**Group members:**

Xinyuan Chang 48541890

Shawn Zhao 11355898

Hexuan Zhang 77668655

Olivia Fang 21290465

Leo Chen 13099197

**Readme:**

1. Need to install environment:

  pip

  nltk

  numpy

  python 3.6

  textblob

  tflearn

  virtualenv

  tensorflow(1.13.1)

  tkinter

  stanford-corenlp-full-2016-10-31





2. This chatbot uses some machine learning technique(by the help of tflearn, tenserflow and etc) to train the program to response to users' input.

3. Chatbot.py is the main program, it takes input from user and compare it with existing questions in qa.json. Then it will find the tag that contains the most matched question. If there is no such one, it will print some message to show it doesn't understand.

4. qa.json refers to QuestionAnswer, it saves questions(patterns) and answers (responses). Tag in qa is used to identify which part the user input belongs to. There can be more than one question and answer saved under the same tag. When users input matches a question most. Chatbot.py will randomly pick an answer under that tag to response. Type "bye" to quit the process.

5. Chatbot.py and qa.json needs to be in the same file

**2 points**: README file in your repository describing what you've done. You are likely to just add onto the README file from A2 in this case. If you've changed a lot since A2, you will have to rewrite the README so it reflects your current submission.

5 points: At the end of your README file, include:

o a list of each feature you programmed for this assignment

* GUI
* Extra topic - Friendship
* Reasonable responses when out of topics
* Spelling error handling
* Coreference Resolution
* Sentiment analysis
* Conversation via sockets
* POS tagging

o for each item on that list, explain briefly **how** you used that feature to improve your agent's conversation or your overall system

**GUI:**

We add a GUI interface for user to input the chat frame. GUI provides a beautiful UI which helps user understand how to use the Chatbot immediately.

**Extra topic - Friendship:**

We add one more topic relate to the previous one, which is friendship. Increasing one more topic means bigger problem set, which will have a higher match rate and probability, which provides a better conversation.

**Reasonable responses when out of topics:**

We add a few more responses and make them more reasonable when the topic is out of range. It makes user feel less weird and awkward, which means a better conversation.

**Spelling error handling:**

We add an spelling error handling function to the system. Right now, user’s misspelling will be covered by the system, which gives a higher chance for our Chatbot to find matched answer.

**Coreference Resolution:**

We add a coreference resolution function to the system. Right now, when a name has been recognized, all the words related to he/she/it will be changed and matched with it, which give our Chatbot higher chance to match answers.

**Sentiment analysis:**

We add a sentiment analysis function to the system. Right now, the sentences user input will be rated between -1 to 1. Negative means feeling bad and postive means feeling good. After Chatbot has done the evaluation, it will reply based on this evaluation.

**Conversation via sockets:**

We have setted up sockets related python docs, which can be used to communicate with other Chatbots. However, in this project, we build another feature, which is self-talking and this will be covered in the following document

**Self-talking:**

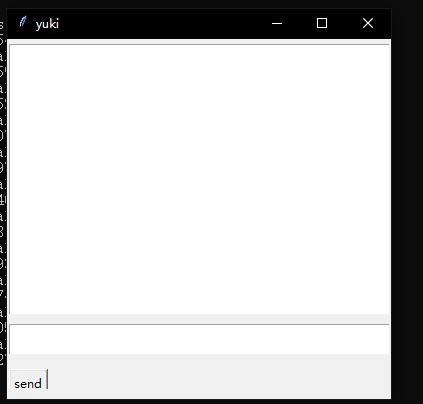
We create a loop in this program, which can be used to let Yuki communicate with itself. Parts of code is now been commented, whenever we uncomment it, Yuki will talk with itself.

