

1 INTRODUCTION

In the recent years, a lot of research has been done to find out how to analyze and process big data. One of the fields is Natural Language Processing. Natural Language Processing is concerned about programming the computer to understand certain language and to enable the interaction between human and computer. Nowadays, Natural Language Processing is used to analyze/understand the language written or spoken in daily conversation. However, in the online forums, the discussed items may not only contain human language, but also contain code snippets, special terms, etc. In Natural Language Processing, those entities must be treated as irregular tokens. To analyze sentences in online forums such as Stack Overflow, additional steps are required on top of the regular NLP. These additional steps will be the main methods to handle the irregular tokens. So, the following section discusses one of the methods to solve this problem.

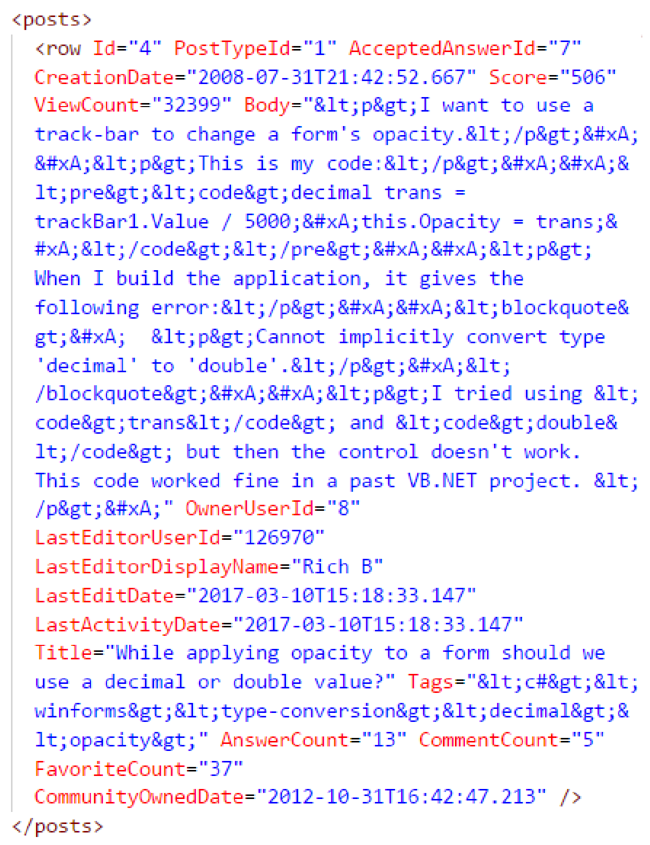
2 EXPERIMENTATION AND RESULTS

2.1 Dataset Collection

Xml parser that is used in this project is the default python library. Default python library to parse xml structure is called lxml.etree. Lxml.etree support the query by using xpath that tremendously useful to get the data based on certain condition.

The most interesting part of the data collection is the way we could access the huge chunk of raw data. The size of the raw data is almost 60 GB, which means it is impossible to put all the data into RAM. Hence, there is a need to divide the large chunk of data into smaller, more accessible parts. Luckily there is a linux command to do so which is *split --bytes=200M file_name*. In the end we successfully divide the xml file to multiple 200 MB data.

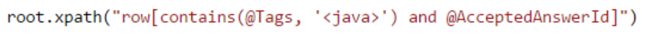
Three requirements that need to be satisfied to complete this project are the dataset should at least contain 500 threads, select thread of one programming language and the last one is the thread should at least contain 2 posts including question or answer. Getting 500 threads is an easy task since we have an abundant of data to choose, however to get thread that is discussing java we need to observe the xml structure of the data. The following xml is the instance of our dataset.



```
<posts>
  <row Id="4" PostTypeId="1" AcceptedAnswerId="7"
  CreationDate="2008-07-31T21:42:52.667" Score="506"
  ViewCount="32399" Body="&lt;p&gt;I want to use a
  track-bar to change a form's opacity.&lt;/p&gt;&#xA;
  &#xA;&lt;p&gt;This is my code:&lt;/p&gt;&#xA;&#xA;&
  lt;pre&gt;&lt;code&gt;decimal trans =
  trackBar1.Value / 5000;&#xA;this.Opacity = trans;&
  #xA;&lt;/code&gt;&lt;/pre&gt;&#xA;&#xA;&lt;p&gt;
  When I build the application, it gives the
  following error:&lt;/p&gt;&#xA;&#xA;&lt;blockquote&
  gt;&#xA;  &lt;p&gt;Cannot implicitly convert type
  'decimal' to 'double'.&lt;/p&gt;&#xA;&lt;
  /blockquote&gt;&#xA;&#xA;&lt;p&gt;I tried using &lt;
  code&gt;trans&lt;/code&gt; and &lt;code&gt;double&
  lt;/code&gt; but then the control doesn't work.
  This code worked fine in a past VB.NET project. &lt;
  /p&gt;&#xA;" OwnerUserId="8"
  LastEditorUserId="126970"
  LastEditorDisplayName="Rich B"
  LastEditDate="2017-03-10T15:18:33.147"
  LastActivityDate="2017-03-10T15:18:33.147"
  Title="While applying opacity to a form should we
  use a decimal or double value?" Tags="&lt;c#&gt;&lt;
  winforms&gt;&lt;type-conversion&gt;&lt;decimal&gt;&
  lt;opacity&gt;" AnswerCount="13" CommentCount="5"
  FavoriteCount="37"
  CommunityOwnedDate="2012-10-31T16:42:47.213" />
</posts>
```

Figure 1: XML details of "post" tag.

