CZ4045 Natural Language Processing Assignment - Online Forum Data Processing

Cui Bolun
Nanyang Technological University
Singapore
CUIB0001@e.ntu.edu.sg

Cui Shengping
Nanyang Technological University
Singapore
SCUI001@e.ntu.edu.sg

Sun Zhihao
Nanyang Technological University
Singapore
ZSUN005@e.ntu.edu.sg

Zhao Jingyi
Nanyang Technological University
Singapore
JZHAO009@e.ntu.edu.sg

1 INTRODUCTION

This project is an affiliated coursework for CZ4045 Natural Language Processing in Nanyang Technological University (Singapore). The group has gone through the full data processing lifecycle using data collected from online forum Stack Overflow, an online forum dedicated to programming related questions and answers.
The main components of this project are: data collection by the self-designed crawler, data analysis and annotation by using stemming followed by POS tagging, tokenizer training and development, further analysis and its applications for sentiment statistics, Shannon generation, grammar parsing checking, and collocation analysis.
Notably, the tokenizer designed and developed by the team is capable to identify irregular name entities such as "European Computer Manufacturer's Association" or "src/main/resources", which greatly improved the accuracy of tokenization, thus enhanced the performance of further analysis.

2 DATASET COLLECTION

2.1 Crawler Design

In order to get threads that satisfy conditions, we designed a web crawler. The crawler contains two parts: A single thread crawler to get all links to the most popular threads attached with tag "java", and a multi-threaded crawler to get the posts text of the threads.
The endpoint for the first crawler is "/questions/tagged/java?sort=frequent".
This link provides all popular questions ranked by a number of answers. Through this endpoint, we can easily find questions that satisfy the problem conditions.
Requests and BeautifulSoup library are used to parse fetched HTML contents.
The crawler will filter out the question with less than two answers, so every thread fetched will have more than two posts. All these questions have tag "java", so they must be related to Java programming language.
After question links are obtained, the multi-threaded crawler will access each link and parse all posts within the thread. Multi-threading manner is used to parallel requests and save network wait time. Since Stack Overflow has rate limit on web request, we design a cool-down time for each sent request.

2.2 Text processing

- Remove code
  We remove all code pieces.
  All text embedded by <code> tag are removed.
- Remove HTML tags BeautifulSoup removes any HTML tags and only keep inside text of the tags.
- Remove special characters UTF-8 encoding is used.
  Any special characters beyond UTF-8 are removed.

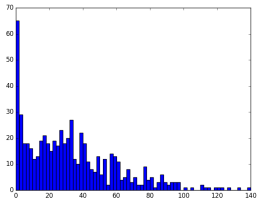

Figure 1: Posts Count Distribution

In total, we fetched 610 popular threads with "java" tag bound. Above is the distribution graph of threads' post count:
The graph below shows the distribution of posts' length:
The mean value of threads' post count is 32.4,
and standard deviation is 27.
This proves that threads collected are most popular ones.
Questions with one post takes up 2.78% questions.
Questions with two post takes up 2.13% questions.
Questions with three post takes up 2.62% questions.

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

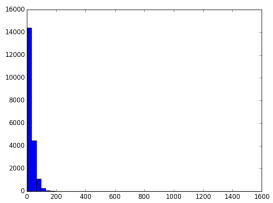

Figure 2: Post Length Distribution

From the statistics, we can observe that most questions have more than three posts and are therefore threads that satisfy conditions.

As we observe the text collected, we find that all code pieces are removed, except some Java API call found in answers. These Java codes will be handled by tokenizer.

3 DATASET ANALYSIS AND ANNOTATION

3.1 Stemming

- Top-20 most frequent origin words:

java	use	class	method	code
object	using	string	one	like
example	also	need	would	way
file	want	get	used	type

- Top-20 most stemmed words:

Stem	Original
use	usings, useful, use, usefulness, used, using, uses, usefully
class	classing, classes, class, classe
method	method, methodically, methods, methode
java	java, javas
object	objects, objective, object, objections, objectively
code	code, codes, coded, coding
string	strings, string
one	ones, one
like	likes, likely, likeness, liking, like, liked
need	needing, neede, needs, need, needed
valu	value, valued, values
exampl	examples, example, exampl
file	files, file, filed
call	call, calling, calls, called
work	worked, work, working, works
get	get, getting, gets
type	typed, typing, types, type
way	way, ways
also	also
creat	created, creat, create, creating, creates

