Probability; Agreement and Types in Natural Language

Informatics 2A: Lecture 23

Adam Lopez

10 November 2016

1. Probabilities in PCFGs
2. Agreement phenomena
3. Types in Semantics
   - Types of entities
   - Subtypes in NL
   - Types as selectional restrictions

Reading:

J&M 2nd edition, ch. 14.2–14.6,
NLTK Book, Chapter 8, final section on Weighted Grammar.

Parsing in the News

(All over tech news outlets, May 2016)

Example PCFG

$S \rightarrow NP\ VP$	[.80]	$Det \rightarrow the$	[.10]
$S \rightarrow Aux\ NP\ VP$	[.15]	$Det \rightarrow a$	[.90]
$S \rightarrow VP$	[.05]	$Noun \rightarrow book$	[.10]
$NP \rightarrow Pronoun$	[.35]	$Noun \rightarrow flight$	[.30]
$NP \rightarrow Proper\text{-}Noun$	[.30]	$Noun \rightarrow dinner$	[.60]
$NP \rightarrow Det\ Nominal$	[.20]	$Proper\text{-}Noun \rightarrow Houston$	[.60]
$NP \rightarrow Nominal$	[.15]	$Proper\text{-}Noun \rightarrow NWA$	[.40]
$Nominal \rightarrow Noun$	[.75]	$Aux \rightarrow does$	[.60]
$Nominal \rightarrow Nominal\ Noun$	[.05]	$Aux \rightarrow can$	[.40]
$VP \rightarrow Verb$	[.35]	$Verb \rightarrow book$	[.30]
$VP \rightarrow Verb\ NP$	[.20]	$Verb \rightarrow include$	[.30]
$VP \rightarrow Verb\ NP\ PP$	[.10]	$Verb \rightarrow prefer$	[.20]
$VP \rightarrow Verb\ PP$	[.15]	$Verb \rightarrow sleep$	[.20]

Where do the probabilities come from?

Parameter Estimation

In a PCFG every rule is associated with a probability.
But where do these rule probabilities come from?

Use a large parsed corpus such as the Penn Treebank.

```
( (S
    (NP-SBJ (DT That) (JJ cold)
      (, ,)
      (JJ empty) (NN sky) )
    (VP (VBD was)
      (ADJP-PRD (JJ full)
        (PP (IN of)
          (NP (NN fire)
            (CC and)
            (NN light) ))))
    (. .) ))
```

$S \rightarrow NP\text{-}SBJ\ VP$

$VP \rightarrow VBD\ ADJP\text{-}PRD$

$PP \rightarrow IN\ NP$

$NP \rightarrow NN\ CC\ NN$

etc.

Parameter Estimation

Corpus of parsed sentences:

'S1: [S [NP grass] [VP grows]]'
'S2: [S [NP grass] [VP grows] [AP slowly]]'
'S3: [S [NP grass] [VP grows] [AP fast]]'
'S4: [S [NP bananas] [VP grow]]'

Construct PCFG:

r	Rule	α	Count
$r1$	S $\rightarrow$ NP VP	S	2
$r2$	S $\rightarrow$ NP VP AP	S	2

What should be the probability for $r1$?

1. 1
2. $1/2$
3. $1/4$

Parameter Estimation: Intuition

Suppose that we have a bag containing two types of marbles: red and black. How would you estimate the ratio of red to black marbles in the bag?

More precisely, what is $p(red)$? (Note: $p(black) = 1 - p(red)$).

Experiment. Draw ten marbles from the bag (replacing them each time). Suppose you draw 7 red and 3 black marbles. What is $p(red)$?

1. .3
2. .5
3. .7
4. 1

Why?

Parameter Estimation: Maximum Likelihood

Since we saw 7 red and 3 black marbles, we can write the likelihood of the observed data in terms of the unknown parameter $p(red)$:

$$p(data) = p(red)^7 \times (1 - p(red))^3 \qquad (1)$$

$p(red)$ is unknown. What's a reasonable way to set it?

