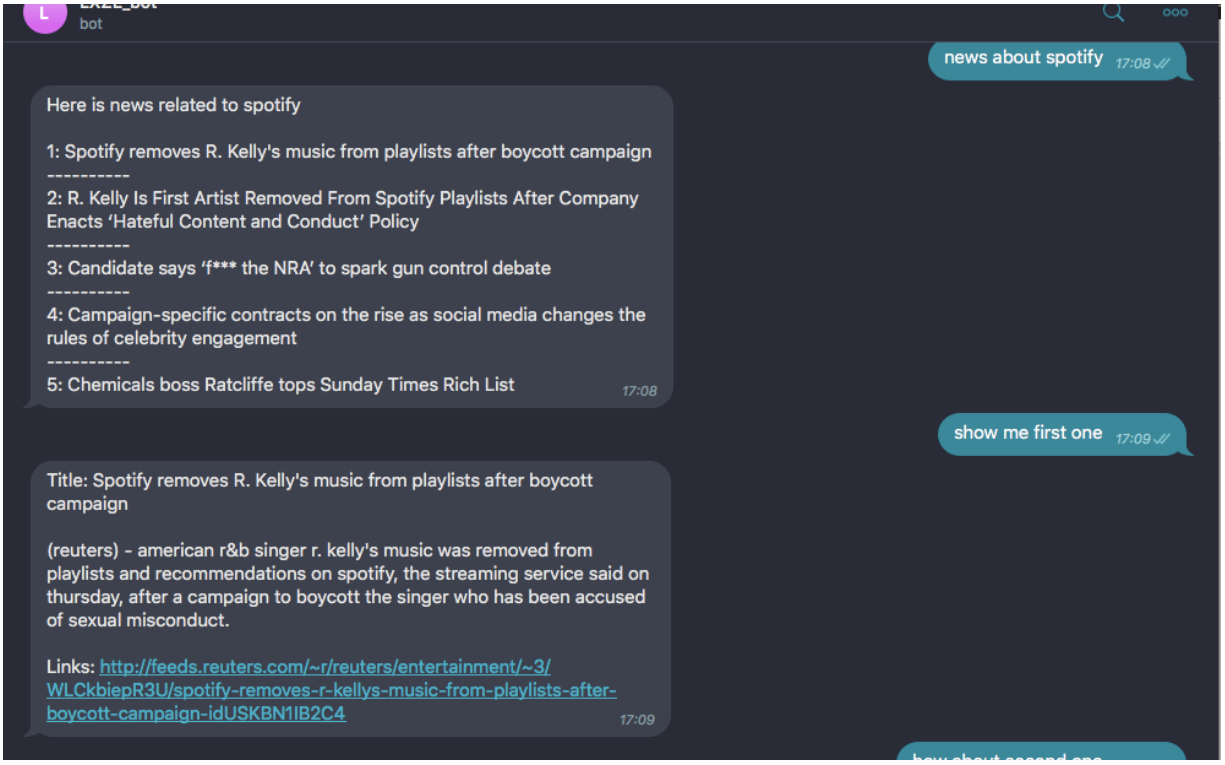
Chayapol Piyaphakdeesakun
59070701607

CPE 376/641 Natural Language Processing

Term Project Report

Topic: News Chatbot



Introduction

It's undeniable that chatbot is start involved gradually into business and services. It makes service which requires human to response or answer become easier and available at any times. So, it's advantageous that we understand how to develop a chatbot from basic.

So I want to start developing a chatbot with the NLP capability which involved with information like news and developed on existed platform like Telegram. Then, it becomes news chatbot which is a bot that automatically crawls news from several websites, extracts topics from its content. After this, a bot will let users query for a news with the certain given topic, then return a news which topic is similar to a given keyword.

My objective is to develop a chatbot that letting users be able to find a news in the pre-defined list of news websites, that matches with their interest. But for the ease of implementation, I let users query for their own topic.

All about related tools and the methodology and a conclusion what I got from implementing this chatbot will be mentioned in further.

