School of Computing and Information Systems
The University of Melbourne
COMP90049 Knowledge Technologies (Semester 2, 2018)
Workshop sample solutions: Week 4

Suppose that we have observed the token `lended`, and we have a dictionary as follows:

```
addendum
blenders
commodity
deaden
end
leader
leant
lent
lemonade
pleading
```

1. Which, if any, of the above dictionary entries, would be returned using a Neighborhood Search with a neighbourhood of 2? 3?

   - With a neighbourhood size of 2, there is a dictionary entry:
     - leader, by *Replacing* the n with a, and the second d with r
   - Along with the above, the following are also within a neighbourhood of 3:
     - blenders, by *Inserting* the b, *Replacing* the second dwith r, and *Inserting* the s
     - deaden(three *Replaces*)
     - end(three *Deletions*)
     - lent (one Replace and two *Deletions*)

2. With respect to the input string `lended` and the dictionary entry `deaden`, calculate the following:

   (a) the Global Edit Distance, using the parameter $[m, i, d, r] = [+1, -1, -1, -1]$

(a)	ε		l		e		n		d		e		d
ε	0	←	-1	←	-2	←	-3	←	-4	←	-5	←	-6
	↑	↖		↖		↖		↖				↖	
d	-1		-1	←	-2	←	-3		-2	←	-3	←	-4
	↑	↖	↑	↖						↖			
e	-2		-2		0	←	-1	←	-2		-1	←	-2
	↑	↖	↑		↑	↖		↖			↑	↖	
a	-3		-3		-1		-1	←	-2		-2		-2
	↑	↖	↑		↑	↖	↑	↖				↖	
d	-4		-4		-2		-2		0	←	-1		-1
	↑	↖	↑	↖	↑	↖	↑		↑	↖			
e	-5		-5		-3		-3		-1		1	←	0
	↑	↖	↑		↑	↖			↑		↑	↖	
n	-6		-6		-4		-2		-2		0		0

From the above table, we can observe that the Global Edit Distance is 0, corresponding to the following sequence of operations: *Replace*, *Match*, *Replace*, *Match*, *Match*, *Replace*, which can be abbreviated as `rmrmmr`. (You can follow along with the highlighted back pointers.)

(b) the Local Edit Distance, using the parameter $[m, i, d, r] = [+1, -1, -1, -1]$

(b)	ε	l	e	n	d	e	d
ε	0	0	0	0	0	0	0
d	0	0	0	0	1	0	1
e	0	0	1	0	0	2	1
a	0	0	0	0	0	1	1
d	0	0	0	0	1	0	2
e	0	0	1	0	0	2	1
n	0	0	0	2	1	1	1

From the above table, we can observe that the Local Edit Distance is 2 (highlighted); there are five equivalent-scoring substring matches that it corresponds to:

- Align -de- in `lended` with the first `de`- in `deaden`: **mm**
- Align -ded with `dead`-: **mmim**
- Align -de- in `lended` with the second -de- in `deaden`: **mm**
- Align -ende- with -eade-: **mrmm**
- Align -en- with -en: **mm**

Note: It worth mentioning that Global Edit Distance algorithms (such as Needleman-Wunsch algorithm or Levenshtein distance algorithm) minimize global edit distance at the price of splitting query string into different fragments and matching different parts of target strings separated by large parts of unmatched strings. Therefore, these algorithm are useful in domains such as spelling corrections.

Whereas Local Edit Distance Algorithms (such as Smith-Waterman algorithm) tries to match much larger parts of query string with the contiguous parts of target string and hence minimizing local edit distance. So in domains such as genome sequence matching and detecting plagiarism where identifying large chunks of contiguous matching strings is essential, Local Edit Distance Algorithms can be very useful.

(c) the N-Gram Distance, using $n = 2$

We begin by generating the 2-grams of the two strings; I will use the terminal marker (#) here:

- `lended: #l, le, en, nd, de, ed, d#`
- `deaden: #d, de, ea, ad, de, en, n#`

Recall that the N-Gram Distance is defined as follows:

$$D(s, t) = |\, G_n(s) \,| + |\, G_n(t) \,| - 2 \times |\, G_n(s) \cap G_n(t) \,|$$

Here we have 7 '2-grams' in `lended`, as well as 7 in `deaden`.

Also, the two sets share 2 '2-grams': '`de`' and '`en`'. (Note that we don't double-count the '`de`'s in `deaden`, because there is only a single '`de`' in `lended`)

Consequently, the 2-gram Distance is 7 + 7 − 2 × 2 = 10.

