EDAN20
Language Technology
http://cs.lth.se/edan20/
Chapter 2: Corpus Processing Tools

Pierre Nugues

Lund University
Pierre.Nugues@cs.lth.se
http://cs.lth.se/pierre_nugues/

August 31 and September 3, 2015

Corpora

A corpus is a collection of texts (written or spoken) or speech
Corpora are balanced from different sources: news, novels, etc.

	English	**French**	**German**
Most frequent words in a collection of contemporary running texts	*the*	*de*	*der*
	of	*le* (article)	*die*
	to	*la* (article)	*und*
	in	*et*	*in*
	and	*les*	*des*
Most frequent words in Genesis	*and*	*et*	*und*
	the	*de*	*die*
	of	*la*	*der*
	his	*à*	*da*
	he	*il*	*er*

Characteristics of Current Corpora

Big: The Bank of English (Collins and U Birmingham) has more than 500 million words

Available in many languages

Easy to collect: The web is the largest corpus ever built and within the reach of a mouse click

Parallel: same text in two languages: English/French (Canadian Hansards), European parliament (23 languages)

Annotated with part-of-speech or manually parsed (treebanks):

- Characteristics/N of/PREP Current/ADJ Corpora/N
- (NP (NP Characteristics) (PP of (NP Current Corpora)))

Lexicography

Writing dictionaries

Dictionaries for language learners should be build on real usage

- *They're just trying to score **brownie points** with politicians*
- *The boss is pleased – that's another **brownie point***

Bank of English: *brownie point* (6 occs) *brownie points* (76 occs)

Extensive use of corpora to:

- Find **concordances** and cite real examples
- Extract **collocations** and describe frequent pairs of words

Concordances

A word and its context:

Language	Concordances
English	`s beginning of miracles did Je`
	`n they saw the miracles which`
	`n can do these miracles that t`
	`ain the second miracle that Je`
	`e they saw his miracles which`
French	`le premier des miracles que fi`
	`i dirent: Quel miracle nous mo`
	`om, voyant les miracles qu’il`
	`peut faire ces miracles que tu`
	`s ne voyez des miracles et des`

Collocations

Word preferences: Words that occur together

	English	French	German
You say	*Strong tea*	*Thé fort*	*Schmales Gesicht*
	Powerful computer	*Ordinateur puissant*	*Enge Kleidung*
You don't say	*Strong computer*	*Thé puissant*	*Schmale Kleidung*
	Powerful tea	*Ordinateur fort*	*Enges Gesicht*

Word Preferences

Strong w			Powerful w		
strong w	*powerful w*	*w*	*strong w*	*powerful w*	*w*
161	0	showing	1	32	than
175	2	support	1	32	figure
106	0	defense	3	31	minority
...					

Corpora as Knowledge Sources

Short term:

- Describe usage more accurately
- Assess tools: part-of-speech taggers, parsers.
- Learn statistical/machine learning models for speech recognition, taggers, parsers
- Derive automatically symbolic rules from annotated corpora

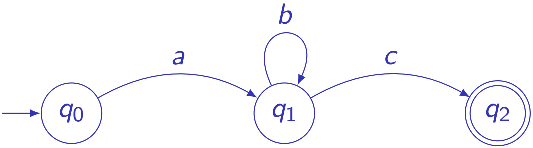
Longer term:

- Semantic processing
- Texts are the main repository of human knowledge

Finite-State Automata

A flexible to tool to search and process text

A FSA accepts and generates strings, here *ac*, *abc*, *abbc*, *abbbc*, *abbbbbbbbbbbbc*, etc.



FSA

Mathematically defined by

- Q a finite number of states;
- Σ a finite set of symbols or characters: the input alphabet;
- q_0 a start state,
- F a set of final states $F \subseteq Q$
- δ a transition function $Q \times \Sigma \rightarrow Q$ where $\delta(q, i)$ returns the state where the automaton moves when it is in state q and consumes the input symbol i.

