Natural Language Processing

Class 02: NLP on the word level

- Morphology: inner structure of words
- Recognizing words
- Morphological parsing
- Finding & correcting spelling errors

VRIJE UNIVERSITEIT BRUSSEL

23 February 2018

Katrien Beuls

Morphology:
The internal structure of words

VRIJE
UNIVERSITEIT
BRUSSEL

What is morphology?

Linguistic definition: The study of the smallest "meaningful" units of language

- Unbound morpheme: *word*, *desk, …*
- Bounded morpheme: -ed, -ing, …
  (to form words such as *walked* and *walking*)

NLP definition: **everything to do with words**

What are the dimensions?

Complexity:

- How many morphemes does a word have?

Morphological processes

- Which functions to morphemes perform?
- E.g. change part of speech: *interact-ion,* change temporal meaning: s*he walk-s* vs. s*he walk-ed*

Morpheme combination

- How do we put morphemes together to form words?

Morphological processes

Derivational: e.g. produce new class of words

- *Modern* (adjective) -> *modern-ize* (verb)

Inflectional: changes "grammatical" meaning

- Temporal distinctions: s*he walk-s* vs. s*he walk-ed*
- Case distinctions ("who does what to whom")

Compounding: creation of new words by combining existing ones

- *Armchair*, *raincoat, blackboard, …*
- *Can get very complex in e.g. Dutch and German: gemeentehuisvuilophaalwagenchauffeurszetelovertrek*

Morpheme combination (1)

Concatenative

- *E.g. re+consider, eas(y)-ier, …*

Infixation

- *E.g. abso+bloody+lutely, …*

Circumfixation

- *E.g. ge-zeg-d* (“said”)

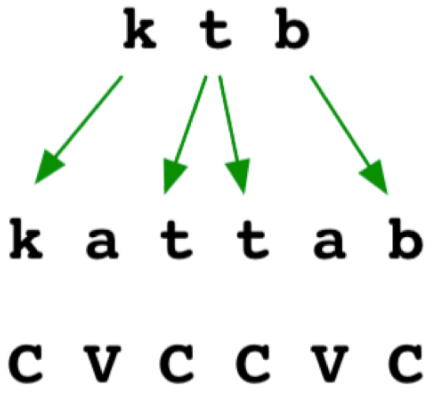
Reduplication

- E.g. Indonesian:
  orang (“man”) -> orangorang (“men”)

Morpheme combination (2)

Templatic combination

- Root is modulated with a template to generate a stem to which other phonemes are added by concatenation, etc.
- E.g. Semitic languages such as Arab
  - Root: *ktb* (semantic domain: “writing”)
  - Template CVCCVC
  - Vocalism

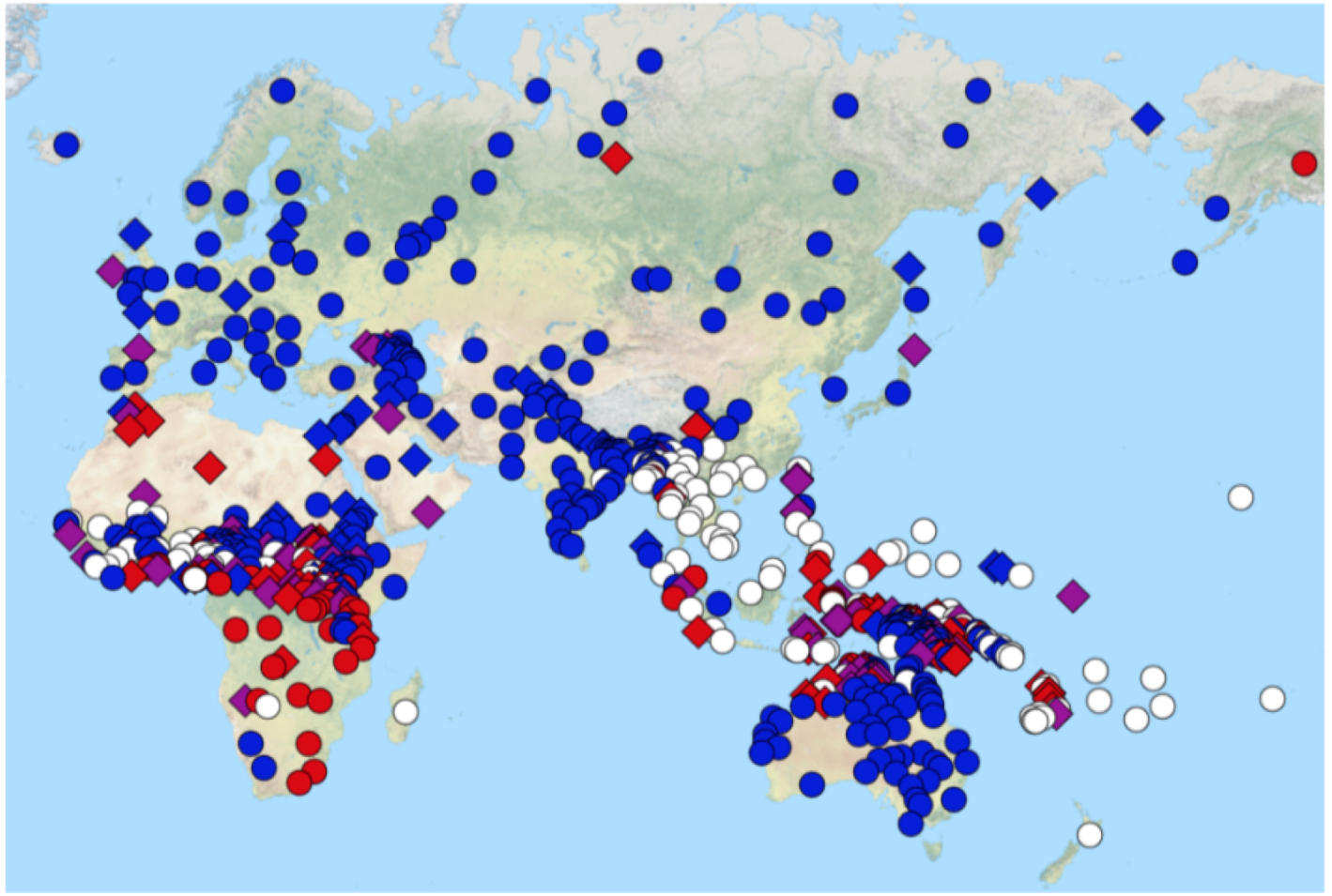


“Zero marking”

Sometimes a distinction is not explicitly marked

- Defaults
- Singular vs. plural (antelope, sheep, …)
- Agreement differences
  (I fly, you fly, we fly, they fly, …)

Massive cross-linguistic variation



Nuaulu

Sentence igt-958:

i -au-sipu api- a

i -au-sipu api- a

3 sg -CAUS-get.down thing- pl

'She put down her things.'

Sentence igt-959:

au we -topi

au we -topi

1SG 1 sg.poss -hat

'my hat'

Source: *World Atlas of Language Structures Online* (http://wals.info)

http://wals.info/feature/26A.html#2/29.2/131.0

Massive cross-linguistic variation

Languages like English:

- Simple morphology
- Can be described using finite-state technologies (or regular languages)

Languages like Arab:

- Complex morphology
- More sophisticated technologies are required

Many languages fall in-between (e.g. Dutch)

Computational models of morphology

VRIJE
UNIVERSITEIT
BRUSSEL

Tasks

Recognition

- Given a string, decide whether that string is accepted by the language model

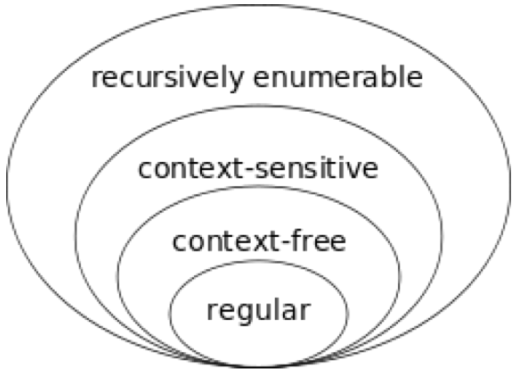
Generation

- Given a lemma, morphological properties, and other relevant representations (e.g. semantics), return an output string

Analysis

- Given a string, return its lemma, morphological properties, semantics, and so on

Naïve solution: word lists



Simple solution

- Just create a list of all possible words and their properties
- Solve tasks by list-checking
- You can even get away with it for English
  (if you have enough memory)

But…

- Not all devices have limitless memory
- Model is brittle (cannot recognize new words)

Instead:

- Use finite-state automata or regular languages

Regular Expressions

VRIJE UNIVERSITEIT BRUSSEL

Slides based on Jurafsky and Martin (3rd ed.), Chapter 2

Regular expressions

Formal language for specifying text strings

How to search for any of these?

