Introduction to Data Science

Natural Language Processing

Outline for this Evening

- N-grams
- Grammars
- Parsing
- Dependencies

Natural Language Processing (NLP)

- In a recent survey (KDNuggets blog) of data scientists, 62% reported working "mostly or entirely" with data about people. Much of this data is text.

- In the suggested CS194-16 projects (a random sample of data science projects around campus), nearly half involve natural language text processing.

- NLP is a central part of mining large datasets.

Natural Language Processing

Some basic terms:

- Syntax: the allowable structures in the language: sentences, phrases, affixes (-ing, -ed, -ment, etc.).
- Semantics: the meaning(s) of texts in the language.
- Part-of-Speech (POS): the category of a word (noun, verb, preposition etc.).
- Bag-of-words (BoW): a featurization that uses a vector of word counts (or binary) ignoring order.
- N-gram: for a fixed, small N (2-5 is common), an n-gram is a consecutive sequence of words in a text.

Bag of words Featurization

Assuming we have a dictionary mapping words to a unique integer id, a bag-of-words featurization of a sentence could look like this:

Sentence:	The	cat	sat	on	the	mat
word id’s:	1	12	5	3	1	14

The BoW featurization would be the vector:

Vector	2,	0,	1,	0,	1,	0,	0,	0,	0,	0,	0,	1,	0,	1
position	1		3		5							12		14

In practice this would be stored as a sparse vector of (id, count)s:

(1,2),(3,1),(5,1),(12,1),(14,1)

Note that the original word order is lost, replaced by the order of id’s.

N-grams

Because word order is lost, the sentence meaning is weakened. This sentence has quite a different meaning but the same BoW vector:

Sentence:	The	mat	sat	on	the	cat
word id s:	1	14	5	3	1	12

BoW featurization:

Vector 2, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1

But word order is important, especially the order of nearby words.

N-grams capture this, by modeling tuples of consecutive words.

N-grams

Sentence: The cat sat on the mat

2-grams: the-cat, cat-sat, sat-on, on-the, the-mat

Notice how even these short n-grams “make sense” as linguistic units. For the other sentence we would have different features:

Sentence: The mat sat on the cat

2-grams: the-mat, mat-sat, sat-on, on-the, the-cat

We can go still further and construct 3-grams:

Sentence: The cat sat on the mat

3-grams: the-cat-sat, cat-sat-on, sat-on-the, on-the-mat

Which capture still more of the meaning:

Sentence: The mat sat on the cat

3-grams: the-mat-sat, mat-sat-on, sat-on-the, on-the-cat

N-grams Features

Typically, its advantages to use multiple n-gram features in machine learning models with text, e.g.

unigrams + bigrams (2-grams) + trigrams (3-grams).

The unigrams have higher counts and are able to detect influences that are weak, while bigrams and trigrams capture strong influences that are more specific.

e.g. “the white house” will generally have very different influences from the sum of influences of “the”, “white”, “house”.

N-grams size

N-grams pose some challenges in feature set size.

