

# Language and Computers

## Writers' Aids

Based on Dickinson, Brew, & Meurers (2013)

### Introduction

### Error causes

Keyboard mistypings

Phonetic errors

Knowledge problems

### Challenges

Tokenization

Inflection

Productivity

### Non-word error detection

Dictionaries

N-gram analysis

### Isolated-word error correction

Rule-based methods

Similarity key techniques

Probabilistic methods

Minimum edit distance

### Error correction for web queries

### Grammar correction

Syntax and Computing

Grammar correction rules

### Caveat emptor

# Why people care about spelling

- ▶ Misspellings can cause misunderstandings
- ▶ Standard spelling makes it easy to organize words & text:
  - ▶ e.g., Without standard spelling, how would you look up things in a lexicon or thesaurus?
  - ▶ e.g., Optical character recognition software (OCR) can use knowledge about standard spelling to recognize scanned words even for hardly legible input.
- ▶ Standard spelling makes it possible to provide a single text, accessible to a wide range of readers (different backgrounds, speaking different dialects, etc.).
- ▶ Using standard spelling can make a good impression in social interaction.

## Introduction

### Error causes

Keyboard mistypings  
Phonetic errors  
Knowledge problems

### Challenges

Tokenization  
Inflection  
Productivity

### Non-word error detection

Dictionaries  
N-gram analysis

### Isolated-word error correction

Rule-based methods  
Similarity key techniques  
Probabilistic methods  
Minimum edit distance

### Error correction for web queries

### Grammar correction

Syntax and Computing  
Grammar correction rules

### Caveat emptor

# How are spell checkers used?

- ▶ **interactive spelling checkers** = spell checker detects errors as you type.
  - ▶ It may or may not make suggestions for correction.
  - ▶ It needs a “real-time” response (i.e., must be fast)
  - ▶ It is up to the human to decide if the spell checker is right or wrong, and so we may not require 100% accuracy (especially with a list of choices)
- ▶ **automatic spelling correctors** = spell checker runs on a whole document, finds errors, and corrects them
  - ▶ A much more difficult task.
  - ▶ A human may or may not proofread the results later.

## Introduction

### Error causes

Keyboard mistypings

Phonetic errors

Knowledge problems

### Challenges

Tokenization

Inflection

Productivity

### Non-word error detection

Dictionaries

N-gram analysis

### Isolated-word error correction

Rule-based methods

Similarity key techniques

Probabilistic methods

Minimum edit distance

### Error correction for web queries

### Grammar correction

Syntax and Computing

Grammar correction rules

### Caveat emptor

- ▶ There are two distinct tasks:
  - ▶ **error detection** = simply find the misspelled words
  - ▶ **error correction** = correct the misspelled words
- ▶ e.g., It might be easy to tell that *ater* is a misspelled word, but what is the correct word? *water*? *later*? *after*?
  - ▶ Note that detection is a prerequisite for correction.

## Introduction

### Error causes

Keyboard mistypings

Phonetic errors

Knowledge problems

### Challenges

Tokenization

Inflection

Productivity

### Non-word error detection

Dictionaries

N-gram analysis

### Isolated-word error correction

Rule-based methods

Similarity key techniques

Probabilistic methods

Minimum edit distance

### Error correction for web queries

### Grammar correction

Syntax and Computing

Grammar correction rules

### Caveat emptor

# Error causes

## Keyboard mistypings

## Space bar issues

- ▶ **run-on** errors = two separate words become one
  - ▶ e.g., *the fuzz* becomes *thefuzz*
- ▶ **split** errors = one word becomes two separate items
  - ▶ e.g., *equalization* becomes *equali zation*
- ▶ Note that the resulting items might still be words!
  - ▶ e.g., *a tollway* becomes *atoll way*

# Error causes

## Keyboard mistypings (cont.)

### Keyboard proximity

- ▶ e.g., *Jack* becomes *Hack* since *h* and *j* are next to each other on a typical American keyboard

### Physical similarity

- ▶ similarity of shape, e.g., mistaking two physically similar letters when typing up something handwritten
  - ▶ e.g., *tight* for *fight*

### phonetic errors

= errors based on the sounds of a language (not necessarily on the letters)

- ▶ **homophones** = two words which sound the same
  - ▶ e.g., *red/read* (past tense), *cite/site/sight*, *they're/their/there*
- ▶ letter/word substitution: replacing a letter (or sequence of letters) with a similar-sounding one
  - ▶ e.g., *John kracked his nuckles.*  
instead of *John cracked his knuckles.*

# Error causes

## Knowledge problems

- ▶ not knowing a word and guessing its spelling (can be phonetic)
  - ▶ e.g., *sientist*
- ▶ not knowing a rule and guessing it
  - ▶ e.g., Do we double a consonant for *ing* words?  
*jog* → *joging*  
*joke* → *jokking*
- ▶ knowing something is odd about the spelling, but guessing the wrong thing
  - ▶ e.g., typing *siscors* for the non-regular *scissors*



# Challenges & Techniques for spelling correction

Before we turn to how we detect spelling errors, we'll look briefly at three issues:

- ▶ **Tokenization:** What is a word?
- ▶ **Inflection:** How are some words related?
- ▶ **Productivity of language:** How many words are there?

How we handle these issues determines how we build a dictionary.

And then we'll turn to the techniques used:

- ▶ Non-word error detection
- ▶ Isolated-word error correction
- ▶ Context-dependent word error detection and correction  
→ grammar correction

Introduction

Error causes

Keyboard mistypings

Phonetic errors

Knowledge problems

Challenges

Tokenization

Inflection

Productivity

Non-word error  
detection

Dictionaries

N-gram analysis

Isolated-word error  
correction

Rule-based methods

Similarity key techniques

Probabilistic methods

Minimum edit distance

Error correction for  
web queries

Grammar correction

Syntax and Computing

Grammar correction rules

Caveat emptor

# Tokenization

Intuitively a “word” is simply whatever is between two spaces, but this is not always so clear.