Literature Review & Related Work

There are a bunch of chatbots working on many websites and services at this moment. In addition, a study says that acquiring a chatbot into a service bring user attention more than provided ordinary user interface. So, instead of swiping into a bunch of news on each website to find one news that I interested, I decided to create this chatbot to help me searching into news database which updated continuously.

As I want to know about which topic are in each news, I have to accomplish the NLP task which is Topic modelling or extraction. For this task, there are some of technique and algorithms that be able to achieve this task. Start with "Latent Semantic Indexing" (LSI) [1] which introduced in 1998, then, the developed version of LSI which is "Probabilistic Latent Semantic Analysis" (PLSA) [2] was introduced later in 1999. Then, in 2002, the most used version of the topic modelling algorithm "Latent Dirichlet Allocation" (LDA) [3] was introduced. Others models are usually the developed version of LDA. At the moment, I decided to do an experiment based on LSI and LDA. All of this kind of topic modelling algorithm is a statistical model which relies upon the frequency of a word inside the given document and corpus. The main concept of these statistical topic modelling is to group keywords in a dictionary in each document, then finding a pattern by sorting a keyword frequency which co-responding with a set of document. The result of found patterns is topics.

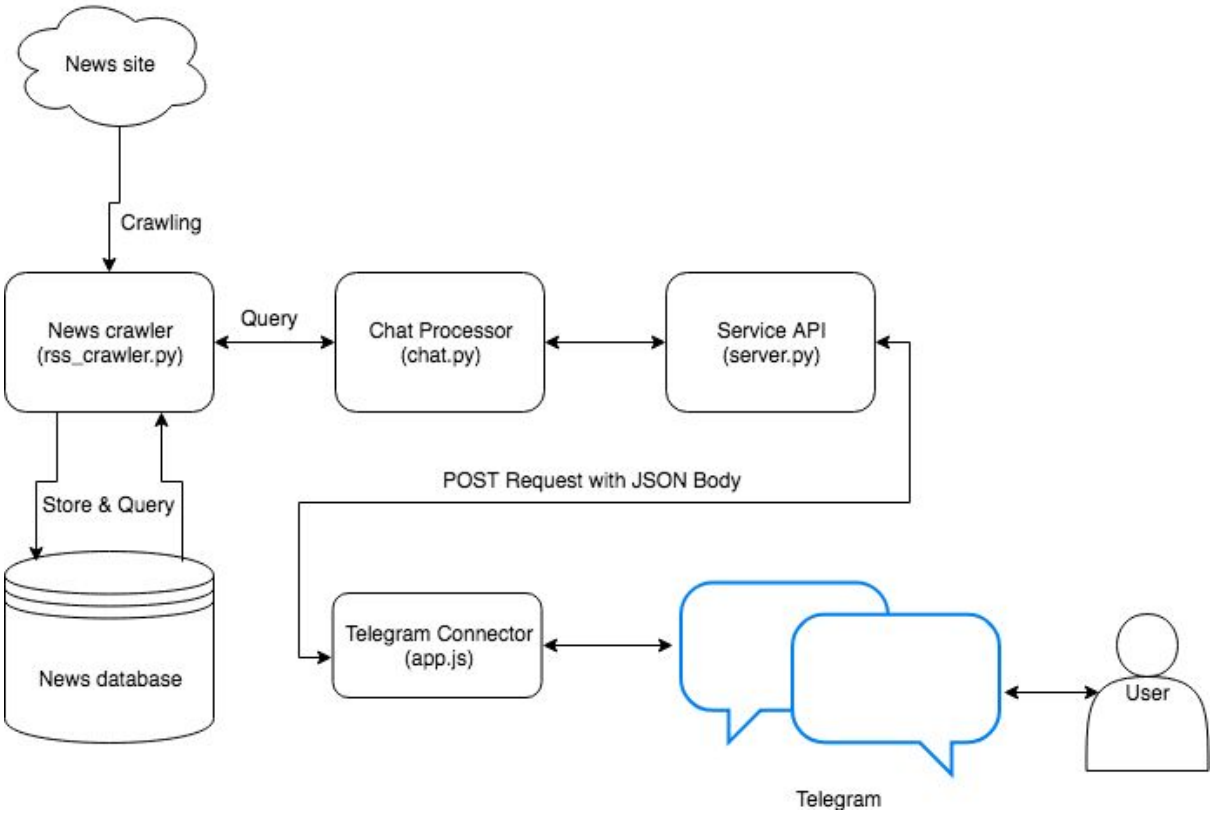
Even though, statistical models sometimes produce an error in a query result, because its process doesn't really interpret on word's semantic. So, there is another approach that obviously accessing to each word semantically, which is the Semantic Topic Modelling [4]. This model relies on word semantics. So it requires a lexical database or Wordnet. This model is an extended version of LDA, by introducing the word sense ability to the model. By the way, as this model requires Wordnet, thus to apply in other languages might be hard due to demands on word semantic within lexical resources.

