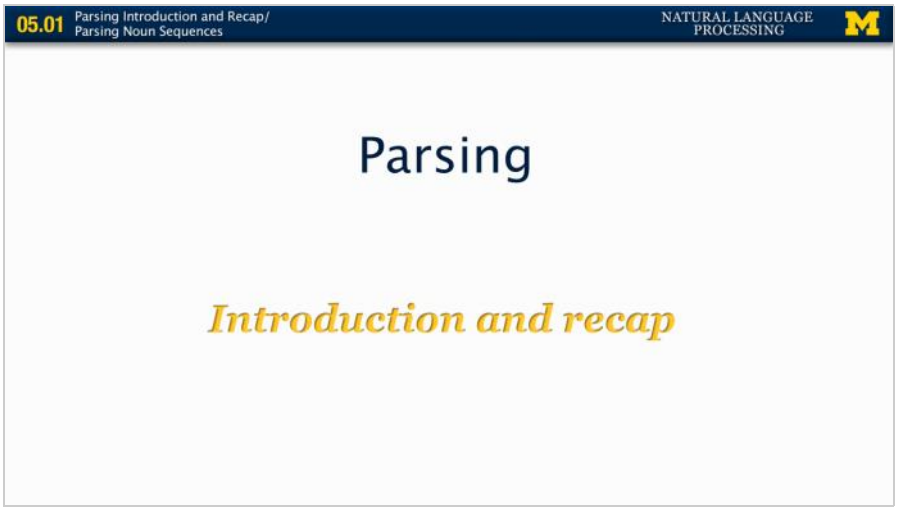
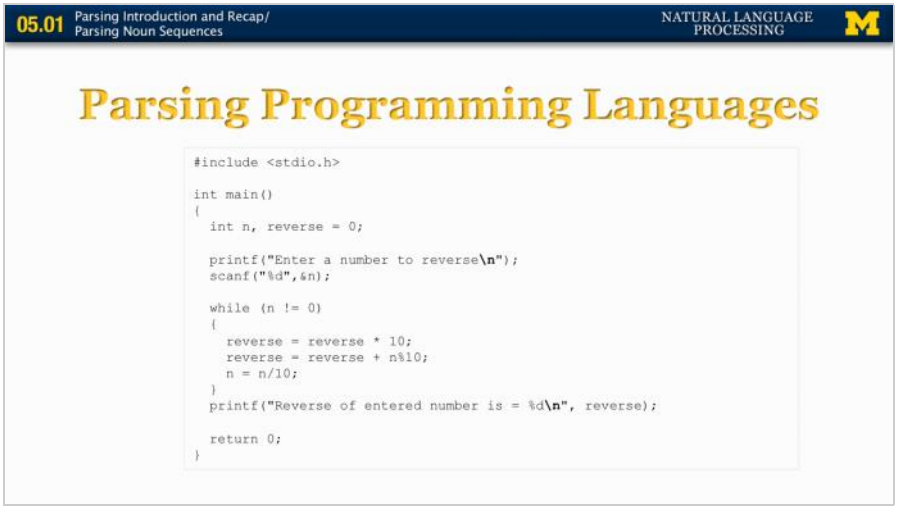
Week 5

2016年7月20日 星期三　　下午4:46

5.1



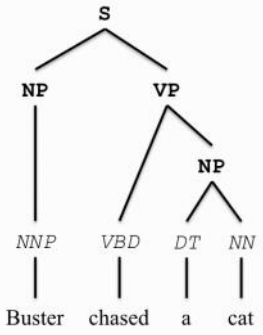


Parsing Human Language

- **Coordination scope**: Small boys and girls are playing.
- **Prepositional phrase attachment**: I saw the man with the telescope.
- **Gaps**: Mary likes Physics but hates Chemistry.
- **Particles vs. prepositions**: She ran up a large bill.
- **Gerund vs. adjective**: Playing cards can be expensive.

Conj. scope
prep. attachment
omitted subject (clear for human unclear for machine)
PP
hill
• to play cards
• a specific card called "playing card" adj

Phrase Structure



Parsing

Parsing noun sequences

Noun-noun Compounds

n. n.