How about this?

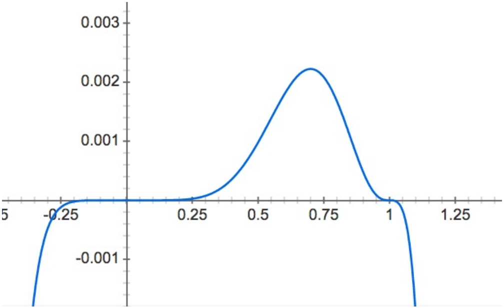
$$\arg\max_{p(red) \in [0,1]} p(data) = p(red)^7 \times (1 - p(red))^3 \qquad (2)$$

Parameter Estimation: Maximum Likelihood

Now we have a basic calculus problem. Solve:

$$\arg \max_{p(red) \in [0,1]} p(data) = p(red)^7 \times (1 - p(red))^3 \tag{3}$$

What $p(data)$ looks like:

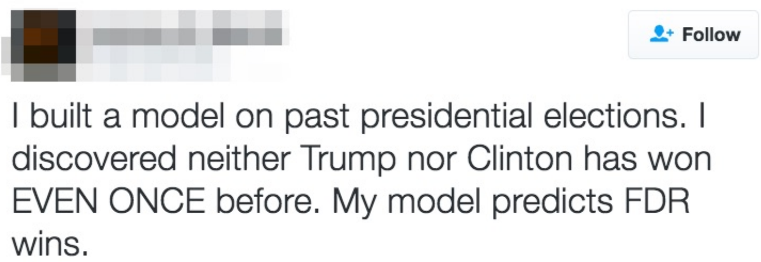


Maximum Likelihood Estimation

MLE is one of the most basic parameter estimation methods. When you have lots of data, it's a reasonable first choice.

What are some cases where it might not work?

Question. What if you *don't* have lots of data (for the parameter you want to estimate)?



Parameter Estimation

Corpus of parsed sentences:

'S1: [S [NP grass] [VP grows]]'
'S2: [S [NP grass] [VP grows] [AP slowly]]'
'S3: [S [NP grass] [VP grows] [AP fast]]'
'S4: [S [NP bananas] [VP grow]]'

Compute PCFG probabilities:

r	Rule	α	$P(r\|\alpha)$
$r1$	S $\rightarrow$ NP VP	S	$2/4$
$r2$	S $\rightarrow$ NP VP AP	S	$2/4$
$r3$	NP $\rightarrow$ grass	NP	$3/4$
$r4$	NP $\rightarrow$ bananas	NP	$1/4$
$r5$	VP $\rightarrow$ grows	VP	$3/4$
$r6$	VP $\rightarrow$ grow	VP	$1/4$
$r7$	AP $\rightarrow$ fast	AP	$1/2$
$r8$	AP $\rightarrow$ slowly	AP	$1/2$

Parameter Estimation

In a PCFG every rule is associated with a probability.
But where do these rule probabilities come from?

Use a large parsed corpus such as the Penn Treebank.

- Obtain grammar rules by reading them off the trees.
- Calculate number of times LHS $\rightarrow$ RHS occurs over number of times LHS occurs.

$$P(\alpha \rightarrow \beta | \alpha) = \frac{\text{Count}(\alpha \rightarrow \beta)}{\sum_{\gamma} \text{Count}(\alpha \rightarrow \gamma)} = \frac{\text{Count}(\alpha \rightarrow \beta)}{\text{Count}(\alpha)}$$