FSA in Prolog

```
% The start state      % The final states
start(q0).              final(q2).

transition(q0, a, q1).
transition(q1, b, q1).
transition(q1, c, q2).

accept(Symbols) :-
  start(StartState),
  accept(Symbols, StartState).

accept([], State) :-
  final(State).
accept([Symbol | Symbols], State) :-
  transition(State, Symbol, NextState),
  accept(Symbols, NextState).
```

Regular Expressions

Regexes are equivalent to FSA and generally easier to use
Constant regular expressions:

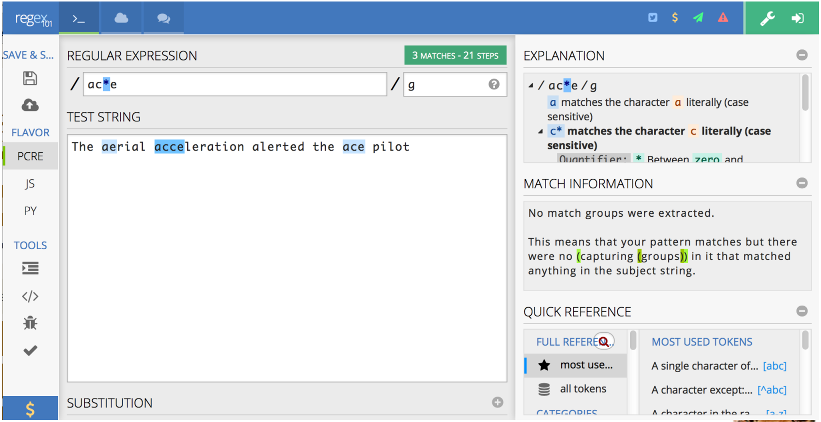
Pattern	String
`regular`	*A section on regular expressions*
`the`	*The book of the life*

The automaton above is described by the regex `ab*c`

`grep 'ab*c' myFile1 myFile2`

regex101.com

regex101.com: A site to experiment and test regular expressions.



Metacharacters

Chars	Descriptions	Examples
*	Matches any number of occurrences of the previous character – zero or more	ac*e matches strings ae, ace, acce, accce, etc. as in "The aerial acceleration alerted the ace pilot"
?	Matches at most one occurrence of the previous character – zero or one	ac?e matches ae and ace as in "The aerial acceleration alerted the ace pilot"
+	Matches one or more occurrences of the previous character	ac+e matches ace, acce, accce, etc. as in as in "The aerial acceleration alerted the ace pilot"

Metacharacters

Chars	Descriptions	Examples
`{n}`	Matches exactly *n* occurrences of the previous character	`ac{2}e` matches `acce` as in "The aerial acceleration alerted the ace pilot"
`{n,}`	Matches *n* or more occurrences of the previous character	`ac{2,}e` matches `acce`, `accce`, etc.
`{n,m}`	Matches from *n* to *m* occurrences of the previous character	`ac{2,4}e` matches `acce`, `accce`, and `acccce`.

Literal values of metacharacters must be quoted using \

The Dot Metacharacter

The dot `.` is a metacharacter that matches one occurrence of any character except a new line

`a.e` matches the strings *ale* and *ace* in:

> *The aerial acceleration alerted the ace pilot*

as well as *age*, *ape*, *are*, *ate*, *awe*, *axe*, or *aae*, *aAe*, *abe*, *aBe*, *a1e*, etc.

`.*` matches any string of characters until we encounter a new line.

The Longest Match

The previous slide does not tell about the match strategy.
Consider the string *aabbc* and the regular expression `a+b*`
By default the match engine is greedy: It matches as early and as many characters as possible and the result is `aabb`
Sometimes a problem. Consider the regular expression `<b>.*</b>` and the phrase

> *They match as early and as many characters as they can.*

It is possible to use a lazy strategy with the `*?` metacharacter instead: `<b>.*?</b>` and have the result:

> *They match as early and as many characters as they can.*

Character Classes

`[...]` matches any character contained in the list.

`[^...]` matches any character not contained in the list.

`[abc]` means one occurrence of either `a`, `b`, or `c`

`[^abc]` means one occurrence of any character that is not an `a`, `b`, or `c`,

`[ABCDEFGHIJKLMNOPQRSTUVWXYZ]` one upper-case unaccented letter

`[0123456789]` means one digit.

`[0123456789]+\.[0123456789]+` matches decimal numbers.

`[Cc]omputer [Ss]cience` matches `Computer Science`, `computer Science`, `Computer science`, `computer science`.

