T-61.5060 — Programming project

October 16, 2014

Due: Friday, Dec 5, 6pm

Instructions

The deliverables for the project assignment are: (1) a writeup; (2) all the source code and scripts that you will develop. More detailed instructions on the format of these deliverables is given below.

All deliverables should be returned by email to Polina Rozenshtein, at `polina.rozenshtein@aalto.fi`.

You can work on the project individually or in two-person teams. You can find your own teammate or ask us to find one for you. In all cases, send an email to Polina by Friday Oct 31, clearly indicating your intention: (*i*) you prefer to work on the project alone; (*ii*) you want to work in a team; also mention who are the team members; one email per team is sufficient; (*iii*) you want to work in a team and you need some help to find a teammate.

The objective of the project is to obtain an in-depth understanding of algorithms for performing nearest-neighbor search in high dimensions. You will get the chance to experiment with algorithms that perform exact search, as well as with algorithms that return approximate answers.

The amount of code required for the project should not be that much, however, in order to be able to develop scalable algorithms you may have to think carefully and you may have to experiment with many different options and strategies. Do not let the project for the last moment. Start early!

You are welcome to propose another programming project instead of the one described below. In this case you should write a description of the proposed project, send it to the course instructors, discuss it with them if needed, and have it approved by them.

The deadline to propose your own project is Friday, Oct 31. The project you will propose should be relevant to the course, it needs to be of equivalent difficulty to the one described here, it should not be already implemented, and it should not be used in another course.

Remember that there is a budget of 5 late days, which you can use in any way you want for the three homeworks and the project. Weekend days count as late days.

Project description

You are given a dataset, stored in a single file, representing a set of tweets collected from `twitter`. The file can be downloaded from

`http://users.ics.aalto.fi/gionis/tweets_15m.txt.gz`

The file is 471 MB compressed and 1.1 GB uncompressed.

The data have been processed so that each line of the file corresponds to a single tweet and it contains the set of terms appearing in the tweet. The terms have been *stemmed* and duplicates have been removed. The dataset contains about 15 million tweets, and 8.7 million unique terms.

We are going to consider the first 1 000 tweets as *queries*, and the rest of the tweets (1 001-st line until the end of the file) as the *database*. We refer to the "query part" of the file by Q and to the "database part" of the file by D.

Additionally, we will consider taking a d-dimensional subspace of the full space. We define three strategies for reducing the dimensionality:

d-frequent : Keep only the d most frequent terms, and ignore the rest. The frequency of each term is defined as the number of tweets in the database that the term appears.

d-infrequent : Keep only the d least frequent terms, and ignore the rest.

d-random : Keep d random terms.

Let $\mathbf{x}$ be a tweet and let $t(\mathbf{x})$ denote the number of terms contained in $\mathbf{x}$. We represent the tweet $\mathbf{x}$ by the set of terms $\{x_1, \ldots, x_{t(\mathbf{x})}\}$. Given two tweets $\mathbf{x} = \{x_1, \ldots, x_{t(\mathbf{x})}\}$ and $\mathbf{y} = \{y_1, \ldots, y_{t(\mathbf{y})}\}$ we define the distance of $\mathbf{x}$ and $\mathbf{y}$ to be angle of the corresponding sets, when those are viewed as vectors in a d-dimensional binary space. One way to compute the angle is

$$\text{angle}(\mathbf{x}, \mathbf{y}) = \arccos \frac{|\mathbf{x} \cap \mathbf{y}|}{\sqrt{t(\mathbf{x})}\sqrt{t(\mathbf{y})}},$$

and note that it takes values from 0 to $\frac{\pi}{2}$.

We are interested in the *nearest-neighbor problem*: given a query tweet $\mathbf{q}$, find the tweet in the database that is nearest to $\mathbf{q}$ according to the angle distance.

In all the questions below, a *full nearest-neighbor search* involves searching for the database tweet $\mathbf{x} \in D$ that is nearest to a query tweet $\mathbf{q}$, *for all* $\mathbf{q} \in Q$

Task 1 [preprocessing]

Compute the number of tweets n_t, in which each term t appears. Compute and plot the *distribution* of n_t, that is, the number of terms t_k that appear in exactly k tweets.