- Analysis:
  The top-20 most frequent words show some features of the corpus:
  – Words related to specific programming language:
    These words represent the programming language itself. The word "java" is exactly the language itself. Some words exhibit the important concept. For example: "class", "method", "object", "type", "get", "call" might probably show that this language relies heavily on object-orientation. Some other words show the frequent APIs of the language: "object", "string", "file" are probably the most APIs used in the language.
  – Words related to the style of Stack Overflow answers:
    In Stack Overflow, most answers are imperative instructions and they have justification on the reliability or correctness of themselves. Hence, other frequently used words show such tone: "use", "one", "like", "need", "work", "way", "also"

3.2 POS Tagging

(1) *Suppose There are two person* `P1`, `P2` *(threads) a* `Washbasin` *(shareable entity) inside a washroom and there is door (lock).*
Suppose/**VB** there/**EX** are/**VBP** two/**CD** persons/**NNS** `P1`/**NNP** `P2`/**NNP** ,/**,** (/**(** threads/**NNS**)/**)** ,/**,** a/**DT** `Washbasin`/**NNP** (/**(** shareable/**JJ** entity/**NN** inside/**IN** a/**DT** washroom/**NN** and/**CC** there/**EX** is/**VBZ** a/**DT** door/**NN** (/**(** lock/**NN**)/**)** ./**.**

(2) *On JDK 7+, you can use try-with-resources construct.*
On/**IN** JDK/**NNP** 7/**NNP** +/**NNP** ,/**,** you/**PRP** can/**MD** use/**VB** try-with-resources/**JJ** construct/**NN** ./**.**

(3) *Here I will show how to remove html tags from string.*
Here/**RB** I/**PRP** will/**MD** show/**VB** how/**WRB** to/**TO** remove/**VB** html/**JJ** tags/**NNS** from/**IN** string/**NN** ./**.**

(4) *There are solutions for Windows 7, Windows Vista, Windows XP, Linux/Solaris and other shells.*
There/**EX** are/**VBP** solutions/**NNS** for/**IN** Windows/**NNP** 7/**NNP** ,/, Windows/**NNP** Vista/**NNP** ,/**,** Windows/**NNP** XP/**NNP** ,/**,** Linux/Solaris/**NNP** and/**CC** other/**JJ** shells/**NNS** ./**.**

(5) *As the link above describes, both of these will register a* `RequestMappingHandlerMapping` *bean (and a bunch of other stuff).*
As/**IN** the/**DT** link/**NN** above/**RB** describes/**VBZ** ,/**,** both/**DT** of/**IN** these/**DT** will/**MD** register/**VB** a/**DT** `RequestMappingHandlerMapping`/**NNP** bean/**NN** (/**(** and/**CC** a/**DT** bunch/**NN** of/**IN** other/**JJ** stuff/**NN**)/**)** ./**.**

(6) *However, a* `HandlerMapping` *isn't very useful without a handler.*
However/**RB** ,/**,** a/**DT** `HandlerMapping`/**NNP** is/**VBZ** n't/**RB** very/**RB** useful/**JJ** without/**IN** a/**DT** handler/**NN** ./**.**

(7) `RequestMappingHandlerMapping` *expects some* `@Controller` *beans so you need to declare those too, through* `@Bean` *methods in a Java configuration or* `<bean>` *declarations in an XML configuration or through component scanning of* `@Controller` *annotated classes in either.*
`RequestMappingHandlerMapping`/**NNP** expects/**VBZ** some/**DT** `@`/**SYM** `Controller`/**NNP** beans/**NNS** so/**CC** you/**PRP** need/**VBP** to/**TO** declare/**VB** those/**DT** too/**RB** ,/**,** through/**IN** `@`/**SYM** `Bean`/**NNP** methods/**NNS** in/**IN** a/**DT** Java/**NNP** configuration/**NN** or/**CC** `<`/**SYM** `bean`/**NN** `>`/**SYM** declarations/**NNS** in/**IN** an/**DT** XML/**NNP** configuration/**NN** or/**CC** through/**IN**

component/**NN** scanning/**VBG** of/**IN** @/**SYM** `Controller`/**NNP** annotated/**JJ** classes/**NNS** in/**IN** either/**DT** ./**.**

