NATURAL LANGUAGE PROCESSING

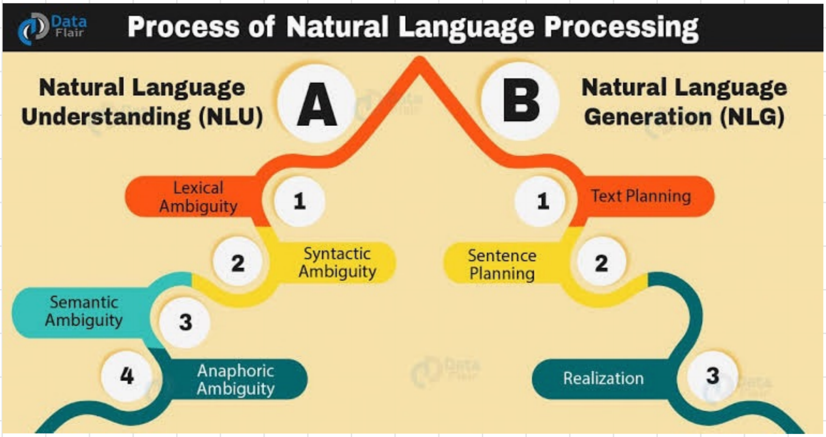
UNIT-1

Introduction

feedback/corrections: vibha@pesu.pes.edu

VIBHA MASTI

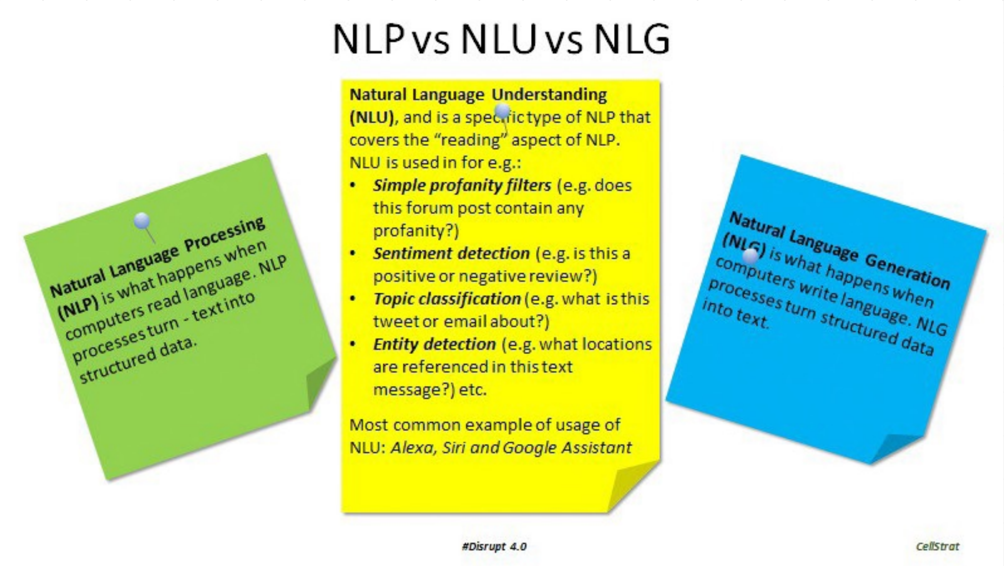
PROCESS OF NLP



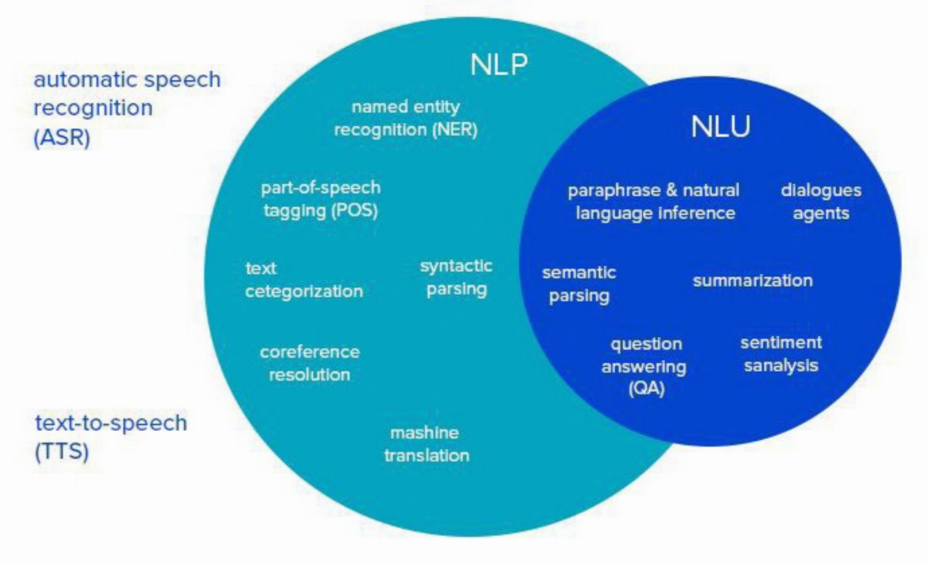
NLP, NLU, NLG

- NLP: CS, AI, computational linguistics
- Turing test: if a machine behaves in a manner that is indistinguishable from a human — artificially intelligent
- See here for more
- Natural language understanding: mapping natural language input into useful representations and analysing the aspects of the language
  - comprehension, sentiment analysis, discover meaning
  - semantic and syntactic analysis
- Natural language generation: producing meaningful phrases and sentences from some internal representation
  - Textual data: Q&A pair generation from a given section
  - Numerical data: create earning summary from earning calendar
  - Pictures: image captioning
  - Diagrams: answer generation using applicable ontology

- More [here]



NLP vs NLU vs NLG
Natural Language Processing (NLP) is what happens when computers read language. NLP processes turn - text into structured data.
Natural Language Understanding (NLU), and is a specific type of NLP that covers the "reading" aspect of NLP.
NLU is used in for e.g.:
Simple profanity filters (e.g. does this forum post contain any profanity?)
Sentiment detection (e.g. is this a positive or negative review?)
Topic classification (e.g. what is this tweet or email about?)
Entity detection (e.g. what locations are referenced in this text message?) etc.
Most common example of usage of NLU: Alexa, Siri and Google Assistant
Natural Language Generation (NLG) is what happens when computers write language. NLG processes turn structured data into text.
#Disrupt 4.0
CellStrat



automatic speech recognition (ASR)
text-to-speech (TTS)
NLP
named entity recognition (NER)
part-of-speech tagging (POS)
text cetegorization
syntactic parsing
coreference resolution
mashine translation
NLU
paraphrase & natural language inference
dialogues agents
semantic parsing
summarization
question answering (QA)
sentiment sanalysis

Challenges - Ambiguity

- Word sense ambiguity

- Pronoun referencing ambiguity

Challenges - Scale

- Bible
- Wikipedia
- WWW

non-standard English

Great job @justinbieber! Were SOO PROUD of what youve accomplished! U taught us 2 #neversaynever & you yourself should never give up either ❤️

segmentation issues

the New York-New Haven Railroad
the New York-New Haven Railroad

idioms

dark horse
get cold feet
lose face
throw in the towel

neologisms

unfriend
Retweet
bromance

world knowledge

Mary and Sue are sisters.
Mary and Sue are mothers.

tricky entity names

Where is *A Bug's Life* playing ...
Let It Be was recorded ...
... a mutation on the *for* gene ...