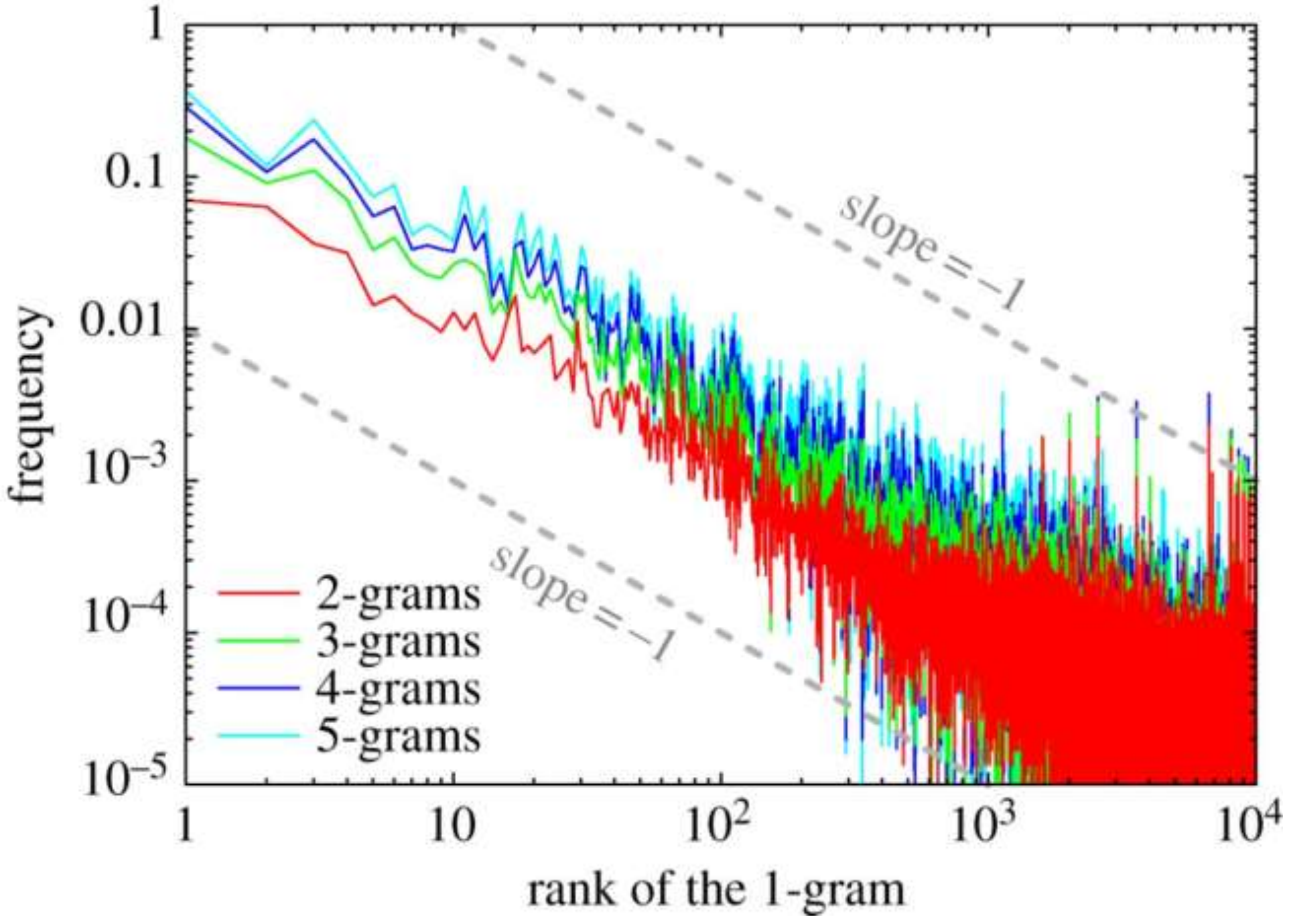
If the original vocabulary size is $|V|$, the number of 2-grams is $|V|^2$

While for 3-grams it is $|V|^3$

Luckily natural language n-grams (including single words) have a **power law** frequency structure. This means that most of the n-grams you see are common. A dictionary that contains the most common n-grams will cover most of the n-grams you see.

Power laws for N-grams

N-grams follow a power law distribution:



N-grams size

Because of this you may see values like this:

- Unigram dictionary size: 40,000
- Bigram dictionary size: 100,000
- Trigram dictionary size: 300,000

With coverage of > 80% of the features occurring in the text.

N-gram Language Models

N-grams can be used to build statistical models of texts.

When this is done, they are called n-gram language models.

An n-gram language model associates a probability with each n-gram, such that the sum over all n-grams (for fixed n) is 1.

You can then determine the overall likelihood of a particular sentence:

The cat sat on the mat

Is much more likely than

The mat sat on the cat

Parts of Speech

Thrax’s original list (c. 100 B.C):

- Noun
- Verb
- Pronoun
- Preposition
- Adverb
- Conjunction
- Participle
- Article

Parts of Speech

Thrax’s original list (c. 100 B.C):

- Noun (boat, plane, Obama)
- Verb (goes, spun, hunted)
- Pronoun (She, Her)
- Preposition (in, on)
- Adverb (quietly, then)
- Conjunction (and, but)
- Participle (eaten, running)
- Article (the, a)

Parts of Speech (Penn Treebank 2014)

1.	CC	Coordinating conjunction
2.	CD	Cardinal number
3.	DT	Determiner
4.	EX	Existential *there*
5.	FW	Foreign word
6.	IN	Preposition or subordinating conjunction
7.	JJ	Adjective
8.	JJR	Adjective, comparative
9.	JJS	Adjective, superlative
10.	LS	List item marker
11.	MD	Modal
12.	NN	Noun, singular or mass
13.	NNS	Noun, plural
14.	NNP	Proper noun, singular
15.	NNPS	Proper noun, plural
16.	PDT	Predeterminer
17.	POS	Possessive ending
18.	PRP	Personal pronoun