(8) *Make sure these beans are present.*
Make/**VB** sure/**JJ** these/**DT** beans/**NNS** are/**VBP** present/**JJ** ./**.**

(9) *As / and * operators are equal in precedence, we need to look at the associativity between those operators.*
As/**CC** //**SYM** and/**CC** */**SYM** operators/**NNS** are/**VBP** equal/**JJ** in/**IN** precedence/**NN,** we/**PRP** need/**VBP** to/**TO** look/**VB** at/**IN** the/**DT** associativity/**NN** between/**IN** those/**DT** operators/**NNS.**

(10) *I did Anagram excersize, which is like Count Change problem but with 50 000 denominations (coins).*
I/**PRP** did/**VBD** Anagram/**NNP** exercise/**NN,** which/**WDT** is/**VBZ** like/**in** Count/**NNP** Change/**NNP** problem/**NN** but/**CC** with/**IN** 50000/**CD** denominations/**NNS** (/**(** coins/**NNS**)/**)** ./**.**

Analysis: There are quite a number of proper-nouns in these sentences, probably because the posts are discussing some APIs used in Java. Some of the posts have syntax errors which makes POS tagging harder, since we need to firstly correct the grammar then apply POS tagging. Many sentences have used parentheses, and this might be related to the nature of Stack Overflow. The writers of posts are responsible for their answers and they want to make it clear to the readers. Therefore, they use parentheses extensively to include additional clarification.

3.3 Token Definition and Annotation

Our token definition consists of three pieces of rules:

- Space separated words
  Empty space is the first level of word tokenizing
- Special use of Dot
  As we observe the text pattern in context of
  Stack Overflow, many Java API call contains
  dots. For example:
  and com.sun.xml.internal.bind.DatatypeConverterImpl.java
  Therefore, sentence segmentation is not
  simply separate by dot.
  Our idea is use regular expression to find
  dots surranded by non-space chars.
  The regex to find API call:
  +[]*[]+ with greedy strategy
  Examples like U.S.A or javax.xml.bind
  can be matched by this regular expression.
  With this regular expression, we find
  out all irregular dots and exclude
  them for sentence segmentation.

- Handle parenthesis
  In context of Stack Overflow, some API call have
  parenthesis following code. For example:
  doGet(), processRequest()
  We decide to leave these parentheses as part of token
  As we observe, all parenthesis in API call have
  no space between its preceding word.
  We utilize this finding to filter out () not part of API
  To achieve this, we firstly find locations of all
  "(". Then from the obtained positions, we
  find the following unenclosed right parenthesis.
  After that, we treat all the left parent with space
  and right paren as a tokenizer.
  In this way, we reserve all () with no space before it.
  The result of this method turns out to be accurate.

- Handle abbreviation
  Common abbreviation should be tokenized,
  such as n't, 's, 're etc.
  Separate the abbreviation out can help further
  negation analysis and syntactic parsing.
  Since the abbreviations are fixed, we use a
  string set to match these fixed patterns.

- Named Entity combination
  After separating sentences, we find some
  named entity existing in posts.
  For example: James Gosling
  European Computer Manufacturer's Association
  Java Runtime Environment

  To help syntactic parsing and semantic analysis,
  we hope to treat these named entities as one token.
  To achieve this, we utilize Conditional Random to
  train a model to identify these named entities
  automatically. We manually annotate some posts
  as seed data to train the model. This is
  the hardest part to implement tokenizer.

To train the CRF model of named entity extraction,
we will manually annotate on 100 posts as input data.
The target is to find all Named Entities
built from more than two space-separated words.
For example:
" The choice of JavaScript as a name was a nod, if you will, to the then-ascendant Java programming language, developed at Sun by Patrick/MISC Naughton/MISC, James/MISC Gosling/MISC, et. al. "
In the example sentence, we label the continuous
words of "James Gosling" with "/MISC" label.
This helps pycrfsuite to recognize the words
as a named entity.