Predefined Character Classes

Expr.	Description	Example
\d	Any digit. Equivalent to [0-9]	A\dC matches A0C, A1C, A2C, A3C etc.
\D	Any nondigit. Equivalent to [^0-9]	
\w	Any word character: letter, digit, or underscore. Equivalent to [a-zA-Z0-9_]	1\w2 matches 1a2, 1A2, 1b2, 1B2, etc
\W	Any nonword character. Equivalent to [^\w]	
\s	Any white space character: space, tabulation, new line, form feed, etc.	
\S	Any nonwhite space character. Equivalent to [^\s]	

Nonprintable Symbols or Positions

Char.	Description	Example
`^`	Matches the start of a line	`^ab*c` matches ac, abc, abbc, etc. when they are located at the beginning of a new line
`$`	Matches the end of a line	`ab?c$` matches ac and abc when they are located at the end of a line
`\b`	Matches word boundaries	`\babc` matches `abcd` but not `dabc` `bcd\b` matches `abcd` but not `abcde`
`\n`	Matches a new line	`a\nb` matches a b
`\t`	Matches a tabulation	

Union and Boolean Operators

Union denoted `|`: `a|b` means either `a` or `b`.

Expression `a|bc` matches the strings `a` and `bc` and `(a|b)c` matches `ac` and `bc`,

Order of precedence:

1. Closure and other repetition operator (highest)
2. Concatenation, line and word boundaries
3. Union (lowest)

`abc*` is the set `ab`, `abc`, `abcc`, `abccc`, etc.

`(abc)*` corresponds to `abc`, `abcabc`, `abcabcabc`, etc.

Perl

Match

```
while ($line = <>) {
  if ($line =~ m/ab+c/) {
    print $line;
  }
}
```

Substitute

```
while ($line = <>) {
  if ($line =~ m/ab+c/) {
    print "Old: ", $line;
    $line =~ s/ab+c/ABC/g;
    print "New: ", $line;
  }
}
```

Perl

Translate

```
tr/ABC/abc/
$line =~ tr/A-Z/a-z/;
$line =~ tr/AEIOUaeiou//d;
$line =~ tr/AEIOUaeiou/$/cs;
```

Concatenate

```
while ($line = <>) {
  $text .= $line;
}
print $text;
```

References

```
while ($line = <>) {
  while ($line =~ m/\$ *([0-9]+)\.?([0-9]*)/g) {
    print "Dollars: ", $1, " Cents: ", $2, "\n";
  }
}
```

Perl

Predefined variables

```
$line = "Tell me, O muse, of that ingenious hero
  who travelled far and wide after he had sacked
  the famous town of Troy.";
$line =~ m/,.*,/;
print $&, "\n";
print "Before: ", $`, "\n";
print "After: ", $', "\n";
```

Arrays

```
@array = (1, 2, 3); #Array containing 1, 2, and 3
print $array[1]; #Prints 2
```

Concordances in Perl

```
# Modified from Doug Cooper
($file_name, $string, $width) = @ARGV;
open(FILE, "$file_name")
  || die "Could not open file $file_name.";
while ($line = <FILE>) {
  $text .= $line;
}
$string =~ s/ /\\s/g; # spaces match tabs and new lines
$text =~ s/\n/ /g; # new lines are replaced by spaces
while ($text =~ m/(.{0,$width}$string.{0,$width})/g ) {
# matches the pattern with 0..width to the right and left
  print "$1\n"; #$1 contains the match
}
```

Java: Match

```
Pattern pattern = Pattern.compile("ab+c");
Scanner scan = new Scanner(System.in).useDelimiter("\\n");
while (scan.hasNext()) {
    String line = scan.next();
    Matcher matcher = pattern.matcher(line);
    if (matcher.find()) {
        System.out.println(line);
    }
}
```

Java: Substitute

```
Pattern pattern = Pattern.compile("ab+c");
Scanner scan = new Scanner(System.in).useDelimiter("\\n");
while (scan.hasNext()) {
    String line = scan.next();
    Matcher matcher = pattern.matcher(line);
    if (matcher.find()) {
        line = matcher.replaceAll("ABC");
        System.out.println(line);
    }
}
```

Java: Concordances

```
String file = args[0], pattern = args[1];
int width = new Integer(args[2]);
String text =
  new Scanner(new File(file)).useDelimiter("\\Z").next();
text = text.replaceAll("\\s+", " ");
String concRegex =
  String.format("(.{0,%s}%s.{0,%s})", width, pattern, width);
//System.out.println(concRE);
Pattern concPattern = Pattern.compile(concRegex);
Matcher matcher = concPattern.matcher(text);
while (matcher.find()) {
    System.out.println(matcher.group());
}
```

Approximate String Matching

A set of edit operations that transforms a source string into a target string: copy, substitution, insertion, deletion, reversal (or transposition).
Edits for *acress* from Kernighan et al. (1990).

