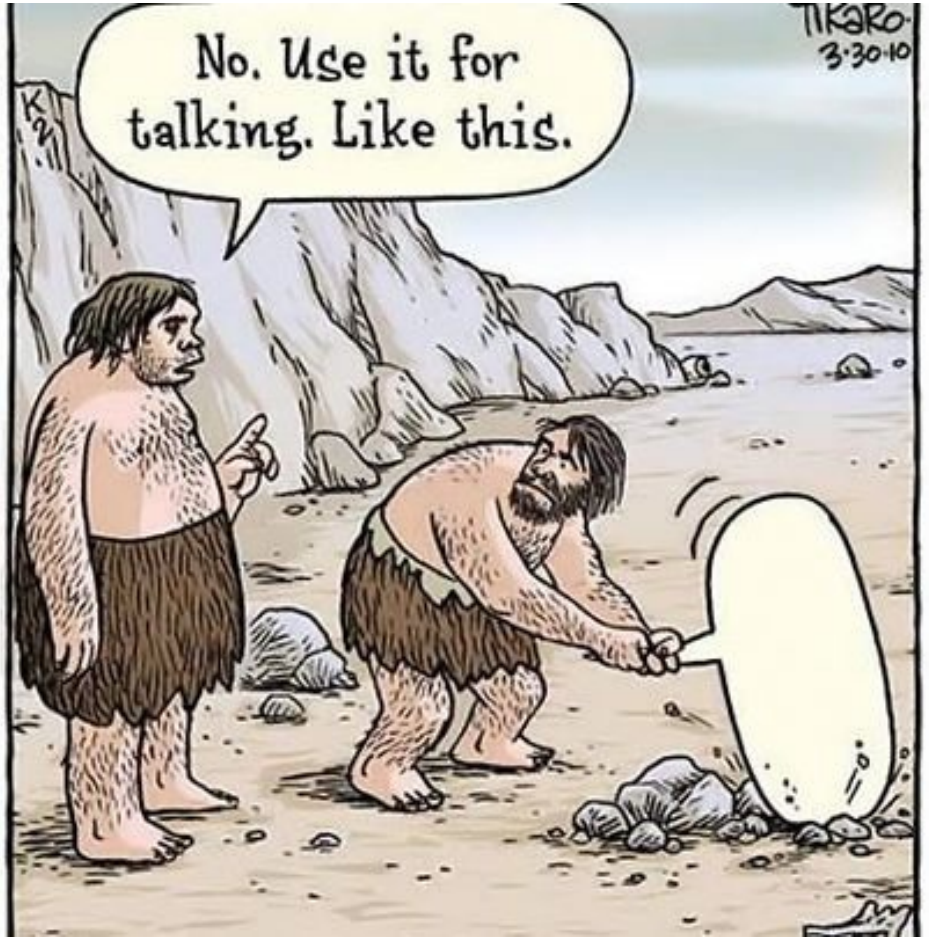
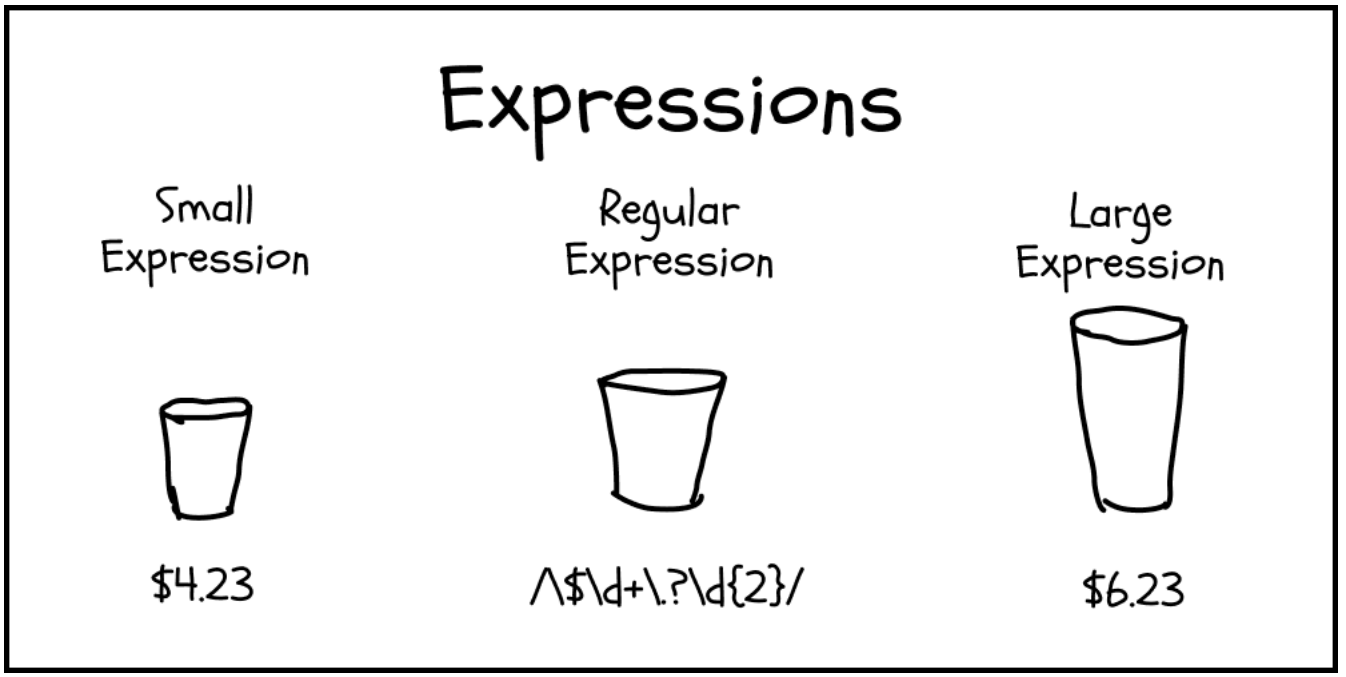
Natural Language Processing