Compute and plot the *cumulative* of the previous distribution. (First you have to define what is the cumulative of a distribution).

Rename the terms as consecutive integers $1, 2, \ldots, d$ so that $n_1 \geq n_2 \geq \ldots \geq n_d$. Plot n_j as a function of j.

How do the three plots above look? What are your conclusions from the plots?

(*hint*: it will be helpful to make the plots in log-log scale).

Task 2 [exact nearest neighbor — brute force]

Implement the brute-force linear-search algorithm to find the nearest neighbor of a given query tweet $\mathbf{q}$. Perform a *full nearest-neighbor search* using this algorithm. Record the total running time.

Perform full nearest-neighbor searches for datasets formed with the three dimensionality-reduction strategies: d-frequent, d-infrequent, and d-random. Use $d = 100 \times 2^j$ with $j = 0, 2, 4, \ldots, 14$.

Plot the running times as a function of d (so you should produce three plots).

Task 3 [exact nearest neighbor — speedups]

Design and implement an algorithm that perfroms exact nearest-neighbor computation and it is as fast as possible.

You can use any speedup idea that you can think of. One observation that can be useful is to realize that for any two tweets to have angle $\text{angle}(\mathbf{x}, \mathbf{y}) < \frac{\pi}{2}$ they need to have at least one common term. If two tweets do not have common terms, their angle is $\frac{\pi}{2}$. You may also want to use early stopping criteria or any other trick you can think.

Repeat the *full nearest-neighbor searches* described in Task 2. This time, however, instead of plotting absolute running time, plot the speedup of your algorithm against the brute-force algorithm. You should again produce three plots, and speedup should be shown as a function of d.

It is likely that your solution will involve building an index as a preprocessing stage. In this case, the time for building the index should not be counted in the times that you will report.

As a definition of speedup you can use the following

$$\text{speedup} = \frac{\text{running time of the brute-force algorithm}}{\text{running time of the improved algorithm}}.$$

Task 4 [approximate search]

Now you allow to return *approximate* nearest neighbors. Design and implement an algorithm for approximate nearest-neighbor search.

Tune the parameters of your algorithm so that you target to achieve an approximation error that is less than 50%. This means that if for a query $\mathbf{q}$ the nearest neighbor is a tweet $\mathbf{x}^*$ with angle distance $\alpha^* = \text{angle}(\mathbf{q}, \mathbf{x}^*)$, your algorithm should return some tweet $\mathbf{x}$ whose angle to $\mathbf{q}$ is at most $\alpha = \text{angle}(\mathbf{q}, \mathbf{x}) \leq (1 + 0.5)\,\alpha^*$. Your algorithm can be probabilistic, so that the approximation error sometimes can be larger than 50% while other times will be smaller, but on average it should be less than 50%.

Perform full nearest-neighbor searches as before. Plot the speedup over the brute-force as a function of d. As before, the time to build an index should not be counted.

Compute and plot the approximation error that your algorithm achieves, averaged over all query tweets in Q. Again, show these plots as a function of d. As a definition of the approximation error you can use

$$\epsilon = \frac{\alpha - \alpha^*}{\alpha^*},$$

where α^* is the angle to the true nearest neighbor and α is the angle returned by your approximation algorithm.

Repeat with target approximation error 20%.

For a sanity check, this task asks for 12 plots:
{ speedup, approximation } × { frequent, infrequent, random } × { 50%, 20% }

Deliverables

You should provide the following deliverables in a `.tar.gz` package. It will be convenient to use the following naming convention:

```
LastNameOfTeammate1_LastNameOfTeammate2/report.pdf
LastNameOfTeammate1_LastNameOfTeammate2/source/
```

Report. Your report should provide answers to the questions asked and it should contain the requested plots.

You should also describe your algorithms in sufficient detail, and you should provide justification for your design choices.

Additionally, you should include a brief discussion with your observations.

Source code. Provide all the source code and scripts, and include a README.txt file explaining how to use it. The program should be able to run locally without installing any other package.