- Woodchuck
- woodchuck
- Woodchucks
- woodchucks

http://en.wikipedia.org/wiki/Image:Closeup_groundhog.jpg

Regular expressions: disjunctions

Use square brackets: []

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

Range: 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]

- Caret 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 caret b	Look up a^b now

Regular Expressions: More Disjunction

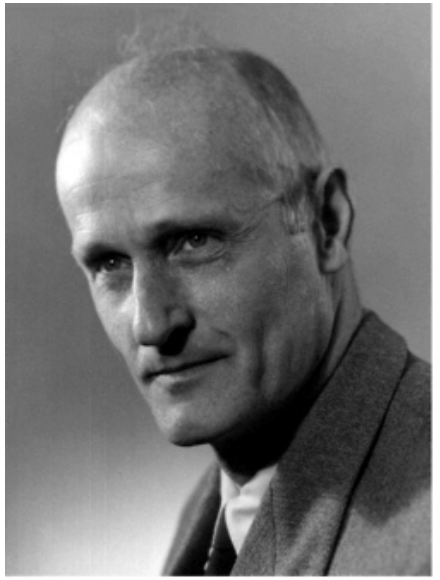
Woodchucks is another name for groundhog!

The pipe | for disjunction

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

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.

1. the — Misses capitalized examples
2. [tT]he — Incorrectly returns “there”, “other”
3. [^a-zA-Z][tT]he[^a-zA-Z]

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)**

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).

Memory function: Registers

Pattern	Matches
([0-9]+) is <\1>	323 is 323
the (.*)er they (.*), the \1er we \2	The harder they run, the harder we run

History: ELIZA

Therapist & patient, MIT, Weizenbaum 1966

Cascade of regular expression substitutions that each match some part of the input lines and changes them

young woman: Men are all alike.
eliza: In what way?
young woman: They're always bugging us about something specific or other.
eliza: Can you think of a specific example?
young woman: Well, my boyfriend made me come here.
eliza: Your boyfriend made you come here?
young woman: He says I'm depressed much of the time.
eliza: I am sorry to hear you are depressed.

Summary

Regular expressions play a surprisingly large role

- Sophisticated sequences of regular expressions are often the first model for any text processing text
- But be careful: they are greedy!

For many hard tasks, we use machine learning classifiers

- But regular expressions are used as features in the classifiers
- Can be very useful in capturing generalizations

Finite State Automata

VRIJE
UNIVERSITEIT
BRUSSEL

Finite-state automata

Mathematical model of computation for computer programs and logic circuits

Consists of finite number of states, transitions between states and actions

Many uses in NLP

- Grapheme to phoneme conversion
- Breaking words into syllables
- Dictionary building
- Parts of machine translation
- Stemming/lemmatization

Formal definition

We define a language to be a set of strings over some alphabet Σ

S is a finite, non-empty set of states

s_0 is a designated initial state, an element of S

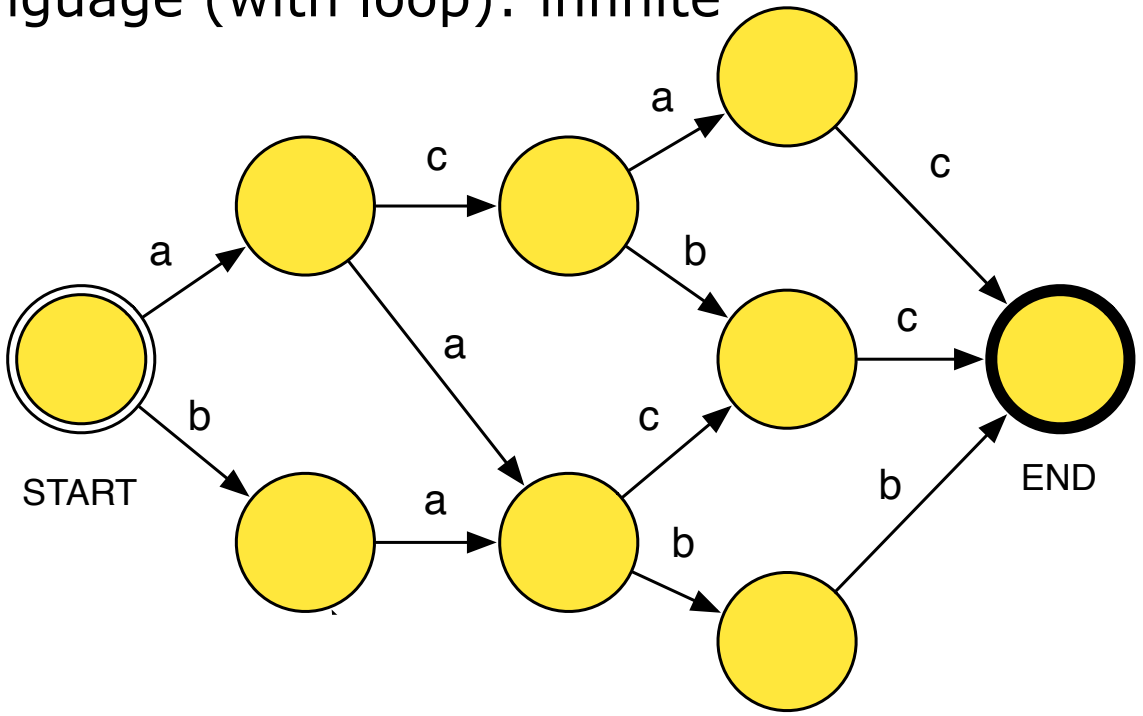
A set of final accepting states $F \subset S$

Edges: given current state s_i and input $x \in \Sigma$, gives new state s_j

Example: FSA over alphabet {a, b, c}

Generated language (without loop):
{ acac, acbc, aacc, aabb, bacc, babb}

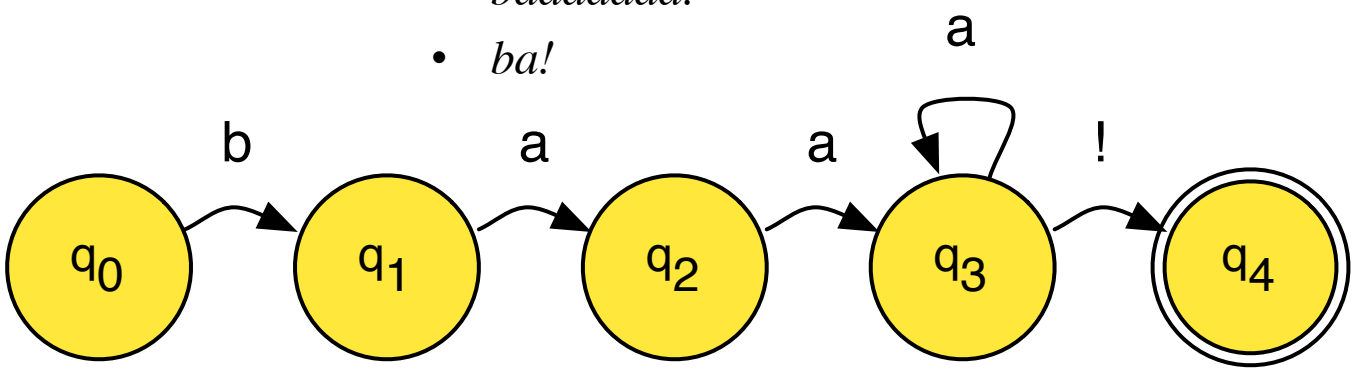
Generated language (with loop): infinite



FSA as a recognizer

Are these strings accepted by the FSA?