- Fish tank = tank that holds fish
- Fish net = net used to catch fish
- Fish soup = soup made with fish
- Fish oil = oil extracted from fish
- Fish sauce = sauce for fish dishes? sauce made of fish?

Noun-noun Compounds

noun-noun compounds

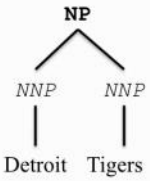
- Head of the compound
  - College junior – a kind of junior
  - Junior college – a kind of college
- Head first?
  - Attorney general
- Adjectives?
  - New Mexico, general manager
- More than two nouns?
  - luxury car dealership

head 放在后面.

head 放在前面

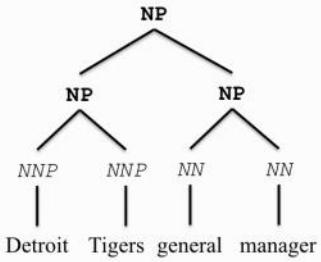
adj.

Noun Phrase Consisting Of Two Nouns



Noun Phrase Consisting Of Four Nouns

!!



Representation Using Parentheses

Salt(Lake City)

- ((Salt Lake) City)
- (Salt (Lake City))

- (((Salt Lake) City) mayor)?

Solution

- (((Salt Lake) City) mayor)

Representation Using Parentheses

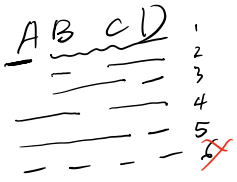
- (((Salt Lake) City) mayor)
- ((Detroit Tigers) (general manager))
- ((Leland Stanford) Junior) University)?

Solution

- (((Leland Stanford) Junior) University)

Combinatorics

- n=2
  (A B)
- n=3
  ((A B) C)
  (A (B C))
- n=4
  ((A B)(C D))



Total # of ways to combine 4 items

Solution

- n=4
  ((A B)(C D))
  (A (B (C D))
  (A ((B C) D))
  ((A (B C)) D)
  (((A B) C) D)

What About n>4?

- n=5
  ((A B)((C D)E))
  …

n How many types of combinations.

Solution

- The general solution is C(n), a notation for the nth Catalan number

  $C_n = 1/(n+1)\ \binom{2n}{n}$, for $n \geq 0$

- 1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, 208012, 742900, …
- Sequence A000108 in the On-Line Encyclopedia of Integer Sequences® (OEIS®)
- https://oeis.org/

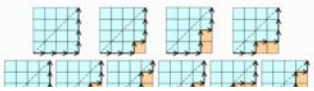
Catalan number

$C_n = \frac{1}{n+1}\binom{2n}{n}$ for $n \geq 0$

Other Uses Of Catalan Numbers

- the number of different ways a convex polygon with $n + 2$ sides can be cut into triangles by connecting vertices with straight lines.

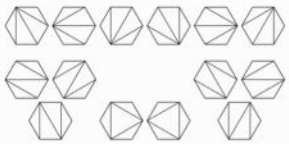
- the number of monotonic paths along the edges of a grid with $n \times n$ square cells, which do not pass above the diagonal.



other usages of Catalan number.

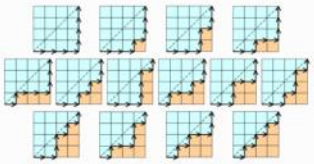
Other Uses Of Catalan Numbers

- the number of different ways a convex polygon with $n + 2$ sides can be cut into triangles by connecting vertices with straight lines.



http://en.wikipedia.org/wiki/File:Catalan-Hexagons-example.svg

- the number of monotonic paths along the edges of a grid with $n \times n$ square cells, which do not pass above the diagonal.



http://en.wikipedia.org/wiki/File:Catalan_number_4x4_grid_example.svg

other usages of Catalan number.

