

# Speaker Recognition in the Transcript Using Keywords

Sumin Han

(hsm6911@kaist.ac.kr)

GitHub: <https://github.com/SuminHan/NLP-SpongeBob>

## Abstract

Recently, many major IT companies are competing with each other to make a better voice recognition device [1]. However, the devices that are currently on the market do not provide personalized service for the multiple users due to the difficulty of the realization of the speaker recognizer. In fact, the speaker recognition can be implemented with the help of accurate sound sensors and machine-learning techniques, but it requires extra hardware and software which slows down the whole system with some security issues. So, I wanted to research about the simple way to detect the speaker by looking at the text itself. Since different people may speak different words, they may use specific words more frequently which make them distinguishable. I set a hypothesis that each speaker has their own keywords in his sentence. However, it was not easy to test with the real spoken data on the web, since I couldn't get the spoken data with the information about the speakers' personalities and backgrounds. On the other hand, the transcript of the animation, which is also a type of a spoken corpus, was attractive for my research, because the characters have strong personalities to make the story more intriguing. So, I decided to use the transcript of "SpongeBob SquarePants", which is one of the popular cartoon animation in the United States. I used AntConc, the corpus analyzation tool, to filter out the keywords for each character from the transcript. I used the metric called "log-likelihood" that to calculate the keyness of each word. I also wanted to categorize the keywords into Part-of-Speech, however, this approach failed with the malfunction of the POS tagger while interpreting the spoken corpus. But, I succeeded to filter the keywords ignoring the POS with the reasonable results. Finally, I successfully built the speaker recognizer that works for each sentence and I qualified it with the original transcript data. Although my research has a clear limitation that it is only tested

with the fictional data, I hope that it suggests a new approach for detection of the speaker in the real world conversation, which recent voice recognizing device may need to apply.

**Author Keywords:** Speaker recognition; Log-likelihood keyness

## 1. Introduction

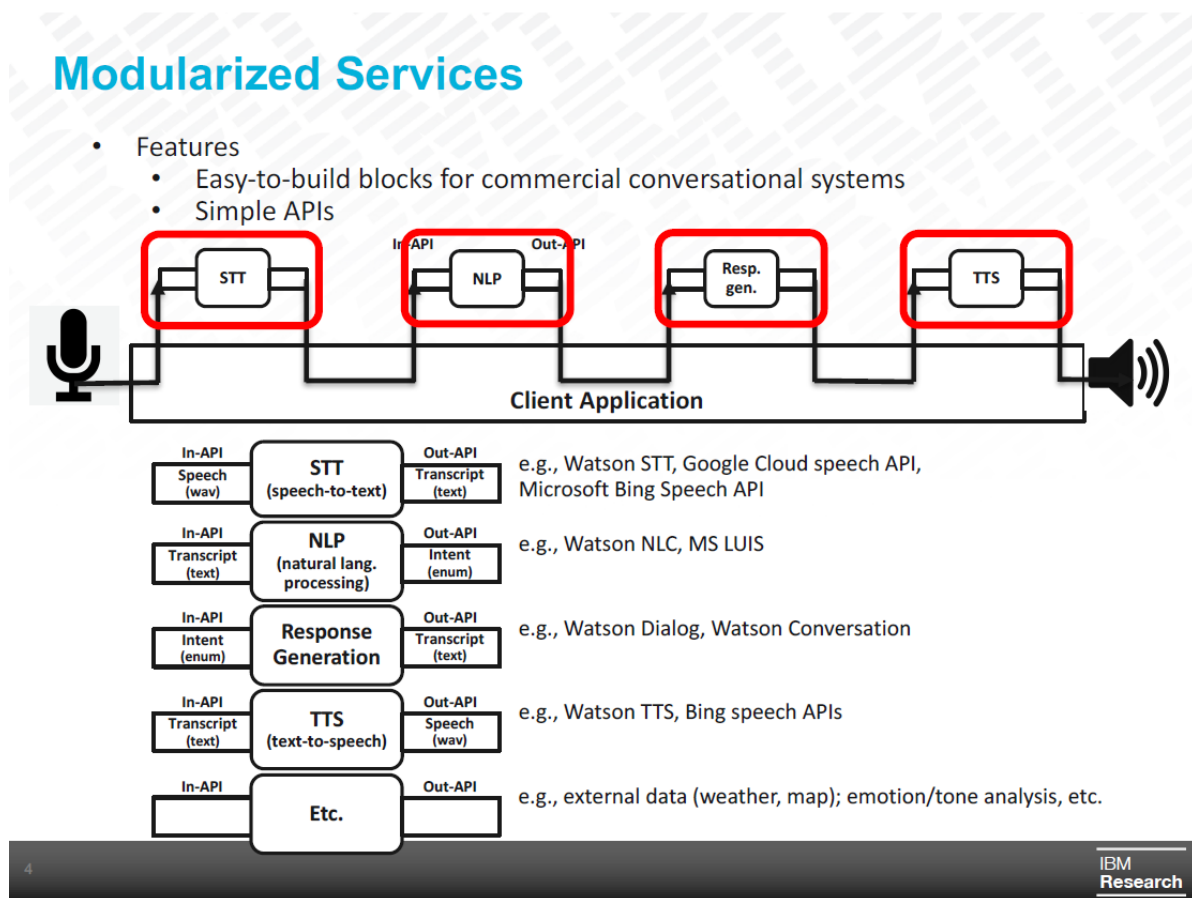


Figure 1. Pipeline of the commercial conversational system. (Reference: IBM Research [3])

Speaker recognition is defined as the identification of a person using the characteristics of voices. It is sometimes confused with the term “speech recognition”, meaning the conversion from sound to text [2]. Basically, the commercial conversational systems are composed of modularized services in the pipeline: Speech-to-text, Natural Language Processing, Response Generation, and Text-to-Speech (Figure 1) [3]. The primary approach to recognize the speaker is to embed a new software in the

speech-to-text phase detecting the real human voice. However, this approach requires sensitive voice sensors and machine-learning techniques to train the software using the user's voice, which makes the performance of the speech-to-text module significantly slow. Also it causes several security problems, because this system saves all the voice records for training data. To solve these problems, I propose a new approach to detect the speaker during the response generation phase: the keyword detection.

In the process of data collection, however, it was not easy to deal with the spoken corpus in the real world. First, I have no information about the speakers' personalities and backgrounds for the spoken corpus that I can get from the internet. Moreover, gathering the spoken data by myself was also difficult because of language problem and its reliability. Instead, the transcript of a cartoon animation looked attractive because we already know about the characters' personalities and backgrounds in the story. This type of data was suitable for my research to test my speaker recognizer module. So, I decided to use the transcript of "SpongeBob SquarePants", one of the popular cartoon animation in the United States. There are some reasons that I particularly chose SpongeBob for my research. First, the characters from the SpongeBob have strong personalities so that I can match the keyword result with my expectation to evaluate my method. Second, SpongeBob is long-run cartoon animation that has more than 10 seasons with 500 episodes with several movies so there are massive data. Third, since there are a lot of episodes, the keywords that each character speak is independent on the topic of the conversation. So, we can simulate the real world situation by testing the conversation of the characters with many topics and situations.