- ▶ contractions = two words combined into one
  - ▶ e.g., *can't, he's, John's [car]* (vs. *his car*)
- ▶ multi-token words = (arguably) a single word with a space in it
  - ▶ e.g., *New York, in spite of, deja vu*
- ▶ hyphens (note: can be ambiguous if a hyphen ends a line)
  - ▶ Some are always a single word: *e-mail, co-operate*
  - ▶ Others are two words combined into one:  
*Columbus-based, sound-change*
- ▶ Abbreviations: may stand for multiple words
  - ▶ e.g., *etc.* = *et cetera*, *ATM* = *Automated Teller Machine*

- ▶ A word in English may appear in various guises due to word **inflections** = word endings which are fairly systematic for a given part of speech
  - ▶ plural noun ending: *the boy* + *s* → *the boys*
  - ▶ past tense verb ending: *walk* + *ed* → *walked*
- ▶ This can make spell-checking hard:
  - ▶ There are exceptions to the rules: *\*mans*, *\*runned*
  - ▶ There are words which look like they have a given ending, but they don't: *Hans*, *deed*

- ▶ part of speech change: nouns can be verbified
  - ▶ *emailed* is a common new verb coined after the noun *email*
- ▶ morphological productivity: prefixes and suffixes can be added
  - ▶ e.g., I can speak of *un-email-able* for someone who you can't reach by email.
- ▶ words entering and exiting the lexicon, e.g.:
  - ▶ *thou*, or *spleet* 'split' (*Hamlet III.2.10*) are on their way out
  - ▶ New words all the time: *omnishambles*, *phablet*, *supersize*, ...

And now the techniques ...

- ▶ **non-word error detection** is essentially the same thing as **word recognition** = splitting up “words” into true words and non-words.
- ▶ How is non-word error detection done?
  - ▶ using a dictionary (construction and lookup)
  - ▶ n-gram analysis

## Intuition:

- ▶ Have a complete list of words and check the input words against this list.
- ▶ If it's not in the dictionary, it's not a word.

## Two aspects:

- ▶ **Dictionary construction** = build the dictionary (what do you put in it?)
- ▶ **Dictionary lookup** = lookup a potential word in the dictionary (how do you do this quickly?)

# Dictionary construction

- ▶ Do we include inflected words? i.e., words with prefixes and suffixes already attached.
  - ▶ Lookup can be faster
  - ▶ But takes more space & doesn't account for new formations (e.g., *google* → *googled*)
- ▶ Want the dictionary to have only the word relevant for the user → **domain-specificity**
  - ▶ e.g., For most people *memoize* is a misspelled word, but in computer science this is a technical term
- ▶ Foreign words, hyphenations, derived words, proper nouns, and new words will always be problems
  - ▶ we cannot predict these words until humans have made them words.
- ▶ Dictionary should be dialectally consistent.
  - ▶ e.g., include only *color* or *colour* but not both

# N-gram analysis

- ▶ An **n-gram** here is a string of  $n$  letters.

<i>a</i>	1-gram (unigram)
<i>at</i>	2-gram (bigram)
<i>ate</i>	3-gram (trigram)
<i>late</i>	4-gram
⋮	⋮

- ▶ We can use this n-gram information to define what the possible strings in a language are.
  - ▶ e.g., *po* is a possible English string, whereas *kvt* is not.

This is more useful to correct optical character recognition (OCR) output, but we'll still take a look.



# Bigram array

- ▶ We can define a **bigram array** = information stored in a tabular fashion.
- ▶ An example, for the letters *k*, *l*, *m*, with examples in parentheses

	...	k	l	m	...
...					
k	0	1 ( <i>tackle</i> )	1 ( <i>Hackman</i> )		
l	1 ( <i>elk</i> )	1 ( <i>hello</i> )	1 ( <i>alms</i> )		
m	0	0	1 ( <i>hammer</i> )		
...					

- ▶ The first letter of the bigram is given by the vertical letters (i.e., down the side), the second by the horizontal ones (i.e., across the top).
- ▶ This is a **non-positional bigram array** = the array 1's and 0's apply for a string found anywhere within a word (beginning, 4th character, ending, etc.).

- ▶ To store information specific to the beginning, the end, or some other position in a word, we can use a **positional bigram array** = the array only applies for a given position in a word.
- ▶ Here's the same array as before, but now only applied to word endings:

	...	k	l	m	...
⋮					
k		0	0	0	
l		1 ( <i>elk</i> )	1 ( <i>hall</i> )	1 ( <i>elm</i> )	
m		0	0	0	
⋮					

- ▶ Having discussed how errors can be detected, we want to know how to correct these misspelled words:
  - ▶ The most common method is **isolated-word error correction** = correcting words without taking context into account.
  - ▶ Note: This technique can only handle errors that result in non-words.
- ▶ Knowledge about what is a typical error helps in finding correct word.

# Knowledge about typical errors

- ▶ word length effects: most misspellings are within two characters in length of original
  - When searching for the correct spelling, we do not usually need to look at words with greater length differences.
- ▶ first-position error effects: the first letter of a word is rarely erroneous
  - When searching for the correct spelling, the process is sped up by being able to look only at words with the same first letter.

- ▶ Many different methods are used; we will briefly look at four methods:
  - ▶ rule-based methods
  - ▶ similarity key techniques
  - ▶ probabilistic methods
  - ▶ minimum edit distance
- ▶ The methods play a role in one of the three basic steps:
  1. Detection of an error (discussed above)
  2. Generation of candidate corrections
    - ▶ rule-based methods
    - ▶ similarity key techniques
  3. Ranking of candidate corrections
    - ▶ probabilistic methods
    - ▶ minimum edit distance

One can generate correct spellings by writing rules:

- ▶ Common misspelling rewritten as correct word:
  - ▶ e.g., *hte* → *the*
- ▶ Rules
  - ▶ based on inflections:
    - ▶ e.g., *VCing* → *VCCing*, where
      - V** = letter representing vowel,  
basically the regular expression [aeiou]
      - C** = letter representing consonant,  
basically [bcdfghjklmnpqrstvwxyz]
  - ▶ based on other common spelling errors (such as keyboard effects or common transpositions):
    - ▶ e.g., *CsC* → *CaC*
    - ▶ e.g., *cie* → *cei*

Introduction

Error causes

Keyboard mistypings

Phonetic errors

Knowledge problems

Challenges

Tokenization

Inflection

Productivity

Non-word error  
detection

Dictionaries

N-gram analysis

Isolated-word error  
correction

Rule-based methods

Similarity key techniques

Probabilistic methods

Minimum edit distance

Error correction for  
web queries

Grammar correction

Syntax and Computing

Grammar correction rules

Caveat emptor

# Similarity key techniques (SOUNDEX)

- ▶ Problem: How can we find a list of possible corrections?
- ▶ Solution: Store words in different boxes in a way that puts the similar words together.
- ▶ Example:
  1. Start by storing words by their first letter (first letter effect),
    - ▶ e.g., *punc* starts with the code P.
  2. Then assign numbers to each letter
    - ▶ e.g., 0 for vowels, 1 for *b, p, f, v* (all bilabials), and so forth, e.g., *punc* → P052
  3. Then throw out all zeros and repeated letters,
    - ▶ e.g., P052 → P52.
  4. Look for real words within the same box,
    - ▶ e.g., *punk* is also in the P52 box.

