Edit distance and applications

Anantharaman Narayana Iyer

Narayana dot Anantharaman at gmail dot com

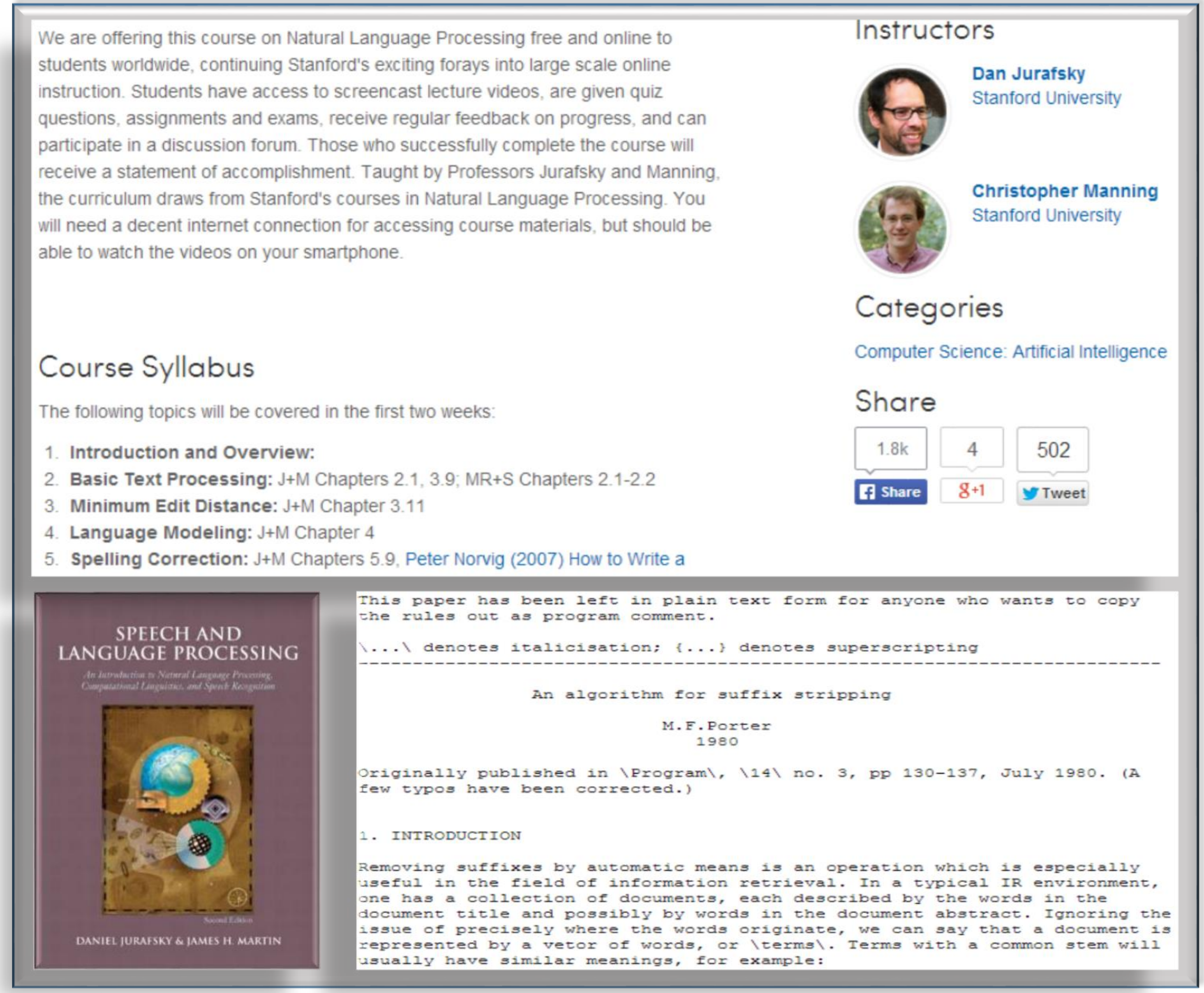
3 Sep 2015

Objective

- Porter stemmer: we will mention it briefly and let you go through this at home
- Discuss in depth the concept of edit distance computation using dynamic programming
- Look at a few applications