4 DEVELOPMENT OF A TOKENIZER

4.1 Trainer

The training model is based on python-crfsuite Trainer.
As the training data is 100 labelled posts, it is a very small training set. Though generative models like HMM tends to perform better in a smaller dataset, their assumptions are hard to meet and it is impossible to enumerate all possible observation sequences to obtain proper joint probability, especially for NLP tasks. In this case, a discriminative model becomes more suitable for the task as it has relaxation of the independent assumption and it is better in capturing dependencies without modelling the distribution. However,

discriminative modes suffer from the label-bias problem where per-state normalization effectively ignores useful observations, as well as from misleading spurious patterns. Unfortunately, our dataset, which is small and biased by nature, is susceptible to the above problems.
With these considerations, conditional random field (CRF), a variation of Markov random field introduced by Lafferty et al. in 2001[1], is applied to counter the above problems. By defining a single log-linear distribution over the joint probability of the entire label sequence given an observation sequence, it avoids the need of per-state normalization, allowing individual states passing on their probability mass to their successor states[2]. In addition, as an undirected probability graphical model, it can capture more dependencies and predict labels with consideration of neighboring sample labels, which is also suitable for our task of identifying the start and end of an entity token. To conclude, using CRF helps to avoid some common pitfalls of discriminative models while allows correlated features to be used in this tokenizer.

4.2 Feature Extraction

4.2.1 Word Pattern Feature. Many special tokens have superficial characteristics: they are easy to obtain but very useful for classification, and word pattern is one of them.
In our project, word pattern features are extracted using regular expressions. For letters, we have:
(r"[A-ZÄÖÜ]", "A"),
(r"[a-zäöüß]", "a"),
and then
(r"[A]2,", "A+"),
(r"[a]2,", "a+"),
After the above regular expression matching, sample input 'Edsger', 'Dijkstra' both become 'Aa+'.
To be more specific about the usefulness of word pattern, people names are usually in word pattern Aa+ Aa+, and the same goes for institution names and other proper nouns. As for java method call, it is usually in the form Aa+.a+Aa+.a+(). Consequently, it is chosen as one of our main features for CRF training.
Other Word Configurations
With the same idea of word pattern, an array of features is extracted concerning digits, capital letters, token length and punctuations.
For each token, we assign the following features represented by integer values indicating:

- whether the token starts with a capital letter
- whether the token contains digits, punctuations
- whether the token contains only digits or only punctuations
- the length of the token

4.2.2 POS Tag. The Part-of-Speech tag is also an input feature of CRF training. It is generated by running the POS tagger from nltk on primary tokens.

4.2.3 Prefix and Suffix. The prefix (first three char if token length >3) and the suffix (last three char) of the token are also taken out as a feature. The motivation is to find frequent prefixes or suffixes that can signal the start/end of an entity special token. For example, 'Mr.' usually starts a people entity, and 'ltd.' usually signals the end of an organizational entity.

4.2.4 Unigram Ranking. To obtain this feature, the unigrams are extracted from the original text and held in an ordered dictionary. All unigrams grouped and ranked by their frequencies. The frequency of a unigram is used to leverage other simple features, reducing the side effects of the unbalanced input corpus.

4.3 Training

The training consists of several steps: 1) tokenize the input sentences to obtain primary tokens using nltk tokenizer, 2) segment input post into training windows (sentences to facilitate POS tagging), 3) compute feature list for each of the window, 4) train the model and 5) save model parameters.
The training dataset is 100 posts labelled as entity tokens and normal tokens. For entity tokens, we maintain a balance in dataset between the different class of entities including the person, organization, API calls, topics and miscellaneous. Since normal tokens outnumbered entity tokens, a simple boosting is applied by training the model using only windows that contain entity tokens between several iterations.

4.4 Result

	precision	recall	f1-score	support
Entity	0.91	0.71	0.80	28
O	0.84	0.96	0.90	45
avg / total	0.87	0.86	0.86	73

4.5 Analysis of Errors

By analyzing resulting tokens, it is observed that one major source of error is the inconsistency in the extracted feature. Because the training data is crawled from Stack Overflow website, word format is very inconsistent. The capitalization of proper nouns is not strictly followed, so as other word patterns. For instance, 'HTML page and 'HTML page' both appear multiple times in the dataset. This confuses the model occasionally as word pattern is an important feature through training. Unfortunately, as the dataset is very small, de-capitalize all the input and remove word pattern feature would lead to worse performance.