- *baa!*
- *baaaaaaa!*
- *ba!*



Regular expression: /baa+!/

FSA for words

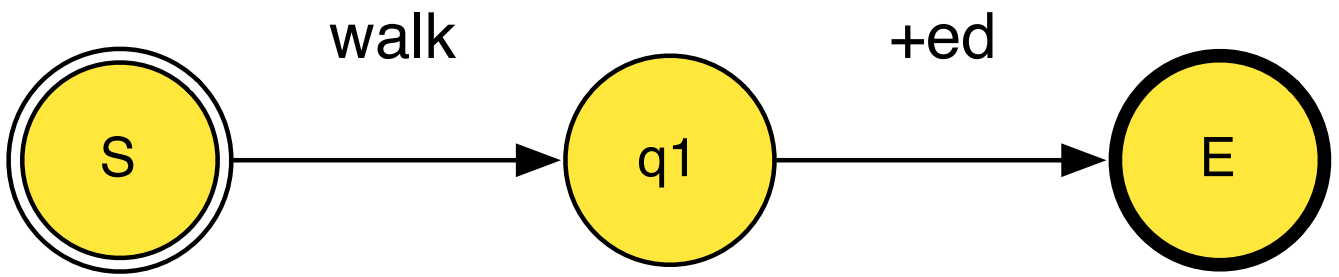
Basic FSA

- Start state
- Transition that emits the word **WALK**
- End state

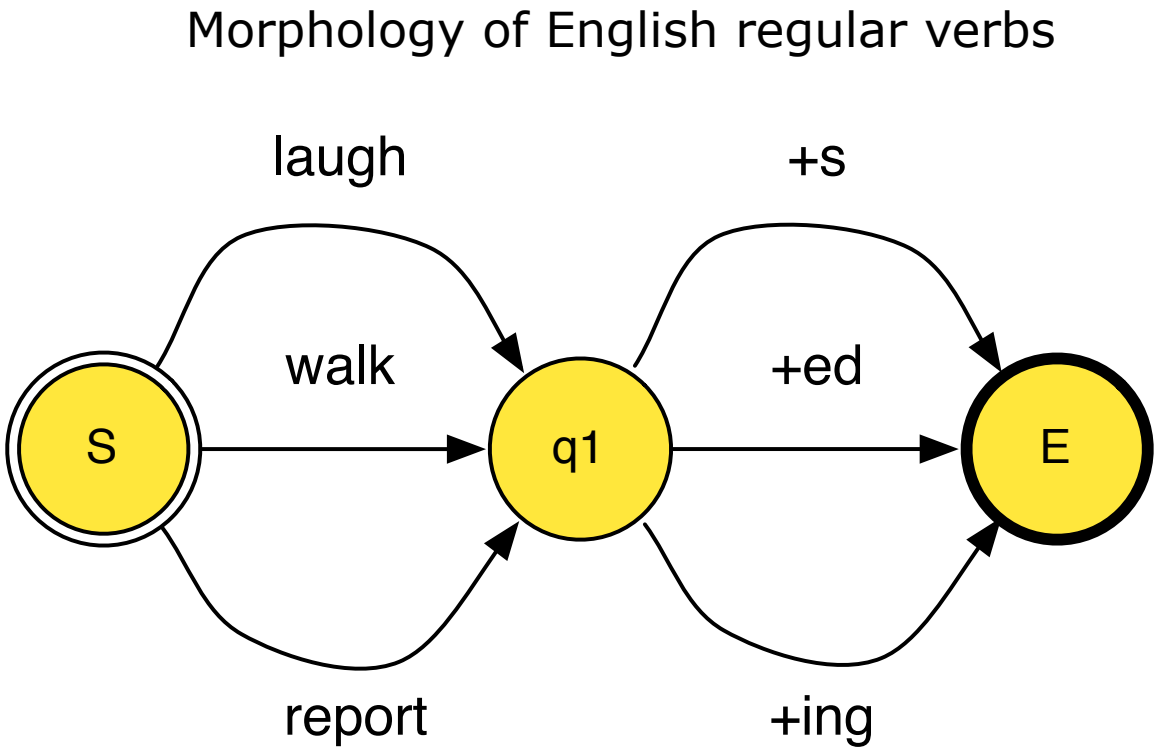
Word + Inflection

Two transitions and an intermediary state

- First transition emits **walk**
- Second transition emits **+ED**

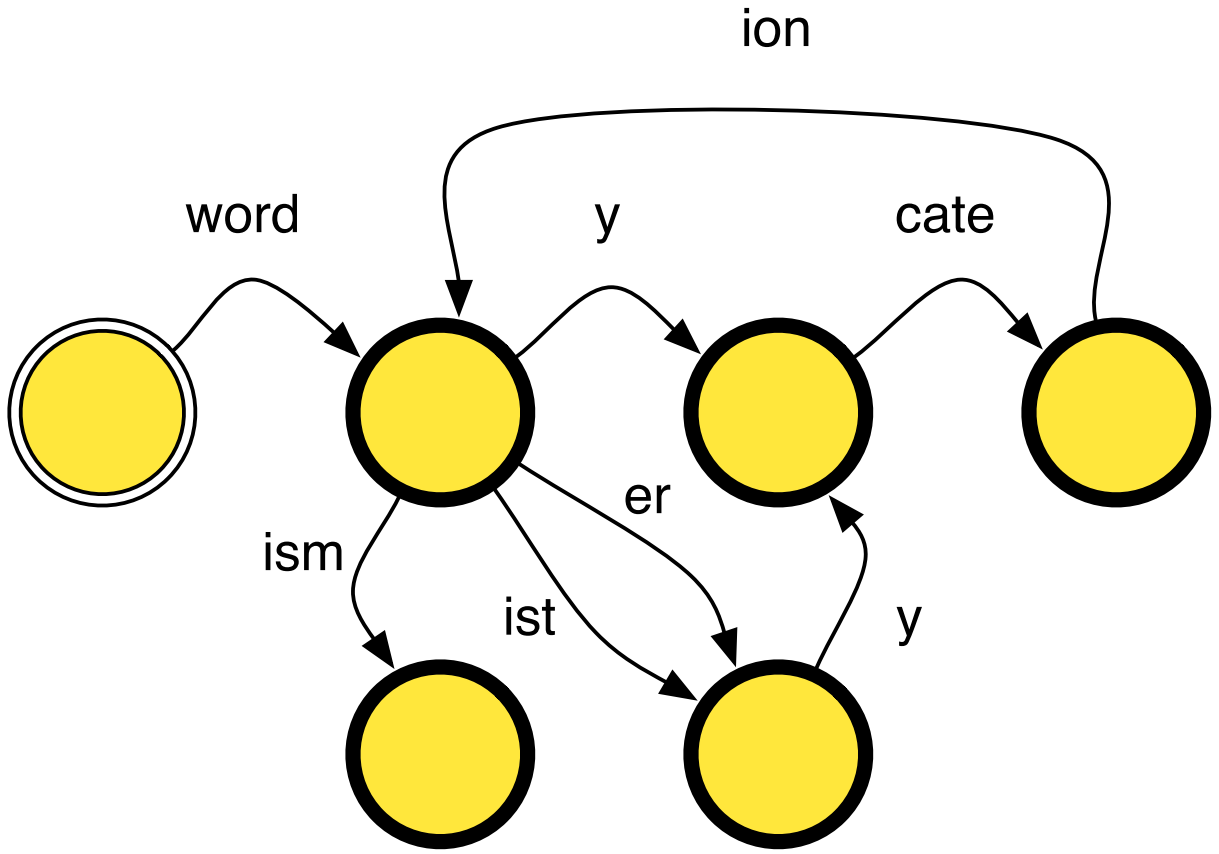


Multiple words and inflections

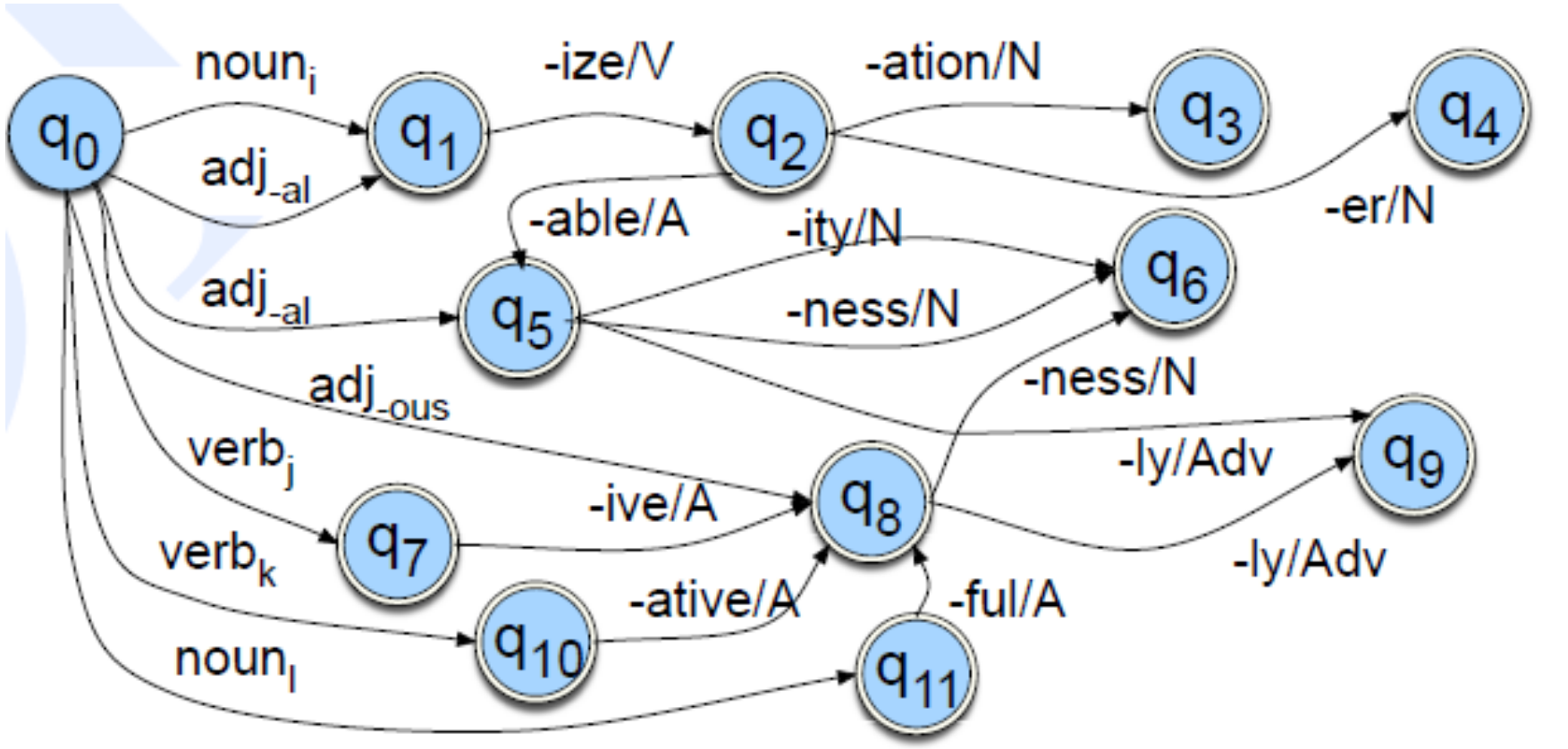


What is missing in this FSA?