Probabilistic Model

- Models built from language data
- P("maison" → "house") high
  P("L'avocat général" → "the general avocado") low
- Requires knowledge about language, world
- Need to extract features

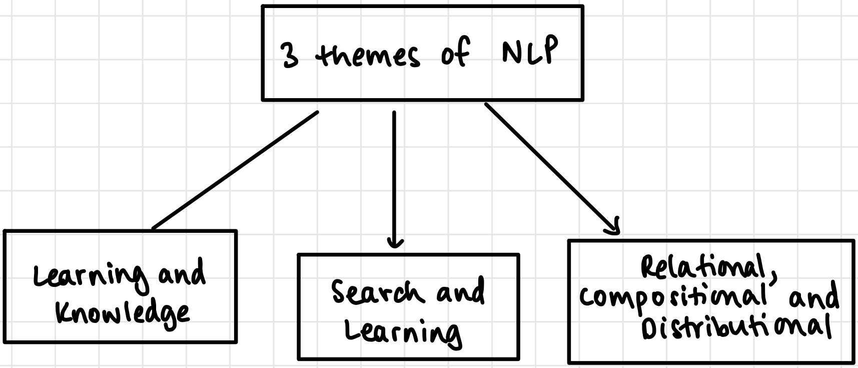
- IBM Watson API
- Chatbot API
- Speech to text API
- Sentiment Analysis API
- Translation API by SYSTRAN
- Text Analysis API by AYLIEN
- Cloud NLP API
- Google Cloud Natural Language API
- MonkeyLearn

Natural Language Processing
ACL, NAACL, EACL, EMNLP, CoNLL, Coling, TACL
aclweb.org/anthology

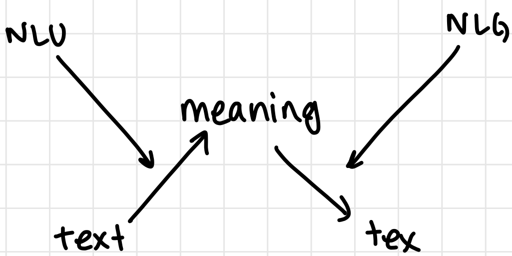
Machine learning
ICML, NIPS, ECML, AISTATS, ICLR, JMLR, MLJ

Artificial Intelligence
AAAI, IJCAI, UAI, JAIR

3 Themes in NLP

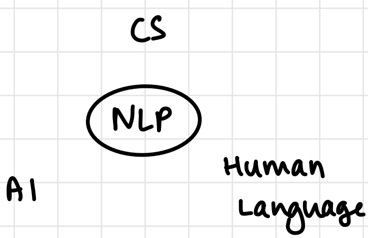


Chatbot NLP System



UNIX WC Command - word count command - data processing command

NLP & its neighboure



Learning & Knowledge

- ML & linguistic knowledge are important to each other
- Identifying stem words — comination, combined, combines are all termed combine
- NLP: combination of Learning & knowledge
- Hierarchical attention networks paper

Search & Learning

- Many NLP problems are optimisation problems

$$\hat{y} = \underset{y \in Y(x)}{\operatorname{argmax}} f(x, y; \theta)$$

parameters

- Search module: search solution space for optimal solution $\hat{y}$ wrt x (combinatorial optimisation as NLP tasks are usually discrete)
- Learning module: learn parameters θ (numerical optimisation as parameters are continuous)
- Expressive model: when model is capable of making subtle linguistic distinctions
- Expressiveness is often traded off against efficiency of search and learning
  - eg: word to word translations make search & learning easy but are not expressive enough to distinguish good translations from bad ones

- Most NLP systems are not expressive in nature
- Complexity of search becomes exponential

Relational, Compositional and Distributional Perspectives

- Any element of language (word, phrase, sentence, sound) can be described from 3 perspectives

1. Relational Perspective:
- consider the word journalist
- journalist is a subcategory of a profession
- anchorwoman is a subcategory of journalist
- journalist performs journalism
- journalism subcategory of writing

- Relational perspective on meaning: basis for semantic ontologies (eg: WORDNET)

2. Compositional Perspective
- Words made of constituent parts
- journalist: journal + ist

- strength: analyse text without training (can address the long tail)

3. Distributional Perspective
- Words are replaceable by other phrases
- eg: idioms
- relational & composational models fail here
- meaning constructed from context - distributional properties

- All three critical to NLP but require seemingly incompatible approaches & representations

TYPES of AMBIGUITY

1. Lexical
   - Ambiguity of a single word (multiple meanings for same word)
   - Eg: She is looking for a match.
     - partner
     - matchstick

2. Syntactic
   - Sentence can be parsed in multiple ways (multiple meanings for a single sentence)
   - Eg: The chicken is ready to eat.
     - is the chicken eating?
     - is the chicken to be eaten?

3. Semantic
   - Meanings of words misinterpreted
   - Eg: The car hit the pole while it was moving.
     - Pole moving
     - Car moving

4. Anaphoric
   - Referential ambiguity (using pronouns)
   - Eg: The boy told his father of the theft. He was very upset.
     - boy
     - father

5. Pragmatic
   - Context of a phrase gives it multiple interpretations
   - Statement is not specific
   - Eg: I like you too.
     - like you do
     - like someone else does

6. Metonymy
   - Phrases in which literal meaning is diff from figurative assertion

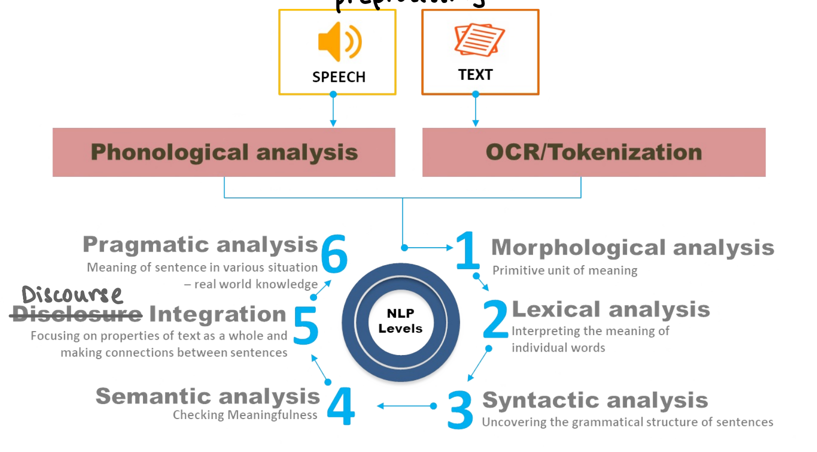
Steps in NLP

0. Phonetics and phonology / tokenisation
1. Morphological analysis
2. Lexical analysis
3. Syntactic analysis
4. Semantic analysis
5. Discourse integration
6. Pragmatic analysis

high ambiguity

low ambiguity

preprocessing



0. Phonetics & Phonology

- Phonetics: branch of linguistics that studies the sounds of human speech
  - Phoneme: smallest sound unit in a language that is capable of conveying a distinct meaning (s of sing, r of ring)

- Homophones: same sound, different meanings, different spellings
  - alter/altar, sell/cell, bore/boar, lone/loan

- **Homonym/ Homograph:** same sound, different meanings, different spellings
  - bank (money / river)

- **Heteronym:** different sound, different meanings, same spelling
  - minute (small, 60 seconds), read (past, present tense), wind (twist, movement of air)

- **Polysemy:** coexistence of many possible meanings for a single word or phrase
  - He is a good man. morals
    He is a good basketball player. skill

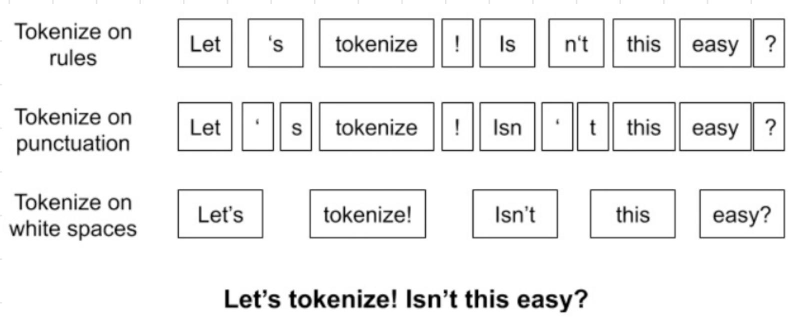
Table 1. The target polysemous words and their meanings

The polysemous words	Meanings
Open	Meaning 1: 'spread out'
	Meaning 2: 'not covered'
	Meaning 3: 'honest'
	Meaning 4: 'not hidden'
	Meaning 5: 'available'
Run	Meaning 1: 'move fast'
	Meaning 2: 'manage'
	Meaning 3: 'provide'
	Meaning 4: 'use'
	Meaning 5: 'flow'
Make	Meaning 1: 'prepare'
	Meaning 2: 'force'
	Meaning 3: 'appoint'
	Meaning 4: 'reach'
	Meaning 5: 'represent'

