Methods for High Degrees of Similarity

Index-Based Methods
Exploiting Prefixes and Suffixes
Exploiting Length
Overview

- LSH-based methods are excellent for similarity thresholds that are not too high.
  - Possibly up to 80% or 90%.
- But for similarities above that, there are other methods that are more efficient.
  - And also give exact answers.
Setting: Sets as Strings

- We’ll again talk about Jaccard similarity and distance of sets.
- However, now represent sets by strings (lists of symbols):
  1. Enumerate the universal set.
  2. Represent a set by the string of its elements in sorted order.
Example: Shingles

- If the universal set is k-shingles, there is a natural lexicographic order.
- Think of each shingle as a single symbol.
- Then the 2-shingling of abcad, which is the set \{ab, bc, ca, ad\}, is represented by the list \[ab], [ad], [bc], [ca] of length 4.
- **Alternative**: hash shingles; order by bucket number.
Example: Words

- If we treat a document as a set of words, we could order the words alphabetically.
- **Better**: Order words lowest-frequency-first.
- **Why?** We shall index documents based on the early words in their lists.
  - Fewer docs in a bucket.
What We Mean by “Index”

Documents containing “aardvark” in prefix

Documents containing “abacus” in prefix

Buckets

aardvark

abacus
Jaccard Distance

- Jaccard distance $J = 1 - \text{Jaccard similarity}$.
- $J = |\text{symmetric difference of sets}|/ |\text{union of sets}|$.
- **Example**: $S = \{1, 2, 3\}$, $T = \{2, 4, 5\}$.
  - Symmetric difference = $\{1, 3, 4, 5\}$.
  - Union = $\{1, 2, 3, 4, 5\}$.
  - $J = 0.8$. 

---

7
Suppose two sets have Jaccard distance $J$ and are represented by strings $s_1$ and $s_2$. Let the LCS of $s_1$ and $s_2$ have length $C$ and the (insert/delete) edit distance of $s_1$ and $s_2$ be $E$. Then:

- $1-J = \text{Jaccard similarity} = \frac{C}{C+E}$.
- $J = \frac{E}{C+E}$.

Secret: $E = |\text{symmetric difference}|$; $C = |\text{intersection}|$.
Relationships Among J, E, C

- We shall be given an upper bound on J.
- Want to compare strings only if E and C are such that \( E/(E+C) \leq J \).
- \( E/(E+C) \) grows with E and shrinks with C.
- Thus, strongest limitations on pairs of strings are when E has its smallest possible value and C its largest possible value.
Indexes

- Build some indexes on the set of strings.
- Then, visit each string once and use the index to find possible candidates for similarity.

For thought: how does this approach compare with bucketizing and looking within buckets for similarity?
Length-Based Indexes

- The simplest thing to do is create an index on the length of strings.
- A string of length \( L \) can be Jaccard distance \( J \) from a string of length \( M \) only if \( L \times (1-J) \leq M \leq L/(1-J) \).
- **Example**: if \( 1-J = 90\% \) (Jaccard similarity), then \( M \) is between 90\% and 111\% of \( L \).
Why the Limit on Lengths?

1-J = M/L
M = L(1-J)

A shortest candidate

1-J = L/M
M = L/(1-J)

A longest candidate
B-Tree Indexes

- The B-tree is a perfect index structure for a length-based index.
- Given a string of length $L$, we can find strings in the range $L \times (1-J)$ to $L/(1-J)$ without looking at any candidates outside that range.
- But just because strings are similar in length, doesn’t mean they are similar.
Aside: B-Trees

- If you didn’t take CS245 yet, a B-tree is a generalization of a binary search tree, where each node has many children, and each child leads to a segment of the range of values handled by its parent.
- Typically, a node is a disk block.
Aside: B-Trees – (2)

From parent

| 50 | 80 | 145 | 190 | 225 |

To values

< 50

≥ 50, < 80

≥ 80, < 145

Etc.
Prefix-Based Indexing

- If two strings are 90% similar, they must share some symbol in their prefixes whose length is just above 10% of the shorter.

- Thus, we can index symbols in just the first $\lfloor JL + 1 \rfloor$ positions of a string of length $L$. 
Why the Limit on Prefixes?

If two strings do not share any of the first $E$ symbols, then $J \geq E/L$.

Thus, $E = JL$ is possible, but any larger $E$ is impossible. Index $E+1$ positions.

Extreme case: second string has none of the first $E$ symbols of the first string, but they agree thereafter.
Indexing Prefixes

◆ Think of a bucket for each possible symbol.

◆ Each string of length L is placed in the bucket for each of its first $\lfloor JL+1 \rfloor$ positions.
Given a *probe* string $s$ of length $L$, with $J$ the limit on Jaccard distance:

for (each symbol $a$ among the first $\lfloor JL+1 \rfloor$ positions of $s$) look for other strings in the bucket for $a$;
Example: Indexing Prefixes

- Let $J = 0.2$.
- String $abc\text{def}$ is indexed under $a$ and $b$.
- String $ac\text{d}f\text{g}$ is indexed under $a$ and $c$.
- String $b\text{c}d\text{e}$ is indexed only under $b$.
- If we search for strings similar to $c\text{d}e\text{f}$, we need look only in the bucket for $c$. 
Prefix + Length

- Sort the buckets by length.
- Instead of comparing the probe string with all members of the bucket, look only at those whose length makes it possible there is a match.
Using Positions Within Prefixes

- If position $i$ of string $s$ is the first position to match a prefix position of string $t$, and it matches position $j$, then the edit distance between $s$ and $t$ is at least $i + j - 2$.