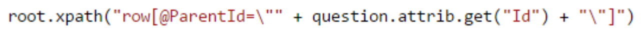
We could observe from above xml structure that there is a 'tags' attribute that explain the type of the thread, so by using tag attribute of 'java' we could easily get the java thread. Fulfilling the last requirement is an easy task as well, because the thread that has an answer is guaranteed to have an attribute of 'AcceptedAnswerId'. So, to get the list of question satisfying above 3 conditions can be realized by using this simple xpath:



```
root.xpath("row[contains(@Tags, '<java>') and @AcceptedAnswerId]")
```

Figure 2: Xpath command used to select JAVA related thread that at least have one answer.

Finally, to get the list of answer of the retrieved question we could easily search the answer with parent ID attribute of the question ID. Following is the xpath to get the list of answers from a question ID:



```
root.xpath("row[@ParentId=\"" + question.attrib.get("Id") + "\"]")
```

Figure 3: Xpath command to retrieve answer(s) bases on the question ID.

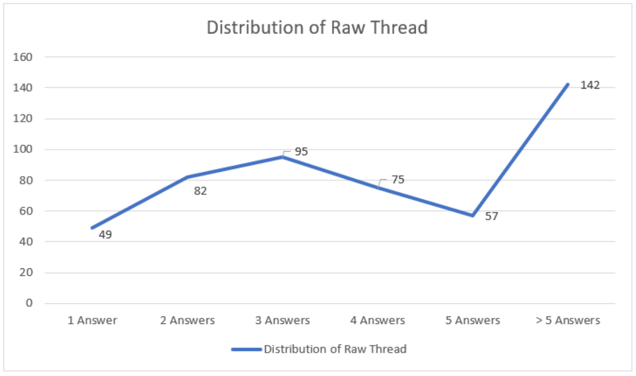


Figure 4: Distribution of Raw Thread

2.2 Dataset Analysis and Annotation

2.2.1 Stemming.

Below are the results of top-20 most used word (excluding English stop words) before (Table 1) and after (Table 2) stemming process.

Table 1: Top-20 Word Count Before Stemming

No	Word	Count
1	Java	1289
2	Use	1171
3	Code	994
4	Like	824
5	Using	806
6	Class	697
7	Way	583
8	Just	559
9	Method	546
10	Need	513
11	Want	463
12	Application	406
13	Used	403
14	Object	396
15	File	369
16	Example	364
17	Work	354
18	Data	348
19	Think	328
20	Time	318

Table 2: Top-20 Word Count After Stemming

No	Word	Original Word	Count
1	use	Using, use, Use, using, uses, used, useful, Useful, usefulness, Used	2713
2	java	Java, java, JAVA	1490
3	code	coding, code, Code codes, Coding, coded	1045

4	class	Class, class, classes, Classes	1032
5	like	Like, like, liked, Likely, likely, liking, likes	904
6	method	Method, method, methods, Methods	807
7	object	Objects, objects, object, Object, objective	724
8	need	Need, need, needed, needing, needs	688
9	work	Work, work, working, works, worked, Works, Working, Worked, workings	658
10	way	Way, way, ways, WAY, Ways	640
11	just	Just, just	622
12	file	File, file, files, Files, fileds, filed	608
13	look	Look, look, looking, looks, Looking, LOOK, looked, Looks	527
14	applic	Application, applicable, application, applications, Applications	526
15	want	want, wants, wanted, wanting	516
16	exampl	Example, example, Examples, examples	442
17	creat	Creating, creating, create, created, Create, creates, Created, Creates	439
18	gener	Generate, general, generic, generics, generator, generally, generated, generation, generating, generically, Generally, generates, generators, Generics, generalizing, generational, Generational, Generator, generalized, generations, General, Generation, Generating, generalize, Generic, generalization, generous, generate	430
19	run	run, running, runs, Run, Running	425
20	tri	Try, try, trying, tried, tries, Trying	405

From the tables above, we can see that the top-20 word are mainly keywords that are often used in Java, such as class, method, and object. This is fitting since Java is our language of choice for the dataset collection. Looking at the two tables above, it is quite clear that there are keywords in Java mixed in with regular English language. This is normal because after all, we are processing the natural language aspect of the posts inside the dataset. If we compare the two tables of top words before and after stemming, we can see that most of the top words remain even after the stemming process. What we noticed, however, is that the top words after stemming prioritizes more important keywords that are related to our language of choice. While stemming does increase recall, it harms precision.