# How is a mistyped word related to the intended?

For ranking errors, it helps to know:

## Types of operations

- ▶ **insertion** = a letter is added to a word
- ▶ **deletion** = a letter is deleted from a word
- ▶ **substitution** = a letter is put in place of another one
- ▶ **transposition** = two adjacent letters are switched

Note that the first two alter the length of the word, whereas the second two maintain the same length.



Two main probabilities are taken into account:

- ▶ **transition probabilities** = probability (chance) of going from one letter to the next.
  - ▶ e.g., What is the chance that *a* will follow *p* in English?  
That *u* will follow *q*?
- ▶ **confusion probabilities** = probability of one letter being mistaken (substituted) for another (can be derived from a confusion matrix)
  - ▶ e.g., What is the chance that *q* is confused with *p*?
  - ▶ We also calculate probabilities for insertions & deletions after particular letters

# Confusion probabilities

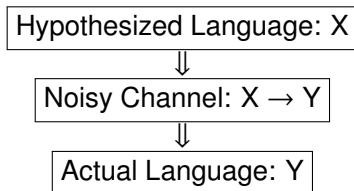
- ▶ It is impossible to fully investigate all possible error causes and how they interact, but we can learn from watching how often people make errors and where.
- ▶ One way is to build a **confusion matrix** = a table indicating how often one letter is mistyped for another

		correct				
		...	r	s	t	...
typed	:					
	r		n/a	12	22	
	s		14	n/a	15	
	t		11	37	n/a	
	:					

(cf. Kernighan et al 1999)

# The Noisy Channel Model

Probabilities can be modeled with the **noisy channel model**



Goal: Recover X from Y

- ▶ The noisy channel model has been very popular in speech recognition, among other fields

(Thanks to Mike White for the slides on the Noisy Channel Model)

Introduction

Error causes

Keyboard mistypings

Phonetic errors

Knowledge problems

Challenges

Tokenization

Inflection

Productivity

Non-word error  
detection

Dictionaries

N-gram analysis

Isolated-word error  
correction

Rule-based methods

Similarity key techniques

Probabilistic methods

Minimum edit distance

Error correction for  
web queries

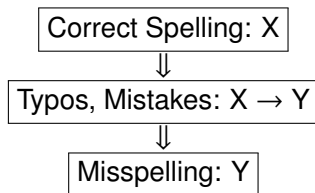
Grammar correction

Syntax and Computing

Grammar correction rules

Caveat emptor

# Noisy Channel Spelling Correction



Goal: Recover correct spelling  $X$  from misspelling  $Y$

- ▶ Noisy word:  $Y$  = observation (incorrect spelling)
- ▶ We want to find the word ( $X$ ) which maximizes:  $P(X|Y)$ , i.e., the probability of  $X$ , given that  $Y$  has been seen

Introduction

Error causes

Keyboard mistypings

Phonetic errors

Knowledge problems

Challenges

Tokenization

Inflection

Productivity

Non-word error  
detection

Dictionaries

N-gram analysis

Isolated-word error  
correction

Rule-based methods

Similarity key techniques

Probabilistic methods

Minimum edit distance

Error correction for  
web queries

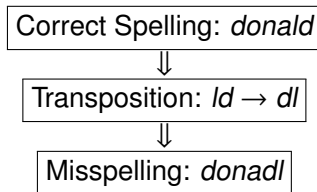
Grammar correction

Syntax and Computing

Grammar correction rules

Caveat emptor

# Example



Goal: Recover correct spelling *donald* from misspelling *donadl* (i.e.,  $P(\text{donald}|\text{donadl})$ )

$p(x|y)$  is the probability of  $x$  given  $y$

- ▶ Let's say that *yogurt* appears 20 times in a text of 10,000 words  
→  $p(\text{yogurt}) = 20/10,000 = 0.002$
- ▶ Now, let's say *frozen* appears 50 times in the text, and *yogurt* appears 10 times after it  
→  $p(\text{yogurt}|\text{frozen}) = 10/50 = 0.20$

With  $X$  as the correct word and  $Y$  as the misspelling ...

$P(X|Y)$  is impossible to calculate directly, so we use:

- ▶  $P(Y|X)$  = the probability of the observed misspelling given the correct word
- ▶  $P(X)$  = the probability of the (correct) word occurring anywhere in the text

Bayes Rule allows us to calculate  $p(X|Y)$  in terms of  $p(Y|X)$ :

$$(1) \text{ **Bayes Rule: } P(X|Y) = \frac{P(Y|X)P(X)}{P(Y)}**$$

Introduction

Error causes

Keyboard mistypings

Phonetic errors

Knowledge problems

Challenges

Tokenization

Inflection

Productivity

Non-word error  
detection

Dictionaries

N-gram analysis

Isolated-word error  
correction

Rule-based methods

Similarity key techniques

Probabilistic methods

Minimum edit distance

Error correction for  
web queries

Grammar correction

Syntax and Computing

Grammar correction rules

Caveat emptor

# The Noisy Channel and Bayes Rule

We can directly relate Bayes Rule to the Noisy Channel:

$$\underbrace{\text{Posterior}}_{\overbrace{Pr(X|Y)}} = \frac{\overbrace{Pr(Y|X)}^{\text{Noisy Channel}} \overbrace{Pr(X)}^{\text{Prior}}}{\underbrace{Pr(Y)}_{\text{Normalization}}}$$

Goal: for a given  $y$ , find  $x =$

$$\arg \max_x \overbrace{Pr(y|x)}^{\text{Noisy Channel}} \overbrace{Pr(x)}^{\text{Prior}}$$

The denominator is ignored because it's the same for all possible corrections, i.e., the observed word ( $y$ ) doesn't change

Introduction

Error causes

Keyboard mistypings

Phonetic errors

Knowledge problems

Challenges

Tokenization

Inflection

Productivity

Non-word error  
detection

Dictionaries

N-gram analysis

Isolated-word error  
correction

Rule-based methods

Similarity key techniques

Probabilistic methods

Minimum edit distance

Error correction for  
web queries

Grammar correction

Syntax and Computing

Grammar correction rules

Caveat emptor



# Finding the Correct Spelling

Goal: for a given misspelling  $y$ , find correct spelling  $x =$

$$\arg \max_x \quad \overbrace{Pr(y|x)}^{\text{Error Model}} \quad \overbrace{Pr(x)}^{\text{Language Model}}$$

1. List “all” possible candidate corrections, i.e., all words with one insertion, deletion, substitution, or transposition
2. Rank them by their probabilities

Example: calculate for *donald*

$$Pr(\text{donadl}|\text{donald})Pr(\text{donald})$$

and see if this value is higher than for any other possible correction.

