Closed-Domain Natural Language Question-Answering System for advising London Restaurant

Author: Xiaofeng Fu

Degree: BSc Computer Science

Supervisor: Prof. Anthony Hunter

Submission Date: 30th April 2018

This report is submitted as part requirement for the BSc Degree in Computer Science at UCL. It is substantially the result of my own work except where explicitly indicated in the text. The report may be freely copied and distributed provided the source is explicitly acknowledged.

Abstract

Nowadays, restaurant recommendation applications have been phenomenally successful. For example, TripAdvisor and Yelp, both are well-known applications in the world. They provide exhaustive restaurants information and users can easily search for a specific restaurant in their applications or websites. However, little has been developed to provide a question-answering (QA) system for restaurant recommendation. Different from reading the discrete restaurant information on the websites, a natural language QA system can provide a reader-friendly textual format of restaurant information that translated from the data on website for users. Yet that QA system involves both natural language understanding (NLU), natural language processing (NLP) and natural language generation (NLG). The development of the natural language system has always been a challenging issue. This project aims to develop a simple natural language QA system that is able to answer questions about London restaurants with well-organized textual format using Python.

A deep research of natural language system and QA system was done to better understand the architecture and the algorithms to implement the system. And a literature review of Natural Language Toolkit (NLTK) for Python was conducted to learn the method to process natural language in Python. Moreover, various meeting with supervisor was organized to sketch the blueprint of the project and decide the core programming language and technologies to be used in this project. Based on the preliminary activities, the QA system for advising London restaurant was designed to process and understand user’s question and then answer the question in reader-friendly textual format. The system was implemented using Python, JavaScript, CSS and HTML with the Django web framework. A testing process and user feedback questionnaire was used to insure the practicability and the stability of the application.

After the implementation of the system, user feedback questionnaires were provided to people that had used the QA system. And the result from the questionnaires has a positive feedback that shows the natural language QA system provides a convenient method for people to search for restaurants information in London.

Contents

1. **Introduction** 
   1. Motivation
   2. Initial Brief
   3. Overview of Approach
   4. Report Structure
2. **Background**
   1. Introduction
   2. Project Context
   3. Programming Language and Frameworks
3. **Requirements and Analysis**
   1. MoSCoW analysis
   2. Use Cases
   3. User Interviews
   4. Discussion
4. **Design and Implementation**
   1. System Overview
   2. System Architecture
   3. Interim System
   4. Database
   5. Natural Language Processing System
   6. Natural Language Generation System
   7. User Interface Design
   8. Challenges and Design Decisions
5. **Testing**
   1. Testing Strategy
   2. Unit Testing
   3. Use Case Testing
6. **User Feedback**
   1. Introduction
   2. Findings and Discussion
7. **Conclusion and Evaluation**
   1. Conclusion
   2. Evaluation
   3. Future work
8. **Reference**
9. **Bibliography**
10. **Appendices**

Appendix A: System Manual

Appendix B: User Manual

Appendix C: Interim Report

Appendix D: Project Plan

Appendix E: User Feedback Questionnaires

Appendix F: Use Cases

Appendix G: User Interviews

Appendix H: Code Listing

Appendix I: Example Video

Appendix J: Regular Expressions

1. Introduction
   1. Motivation

With the development of Internet, people can easily find a specific restaurant information through a search engine or restaurant recommendation applications. According to the study of the Telmetrics-xAd, restaurants are always the top mobile and local search category on search engines based on sizable query log data from different sources (Greg, 2012). Moreover, from the study, the most popular applications that used by mobile users to search for a restaurant are multi-purpose online restaurant review aggregators like TripAdvisor and Yelp (Kevin, 2016).

For these application, the method to show the information of restaurants are not direct enough. People must enter the name of the restaurant first and then find the information they need from the search result page, which may contain many useless information for them and it is ineffective. Compare with the strong review aggregation system and the huge restaurants database of these applications, few of them has been developed to provide a natural language question-answering (QA) system for restaurant recommendation. Different from the traditional method to search for a restaurant in these restaurant review aggregators, a QA system can provide a reader-friendly textual format of restaurant information that translated from the data on website for users. However, the implementation of that QA system involves both natural language understanding (NLU), natural language processing (NLP) and natural language generation (NLG). It has always been a challenge for people to develop a natural language system. In this project, the implementation of a simple natural language QA system is the primary objective.

* 1. Initial Brief

This section includes the main challenge to implement the system and the aims & goals of this project.

* + 1. Main Challenge

The main challenge of the project is to find a feasible method to implement the natural language process system and the natural language generation system for this project based on limited searchable resources. In addition, this project involves new programming technologies. Not only a new programming language Python, but also Natural Language Toolkit (NLTK), a platform for building Python programs to work with human language.

* + 1. Aims and Goals

Aims:

* Collect enough London restaurants information from the Internet and put them into a local database.
* Understand background of NLP and NLG and be familiar with the core implementation process of them.
* Learn Python and Django web framework and apply them to build the system.
* Design and implement a QA system that translates the discrete restaurant information into reader-friendly textual format.

Goals:

* **NLP system:** The QA system must ‘understand’ users questions (the system should take the question as an input and then identify the question type and process the question to a set of keywords
* **MySQL database:** The system must contain a MySQL database to put the restaurants data. And the system must have the ability to search and operate the database.
* **NLG system:** The system must generate a paragraph of the restaurant information with well-organized reader-friendly textual format. And this system must have the ability to provide different answers for different types of question.
* **User Interface:** The QA system must have a simple user interface for users to ask questions. This user interface should contain at least a chat window and an input field.
* **Usability:** The system must achieve a high efficiency and user satisfaction in a specified context of use.
  1. Overview of Approach

In this project, several weekly meetings with supervisor were organized first to create ideas of the implementation of the system. These meetings not only gave the opportunity to discuss about the main challenges, but also to receive feedback from the supervisor. Moreover, email was used to daily communicate with the supervisor.

Throughout the project, the implementation approach was step-by-step. In each work period, a part of the QA system had to be implemented. In the project plan document (Appendix D), the work plan of the completion of the project is shown. This work plan outlined objectives and works that had to be completed in each period. As all works were completed, testing process and user feedback questionnaire were used to improve and evaluate the quality of this project.

* 1. Report Structure

This report consists of seven chapters. In chapter 1 and 2, the background information and the overview of the project will be introduced. This section will include the introduction of related programming language and web frameworks used in the project. Chapter 3 covers the detailed requirements and related analysis. The requirements will be shown as a MoSCoW table. Furthermore, use cases and user interviews will be covered in this section. Chapter 4 will discuss the design and implementation process of the system. In this section, an overview and the architecture of the system will be given. And this report will provide the implementation detail of each component of the system separately. In chapter 5, testing strategy of the project will be descripted. Chapter 6 will cover the introduction and the result of the user feedback questionnaire. And the result will be analyzed in this report. Finally, chapter 7 contains the conclusion and the evaluation of the project.

1. Background
   1. Introduction

At the start of the project, learning the basic natural language knowledges from the NLTK textbook and the Internet was an important process. Although there are numerous example NLP and NLG systems on the Internet, these systems are complex, and it was difficult for a beginner to read the source code and learn the algorithms. To implement the project, much time was spent on the preliminary process and the period to get familiar with these knowledges. Background information below are the main technologies and resources that are learnt and used during the implementation of this project.

* 1. **Project Context**

This section provides the contextual knowledge required to understand the project and it involves the technologies that were used during this project.

* + 1. **Question-answering (QA) System**

Question-answering is a computer science technology within the field of NLP. It is concerned with developing applications that can answer human questions automatically in the form of short natural language texts. Open domain and closed domain are two main methods to implement the QA system. For the open domain QA system, it involves the knowledges of computational linguistics, information retrieval and knowledge representation. It is hard to implement an open domain QA system since there are a huge number of question types and this system must identify the correct one to return an answer.