**POS tagging:**

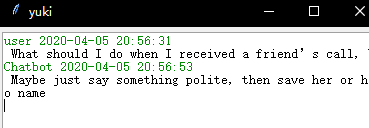
We add a pos tagging function for our Chatbot. This function prints out the part of speech of the words. However, it is not used to calculate for now. It may be implemented in furure.

o for each explanation, give a snippet of a conversation that demonstrates your feature

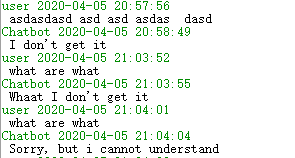
**GUI:**



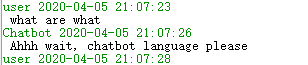
**Extra topic - Friendship:**



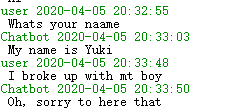
**Reasonable responses when out of topics:**







**Spelling error handling:**

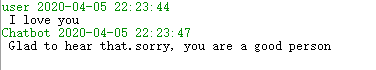


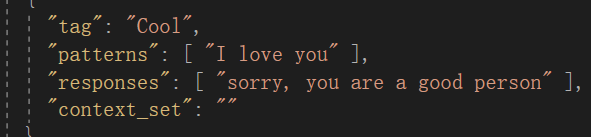
**Coreference Resolution:**



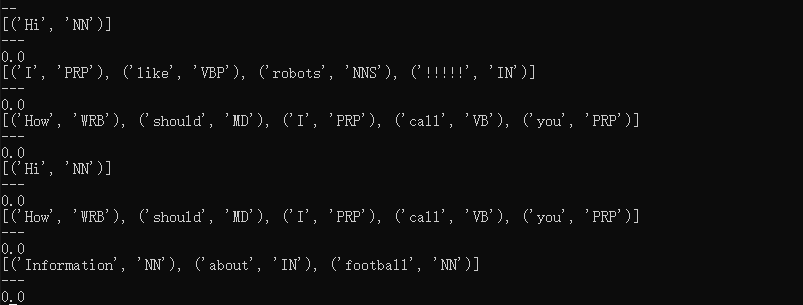


**Sentiment analysis:**

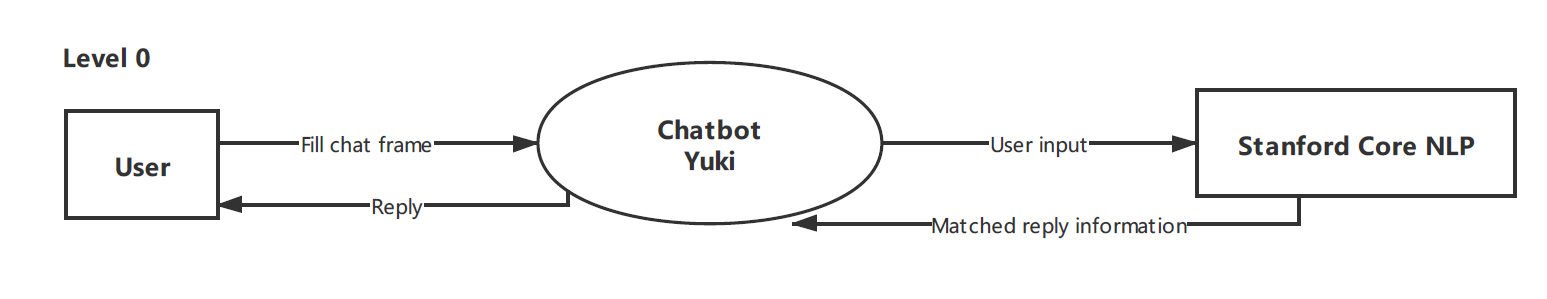




**POS tagging:**

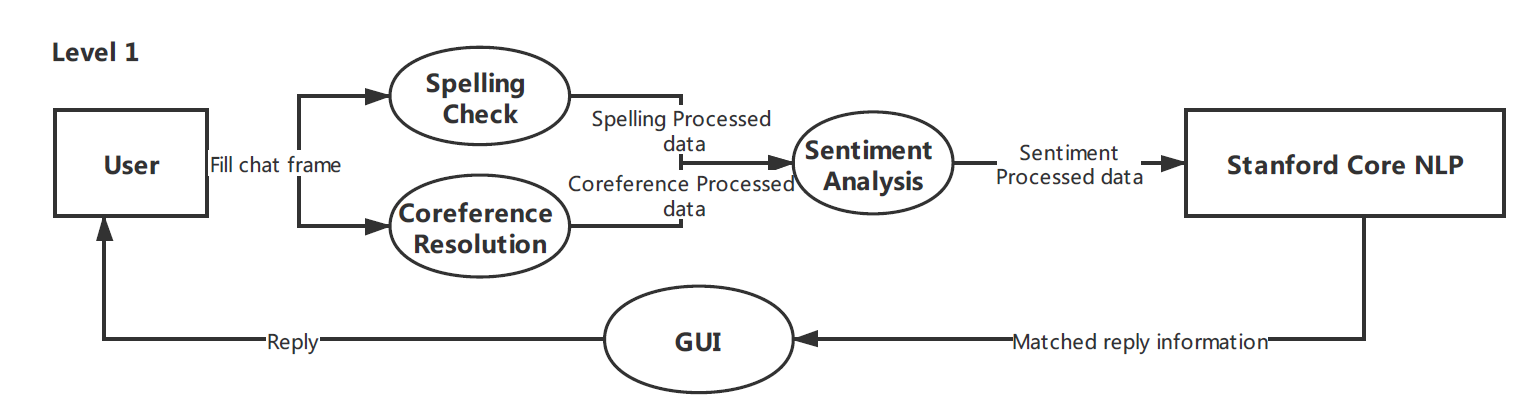


**3 points**: Provide a Level 0 DFD for your system with description.



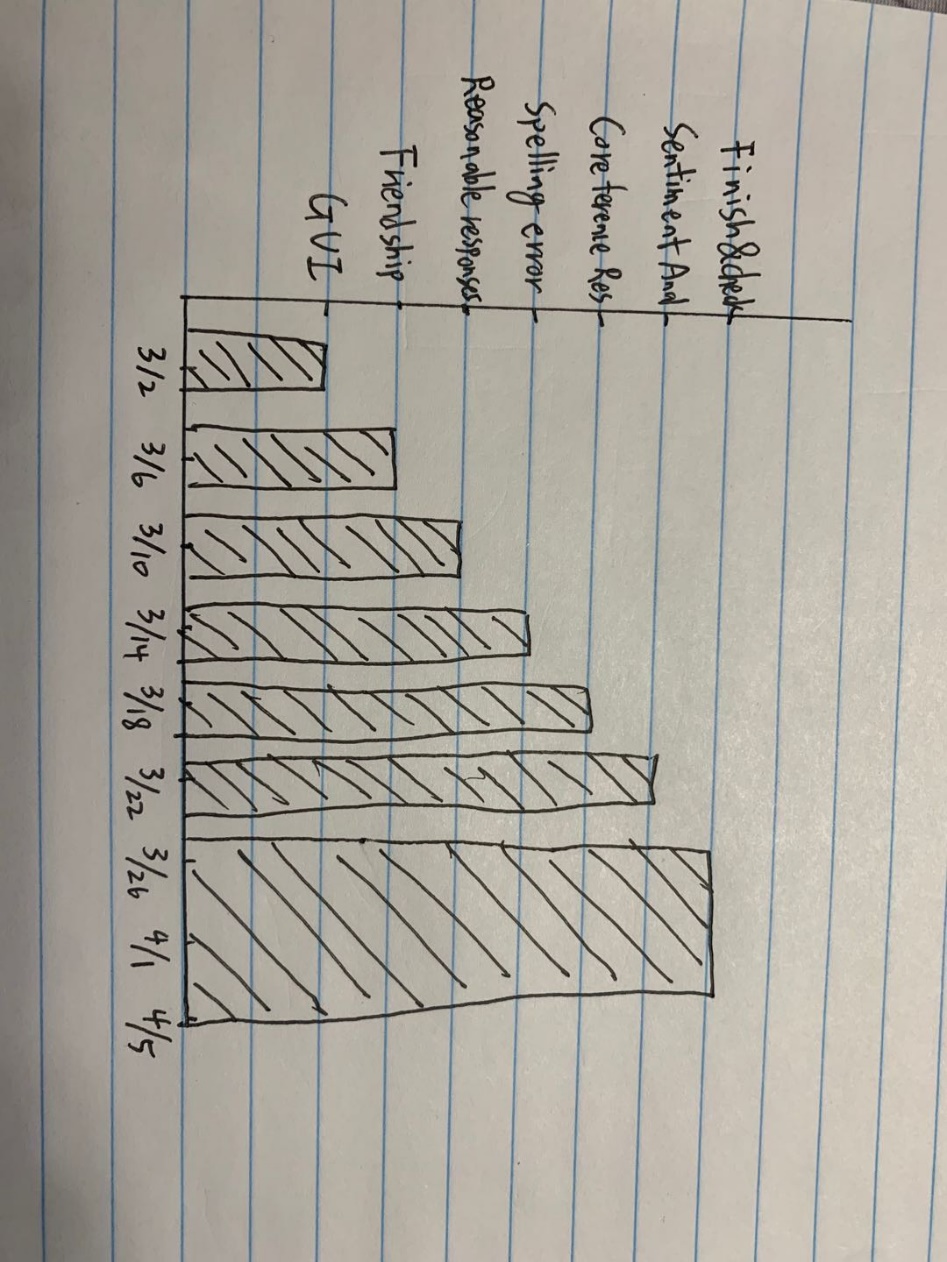
After User filling the chat frame, Chatbot will send the information to the Stanford Core NLP. Next, the Stanford Core NLP will give the matched reply information to the Chatbot and then the Chatbot will reply these information to the User.

