

UNIVERSITY OF NEW SOUTH WALES

School of Computer Science and Engineering



UNSW

**X-o-Bot: A Generic Dialog Framework for
Conversational Agents**

Project Report

Computer Science Project COMP9900

Team: **TIGER**

Scrum Master/Developer:

Jiahui Wang (z5171973)

Developers:

Haichuan Wang(z5144159)

Taiyan Zhu(z5089986)

Xingchen Guo(z5109353)

Table of Content

TABLE OF CONTENT	- 1 -
INTRODUCTION	- 2 -
EXISTING SYSTEMS:	- 2 -
PROBLEMS OF THE EXISTING SYSTEM AND WHY OUR SYSTEM IS BETTER:	- 2 -
OUR SYSTEMS:	- 2 -
STRUCTURE OF THE REPORT:	- 3 -
BACKGROUND	- 4 -
EXAMPLE: USAGE SCENARIO	- 4 -
SYSTEM ARCHITECTURE	- 5 -
CORE COMPONENT 1: DATA OBTAINING AND CLEANING	- 6 -
BEAUTIFUL SOUP	- 6 -
DATA CRAWLING STEP BY STEP	- 6 -
ISSUES	- 7 -
CORE COMPONENT 2: DATA HANDLING AND PROCESSING	- 8 -
CORE COMPONENT 3: MACHINE LEARNING MODEL	- 10 -
CORE COMPONENT 4: DIALOGFLOW	- 12 -
CORE COMPONENT 5: UI	- 14 -
CONCLUSION	- 15 -
REFERENCE	- 16 -
APPENDIX A: SOURCE CODE EXPLANATION FOR COMPONENT 1	- 17 -
APPENDIX B: SOURCE CODE EXPLANATION FOR COMPONENT 2	- 19 -
APPENDIX C: SOURCE CODE EXPLANATION FOR COMPONENT 5	- 21 -
APPENDIX D: SOURCE CODE STRUCTURE	- 23 -
APPENDIX E: USED TECHNIQUES	- 26 -
APPENDIX F: INSTALLATION GUIDE	- 27 -
<i>Environment</i>	- 27 -
<i>Database</i>	- 27 -
<i>Run Program</i>	- 27 -

Introduction

Chatbots have become a vital part of nearly every business – whether we're talking about providing 24/7 support or increasing human productivity, chatbots are a valid reason to be added as part of customer service management. In addition, branding is improved with less customer effort. Similarly, chatbots can be used as a powerful tool for educational purpose. For example, answering simple questions which students do not have to wait for the lecturer in charge to answer.

Existing Systems:

One of the existing systems is called Jill Watson. Jill Watson is a graduate teaching assistant that use in Georgia Tech course on artificial intelligence: Jill answers routine, frequently asked questions on the class discussion forum (Ashok Goel, 14-11-2017). The teaching assistant use mirror forum, as the student post the questions on the actual forum After that, Jill will post the answer on the mirror forum, then a human teaching assistance will assess the answer using the previous data to confirm if the answer is right or not. When the confidence is greater than 97%, then Jill will remove the mirror forum and directly post on the real forum alone.

Problems of the Existing System and Why Our System Is Better:

There are various advantages on Jill Watson, first the teaching assistant can immediately answer the question regardless the time and space. This will increase the efficiency of the students by minimize the time of waiting. On the other hand, the teaching assistance can reduce the workload of human teaching since the whole system is process without human intervention. However, there still contain some disadvantages for this teaching assistance, the assistance can only answer the question for relative courses and topics, the system immediately eliminates the questions outside of its limit by not replying the questions. Moreover, the system is still a prototype and is only available for Georgia Tech course (Ashok Goel, 1-11-2016). In this report, our chatbot is also teaching assistance use as the same purpose as Jill Watson. However, there are several differences between two teaching assistances. Our chatbot is completer and more stable chatbot compare to Jill Watson, the chatbot contain all the advantages from Jill Watson and already release online. In addition, the self-learning technique of our chatbot also provide the advantage to answer some question outside the courses or topics. Hence, by comparing both teaching assistances, our chatbot is more prefer than the Jill Watson

Our systems:

The whole project consists of several components. One of the core components is machine learning module. It matches the key words from the users' input with the key words of the questions which are stored in the cloud storage. Then return the top three questions with the highest similarities. Before training the machine learning model, it is important to obtain enough data sets to train the model. Another core component is data obtaining and cleaning. There are several ways of obtaining data, in our project, all the data sets were obtained by using beautiful soup from websites including Kaggle and Stack Overflow. There is also a graphical

user interface which allows users communicate with our chatbot and also try to make it more user-friendly.

The aim of this project is to design and implement a “smart enough” chatbot with ability of learning from new questions for the course Comp9021 Principles of Programming. The chatbot will answer questions regarding course content, lecture location, information about the lecturer in charge and many other matters for a particular course.

Regarding the scope, the project is limited to one particular course, Comp9021 Principles of Programming. Mainly because users might get confused if too many answers are replied by the chatbot, so we want to limit the answer to be only one at a time. The aim of the chatbot is to help the user obtain a clear answer, not to make them confused. Our project uses Alibaba cloud as the requested cloud-based architecture, so our implemented chatbot can be visited at <http://47.254.85.250>. We also use it as our cloud storage.

Structure of The Report:

Please see the [Table of Content](#)

Background

Example: Usage Scenario

The users of our system are the student who enrolled in Comp9021. A student who is taking the course can use the chatbot to obtain information such as “How many assignments of this course? ”, “The contact details of the Lecturer in charge” and “Location of the lecture room (integrated with google map)”

Our system process users’input by using both Dialogflow and our trained machine learning model (details of this part will be described in [Core Component 3](#)). The Figure 1 shows that the links for course information such as class time, class location, lab time and exam info can be easily found from the top of the UI.