Derivational morphology



Combining verbal and nominal derivational morphology



Concatenation

Developing an FSA can get very complex

Build smaller modules

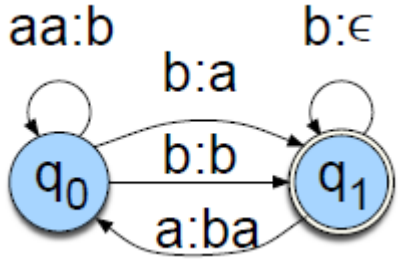
- L: Lexicon
- D: Derivational morphology
- I: Inflectional morphology

Concatenate L+D+I

Morphological analyses
Finite State Transducers

VRIJE
UNIVERSITEIT
BRUSSEL

Finite State Transducers (FST)

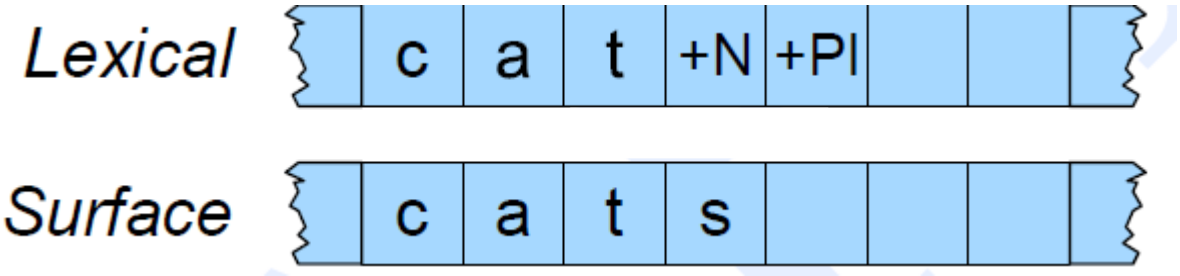


Automaton that maps between 2 sets of symbols

2-tape automaton to generate / recognize pairs of strings

Regular relations

FST as translator: produce a morphological analysis based on a surface form



A set of tuples of strings that can be recognized by an FST is called a regular relation

Each relation is a set of ordered pairs of strings

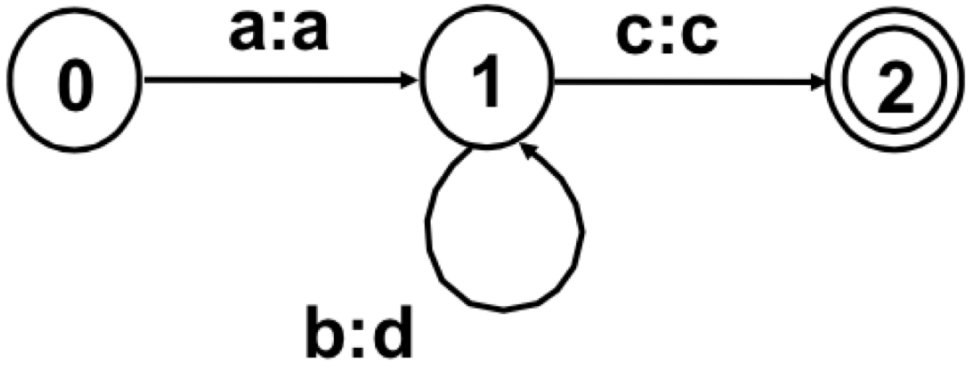
<“cat”, “cats”>, <“child”, “children”>

Regular relation:

{ <ac,ac>, <abc,adc>, <abbc,addc>,<abbbc,adddc>,…}

Description as regular expression:

a [b:d]* c



Two-level morphology:
Assumptions

Koskenniemi (1983)

Simplifying assumption (1)

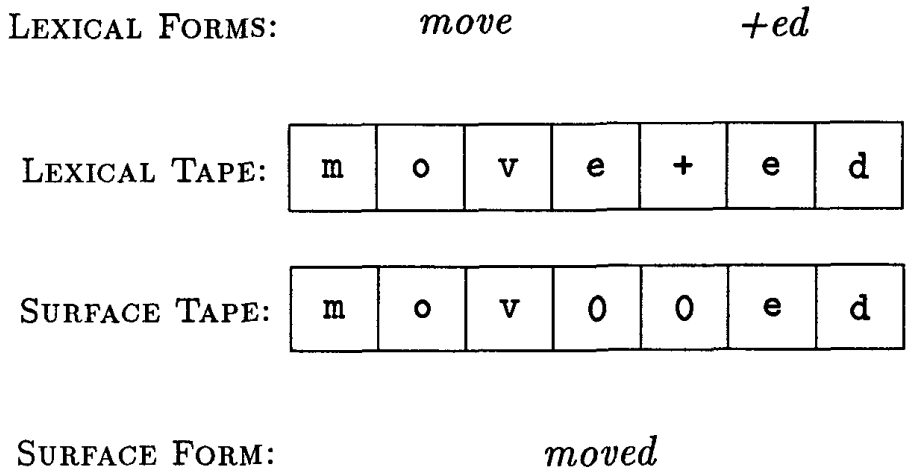
- The set of words in a human language is a regular language.
- In practice: very good approximation, especially for Western European languages

Simplifying assumption (2)

- Words have the following pattern:
  (prefix) + root + (suffix)
  (un+tie, embark+ing, …)

History: http://www.ling.helsinki.fi/~koskenni/esslli-2001-karttunen/

Two-level morphology: Terminology



Representation

- Surface string: *happiest*
- Lexical string: *happy+est*

Aligned correspondence (~ two symbols in FST)

- `happy+est`
- `Happi0est`

Two-level morphology:
Feasible pairs

Aligned correspondence (~ two symbols in FST)

- `happy+est`
- `Happi0est`

Feasible pairs = set of possible lexical and surface correspondences

- Contains insertions, deletions, changes
- {y:i, +:0, …}

Two-level morphology:
Challenge

Examples to accept:

- `stop0+ed` `try+ing` `try+ed`
- `stopp0ed` `try0ing` `tri0ed`

Examples to reject

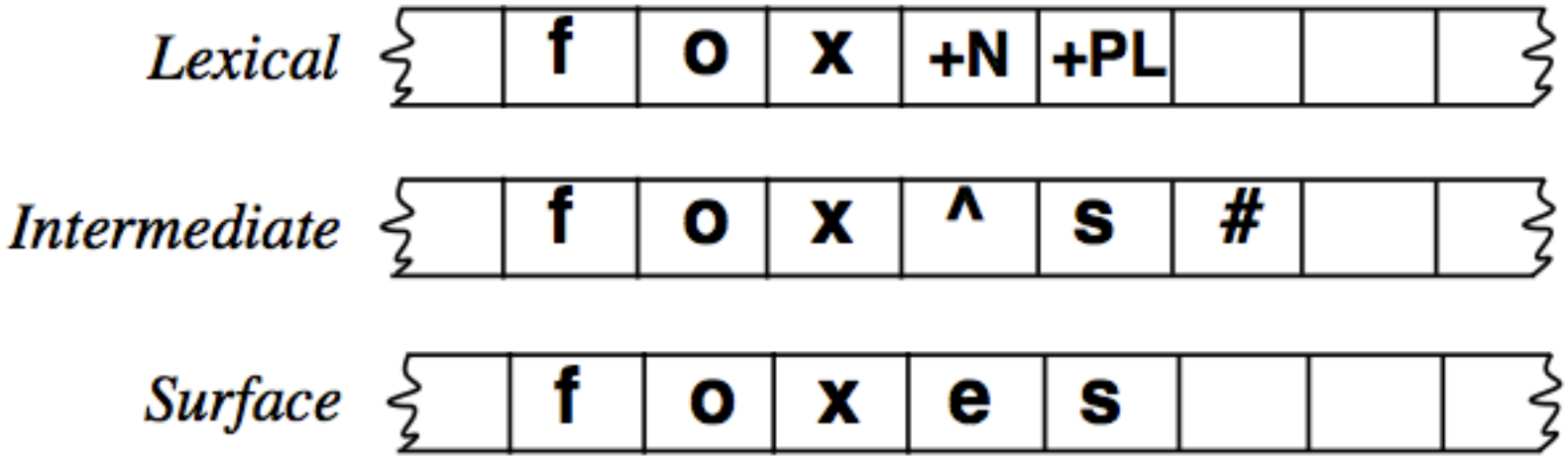
- stop+ed try+ing try+ed
- stop-0ed tri0ing try0ed

Only apply rules under certain conditions!

- Context
- Optional vs. obligatory

Two-level morphology:
Parallel rules

Rules that describe constraints on valid lexical-surface pairings
All rules operate in parallel



Available tools

AT&T Library

http://www2.research.att.com/~fsmtools/fsm/

OpenFST (Google and NYU)

http://www.openfst.org

Carmel Toolkit

http://www.isi.edu/licensed-sw/carmel

FSA Toolkit

http://www-i6.informatik.rwth-aachen.de/~kanthak/fsa.html

Useful references

Hopcroft, Motwani and Ullman, Formal Languages and Automata Theory, Addison Wesley

Beesley and Karttunen, Finite State Morphology, CSLI, 2003

Kaplan and Kay. “Regular Models of Phonological Rule Systems” in Computational Linguistics 20:3, 1994. Pp. 331-378.

Karttunen et al., Regular Expressions for Language Engineering, Natural Language Engineering, 1996

Applications:
Basic Text Processing

VRIJE
UNIVERSITEIT
BRUSSEL

Basic text processing

1. **Segmenting/tokenizing words in running text**
2. Normalizing word formats
3. Segmenting sentences in running text

How many words?