**5 points**: Provide a Level 1 DFD for your system with description.



After User filling the chat frame, Spelling Check and Coreference Resolution functions will get the information. After the processing, Sentiment Analysis will also do the processing and finally Stanford Core NLP will get the ultimate data. It will then match reply information with the qa.json and display it by the GUI. User finally will see the reply from GUI.

**5 points:** Submission of your GitHub repository. Graph showing different features developed on a separate branch and the commits made in the repository.

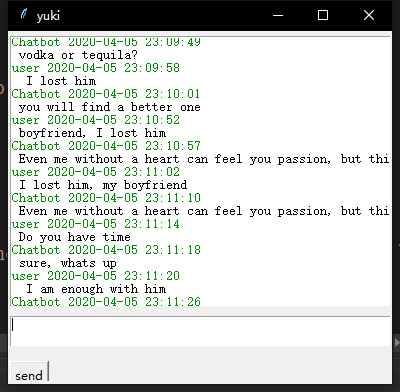
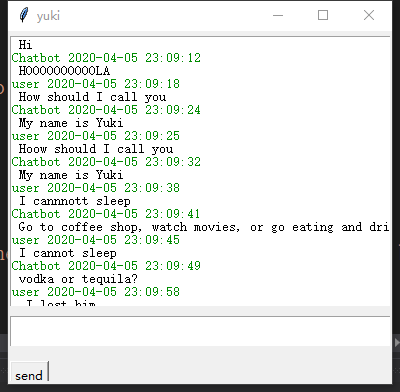


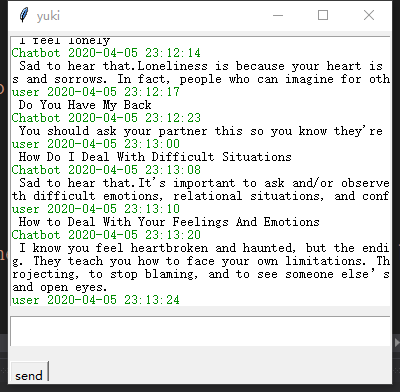
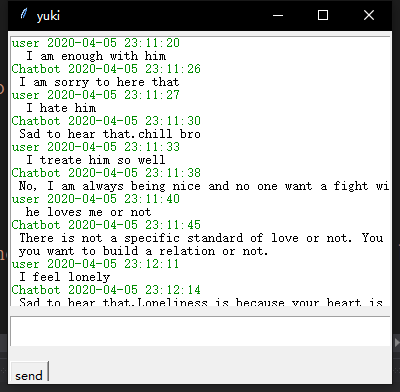
GitHub: <https://github.com/Aoishion/COSC310-Group9-Assignment3>

