Knowledge Technologies, Semester 2 2014

Project 1: Approximate String Search for Geolocation of Tweets

Due:	5pm, Wed September 03, 2014
Submission mechanism:	To be described on the LMS.
Submission materials:	Source code, `README` (see below); Report (in PDF; also below)
Assessment criteria:	Critical Analysis; Soundness; Report Quality; Creativity.
Marks:	15% (undergraduate), 20% (masters)

Overview

For this project, we will be working with tweets from Twitter. Your objective is to develop software that identifies location names in these tweets.

The particular focus of this project is a critical analysis of methodologies. That is, you will describe what you have learned about the problem and can pass along to another person attempting it. Technical merit and good programming practice will not be of primary importance; impeccably-coded software with little insight will not receive a strong mark.

Resources

Given a set of target strings (in this case, the tweets) and a set of queries (US locations), you will find the best approximate matches to each query in the texts. The location names are therefore the "queries", and the tweets are the target texts. Please note that many tweets will not contain any location names at all, and many of the locations may not occur in any tweet text.

The directory `/home/subjects/comp90049/2014-sm2/project1/` on the new CIS servers (e.g., `nutmeg.eng.unimelb.edu.au` or `dimefox.eng.unimelb.edu.au`) contains the principal materials required for the project.

The tweets are available in the file `training_set_tweets.txt`. The format of this file is {`user_id (tab) tweet_id (tab) tweet_text`}.

The (United States) location names are available in the file `US.txt`. The format of this file is described in the file `readme.geonames.txt`.

Terms of Use

As part of the terms of use of Twitter, in using the data you agree to the following:

- You are strictly forbidden from redistributing (sharing) the dataset with others or reusing it for any purpose other than this project.

- You are strictly forbidden from reproducing documents in the document collection in any publication, other than in the form of isolated examples.

Additionally note that the document collection is a sub-sample of actual data posted to Twitter, without any filtering whatsoever. As such, the opinions expressed within the documents in no way express the official views of The University of Melbourne or any of its employees, and my using them does not constitute endorsement of the views expressed within. I recognise that some of you may find certain of the documents in bad taste and possibly insulting, but please look beyond this to the task at hand. The University of Melbourne accepts no responsibility for offence caused any content contained in the documents.

Task 1: Software

Task

Your code should broadly perform the following:

1. Preprocess the data files to extract the tweet texts, and the location names (the `asciiname` column of `US.txt`).
2. Preprocess the tweet texts to remove all non-alphabetic characters, other than spaces, and convert all letters to lower-case. The resulting text should have an alphabet of size 27.
3. Make use of suitable data structures (see below).
4. Process each query in turn (see below).
5. For each query, return the tweet ID (or IDs!) from the text that your system believes to be a plausible instance(s) of the query. Note that this set might be empty, and that there may be a great deal of subjectivity involved.

You may work with samples of either the query or the target datasets (or both) if you find working with the full data set too challenging, but please be sure that it is clear in your submission (`README` or report) how you created these samples.

Methods

There are a number of different methods for doing the matching. Students taking COMP30018 should aim to try one (or more) of these strategies; students taking COMP90049 should aim to try two (or more) of these strategies.

1. You may process the texts (and possibly the queries) into separate tokens ("words"). This isn't in the spirit of how string search has been presented in the subject, but is a legitimate strategy for solving the problem. If you use this approach, you should identify its advantages and disadvantages in your report. You might then:

   (a) Use a Global Edit Distance matching strategy for each token. You will need to choose appropriate values for the operation scores, and threshold sensibly. Looking at all of the tokens may take a long time.

(b) Use a N-gram Distance matching strategy for each token. Thresholding is the most difficult problem here. This is the fastest approach, but may give poorer results.

2. You may treat the texts as a monolithic entity (i.e., treat the full tweets file as a single text), and apply the approximate matching technique(s) to this directly. You might need to do some back-tracking (or use some other book-keeping strategy) to determine the tweet ID for a given match. If you use this approach, you should identify its advantages and disadvantages in your report. Using the documents as a single string, you might:

   (a) Build and search a trie. Be *extremely* careful with this. A simple implementation is all that is required, but memory requirements may grow much faster than you expect. Test thoroughly on small volumes of text (e.g., small numbers of tweets)! You will then need to implement search procedures allowing 1 or 2 mismatches. (Or 3 mismatches, if your tests show that this will complete in a reasonable length of time.) This will, in general, be the fastest solution, but the most difficult to implement.

   (b) Use edit distance with local alignment. Again, be careful to test on small volumes of data; you may find that you only have time to run a relatively small number of queries.
   (C code for a simple edit distance will be given in lectures. But this code does not show how to *trace-back* to find the alignment. Source code examples for trace-back can be readily found on the web; feel free to make use of them.)

3. These are not exhaustive, other strategies are possible as well. You may use external packages or supplied code if you wish; if you do so, the source of the package should be clearly indicated in your `README`, and your strategy should be briefly explained in the report.

You should also attempt to ascertain the effectiveness of your system. Take a look at the results for a few queries (by hand), and decide whether the tweet is relevant to the query (to the best of your ability). Choose a few examples to demonstrate your system's performance for your report.

Task 2: Report

The report will consist of about 750–1000 words (for students taking COMP30018) or 1000–1500 words (for students taking COMP90049) and should aim to discuss:

1. A basic description of the task and methods you explored;
2. Whether the method worked – use example queries to demonstrate the effectiveness of the methods you tried. What worked well? What went wrong?
3. Considerations for scalability (that is, performance and behaviour as the text size or query length grows), and what effect this might have on the results.
4. A discussion of challenges you faced in solving this task, and the suitability of the method for the task. How might you approach the task differently?

Note that we will be looking for evidence that you have thought about the task and have determined reasons for the performance of the method(s).

While 1000 words might seem long at first glance, it isn't: being concise will be invaluable, and overlong reports will be penalised. You will not have space to discuss the technical elements of your implementation; you should focus primarily on the knowledge you have gained in your experiments. Spending more time on your report will allow you to use the word limit more effectively.

Submission

Submission will entail two parts:

1. An archive which contains your code and a `README` file which briefly explains how to compile your submission on the CIS servers (preferably as a Makefile or `run.sh`) and the location and format of the output.

2. Your written report, as a single file, submitted in Portable Document Format (PDF).

Details of how to submit will be given on the LMS closer to the deadline.

Your software can be implemented in any programming language or combination of programming languages. There is no requirement that it runs on the CIS servers; please be considerate of other users if you do choose to run your system there — the task may be very memory-intensive.

If your report is submitted in any format other than PDF, we reserve the right to return it with a mark of 0.

Changes/Updates to the Project Specifications

If we require any (hopefully small-scale) changes or clarifications to the project specifications, they will be posted on the LMS. Any addendums will supersede information included in this document.

Academic Misconduct

For most people, collaboration will form a natural part of the undertaking of this project. However, it is still an individual task, and so reuse of code or excessive influence in algorithm choice and development will be considered cheating. We will be checking submissions for originality and will invoke the University's Academic Misconduct policy (`http://academichonesty.unimelb.edu.au/policy.html`) where inappropriate levels of collusion or plagiarism are deemed to have taken place.