Natural Language Processing

Part 1: Words….

- Morphology
- Spell checking
- N-gram models

Language & knowledge - 1

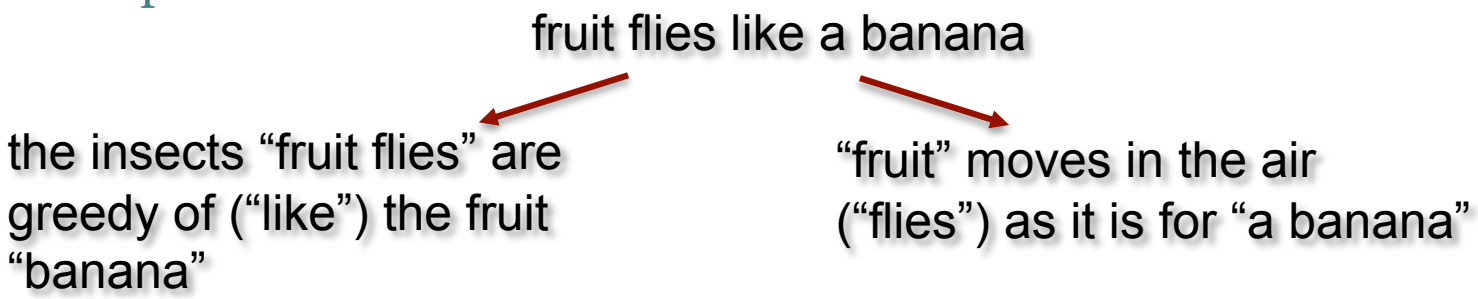
- The understanding/generation of natural language requires knowledge at different levels
  - Lexical
    The dictionary of used words, the concept of “word”
    - dog, cat, Marco, 2011 (the year)
  - Morphological
    The rules for modifying words given their use and context (plural, gender, contraction, conjugations)
    - the black dog (il cane nero), the black dogs (i cani neri)
    - the white –female- cat (la gatta bianca), the white –male- cat (il gatto bianco)
  - Syntactical
    The rules for composing word sequences and to build sentences (the meaning depends on the word ordering..)
    - The dog bites the cat, The dog the cat bites (?)

Language & knowledge- 2

- **Lexical and compositional semantics**
  The interpretation of a sentence requires knowledge on the meanings of its words and how these components are combined to yield the overall meaning
  - The dog bites the cat, The cat bites the dog
- **Pragmatics**
  Conventions and indirect expressions are used to formulate the sentences. The way to express a concept may depend on a specific context or situation that does not concern the language in itself (social, environmental, psychological). The context allows a correct correlation between the (lexical) meaning of the sentence with the speaker intentions (the concept he/she wants to deliver)
  - I am sorry I may not be able to come (I will not come)
  - This is cool, See you later, I'll be back soon (deixis)
- The context includes social/cultural knowledge shared by the source and recipient of the message, their current spatio/temporal state, their roles, expectations, motivations, the progress in their conversation

Ambiguities

- Many phases of natural language processing aim at solving the ambiguities
  - A linguistic structure (f.i. sentence) is ambiguous if it has more than one interpretation



  - The ambiguities can be at different linguistic levels
    - lexical-morphology (fruit - noud/adj, flies - noun/verb, like - verb/prep)
    - semantics
    - syntax (subject: fruit flies/fruit; verb: flies/like)

Other examples of ambiguities

- Syntactic ambiguity
  - I saw the man with the binoculars
    - Who has the binoculars?
  - One morning I shot an elephant in my pajamas (*Groucho Marx*)
    - Who wears the pajamas? The elephant in the joke!
  - Flying planes can be dangerous
    - Being a pilot is a dangerous job?
  - Toilet out of order. Please use floor below
    - I hope you don't misunderstand the advice!!
  - I once saw a deer riding my bicycle
    - The deer likes cycling….
- Incremental parsing ambiguity (Reduced relative clauses)
  - The horse raced past the barn fell
    - The syntax changes when the last word arrives… "raced" is a Past Participle not Past Tense

Eliza (1966)

- Eliza, developed by Weizenbaum in 1966, is one of the first software applications that can manage a constrained conversation with an user
  - It emulates the answers of a Rogerian psychotherapist
  - It is a simple program that exploits pattern-matching to process the input sentence and to generate the reply
  - Actually it does not implement any knowledge/model of the real behavior of the psychotherapist
  - It creates a credible interaction since it simulates a kind of dialog where the agent can act without a real knowledge of the world (the reply is always built upon the sentence proposed by the other agent)
  - It is the first example of a chatterbot
  - The surprising outcome is that some users were actually believing that Eliza was really understanding their problems! (even after that the program characteristics had been clearly explained to them)

Eliza - example

- Replies are based on a script
  - When no specific reply can be obtained from the small knowledge base, the engine uses some rules for reformulating the question starting from the sentence proposed by the “patient”

>> Hello.
How do you do. Please state your problem.
>> I feel sad
Tell me more about such feelings.
>> My mother hates me
Tell me more about your family.
>> My father fled when I was young
Who else in your family fled when you was young ?
>> noone
Really, noone ?
>> Also my sister fled away
Your sister ?

Eliza – processing 1

- The input sentences is split into words
- A set of pre-replacements is applied

```
pre: how what
pre: you're you are
```

- All the found keywords are listed in descending weight value

```
key: remember 5
key: if 3
key: am 0
key: i 0
```

  - the keywords are sequentially processed until an output is generated

Eliza – processing 2

- For a given keyword, the corresponding decomposition rules in the knowledge base are checked
  - The first matching the input pattern is selected. If there is no match the following keyword is checked

```
key: i 0
  decomp: * i am* @sad *
```

- For the matching decomposition rule, one of the listed reassemble rules is applied (they are selected with a round robin policy)

```
 key: i 0
decomp: * i feel *
   reasmb: Tell me more about such feelings.
   reasmb: Do you often feel (2) ?
   reasmb: Do you enjoy feeling (2) ?
   reasmb: Of what does feeling (2) remind you ?
```

Eliza – processing 3

- The post-replacements are applied

```
post: me you
post: i you
post: you I
```

- The resulting string is sent to the output

- The script also allows the definition of
  - What can be said at the beginning (initial) or at the end (final)
  - a list of synonyms (synon: sad unhappy depressed sick)
  - the actions to be performed when no keywords are found (key: xnone - reasmb: Please go on.)
  - a set of candidate replies that can be stored for random selection (the decomposition rule starts with $)

Words and morphology

- Words are the atomic elements in a language
  - Any application for automatic language processing heavily exploits the lexical knowledge
  - Regular expressions are a useful model for lexical entities
    - Modeling of word inflections (f.i. boy(ε|s), bell(a|o|e|i))
    - Modeling of lexical categories (f.i. price [09]+'.'[0-9]{2}' '€)
  - An important role in word analysis is played by morphological rules that model the transformations that can be applied to yield inflections or derivations of the same lexical unit (stem)
    - Plurals, verbal modes
    - The morphological rules are related to spelling rules
  - Morphological parsing aims at the decomposition of an input word into its component morphemes
    - seeing -> see – ing, boys -> boy - s