5 FURTHER ANALYSIS

5.1 Irregular Tokens

After tokenizing the dataset using the tokenizer developed, the top-20 most frequent irregular tokens (NOT standard English words) identified are:

#	Token	Morphological Forms	Frequency
1	Brendan Eich	NNP	18
2	Patrick Naughton	NNP	18
3	James Gosling	NNP	18
4	HTTP POST requests	NNS	5
5	.jar file	RB	4
6	HTML page	NN	4
7	Sun Microsystems	NNP	3
8	HTML pages	NNP	3
9	HTTP GET requests	NNS	3
10	HTML form	NNP	3
11	ClassCastException	NN	3
12	languages/models/platforms	NNS	2
13	JavaScript interpreter	NNP	2
14	src/main/resources	NNS	2
15	Thread.currentThread	NNP	2
16	.getContextClassLoader	NNP	2
17	File.separator	NNP	2
18	authorization/authentication	NN	2
19	HTTP GET request	NNP	2
20	POST form	NNP	2

5.2 POS Tagging

The randomly selected 10 sentences from the dataset where each sentence contains at least one irregular token are:

- But, JavaScript was created at Netscape (now Mozilla) in the early days of the Web, and technically, "Java-Script" is a trademark licensed from Sun Microsystems used to describe
- JavaScript is an object-oriented scripting language that allows you to create dynamic HTML pages, allowing you to process input data and maintain data, usually within the browser.
- While the two have similar names, they are really two completely different programming languages/models/platforms, and are used to solve completely different sets of problems.
- They are independent languages with unrelated lineages. Brendan Eich created JavaScript originally at Netscape. It was initially called Mocha. The choice of JavaScript as a name was a nod, if you will, to the then-ascendant Java programming language, developed at Sun by Patrick Naughton, James Gosling , et. al.
- Netscape's implementation of the language. Netscape submitted the language for standardization to ECMA (European Computer Manufacturer's Association)
- Practically every PC in the world sells with at least one JavaScript interpreter installed on it.
- If you are using spring, then you can use the the following method to read file from src/main/resources:
- Place the file at the root (after extracting .jar file, it should be in the root), then access it using Thread. currentThread(). getContextClassLoader(). getResourceAsStream("file.txt")
- JavaScript is an object-oriented scripting language that allows you to create dynamic HTML pages, allowing you to process input data and maintain data, usually within the browser.
- You should use doGet() when you want to intercept on HTTP GET requests. You should use doPost() when you want to intercept on HTTP POST requests. That's all. Do not port the one to the other or vice versa (such as in Netbeans ' unfortunate auto-generated processRequest() method). This makes no utter sense.

Based on our own tokenization results, POS tagging is applied on these sentences as shown below:

- But/**CC**, JavaScript/**NNP** was/**VBD** created/**VBN** at/**IN** Netscape/**NNP** (now/**RB** Mozilla/**NNP**) in/**IN** the/**DT** early/**JJ** days/**NNS** of/**IN** the/**DT** Web/**NNP**, and/**CC** technically/**RB**, "Java-Script/**NNP**" is/**VBZ** a/**DT** trademark/**NN** licensed/**VBN** from/**IN** Sun Microsystems/**NNP** used/**VBD** to/**TO** describe/**VB**
- JavaScript/**NNP** is/**VBZ** an/**DT** object-oriented/**JJ** scripting/**NN** language/**NN** that/**WDT** allows/**VBZ** you/**PRP** to/**TO** create/**VB** dynamic/**JJ** HTML pages/**NNP**, allowing/**VBG** you/**PRP** to/**TO** process/**VB** input/**NN** data/**NNS** and/**CC** maintain/**NN** data/**NNS**, usually/**RB** within/**IN** the/**DT** browser/**NN**.
- While/**IN** the/**DT** two/**CD** have/**VBP** similar/**JJ** names/**NNS**, they/**PRP** are/**VBP** really/**RB** two/**CD** completely/**RB** different/**JJ** programming/**NN** languages/models/platforms/**NNS**, and/**CC** are/**VBP** used/**VBN** to/**TO** solve/**VB** completely/**RB** different/**JJ** sets/**NNS** of/**IN** problems/**NNS**.
- They/**PRP** are/**VBP** independent/**JJ** languages/**NNS** with/**IN** unrelated/**JJ** lineages/**NNS**. Brendan Eich/**NNP** created/**VBD** Javascript/**NNP** originally/**RB** at/**IN** Netscape/**NNP**. It/**PRP** was/**VBD** initially/**RB** called/**VBN** Mocha/**NNP**. The/**DT** choice/**NN** of/**IN** Javascript/**NNP** as/**IN** a/**DT** name/**NN** was/**VBD** a/**DT** nod/**NN**, if/**IN** you/**PRP** will/**MD**, to/**TO** the/**DT** then/**RB** ascendant/**JJ** Java/**NNP** programming/**NN** language/**NN**, developed/**VBN** at/**IN** Sun/**NNP** by/**IN** Patrick Naughton/**NNP**, James Gosling/**NNP**, et/**NN**. al/**NN**.
- Netscape/**NNP**'s/**JJ** implementation/**NN** of/**IN** the/**DT** language/**NN**. Netscape/**NNP** submitted/**VBD** the/**DT** language/**NN** for/**IN** standardization/**NN** to/**TO** ECMA/**NNP** (European Computer Manufacturer's Association/**NNP**)
- Practically/**NNP** every/**DT** PC/**NN** in/**IN** the/**DT** world/**NN** sells/**VBZ** with/**IN** at/**IN** least/**JJS** one/**CD** JavaScript interpreter/**NNP** installed/**VBD** on/**IN** it/**PRP**.
- If/**IN** you/**PRP** are/**VBP** using/**VBG** spring/**NN**, then/**RB** you/**PRP** can/**MD** use/**VB** the/**DT** the/**DT** following/**JJ** method/**NN** to/**TO** read/**VB** file/**NN** from/**IN** src/main/resources/**NNS**:
- Place/**VB** the/**DT** file/**NN** at/**IN** the/**DT** root/**NN** (after/**IN** extracting/**VBG** .jar file/**RB**, it/**PRP** should/**MD** be/**VB** in/**IN** the/**DT** root/**NN**), then/**RB** access/**NN** it/**PRP** using/**VBG** Thread. currentThread(). getContextClassLoader(). getResourceAsStream("file.txt")/**NNP**
- JavaScript/**NNP** is/**VBZ** an/**DT** object-oriented/**JJ** scripting/**NN** language/**NN** that/**WDT** allows/**VBZ** you/**PRP** to/**TO** create/**VB** dynamic/**JJ** HTML pages/**NNP**, allowing/**VBG** you/**PRP** to/**TO** process/**VB** input/**NN** data/**NNS** and/**CC** maintain/**NN** data/**NNS**, usually/**RB** within/**IN** the/**DT** browser/**NN**.
- You/**PRP** should/**MD** use/**VB** doGet()/**NN** when/**WRB** you/**PRP** want/**VBP** to/**TO** intercept/**VB** on/**IN** HTTP GET requests/**NNS**.

You/**PRP** should/**MD** use/**VB** doPost()/**NN** when/**WRB** you/**PRP** want/**VBP** to/**TO** intercept/**VB** on/**IN** HTTP POST requests/**NNS**. That/**DT**'s/**VBZ** all/**DT**. Do/**NNP** not/**RB** port/**VB** the/**DT** one/**CD** to/**TO** the/**DT** other/**JJ** or/**CC** vice/**NN** versa/**NNS** (such/**JJ** as/**IN** in/**IN** Netbeans/**NNP**' unfortunate/**JJ** auto-generated/**JJ** processRequest()/**NN** method/**NN**). This/**DT** makes/**VBZ** no/**DT** utter/**JJ** sense/**NN**.

6 APPLICATION

6.1 Sentiment Statistics

We did a simple sentiment analysis on collected corpus.
We check word sentiment from AFINN word list, and compute sentiment score for each sentence.
We then do statistics on all sentence score, and get a distribution graph.