For this project, a closed domain question-answer system was the objective to implemented. By contrast with an open domain system, this system has a restricted domain, which means it has fewer question types to identify. And it is easier for a beginner to implement it. Especially for the project, the system only has to answer the questions about London restaurants. Moreover, it is also easier for the system to find the correct answer.

To have a deeper understanding about the implementation process of a closed domain QA system, a review of literatures and reading materials from the Internet were conducted, a **Keywords Based Closed Domain Question Answering System** report written by Rohini P. Kamdi and Avinash J. Agrawal gives a systematic introduction about how to implement a closed domain QA system. And this report also introduces the overview of the system architecture and several core algorithms in this system. Furthermore, there are several reading materials on **Quora**, which provide useful information of the development of a QA system.

* + 1. **Natural Language Processing (NLP)**

Natural language processing is a computer technology that concerns with the interactions of natural languages and computer. It focuses on the processing procedure of natural language data and translate these data into an organized format. The processing of natural language involves both syntax part and semantics part.

For the syntax part, it covers part-of-speech (POS) tagging, parsing and word segmentation. POS tagging aims to determine the POS for each word in a sentence. In this project, the POS tagging tool provided by NLTK was used. This tool not only tag each word with POS, but also consider the context of the word to eliminate the ambiguity of many words that have multiple POS. Word segmentation part aims to separate a chunk of continuous text into words in a sentence. For the semantics part, named entity recognition, question-answering and natural language generation are included. Named entity recognition is used to determine which items in a sentence map to proper names. In this project it is used to identify the name of the restaurant.

**The book of NLTK** was used to learn the natural language processing. This book includes the knowledges of NLP and also the related functions that can be used in Python programming.

* + 1. **Natural Language Generation (NLG)**

Natural language generation is the process that takes input data and translate these input into a sensible reader-friendly text. Four core elements are important in a NLG system: knowledge source, communicative goal, user model and discourse model. Knowledge source is the knowledge that the system has, for example, a database. And communication goal is the expected text the system output. User model is the group of people that will read the output text and discourse model is the style of the output text (Swizec, 2012).

In a traditional architecture of a NLG system, content determination is the first step. The system will determine what content should be included in the output, more generally, what the system going to say. Secondly, the system will plan the sentences semantically. Finally, specific words that used in the sentences will be determined. And the system will check the connections between each sentence to make sure all connections are smoothly.

* + 1. **Natural Language Toolkit (NLTK)**

*“NLTK is a leading platform for building Python programs to work with human language data.” (NLTK 3.2.5 documentation, 2017)*

As the introduction on the website of NLTK website, it is a powerful programming library for Python application developers to process natural language data. In involves more than 50 corpora and related resources. Throughout the project, NLTK documents was used as a reference and NLTK library was used to implement the functionalities of the QA system.

* 1. **Programming Language and Frameworks**

This section involves the programming language and related web frameworks that used in the project.

* + 1. **Python**

Python is used as the programming language of this project since it has many powerful libraries like NLTK and PyDictionary. NLTK is one of the leading platform for NLP programming and PyDictionary is a similar library as WordNet to get meaning, synonyms, translations and Antonyms of words (PyDictionary 1.5.2 documentation, 2018). Furthermore, compare with other programming languages, Python has more powerful system functions and it is easier for developers to implement data structures and some complex algorithms.

* + 1. **Django**

The system was designed to develop as a web application. To combine Python and web development together, Django is a reliable platform to implement it. This web framework provides a platform for developers to write Python, HTML and JavaScript together. Moreover, Django encourages agile development and clean design and it can prevent much of the problems of web development to give developers a better programming environment without the reinvent the wheel (Django, 2018).

* + 1. **Materialize**

Because of the related development experiences before, Materialize was used to build the user interface of the project. Materialize is a responsive front-end framework based on Material Design. It focuses on a better user experience and it is easy for developers to use.

* + 1. **BeautifulSoup**

Web crawler was used in this project to get sufficient restaurants data from the internet. BeautifulSoup is a powerful library in Python to get data from the websites. It can find the specific components of HTML statements of a website and get the data of the components and return it to the local program.

1. Requirements and Analysis

The establishment of the technical requirements is the most difficult part of the conceptual work in software engineering (Brooks, 1986). In Chapter 3, requirements analysis and user analysis will be covered. Requirements analysis involves a MoSCoW table. Use cases diagram and user interviews are used to analyze the interaction between users and the QA system. Requirements discussion with the supervisor is also mentioned in this section.

* 1. MoSCoW analysis

MoSCoW table is a technique used to identify the priority of each requirement. It divides all requirements into four categories: Must have, Should have, Could have and Won’t have (Clegg, 2004). By using the MoSCoW method, developers will firstly try to implement all the requirements but the Should have and Could have requirements will have lower priority.

An initial MoSCoW table was made at the beginning of the project (Appendix D). With the development of the project, the requirements were changed and below is the final MoSCoW table for this project.

Functional Requirements:

|  |  |  |
| --- | --- | --- |
| **ID** | **Requirement** | **Priority** |
| **Database** | | |
| DB01 | The system shall have a database of London restaurants data | Must have |
| DB02 | The system shall have a database to store all question types | Must have |
| **Natural Language Processing** | | |
| NLP01 | The system shall be able to identify the type of users’ questions | Must have |
| NLP02 | The system shall be able to get the keywords of users’ questions | Must have |
| NLP03 | The system shall be able to identify entities (Restaurant name) | Must have |
| **Natural Language Generation** | | |
| NLG01 | The system shall be able to find the required data from the database | Must have |
| NLG02 | The system shall be able to answer all possible questions that relate to London restaurants. | Could have |
| NLG03 | The system shall be able to return answer to the user. | Must have |
| NLG04 | The system shall be able to generate answer with different sentences each time | Should have |
| NLG05 | The system shall have a regular expression parser that identifies the regular expressions in .txt files and then translate them into sentences | Should have |
| NLG06 | The answer should include the restaurant pictures | Won’t have |
| NLG07 | The answer should include the link of the restaurant website | Won’t have |
| **User Interface** | | |
| UI01 | The system shall have a user interface for users to input their questions and get answers from the system | Must have |
| UI02 | The user interface shall be like a chat window for users to communicate with the QA system | Should have |

*Table 1*

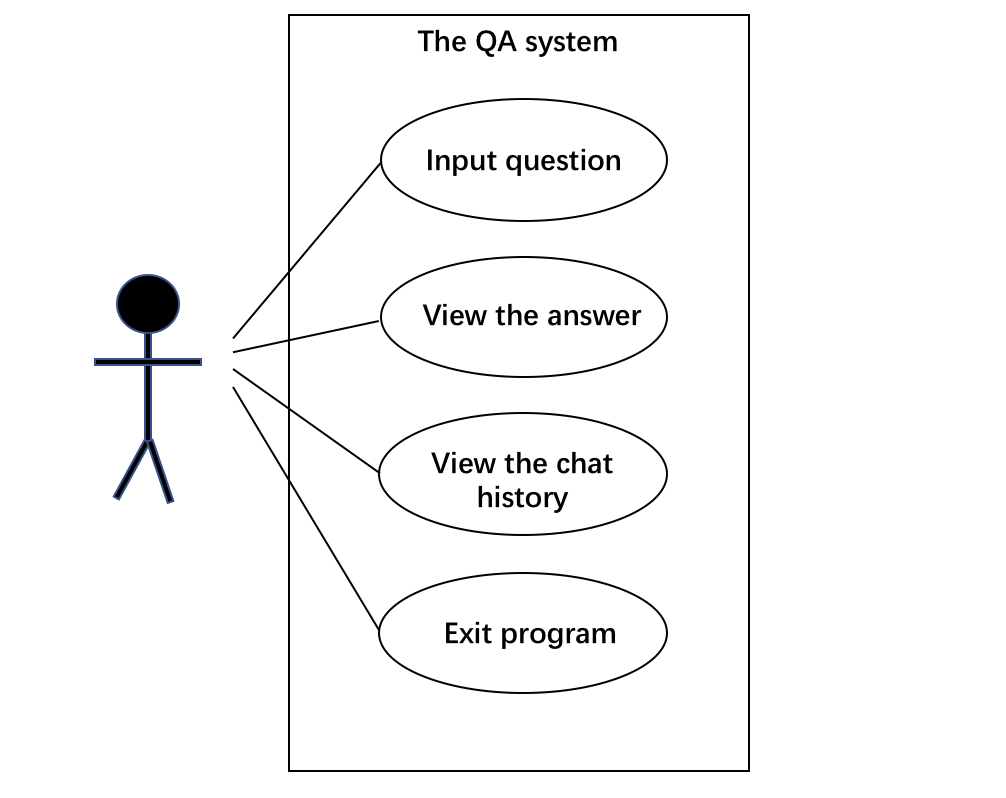
Non-Functional Requirements:

|  |  |  |
| --- | --- | --- |
| **ID** | **Requirement** | **Priority** |
| **Implementation** | | |
| IP01 | The system shall put the databases and website on a server. | Could have |
| IP02 | The system shall be implemented using Python and NLTK | Must have |
| **Performance** | | |
| P01 | The system shall return answers less than 1 second | Could have |
| P02 | The system shall have a concise and elegant user interface | Should have |
| P03 | The system shall be able to return reasonable answers without any bugs or exceptions. | Should have |

*Table 2*

* 1. **Use Cases**

Use case is numerous actions that define the interactions between the system and users. The diagram of use case is given below, and the full use cases can be seen in Appendix F.



*Use case diagram*

* 1. **User Interviews**

*“You cannot understand good design if you do not understand people; design is made for people” – Dieter Rams (Rob, 2012)*

As Dieter said in the 1976 speech, users determine the design of an application. It is important to establish the requirements from a user’s point of view. In addition, there are few similar applications on the Internet as a reference. Therefore, user interviews are used to identify what users expect from the application and what users do not like in a QA system to determine the required requirements of the system. The document of user interview will be attached in the appendices part (Appendix G).

* 1. **Discussion**

The weekly meetings and email communications with Prof. Anthony Hunter were used to generate and exchange ideas of the design of the QA system. The feedbacks collected from the supervisor and user interviews were considered to design the system and gather the required requirements.

1. Design and Implementation

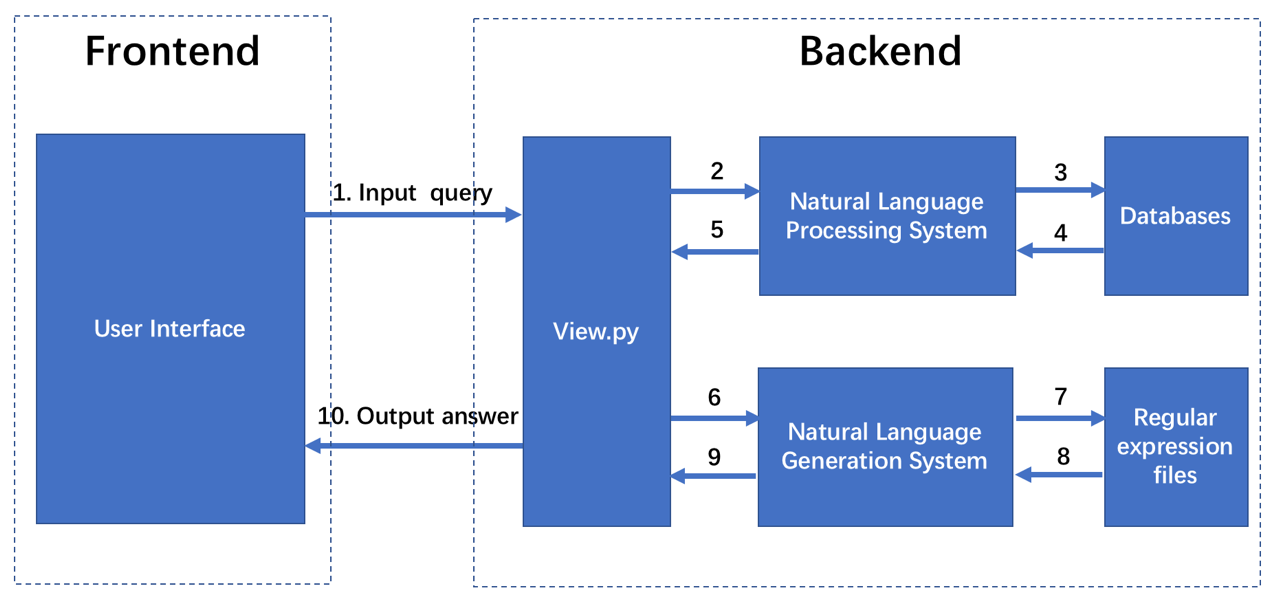
This section covers the design and Implementation of the closed domain QA system. The overview and the architecture of the system will be described. Moreover, the report will outline the algorithms to implement each part of the system. And the design of the user interface will be explained. Finally, this section will discuss challenges and design decisions of the project.

* 1. System Overview

The closed domain natural language QA system has three main components. The first part is the NLP parser. The parser is used for identifying the question type of the user query and extracting the keywords in the query. And then the parser will generate a MySQL query to get the required data from the London restaurant database. A NLG section is the second part of the system. This system requires numerous parameters from the external system and then it will determine the specific contents it generates that required by the QA system. Then the NLG system will generate the specific natural language contents based on the regular expression files and return the text. The final part is the user interface. The UI system is based on the Django web framework and it is used for users to interact with the QA system. These components will be introduced in detail in following sections.

* 1. **System Architecture**

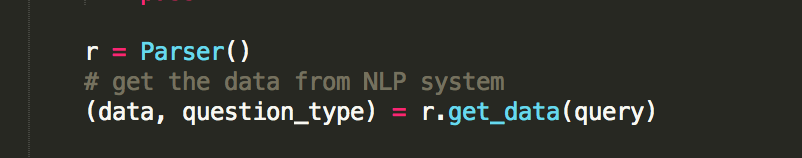
The architecture of the QA system is shown in figure 2. The system structure can be divided into two parts; the frontend interface and the backend system. In figure 2, the square boxes represent different components in the system and the arrows between them represent the transmission of data. And the numbers on the arrows represent the processing steps throughout the running of the QA system.



*Figure 2*

For the frontend system, it includes a website user interface and the only function of the frontend system is to get the input query and return the answer to the user. This system is written in HTML, CSS and JavaScript. And Materialize frontend framework is used to build the website components in the user interface.

For the backend system, a NLP system, a NLG system and an interim system are included. The interim system is an interim platform of the NLP and NLG system and it is written on view.py, which is a Django file that takes a Web request and returns a Web response to the HTML page. Throughout the using of the QA system, as the code in figure 3, view.py gets the user query posted from the frontend (**Step 1 in figure 2**) and then pass it into the NLP system (**Step 2, 5**). The code*r = Parser()* represents the initialization of the NLP system and the function *get\_data* is used to get required data from the database based on the input user query.



*Figure 3*

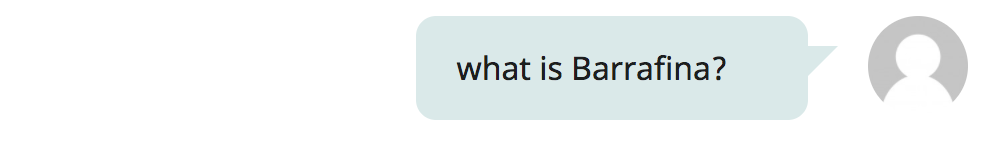
Then it will process the data received from the NLP system into a formatted array. After that, according to the identified question type received from the NLP system, view.py will determine different parameters that pass to the NLG system and return the natural language text received from the NLG system to the frontend (**Step 6, 9, 10**).

