Chative

-make you talkative

Abstract

Nowadays, as the advancement of technology, the interpersonal distance could not be defined spatially anymore. More and more social media make people closure at ease. For instance, we are now familiar with chatting on Facebook or having a meeting on skype. Those things do bring a lot of values to humankind.

However, it's diversity makes the world intriguing. Some may good at interacting with others, of course, some would be clumsy at it. Therefore, we have a dream. For those who are not good at talking, interacting or flirting with others, we hope to accomplish them. Making them charming, talkative, well-received is our vision.

To fulfill that idea, in this project, we introduced a Google Chrome extension, Chative, that helps user keep the chatting flow going, smoothing and exciting. When a user is chatting online and facing some awkward situation that he/she can not deal with it alone, Chative could be an intelligent assistant. User can type the sentence from counterpart, which make your feel hard to continue, and Chative will give some awesome suggestions (sentences). The results are based on our fantastic database with interesting corpus. We first vectorize the sentence from the counterpart, then query it in the database to find the most similar one. After sorting them by the similarity, we use the next sentence as the suggestions. The suggestions would be already sorted by their own developing ability. With our interesting suggestions, user can create a warming chatting environment.

Goals of the project

Our goal is to build a lite and easy-used chrome extension that can provide users with awesome suggestions immediately when they are chatting on the Net. In order to reach the goal, we used a chatting corpus to provide suggestions from others' words. Comparing to generate replies by robot or some Seq2Seq techniques, our approach, kind of crowdsourcing, gives words with humanity.

Usage

https://github.com/ju5td0m7m1nd/chat_aid

Our system is intuitive to use. Just key in the sentence from the counterpart, it will provide you some amazing suggestions!

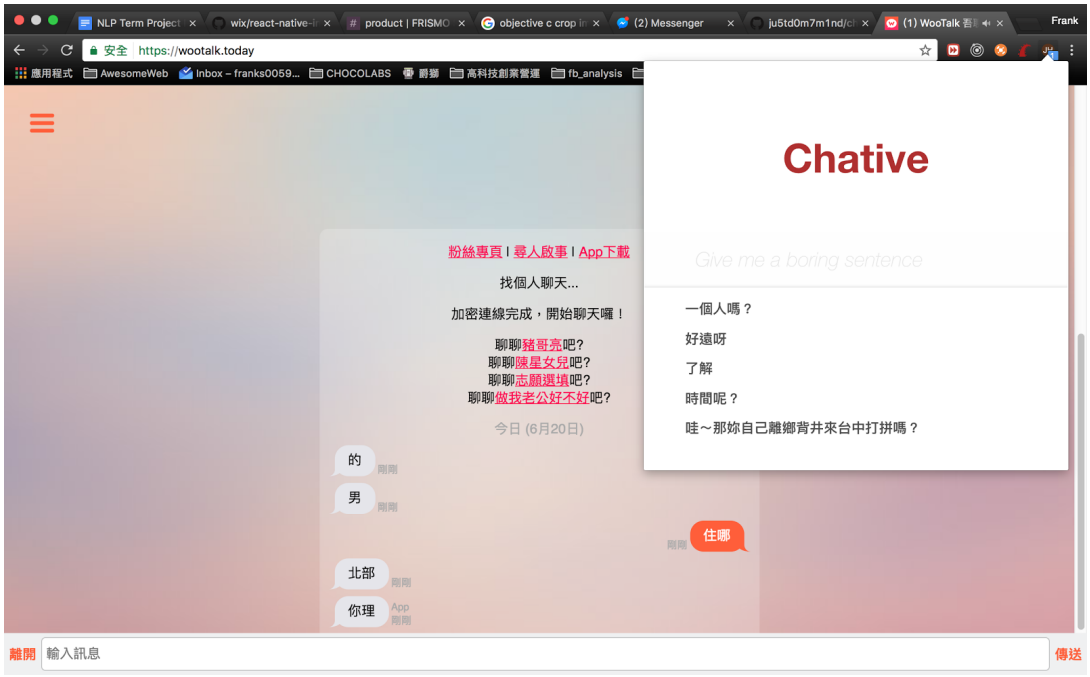

Figure1. Application Usage

Methodology

There mainly three main part of our works. Firstly, we preprocessed and cleaned the data that is useful for us. Secondly, based on the processed data, we trained our own Word2Vec for word-embedding, then build the similarity matrix. Lastly, we implement the whole system on a Google Chrome extension. The framework is shown in below:

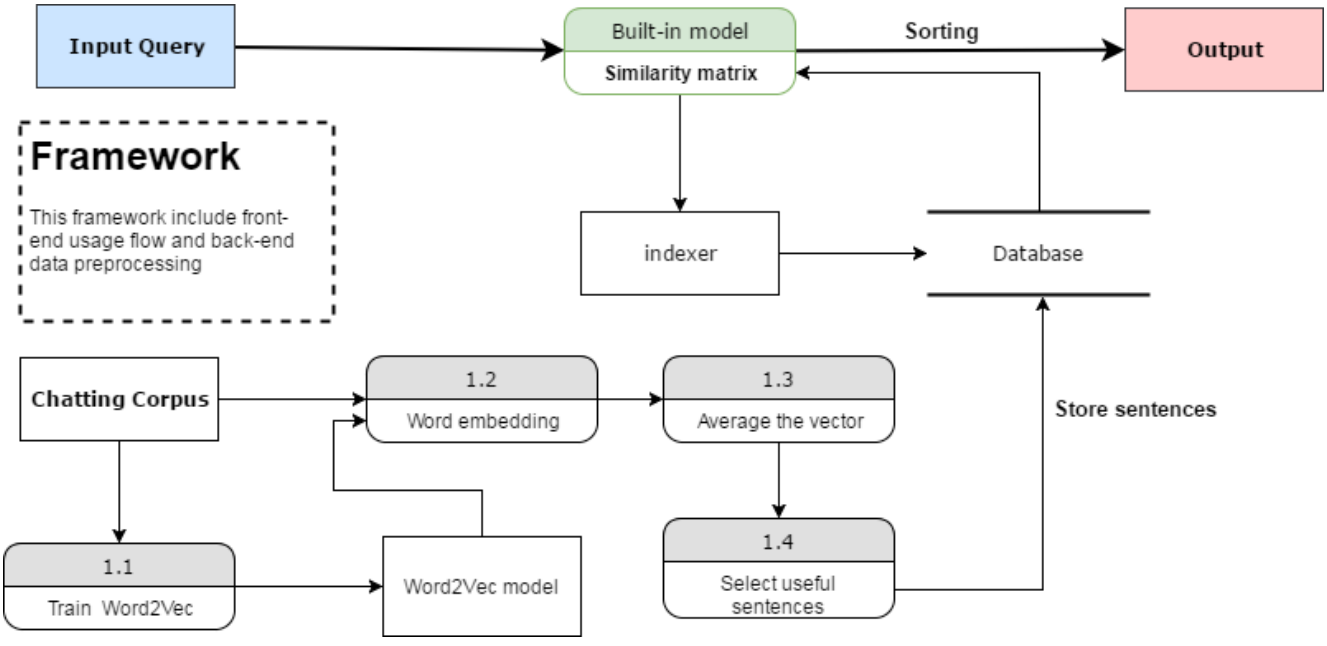

Figure2. System Framework

1. Data Preprocessing

Before we train the Word2Vec model, we did some preprocessing works to reduce the time expected to spend and generate additional data to support the selection of the candidate sentences. To begin with, we remove the sentences in duplicate by keeping data only from single gender. Next, remove the sentences that are not in so-called "frequent interval", which means there are at least ***n*** sentences in duration ***d*** follow each sentence in a time interval. Finally, mark the distance(time, number of sentence) from current sentence to the end of current interval for each sentence.

2. Sentence Vectorization and similarity matrix construction

In this section, we have to build the similarity matrix for query. First, the each sentence was separated by word using Jieba[1]. Instead of using Wiki corpus, here we used those word from chatting data to train our Word2Vec model[2]. In this way, we can find out some specific word relationships in our corpus. The example is as below:

```
model.most_similar("賴")
```

```
[('line', 0.855745792388916),
 ('Line', 0.8464864492416382),
 ('LINE', 0.8412118554115295),
 ('我賴', 0.8075134754180908),
 ('的賴', 0.801491916179657),
 ('賴啊', 0.794263482093811),
 ('用賴', 0.791531503200531),
 ('你賴', 0.7817888855934143),
 ('賴吧', 0.7787739038467407),
 ('賴呢', 0.7736353278160095)]
```

Figure3. Similar word example

From the above example, the word "賴" is very similar to "line". That's the relationship which may not be captured by using Wiki corpus. After we have the Word2Vec model, we embedded the word and average the vector. Since the sentence always consist with a few words, the performance of averaging word vector is good enough. Finally, each sentence was transformed into a 100-dimension vector. The below table shows our data frame in database.