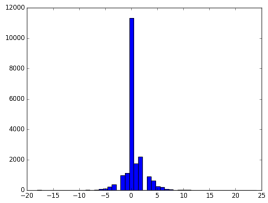

Figure 3: Sentiment Distribution

From the result, we can conclude that most posts are very neutral and have no subjectivity. Positive posts are more than negative posts. We can feel that, as a technical forum, Stack Overflow is very neutral and encouraging.
We also count 10 most frequent positive words

- good
- best
- super
- nice
- great
- perfect
- perfectly
- greater
- popular
- excellent

10 most frequent negative words

- bad
- warning
- evil
- illegal
- worry
- worse
- lost
- worst
- warnings
- ugly

We can feel that most discussions are polite and objective

6.2 Shannon Generation

We try to generate a 20-word sentence based on NGram model
Starting words are all "I like".
Result of 3-Gram model:
"I like the old " eclipse " folder , because they are) . No one mentioned anyMatch yet. This is the"
Result of 4-Gram model:
"I like it. What are the " Best Practices " ? Will I actually get Java code , but the google web"
Result of 5-Gram model:
"I like to use this annotation every time I am overriding a method independently , if the base is an interface"
Result of 6-Gram model:
"I like the idea to use an almost correct method on some of the input. Here is a version with a higher"

As we observe from the generated results, a higher Gram model can generate a more meaningful sentence.

6.3 Grammar Parsing and Checking

We select 1129 sentences from the original data. Each sentence contains at most 10 tokens and it must start with a capital word and end with symbols ".", "?", or "!".
Then we run Stanford's CoreNLP parser to get the parse tree of each sentence, as well as determine whether each sentence is grammatically correct.
The result is amazing, out of the 1129 sentences none contains grammatical errors. Originally we planned to find and correct errors in the sentences and then compare the parse trees of the incorrect and the correct sentences. It turns out that although Stack Overflow is an online platform, people still use proper language. This result shows that most of the post authors are responsible to maintain the high quality of the platform and try to make answer or explanation as clear as possible.
Some of the parse trees of the sentences are shown below (See Figures 4-8):

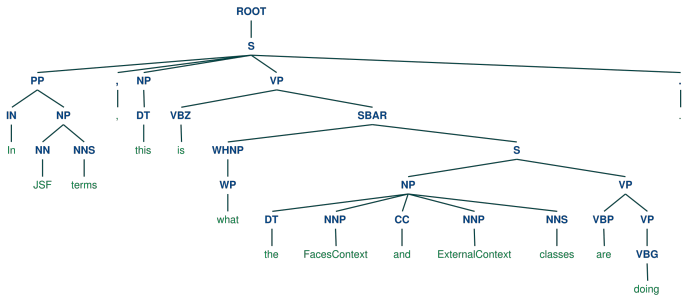

Figure 4: Parse tree of sentence: In JSF terms, this is what the `FacesContext` and `ExternalContext` classes are doing.

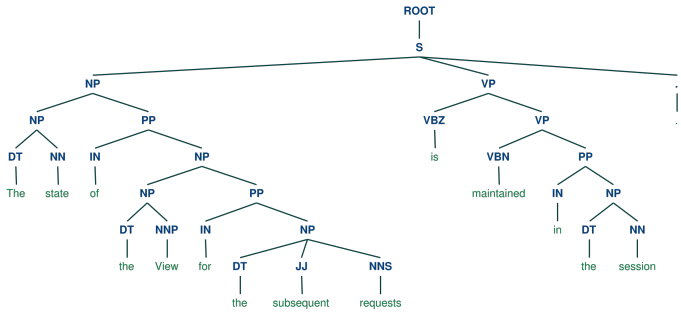

Figure 5: Parse tree of sentence: The state of the View for the subsequent requests is maintained in the session.

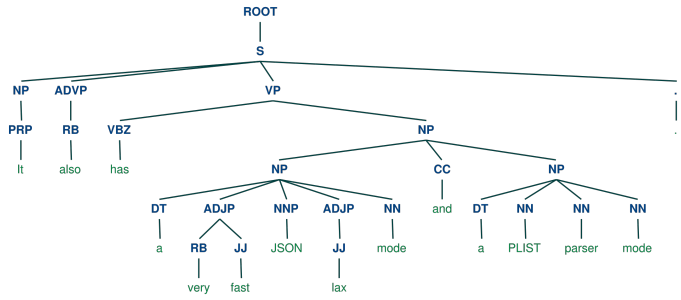

Figure 6: Parse tree of sentence: It also has a very fast JSON lax mode and a PLIST parser mode.