Based on the interim system, the NLP system receives the input query from the frontend and then it will get required data from the database **(Step 3, 4)** and pass the data back to view.py. The information of the databases and the implementation method of NLP system will be given in section 4.4 and section 4.5. The NLG system gets the parameters passed by the interim system and use them to determine the contents it generates. The system will generate sentences by reading regular expression files **(Step 7, 8)** and then return the sentences back to the interim system. The details of NLG system will be described in section 4.6.

* 1. **Interim System**

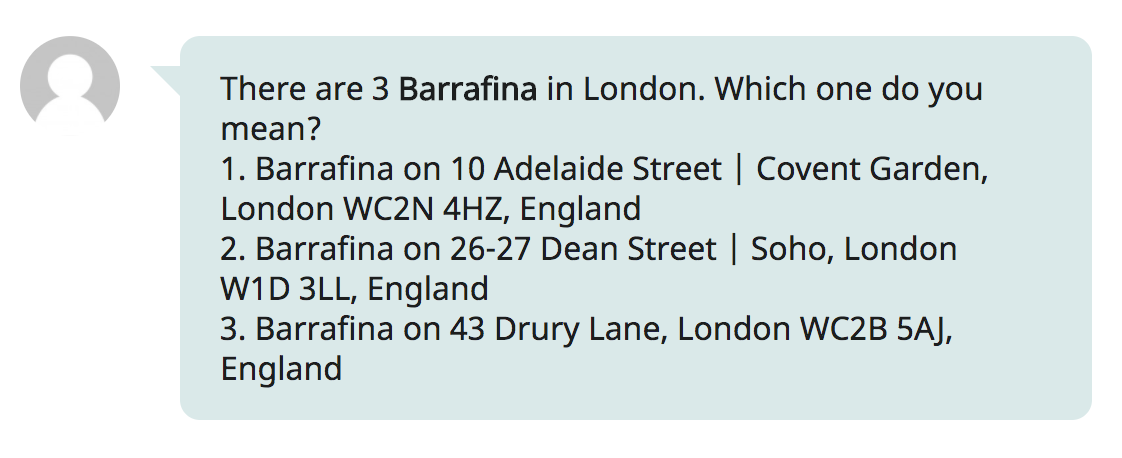
Interim system is an important part of the QA system. As the report mentioned in last section, it is written on view.py and it is responsible for the data transmission between the frontend and the backend. From the figure 2, it shows that the operation of the NLP system and NLG system is processed in the interim system. Since each time the user query is different, the data received from the NLP system will be different. Therefore, it is also important for the system to process the data returned from the NLP system differently to meet the requirements of the parameters of NLG system for different questions.

Moreover, the interim system is able to solve the ambiguous query problem. Since the existing NLG system can only generate texts for a single restaurant, the QA system will face an issue when the user asks an ambiguous question. For example, the user may ask a question like this:



*Figure 4*

Barrafina is a Spanish tapas bar and it has a chain of 3 restaurants in London. The existing NLG system cannot generate a text for 3 restaurants. To solve this problem, the system will first get the data from the NLP system and check whether the number of restaurants received is more than 1. If the number is higher than 1, the system will generate options and return this text to the web.

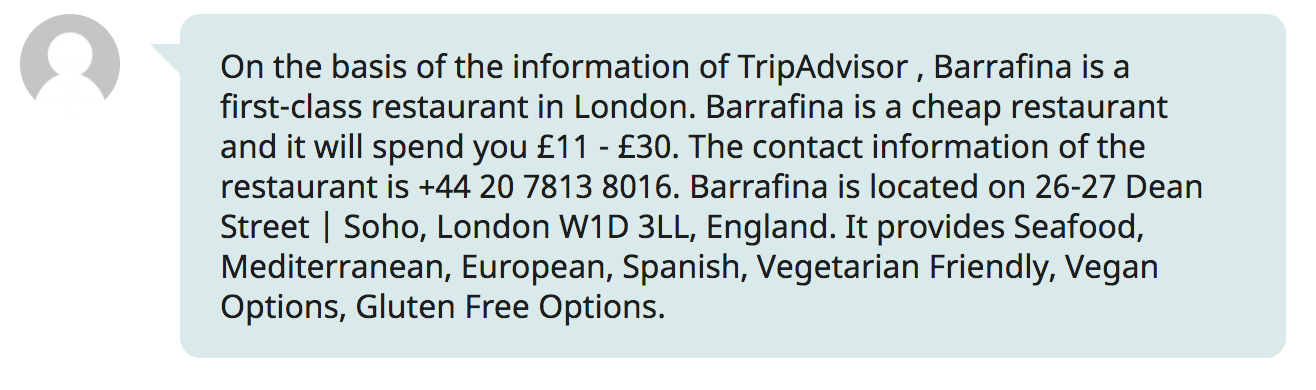


*Figure 5*

And the user can reply a number to get further information. The JavaScript code written in index.html will check whether the user query is a number. If it is, the code will check the last user query. If the last user query exists, the code will generate a new query:

  
*Figure 6*

For example, if the last user query is figure 4 and the user has entered 2 in the QA system. The new query that passed to the backend system will be: “what is Barrafina?<2>”. Then the interim system will find whether the query has the specific format *<number>*. If it has, as the text in figure 7, the system will get the natural language description of the restaurant from the NLG system corresponding to the number and then return the text to the web.



*Figure 7*

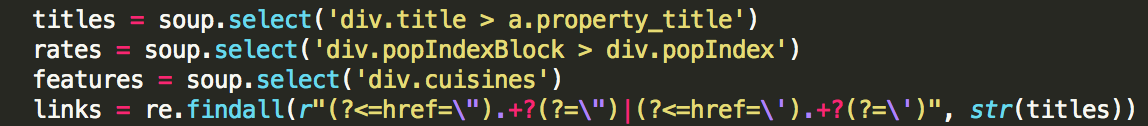
* 1. **Database**

Since the identification of keywords and entities of the NLP system is based on the London restaurant database, and the identification of question types of the NLP system is also based on the question type database. Therefore, the performance of the system is partly depended on the databases. And database is one of the key system in the project. Because of the development experiences of MySQL database before, MySQL database is used to manage data for the QA system in this project. The closed domain QA system includes two databases, the first one is used to store the data of London restaurants and the second one is used to store the information of different question types.

For the London restaurant database, the source of the data is a significant problem at the beginning of the project. It is difficult to find an available London restaurant database on the Internet. The use of API was considered but the API of TripAdvisor is only available for businesses but not for students and research. And the API of Yelp is not sufficient for the functionality of the QA system. After discussion with the supervisor and research, web crawler was used to get restaurant data.