In addition, to allow the user to be able to ask for the news, the chatbot requires a question answering technique. There are 2 types of question answering, which are closed-domain and open-domain. Closed-domain QA is to deal with the specific territories, such as facts or question that have an exact answer. The answer can search through the database, knowledge resource or the specific domain ontology. On the other hand, open-domain QA is dealing with a various question. In order to answer this kind of question, the system absolutely relies upon the general ontology and ordinary knowledge. So, the database would be really large and have a sufficient data in favour of find and generate an answer.

But fortunately, out chatbot just need to answer for the closed domain question, such as asking for the news and selecting the number of news in the provided list. So, the main part of our question answering system is to classify the question into the above-mentioned 2 types and to extract and obtain the right information from the given question.

Methodology

Here is my chatbot system architecture.



Chayapol Piyaphakdeesakun
59070701607

First of all, I decided to use Telegram as a conversation platform. Because of its library is really easy to access and implement with. I use library "telegraf.js" which is a library that wraps up all connection method and API calling into a simple function. At the very first, I've tried python library for telegram but there is some struggle on development, so I change to this one instead.

Then the difficulty is making Javascript and Python program be able to communicate to each other, that makes me have to create some API for my chatbot process. I choose 'Flask' which is simple microframework for web development, thus its ease of use allows me to simply create API with small numbers of lines of code. In the part of communication, I make a simple POST request on the Javascript side with JSON body that contains user input to Python processor program. Then chatbot waits for the response and returns to the user as soon as it receives the result.

In part of the chat processor, there are three main parts, which are question type detector, news retriever and news selector. I implement these modules by creating simple keyword detector, then extract the main topic or content and let process decide what to do depends on the type of question.

The last part is news crawler. In this part, I decided to use the RSS news list which is kind of web feed that allows users to retrieve websites content in a specific format. Firstly, it read the news list from the pre-defined list. I've put the RSS URL from the famous news websites such as Times, Yahoo, The Guardian and FoxNews. After that, I use the library "Feedparser" to crawl the data from URL list and stored temporarily in memory. Then process each news as a document with library "NLTK" and "spaCy", and sum it up into the corpus; subsequently creating a model using corpus and its dictionary using "Gensim". After that, I stored the news data and model with a timestamp that makes system know how old model is, So it will know which time it should update the news database and model.

Afterwards, for the query process, which to retrieve the news that similar to the given topic, this process requires the created model from a previous part, a corpus and document which processed from the crawled news site. At first, the system tokenizes the topic from chat processor and look up for its vector value compared to the model; subsequently, calculate the similarity in each document with matrix similarity function, and then sorting and filtering the result that has similarity value more than 0.5 back to the user.

Chayapol Piyaphakdeesakun
59070701607

Implementation & Result

In my chatbot, as I follow my architecture, there are 4 parts of code. Each module working on its tasks.

