University of Malta
BSc(IT) HonsYear IV

CSA4050: Advanced Topics in NLP

Statistical NLP

Minimum Edit Distance

- Multiple Errors
- Minimum Edit Distance Concepts
- MED Algorithm

Dept Computer Science and AI 2003/04
Lecturer: Michael Rosner

Distances Between Strings

- So far we have only considered single spelling errors. To handle multiple spelling errors within the same word, we need a *richer notion of distance*.
- The distance from a string **a** to a string **b** is defined as the *minimum length* of any acceptable analysis of the difference between **a** and **b**.
- A basic approach is to analyse the total difference between 2 strings as into a series of individual elementary differences, each achieved by a *primitive editing operation*.
- Obviously, the length of such an analysis will depend on the on the chosen set of primitive operations.

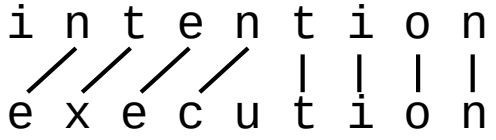
Primitive Editing Operations

- Usually the following are used:
  - substitution
  - deletion
  - insertion
- Others are possible, e.g. transposition. However this can be defined in terms of the more elementary operations (in this case, a pair of substitutions).
- It is also crucial to realise that in general there will be more than one series of editing operations that will achieve a given result. To give a simple example, there are two ways to turn "acress" into "acres".

Three Different Styles

The following illustrates three ways in which the difference between strings can be presented:

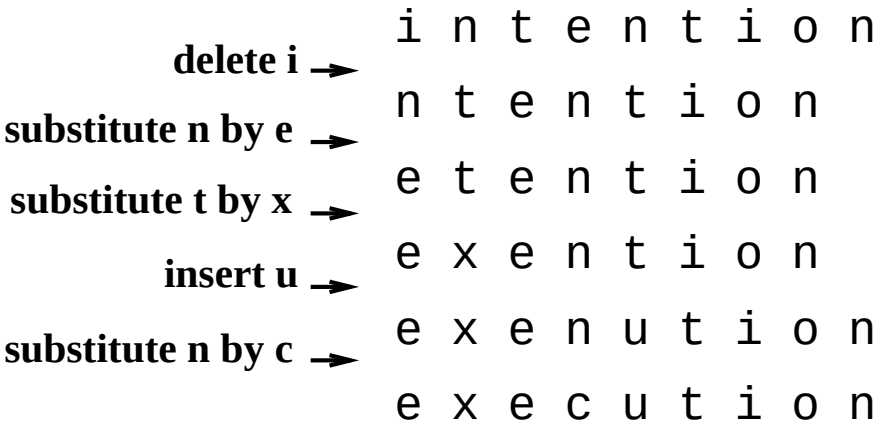
Trace



Alignment

i n t e n ε t i o n
ε e x e c u t i o n

Operation List



These represent different modes of *analysis* as well as of presentation.

Trace, Alignment and List

- Trace: merely records corresponding character positions (different letters at each end of a line indicate a substitution).
- Alignments are more detailed than traces (same trace can result in several alignments).

1 trace

```
g  a  b  h
|        |
g  c  d  h
```

2 alignments

```
g  a  b  -  -  h
g  -  -  c  d  h

g  a  -  b  -  h
g  -  c  -  d  h
```

- List is yet more detailed than alignment. Again, a given alignment can be realised by several different operation lists.

Levenshtein Distance

The Levenshtein distance between two sequences assumes that each of the three operations has a cost of 1. Thus the Levenstein distance between *intention* and *execution* is 5.

Levenstein also proposed an alternate version of his metric in which each insertion or deletion has a cost of 1 and substitution (represented as 1 insertion followed by 1 deletion) has a cost of 2. Under this model the distance between *intention* and *execution* would be 7.

We can also weight operations by more complex functions, e.g. using confusion matrices which assign a probability to each. We can then talk about the "maximum probability alignment" between strings rather than the edit distance.

Computing Minimum Edit Distance

- The minimum edit distance can be computed by **dynamic programming**, the name for a class of algorithms, first introduced in 1957 by Bellman.
- The main characteristic of this class is that *table driven methods* are used to solve a problem for properly *combining the solutions to subproblems*.
- Various tabular parsing algorithms fall into this class (e.g. Earley's Algorithm - cf. the Star operation). Another item in the class is the Viterbi algorithm which is used to discover which word best fits a given pattern of phonemes.
- In the case of the minimum edit distance algorithm we focus on the minimum edit distance between s(i) (the first i characters of the source) and t(j), the first j characters of the target.

Computing Minimum Edit Distance

The basic intuition is that MED between s(i) and t(j) is related to the minimum of the MEDs of three other pairs of strings, namely:

MED(s(i-1),t(j))
MED(s(i),t(j-1))
MED(s(i-1),t(j-1))

For example, suppose the word on the page is FLYD instead of FLIES.

MED(FLIES,FLYD) is related to each of

1. MED(FLIE,FLYD)
2. MED(FLIES,FLY)
3. MED(FLIE,FLY)

Three Ways to Compute MED(FLIES,FLYD)

1. MED(FLIES,FLYD) = MED(FLIE,FLYD)
   + cost(del(S) from source)

2. MED(FLIES,FLYD) = MED(FLIES,FLY)
   + cost(ins(D) into target))

3. MED(FLIES,FLYD) = MED(FLIE,FLY)
   + cost(subst(source(S) with target(D))

The Recurrence Relation

To get the *minimum* edit distance we have to take the minumum of these quantities. A relation can be defined that specifies the value of $d_{i,j}$ as follows:

$$d_{i,j} = \text{min} \begin{cases} d_{i-1,j} + \text{del-cost(s[i])} \\ d_{i-1,j-1} + \text{subst-cost(s[i],t[j])} \\ d_{i,j-1} + \text{ins-cost(t[j])} \end{cases}$$

The Algorithm

Here is the corresponding algorithm which stores intermediate results in table.

```
function med(s,t) returns min distance

% initialisation
%
m <- length(s)
n <- length{t)
new array d[m+1,n+1]
d[0,0] = 0
for i = 1 to m do d[i,0] = d[i-1,0] + del-cost(s[i])
for j = 1 to n do d[0,j] = d[0,j-1] + ins-cost(t[j])

% main loop
%
for i = 1 to m do
  for j = 1 to n do
      d[i,j] = min( d[i-1,j] + del-cost(s[i]),
                    d[i,j-1] + ins-cost(t[j]),
                    d[i-1,j-1] + subst-cost(s[i],t[j]) )
```

Result of Running MED Algorithm

source: FLIES - m = 5

target: FLYD - n = 4

4	D						
3	Y						
2	L						
1	F						
0	#						
j		#	F	L	I	E	S
	i	0	1	2	3	4	5

After Initialisation

4	D	4					
3	Y	3					
2	L	2					
1	F	1					
0	#	0	1	2	3	4	5
j		#	F	L	I	E	S
	i	0	1	2	3	4	5

Result of Running MED Algorithm

After 1st Inner Loop

4	D	4	3				
3	Y	3	2				
2	L	2	1				
1	F	1	0				
0	#	0	1	2	3	4	5
j		#	F	L	I	E	S
	i	0	1	2	3	4	5

Final Result

4	D	4	3	2	3	4	5
3	Y	3	2	1	2	3	4
2	L	2	1	0	1	2	3
1	F	1	0	1	2	3	4
0	#	0	1	2	3	4	5
j		#	F	L	I	E	S
	i	0	1	2	3	4	5