Faculty of Computer Science, Dalhousie University *1-Oct-2024*

CSCI 4152/6509 — Natural Language Processing

Lecture 8: Similarity Based Classification

Location: Carleton Tupper Building Theatre C   Instructor: Vlado Keselj
Time: 16:05 – 17:25

Previous Lecture

- IR evaluation measures review
- Recall-precision curve review
- Text classification review
- Evaluation measures for Text Classification review
- Discussion about evaluation methods for classifiers
- Similarity-based Text Classification

10.2 Common N-Grams Method for Text Classification (CNG)

We will now take a closer look at a specific kind of text classification problem, called *authorship attribution,* and a simple character n-grams based method that works well on this task, called the *CNG method* (Common N-Gram analysis method). The method was initially published in 2003 and used in the authorship attribution task, but later showed some good results on other tasks as well. Figure 2 illustrates the general authorship attribution problem.

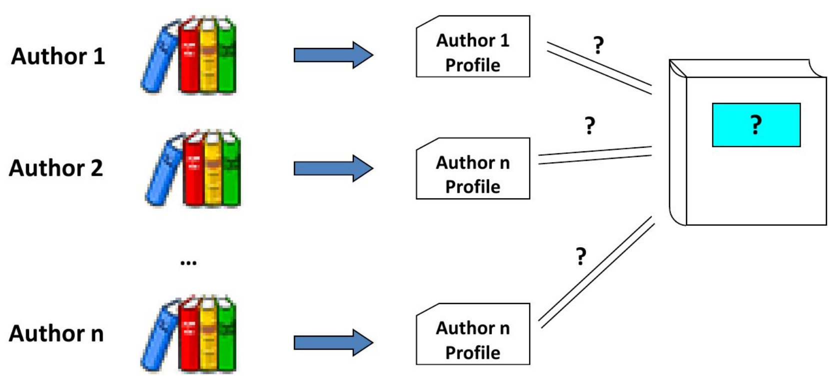


Figure 2: Authorship Attribution Problem

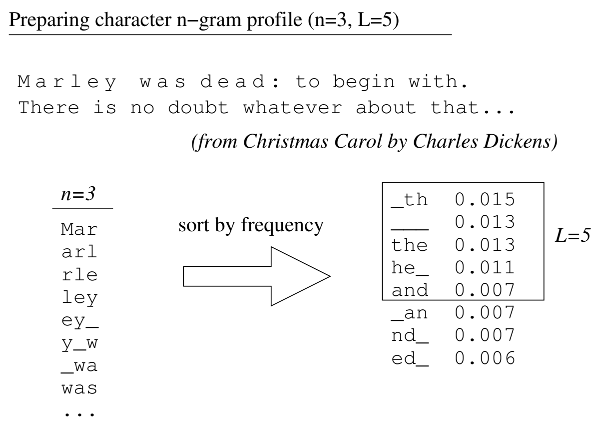
CNG Method Overview

- Method based on character n-grams
- Language independent
- Based on creating n-gram based author profiles
- Similarity based (a type of kNN method—k Nearest Neighbours)
- Similarity measure:

$$\sum_{g \in D_1 \cup D_2} \left(\frac{f_1(g) - f_2(g)}{\frac{f_1(g)+f_2(g)}{2}} \right)^2 = \sum_{g \in D_1 \cup D_2} \left(\frac{2 \cdot (f_1(g) - f_2(g))}{f_1(g) + f_2(g)} \right)^2 \tag{1}$$

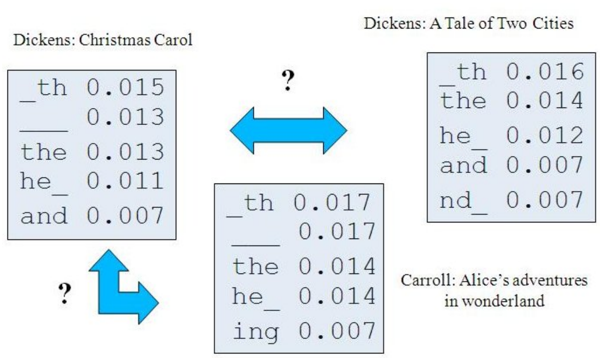
where $f_i(g) = 0$ if $g \notin D_i$.

