Error detection of text queries transcribed from voice input

STUDENTS: Aravind Narayanan, Zhuoran Zhou, Chingpo Lin

Problem Statement

- People use voice for in-car navigation systems. However, the system might have trouble processing what exactly the user says, depending on noise from surroundings as well as accents.
- The system will come up with a few strings for what the user could have said. For example, if the user said "university," then the system’s voice processor might hear two possibilities: “university” and “universe city”.
- Our goal is to rank these possible queries in order of likelihood of what the user actually meant.

Requirements

- Given a list of possible input strings, order a list of the same strings ordered correctly based on relevance.
- Eliminate errors from the input queries that were transcribed from speech (nonsensical phrase and misspelled words should be filtered out.)
- Our program should be organized into a Java library (and be able to be run through a JAR file).
- We need to create an Android test application in order to run the library.
- Finally, we need a full report and documentation detailing our entire implementation.

Implementation - Algorithm (Cont’d)

- Now, how do we rank input queries when given a training data set like this? We can rank phrases by their frequency, or how often they are mentioned.
- Our algorithm counts the frequency of every bigram (set of two words) in “local.txt”. Some examples of bigrams from the snippet above would be “golf club” and “club calabash”.
- When prepositions like “of” or “in” are encountered, we include the word after the preposition as well. For example, we count the word “city of detroit” instead of splitting it up into two bigrams. This is done to not give too much importance to common filler words.
- When full queries themselves are just a single word like "Gas" we just store that word alone.
- Our dictionary is stored as a text file called “word.txt”. Here is a snippet of the dictionary. The count of each phrase is shown below the phrase.

<table>
<thead>
<tr>
<th>Phrase</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>university</td>
<td>700</td>
</tr>
<tr>
<td>wayne state</td>
<td>54</td>
</tr>
<tr>
<td>state university</td>
<td>646</td>
</tr>
</tbody>
</table>

Once we have a dictionary of phrases, we can rank queries based on their score. If a query is longer than two words, its score is calculated by getting the sum of the counts of all of the bigrams in that query. For example, let us take the query “wayne state university”.

Rutgers 32,783 -96,764
direction to stadium on 79,193 -83,599
gandels golf club calabash 33,823 -75,619
gandels golf club calabash 33,823 -75,619
Sears 8,482 -25,978
Sears 8,482 -25,978
Sunw 3,805 -32,73
gas 3,865 -34,312
big main street astonhe us 42,321 -71,111
china 42,839 -91,609
cleanwater seeke shop 27,991 -92,73
inventor crossing 45,766 -108,581

Implementation - Library and Android App

- After algorithm implementation, we build a jar file as a library for ease of use in both Android development and command-line-execution.
- As shown below, calling the jar library through command line requires a quite a few lines of code every time.

未来工作

- The Android App should include a function to support the .jar file library so that it can save plenty of time by not re-loading the same dictionary whenever it runs.
- We need to improve our address checker, which checks whether queries are addresses or non-addresses. Currently, many edge cases are not handled.
- To simplify the user interface, we also built a simple Android App for our program.
- As shown below, the program takes a number of possible text queries as inputs and sorts them by their score.

Future Work

- Industry Mentors: Srinivasa Parvathareddy, Changzheng Jang, Akira Zhang, Kumar Maddali
- Faculty Advisors: Lillian Ratliff, Mari Ostendorf
- TA: Haobo Zhang

Acknowledgements