References

- Speech And Language Processing, 2nd Edition by Dan Jurafsky and James Martin
- Natural Language Processing, Coursera online learning course by Dan Jurafsky and Christopher Manning
- Home page of Dr Martin Porter: http://tartarus.org/~martin/PorterStemmer/
- Wikipedia (for certain definitions of linguistic terminology)



References: Tools

Fuzzy String Matching in Python

Fuzzy String Matching, also called Approximate String Matching, is the process of finding strings that approximatively match a given pattern.

```
def levenshteinDistance(s1,s2):
    if len(s1) > len(s2):
        s1,s2 = s2,s1
    distances = range(len(s1) + 1)
    for index2,char2 in enumerate(s2):
        newDistances = [index2+1]
        for index1,char1 in enumerate(s1):
            if char1 == char2:
                newDistances.append(distances[index1])
            else:
                newDistances.append(1 + min((distances[index1],
                                             distances[index1+1],
                                             newDistances[-1])))
        distances = newDistances
    return distances[-1]

print(levenshteinDistance("kitten","sitting"))
print(levenshteinDistance("rosettacode","raisethysword"))
```

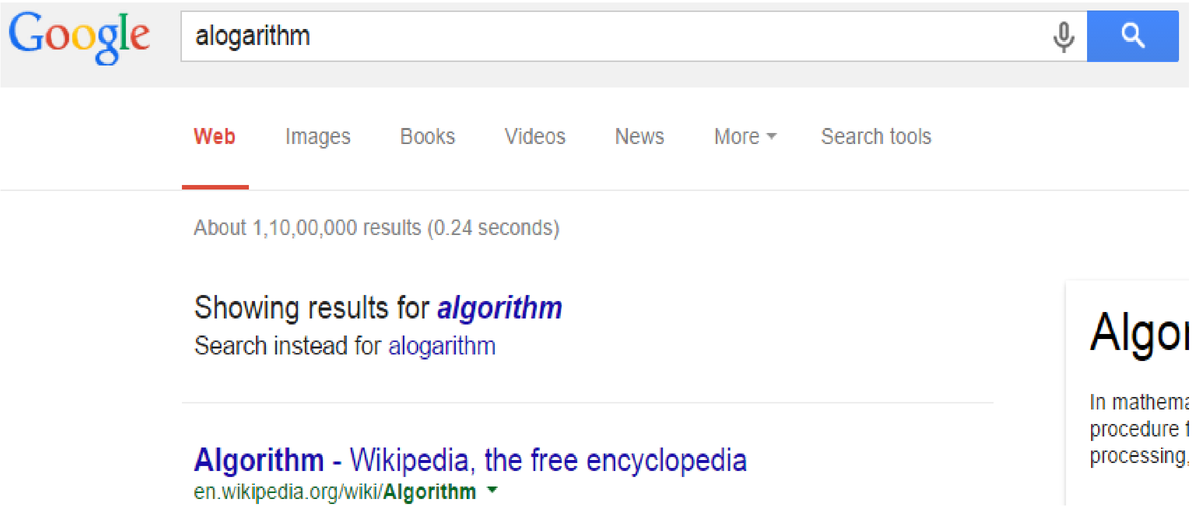
http://rosettacode.org/wiki/Levenshtein_distance