- The LCS of $s$ and $t$ is no longer than $L - i + 1$, where $L$ is the length of $s$. 
If J is the limit on Jaccard distance, then $E/(E+C) \leq J$.
- $E \geq i + j - 2$.
- $C \leq L - i + 1$.

For smallest $E$, largest $C$:
- $(i + j - 2)/(L + j - 1) \leq J$.
- Or, $j \leq (JL - J - i + 2)/(1 - J)$.

Makes sense because $E/(E+C)$ grows with E and shrinks with C.
Positions/Prefixes – (3)

* We only need to find a candidate once, so we may as well:
  1. Visit positions of our given string in numerical order, and
  2. Assume that there have been no matches for earlier positions.
Create a 2-attribute index on (symbol, position).
If string $s$ has symbol $a$ as the $i^{th}$ position of its prefix, add $s$ to the bucket $(a, i)$.
A B-tree index with keys ordered first by symbol, then position is excellent.
If we want to find matches for probe string $s$ of length $L$, do:

```plaintext
for (i=1; i<=J*L+1; i++) {
    let s have $a$ in position i;
    for (j=1;
        j<=(J*L-J-i+2)/(1-J); j++)
        compare s with strings in bucket ($a$, j);
}
```
Example: Lookup

- Suppose $J = 0.2$.
- Given probe string adegjkmprz, $L=10$ and the prefix is ade.
- For the $i$th position of the prefix, we must look at buckets where $j \leq (JL - J - i + 2)/(1 - J) = (3.8 - i)/0.8$.
- For $i = 1$, $j \leq 3$; for $i = 2$, $j \leq 2$, and for $i = 3$, $j \leq 1$. 
Example: Lookup – (2)

Thus, for probe adegkmprz we look in the following buckets: (a, 1), (a, 2), (a, 3), (d, 1), (d, 2), (e, 1).

Suppose string \( t \) is in (d, 3). Either:

- We saw \( t \), because a is in position 1 or 2, or
- The edit distance is at least 3 and the length of the LCS is at most 9 (thus the Jaccard distance is at least \( \frac{1}{4} \)).
We Win Two Ways

1. Triangular nested loops let us look at only half the possible buckets.
2. Strings that are much longer than the probe string but whose prefixes have a symbol far from the beginning that also appears in the prefix of the probe string are not considered at all.
Adding Suffix Length to the Mix

- We can index on three attributes:
  1. Character at a prefix position.
  2. Number of that position.
  3. Length of the suffix = number of positions in the entire string to the right of the given position.
Lower Bound on Edit Distance

- Suppose we are given probe string $s$, and we find string $t$ because its $j^{th}$ position matches the $i^{th}$ position of $s$.
- A lower bound on edit distance $E$ is:
  1. $i + j - 2$ plus
  2. The absolute difference of the lengths of the suffixes of $s$ and $t$ (what follows positions $i$ and $j$, respectively).
Longest Common Subsequence

- Suppose we are given probe string $s$, and we find string $t$ first because its $j$th position matches the $i$th position of $s$.

- If the suffixes of $s$ and $t$ have lengths $k$ and $m$, respectively, then an upper bound on the length $C$ of the LCS is $1 + \min(k, m)$. 
Bound on Jaccard Distance

◆ If J is the limit on Jaccard distance, then $E/(E+C) \leq J$ becomes:

$\begin{align*}
&i + j - 2 + |k - m| \leq J(i + j - 2 + |k - m| + 1 + \min(k, m)). \\
&\text{Thus: } j + |k - m| \leq (J(i - 1 + \min(k, m)) - i + 2)/(1 - J). 
\end{align*}$
Positions/Prefixes/Suffixes – Indexing

- Create a 3-attribute index on (symbol, position, suffix-length).
- If string $s$ has symbol $a$ as the $i^{th}$ position of its prefix, and the length of the suffix relative to that position is $k$, add $s$ to the bucket $(a, i, k)$. 
Example: Indexing

- Consider string \texttt{abcde} with J = 0.2.
- Prefix length = 2.
- Index in: (\texttt{a}, 1, 4) and (\texttt{b}, 2, 3).
As for the previous case, to find candidate matches for a probe string \( s \) of length \( L \), with required similarity \( J \), visit the positions of \( s \)'s prefix in order.

If position \( i \) has symbol \( a \) and suffix length \( k \), look in index bucket \((a, j, m)\) for all \( j \) and \( m \) such that \( j + |k - m| \leq (J(i - 1 + \min(k, m)) - i + 2)/(1 - J)\).
Example: Lookup

- Consider abcde with $J = 0.2$.
- **Require:** $j + |k - m| \leq (J(i - 1 + \min(k, m)) - i + 2)/(1 - J)$.
- For $i = 1$, note $k = 4$. We want $j + |4 - m| \leq (0.2\min(4, m)+1)/0.8$.
- Look in $(a, 1, 3), (a, 1, 4), (a, 1, 5), (a, 2, 4), (b, 1, 3)$.
- From $i = 2, k = 3$, $j + |3 - m| \leq 0.2(1+\min(4, m))/0.8$. 

Pattern of Search

\[ i = 1 \]

Position

Length of suffix

\( k \)
Pattern of Search

Pattern of Search

Pattern of Search

Pattern of Search

Pattern of Search
Pattern of Search

Position

$k$

Length of suffix

$i = 3$
Physical-Index Issues

- A B-tree on (symbol, position, length) isn’t perfect.
  - For a given symbol and position, you only want some of the suffix lengths.
  - Similar problem for any order of the attributes.

- Several two-dimensional index structures might work better.