Why people care about spelling

- People want to appear to be educated.
- Misspellings can cause misunderstandings and real-life problems:
  - For example:
    - Did you see her god yesterday? It's a big golden retriever.
    - This will be a fee [free] concert.
  - 1991 Bell Atlantic & Pacific Bell telephone network outages were partly caused by a typographical error: A 6 in a line of computer code was supposed to be a D. "That one error caused the equipment and software to fail under an avalanche of computer-generated messages." (Wall Street Journal, Nov. 25, 1991)

Why people care about spelling (cont.)

- Standard spelling makes it easy to organize words and text:
  - e.g., Without standard spelling, how would you look up things in a lexicon or thesaurus?
  - e.g., Optical character recognition software can use knowledge about standard spelling to recognize scanned words even for barely legible input.
- Standard spelling makes it possible to provide a single text, which is accessible to a wide range of readers (different backgrounds, speaking different dialects, etc.).
- Using standard spelling is associated with being well-educated, i.e., is used to make a good impression in social interaction.

Detection vs. Correction

- 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?
- Depends on what we want to do with our results as to what we want to do.
  - Note, though, that detection is a prerequisite for correction.

What causes errors?

- Keyboard mistypings
- Phonetic errors
- Knowledge problems

Who cares about spelling?

Acounting to a rscheearch at Cmbrigde Univeristy, it doesn't mtaa in wtr crrdit the letres in a woord are, the olny ipromoent lting is taht the frist and lsat lette be at the rght pclae. The rset can be a toatl mse and you can stll raud it wouhht porbelm. This is bcsusea the huamn mnid does not raevy iterter by istlet, but the wrold as a whloe.

(See http://www.mrc-cbu.cam.ac.uk/personal/matt.davis/Cmbrigde/ for the story behind this supposed research report.)

A doctcr has aitmited the magiltheansr of a tageene cacrnr pintaet who deid aetf a hatospil durg blendor.

How are spell checkers used?

- Interactive spell checkers = spell checker detects errors as you type.
  - It may or may not make suggestions for correction.
  - Requires 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.
  - If there are a list of choices, we may not require 100% accuracy in the corrected word
- Automatic spell 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.

Keyboard mistypings

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

Space bar issues
Phonetic errors

- homophones = two words which sound the same
  - e.g., red/read (past tense), cite/site/sight, they/their/there
- Spoonerisms = switching two letters/sounds around
  - e.g., Pleating and humming

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 → jogging
    - jake → jokking

What makes spelling correction difficult?

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

Productivity

- part of speech change: nouns can be verbified
  - e.g., Jono cracked his nuckle.
  - instead of John cracked his knuckles.
  - e.g., I study sikoiloege.
- word replacement: replacing one word with some similar-sounding word
  - e.g., John battled me on the back. instead of John patted me on the back.

Physical similarity

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

Tokenization

- Intuitively a "word" is simply whatever is between two spaces (not necessarily on the letters)
  - e.g., Hans was a thing? Now, it's something you do. It got verbed. Verbing weirds language.
  - e.g., pleating and humming
  - e.g., Hans
  - e.g., caveat emptor

More examples for phonetic errors

(1) a. death in Venice
   b. deal in Venice
(2) a. give them an ice bucket
   b. give them a nice bucket
(3) a. the stuffy nose
   b. the stuff he knows
(4) a. the biggest hurdle
   b. the biggest hurdle
(5) a. a Coke danish
   b. a coconut danish

Phoneticians = 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/their/there
- Spoonerisms = switching two letters/sounds around
  - e.g., Pleating and humming

Physical similarity

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

There are exceptions to the rules:

- 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

Physical similarity

- similarity of shape, e.g., mistaking two physically similar letters when typing up something handwritten
  - e.g., tight for fight
## Non-word error detection

- 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?
  - by using a dictionary (construction and lookup)
  - n-gram analysis

## Dictionary construction

- Do we include inflected words? i.e., words with prefixes and suffixes already attached.
  - Pro: lookup can be faster
  - Con: takes much 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 and spelled correctly.
- Foreign words, hyphenations, derived words, proper nouns, and new words will always be problems for dictionaries since we cannot predict these words until humans have made them words.
- Dictionary should probably be dialectically consistent.
  - e.g., include only color or colour but not both

## Dictionary lookup

Several issues arise when trying to look up a word:

- Have to make lookup fast by using efficient lookup techniques, such as a hash table
- Have to strip off prefixes and suffixes if the word isn’t an entry by itself.
  - running → run
  - antireligious → religious

## Dictionaries

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?)

