Similar Items

A small program that calculates similarities between text files

Solution

The solution is split into smaller parts

Shingles

Each file is turned into *shingles* by cutting it into small chunks, these chunks are then hashed (with 32 bit CRC) to save space and comparison complexity. Before cutting any file, its text is turned into lowercase and spaces are replaced with underscores.

Signature vectors

Each set of shingles (located in a file) is also represented as a shortened version with its most representative shingles. This is achieved by applying min-hash to the set and picking the smallest value (hashed shingle) many times. A linear hash function is used (y = kx + b % c) which is unique for each min-hash iteration.

Locality hashing

In order to filter out candidates and reduce the amount of comparisons needed, locality hashing is used. The signature vector of each file is split into bands that are hashed and used for indexing buckets in a hash table. Files that end up in the same bucket(s) are candidates of being similar and is what comparisons are based upon.

By choosing the amount of bands (and rows per band) one can alter the probability of false-negatives vs false-positives. This project chose a band size that gave the smallest false-positive probability possible while avoiding false-negatives. The amount of buckets in the hash table also determines the amount of missed candidates. A rather large size was used in this project to increase comparison speed by increasing the chance of false-negatives.

Finding similar items

Each file is run through the above procedure and then uses the following logic to find similar items:

```ruby
# Compares a given file to its candidates
def compare_file(file)
  candidates = possible_candidates_for(file)
  candidates = SignatureMatrix.filter_candidates(file, candidates)
  file.compare_to_candidates(candidates)
end
```

A file finds its candidates by merging all the buckets it was placed in. These bucket ids are saved earlier to avoid recalculating hashes. The file first removes all candidates that it already knows it is simlar to, by looking in its *similar_files* list.

The file then removes all candidates that are *likely* to be un-similar by comparing its signature vector to the others. This is done by calculating the fraction of elements that are equal (value and index) and compares it against the wanted similarity threshold.

The files that remain are compared by using Jaccard similarity, calculating the intersection of shingles divided by the union of shingles. Files that passes this far are considered similar and are added to the files (and, since comparison is commutative, the similar files) list of similar files.

How to run

Prerequisites

The program is written in ruby, but uses jruby to fully utilize threading to increase speed.Jruby is highly recommended but ruby can of course be used.

Parameters

The only argument passable parameters are similarity threshold and files to compare:

- `threshold:` (optional) float (0, 1], defaults to 0.8
- `files:` (required) any amount of files or folders that will be scanned for files

Other settings

You can also alter the default parameters by changing **lib/settings.rb** to see how the program behaves.

```
CONFIG = OpenStruct.new(
  table_size:     20000,  # number of buckets in hash table
  similarity:     0.8,    # similarity threshold between files
  min_hashes:     100,    # number of min hashes for signature matrix
  shingle_size:   10,     # size of shingles
  pool_size:      8       # number of workers
  )
```

Examples

Using jruby

note: when using jruby you are required to set the jvm heap size to an appropriate value depending on the amount of files

```
jruby -J-Xmx2048m lib/main.rb 0.75 data/20_newsgroups/sci.* data/file.txt
```

Using ruby

note: if rvm (or rbenv etc) is installed, this will run as jruby because of .ruby-version

```
ruby lib/main.rb 0.75 data/20_newsgroups/sci.* data/another_file.txt
```

The program will tell which file is similar to which and to what degree.

Estimated run properties: 40 seconds runtime and 400MB ram usage per 1k files using the 20 newsgroup dataset.