2.2.2 *POS Tagging.*

Here are the results for the POS Tagging:

1. Sentence: “I have an opensource framework for working with CSV and flat files in general”
   Result: [('I', 'PRP'), ('have', 'VBP'), ('an', 'DT'), ('opensource', 'NN'), ('framework', 'NN'), ('for', 'IN'), ('working', 'VBG'), ('with', 'IN'), ('CSV', 'NNP'), ('and', 'CC'), ('flat', 'JJ'), ('files', 'NNS'), ('in', 'IN'), ('general', 'JJ')]
2. Sentence: “I'm fairly surprised at how unanimous the sentiment is that getters and setters are fine and good”
   Result: [('I', 'PRP'), ("'m", 'VBP'), ('fairly', 'RB'), ('surprised', 'VBN'), ('at', 'IN'), ('how', 'WRB'), ('unanimous', 'JJ'), ('the', 'DT'), ('sentiment', 'NN'), ('is', 'VBZ'), ('that', 'IN'), ('getters', 'NNS'), ('and', 'CC'), ('setters', 'NNS'), ('are', 'VBP'), ('fine', 'JJ'), ('and', 'CC'), ('good', 'JJ')]
3. Sentence: “Sun itself recommends staying away from StringTokenizer and using the String.spilt method instead”
   Result: [('Sun', 'NNP'), ('itself', 'PRP'), ('recommends', 'VBZ'), ('staying', 'VBG'), ('away', 'RB'), ('from', 'IN'), ('StringTokenizer', 'NNP'), ('and', 'CC'), ('using', 'VBG'), ('the', 'DT'), ('String.spilt', 'NNP'), ('method', 'NN'), ('instead', 'RB')]
4. Sentence: “At work we use teraterm and realterm for checking serial data is correctly formatted.”
   Result: [('At', 'IN'), ('work', 'NN'), ('we', 'PRP'), ('use', 'VBP'), ('teraterm', 'JJ'), ('and', 'CC'), ('realterm', 'NN'), ('for', 'IN'), ('checking', 'VBG'), ('serial', 'JJ'), ('data', 'NNS'), ('is', 'VBZ'), ('correctly', 'RB'), ('formatted', 'VBN')]
5. Sentence: “You should throw an IllegalArgumentException, as it will make it obvious to the programmer that he has done something invalid.”
   Result: [('You', 'PRP'), ('should', 'MD'), ('throw', 'VB'), ('an', 'DT'), ('IllegalArgumentException', 'NN'), (',', ','), ('as', 'IN'), ('it', 'PRP'), ('will', 'MD'), ('make', 'VB'), ('it', 'PRP'), ('obvious', 'JJ'), ('to', 'TO'), ('the', 'DT'), ('programmer', 'NN'), ('that', 'IN'), ('he', 'PRP'), ('has', 'VBZ'), ('done', 'VBN'), ('something', 'NN'), ('invalid', 'NN'), ('.', '.')]
6. Sentence: “In C#, int is just an alias for System.Int32, string for System.String, double for System.Double etc.”
   Result: [('In', 'IN'), ('C', 'NNP'), ('#', '#'), (',', ','), ('int', 'NN'), ('is', 'VBZ'), ('just', 'RB'), ('an', 'DT'), ('alias', 'NN'), ('for', 'IN'), ('System.Int32', 'NNP'), (',', ','), ('string', 'VBG'), ('for', 'IN'), ('System.String', 'NNP'), (',', ','), ('double', 'NN'), ('for', 'IN'), ('System.Double', 'JJ'), ('etc', 'NN'), ('.', '.')]
7. Sentence: “One should always use the Integer.equals() method when comparing Integer objects.”
   Result: [(‘One’, ’CD’), (‘should’, ’MD’), (‘always’, ’RB’), (‘use’, ‘VB’), (‘the’, ‘DT’), (‘Integer.equals’, ‘NNP’), (‘(‘, ‘(‘), (‘)’, ‘)’), (‘method’, ‘NN’), (‘when’, ‘WRB’), (‘comparing’, ‘VBG’), (‘Integer’, ‘NNP’), (‘objects’, ‘NNS’)]
8. Sentence: “A NullPointerException seems to make more sense in the case where you're attempting to actually use the null.”
   Result: [('A', 'DT'), ('NullPointerException', 'NN'), ('seems', 'VBZ'), ('to', 'TO'), ('make', 'VB'), ('more', 'JJR'),

('sense', 'NN'), ('in', 'IN'), ('the', 'DT'), ('case', 'NN'), ('where', 'WRB'), ('you', 'PRP'), ("'re", 'VBP'), ('attempting', 'VBG'), ('to', 'TO'), ('actually', 'RB'), ('use', 'VB'), ('the', 'DT'), ('null', 'NN'), ('.', '.')]

9. Sentence: "We are using FindBugs and Checkstyle as well as Clover for Code Coverage."
   Result: [('We', 'PRP'), ('are', 'VBP'), ('using', 'VBG'), ('FindBugs', 'NNP'), ('and', 'CC'), ('Checkstyle', 'NNP'), ('as', 'RB'), ('well', 'RB'), ('as', 'IN'), ('Clover', 'NNP'), ('for', 'IN'), ('Code', 'NNP'), ('Coverage', 'NNP'), ('.', '.')]
10. Sentence: "Type safety: The cast from Object to List is actually checking against the erased type List"
   Result: [('Type', 'JJ'), ('safety', 'NN'), (':', ':'), ('The', 'DT'), ('cast', 'NN'), ('from', 'IN'), ('Object', 'NNP'), ('to', 'TO'), ('List', 'NNP'), ('is', 'VBZ'), ('actually', 'RB'), ('checking', 'VBG'), ('against', 'IN'), ('the', 'DT'), ('erased', 'JJ'), ('type', 'NN'), ('List', 'NN')]

