Individual Report on Project (Group 33)

# Voice-based Flight Ticket Booking System

Name: Xie He

Email address: xieh@usc.edu

Rest of my group: Junqiang Chen, Honglin Li, Yanhua Zheng

GitHub URL of my group: https://github.com/weidadeqiangge/CSCI544_Group33

*I did further work since my group's project completed. So, for my latest and complete code please see the GitHub repository of myself.*

GitHub URL of myself: https://github.com/Shane-Xie-He/NLU-flight-ticket

## 1. Project Overview

### 1.1. Goal

The goal of our group's project is to build a dialogue system that can interact with the user via voice, and help the user to book a flight ticket. Several tasks need to be completed to accomplish this goal. First, we need to build a system that can recognize the voice input of the user and transform it to text. Second, we need to build a natural language understanding (NLU) module that can extract structured information from the text. Then, we need to have a dialogue manager that takes in the structured information extracted, does appropriate processing according to it, and generates appropriate responses to the user. Finally, we need to have a text-to-speech system that transforms the dialogue manager's response to voice, to give the user a more natural experience. We decided to use existing speech synthesizer, and implement our own speech recognizer, NLU module, and dialogue manager. We divided the project into four parts, so that each of the four members of our group can focus on one. The four parts are:

1. Build an acoustic model for speech recognition;
2. Build a language model for speech recognition;
3. Build a NLU module to extract structured information;
4. Build a dialogue manager to process input information and interreact with the user.

My task is the third task above, to build a NLU module.

### 1.2. Method

We use different methods for different parts of our system. For speech recognition, we use the hidden Markov model. We use a toolkit called HTK (Hidden Markov Model Toolkit). We trained our own acoustic model using our own voice, and we built our language model using hand-written grammar based on our assumption of what we should expect for user input. For NLU, we use both rule-based and machine learning (conditional random field) approach to extract information for text. Since this is my part, I will explain it in details in the next section. For

dialogue manager, we implemented a finite state machine to track the status of the conversation and generate appropriate responses.

### 1.3.Evaluation

We didn't have systematic evaluation method to evaluate the performance of our system as a whole, or the performance of the dialogue system. Instead, we tested our resulting system by interreact with it as a normal user, and found out it was mostly satisfying. However, we did evaluate our speech recognizer and NLU module. For the speech recognizer, we evaluate its performance using both the sentence error rate, the ratio of sentences with at least one mistake to all sentences, and the word error rate, the ratio of all substituted, deleted and inserted words to all actual words. For NLU, we mainly use precision, recall and F1-score of the extracted pieces of information to evaluate it. Since this is my part, I will explain it in details in the next section.

## 2. My Primary Responsibility

As stated above, my primary responsibility is to build a NLU module, to extract structured information from input text.

### 2.1.Goal

The goal is to extract information from input text. To do this, my program needs to recognize city names, person names, expressions of time, and other flight ticket related information, distinguish origin / destination city and outbound / return time, and process the information into formatted output.

According to what is provided by my teammate responsible for the language model, further discussions of my group, and a little extension of myself, the text input I expect is like this:

| | |
|---|---|
| 1 | i want [to buy \| to book \| to reserve \| to get] (a ticket \| an air ticket \| a flight ticket) from CITY to CITY [departing on MONTH DATE [YEAR]] |
| 2 | yes \| yeah \| right \| correct \| confirm |
| 3 | [no] (([my name is \| the name is] PERSON_NAME) \| ((the \| my) (origin \| destination) [city] is CITY) \| ([on] MONTH DATE [YEAR]) \| (i want (economy \| business \| first) class) \| (i want (roundtrip \| one way trip))) |
| 4 | [i want to] change (([my \| the] name to PERSON_NAME) \| (the (origin \| destination) [city] to CITY) \| (the [departure \| return] date to MONTH DATE [YEAR]) \| (the travel class to (economy \| business \| first) class) \| (it to (roundtrip \| one way trip))) |

A typical input text could be:

| |
|---|
| i want to book an air ticket from hong kong to london departing on march twenty ninth twenty twenty two |

I implemented both a rule based version and machine learning based version of my NLU module. In the rule based version, I used the above grammar as a reference of my rules for information extraction. However, I didn't adhere to it, and tried to make my code as general as possible to handle input not conforming to the above grammar. In the machine learning based version, I used the above grammar to generate my training data. I also use the above grammar to generate test data to test both versions of my NLU module.