# Obtaining probabilities

How do we get these probabilities?

We can count up the number of occurrences of  $X$  to get  $P(X)$ , but where do we get  $P(Y|X)$ ?

- ▶ We can use confusion matrices, as we saw before: one matrix each for insertion, deletion, substitution, and transposition
- ▶ These matrices are calculated by counting how often, e.g., *ab* was typed instead of *a* in the case of insertion

To get  $P(Y|X)$ , then, we find the probability of this kind of typo in this context. For insertion, for example ( $X_p$  is the  $p^{\text{th}}$  character of  $X$ ):

$$(2) \quad P(Y|X) = \frac{\text{ins}[X_{p-1}, Y_p]}{\text{count}[X_{p-1}]}$$

- ▶ In order to rank possible spelling corrections, it can be useful to calculate the **minimum edit distance** = minimum number of operations it would take to convert one word into another.
- ▶ For example, we can take the following five steps to convert *junk* to *haiku*:
  1. *junk* → *juk* (deletion)
  2. *juk* → *huk* (substitution)
  3. *huk* → *hku* (transposition)
  4. *hku* → *hiku* (insertion)
  5. *hiku* → *haiku* (insertion)
- ▶ But is this the minimal number of steps needed?

# Computing edit distances

## Figuring out the upper bound

- ▶ To be able to compute the edit distance of two words at all, we need to ensure there is a finite number of steps.
- ▶ This can be accomplished by
  - ▶ requiring that letters cannot be changed back and forth a potentially infinite number of times, i.e., we
  - ▶ limit the number of changes to the size of the material we are presented with, the two words.
- ▶ Idea: Never deal with a character in either word more than once.
- ▶ Result:
  - ▶ We could delete each character in the first word and then insert each character of the second word.
  - ▶ Thus, we will never have a distance greater than  $length(word1) + length(word2)$

### Introduction

#### Error causes

Keyboard mistypings  
Phonetic errors  
Knowledge problems

#### Challenges

Tokenization  
Inflection  
Productivity

#### Non-word error detection

Dictionaries  
N-gram analysis

#### Isolated-word error correction

Rule-based methods  
Similarity key techniques  
Probabilistic methods  
Minimum edit distance

#### Error correction for web queries

#### Grammar correction

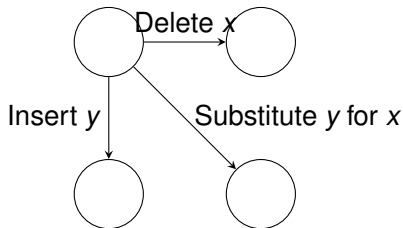
Syntax and Computing  
Grammar correction rules

#### Caveat emptor

# Computing edit distances

Using a graph to map out the options

- ▶ To calculate minimum edit distance, we set up a **directed, acyclic graph**, a set of nodes (circles) and arcs (arrows).
- ▶ Horizontal arcs correspond to deletions, vertical arcs correspond to insertions, and diagonal arcs correspond to substitutions (a letter can be “substituted” for itself).

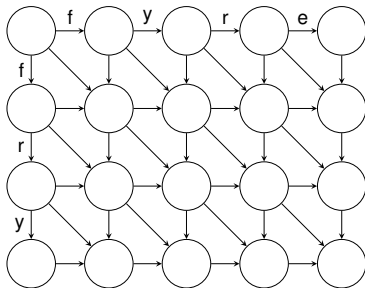


Discussion here based on Roger Mitton's book *English Spelling and the Computer*.

# Computing edit distances

## An example graph

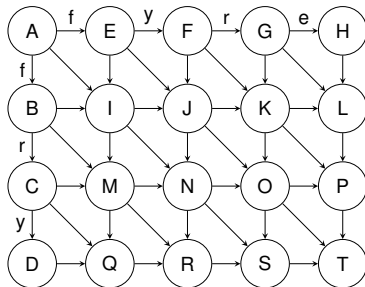
- ▶ Say, the user types in *fyre*.
- ▶ We want to calculate how far away *fry* is (one of the possible corrections). In other words, we want to calculate the minimum edit distance (or minimum edit cost) from *fyre* to *fry*.
- ▶ As the first step, we draw the following directed graph:



# Computing edit distances

## Adding numbers to the example graph

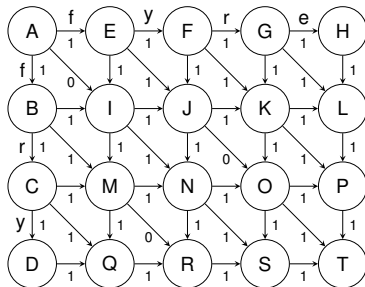
- ▶ The graph is **acyclic** = for any given node, it is impossible to return to that node by following the arcs.
- ▶ We can add identifiers to the states, which allows us to define a **topological order**:



# Computing edit distances

## Adding costs to the arcs of the example graph

- ▶ We need to add the costs involved to the arcs.
- ▶ In the simplest case, the cost of deletion, insertion, and substitution is 1 each (and substitution with the same character is free).



- ▶ Instead of assuming the same cost for all operations, in reality one will use different costs, e.g., for the first character or based on the confusion probability.



# Computing edit distances

How to compute the path with the least cost

We want to find the path from the start (A) to the end (T) with the least cost.

- ▶ The simple but dumb way of doing it:
  - ▶ Follow every path from start (A) to finish (T) and see how many changes we have to make.
  - ▶ But this is very inefficient! There are many different paths to check.