Morphological parsing

- Morphological parsing aims at the detection of word components (morphemes) given an inflected (or derived) input form (surface)
  - A similar task is stemming whose goal is to remove variations in words mapping them to a reference form (stem)
  - This task requires to model the morphological knowledge that is language dependent
  - An alternative solution is to store all the variations of the stem in the dictionary
    - It is a potentially inefficient solution
    - F.i. in English, the –ing suffix allows the generation of the ing form for any verb and the –s suffix is used for the plural of most nouns (productive suffix)
    - Generally standard suffixes are used for new words and the dictionary can be automatically extended (fax – faxare, click – cliccare for Italian)
    - The listing of all the morphological variants can be complex for some languages (composed words as in German, Turkish, Arabic)

Morphemes & affixes

- A morpheme is the atomic units carrying meaning in a language
- There are two main categories of morphemes
  - Stem – the main morpheme in a word (it defines the word meaning)
  - Affixes – they add additional meanings of different types
    - prefixes – the recede the stem (unset : un- set; in Italian in-, ri- , dis-,...)
    - suffixes – they follow the stem (boys: boy –s; in Italian –mente, -tore, -zione, ..)
    - infixes – they are inserted into the stem (in some languages – Tagalog Philippines)
    - circumfixes – they precede and follow the stem (in German sagen [to say] the past participle is ge- sag -t)
  - Prefixes and suffixes are often referred to concatenative morphology since a word is obtained by the concatenation of morphemes
  - Some languages have more complex compositional rules and follow a non-concatenative morphology (f.i. involving infixes)

Morphology – inflection & derivation

- A non-concatenative morphology may be based on templates or root-and-pattern
  - In Semitic languages the word root consists of a sequences of consonants and derived morphemes are obtained by inserting a specific vowel pattern
    - lmd [study/learn]: lamad [he studied], limed [he taught], lumad [he was taught]
- A word may have more than one affix
  - rewritten: re- writ –ten; unbelievably: un- believ –able –ly
  - disconnected: dis- connect –ed; unreasonably: un- reason -abl –y
  - The maximum number of affixes depends on the language. Languages that may have many affixes are said to be agglutinative (Turkish up to 9-10 affixes/word)
- There two main modalities to yield words from morphemes
  - inflection – the stem is combined with a morpheme to obtain a word of the same grammatical class (f.i. English plurale–s or past tense –ed)
  - derivation – the stem is combined with a morpheme to obtain a word of a different grammatical class (f.i. compute [Verb] – computation [Noun])

Morphology – inflections in English: nouns

- Inflection system in English is quite simple
- Only Nouns, Verbs and some Adjectives can be inflected using a very small set of affixes
  - Nouns
    - plural
      Most of the nouns appear in the root stem corresponding to the singular or the plural suffixes is added (-s ;-es for stems ending in –s, -z, -sh, -ch, sometimes in –x and for nouns ending in y preceded by a consonant for which the y is transformed into i -ies)
    - Saxon genitive
      It is obtained with the suffix –'s for singular nouns and plurals not ending in s (f.i. children's) and with the suffix –' for regular plurals and some nouns ending in s or z)

Morphology – inflections in English: verbs

- Verbs
  The inflection rules for verbs are more complex than for nouns (but other languages as Italian they are much more complex…)
  - There are 3 verb categories: main verbs (eat,sleep,walk,…), modal verbs (can, will, may,..) and primary verbs (have, be, do)
  - Most of main verbs are regular, that is the share the same set of suffixes with the same functionality

stem	walk	kiss	map	cry
-s form	walks	kisses	maps	cries
-ing participle	walking	kissing	mapping	crying
Past form or -ed participle	walked	kissed	mapped	cried

  - For regular verbs it is possible to generate all the inflections given the stem by adding the suffixes (eventually some spelling rules must be applied)

Morphology – inflections in English: irregular verbs

- The class of regular verbs is also said *productive* since it is the one used to generate new words (in Italian is the first inflection form with infinite ending in are)
- Irregular verbs have inflection forms that are not easy to be explained (usually they have 5 different forms, but there are up to 8 for "to be")
  - They consist of few verbs (about 250) but they are among the most used in the language

stem	eat	catch	cut
-s form	eats	catches	cuts
-ing participle	eating	catching	cutting
Past form	ate	caught	cut
- ed participle	eaten	caught	cut

Morphology – derivations in English

- In general morphological derivations are more complex and less predictable (common derivations are often not applicable in general)
  - nominalization – Generation of nouns from verbs and adjectives

-ation	compute (V)	computation
-ee	attend (V)	attendee
-er	kill (V)	killer
-ness	happy (A)	happiness

  - Generation of adjectives from verbs and nouns

-al	computation (N)	computational
-able	drink (V)	drinkable
-less	fear (N)	fearless

Morphological parsing

- A morphological parser associates an input word to its stem and a set of word features (f.i. number/person/mode/tense/…)

student	student +N +SG
students	student +N +PL
cities	city +N +PL
children	child +N +PL
looks	look +V +3SG look +N +PL
carrying	carry +V +PRES-PART
taught	teach +V +PAST-PART teach +V +PAST

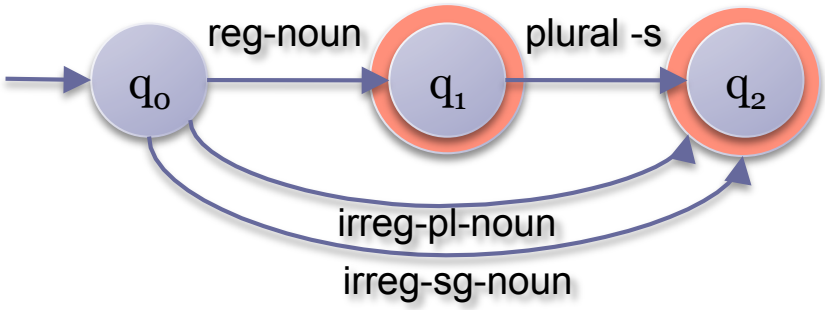
+N : Noun
+PL: Plural
+V : Verb
+3SG : third singular person
+PAST : Past form
+PAST-PART : Past Participle
+PRES-PART : Present Participle

Morphological parsers

- A morphological parser exploits the following sources of knowledge
  - Lexicon
    The list of the stems and of the affixes with the core information on them (f.i. if a stem is a verb or a noun)
  - Morphotactics
    The model for the morpheme composition rules (f.i. which morpheme classes can follow another class of morphemes in a word).
  - Orthographic rules (spelling rules)
    These rules describe how orthographic changes are applied when morphemes are combined to form a word

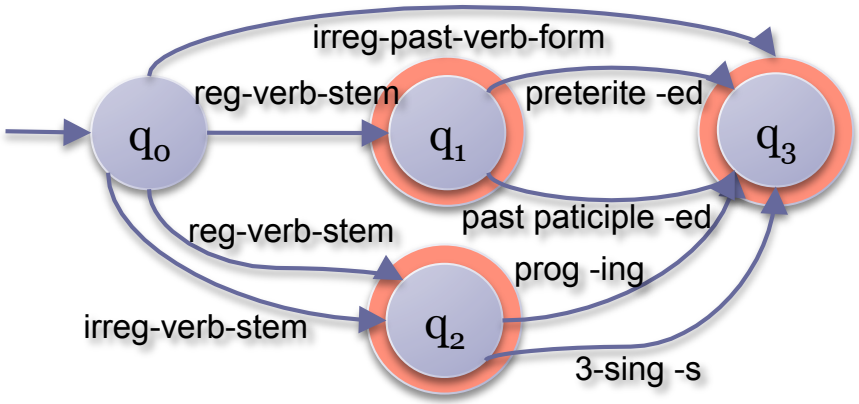
  - In general, rules on morpheme combination can be modeled by Finite State Automata (FSA)