I do uh main- mainly business data processing

- Fragments, filled pauses

Seuss's cat in the hat is different from other cats!

- **Lemma**: same stem, part of speech, rough word sense
  - cat and cats = same lemma
- **Word form**: the full inflected surface form
  - cat and cats = different word forms

How many words?

they lay back on the San Francisco grass and looked at the stars and their

Type: an element of the vocabulary.
Token: an instance of that type in running text.

How many?

- 15 tokens (or 14)
- 13 types (or 12) (or 11?)

How many words?

N = number of tokens
V = vocabulary = set of types
$|V|$ is the size of the vocabulary

	Tokens = N	**Types = |V|**
Switchboard phone conversations	2.4 million	20 thousand
Shakespeare	884,000	31 thousand
Google N-grams	1 trillion	13 million

Issues in Tokenization

Finland's capital	→	Finland Finlands Finland's *?*
what're, I'm, isn't	→	What are, I am, is not
Hewlett-Packard	→	Hewlett Packard ?
state-of-the-art	→	state of the art ?
Lowercase	→	lower-case lowercase lower case ?
San Francisco	→	one token or two?
m.p.h., PhD.	→	??

Simple tokenization in UNIX

Unix for Poets

Kenneth Ward Church
AT&T Research

The problem is to input a text file, say Genesis (a good place to start),[2] and output a list of words in the file along with their frequency counts. The algorithm consists of three steps:

1. Tokenize the text into a sequence of words (tr),
2. Sort the words (sort), and
3. Count duplicates (uniq –c).

The algorithm can be implemented in just three lines of Unix code:

```
tr -sc '[A-Z][a-z]' '[\012*]' < genesis |
sort |
uniq -c > genesis.hist
```

Change all non-alpha to newlines

Sort in alphabetical order

Merge and count each type

This program produces the following output:

```
      1
      2 A
      8 Abel
      1 Abelmizraim
      1 Abidah
      1 Abide
      1 Abimael
     24 Abimelech
    134 Abraham
     59 Abram
        ...
```

Basic text processing

1. Segmenting/tokenizing words in running text
2. **Normalizing word formats**
3. Segmenting sentences in running text

Normalization

Need to “normalize” terms

- Information Retrieval: indexed text & query terms must have same form.
  - We want to match ***U.S.A.*** and ***USA***

We implicitly define equivalence classes of terms

- Deleting periods in a term
- “case folding”: Reducing everything to lowercase
- …

Lemmatization

Reduce inflections or variant forms to base form

- *am, are, is* → *be*
- *car, cars, car's, cars'* → *car*

the boy's cars are different colors
→ *the boy car be different color*

Lemmatization: have to find correct dictionary headword form

Machine translation

- Spanish quiero ('I want'), quieres ('you want') same lemma as querer 'want'

Stemming

Reduce terms to their stems

Stemming is chopping of affixes

- language dependent
- e.g., ***automate(s), automatic, automation*** all reduced to ***automat***.

for example compressed and compression are both accepted as equivalent to compress.

for exampl compress and compress ar both accept as equival to compress

Porter's algorithm
The most common English stemmer

Step 1a

sses	→ ss	caresses	→ caress
ies	→ i	ponies	→ poni
ss	→ ss	caress	→ caress
s	→ ø	cats	→ cat

Step 1b

(*v*)ing	→ ø	walking	→ walk
		sing	→ sing
(*v*)ed	→ ø	plastered	→ plaster
…			

Step 2 (for long stems)

ational	→ ate	relational	→ relate
izer	→ ize	digitizer	→ digitize
ator	→ ate	operator	→ operate
…			

Step 3 (for longer stems)

al	→ ø	revival	→ reviv
able	→ ø	adjustable	→ adjust
ate	→ ø	activate	→ activ
…			

Porter Stemmer

M.F. Porter, 1980, An algorithm for suffix stripping, Program, 14(3), pp 130-137.

You can download versions for many programming languages at:

http://tartarus.org/martin/PorterStemmer/

Viewing morphology in a corpus
Why only strip "–ing" if there is a vowel?

(*v*)ing → Ø	walking	→ walk
	sing	→ sing

Viewing morphology in a corpus
Why only strip –ing if there is a vowel?

(*v*)ing → Ø walking → walk
sing → sing

```
tr -sc 'A-Za-z' '\n' < shakes.txt | grep 'ing$' | sort | uniq -c | sort –nr
```

```
1312 King
 548 being
 541 nothing
 388 king
 375 bring
 358 thing
 307 ring
 152 something
 145 coming
 130 morning
```

```
548 being
541 nothing
152 something
145 coming
130 morning
122 having
120 living
117 loving
116 Being
102 going
```

```
tr -sc 'A-Za-z' '\n' < shakes.txt | grep '[aeiou].*ing$' | sort | uniq -c | sort –nr
```

Dealing with complex morphology is sometimes necessary

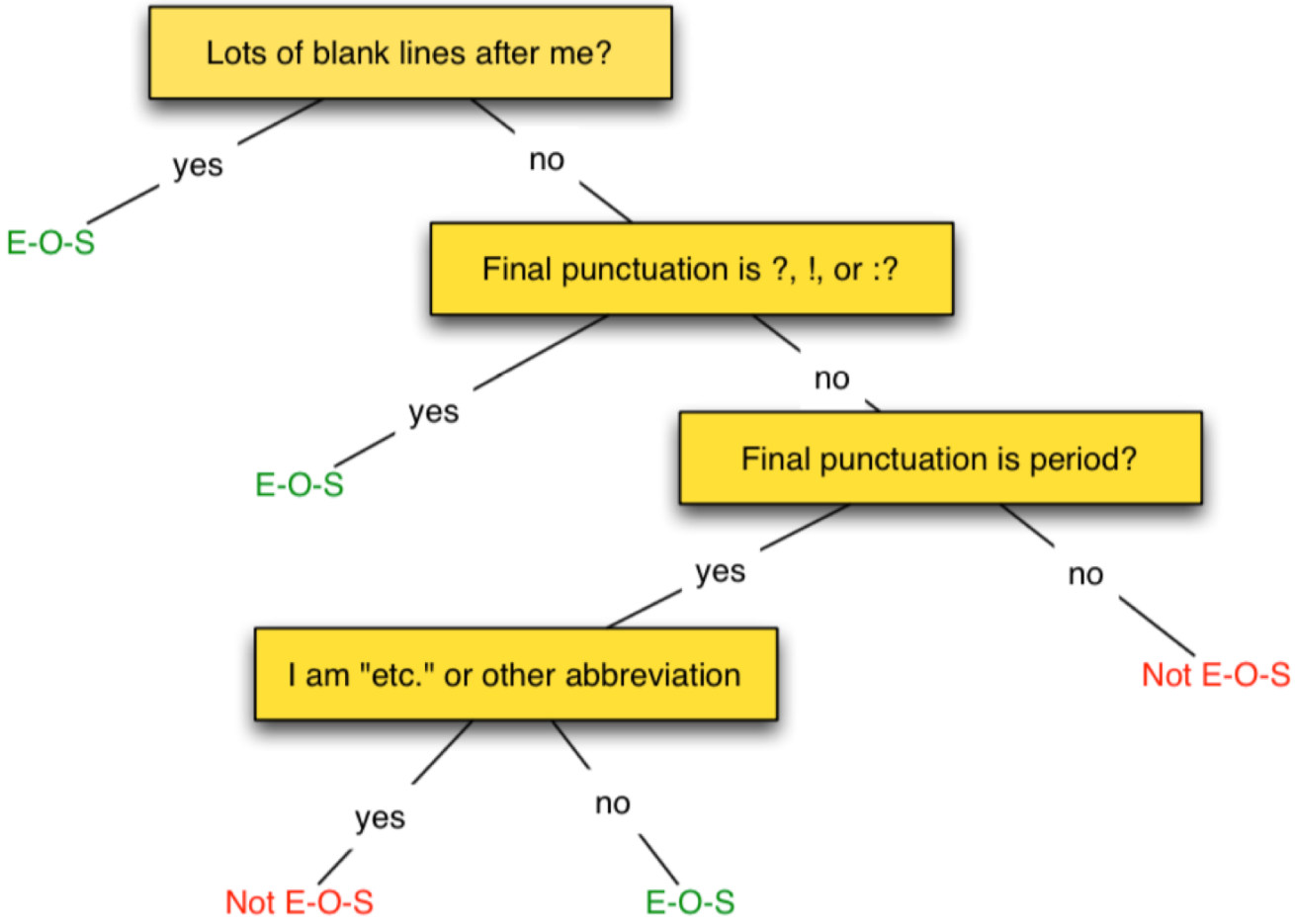
Some languages requires complex morpheme segmentation

- Turkish
- Uygarlastiramadiklarimizdanmissinizcasina
- `(behaving) as if you are among those whom we could not civilize'
- Uygar `civilized' + las `become'
  + tir `cause' + ama `not able'
  + dik `past' + lar 'plural'
  + imiz 'p1pl' + dan 'abl'
  + mis 'past' + siniz '2pl' + casina 'as if'

Basic text processing

1. Segmenting/tokenizing words in running text
2. Normalizing word formats
3. **Segmenting sentences in running text**

Determining if a word is end-of-sentence



More sophisticated decision tree features

Case of word with ".": Upper, Lower, Cap, Number
Case of word after ".": Upper, Lower, Cap, Number

Numeric features

- Length of word with "."
- Probability(word with "." occurs at end-of-s)
- Probability(word after "." occurs at beginning-of-s)

Implementing Decision Trees

A decision tree is just an if-then-else statement

The interesting research is choosing the features

Setting up the structure is often too hard to do by hand

- Hand-building only possible for very simple features, domains
  - For numeric features, it's too hard to pick each threshold
- Instead, structure usually learned by machine learning from a training corpus

We can think of the questions in a decision tree as features that could be exploited by any kind of classifier

- Logistic regression
- SVM
- Neural Nets
- etc.