## Introduction

### Error causes

Keyboard mistypings

Phonetic errors

Knowledge problems

### Challenges

Tokenization

Inflection

Productivity

### Non-word error detection

Dictionaries

N-gram analysis

### Isolated-word error correction

Rule-based methods

Similarity key techniques

Probabilistic methods

Minimum edit distance

### Error correction for web queries

### Grammar correction

Syntax and Computing

Grammar correction rules

### Caveat emptor

# Computing edit distances

The smart way to compute the least cost

- ▶ The smart way to compute the least cost uses **dynamic programming** = a program designed to make use of results computed earlier
  - ▶ We follow the topological ordering.
  - ▶ As we go in order, we calculate the least cost for that node:
    - ▶ We add the cost of an arc to the cost of reaching the node this arc originates from.
    - ▶ We take the minimum of the costs calculated for all arcs pointing to a node and store it for that node.
  - ▶ The key point is that we are storing partial results along the way, instead of recalculating everything, every time we compute a new path.

## Introduction

### Error causes

Keyboard mistypings

Phonetic errors

Knowledge problems

### Challenges

Tokenization

Inflection

Productivity

### Non-word error detection

Dictionaries

N-gram analysis

### Isolated-word error correction

Rule-based methods

Similarity key techniques

Probabilistic methods

Minimum edit distance

### Error correction for web queries

### Grammar correction

Syntax and Computing

Grammar correction rules

### Caveat emptor

# Spelling correction for web queries

A nice little side topic ...

Spelling correction for web queries is hard because it must handle:

- ▶ Proper names, new terms, etc. (*blog, shrek, nsync*)
- ▶ Frequent and severe spelling errors
- ▶ Very short contexts

## Introduction

### Error causes

Keyboard mistypings

Phonetic errors

Knowledge problems

### Challenges

Tokenization

Inflection

Productivity

### Non-word error detection

Dictionaries

N-gram analysis

### Isolated-word error correction

Rule-based methods

Similarity key techniques

Probabilistic methods

Minimum edit distance

### Error correction for web queries

### Grammar correction

Syntax and Computing

Grammar correction rules

### Caveat emptor

## Main Idea (Cucerzan and Brill (EMNLP-04))

- ▶ Iteratively transform the query into more likely queries
- ▶ Use query logs to determine likelihood
  - ▶ Despite the fact that many of these are misspelled!
  - ▶ Assumptions: the less wrong a misspelling is, the more frequent it is; and correct > incorrect

## Example:

*anol swartegger*  
→ *arnold schwartnegger*  
→ *arnold schwarznegger*  
→ *arnold schwarzenegger*

# Algorithm (2)

- ▶ Compute the set of all *close* alternatives for each word in the query
  - ▶ Look at word unigrams and bigrams from the logs; this handles concatenation and splitting of words
  - ▶ Use weighted edit distance to determine closeness
- ▶ Search sequence of alternatives for best alternative string, using a noisy channel model

## Constraint:

- ▶ No two adjacent in-vocabulary words can change simultaneously

# The formal algorithm

(just for fun)

Given a string  $s_0$ , find a sequence  $s_1, s_2, \dots, s_n$  such that:

- ▶  $s_n = s_{n-1}$  (stopping criterion)
- ▶  $\forall i \in 0 \dots n-1$ ,
  - ▶  $\text{dist}(s_i, s_{i+1}) \leq \delta$  (only a minimal change)
  - ▶  $P(s_{i+1}|s_i) = \max_t P(t|s_i)$  (the best change)

## Introduction

## Error causes

Keyboard mistypings

Phonetic errors

Knowledge problems

## Challenges

Tokenization

Inflection

Productivity

## Non-word error detection

Dictionaries

N-gram analysis

## Isolated-word error correction

Rule-based methods

Similarity key techniques

Probabilistic methods

Minimum edit distance

## Error correction for web queries

## Grammar correction

Syntax and Computing

Grammar correction rules

## Caveat emptor

## Context Sensitivity

- ▶ *power crd* → *power cord*
- ▶ *video crd* → *video card*
- ▶ *platnuin rings* → *platinum rings*

## Known Words

- ▶ *golf war* → *gulf war*
- ▶ *sap opera* → *soap opera*

# Examples (2)

## Introduction

### Error causes

Keyboard mistypings

Phonetic errors

Knowledge problems

### Challenges

Tokenization

Inflection

Productivity

### Non-word error detection

Dictionaries

N-gram analysis

### Isolated-word error correction

Rule-based methods

Similarity key techniques

Probabilistic methods

Minimum edit distance

### Error correction for web queries

### Grammar correction

Syntax and Computing

Grammar correction rules

### Caveat emptor

## Tokenization

- ▶ *chat inspanich* → *chat in spanish*
- ▶ *ditroitigers* → *detroit tigers*
- ▶ *britenetspear inconcert* → *britney spears in concert*

## Constraints

- ▶ *log wood* → *log wood* (not *dog food*)



**Context-dependent word correction** = correcting words based on the surrounding context.

- ▶ This will handle errors which are real words, just not the right one or not in the right form.
- ▶ This is very similar to a **grammar checker** = a mechanism which tells a user if their grammar is wrong.

# Grammar correction—what does it correct?

- ▶ Syntactic errors = errors in how words are put together in a sentence: the order or form of words is incorrect, i.e., ungrammatical.
- ▶ **Local** syntactic errors: 1-2 words away
  - ▶ e.g., *The study was conducted mainly **be** John Black.*
  - ▶ A verb is where a preposition should be.
- ▶ **Long-distance** syntactic errors: (roughly) 3 or more words away
  - ▶ e.g., *The **kids** who are most upset by the little totem **is** going home early.*
  - ▶ Agreement error between subject *kids* and verb *is*

# More on grammar correction

- ▶ Semantic errors = errors where the sentence structure sounds okay, but it doesn't really mean anything.
  - ▶ e.g., *They are leaving in about fifteen **minuets** to go to her house.*

⇒ *minuets* and *minutes* are both plural nouns, but only one makes sense here

There are many different ways in which grammar correctors work, two of which we'll focus on:

- ▶ *N*-gram model
- ▶ Rule-based model

# N-gram grammar correctors

We can look at **bigrams** of words, i.e., two words appearing next to each other.