Morphological parsers- example 1



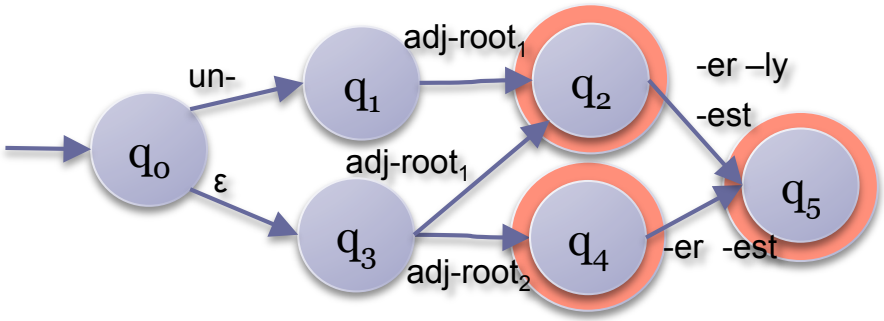
- The parser models the presence of regular nouns (reg-noun) whose plural is obtained by adding the suffix –s
  - It models orthographic variations as in *city-cities*
- The parser manages separately irregular nouns for which the singular (irreg-sg-noun) and the plural (irreg-pl-noun) are not easily correlated
  - f.i. *mouse-mice, child-children*

Morphological parsers- example 2



- The parser exploits 3 different stem classes for verbs
  - reg-verb-stem, irreg-verb-stem, irreg-past-verb-form
- There are 4 classes for the affixes
  - -ed past, -ed past participle, -ing participle, 3rd singular -s

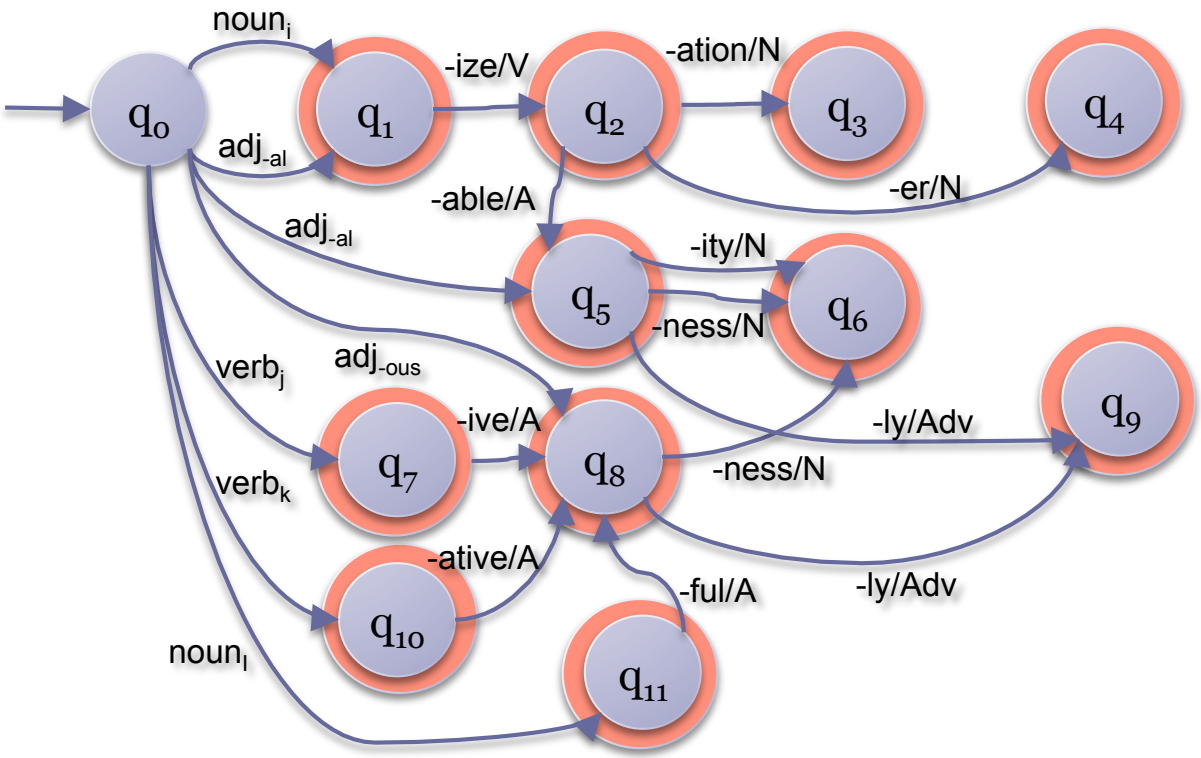
Morphological parsers- example 3



big, bigger, biggest
cool, cooler, coolest, cooly
clear, clearer, clearest, clearly
unclear, unclearly
happy, happier, happiest, happily
unhappy, unhappier, unhappiest, unhappily

- The adjectives may have different types of affixes
  - an optional prefix –un (but not for all!)
  - a mandatory root adj-root
  - an optional suffix (-er –ly –est) with eventual orthographic fixes
  - The adjectives are split into two classes depending on if they admit the prefix –un and the suffix -ly (Antworth, 1990)

Morphological parsers- example 4



verbs ending in –ize may be nominalized with the suffix -ation

fossil**e**-iz**e**-ation/N

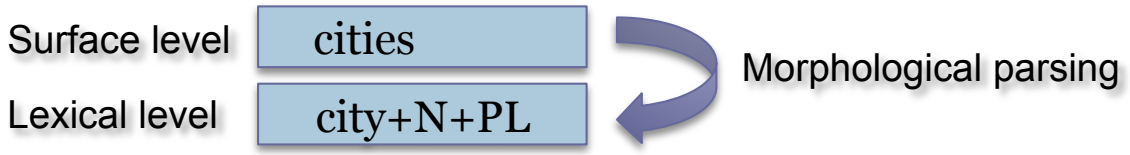
adjectives ending in –al or –able may be combined with the suffix –ity or sometimes -ness

natural-ness/N

- FSA for nominal and verbal derivations
  - there are many exceptions to these rules

Morphological recognizers/parsers

- Morphological recognizers can be obtained from the models based on FSA
  - The morphological recognizer decides if a word belongs to a given language
  - FSA modeling morphotactics and exploiting sub-lexicons that include lists of stems
    - sub-lexicons allow the expansion of some arcs (adj-root$_1$, adj-root$_2$,noun$_i$,..)
- In general we are interested in morphological parsing
  - The stem (root morpheme) and the features are extracted from the word