- **Phonology:** study of sound structures in language

0. Tokenisation

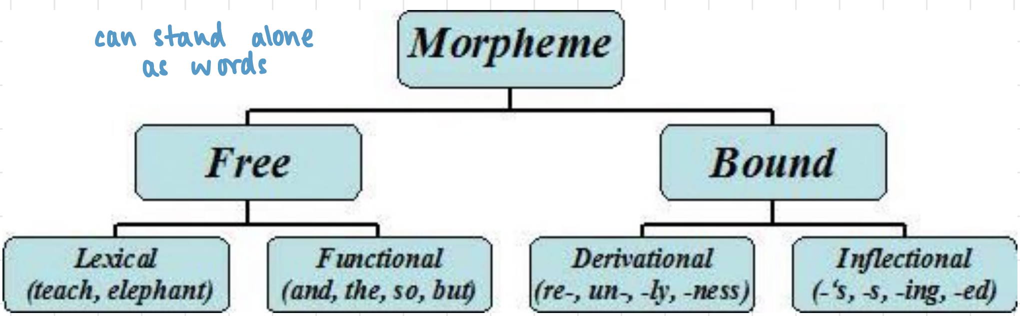
- separating out words from running text
- Throw away certain characters



1. Morphological anaysis

- Morpheme: smallest unit of meaning
- Understanding of diff words according to their morphemes
- Study of structure and formation of words

can stand alone as words

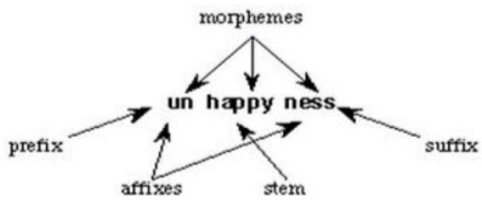


noun, verb

prefixes suffixes

suffixes

Consider a word like: "unhappiness". This has three parts:



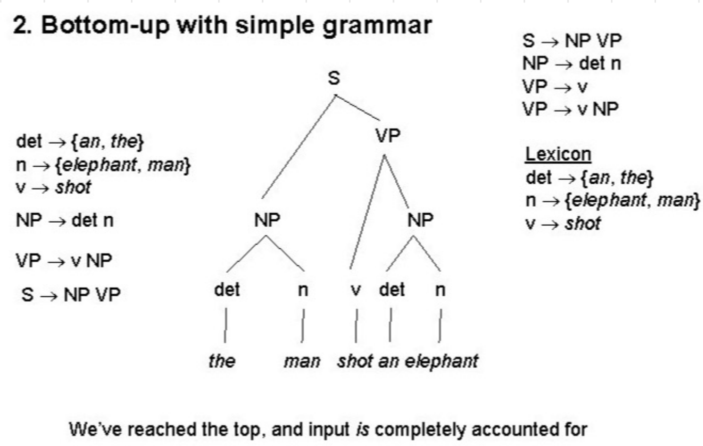
There are three morphemes, each carrying a certain amount of meaning. *un* means "not", while *ness* means "being in a state or condition". *Happy* is a *free morpheme* because it can appear on its own (as a "word" in its own right).

2. Lexical analysis

- Identifying and analysing structure of words
- Lexicon: collection of words & phrases in a language
- Divide chunk of text into paragraphs, sentences and words
- Obtaining properties of a word
  - Eg: dog ⇒ image of dog & its properties ⇒ 4 leg, carnivore, animate
- Stemming: rudimentary rule-based process of stripping the suffixes (ing, ly, es, s etc) from word
- Lemmatisation: organised, step by step procedure to find root form of word using vocabulary and morphological analysis
  - Eg: go, going, went → go
  - Better than stemming

3. Syntactic Analysis (Parsing)

- Analysis of words in a sentence for grammar
- Eg: The school goes to boy. → rejected by English syntactic analyser

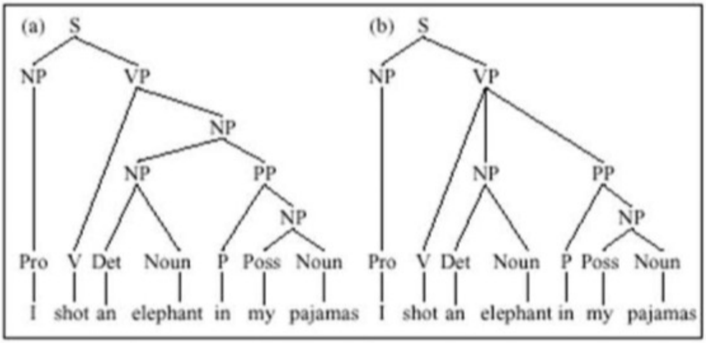


Sentence constructed from Noun Phrase (NP) and Verb Phrase (VP)

Noun Phrase: article (art) and noun (n)

Verb Phrase: verb (v) and noun phrase (NP)

- Ambiguity



4. Semantic analysis

- Draws exact meaning from text, checks for meaningfulness
- Disregards "I am eating hot ice cream"
- Finding synonyms, word sense disambiguation, constructing Q&A systems, translating from one NL to another
- Must first do morphological & syntactic analysis before semantic analysis
- Semantic & pragmatic analysis make up the most complex phase of NLP

5. Discource integration

- Sense of context
- Meaning of a single sentence that depends on surrounding sentences.
- Eg: Ram saw a hat. He wanted to buy it.
- Anaphora: use of a word referring back to a word used earlier in text
- Active & passive voice: My house was broken into last week. They took my TV, knowledge bases

  burglars

6. Pragmatic analysis

- Extra meaning read into text without actually being encoded
- World knowledge, intentions, plans, goals
- Eg:
  1. The city police refused the demonstrators a permit because they feared violence.
  2. The city police refused the demonstrators a permit because they advocated revolution.

  - In 1, they refers to police
  - In 2, they refers to the demonstrators
- World knowledge in knowledge bases and inference modules to be utilised
- Interpretation of ambiguity, intent

TEXT NORMALISATION

- Segmenting / tokenising words
- Normalisation of word formats (case conversion)
- Results in smaller vocabulary and smaller feature vectors
- Standardisation of numbers (1000 / 1,000) and dates

Parts of Speech

POS

Content words	Function words
words with meaning	words without meaning
stored in dictionary	explain structural relationships, create grammar
open class (words added & removed)	generally a closed class
+ covid-19	eg: to, of, the
- aerodrome	prepositions, pronouns, conjunctions, determiners, qualifiers, intensifiers, interrogatives
nouns, verbs, adjectives, adverbs	

Types vs Tokens

1 2 3 4 5 6 7 8 9 10 11 12

They picnicked by the pool, then lay back on the grass and looked at the stars.

13 14 15 16 17 18

- 18 tokens (including punctuation) — space delimiter - instance of type in running text
- 16 types (unique words) - element of vocabulary

Tokenization

- Breaking up sequence of characters in text by locating word boundaries.

