Implementation exercise

GORILLA–SEA CUCUMBER HASH

Thore Husfeldt (Rasmus Pagh)
2010 (updated 2011)

Data mining

Implement the similarity algorithm you find on the next page. You can use whatever hash function you want, and play around with parameters k and d until you get output that you are biologically comfortable with. For me, k=20 and d=10000 works pretty well.

You may need to consult some kind of reference work to remember (or learn) what a dot product is, and how to compute |a|, the length of a vector.

Description

We use hashing to compare protein sequences in order to learn if we are closer (genetically) to a gorilla or a sea cucumber. This problem can be solved exactly and efficiently using a technique called "dynamic programming", but here we'll settle for a simpler and more brutal approach. (The dynamic programming solution is the topic of another exercise that you may see in another course.)

Inputs

The input files "HbB_FASTAs.in" contains proper data from a protein sequence database. The file bestmovies.in contains information on actors in some of the best movies ever.

Output

For each pair of organisms (movies), output a number between 0 and 1 that describes their similarity, based on the given strings. If you did everything right, organisms like "human" and "gorilla" should be judged more similar than "cow" and "sea cucumber."

Requirements

You output will depend on your choice of hash function and several parameters. Thus, I cannot provide sample output. Thus, you're on you own for evaluating the correctness of your code.

At the very least, you need to include some kind of documentation that you code for computing the length of a vector, and the angle between two vectors works as expected. For example, you can write a test method that calls your "compute_angle" method on some known vector pairs.

```
>Human 2144721 HBHU 4HHB
MVHLTPEEKSAVTALWGKVNVDEVGGEALGRLLVVYPWTQRFFESFGDLSTPDAVMGNPKVKAHGKKVLG
AFSDGLAHLDNLKGTFATLSELHCDKLHVDPENFRLLGNVLVCVLAHHFGKEFTPPVQAAYQKVVAGVAN
ALAHKYH
>Human-sickle 2392691 2HBS
VHLTPVEKSAVTALWGKVNVDEVGGEALGRLLVVYPWTQRFFESFGDLSTPDAVMGNPKVKAHGKKVLGA
FSDGLAHLDNLKGTFATLSELHCDKLHVDPENFRLLGNVLVCVLAHHFGKEFTPPVQAAYQKVVAGVANA
LAHKYH
...
```

First few lines of HbB_FASTAs.in. The Human protein takes up three lines, "MVHL...KYH". Ignore things like "2144721 HBHU 4HHB" — I have no idea what that stuff even means.

Similarity test based on hashing

For integer k, a k-gram of a string is a substring of length k. Every string of length n has up to $n–k$ such k-grams, but there may be repetitions. For example, in "ABACADABRA", the 2-gram "AB" appears twice.

Choose a hash function h from the set of k-grams to $\{0,\ldots,d–1\}$. For example, in Java, you could take `String.hashcode() % d`. Given a string S, compute for each $0 \le i < d$ the value $p[i]$ as the number of k-grams T of S for which $h(T) = i$. The resulting array p is called a *profile* of S. Two identical strings will have exactly the same profile. More interestingly, two similar strings will have similar profiles: ABACADABRA and BABABLACKSHEEP both contain "AB" the same number of times, both of which contribute to the profile index counting `h("AB") % d`. Assuming uniform hashing, no other 2-grams are likely to get the same hash value, so the profile index is roughly the same.

So, two strings are similar if their profiles are similar. Profiles are easier to work with because they all have the same length: they are sequences of exactly d integers. This puts us on well-understood ground: sequences of d integers can be viewed as vectors in d-dimensional space, and they are similar if they "point in the same direction, more or less."

That's the concept of *angle*: the angle a between two vectors p and q is defined as $a = p \cdot q / |p|\,|q|$, where $p \cdot q$ is the *dot product*. of p and q, and $|p|$ is the length of p. The trigonometric function cosine transforms the angle into a number between 0 and 1 such that $\cos a$ is close to 1 when the angle between p and q is very small, so that p and q are similar.

In summary: Fix d and k, and a hash function. Compute the profile of a string by counting the number of k-grams that map to each hash value. Compare two strings by computing the cosine of the angle of their profiles.

REALITY CHECK

Is this really how it's done?
Well, close enough.
In particular, techniques for comparing documents (for example to detect near-duplicates for web search engine reporting, data mining, or fraud dectection) are based on comparing hash values. The details are slightly different, and if you want to read up on this, start with Wikipedia's article on min-wise independent hashing.
For comparing protein sequences, one normally uses a more precise distance estimate called the *Levenshtein distance*, sometimes called *edit distance*. For that particular distance there's actually a better algorithm called *dynamic programming*, which works fast enough for such small inputs. The same thing is used by your word processor's spell checker to find the word you meant to type.

WHY HASHING?

Q: What role does hash function play in the algorithm? Why not just compute the profiles of the k-grams themselves, instead of their hash values?

A: This would indeed work. In principle. The problem is that the profiles become too large. Assume for a moment that there are only 24 letters in the alphabet. Then there are 24^k different k-grams, and your profile vector p would be 24^k-dimensional. Good luck storing that on your machine for k=20, say! Also, most of the entries in p would end up being 0 anyway. With hashing, you need only d dimensions. So this is a prototypical hashing application: avoid storing a (largely useless) sparse table by hashing it into a much smaller table without losing too much information.