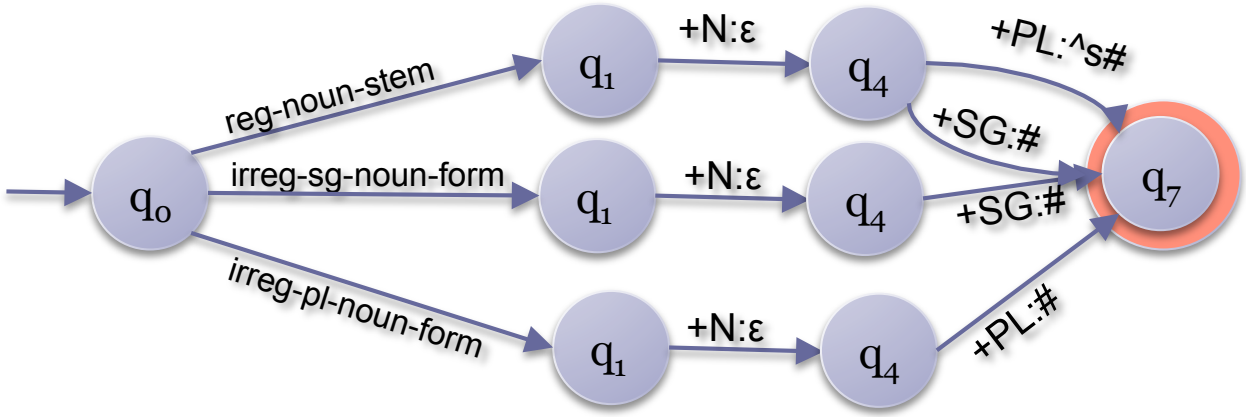


Finite-State Transducers (FST)

- The mapping between two strings can be modeled with a Finite-State Transducer
  - A FST defines a relation between two sets of strings
  - It is an extension of FSA as a Mealy machine
    - $Q = \{q_0,\ldots,q_N\}$ is the finite set of states
    - Σ is a finite alphabet of complex symbols. Each symbol is an input-output pair $i{:}o$ where i is a symbol from the input alphabet I and o is a symbol from the output alphabet O (that includes also the symbol ε)
    - q_0 is the start state
    - $F \subseteq Q$ is the set of final accepting states
    - $\delta(q,i{:}o)$ is the state transition function
  - The set of pairs in Σ are also referred as admissible pairs
  - A FST models regular relations, an extension of regular languages to relations between sets of strings

FST – computation model

- The input of the FST corresponds to the lexical level and the output to the surface level
  - in the pair *i*:*o*, *i* corresponda to a character in the morphemes (lexical level) and *o* to a character in the word or ε (surface level)
    - Each pair *i*:*o* defines how the input symbol *i* is mapped to the output symbol *o*
    - Since it is common that an input symbol is mapped to itself, the default pairs *a*:*a* can be expressed as *a*



is the word end marker

^s inserts s by applying the orthographic rules

FST - categories

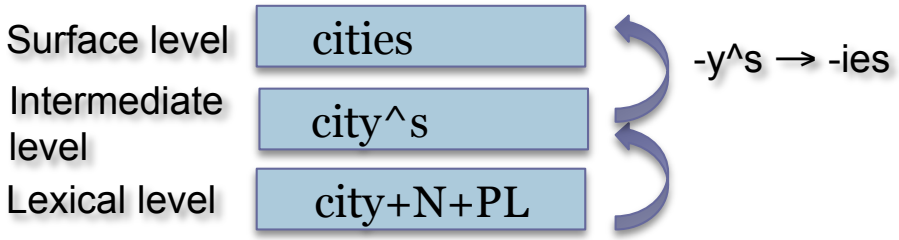
- Le transitions labeled with sub-lexicon categories must be expanded
  - The lexical must be updated such that irregular plurals are correctly mapped to the singular morpheme

reg-noun	irreg-pl-noun	irreg-sg-noun
c:c a:a t:t	g:g o:e o:e s:s e:e	g:g o:o o:o s:s e:e
d:d o:o g:g	m:m o:i u:ɛ s:c e:e	m:m o:o u:u s:s e:e

  - The complete automaton is obtained by the cascade or composition of the FST that maps the stems and the FST that adds the affixes
  - However the concatenation of morphemes does not work when there is a spelling fix (f.i. box -> boxes)
    - Orthographic rules must be applied at the morpheme boundaries
    - These rules can be implemented by transducer automata

FST – orthographic rules

- The application of orthographic rules at the morpheme boundaries after their concatenation can be modeled by another FST



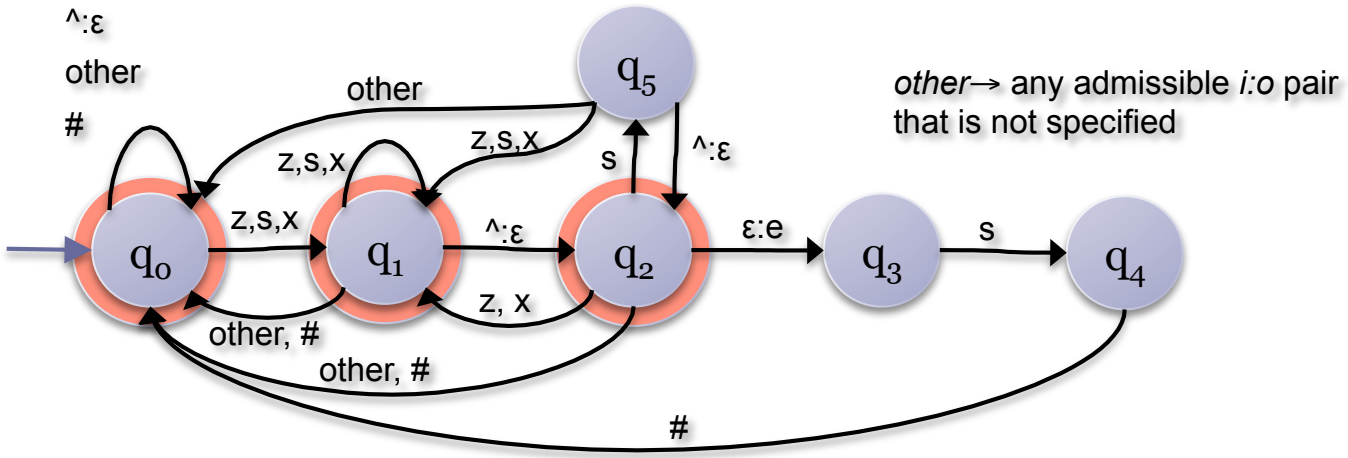
  - The rule for the insertion of the E character when building a plural can be defined with the notation

$$\epsilon \rightarrow \mathrm{e} / \left\{ \begin{array}{c} x \\ s \\ z \end{array} \right\} \hat{} \_\_ s\#$$

replace ε with e when it appears between (x|s|z)^ and s#

(the replacement of ε is actually an insertion)