19.	PRP$	Possessive pronoun
20.	RB	Adverb
21.	RBR	Adverb, comparative
22.	RBS	Adverb, superlative
23.	RP	Particle
24.	SYM	Symbol
25.	TO	*to*
26.	UH	Interjection
27.	VB	Verb, base form
28.	VBD	Verb, past tense
29.	VBG	Verb, gerund or present participle
30.	VBN	Verb, past participle
31.	VBP	Verb, non-3rd person singular present
32.	VBZ	Verb, 3rd person singular present
33.	WDT	Wh-determiner
34.	WP	Wh-pronoun
35.	WP$	Possessive wh-pronoun
36.	WRB	Wh-adverb

Grammars

Grammars comprise rules that specify acceptable sentences in the language: (S is the sentence or root node)

- S → NP VP
- S → NP VP PP
- NP → DT NN
- VP → VB NP
- VP → VBD
- PP → IN NP
- DT → “the”
- NN → “mat”, “cat”
- VBD → “sat”
- IN → “on”

Grammars

Grammars comprise rules that specify acceptable sentences in the language: (S is the sentence or root node) “the cat sat on the mat”

- S → NP VP
- S → NP VP PP (the cat) (sat) (on the mat)
- NP → DT NN (the cat), (the mat)
- VP → VB NP
- VP → VBD
- PP → IN NP
- DT → “the”
- NN → “mat”, “cat”
- VBD → “sat”
- IN → “on”

Grammars

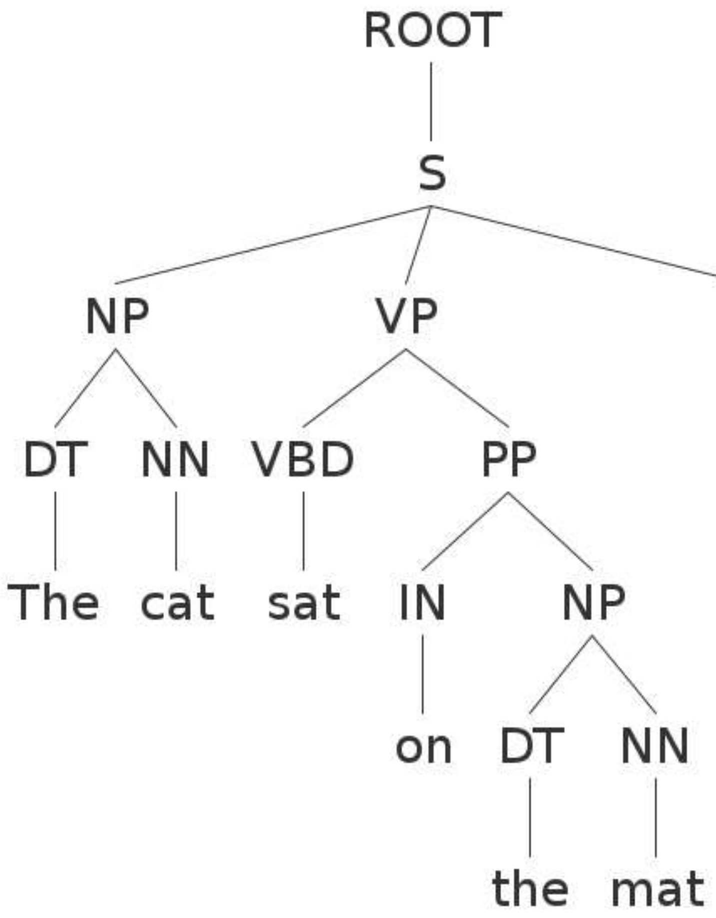
English Grammars are context-free: the productions do not depend on any words before or after the production.

The reconstruction of a sequence of grammar productions from a sentence is called “parsing” the sentence.

It is most conveniently represented as a tree:

Parse Trees

"The cat sat on the mat"



Parse Trees

In bracket notation:

```
(ROOT
  (S
    (NP (DT the) (NN cat))
    (VP (VBD sat)
      (PP (IN on)
        (NP (DT the) (NN mat))))))
```