Summary

Tokenization, lemmatization, stemming and sentence segmentation are almost inevitable when working with raw text.

They are not rocket science but also not trivial. Can be very tedious.

Often highly language dependent.

Use of machine learning when necessary but often regular expressions or iterative rules suffice.

Comparing words: Editing Distance

VRIJE UNIVERSITEIT BRUSSEL

Google

speling

All Images Videos News Shopping More Settings Tools

About 5.000.000 results (0,55 seconds)

Did you mean: ***spelling***

Speling - 7 definities - Encyclo

www.encyclo.nl/begrip/speling ▾ Translate this page

1) Uit `De lagere vaktalen: Timmermanstaal` 1914 wordt gezegd van timmerwerk dat te los in de opening sluit. Dit paneel heeft te veel **speling**.

speling Nederlands woordenboek - Woorden.org

www.woorden.org/woord/speling ▾ Translate this page

Meer info. Slot Webcommerce B.V. maakt gebruik van cookies en daarmee vergelijkbare technieken. woorden.org gebruikt functionele en analytische cookies om u een optimale bezoekerservaring te bieden. Bovendien plaatsen derde partijen tracking cookies om u gepersonaliseerde advertenties te tonen en om buiten de ...

Urban Dictionary: speling

Spelling

Spelling is the combination of alphabetic letters to form a written word. It is a linguistic process of correct writing with the necessary letters and diacritics present in a comprehensible, usually standardized order.
Wikipedia

Feedback

Problem 1:
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:

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

Also for Machine Translation, Information Extraction, Speech Recognition

Solution: Edit Distance

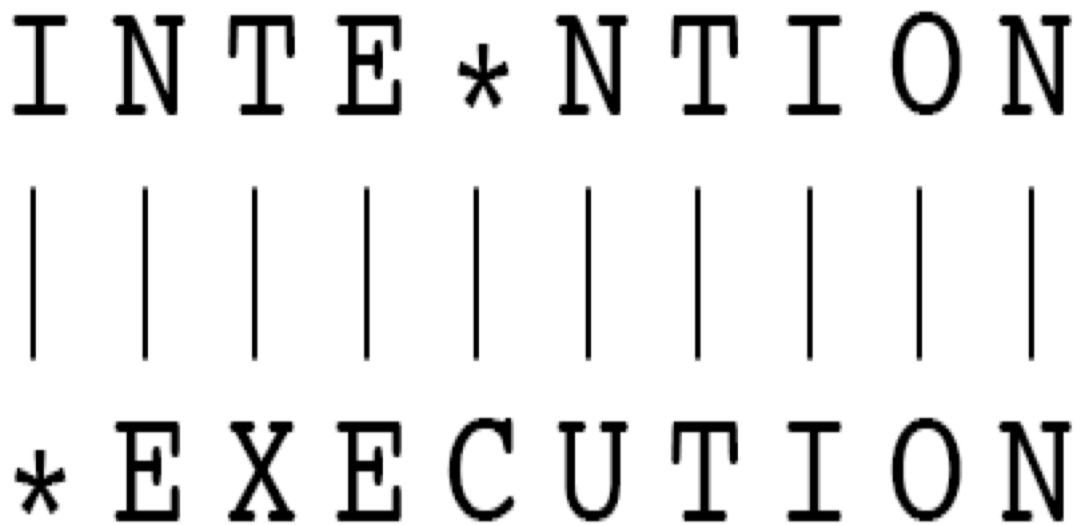
The minimum edit distance between two strings
Is the minimum number of editing operations

- Insertion
- Deletion
- Substitution

Needed to transform one into the other

Minimum Edit Distance

Two strings and their **alignment**:



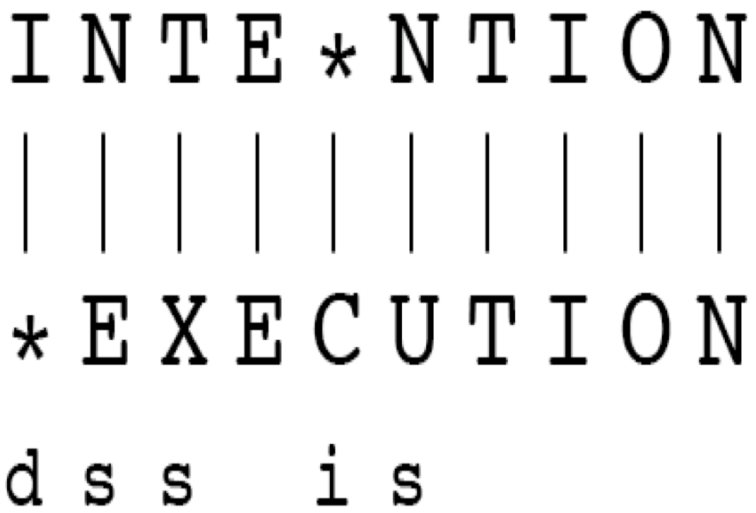
Minimum Edit Distance

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:

–AGG**CTATCAC**CT**GACC**T**C**CA**GG**C**CGA--TGCCC**---

T**AG–CTATCAC--GACC**G**C--GG**T**CGA**TT**TGCCC**GAC

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
```

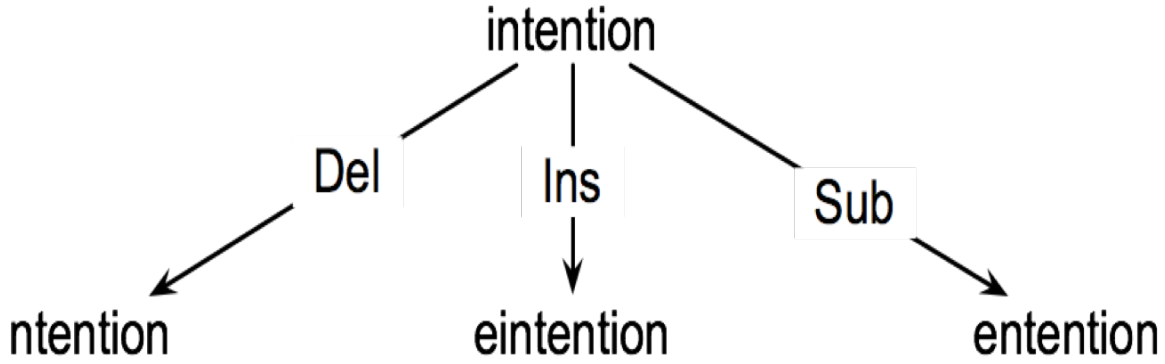
Named Entity Extraction and Entity Co-reference

- 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*)**

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)

Defining Min Edit Distance (Levenshtein)

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
```

Edit distance table

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

Edit distance table

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

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

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

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

Result

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

Computing alignments

Edit distance isn’t sufficient

- We often need to **align** each character of the two strings to each other

We do this by keeping a “backtrace”

Every time we enter a cell, remember where we came from

When we reach the end,

- Trace back the path from the upper right corner to read off the alignment

MinEdit with Backtrace

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**

Adding Backtrace to Minimum Edit Distance

Base conditions:

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

Termination:

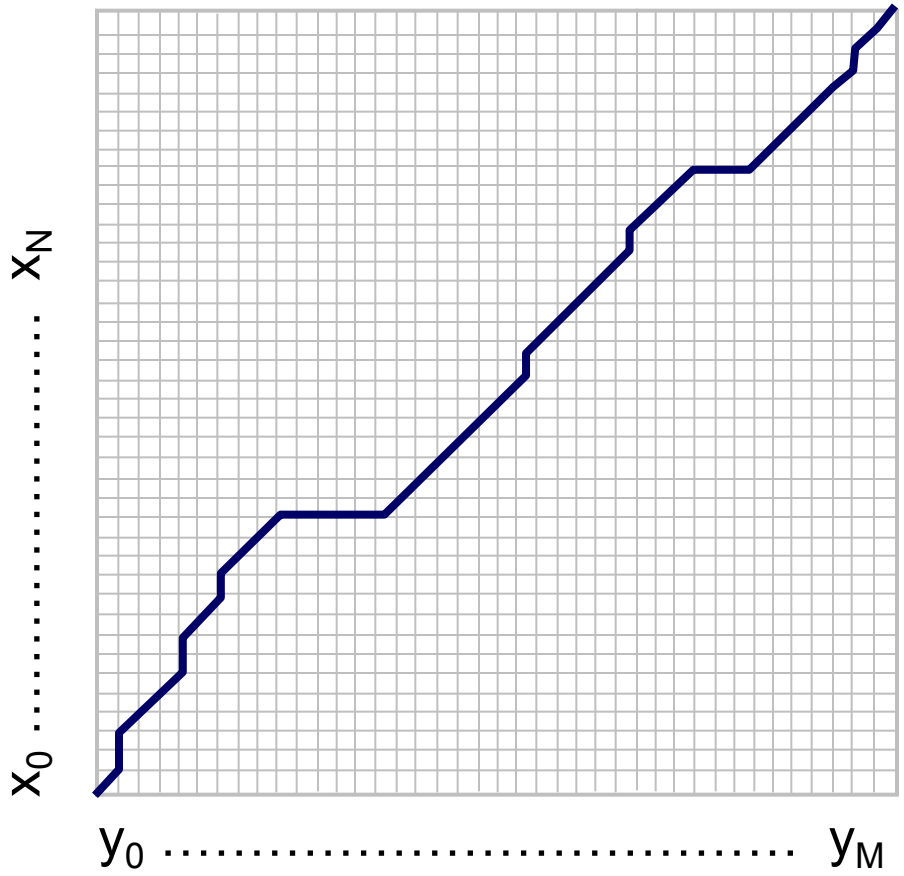
```
D(N,M) is distance
```

Recurrence Relation:

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

                          LEFT          insertion
        ptr(i,j)=         DOWN          deletion
                          DIAG          substitution
```

The Distance Matrix



Every non-decreasing path

from (0,0) to (M, N)

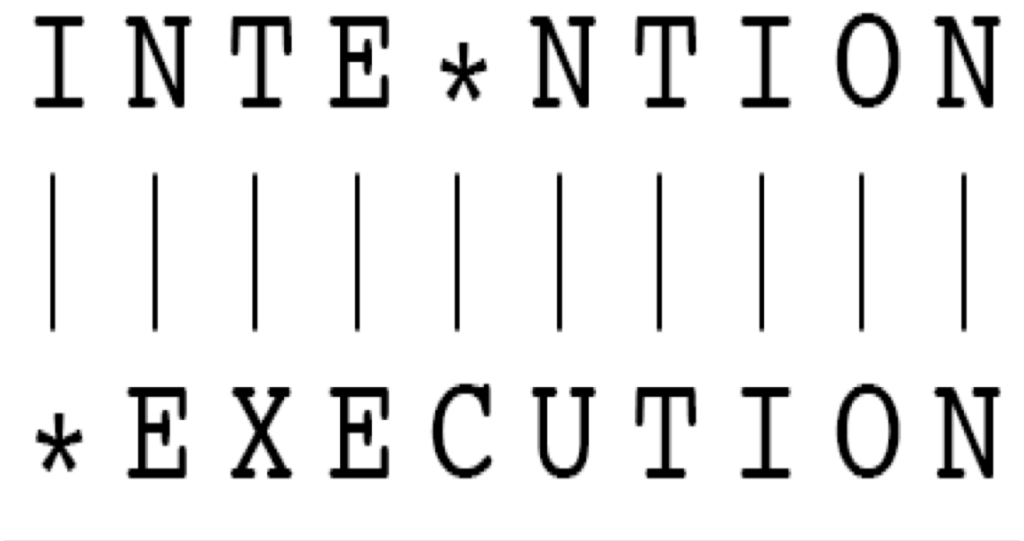
corresponds to
an alignment
of the two sequences

An optimal alignment is composed
of optimal subalignments

Slide adapted from Serafim Batzoglou

Result of Backtrace

Two strings and their alignment



Weighted Edit Distance

Why would we add weights to the computation?

- Spell Correction: some letters are more likely to be mistyped than others
- Biology: certain kinds of deletions or insertions are more likely than others

Confusion matrix for spelling errors

sub[X, Y] = Substitution of X (incorrect) for Y (correct)

X	Y (correct)																									
	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
a	0	0	7	1	342	0	0	2	118	0	1	0	0	3	76	0	0	1	35	9	9	0	1	0	5	0
b	0	0	9	9	2	2	3	1	0	0	0	5	11	5	0	10	0	0	2	1	0	0	8	0	0	0
c	6	5	0	16	0	9	5	0	0	0	1	0	7	9	1	10	2	5	39	40	1	3	7	1	1	0
d	1	10	13	0	12	0	5	5	0	0	2	3	7	3	0	1	0	43	30	22	0	0	4	0	2	0
e	388	0	3	11	0	2	2	0	89	0	0	3	0	5	93	0	0	14	12	6	15	0	1	0	18	0
f	0	15	0	3	1	0	5	2	0	0	0	3	4	1	0	0	0	6	4	12	0	0	2	0	0	0
g	4	1	11	11	9	2	0	0	0	1	1	3	0	0	2	1	3	5	13	21	0	0	1	0	3	0
h	1	8	0	3	0	0	0	0	0	0	2	0	12	14	2	3	0	3	1	11	0	0	2	0	0	0
i	103	0	0	0	146	0	1	0	0	0	0	6	0	0	49	0	0	0	2	1	47	0	2	1	15	0
j	0	1	1	9	0	0	1	0	0	0	0	2	1	0	0	0	0	0	5	0	0	0	0	0	0	0
k	1	2	8	4	1	1	2	5	0	0	0	0	5	0	2	0	0	0	6	0	0	0	4	0	0	3
l	2	10	1	4	0	4	5	6	13	0	1	0	0	14	2	5	0	11	10	2	0	0	0	0	0	0
m	1	3	7	8	0	2	0	6	0	0	4	4	0	180	0	6	0	0	9	15	13	3	2	2	3	0
n	2	7	6	5	3	0	1	19	1	0	4	35	78	0	0	7	0	28	5	7	0	0	1	2	0	2
o	91	1	1	3	116	0	0	0	25	0	2	0	0	0	0	14	0	2	4	14	39	0	0	0	18	0
p	0	11	1	2	0	6	5	0	2	9	0	2	7	6	15	0	0	1	3	6	0	4	1	0	0	0
q	0	0	1	0	0	0	27	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
r	0	14	0	30	12	2	2	8	2	0	5	8	4	20	1	14	0	0	12	22	4	0	0	1	0	0
s	11	8	27	33	35	4	0	1	0	1	0	27	0	6	1	7	0	14	0	15	0	0	5	3	20	1
t	3	4	9	42	7	5	19	5	0	1	0	14	9	5	5	6	0	11	37	0	0	2	19	0	7	6
u	20	0	0	0	44	0	0	0	64	0	0	0	0	2	43	0	0	4	0	0	0	0	2	0	8	0
v	0	0	7	0	0	3	0	0	0	0	0	1	0	0	1	0	0	0	8	3	0	0	0	0	0	0
w	2	2	1	0	1	0	0	2	0	0	1	0	0	0	0	7	0	6	3	3	1	0	0	0	0	0
x	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	9	0	0	0	0	0	0	0
y	0	0	2	0	15	0	1	7	15	0	0	0	2	0	6	1	0	7	36	8	5	0	0	1	0	0
z	0	0	0	7	0	0	0	0	0	0	0	7	5	0	0	0	0	2	21	3	0	0	0	0	3	0


Tab
Caps Lock
Shift
Ctrl
Alt
Alt
Enter
Shift

Peter Norvig's Spell Corrector

Spell corrector in less than 1 page of code

- Accuracy about 70%
- Remember third law of computational linguistics

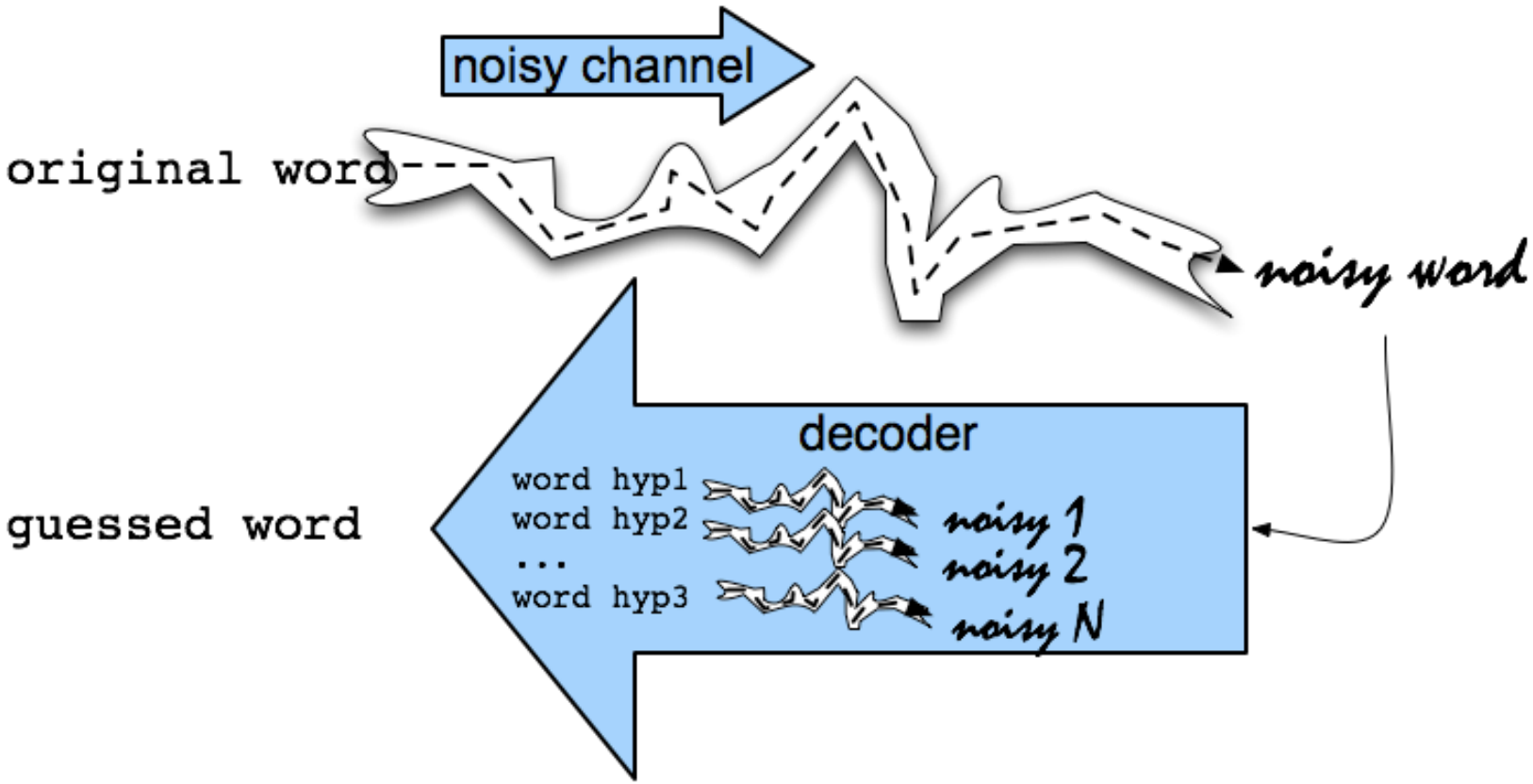
Code & explanation available at
http://norvig.com/spell-correct.html

http://nbviewer.jupyter.org/url/norvig.com/ipython/How%20to%20Do%20Things%20with%20Words.ipynb

Spelling correction:
The noisy channel approach

VRIJE
UNIVERSITEIT
BRUSSEL

Noisy channel intuition



Noisy channel intuition

We see the observation *x* of a word
Find the intended word among corrections *w*

$$\hat{w} = \underset{w \in V}{\operatorname{argmax}} P(w \mid x)$$

$$= \underset{w \in V}{\operatorname{argmax}} \frac{P(x \mid w) P(w)}{P(x)}$$

$$= \underset{w \in V}{\operatorname{argmax}} P(x \mid w) P(w)$$

Control mechanism

language model

error model

Noisy Channel proposed for spelling around 1990

IBM

- Mays, Eric, Fred J. Damerau and Robert L. Mercer. 1991. Context based spelling correction. Information Processing and Management, 23(5), 517–522

AT&T Bell Labs

- Kernighan, Mark D., Kenneth W. Church, and William A. Gale. 1990. A spelling correction program based on a noisy channel model. Proceedings of COLING 1990, 205-210

Example: **acress**

Words at distance 1 of **acress**

Error	Candidate Correction	Correct Letter	Error Letter	Type
acress	actress	t	-	deletion
acress	cress	-	a	insertion
acress	caress	ca	ac	transposition
acress	access	c	r	substitution
acress	across	o	e	substitution
acress	acres	-	s	insertion
acress	acres	-	s	insertion

Example

Candidate generation

- Roughly 75% of the errors are at distance 1
- Almost all errors at distance 2

Also allow transposition and insertion of spaces or hyphens

- inlaw → in-law
- thisidea → this idea

Example: noisy channel model

Candidate Correction	Correct Letter	Error Letter	x|w	P(x|word)
actress	t	-	c|ct	.000117
cress	-	a	a|#	.00000144
caress	ca	ac	ac|ca	.00000164
access	c	r	r|c	.000000209
across	o	e	e|o	.0000093
acres	-	s	es|e	.0000321
acres	-	s	ss|s	.0000342

Channel model

Kerninghan, Church and Gale 1990

$$P(x|w) = \begin{cases} \frac{\text{del}[w_{i-1}, w_i]}{\text{count}[w_{i-1} w_i]}, & \text{if deletion} \\ \frac{\text{ins}[w_{i-1}, x_i]}{\text{count}[w_{i-1}]}, & \text{if insertion} \\ \frac{\text{sub}[x_i, w_i]}{\text{count}[w_i]}, & \text{if substitution} \\ \frac{\text{trans}[w_i, w_{i+1}]}{\text{count}[w_i w_{i+1}]}, & \text{if transposition} \end{cases}$$