- In written languages (Chinese, Japanese, Turkish etc.) - no explicit word boundaries in writing system
  - Word segmentation

- Sentence segmentation also a part of preprocessing
  - sentence boundaries

- In general, binary classifier to decide if a period (.) marks the end of a sentence or is a part of a word
  - Abbreviation dictionary helpful

- SOTA methods for sentence tokenisation based on ML

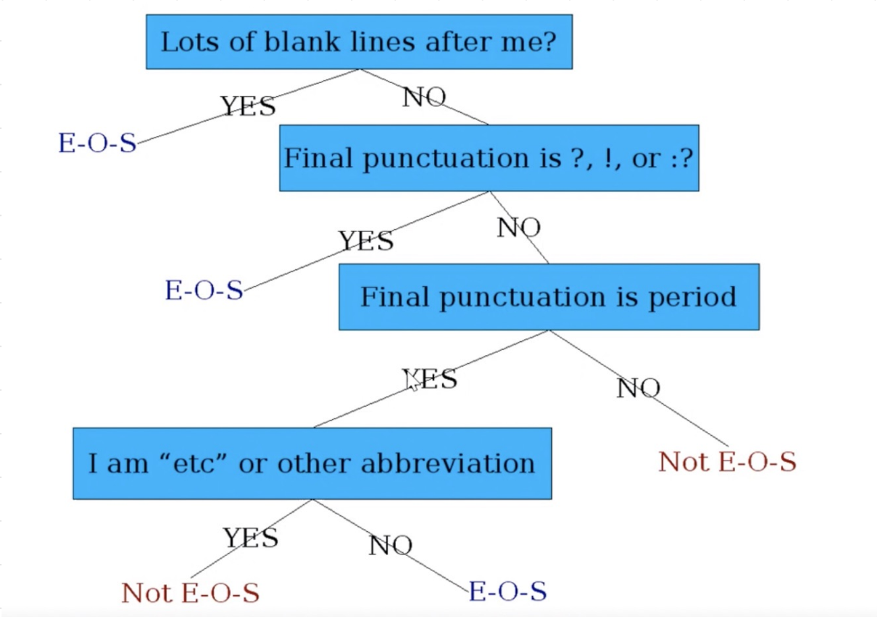
- Accuracy of tokenisation affects results of higher level processing

- Problems / ambiguity
  - United States, AT&T, 3-year-old
  - Prof. Dr. J. M.
  - 123,456.78

- !, ? - not as ambigious

- . - quite ambiguous
  - sentence boundary
  - Dr., Inc. (abbreviations)
  - .2%, 0.234 (numbers)

- Eg classifier: decision tree



Text Normalization

- Elimination of inflectional affixes (-ed, -s suffixes etc.)
- Stemmer: eliminate affixes using series of regex substitutions
- Character-based stemming algorithms: necessarily approximate

Original	The	Williams	sisters	are	leaving	this	tennis	centre
Porter stemmer	the	william	sister	are	leav	thi	tenni	centr
Lancaster stemmer	the	william	sist	ar	leav	thi	ten	cent
WordNet lemmatizer	The	Williams	**sister**	are	leaving	this	tennis	centre

↑ performs better than stemmers

- Snowball stemmer: built on SNOBOL and Porter (same creator as Porter) – supports many languages – also called Porter 2

- Lemmatiser: identify underlying lemma of wordform
  - avoid over-generalisation errors of stemmers

- Both stemming & lemmatisation are language specific

- Stemming and lemmatisation used in
  - tagging systems
  - indexing
  - search engine optimisation (SEO)
  - web search results
  - information retrieval

- Lemma: dictionary form of words (headword)
  - run, runs, ran, running— forms of same lexeme with run as the lemma

- Lemmatisation takes into consideration of morphological analysis of the words
  - look up dictionary

- Lemmatisers are more complex as sometimes stemmers are preferred

- Stem: root + derivational morphemes / affixes
  - root: run, bat, chat
  - morphologically complex (eg: compound words): bottle opener
  - words with derivational morphemes: blacken, standardize, unkind

- Crude chopping of suffixes ; not always acceptable words

N-grams

- n-gram: contiguous sequence of n items from text | speech
- Items: letters, words, base pairs, phonemes, syllables etc
- Eg: She was laughing at him.
  1-grams: She, was, laughing etc
  2-grams: She was, was laughing etc
  3-grams: She was laughing, was laughing at etc.
- n-gram model: type of probabilistic language model for predicting the next item in the sequence

PORTER STEMMER

- Stripping of suffixes – varies depending on
  - whether stem dictionary used
  - whether suffix list with various rules used
  - https://tartarus.org/martin/PorterStemmer/
- Using stem dictionary: difficult, time-consuming
- Instead, explicit set of suffixes with removal rules
- Simple, rule-based, suffix stripping algorithm (heuristic method)
- Five sets of rules applied in order; practical method and not guaranteed to be optimal
- Paper in 1980

Porter Stemmer Definitions

- Consonants: a letter other than A, E, I, O and U and the letter Y unless it is preceded by a vowel
  - consonant: Y in TOY
  - vowel: Y in RHYTHM

- Vowel: a letter other than a consonants

Porter Stemmer

- Consonant denoted by c, vowel denoted by v
- A list of consonants of length > 0 is denoted as C
- A list of vowels of length > 0 is denoted as V
- A word has one of four forms
  1. CVCV...C
  2. CVCV...V
  3. VCVC...C
  4. VCVC...V

optimal

- Generically, [C]VCVC...[V]
- Using (VC){m} to denote VC denoted m times,

[C](VC){m}[V]

- ∴ all words are (m denotes measure of word or word part)

[C](VC){m}[V]

- Eg: TROUBLES

$$\begin{matrix} TR & OU & BL & E & S \\ \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\ C & V & C & V & C \end{matrix}$$

C (VC) {2}

- m is measure of word or word part

1. m=0 [C][V]
   - eg: TREE, BY
2. m=1 [C] VC [V]
   - eg: TINY, OATS, TROUBLE, IVY
3. m=2 [C] VCVC [V]
   - eg: TROUBLES, PRIVATE, EATEN

Rules for Suffix Removal

- Form: (condition) S1 → S2
  - meaning: if a word ends with suffix S1 and the stem before S1 satisfies the condition, S1 replaced with S2

- Condition usually given in terms of m

- Eg: (m > 1) EMENT →
  - S1 = EMENT
  - S2 = null

  - REPLACEMENT → REPLAC
    $C(VC)^2$

  - ELEMENT
    $(VC)^1$ → does not satisfy

- MEASUREMENT → MEASUR

C (VC)^2 C

Porter Stemmer Rule Format

1. m ——— measure of stem
2. *S ——— stem ends with S (and for other letters)
3. *v* ——— stem contains a vowel
4. *d ——— stem ends with double consonant (eg: -TT, -SS)
5. *o ——— stem ends in cvc where second c not W, X, or Y (eg: -WIL, -HOP)

Conditions may also contain logical expressions (and, or, not)

(m>1 and (*S or *T))

tests for a stem with m>1 ending in S or T, while

(*d and not (*L or *S or *Z))

tests for a stem ending with a double consonant other than L, S or Z. Elaborate conditions like this are required only very rarely.

Step 1 - plurals & past participles

Step 1a

plurals

SSES	→ SS	caresses	→ caress
IES	→ I	ponies	→ poni
		ties	→ ti
SS	→ SS	caress	→ caress
S	→	cats	→ cat

Step 1b

past tense, progressive

(m>0)	EED → EE	feed	→ feed
		agreed	→ agree
(*v*)	ED →	plastered	→ plaster
		bled	→ bled
(*v*)	ING →	motoring	→ motor
		sing	→ sing

· If 2^{nd} or 3^{rd} rules in 1b successful, following is done

cleaning up

AT → ATE		conflat(ed)	→ conflate
BL → BLE		troubl(ing)	→ trouble
IZ → IZE		siz(ed)	→ size
(*d and not (*L or *S or *Z)) → single letter		hopp(ing)	→ hop
		tann(ed)	→ tan
		fall(ing)	→ fall
		hiss(ing)	→ hiss
		fizz(ed)	→ fizz
(m=1 and *o) → E		fail(ing)	→ fail
		fil(ing)	→ file

The rule to map to a single letter causes the removal of one of the double letter pair. The –E is put back on –AT, –BL and –IZ, so that the suffixes –ATE, –BLE and –IZE can be recognised later. This E may be removed in step 4.

replacement of y

Step 1c

(*v*) Y → I	happy	→ happi
	sky	→ sky

Step 2 - Derivational Morphology

Step 2

(m>0)	ATIONAL	→ ATE	relational	→ relate
(m>0)	TIONAL	→ TION	conditional	→ condition
			rational	→ rational
(m>0)	ENCI	→ ENCE	valenci	→ valence
(m>0)	ANCI	→ ANCE	hesitanci	→ hesitance
(m>0)	IZER	→ IZE	digitizer	→ digitize
(m>0)	ABLI	→ ABLE	conformabli	→ conformable
(m>0)	ALLI	→ AL	radicalli	→ radical
(m>0)	ENTLI	→ ENT	differentli	→ different
(m>0)	ELI	→ E	vileli	→ vile
(m>0)	OUSLI	→ OUS	analogousli	→ analogous
(m>0)	IZATION	→ IZE	vietnamization	→ vietnamize
(m>0)	ATION	→ ATE	predication	→ predicate
(m>0)	ATOR	→ ATE	operator	→ operate
(m>0)	ALISM	→ AL	feudalism	→ feudal
(m>0)	IVENESS	→ IVE	decisiveness	→ decisive
(m>0)	FULNESS	→ FUL	hopefulness	→ hopeful
(m>0)	OUSNESS	→ OUS	callousness	→ callous
(m>0)	ALITI	→ AL	formaliti	→ formal
(m>0)	IVITI	→ IVE	sensitiviti	→ sensitive
(m>0)	BILITI	→ BLE	sensibiliti	→ sensible

The test for the string S1 can be made fast by doing a program switch on the penultimate letter of the word being tested. This gives a fairly even breakdown of the possible values of the string S1. It will be seen in fact that the S1-strings in step 2 are presented here in the alphabetical order of their penultimate letter. Similar techniques may be applied in the other steps.