According to the agreement between me and my teammate responsible for the dialogue system, my output should be like this:

```
A Python list, which contains zero or more structures of the following:
("=", "OriginCity", "Los Angeles")
("=", "DestinationCity", "Paris")
("=", "MyName", "Xie He") # The name of the passenger
("True",) # When the user gives a yes answer like "yes", "yeah", "right", "correct", "confirm"
("False",) # When the user gives a no answer
("=", "Class", "Economy") # for travel class
("=", "RoundTrip", "True") # "True" for roundtrip, "False" for one way trip

# Outbound time
("=", "OutDepartureTimeMonth", 12)
("=", "OutDepartureTimeDate", 2)
("=", "OutDepartureTimeYear", 2016)

# Return time
("=", "RetDepartureTimeMonth", 12)
("=", "RetDepartureTimeDate", 2)
("=", "RetDepartureTimeYear", 2016)

# Time, not provided in the utterance to be outbound time or return time
("=", "DepartureTimeMonth", 12)
("=", "DepartureTimeDate", 2)
("=", "DepartureTimeYear", 2016)
```

As said above, I used two different method to accomplish the goal, one of rule based and the other one of machine learning based. Here I describe them as follows.

## 2.2.Method 1 – Rule Based

The first approach I tried is a rule based approach. Since I already have the grammar of the expected input, I can directly look for patterns in the input text, and extract structured information from it, including origin / destination city names, person names, outbound / return time, and other flight ticket related information. However, as said above, I didn't strictly adhere to the grammar provided above, and tried to make my code as general as possible to handle input not conforming to the grammar. Here I briefly describe the rules I used to extract information.

1. Keywords or patterns for yes / no answers, months, travel classes and roundtrip / one way trip words, are so specific that can be recognized directly.
2. If 1 or more consecutive unknown words is encountered, then this chunk of words could be a proper name, which could be origin city name, destination city name, or person name (of the passenger).
3. If a proper name is preceded by "from", "origin is", or similar structures, then it is an origin city name. If it is preceded by "to", "destination is", or similar structures, then it is a destination city name. If the proper name is preceded by "name is" or similar structures, then it is a name of the passenger.
4. The structure is an expression of date if it is in any of these forms: "first" (1~9), "eleventh" (10~19), "twentieth" (20 & 30), and "twenty first" (21~29, 31).
5. The structure is an expression of year if it is in any of these forms: "two thousand (and)" + "one" (1~9) / "eleven" (10~19) / "twenty" (20, 30, …, 90) / "twenty one" (21~29, 31~39, …, 91~99); or "twenty" + "oh one" (1~9) / "eleven" (10~19) / "twenty" (20, 30, …, 90) / "twenty one" (21~29, 31~39, …, 91~99).

6. The structure of month + date + year is usually an outbound departure date. However, if the word “return” is nearby then it is a return date. Also, if the whole utterance is too short, it may not contain information to decide whether it is an outbound date or return date, and thus the property of this time information cannot and should not be decided.

After I located these information chunks, I extract them from the text and transform them into structured information. I transform all date related expression into numeric representation using the knowledge of English number and month representations. I also transform chunks of yes / no answers, travel classes and roundtrip / one way trip words to standardized representations.

## 2.3.Method 2 – Machine Learning (CRF)

When I was writing the code of my rule based information extraction module, I tried to make my code as general as possible to handle many different forms of expressions. However, I realized that this is very hard since natural languages have so many different forms and variations of expressions. So, I started to try machine learning. I used conditional random fields (CRF) to do sequential labeling, in order to recognize word chunks of information, and then extract the information in the chunks.

For sequential labeling, I used python-crfsuite to do it. I use the following labels to chunk the utterance. The “B-” prefix indicates a beginning of a chunk and the “I-” prefix indicates other positions in a chunk.

| | |
|---|---|
| B-city-ori, I-city-ori | names of origin cities |
| B-city-des, I-city-des | names of destination cities |
| B-personname, I-personname | person names |
| month-out | outbound flight month |
| month-ret | return flight month |
| month | month, not provided to be outbound or return month |
| B-date-out, I-date-out | outbound flight date |
| B-date-ret, I-date-ret | return flight date |
| B-date, I-date | date, not provided to be outbound or return date |
| B-year-out, I-year-out | outbound flight year |
| B-year-ret, I-year-ret | return flight year |
| B-year, I-year | year, not provided to be outbound or return year |
| Yes | yes words |
| No | No words |
| B-class, I-class | Travel class (economy class \| first class …) |
| B-RoO, I-RoO | “roundtrip” or “one way trip” |
| the-out | “the” preceding outbound flight date (like in “the sixteenth”) |
| the-ret | “the” preceding return flight date |
| the | “the” preceding a date not provided to be outbound / return |
| O | Other words |