Grammars

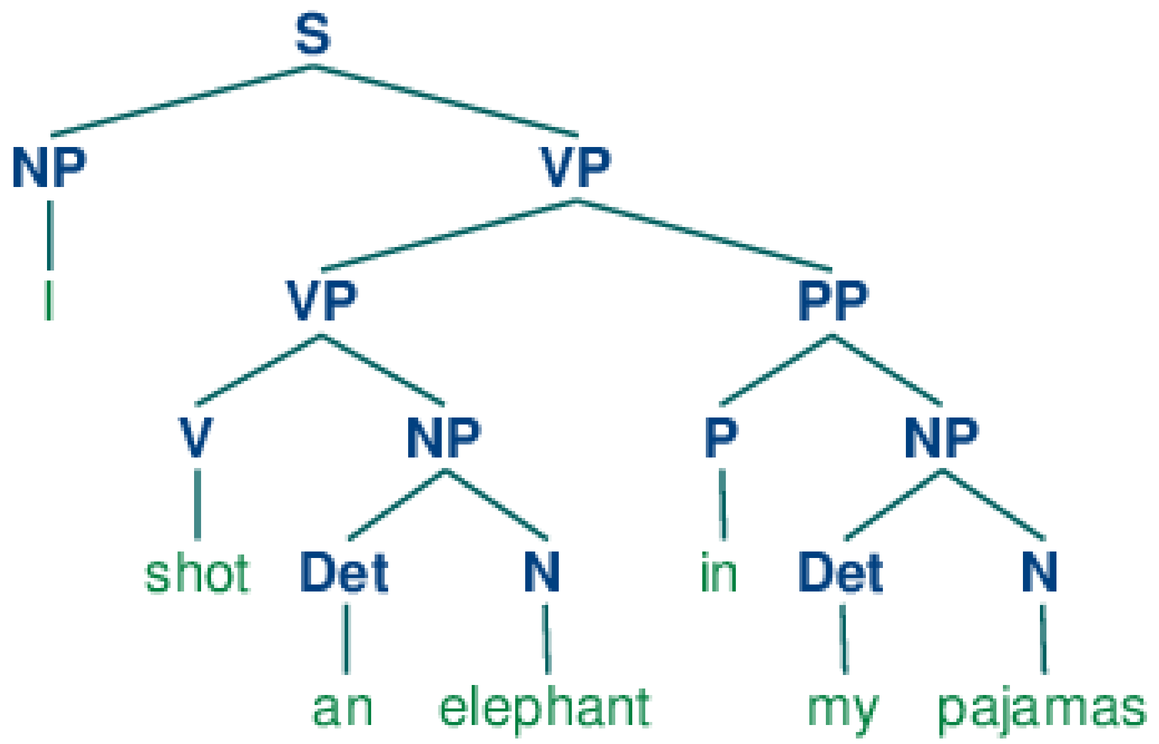
There are typically multiple ways to produce the same sentence. Consider the statement by Groucho Marx:

“While I was in Africa, I shot an elephant in my pajamas”

“How he got into my pajamas, I don’t know”