- ▶ **Question:** Given the previous word, what is the probability of the current word?
  - ▶ e.g., given *these*, we have a lower chance of seeing *report* than of seeing *reports*
  - ▶ Since a confusable word (*reports*) can be put in the same context, resulting in a higher probability, we flag *report* as a potential error
- ▶ But there's a major problem: we may hardly ever see *these reports*, so we won't know its probability.
- ▶ **Possible Solutions:**
  - ▶ use bigrams of **parts of speech**
  - ▶ use massive amounts of data and only flag errors when you have enough data to back it up

## Introduction

### Error causes

Keyboard mistypings

Phonetic errors

Knowledge problems

### Challenges

Tokenization

Inflection

Productivity

### Non-word error detection

Dictionaries

N-gram analysis

### Isolated-word error correction

Rule-based methods

Similarity key techniques

Probabilistic methods

Minimum edit distance

### Error correction for web queries

### Grammar correction

Syntax and Computing

Grammar correction rules

### Caveat emptor

We can write regular expressions to target specific error patterns. For example:

- ▶ *To a certain extend, we have achieved our goal.*
  - ▶ Match the pattern *some* or *certain* followed by *extend*, which can be done using the regular expression `some|certain extend`
  - ▶ Change the occurrence of *extend* in the pattern to *extent*.

See, e.g., <http://www.languagetool.org/>

# Beyond regular expressions

## Introduction

### Error causes

Keyboard mistypings

Phonetic errors

Knowledge problems

### Challenges

Tokenization

Inflection

Productivity

### Non-word error detection

Dictionaries

N-gram analysis

### Isolated-word error correction

Rule-based methods

Similarity key techniques

Probabilistic methods

Minimum edit distance

### Error correction for web queries

### Grammar correction

Syntax and Computing

Grammar correction rules

### Caveat emptor

- ▶ But what about correcting the following:
  - ▶ *A baseball teams were successful.*
- ▶ We should see that *A* is incorrect, but a simple regular expression doesn't work because we don't know where the word *teams* might show up.
  - ▶ ***A** wildly overpaid, horrendous baseball **teams** were successful.* (Five words later; change needed.)
  - ▶ ***A** player on both my **teams** was successful.* (Five words later; no change needed.)
- ▶ We need to look at how the sentence is constructed in order to build a better rule.

- ▶ **Syntax** = the study of the way that sentences are constructed from smaller units.
- ▶ There cannot be a “dictionary” for sentences since there is an infinite number of possible sentences:

(3) The house is large.

(4) John believes that the house is large.

(5) Mary says that John believes that the house is large.

There are two basic principles of sentence organization:

- ▶ Linear order
- ▶ Hierarchical structure (Constituency)

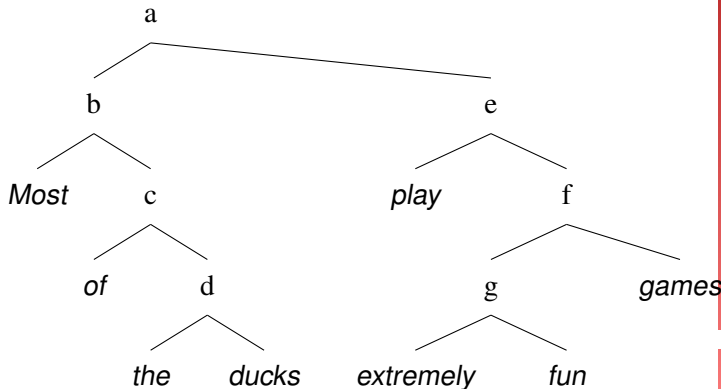
- ▶ **Linear order** = the order of words in a sentence.
- ▶ A sentence can have different meanings, based on its linear order:
  - (6) John loves Mary.
  - (7) Mary loves John.
- ▶ Languages vary as to what extent this is true, but linear order in general is used as a guiding principle for organizing words into meaningful sentences.
- ▶ Simple linear order as such is not sufficient to determine sentence organization though. For example, we can't simply say "The verb is the second word in the sentence."
  - (8) I **eat** at really fancy restaurants.
  - (9) Many executives **eat** at really fancy restaurants.



- ▶ What are the “meaningful units” of a sentence like *Most of the ducks play extremely fun games*?
  - ▶ Most of the ducks
  - ▶ of the ducks
  - ▶ extremely fun
  - ▶ extremely fun games
  - ▶ play extremely fun games
- ▶ We refer to these meaningful groupings as **constituents** of a sentence.

# Hierarchical structure

- ▶ Constituents can appear within other constituents
- ▶ Constituents shown through brackets:  
[[Most [of [the ducks]]] [play [[extremely fun] games]]]
- ▶ Constituents displayed as a **syntactic tree**:



Introduction

Error causes

Keyboard mistypings

Phonetic errors

Knowledge problems

Challenges

Tokenization

Inflection

Productivity

Non-word error  
detection

Dictionaries

N-gram analysis

Isolated-word error  
correction

Rule-based methods

Similarity key techniques

Probabilistic methods

Minimum edit distance

Error correction for  
web queries

Grammar correction

Syntax and Computing

Grammar correction rules

Caveat emptor

- ▶ We would also like some way to say that
  - ▶ *the ducks*, and
  - ▶ *extremely fun games*are the same type of grouping, or constituent, whereas
  - ▶ *of the ducks*seems to be something else.
- ▶ For this, we will talk about different **categories**
  - ▶ Lexical
  - ▶ Phrasal

**Lexical categories** are simply word classes, or what you may have heard as **parts of speech**. The main ones are:

- ▶ verbs: *eat, drink, sleep, ...*
- ▶ nouns: *gas, food, lodging, ...*
- ▶ adjectives: *quick, happy, brown, ...*
- ▶ adverbs: *quickly, happily, well, westward*
- ▶ prepositions: *on, in, at, to, into, of, ...*
- ▶ determiners/articles: *a, an, the, this, these, some, much, ...*

Introduction

Error causes

Keyboard mistypings

Phonetic errors

Knowledge problems

Challenges

Tokenization

Inflection

Productivity

Non-word error  
detection

Dictionaries

N-gram analysis

Isolated-word error  
correction

Rule-based methods

Similarity key techniques

Probabilistic methods

Minimum edit distance

Error correction for  
web queries

Grammar correction

Syntax and Computing

Grammar correction rules

Caveat emptor

How do we determine which category a word belongs to?