05.01 Parsing Introduction and Recap/ Parsing Noun Sequences NATURAL LANGUAGE PROCESSING

NLP

5.2

NLP

Parsing

Prepositional Phrase Attachment (1)

05.02 Prepositional Phrase Attachment 1/3 NATURAL LANGUAGE PROCESSING

Penn Treebank Representation

```
( (S
    (NP-SBJ
      (NP (NNP Pierre) (NNP Vinken) )
      (, ,)
      (ADJP
        (NP (CD 61) (NNS years) )
        (JJ old) )
      (, ,) )
    (VP (MD will)
      (VP (VB join)
        (NP (DT the) (NN board) )
        (PP-CLR (IN as)
          (NP (DT a) (JJ nonexecutive) (NN director) ))
        (NP-TMP (NNP Nov.) (CD 29) )))
    (. .) ))
```

05.02 Prepositional Phrase Attachment 1/3 NATURAL LANGUAGE PROCESSING

Penn Treebank Representation

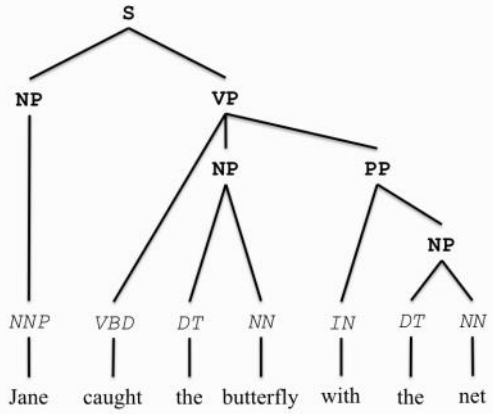
```
( (S
    (NP-SBJ (NNP Mr.) (NNP Vinken) )
    (VP (VBZ is)
      (NP-PRD
        (NP (NN chairman) )
        (PP (IN of)
          (NP
            (NP (NNP Elsevier) (NNP N.V.) )
            (, ,)
            (NP (DT the) (NNP Dutch) (VBG publishing) (NN group) )))))
    (. .) ))
```

Prepositional Phrase Attachment

- High (verbal): join board **as director** → nearest verb.
- Low (nominal): is chairman **of Elsevier** modify chairman

the way the person joins the board.

not modify board



Examples

- Examples:
  - Lucy's plane leaves Detroit on Monday. – high
  - Jenna met Mike at the concert. – high
  - This painting must cost millions of dollars. – low
- High or low attachment?
  - Alicia ate spaghetti from Italy. l
  - Alicia ate spaghetti with meatballs. l
  - Alicia ate spaghetti with a fork. h
  - Alicia ate spaghetti with Justin. h
  - Alicia ate spaghetti with delight. h
  - Alicia ate spaghetti on Friday. l

→ modify verb: leave

the concert modify verb

Solution

- High or low attachment?
  - Alicia ate spaghetti **from Italy.** – low
  - Alicia ate spaghetti **with meatballs.** – low
  - Alicia ate spaghetti **with a fork.** – high
  - Alicia ate spaghetti **with Justin.** – high
  - Alicia ate spaghetti **with delight.** – high
  - Alicia ate spaghetti **on Friday.** – high

Actual Headline

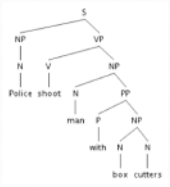
- Police shoot man with box cutters.

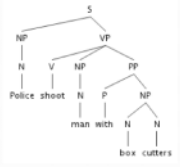
high attachment

straight forward for human.

(S (NP (N Police)) (VP (V shoot) (NP (N man) (PP (P with) (NP (N box) (N cutters))))))

(?) (S (NP (N Police)) (VP (V shoot) (NP (N man)) (PP (P with) (NP (N box) (N cutters)))))





Binary classification

Prepositional Phrase Attachment

- Input: a prepositional phrase and the surrounding context
- Output: a binary label: 0(high) or 1(low)
- In practice: the context consists only of four words: the preposition, the verb before the preposition, the noun before the preposition, and the noun after the preposition
- Example: *join board as director*
- Why?

Why only focus on the four words

Answer

- Because almost all the information needed to classify a prepositional phrase's attachment as high or low is contained in these four features.
- Furthermore, using only these *tuples* of four features allows for a consistent and scaleable approach.