RE & Word Syntax



NLP: Regular Expressions



NLP: Regular Expressions

....maybe more realistic....

Regular expressions

- A formal language for specifying text strings
- How can we search for any of these?
  - woodchuck
  - woodchucks
  - Woodchuck
  - Woodchucks

Regular Expressions: Disjunctions

- Letters inside square brackets []

Pattern	Matches
`[wW]oodchuck`	Woodchuck, woodchuck
`[1234567890]`	Any digit

- Ranges `[A-Z]`

Pattern	Matches	
`[A-Z]`	An upper case letter	`Drenched Blossoms`
`[a-z]`	A lower case letter	`my beans were impatient`
`[0-9]`	A single digit	`Chapter 1: Down the Rabbit Hole`

Regular Expressions: Negation in Disjunction

- Negations [^Ss]
  - Carat means negation only when first in []

Pattern	Matches	
[^A-Z]	Not an upper case letter	Oyfn pripetchik
[^Ss]	Neither 'S' nor 's'	I have no exquisite reason"
[^e^]	Neither e nor ^	Look here
a^b	The pattern a carat b	Look up a^b now

Regular Expressions: More Disjunction

- Woodchucks is another name for groundhog!
- The pipe | for disjunction

Pattern	Matches
`groundhog|woodchuck`	
`yours|mine`	yours mine
`a|b|c`	= [abc]
`[gG]roundhog|[Ww]oodchuck`	

Regular Expressions: ? * + .

Pattern	Matches	
`colou?r`	Optional previous char	`color` `colour`
`oo*h!`	0 or more of previous char	`oh!` `ooh!` `oooh!` `ooooh!`
`o+h!`	1 or more of previous char	`oh!` `ooh!` `oooh!` `ooooh!`
`baa+`		`baa` `baaa` `baaaa` `baaaaa`
`beg.n`		`begin` `begun` `begun` `beg3n`

Stephen C Kleene

Kleene *, Kleene +

Regular Expressions: Anchors ^ $

Pattern	Matches
^[A-Z]	Palo Alto
^[^A-Za-z]	1 "Hello"
\.$	The end.
.$	The end? The end!

Example

- Find me all instances of the word “the” in a text.

```
the
```

Misses capitalized examples

```
[tT]he
```

Incorrectly returns `other` or `theology`

```
[^a-zA-Z][tT]he[^a-zA-Z]
```

Errors

- The process we just went through was based on fixing two kinds of errors
  - Matching strings that we should not have matched (there, then, other)
    - False positives (Type I)
  - Not matching things that we should have matched (The)
    - False negatives (Type II)

Errors cont.

- In NLP we are always dealing with these kinds of errors.
- Reducing the error rate for an application often involves two antagonistic efforts:
  - Increasing accuracy or precision (minimizing false positives)
  - Increasing coverage or recall (minimizing false negatives).

Regular Expressions

Operator	Meaning	Example	Example meaning
+	one or more	a+	look for 1 or more "a" characters
*	zero or more	a*	look for 0 or more "a" characters
?	optional	a?	look for 0 or 1 "a" characters
[]	choose 1	[abc]	look for "a" or "b" or "c"
[-]	range	[a-z]	look for any character between "a" and "z"
[^]	not	[^a]	look for character that is not "a"
()	grouping	(a-z)+	look for one of more occurences of chars between "a" and "z"
(|)	or operator	(ey|ax)	look for strings "ey" or "ax"
ab	follow	ab	look for character "a" followed by character "b"
^	start	^a	look for character "a" at start of string/line
$	end	a$	look for character "a" at end of string/line
\s	whitespace	\sa	look for whitespace character followed by "a"
.	any character	a.b	look for "a" followed by any char followed by "b"

Regular Expressions

- Eliza: https://www.masswerk.at/elizabot/
- https://regex101.com/

Regular Expressions


```
#   Regular expressions for Eliza (to be applied in earnest)
# To execute the file: execfile(r'./RE.py')

import re

str = "I feel sad, my heart is broken"
print("The initial string is:", str)
str = re.sub("I", "you", str)
print("After first substitution:", str)
str = re.sub("my", "your", str)
print("After second substitution:", str)
str = re.sub(r"you (feel|are) (sad|sorry|desperate)(.*)",
r"Why do you say that you \1 \2\3?", str)
print("After third substitution:", str)
```

Regular Expressions in Python

see code available on AulaWeb

File name: RE.py

NLP: Syntax (Word Level)

Twenty years later, Kim turns the tables on her loathsome former English teacher.