Start with the Telegram connector, app.js. This file works on connecting and receive the text input from the user via Telegram application. Most of "telegraf.js" function is an event-based function. So, when user input is coming, it will trigger my defined function, which is sending the input text to my Python service API and waiting for the result. After the result is been sent back, the program with selecting the result to reply to the user based on result type. If there is an error, it will tell the user that it doesn't know about input.

```
bot.on('text', async (ctx) => {
    if(userTopic[`${ctx.message.from.id}`] == undefined){
        userTopic[`${ctx.message.from.id}`] = ''
    }
    try{
        ctx.replyWithChatAction('typing')
        let res = await process(
            ctx.message.text,
            userTopic[`${ctx.message.from.id}`]
        )
        if(res['topic'] != ''){
            userTopic[`${ctx.message.from.id}`] = res['topic']
            console.log(
                `topic updated: ${ctx.message.from.id} -> ${res['topic']}`
            )
        }else if(res['type'] == 'error' && res['topic'] == ''){
            ctx.reply('Topic not told yet')
            return
        }
        ctx.reply(res.content)
    }catch(e){
        cons any error(`[ERR] ${e}`)
        ctx.reply('Error on process', e)
    }
})
```

Event receiver function

```
const process = (text, topic) => {
    return new Promise((resolve,reject)=>{
        request({
            method: 'post',
            url: 'http://127.0.0.1:5000/chat',
            body: {text: text, topic: topic},
            headers: headers,
            json: true,
        },(err,res,body)=>{
            if(!err){
                resolve(body)
            }else{
                reject(err)
            }
        })
    })
}
```

Request sender

Next is Python service API, server.py. This file acquires 'Flask' Library to create a service. The only HTTP request method that it allows access is POST method with JSON body. If there are any errors in there, it would return the error back to the user.

```python
import chat
from flask import Flask, request, jsonify

app = Flask(__name__)

@app.route('/chat', methods=['POST'])
def index_route():
    if not request.is_json:
        print('Input not json')
        return ''
    json = request.get_json()
    [input_type, result, topic] = chat.chat(json['text'], json['topic'])
    return jsonify({
        'type': input_type,
        'content': result,
        'topic': topic
    })

if __name__ == '__main__':
    app.run(host='127.0.0.1', port=5000, debug=True)
```

Request receiver site's code

The next module is the chatting processor, chat.py. There are 5 methods that working on to make a chatbot be able to talk with users.

At first, this chat function will be called and receive arguments from the service API. Then, in the chatbot, the main function will decide what function need to be done next according to a type of input query.

In the input type checker, I have defined lists of keyword for each type of question, then return its class name. My method to indicate its type is checking each word in a given sentence backwards.

```python
greet_Checklist = ['hello', 'hi', 'sup', 'greetings', 'hola', 'yo']
news_request_Checklist = ['update', 'new', 'news', 'anything', 'about', 'highlight', 'hot', 'today', 'happen']
accept_Checklist = ['one', 'this', 'show', 'display', 'pick']
def check_type(raw_sentence):
    for token in rss_crawler.clean(raw_sentence).split(' ')[::-1]:
        if token in greet_Checklist:
            return 'greet'
        elif token in news_request_Checklist:
            return 'request'
        elif token in accept_Checklist:
            return 'select'
    return 'unknown'
```

For the topic extraction, it tries to indicate which word in a sentence which is the main topic. For me, I try to use the words after "about" as a topic. I also remove irrelevant word like "today" from a topic too.

```
def findTopic(raw_sentence):
    doc = rss_crawler.clean(raw_sentence)
    token = doc.split(' ')
    if 'today' in token:
        token.remove('today')
    if 'about' in token:
        idx = token.index('about')
        topic = token[idx+1:]
    else:
        topic = [token[-1]]

    return ' '.join([t for t in topic if t not in gr
```

Use words after “about” or last word in sentence.

If the input sentence has been classified as news request, it will call the news retriever function, which passes a topic as an argument and calls the crawler's function and returns a value as a news list.

For the news selecting function, it reads the text input and detects which number the user want to read. In order to that, I defined the keyword for each number which is being shown in below image.

```
def selectNews(raw_sentence, topic):
    result = rss_crawler.query(topic, 5)
    idxLists = [
        ['first', 'one', '1'],
        ['second', 'two', '2'],
        ['third', 'three', '3'],
        ['fourth', 'four', '4'],
        ['fifth', 'five', '5'],
        ['last', 'lastest', '0']
    ]
    try:
        tokens = rss_crawler.clean(raw_sentence).split(' ')
        idx = 0
        for token in tokens:
            for idxList in idxLists[::-1]:
                if token in idxList:
                    idx = int(idxList[-1]) - 1
                    if idx != 0:
                        if 'one' in tokens:
                            tokens.remove('one')
        print(idx)
        try:
            txt = 'Title: {}\n\n{}'.format(result[idx]['title'], result[idx]['content'])
            txt += '\n\nLinks: {}'.format(result[idx]['link'])
            return txt
        except IndexError:
            raise Exception('Index not found')
    except KeyError:
        raise Exception('key not found')
```

In case of the word "last" or "latest" news, it means picking the last one. Fortunately, in Python language, it has a list slicing built-in function which can indicate the last element in a list using index "-1". So I use this index, especially for the "last" keyword.