Parameter Estimation

With these parameters (rule probabilities), we can now compute the probabilities of the four sentences S1–S4:

$$\begin{aligned} P(S1) &= P(r1|S)P(r3|NP)P(r5|VP) \\ &= 2/4 \cdot 3/4 \cdot 3/4 = 0.28125 \end{aligned}$$

$$\begin{aligned} P(S2) &= P(r2|S)P(r3|NP)P(r5|VP)P(r7|AP) \\ &= 2/4 \cdot 3/4 \cdot 3/4 \cdot 1/2 = 0.140625 \end{aligned}$$

$$\begin{aligned} P(S3) &= P(r2|S)P(r3|NP)P(r5|VP)P(r7|AP) \\ &= 2/4 \cdot 3/4 \cdot 3/4 \cdot 1/2 = 0.140625 \end{aligned}$$

$$\begin{aligned} P(S4) &= P(r1|S)P(r4|NP)P(r6|VP) \\ &= 2/4 \cdot 1/4 \cdot 1/4 = 0.03125 \end{aligned}$$

Motivation behind such estimation

One criterion for finding rule weights of a PCFG (or parameters in general) is the *maximum likelihood* criterion.

It means we want to find rule weights which make the treebank we observe most likely if we multiply in all probabilities together (we assume the trees are independent)

Counting and normalising satisfies this criterion

Type-related phenomena in NL

- We've met the concept of types in programming languages, along with the idea of typing constraints on programs.
- Types also play a variety of roles in NL: e.g.
  - for disambiguation (via selectional restrictions),
  - for NL semantics (as in upcoming lecture).
- Furthermore, some phenomena that would be typically handled via types in a PL context (notably agreement) are often handled in other ways in NL.

We'll briefly survey this material in this lecture.

Agreement phenomena

In PLs, typing rules enforce type agreement between different (often separated) constituents of a program:

```
int i=0; ...; if (i>2) ...
```

There are somewhat similar phenomena in NL: constituents of a sentence (often separated) may be constrained to agree on an attribute such as person, number, gender.

- You, I imagine, are unable to attend.
- The hills are looking lovely today, aren't they?
- He came very close to injuring himself.

Agreement in various languages

These examples illustrate that in English:

- Verbs agree in person and number with their subjects;
- Tag questions agree in person, number, tense and mode with their main statement, and have the opposite polarity.
- Reflexive pronouns follow suit in person, number and *gender*.

French has much more by way of agreement phenomena:

- Adjectives agree with their head noun in gender and number.

  Le petit chien, La petite souris, Les petites mouches

- Participles of *être* verbs agree with their subject:

  Il est arrivé, Elles sont arrivées

- Participles of other verbs agree with preceding direct objects:

  Il a vu la femme, Il l'a vue

How can we capture these kinds of constraints in a grammar?

Agreement rules: why bother?

Modelling agreement is obviously important if we want to generate grammatically correct NL text.

But even for understanding input text, agreement can be useful for resolving ambiguity.
E.g. the following sentence is ambiguous . . .

The boy who eats flies ducks.

. . . whilst the following are less so:

The boys who eat fly ducks.
The boys who eat flies duck.

Node-splitting via attributes

One solution is to refine our grammar by splitting certain non-terminals according to various *attributes*. Examples of attributes and their associated values are:

- Person: 1st, 2nd, 3rd
- Number: singular, plural
- Gender: masculine, feminine, neuter
- Case: nominative, accusative, dative, . . .
- Tense: present, past, future, . . .

In principle these are language-specific, though certain common patterns recur in many languages.

We can then split phrase categories as the language demands, e.g.

- Split NP on person, number, case (e.g. NP[3,sg,nom]),
- Split VP on person, number, tense (e.g. VP[3,sg,fut]).

Parameterized CFG productions

We can often use such attributes to enforce agreement constraints.
This works because of the head phrase structure typical of NLs.
E.g. we may write parameterized rules such as:

$$\begin{array}{rcl} \mathrm{S} & \rightarrow & \mathrm{NP[p,n,nom]\ VP[p,n]} \\ \mathrm{NP[3,n,c]} & \rightarrow & \mathrm{Det[n]\ Nom[n]} \end{array}$$

Each of these really abbreviates a finite number of rules obtained by specializing the attribute variables. (Still a CFG!)
When specializing, each variable must take the same value everywhere, e.g.

$$\begin{array}{rcl} \mathrm{S} & \rightarrow & \mathrm{NP[3,sg,nom]\ VP[3,sg]} \\ \mathrm{S} & \rightarrow & \mathrm{NP[1,pl,nom]\ VP[1,pl]} \\ \mathrm{NP[3,pl,acc]} & \rightarrow & \mathrm{Det[pl]\ Nom[pl]} \end{array}$$

Parsing algorithms can be adapted to work with this machinery: don't have to 'build' all the specialized rules individually.