Word Segmentation

Word segmentation is the problem of dividing a string of written language into its component words.

In English and many other languages using some form of the Latin alphabet, the space is a good approximation of a word divider (word delimiter), although this concept has limits because of the variability with which languages regard collocations and compounds. Many English compound nouns are variably written (for example, multi-agent = multiagent = multi agent) with a corresponding variation in whether speakers think of them as noun phrases or single nouns.

However, the equivalent to the word space character is not found in all written scripts, and without it word segmentation is a difficult problem. Languages which do not have a trivial word segmentation process include Chinese, Japanese, where sentences but not words are delimited, Thai and Lao, where phrases and sentences but not words are delimited, and Vietnamese, where syllables but not words are delimited.

Word Segmentation in Chinese: the MaxMatch algorithm

The algorithm requires a dictionary (wordlist) of the language. The maximum matching algorithm starts by pointing at the beginning of a string. It chooses the longest word in the dictionary that matches the input at the current position. The pointer is then advanced to the end of that word in the string. If no word matches, the pointer is instead advanced one character (creating a one-character word). The algorithm is then iteratively applied again starting from the new pointer position.

Input: wecanonlyseeashortdistanceahead

Output: we canon l y see ash o r t distance ahead

NLTK

NLTK is a platform for building Python programs to work with human language data. It provides easy-to-use interfaces to over 50 corpora and lexical resources such as WordNet, along with a suite of text processing libraries for classification, tokenization, stemming, tagging, parsing, and semantic reasoning.

https://www.nltk.org/

During this course, we will sometimes use NLTK to exemplify some problems and some of their possible solutions

Sometimes, we will also use less known and widely adopted, but still elegant and nice, tools.

Word Segmentation in Python

see code available on AulaWeb

File name: wordToken.py

```
# (download NLTK only once by de-commenting the line below)
# nltk.download()
```