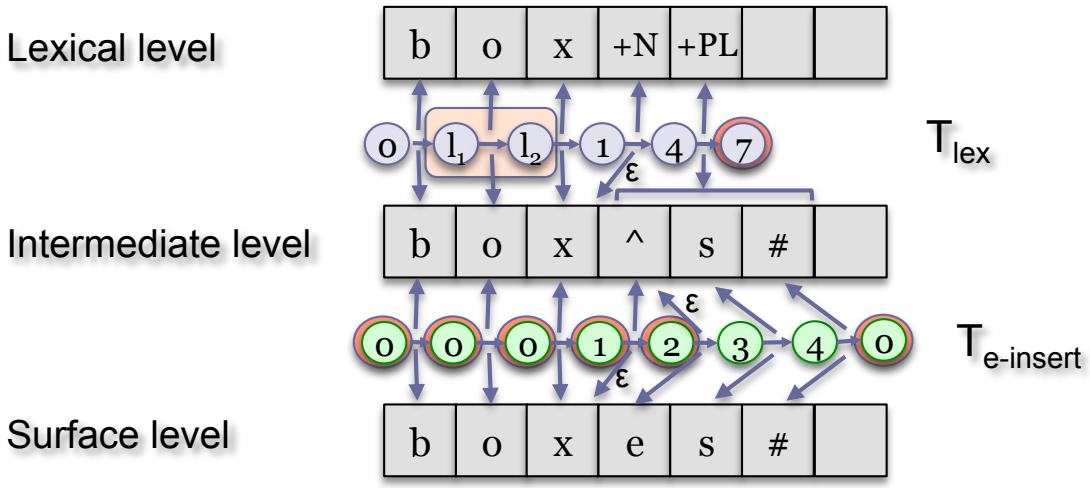
FST – E-insertion rule



- The pair ε:e between q_2 and q_3 models the insertion
- The pair ^:ε corresponds to the deletion of the start markers of the morphemes to generate the word
- The FST applies the rule only to the configurations that require it making no changes in the other cases (the rule ε:e is applied only in the right context)
  - State q_3 models the insertion of the e that requires a further insertion of s and the subsequent word ending (#)

FST+Lexicon+Rules

- A morphological generator/parser can be built by stacking the modules for the morphological/orthographic parsing



- The same FST stack can be used in modeling the generation lexical→surface or parsing surface→lexical
  - Parsing may be more complex due to ambiguities (they can be solved only given the word context at the sentence level)
    - box +N +PL and box +V +3SG are both correct

SFTS

- There are tools for the automatic generation of FSTs
  - Stuttgart FST tools
    - A programming language for specifying a FST (SFST-PL)
    - A compiler for programs in SFST-PL (fst-compiler)
    - A set of tools for using, printing, compare FSTs
    - A library for FSTs in C++
  - A FST can be specified by REs defined on the pairs i:o
    - (a:b|b:a|c:c)* maps the input string into an output string where the symbols c are left unchanged whereas the symbols a and b are swapped
    - a:b maps a to b in generation and b to a in parsing
    - the pair c:c can be shortened as c

```
(house|foot|mouse) <N>:<> <sg>:<> | \
(house<>:s | f o:e o:e t | {mouse}:{mice}) <N>:<> <pl>:<>
```

Stemming – Porter stemmer

- A stemmer is a simple morphological parser that does not exploit a lexicon
  - It is used in Information Retrieval for keyword normalization
  - The stemming algorithm proposed by Porter (1980) for English is based on a cascade of simple rewriting rules
    - ATIONAL → ATE (relational → relate)
    - ING → ε if the stem contains a vowel (monitoring → monitor)
  - The oversimplification makes stemmers prone to errors and failures
    - university → universal, organization → organ
    - sparse … sparsity, matrices … matrix, explain… explanation (failures)
  - The Retrieval improvement is generally low (they are not used often)

Porter stemmer - 1

- Sets of literals
  - consonants (c): all literals excluding A, E, I, O, U and Y preceded by a consonant
  - vowels (v): A, E, I, O, U and Y not proceded by a consonant
- Each English word is approximated with the regular expression
  - $(C|\varepsilon)(VC)^m(V|\varepsilon)$ where C is a sequence of one or more consonants and V is a sequence of vowels
  - m is referred to as word (or part-of-word) measure
    - m=0 [TR][][], [][][EE], [TR][][EE]
    - m=1 [TR][OUBL][E],[TR][EES][]
    - m=2 [TR]([OUBL][ES])[], [PR]([IV][AT])[E]
- The rewriting rules have the following structure
  - (condition) S1 → S2 : If the word has the suffix S1 and the stem preceding S1 satisfies the condition, then S1 is rewritten with S2

Porter stemmer - 2

- Conditions are built using the following atoms combined with logic operators
  - m - the stem measure
  - *S - the stem ends in S (similar to other literals)
  - *v* - the stem contains a vowel
  - *d - the stem ends with a double consonant (-TT, -SS)
  - *o - the stem ends with the sequence CVC where the second C is not W, X or Y (-WIL, -HOP)
- The stemmer consists in 7 subsets of rules applied in sequence
  1. Plural nouns and 3rd person verbs
     - SSES → SS (caresses → caress)
     - IES → I (ponies → poni, ties → ti)
     - SS → SS (caress → caress)
     - S → ε (cats → cat)

Porter stemmer - 3

2. Verb past tense and gerund
   - (m>1) EED → EE (agreed → agree, feed → feed)
   - (*v*) ED → ε (computed → comput, red → red)
   - (*v*) ING → ε (monitoring → monitor, sing → sing)

   1. Cleanup – when the last two are used further cleanup rules are used (deletion of double literals – hopp[ing] → hop, insertion of the e at the end f.i. AT → ATE – comput[ed] → compute)
3. (*v*) Y → I
   - happy → happi
4. Morphological derivations I (multiple suffixes)
   - (m>0) ATIONAL → ATE (relational → relate)
   - (m>0) ATOR → ATE (operator → operate)
   - (m>0) BILITI → BLE (sensibiliti → sensible)
   - etc… (20 rules)

Porter stemmer - 4

5. Morphological derivation II (further multiple suffixes)
   - (m>0) ICATE → IC (triplicate → triplic)
   - (m>0) NESS → ε (goodness → good)
   - etc.. (6 rules)
6. Morphological derivations III (single suffixes)
   - (m>1) ENCE → ε (inference → infer)
   - (m>1) EMENT → ε (replacement → replac)
   - (*S or *T) & ION → ε (adoption → adopt)
   - etc.. (18 rules)
7. Cleanup
   - (m>1) E → ε (probate → probat, rate → rate)
   - (m=1 & ! *o) E → ε (cease → ceas)

   - (m>1 & *d*L) → [single letter] (controll → control)

Orthographic errors – correction/detection

- Given the character sequence of a word containing orthographic errors the goal is to compute the list of the candidate corrections
  - It is a problem of probabilistic translation – the misspelled word may correspond to more lexical forms each associated to a probability
    - spone -> spine, sponge, spooned, spoke, shone, stone
  - The surface level form is available with an orthographic error and more lexical forms are compatible with it
- There are three categories of tasks that need to be faced
  - Detection of errors for non-words – the orthographic error yields a word that does not belong to the language (f.i. spone)
  - Error correction for isolated non-words
  - Error detection and correction given the context – We consider also errors that yield correct words (f.i. worm -> warm)

Types of errors

- The types of errors depend on the text generation modality
  - keyboard typing (1%-3% error rate)… but also handwriting
  - OCR for printed or handwritten characters (the error rate is quite variable and depends of the print/write quality)
  - Most of the errors are single (~94% for typed text)
    - insertion – a character is added (change -> chanhge)
    - deletion - a character is deleted (change -> cange)
    - substitution – a character is changed (change -> cjange)
    - transposition – two characters are swapped (change-> cahnge)
  - Most of the algorithms are designed and optimized for single errors
    - Most of the errors are typographical errors due to the use of the keyboard (f.i. substitution of characters corresponding to keys that are close to each other in the keyboard layout)