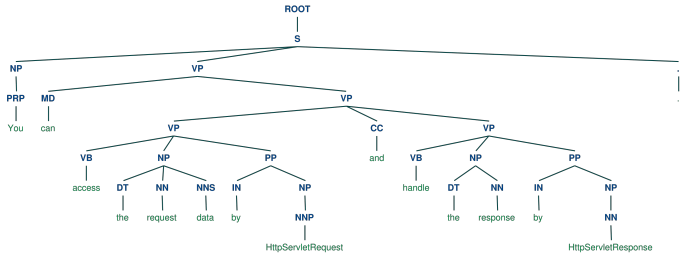

Figure 7: Parse tree of sentence: You can access the request data by `HttpServletRequest` and handle the response by `HttpServletResponse`.

6.4 Collocation Analysis

Another useful application for stack overflow content would be an analysis of topics/entity relations appeared in the crawled threads, which can provide insight into knowledge correlation. This is realized by collocation analysis. The full Stack Overflow content is first tokenized, disregarding all punctuation and stop words, then filtered regarding their POS tag. Bi-grams and tri-grams are extracted from the filtered tokens to identify co-occurrence of topics. The top 20 highly collocated topics are visualized using a chord graph.

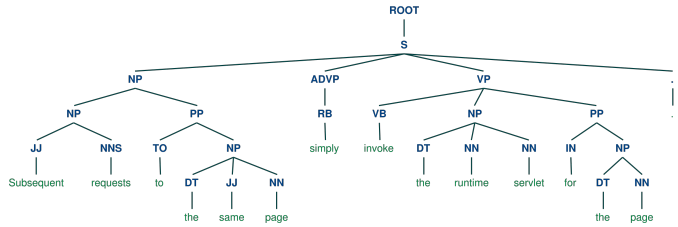

Figure 8: Parse tree of sentence: Subsequent requests to the same page simply invoke the runtime servlet for the page.

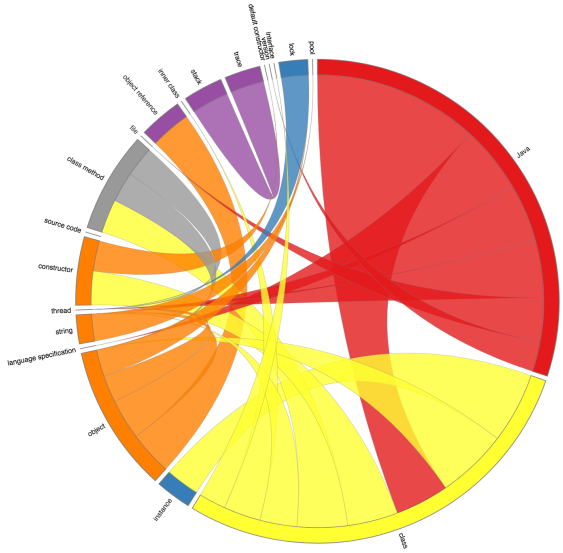

Figure 9: Top 20 collocated topics in Java

7 APPENDIX

7.1 Third-party Libraries

All the third-party libraries used in this project and their respective download links can be found below:

(1) **requests**: https://pypi.python.org/pypi/requests
(2) **BeautifulSoup**: https://pypi.python.org/pypi/BeautifulSoup
(3) **python-crfsuite**: https://pypi.python.org/pypi/python-crfsuite
(4) **scikit-learn**: https://pypi.python.org/pypi/scikit-learn
(5) **shelve**: https://pypi.python.org/pypi/shelve
(6) **nltk**: https://pypi.python.org/pypi/nltk
(7) **CoreNLP**: https://stanfordnlp.github.io/CoreNLP/index.htmldownload

REFERENCES

[1] John Lafferty, Andrew McCallum, and Fernando Pereira. Conditional random *Conditional random fields: Probabilistic models for segmenting and labeling sequence data.* In Proc. ICML 2001, 2001.
[2] Hanna Wallach *Efficient Training of Conditional Random Fields.* 2002