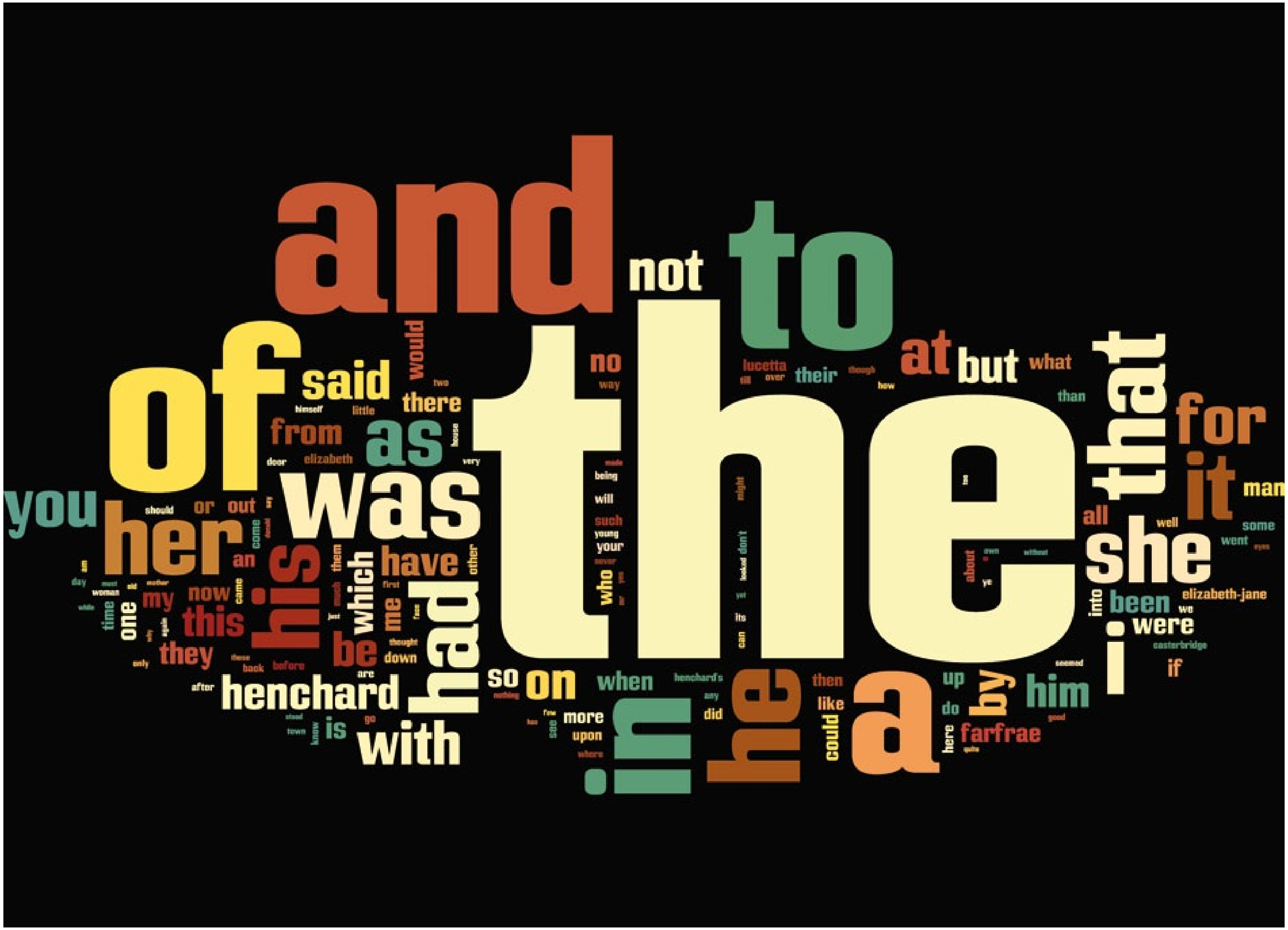
Stop words

Stop words are words which are filtered out before or after processing of natural language data (text). Though "stop words" usually refers to the most common words in a language, there is no single universal list of stop words used by all natural language processing tools, and indeed not all tools even use such a list. Some tools specifically avoid removing these stop words to support phrase (namely, “exact” and not “keyword-based”) search.

The general strategy for determining a stop list is to sort the terms by collection frequency (the total number of times each term appears in the document collection), and then to take the most frequent terms, often hand-filtered for their semantic content relative to the domain of the documents being indexed, as a stop list , the members of which are then discarded during indexing.

Example: https://www.ranks.nl/stopwords

Stop words



the
and
to
of
was
that
not
a
he
in
her
his
had
you
she
it
for
i
at
but
said
as
with
henchard
have
this
be
on
him
by
from
which
they
there
would
my
now
is
so
when
been
were
no
what
their
all
man
farfrae
elizabeth-jane
lucetta
could
like
up
more
upon
if
me
one
an
out
or
time
then
did
do
here
into
some
went
well

Stop words in Python

see code available on AulaWeb

File name: stopWords.py

Stop words

Is a word that appears often in a document, always a stop word?

How can we discriminate between words which are frequent in a document **d** because they are related with the document's topic, and words which are frequent in **d** because they are frequent in the language?

Stop words

Our simple corpus C1 consists of Doc1

- **Doc1 (length 54)**
- The cat (Felis catus) is a small carnivorous mammal. It is the only domesticated species in the family Felidae and often referred to as the domestic cat to distinguish it from wild members of the family. The cat is either a house cat or a farm cat, which are pets, or a feral cat.
-
-
-

Stop words

- Our simple corpus C1 consists of Doc1
- **Doc1 (length 54)**
- The cat (Felis catus) is a small carnivorous mammal. It is the only domesticated species in the family Felidae and often referred to as the domestic cat to distinguish it from wild members of the family. The cat is either a house cat or a farm cat, which are pets, or a feral cat.
- number of “the” in Doc1 = 6
- number of “cat” in Doc1 = 6
- number of “car(s)” in Doc1 = 0
-

Stop words

- Our simple corpus C2 consists of Doc1 and Doc2
- **Doc2 (length 42)**
- A car (or automobile) is a wheeled motor vehicle used for transportation. Cars became widely available in the early 20th century. One of the first cars accessible to the masses was the 1908 Model T. Cars were rapidly adopted in the US.
-
-
-

Stop words

- Our simple corpus C2 consists of Doc1 and Doc2
- **Doc2 (length 42)**
- A car (or automobile) is a wheeled motor vehicle used for transportation. Cars became widely available in the early 20th century. One of the first cars accessible to the masses was the 1908 Model T. Cars were rapidly adopted in the US.
- number of “the” in Doc2 = 4
- number of “cat” in Doc2 = 0
- number of “car(s)” in Doc2 = 4

Stop words

- Our simple corpus C2 consists of Doc1 and Doc2
- number of “the” in C2 = 10
- number of “cat” in C2 = 6
- number of “car(s)” in C2 = 4

Stop words & TF-IDF

Term frequency tf(t,d), is the raw count of a term t in a document d, i.e., the number of times that term t occurs in document d.