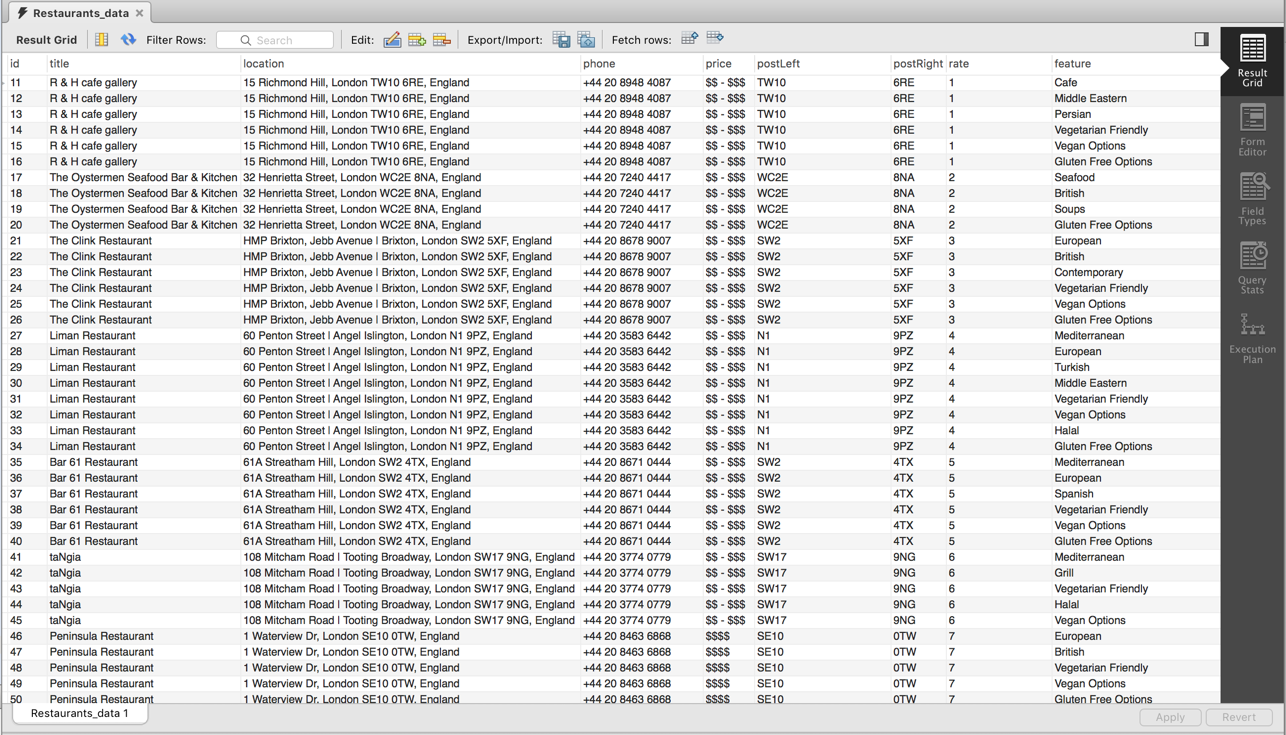
BeautifulSoup was used to collect London restaurants data from the Internet. Because of the lack of knowledge of web crawler before, research and learning of BeautifulSoup was conducted at the beginning of the project. Finally, the website of TripAdvisor was decided as the target to collect restaurants data because the website of TripAdvisor has a clear structure and it is easier for the codes to identify the web components comparing with other websites.

Because of the huge quantity of restaurants information on the website, several of them were selected to be included in the database: **restaurant name, location, post code, price, phone number, rate and the features**. To get these information, the web components that include the required information was identified as the preliminary process and then the web crawler will use the function of BeautifulSoup to find all these components in each page of the list of restaurants of TripAdvisor and extract the restaurant information from the components and store them in the variables. The codes can be find in Figure 8.



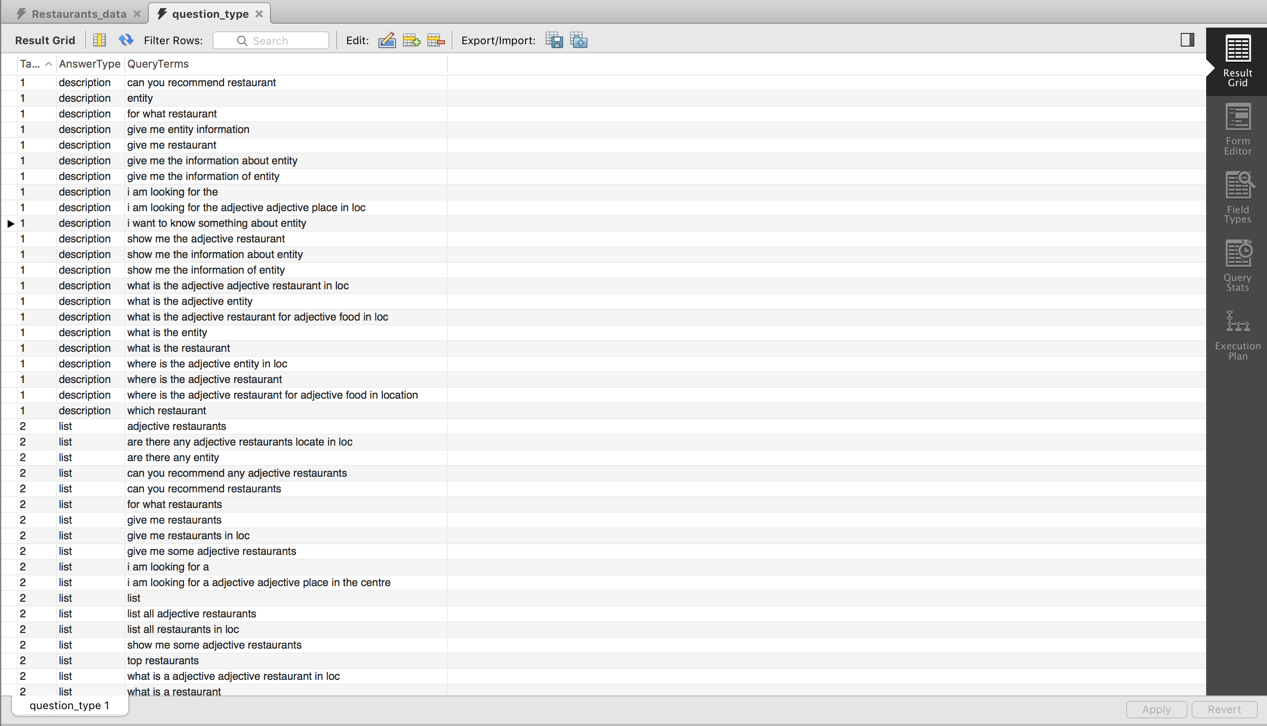
*Figure 8*

After that, dictionary is used to store these variables and each dictionary represents a set of information for a restaurant. These dictionaries will be inserted into the local Restaurants\_data database. Finally, the complete database is shown in Figure 9, more than 16000 restaurants in London were inserted in the database.

**

*Figure 9*

For the question type database, it is used for the NLP system to identify the question type of the user query. It includes 3 columns, target, answer type and query terms. **Target column** represents the question type of this row and there are totally 5 question types now. **Answer type column** is the description of the previous column. **QueryTerm column** includes a set of terms that represent the characteristic of question types. The generation of these query terms is based on a user questionnaire. The questionnaire asked for users to write a question for the QA system and then these questions are collected. The adjective, location and entities in these questions were replaced by different significant words. For example, *what is Barrafina* will be replaced by *what is entity.* What is the location of Barrafina will be replaced by What is the loc of entity. And then these query terms will be inserted into the database. The full documentation of sample questions that collected from the questionnaire will be referenced in the appendices part. The using of these columns will be introduced in section 4.5 with the NLP system. Different from the London restaurant database, the existing row insertion method of the database is manual. In future work, a user interface for inserting a row will be developed. The database is shown in Figure 10.



*Figure 10*

* 1. **Natural Language Processing System**

The implementation of the NLP system of the project is based on the functions of the NLTK textbook and the book has a clear description about how to use the functions to process with natural language texts. At the beginning of the project, long time had been spent to discuss about the design of the NLP system because there are two options to implement it. One is focus on the relationship extraction; this algorithm will identify the context relationship for every word by using a parser tree. It requires a deep understanding of NLP and NLTK functions. Another algorithm is focus on the keyword extraction and this algorithm is easier to implement comparing with the first method. This algorithm involves following steps: 1. Remove stop words and punctuations from a query. 2. Extract keywords from a query. 3. Identify the question type of the query. By balancing the time for implementing the NLP system and the NLG system, the second algorithm is used in this project.