Step 3 - Derivational Morphology II

Step 3

(m>0)	ICATE	→ IC	triplicate	→	triplic
(m>0)	ATIVE	→	formative	→	form
(m>0)	ALIZE	→ AL	formalize	→	formal
(m>0)	ICITI	→ IC	electriciti	→	electric
(m>0)	ICAL	→ IC	electrical	→	electric
(m>0)	FUL	→	hopeful	→	hope
(m>0)	NESS	→	goodness	→	good

Step 4 - Derivational Morphology III

Step 4

(m>1)	AL	→	revival	→	reviv
(m>1)	ANCE	→	allowance	→	allow
(m>1)	ENCE	→	inference	→	infer
(m>1)	ER	→	airliner	→	airlin
(m>1)	IC	→	gyroscopic	→	gyroscop
(m>1)	ABLE	→	adjustable	→	adjust
(m>1)	IBLE	→	defensible	→	defens
(m>1)	ANT	→	irritant	→	irrit
(m>1)	EMENT	→	replacement	→	replac
(m>1)	MENT	→	adjustment	→	adjust
(m>1)	ENT	→	dependent	→	depend
(m>1)	and (*S or *T)) ION	→	adoption	→	adopt
(m>1)	OU	→	homologou	→	homolog
(m>1)	ISM	→	communism	→	commun
(m>1)	ATE	→	activate	→	activ
(m>1)	ITI	→	angulariti	→	angular
(m>1)	OUS	→	homologous	→	homolog
(m>1)	IVE	→	effective	→	effect
(m>1)	IZE	→	bowdlerize	→	bowdler

The suffixes are now removed. All that remains is a little tidying up.

Step 5 - Tidying Up

Step 5a

(m>1) E	→	probate	→	probat
		rate	→	rate
(m=1 and not *o) E	→	cease	→	ceas

Step 5b

(m>1 and *d and *L)	→	single letter		
		controll	→	control
		roll	→	roll

Why it works

The algorithm is careful not to remove a suffix when the stem is too short, the length of the stem being given by its measure, m. There is no linguistic basis for this approach. It was merely observed that m could be used quite effectively to help decide whether or not it was wise to take off a suffix. For example, in the following two lists:

list A	list B
RELATE	DERIVATE
PROBATE	ACTIVATE
CONFLATE	DEMONSTRATE
PIRATE	NECESSITATE
PRELATE	RENOVATE

–ATE is removed from the list B words, but not from the list A words. This means that the pairs DERIVATE/DERIVE, ACTIVATE/ACTIVE, DEMONSTRATE/DEMONSTRABLE, NECESSITATE/NECESSITOUS, will conflate together. The fact that no attempt is made to identify prefixes can make the results look rather inconsistent. Thus PRELATE does not lose the –ATE, but ARCHPRELATE becomes ARCHPREL. In practice this does not matter too much, because the presence of the prefix decreases the probability of an erroneous conflation.

- Better to ignore irregular forms and exceptions instead of making complicated rules

- Porter stemmers can repair fairly well

Input	Strip -ed Affix	Repair
hoped	hop	hope (add -e if word is short)
hopped	hopp	hop (delete one if doubled)

- Online demo: https://textanalysisonline.com/nltk-porter-stemmer

- Online demo gives:
  - *gas* (noun) → *ga*
  - *gases* (plural) → *gase*
  - *gasses* (verb, present tense) → *gass*
  - *gassing* (verb, present continuous) → *gass*
  - *gaseous* (adjective) → *gaseou*

SPELLING CORRECTION

- Word processing, search engines, texting

- Spelling tasks
  - spelling error detection
  - spelling error correction

3 reasons for spelling errors

insertion
extra letter

deletion
missing letter

substitution
wrong letter

Types of spelling errors

1. Non-word errors
   - misspelled word is not a dictionary word
   - eg: giraffe → graffe

2. Real-word errors
   - context has to be learnt

   (a) Typographical errors
   - rearrangement of letters / wrong letters to form dictionary word
   - eg: there → three

   (b) Cognitive errors (speech input)
   - due to homophones/misunderstandings
   - eg: peace → piece
   - eg: two → too

Another form of classification of errors

1. Typographic: typing errors
2. Orthographic: lack of comprehension
3. Phonetic: cognition of listener

Rate of Spelling Errors

26%: Web queries Wang *et al.* 2003

13%: Retyping, no backspace: Whitelaw *et al.* English&German

7%: Words corrected retyping on phone-sized organizer

2%: Words uncorrected on organizer Soukoreff &MacKenzie 2003

1-2%: Retyping: Kane and Wobbrock 2007, Gruden et al. 1983

Categories of Spell Checking Techniques

1. Non-word
2. Isolated
3. Context

1. Non-word errors

- Any word not in dictionary
- Larger dictionary better
- Generate candidate real words
  - shortest weighted edit distance
  - highest noisy channel probability
- Detection of non-words

2. Isolated-word error

- Find nearest meaningful word
- No context required
- Minimum edit distance, similarity key, rule-based methods, N-gram, Neural networks

3. Real-word / context dependent

- Candidate word with similar pronunciation, spelling
- Choose best candidate with noisy channel, classifier
- Context-dependent
- Peace of mind, piece of my mind

EDIT DISTANCE

- No. of edits to get from source string to destination string
- Operations
  - insert
  - delete
  - substitution

Minimum Edit Distance

- Minimum no. of operations required for editing
- Cost of operations (Levenshtein Distance)
  - insertion : 1
  - deletion : 1
  - substitution : 2
- Dynamic programming

Q: Transform vinter to writers

- vinter → vrinter insert r
- vrinter → vrinters insert s
- vrinters → writers substituted v → w
- wrinters → writers delete n

String Alignment

- Global alignment of strings S_1 and S_2
- Align S_1 & S_2 such that each char/space in one string is opposite a unique char/space in another string
- S_1 = qacdbd , S_2 = qawxb

S_1	q	a	c	d	b	d
S_2	q	a	w	x	b	-

Algorithm

- Let $D(i,j)$ denote edit distance of $S_1[1 \dots i]$ and $S_2[1 \dots j]$
  - minimum number of edits to transform first i chars of S_1 to first j chars of S_2

- Three parts of DP
  - recurrence relation
  - tabular computation
  - traceback

1. Recurrence relation

$$D(i,j) = \min \{ D(i-1,j)+1, \quad \text{deletion}$$
$$D(i,j-1)+1, \quad \text{insertion}$$
$$D(i-1,j-1) + t(i,j) \quad \text{substitution}$$
$$\}$$

$S_1[i] = S_2[j] \Rightarrow t(i,j) = 0$

else $\Rightarrow t(i,j) = 1$ or 2

- Initialisation

$D(i,0) = i$ i deletions

$D(0,j) = j$ j insertions

- Termination

$D(n,m)$ is distance

2. Tabular computation

- Bottom-up approach to compute $D(n,m)$ using $D(i,j)$ for smaller i,j

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

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

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

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

Figure 2.17 The minimum edit distance algorithm, an example of the class of dynamic programming algorithms. The various costs can either be fixed (e.g., $\forall x$, ins-cost$(x) = 1$) or can be specific to the letter (to model the fact that some letters are more likely to be inserted than others). We assume that there is no cost for substituting a letter for itself (i.e., sub-cost$(x,x) = 0$).

