CS224N Final Project

Geo-location Route Recognition

June 2, 2010

Yingjie (Roger) Zheng (yzheng@stanford.edu)
Philip (Tony) Hairr (thairr@stanford.edu)

Introduction

In this project, we explored the methods to identify possible travel routes described in a text paragraph. We built a system to retrieve such information from free web text, geocoded and annotated the travel directions on a map. We also developed methods to evaluate the performance of the system, investigated various problems with our solutions and proposed future efforts in this domain.

Objective

For example, below is a paragraph adapted from the Lonely Planet web site (http://www.lonelyplanet.com/usa/california/travel-tips-and-articles/42/2500):

"Start at the bottom of the state map, just outside San Diego, where the pretty peninsular beach town of Coronado is connected to the mainland by a long, narrow spit of sand called the Silver Strand. You can't help but feel a thrill while speeding over the 2-mile-long Coronado Bridge as it snakes its way across San Diego Bay.
Further along the track, head down toward the sea and Torrey Pines State Natural Reserve, harboring the rarest pines in North America and a lagoon frequented by migrating seabirds. ... On to Long Beach and the often overlooked Long Beach Museum of Art, sitting pretty on an ocean bluff. ..."

We would like that our system can generate a list of locations that represents the direction of the route and plot the route on a map.

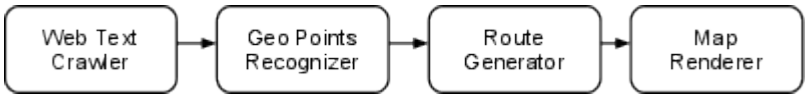
Coronado -> Coronado Bridge -> Torrey Pines State Natural Reserve -> Long Beach -> Long Beach Museum of Art

Related Work

Our work is related to the intersection of natural language processing and geographic information retrieval. [Adam, Jurafsky, 2010] presented a system that learns to follow navigational natural language directions by apprenticeship learning through a map paired with English descriptions. [Davidov, Rappoport, 2009] introduced a framework to discover connectivity and transport routes in a certain region based on a small set of seeding descriptions. [Leidner, et. al, 2003] described details of how to disambiguate geographical names and how to ground the named entity on the map.

General Architecture

The system we aimed to build is composed of 4 parts working in a pipeline.



Geographical Entity Recognition

The first task in the pipeline is to identify geographical points described in the text. One approach is to search location names according to a location dictionary. Another approach is to use a machine learned NER (Named Entity Recognition) system which identifies place names based on various text features. We chose an NER system in our solution, because an NER system, once properly trained, is more capable of identifying location names that are not in the dictionary thus increasing the precision and recall. We used Stanford NER software package for this task. Stanford NER package implements linear chain Conditional Random Field (CRF) sequence models with 3 class (PERSON, ORGANIZATION, LOCATION) named entity recognizers for English.

For geographical points recognition, we found that people tend to use location names and organization names to indicate the place they go. For example,

"After I visited *Stanford University*, I walked on the street in downtown Palo Alto."
"At long last, the highway reaches *Redwood National & State Parks.*

Here, both Stanford University and Redwood National & State Parks are identified as ORGANIZATION in Stanford NER system, but they are crucial in building up the routes. So we used both ORGANIZATION and LOCATION entities in our system.

Route Disambiguation

We assume that in the input text, the order of occurrence of geographical points will generally correspond to their order of appearance on the route, and we assume that our input text will talk about one route, although this assumption is challenged in our actual test as discussed below. We focused on increasing the correctness of route direction with one sentence, because in English, the navigational expression can be placed flexibly within a sentence. For example, the relation <FROM A TO B> expressed by preposition "from" and "to" fits in the following case:

1. I flew to Shanghai from San Francisco.
2. I flew from San Francisco to Shanghai.

Both mean a route <FROM Shanghai TO San Francisco>. It is obvious that another prepositional expression such as "via" would add the complexity of this example. From the observation that most route information can be described in a <FROM A VIA B TO C> relation, our approach to solve the ambiguity is to fit every sentence containing location information to the form <FROM A VIA B TO C>. The problem then transfers to determining the correct place names for A, B and C. We can use syntactic information obtained from a NLP parser to determine the dependencies between prepositions and place name.

In our system, we used Stanford Parser to do the job. Stanford Parser is a highly lexicalized PCFG parser, in addition to Penn TreeBank-style output, we utilized its word dependency output as our tool to understand the relationships between words.

