Gorilla–Sea Cucumber Hash Report

Vlad Alexandru Ilie

April 22, 2018

Description

This is a simple data mining application that uses hashing to compare protein sequences in order to learn if we are closer (genetically) to a gorilla or a sea cucumber.

Results

The following table gives the similarity between each pair of species as a number between 0 and 1, higher values meaning "more similar." We have used the hash function $h(S) = java\_string\_hash(S)\%d$ with $d = 10000$ and k-grams of length $k = 20$. As can be seen, the species closest to us is the *Gorilla*.

	Human	Gorilla	Monkey	Horse	Deer	Pig	Cow	Gull	Trout	R. Cod	Lamprey	Sea Cuc.
Human	1.0000	0.6712	0.3871	0.2763	0.2424	0.2887	0.2729	0.2253	0.2083	0.1973	0.1973	0.1995
Gorilla	0.6712	1.0000	0.3563	0.2653	0.2368	0.2847	0.2802	0.2165	0.2048	0.1981	0.1937	0.1968
Spider-Monkey	0.3871	0.3563	1.0000	0.2734	0.2394	0.2867	0.2713	0.2274	0.2035	0.1986	0.1953	0.1995
Horse	0.2763	0.2653	0.2734	1.0000	0.2328	0.2679	0.2676	0.2253	0.2023	0.1981	0.1911	0.1998
Deer	0.2424	0.2368	0.2394	0.2328	1.0000	0.2428	0.2430	0.2100	0.2035	0.2018	0.1912	0.2009
Pig	0.2887	0.2847	0.2867	0.2679	0.2428	1.0000	0.2662	0.2212	0.2076	0.1976	0.1907	0.1988
Cow	0.2729	0.2802	0.2713	0.2676	0.2430	0.2662	1.0000	0.2155	0.2054	0.1985	0.1986	0.1995
Gull	0.2253	0.2165	0.2274	0.2253	0.2100	0.2212	0.2155	1.0000	0.2060	0.2022	0.1973	0.2048
Trout	0.2083	0.2048	0.2035	0.2023	0.2035	0.2076	0.2054	0.2060	1.0000	0.2038	0.2015	0.1995
R. Cod	0.1973	0.1981	0.1986	0.1981	0.2018	0.1976	0.1985	0.2022	0.2038	1.0000	0.1954	0.1966
Lamprey	0.1973	0.1937	0.1953	0.1911	0.1912	0.1907	0.1986	0.1973	0.2015	0.1954	1.0000	0.1986
Sea-Cucumber	0.1995	0.1968	0.1995	0.1998	0.2009	0.1988	0.1995	0.2048	0.1995	0.1966	0.1986	1.0000

Tests

The method `similarity(p, q)` computes the cosine of the angle of two vectors of the same length d. The similarity of two complex vectors is defined by the angle between them, in this case $\cos\alpha = (p \cdot q)/(|p|\,|q|)$. Where $p \cdot q$ is the dot product of p and q, and |p| is the (Euclidean) length of p. It has been tested on the following examples:

p	q	d	value returned
$(0,1)$	$(0,1)$	2	1
$(0,1)$	$(0,2)$	2	1
$(0,1)$	$(1,0)$	2	0
$(0,1)$	$(0,-1)$	2	-1
$(0,\ldots,0,1)$	$(1,0,\ldots,0)$	1000	0
$(0,1,1)$	$(1,0,1)$	2	0.5
$(0,1,1)$	$(0,0,2)$	2	0.7

Similarly, our static method `EuclidLen(p)` computes the Euclidean length of p. $|p| = \sqrt{\sum_{i=1}^{n} p(i)^2}$

Performance

With K = 30 and D = 100000, this implementation took 10.074*seconds* to compute the output.

With K = 20 and D = 10000, this implementation took 2.591*seconds* to compute the output.