CMPE 492 Senior Project II

Plan My Day

Final Report

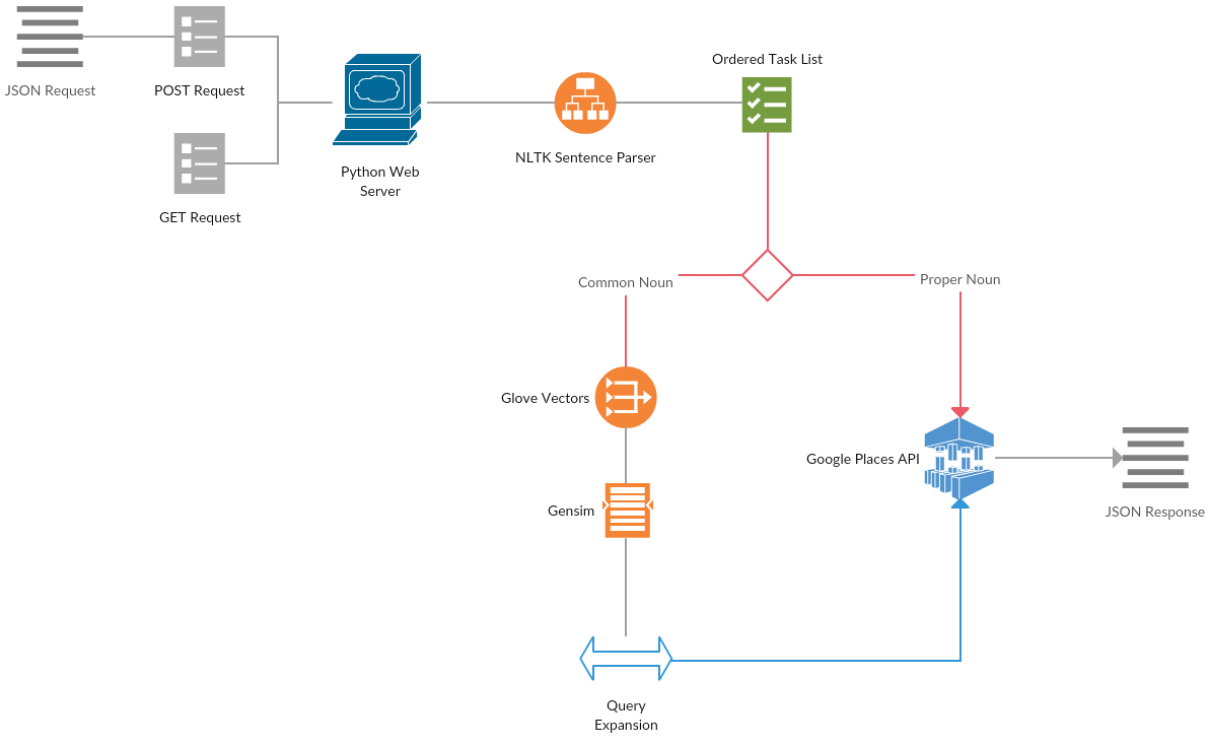
Project Group Members: Nehir Yiğit Karahan & Gökalp Fırat

Supervisor: TAYFUN KÜÇÜKYILMAZ

May 15, 2019

Contents

1. Final Architecture and Design



The figure above represents the final architecture and design of our system. The system starts with "POST" or "GET" request sent to our Python web server. If the request type is "POST" it needs to be content type of JSON and includes a JSON body to send. The purpose and the result of both types are identical the difference is "GET" requests had a character limit which may restrict applications. So the request type depends on developer usage. After the request arrived at our web server the text input will be processed at our NLTK sentence parsing algorithm. Our sentence parser returns an ordered task list depending on text's grammar. This list will be iterated on the server depending on the task's grammar; it will be either sent for a query expansion or directly sent to Google Places API. If the task needs a query expansion it will use glove vectors to find similarity from place type list which is obtained from the Places API. This query expansion leads to get best results for the task. Finally, after all requests are done the server will return the JSON response.

2. Test Results

In this part, we will be discussing the test result in detail. Questions of which tests are passed and failed, the reasons for the failure will be answered.

First of all, we want to remind our test cases.