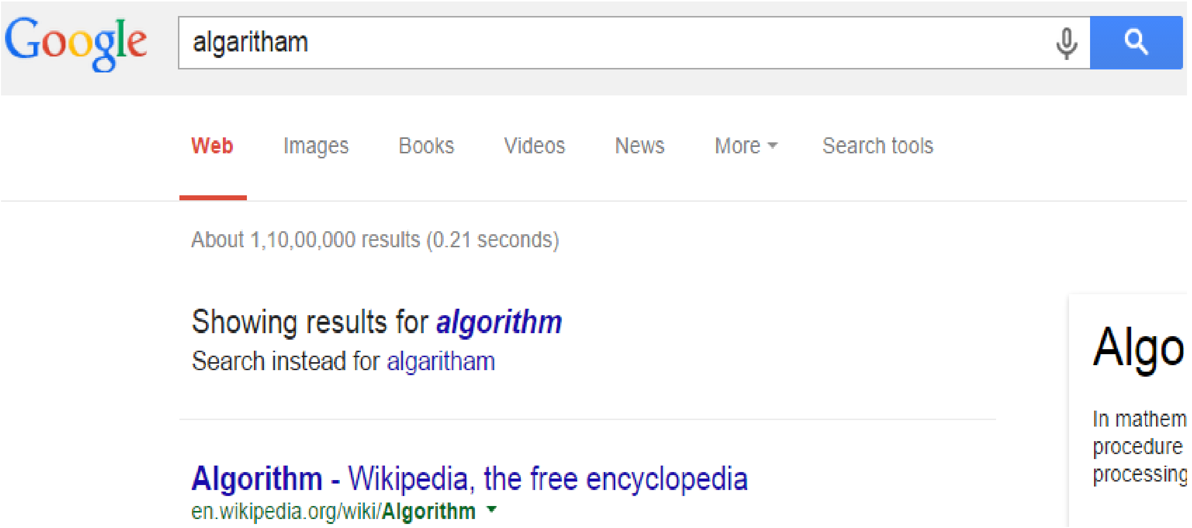
```
import re, collections

def words(text): return re.findall('[a-z]+', text.lower())

def train(features):
    model = collections.defaultdict(lambda: 1)
    for f in features:
        model[f] += 1
    return model

NWORDS = train(words(file('big.txt').read()))

alphabet = 'abcdefghijklmnopqrstuvwxyz'

def edits1(word):
   splits     = [(word[:i], word[i:]) for i in range(len(word) + 1)]
   deletes    = [a + b[1:] for a, b in splits if b]
   transposes = [a + b[1] + b[0] + b[2:] for a, b in splits if len(b)>1]
   replaces   = [a + c + b[1:] for a, b in splits for c in alphabet if b]
   inserts    = [a + c + b     for a, b in splits for c in alphabet]
   return set(deletes + transposes + replaces + inserts)

def known_edits2(word):
    return set(e2 for e1 in edits1(word) for e2 in edits1(e1) if e2 in NWORDS)

def known(words): return set(w for w in words if w in NWORDS)

def correct(word):
    candidates = known([word]) or known(edits1(word)) or known_edits2(word) or [word]
    return max(candidates, key=NWORDS.get)
```

http://norvig.com/spell-correct.html

Edit Distance

- Suppose we have 2 strings x and y and our goal is to transform x to y.
  - Example 1: x = alogarithm, y = logarithm
  - Example 2: x = alogarithm, y = algorithm
- Edit operators
  - Deletion
  - Insertion
  - Substitution
- The weighted number of operations that are needed to transform x to y using the above operators is called the edit distance
  - Typically, deletion and insertion are assigned a weight 1 while substitution has weight 2 (Levenshtein)
- Edit distance is a measure of string similarity