Based on the NLTK library and the algorithm discussed above, the NLP system can be separated into four parts: query formulation, answer type classification, relation extraction and SQL query generation. For the query formulation and relation extraction part, it covers following main functions:

* get\_keywords
* get\_phrases
* generate\_phrases

In this part, the system will first find the entity in the set of phrases by comparing each phrase with the restaurant name in the London restaurants database in the generate\_phrases function. The phrase with the longest match will be identified as the entity. To simplify subsequent processes, the entity will be replaced by the word ‘entity’. Here is the example:

*Example: What is you me sushi? -> (You Me Sushi, entity) -> What is entity?*

Then the system will first tokenize the query into words in the get\_keywords function. Secondly, by using the NLTK POS-tagger, the system will ignore stop words and verbs from the query in the generate\_phrases function. After this process, the system will get a set of phrases. Thirdly, the adjectives in the phrases will be extracted. Then a set of keywords will be extracted from the user query. And the system will tag each word with its characteristic. Finally, the system will traverse the keywords array; if an unknown keyword found, remove it. For example, the following examples show the query processing step in the NLP system:

*Example 1:*

*The user query: Does Burger King provide sushi?*

*Step 1: Entity detection: (Burger King, entity)*

*Step 2: Replace the entity: Does entity provide sushi?*

*Step 3: Remove stop words and verbs from the query: [entity, sushi]*

*Step 4: Adjective extraction: No adjective found*

*Step 5: Tag each word with its characteristic:* ***[(entity, ‘title’), (sushi, ‘feature’)]***

*Example 2:*

*The user query: Show me some great restaurants in NW1.*

*Step 1: Entity detection: No entity found*

*Step 2: Remove stop words and verbs from the query: [great restaurants, NW1]*

*Step 3: Adjective extraction: [great, restaurants, NW1]*

*Step 4: Tag each word with its characteristic: [(great, ‘goodrate’), (restaurants, ‘unknown’),* *(NW1, ‘postleft’)]*

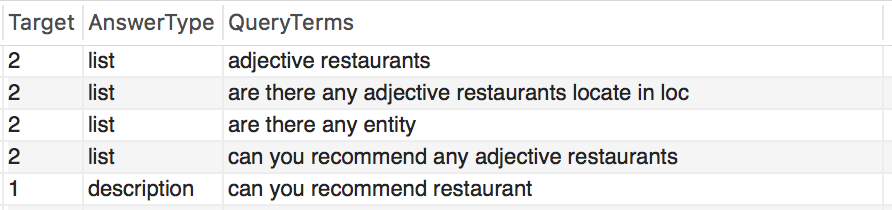
*Step 5: Remove unknown:****[(great, ‘goodrate’), (NW1, ‘postleft’)]***

After extracting keywords from the query, the answer type classification is followed. This part contains following main functions:

* calculate\_probability
* define\_question\_type

Answer type classification is the most important component in a QA system and the method to identify the question type of the user query is major task for implementing the QA system. After reviewing of numerous literatures, the question classification method according to the answer type described by Robini is used in the project (Robini, 2015). Because for a closed domain QA system, the question types are limited and also the answer types are restricted, this method divides the answers of the QA system into different types: description, yes or no, a list of restaurants and the details of a restaurant. For these different answer types, the questions can be classified accordingly into different target numbers.

To identify the question type of a user query, numerous query terms were prepared in the question type database. The query terms are generated manually according to the user questions collected from the questionnaire. Firstly, the answer type of each question is identified. Secondly, the adjectives, the postcodes and the entity of the question will be replaced by ‘adjective’, ‘loc’, and ‘entity’. For example, “Are there any good Chinese restaurants locate in N1C?” will be first identified as target 2 because the answer will be a list of restaurants. And then the query will be replaced by “are there any adjective restaurants locate in loc”. Finally, this query term will be inserted into the question type database in Figure 11.



*Figure 11*

After the preparation of the database, the Jaccard coefficient mechanism is used to identify the question type. When a user input a query, the NLP system will first replace the query like the process above and then the user query will have the same format as the query terms. And the Jaccard coefficient mechanism uses the formula below to obtain the probability for each query terms:

Probability = Query ∩ Query Terms / Query U Query Terms

The system will compare the user query word by word with each query terms and calculate the number of intersection words of query and the query terms and the number of union words of them. Then the probability of the query terms is the division of the intersection number and the union number. After the calculation process, an array of probabilities will be obtained and the query terms with the highest probability will be identified. Finally, the target and the answer type corresponding to the query terms can be determined and the target number is identified as the question type.

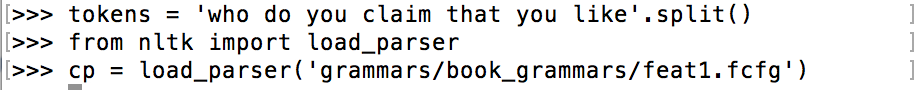
The potential problem of this mechanism is that when the size of the question type database growing, the calculation process will take a relatively long time to get the array of probability and it takes a large space to store all these probabilities in an array. In the future work, the calculation algorithm of the Jaccard coefficient mechanism need to be optimized.

The fourth part is SQL query generation. The formation of the SQL queries is based on the set of keywords and the question type obtained before. For each question type, different SQL query will be generated, and different keywords will also determine the condition of the query. For example, if the list of the keywords is *[(great, ‘goodrate’), (NW1, ‘postleft’)]*and the question type is 1 (Description), the SQL query will be *SELECT \* FROM Restaurants\_data WHERE rate = (SELECT max(rate) FROM Restaurants\_data WHERE postLeft like ‘NW1’)*. Then the SQL query will be returned to the interim system.

* 1. **Natural Language Generation System**

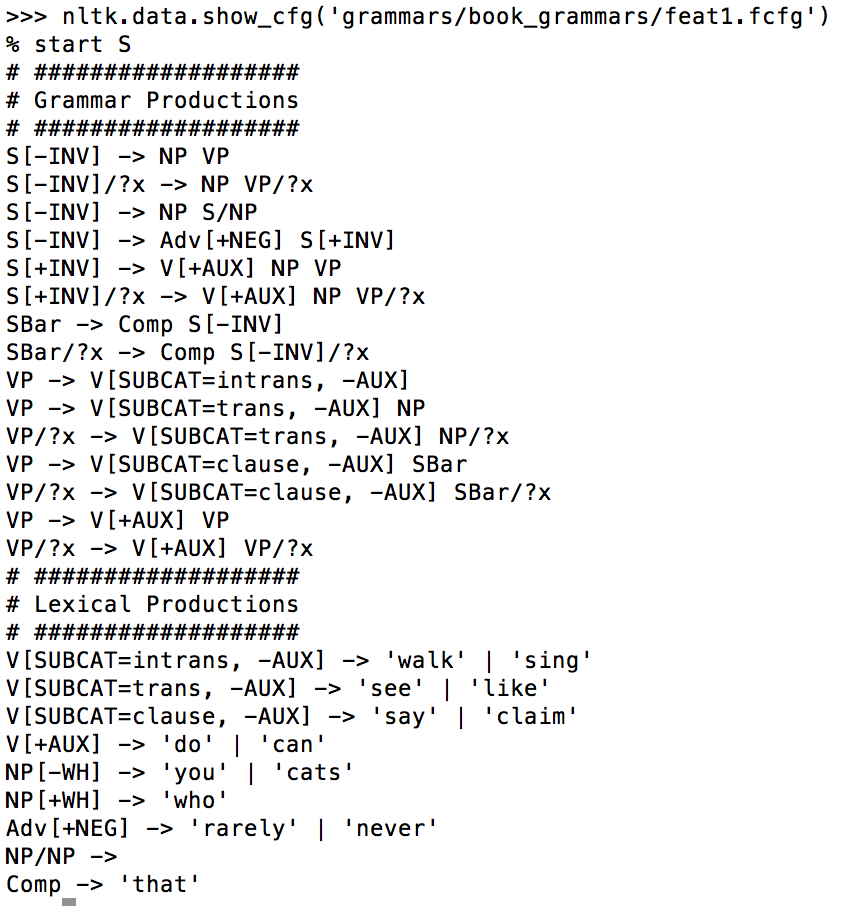
After a research about the existing NLG system on the Internet, most of them are based on template-based method, which generate texts by putting keywords in a template. Moreover, there have not exist a common method for implementing every NLG application. Therefore, for developing a ‘smarter’ NLG system, several discussions about the design of the NLG system with the supervisor were scheduled. From these discussions, the feature-based grammar analysis in the NLTK book was advised by the supervisor.