In this research, I used Beautiful Soup and NLTK package in the Python to crawl the data from the web. With the help of a well-organized web page that stores all the transcript from the SpongeBob episodes [4], I could gather about 538 files (2.94MB) for my research. Then I used AntConc, the freeware corpus analysis tool, to filter out the keywords. I referred to a Youtube video [5] to use this functionality, but later I found out AntConc uses the metric called 'log-likelihood keyness' calculated by using a well-defined formula that I will introduce later. After that, I made a web service that the

robot that answers the correct speaker for the inputted sentence so that people can test my program interactively. Finally, I tested with the original transcript to evaluate my speaker detection module.

## 2. Challenges and Key Approach

The key challenge for filtering out the keywords is the numeration of the keyness. In this research, I decided to use the metric called “log-likelihood keyness”. The formula of the algorithm is written as below [6][7].

**function** Log-Likelihood (**a**: Frequency of word in the speaker’s corpus, **b**: Frequency of word in the whole corpus, **c**: Corpus size of the speaker, **d**: Total corpus size [tokens])

|  $E1 = c * (a + b) / (c + d)$

|  $E2 = d * (a + b) / (c + d)$

| **return**  $2 * ((a * \ln(a / E1)) + (b * \ln(b / E2)))$

**end function**

The function determines the keyness of a word considering both the relative frequency and the size of the whole corpus. Then it takes the logarithm to soothe the proportional part. In this way, we can also compute the relative keyness among the keywords that filtered out for one character and rank their importance.

Also, I wanted to categorize the keywords into the Part-of-Speech tags (e.g. nouns, pronouns, verbs, adverbs, adjectives). I tried with both TagAnt and NLTK pos tagger for this research, however, both of them did not work well with the spoken corpus. These taggers only worked well with the finely structured sentence, and they confused many imperative sentence and interjections as nouns. Although this approach to categorizing the keywords into POS failed, I still could successfully use the keywords that calculated from the AntConc, without considering the POS.

The resulting keyness for each word using this log-likelihood approach was used for my speaker recognizer design. Since each keyword shows different importance by their numbers, I thought this can be used for the scoring system. For example, if the keyness of the word “formula” is 20.0 for Plankton and 10.0 for Mr. Krabs, this word is more likely to be used by Plankton. However, if the keyness of the word “money” is 10.0 for plankton and 100.0 for Mr. Krabs, the sentence that both includes “formula” and “money” each once results in 30.0 for Plankton and 110.0 for Mr. Krabs for the total. So this sentence is more likely to be said by Mr. Krabs rather than Plankton. By using this property, I could design the speaker recognizer that outputs the result with input sentence, and evaluated with the original transcript to check its average correct answers.

### **3. Data Collection**

For the data collection, I used a well-organized web page which stores the list of the transcripts from SpongeBob and crawled the data. I used Python and BeautifulSoup [8] to download and categorize into each speaker. In this section, I will show how I could crawl all the transcript data in more detailed level.

#### **3.1. Crawling the Transcript List**

The web page shows the list of the transcripts [4]. When you look into the code, these tables contains the URL address to each transcript page.

#	Title	Transcript
1a	<a href="#">Help Wanted</a>	<a href="#">View transcript</a>
1b	<a href="#">Reef Blower</a>	<a href="#">View transcript</a>
1c	<a href="#">Tea at the Treedome</a>	<a href="#">View transcript</a>
2a	<a href="#">Bubblestand</a>	<a href="#">View transcript</a>
2b	<a href="#">Ripped Pants</a>	<a href="#">View transcript</a>
3a	<a href="#">Jellyfishing</a>	<a href="#">View transcript</a>

Figure 2. The list of the transcripts

#	Title	Transcript
1a	<a href="#">Help Wanted</a>	<a href="#">View transcript</a>
1b	<a href="#">Reef Blower</a>	<a href="#">View transcript</a>
1c	<a href="#">Tea at the Treedome</a>	<a href="#">View transcript</a>
2a	<a href="#">Bubblestand</a>	<a href="#">View transcript</a>
2b	<a href="#">Ripped Pants</a>	<a href="#">View transcript</a>

Figure 3. The main table with “wikitable” class

```

<table class="wikitable" style="width: 690px;">
  <tbody>
    <tr>...</tr>
    <tr>
      <td>...</td>
      <td style="text-align:left">
        <a href="/wiki/Help Wanted" title="Help Wanted">Help
          Wanted</a>
      </td>
      <td>
        <center>
          <a href="/wiki/Help Wanted (transcript)" title="Help
            Wanted (transcript)">View transcript</a> == $0
        </center>
      </td>
    </tr>
    <tr>...</tr>
    <tr>...</tr>
    <tr>...</tr>
  </tbody>
</table>

```

Figure 4. The HTML code that directs the link

I first downloaded this HTML file containing the list of transcripts and extracted the table elements using BeautifulSoup. The tables had “wikitable” class, so BeautifulSoup could extract only this type of table elements. Each row of the table contains the episode number, title, and URL to view the transcript. The transcript URL is saved as “href” information in the “a” tag in the HTML, so I extracted this information and made the list of the links to download the transcript data later. I made a new text file to arrange the list information. There were 538 files. Refer to the source code “ListCrawling.py” in “data/crawling\_code”.

```

1 http://spongebob.wikia.com
2 1 Help Wanted /wiki/Help_Wanted_(transcript)
3 2 Reef Blower /wiki/Reef_Blower_(transcript)
4 3 Tea at the Treedome /wiki/Tea_at_the_Treedome_(transcript)
5 4 Bubblestand /wiki/Bubblestand_(transcript)
6 5 Ripped Pants /wiki/Ripped_Pants_(transcript)
7 6 Jellyfishing /wiki/Jellyfishing_(transcript)
8 7 Plankton! /wiki/Plankton!_(transcript)

```

Figure 5. SpongeBob\_Script\_List.txt (the first line is the reference URL)

### 3.2. Crawling the Transcript Text

Now, use the arranged list file to download by accessing each URL address. I used urllib2 library in Python to download the web data. First, we need to make a full URL address by combining the main address (<http://spongebob.wikia.com>) with the sub address (/wiki/Help\_Wanted\_(transcript)). Then we download the web content using the urllib2 library.

```

response = urllib2.urlopen(URL + href)
webContent = response.read()

```

This code downloads the HTML file of the transcript web page.



Figure 6. The actual transcript page that URL addresses

When you use the developer tools in the Google Chrome, you can check its HTML tag that has id “mw-content-text”. I used this information to extract this part using Beautiful Soup.