Often normalized w.r.t. the document's length

- tf(t,d) = number of t in d / length(d)

Stop words & TF-IDF

Inverse document frequency, idf(t, C), is the logarithmically scaled inverse fraction of the documents in the corpus C that contain the term t (obtained by dividing the total number of documents in C by the number of documents containing t, and then taking the logarithm of that quotient):

- **idf(t, C)** = log (|C|/(|documents in C which contain t|))

Stop words & TF-IDF

Inverse document frequency, idf(t, C), is the logarithmically scaled inverse fraction of the documents in the corpus C that contain the term t (obtained by dividing the total number of documents in C by the number of documents containing t, and then taking the logarithm of that quotient):

- **idf(t, C)** = log (|C|/(|documents in C which contain t|))
- |C| dimension of the corpus

Stop words & TF-IDF

Inverse document frequency, idf(t, C), is the logarithmically scaled inverse fraction of the documents in the corpus C that contain the term t (obtained by dividing the total number of documents in C by the number of documents containing t, and then taking the logarithm of that quotient):

- **idf(t, C)** = log (|C|/(|documents in C which contain t|))
- |documents in C which contain t| (we might add 1 to avoid dividing by 0, but this makes sense if C is “large enough”)

Stop words & TF-IDF

tf-idf(t, d, C) = tf(t, d) * idf(t, C)

Stop words & TF-IDF

tf-idf(“the”, Doc1, C2) =

Stop words & TF-IDF

tf-idf(“the”, Doc1, C2) =

tf(“the”, Doc1) * idf(“the”, C2) =

6/54 * log(2/2) =

0

Stop words & TF-IDF

tf-idf(“the”, Doc2, C2) =

Stop words & TF-IDF

tf-idf(“the”, Doc2, C2) =

tf(“the”, Doc2) * idf(“the”, C2) =

4/42 * log(2/2) =

0

Stop words & TF-IDF

tf-idf(“cat”, Doc1, C2) =

Stop words & TF-IDF

tf-idf(“cat”, Doc1, C2) =

tf(“cat”, Doc1) * idf(“cat”, C2) =

6/54 * log(2/1) =

0.1 * 0.3 = 0.03

Stop words & TF-IDF

tf-idf(“cat”, Doc2, C2) =

Stop words & TF-IDF

tf-idf(“cat”, Doc2, C2) =

tf(“cat”, Doc2) * idf(“cat”, C2) =

0 * ... =

0

Stop words & TF-IDF

tf-idf(“car”, Doc1, C2) =

Stop words & TF-IDF

tf-idf(“car”, Doc1, C2) =

tf(“car”, Doc1) * idf(“car”, C2) =

0 * ... =

0

Stop words & TF-IDF

tf-idf(“car”, Doc2, C2) =

Stop words & TF-IDF

tf-idf(“car”, Doc2, C2) =

tf(“car”, Doc2) * idf(“car”, C2) =

4/42 * log(2/1) =

0.09 * 0.3 = 0.028

Stop words

https://www.online-utility.org/text/analyzer.jsp

Stemming and Lemmatization

Documents are going to use different forms of a word, such as organize, organizes, and organizing. Additionally, there are families of derivationally related words with similar meanings, such as democracy, democratic, and democratization. In many situations, it is useful for a search for one of these words to return documents that contain another word in the set.The goal of both stemming and lemmatization is to reduce inflectional forms and sometimes derivationally related forms of a word to a common base form. For instance:

am, are, is -> be

car, cars, car's, cars' -> car

The result of this mapping of text will be something like:

the boy's cars are different colors -> the boy car be differ color

Stemming and Lemmatization

Stemming and lemmatization differ in their flavor. Stemming usually refers to a crude heuristic process that chops off the ends of words in the hope of achieving this goal correctly most of the time, and often includes the removal of derivational affixes.

Lemmatization usually refers to doing things properly with the use of a vocabulary and morphological analysis of words, normally aiming to remove inflectional endings only and to return the base or dictionary form of a word, which is known as the lemma.

The two may also differ in that stemming most commonly collapses derivationally related words, whereas lemmatization commonly only collapses the different inflectional forms of a lemma.

The Porter Stemmer (Porter, 1980)

- A simple rule-based algorithm for stemming
- An example of a HEURISTIC method
- Based on rules like:
  - ATIONAL -> ATE (e.g., *relational* -> *relate*)
- The algorithm consists of seven sets of rules, applied in order

The Porter Stemmer: definitions

- Definitions:
  - CONSONANT: a letter other than A, E, I, O, U, and Y preceded by consonant
  - VOWEL: any other letter
- With this definition, all words are of the form:

  $(C)(VC)^m(V)$

  C=string of one or more consonants (con+)

  V=string of one or more vowels
- E.g.,
  - Troubles
  - C V CVC

The Porter Stemmer: rule format

- The rules are of the form:

  (condition) S1 -> S2

  Where S1 and S2 are suffixes
- Conditions:

m	The measure of the stem
*S	The stem ends with S
v	The stem contains a vowel
*d	The stem ends with a double consonant
*o	The stem ends in CVC (second C not W, X, or Y)