For example, for the sentence *"I took a bus to ride from Chicago to Sacramento."* the parser would output the following dependency in the form of *relationship(governor, dependent)*:

```
nsubj(took-2, I-1)
det(bus-4, a-3)
dobj(took-2, bus-4)
aux(ride-6, to-5)
infmod(bus-4, ride-6)
prep(ride-6, from-7)
pobj(from-7, Chicago-8)
prep(ride-6, to-9)
pobj(to-9, Sacramento-10)
```

For pobj(to-9, Sacramento-10) we know that the 10th word of the sentence, "Sacramento" depends on the ninth word "to". "Sacramento" is the prepositional object of to. By looking at the pobj relationships of (to, Sacramento), and (from, Chicago) we can extract the A and C in <FROM A VIA B TO C> model, this is same for B, VIA as well.

To make this rule more robust, we observed that "from", "via" and "to" are not the only prepositions that possibly represent relationship of <FROM A VIA B TO C>. Such as "following San Diego, I went to Long Beach". We hand categorized English prepositions to 3 classes that has the similar meanings as "from", "to" and "via" by matching the governor part of pobj(preposition, PLACE NAME) with following regular expressions.

```
String fromCases = "from|after|following|out";
String viaCases =
"via|across|along|alongside|around|between|including|next|over|past|round|through";
String toCases ="to|before|into|till|toward|towards|until|upon";
```

The matching looks like following in our source code.

```
if(relationship.matches("pobj")){
if (gov.toLowerCase().matches(fromCases))  return "from";
if (gov.toLowerCase().matches(viaCases))     return "via";
if (gov.toLowerCase().matches(toCases))     return "to";
```

Apart from prepositions, we observed that transitive verbs can also indicate a start or stop action in the route description. For example, I *leave* San Diego for Shanghai this morning. This dependent relation is represented by dobj(governor, dependent). Currently, we only categorize this kind of relation as "from". Further investigation need to be done on more text samples.

Connecting Geographical Entity Recognition to Route Disambiguation

Some place names are represented by more than one word, these names needs to be chunked together after NER part in order for the parser to generate correct dependency information on the whole place name. We put an under bar "_" between terms to achieve this. After parsing, the words are unchuncked to restore their original representation.

NER Stage: San Francisco
Parsing Stage: San_Francisco
Geocoding and plotting stage: San Francisco

Grounding Geographical Entity

After obtaining a list of location names, we can geocode them into latitude and longitude coordinates for map display. Our system used Google Maps API web service to retrieve these coordinates and put them into an xml file. For the sake of simplicity, we

used an HTML file containing Javascript file to show the routes in data.xml in Google Map.

A challenge in this process is that different geographical locations can have same names. For example, San Diego Bay can mean a place in San Diego , California, while at the same time, it can mean "San Diego Bay, Aleutians East, AK". To resolve this from linguistic context, two minimality heuristic was proposed by Leidner, et al[4]. 1. Assuming that within a discourse, all place names refers to the same location. 2. If more than one place name mentioned in a span of text, the location interpretations are the set that give the smallest region.

In our case, since we couldn't obtain a list of alternative geographical coordinates from Google's service, we weren't able to implement these two heuristics. This will be our future work to find an alternative geocoding service allowing us to get different coordinates on one location name.

Preparing Test Data

Web Crawling and Text Extraction

For the subject text from which we extract routes, we chose the Thorn Tree travel forum at the Lonely Planet web site, because users of the site frequently ask how to get from one part of the world to another. We used a simple web search to get a list of documents using the "site:" search parameter in the query (site:http://www.lonelyplanet.com/thorntree). We then downloaded the first 300 result documents. Two of these documents were not useful in our study, so we limited the set of documents to the remaining 298. The actual text of each forum entry consists of a question and a series of answers. In order to focus specifically on that text, we used cues inside the HTML template of the site to tell us where the text of interest starts and stops, the extracted that text and removed any HTML it contained. Finally, for each document, combined the text into one long line, then split it on the expected final punctuation (period, question mark, exclamation mark) to divide up the text into one sentence per line.

Human Judgments

For each of the documents, we read the content ourselves and determined the main route being discussed in that forum thread, and listed the locations on that route, one location per line, in a file that corresponds to the original text file. Using this format we could conveniently compare our (assumed) correct answers with the output of our extraction experiments.

Evaluation Metrics

Score (based on minimum edit distance)

We calculate a score to attempt to quantify the difference between the program output and the test data, with an attempt at balance so that big differences in small files do not "count" too much. Using the Unix "diff" command (-i for case insensitive) we calculate the minimum edit distance between the files on a per-line basis. Inserted and deleted lines (locations) each count once; substitutions and moved lines each count twice as one insertion and one deletion. We divide the number of lines in the golden test file by the same number plus the edit distance to get a number between 0 and 1. If the edit distance is 0, the score is therefore 1 or perfect. As the edit distance grows larger, the score approaches 0.