Q: S_1 = HELLO, S_2 = ELDER, use levenshtein dist (subs = 2)

target j

source i

	#	E	L	D	E	R
#	0	1	2	3	4	5
H	1					
E	2					
L	3					
L	4					
O	5					

D(H,E) = D(#,E) + del = 2

D(H,E) = D(#,#) + sub(H,E) = 2

D(H,E) = D(H,#) + ins = 2

target j

source i

	#	E	L	D	E	R
#	0	1	2	3	4	5
H	1	2				
E	2					
L	3					
L	4					
O	5					

D(H, EL) = D(#, EL) + del = 3

D(H,EL) = D(#, E) + sub (H,L) = 3

D(H, EL) = D(H,E) + ins = 3

target j

source i

	#	E	L	D	E	R
#	0	1	2	3	4	5
H	1	2	3	4	5	6
E	2	1	2	3	4	5
L	3	2	1	2	3	4
L	4	3	2	3	4	5
O	5	4	3	4	5	6

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

Figure 2.18 Computation of minimum edit distance between *intention* and *execution* with the algorithm of Fig. 2.17, using Levenshtein distance with cost of 1 for insertions or deletions, 2 for substitutions.

3. Traceback

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

Figure 2.19 When entering a value in each cell, we mark which of the three neighboring cells we came from with up to three arrows. After the table is full we compute an **alignment** (minimum edit path) by using a **backtrace**, starting at the **8** in the lower-right corner and following the arrows back. The sequence of bold cells represents one possible minimum cost alignment between the two strings. Diagram design after Gusfield (1997).

del i
n → e
t → x
ins c after e
n → u

Complexity

- Time: O(nm)
- Space: O(nm)
- Backtrace: O(n+m)

Weighted Edit Distance

- Some letters more likely to be mistyped (ie → ei, close by on keyboard)

- Frequency of substitution

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

Y (correct)

X	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

NOISY CHANNEL MODEL for SPELLINGS

- Most probable correct word from a misspelled word can be computed by
  1. Generating possible correct words with some edit distance model
  2. Estimating prior and likelihood probabilities using a language model
  3. Computing posteriors and MAP using Bayes Theorem

- Noisy Channel Model: framework used in Q&A systems, machine translators, spell checkers etc
- Goal: find intended word from misspelled word

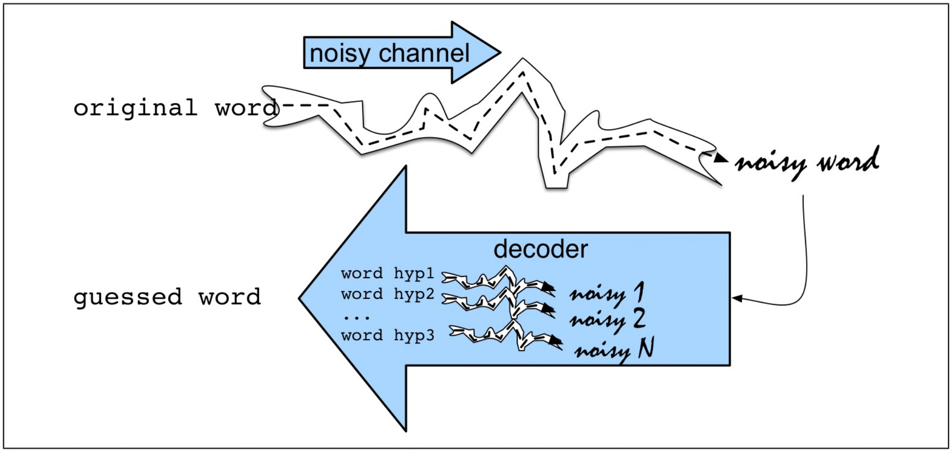


Figure B.1 In the noisy channel model, we imagine that the surface form we see is actually a "distorted" form of an original word passed through a noisy channel. The decoder passes each hypothesis through a model of this channel and picks the word that best matches the surface noisy word.

- Probabilistic model to generate list of probable words
- Given an alphabet Σ, let Σ^* = set of all finite strings over Σ
- Let dictionary D of valid words be some subset of Σ^*,

$$D \subseteq \Sigma^*$$

- The noisy channel is the matrix M, where every entry M_{wx} is the probability that $x \in \Sigma^*$ is the noisy word obtained as output given that $w \in D$ is the true word

$$M_{wx} = P(x|w)$$

The intuition of the **noisy channel** model (see Fig. B.1) is to treat the misspelled word as if a correctly spelled word had been "distorted" by being passed through a noisy communication channel.

This channel introduces "noise" in the form of substitutions or other changes to the letters, making it hard to recognize the "true" word. Our goal, then, is to build a model of the channel. Given this model, we then find the true word by passing every word of the language through our model of the noisy channel and seeing which one comes the closest to the misspelled word.

- The noisy channel model is a kind of Bayesian inference
- Out of all possible words in the dictionary D, we want to predict the true word w such that $P(w|x)$ is highest

$$\hat{w} = \underset{w \in D}{\operatorname{argmax}} \underbrace{P(w|x)}_{\text{posterior}}$$

- Bayes Theorem

$$\hat{w} = \underset{w \in D}{\operatorname{argmax}} \frac{\overbrace{P(x|w)}^{\text{noisy channel / likelihood / error model}} \overbrace{P(w)}^{\text{prior}}}{P(x)}$$

$$\hat{w} = \underset{w \in D}{\operatorname{argmax}} P(x|w) P(w)$$

Damerau-Levenshtein's Distance

- Possible edits
  1. Insertion
  2. Deletion
  3. Substitution
  4. Transposition of 2 adjacent letters

- ~80% of errors are within edit distance 1
- Almost all are within edit distance 2
- Allow insertion of space and hyphen
  - thisidea → this idea

language model

function NOISY CHANNEL SPELLING(*word x*, *dict D*, lm, *editprob*) **returns** *correction*

if $x \notin D$
  candidates, edits ← All strings at edit distance 1 from x that are $\in D$, and their edit
  for each c, e in candidates, edits
    channel ← editprob(e)
    prior ← lm(x)
    score[c] = log *channel* + log *prior*
  return argmax_c *score*[c]

Figure B.2 Noisy channel model for spelling correction for unknown words.

Q: Confusion matrix and candidate corrections for the misspelling "acress" — assuming edit distance of 1

		Transformation			
Error	Correction	Correct Letter	Error Letter	Position (Letter #)	Type
acress	actress	t	—	2	deletion
acress	cress	—	a	0	insertion
acress	caress	ca	ac	0	transposition
acress	access	c	r	2	substitution
acress	across	o	e	3	substitution
acress	acres	—	s	5	insertion
acress	acres	—	s	4	insertion

Figure B.3 Candidate corrections for the misspelling *acress* and the transformations that would have produced the error (after Kernighan et al. (1990)). "—" represents a null letter.

- P(w) — prior — computed from unigram language model (COCA)

w	count(w)	p(w)
actress	9,321	.0000231
cress	220	.000000544
caress	686	.00000170
access	37,038	.0000916
across	120,844	.000299
acres	12,874	.0000318

- Likelihood | channel model — P(x|w)
  - local context
  - corpus of errors

Candidate Correction	Correct Letter	Error Letter	x|w	P(x|w)
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

Figure B.4 Channel model for acress; the probabilities are taken from the *del*[], *ins*[], *sub*[], and *trans*[] confusion matrices as shown in Kernighan et al. (1990).