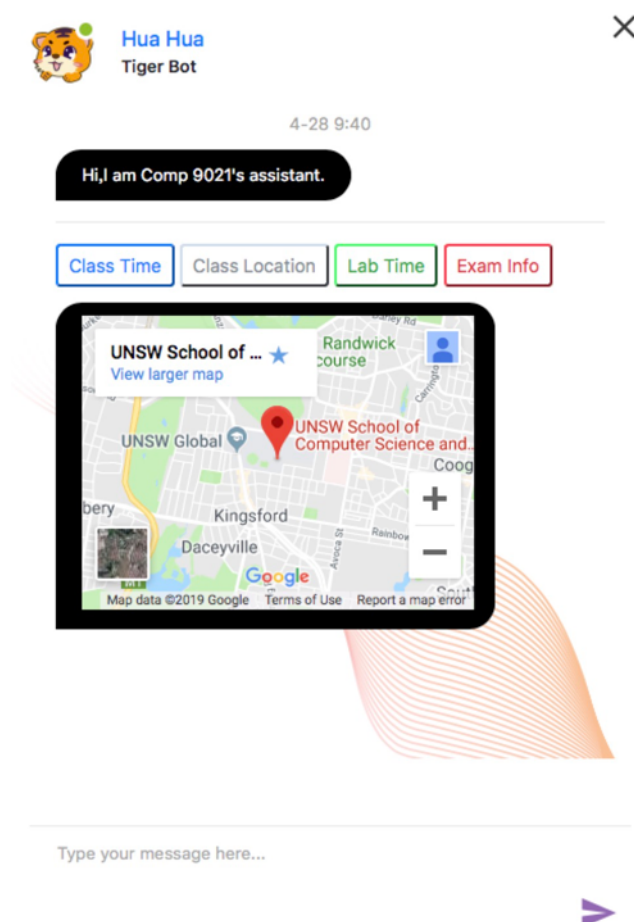


Figure 1 A screen shot of the UI returning the location of the lecture room

System Architecture

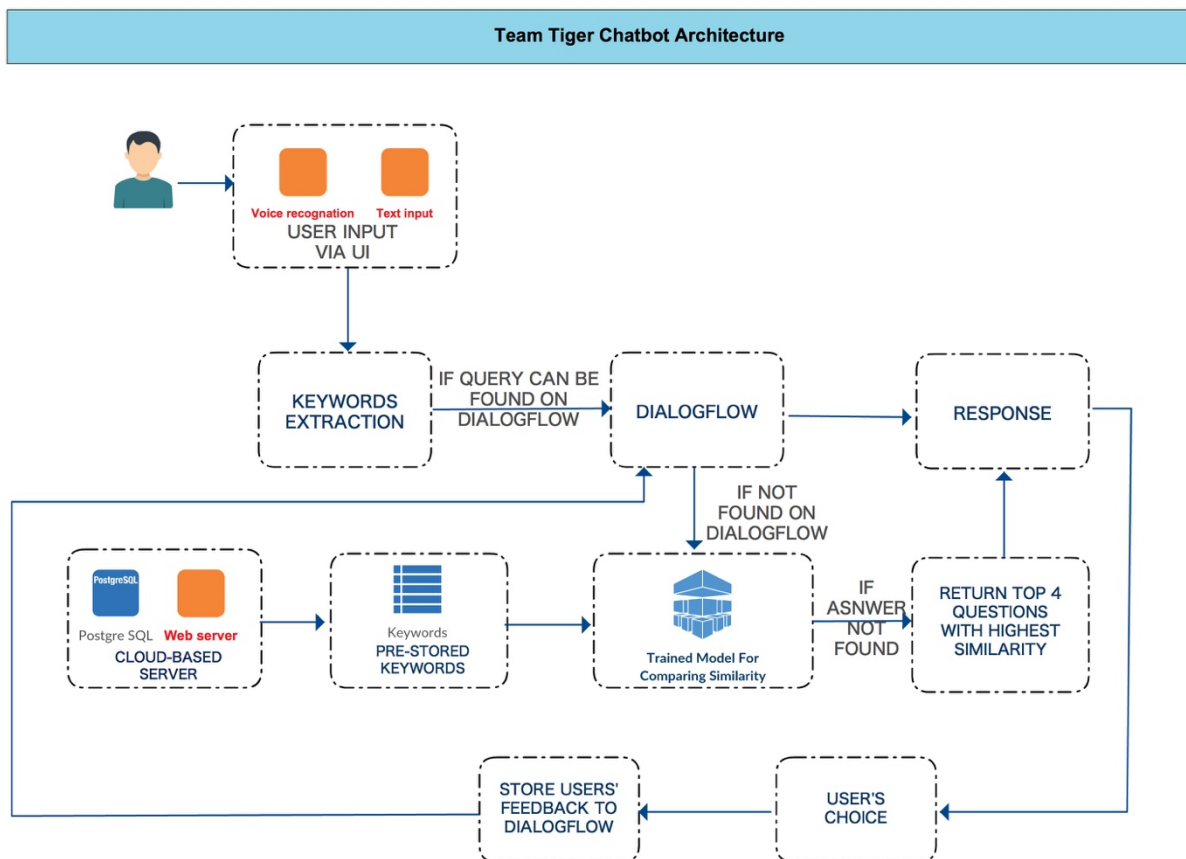


Figure 2 Architecture of the Chatbot

Apart from data obtaining and cleaning (details will be described in [Core Component 1](#)), there are four major components in our systems. These four components are UI(details will be described in [Core Component 5](#)), Dialogflow (details will be described in [Core Component 4](#)), Trained model for comparing similarity (details will be described in [Core Component 3](#)) and our cloud-based server.

When the bot receives an input from a user, it first extracts the keywords and tries to match the keywords to the keywords of the questions on the Dialog flow. If a question is matched, then simply return the answer related to the question. However, if there are no questions can be matched with the users' input, then the bot would try to match the pre-stored keywords which are stored on our cloud-based server. If an answer cannot be found, then return three options with the highest similarity. When one of the options is selected by the user, the bot stores the question and the choice on Dialogflow.

Core Component 1: Data Obtaining and Cleaning

Data is vital for training the machine learning model. For this purpose, data for both model training and question and answer bank need to be cleaned before hand. In order to make our system more powerful and accurate, the data should be obtained and saved into a CSV file with two columns for questions and answers. Since most of the data forms internet, programming techniques were applied for data crawling.

Since this course is about data structure & algorithm in Python, we were targeting the websites with Python interview questions as well as Stack Overflow for crawling the data.