As we can see from the results of the POS tagging above, the default tokenizer can appropriately apply POS tagging to sentences that contain mostly natural language. An example would be sentence 1 whereby each word is appropriately tagged based on their contexts in the sentence. However, when the sentence contains bits and pieces of code, the tokenizer begins to show its flaws. For instance, in sentence 6, the word "string" was incorrectly tagged as a verb, whereby in reality it should have been a variable type. Another glaring example of the weakness of the default POS tagger can be seen in sentence 7. The string "Integer.equals()" should have been regarded as one token because it is a Java method. However, in this case the POS tagger treats it as 3 separate tokens, separating both parentheses.

2.3 Token Definition and Annotation

We define our token as group(s) of characters from the original dataset that can be treated as a syntax in Java. For instance, the following strings are considered as one JAVA token:

String.split()
NullPointerException
@Override
int[]
List<String>

The rationale for the above is simple – some of the above strings are previously treated as more than one token. We don't want this to happen, as it can interfere in the data processing for our application. Hence to get rid of this problem, we annotated 100 posts manually to achieve the result that we want.

In the annotation process, at first, we used the default POS Tagger to tag the 100 posts we selected. After that, we went on to annotate manually which strings belong to one token with regards to our final application.

2.4 Tokenizer

2.4.1 Regex.

We use Regex as one of our tokenizer to tokenize all the Java related words to JAVA and other unknown words to UNK (unknown). For JAVA token, we tried to capture as many cases as possible. Following are the list of all Regex we created for both JAVA token and UNK token.

Java Token

1. [a-zA-Z0-9_\.]+\([a-zA-Z0-9_]*\)
   This Regex captures Java function or method call.
   For ex.: test._this_method()
   The first part: [a-zA-Z0-9_\.]+ captures all words including dot and underscore.
   The second part: \(captures the open parenthesis
   The third part : [a-zA-Z0-9_]* captures optional arguments for the method.
   The fourth part: \) captures the close parenthesis of the method.
2. [a-zA-Z0-9_]*@[a-zA-Z0-9_]+
   This Regex captures Java annotation.
   For ex.: @Override
   The first part: [a-zA-Z0-9_]* captures optional words that comes before the annotation symbol.
   The second part: @ captures the annotation symbol.
   The third part: [a-zA-Z0-9_]+ captures the words that follow the annotation symbol.
3. (List<[a-zA-Z0-9_]+>|ArrayList<[a-zA-Z0-9_]+>)
   This Regex captures Java List or Java ArrayList.
   For ex.: List<Integer> or ArrayList <String>
   The first part: List<[a-zA-Z0-9_]+> captures List with its parameter.
   The second part: ArrayList<[a-zA-Z0-9_]+> captures ArrayList with its parameter.
4. [a-zA-Z0-9_]*Exception\(?[a-zA-Z0-9_]*\)?
   This Regex captures Java Exceptions.
   For ex.: NullPointerException(test)
   The first part: [a-zA-Z0-9_]* captures all words that are prefix of Java Exception
   The second part: Exception captures the words Exception, means the original word must contain Exception in order to become Java Exception.
   The third part: \(?[a-zA-Z0-9_]*\)? captures optional arguments that enclosed by parentheses.
5. [a-zA-Z_]{2,}\.[\.a-zA-Z_]*[a-zA-Z_]+
   This Regex captures Java words with it packages.
   For ex.: java.lang.NullPointerException
   The first part: [a-zA-Z_]{2,}\. captures a certain words more than 2 characters followed by a dot.
   The second part: [\.a-zA-Z_]*[a-zA-Z_]+ captures the repetition of alphabetical words follow by dot and ended with words with length longer than one character.
6. [a-zA-Z]+[a-z]+[A-Z][A-Za-z0-9_]+
   This Regex captures CamelCase words.
   For ex.: testParser
   The first part: [a-zA-Z]+[a-z]+[A-Z] captures the first words which can start from one or more Capital or lowercase letter and follow by one or more lowercase letter and follow by one Capital letter. Ex.: TesT, tesT, TEsT.
   The second part: [A-Za-z0-9_]+ captures all the characters that followed the first CamelCase.

7. `[a-zA-Z0-9\_\.]+\[.*\][a-zA-Z0-9\.\_]*`
   This Regex captures Java Array.
   For ex.: bytes[ ].
   The first part: `[a-zA-Z0-9\_\.]+` captures the Array type or Array name
   The second part: `\[.*\][a-zA-Z0-9\.\_]*` captures the Square brackets with its optional arguments and the optional function that follow the square brackets.

Unknown Token

1. `[A-Za-z0-9\_\$\-\>]*\$[A-Za-z0-9\_\$\-\>]+|[A-Za-z0-9\_\$\-\>]+\$[A-Za-z0-9\_\$\-\>]*`
   This Regex captures PHP codes.
   For ex.: testing->Result, $Callback
2. `\https?\:\/\/[a-zA-Z0-9\S]*\.[\com|\org\|net][\/a-zA-Z0-9\S]*`
   This Regex captures URL links.
   For ex.: https://www.google.com.sg/

2.4.2 *CRF.*

Conditional Random Field was used as another tokenizer on this experiment. The CRF used using k-fold as cross validation to select the best model from training data. To use CRF as tokenizer IOB tagging is done based on the manual annotation.

