CSE 584A Class 24

Jeremy Buhler

April 23, 2018

1 Another Kind of Distance via Hashing

- In the last two classes, we looked at strategies for finding approximate matches between single k-mers, for short-ish k (tens of bases).
- We'll now look at strategies for finding matches in much longer sequences.
- **Key Relaxation**: instead of estimating global similarity, as in alignment methods, we will assess whether two sequences contain *many local similarities.*
- This "alignment-free" approach will lose some information compared to global approaches but avoids the need for expensive alignment algorithms.
- It is also relatively insensitive to many kinds of genomic rearrangement, which is not true of alignment.

The basic idea is to treat a sequence S as a bag containing all its constituent k-mers, for some fixed k.

- Let $\Sigma_k[S]$, the *k-spectrum* of S, be a set containing every k-mer that appears as a substring of S.
- Consider two sequences S and T.
- Intuitively, if S and T are similar, they have similar k-spectra.
- Moreover, if S is similar to a substring of T, $\Sigma_k[S]$ is approximately a subset of $\Sigma_k[T]$.
- We can make these ideas more precise by measuring the *Jaccard index* J between $\Sigma_k[S]$ and $\Sigma_k[T]$:
$$J_k(S,T) = \frac{|\Sigma_k[S] \cap \Sigma_k[T]|}{|\Sigma_k[S] \cup \Sigma_k[T]|}.$$
- The Jaccard index lies between 0 and 1. It is 1 if S and T have identical k-spectra.
- Note that two distinct sequences can have identical k-spectra, e.g. AAACCCAAA and CCCAAACCC for $k = 3$.
- But if we make k long enough, non-pathological sequences are unlikely to have a high Jaccard index unless they are nearly the same.

- "Long enough" could be chosen so that it is very unlikely that two unrelated sequences, say under an iid model, share a k-mer purely by chance.
- For eukaryotic genome-sized sequences, k in the range 21-35 achieves this property.

How is the Jaccard index related to evolutionary distance between sequences?

- Suppose S and T are two homologous sequences of common length n that have diverged by point *substitutions* over time.
- In particular, suppose that for each base of S, the corresponding base of T has mutated with probability δ.
- Then for any fixed k-mer of S, the expected number of accumulated mutations is $k\delta$.
- Assuming that mutations appear independently in each position of the sequence, accumulation of mutations in a k-mer is an (approximately) *Poisson process* with intensity $k\delta$.
- Hence, the probability that a given k-mer accumulates zero mutations is (approximately) $e^{-k\delta}$.
- Now let w be the number of k-mers shared between two (non-repetitive) sequences of length n under this mutation process.
- Then we have (modulo some very unlikely events)

$$E[w] = ne^{-k\delta}.$$

- We can estimate $E[w]$ simply by counting the observed number of shared k-mers between S and T, then solve for δ above to estimate the mutation distance between S and T.

But what if S and T have different sizes?

- Let n be the average size $(|S| + |T|)/2$.
- Then we have that

$$\begin{aligned} J_k(S,T) &= \frac{|\Sigma_k[S] \cap \Sigma_k[T]|}{|\Sigma_k[S] \cup \Sigma_k[T]|} \\ &= \frac{w}{|\Sigma_k[S]| + |\Sigma_k[T]| - w} \\ &= \frac{w}{2n - w}. \end{aligned}$$

or, equivalently,

$$\frac{w}{n} = \frac{2J_k(S,T)}{1 + J_k(S,T)}.$$

Hence, we can use the Jaccard similarity to get our point estimate of w, and thence δ.

- Technically, estimating δ this way is a bit suspect because our model does not formally permit sequences of different sizes.
- Empirically, however, Ondov et al. (see MASH link below) argue that it does a reasonable job.

Once we can compute distances δ between pairs of genomes, we can cluster them by evolutionary distance, build phylogenetic trees, and so forth.

2 Efficiently Computing the Jaccard Index

We can do lots of cool stuff if we can compute $J_k(S, T)$. So how do we do that?

- *One obvious approach*: hash all the k-mers in each sequence.
- We can then intersect the two hash tables to obtain the Jaccard index.
- This is the approach taken by tools such as Jellyfish (Marçais and Kingsford, *Bioinformatics* 27:764, 2011).
- Alternatively, we can build the BWT for each of S and T and use an extension of the k-mer counting hack from Homework 3 to count the number of shared k-mers, as well as the numbers of k-mers in each of S and T.
- See Mäkkinen et al.'s book for more details.
- *Problem*: both of these approaches take time and space $\Theta(|S| + |T|)$.
- For genome-sized sequences, this is undesirable.
- Even for smallish genomes like bacteria, trying to compute distances from a given genome to many others (e.g. 10^5 other genomes in GenBank) takes quite a lot of time and space.
- *Can we reduce the storage and time costs of comparison?*

What if we could *estimate* the Jaccard index much faster than computing it exactly?

- *Key idea*: sample k-mers uniformly from the set $\Sigma_k[S] \cup \Sigma_k[T]$.
- If we sample m k-mers, and w of them are in $\Sigma_k[S] \cap \Sigma_k[T]$, then we expect that

$$J_k(S, T) \approx w/m.$$

  More precisely, can easily prove that w/m is an unbiased estimator: $E[w] = mJ_k(S, T)$.