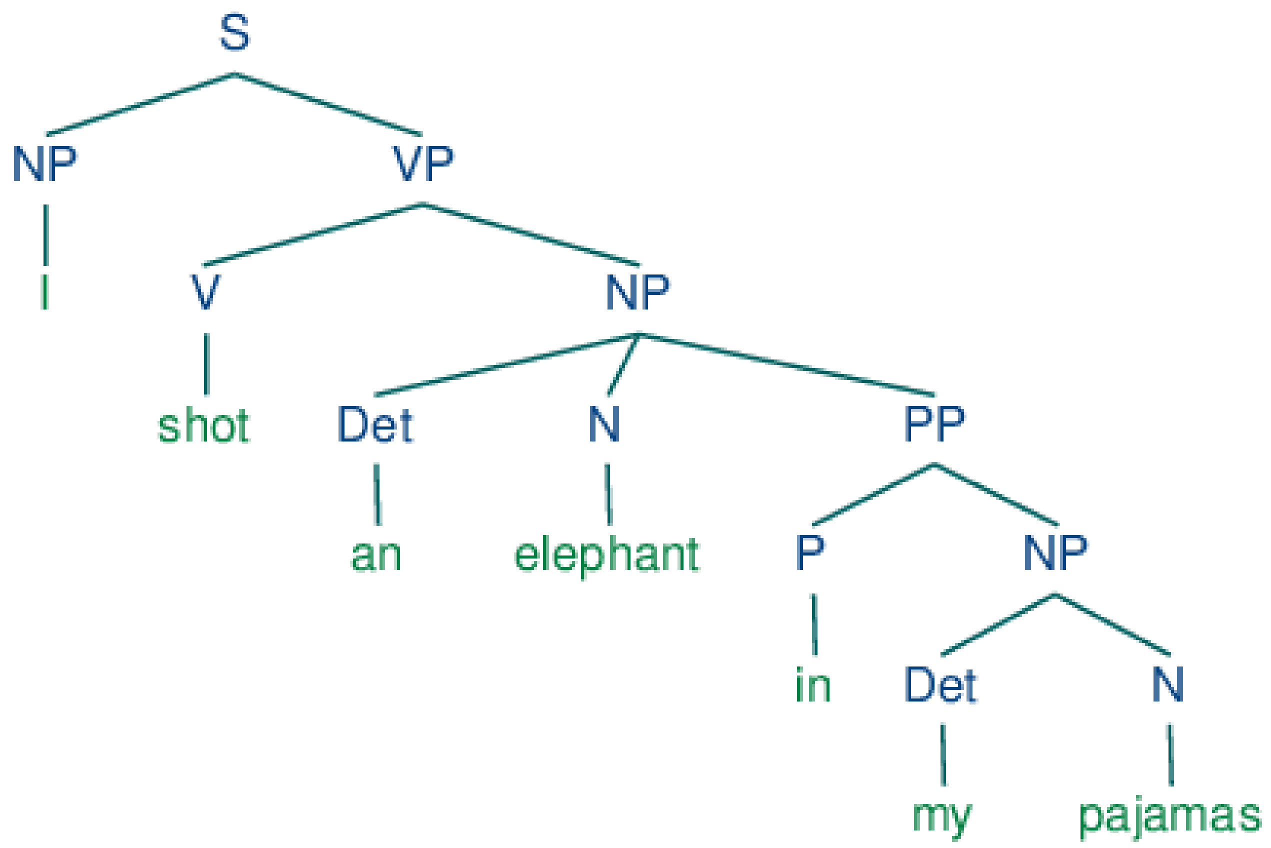
Parse Trees

"…,I shot an elephant in my pajamas" -what people hear first



Parse Trees

Groucho’s version



Grammars

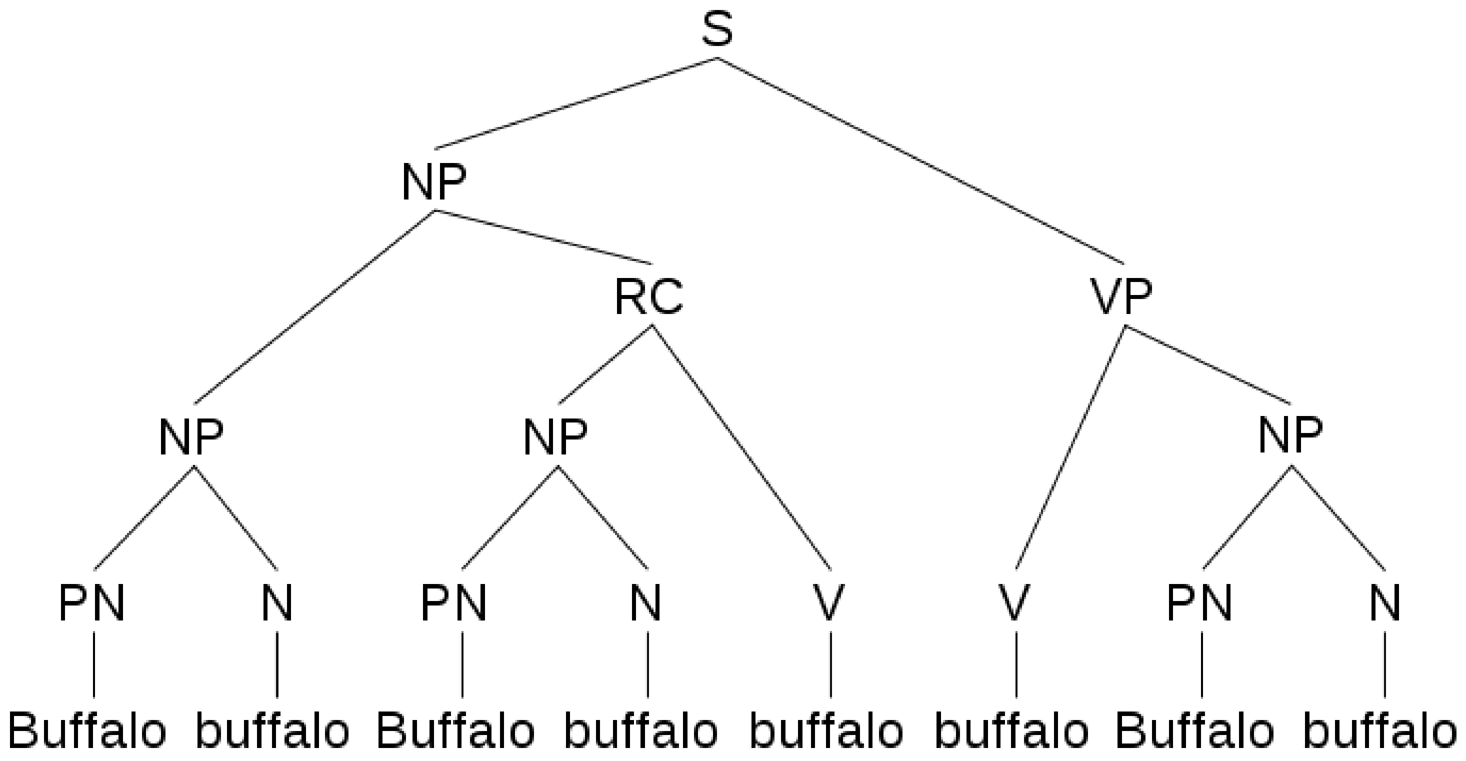
Recursion is common in grammar rules, e.g.