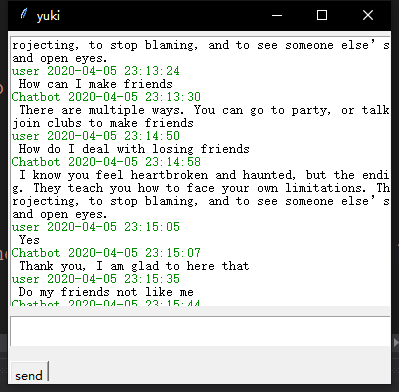
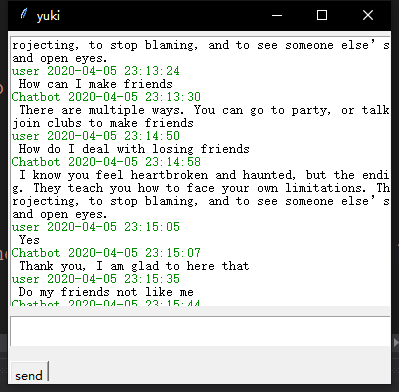
We work in group locally, and the process is shown as above. At last, we collect all of our code together, and post on our GitHub repository after we finish and test the code. Therefore, there is only one push shown on the github.

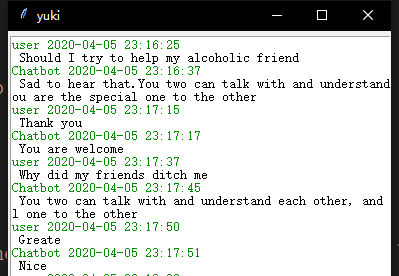
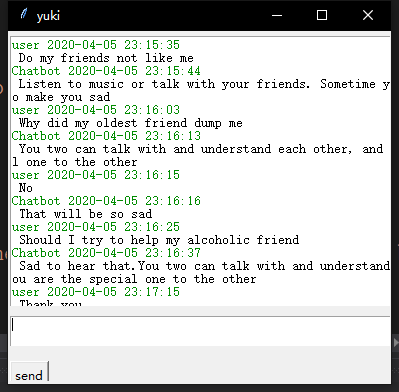
**5 points**: Include sample output in your project report. Have one dialogue (at least 30 turns) that show a good or feasible conversation -- ensure your new features are demonstrated! Document a list of limitations of your program, and have at least two short dialogues that show when your agent is not able to handle the conversation properly.

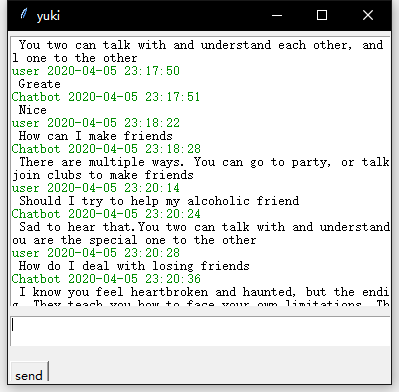
**Good conversation:**

****

****

****

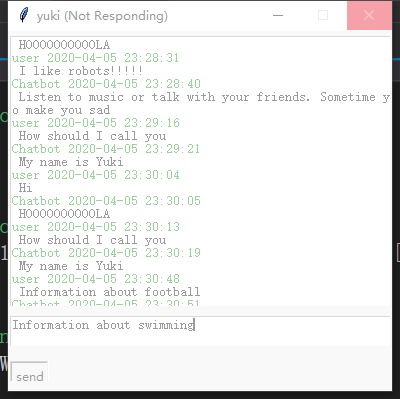
****

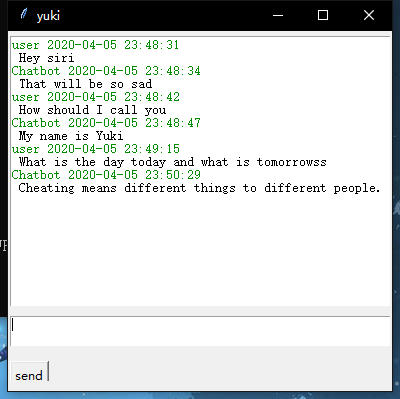
****

**Limitations:**

1. Even though user can see the history when they are using the Chatbot, however, when the software is closed, the history will not be documented.
2. The reply time needed for the Chatbot is pretty long, which may cause a negative impact on the usage.
3. The initial time needed for the Chatbot is also long, which spends user too much time for using.

**Short conversations:**

**1: **

**2: **

**5 points**: Based on your system, include a list of at least 5 features that you can extract from your code or design that can be shared with others as an API.

1. Reasonable responses function
2. Spelling error handling function
3. Coreference Resolution function
4. Sentiment analysis function
5. Beautiful GUI