I used the grammar listed before to generate my training and testing sets of utterances, along with the labeling and structured information contained in each utterance. An example of a typical utterance and its labeling and structured information is as follows.

| |
|---|
| Utterance: 'i want to book an air ticket from hong kong to london departing on march twenty ninth twenty twenty two'<br>Labeling: ['O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'B-city-ori', 'I-city-ori', 'O', 'B-city-des', 'O', 'O', 'month-out', 'B-date-out', 'I-date-out', 'B-year-out', 'I-year-out', 'I-year-out'] |

```
Structured information:
[('=', 'OriginCity', 'hong kong'),
('=', 'DestinationCity', 'london'),
('=', 'OutDepartureTimeMonth', 3),
('=', 'OutDepartureTimeDate', 29),
('=', 'OutDepartureTimeYear', 2022)]
```

I use generated utterances like this to train and test my CRF sequential labeling model, I also use them to test the final NLU module.

I use the following features to train my CRF model and make it do prediction.

```
The word itself;
The word's previous word;
The word's next word;
Whether this word is the first word in the utterance;
Whether this word is the last word in the utterance;
Distance to the word "departure" in this utterance (if present), set to be 100 when not present;
Distance to the word "return" in this utterance (if present), set to be 100 when not present;
Distance to the word "origin" in this utterance (if present), set to be 100 when not present;
Distance to the word "destination" in this utterance (if present), set to be 100 when not present;
Distance to the word "from" in this utterance (if present), set to be 100 when not present;
Distance to the word "to" in this utterance (if present), set to be 100 when not present;
Distance to the beginning of the utterance;
Distance to the end of the utterance;
Whether it is a month name;
Whether it is a cardinal number of 1~9;
Whether it is a cardinal number of 10~19;
Whether it is a cardinal number of 20, 30, …, 90;
Whether it is an ordinal number of 1~9;
Whether it is an ordinal number of 10~19;
Whether it is an ordinal number of 20, 30, …, 90;
```

After being able to label all the words in an utterance, it's easy to extract the information chunks in the utterance, by merely following the "B-" and "I-" indicators. After I extract the information chunks, I examine them to see if they are indeed valid representations of the corresponding information, and if so, transform them to standardized representation. The transforming process is the same as the rule based version's, transforming all date related expression into numeric representation using the knowledge of English number and month representations, and also transforming chunks of yes / no answers, travel classes and roundtrip / one way trip words to standardized representations.

## 2.4.Evaluation

I used the grammar listed before to generate my testing set. I use the generated data to test both my rule based and my machine learning versions of NLU module.

Since the rule based version of NLU module is hand-crafted according to the same grammar specification I use to generate my testing set, the accuracy of it should be 100%, otherwise there are bugs. After I removed all the bugs, I successfully achieved the accuracy of 100% for the rule based version of NLU module.

It would be more interesting to test the machine learning based version of my code, since any machine learning technique cannot guarantee an accuracy of 100%. However, I found out that if I set the size of generated training set to 10000 and then train the model, the model would indeed achieve an accuracy of 100% on a different testing set. As a result, the extracted information also has 100% accuracy and 100% recall. This means that my implementation is successful.