And the last one which is news crawler, rss_crawler.py. This file is working on crawling news, processing all documents, creating a model and finding similar news on a given topic. When a chatting processor request for a news, all it does is retrieve all news in each every URL in the list, cleaning with my specific regular expression, tokenizing with libraries, creating bigram and trigram, modelling with Latent Dirichlet Allocation algorithm then stored both news data and model for a query in further.

```
def clean(txt):
    tag_remover     = re.compile(r'<[^>]+>|(@[A-Za-z0-9]+)|\.\.\.|Continue reading|(https?:\S*)|\?|\!')
    url_remover     = re.compile(r'((?:[\w-]+\.)+[a-z]{3,6}/([A-Za-z0-9]+))|\S*@\S*')
    script_remover  = re.compile(r'<script(.*)/script>')
    space_remover   = re.compile(r'^\s+|\s+$|\s+(?=\s)')
    char_replacer   = re.compile(r'\uFFFF')

    rm_space    = lambda txt: space_remover.sub('', txt)
    rm_url      = lambda txt: url_remover.sub('', txt)
    rm_tag      = lambda txt: tag_remover.sub('',txt)
    rm_script   = lambda txt: script_remover.sub('', txt)
    replace_n   = lambda txt: char_replacer.sub(' ', txt.replace('\n',u'\uFFFF'))
    return rm_space(
        rm_url(
            rm_tag(
                rm_script(
                    replace_n(txt)
                )
            )
        )
    ).lower()
```

Cleaning function

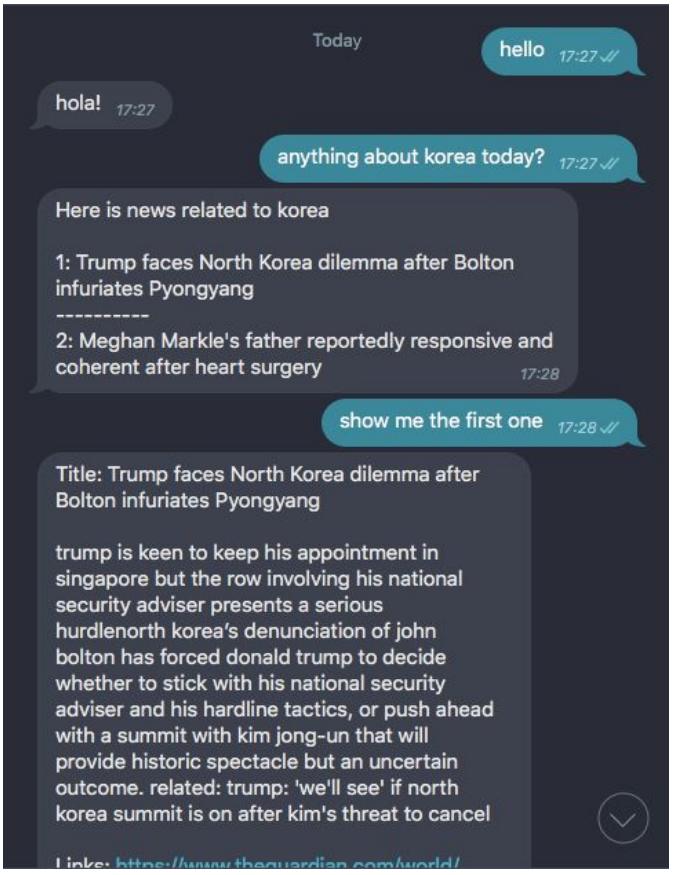
In part of query the similar topic's news, it loads the model and data back to the system, then finding the similar news using the "MatrixSimilarity" function. Within the similarities.MatrixSimilarity function, it is a function that computes Cosine similarity the vector against all document's matrix. After that, it will select only 5 news that has the highest similarity then return news that has a similarity value more than 0.5.

```
topicTokens = list(tokenize([topic]))[0]
bowTopic = dictionary.doc2bow(topicTokens)
vecTopic = model[bowTopic]

lda_idx = similarities.MatrixSimilarity(model[corpus])
similarity = lda_idx[vecTopic]
similarity = sorted(enumerate(similarity), key=lambda item: -item[1])

res = []
for doc_id, sim in similarity[:limit]:
    if(sim > 0.5):
        res.append({
            'title': html.unescape(textTitle[doc_id]),
            'content': html.unescape(clean(textList[doc_id])),
            'link' : rawData[doc_id]['link'],
            'similarity': sim,
            'doc_id': doc_id
        })
pp.pprint(res)
return res
```