- 4 confusion matrices used to predict P(x|w) (likelihood)

$$\mathrm{del}[x,y]: \mathrm{count}(\mathtt{xy} \text{ typed as } \mathtt{x})$$
$$\mathrm{ins}[x,y]: \mathrm{count}(\mathtt{x} \text{ typed as } \mathtt{xy})$$
$$\mathrm{sub}[x,y]: \mathrm{count}(\mathtt{x} \text{ typed as } \mathtt{y})$$
$$\mathrm{trans}[x,y]: \mathrm{count}(\mathtt{xy} \text{ typed as } \mathtt{yx})$$

- From https://aclanthology.org/C90-2036.pdf

del[X, Y] = Deletion of Y after X

Y (Deleted Letter)

X	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	7	58	21	3	5	18	8	61	0	4	43	5	53	0	9	0	98	28	53	62	1	0	0	2	0
b	2	2	1	0	22	0	0	0	183	0	0	26	0	0	2	0	0	6	17	0	6	1	0	0	0	0
c	37	0	70	0	63	0	0	24	320	0	9	17	0	0	33	0	0	46	6	54	17	0	0	0	1	0
d	12	0	7	25	45	0	10	0	62	1	1	8	4	3	3	0	0	11	1	0	3	2	0	0	6	0
e	80	1	50	74	89	3	1	1	6	0	0	32	9	76	19	9	1	237	223	34	8	2	1	7	1	0
f	4	0	0	0	13	46	0	0	79	0	0	12	0	0	4	0	0	11	0	8	1	0	0	0	1	0
g	25	0	0	2	83	1	37	25	39	0	0	3	0	29	4	0	0	52	7	1	22	0	0	0	1	0
h	15	12	1	3	20	0	0	25	24	0	0	7	1	9	22	0	0	15	1	26	0	0	1	0	1	0
i	26	1	60	26	23	1	9	0	1	0	0	38	14	82	41	7	0	16	71	64	1	1	0	0	1	7
j	0	0	0	0	1	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	1	0	0	0	0	0
k	4	0	0	1	15	1	8	1	5	0	1	3	0	17	0	0	0	1	5	0	0	0	1	0	0	0
l	24	0	1	6	48	0	0	0	217	0	0	211	2	0	29	0	0	2	12	7	3	2	0	0	11	0
m	15	10	0	0	33	0	0	1	42	0	0	0	180	7	7	31	0	0	9	0	4	0	0	0	0	0
n	21	0	42	71	68	1	160	0	191	0	0	0	17	144	21	0	0	0	127	87	43	1	1	0	2	0
o	11	4	3	6	8	0	5	0	4	1	0	13	9	70	26	20	0	98	20	13	47	2	5	0	1	0
p	25	0	0	0	22	0	0	12	15	0	0	28	1	0	30	93	0	58	1	18	2	0	0	0	0	0
q	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	18	0	0	0	0	0
r	63	4	12	19	188	0	11	5	132	0	3	33	7	157	21	2	0	277	103	68	0	10	1	0	27	0
s	16	0	27	0	74	1	0	18	231	0	0	2	1	0	30	30	0	4	265	124	21	0	0	0	1	0
t	24	1	2	0	76	1	7	49	427	0	0	31	3	3	11	1	0	203	5	137	14	0	4	0	2	0
u	26	6	9	10	15	0	1	0	28	0	0	39	2	111	1	0	0	129	31	66	0	0	0	0	1	0
v	9	0	0	0	58	0	0	0	31	0	0	0	0	0	2	0	0	1	0	0	0	0	0	0	1	0
w	40	0	0	1	11	1	0	11	15	0	0	1	0	2	2	0	0	2	24	0	0	0	0	0	0	0
x	1	0	17	0	3	0	0	1	0	0	0	0	0	0	0	6	0	0	0	5	0	0	0	0	1	0
y	2	1	34	0	2	0	1	0	1	0	0	1	2	1	1	1	0	0	17	1	0	0	1	0	0	0
z	1	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2
@	20	14	41	31	20	20	7	6	20	3	6	22	16	5	5	17	0	28	26	6	2	1	24	0	0	2

add[X, Y] = Insertion of Y after X

Y (Inserted Letter)

X	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	15	1	14	7	10	0	1	1	33	1	4	31	2	39	12	4	3	28	134	7	28	0	1	1	4	1
b	3	11	0	0	7	0	1	0	50	0	0	15	0	1	1	0	0	5	16	0	0	3	0	0	0	0
c	19	0	54	1	13	0	0	18	50	0	3	1	1	1	7	1	0	7	25	7	8	4	0	1	0	0
d	18	0	3	17	14	2	0	0	9	0	0	6	1	9	13	0	0	6	119	0	0	0	0	0	5	0
e	39	2	8	76	147	2	0	1	4	0	3	4	6	27	5	1	0	83	417	6	4	1	10	2	8	0
f	1	0	0	0	2	27	1	0	12	0	0	10	0	0	0	0	0	5	23	0	1	0	0	0	1	0
g	8	0	0	0	5	1	5	12	8	0	0	2	0	1	1	0	1	5	69	2	3	0	1	0	0	0
h	4	1	0	1	24	0	10	18	17	2	0	1	0	1	4	0	0	16	24	22	1	0	5	0	3	0
i	10	3	13	13	25	0	1	1	69	2	1	17	11	33	27	1	0	9	30	29	11	0	0	1	0	1
j	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
k	2	4	0	1	9	0	0	1	1	0	1	1	0	0	2	1	0	0	95	0	1	0	0	0	4	0
l	3	1	0	1	38	0	0	0	79	0	2	128	1	0	7	0	0	0	97	7	3	1	0	0	2	0
m	11	1	1	0	17	0	0	1	6	0	1	0	102	44	7	2	0	0	47	1	2	0	1	0	0	0
n	15	5	7	13	52	4	17	0	34	0	1	1	26	99	12	0	0	2	156	53	1	1	0	0	1	0
o	14	1	1	3	7	2	1	0	28	1	0	6	3	13	64	30	0	16	59	4	19	1	0	0	1	1
p	23	0	1	1	10	0	0	20	3	0	0	2	0	0	26	70	0	29	52	9	1	1	1	0	0	0
q	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
r	15	2	1	0	89	1	1	2	64	0	0	5	9	7	10	0	0	132	273	29	7	0	1	0	10	0
s	13	1	7	20	41	0	1	50	101	0	2	2	10	7	3	1	0	1	205	49	7	0	1	0	7	0
t	39	0	0	3	65	1	10	24	59	1	0	6	3	1	23	1	0	54	264	183	11	0	5	0	6	0
u	15	0	3	0	9	0	0	1	24	1	1	3	3	9	1	3	0	49	19	27	26	0	0	2	3	0
v	0	2	0	0	36	0	0	0	10	0	0	1	0	1	0	1	0	0	0	0	1	5	1	0	0	0
w	0	0	0	1	10	0	0	1	1	0	1	1	0	2	0	0	1	1	8	0	2	0	4	0	0	0
x	0	0	18	0	1	0	0	6	1	0	0	0	1	0	3	0	0	0	2	0	0	0	0	1	0	0
y	5	1	2	0	3	0	0	0	2	0	0	1	1	6	0	0	0	1	33	1	13	0	1	0	2	0
z	2	0	0	0	5	1	0	0	6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	4
@	46	8	9	8	26	11	14	3	5	1	17	5	6	2	2	10	0	6	23	2	11	1	2	1	1	2

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

Y (correct)