Beautiful Soup

The naivest way of getting data is just “copy & paste”, but this way is time consuming and people who doing it can get tired very quickly. For our project, Beautiful Soup was used for obtaining data.

Beautiful Soup [2] is a Python library to extract data from HTML and XML files. It works with parser to provide idiomatic ways of navigating, searching, and modifying the parse tree. It commonly saves programmers hours or days of work.

The input of this part is URLs of the webpages, and the output will be two CSV files which are “questions.csv” and “answers.csv”, then finally we combine these two files into one CSV file. The explanation of the source code will be in Appendix A. The source code for this part will not be included in the final source code submission, because the code will not be used for deployment of the project.

Data Crawling Step by Step

1. The first step is to find the “tag” of the wanted data, in this case, questions. We will use this link: “<https://www.sanfoundry.com/python-questions-answers-variable-names/>” as an example. Using Chrome to open the website, then select a questions and then right click select “inspect”. From the Diagram 3 we can see that the tag for all questions is “q”.
2. Then apply regular expression for removing the unwanted data. Because there are many useless data also uses the tag. In this case, “`^(\d+.+ ?\d+.+ .)`”. It means only obtain the data start with a digit + “.” + “space”, and end with a “.” or “?”. For example, “1. Is Python case sensitive when dealing with identifiers?”, our regular matches this questions because it starts with a digit + a dot + a space, and ending with a question mark. Because some of the explanation question has a full stop at the end, so I added the full stop for this case.

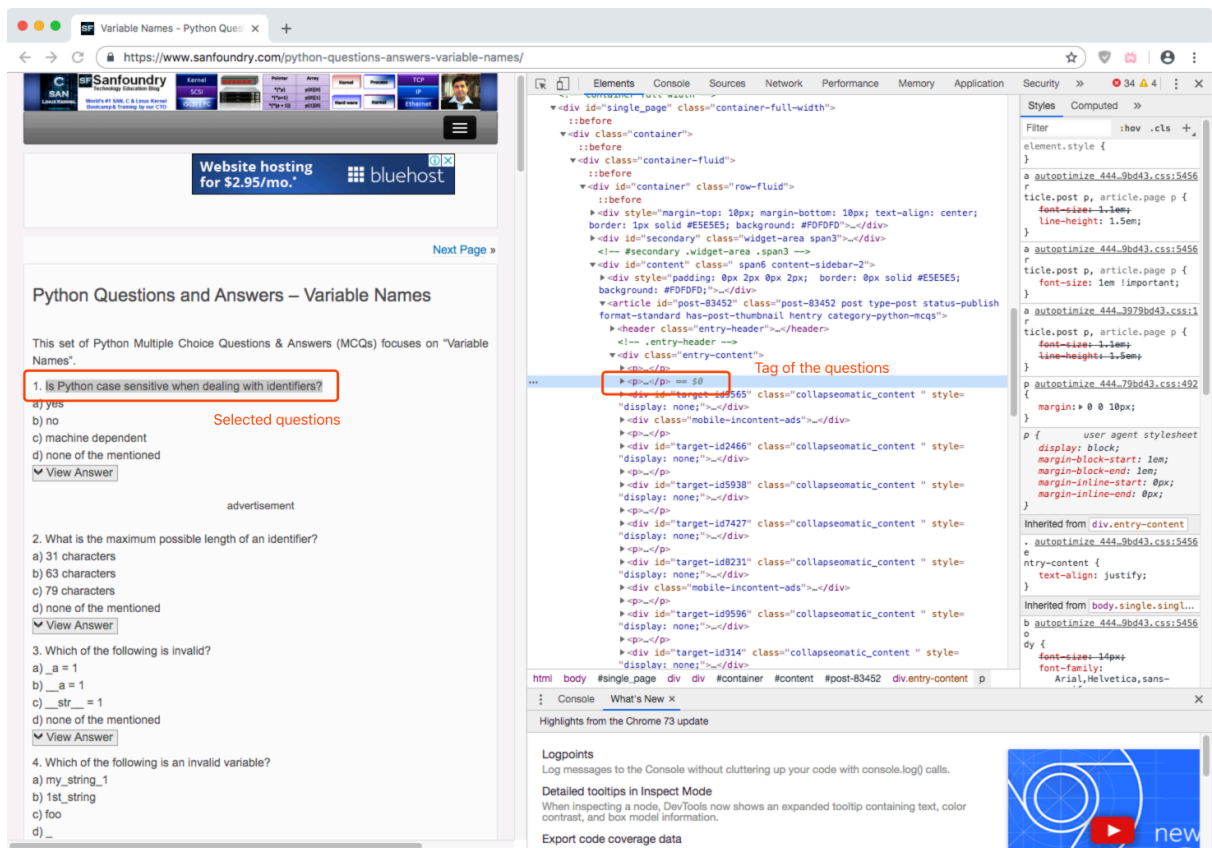


Figure 3 Tag for questions

3. Then we use a list to store all the questions we obtained from the website. Apart from regular expression, we also used “[3:]” to remove the digit number + a dot + a space at the beginning of the question. Then write them into a CSV file called “questions.csv”.
4. Likewise, we do the same to the answers, then we have file called “answers.csv”.
5. Finally, we combine these two csv file into one. Since it only needs to copy one column, we can only use “copy & paste”.

Issues

How to tackle these issues will be explained in conclusion.

The first issue for this part was the website has anti-crawling system. The solution for fixing this issue is to provide a user-agent header: “headers = {'User-Agent': 'Mozilla/5.0'}” for faking a browser visit.

Core Component 2: Data Handling and Processing

As our original data source is fetched from the website Stack Overflow, including 3 main tables, one question table containing around 600,000 questions, one reply table containing more than 1 million replies recode as well as a label table. Each question in question table is unique, and each reply has a unique id, linked via question's id to each question in question table. Moreover, each question has up to 4 labels, like classification, stored in the label table, linked via question's id.

In addition, the main body of questions and replies is the HTML form. Therefore, the first job is removing the HTML tags and extracting the main content of each question and reply via using function BeautifulSoup from library bs4. Such clean data is updated to the database in 4 tables respectively.

For the database, we use the SQLite combining the library Sqlalchemy in our code.