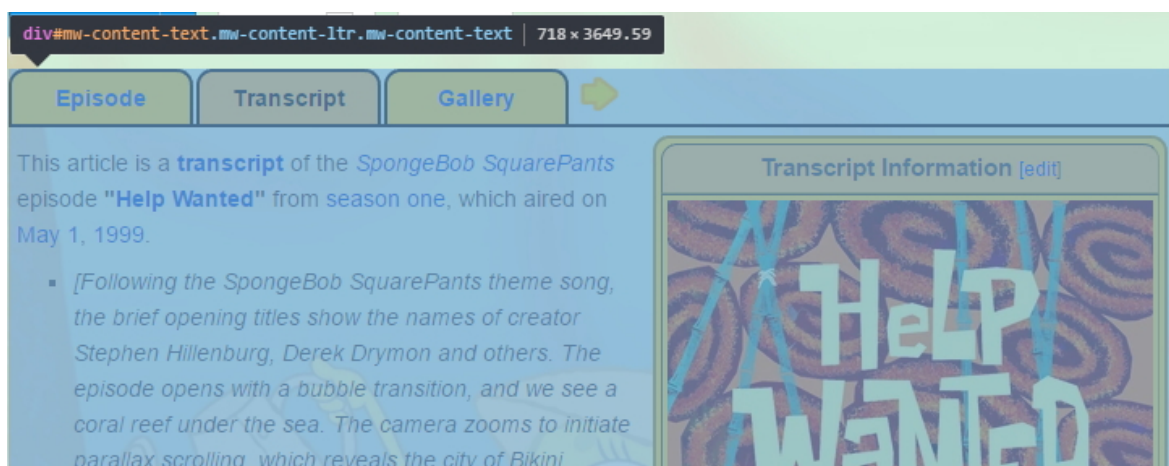


Figure 7. div element with id “mw-content-text”



Each sentence has “li” tag contained in the “ul” tagged element, which means each sentence considered as an element inside unordered list. In the next section, I will explain more detail about this process together with speaker categorization. Refer to the source code “TranscriptCrawling.py” in “data/crawling\_code”.

### 3.3. Categorization into Each Speaker

```

▼ <li>
  <b>French Narrator:</b>
  " Ah, the sea... so fascinating. So wonderful. Here, we see Bikini
  Bottom, teeming with life. "
  ▼ <i>
    "[Shows from left to right Patrick's, Squidward's, and SpongeBob's
    houses. Zooms in on SpongeBob's house]"
    </i>
    " Home to one of my favorite creatures, SpongeBob SquarePants. Yes,
    of course he lives in a pineapple, you silly. "
    ▼ <i>
      "[SpongeBob's alarm sounds; he wakes, but is unaffected by the
      annoying sound, and with a smile turns it off. He climbs from his
      bed to a ladder leading to his diving board]"
      </i>
    </li>
  ▼ <li>
    <b>SpongeBob:</b>
    " Today's the big day, Gary!
    "
    </li>

```

Figure 8. HTML code for the list of sentences in the transcript

Each sentence is tagged as “li” meaning the list element. The sentence contains the information about the speaker inside the “b” tag (bold), and the lines. There is “i” tag (italic) which describes the action or the situation. I used “b” tag to categorize the speaker and write the sentence ignoring the content inside the “i” tag. (Figure 8)

```

for div in page:
    if div.find('ul'):
        for li in div.ul.select('li'):
            v = li.b
            if v:
                speaker = v.text
                li.b.replace_with('')

                while li.i:
                    li.i.replace_with('')

                content = li.text

                #print speaker
                #print content
                file.write(speaker + "\t" + content)

```

Figure 9. Python code for speaker categorization using “b” tag, and “i” tag

Later, we read all the downloaded files and split each sentence with “\t” token (tab character) because it divides the speaker and the sentence [note this line: `file.write(speaker + “\t” + content)`]. In this way, I could successfully categorize the scripts for each character. In this way, I could crawl the script data about 2.94 MB (2977 KB).




 SpongeBob.txt	2017-04-09 오후...	Notepad++ Docu...	785KB
 Mr. Krabs.txt	2017-04-09 오후...	Notepad++ Docu...	318KB
 Squidward.txt	2017-04-09 오후...	Notepad++ Docu...	268KB
 Patrick.txt	2017-04-09 오후...	Notepad++ Docu...	218KB
 Plankton.txt	2017-04-09 오후...	Notepad++ Docu...	151KB
 Sandy.txt	2017-04-09 오후...	Notepad++ Docu...	88KB
 Narrator.txt	2017-04-09 오후...	Notepad++ Docu...	27KB
 Mrs. Puff.txt	2017-04-09 오후...	Notepad++ Docu...	26KB
 Patchy.txt	2017-04-09 오후...	Notepad++ Docu...	25KB
 Karen.txt	2017-04-09 오후...	Notepad++ Docu...	19KB
 Pearl.txt	2017-04-09 오후...	Notepad++ Docu...	19KB

Figure 10. Categorized files for each speaker

## 4. Keyword Calculation

In this section, I will describe the way to calculate and rank the keywords using the log-likelihood keyness. I used AntConc for the primary tool to calculate the keyness. I also tried to categorize the keywords into POS using Python NLTK package, but it failed since the POS tagger did not work with

the spoken corpus correctly. So, I used the result of the AntConc, which does not actually categorize into keywords to build the speaker recognizer. It uses the scoring system that gets the value of the keyness when the word hits in the list.

#### 4.1. Keyness Calculation Using AntConc

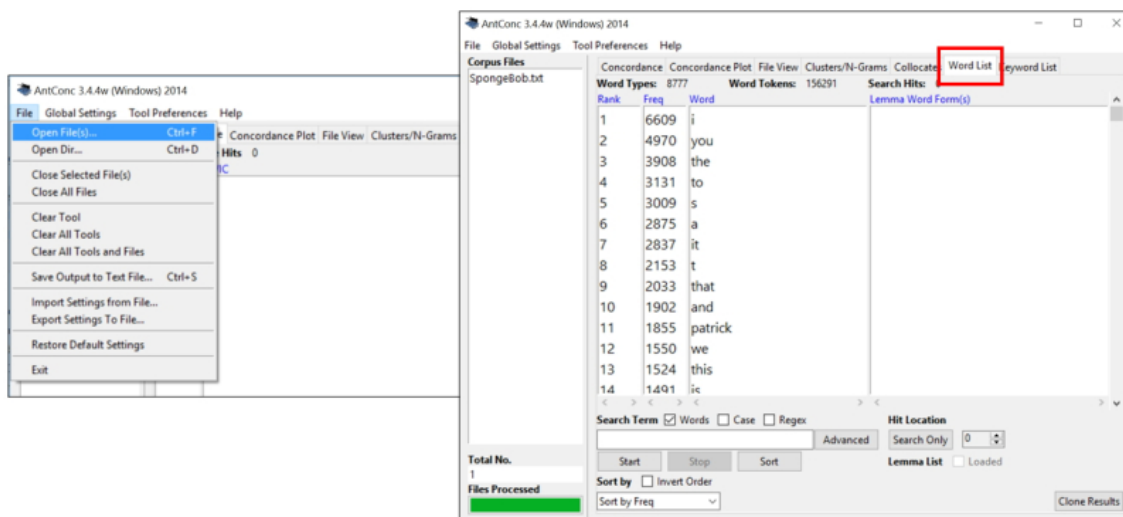


Figure 11. Open the script file and find the word frequency.

For the first step, you choose the target speaker's script and open in the AntConc. Then go to Word List tab, and calculate the frequency of each word by clicking on the Start button.