NP → NP RC

Because of this, sentences of arbitrary length are possible.

Recursion in Grammars

"Buffalo buffalo Buffalo buffalo buffalo buffalo Buffalo buffalo".



Grammars

Its also possible to have “sentences” inside other sentences…

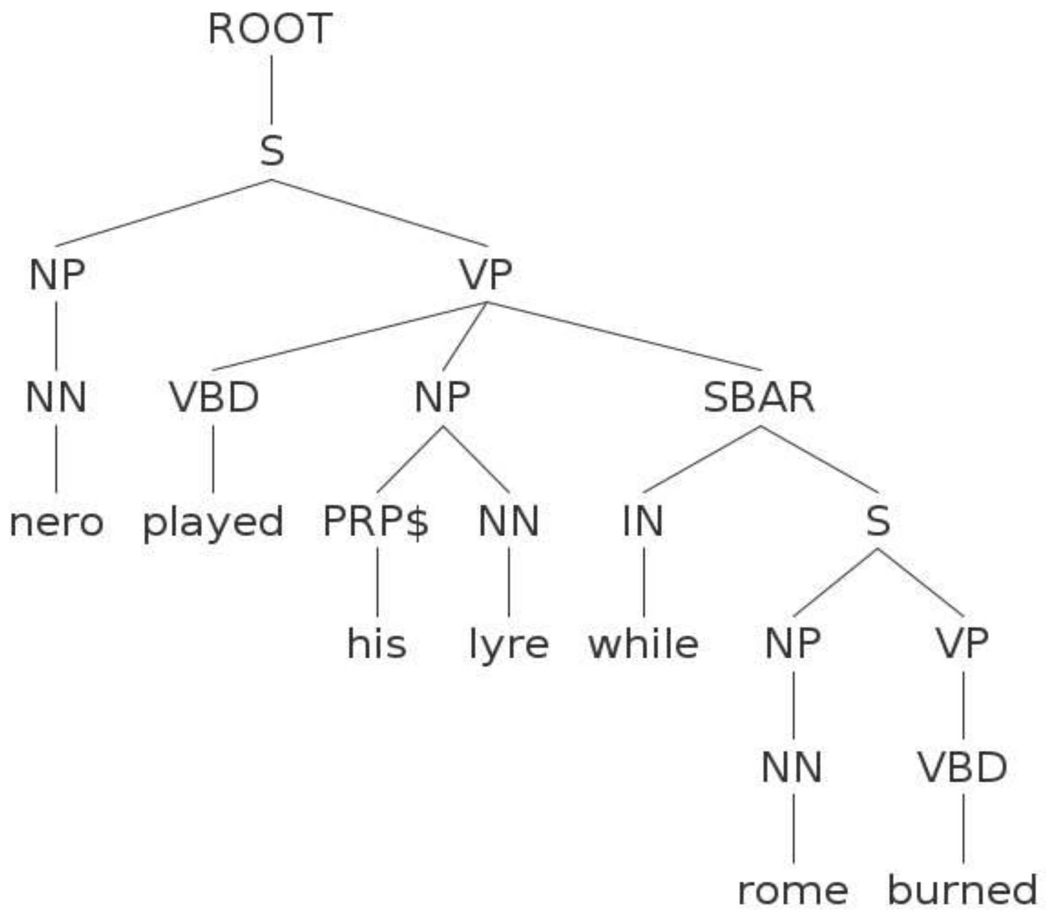
S → NP VP

VP → VB NP SBAR

SBAR → IN S

Recursion in Grammars

"Nero played his lyre while Rome burned".



PCFGs

Complex sentences can be parsed in many ways, most of which make no sense or are extremely improbable (like Groucho's example).

Probabilistic Context-Free Grammars (PCFGs) associate and learn probabilities for each rule:

$S \rightarrow NP\ VP$ 0.3

$S \rightarrow NP\ VP\ PP$ 0.7

The parser then tries to find the most likely sequence of productions that generate the given sentence. This adds more realistic "world knowledge" and generally gives much better results.

Most state-of-the-art parsers these days use PCFGs.