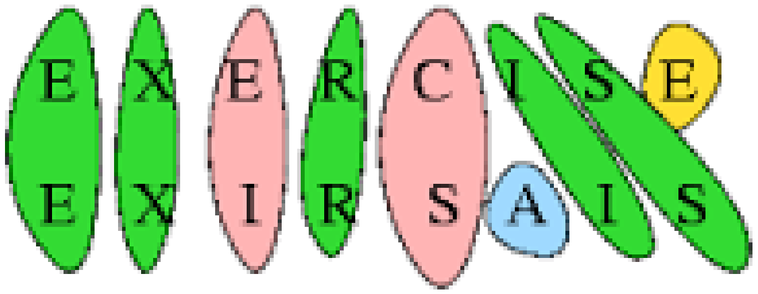


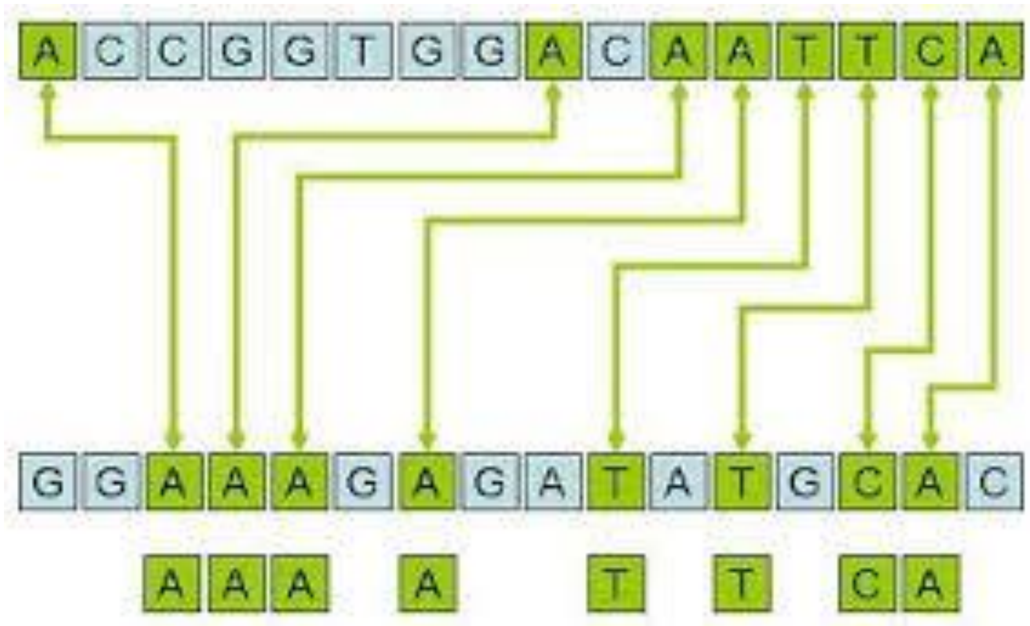


Example

- What is the edit distance between:
  1. x = alogarithm, y = logarithm
  2. x = alogarithm, y = algorithm
- In the example 1, by deleting the letter a from string x, we get the string y. Hence edit distance is 1
- In the example 2, we need the following steps:
  - Step 1: Delete o from x   =>   algarithm
  - Step 2: Replace the 4^{th} letter a with o   => algorithm
- In example 2, we performed 1 deletion and 1 replacement and hence the edit distance = 1 + 2 = 3

Why is this needed?