There are 7 columns in Question Table (Figure 4), the column ID is a autoincrement column as primary key, the column QUESTION_ID is unique for questions, the column TITLE stores the title for each Q&A post, the column QUESTIONS is the detail for the post, the column KEYWORDS stores the keywords extracted from the title, which are words the meaningless words have been removed as well as the stem extraction of NLTK library has been used. (the pos_tag tech of NLTK library has been also used in keywords extraction, such verbs and nouns are remained in the column nltkwords. However, the words remained are too general to improve the accuracy of similarity.)



Question	
id	INTEGER
question_id	INTEGER
title	TEXT
questions	TEXT
update_time	DATETIME
keywords	TEXT
nltkwords	TEXT

Figure 4 The Question Table

There are 6 columns in Answer Table (Figure 5), the column ID is a autoincrement column as primary key, the column ANSWER_ID is unique for questions, the column QUESTION_ID is the foreigner key, linking the question table, each answer has only one question id, match the question in question table, the column answers are the content of reply. the score is the evaluation of this reply.

Answer		
	id	INTEGER
	answer_id	INTEGER
	question_id	INTEGER
	answers	TEXT
	score	INTEGER
	update_time	DATETIME

Figure 5 The Answer Table

There are 3 columns in Label Table (Figure 6),

The column ID is a autoincrement column as primary key, also the unique id of the label.

The label is classification.




Label		
	id	INTEGER
	label	TEXT
	update_time	DATETIME

Figure 6 The Label Table

There are 4 columns in Label Table (Figure 7), The column ID is a auto increment column as primary key. The column QUESTION_ID is the foreigner key, linking the Question Table. The column LABEL_ID is the foreigner key, linking the Label Table. Each question can be classified up to 4 classes.





Tag		
	id	INTEGER
	question_id	INTEGER
	label_id	INTEGER
	update_time	DATETIME

Figure 7 The Tag Table

There are about 200,000 specify questions are chosen to train the model according their labels which is more relative to the COMP9021 course. These records are updated to a new table (Figure 8), which can reduce the training time.




CQuestion		
	id	INTEGER
	question_id	INTEGER
	label	TEXT

Figure 8 New table for reducing training time

The DAL(data access layer) provides simplified access to the database in persistent storage.

To combine with the SQLAlchemy, database table classes have been defined for each table in the database(will be described in [Appendix B](#)). Also, some commands such as insert, delete and update to a single table will be packaged in a specified class (will be described in [Appendix B](#)).

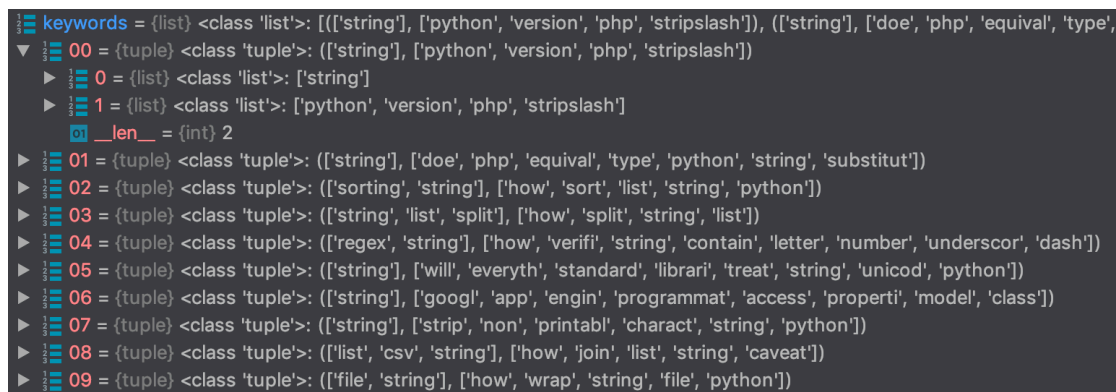
Core Component 3: Machine Learning Model

The source code for this part is huge, please see the comment in the submitted source code.

In the machine learning part, a words similarity model has been trained by employing the Gensim Doc2vec Model.

The training input is a batch of paragraph or sentence, each sentence or paragraph will be regarded as a document indexed in the model. In our project, the training dataset will be fetched from the database, which already is preprocessed and stored as keywords in Question Table.

The data, fetched from the database, will be formed into a three-dimensional list. (Figure 9)



```
keywords = (list) <class 'list'>: ([('string'), ('python', 'version', 'php', 'stripslash')], ([('string'), ('doe', 'php', 'equival', 'type', 'python', 'string', 'substitut')], ([('sorting', 'string'), ('how', 'sort', 'list', 'string', 'python')], ([('string', 'list', 'split'), ('how', 'split', 'string', 'list')], ([('regex', 'string'), ('how', 'verifi', 'string', 'contain', 'letter', 'number', 'underscor', 'dash')], ([('string', 'will', 'everyth', 'standard', 'librari', 'treat', 'string', 'unicod', 'python')], ([('string', 'googl', 'app', 'engin', 'programmat', 'access', 'properti', 'model', 'class')], ([('string', 'strip', 'non', 'printabl', 'charact', 'string', 'python')], ([('list', 'csv', 'string'), ('how', 'join', 'list', 'string', 'caveat')], ([('file', 'string'), ('how', 'wrap', 'string', 'file', 'python')])
00 = (tuple) <class 'tuple'>: ([('string'), ('python', 'version', 'php', 'stripslash')])
0 = (list) <class 'list'>: ['string']
1 = (list) <class 'list'>: ['python', 'version', 'php', 'stripslash']
len_ = (int) 2
01 = (tuple) <class 'tuple'>: ([('string'), ('doe', 'php', 'equival', 'type', 'python', 'string', 'substitut')])
02 = (tuple) <class 'tuple'>: ([('sorting', 'string'), ('how', 'sort', 'list', 'string', 'python')])
03 = (tuple) <class 'tuple'>: ([('string', 'list', 'split'), ('how', 'split', 'string', 'list')])
04 = (tuple) <class 'tuple'>: ([('regex', 'string'), ('how', 'verifi', 'string', 'contain', 'letter', 'number', 'underscor', 'dash')])
05 = (tuple) <class 'tuple'>: ([('string', 'will', 'everyth', 'standard', 'librari', 'treat', 'string', 'unicod', 'python')])
06 = (tuple) <class 'tuple'>: ([('string', 'googl', 'app', 'engin', 'programmat', 'access', 'properti', 'model', 'class')])
07 = (tuple) <class 'tuple'>: ([('string', 'strip', 'non', 'printabl', 'charact', 'string', 'python')])
08 = (tuple) <class 'tuple'>: ([('list', 'csv', 'string'), ('how', 'join', 'list', 'string', 'caveat')])
09 = (tuple) <class 'tuple'>: ([('file', 'string'), ('how', 'wrap', 'string', 'file', 'python')])
```

Figure 9 Data in three-dimensional list

An element in the first-dimensional list is a record fetched from the database, each element is a tuple containing a list of labels, which are the classification of the question, and a list of keywords of the question.