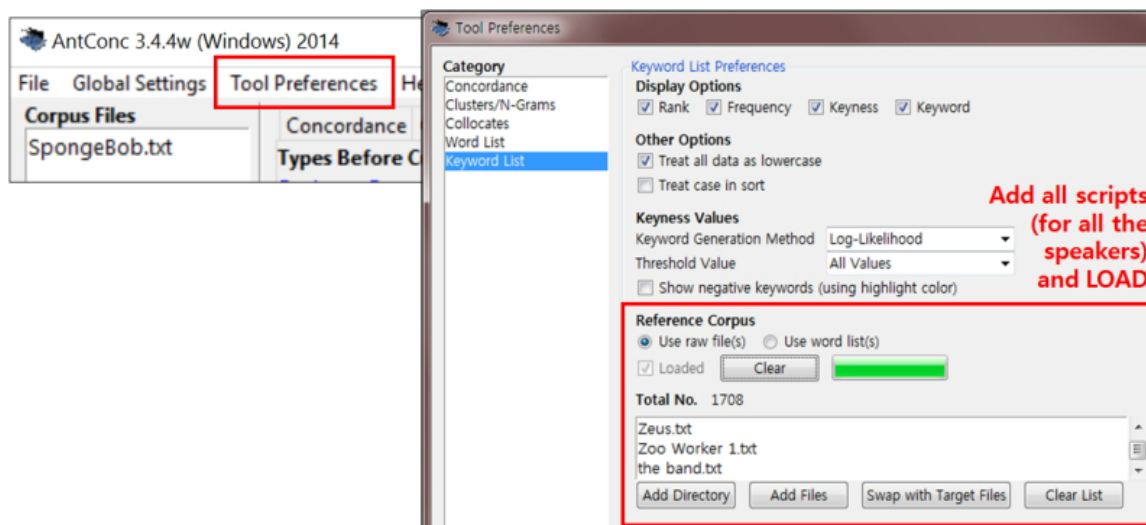


Figure 12. Load all the other script files as reference corpus.

Next, load the scripts for all the speakers (including the targeted speaker) and click on load button. Recall that we need the reference corpus to calculate the log-likelihood keyness that is introduced in the Section 2.

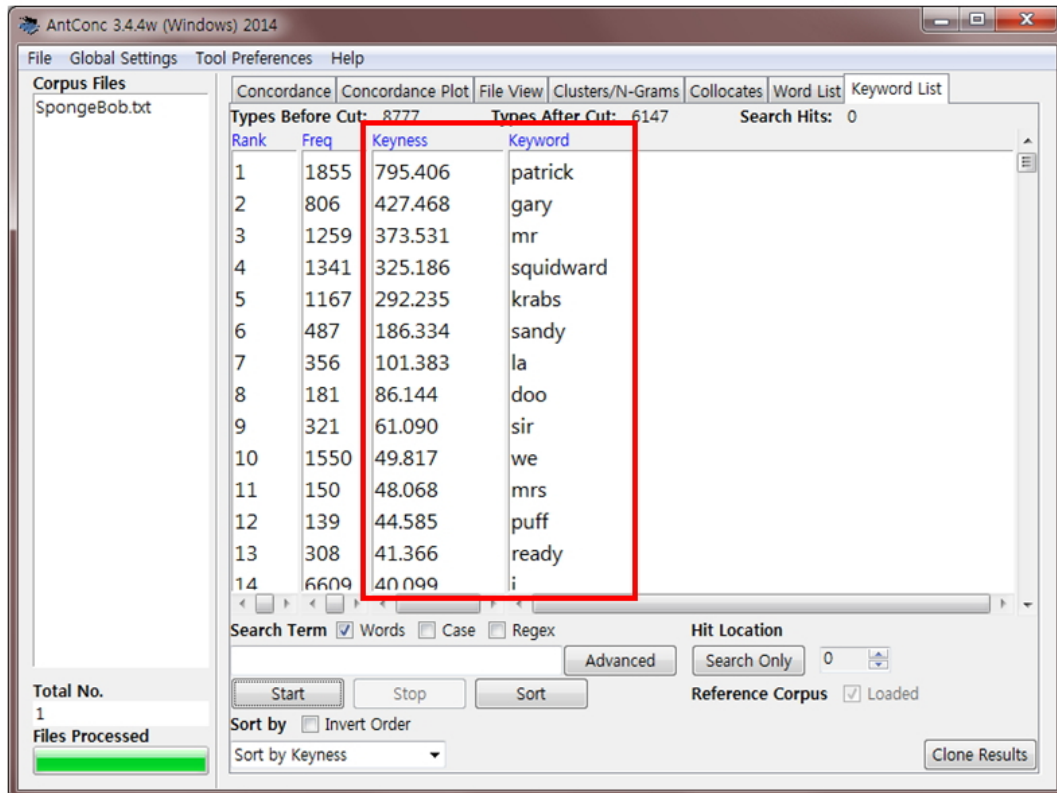


Figure 13. Calculate the keyness

Finally, go to Keyword List tab and click on the Start button. The result is sorted by the keyness as a default. You can change the order by clicking on the dropdown button below. When you get the result, you can save the file by pressing “Ctrl + s” on the keyboard. I did the same process with the scripts of Gary, Mr. Krabs, Mrs. Puff, Patrick, Plankton, Sandy, SpongeBob, and Squidward who are the main speakers in the story. I will discuss about the result in the Section 4.3 Evaluation.

## 4.2. Part-of-Speech Categorization

Although I failed with this approach, I wanted to put this section for other researchers who wants to deal with the spoken corpus to give them some precautions. In fact, the reason why I did this approach

is that I wanted to ignore redundant words other than the nouns, verbs, adverbs, adjectives, which are not considered to be keywords.

Basically, the key idea to find out the keyword is same with AntConc: using the log-likelihood keyness. In this case, I made up my own custom function to calculate keyness.

```
def keyness(a, b, c, d):
    a = float(a)
    b = float(b)
    c = float(c)
    d = float(d)
    E1 = c*(a+b) / (c+d)
    E2 = d*(a+b) / (c+d)
    ka = (a*math.log(a/E1))
    kb = (b*math.log(b/E2))
    return 2*(ka+kb)
```

Figure 14. Python code to calculate the keyness

Then we categorize into POS tag. We first categorize and calculate the frequency of the words.

```
ttypes = ['NN', 'NNS', 'NNP', 'NNPS', 'VB', 'VBD', 'VBG', 'VBN', 'VBP', 'VBZ', 'JJ', 'RB']
```

Figure 15. Types of POS tag to extract: [NN: nouns, NNS: plural nouns, NNP: pronoun, NNPS: plural nouns, VB: normal verb, VBD: verb past tense, VBG: verb gerund or present participle, VBN: verb past principles, VBP: verb non-3rd person singular present forms, VBZ: verb 3rd person singular present form, JJ: Adjective, RB: Adverb]

```

f = open(fname + ".txt", "r");
for line in f:
    line = line.strip()
    if not line: break
    words = nltk.pos_tag(nltk.word_tokenize(line))
    for (w, tag) in words:
        w = w.lower()
        if w in stopwd: continue
        if len(w) <= 1: continue

        for t in ttypes:
            if tag == t:
                if ddic[fname][t].has_key(w):
                    ddic[fname][t][w] += 1
                else: ddic[fname][t][w] = 1

                if ttwcount[fname][t].has_key(w):
                    ttwcount[fname][t][w] += 1
                else: ttwcount[fname][t][w] = 1

            ftcoun[fname][t] += 1
            ttcoun[t] += 1

```

Figure 16. Do the math for further keyness calculation

However, during this phase, I found that lots of POS tags were turned out to be incorrect. For example, this is the POS tagging result from Mr. Krabs' script.