To efficiently sample from the union, we turn to a technique called *min-hashing.*

- Define a hash function h from k-mers to integers of at least $2k$ bits.
- A "good" hash function h should resemble a random permutation of k-mer space.
- For each k-mer $s \in S$, compute $h(s)$.

- Choose a *sketch size* m and set the *sketch* $\Phi_h[S]$ to be the k-mers with the m smallest distinct hash values obtained from S.
- Finally, find the set $\Psi(S,T)$ of k-mers with the m smallest unique hash values in $\Phi_h[S] \cup \Phi_h[T]$.
- **Claim**: If h is a random permutation on the space of k-mers, then $\Psi(S,T)$ is a uniform random sample from $\Sigma_k[S] \cup \Sigma_k[T]$.
- **Pf**: $\Phi_h[S]$ and $\Phi_h[T]$ contain the m smallest k-mers (under h) in $\Sigma_k[S]$ and $\Sigma_k[T]$ respectively.
- Hence, the m smallest k-mers in $\Phi_h[S] \cup \Phi_h[T]$ are also the m smallest in $\Sigma_k[S] \cup \Sigma_k[T]$.
- But h is a random permutation, so any subset of size m from $\Sigma_k[S] \cup \Sigma_k[T]$ is equally likely to be smallest under h. QED
- **Claim**: a k-mer x in $\Psi(S,T)$ is in $\Sigma_k[S] \cap \Sigma_k[T]$ iff $x \in \Phi_h[S] \cap \Phi_h[T]$.
- **Pf**: if x is in $\Psi(S,T)$, it is among the m smallest in the union $\Sigma_k[S] \cup \Sigma_k[T]$.
- If x is in both $\Sigma_k[S]$ and $\Sigma_k[T]$, it must therefore be among the smallest m elements in each, and so will appear in both $\Phi_h[S]$ and $\Phi_h[T]$.
- Conversely, if x is in both $\Phi_h[S]$ and $\Phi_h[T]$, it is trivially in both $\Sigma_k[S]$ and $\Sigma_k[T]$. QED
- Conclude that we can count the number w of k-mers in $\Psi(S,T)$ that are in $\Phi_h[S] \cap \Phi_h[T]$ and estimate $J_k(S,T)$ as w/m.

A few practical notes on min-hashing...

- We want to choose h to look like a random permutation on k-mers.
- It should disrupt any biologically meaningful structure, e.g. our sampling should not be biased toward AT- or GC-rich k-mers.
- Many good choices, e.g. MurmurHash3 (Appleby 2012).
- Typically, sketch size m is on the order of hundreds to thousands (analysis to follow next time).
- We can compute a sketch of size m from a genome of size n in space $O(m)$ and time $O(n \log m)$ using a priority queue.
- If we keep our sketches sorted by hash value, we can use merging to compute w for a pair of sketches in time $O(m)$.
- The larger m, the more accurate the estimate of J_k, but the more space and time needed to store the sketches.
- Also, more divergent sequences share fewer k-mers, so sampling with a fixed sketch size m yields more uncertain estimates of J_k.

For more details, see Ondov et al.'s MASH tool for min-hash-based sequence comparison (*Genome Biology* 17:132, 2016): `https://genomebiology.biomedcentral.com/articles/10.1186/s13059-016-0997-x`

3 Assembly-Free Genome Comparison

Min-hashing gives a way to compare genomes without alignment. But we still have to obtain the genomes in the first place.

- Assembly is used to turn raw sequence reads into a finished genome.
- It is very helpful for inferring how parts of the genome more than a read length apart are organized, and in particular, it is essential for long-range alignment.
- But Jaccard-based comparison is insensitive to long-range order!
- Hence, why not skip assembly altogether and directly compute a genome's k-mer spectrum from its raw read set?

A couple of caveats...

- *Problem*: we must sequence enough to be fairly sure our reads contain nearly every k-mer in the genome.
- Typical "coverage" of genome sequencing is between, say, 5x and 100x the genome size, depending on budget. The greater the coverage, the lower the chance of missing a given k-mer.
- *Problem*: raw reads have a much higher error rate than the finished genome.
- Result is that read set, and hence its k-spectrum, contains many k-mers not present in the genome.
- If coverage is high enough that nearly all true genomic k-mers appear in at least 2 reads, we can discard any k-mer that appears just once as noise.
- (don't do this if your coverage is only 5x – you'll throw away a lot of real k-mers that were sampled once. 20x is better.)

How do we apply sketching techniques to unassembled reads?

- Do not add a k-mer to the sketch unless it has been seen twice.
- *More expensive*: keep a table of all k-mers seen exactly once so far, and consult it for each new k-mer to decide whether to add it to the sketch.
- *Less expensive*: use an approximate membership data structure, such as *Bloom filter*, to remember the set of k-mers seen so far.
- Bloom filters have a small false positive rate for membership testing but no false negatives.
- They can be stored in much less space than a full list of all k-mers seen.
- Variants of the above two approaches can be used to disallow k-mers that have been seen less than q times, for any fixed $q > 0$.