The Porter Stemmer: Step 1

- SSES -> SS
  - *caresses* -> *caress*
- IES -> I
  - *ponies* -> *poni*
  - *ties* -> *ti*
- SS -> SS
  - *caress* -> *caress*
- S -> ϵ
  - *cats* -> *cat*

The Porter Stemmer: Step 2a

- (m>1) EED -> EE
  - Condition verified*: agreed -> agree*
  - Condition not verified*: feed -> feed*
- (*V*) ED -> ϵ
  - Condition verified*: plastered -> plaster*
  - Condition not verified*: bled -> bled*
- (*V*) ING -> ϵ
  - Condition verified*: motoring -> motor*
  - Condition not verified*: sing -> sing*

The Porter Stemmer: Step 2b

- (These rules are ran if second or third rule in 2a apply)
- AT-> ATE
  - *conflat(ed) -> conflate*
- BL -> BLE
  - *Troubl(ing) -> trouble*
- (*d & ! (*L or *S or *Z)) -> single letter
  - Condition verified*: hopp(ing) -> hop, tann(ed) -> tan*
  - Condition not verified*: fall(ing) -> fall*
- (m=1 & *o) -> E
  - Condition verified*: fil(ing) -> file*
  - Condition not verified*: fail -> fail*

The Porter Stemmer: Steps 3 and 4

- Step 3: Y Elimination (*V*) Y -> I
  - Condition verified*: happy -> happi*
  - Condition not verified*: sky -> sky*
- Step 4: Derivational Morphology, I
  - (m>0) ATIONAL -> ATE
    - *Relational -> relate*
  - (m>0) IZATION -> IZE
    - *generalization-> generalize*
  - (m>0) BILITI -> BLE
    - *sensibiliti -> sensible*

The Porter Stemmer: Steps 5 and 6

- Step 5: Derivational Morphology, II
  - (m>0) ICATE -> IC
    - *triplicate -> triplic*
  - (m>0) FUL -> ϵ
    - *hopeful -> hope*
  - (m>0) NESS -> ϵ
    - *goodness -> good*
- Step 6: Derivational Morphology, III
  - (m>0) ANCE -> ϵ
    - *allowance-> allow*
  - (m>0) ENT -> ϵ
    - *dependent-> depend*
  - (m>0) IVE -> ϵ
    - *effective -> effect*

The Porter Stemmer: Step 7

- Step 7a
  - (m>1) E -> ϵ
    - *probate -> probat*
  - (m=1 & !*o) NESS -> ϵ
    - *goodness -> good*
- Step 7b
  - (m>1 & *d & *L) -> single letter
    - Condition verified*: controll -> control*
    - Condition not verified*: roll -> roll*

Examples

- *computers*
  - Step 1, Rule 4: -> *computer*
  - Step 6, Rule 4: -> *compute*
- *singing*
  - Step 2a, Rule 3: -> *sing*
  - Step 6, Rule 4: -> *compute*
- *controlling*
  - Step 2a, Rule 3: -> *controll*
  - Step 7b : -> *control*
- *generalizations*
  - Step 1, Rule 4: -> *generalization*
  - Step 4, Rule 11: -> *generalize*
  - Step 6, last rule: -> *general*

Problems

- *elephants -> eleph*
  - Step 1, Rule 4: -> *elephant*
  - Step 6, Rule 7: -> *eleph*
- *doing - > doe*
  - Step 2a, Rule 3: -> *do*

References

- The Porter Stemmer home page (with the original paper and code): http://www.tartarus.org/~martin/PorterStemmer/
- Jurafsky and Martin, chapter 3.4
- The original paper: Porter, M.F., 1980, An algorithm for suffix stripping, *Program*, **14**(3) :130-137.

Stemming and Lemmatization

http://9ol.es/porter_js_demo.html

...but...

http://www.dtsearch.co.uk/support/webhelp/la100/porter_stemming_errors.htm

Stemmers for other languages:

http://proiot.ru/jssnowball/

Stemming and Lemmatization

https://text-processing.com/demo/stem/

Stemming and Lemmatization

Sono stato in uno stato il cui statuto prevede che la biscia non possa entrare in una bisca. Una messe di polemiche è piovuta sullo statuto, per le limitazioni messe alla biscia, che è andata a messa insieme al console per consolarsi.

Stemming and Lemmatization

Once Miss Marple had a cold and she went to the doctor: "I go!", she said, and she read the doctor's doctoral thesis. "These are good theses!", and -- although she missed her train -- Miss Marple understood that University has a universal mission.

Stemming in Python

see code available on AulaWeb

File name: textNormalization.py

Minimum Edit Distance

How similar are two strings?

- Spell correction
  - The user typed “graffe”

  Which is closest?
    - graf
    - graft
    - grail
    - giraffe

- Computational Biology
  - Align two sequences of nucleotides

    AGGCTATCACCTGACCTCCAGGCCGATGCCC
    TAGCTATCACGACCGCGGTCGATTTGCCCGAC
  - Resulting alignment:

    –AGG**CTATCAC**C**TGACCTC**C**AGGC**G**A--TGCCC---**
    TAG-CTATCAC--GACCG**C--GG**T**CGA**TT**TGCCC**GAC

- Also for Machine Translation, Information Extraction, Speech Recognition

[...w.r.t. computational biology...]