Query process

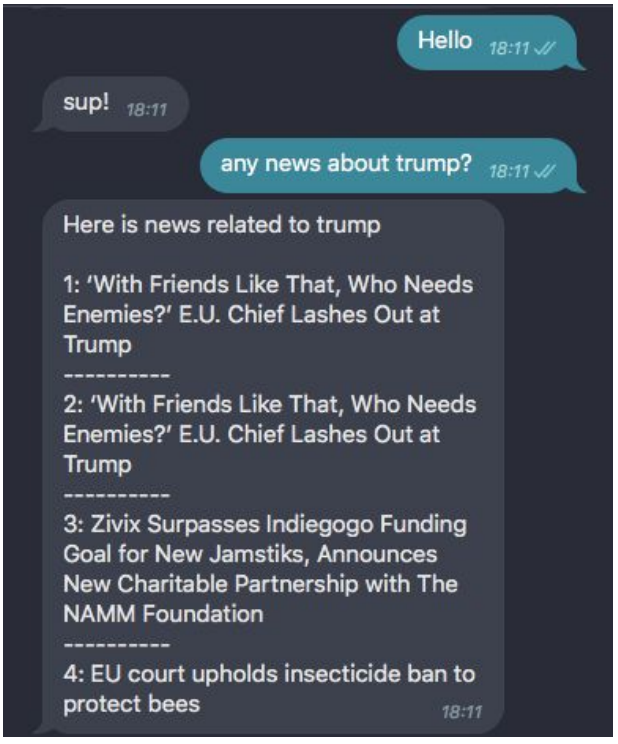
Chayapol Piyaphakdeesakun
59070701607

Result from the system after query for news about korea



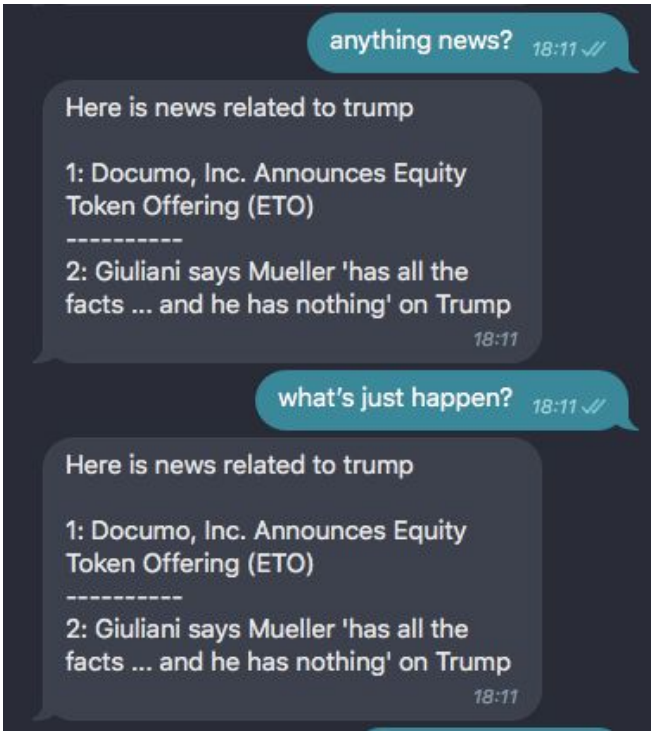
Test sentence:

- Hello
- Any news about trump?