The noisy channel

- Errors can be modeled as the effect of a noisy channel on an instance of the lexical form yielding a surface level form

  - The model of the noisy channel describe the errors that can be generated
  - A bayesian classification approach can be used

$$correct\text{-}word = \underset{word \in V}{\operatorname{argmax}} \; P(word|observation)$$

    - The word w in the dictionary V that maximizes the probability P(w|o) of generating by the observation o
      - P(w|o) is a model of the errors introduced by the channel

Bayesian classification

- The maximization of P(w|o) requires the computation of P(w|o) for each word in the dictionary
  - A more manageable form is obtained applying the Bayes theorem

$$\hat{w} = \text{argmax}_{w \in V} \frac{P(o|w)P(w)}{P(o)}$$

  - P(w) is the prior probability of the word w and it may be estimated from its frequency in a reference corpus for the language
  - P($o|w$) models the possible variations (the observation o) that can be obtained from a dictionary word w. The implementation of the model requires additional hypotheses
  - P(o) is the prior probability of the observation o, but being a term shared by all the dictionary words it may be neglected when maximizing the probability

$$\hat{w} = \text{argmax}_{w \in V} P(o|w)P(w)$$

Likelihood estimation

- The prior may be estimated on a corpus by applying smoothing techniques to balance the estimate for rare words (some words may not even appear in the corpus)

$$P(w) = \frac{\#w + 0.5}{N + 0.5|V|}$$

  - N is the number of words in the corpus, |V| is the number of words in the dictionary. Each word has a bias value of 0.5 for the counting
- The likelihood p(o|w) may be estimated given a model of the errors (insertion/deletion/substitution/transposition) based on their context
  - The probability of substituting a character with another one can be estimated on a reference corpus with errors. The confusion matrix in the entry [‘a’,’s’] reports the number of times that ‘a’ is substituted by a ‘s’ (Kernighan et al. 1990)
    - ‘a’ and ‘s’ are two adjacent keys on the keyboard

Confusion matrix

- In the system proposed by Kernighan et al. (1990), 4 confusion matrices, computed form examples, are exploited
  - del[*x*,*y*] counts the number of times that the characters *xy* in the correct word have been written as *x*
  - ins[*x*,*y*] counts the number of times that the character *x* in the correct word has been written as *xy*
  - sub[*x*,*y*] is the number of times that *x* has been written as *y*
  - trans[x,y]] is the number of times that *xy* has been written as *yx*
  - With the previous choice insertions/deletions are modeled as dependent on the preceding character
- The matrices can be estimated by an iterative procedure
  - the matrix entries are initialized with the same value
  - the corrections are computed exploiting the current model *o* -> *w*
  - the procedure is iterated using the updated model

Computation of P(*o*|*w*)

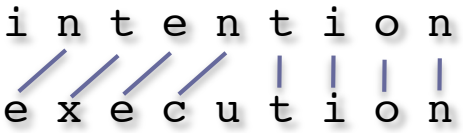
- The Kernighan algorithm is based on the hypothesis of single errors
  - Given an observation *t*, the algorithm finds the dictionary words *c* that can be obtained from *c* with just one error of the considered types
  - The likelihood can be computed as follows

$$P(t|c) = \begin{cases} \frac{\text{del}[c_{p-1}, c_p]}{\#[c_{p-1}, c_p]} & \text{if deleletion of } c_p \\ \frac{\text{ins}[c_{p-1}, t_p]}{\#[c_{p-1}]} & \text{if insertion of } t_p \\ \frac{\text{sub}[t_p, c_p]}{\#[c_p]} & \text{if substitution of } t_p \\ \frac{\text{trans}[c_p, c_{p+1}]}{\#[c_p, c_{p+1}]} & \text{if transposition of } c_p \end{cases}$$

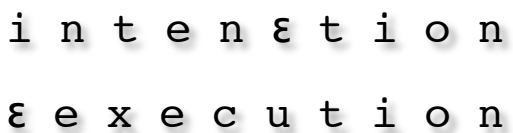
  - where p is the position of the character for which the observed string *t* and the correct one *c* differ

Minimum edit distance

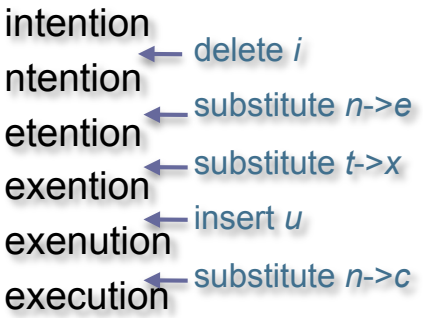
- The general case in which there are no assumptions on the number of errors can be faced by exploiting the string edit distance
  - The bayesian technique can be seen as a method to compute a distance between a pair of strings
    - The "closest" string with respect to the observation is the dictionary string having the highest probability
  - The minimum edit distance is defined as the minimal number of edit operations needed to transform one string into the other



Track



Alignment



List of the operations

Levenshtein distance

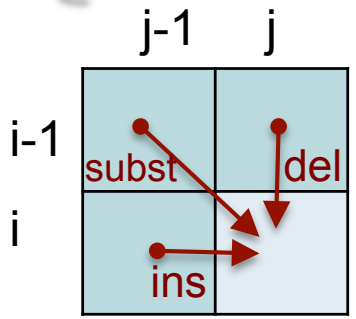
- A cost can be associated to each editing operation
  - For the original Levenshtein distance (1966) each operation has cost 1
    - $d_L(intention,execution) = 5$
  - More complex weighting functions can be defined (f.i. based on the confusion matrix). The most probable alignment is obtained
- The minimum edit distance is computed by the dynamic programming scheme (Bellman 1957)
  - The basic principle is that the optimal sequence of edit operations to map the input string into the output string is optimal also at each intermediate step
  - In fact, if there was a better path up to an intermediate configuration with respect to the one computer for the optimal alignment, it would be possible to replace it in the overall optimal sequence yielding a better value for the total cost (this would be a contradiction)

Edit distance computation

- The computation exploits the distance matrix having a column for each symbol in the goal sequence and one row for each symbol of the input string (edit-distance matrix)

	#	e	x	e	c	u	t	i	o	n
#	0	1	2	3	4	5	6	7	8	9
i	1	1	2	3	4	5	6	6	7	8
n	2	2	2	3	4	5	6	7	7	7
t	3	3	3	3	4	5	5	6	7	8
e	4	3	4	3	4	5	6	6	7	8
n	5	4	4	4	4	5	6	7	7	7
t	6	5	5	5	5	5	5	6	7	8
i	7	6	6	6	6	6	6	5	6	7
o	8	7	7	7	7	7	7	6	5	6
n	9	8	8	8	8	8	8	7	6	5

$$dist[i,j] = \min \begin{cases} dist[i-1,j]+del\text{-}cost(s_i) \\ dist[i-1,j-1]+subst\text{-}cost(s_i,t_j) \\ dist[i,j-1]+ins\text{-}cost(t_j) \end{cases}$$



Distance computation– subst-cost(s)=2

- The substitution cost can be considered equal to 2 (corresponds to consider a substitution as a deletion followed by an insertion)

	#	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

```
intention
entention    subst(i,e)    2
exntention   ins(x)         1
ex□tention   del(n)         1
ex□ention    del(t)         1
exention     subst(e,e)     0
execntion    ins(c)         1
execuntion   ins(u)         1
exec□ution   del(n)         1

        iεnteεεntion
        exεεecuεtion
```

context-sensitive spelling

- The are errors resulting in correct words
  - The error can be detected only taking into account the surrounding words (the context)

    I will arrive in fine [five] minutes

    Did you witch [watch] TV?