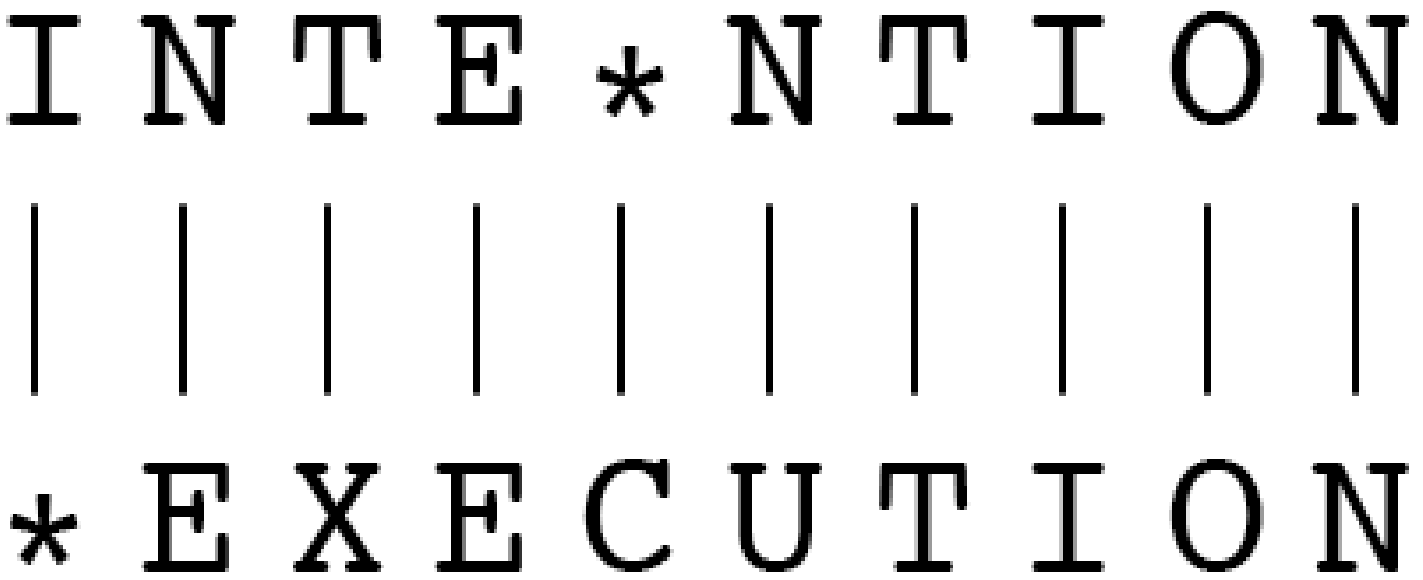
- Why should we care about aligning sequences of nucleotides (= building blocks of the nucleic acids RNA and DNA; in DNA, the four bases are Adenine, Guanine, Thymine and Cytosine. RNA uses Uracil in place of Thymine)?
- Molecular phylogenetics: hereditary molecular differences, predominately in DNA sequences, to gain information on an organism's evolutionary relationships
- Genome evolution and comparison: comparative analysis of genomes of individuals with genetic disease against healthy individuals may reveal clues of eliminating that disease.
- Understanding the function of unknown proteins.
- ....

Edit Distance

- The minimum edit distance between two strings
- Is the minimum number of editing operations
  - Insertion
  - Deletion
  - Substitution
- Needed to transform one into the other

Minimum Edit Distance

- Two strings and their **alignment**:



Minimum Edit Distance

```
I N T E * N T I O N
| | | | | | | | | |
* E X E C U T I O N
d s s   i s
```

- If each operation has cost of 1
  - Distance between these is 5
- If substitutions cost 2 (Levenshtein)
  - Distance between them is 8

Alignment in Computational Biology

- Given a sequence of bases

```
AGGCTATCACCTGACCTCCAGGCCGATGCCC
TAGCTATCACGACCGCGGTCGATTTGCCCGAC
```