IOB Tagging

The IOB tag is created using the training data that is manually annotated. So, for each character in a sentence tagged with "S", "T", "I", "O". "S" indicating the start of a sentence (it marked based on the POS tagging result of end sentences). So, using the POS tagging result the start of the next token after end sentence marked as "S". "T" indicating the start of a token (that is not the start of sentence) so all the start of token that is not "S" will be marked as "T". Next step is to label all character inside the token with "I" and the remaining as "O".

CRF Features

From the IOB tagging some features are extracted from the character for CRF. Window size is used to determine how many contexts used as feature in the CRF training. In this experiment 4 is used as window size. For each 4 characters to the left and 4 characters to the right of current character, the extracted feature used are:

- The character itself
- The character is an upper case
- The character is a number

So, in total it will be 27 features used.

Result

From the experiment the regular token is successfully determined using the CRF tokenizer. But the JAVA token is not tokenized correctly (e.g. "String()" became "String(" and ")") This may be caused because the small number of test data. But some JAVA token is already correctly tokenized. So, given more training data the CRF tokenizer might give better result.

The metrics of the tokenizer are:

- Accuracy: 0.635
- F1: 0.624
- Precision: 0.625
- Recall: 0.624

2.5 Further Analysis

2.5.1 *Irregular Tokens.*

Here are the results for the top-20 non-English words.

Table 3: Top-20 Non-English Words

No	Word	Count
1	Java	1285
2	java	196
3	ArrayList	31
4	HashMap	28
5	Exception	27
6	TestNG	27
7	NullPointerException	22
8	JavaScript	19
9	equals()	18
10	JFreeChart	18
11	Exceptions	18
12	IllegalArgumentException	17
13	toString()	16
14	StringBuilder	16
15	serialVersionUID	15
16	WebSphere	14
17	hashCode	14
18	java.util.concurrent	14
19	int[]	13
20	finalize()	12

As we can see from the above table, we have successfully managed to capture tokens that would have been previously treated as separate tokens. An example of this is entry 9, "equals()". Previously, it would have been recognized as 3 separate tokens – the word and the parentheses. The same case goes for entry 19 as well.

2.5.2 *Normal POS Tagging with Regex Tokenizer.*

Here are the results of the POS tagging with regex tokenizer.

1. Sentence: "Also, PMD supports Java's @SuppressWarnings annotation."
   Result: ["Also", "RB"], [",", ","], ["PMD", "NNP"], ["supports", "VBZ"], ["Java", "NNP"], ["'s", "POS"], ["@SuppressWarnings", "NNS"], [".", "."]
2. Sentence: "On the other hand, NullPointerExceptions and IndexOutofBoundsExceptions might actually often be appropriate."
   Result: ["On", "IN"], ["the", "DT"], ["other", "JJ"], ["hand", "NN"], [",", ","], ["NullPointerExceptions", "NNP"], ["IndexOutofBoundsExceptions", "NNP"], ["actually", "RB"], ["often", "RB"], ["be", "VB"], ["appropriate", "JJ"], [".", "."]
3. Sentence: "Whenever a.equals(b), then a.hashCode() must be same as b.hashCode()."

Result: ["Whenever", "WRB"], ["a.equals(b)", "VBP"], ["then", "RB"], ["a.hashCode()", "RB"], [“must”, “MD”], ["be", "VB"], ["same", "JJ"], ["as", "IN"], ["b.hashCode()", "NN"]