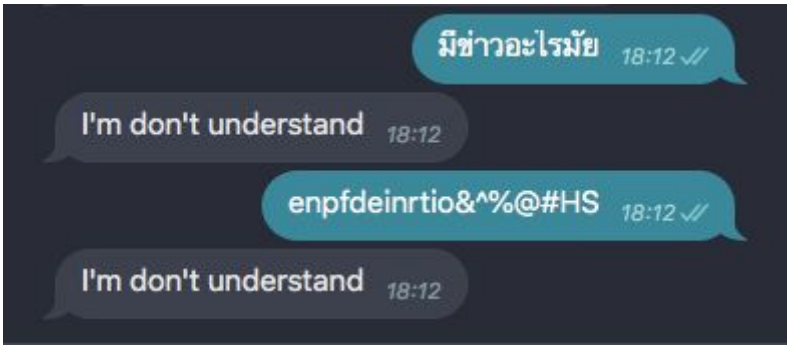
The user can greet a chatbot with formal word like hello, hi.
In case of news query, it would trigger the system to crawl and filter the most similar news with the topic to user.

- Anything news?
- what's just happen?



The result show that if user isn't provide a topic, the system will assume that user wants news with a previous given topic.

- มีข่าวอะไรมั้ย
- enpfdeinrtio&^%@#HS



In case of unknown words, the system will return an error which tell the user that it doesn't understand an input.

Conclusion

There are a lot of things that I have learned while I am implementing this chatbot. First of all, I have learned how to deal with the internet-based content, which is my crawled news. There is numerous error in its content. So, I learned the way to clean the data with my own implemented regular expression, to detect and eliminate unnecessary character that could not be tokenized. Also, I have experienced with NLP library. I used spaCy and NLTK to parsing, tokenize, creating bigram and trigram. In addition, I have used Gensim library to create an LDA model as a core function of this chatbot. The last thing is now I can develop the chatbot system to serve for the existed application like Telegram. Even there are numerous mistakes I have done in the middle of implementation, however, I can figure it out in time. The experienced I have gained from this work is really worthy and valuable.

Suggestion

As the result from this chatbot is not 100% accurate on retrieving the correct topic in news database, I would like to involve more NLP techniques to improving this system. First of all, I would like to include a model that I mentioned above, Semantic Topic Modelling. This model helps system be able to understand word semantic, so the system will know which word related to other word and be able to retrieve more news even that news might not contain the certain keyword. Also using ensemble technique, which is the technique that acquires results from several models and selects a result that passes overall criteria, might yield a better and more accurate result.

Also, this chatbot should have one NLP capability which is spell checking. Because most of the conversation scenario is start from the user. So, if the user has spelt some words wrong, then the system will not recognize the real meaning and user's intention.

Furthermore, As I develop this system to serve on my own, the question processing which detects the keyword to classify question type is absolutely limited by my knowledge. Thus, If I have a chance, I would let this service become globally accessible. So I can retrieve more user input, which means I would have a large training dataset for finding a pattern and training a question answering system. By the way, I already implement this chatbot on Telegram, so now it can be served to anyone immediately; However, the problem is just making news processing and conversation module working rapidly to handle more input queries.

References

1. Papadimitriou, Christos H., Hisao Tamaki, Prabhakar Raghavan, and Santosh Vempala. "Latent semantic indexing: A probabilistic analysis." In Proceedings of the seventeenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems, pp. 159-168. ACM, 1998.
2. Hofmann, Thomas. "Probabilistic latent semantic analysis." In Proceedings of the Fifteenth conference on Uncertainty in artificial intelligence, pp. 289-296. Morgan Kaufmann Publishers Inc., 1999.
3. Blei, David M., Andrew Y. Ng, and Michael I. Jordan. "Latent dirichlet allocation." In Advances in neural information processing systems, pp. 601-608. 2002.
4. Ferrugento, Adriana Figueiredo. "Semantic Topic Modelling." In Semantic Topic Modelling. 2015.