This dataset will be formed into a Tagged Document Object, which is the "Represents a document along with a tag, input document format for Doc2Vec"[5], passed to Doc2Vec Model.

Before starting the training, some Hyper-parameter should be set.



```
model = doc2vec.Doc2Vec(sents, size=300, window=12, min_count=5, workers=4, dm=0)
```

Figure 10 Parameters for training

dm ({1,0}, optional) – Defines the training algorithm. If dm=1, ‘distributed memory’ (PV-DM) is used. Otherwise, a distributed bag of words (PV-DBOW) is employed. [5]

In our case, we adopt the DBOW mode, which is a transformation from the CBOW model, as this model can provide a more accurate similarity result compared to DM model.

vector_size (int, optional) – Dimensionality of the feature vectors. [5]

We set it as 300, hopefully, the 300 dimensions can outcome the more accurate vectors.

min_count (int, optional) – Ignores all words with a total frequency lower than this. [5]

Here we set it the default value.

window (int, optional) – The maximum distance between the current and predicted word within a sentence. [5]

The Business logic layer coordinates the application, processes commands make logical decisions and evaluations, and performs calculations.

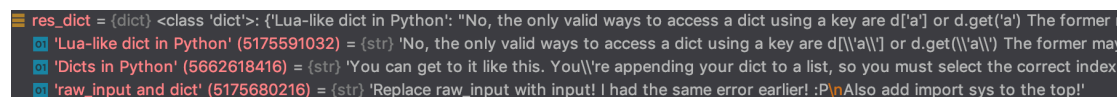
There are two business logics in this layer, one is training model (described in component "Machine Learning"), the other one is replying generation.

The input of the reply generation system is a sentence (string type), passed from the UI layer. Through the keywords selection, certain keywords will be extracted and split into a list, passed to similarity system.

In similarity system, models (trained on machine learning part) will be loaded at the beginning of running the program. Once the list of keywords passed to the system, it will be transformed into a 300-dimension vector. Such a vector will be used to match the question from the data source and general the similarity of the words.

Like the description above, the index of question from the data source has been stored in the model. Through the matching, the index of the top 10 most similar question will be returned with its similarity.

Moreover, the average of multiple model outputs will be used to evaluate the similarity comprehensively. The output index will be used to query the question and its answer from the database. The output of the similarity is a dictionary whose keys are the title of question returned from the query, as well as the corresponding value is the answer queried from Answer Table. (Figure 11)



```
res_dict = {dict: <class 'dict'>: {'Lua-like dict in Python': "No, the only valid ways to access a dict using a key are d['a'] or d.get('a') The former may raise a KeyError if the key is not present, while the latter will return None. The latter may be more convenient if you are not sure if the key is present.", 'Dicts in Python': "You can get to it like this. You\\re appending your dict to a list, so you must select the correct index", 'raw_input and dict': "Replace raw_input with input! I had the same error earlier! :P\\nAlso add import sys to the top!"}}
```

Figure 11 The result dictionary

Finally, the dictionary will be transformed into JSON form in reply system, returning to the UI layer.

Core Component 4: Dialogflow

The Dialogflow was used for two purposes. The first one is to handle questions related to course outline such as “Who is the lecturer in charge?”. The other one is to help with our feedback loop functionality. We have to pre-store these questions before hand.

The above code is obtained from official Google Documentation.

The first step is to make a connection with the Dialogflow. The necessary parameters such as Project_ID, Session_ID, Credentials and Language_Code.

```
DIALOGFLOW_PROJECT_ID = 'tigerbot-943ba'
DIALOGFLOW_LANGUAGE_CODE = 'en-US'
GOOGLE_APPLICATION_CREDENTIALS = 'tigerbot-943ba-e785ca33e7a0.json'
SESSION_ID = 'tigerbot'
PROJECT_ID = 'tigerbot-943ba'
credentials = service_account.Credentials.from_service_account_file(GOOGLE_APPLICATION_CREDENTIALS)
```

Figure 12 The necessary parameters for Dialogflow

There are several functions in the Python file. These two functions are used to analyze users' input and provide a response:

```
def get_analyzed_text_response(text_to_be_analyzed):
    text_input = dialogflow.types.TextInput(text=text_to_be_analyzed, language_code=DIALOGFLOW_LANGUAGE_CODE)
    query_input = dialogflow.types.QueryInput(text=text_input)
    response = session_client.detect_intent(session=session, query_input=query_input)
    return response

def get_response(response):
    response_intent = response.query_result.intent.display_name
    response_text = response.query_result.fulfillment_text
    return response_intent, response_text
```

Figure 13 The two main functions for general conversation

When an input which is a string is received from a user, the function starts to analyze the input string. Then the Dialogflow convert test_input into query_input and finally provides a response. The function get_response will get the response from the get_analyzed_text_response. This function will try to match the users' input with pre-stored questions and provide an answer for the user as the output.