4. Sentence: “It is the output of the built-in Date.toString() method (at least in the UK and US locales).”
Result: ["It", "PRP"], ["is", "VBZ"], ["the", "DT"], ["output", "NN"], ["of", "IN"], ["the", "DT"], ["built-in", "JJ"], ["Date.toString()", "NNP"], ["method", "NN"], ["(", "("], ["at", "IN"], ["least", "JJS"], ["in", "IN"], ["the", "DT"], ["UK", "NNP"], ["and", "CC"], ["US", "NNP"], ["locales", "NNS"], [")", ")"], [".", "."]
5. Sentence: “IllegalArgumentException: Thrown to indicate that a method has been passed an illegal or inappropriate argument.”
Result: ["IllegalArgumentException", "NNP"], [":", ":"], ["Thrown", "VBN"], ["to", "TO"], ["indicate", "VB"], ["that", "IN"], ["a", "DT"], ["method", "NN"], ["has", "VBZ"], ["been", "VBN"], ["passed", "VBN"], ["an", "DT"], ["illegal", "JJ"], ["or", "CC"], ["inappropriate", "JJ"], ["argument", "NN"], [".", "."]
6. Sentence: “Instead, use Runtime.exec(String[]) which takes an array of already-separated arguments.”
Result: ["Instead", "RB"], [",", ","], ["use", "NN"], ["Runtime.exec", "NNP"], ["(", "("], ["String[]", "NNP"], [")", ")"], ["which", "WDT"], ["takes", "VBZ"], ["an", "DT"], ["array", "NN"], ["of", "IN"], ["already-separated", "JJ"], ["arguments", "NNS"], [".", "."]
7. Sentence: “Have a look at the APIs available in the java.lang.management package.”
Result: ["Have", "VBP"], ["a", "DT"], ["look", "NN"], ["at", "IN"], ["the", "DT"], ["APIs", "NNP"], ["available", "JJ"], ["in", "IN"], ["the", "DT"], ["java.lang.management", "NN"], ["package", "NN"], [".", "."]
8. Sentence: “I have a list of integers, List<Integer> and I'd like to convert all the integer objects into Strings, thus finishing up with a new List<String>.”
Result: ["I", "PRP"], ["have", "VBP"], ["a", "DT"], ["list", "NN"], ["of", "IN"], ["integers", "NNS"], [",", ","], ["List<Integer>", "NNP"], ["and", "CC"], ["I", "PRP"], ["'d", "MD"], ["like", "VB"], ["to", "TO"], ["convert", "VB"], ["all", "PDT"], ["the", "DT"], ["integer", "NN"], ["objects", "VBZ"], ["into", "IN"], ["Strings", "NNP"], [",", ","], ["thus", "RB"], ["finishing", "VBG"], ["up", "RP"], ["with", "IN"], ["a", "DT"], ["new", "JJ"], ["List<String>", "NNP"], [".", "."]
9. Sentence: “One should always use the Integer.equals() method when comparing Integer objects.”
Result: ["One", "CD"], ["should", "MD"], ["always", "RB"], ["use", "VB"], ["the", "DT"], ["Integer.equals()", "NNP"], ["method", "NN"], ["when", "WRB"], ["comparing", "VBG"], ["Integer", "NNP"], ["objects", "NNS"], [".", "."]
10. Sentence: “I've also seen cases where an int[] of size 1 is passed in.”
Result: ["I", "PRP"], ["'ve", "VBP"], ["also", "RB"], ["seen", "VBN"], ["cases", "NNS"], ["where", "WRB"], ["an", "DT"], ["int[]", "NN"], ["of", "IN"], ["size", "NN"], ["1", "CD"], ["is", "VBZ"], ["passed", "VBN"], ["in", "IN"], [".", "."]

From the results above, we can see that the regex tokenizer has improved the POS tagger by quite a significant margin. For example, if we compare sentence 9 above and sentence 7 in section 2.2.2, we can see that the POS tagger now correctly considers the string “Integer.equals()” as one token. In this case “Integer.equals()” is correctly tagged as NNP (proper noun).

2.5.3 *CRF POS Tagging with Regex Tokenizer.*

Data Preprocessing

From the data collected from the forum, 100 posts are selected and tagged with the NLTK library. Then, the tagged comments are re-labelled to have the appropriate tag based on the rule defined.

For each word in the posts, the relevant features are extracted and will be used later to train the CRF model.

Current list of features:

1. The word itself.
2. Last 3 letters of the word.
3. Last 2 letters of the word.
4. The word is in uppercase.
5. The word is a title.
6. The word is a number.
7. Start of a sentence.
8. The next word after the current word.
9. The next word is in uppercase.
10. The next word is a title
11. End of a sentence.
12. The previous word before the current word.
13. The previous word is in uppercase.
14. The previous word is a title.

K-Folds

The data from the previous step is then separated into K-folds. The number of K-folds tested for this project is 3,4,5 and 6. The number chosen for the model is 5 since the 5-fold model gives the highest accuracy and the improvement from 5-fold to 6-fold is insignificant.

Evaluation of Results

1. Sentence : ["Its includeMethod() can exclude any method we want , like a public not-annotated method ."]
POS tag : [('Its','NNS'), ('includeMethod()','JAVA'), ('can','MD'), ('exclude','VB'), ('any','DT'), ('method','NN'), ('we','PRP)', ('want','VBP'), (',' , ','), ('like','IN'), ('a','DT'), ('public','JJ'), ('not-annotated','JJ'), ('method','NN'), ('.' , '.')]
2. Sentence : ["There’s nothing special about BigDecimal.round() vs. any other BigDecimal method."]
POS tag : [('There','EX'), (''s','VBZ'), ('nothing','VBG'), ('special','JJ'), ('about','IN'), ('BigDecimal.round()','JAVA'), ('vs', 'NN'), ('.', '.'), ('any','DT'), ('other','JJ'), ('BigDecimal','JJ'), ('method','NN'), ('.' , '.')]

3. Sentence : ["From the EDT, you can read the current event from the queue (java.awt.EventQueue.getCurrentEvent()) , and possibly find a component from that ."]

POS tag : [('From','IN'), ('the','DT'), ('EDT','NNP'), (',' , ','), ('you','PRP'), ('can','MD'), ('read','VB'), ('the','DT'), ('current','JJ'), ('event','NN'), ('from','IN'), ('the','DT'), ('queue','NN'), ('(' , '('), ('java.awt.EventQueue.getCurrentEvent()','JAVA'), (')' , ')'), (',' , ','), ('and','CC'), ('possibly','RB'), ('find','VB'), ('a','DT'), (('component','NN'), ('from','IN'), ('that','IN'), ('.' , '.')]