Precision

We used the Unix "sort" and "join" commands to generate lists of unique places appearing in the test program output and the golden test data separately, then match them to find out how many locations appear in both, then calculated precision using the matching and total line counts. For example, if the program output contained 5 lines and the golden test data contained 6 lines, and 3 lines match, we divide the matching lines (3) by the total lines in the program output (5) to get 60% precision. In other words, out of 5 lines that were returned, 3 were correct. This does not take into account the order of locations in the correct route, and we ignored uppercase and lowercase differences.

Recall

Using the same Unix commands as for precision, we calculate recall by dividing the matching lines by the total lines in the golden test data. Given the same example as above, we would divide 3 matching lines by 6 lines in the golden data to get 50% recall. In other words, of the 6 lines that should have been returned, 3 actually were. Again, this does not take into account the correct order or character case.

Baseline Heuristic Approach

The baseline we use for performance comparison uses virtually no text understanding at all. The algorithm is to take the first line of every file (typically the question that started the thread) and output the first word following the last occurrence of the word "from" and the first word following the last occurrence of "to". This is actually an effective method in very simple cases, where the entire route consists of only the starting and stopping point and both are mentioned in the question, and the location names are single words that immediately follow a "from" or a "to".

Baseline Results

Score

The average score for the baseline was 0.588188. What this means is that on average, the routes (location lists) produced had almost as many edits as they had correct locations. Since deletions are counted as single edits, we can expect a score of 0.5 in any specific case where all of the locations were missed by the baseline script.

Precision

Precision in the baseline was on average 0.536913. Ignoring order, the baseline script was correct regarding about half of the locations in its output.

Recall

Recall for the baseline averaged 0.453926. Again ignoring order, the baseline found a little less that half of the locations that should have been extracted.

First Experiment: NER-only Approach

In this experiment, we only connected NER to our pipeline, without route disambiguations, i.e. if we have the input text:

I fly to San Diego from San Francisco.

The route will be San Diego -> San Francisco. We expect this to eliminate the issue of multi-word place names not being recognized in the baseline.

Interim Results

Score

The average score in this experiment was 0.110268. This means that there were about eight times as many edits counted in the output as there were correct locations. In particular these tend to be insertions due to entities being recognized that play no role in the main route in each context.

Precision

The average precision in this experiment was 0.162685. Precision also dropped because the entity extraction did not appear to take into account the usage of the entities in the sentence; therefore, named entities which were not on the route were be output, and entities which were on the route but were not recognized (possibly due to capitalization or word shape features).

Recall

The average recall for this experiment was 0.785805. Consistent with the remarks for precision, the NER-only approach output a fairly thorough (if often inappropriate) list of entities, so not as much was missed as when we ran the baseline script.

Second Experiment: NER with Parsing and Re-ordering

This experiment included full algorithms for NER and route disambiguation described above, i.e. for the following input text:

I fly to San Diego from San Francisco.

The route will be San Francisco -> San Diego. For input text:

I will go to Shanghai from San Francisco via Tokyo.

The route output will be San Francisco -> Tokyo -> Shanghai.

Interim Results

Score

The average score in this experiment was 0.258591. This means the edit distance has come down to 3 times the number of correct locations, from 8 in the previous experiment.

Precision

The average precision in this experiment was 0.272785. This is a significant improvement over the last experiment, but still only a quarter of locations output are correct.

Recall

The average recall in this experiment was 0.612416. Places that did not fit into the

explicitly defined grammatical relationships in the parser were dropped, and some of them should not have been.

Third Experiment: Using fewer sentences

Noting that the baseline heuristic approach is as successful as it is using only the first sentence (the subject) of each forum thread, and the first and second language-understanding-based approaches are hindered by information overload, we thought we should try analyzing fewer sentences. Often the main route being discussed is not in the subject, but may appear in the elaborated question section (e.g. users asking "Is this the right route?") or the first answer in the thread. For the next experiment, without modifying the core algorithm, we simply reduced the length of the input files to 5 lines (sentences) each. An example file follows:

1. `Kyrgyzstan to China Hi folks I am posting to find about travelling from Kyrgyzstan to China and onward travel from China.`
2. `I am hoping to travel from Kyrgyzstan to China next year My original plan was to cross the border overland from Kyrgyzstan to China but it sounds very complicated!`
3. `I was looking to take a 1 way flight from Bishkek to Urumqi and from Urumqi travel all the way across to Shanghai by train, from Shanghai I am hoping to travel down to Hong Kong by train However I am reading conflicting reports on-line about people having difficulty travelling from Kazakhstan or Kyrgyzstan into China without proof of onward travel Can anyone tell me if this is the case?`
4. `If this is the case could I still travel into Urumqi from Bishkek by flying and then maybe book a 1 way flight from Shanghai to Hong Kong but still take the train from Urumqi across to Shanghai with a few stops on route?`
5. `I plan to get my Chinese visa before I travel to Kyrgyzstan Cheers Lucas 1 Complicated?`

In the above case, the question is about a trip from Bishkek in Kyrgyzstan to Hong Kong in China, but the subject only mentions the country names, while the specific locations on the route begin on line 3 of the file. By line 5, each of the places on the route are mentioned multiple times, so entity extraction alone can score perfect recall. It is difficult to tell the author is describing several alternative routes just from syntactic informations we obtain.

Final Results

Score

The average score in this experiment was 0.438389. This is the largest improvement in score so far, among the language-understanding approaches.

Precision

The average precision in this experiment was 0.549765. Precision effectively doubled because the program was no longer returning as many incorrect locations, which was a function of the amount of text it was analyzing.

Recall

The average recall in this experiment was 0.602047. The above improvements in score and precision came at a modest cost of about 1% recall. This also reflects that the important locations were originally only appearing in the top portion of the text.

A graphical representation of "I first fly to Sydney from Shanghai, then to Singapore." generated by our system is shown here.

Conclusion, Future Work

These experiments demonstrate that in the domain of forum threads, a language-understanding approach can outperform a naive heuristic approach by taking advantage of some observations about sentence structure and an effective place-name entity extractor, focused on the first five sentences of the thread. However, outside of the achievements of these experiments there remains much to explore.

One failing of the current system is that in situations where the thread authors say the route explicitly, such as in, "I took this route: San Diego-San Jose-San Francisco", our parser-enabled method does not identify the route because it does not find cue words such as "from" and "to". Of course the baseline heuristic fails in this respect as well, but it seems within reach to treat list-delimiting symbols such as dashes and commas as cues for identifying routes.

A more complex failure occurs when, due to sentence structure, the route-indicating prepositions are located outside the clause that describes the path. For example, "After I visited San Diego, i went to San Jose, then San Francisco." The preposition "after" represents a "from" relationship, but it appears outside the verb phrase that contains the relevant place name. The clause "I visited San Diego" has "visited" as its root verb which attaches to "after". Using the parser in our project, the trace from candidate locations stops when it encounters a verb, which is too early. Further, "San Jose, then San Francisco" are both places that "I went", but San Jose is not the final destination; there is an implicit "from" relation indicating a path from San Jose to San Francisco that is also not captured by our current parser.

Improvements in each of our metrics could be achieved in our data set to the extent that these issues occur (which they do) and can be overcome. Further, a route extraction solution that is able to solve these problems for forum threads would also be applicable in a wider domain, to include blogs, books, and any other natural language media.

References

[1] Adam Vogel and Dan Jurafsky. 2010. Learning to Follow Navigational Directions. In *Proceedings of ACL.*

[2] Paul Clough, 2005, Extracting Metadata for Spatially-Aware Information Retrieval on the Internet, In the 2005 workshop on Geographic information retrieval

[3] Dmitry Davidov, Ari Rappoport. 2009. Geo-mining: Discovery of Road and Transport Networks Using Directional Patterns. In: EMNLP 2009

[4] Leidner, G. Sinclair, and B. Webber. 2003. Grounding spatial named entities for information extraction and question answering. In *Workshop on the Analysis of Geographic References, Edmonton, Alberta, Canada. NAACL-HLT*

Natural language processing (NLP) is one of the most important technologies of the information age. Understanding complex language utterances is also a crucial part of artificial intelligence. Applications of NLP are everywhere because people communicate most everything in language: web search, advertisement, emails, customer service, language translation, radiology reports, etc. There are a large variety of underlying tasks and machine learning models behind NLP applications. Recently, deep learning For Android : Modify AndroidManifest.xml located at app/App_Resources/Android/src/main/AndroidManifest.xml and insert this between tags. . For iOS: Add the following snippet at the top of main.js. import * as platform from 'platform'if (platform.isIOS) { GMSServices.provideAPIKey("PUT_API_KEY_HERE") }.