## N-gram analysis

An n-gram is here a string of n letters.

```
  a
  1-gram (unigram)
  at
  2-gram (bigram)
  ate
  3-gram (trigram)
  late
  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.

## How do we store and use n-gram information?

- Store the number of times an n-gram appears (like in Language Identification). But, maybe we just want to know if an n-gram is possible.
- We could have a list of possible and impossible n-grams (1 = possible, 0 = impossible):

<table>
<thead>
<tr>
<th>k</th>
<th>l</th>
<th>m</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1 (tackle)</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1 (Hammer)</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1 (alms)</td>
</tr>
</tbody>
</table>

- Any word which has a 0 for any substring is a misspelled word.
- Problems with such an approach:
  - Information is repeated (po in police)
  - Requires a lot of computer storage space
  - Inefficient (slow) when looking up every string

## Bigram array

- Instead, 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 . .
  . . . . . . .
  . . . 0 0 1 (elk) 1 (hell) 1 (alms)
  . . . 1 0 0 1 (hammer)
```

- 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.)

## Positional bigram array

- 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 .
  . . . . . . .
  . . . 0 0 0 .
  . . . 1 (elk) 1 (half) 1 (elm)
  . . . 0 0 0 .
  . . . . . . .
  . . . 0 0 0 .
```

- Caveat emptor
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.

## Rule-based methods

One can generate correct spellings by writing rules:

- **Common misspelling rewritten as correct word:**
  - e.g., *h*te → *th*
- **Rules**
  - based on inflections:
    - e.g., Cie → Cei
  - based on other common spelling errors (such as keyboard effects or common transpositions):
    - e.g., CaC → C&C
    - e.g., Cie → Cei

## Similarity key techniques

### 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., *punk* starts with the code *P*.
  2. Then assign numbers to each letter:
     - e.g., 0 for vowels, 1, 2, 3, or 5 (all non-bilabials), and so forth, e.g., pun → *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?

#### Types of errors

- **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.

#### General properties

- **single-error misspellings** = only one instance of an error
- **multi-error misspellings** = multiple instances of errors (harder to identify)

### Minimum edit distance

- 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 haku:

1. **junk** → **juk** (deletion)
2. **juk** → **huk** (substitution)
3. **huk** → **hku** (transposition)
4. **hku** → **hku** (insertion)
5. **hku** → **haku** (insertion)

But is this the minimal number of steps needed?
Computing edit distances
Figuring out the worst case

- 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:
  - In the worst case, we delete each character in the first word and then insert each character of the second word.
  - The worst case edit distance for two words is length(word1) + length(word2).

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 (and a letter can be “substituted” for itself).

Omit x

Insert y

Substitute x for y

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:

```
1   2   3   4   5   6   7
p   c   f   g   e   d   b
```

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
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.

Context-dependent word correction

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.
- Essentially a fancier name for 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 minutes to go to her house.
  - ⇒ minutes 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:

- Bigram model (igrams of words)
- Rule-based model

Bigram grammar correctors

We could also look at bigrams: now we are talking about 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 5% chance of seeing reports and a 0.001% chance of seeing report (these report cards).
  - Thus, we will change report to reports

- But there’s a major problem: we may hardly ever see these reports, so we won’t know the probability of that bigram.

  - **(Partial) Solution:** use bigrams of parts of speech.
    - e.g., What is the probability of a noun given that the previous word was an adjective?

Beyond regular expressions

- But what about correcting the following:
  - A baseball teams were successful.
  - We should change A to The, 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. (Four words later; no change needed.)

- We need to look at how the sentence is constructed in order to build a better rule.

Syntax

- **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:
  - (6) The house is large.
  - (7) John believes that the house is large.

- Mary says that John believes that the house is large.

- There are two basic principles of sentence organization:
  - Linear order
  - Hierarchical structure (Constituency)

Rule-based grammar correctors

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 certain followed by extend, which can be done using the regular expression some\(\)certain extend
  - Change the occurrence of extend in the pattern to extend

- Naber (2003) uses 56 such rules to build a grammar corrector which works nearly as well as that in commercial products.

Linear order

- **Linear order** = the order of words in a sentence.

- A sentence has different meanings based on its linear order.
  - (9) John loves Mary.
  - (10) 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.”
  - (11) I eat at really fancy restaurants.
  - (12) Many executives eat at really fancy restaurants.

Constituency

- **Constituency** = the study of the way that sentences are organized into meaningful smaller groups.

- What are the “meaningful units” of a sentence like Many executives eat at really fancy restaurants?
  - Many executives
  - really fancy
  - really fancy restaurants
  - eat at really fancy restaurants

- We refer to these meaningful groupings as constituents of a sentence.

- There are many “tests” to determine what a constituent is, but we will not concern ourselves with them here.

Hierarchical structure

- Constituents can appear within other constituents. We can represent this in a bracket form or in a syntactic tree:

```
[Many executives [eat [at [really fancy restaurants]]]]
```

- Constituents displayed as a tree:

```
Many executives eat at really fancy restaurants
```

Categories

- We would also like some way to say that Many executives and really fancy restaurants are the same type of grouping, or constituent, whereas at really fancy restaurants seems to be something else.

- For this, we will talk about different categories
  - Lexical
  - Phrasal
**Lexical categories**

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, these, some, much, ...

**Phrasal categories**

What about phrases? Can we assign them categories? We can also look at their distribution and see which ones behave in the same way.

- The joggers ran through the park.
- What other phrases can we put in place of The joggers?

**Determining lexical categories**

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 the, 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.

**Phrase Structure Rules**

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.

- **NP → Det N (the cat, a house, this computer)**
- **NP → 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 → Det (AdJP) N**
  - Note that this is different and has nothing to do with the use of parentheses in regular expressions.
  - **AdJP → (Adv) Adj (really happy)**

**Some other English rules**

- **VP → V (laugh, run, eat)**
- **VP → V NP (love John, hit the wall, eat cake)**
- **VP → V NP NP (give John the ball)**
- **PP → P NP (to the store, at John, in a New York minute)**
- **NP → NP PP (the cat on the stairs)**

**Closed & Open classes**

We can add words to some classes, but not to others. This also seems to correlate with whether a word is "meaningful" or just a function word = only meaning comes from its usage in a sentence.

**Open classes**: new words can be easily added:

- verbs
- nouns
- adjectives
- adverbs

**Closed classes**: new words cannot be easily added:

- prepositions
- determiners

**Building a tree**

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

![Tree Diagram]

**Phrasal categories (cont.)**

- SusanStudents you
- some 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.

**Phrase Structure Rules and Trees**

With every phrase structure rule, you can draw a tree for it.
Try analyzing these sentences and drawing trees for them, based on the phrase structure rules given above.

- The man in the kitchen drives a truck.
- That dang cat squeezed some fresh orange juice.
- The mouse in the corner by the stairs ate the cheese.

Properties of Phrase Structure Rules

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

(13) We need more intelligent leaders.
(14) Paragraphs:
   a. We need leaders who are more intelligent.
   b. Intelligent leaders? We need more of them!

- **hierarchical** = categories have internal structure; they aren't just linearly ordered.
- **recursive** = property allowing for a rule to be reapplied (within its hierarchical structure).

i.e., \( \text{NP} \rightarrow \text{NP PP} \rightarrow \text{P NP} \)

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

### Parsing

So, using these phrase structure (context-free) rules, **parse** a sentence as **assign a structure to a sentence**.

Do you parse top-down or bottom-up (or a mixture)?
- **top-down**: build a tree by starting at the top (i.e., \( \text{S} \rightarrow \text{NP VP} \) and working down the tree.
- **bottom-up**: build a tree by starting with the words at the bottom and working up to the top.

There are many, many parsing techniques out there.

### 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 → The
- **John at the taco.**

The structure of this sentence is NP PP, but that doesn't make up a whole sentence. We need a verb somewhere.

### Context-free grammars

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

**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 \( \rightarrow \text{P NP} \)" when there is a verb phrase (VP) to the left.

### Dangers of spelling and grammar correction

- 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** when using a spell checker!

<table>
<thead>
<tr>
<th>use checker</th>
<th>high SAT scores</th>
<th>low SAT scores</th>
</tr>
</thead>
<tbody>
<tr>
<td>no checker</td>
<td>16 errors</td>
<td>17 errors</td>
</tr>
<tr>
<td></td>
<td>5 errors</td>
<td>12.3 errors</td>
</tr>
</tbody>
</table>

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

### A Poem on the Dangers of Spell Checkers

**Michael Livingston**
(http://www.courses.rochester.edu/livingston/guide/phorix.html)

Eye have a spelling chequer
It came with my pea sea.
It plainly marques four my reve
Miss steaks eye kin knot sea.
Eye strike a key and type a word
And weight four it two say
Weather eye am wrong car write
It shows me strait a weigh
As soon as a mist ached is mail
It nose bee fore two long
And eye can put the error rite
Its rare lea ever long.
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.

### References