Another function in this Python file is:

```
def create_intent(display_name, training_phrases_parts, message_texts):
    """Create an intent of the given intent type."""

    training_phrases = []

    for training_phrases_part in training_phrases_parts:
        training_phrases_part = ' '.join(filterpy(training_phrases_part.split(' ')))
        part = dialogflow.types.Intent.TrainingPhrase.Part(
            text=training_phrases_part)
        # Here we create a new training phrase for each provided part.
        training_phrase = dialogflow.types.Intent.TrainingPhrase(parts=[part])
        training_phrases.append(training_phrase)

    text = dialogflow.types.Intent.Message.Text(text=message_texts)
    message = dialogflow.types.Intent.Message(text=text)

    intent = dialogflow.types.Intent(
        display_name=display_name,
        training_phrases=training_phrases,
        messages=[message])

    response = intents_client.create_intent(parent, intent)

    print('Intent created: {}'.format(response))
```

Figure 14 Function for adding users' feedback answer & question

This function is mainly for feedback loop. When a user selects an answer that the user believes is the right answer for the question, the Dialogflow will create the question as a new intent and store the selected answer as the answer of this question.

Core Component 5: UI

In the UI part, Bootstrap was used as Front-end Framework of the project, which contain HTML, CSS, and JavaScript framework. Bootstrap is a source that is free and able to access by general public, mobile-first front-end web development. It contains CSS- and (optionally) JavaScript-based design templates for typography, forms, buttons, navigation and other interface components (Mark Otto, 2018). Moreover, jQuery backstretch, material_icon and perfect-scrollbar was used as the plugins to increase the vision of the page.

The training input is a batch of paragraph/sentence, each sentence/paragraph will first generate to a JSON type. Since the postJSON is the first output therefore, postJSON will send to backend after each sentence/paragraph generated to a JSON type. The source code will be explained in [Appendix C](#).

The displayed output in html page is pass from the back-end. One of the outputs is from the dialogflow, whereas one output is from our training model. The display function will demonstrate the feedback on user interface.

If the users do not get the answer from the following questions, the chatbot will learn from the back-end and obtain the 3 answers that has the highest similarity to let the users choose. If the users accept the answer, the new questions and answers will pass through back-end from the hyper link. It will help the system to develop and answer more questions in the future.

Conclusion

Chatbots for education can provide a welcoming interface between students and a course website. It could provide all useful information such as academic, admission, course content and timetable to their enrolled students 24/7. In our project, we have successfully implemented a chatbot for comp9021 with functionality with feedback looping. We encountered several issues which has been explained in each core component.

There are several improvements can be made if we had longer time to complete. Firstly, we did not have enough time to finish the voice input. We were on the right track. We were planning to use Google speech to test for voice input.

Secondly the university can hire a few number of the post-graduated students that contain enough background knowledge to do research and give recommendation base on the chatbot. however, this will cost the university a lot of money to hiring and interview the student.

In addition, the chatbot can gathering the feedback from the users, and improve the accuracy of the answer. However, it requires large amount of data from the users to obtain high accurate answer.

Reference

1. Zakharov, K., Mitrovic, A., Ohlsson, S. (2005). Feedback Micro-engineering in EER-Tutor. . Artificial Intelligence in Education AIED 2005, IOS Press. G. M. C-K Looi, B. Bredeweg, J. Breuker (eds) Proc. . **pp. 718-725**.
2. Definition of Jill Watson, Jill Watson Doesn't Care if You're Pregnant: Grounding AI Ethics in Empirical Studies-Bobbie Eicher, Lalith Polepeddi, Ashok Goel-
http://www.aies-conference.com/wpcontent/papers/main/AIES_2018_paper_104.pdf -
page 1-posted on 14-11-2017
3. Information for Jill Watson, A teaching assistant named Jill Watson- Ashok Goel-
<https://www.youtube.com/watch?v=WbCguICyfTA> – posted on 1-11-2016
4. <https://www.crummy.com/software/BeautifulSoup/bs4/doc/>
5. <https://radimrehurek.com/gensim/models/doc2vec.html>
6. <https://www.tornadoweb.org/en/stable/>

Appendix A: Source Code Explanation for Component 1

<pre>from bs4 import BeautifulSoup import requests import re import csv</pre>	<p>The first few lines are for importing the necessary packages.</p> <p>bs4 is the BeautifulSoup library.</p> <p>Requests is the HTTP library. we use it for reading the content of the server's response.</p> <p>Re is the library for regular expression</p> <p>Csv is the library for handling CSV files.</p>
<pre>def criterion(tag): return (tag.name == 'p' and re.search('^(\\d+\\.+ ? \\d+\\.+ .)', tag.text))</pre>	<p>Create a function with regular expression and the tag for a certain website. Tag and regular expression can be modified based on different website.</p>
<pre>url = ["https://www.sanfoundry.com/python- questions-answers-variable-names/"]</pre>	<p>I used a list to contain URLs, so I can add more URLs and crawling them at the same page, if they can be applied with the same tag and regular expression.</p>
<pre>headers = {'User-Agent':'Mozilla/5.0'}</pre>	<p>This line is for dealing with anti-crawler.</p>
<pre>for i in range(len(url)): page = requests.get(url[i]) soup = BeautifulSoup(page.text, "lxml") pars = soup.find_all(criterion)</pre>	<p>I use a for loop to visit all URLs in the url list. Then use requests.get to connect the target server, and use BeautifulSoup with lxml parser to read the responded web content. Then apply the previously created function to crawl.</p>
<pre>L = [] for m in range(len(pars)): questions = pars[m].get_text()[3:] L.append(questions.lstrip()) with open("questions.csv", 'a') as g: writer = csv.writer(g) writer.writerow([item] for item in L))</pre>	<p>A list is created to store all the questions. For all the questions stored in pars, we remove the first three characters and also remove all the spaces. Then store them into the list L. Final, write all the questions into a CSV file.</p>