Part-of-Speech	Word	Frequency
NN	Spongebob	105
NNP	Spongebob	415
NNP	Spongebobs	1
NNP	Spongebob-bob-bob-bob-bob-bob-ob	1
VP	Spongebob	1
VBP	Spongebob	2
JJ	spongebob	4

Table 1. Incorrectly tagged word "SpongeBob" in Mr. Krabs' script

SpongeBob which must be considered as Pronoun was detected as a noun in many times and sometimes verbs or adjectives. This is because the POS tagger does not work correctly with

imperative sentences that are common in the spoken corpus. For example, “SpongeBob!” is categorized into the noun, not the pronoun. Also, the script was written informally, so there were several words that are not in the dictionary. This caused the severe problems in the keyword detection.

Keyword	Value	Keyword	Value	Keyword	Value
Mr. Krabs		NNS		NNP	
money	428.243331949	customers	125.084599319	spongebob	715.553146006
boy	201.144595309	patties	73.859477693	squidward	470.713274361
time	182.976696378	thanks	40.5035845413	krabby	317.25729847
spongebob	136.259241984	eyes	32.1646112534	plankton	263.806340576
day	119.389050119	boys	30.9733293551	krusty	253.460993887
way	105.114272387	people	29.7820474569	mr.	232.770300508
work	99.9234441213	kids	28.5907655586	krab	231.04607606
something	98.6257370548	dollars	28.5907655586	patty	200.010035992
ya	76.5647169241	things	22.6343560672	hey	158.628649235
formula	75.2670098576	friends	22.6343560672	krabs	124.144160271
thing	71.3738886581	pants	21.4430741689	uh	113.798813582
right	67.4807674586	times	20.2517922707	okay	101.729242444
nothing	67.4807674586	years	19.0605103724	come	96.5565690996
lad	66.1830603921	bucks	19.0605103724	huh	94.8323446514
business	58.396817993	hours	16.6779465758	patrick	93.1081202032
job	58.396817993	hands	16.6779465758	pearl	87.9354468586
look	54.5036967935	days	16.6779465758	well	84.4869979622
course	54.5036967935	folks	16.6779465758	ah	82.762773514
anything	54.5036967935	minutes	15.4866646776	ooh	62.0720801355
wait	53.2059897269	boots	14.2953827793	neptune	60.3478556873

Figure 17. The keywords are categorized into each POS, but many keywords do not actually belong to their group.

[NN: nouns, NNS: plural, NNP: pronoun]

Therefore, I decided not to go over the keyness approach with POS categorization. But still, I could use the result from AntConc keyness calculation which does not care the POS tag. Refer to the “taggedrun.py” file and keyword\_result.txt in the “data/my\_keyword\_result” directory.

### 4.3. Keyword Result Evaluation

First of all, I expected the keywords for each character based on my personal view:

- **SpongeBob:** Krabby Patty, spatula, jellyfish, bubble blowing, driving license
- **Patrick:** buddy, friends, play outside
- **Mr. Krabs:** money, business


- **Squidward:** clarinet, art
- **Plankton:** secret, formula, machine, fail
- **Sandy:** Karate, helmet, science
- **Gary:** meow

These are the reason why I expected above keywords:

- **SpongeBob** likes his job and he is working at the Krusty Krab. He cooks Krabby Patty with his lovely spatula. His hobby is catching jellyfish and bubble-blowing, and he always fails to pass the driving test.
- **Patrick** likes to play with SpongeBob. In fact, he doesn't have any job, so he always wants to hang out with his best friend SpongeBob.
- **Mr. Krabs** loves money. He is a dreadful penny pincher. He likes to talk about new businesses.
- **Squidward** hates his job. His hobby is playing the clarinet or painting and enjoying art.
- **Plankton** tries to get the secret or formula of Krabby Patty. He invents lots of machines to supplement his small body, however, he always fails his plan.
- **Sandy** likes science. She uses helmet under the sea to breath. She is also good at karate.
- **Gary** is the SpongeBob's pet and the only word it can say is meow.

The resulting keywords for each speaker from Section 4.1. are listed below. I ignored some of the Pronouns especially the name of the characters, and interjections (e.g. wow, yeah, oh) to focus on the character's own personality, not the relationship with other characters. I added my comment for each result.





Rank	Frequency	Keyness	Word
13	<b>308</b>	41.366	<b>ready</b>
18	281	27.577	<b>sorry</b>
19	210	25.872	guess
20	197	24.558	<b>best</b>
21	178	23.572	<b>friend</b>
22	<b>406</b>	20.901	<b>okay</b>
25	102	18.692	<b>jellyfish</b>
26	138	17.694	<b>worry</b>
34	63	13.799	<b>spatula</b>
36	155	12.487	<b>buddy</b>

Figure 18. Keywords of SpongeBob

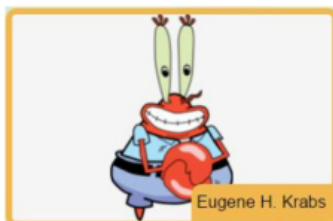
As I expected, there are words like jellyfish, spatula. Actually, SpongeBob seems very kind because he uses the words such as ready, sorry, best friend, okay, buddy. Although it is not listed in the figure, SpongeBob has many pronoun keywords such as Patrick (rank: 1, frequency: 1855, keyness: 795.406), Gary (rank: 2, frequency: 806, keyness: 427.468), Mr (rank: 3, frequency: 1259, keyness: 373.531), Squidward (rank: 4, frequency: 1341, keyness: 325.186), krabs (rank: 5, frequency: 1167, keyness: 292.235), Sandy (rank: 6, frequency: 487, keyness: 186.334). This may show that SpongeBob is the main character who calls the other characters' name most frequently in the story.



Rank	Frequency	Keyness	Word
5	98	81.819	buddy
8	33	36.387	rock
9	16	32.688	donut
10	30	32.083	glove
13	67	26.959	friend
14	33	25.547	cream
17	10	23.797	ruggie
18	24	23.601	hide
19	14	22.415	speech
20	52	22.036	play

Figure 19. Keywords of Patrick

Patrick calls his best friend SpongeBob all the time by saying the word 'buddy'. Also, he lives under the hemisphere rock. He likes to play with SpongeBob. But, it seems that he doesn't have strong keywords that distinguish him. In fact, there were many interjections such like yeah (rank: 2, frequency: 290, keyness: 142.345), hey (rank: 3, frequency: 333, keyness: 119.861), oh (rank: 7, frequency: 533, keyness: 63.824). In this aspect, Patrick is a very emotional character.



Rank	Frequency	Keyness	Word
1	<b>354</b>	467.163	<b>money</b>
2	<b>408</b>	338.615	<b>boy</b>
3	<b>1027</b>	226.042	<b>me</b>
4	90	149.547	lad(젊은이)
5	105	115.565	<b>customers</b>
6	162	84.869	<b>ya</b>
7	78	68.870	<b>boys</b>
10	138	42.642	<b>patties</b>
11	44	39.369	<b>dollar</b>
13	80	37.533	<b>free</b>

Figure 20. Keywords of Mr. Krabs

As I expected, Mr. Krabs talks about money (dollar) and the customers in his restaurant which sells the Krabby patties. Also, he is older than others and calls other characters especially SpongeBob as boy or lad.

