CSEP 521: Applied Algorithms
Lecture 16 – Document Similarity

Richard Anderson, February 25, 2021

Announcements

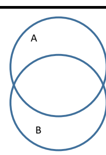
Topics

- Document similarity
- MinHash
- String similarity
  - Edit Distance / Longest Common Subsequence
  - Sharding strings
- Dimension reduction

Document Similarity

- Want to be able to identify documents that are "very close" to each other
- Very large number of documents
- Individually pre-process documents
  - Save a small amount of data per document (sketch)
  - Perform similarity tests based on sketch

Jaccard Similarity



$$\text{Jaccard}(A, B) = \frac{A \cap B}{A \cup B}$$

Let X be the characteristic vector for A where x_j is the multiplicity of item j and Y be the characteristic vector for B where y_j is the multiplicity of item j.

$$\text{Jaccard}(A, B) = \frac{\sum_j \min(x_j, y_j)}{\sum_j \max(x_j, y_j)}$$

Representation scheme

- Tokenize document
- Break document into shards
- Hash each shard into a domain of size 2^{64} (unsigned long)
- Treat as a bag of words*
- Use Jaccard similarity measure

far out in the uncharted backwaters of the unfashionable end of the western spiral arm of the galaxy lies a small unregarded yellow sun

1. far out in the uncharted
2. out in the uncharted backwaters
3. in the uncharted backwaters of
4. the uncharted backwaters of the
5. uncharted backwaters of the unfashionable
6. backwaters of the unfashionable end
7. of the unfashionable end of
8. the unfashionable end of the
9. unfashionable end of the western
10. end of the western spiral
11. of the western spiral arm
12. the western spiral arm of

* In this application, we use bag of words without multiplicity

Similarity testing

- Identify document pairs that have high similarity by doing pairwise comparison
- Precompute hashes of shards – n shards for document of n tokens
- Cost of comparison is O(n)

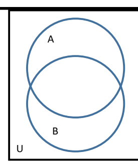
- How to improve this: reduce the amount of information stored per document

MinHash

- U is the domain (in this case, the hash of the shards, $[0 \ldots 2^{64}]$)
- Choose a random permutation π on U
- Let $A \subseteq U$
- $\text{MinHash}(A) = \text{argmin}_{x \in A}\, \pi(x)$
  - MinHash is the smallest element of A under the random permutation

U = {a, b, c, d, e, f, g, h, i, j }
A = { b, c, e, g }
B = { c, e, f }
π_1 = [a, c, d, i, j, h, b, e, f, g]
π_2 = [j, i, g, c, b, h, e, a, d, f]

An amazing result



$$\Pr[\text{MinHash}(A) = \text{MinHash}(B)] = \frac{|A \cap B|}{|A \cup B|} = \text{Jaccard}(A, B)$$

Using the MinHash

- Identify document pairs where Jaccard(A,B) ≥ 0.95
- Run MinHash with k independent random permutations
- Number of times MinHash(A)=MinHash(B) is a good estimate of Jaccard Similarity
- Compute the k MinHashes for each documents as a sketch
- Comparison of documents requires k comparisons

Similarity of Strings

- String edit distance – how many edits to convert S_1 into S_2
- Edit operations: Add character, Remove character, (Change character)

BARTHOLEMEWSIMPSON → KRUSTYTHECLOWN

B	A	R	T	H	O	L	E	M	E	W	S	I	M	P	S	O	N				B-
A	R	T	H	O	L	E	M	E	W	S	I	M	P	S	O	N					A-
R	T	H	O	L	E	M	E	W	S	I	M	P	S	O	N						K+
K	R	T	H	O	L	E	M	E	W	S	I	M	P	S	O	N					U+
K	R	U	T	H	O	L	E	M	E	W	S	I	M	P	S	O	N				S+
K	R	U	S	T	H	O	L	E	M	E	W	S	I	M	P	S	O	N			Y+
K	R	U	S	T	Y	H	O	L	E	M	E	W	S	I	M	P	S	O	N		T+
K	R	U	S	T	Y	T	H	O	L	E	M	E	W	S	I	M	P	S	O	N	O-
K	R	U	S	T	Y	T	H	L	E	M	E	W	S	I	M	P	S	O	N		L-
K	R	U	S	T	Y	T	H	E	M	E	W	S	I	M	P	S	O	N			M-
K	R	U	S	T	Y	T	H	E	E	W	S	I	M	P	S	O	N				E-
K	R	U	S	T	Y	T	H	E	W	S	I	M	P	S	O	N					C+
K	R	U	S	T	Y	T	H	E	C	W	S	I	M	P	S	O	N				L+

BARTHOLEMEWSIMPSON → KRUSTYTHECLOWN

K	R	U	S	T	Y	T	H	E	C	L	W	S	I	M	P	S	O	N			O+
K	R	U	S	T	Y	T	H	E	C	L	O	W	S	I	M	P	S	O	N		S-
K	R	U	S	T	Y	T	H	E	C	L	O	W	I	M	P	S	O	N			I-
K	R	U	S	T	Y	T	H	E	C	L	O	W	M	P	S	O	N				M-
K	R	U	S	T	Y	T	H	E	C	L	O	W	P	S	O	N					P-
K	R	U	S	T	Y	T	H	E	C	L	O	W	S	O	N						S-
K	R	U	S	T	Y	T	H	E	C	L	O	W	O	N							O-
K	R	U	S	T	Y	T	H	E	C	L	O	W	N								

Longest Common Subsequence

- $C=c_1\ldots c_g$ is a subsequence of $A=a_1\ldots a_m$ if C can be obtained by removing elements from A (but retaining order)
- LCS(A, B): A maximum length sequence that is a subsequence of both A and B