4. Sentence : ["So with autoboxing, I would expect the above to convert myInt to an Integer and then call toString() on that ."]

POS tag : ['RB', 'IN', 'VBG', ',', 'PRP', 'MD', 'VB', 'DT', 'NN', 'TO', 'VB', 'NN', 'TO', 'DT', 'NNP', 'CC', 'RB', 'VB', 'JAVA', 'IN', 'DT', '.']

5. Sentence : ["In C#, integers are neither reference types nor do they have to be boxed in order for ToString() to be called ."]

POS tag : [('So','RB'), ('with','IN'), ('autoboxing','VBG'), (',' , ','), ('I','PRP'), ('would','MD'), ('expect','VB'), ('the','DT'), ('above','NN'), ('to','TO'), ('convert','VB'), ('myInt','NN'), ('to','TO'), ('an','DT'), ('Integer','NNP'), ('and','CC'), ('then','RB'), ('call','VB'), ('toString()','JAVA'), ('on','IN'), ('that','DT'), ('.' , '.)']

6. Sentence : ["POST variables should be accessible via the request object : HttpRequest.getParameterMap() ."]

POS tag : [('POST','NNP'), ('variables','NNS'), ('should','MD'), ('be','VB'), ('accessible','JJ'), ('via','IN'), ('the','DT'), ('request','JJ'), ('object','NN'), (':', ':'), ('HttpRequest.getParameterMap()','JAVA'), ('.' , '.')]

7. Sentence : ["It would be impossible to call the toString() or equals() methods , for example , since they are inherited in most cases ."]

POS tag : [('It','PRP'), ('would','MD'), ('be','VB'), ('impossible','JJ'), ('to','TO'), ('call','VB'), ('the','DT'), ('toString()','JAVA'), ('and','CC'), ('equals()','JAVA'), ('methods','NNS'), (',' , ','), ('for','IN'), ('example','NN'), (',' , ','), ('since','IN'), ('they','PRP'), ('are','VBP'), ('inherited','VBN'), ('in','IN'), ('most','JJ'), ('cases','NNS'), ('.' , '.')]

8. Sentence : ["Java may allow a try / catch around the super() call in the constructor if 1 . you override ALL methods from the superclasses , and 2 . you don't use the super.XXX() clause , but that all sounds too complicated to me ."]

POS tag : [('Java','NNP'), ('may','MD'), ('allow','VB'), ('a','DT'), ('try','JJ'), ('/','NN'), ('catch','MD'), ('around','VB'), ('the','DT'), ('super()','JAVA'), ('call','NN'), ('in','IN'), ('the','DT'), ('constructor','NN'), ('if','IN'), ('1','CD'), ('.' , '.'), ('you','PRP'), ('override','VBP'), ('ALL','DT'), ('methods','NNS'), ('from','IN'), ('the','DT'), ('superclas','NNS'), (',' , ','), ('and','CC'), ('2','CD'), ('.' , '.'), ('you','PRP'), ('do','VBP'), ('n't','RB'), ('use','VB'), ('the','DT'), ('super.XXX()','JAVA'), ('clause','NN'), (',' , ','), ('but','CC'), ('that','IN'), ('all','DT'), ('sounds','NNS'), ('too','RB'), ('complicated','VBN'), ('to','TO'), ('me','PRP'), ('.' , '.')]

9. Sentence : ["The NullPointerException is an exception thrown by the Java Virtual Machine when you try to execute code on null reference (Like toString()) ."]

POS tag : [('The','DT'), ('NullPointerException','JAVA'), ('is','VBZ'), ('an','DT'), ('exception','NN'), ('thrown','VBN'), ('by','IN'), ('the','DT'), ('Java','NNP'), ('Virtual','NNP'), ('Machine','NNP'), ('when','WRB'), ('you','PRP'), ('try','VBP'), ('to','TO'), ('execute','VB'), ('code','NN'), ('on','IN'), ('null','JJ'), ('references','NN'), ('(' , '('), ('Like','IN'), ('toString()','JAVA'), (')' , ')'), ('.' , '.')]

10. Sentence : ["Short version - use flush() or whatever the relevant system call is to ensure that your data is actually written to the file ."]

POS tag : [('Short','NNP'), ('version','NN'), (' ','NN'), ('-',':'), ('use','NN'), ('flush()','JAVA'), ('or','CC'), ('whatever','WDT'), ('the','DT'), ('relevant','JJ'), ('system','NN'), ('call','NN'), ('is','VBZ'), ('to','TO'), ('ensure','VB'), ('that','IN'), ('your','PRP$'), ('data','NNS'), ('is','VBZ'), ('actually','RB'), ('written','VBN'), ('to','TO'), ('the','DT'), ('file','NN'), ('.' , '.')]

Using the CRF POS tagger the result became more accurate because now the POS tag for JAVA token has its own tag. This will give better information when used in the application.

2.6 Application

As for the application, we would like to explore all the possible implementations for natural language processing. Therefore, we built 3 models, namely: model to recognize negative sentence, model to recognize semantic meaning and model to recognize sentences related to error handling.