S	$\rightarrow$	NP[p,n,nom] VP[p,n]
NP[p,n,c]	$\rightarrow$	Pro[p,n,c]
Pro[1,sg,nom]	$\rightarrow$	*I*, etc.
Pro[1,sg,acc]	$\rightarrow$	*me*, etc.
NP[3,n,c]	$\rightarrow$	Det[n] Nom[n] RelOpt[n]
Nom[n]	$\rightarrow$	N[n] | Adj Nom[n]
N[sg]	$\rightarrow$	*person*, etc.
N[pl]	$\rightarrow$	*people*, etc.
RelOpt[n]	$\rightarrow$	ϵ | *who* VP[3,n]
VP[p,n]	$\rightarrow$	VV[p,n] NP[p',n',acc]
VV[p,n]	$\rightarrow$	V[p,n] | BE[p,n] VG
V[3,sg]	$\rightarrow$	*teaches*, etc.
BE[p,n]	$\rightarrow$	*is*, etc.
VG	$\rightarrow$	*teaching*, etc.

(Other rules omitted.)

Some disadvantages of rule splitting for agreement

There is a huge proliferation of primitive grammatical categories

Example

Non3sgVPto, NPmass, 3sgNP, Non3sgAux, ...

This leads to a large number of grammar rules and a loss of generality in the grammar

A fix: constraint-based representation scheme based on unification

Feature Structures

Defined in terms of attribute-value matrices (AVMs):

$$\begin{bmatrix} \textsf{subj} & \begin{bmatrix} \textsf{pers} & \textsf{3} \\ \textsf{num} & \textsf{sg} \\ \textsf{gend} & \textsf{masc} \\ \textsf{pred} & \text{pro} \end{bmatrix} \\ \textsf{pred} & \text{eat}\langle\text{SUBJ, OBJ}\rangle \\ \textsf{obj} & \begin{bmatrix} \textsf{pers} & \textsf{3} \\ \textsf{num} & \textsf{pl} \\ \textsf{gend} & \textsf{fem} \\ \textsf{pred} & \text{pro} \end{bmatrix} \end{bmatrix}$$

Nested set of attributes

How to use Feature Structures?

Each lexical rule is attached with a lexical AVM

Example

Det $\rightarrow$ this	[Det AGREEMENT [NUMBER Sg]]
Det $\rightarrow$ these	[Det AGREEMENT [NUMBER Pl]]
Aux $\rightarrow$ do	[Det AGREEMENT [NUMBER Pl, PERSON 3rd]]
Aux $\rightarrow$ does	[Det AGREEMENT [NUMBER Sg, PERSON 3rd]]

Grammar rules are specified with constraints and copying instructions

Example

VP $\rightarrow$ Verb NP	[VP AGREEMENT] = [Verb AGREEMENT]
NP $\rightarrow$ Det Nominal	[NP HEAD] = [Nominal Head]
	[Det HEAD AGREEMENT]
	= [Nominal HEAD AGREEMENT]

Parsing with Feature Structures

Whenever phrases are conjoined, for example, two phrases in the CYK parser, we do *feature unification*.

This means we check whether we satisfy the constraints attached to the rule, and if so, when we create the new phrase, we also create a new feature structure for it

Unification is not a trivial algorithm because the attribute-values can be *shared* in an AVM, using pointers

Example of Shared Attributes

$$\begin{bmatrix} \text{CAT} & \text{VP} \\ \text{AGREEMENT} & \boxed{1} \\ \text{HEAD} & \begin{bmatrix} \text{AGREEMENT} \ \boxed{1} \begin{bmatrix} \text{PERSON} & 3 \\ \text{NUMBER} & \text{SG} \\ \text{GENDER} & \text{MASC} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

The $\boxed{1}$ refers to the same sub-AVM

Types in Natural Language Semantics

Types are also very useful if we wish to describe the semantics (i.e., meaning) of natural languages. For example, we can use types employed in logic to model the meanings of various phrase types.