Typo	Correction	Source	Target	Position	Operation
acress	actress	–	t	2	Deletion
acress	cress	a	–	0	Insertion
acress	caress	ac	ca	0	Transposition
acress	access	r	c	2	Substitution
acress	across	e	o	3	Substitution
acress	acres	s	–	4	Insertion
acress	acres	s	–	5	Insertion

Building a Spell Checker

Spell checkers use a dictionary and a set of transformations to suggest corrections to misspelled words in a text.
Dictionaries are collected from well-written texts: novels, newspapers, etc.

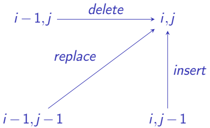
- Given a word in a text not in the dictionary, the spell checker generates all the transformations of this word.
- If we allow only one edit operation on a source string of length n, and if we consider an alphabet of 26 unaccented letters,
  - the deletion will generate n new strings;
  - the insertion, $(n+1) \times 26$ strings;
  - the substitution, $n \times 25$; and
  - the transposition, $n-1$ new strings.
- The spell checker keeps the transformations that are in the dictionary and orders them by frequency to suggest the correct word.

For an implementation, see `http://norvig.com/spell-correct.html`

Building a Spell Checker

```
freq('acres') = 36.
freq('caress') = 3.
freq('cress') = false.
freq('actress') = 7.
freq('access') = 56.
freq('across') = 222.
```

Distance between *ab* and *cb*



Edit distances measure the similarity between strings.

Let us align

a	b	Source
c	b	Destination

b	2		
c	1		
Start	0	1	2
	Start	a	b

Minimum Edit Distance

We compute the minimum edit distance using a matrix where the value at position (i,j) is defined by the recursive formula:

$$edit\_distance(i,j) = \min \begin{pmatrix} edit\_distance(i-1,j) + del\_cost \\ edit\_distance(i-1,j-1) + subst\_cost \\ edit\_distance(i,j-1) + ins\_cost \end{pmatrix}.$$

where $edit\_distance(i,0) = i$ and $edit\_distance(0,j) = j$.

Edit Operations

$$
\begin{tikzcd}
i-1,j \arrow[r, "delete"] & i,j \\
i-1,j-1 \arrow[ur, "replace"] & i,j-1 \arrow[u, "insert"']
\end{tikzcd}
$$

Usually, $del\_cost = ins\_cost = 1$
$subst\_cost = 2$ if $source(i) \neq target(j)$
$subst\_cost = 0$ if $source(i) = target(j)$.

Distance between *ab* and *cb*

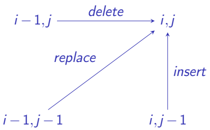
$$\begin{tikzcd}[column sep=large] i-1,j \arrow[rr, "delete"] & & i,j \\ i-1,j-1 \arrow[urr, "replace"] & & i,j-1 \arrow[u, "insert"'] \end{tikzcd}$$

Let us align

a	b	Source
c	b	Destination

b	2		
c	1		
Start	0	1	2
	Start	a	b

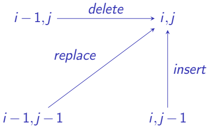
Distance between *ab* and *cb*



Let us align $\frac{\text{a} \quad \text{b} \quad \text{Source}}{\text{c} \quad \text{b} \quad \text{Destination}}$

b	2		
c	1	2	
Start	0	1	2
	Start	a	b

Distance between *ab* and *cb*

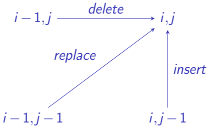


Let us align

a	b	Source
c	b	Destination

b	2	3	
c	1	2	3
Start	0	1	2
	Start	a	b

Distance between *ab* and *cb*



Let us align $\dfrac{\text{a} \quad \text{b} \quad \text{Source}}{\text{c} \quad \text{b} \quad \text{Destination}}$

b	2	3	**2**
c	1	2	3
Start	0	1	2
	Start	a	b

Distance between *language* and *lineage*

e	7								
g	6								
a	5								
e	4								
n	3								
i	2								
l	1								
Start	0	1	2	3	4	5	6	7	8
	Start	l	a	n	g	u	a	g	e

Distance between *language* and *lineage*

e	7	6	5						
g	6	5	4						
a	5	4	3						
e	4	3	4						
n	3	2	3						
i	2	1	2	3	4	5	6	7	8
l	1	0	1	2	3	4	5	6	7
Start	0	1	2	3	4	5	6	7	8
	Start	l	a	n	g	u	a	g	e