The concern of developing any machine learning algorithm lies in the input. The input of every machine learning algorithm is in the form of matrices containing numerical value, therefore we need to transform or embed the sentence to its matrices of numerical value. Matrices in the program can be easily represented in two-dimensional array. If the column of two-dimensional array represents unique token and the row represents the sentence, then the value of column 'a' and row 'b' is the number of the token 'a' contained in the sentence 'b'. Following is an example of the words-to-matrices transformation.

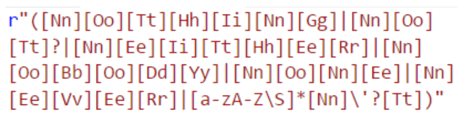
The list of sentences "I like Java" and "I test Java" contains 4 token which are I, like, test, and java. Tokens 'I', 'like', 'test' and 'java' will be identified by index 0,1,2,3 respectively. Therefore, eventually the end matrices of the sentences are the following.

Table 4: Word-to-Matrix Transformation

Token/Sentence	I	Like	Test	Java
1	1	1	0	1
2	1	0	1	1

2.6.1 *Negative Sentence Model.*

Initially negative sentence exploration is done by regex pattern. The rationale of choosing regex is because negative sentence has an obvious token such as not, doesn't etc. Those negative tokens are independent too which means any sentence that has ‘NOT’ token will be considered as the negative sentence such that regex can be used. The following is the regex pattern used to capture the negative sentence.



```
r"([Nn][Oo][Tt][Hh][Ii][Nn][Gg]|[Nn][Oo]
[Tt]?|[Nn][Ee][Ii][Tt][Hh][Ee][Rr]|[Nn]
[Oo][Bb][Oo][Dd][Yy]|[Nn][Oo][Nn][Ee]|[Nn]
[Ee][Vv][Ee][Rr]|[a-zA-Z\S]*[Nn]\'?[Tt])"
```

Figure 4: Regex for Negative Sentence.

Aside from regex, several machine learning algorithms are used with hope that the algorithm can capture the negative sentence pattern. The machine learning algorithms used are naive bayes and svm. The table below will show the result of the algorithm trained.

Furthermore, we will try to tune the model by using K-Fold, N-Gram and TF-IDF. K-Fold is used to search the best model among the K model. N-gram means we will train the model by using bigram. If it performs worse, then we will choose the unigram instead. TF-IDF means that we will transform the inputted vector to a matrix of TF-IDF features, then train the model by using the TF-IDF matrix. TF-IDF essentially will determine how important the token is in the corpus. TF means term frequency it will count the number of token in the sentence, however token ‘the’ might be the top most frequent word since we do not put any weightage on each of the tokens, therefore IDF term is introduced. IDF is inverse document frequency which will reduce the weightage of the most common words such as ‘the’. Both N-Gram and TF-IDF are not important in this context since there is no need to do context preservation.

Table 5: Algorithm Performance Statistics on Negative Sentences

Algorithm	Accuracy	F1	Precision	Recall
Naive Bayes	0.770	0.450	0.506	0.880
Tuned Naive Bayes	0.840	0.753	0.729	0.793
SVM	0.930	0.890	0.870	0.920
Tuned SVM	0.920	0.870	0.830	0.930

Above is the naive test result of the application.

2.6.2 Semantic Meaning Model and Error Sentence Model. Semantic and error recognizer is a challenging problem. There are a lot of reasons behind the complexity of this problem. Firstly, both problems require the model to capture the context behind the sentence which is almost impossible to do. Secondly, the quality of data used for training is not that good, since everyone has different perceptions regarding the sentence which means even humans cannot do it perfectly.

Both models can be built by using Recurrent Neural Network(RNN) since RNN can preserve the context of the sentence. Aside of RNN we will try the tuned SVM and Bayesian to do this task as well, since it will do few technique of context preservation such as Bigram and TF-IDF transformation.

Table 6: Algorithm Performance Statistics on Error-related Sentences

Algorithm	Accuracy	F1	Precision	Recall
RNN	0.95	0.48	0.50	0.47
Tuned Naive Bayes	0.95	0.60	0.56	0.97
Tuned SVM	0.95	0.48	0.50	0.47

Table 7: Algorithm Performance Statistics on Semantic Meaning Analysis

Algorithm	Accuracy	F1	Precision	Recall
RNN	0.860	0.300	0.330	0.286
Tuned Naive Bayes	0.860	0.360	0.360	0.530
Tuned SVM	0.860	0.360	0.360	0.480

All the above results make a lot of sense since we lack data to be inserted to the model. Almost 90% of our data is either neutral or not an error such that all the test result show either neutral in the error problem context or neutral in the semantic context.

4 CONCLUSIONS

This experiment indicates that processing natural language is not an easy task since it is context-based. Stackoverflow has its own context such that irregular token dominates the overall token collection. Further processing of an irregular token is needed so that it can be used for a Stackoverflow-related application.

5 APPENDICES

5.1 3rd Party Library

Keras==2.0.8

nltk==3.2.5

numpy==1.13.3

scikit-learn==0.19.1

scipy==1.0.0

sklearn==0.0

tensorflow==1.3.0

tensorflow-tensorboard==0.1.8

pandas==0.21.0

python-crfsuite==0.9.5

sklearn-crfsuite==0.3.6

ACKNOWLEDGMENTS

This work was supported by School of Computer Science and Engineering of Nanyang Technological University and by Dr. Sun Aixin.