X	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

rev[X, Y] = Reversal of XY

Y

X	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	2	1	1	0	0	0	19	0	1	14	4	25	10	3	0	27	3	5	31	0	0	0	0	0
b	0	0	0	0	2	0	0	0	0	0	0	1	1	0	2	0	0	0	2	0	0	0	0	0	0	0
c	0	0	0	0	1	0	0	1	85	0	0	15	0	0	13	0	0	0	3	0	7	0	0	0	0	0
d	0	0	0	0	0	0	0	0	7	0	0	0	0	0	0	0	0	1	0	0	2	0	0	0	0	0
e	1	0	4	5	0	0	0	0	60	0	0	21	6	16	11	2	0	29	5	0	85	0	0	0	2	0
f	0	0	0	0	0	0	0	0	12	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
g	4	0	0	0	2	0	0	0	0	0	0	1	0	15	0	0	0	3	0	0	3	0	0	0	0	0
h	12	0	0	0	15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	0	0	0	0	0	0
i	15	8	31	3	66	1	3	0	0	0	0	9	0	5	11	0	1	13	42	35	0	6	0	0	0	3
j	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
k	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
l	11	0	0	12	20	0	1	0	4	0	0	0	0	0	1	3	0	0	1	1	3	9	0	0	7	0
m	9	0	0	0	20	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	4	0	0	0	0	0
n	15	0	6	2	12	0	8	0	1	0	0	0	3	0	0	0	0	0	6	4	0	0	0	0	0	0
o	5	0	2	0	4	0	0	0	5	0	0	1	0	5	0	1	0	11	1	1	0	0	7	1	0	0
p	17	0	0	0	4	0	0	1	0	0	0	0	0	0	1	0	0	5	3	6	0	0	0	0	0	0
q	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
r	12	0	0	0	24	0	3	0	14	0	2	2	0	7	30	1	0	0	0	2	10	0	0	0	2	0
s	4	0	0	0	9	0	0	5	15	0	0	5	2	0	1	22	0	0	0	1	3	0	0	0	16	0
t	4	0	3	0	4	0	0	21	49	0	0	4	0	0	3	0	0	5	0	0	11	0	2	0	0	0
u	22	0	5	1	1	0	2	0	2	0	0	2	1	0	20	2	0	11	11	2	0	0	0	0	0	0
v	0	0	0	0	1	0	0	0	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
w	0	0	0	0	0	0	0	4	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	8	0
x	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
y	0	1	2	0	0	0	1	0	0	0	0	3	0	0	0	2	0	1	10	0	0	0	0	0	0	0
z	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Sources of data

- Roger Mitton: https://www.dcs.bbk.ac.uk/~roger/corpora.html
- Peter Norvig: https://norvig.com/ngrams/

Estimation of P(x|w)

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

- $del[x_1, w_2]$ = count $(x_1 w_2$ typed as $x_1)$

Candidate Correction	Correct Letter	Error Letter	x|w	P(x|w)	P(w)	10^9*P(x|w)P(w)
actress	t	-	c|ct	.000117	.0000231	2.7
cress	-	a	a|#	.00000144	.000000544	0.00078
caress	ca	ac	ac|ca	.00000164	.00000170	0.0028
access	c	r	r|c	.000000209	.0000916	0.019
across	o	e	e|o	.0000093	.000299	2.8
acres	-	s	es|e	.0000321	.0000318	1.0
acres	-	s	ss|s	.0000342	.0000318	1.0

Figure B.5 Computation of the ranking for each candidate correction, using the language model shown earlier and the error model from Fig. B.4. The final score is multiplied by 10^9 for readability.

Using a Bigram Language Model

- "a stellar and versatile **acress** whose combination of sass and glamour…"
- Counts from the Corpus of Contemporary American English with add-1 smoothing
- P(actress|versatile)=.000021 P(whose|actress) = .0010
- P(across|versatile) =.000021 P(whose|across) = .000006

- **P("versatile actress whose") = .000021*.0010 = 210 $\times 10^{-10}$**
- P("versatile across whose") = .000021*.000006 = 1 $\times 10^{-10}$

Real Word Errors

- Generate candidate set containing
  - the word itself
  - all single-letter edits that are in the dictionary
  - homophones

- Choose best candidates
  - noisy channel model
  - task-specific classifier

NOISY CHANNEL MODEL for REAL WORDS

- Given a sentence $w_1, w_2, w_3, \ldots, w_n$

- Generate a set of candidates for each word w_i
  - Candidate $(w_1) = \{w_1, w_1', w_1'', w_1''', \ldots\}$
  - Candidate $(w_2) = \{w_2, w_2', w_2'', w_2''', \ldots\}$
  - Candidate $(w_3) = \{w_3, w_3', w_3'', w_3''', \ldots\}$

- Choose sequence that maximises P(W)

Eg. The candidate set for the real word error thew might be
C(thew) = {the, thaw, threw, them, thwe}

- Make simplifying assumption: only one misspelled word in each sentence

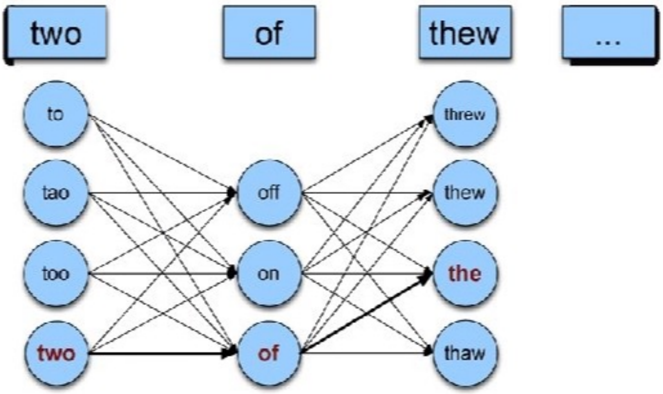
  - Thus the set of candidate sentences $C(X)$ for a sentence X = Only two of thew apples would be:

```
only two of thew apples
oily two of thew apples
only too of thew apples
only to of thew apples
only tao of the apples
only two on thew apples
only two off thew apples
only two of the apples
only two of threw apples
only two of thew applies
only two of thew dapples
...
```

- Find W that maximises P(W)

W = two of the

Noisy channel for real-word spell correction



- Probability of No Error – P(the | the)
- Channel probability for a correctly typed word
- Noisy channel scores every sentence

$$\hat{W} = \underset{W \in C(X)}{\operatorname{argmax}} P(W|X)$$

- Can use uni, bi or trigram probability of the sentence to compute P(W)

Compute Channel Model Probability $P(x|w)$

- x - real word error
- Let channel model $P(x|w) = \alpha$ when $x = w$ ← probability of no error
- Distribute $1-\alpha$ evenly over all other candidate corrections $C(x)$

$$P(x|w) = \begin{cases} \alpha & \text{if } x = w \\ \dfrac{1-\alpha}{|C(x)|} & \text{if } x \in C(x) \\ 0 & \text{Otherwise} \end{cases}$$

→ no error (α decided by design)

→ candidate errors equally likely

- For the above example two of thew, using 3-gram Stupid Backoff model trained on Google n-grams

$$\begin{aligned}
P(\text{the}|\text{two of}) &= 0.476012 \\
P(\text{thew}|\text{two of}) &= 9.95051 \times 10^{-8} \\
P(\text{thaw}|\text{two of}) &= 2.09267 \times 10^{-7} \\
P(\text{threw}|\text{two of}) &= 8.9064 \times 10^{-7} \\
P(\text{them}|\text{two of}) &= 0.00144488 \\
P(\text{thwe}|\text{two of}) &= 5.18681 \times 10^{-9}
\end{aligned}$$

Following Norvig (2009), we assume that the probability of a word being a typo in this task is .05, meaning that $\alpha = P(w|w)$ is .95. Fig. B.6 shows the computation.

x	w	x|w	P(x|w)	$P(w\|w_{i-2}, w_{i-1})$	$10^8 P(x\|w) P(w\|w_{i-2}, w_{i-1})$
thew	the	ew|e	0.000007	0.48	333
thew	thew		α=0.95	9.95×10^{-8}	9.45
thew	thaw	e|a	0.001	2.1×10^{-7}	0.0209
thew	threw	h|hr	0.000008	8.9×10^{-7}	0.000713
thew	thwe	ew|we	0.000003	5.2×10^{-9}	0.00000156

Figure B.6 The noisy channel model on 5 possible candidates for `thew`, with a Stupid Backoff trigram language model computed from the Google N-gram corpus and the error model from Norvig (2009).

For the error phrase *two of thew*, the model correctly picks *the* as the correction. But note that a lower error rate might change things; in a task where the probability of an error is low enough (α is very high), the model might instead decide that the word *thew* was what the writer intended.

State-of-the-Art Systems

- Autocorrect (confident in correction)
- Give best correction (less confident)
- Give correction list (even less confident)
- Flag as error (unconfident)
- Weigh probabilities

$$\hat{w} = \text{argmax } P(x|w)\, P(w)^{\lambda}$$

- Learn λ from development test set
- Phonetic error model

- Improvements to Channel Model

  1. Richer edits – Brill and Moore, 2000
     ent → ant
     ph → f
     le → al

  2. Incorporate pronunciation into channel – Toutanova and Moore, 2002

- Classifier-based methods for specific pairs etc
  - weather / whether