Basic Types

1. e — the type of real-world *entities* such as Inf2a, Stuart, John.
2. t — the type of *facts with truth value* like 'Inf2a is amusing'.

From these two basic types, we may construct more complex types via the function type constructor.

From basic to complex formal types

Where PL people write $\sigma \to \tau$, NL people often write $< \sigma, \tau >$. E.g.:

- $<e,t>$: **unary predicates** – functions from entities to facts.
- $<e, <e,t>>$: **binary predicates** – functions from entities to unary predicates.
- $<<e,t>, t>$: **type-raised entities** – functions from unary predicates to truth values.

- Inf2a, Stuart **:** e
- enjoys **:** $<e, <e,t>>$
- enjoys Inf2a, is amusing **:** $<e,t>$
- Inf2a is amusing, Stuart enjoys Inf2a **:** t
- every student **:** $<<e,t>, t>$

This simple system of types will be enough to be going on with (see Lecture 24). But for more precise semantic modelling, a much richer type system is desirable.

Different types of entities in NL

We can distinguish those entities we can count and those we can't:

- ◦ A student kept **a chicken** in her room.
- ◦ A student kept **two chickens** in her room.
- ◦ I ate **rice** and drank **milk**.
- ◦ *I ate **two rices** and drank **two milks**.

- individuals (things we can count): one student, two students, one chicken, many chickens, one room, many rooms
- mass (things we can't count): rice, milk

Subtypes in NL

```
hamburger <: sandwich <: food item <: food
<: substance <: matter <: physical entity <: entity
```

- To deal with meanings in NL, more fine-grained classifications (of varying levels of specificity) are often useful.
- There are also many other more abstract types of entities to which a NL expression may refer: e.g., locations, points in time, time spans, events, beliefs, desires, possibilities, . . .
- This leads to a vast system of subtypes capturing information about real-world concepts and their relationships.
  (Cf. the **WordNet** database.)

Selectional restrictions

We can often characterize verbs and other predicates in terms of their selectional restrictions — constraints on the type of entities or expression can serve as their arguments.

- I want to eat somewhere close to Appleton Tower.
- I want to eat something close to Thai food.

How do we know that *Thai food* is the object of the eating event in the second sentence, and that *somewhere close to AT* is the location of the eating event in the first?

- The object of eating is usually something *edible*: Its semantic type is *edible things*.
- The location of an event is usually a *place*: Its semantic type is *location*.

Selectional restrictions

Selectional restrictions are associated with word senses, not words:

- Do any international airlines serve vegan meals?
  (ie, *provide food or drink*)
- Do any international airlines serve Edinburgh?
- (ie, *provide a service*)
- ?? Do any international airlines serve Edinburgh and vegan meals?

Selectional restrictions vary in their specificity:

OBJECT(imagine): a situation
OBJECT(diagonalise): a matrix

⇒ Verbs vary in the specificity of their argument types.

Selectional restrictions and type coercion

Selectional restrictions can change the way we interpret a term:

- Jane Austen wrote ‘Emma’.
- I used to read Jane Austen a lot.

- The chicken was domesticated in Asia.
- The chicken was overcooked.

Metonymy is when the referent of a term changes to a related entity, often associated with the demands of a verb's selectional restrictions.

Summary

- Many agreement phenomena in NL can be modelled using CFGs with attributes.
- Type systems are also useful in semantic modelling.
- To capture selectional restrictions associated with verb arguments, a very rich system of subtypes is desirable.
- Type coercion is common in Natural Language: changing the type (and often the referent) of an expression to one that fits the verb (predicate) to which it serves as an argument.