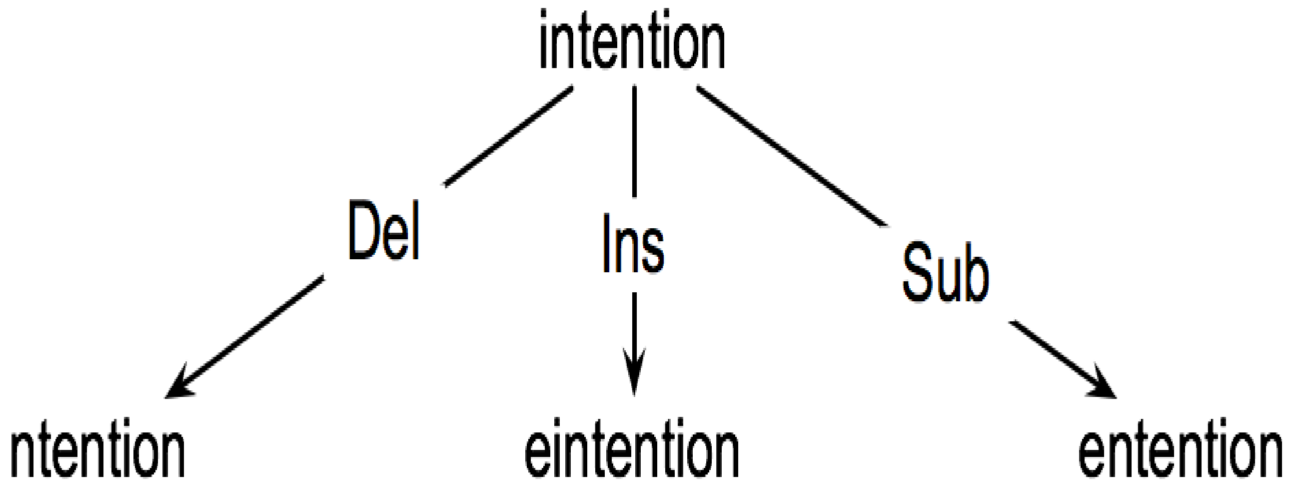




- Applications in Natural Language Processing
  - Spelling correction
  - Approximate string matching
  - Spam filtering
  - Longest Common Subsequence
  - Finding curse words or inappropriate words
  - Finding similar sentences where one sentence differs from the other within the edit distance with respect to the words: Word level matching
    - Adobe announced 4th quarter results today
    - Adobe announced quarter results today
- Computational Biology
  - Aligning DNA sequences
- Speech Recognition

How to find min edit distance?

- Let us consider x = intention y = execution (Ref Dan Jurafsky)
- Initial State: The word we are transforming (intention)
- Final State: The word we are transforming to (execution)
- Operators: Insert, Delete, Substitute
- Path cost: This is what we need to minimize, ie, the number of edit operations weighted appropriately



Minimum Edit Distance

- The space of all edit sequence is huge and so it is not viable to compute the edit distance using this approach
- Many paths to reach the end state from the start state.
- We only need to consider the shortest path: Minimum Edit Distance
- Let us consider 2 strings X, Y with lengths n and m respectively
- Define D(i, j) as the distance between X[1..i], Y[1..j]. This the distance between X and Y is D(n, m)

Dynamic Programming algorithm To compute minimum edit distance

- Create a matrix where the column represents the target string, where there is one column for each symbol and the row represents the source string with one symbol per row.
- For minimum edit distance this is the edit-distance matrix where the cell (i, j) contains the distance of first i characters of the target and first j characters of the source
- Each cell can be computed as a simple function of surrounding cells; thus, from beginning of the matrix it is possible to compute every entry
  - We compute D(i,j) for small *i,j*
  - And compute larger D(i,j) based on previously computed smaller values
  - i.e., compute D(*i,j*) for all *i* ($0 < i < n$) and *j* ($0 < j < m$)

Minimum Edit Distance Algorithm

- Initialization

```
D(i,0) = i
D(0,j) = j
```

- Recurrence Relation*:*

```
For each  i = 1…M
      For each  j = 1…N
                        D(i-1,j) + 1
          D(i,j)= min  D(i,j-1) + 1
                        D(i-1,j-1) +   2; if X(i) ≠ Y(j)
                                       0; if X(i) = Y(j)
```

- Termination*:*

```
D(N,M) is distance
```

Example

- Let X = “alogarithm” and Y = “algorithm”
- We need to transform X to Y using the dynamic programming algorithm
- Let us start off with the base cases:
  - (ε, ε) = 0, (εa, ε) = 1, …
  - (ε, a) = 1, (ε, al) = 2, …
- Recurrence:
  - We need to evaluate the distance between substrings: (εa, εa), (εa, εal), (εa, εalo), (εa, εalog)…(εal, εa), (εal, εal), …, (εalg, εa), (εalg, εal), (εalg, εalo), (εalg, εalog), …