ID	Test Case	Start Date	End Date
1	Register/Login	05.02.2019	05.02.2019
2	UI Functionality	02.02.2019	04.02.2019
3	Usage of NLTK	01.04.2019	01.04.2019
4	Voice Recognition API	15.04.2019	16.04.2019
5	Sending Request to Python Server From the Android Client and Managing the Response	25.04.2019	-
6	Google Map API With the Response(Parsed Locations)	-	25.04.2019
7	Gensim/Glove Train	01.05.2019	05.05.2019
8	Testing the scenarios all in one breath / Running the whole system	05.05.2019	-
9	Secondary Tests (Section 4)	-	-

Figure 1: Test Plan Report (Sec. 5 - Test Schedule)

TEST LEVEL	Project Team
Unit Testing	P
Integration Testing	P
Regression Testing	P
Subsystem Testing	P
Security Testing	S
Performance Testing	S
User Acceptance Testing	S

Figure 2: Test Plan Report (Sec. 4 - Test Strategy)

As another reminder, secondary tests were including security testing, performance testing, user acceptance testing. The project has not in the state of complete. Since we are still implementing some parts, security tests and user acceptance tests are not done yet.

2.1 Performance Test

For the performance testing part, we measured the execution time, which starts when the request is sent and ends when the response is ready to send, like 6-7 seconds for 1 request at a time. In our test, we used a sentence which includes total of 7 location and object:

```
"I will buy some keshar cheese or milk from migros or ikea and I will eat hamburger or tea and I will buy meat"
```

During this process, there are serious steps which we discussed in the previous sections and executions times for each is given below;

- Usage of nltk

- Usage gensim/glove part
- Preparing a json as a response

For multiple requests at a time, we may increase the performance by having a stronger server.

2.2 Primary Tests

Primary tests includes register/login, UI functionality, usage of nltk, voice recognition API, google map API for marking the locations, gensin/glove.

- Register/Login
  - We tried to log in with any google account(Gmail) and the test is passed.
- Usage of NLTK
  - We used lots of input test case sentences in our python server and observe the process.
  - We checked if every sentences are parsed into expected meaningful parts and it is working well. The test is passed.
- Voice Recognition API
  - We tried Google's API on an Android device by just speaking and we observed that it sometimes fails and we cannot take the risk so for us, it failed.
  - To fix this, we decided to let our users edit the text after they speak.
- Google Map API
  - We used this API on our server-side and we used longitude and latitude data set for a certain location as an input for the API and it found and marked that place. The test is passed.
- Gensim/Glove Train
  - We used this for query expansion method(Sec. NO), and we gave “hamburger” food as an input and we observed the expected outcome which is “hamburger food” in return. So, this test is passed.
- Testing the whole running system
  - Since the implementation is not completed yet, we delayed this part. So the test result is unknown yet.

2.3 Potential Enhancements For The Future

- We can buy a stronger server in order to support more request at a time and reduce the bottleneck to improve the overall performance.
- With time, we can add new futures accordingly to the user feedback.
- We can run tests to find the security vulnerabilities.
- We can work on Android application’s UI part.
- After completed the tasks above, we can put our application and API to the market.

3. Engineering Solutions

Our main process on the server can be mentioned as;

Nltk parses the sentence into meaningful parts and understands and detects the locations and objects(anything the user wants to have) and ordering them accordingly to the user’s input sentence’s structure and returns a list, then the gensim/glove part gets the part of this data set as an input and makes query expansion for that part and finds the relative places for the objects and picks those locations’ coordinates and send those to the Google Places API. And the other part which the gensim didnt take will be sent to the Google places API directly.

After those steps, the server prepares a Json which includes these coordinates and their relative data (name of the place, name of the object, distance between the user) and returns this Json by the response to the client.

Our service's architectural design is REST. The concept of RESTful services is the notion of resources. The clients send request to these URIs using the methods defined by the HTTP protocol. We have been using GET and POST methods in our application. The characteristics of a REST system are defined by six design rules:

1) Client-Server: Separation between server that offers the service and client that consumes it.
2) Stateless: Every request from a client must contain all the requirements of the server to carry out request.
3) Cacheable: The server must indicate to the client if requests can be cached or not.
4) Layered System: Communication between a client and a server should be standardized in such a way that allows intermediaries to respond to requests instead of the end server, without the client having to do anything different.
5) Uniform Interface: The method of communication between a client and a server must be uniform.
6) Code on demand: Servers can provide executable code or scripts for clients to execute in their context. This constraint is the only one that is optional.