Sample Tuples

Sent #	Verb	$Noun_1$	Preposition	$Noun_2$	Class
0	join	board	as	director	V
2	named	director	of	conglomerate	N
3	caused	percentage	of	deaths	N
6	bring	attention	to	problem	V
12	led	team	of	researchers	N
16	including	three	with	cancer	N
24	imposed	ban	on	uses	N
26	made	paper	for	filters	N
28	dumped	sacks	of	material	N
28	dumped	sacks	into	bin	V

Sidebar (1/2)

- The linguistics (and psycholinguistics) literature offers competitive explanations for attachment.
- One theory (Kimball 1973) favors the so-called *right association* rule. It says that, given a new phrase and two choices for attachment, people tend to attach the new phrase with the more recent ("rightmost" within the sentence) of the candidate nodes, resulting in low attachment.
- Alternatively, the *minimal attachment* principle (Frazier 1978) favors an attachment that results in the syntax tree having fewer additional syntactic nodes (in this case, favoring high attachment).
- As one can see from the statistics, none of these methods alone can explain the high prevalence of both types of attachment.

Sidebar (2/2)

- Some other observations can be made by performing statistical analysis of a training set.
- The standard corpus used for this sort of analyses comes from (RRR 1994) and includes 27,937 prepositional phrases extracted from the Penn Treebank (Marcus et al. 1993), divided into three groups (20,801 training, 4039 development, and 3097 test).
- This data representation makes the assumption that additional context is only marginally more useful for classification purposes compared to the four features in the table (verb, noun1, preposition, and noun2).
- For comparison, the sentence matching the data point "bring attention to problem" is actually "Although preliminary findings were reported more than a year ago, the latest results appear in today's New England Journal of Medicine, a forum likely to bring new attention to the problem." It is unlikely that the information in the first ¾ of the sentence will affect the classification of the prepositional phrase "to the problem".

05.02 Prepositional Phrase Attachment 1/3 NATURAL LANGUAGE PROCESSING

NLP

Supervised learning
good for this problem
manually label
→ train a system
→ evaluate
by ~accuracy

The simplest baseline method?
Find the most common label
⇒ Assign it to all.

5.3

NLP

05.03 Prepositional Phrase Attachment 2/3 NATURAL LANGUAGE PROCESSING

Parsing

Prepositional Phrase Attachment (2)

Supervised Learning: Evaluation

- Manually label a set of instances.
- Split the labeled data into training and testing sets.
- Use the training data to find patterns.
- Apply these patterns on the testing data set.
- For evaluation: use **accuracy** (the percentage of correct labels that a given algorithm has assigned on the testing data).
- Compare with a simple baseline method.
- What is the simplest baseline method?

Answer

- The simplest supervised baseline method is to find the more common class (label) in the training data and assign it to **all** instances of the testing data set.

Algorithm 1

label the tuple as "low" (default)

Random baseline

- A **random** unsupervised baseline would have been to label each instance in the testing data set with a random label, 0 or 1.
- Practically, random performance is the lower bound against which any non-random methods should be compared.

Randomly label the data

Measuring The Accuracy Of The Supervised Baseline Method (Algorithm 1)

- In the official training data set (RRR94), the rate of occurrence of the 1 label (low attachment) is 52.2% (10,865/20,801).
- Is the accuracy of this baseline method then equal to 52.2%?

Solution

- No, this is not how accuracy is computed. It has to be computed on the testing set, not the training set.
- Using the official split, the accuracy of this method on the testing set is 59.0% (1,826/3,097). One shouldn't think of this number as a good result. The difference (+6.8% going from training to testing) could have been in the opposite direction, resulting in a performance below random.

should be calculated on the testing set, not the training set

Observations

- If the baseline method is simple and if the testing set is randomly drawn from the full data set and the data set is large enough, one could expect that the accuracy on the testing set is comparable to the one on the training set. Note that the PTB data set is drawn from business news stories. If one were to train a method on this data and test it on a different set of sentences, e.g., from a novel, it is possible that the two sets will have very different characteristics.
- The more complicated the method is, however, the more likely it will be that it will "overfit" the training data, learning patterns that are too specific to the training data itself and which may not appear in the testing data or which may be associated with the opposite class in the testing data.