Example (Ref Dan Jurafsky Coursera)

N	9									
O	8									
I	7									
T	6									
N	5									
E	4									
T	3									
N	2									
I	1									
#	**0**	1	2	3	4	5	6	7	8	9
	#	E	X	E	C	U	T	I	O	N

$$D(i,j) = \min \begin{cases} D(i-1,j) + 1 \\ D(i,j-1) + 1 \\ D(i-1,j-1) + \begin{cases} 2; & \text{if } S_1(i) \neq S_2(j) \\ 0; & \text{if } S_1(i) = S_2(j) \end{cases} \end{cases}$$

Example (contd): (Ref Dan Jurafsky Coursera)

N	9	8	9	10	11	12	11	10	9	**8**
O	8	7	8	9	10	11	10	9	8	9
I	7	6	7	8	9	10	9	8	9	10
T	6	5	6	7	8	9	8	9	10	11
N	5	4	5	6	7	8	9	10	11	10
E	4	3	4	5	6	7	8	9	10	9
T	3	4	5	6	7	8	7	8	9	8
N	2	3	4	5	6	7	8	7	8	7
I	1	2	3	4	5	6	7	6	7	8
#	**0**	1	2	3	4	5	6	7	8	9
	#	E	X	E	C	U	T	I	O	N

Backtrace

- Often it is not adequate to just compute minimum edit distance alone. It may also be necessary to find the path to the minimum edit distance. This for example, is needed for finding the alignment between 2 sequences
- Example:
  - Intention and Execution
- We can implement backtracing by keeping pointers for each cell to record the possible preceding cells that contributed to the minimum distance for this cell (i, j). It is possible that this cell might have been reached just from one previous cell, or two cells or all the three.
- When we reach the termination condition, start from the pointer from the final cell (i = M, j = N) and trace back to find the alignment

Example – Backtracing (Adopted from Dan Jurafsky)

n	9	↓ 8	↙←↓ 9	↙←↓ 10	↙←↓ 11	↙←↓ 12	↓ 11	↓ 10	↓ 9	↙ **8**
o	8	↓ 7	↙←↓ 8	↙←↓ 9	↙←↓ 10	↙←↓ 11	↓ 10	↓ 9	↙ **8**	← 9
i	7	↓ 6	↙←↓ 7	↙←↓ 8	↙←↓ 9	↙←↓ 10	↓ 9	↙ **8**	← 9	← 10
t	6	↓ 5	↙←↓ 6	↙←↓ 7	↙←↓ 8	↙←↓ 9	↙ **8**	← 9	← 10	←↓ 11
n	5	↓ 4	↙←↓ 5	↙←↓ 6	↙←↓ 7	↙←↓ **8**	↙←↓ 9	↙←↓ 10	↙←↓ 11	↙↓ 10
e	4	↙ 3	← 4	↙← **5**	← **6**	← 7	←↓ 8	↙←↓ 9	↙←↓ 10	↓ 9
t	3	↙←↓ 4	↙←↓ **5**	↙←↓ 6	↙←↓ 7	↙←↓ 8	↙ 7	←↓ 8	↙←↓ 9	↓ 8
n	2	↙←↓ **3**	↙←↓ 4	↙←↓ 5	↙←↓ 6	↙←↓ 7	↙←↓ 8	↓ 7	↙←↓ 8	↙ 7
i	**1**	↙←↓ 2	↙←↓ 3	↙←↓ 4	↙←↓ 5	↙←↓ 6	↙←↓ 7	↙ 6	← 7	← 8
#	**0**	1	2	3	4	5	6	7	8	9
	#	**e**	**x**	**e**	**c**	**u**	**t**	**i**	**o**	**n**

Backup slides