    I feel very worm [warm]!

  - The problem can be faced by considering that some combinations of words are less likely than others
  - In general we can think to build a language model that allows the computation of the probability of a given sequence of words
  - The same model can be used to predict the most likely word to be inserted into an incomplete sentence
  - The probabilities can be computed from collections of texts

Word counting & N-grams

- The counting strategy should be well defined
  - Should the punctuation marks be considered?
  - Should character case be considered?
  - Should morphological parsing/stemming be exploited?
- The simplest (inaccurate) language model assumes that each word in a text appears independently on the others
  - We consider p(*word*) estimated as *#word/TotWords* on a significant text corpus
  - The text is modeled as generated by a sequence of independent events
- A more accurate model takes into account the conditional probabilities among adjacent words (bi-grams)
  - We consider $p(word_p|word_{p-1})$ (f.i. *computer network* vs *computer pear*)
  - The model is more accurate but it is more difficult to be estimated with accuracy

N-grams & sequences

- If words at a given position in a text are considered as events, a sentence with *n* words is the joint event $w_1,w_2,\ldots,w_n$ characterized by a probability distribution

$$p(w_1, w_2, \ldots, w_n)$$

  - the probability can be factorized by applying the chain rule

$$p(w_1, w_2, \ldots, w_n) = p(w_1)p(w_2|w_1)p(w_3|w_1w_2)\ldots p(w_n|w_1\ldots w_{n-1})$$

    - the probabilities $p(w_i|w_1..w_{i-1})$ model the generation of a word given the preceding words
    - Without any assumption the expansion is exactly equal to the joint probability (the derivation is based on the definition of conditional probability)
    - The problem is the estimation of $p(w_i|w_1..w_{i-1})$ from observations, especially when i increases

Bi-grams Language Models

- A simplifying assumption is to consider the dependence only between adjacent words
  - With this assumption we only need to estimate the conditional probabilities for bi-grams (pairs of adjacent words)
  - Hence we need to estimate the probabilities $p(w_i|w_{i-1})$
  - Given this assumption it results that

$$p(w_p|w_1..w_{p-1}) = p(w_p|w_{p-1})$$

    - the p-th word in the sequence is independent of all the others given the preceding one
    - The assumption correspond to a Markovian property, i.e. the sequences are generated by a Markov chain whose states are the words
    - actually the p-th words depends on all the preceding ones but through the chain of choices that have been taken at each step of the generation

N-grams Language Models

- The model can be generalized by considering N-grams
  - We assume that a word in the sequence is independent given the preceding N-1 words
  - The assumption corresponds to the following approximation

$$p(w_p|w_1..w_{p-1}) \approx p(w_p|w_{p-N+1}..w_{p-1})$$

  - The sequence probability is approximated as follows

$$p(w_1, w_2, \ldots, w_n) \approx \prod_{k=1}^{n} p(w_k|w_{k-N+1}..w_{k-1})$$

    - where we assumed that the preceding words with index less than 1 are equal to a special symbol “sequence start”, f.i. for bigrams it holds the following

$$p(w_1|w_0) = p(w_1|\#0) = p(w_1)$$

N-gram Language Models- example

- The N-grams model dependencies deriving from
  - grammatical rules (f.i. an adjective is likely to be followed by a noun)
  - semantics restrictions (f.i. eat a pear vs. eat a crowbar)
  - cultural restrictions (es. eat a cat)
- The probabilities depend on the considered context
  - the language use in a restricted specific context can be modeled
  - f.i. Berkely Restaurant Project (Jurafsky et al., 1994)

$p(w|\text{eat})$

eat on	0.16	eat Thai	0.03
eat some	0.06	eat breakfast	0.03
eat lunch	0.06	eat in	0.02
eat dinner	0.05	eat Chinese	0.02
eat at	0.04	eat Mexican	0.02
eat a	0.04	eat tomorrow	0.01
eat indian	0.04	eat dessert	0.007
eat today	0.03	eat British	0.001

Estimation of the models

- The probabilities have values less than 1
  - The combination formula is a product that tends to become quite small with potential underflow problems
  - In general the computation is performed using the (logprob)

$$\log p(w_1, w_2, \ldots, w_n) = \sum_{k=1}^{n} \log p(w_k | w_{k-N+1} .. w_{k-1})$$

- The model estimation can be obtained by counting the N-grams and by normalizing the counts on a reference corpus
  - f.i. for bigrams the relative frequency is computed as follows

$$p(w_p | w_{p-1}) \approx \frac{\#(w_{p-1} w_p)}{\sum_w \#(w_{p-1} w)} = \frac{\#(w_{p-1} w_p)}{\#w_{p-1}}$$

  - the estimate of the relative frequency is an example of Maximum Likelihood Estimation (Max P(*corpus*|M))

Limits of N-grams based LMs

- The model accuracy increases with N
  - The syntactic/semantic contexts are better modeled
  - The drawback is the difficulty in the model parameter estimation (the conditional probabilities)
  - If the dictionary contains D terms (word forms with inflections) there are D^N N-grams
    - A corpus C words "long" contains C N-grams (each word generates exactly a sample for one N-gram)
    - For a significant estimate of the parameters, the corpus size should increase exponentially in the order N of N-grams
    - f.i. given D=30000 there are 900 millions bigrams and a corpus with C=1.000.000 words would not be adequate to compute an accurate estimate for the language (especially for the most rare bigrams)
    - Hence, the resulting model can be heavily dependent on the corpus exploited in the estimation of the parameters

Smoothing

- One of the problems in the estimation of the N-gram probabilities on a corpus with limited size is that some of the "valid" N-grams might not appear at all
  - The matrix for the computation of the bigram frequencies is very sparse in general
  - Bigrams with 0 occurrences would be considered with 0 probability
  - Analogously the estimate of the probabilities for the N-grams with low frequencies is not very accurate
  - Smoothing techniques can be applied to give a better estimate for the N-grams with low (zero) probability
  - In general these techniques apply a fix on the counts given some criteria

Smoothing – add-one

- The most simple technique is the add-one
  - 1 is added to all counts
  - It is simple but its performances are not good
  - f.i. for the estimate of the unigram (single words) probability estimate

$$p(w) \approx \frac{\#w + 1}{N + |D|} \qquad p(w) \approx \frac{\#w^*}{N},\ \#w^* = (\#w + 1)\frac{N}{N + |D|}$$

    - $\#w^*$ is the count with the add-one "fix"
  - For bigrams the estimate is fixed as follows

$$p(w_p|w_{p-1}) \approx \frac{\#(w_{p-1}w_p) + 1}{\#w_{p-1} + |D|}$$

    - in this case the variation may be significant since |D| is usually much greater than $\#w_{p-1}$ (all the bigrams are considered valid)