The NLTK book introduces a feature-based method to parse the structure of sentences. This method takes a sentence as an input and then the input will be loaded as a parameter of a built-in parser of the NLTK. And then the parser will generate a parser tree for the sentence. For the built-in parser of the NLTK, it needs a regular expression file as its background knowledge to generate a parse tree. For example, to generate a parse tree of a sentence by using the NLTK, the grammar of the sentence must be loaded into the parser like the code in Figure 12.



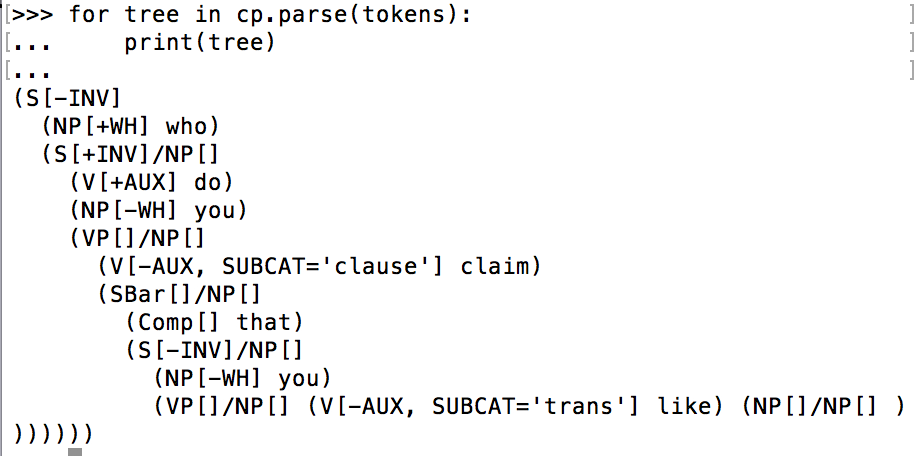
*Figure 12*

And the grammar is written in the form of regular expressions in Figure 13.



*Figure 13*

It is clearly to be seen that the regular expressions start with the non-terminal *S[-INV]* and there are several different sentence structures that can be generated by this grammar file. For example, *S[-INV] -> NP VP* is one of the sentences can be generated by these regular expressions. NP means Noun Phrases and VP means Verb Phrase; it means the sentence S can be formed by a noun phrase followed with a verb phrase. And the structure of NP and VP is also claimed in this file. Through the traverse of the Non-Terminals, terminals will be reached as the expressions in Figure 13; For example, *NP[-WH] -> ‘you’ | ‘cats’* and *Adv[+NEG] -> ‘rarely’ | ‘never’*. A non-terminal *NP[-WH]* can be represents as ‘you’ or ‘cats’ in this grammar. To generate the parse tree of a sentence, the tokens of the sentence will be put in the parser and then the parse tree and the part-of-speech of each word will be printed in the terminal.



*Figure 14*

The parse tree in Figure 15 is a readable version of the tree in Figure 14.

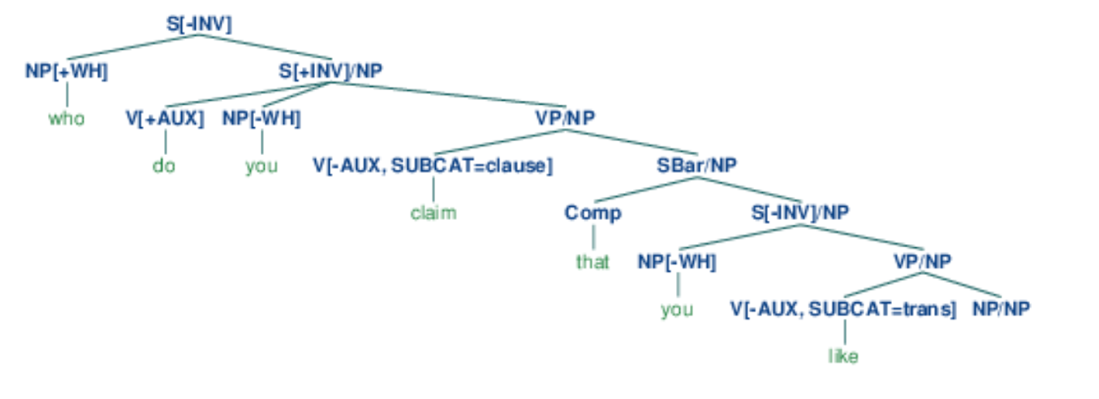


Figure 15 (NLTK book, Chapter 09, 2015)

Based on the feature-based grammars analysis of the NLTK book, a specific NLG algorithm for the project was designed. The main functions of the NLG system in the project are:

* generate\_long\_sentence
* list\_to\_string
* break\_down
* read\_expressions
* split\_to\_part

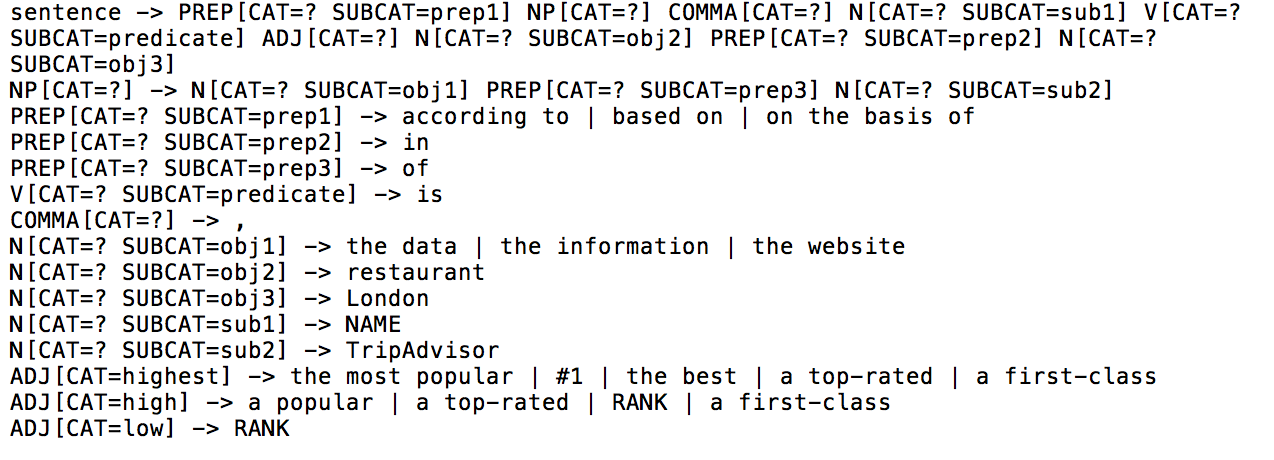
And the regular expression files of the NLG system are listed below:

* feature.txt
* phone.txt
* price.txt
* rank.txt
* address.txt

The contents of these regular expression files can be seen in Appendix J.

The NLG system of the project has three parts, content determination, expression reading and grammar correction. For the first part, the NLG system will first analyse the question type received from the interim system in the generate\_long\_sentence function. And different question type will have different sentence generated in the system. If the question type is target 1, which will have a descriptive answer, the NLG system will generate sentences to describe the restaurant completely. Otherwise the system will generate a more specific answer corresponding to the question type received.