The impact of our engineering solution in global context can be analyzed as usability. Any devices with a network connection can send HTTP requests; therefore our service can be used by a lot of devices. To talk about societal context that the service can be used by blind people with a combination of voice recognition systems. That is mostly depending on voice recognition systems because this technology is still at the beginning of its development. The blind people can use it to plan their day by just telling the text to our server and the rest of it will be handled by server. There may be some benefits for environments too such as decreased usage of maps or lists for plans that will decrease usage of paper proportionally.

3.1 NLTK

Firstly, nltk is a library and it has some default settings.

```
tagged = nltk.pos_tag(nltk.word_tokenize(text))
```

In this example, the text is the input. Using nltk library it tags the words in the sentence. For example, if our text is:

```
"I will buy milk and bread"
```

- Then the tagged version of this sentence is:

```
[       ('I', 'PRP'), ('will', 'MD'), ('buy', 'VB'), ('milk', 'NN'),
        ('and', 'CC'), ('bread', 'NN')                                ]
```

This is default behavior of nltk and these tags have meanings, for example, NN means that "milk" is a noun. VB means that "buy" is a verb. There is a list of these tags and their meanings provided in the nltk's website.

Our aim was to understand the user's plan by using this library. Of course, cases were not this easy always. Example:

```
“I will buy some keshar cheese or milk from migros or ikea and I will eat
hamburger or tea and I will buy meat”
```

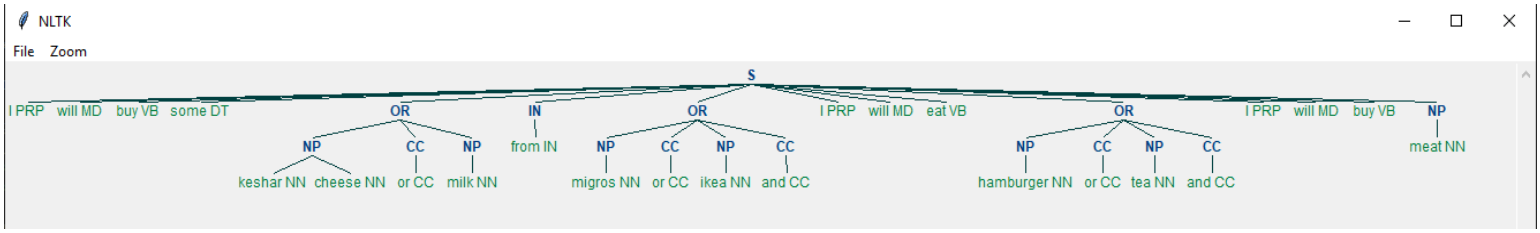
In this case, we put some priorities on the tags. If there is a word with the tag of “IN”, which includes these words: “at”, “in”, “from”, “on”, it means thats a special location for user. In the example above, its migros or ikea, so user will buy some stuff but we do not need to pick those because we know where the user actually wants to go. And getting migros and ikea was another challenging part because figuring that there is “from” in the sentence does not make picking “migros” or “ikea” easy by any means.

To have those two keywords(migros and ikea), we used the chunking method which nltk provides. With this method, we could write a regular expression to combine the words together accordingly to their tags. For example;

```
grammar = """NNP:{<NNP><NN>*<NNP>*}
NP:{<NN>*}
IN:{<IN>}
"""
chunkparser = nltk.RegexpParser(grammar)
mytree = chunkparser.parse(tagged)
```

“mytree” is a tree and it has several subtrees accordingly to the chunks which we described in the grammar. So, NNP, NP and IN are the labels which are created by the rules. By checking the existence of these labels in a sentence, we can have some serious ideas about the user’s input.

This is how this tree looks like accordingly to the different grammar.



So for this case, since the “IN” label has the highest priority we are focusing on the migros or ikea part by skipping the keshar cheese and milk. Then we are getting into the “OR” label and finding “NP” labels by observing their conjunctions. Because if there is “and” between nouns, which means the user will go to the migros and ikea both, but if there is or between them, we need to pick the closest one for the user.