Regarding the answers part, the only difference is we use a different function to get the data:

```
p = soup.find_all('div', {"class": "collapseomatic_content"})
```

```
crawler.py - /Users/jack/Desktop/Comp9900/crawler.py (3.6.2)
# import the necessary packages
from bs4 import BeautifulSoup
import requests
import re
import csv

# create a fonction with regular expression for a certain website
def criterion(tag):
    return (tag.name == 'p' and re.search('^(\\d+.+ ?|\\d+.+ .)', tag.text))

##"https://www.sanfoundry.com/python-questions-answers-precedence-associativity-
##      "https://www.sanfoundry.com/python-questions-answers-variable-names/",
##      "https://www.sanfoundry.com/python-mcqs-core-datatypes/",
##      "https://www.sanfoundry.com/python-questions-answers-numeric-types/",
##      "https://www.sanfoundry.com/python-questions-answers-precedence-associativity-
url = [ "https://www.sanfoundry.com/python-questions-answers-variable-names/"]

# this line is for dealing with anti-crawler
headers = {'User-Agent': 'Mozilla/5.0'}
for i in range(len(url)):
    page = requests.get(url[i])
    soup = BeautifulSoup(page.text, "lxml")
    L = []
    pars = soup.find_all(criterion)

    # this part is for the questions
    for m in range(len(pars)):
        questions = pars[m].get_text()[3:]
        L.append(questions.lstrip())
    with open("questions.csv", 'a') as g:
        writer = csv.writer(g)
        writer.writerows([[item] for item in L])

    # this part is for the answers
    p = soup.find_all('div', {"class": "collapseomatic_content"})
    K = []
    for m in p:
        answers = m.get_text()
        K.append(answers.lstrip())
    with open("answers.csv", 'a') as o:
        writer = csv.writer(o)
        writer.writerows([[item] for item in K])
```

Ln: 43 Col: 0

Appendix B: Source Code Explanation for Component 2

<pre> from flask_sqlalchemy import SQLAlchemy app = Flask(__name__) app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///collection.sqlite' #setting the database path base = SQLAlchemy(app) </pre>	<p>Import SQLAlchemy</p> <p>Setting the path of database</p>
<pre> class Question(base.Model): __tablename__ = 'Question' id = Column(Integer, primary_key=True) question_id = Column(Integer, unique=True, nullable=False) title = Column(Text, nullable=False) questions = Column(Text, nullable=True) update_time = Column(DateTime, nullable=False) keywords = Column(Text, nullable=True) nltkwords = Column(Text, nullable=True) </pre>	<p>A databases table class, inherit the class SQLAlchemy. Attributes declared as object of column in the class are the existing columns of the table in database.</p>
<pre> class DBQue(): def __init__(self): self.que = Question() def get_keywords_all(self, count=0): ''' Connect to the Question Table, return query result :param count: the number of rows will be returned. :return:a list of object of Question class, containing only value of keyword column, which is the result of query </pre>	<p>Internal functions to connect the database are provided by class SQLAlchemy, for example:</p> <p>"Question.query.with_entities(Question.keywords).all()", calls the internal function query, which is inherited by class Question, to access the question table from database.</p> <p>Such functions to access the same data table are packaged in a class, for example the get_keywords_all() and update_keywords_all() are the functions access the same table question.</p>

```

'''
if count:
data_lines =
Question.query.with_entities(Question.keywords).l
imit(count).all()

return data_lines

else:
return
Question.query.with_entities(Question.keywords).
all()


def update_keywords_all(self):
'''
update keyword columns
:return:
'''

data_lines = self.query_all()
title_list = [i.title for i in data_lines]

data_words = db.build_dictionary(lines=title_list,
freq=0)
max = len(data_words)
for i, j in enumerate(data_lines):
print(i)
j.keywords = ','.join(data_words[i])
base.session.commit()

```

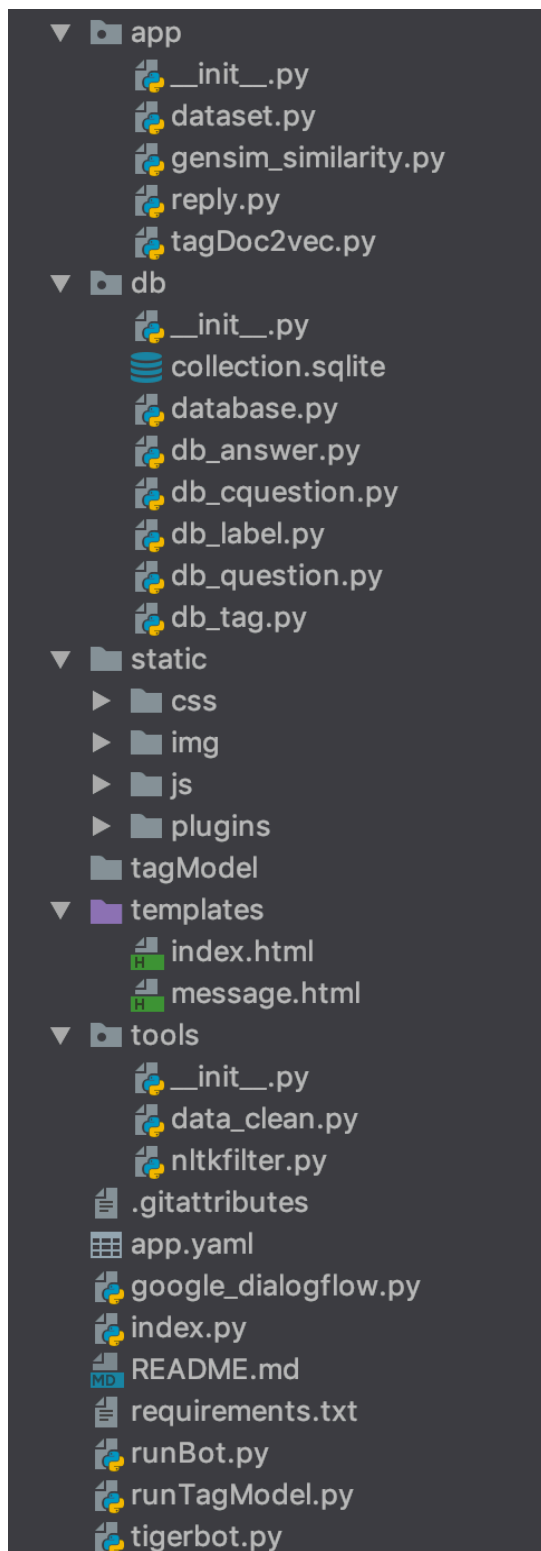
Appendix C: Source Code Explanation for Component 5

