DSCI 575: Advanced Machine Learning

Sequence Mining
Winter 2017
Sequence Mining

• Finding patterns in data organized according to a sequence:
  – Customer purchases:
    • ‘Star Wars’ followed by ‘Empire Strikes Back’ followed by ‘Return of the Jedi’.
  – Stocks/bonds/markets:
    • Stocks going up followed by bonds going down.
  – Environmental:
    • CO$_2$ going up is followed by temperatures going up.
  – Website/telephone system navigation.
  – Biological sequences.
    • DNA: ATGCTTCGGCAAGACTCAAAAAATA...
    • RNA: ATGCUUCGGCAAGACUCAAAAAAUA...
    • Protein: GIVEQCCTSICSLYQLENYCN
Sequential Pattern Analysis

• In data mining, called **sequential pattern analysis**:  
  – If you buy product A, are you likely to buy product B at a later time?

• **Similar to association rules, but now order matters.**  
  – Many issues stay the same.

• Exist **sequential generalization of association rule methods**:  
  – Generalized sequential pattern (GSP) algorithm is like a priori algorithm.

• We’re going to instead focus on methods from **bioinformatics**...
Biological Sequences

• We are generated huge quantities of biological data.
• Much of it is stored as sequences.
  – DNA, RNA, and proteins.

Whole Genome Sequencing

• First single-celled organisms’ genomes sequenced in late 90s.
• Many animals/plants in early 2000s.
• Human genome project finished in 2003.
• Late 2000s and 2010s:
  – Characterizing variation and function.
  – HapMap, ENCODE, 1000 genomes, 23andMe.
  – Potential to study infrequent variations.
  – New insights into rare diseases.
  – Promise of personalized medicine.
• Way more data than understanding:
  – One of most important scientific problems.

Bioinformatics

• **Bioinformatics**: biology and databases and data analysis.
  – It’s a huge area, with many interesting variations on DM/ML methods.
• Big focus on **analyzing sequences**.
  – We’ll discuss some of the classic ideas today.
• But **sequences aren’t everything**:
  – How do molecules ‘fold’ in three-dimensions?
  – Which molecules can ‘fit’ together?
  – What genes perform similar functions?
  – How do molecule concentrations affect each other?
  – What are signaling ‘pathways’?

https://en.wikipedia.org/wiki/Bioinformatics
Finding/Testing Similar Sequences

• A classic bioinformatics problem:
  – You find an interesting part of a biological sequence.
    • E.g., this gene makes your mice live much longer or immune to a disease.
  – Do similar sequences appear elsewhere?
    • Either in the same organism, or in other organisms.

• Want to test relatedness of sequences and find related sequences.
  – Heavy use of dynamic programming.
  – Other tricks to handle huge datasets.

• We’ll start from simplest case, and get more complicated.
String Search

- Simplest variant is string search:
  - We have a sequence of length ‘n’
  - We have a query of length ‘m’.
  - Does query occur in sequence?

- Example:
  - Sequence: “GIVEQCCTSICSQLYQLENYCN” (insulin).
  - Query: “TSI”.

- Naïve algorithm:
  - For each of ‘n’ positions, test whether the string starts there.
  - Cost is O(nm).

- Several algorithms reduce this to O(n + m) (e.g., Knurth-Morris-Pratt).
Longest Common Substring

• What if we have multiple queries for same sequence?
  – Sequence: “GIVEQCCTSICSLYQLENYCN”.
  – With ‘k’ queries of length <= 3, cost is $O(n + km)$ with suffix trees.

• A related problem is longest common substring:
  – Sequence 1: “GIVEQCCTSICSLYQLENYCN” (human).
  – Sequence 2: “GIVEQCCASVCSLYQLENYCN” (cow).
  – What is longest string that occurs in both sequences?
    • In this case it’s “CSLYQLENYCN”.

• Suffix trees solve this problem in $O(n + m)$. 
(pause)
Longest Common Substring vs. Subsequence

• Consider human/pig/cow insulin:
  – Sequence 1: “GIVEQCTSICSLOYQLENYCN” (human).
  – Sequence 2: “GIVEQCCASVCSLOYQLENYCN” (cow).
  – Sequence 3: “GIVEQCTSICSLOYQLENYCN” (pig).

• Longest substring between human/pig is 22 (entire sequence).
• Longest substring between human/cow is 11: “CSLOYQLENYCN”.
  – But have we really cut the similarity in half?