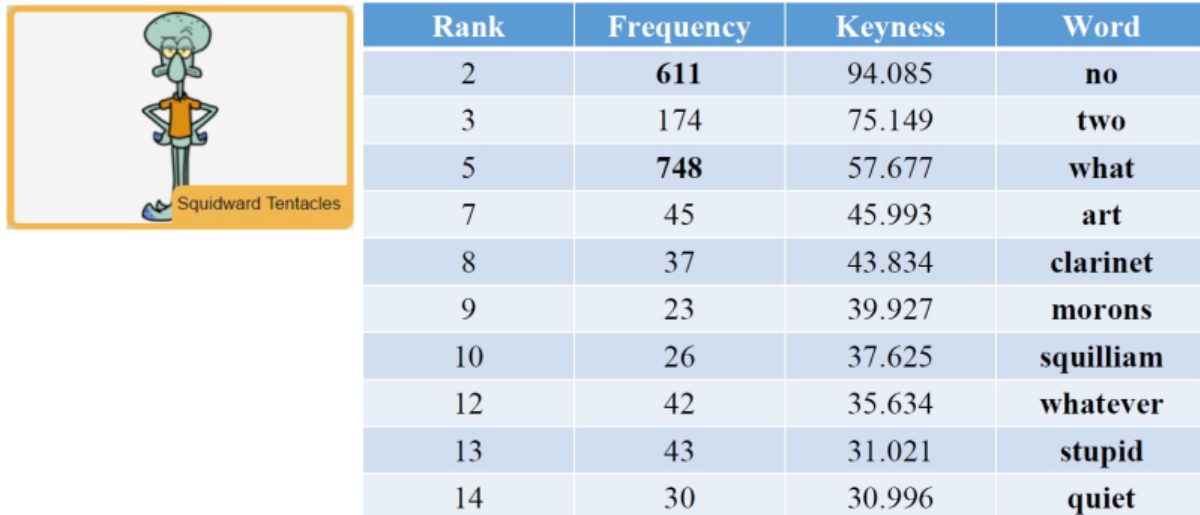


Figure 21. Keywords of Squidward

Squidward is a very cynical person. He really hates SpongeBob and says “no” for all the suggestions. He frequently indicates both SpongeBob and Patrick as “two” of you. His hobbies are enjoying art and playing the clarinet. He thinks SpongeBob and Patrick are morons and stupid. He wants to be in a quiet place because SpongeBob always makes noises.

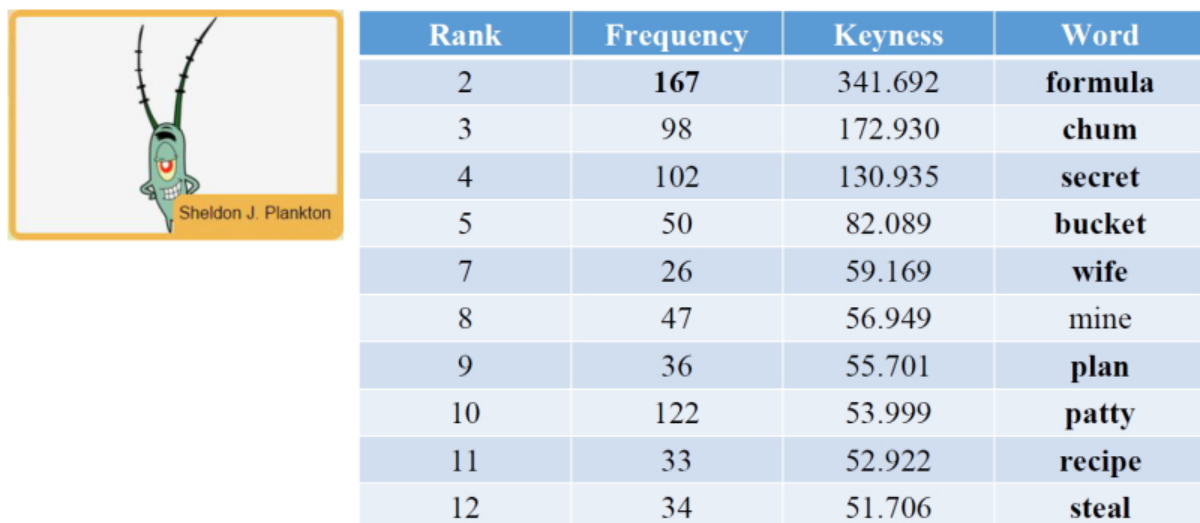


Figure 22. Keywords of Plankton

Plankton tries to steal the secret formula to make Krabby Patties and makes a plan. Also, he is the only one who has a wife. The keywords reflect his personality clearly.



Rank	Frequency	Keyness	Word
6	20	74.906	<b>critter</b>
7	28	56.301	<b>karate</b>
8	27	56.189	<b>air</b>
9	17	56.160	<b>rodeo</b>
10	13	51.327	<b>critters</b>
14	16	45.009	<b>nuts</b>
15	12	43.866	<b>experiment</b>
16	14	36.561	<b>science</b>
18	8	31.115	<b>tarnation</b>
20	11	29.212	<b>helmet</b>

Figure 23. Keywords of Sandy

Sandy is curious about other critters. She is also good at karate and sometimes teaches SpongeBob. She eats nuts, as she's a squirrel. She conducts many experiments to find new scientific discovery. She uses her helmet to breath in the ocean.



Rank	Frequency	Keyness	Word
1	<b>521</b>	5156.911	<b>meow</b>
2	4	40.943	<b>moowww</b>
3	3	30.707	<b>reow</b>
4	3	29.467	<b>mah</b>
5	3	28.446	<b>meooooow</b>
6	2	20.471	<b>meowow</b>
7	2	20.471	<b>moowww</b>
8	2	20.471	<b>mrloooow</b>
...			

Figure 24. Keywords of Gary

Gary only speaks meow, and he's the only one who speaks this word. The word "meow" shows very strong keyness.

#### 4.4. Speaker Recognizer Design

I used the resulting keyness value in Section 4.1. for the speaker recognizer design. For each speaker, we can get the keyness for each word. So, if we want to test the new sentence, we can tokenize the sentence into words and compare the sum of the keyness of each word for each speaker. For example, suppose that the word "hello" have keyness value 30 for speaker A and 20 for speaker B, and "world" have keyness value 40 for speaker A and 60 for speaker B, then the sentence "hello world" output the value 70 for speaker A and 80 for speaker B by summation. Therefore, this sentence is more likely said by speaker B. Based on this idea, I designed the speaker recognizer. But I ignored the character's name and stop words (e.g. I, me, you, can, about, after, get) [9], which is defined in the NLTK package, because it should not be considered as one's keyword. Here is the sample result for the input sentence. Refer to the score\_app.py file in the "data/antconc\_keyword\_result" directory.

**Stopword list**

a	been	get
about	before	getting
after	being	go
again	between	goes
age	but	going
all	by	gone
almost	came	got
also	can	gotte
am	cannot	had
an	come	has
and	could	ha

Figure 25. List of the stopwords

::: I can't give you the formula, even though you give me money.