<pre>function handle_message() { var content = \$('#message-input').val(); if (content == "") { return false; } \$('.chat-messages').append('<div class="message self"><div class="message-content">' + content + '</div></div>'); \$('#message-input').val(""); message = formatToMessage(content); }</pre>	<p>This function is used as handling user input.</p> <ol style="list-style-type: none"> 1. get message from message.html 2. display the user input on conversation interface 3. clear message box 4. use this function change the input to a JSON type <pre>function formatToMessage()</pre>
<pre>jQuery.postJSON = function (url, args, callback) { \$.ajax({ url: url, data: \$.param(args), dataType: "text", type: "POST", success: function (response) { if (callback) callback(eval("(" + response + ")")); } }); }</pre>	<p>@param: url @param: args @param: callback Wrapped AJAX method by jQuery, send args data to url in HTTP POST method.</p>
<pre>\$.postJSON("/message/new", message, function (response) { if (response.show_hints == "yes") { showHints(response.reply_other, content); } else { printBotMsg(response.reply_msg); } });</pre>	<p>This function is for display reply message. If the back end cannot provide one unique answer. It will send a dictionary which contains 3 questions and answers. Then those will be handled by</p> <pre>function showHints(hints, userQuestion)</pre>
<pre>function showHints(hints, userQuestion) ...(more details in source code) answers.push(answer); questions.push(question); question_ids.push(hint_id); \$.postJSON(apiUrl, dict, function (response) { printBotMsg(response.msg) });</pre>	<p>This function is to implement feedback loop.</p> <ol style="list-style-type: none"> 1. I put questions ID, questions and answers into 3 arrays 2. use method to give user choices and get feedback. 3. send feedback to backend and then get reply.
<pre>\$(function () { \$('#message-input').on('keypress', function (e) { if (e.which == 13) { handle_message(); } }); });</pre>	<p>The user can click the 'send' or just use 'enter' to send message.</p>
<pre>\$(function () { \$('#pvr_chat_wrapper').toggleClass('active'); \$('#pvr_chat_button, .pvr_chat_wrapper .close_chat') .on('click', function () {</pre>	<p>Use 'toggleClass()' to achieve show message window or hide the window</p>

```
$('.pvr_chat_wrapper').toggleClass('active');  
return false;  
});  
$('#class_time').on('click', function ()  
$('#class_loc').on('click', function ()  
$('#lab_time').on('click', function ()  
$('#exam_info').on('click', function ()
```

Four buttons: basic
questions and answers for
the course

Appendix D: Source Code Structure



./app	A python package, containing all the classes of business logic
./app/__init__.py	
./app/dataset.py	Packaging some functions, calling the data tables connection function in data access layer, as well as some data process functions.
./app/gensim_similarity.py	Receiving a sentence(string) as input, through the similarity system, a dictionary with Q&A as key-value.
./app/reply.py	This is designed as router, which can call different similarity system. Once the user input passed to the back-end, function reply() will be called, as well as the tokenization will be processed. After that, a list of key words will be passed to similarity system.
./app/tagDoc2vec.py	To process the model training.
./db	A python package, containing all the classes of data access logic
./db/__init__.py	
./db/collection.sqlite	Sqlite database file.
./db/database.py	5 data table classes are defined in this file, as well as some data process functions.
./db/db_answer.py	Packaging all the basic operation functions to the Answer table.
./db/db_cquestion.py	Packaging all the basic operation connection functions to the CQuestion table.
./db/db_label.py	Packaging all the basic operation connection functions to the Label table.
./db/db_question.py	Packaging all the basic operation connection functions to the Question table.
./db/db_tag.py	Packaging all the basic operation connection functions to the Tag table.
./static	
./static/css	

./static/img	
./static/js	
./static/plugins	
./tagModel	Saving the trained models
./templates	
./templates/index.html	
./templates/message.html	
./tools	A python package, containing some classes for data processing.
./tools/data_clean.py	Some data cleaning functions, such as stopword remove, punctuation remove.
./tools/nltkfilter.py	A class used to extract the key words by using NLTK pos_tag tech.
Google_dialogflow.py	
Index.py	To run the back-end system. In current program, we combine the back-end and front-end, so that this file will no be used anymore.
runBot.py	To run a chatbot using the terminal, without UI.
Tigerbot.py	To run the project.

Appendix E: Used Techniques

There are several techniques we applied on our project. Regarding the DBMS (database management system), we changed quite a lot of cloud storage. At the beging, we used ElephantSQL, then moved to Google Cloud and finally we decided to use Alibaba Cloud. The reason we choose is it provides 12/7 tech support, fast response times.

About the beck-end, tornado was used. Tornado is a Python web framework and asynchronous networking library, originally developed at FriendFeed [6].

We also use Dialogflow for helping us with the simple question answering and feedback looping. We added a google map API to show the location of the lecture room.

Apart from the front-end, the whole project was created by using Python. The front-end was constructed by using JavaScript and HTML.

Appendix F: Installation Guide

The following guidelines will help you install and run the project on your local machine for development and testing. For instructions on how to deploy the project to an online environment, please refer to the Deployment section. If you have any difficulties of the installation, please send an email to taiyan.zhu@student.unsw.edu.au.

Environment

version control: python 3.7.3

libraries requirement: showed on requirement.txt

run the command to build the project environment: `pip install -r requirement.txt`
as nltk library is applied in the project, some data packages should be download:

```
>>> nltk.download('stopwords')
>>> nltk.download('punkt')
>>> nltk.download('brown')
```

Database

SQLite database is applied in this program. The database file, collection.sqlite is about 1GB under the db folder. The data, collection.sqlite, is too big to submit to the CSE server, so it needs to be downloaded by yourself. Link: <https://github.com/comp3300-comp9900-term-1-2019/capstone-project-tiger/blob/master/db/collection.sqlite>

If this file need to be downloaded separately, please put it in the db folder.

Run Program

For localhost

1. Download the whole project or use clone command:

```
$ git clone https://github.com/comp3300-comp9900-term-1-2019/capstone-project-tiger.git
```

2. Run the runTagModel.py to train the models. It may take more than half hour or longer.
3. Once the models have been trained, the system can be accessed by

running runBot.py, the program will be run in terminal without UI

running tigerbot.py to start the web service, the program can be accessed via url <http://127.0.0.1:8888>

For remote server

Access the website <http://47.254.85.250>

This service is deployed on the server of alibabacloud.

Deployment method are also following the same steps as local host.

The port in tigerbot.py should be change to 80.