Distance between *language* and *lineage*

e	7	6	5	6	5	6	7	6	**5**
g	6	5	4	5	4	5	6	5	6
a	5	4	3	4	5	6	5	6	7
e	4	3	4	3	4	5	6	7	6
n	3	2	3	2	3	4	5	6	7
i	2	1	2	3	4	5	6	7	8
l	1	0	1	2	3	4	5	6	7
Start	0	1	2	3	4	5	6	7	8
	Start	l	a	n	g	u	a	g	e

Perl Code

```
($source, $target) = @ARGV;
$length_s = length($source);
$length_t = length($target);
# Initialize first row and column
for ($i = 0; $i <= $length_s; $i++) {
  $table[$i][0] = $i;
}
for ($j = 0; $j <= $length_t; $j++) {
  $table[0][$j] = $j;
}
# Get the characters. Start index is 0
@source = split(//, $source);
@target = split(//, $target);
```

Perl Code

```
# Fills the table. Start index of rows and columns is 1
for ($i = 1; $i <= $length_s; $i++) {
  for ($j = 1; $j <= $length_t; $j++) {
  # Is it a copy or a substitution?
    $cost = ($source[$i-1] eq $target[$j-1]) ? 0 : 2;
    # Computes the minimum
    $min = $table[$i-1][$j-1] + $cost;
    if ($min > $table[$i][$j-1] + 1) {
      $min = $table[$i][$j-1] + 1;
    }
    if ($min > $table[$i-1][$j] + 1) {
      $min = $table[$i-1][$j] + 1;
    }
    $table[$i][$j] = $min;
  }
}
```

Java Code

```
String source = args[0], target = args[1];
int length_s = source.length();
int length_t = target.length();

int[][] table = new int[length_s + 1][length_t + 1];

// Initialize first row and column
for (int i = 0; i <= length_s; i++) {
    table[i][0] = i;
}
for (int j = 0; j <= length_t; j++) {
    table[0][j] = j;
}
String[] source_char = source.split("");
String[] target_char = target.split("");
```

Java Code

```
// Fills the table. Start index of rows and columns is 1
for (int i = 1; i <= length_s; i++) {
    for (int j = 1; j <= length_t; j++) {
        // Is it a copy or a substitution?
        int cost =
          source_char[i - 1].equals(target_char[j - 1])? 0:2;
        // Computes the minimum
        int min = table[i - 1][j - 1] + cost;
        if (min > table[i][j - 1] + 1) {
            min = table[i][j - 1] + 1;
        }
        if (min > table[i - 1][j] + 1) {
            min = table[i - 1][j] + 1;
        }
        table[i][j] = min;
    }
```

Prolog Code

```
% edit_operation carries out one edit operation
% between a source string and a target string.
edit_operation([Char | Source], [Char | Target], Source,
    Target, ident, 0).
edit_operation([SChar | Source], [TChar | Target], Source,
    Target, sub(SChar,TChar), 2) :-
  SChar \= TChar.
edit_operation([SChar | Source], Target, Source, Target,
    del(SChar), 1).
edit_operation(Source, [TChar | Target], Source, Target,
    ins(TChar), 1).
```

Prolog Code

```
% edit_distance(+Source, +Target, -Edits, ?Cost).
edit_distance(Source, Target, Edits, Cost) :-
  edit_distance(Source, Target, Edits, 0, Cost).

edit_distance([], [], [], Cost, Cost).
edit_distance(Source, Target, [EditOp | Edits], Cost,
    FinalCost) :-
  edit_operation(Source, Target, NewSource, NewTarget,
    EditOp, CostOp),
  Cost1 is Cost + CostOp,
  edit_distance(NewSource, NewTarget, Edits, Cost1,
    FinalCost).
```

Distance between *language* and *lineage*

<table>
<tr><th></th><th>First alignment</th><th>Third alignment</th></tr>
<tr><td>Without epsilon symbols</td><td>l a n g u a g e
| | | | / / /
l i n e a g e</td><td>l a n g u a g e
| | | / / /
l i n e a g e</td></tr>
<tr><td>With epsilon symbols</td><td>l a n g u a g e
| | | | | | | |
l i n e ε a g e</td><td>l a n g u ε a g e
| | | | | | | | |
l i n ε ε e a g e</td></tr>
</table>