['I', 'ca', "n't", 'give', 'you', 'the', 'formula', ',', 'even', 'though', 'you', 'give', 'me', 'money', '.']

why? gary 0  
why? mr. krabs 490.368  
why? mrs. puff 2.45  
why? narrator 0.647  
why? patrick 3.181  
why? plankton 344.606  
why? sandy 0  
why? spongebob 0.222  
why? squidward 27.356  
mr. krabs 490.368

Later, I created my own web server that guesses the speaker so that users can test my program interactively. I used Node.js [10], a simple web server technology, to connect my speaker recognizer module and made a simple web application. I will discuss the reliability of this system in the next section. Refer “data/myapp” for further information about Node.js server and internal system.

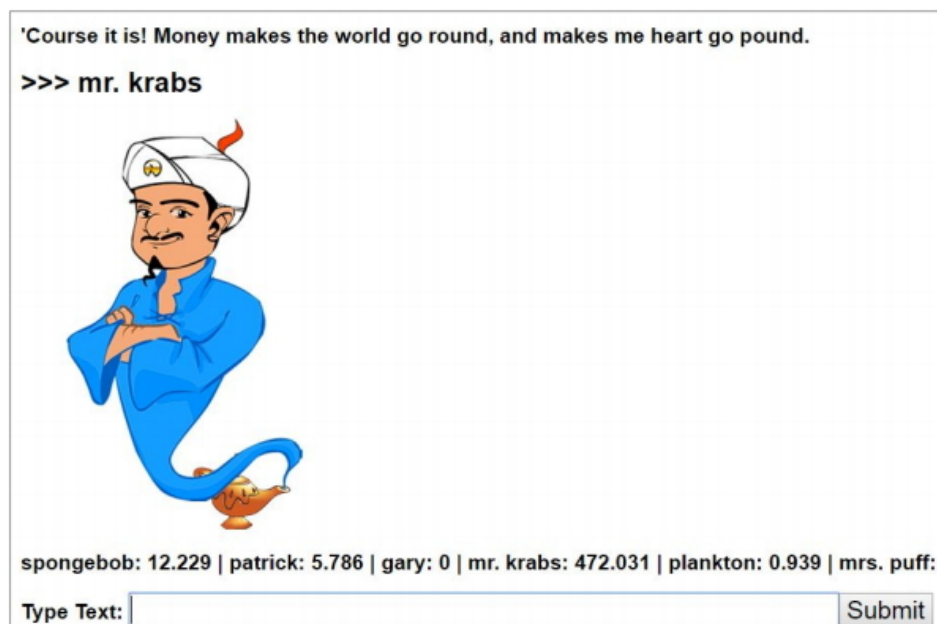


Figure 26. Speaker detector web application with Akinator's image

#### 4.5. Speaker Recognizer Evaluation

Character	Correct	Total	Percentage
SpongeBob	3150	13348	23.60 %
Patrick	2421	5206	46.50 %
Mr. Krabs	2168	4812	45.05 %
Squidward	1416	4796	29.52 %
Plankton	848	2164	39.19 %
Sandy	517	1403	36.85 %
Gary	415	420	98.81 %

Table 2. Speaker Recognizer Testing Result

(average:  $10935 / 32149 = 34.01\%$ )

I conducted the testing with the original transcript data sentence by sentence. For example, 3150 sentences out of 13348 sentences for SpongeBob turned out to be correct with 23.60 % of correction rate. In this testing process, if a sentence got no score, then I skipped that sentence. This testing module also ignores names (e.g. SpongeBob, Patrick, Krusty Krabs) and stop words. The result is listed below.

First of all, Gary got the highest score, 98.81 %. This is because he is the only one who speaks “meow” sound. However, there was an episode that SpongeBob go into Gary’s dream and Gary spoke English fluently. That resulted in the little incorrectness.

For Patrick, Mr. Krabs, Plankton, and Sandy, they showed the quite good correction rate around 40 %. That is because they have strong personalities and has their own keywords. Patrick speaks lots of

interjections, Mr. Krabs talks about business and money-related words, Plankton speaks a lot about his plan to steal the Krabby Patties secret, and Sandy is the only land animal that lives in the ocean.

Squidward showed about 30 % correction rate, which is lower than the above group. Squidward has his own hobby like making art and playing clarinet, but he is not the main story maker in many episodes. He is more like an indispensable character that makes the SpongeBob's action remarkable. So, he has slightly lower value.

Finally, SpongeBob got the lowest score 23.60 %. In fact, the major reason is that he speaks a lot more than any other characters. Note that the number of total sentences of his script is 13348. The second is Patrick and he spoke 5206 sentences. As SpongeBob is the main character, he has to speak the words that are also related to his opponent. For example, if Mr. Krabs teaches SpongeBob how to deal with money, SpongeBob may also speak about money. If Patrick asks SpongeBob to go out to play, SpongeBob may also say the similar interjections as Patrick does. So, he gets the lowest characteristics among the other characters, which resulted in the lowest score.

## **5. Discussion**

The biggest question about my research is that is this approach can really be applied to the real devices on sales. Although exceptionally my speaker recognizer could detect Gary perfectly for about 99 %, this is because he is a pet. Generally, the average correction rate was 34 % and this number seems low if we target to support multiple users in the general situation.

However, we don't have to care too much about 56 % of incorrect answers. Those sentences that my module failed to recognize are the more likely the sentence with general words like "bring me water", "where is the nearest restaurant?". In these cases, we don't need to provide customized service for each user. But still, I admit one of the biggest disadvantages of my approach is that my module can not support instant voice detection, which can't identify the user with sentences with common words.



On the other hand, The situations that my module can show the performance is during the conversation. For example, when there is a family talk, my module may listen to the conversation and do the speech-to-text process to convert into sentences. Then for each sentence, my module can detect the speaker and support the customized service instantly.

There is a limitation that my module gets low correction rate as more people use. This is because, if there are more users, keywords could be overlapped and result in the increase of the incorrection rate. However, if there are the small number of common users may show better performance. For example, my module can run effectively with the family members because the number of users is 3~5. In addition, for the family users, there are mom and dad who have different gender, and kids with different ages and personalities, so they may have distinguishable keywords.

In addition, there is less security issue because all my module utilize is the frequency of each word token from each sentence. The log-likelihood keyness does not require the relationship between other words so my module can get the result by simply counting the number of each word token. On the other hand, the voice recognizer with high technology can cause privacy issues as it saves all the recorded voice files. This approach can cause some legal problems with privacy issues.

Furthermore, my approach is very simple and do not require lots of costs. This approach can still use the speech-to-text module (e.g. Watson STT, Google Cloud speech API, Microsoft Bing Speech API) that is available on the market. We can still use all the available services that companies provide. My approach would suggest a better approach for the small companies or individual developers aside the big IT companies.

## **6. Conclusion**

Using the log-likelihood keyness, I could successfully filter out the keywords for each character. This metric also considers the relative keyness between other words so I could utilize for my speaker

recognizer design. My speaker recognizer showed average 34 % of accuracy but for some characters (except for Gary) showed considerably high values like 40~45 %. My speaker recognizer can not support instant voice recognition using sound, but it may perform nicely when there is a conversation between the static users. Also, it is a very simple method and does not require extra time for calculation and cost for implementation. My approach is well modularized, so we can still use the Speech-to-Text modules that major IT companies provide. Plus, there is less security issue than the accurate voice sound recognition. Therefore, for the small companies or the individual developers, my approach could be better if they need to implement the functionality using speaker recognizer.