Smoothing – Witten-Bell Discounting 1

- It is based on the idea that a non observed event is an event that has not occurred yet (but it can)
  - the probability of the occurrence of an N-gram with null frequency is modeled with the probability of observing an N-gram for the first time
    - The count T of the events observed only one time is used to estimate the count of those that have never been observed
    - if many events have been observed it is likely that many events will be observed in the future
  - The number of N-gram observed for the first time is exactly the number of different N-grams observed in the corpus
  - The total probability of the non-observed N-grams - p(*new*) – is estimated by

$$\sum_{i:\#c_i=0} p_i^* = \frac{T}{T+N}$$

T is the number of observed new events

N is the number of observed events

Smoothing – Witten-Bell Discounting 2

- If we assume that all the non-observed event have the same probability

$$p_i^* = \frac{T}{Z(T+N)} \quad Z = \sum_{i:c_i=0} 1$$

  - The probabilities of the observed events should be coherently reduced

$$p_i^* = \frac{c_i}{T+N} \quad \text{if } c_i > 0$$

- For bigrams (N-grams) the considered counts are conditioned with respect to the context
  - The considered counts are T(w_x) – number of new types of observed bigrams where w_x is the preceding word – and N(w_x) – total number of bigrams that have w_x as preceding word

Smoothing – Witten-Bell Discounting 3

- The total probability of the non-observed bigrams related to w_x is

$$\sum_{i:\#(w_x w_i)=0} p^*(w_i|w_x) = \frac{T(w_x)}{N(w_x)+T(w_x)}$$

  - By a uniform distribution of this probability on all the non-observed bigrams it can be obtained

$$p^*(w_i|w_{i-1}) = \frac{T(w_{i-1})}{Z(w_{i-1})(N(w_{i-1})+T(w_{i-1}))} \quad \text{if } (\#(w_{i-1}w_i)=0)$$

  - For bigrams having non-null frequency the probability is computed as follows

$$p^*(w_i|w_{i-1}) = \frac{\#(w_{i-1}w_i)}{\#w_{i-1}+T(w_{i-1})} \quad \text{if } (\#(w_{i-1}w_i)>0)$$

Smoothing – Good-Turing

- The probability of the N-grams with higher counts is redistributed to the N-grams with null (or low) counts
  - N_c is the number of N-grams occurring c times ($N_0$is the number of bigrams appearing 0 times, N_1 those with counts equal to 1,..)
  - The Good-Turing estimate modifies the counts such that

$$c^* = (c+1)\frac{N_{c+1}}{N_c}$$

    - f.i. the count for non-observed bigrams (c=0) is estimated as the number of bigrams observed just once divided by the number of the never observed bigrams (the $|D|^2$ valid bigrams except the observed ones)
    - in general it is applied only for $c \leq k$ (f.i. k=5) that is only the counts above a certain threshold are considered reliable

$$c^* = \frac{(c+1)\frac{N_{c+1}}{N_c} - c\frac{(k+1)N_{k+1}}{N_1}}{1 - \frac{(k+1)N_{k+1}}{N_1}} \quad 1 \leq c \leq k$$

Backoff

- If there are no observations for a given N-gram, its statistics can be estimated exploiting the correlated N-1-grams
  - to estimate the probability P($w_p|w_{p-1}w_{p-2}$), we can exploit the knowledge on P($w_p|w_{p-1}$)
  - to estimate the probability of the bigram P($w_p|w_{p-1}$, we can exploit the available information on the unigram P(w_p)
- The backoff techniques exploit the hierarchy among the N-grams to improve the estimates
  - f.i. for trigrams the following estimate can be exploited

$$\hat{p}(w_i|w_{i-2}w_{i-1}) = \begin{cases} p(w_i|w_{i-2}w_{i-1}) & \text{if } \#(w_{i-2}w_{i-1}w_i) > 0 \\ \alpha_1 p(w_i|w_{i-1}) & \text{if } \#(w_{i-2}w_{i-1}w_i) = 0 \ , \#(w_{i-1}w_i) > 0 \\ \alpha_2 p(w_i) & \text{otherwise} \end{cases}$$

backoff & discounting

- The smoothing/discounting techniques "extract" part of the probability from the observed events to distribute it (uniformly) to non-observed events
  - A smoothing technique defining the amount of probability to be assigned to non-observed events can be combined with the backoff method giving a modality to distribute the probability among the considered events
  - The backoff assigns a non null probability to events that with the MLE estimate would have a zero probability. This added probability is to be divided among all the events
  - The general probability estimate can be computed as follows

$$\hat{p}(w_n|w_{n-N+1}..w_{i-1}) = \tilde{p}(w_n|w_{n-N+1}..w_{i-1}) + \theta(p(w_n|w_{n-N+1}..w_{i-1}))\cdot \cdot\alpha(w_{n-N+1}..w_{i-1})\hat{p}(w_n|w_{n-N+2}..w_{i-1})$$

    - where $\theta(p) = 0$ if $p>0$ and $\theta(p) = 1$ is $p=0$ selects/deselects the backoff

Backoff & discounting - computation 1

- The probabilities of the non null N-grams are estimated as

$$\tilde{p}(w_n|w_{n-N+1}..w_{n-1}) = \frac{\#^*(w_{n-N+1}..w_{n-1}w_n)}{\#(w_{n-N+1}..w_{n-1})} < \frac{\#(w_{n-N+1}..w_{n-1}w_n)}{\#(w_{n-N+1}..w_{n-1})}$$

  - where the count with discount $\#^*$ allows the transfer of the probability to the non observed events
  - The amount of probability to be transferred from a given context that represents a (N-1)-gram is

$$\beta(w_{n-N+1}..w_{n-1}) = 1 - \sum_{w_n:\#(w_{n-N+1}..w_n)>0} \tilde{p}(w_n|w_{n-N+1}..w_{n-1})$$

  - Each (N-1)-gram will receive only a fraction of β

Backoff & discounting - computation 2

- The probabilities of the (N-1)-grams are computed from β by normalizing with respect to the total assigned probability
  - the value of the backoff coefficients α is derived

$$\alpha(w_{n-N+1}..w_{n-1}) = \frac{1 - \sum_{w_n:\#(w_{n-N+1}..w_n)>0} \tilde{p}(w_n|w_{n-N+1}..w_{n-1})}{1 - \sum_{w_n:\#(w_{n-N+1}..w_n)>0} \tilde{p}(w_n|w_{n-N+2}..w_{n-1})}$$

  - the amount of discount applied to each N-gram and the fraction redistributed to the (N-1)-grams are peculiar of each N-gram

  - Summarizing for trigrams the backoff computation is the following

$$\hat{p}(w_i|w_{i-2}w_{i-1}) = \begin{cases} \tilde{p}(w_i|w_{i-2}w_{i-1}) & \text{if } \#(w_{i-2}w_{i-1}w_i) > 0 \\ \alpha(w_{i-2}w_{i-1})\tilde{p}(w_i|w_{i-1}) & \text{if } \#(w_{i-2}w_{i-1}w_i) = 0 \text{ , } \#(w_{i-1}w_i) > 0 \\ \alpha(w_{i-1})\tilde{p}(w_i) & \text{otherwise} \end{cases}$$