The same logic is applied to the “hamburger or tea” part too. After all, the list of tasks in order is below;

['migros or ikea', 'hamburger or tea', 'meat']

If there is “or” word, in the “OR” label, we are adding it between the “NP” labels (migros or ikea), but if there is “and” word, we are separating our “NP” labels and putting them into the array as different elements. So, if we write “and” instead of “or” then the returned list is;

['migros', 'ikea', 'hamburger', 'tea', 'meat']

As we treat the sentences differently for some cases, we are changing our chunk parser accordingly to these cases.

These cases can be listed as below:

- If there is a word with the tag of “IN”
- If there is special name which are tagged with the tag of “NNP”
- If there is common nouns which are taged with the tag of “NP”
- If there is a label with the name of “OR”
- If any two of them are together in the sentence

These topics will be discussed in the presentation with all the details.

4. Issues with the topic

4.1 NLTK

If the input sentence is:

“I will Visit Ted university or Bilkent university and I will eat meat at burgerking and I will eat meat”

Then the result list has “burgerking, meat” in it. It won’t pick the two universities where the user wants to visit. The reason is the priority, if there is a word with the tag of “IN”, we are taking the rest of the sentence:

“at burgerking and I will eat meat”

Then parsing it for the “and” word, and sending these parsed parts to the method recursively.

“at burgerking”

“I will eat meat”

This is how it works better for many cases. Because otherwise, we could not support the better case, for example;

“I will buy milk, cheese, water, and french fries from migros”

For this case, if we split the sentence accordingly to the “and” before checking if it has a word with the tag of “IN”;

“I will buy milk, cheese, water”

“french fries from migros”

Then the list would be;

['milk', 'cheese', 'migros']

And we could needlessly pick the milk and cheese when just picking the migros was enough.

So, this is not actually a problem or bug. We know the reason for this problem and we know the solution so we are considering this as a trade-off.

Additionally, if the user enters this sentence as;

- "I will Visit Ted university or Bilkent university THEN/AFTER/BEFORE I will eat meat at burgerking and I will eat meat"
- "I will Visit Ted university or Bilkent University**.** I will eat meat at burgerking and I will eat meat"

There will be no problem.

5. New Tools and Technologies

5.1 Glove & Gensim

There are two learning algorithm for word vector representation these are word2vec and Glove. In word2vec, Skipgram models try to capture co-occurrence one window at a time. In Glove it tries to capture the counts of overall statistics how often it appears. Glove basically counts how frequently a word appears in a context. Depending on our needs Glove is the best choice. We need to apply query expansion for common nouns. Query expansion is basically adding another text at the end of the text. Google Place API allows us to do that for given keyword. We can expand our query with the place types allowed by Google. This process is giving us the best results for common noun place search. But first we need to get most similar place type to the noun. For this purpose doing "Cosines Distance Formula" will give us the similarity we need. We used Gensim library to use this algorithms easily. Gensim is designed to process raw, unstructured digital texts. The algorithms in Gensim, such as Word2Vec, FastText, Latent Semantic Analysis (LSI, LSA, see LsiModel), Latent Dirichlet Allocation (LDA, see LdaModel) etc, automatically discover the semantic structure of documents by examining statistical co-occurrence patterns within a corpus of training documents.

5.2 Flask

Flask is a web framework. This means flask provides you with tools, libraries and technologies that allow you to build a web application. This web application can be some web pages, a blog, a wiki or go as big as a web-based calendar application or a commercial website. We had used this framework for our API's homepage and request/responses.

5.3 Googletrans

Googletrans is a free and unlimited python library that implemented Google Translate API. This uses the Google Translate Ajax API to make calls to such methods as detect and translate. We need this library because even our service text input should be in English the client's location may be not in UK or USA. Therefore common nouns should be translated to the client's language before making requests to Google Place API.

6. References

- https://radimrehurek.com/gensim/intro.html
- https://www.quora.com/How-is-GloVe-different-from-word2vec
- https://blog.miguelgrinberg.com/post/designing-a-restful-api-with-python-and-flask