• Longest common subsequence:
  – Longest exact match by deleting characters.
Longest Common Subsequence

• **Longest common subsequence (LCS):**
  – Sequence 1: “GIVEQCTSI\text{CSLYQLENYCN}” (human).
  – Sequence 2: “GIVEQCCASV\text{CSLYQLENYCN}” (cow).
  – LCS: “GIVEQCC[ ]S[ ]CSLYQLENYCN”.

• Basis of most ‘diff’ commands (and version control like git).

• Finding LCS by brute force:
  – $2^n$ possible deletions in sequence 1.
  – $2^m$ possible deletions in sequence 2.
  – $O(\min(n,m)2^{n+m})$.

• Can we do better?
Longest Common Subsequence

• Suppose we have the LCS for two sequences:
  – Sequence 1: “ACE”.
  – Sequence 2: “ABCD”.
  – LCS: “AC”.
• Key idea: it’s easy to update LCS if we append one character.
  – Updated sequence 2: “ABCDE”.
  – New LCS: “ACE”.
  – Either the new character extends LCS or not: compute this in O(m).
• O(mn)-time Algorithm:
  1. Start with all of sequence 1 and empty sequence 2 (LCS = []).
  2. Sequentially append sequence 2 to sequence 2, tracking LCS.
Dynamic Programming

• LCS algorithm is special case of dynamic programming.

• Dynamic programming efficiency requires two ingredients:
  1. Optimal substructure:
     • Can efficiently solve the problem given solutions to ‘sub-problems’ (i.e. recursion).
     • For LCS: we can quickly solve problem of length ‘m’ given solution of length (m-1).
  2. Overlapping sub-problems:
     • Limited number of *different* possible sub-problems.
     • For LCS: there are only $O(mn)$ possible lengths for the two strings.

• Key trick: store solutions of sub-problems, instead re-computing.
  – Guarantees each sub-problem is solved at most once.
LCS with Dynamic Programming

• Let’s define the LCS recursively:

\[
LCS(X_i, Y_j) = \begin{cases} 
\emptyset & \text{if } i = 0 \text{ or } j = 0 \\
LCS(X_{i-1}, Y_{j-1}) \sim x_i & \text{if } x_i = y_j \\
\text{longest}(LCS(X_i, Y_{j-1}), LCS(X_{i-1}, Y_j)) & \text{if } x_i \neq y_j 
\end{cases}
\]

• Exponential number of recursive calls in naïve method:
LCS with Dynamic Programming

- Let’s define the LCS recursively:

\[
LCS(X_i, Y_j) = \begin{cases} 
\emptyset & \text{if } i = 0 \text{ or } j = 0 \\
LCS(X_{i-1}, Y_{j-1}) \cup x_i & \text{if } x_i = y_j \\
\text{longest}(LCS(X_i, Y_{j-1}), LCS(X_{i-1}, Y_j)) & \text{if } x_i \neq y_j
\end{cases}
\]

- \(O(mn)\) recursive calls with dynamic programming method:

Store results in table:

\[
\begin{array}{c}
(n, m) \\
| \\
| \\
(n-1, m) \\
| \\
(n-2, m) \\
| \\
(n-3, m) \\
\end{array}
\]

\[
\begin{array}{c}
(n, m-1) \\
| \\
(n-1, m-1) \\
| \\
(n-2, m-1) \\
\end{array}
\]

\[
\begin{array}{c}
(n, m-2) \\
| \\
(n-1, m-2) \\
| \\
(n-2, m-2) \\
\end{array}
\]

Don't recompute

https://en.wikipedia.org/wiki/Longest_common_subsequence_problem
Edit Distance

• LCS considers deletions of elements.

• We might also consider replacements:
  – Sequence 1: “GIVEQCTSICSLYQLENYCN”.
  – Sequence 2: “GIVEQCCASVCSLYQLENYCN”.
  – Where different replacements have different ‘costs’.
    • Some proteins can be substituted and molecule will be similar, some are disastrous.