(d) the Jaro-Winkler Similarity, using $\ell <= 4$ and $p = 0.1$

We first need to find the *m* (number of matches in our window of search). Here our window of search (the matches we consider good enough) is :

$$\left\lfloor \frac{\max(|a|,|b|)}{2} - 1 \right\rfloor = \left\lfloor \frac{\max(6,6)}{2} - 1 \right\rfloor = 2$$

This means that for each letter in 'lended' we are looking for a match letter in 'deaden' in a window of ± 2 position.

```
d , e , a , d , e , n
    |       |   |
l , e , n , d , e , d
```

Here *m* is 3. Also all these matches have the same position (no transposition), therefore in this example t = 0.

Recalling the definition of Jaro-Winkler Similarity:

$$Sim_j(a,b) = \frac{1}{3}\left(\frac{m}{|a|} + \frac{m}{|b|} + \frac{m-t}{m}\right)$$

$$Sim_w(a,b) = Sim_j(a,b) + \ell \times p \times (1 - Sim_j(a,b))$$

For the first part (Jaro part), we have:

$$Sim_j('deaden','lended') = \frac{1}{3}\left(\frac{3}{6} + \frac{3}{6} + \frac{3-0}{3}\right) = \frac{2}{3} \cong 0.67$$

For the Winkler part we need to find the true value of ℓ. Here ℓ is the length of true common prefix *between* the two words. Since 'deaden' and 'lended' do not have a common prefix, in this case the value of ℓ is 0. Put it all together we have:

$$Sim_w('deaden',' lended') = 0.67 + 0 \times 0.1 \times (1 - 0.67) = 0.67$$

3. Find the best approximate match (or matches, if there are ties) in the dictionary for the string `lended`, based on the following methods; consider different parameters where necessary:

(a) the Global Edit Distance

Using the above scoring parameter, the closest matches are `blenders (+2)` and `leader(+2)`. You might like to try some other parameter setting, to see if they give different results.

(b) the Local Edit Distance

Using the above scoring parameter, the closest match is `blenders (+5)`. In this case, changing the parameter is unlikely to result in a different answer. (Why?)

(c) the n-Gram Distance (n = 2)

If we are using n = 2 and padding with `#`, the best dictionary entry is `end`, with a 2-Gram Distance of 4.

(d) the n-Gram Distance (n =3)

If we are using n = 3 and padding with `#`, the best dictionary entry is `lent`, with a 3-Gram Distance of 6.

(e) the Jaro-Winkler Similarity ($\ell <= 4$ and $p = 0.1$)

Using Jaro-Winkler similarity, the most similar dictionary entry is ~~`lent`, with the similarity of 0.825~~ `end` with the similarity of 0.833.

```
e,n,d
 \ \ \
l,e,n,d,e,d
```

Here *m* is 3. Also although these matches do not have the same position, they are at the same order (no transposition) therefore in this example t = 0.

$$Sim_j('lended','end') = \frac{1}{3}\left(\frac{3}{6} + \frac{3}{3} + \frac{3-0}{3}\right) \cong 0.833$$

The ℓ parameter is 0 (no common prefix) and so the Winklers part is equal to the Jaro part.

4. Assuming that the "correct" (intended) dictionary entry was `lent`, calculate the *precision* of each of the above methods of finding approximate entries from the dictionary.

   For each method, we will consider how many dictionary entries it returns as a result (predicts as a good match), as well as how many it got correct — in this case, there is only a single correct answer, so the value will be 0 or 1.

   We have quite a few methods above!

   - Neighbourhood Search (with the neighbourhood of 2): there was one entry returned from the dictionary (`leader`), but it wasn't `lent`, so the precision is $\frac{0}{1} = 0$.
   - Neighbourhood Search (with a neighbourhood of 3): there were five entries returned from the dictionary, and `lent` was one of them. The precision of this system is the number of correct responses (1) out of the total number of attempted responses (5), $\frac{1}{5} = 20\%$.
   - Global Edit Distance: there were two results from the dictionary (`blenders` and `leader`), but no `lent`, so the precision is $\frac{0}{2} = 0$.
   - Local Edit Distance: there was just a single result (`blenders`) which isn't `lent`, so the precision is $\frac{0}{1} = 0$.
   - 2-Gram Distance: there was a single result which was `end`, so the precision is $\frac{0}{1} = 0$.
   - 3-Gram Distance: there was a single result which was `lent`, so the precision is $\frac{1}{1} = 100\%$.
   - Jaro-Winkler similarity: there was a single result which was `lent`, so the precision is $\frac{1}{1} = 100\%$.

   Here, the best methods, according to precision, were Jaro-Winkler similarity and 3-Gram Distance. However, if we wanted to compare these methods more accurately, we would need to aggregate these results over a large number of inputs; just considering a single word is not enough information to draw reliable conclusions.