Example of Creating an Author Profile



The profile is created by collecting all character n-grams of certain size, sorting them according to the normalized frequency; i.e., frequency obtained by taking n-grams count divided by the total number of n-grams; and keeping L most frequent n-grams, where L is a positive integer called the profile length.

How to measure profile similarity?



CNG Similarity Measure

- Euclidean-style distance with relative differences, rather than absolute
- Example: instead of using $0.88 - 0.80 = 0.10$, we say it is about 10% difference, which is the same for 0.088 and 0.080
- To be symmetric, divide by the arithmetic average:

$$d(f_1, f_2) = \Sigma_{n \in \mathrm{dom}(f_1) \cup \mathrm{dom}(f_2)} \left(\frac{f_1(n) - f_2(n)}{\frac{f_1(n)+f_2(n)}{2}} \right)^2$$

- $dom(f_i)$ is the domain of function f_i, i.e., of the profile i

Motivation for Similarity Measure: The idea for this particular similarity measure comes from the standard Euclidean distance:

$$\sum_{g \in D_1 \cup D_2} (f_1(g) - f_2(g))^2 \tag{2}$$

However, the Euclidean distance would be dominated by the most frequent n-grams, since their frequency is orders of magnitude higher than lower frequency n-grams. This is due to a Zipf's-like distribution law for n-grams; i.e., a power-law distribution of n-gram frequencies. To increase the impact of lower-frequency n-grams, we calculate an Euclidean-style distance for relative n-gram frequency differences. This is how we obtain the CNG similarity measure:

$$\sum_{g \in D_1 \cup D_2} \left(\frac{f_1(g) - f_2(g)}{\frac{f_1(g)+f_2(g)}{2}} \right)^2 = \sum_{g \in D_1 \cup D_2} \left(\frac{2 \cdot (f_1(g) - f_2(g))}{f_1(g) + f_2(g)} \right)^2$$

where $f_i(g) = 0$ if $g \notin D_i$. It is important that we take a union of the n-grams in profiles, rather than an intersection, since taking an intersection would lead to a low distance value for profiles with small overlap, which are intuitively dissimilar.

CNG Similarity Example

Let us consider an example of comparing two very simple documents, each one consisting of one line:

```
d1: the dog eat homework
d2: the cat eat homework
```

In other words, the first document `d1` contains only the string '`the dog eat homework`" and the second document `d2` contains the string "`the cat eat homework`". The two strings are not grammatical sentences for simplicity reasons, although they may realistically occur after stemming grammatical sentences. If we collect all character tri-grams from these strings, we will obtain the following trigrams from the first document:

```
the  he_  e_d  _do  dog  og_  g_e  _ea  eat
at_  t_h  _ho  hom  ome  mew  ewo  wor  ork
```

If we sort the n-grams, count them, and normalize their frequency we obtain the following results:

Trigram	count	normalized frequency (f_1)
`_do`	1	0.0555555555555556
`_ea`	1	0.0555555555555556
`_ho`	1	0.0555555555555556
`at_`	1	0.0555555555555556
`dog`	1	0.0555555555555556
`e_d`	1	0.0555555555555556
`eat`	1	0.0555555555555556
`ewo`	1	0.0555555555555556
`g_e`	1	0.0555555555555556
`he_`	1	0.0555555555555556
`hom`	1	0.0555555555555556
`mew`	1	0.0555555555555556
`og_`	1	0.0555555555555556
`ome`	1	0.0555555555555556
`ork`	1	0.0555555555555556
`t_h`	1	0.0555555555555556
`the`	1	0.0555555555555556
`wor`	1	0.0555555555555556
sum	18	1.0

The list of frequencies is very simply since we have no repeated trigrams in this simple string. Similarly, for the second string we obtain frequencies:

Trigram	count	normalized frequency (f_2)
`_ca`	1	0.0555555555555556
`_ea`	1	0.0555555555555556
`_ho`	1	0.0555555555555556
`at_`	2	0.111111111111111
`cat`	1	0.0555555555555556
`e_c`	1	0.0555555555555556
`eat`	1	0.0555555555555556
`ewo`	1	0.0555555555555556
`he_`	1	0.0555555555555556
`hom`	1	0.0555555555555556
`mew`	1	0.0555555555555556
`ome`	1	0.0555555555555556
`ork`	1	0.0555555555555556
`t_e`	1	0.0555555555555556
`t_h`	1	0.0555555555555556
`the`	1	0.0555555555555556
`wor`	1	0.0555555555555556
sum	18	1.0