When I lower the size of generated training set to 1000, the accuracy reduced to below 100%. The detailed labeling testing result is as follows (testing set size 10000):

```
Whole utterance accuracy: 0.9973
Overall labeling accuracy: 0.9993727760689947
Accuracy = 1.0000, Recall = 1.0000 for B-RoO
Accuracy = 1.0000, Recall = 0.9993 for B-city-des
Accuracy = 1.0000, Recall = 1.0000 for B-city-ori
Accuracy = 1.0000, Recall = 1.0000 for B-class
Accuracy = 1.0000, Recall = 0.9960 for B-date
Accuracy = 1.0000, Recall = 1.0000 for B-date-out
Accuracy = 0.9950, Recall = 0.9950 for B-date-ret
Accuracy = 0.9814, Recall = 1.0000 for B-personname
Accuracy = 1.0000, Recall = 0.9356 for B-year
Accuracy = 1.0000, Recall = 1.0000 for B-year-out
Accuracy = 0.9858, Recall = 0.9811 for B-year-ret
Accuracy = 1.0000, Recall = 1.0000 for I-RoO
Accuracy = 1.0000, Recall = 0.9972 for I-city-des
Accuracy = 1.0000, Recall = 1.0000 for I-city-ori
Accuracy = 1.0000, Recall = 1.0000 for I-class
Accuracy = 1.0000, Recall = 1.0000 for I-date
Accuracy = 1.0000, Recall = 1.0000 for I-date-out
Accuracy = 1.0000, Recall = 0.9722 for I-date-ret
Accuracy = 0.9814, Recall = 1.0000 for I-personname
Accuracy = 1.0000, Recall = 0.9713 for I-year
Accuracy = 1.0000, Recall = 1.0000 for I-year-out
Accuracy = 0.9903, Recall = 0.9961 for I-year-ret
Accuracy = 1.0000, Recall = 1.0000 for No
Accuracy = 0.9999, Recall = 1.0000 for O
Accuracy = 1.0000, Recall = 1.0000 for Yes
Accuracy = 1.0000, Recall = 0.9960 for month
Accuracy = 1.0000, Recall = 1.0000 for month-out
Accuracy = 1.0000, Recall = 1.0000 for month-ret
Accuracy = 1.0000, Recall = 1.0000 for the
Accuracy = 1.0000, Recall = 1.0000 for the-out
Accuracy = 0.9899, Recall = 1.0000 for the-ret
```

The final accuracy and recall of extracted information is as follows:

```
Whole utterance accuracy: 0.9973
Overall info accuracy: 0.9991674828599413
Overall info recall: 0.998434059212136
Accuracy = 1.0000, Recall = 1.0000 for =Class
Accuracy = 1.0000, Recall = 0.9960 for =DepartureTimeDate
Accuracy = 1.0000, Recall = 0.9960 for =DepartureTimeMonth
Accuracy = 1.0000, Recall = 0.9356 for =DepartureTimeYear
```

```
Accuracy = 1.0000, Recall = 0.9993 for =DestinationCity
Accuracy = 0.9814, Recall = 1.0000 for =MyName
Accuracy = 1.0000, Recall = 1.0000 for =OriginCity
Accuracy = 1.0000, Recall = 1.0000 for =OutDepartureTimeDate
Accuracy = 1.0000, Recall = 1.0000 for =OutDepartureTimeMonth
Accuracy = 1.0000, Recall = 1.0000 for =OutDepartureTimeYear
Accuracy = 1.0000, Recall = 0.9851 for =RetDepartureTimeDate
Accuracy = 1.0000, Recall = 1.0000 for =RetDepartureTimeMonth
Accuracy = 1.0000, Recall = 0.9811 for =RetDepartureTimeYear
Accuracy = 1.0000, Recall = 1.0000 for =RoundTrip
Accuracy = 1.0000, Recall = 1.0000 for False
Accuracy = 1.0000, Recall = 1.0000 for True
```

When I reduce the size of generated training set to 100, the accuracy further reduced (test set size 10000):

```
Whole utterance accuracy: 0.7565
Overall labeling accuracy: 0.9337434412882215
Accuracy = 1.0000, Recall = 0.6145 for B-RoO
Accuracy = 0.9646, Recall = 0.9036 for B-city-des
Accuracy = 0.9724, Recall = 0.9636 for B-city-ori
Accuracy = 1.0000, Recall = 1.0000 for B-class
Accuracy = 0.0000, Recall = 0.0000 for B-date
Accuracy = 0.9710, Recall = 0.9247 for B-date-out
Accuracy = 0.5302, Recall = 0.3688 for B-date-ret
Accuracy = 0.5312, Recall = 1.0000 for B-personname
Accuracy = NA    , Recall = 0.0000 for B-year
Accuracy = 0.8469, Recall = 0.7105 for B-year-out
Accuracy = 0.7022, Recall = 0.5896 for B-year-ret
Accuracy = 0.9770, Recall = 1.0000 for I-RoO
Accuracy = 1.0000, Recall = 0.5460 for I-city-des
Accuracy = 1.0000, Recall = 0.8968 for I-city-ori
Accuracy = 1.0000, Recall = 1.0000 for I-class
Accuracy = NA    , Recall = 0.0000 for I-date
Accuracy = 0.9963, Recall = 0.8280 for I-date-out
Accuracy = 0.1754, Recall = 0.2569 for I-date-ret
Accuracy = 0.5312, Recall = 1.0000 for I-personname
Accuracy = 1.0000, Recall = 0.0880 for I-year
Accuracy = 0.8276, Recall = 0.9429 for I-year-out
Accuracy = 0.8436, Recall = 0.6451 for I-year-ret
Accuracy = 0.8264, Recall = 0.9575 for No
Accuracy = 0.9610, Recall = 0.9913 for O
Accuracy = 0.9446, Recall = 1.0000 for Yes
Accuracy = 1.0000, Recall = 0.2798 for month
Accuracy = 0.9928, Recall = 0.9689 for month-out
Accuracy = 1.0000, Recall = 0.9109 for month-ret
Accuracy = NA    , Recall = 0.0000 for the
Accuracy = 0.9889, Recall = 0.9780 for the-out
Accuracy = 0.6320, Recall = 0.8020 for the-ret
```

