CS 181: Natural Language Processing

Lecture 4: Dynamic Programming & Minimum String Edit Distance

Kim Bruce
Pomona College
Spring 2008

Disclaimer: Slide contents borrowed from many sources on web!

Catching Typos

- Recognizing misspellings easy - check dictionary
  - But lots of suffixes and prefixes: use fsa!
- What about making corrections to isolated words?
  - Look for spellings that are "close" to word
- Context-dependent errors detection/correction
  - Transpositions may accidentally create real words!

Spelling Correction

- Map words into equivalence classes that likely hold correct spelling.
  - CS 51 lab: canonize words by removing vowels and doubled consonants: canonize lab
  - Find all words w/same canonization as word.
- Alternatively, develop metric and find real world closest to word.
  - Use minimum edit distance

Minimum Edit Distance

- Can convert any word to another by series of additions, deletions, and substitutions.
  - Once specify cost of each operation then can measure distance between them
  - We'll use 1 for cost of addition/deletion, 2 for substitution.
  - Use same algorithm if choose different costs, but get different answer.

Example

- Convert "INTENTION" to "EXECUTION"
- `INTENTION`

Example

- Convert "INTENTION" to "EXECUTION"
- ` NTENTION`   *delete I*

cost = 1

Example

- Convert "INTENTION" to "EXECUTION"
- `ETENTION` *subst E for N*

cost = 3

Example

- Convert "INTENTION" to "EXECUTION"
- `EXENTION` *subst X for T*

cost = 5

Example

- Convert "INTENTION" to "EXECUTION"
- `EXECTION` *subst C for N*

cost = 7

Example

- Convert "INTENTION" to "EXECUTION"
- `EXECUTION` *insert U*

cost = 8

Optimality

- How can we know if that is the minimal edit distance?
- Check all possible conversions? Too many!

Solving Problems

- Optimal substructure property: The optimal solutions to a problem contain optimal solutions to its subproblems.
- Ex: Shortest distance from LA to NYC
  - If shortest path goes through Chicago then portion of the path from LA to Chicago and from Chicago to NYC are also optimal.

Dynamic Programming

- If problem has overlapping subproblems (solve same problem repeatedly) & optimal substructure property then can use dynamic programming.
- Key idea is to save solutions to subproblems so don't have to recalculate
  - Memoization!
- Can do top-down or bottom-up
  - We'll do bottom-up

Minimum Edit Distance

- What is minimal cost of transforming v to w?
- Transform to problem with subproblems.
- Define distance[i,j] to be min cost of transforming v[1..j] to w[1..i]
- Does it satisfy optimal substructure property?
- Does it have overlapping subproblems?

Minimum Edit Distance

- Let |v| = m, |w| = n
- Consider last move in aligning. 3 choices:
  - Add move: Take moves changing v to w[1:n-1] & insert w[n]
  - Delete move: Take moves changing v[1:m-1] to w & delete v[m]
  - Replace move: Take moves changing v[1:m-1] to w[1:n-1] & change v[m] to w[n]

Minimum Edit Distance

- Recursive solution:

$$\text{distance}[i,j] = \min \begin{cases} \text{distance}[i-1,j] + \text{ins\_cost}(w_i) \\ \text{distance}[i-1,j-1] + \text{sub\_cost}(w_i,v_j) \\ \text{distance}[i,j-1] + \text{del\_cost}(v_j) \end{cases}$$

ins_cost = 1

del_cost = 1

sub_cost = 0 if $w_i = v_j$, 2 otherwise

DISTANCE[I,J]

	#	P	A	R	K
#	0	1	2	3	4
S	1	2	3	4	5
P	2	1	2	3	4
A	3	2	1	2	3
N	4	3	2	3	4
K	5	4	3	4	3

Recover edits from table

```
def minEditDist(target, source):
    n = len(target)
    m = len(source)

    # m+1 rows, n+1 cols
    distance = [[0 for i in range(n+1)] for j in range(m+1)]

    for col in range(1,n+1):
        distance[0][col] = distance[0][col-1] + 1
    for row in range(1,m+1):
        distance[row][0]= distance[row-1][0] + 1

    for col in range(1,n+1):
        for row in range(1,m+1):
            distance[row][col] = min(
                distance[row-1][col] + 1,
                distance[row][col-1] + 1,
                distance[row-1][col-1]+substCost(source[row-1],target[col-1]))
    return distance[m][n]
```

Variants & Improvements

- Needleman-Wunch distance: cost of substitution varies depending on characters
  - E.g., distance btn characters nearby is less
- Want to match names: Kim Barry Bruce, Kim B. Bruce, K. B. Bruce, Kim Bruce, K. Bruce.
  - One idea: n character gap costs less than n gaps of length 1.

N-Grams

N-grams

- N-gram is sequence of N words that occur sequentially in text
- Determine probabilities of N-grams
- Use to predict which word is most likely to be correct in context.
- Can help in spelling correction

Using context

- Spell-checking:
  - They are leaving in about 15 minuets.
- Part of speech tagging
  - Which meaning of "dogs"
- Machine translation
- Speech & handwriting recognition
  - Compare possible word decodings
- Authorship identification

Which is most probable?

- First Example:
  - ... I think they're OK ...
  - ... I think there OK ...
  - ... I think their OK ...
- Second Example:
  - ... by the way, are they're likely to ...
  - ... by the way, are there likely to ...
  - ... by the way, are their likely to ...

Which is most probable?

- Third Example:
  - How do you wreck a nice beach?
  - How do you recognize speech?
- Fourth Example:
  - Put the file in the folder
  - Put the file and the folder

Counting Words

- Types vs Tokens
  - "They picnicked by the pool, then lay back on the grass and looked at the stars"
  - 16 tokens, 14 types
  - Shakespeare: 884,647 tokens, 29,006 types
  - Also interested in number of lemmas
    - Remove affixes

Language Models

- Develop a "language model" to help us predict the likelihood of strings.
- In English:
  - P(*the big dog*) > P(*dog big the*) > P(*dgo gib eth*)
- How can the computer know this?
- Each sentence is sequence $w_1, \ldots, w_n$
- How determine $P(w_1, \ldots, w_n)$

N-Grams

- Computes a probability for observed input
- Probability is likelihood of observation being generated by same source as training data.
- Different models arise from different training sets: English vs. French
- Problems!

Any Questions?