- ▶ **Distribution:** Where can these kinds of words appear in a sentence?
  - ▶ e.g., Nouns like *mouse* can appear after articles (“determiners”) like *some*, while a verb like *eat* cannot.
- ▶ **Morphology:** What kinds of word prefixes/suffixes can a word take?
  - ▶ e.g., Verbs like *walk* can take a *ed* ending to mark them as past tense. A noun like *mouse* cannot.

## What about phrasal categories?

- ▶ What other phrases can we put in place of *The joggers* in a sentence such as the following?
  - ▶ The joggers ran through the park.
- ▶ Some options:
  - ▶ Susan
  - ▶ students
  - ▶ you
  - ▶ most dogs
  - ▶ some children
  - ▶ a huge, lovable bear
  - ▶ my friends from Brazil
  - ▶ the people that we interviewed
- ▶ Since all of these contain nouns, we consider these to be *noun phrases*, abbreviated with NP.

Introduction

Error causes

Keyboard mistypings

Phonetic errors

Knowledge problems

Challenges

Tokenization

Inflection

Productivity

Non-word error  
detection

Dictionaries

N-gram analysis

Isolated-word error  
correction

Rule-based methods

Similarity key techniques

Probabilistic methods

Minimum edit distance

Error correction for  
web queries

Grammar correction

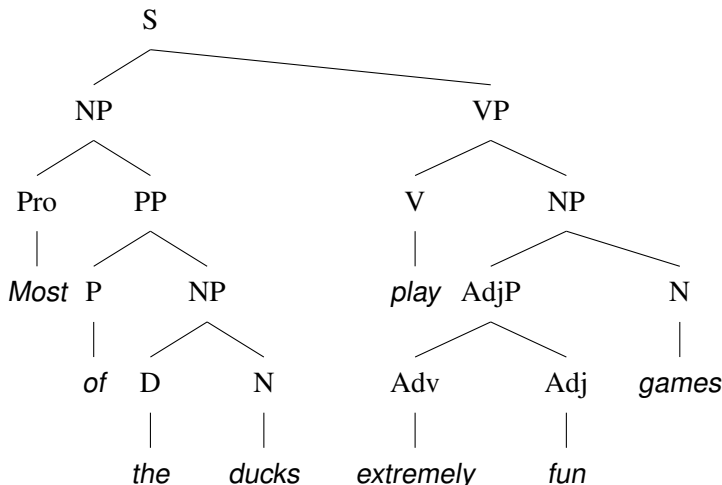
Syntax and Computing

Grammar correction rules

Caveat emptor

# Building a tree

Other phrases work similarly (S = sentence, VP = verb phrase, PP = prepositional phrase, AdjP = adjective phrase):



- ▶ We can give rules for building these phrases. That is, we want a way to say that a determiner and a noun make up a noun phrase, but a verb and an adverb do not.
- ▶ **Phrase structure rules** are a way to build larger constituents from smaller ones.
  - ▶ e.g.,  $S \rightarrow NP VP$   
This says:
    - ▶ A sentence (S) constituent is composed of a noun phrase (NP) constituent and a verb phrase (VP) constituent. [hierarchy]
    - ▶ The NP must precede the VP. [linear order]



# Some other possible English rules

- ▶ NP  $\rightarrow$  Det N (*the cat, a house, this computer*)
- ▶ NP  $\rightarrow$  Det AdjP N (*the happy cat, a really happy house*)
  - ▶ For phrase structure rules, as shorthand parentheses are used to express that a category is optional.
  - ▶ We thus can compactly express the two rules above as one rule:
    - ▶ NP  $\rightarrow$  Det (AdjP) N
    - ▶ Note that this is different and has nothing to do with the use of parentheses in regular expressions.
- ▶ AdjP  $\rightarrow$  (Adv) Adj (*really happy*)
- ▶ VP  $\rightarrow$  V (*laugh, run, eat*)
- ▶ VP  $\rightarrow$  V NP (*love John, hit the wall, eat cake*)
- ▶ VP  $\rightarrow$  V NP NP (*give John the ball*)
- ▶ PP  $\rightarrow$  P NP (*to the store, at John, in a New York minute*)
- ▶ NP  $\rightarrow$  NP PP (*the cat on the stairs*)

# Phrase Structure Rules and Trees

With every phrase structure rule, you can draw a tree for it.

## Lexicon:

Vt  $\rightarrow$  *saw*

Det  $\rightarrow$  *the*

Det  $\rightarrow$  *a*

N  $\rightarrow$  *dragon*

N  $\rightarrow$  *boy*

Adj  $\rightarrow$  *young*

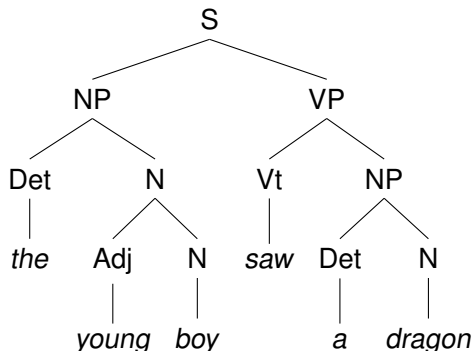
## Syntactic rules:

S  $\rightarrow$  NP VP

VP  $\rightarrow$  Vt NP

NP  $\rightarrow$  Det N

N  $\rightarrow$  Adj N



Introduction

Error causes

Keyboard mistypings

Phonetic errors

Knowledge problems

Challenges

Tokenization

Inflection

Productivity

Non-word error  
detection

Dictionaries

N-gram analysis

Isolated-word error  
correction

Rule-based methods

Similarity key techniques

Probabilistic methods

Minimum edit distance

Error correction for  
web queries

Grammar correction

Syntax and Computing

Grammar correction rules

Caveat emptor

# Properties of Phrase Structure Rules

- ▶ **generative** = a schematic strategy that describes a set of sentences completely.
- ▶ potentially (**structurally**) **ambiguous** = have more than one analysis

(10) We need more intelligent leaders.

(11) Paraphrases:

- a. We need leaders who are more intelligent.
- b. Intelligent leaders? We need more of them!

- ▶ **recursive** = property allowing for a rule to be reapplied (within its hierarchical structure).

e.g., **NP** → NP PP

PP → P **NP**

- ▶ The property of recursion means that the set of potential sentences in a language is **infinite**.

A **context-free grammar** (CFG) is essentially a collection of phrase structure rules.

- ▶ It specifies that each rule must have:
  - ▶ a left-hand side (LHS): a single **non-terminal** element = (phrasal and lexical) categories
  - ▶ a right-hand side (RHS): a mixture of non-terminal and **terminal** elements = actual words
- ▶ A CFG tries to capture a natural language completely.