## 7. Implementation



Figure 27. The example of implication in the family talk.

My speaker recognition module can be used for the speaker AI assistant to support customized service for multiple users. For example, if there is a family conversation and someone said “Mom, I’m hungry”, the AI may recognize that speaker is Mike and guide him to the location of his favorite

cookie (Figure 27). Also, speaker recognizer can make a better conversation between the users. For example, the AI assistant listens to the family talk and point out whether parents are treating their children well based on the pedagogy knowledge. So, if there is a problem in the parent's attitude for children, the AI assistant can suggest a better way treat with their lovely children to educate them well. Refer to the paper about the parenting software that directs the parents to play with their children more educative way using personalized AI service [11].

## 8. Future Research

### 8.1. Hypernym Approach

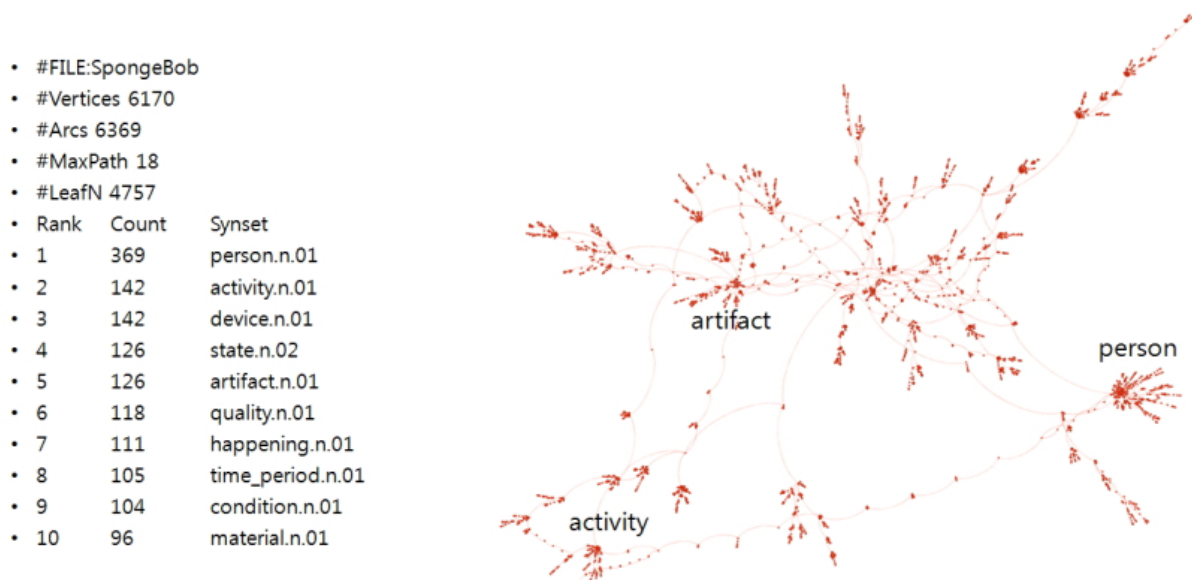


Figure 28. Hypernym network of SpongeBob script for nouns

In order to look at the more abstract concepts between the words and their relationships, we may apply the hypernym approach. According to Wikipedia, hypernym is defined as below.

In linguistics, a hyponym (from Greek *hupó*, “under” and *ónoma*, “name”) is a word or phrase whose semantic field is included within that of another word, its hyperonym or hypernym (from Greek *hupér*, “over” and *ónoma*, “name”) [12].

NLTK provides hypernym functionality as free. There are several steps to make a network (Figure 28) using NLTK package: First, you need to tokenize the words and POS tag them. Second, find the synonym sets for that word matching its POS tag. Third, follow up the hypernym path. Finally, using the relationships between the word in the hypernym path, you can create the desired network.

However, note that POS tagger did not work well with the spoken corpus (Section 4.2). You need to make sure that all the sentences are well tagged, and make the hypernym tree. Also, there is need for calculating the similarity with other characters with a random hypernym network.

## 8.2. Age Detection using NPS chat corpus

```
>>> from nltk.corpus import nps_chat
>>> chatroom = nps_chat.posts('10-19-20s_706posts.xml')
>>> chatroom[123]
['i', 'do', 'n't', 'want', 'hot', 'pics', 'of', 'a', 'female', ',',
 'i', 'can', 'look', 'in', 'a', 'mirror', '.']
```

Figure 29. NPS Chat Corpus

NPS corpus is the chat corpus that is originally collected by the Naval Postgraduate School and supported in the NLTK package [13]. It contains more than 10,000 posts with anonymous usernames for privacy issue. This also categorizes into age groups (teens, 20s, 30s, 40s, and adults chat room) so we may calculate the keyness to recognize the speaker's age.

## 9. References

- [1] Blaine Kylo, "The battle of the home speaker AI assistant is coming to Canada", <http://www.straight.com/blog/921981/battle-home-speaker-ai-assistant-coming-canada> (June 8th, 2017)
- [2] Wikipedia, "Speaker Recognition", [https://en.wikipedia.org/wiki/Speaker\\_recognition](https://en.wikipedia.org/wiki/Speaker_recognition) (June 9th, 2017)

- [3] Jinho Lee, Inseok Hwang, Thomas S. Hubregtsen, Anne E. Gattiker, and Christopher M. Durham, “SCI-FII: Speculative Conversational Interface Framework for Incremental Inference on Modularized Services”, IEEE MDM 2017
- [4] Fandom, [http://spongebob.wikia.com/wiki/List\\_of\\_transcripts](http://spongebob.wikia.com/wiki/List_of_transcripts) (June 9th, 2017)
- [5] Youtube, “AntConc 3.4.0 Tutorial 10: Keyword List Tool”, <https://youtu.be/tMNm5Kb9LLE> (June 10th, 2017)
- [6] Paul Rayson, “Log-likelihood keyness”, <http://ucrel.lancs.ac.uk/llwizard.html> (June 10th, 2017)
- [7] Rayson P., Berridge D. and Francis B., “Extending the Cochran rule for the comparison of word frequencies between corpora”, *JADT 2004*, pp. 926 – 936.
- [8] Leonard Richardson, “Beautiful Soup Documentation”, <https://www.crummy.com/software/BeautifulSoup/bs4/doc/> (June 10th, 2017)
- [9] Wikipedia, “Stop Words”, [https://en.wikipedia.org/wiki/Stop\\_words](https://en.wikipedia.org/wiki/Stop_words) (June 10th, 2017)
- [10] Node.js Foundation, <https://nodejs.org/> (June 10th, 2017)
- [11] Inseok Hwang, Chungkuk Yoo, Chanyou Hwang, Dongsun Yim, Youngki Lee, Chulhong Min, John Kim, Junehwa Song, “TalkBetter: family-driven mobile intervention care for children with language delay”, CSCW 2014.
- [12] Wikipedia, “Hyponym”, [https://en.wikipedia.org/wiki/Hyponymy\\_and\\_hypernymy](https://en.wikipedia.org/wiki/Hyponymy_and_hypernymy) (June 10th, 2017)
- [13] Steven Bird, Ewan Klein and Edward Loper, “Natural Language Processing with Python”, <http://www.nltk.org/book/ch02.html#web-and-chat-text> (July 1st, 2015)