- An alignment:

```
-AGGCTATCACCTGACCTCCAGGCCGA--TGCCC---
TAG-CTATCAC--GACCGC--GGTCGATTTGCCCGAC
```

- Given two sequences, align each letter to a letter or gap

Other uses of Edit Distance in NLP

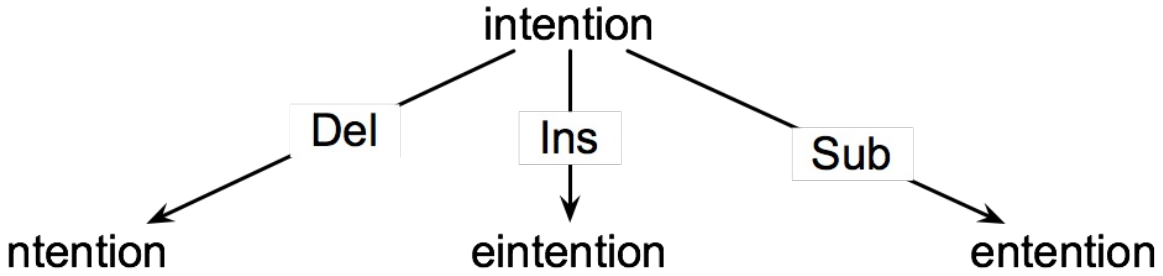
- Evaluating Machine Translation and speech recognition

```
R Spokesman confirms    senior government adviser was shot
H Spokesman said    the senior            adviser was shot dead
              S      I              D                        I
```

- Named Entity Extraction and Entity Coreference
  - IBM Inc. announced today
  - IBM profits
  - Stanford President John Hennessy announced yesterday
  - for Stanford University President John Hennessy

How to find the Min Edit Distance?

- Searching for a path (sequence of edits) from the start string to the final string:
  - **Initial state**: the word we're transforming
  - **Operators**: insert, delete, substitute
  - **Goal state**: the word we're trying to get to
  - **Path cost**: what we want to minimize: the number of edits



Minimum Edit as Search

- But the space of all edit sequences is huge!
  - We can't afford to navigate naïvely
  - Lots of distinct paths wind up at the same state.
    - We don't have to keep track of all of them
    - Just the shortest path to each of those revisted states.

Defining Min Edit Distance

- For two strings
  - X of length *n*
  - Y of length *m*
- We define D(*i*,*j*)
  - the edit distance between X[1..*i*] and Y[1..*j*]
    - i.e., the first *i* characters of X and the first *j* characters of Y
  - The edit distance between X and Y is thus D(*n*,*m*)

Minimum Edit Distance

Computing Minimum Edit Distance

Dynamic Programming for Minimum Edit Distance

- **Dynamic programming**: A tabular computation of D(n,m)
- Solving problems by combining solutions to subproblems.
- Bottom-up
  - 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 ($0 < j < m$)

```
function MIN-EDIT-DISTANCE(source, target) returns min-distance

  n ← LENGTH(source)
  m ← LENGTH(target)
  Create a distance matrix distance[n+1,m+1]

  # Initialization: the zeroth row and column is the distance from the empty string
      D[0,0] = 0
      for each row i from 1 to n do
          D[i,0] ← D[i-1,0] + del-cost(source[i])
      for each column j from 1 to m do
          D[0,j] ← D[0,j-1] + ins-cost(target[j])

  # Recurrence relation:
  for each row i from 1 to n do
      for each column j from 1 to m do
          D[i,j] ← MIN( D[i−1,j] + del-cost(source[i]),
                        D[i−1,j−1] + sub-cost(source[i], target[j]),
                        D[i,j−1] + ins-cost(target[j]))
  # Termination
  return D[n,m]
```

Figure 2.16 The minimum edit distance algorithm, an example of the class of dynamic programming algorithms. The various costs can either be fixed (e.g., $\forall x$, ins-cost$(x) = 1$) or can be specific to the letter (to model the fact that some letters are more likely to be inserted than others). We assume that there is no cost for substituting a letter for itself (i.e., sub-cost$(x,x) = 0$).

Defining Min Edit Distance (Levenshtein)

- Initialization

  $D(i,0) = i$

  $D(0,j) = j$

- Recurrence Relation:

  For each $i = 1 \ldots M$

  For each $j = 1 \ldots N$

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

- Termination:

  $D(N,M)$ is distance

The Edit Distance Table

Src\Tar	#	e	x	e	c	u	t	i	o	n
#	0	1	2	3	4	5	6	7	8	9
i	1	2	3	4	5	6	7	6	7	8
n	2	3	4	5	6	7	8	7	8	7
t	3	4	5	6	7	8	7	8	9	8
e	4	3	4	5	6	7	8	9	10	9
n	5	4	5	6	7	8	9	10	11	10
t	6	5	6	7	8	9	8	9	10	11
i	7	6	7	8	9	10	9	8	9	10
o	8	7	8	9	10	11	10	9	8	9
n	9	8	9	10	11	12	11	10	9	8

Figure 2.17 Computation of minimum edit distance between *intention* and *execution* with the algorithm of Fig. 2.16, using Levenshtein distance with cost of 1 for insertions or deletions, 2 for substitutions.

The Edit Distance Table with Backtrace

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

Figure 2.18 When entering a value in each cell, we mark which of the three neighboring cells we came from with up to three arrows. After the table is full we compute an **alignment** (minimum edit path) by using a **backtrace**, starting at the **8** in the lower-right corner and following the arrows back. The sequence of bold cells represents one possible minimum cost alignment between the two strings. Diagram design after Gusfield (1997).