• Edit distance:
  – Min ‘cost’ of turning string 1 into 2 via additions/deletions/replacements.
  – Can also be computed by dynamic programming:
    • Minimize over the 3 operations.
Edit Distance

- Edit distance between strings ‘X’ and ‘Y’ is $ED(X_m, Y_n)$ where is min of:

$$ED(X_i, Y_j) = \begin{cases} 
ED(X_{i-1}, Y_{j-1}) & \text{if } X_i = Y \\
\min \left\{ \begin{array}{ll}
ED(X_{i-1}, Y_j) + \text{cost}('delete X_i') \\
ED(X_i, Y_{j-1}) + \text{cost}('insert Y_j') \\
ED(X_{i-1}, Y_{j-1}) + \text{cost}('replace X_i with Y_j') \\
\end{array} \right. & \text{if } 1 < i \leq n \text{ and } X_i \neq Y, \\
\text{cost('delete X_1')} & \text{if } j = 0, \\
\text{cost('insert Y_1')} & \text{if } i = 0, \\
\end{cases}$$

- Cost is still $O(mn)$, and if costs are non-negative this is a distance.
(pause)
Local Edit Distance / Local Alignment

• Local alignment (Smith-Waterman):
  – Positive ‘score’ for matches, negative ‘score’ for add/delete/replace.
  – Set negative ‘d_{ij}’ values to zero, and maximize d_{ij} over ‘i’ and ‘j’.
    • Note that in bioinformatics you maximize ‘score’ rather than minimize ‘distance’.
  – Finds substrings with small edit distance:

Smith-Waterman Scoring

<table>
<thead>
<tr>
<th></th>
<th>D</th>
<th>E</th>
<th>S</th>
<th>I</th>
<th>G</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>D</td>
<td>0</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>E</td>
<td>4</td>
<td>10</td>
<td>9</td>
<td>8</td>
<td>7</td>
<td>6</td>
</tr>
<tr>
<td>A</td>
<td>3</td>
<td>9</td>
<td>9</td>
<td>8</td>
<td>7</td>
<td>6</td>
</tr>
<tr>
<td>S</td>
<td>0</td>
<td>2</td>
<td>8</td>
<td>14</td>
<td>13</td>
<td>12</td>
</tr>
</tbody>
</table>

Match = +5
Mismatch = -1
Gap = -1

Aligned:
1: DESIGN
2: IDEAS

http://2012.igem.org/Team:Johns_Hopkins-Software/Cloud
**BLAST**

- **Basic Local Alignment Search Tool (BLAST):**
  - A method for searching biological sequences.
  - Most cited paper in 1990s of all of science.
- **Setup:**
  - We have a huge database of sequences.
  - Individual sequences may be very long (human genome: ~3.2 billion).
  - Quickly find similar sequences to a query sequence.
- **Key ideas:**
  - Find interesting and short substrings in query.
  - Fast phase: Find ‘candidates’ that contain any substring.
  - Slow phase: apply dynamic programming on the ‘candidates’.
  - Some other tricks to make it faster.

\[
\text{If there are all 'l' strings in database, finding indices of 'k' substrings of length 'm' costs } O(k \cdot m \cdot l) .
\]

\[
\text{No dependence on length of database sequences.}
\]
BLAST

- BLAST:
  - Sequence database:
  - Disadvantage:
    - You could have false negatives in the first phase (you miss distantly-related sequences).
- PSI-BLAST:
  - Re-run with related sequences to find more distantly-related sequences.
- Related to hashing tricks for finding elements of a set:
  - Bloom filter: guaranteed to have no false negatives.
  - Count-min sketch: more recent probabilistic/online method.
Generalizations of Edit Distance

• We can have score based on insertion/deletion length (‘gap score’)
• Other common mutations:
  – Reversal:
  
  \[ \text{ATGATATCGCGA} \quad \text{ATGGCGCTATAA} \]

  – Transposition:

  – In general, we can’t handle these efficiently: sub-problems don’t overlap.
• But some special cases exist:
  – If reversals are ‘contained’ in each other, solve as ‘context-free grammar’.
Multiple Sequence Alignment

- Multiple Sequence Alignment:
  - We have several sequences and want to jointly align them:

  Dynamic programming is exponential in number of sequences.

https://en.wikipedia.org/wiki/Multiple_sequence_alignment
Multiple Sequence Alignment and Clustering

• Heuristic to avoid exponential cost of multiple sequence alignment:
  – First perform hierarchical clustering.
    • Clustering could be interesting on its own.
  – Align sequences as we go up the tree.

• Popular method is Clustal:
  – 3 of top 15 all-time most-cited science papers:
    • BLAST, PSI-BLAST, Clustal.

http://figshare.com/articles/_Multiple_sequence_alignment_and_phylogenetic_tree_of_the_OVATE_domains_of_OFP_proteins_from_Arabidopsis_and_tomato_/1333022
Summary

• **Sequence data** arises in applications involving time/strings.
  – Common substrings can be found in linear time.
  – Edit distance can be found efficiently using dynamic programming.
  – BLAST combines the above two with other tricks.

• **Multiple sequence alignment** considers multiple sequences.