Sequences and Time Series

W-Grams and Other Approaches to Filtering

K. Selçuk Candan, Professor of Computer Science and Engineering

CASCADE
ASU Center for Assured and SCAlable Data Engineering

Ira A. Fulton Schools of Engineering
Arizona State University

Strings, sequences, time series

A *string* or *sequence*, $S = (c_1, c_2, \ldots, c_N)$, *is a finite sequence of symbols.*

abcbbbaabbaabcbbbaaabbc

- Prefix search:
  - Find all strings that start with “tab”:
    - “table”; “tabular”; “tablet”; ….
- Subsequence search:
  - Find all strings that contain the subsequence “ark”:
    - “marketing”; “spark”; “quark”
  - Find all occurrences of “acd”:
    - “aabacdcdabdcababdacddcab.”
- Sequence similarity:
  - “table” vs. “cable”?
  - “table” vs. “tale”?
  - “table” vs. “tackle”?

Reminder: Edit Distance

- Let P (of size N) and Q (of size M) be two sequences.

- Given a sequence of edit operations, E

$$C(E) = \sum_{e_i \in E} C(e_i)$$

- Edit distance

$$D(String_1, String_2) = \min_{E \text{ takes } String_1 \text{ to } String_2} \{C(E)\}$$

- Cost: O(N*M)

Cross-parsing Distance

- Let P (of size N) and Q (of size M) be two sequences
- c(P|Q): cost of parsing P with respect to Q
  1. Find the longest (possibly empty) prefix of P that appears as a string somewhere in Q
  2. Restart from the very next position in P and continue until P is completely parsed.

Cross-parsing Distance

- Let P (of size N) and Q (of size M) be two sequences
- c(P|Q): cost of parsing P with respect to Q
  1. Find the longest (possibly empty) prefix of P that appears as a string somewhere in Q
  2. Restart from the very next position in P and continue until P is completely parsed.

$$D_{crossparse}(P,Q) = \frac{\left(c(P \mid Q) - 1\right) + \left(c(Q \mid P) - 1\right)}{2}$$

- Linear time if the strings are indexed using a suffix tree

Compression Distance

- C(P) is the compressed size of P,
- C(Q) is the compressed size of Q, and
- C(P:Q) is the compressed size of the sequence obtained by concatenating P and Q.

$$D_{NCD}(P,Q) = \frac{C(P:Q) - \min\{C(P), C(Q)\}}{\max\{C(P), C(Q)\}}$$

Filtering

- Given a string P of (length N) and a pattern Q (of length M), determine whether the string P may contain an approximate match to Q

Filtering

- Given a string P of (length N) and a pattern Q (of length M), determine whether the string P may contain an approximate match to Q with at most k errors

- Approach 1: Given a maximum error rate, k,
  - cut the pattern Q into k + 1 pieces
  - verify that at least one piece of Q exists in P exactly

  This is because k errors cannot affect more than k pieces.

Filtering

- Given a string P of (length N) and a pattern Q (of length M), determine whether the string P may contain an approximate match to Q with at most k errors

- Approach 2: Given a maximum error rate, k,
  - slide a window of length M over the string P and count the number of symbols that are included in the pattern Q
  - only windows that have at least M – k matching symbols need to be considered

Fingerprinting with w-grams

- **w-grams:** Given a sequence P, its w-grams are obtained by sliding a window of length w over P.

Common w-gram counting

- Given a string P of (length N) and a string Q (of length M), determine whether the two strings may match each other with at most k errors
  - **w-grams:** Given a sequence P, its w-grams are obtained by sliding a window of length w over P.

- Approach (common w-gram counting):
  - Identify (M − w + 1) w-grams of the query string Q
  - Each mismatch between Q and P can affect w many w-grams
    - given an upper bound of k errors, at least (M − w + 1 − kw) w-grams must match
  - Search for these matches using a suffix tree (in linear time)

String kernels

- Given a string P of (length N) and a string Q (of length M), determine whether the strings may approximately match each other
  - **w-grams:** Given a sequence P, its w-grams are obtained by sliding a window of length w over P.

- Approach (string kernels):
  - Identify all w-grams of the query sequence Q; create a counting vector, q
  - Identify all w-grams of the data sequence P; create a counting vector, p
  - Measure the (dot product) similarity of the two counting vectors
    - If the dot product similarity is low, then P is not likely to match Q

Min-sampling similarity

- Given a string P of (length N) and a string Q (of length M), determine whether the strings may approximately match each other
  - **w-grams:** Given a sequence P, its w-grams are obtained by sliding a window of length w over P.
- Approach (min-sampling similarity):
  - Consider r random hash orders of the w-grams of P and Q
  - For each order o_i
    - Find the smallest B w-grams of string P based on the chosen order
    - Find the smallest B w-grams of string Q based on the chosen order
    - If they agree, $match_i(P, Q) = 1$
  - After computing the match for all r orders

$$sim(P,Q) = \frac{\sum_{o_i} match_i(P,Q)}{r}$$

Summary

- Edit distance can be costly for matching long strings
  - Cross-Parsing and Compression-Distance can be used to approximate the edits distance comparison

- W-grams (more commonly known as n-grams) can be used to help filter unpromising candidates before more costly distance computations