Systems

- **NLTK:** Python-based NLP system. Many modules, good visualization tools, but not quite state-of-the-art performance.
- **Stanford Parser:** Another comprehensive suite of tools (also POS tagger), and state-of-the-art accuracy. Has the definitive dependency module.
- **Berkeley Parser:** Slightly higher parsing accuracy (than Stanford) but not as many modules.
- Note: high-quality parsing is usually very slow, but see: https://github.com/dlwh/puck

Dependencies

In a constituency parse, there is no direct relation between the constituents and words from the sentence (except for leaf nodes which produce a single word).

In dependency parsing, the idea is to decompose the sentence into relations directly between words.

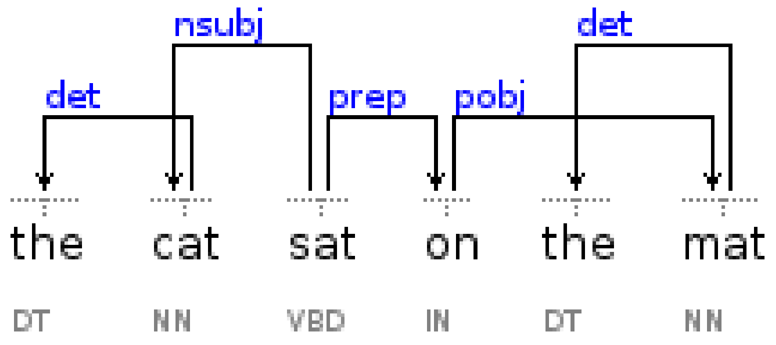
This is an older, and some argue more natural, decomposition of the sentence. It also often makes semantic interpretation (based on the meanings of the words) easier.

Lets look at a simple example:

Dependencies

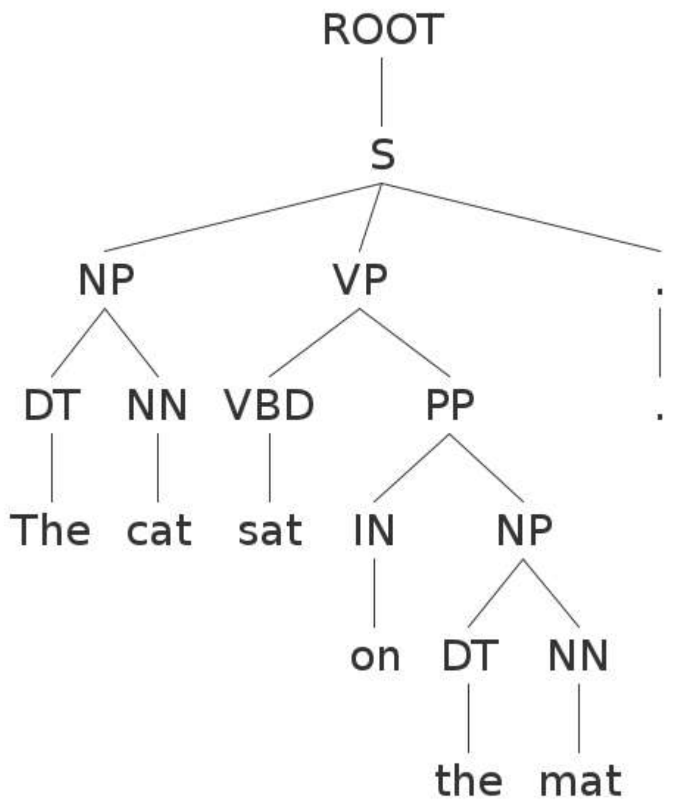
"The cat sat on the mat"

dependency tree



constituency labels of leaf nodes

parse tree



Dependencies

From the dependency tree, we can obtain a “sketch” of the sentence. i.e. by starting at the root we can look down one level to get:

“cat sat on”

And then by looking for the object of the prepositional child, we get:

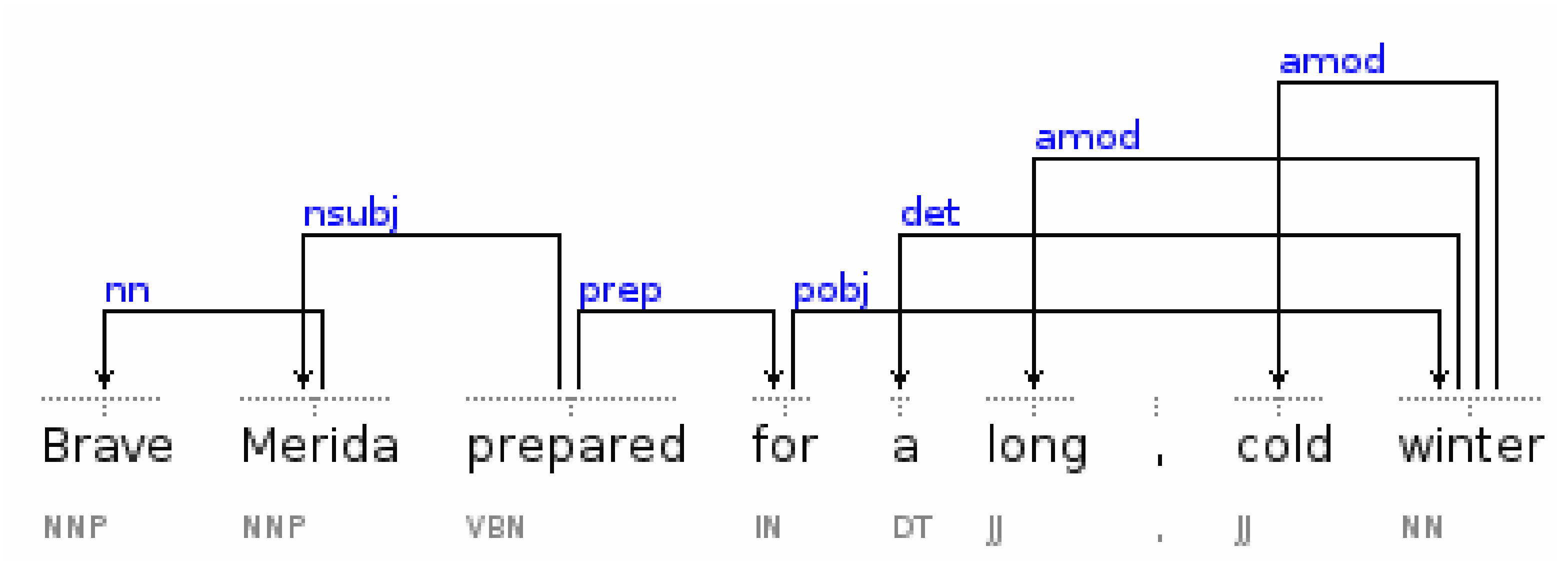
“cat sat on mat”

We can easily ignore determiners “a, the”.

And importantly, adjectival and adverbial modifiers generally connect to their targets:

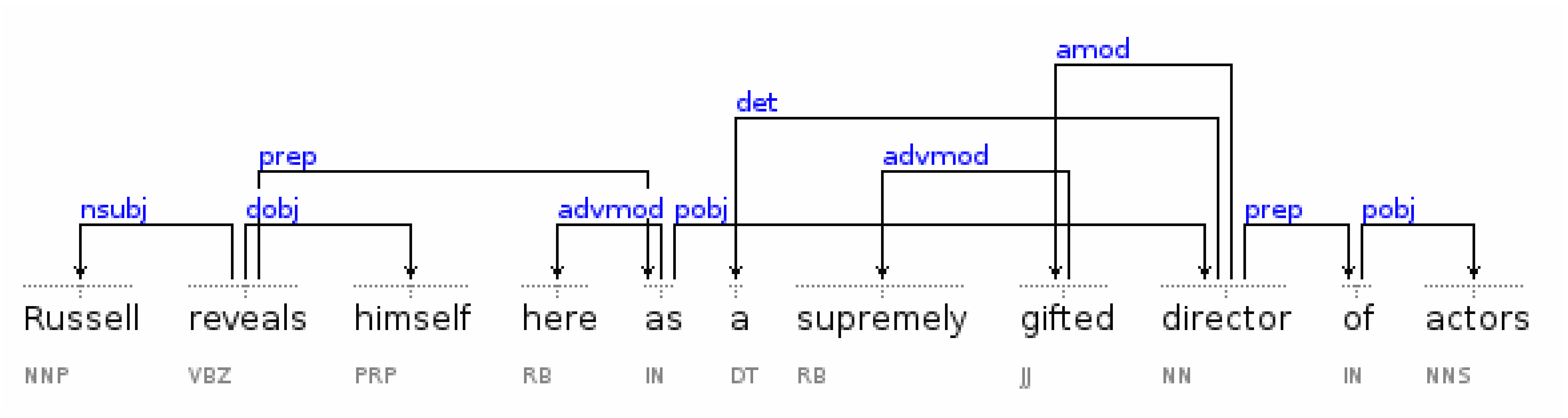
Dependencies

“Brave Merida prepared for a long, cold winter”



Dependencies

"Russell reveals himself here as a supremely gifted director of actors"



Dependencies

Stanford dependencies are constructed from the output of a constituency parser (so you can in principle use other parsers).

The mapping is based on hand-written regular expressions.

Dependency grammars have been widely used for sentiment analysis and for semantic embedings of sentences.