Expression reading is the core part of the NLG system. From the content determination part, the regular expression files that need to be read by the system was determined. And then the system will call the read\_expressions function to enter the specific regular expression file and construct a sentence. The regular expression file is written as the example in Figure 16:



*Figure 16*

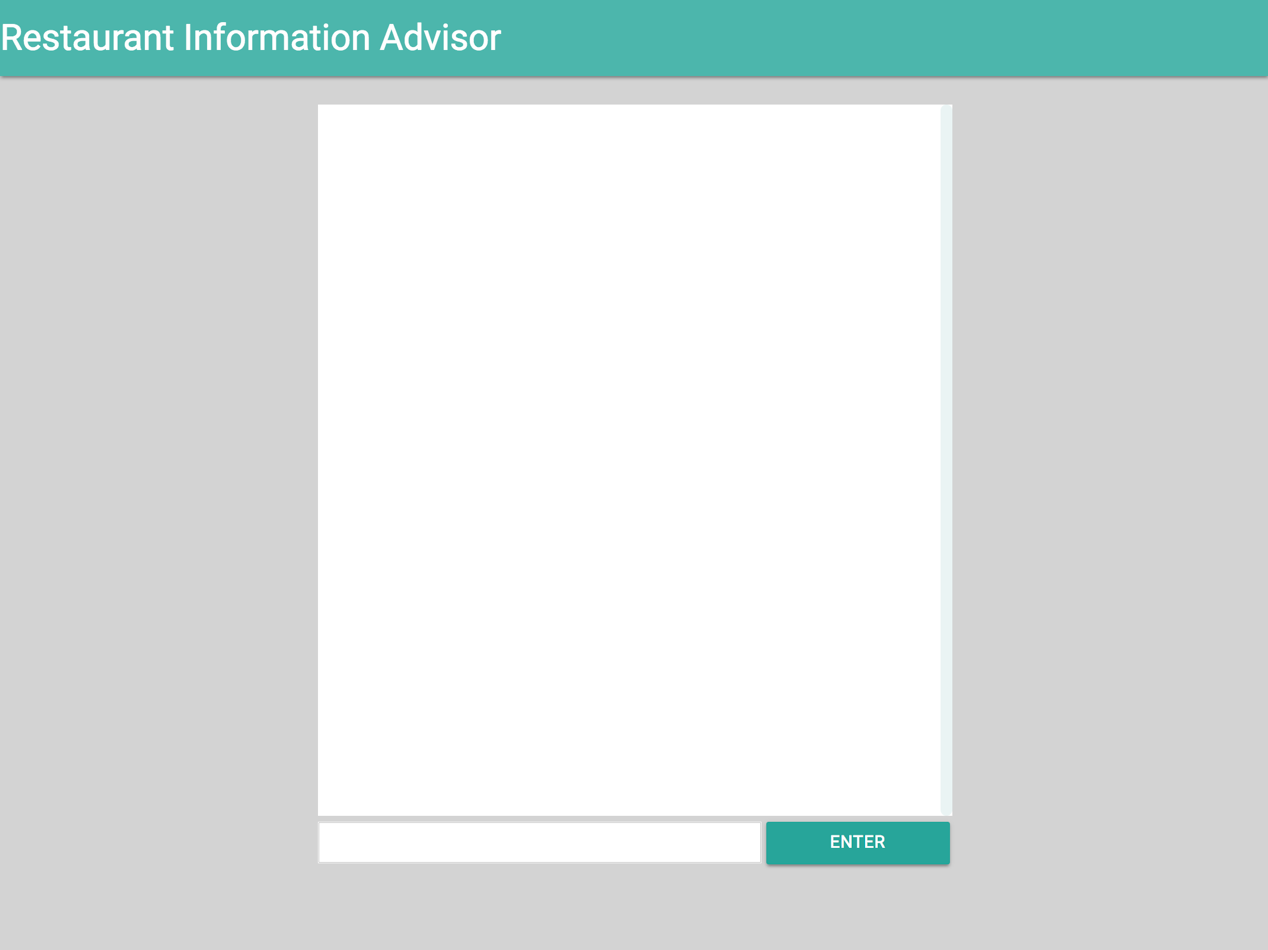
This grammar has a similar structure like the example in Figure 13 and the non-terminal ‘sentence’ is the start point for the code to read. The non-terminals on the right-hand side of the right arrow represent the non-terminals on the left. And the non-terminals have brackets to contain additional information. For example, *N[CAT=? SUBCAT=obj1]* means it is a noun with unknown category and the sub-category of the noun is obj1. When the system traversing through the non-terminals, the unknown category that represented by a question mark will be replaced by the parameter CAT of the read\_expressions. For example, if CAT = highest in the read\_expressions, then *N[CAT=? SUBCAT=obj1]* will be replaced by *N[CAT=highest SUBCAT=obj1]*. The use of category is used to generate different sentences in a single regular expression file. For example, *ADJ[CAT=highest]*, *ADJ[CAT=high]*, *ADJ[CAT=low]* are defined in Figure 16. When the question mark of *ADJ[CAT=?]* is replaced by ‘highest’, *ADJ[CAT=highest]* will be used to generate a sentence. Furthermore, ‘|’ is used in the regular expressions and it represents a OR gate. For example, there are 3 terminals in *N[CAT=? SUBCAT=ojb1]* and each time the NLG system will randomly choose one terminal from the terminals.

To generate a sentence, the system will first read the file line by line to identify the non-terminal ‘sentence’ in the file and then it will split the regular expression into two parts by the right arrow in the expression. The left part is ‘sentence’ and the right part is the non-terminals that represent it. Then the code will traverse every non-terminal in the right part. For example, the first term of the right part of the non-terminal ‘sentence’ is the non-terminal *PREP[CAT=? SUBCAT=prep1]*. The system will then read the file line by line again to find this non-terminal in the left part of a regular expression. When the non-terminal found, the right part of the non-terminal will be traversed. The implementation of this process is implemented by using iteration in the read\_expressions function. The iteration of the traverse of the non-terminals will end when the system finds terminals in the right part of an expression. And the terminals will be returned. After all iterations finished, an ordered list of terminals is generated. Then the system will use list\_to\_string and break\_down function to transform the list to a string. Finally, a sentence is generated. For example:

*Initial sentence: NAME is a RANK restaurant.*

After the initial generation of a sentence, the grammar correction part is processed. This process will replace the keywords received from the interim system into the sentence. For example, the terminals RANK and NAME in the sentence will be replaced by the actual rank and the name of the restaurant. After that, since sometimes more than one sentence will be generated, the first digit of sentences will be capitalized, and a space will be added after the full stop of each sentence. Finally, the NLG system will return the generated sentences to the interim system.

* 1. **User interface Design**



*Figure 17*

* 1. **Challenges and Design Decisions**

Sss

1. Testing
   1. Testing Strategy
   2. **Unit Testing**
   3. **Use Case Testing**

1. User Feedback
   1. Introduction
   2. **Findings and Discussion**
2. Conclusion and Evaluation
   1. Conclusion
   2. **Evaluation**
   3. **Future Work**
3. References

Greg Sterling, 2012, <https://searchengineland.com/study-64-percent-of-mobile-restaurant-searchers-convert-immediately-or-within-an-hour-135188>

Kevin Kent, 2016, <https://www.reviewtrackers.com/mobile-customers-food/>

Swizec Teller, 2012, <https://swizec.com/blog/natural-language-generation-system-architectures/swizec/4535>

NLTK 3.2.5 documentation, 2017, <http://www.nltk.org/>

PyDictionary 1.5.2 documentation, 2018, <https://pypi.python.org/pypi/PyDictionary/>

Django, 2018, <https://www.djangoproject.com/>

Brooks, 1986,

Clegg Dai & Barker Richard, 2004, Case Method Fast-Track: A RAD Approach. Addison-Wesley.

Rob Alderson, 2012, <https://www.itsnicethat.com/articles/dieter-rams-speech>

Rohini P. Kamdi, Avinash J. Agrawal, 2015, <http://www.mecs-press.org/ijisa/ijisa-v7-n12/IJISA-V7-N12-6.pdf>

NLTK book, 2015, <http://www.nltk.org/book/ch09.html#ex-gapparse>

1. Bibliography

Amit Gunjal, 2017, how do I develop a closed-domain natural language QA system. <https://www.quora.com/How-do-I-develop-a-closed-domain-natural-language-QA-system>

Brad Miller & David Ranum, 2018, Problem Solving with Algorithms and Data Structures using Python, <http://interactivepython.org/runestone/static/pythonds/index.html>

1. Appendices