How to train a spell corrector?

Supervised learning

- Create an annotated training set
- Errors and manual corrections
- Reliable, but expensive to build

Unsupervised learning

- Work with only raw text of mistakes
- Here: we can exploit a reference point (e.g. English dictionary)

Expectation-Maximization Algorithm (Dempster et al. 1977)

Initialize the model

- for instance uniformly: all misspellings are equally likely

Expectation step: apply the model to the data

- consider all possible corrections, and score them

Maximization step: estimate the model from the data

- given the corrections labeled in the data, learn a model
- using weighted counts based on likelihood of correction

Iterate: go back to step 2 until convergence

Example sentence

The acress played an importamt role im the theare

actress	important	in	theatre
acres	import	is	there
across			

Score possible paths

- The actress played an important role in the theatre.
  $LM(\ldots) \times p_{del}(t) \times p_{sub}(n,m) \times p_{sub}(n,m) \times p_{ins}(t,\epsilon) = 0.000023$
- The actress played an import role in the theatre.
  $LM(\ldots) \times p_{del}(t) \times p_{del}(a) \times p_{del}(n) \times p_{del}(t) \times p_{sub}(n,m) \times p_{ins}(t,\epsilon) = 0.000015$
- …

Count collection

- Normalize the scores, so that they add up to 1
  - *The actress played an important role in the theatre.* del(t), sub(n,m), sub(n,m), ins(t,∈) → 0.46
  - *The actress played an import role in the theatre.* del(t), del(a), del(n), del(t), sub(n,m), ins(t,∈) → 0.30
  - other paths adding up to 1 - 0.46 - 0.30 = 0.24
- Collect counts for correction model
  - del(t) += 0.46 + 0.30 + ...
  - del(a) += 0.30 + ...
  - sub(n,m) += 0.46 + 0.46 + 0.30 + ...

Next iteration

- Score paths with new model

The acress played an importamt role im the theare

actress	**important**	**in**	**theatre**
acres	import	is	there
across			

- We have preference for corrections with seen changes (in many sentences)
- Higher probability is given to them → larger counts are collected → …
- Convergence to improved model

Factors that could influence p(misspelling|word)

The source letter
The target letter
Surrounding letters
The position in the word
Nearby keys on the keyboard
Homology on the keyboard
Pronunciations
Likely morpheme transformations

Interface issues in spell correction

Confident
- Autocorrection

Fairly confident
- Suggest best correction

Less confident
- Give a list of options

Unconfident
- Flag an error

Evaluation

Some spelling error test sets

- Wikipedia's list of common English misspelling
- Aspell filtered version of that list
- Birkbeck spelling error corpus
- Peter Norvig's list of errors (includes Wikipedia and Birkbeck, for training or testing)

State-of-the-Art

Exploit pronunciation

- Toutanova and Moore 2002
- GNU aspell algorithm (Atksinson 2011)

Richer edits

- Brill and Moore 2000
- E.g. ent -> ant, ph→f, le→al

Confusion sets: peace/piece, weather/wether
(Golding and Roth 1999; Bergsma et al 2010)

Summary and next week

Most word-related tasks can be done using regular languages (REs and FSAs)

Basic text processing: normalization

Correcting and aligning sequences: edit distance

Study material:

Jurafsky & Martin 3rd edition: Chapters 2 and 5
(https://web.stanford.edu/~jurafsky/slp3/)

Jurafsky & Martin 2nd edition: Chapter 3

Date	Class	Lecturer	Linguistic phenomenon	techniques	Applications	Assignments
16/02/2018	1	Beuls	introduction / non-randomness	Frequency Distributions: words & collocations	Overview of applications	
23/02/2018	2	Beuls	morphology	regular expressions, finate state automata, string edits	tokenization, lemmatization, spelling correction	
02/03/2018	3	Beuls	n-grams	language models, entropy, perplexity	language guessing	Assignment 1
09/03/2018	4	Van Eecke	Parts of speech	Hidden Markov Models	Labelling tasks	
16/03/2018	5	Beuls	syntax	Dynamic programming	Dependency parsing	
23/03/2018	6	Beuls	Fluid Construction Grammar	Feature structures, unification	Semantic parsing	
30/03/2018	7	Beuls	Pragmatics	Inferencing	Language grounding	Assignment 2
06/04/2018	Easter	break				
13/04/2018	Easter	break				
20/04/2018	8	Beuls	Document semantics	Vector space models, topic models	Information retrieval	
27/04/2018	9	Beuls	Lexical semantics	word spaces, word embeddings	thesaurus extraction, lexical substitution	
04/05/2018	10	Beuls	compositional semantics	formal semantics, distrubutional semantics	summarization, logical inference	
11/05/2018	11	Beuls	opinion mining	sentiment analysis, paraphrase detection	content analytics	Assignment3
18/05/2018	12	Beuls	Machine translation	rule-based, statistical, neural MT	general vs domain-specific MT	
25/05/2018	13	Beuls	reserve class/ question time			