Table1. Dataframe content

sentence	sentence_seg	sender_gender	time_to_end	number_to_end	Vector
他都不顯示	他 都 不 顯示	f	266	30	-0.9, 0.21, ...

[1] https://github.com/fxsjy/jieba
[2] https://radimrehurek.com/gensim/models/word2vec.html

Once we obtained the vector that represent a sentence, the similarity between two sentences was measurable. We then used gensim to build high performance indexing matrix[3]. To speed-up similarity calculation and comparison, we store the matrix in a spare way; that is the dense matrix was transformed into 2-tuple list. The former element of tuple is the index of the dimension and the second element is the value of that dimension.

```
[(0, 0.26488852780299998), (1, 1.28938772157), (2, -0.36831588065300003), (3, -0.18018837366300003), (4, 0.50038379803300004),
 (5, -0.77360287727799992), (6, 0.70233993418500007), (7, -0.51010624296000007), (8, -0.43425592112899997), (9, 1.8824235415099
999), (10, -1.2364192623600001), (11, -1.0471249495200001), (12, 0.101941561734), (13, 0.180532062572), (14, -0.503726689144999
96), (15, -0.30742569454000002), (16, -0.055875984020499994), (17, -0.75108227506300007), (18, 0.37381297815600001), (19, -0.01
8399186432400003), (20, 0.048028611578000001), (21, 0.72577700624200003), (22, 1.1158218823799999), (23, 0.56103913486000001),
 (24, 1.0919681675700001), (25, 1.54076870158), (26, -0.49843294546000005), (27, 0.53009905800000001), (28, 0.6801868751649999
8), (29, 0.0018918607384), (30, 0.49334737099699999), (31, -0.54979431722299998), (32, -1.1570228659499999), (33, 1.10214737151
00001), (34, 0.38517456315499998), (35, -1.06875236845), (36, 0.301012304146), (37, -1.2280117068400001), (38, 0.88487382978200
002), (39, -0.31998057011500003), (40, -1.2886287593299999), (41, 0.41184223559699995), (42, -0.313372240867), (43, 1.935604627
```

Figure4. 2-tuple format vector representation

Every time the matrix receiving a vector input, it'll compute the similarity between all of the vectors in database and input vector then return the vector's index and similarity. Chative will adopt top 50 similar sentences and extract the index+1 sentences in database as our suggestions. The output will show sorted suggestion as figure below.

```
Input: hi

Out[60]: [('hi', '謝謝^^', 238),
          ('hi', 'hi  很高興認識你 我叫小琪 先生怎麼稱呼呀？', 233),
          ('hi', '你好', 165),
          ('hi', '老師好  哈哈  帶著小朋友會累嗎？', 154),
          ('hi', '我叫小豪', 139),
          ('hi', '!!!!也太快回', 133),
          ('hi', '顆顆', 125),
          ('hi', '哈', 117),
          ('hi', '下班了', 114),
          ('hi', 'beauty lips', 85)]
```

Figure5. Raw Output format

The tuples are in format (input query, suggestions, number to end).

3. Deploy on Google Chrome Extension

・Backend：

We used ***djangorestframwork*** as backend framework, and we wrapped our Chative as a class in python. When the server started, it will first create a Chative class instance, loaded the word2vec index model and the corpus and waiting for the input. As the restful api, we setup a url "/recommend/${userinput}/", ${userinput} stands for the query user want to search.

[3] https://radimrehurek.com/gensim/similarities/docsim.html

・Frontend：

We use React as the framework of frontend which will handle the render logic. Therefore, after we get the response from server, React can dynamically re-render the frontend, so it is easy to maintain and implement.

As the data flow, we used Redux. Redux can dispatch different customized action, and in this application, it's the ajax call to server. Also Redux can handle the state while processing, that is, we can have the corresponding render strategy in three ajax state: waiting for response, success or failed.

Figure6. Application Interface

Future Work

Basically, our system has three limitation:

1. We don't have machine with enough memory, so the corpus size is limited.
2. It's a little bit slow when indexing or sorting the suggestions since it has to manipulate millions of sentences.
3. The corpus is lack of diversity. Most of the chatting goal is to hookup with others, which is unhealthy.

Therefore, in the future, we could try some distributed way to store the data and enhance the sorting speed. In terms of the corpus content, we consider it's useful now. In respect of usability of our extension, we will feature it by capturing the sentences from counterparts automatically.

Data and Resources

https://radimrehurek.com/gensim/index.html
https://github.com/fxsjy/jieba