The final accuracy and recall of extracted information is as follows:

```
Whole utterance accuracy: 0.7565
```

```
Overall info accuracy: 0.9130434782608695
Overall info recall: 0.8354783459750428
Accuracy = 1.0000, Recall = 1.0000 for =Class
Accuracy = NA    , Recall = 0.0000 for =DepartureTimeDate
Accuracy = 1.0000, Recall = 0.2798 for =DepartureTimeMonth
Accuracy = NA    , Recall = 0.0000 for =DepartureTimeYear
Accuracy = 0.9171, Recall = 0.8592 for =DestinationCity
Accuracy = 0.5312, Recall = 1.0000 for =MyName
Accuracy = 0.9724, Recall = 0.9636 for =OriginCity
Accuracy = 1.0000, Recall = 0.8886 for =OutDepartureTimeDate
Accuracy = 1.0000, Recall = 0.9689 for =OutDepartureTimeMonth
Accuracy = 1.0000, Recall = 0.6442 for =OutDepartureTimeYear
Accuracy = 0.6258, Recall = 0.2525 for =RetDepartureTimeDate
Accuracy = 1.0000, Recall = 0.9109 for =RetDepartureTimeMonth
Accuracy = 1.0000, Recall = 0.2877 for =RetDepartureTimeYear
Accuracy = 1.0000, Recall = 0.6145 for =RoundTrip
Accuracy = 0.8264, Recall = 0.9575 for False
Accuracy = 0.9446, Recall = 1.0000 for True
```

According to the test results above, my NLU module is indeed functioning as expected. For the machine learning version of NLU module, we can see that the accuracy and recall change as the size of training set changes. When the training set is of size 10000, it achieves 100% accuracy. However, when the training set is merely 100, its accuracy is not so good. So, it's better to have a bigger training set.

# 3. Other Project Work

Firstly, I did some work to assist with my primary responsibility of building a NLU module.

The first work I did is to write code generating training and testing data based on the grammar listed before. The data generated include not only the utterances, but also the labeling and structured information of each utterance. The amount of work of this is comparable to the amount of work of the rule based version of my NLU module.

The second work I did is to write code testing my NLU modules, and generating detailed accuracy and recall report as shown above. For the testing of sequential labeling result, I calculated sentence accuracy, label accuracy, and accuracy and recall for each label. For the testing of the final information extraction result, I calculated sentence accuracy, accuracy and recall for all information types, and accuracy and recall for each information types.

I also participated in some other work of my group, including:

1. Define the goal of our project and design the overall structure of our system;
2. Test and debug our overall system;
3. Help prepare the presentation.

# 4. Online Resources

[1] python-crfsuite's documentation. I used python-crfsuite to implement CRF. http://python-crfsuite.readthedocs.io/en/latest/

[2] python-crfsuite's example. I used python-crfsuite to implement CRF. http://nbviewer.jupyter.org/github/tpeng/python-crfsuite/blob/master/examples/CoNLL%202002.ipynb

[3] CRFsuite - Tutorial on Chunking Task. I read this webpage for the use of sequential labeling for NLU. http://www.chokkan.org/software/crfsuite/tutorial.html

[4] HTK speech recognition toolkit. My teammate used this toolkit to build speech recognizer. http://htk.eng.cam.ac.uk/

# 5. References

I don't have other references except the online resources listed above.