`ocurranec`	`attacggct`	`bartholemewsimpson`
`occurrence`	`tacgacca`	`krustytheclown`

Edit Distance and LCS

- String A has length n and B has length m
- Suppose that A is converted to B by removing k characters and adding j characters
  - Number of unchanged characters is $c = n - k = m - j$
  - Edit distance d is $k + j$
  - $n + m = 2c + k + j = 2c + d$
  - $d = n + m - 2c$
- Minimizing the edit distance is maximizing the length of the common sequence

LCS Optimization

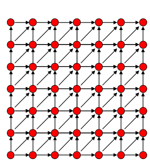
- $A = a_1 a_2 \ldots a_m$
- $B = b_1 b_2 \ldots b_n$
- Opt[j, k] is the length of $LCS(a_1 a_2 \ldots a_j, b_1 b_2 \ldots b_k)$
- Optimization recurrence

If $a_j = b_k$, Opt[j,k] = 1 + Opt[j-1, k-1]

If $a_j \neq b_k$, Opt[j,k] = max(Opt[j-1,k], Opt[j,k-1])

Opt[j,0] = Opt[0,k] = 0

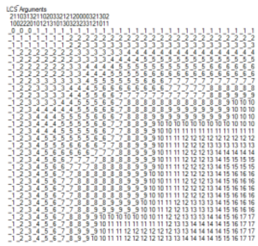
Dynamic Programming Computation



Code to compute Opt[n, m]

```
for (int i = 0; i < n; i++)
  for (int j = 0; j < m; j++)
    if (A[ i ] == B[ j ] )
      Opt[ i,j ] = Opt[ i-1, j-1 ] + 1;
    else if (Opt[ i-1, j ] >= Opt[ i, j-1 ])
       Opt[ i, j ] := Opt[ i-1, j ];
    else
       Opt[ i, j ] := Opt[ i, j-1];
```

Computing the Longest Common Subsequence



LCS Performance

- Runtime is $O(n^2)$ for a pair of strings of length n
- Space requirement is $O(n^2)$
  - Which can be reduced to O(n) be reusing rows
  - Recovering the actual LCS is more work, but can also be done in O(n) space

Experiment: compute the length of two random bit strings (alphabet size 2)

N: 10000 Base 2 Length: 8096 Gamma: 0.8096 Runtime:00:00:01.86
N: 20000 Base 2 Length: 16231 Gamma: 0.81155 Runtime:00:00:07.45
N: 30000 Base 2 Length: 24317 Gamma: 0.8105667 Runtime:00:00:16.82
N: 40000 Base 2 Length: 32510 Gamma: 0.81275 Runtime:00:00:29.84
N: 50000 Base 2 Length: 40563 Gamma: 0.81126 Runtime:00:00:46.78
N: 60000 Base 2 Length: 48700 Gamma: 0.8116667 Runtime:00:01:08.06
. . . . .
N: 300000 Base 2 Length: 243605 Gamma: 0.8120167 Runtime:00:28:07.32

Space efficient implementation

```
public int SpaceEfficientLCS() {
        int n = str1.Length;
        int m = str2.Length;
        int[] prevRow = new int[m + 1];
        int[] currRow = new int[m + 1];

        for (int j = 0; j <= m; j++)
            prevRow[j] = 0;

        for (int i = 1; i <= n; i++) {
            currRow[0] = 0;
            for (int j = 1; j <= m; j++) {
                if (str1[i - 1] == str2[j - 1])
                    currRow[j] = prevRow[j - 1] + 1;
                else if (prevRow[j] >= currRow[j - 1])
                    currRow[j] = prevRow[j];
                else
                    currRow[j] = currRow[j - 1];
            }
            for (int j = 1; j <= m; j++)
                prevRow[j] = currRow[j];
        }

        return currRow[m];
    }
```

String similarity

- Edit distance
  - Advantages – measure of distance between strings
  - Flexibility in edit operation and weighting
- Disadvantages
  - Relatively inefficient: $O(n^2)$, heuristics may help for large strings or looking for similarity
  - Requires looking at the entire string

String similarity with shards

- Same basic idea as with documents
- Consider alphabets with a small number of characters, e.g., {a, c, t, g}
- Take shards as being strings of length k
  - k a multiple of 32 would make sense for packing into long ints
  - Hashing and minhash sketches apply as for documents
- Domain characteristics may be important
  - Mutation rate / distribution in sequences

Coming next: Dimension reduction for R^n

- Consider the distance function $D(x,y) = 0$ if $x = y$, $D(x,y) = 1$ if $x \neq y$
- Suppose we have a domain U and want to answer distance queries between a set of n elements
- Natural solution is to use $\log_2 U$ bits to describe the elements
- Can we use less space if we want to approximately answer distance queries

Of course this is going to be hashing

- Choose a good hash function h: $U \rightarrow 2^{32}$
- Let $f_1(x) = h(x) \bmod 2$
- 1 bit representation
  - If $x = y$, then $f_1(x) = f_1(y)$
  - If $x \neq y$, then $\mathbf{Pr}[f_1(x) = f_1(y)] \leq ½$
  - Property preserved with probability at least 50%
- Repeat with k independent has functions $h_1, \ldots, h_k$
  - If $x = y$, then $f_i(x) = f_i(y)$ for all $i = 1, \ldots, k$
  - If $x \neq y$, then $\mathbf{Pr}[f_i(x) = f_i(y)$ for all $i = 1, \ldots, k] \leq 2^{-k}$
- To achieve error of δ, we need to use $k = \lceil \log_2 1/\delta \rceil$