Final Project Report: SMASH

Nathan Roach, Patricia Walchessen, Charlie Wang, and Yue Zhang

December 8, 2017

Abstract

[1 paragraph] Final Project Abstract

Introduction

An intriguing and common problem that arises in computation is efficiently comparing the similiarities between high-dimensional data. The problem reaches to the far corners of computation, including with machine learning and kernels. In bioinformatics, the problem takes the form of long genomic sequences and the aim of determining the similarity between both close and disparate sequences. There are many purposes for solving the problem in bioinformatics, including evaluating evolutionary relationships, determining the organismal source of sequencing data, and comparing metagenomic data.

One class of algorithms that has been developed is Locality-Sensitive Hashing (LSH) algorithms. LSH aims to solve the problem by reducing data dimensionality and binning inputs in such a manner that similar data sets can be grouped together with high probability. There are two prevalent LSH algorithms in the wild: MinHash and SimHash.

MinHash computes the differences between two genomic sequences through the estimation of a Jaccard Index in which a ratio is computed that evaluates

the percent shared identity between the sets. SimHash, on the other hand, estimates the cosine similarity through the use of weighting of the frequency of bits and then compiling the signs from the summations.

Prior Work

[1 page] Some prior work include a paper published in 2016 about using MinHash to evaluate genomic sequences. In 2016, Ondov et al. used MinHash to analyze the resemblance between genomes and metagenomes[3]. After additional research, we saw no reference of using any other LSH to hash and compare genomic sequences, so we pursued a research project on the competitor, SimHash. The literature suggests that SimHash can be used for the same purpose as MinHash in analyzing resemblance.

Methods and Software

0.1 Exploration with SimHash

We started exploring the existing SimHash literature and implementations out in the wild to see how the original SimHash algorithm performs. Of the many that we looked into, the two that we spent the most time with and explored the most extensively were: one that is written in Python (https://github.com/leonsim/simhash) and another that is written in Go (https://github.com/mfonda/simhash).

What is interesting of note about these packages is that they are primarily built on shingling, or dividing up into k-mers. method, combined with For both of these SimHash libraries, we took a deep dive into each of the respective codebases and modified the code to accept genomic data as input and to spit out comprehensible output for genomic data. We also modified other parts of the code to make it more flexible. For example, in the Python library, there were some hard-coded values such as the size of the shingle or k-mer. Next, we fed in different genomic sequences, and using each of the

libraries' interfaces, observed the outputs from the SimHash library.

To our surprise, when putting in the entire genome, SimHash was returning quite inaccurate outputs. We first noted that it may be because of the hashing function used, so we dove into what the libraries used. The Python library was using a MD5 hashing function, while the Go library was doing manual bit shifting operations. Since the hashing functions were different yet both were not returning great results, we concluded that it might not be due to the hashing function.

We scoured the Internet and found that it might be due to how SimHash is primarily an algorithm built to detect duplicity and cannot handle too big of a hamming distance, say a hamming distance of over 20. Thus, we needed to do more than just feed the whole genome into the SimHash libraries.

We took a closer look at the genome file, and the first pattern we discovered was the substrings of uppercase letters and lowercase letters. We discovered that uppercase letters corresponded to non-repeats while lowercase letters correspond to repeats. The first approach of ours was to make all the bases in the sequence uppercase letters. Then, we ran the SimHash libraries again. Once again, the results were inaccurate.

Finally, we decided to take one last stab at it by intelligently dividing up the genome. We grouped all the sequences that were uppercase together, and all the sequences that were lowercase together. This is based on the assumption that the repeats, when they exist, are long. Then, we constructed the shingles or k-mers, and ran them through the SimHash libraries. Lo and behold, the results were still inaccurate.

0.2 Our Flavor of SimHash

[1.5 pages] Seeing how existing SimHash libraries were not performing as expected, we went back to the theory behind the SimHash algorithm. We

0.2.1 Data

0.2.2 Algorithm

0.3 Exploration with MinHash

[0.5 of a page]. Patricia's description of what she has done with MinHash.

Results

[1.5 pages] Insert results of different k-mer sizes and of different genomic sequences.

Conclusions

[0.5 pages] There are more ways to judge similarity between two genomic sequences than just hashing the entirety of each of the genome and comparing how close in relation both are.

References

[1] Caitlin Sadowski and Greg Levin. *SimHash: Hash-based Similarity Detection*. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.473.7179&rep=rep1&type=pdf, 2007.

[2] Anshumali Shrivastava and Ping Li. *In Defense of MinHash Over SimHash*. Journal of Machine Learning, 33, 2014.

[3] Brian D. Ondov, Todd J. Treangen, Páll Melsted, Adam B. Mallonee, Nicholas H. Bergman, Sergey Koren and Adam M. Phillippy. *Mash: fast genome and metagenome distance estimation using MinHash*. Genome Biology, 17:132, 2016.