Since the documents are very short, we are not going to use the profile cut-off length L; i.e., we will use all n-grams. In order to calculate the CNG distance, we now make a union of all n-grams and compare their frequencies. This is how we obtain the following table:

Trigram	f_1	f_2	$(2(f_1 - f_2)/(f_1 + f_2))^2$
`_ca`	0	0.0555555555555556	4.0
`_do`	0.0555555555555556	0	4.0
`_ea`	0.0555555555555556	0.0555555555555556	0.0
`_ho`	0.0555555555555556	0.0555555555555556	0.0
`at_`	0.0555555555555556	0.111111111111111	0.444444444444444
`cat`	0	0.0555555555555556	4.0
`dog`	0.0555555555555556	0	4.0
`e_c`	0	0.0555555555555556	4.0
`e_d`	0.0555555555555556	0	4.0
`eat`	0.0555555555555556	0.0555555555555556	0.0
`ewo`	0.0555555555555556	0.0555555555555556	0.0
`g_e`	0.0555555555555556	0	4.0
`he_`	0.0555555555555556	0.0555555555555556	0.0
`hom`	0.0555555555555556	0.0555555555555556	0.0
`mew`	0.0555555555555556	0.0555555555555556	0.0
`og_`	0.0555555555555556	0	4.0
`ome`	0.0555555555555556	0.0555555555555556	0.0
`ork`	0.0555555555555556	0.0555555555555556	0.0
`t_e`	0	0.0555555555555556	4.0
`t_h`	0.0555555555555556	0.0555555555555556	0.0
`the`	0.0555555555555556	0.0555555555555556	0.0
`wor`	0.0555555555555556	0.0555555555555556	0.0
sum			36.444444444444444

Hence, the CNG distance between documents `d1` and `d2` is 36.444444444444444.

Classification using CNG

- Create profile for each class using training text
  - done by merging all texts in each class into one long document
  - another option: centroid of profiles of individual documents
- Create profile for the test document
- Assign class to the document according to the closest class profile according to the CNG distance

10.3 Edit Distance

The CNG similarity is one way of measuring text similarity, which is quite robust to typos, morphological variations, and similar general string differences. It also somewhat captures word ordering and punctuation, since n-grams can span two words. These characteristics are particularly noticable when comparing this similarity to the standard bag-of-words approach, which may or may not use stemming, and which relies on cosine similarity. Another similarity measure that is very string-oriented, with a similar set of characteristics, is the *edit distance.*

Slide notes:

Edit Distance: Introduction

- Edit distance is a similarity measure convenient for words and short texts, robust for typos and morphological differences
- Tends to be too expensive for longer texts
- Consider typical errors that cause typos:
  - there → thre (missed a letter)
  - there → theare (inserted an extra letter)
  - there → yhere (mistyped a letter)
- Task: find a word in lexicon most likely to produce incorrect word found in text

Slide notes:

Edit Distance: Brute Force Approaches

- one approach: search lexicon and try deleting, inserting, and replacing each of the letters, and compare with mistyped word
- this is already quite expensive, but what with multiple errors?
- Can we find the minimal number of edit operations (deletes, inserts, or substitutions) that would lead from a source string s to the target string t?
- This is *minimal edit distance* — it always exists because we can always delete $|s|$ letters and insert $|t|$ letters, so it is always $\leq |s| + |t|$

Slide notes:

Edit Distance: Properties

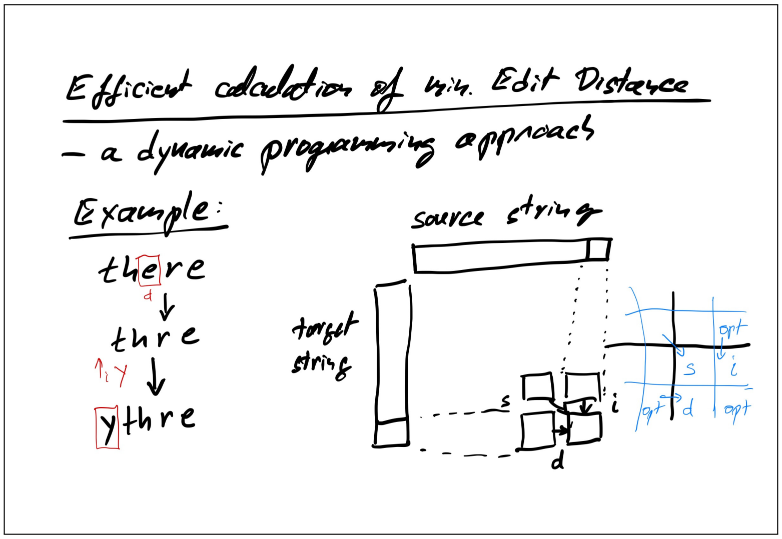
- Reflexive: $d(s,t) = 0$ if and only if $s = t$
- Symmetric: $d(s,t) = d(t,s)$, because edit operations are reversible
- Transitive: $d(s,t) + d(t,v) \geq d(s,v)$
- Can be parametrized with *cost_d*(c), *cost_i*(c), *cost_s*(c,d) for all characters c and d; positive cost functions with exception *cost_s*$(c,c) = 0$
- If cost is 1 for delete and insert, and 2 for substitute operations, it is also known as the Levenshtein distance [JM] (all cost= 1 according to some sources)

Slide notes:

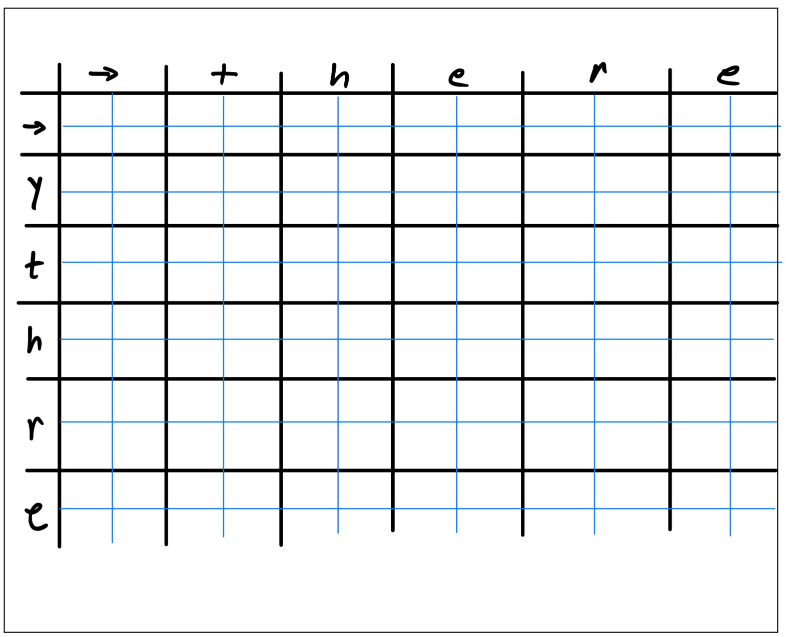
Edit Distance: Dynamic Programming Idea

– calculate optimal distance between $s = xe$ and $t = yf$ using optimal distances between xe and y, x and yf, and x and y

Slide notes:



Slide notes:



Slide notes:

	→	t		h		e		r		e	
→	0	1	1	2	2	3	3	4	4	5	5
y	1	1	2	2	3	3	4	4	5	5	6
	1	2	1	2	2	3	3	4	4	5	5
t	2	1	2	2	3	3	4	4	5	5	6
	2	3	1	2	2	3	3	4	4	5	5
h	3	3	2	1	3	3	4	4	5	5	6
	3	4	2	3	1	2	2	3	3	4	4
r	4	4	3	3	2	2	3	2	4	4	5
	4	5	3	4	2	3	2	3	2	3	3
e	5	5	4	4	3	2	3	3	3	2	4
	5	6	4	5	3	4	2	3	3	4	2

Edit Distance Algorithm

```
Algorithm EditDistance(s,t)
n = len(s); m  = len(t)
d[m+1,n+1] - initialize to 0s
for i=1 to n do d[0,i] = d[0,i-1] + cost_d(s[i-1])
for j=1 to m do d[j,0] = d[j-1,0] + cost_i(t[j-1])
for j=1 to m do
  for i=1 to n do
    d[j,i] = min( d[j-1,i-1] + cost_s(s[i-1],t[j-1]),
                  d[j-1,i]   + cost_i(t[j-1]),
                  d[j,i-1]   + cost_d(s[i-1]) )
return d[m,n]
```