Why “context-free”? Because these rules make no reference to any context surrounding them. i.e. you can't say “PP → P NP” *when* there is a verb phrase (VP) to the left.

**Pushdown automaton** = the computational implementation of a context-free grammar.

It uses a **stack** (its memory device) and has two operations:

- ▶ **push** = put an element onto the top of a stack.
- ▶ **pop** = take the topmost element from the stack.

This has the property of being **Last In First Out (LIFO)**.

Consider a rule like  $PP \rightarrow P NP$

- ▶ Push NP onto the stack
- ▶ Push P onto it
- ▶ If you find a preposition (e.g., *on*), pop P off of the stack
  - ▶ Now, the next thing you need is an NP ... when you find that, pop NP and push PP onto the stack

Introduction

Error causes

Keyboard mistypings

Phonetic errors

Knowledge problems

Challenges

Tokenization

Inflection

Productivity

Non-word error  
detection

Dictionaries

N-gram analysis

Isolated-word error  
correction

Rule-based methods

Similarity key techniques

Probabilistic methods

Minimum edit distance

Error correction for  
web queries

Grammar correction

Syntax and Computing

Grammar correction rules

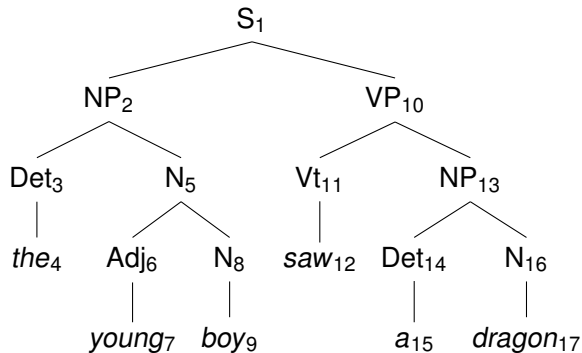
Caveat emptor

Using these context-free rules and something like a pushdown automaton, we can get a computer to **parse** a sentence = assign a structure to a sentence.

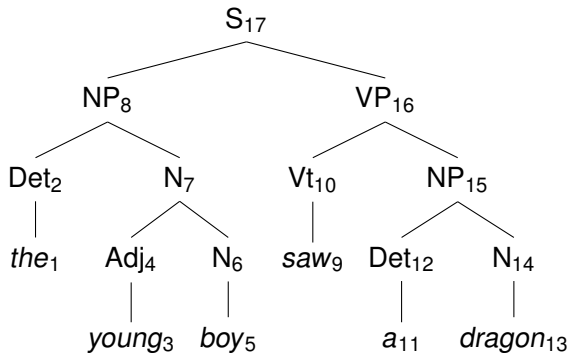
There are many, many parsing techniques out there.

- ▶ **top-down**: build a tree by starting at the top (i.e.  $S \rightarrow NP VP$ ) and working down the tree.
- ▶ **bottom-up**: build a tree by starting with the words at the bottom and working up to the top.

# Trace of a top-down parse



# Trace of a bottom-up parse





# Writing grammar correction rules

So, with context-free grammars, we can now write some correction rules, which we will just sketch here.

- ▶ *A baseball teams were successful.*
  - ▶ A followed by PLURAL NP: change  $A \rightarrow The$
- ▶ *John at the pizza.*
  - ▶ The structure of this sentence is NP PP, but that doesn't make up a whole sentence.
  - ▶ We need a verb somewhere.

# Dangers of spelling and grammar correction

## Introduction

### Error causes

Keyboard mistypings

Phonetic errors

Knowledge problems

### Challenges

Tokenization

Inflection

Productivity

### Non-word error detection

Dictionaries

N-gram analysis

### Isolated-word error correction

Rule-based methods

Similarity key techniques

Probabilistic methods

Minimum edit distance

### Error correction for web queries

### Grammar correction

Syntax and Computing

Grammar correction rules

### Caveat emptor

- ▶ The more we depend on spelling correctors, the less we try to correct things on our own. But spell checkers are not 100%
- ▶ A study at the University of Pittsburgh found that students made **more** errors (in proofreading) when using a spell checker!

	high SAT scores	low SAT scores
use checker	16 errors	17 errors
no checker	5 errors	12.3 errors

(cf., <http://www.wired.com/news/business/0,1367,58058,00.html>)

# A Poem on the Dangers of Spell Checkers

Michael Livingston

*Eye halve a spelling chequer  
It came with my pea sea.  
It plainly marques four my revue  
Miss steaks eye kin knot sea.  
Eye strike a key and type a word  
And weight four it two say  
Weather eye am wrong oar write  
It shows me strait a weigh.  
As soon as a mist ache is maid  
It nose bee fore two long  
And eye can put the error rite  
Its rare lea ever wrong.  
Eye have run this poem threw it  
I am shore your pleased two no  
Its letter perfect awl the weigh  
My chequer tolled me sew.*

Language and  
Computers

Writers' Aids

Introduction

Error causes

Keyboard mistypings

Phonetic errors

Knowledge problems

Challenges

Tokenization

Inflection

Productivity

Non-word error  
detection

Dictionaries

N-gram analysis

Isolated-word error  
correction

Rule-based methods

Similarity key techniques

Probabilistic methods

Minimum edit distance

Error correction for  
web queries

Grammar correction

Syntax and Computing

Grammar correction rules

Caveat emptor

- ▶ The discussion is based on Markus Dickinson (2006). Writer's Aids. In Keith Brown (ed.): *Encyclopedia of Language and Linguistics. Second Edition..* Elsevier.
- ▶ A major inspiration for that article and our discussion is Karen Kukich (1992): *Techniques for Automatically Correcting Words in Text*. ACM Computing Surveys, pages 377–439; as well as Roger Mitton (1996), *English Spelling and the Computer*.
- ▶ For a discussion of the confusion matrix, cf. Mark D. Kernighan, Kenneth W. Church and William A. Gale (1990). A spelling Correction Program Based on a Noisy Channel Model. In *Proceedings of COLING-90*. pp. 205–210.
- ▶ An open-source style/grammar checker is described in Daniel Naber (2003). *A Rule-Based Style and Grammar Checker*. Diploma Thesis, Universität Bielefeld.  
<http://www.danielnaber.de/language-tool/>