Upper Bounds On Accuracy

- The 52% accuracy we've seen so far is our current lower bound. Now, what is the upper bound?
- Usually, human performance is used for the upper bound. For PP attachment, using the four features mentioned earlier, human accuracy is around 88%.
- So, a hypothetical algorithm that achieves an accuracy of 87% is in fact very close to the upper bound (on a scale from 52% to 88%, it is 97% of the way to the upper bound).

human performance as upper bound

Using Linguistic Knowledge

- One way to beat the two baselines is to use linguistic information. For example, the preposition "of" is much more likely to be associated with low attachment than high attachment.
- In the training data set, this number is an astounding 98.7% (5,534/5,607)
- Therefore the feature *prep_of* is very valuable. What are the two main reasons?

Answer

- Reason 1 – it is very informative (98.7% of the time it is linked with the low attachment class)
- Reason 2 – it is very frequent (27.0% of the entire training set – 5607/20801).
- Reason 1 alone would not be sufficient!

Why this linguistic knowledge is important?
① informative
② frequent

Sidebar 1/3

- The PTB (Penn Treebank) data set has been used for competitive evaluation of pp attachment algorithms since 1994.
- Each new algorithm is allowed to look at the training and development sets and use any knowledge extracted from them.
- The test set data can never be looked at and can be used only once per algorithm for its evaluation.
- Doing the contrary (repeatedly tuning a new algorithm based on its performance on the designated test set) results in a performance level that is irreproducible on new data and such approaches are not allowed in NLP research.
- Note that the development set can be used in a well-specified way as part of the training process.

if look at it many time will over-train the model

Sidebar 2/3

- Let's look at the training data then and see if we can learn some patterns that would help us improve over the silly "label everything as noun attachment" baseline and its 52% expected accuracy.
- For example, some prepositions tend to favor particular attachments.
- Let's start with "against". It appears 172 times in the training set, of which 82 (48%) are noun attached, the rest (52%) being high attached. This ratio (48:52) is very similar to the baseline (52:48), so clearly, knowing that the preposition is "against" gives us very little new information.

Sidebar 3/3

- Furthermore, the total occurrence of "against" in the training corpus is rather small (less than 1% of the prepositional phrases).
- In two special cases, however, the identity of the preposition gives a lot of information.
- At one extreme, "of" is associated with low attachment in 99% of the cases, whereas at the other end of the scale, "to" is associated with high attachment in 82% of its appearances.
- It is also important to note that both of these prepositions are fairly common in the training set ("to" representing 27% of all prepositions and "of" accounting for another 11% of them).
- With this knowledge, we can build a very simple decision list algorithm (Brill and Resnik) that looks like this (the rules are sorted by their expected accuracy on their majority class).

Are There Any Other Such Features?

- Yes, *prep_to*: 2,182/2,699 = 80.8% in favor of high attachment.
- Which leads us to our next algorithm:

Algorithm 2

If the preposition is "of", label the tuple as "low".
Else
  If the preposition is "to", label the tuple as "high".
  Else
    label the tuple as "low" (default)

still constructing the baseline

Sidebar 1/2

- Let's compare the performance of this simple decision list algorithm with that of the baseline.
- On the training set, the first rule would fire in 5,577 cases, of which 5,527 will be correctly labeled as low attachment and another 50 will be incorrectly labeled as high attachment (the accuracy of this rule is expected to be 99% on the training set).
- The second rule (with an expected accuracy of 82% on the training set) would result in 2,672 decisions, of which 2,172 will be correctly processed as high attachment and 500 will be mislabeled as low attachment.
- Finally, the default rule will kick in, whenever the first two rules are not applicable. In this case, it will apply on all remaining phrases (20,801 – 5,577 – 2,672 = 12,552 cases).
- Specifically, among these, it will result in 4,837 phrases correctly labeled as low and 7,714 incorrectly labeled as high.

Sidebar 2/2

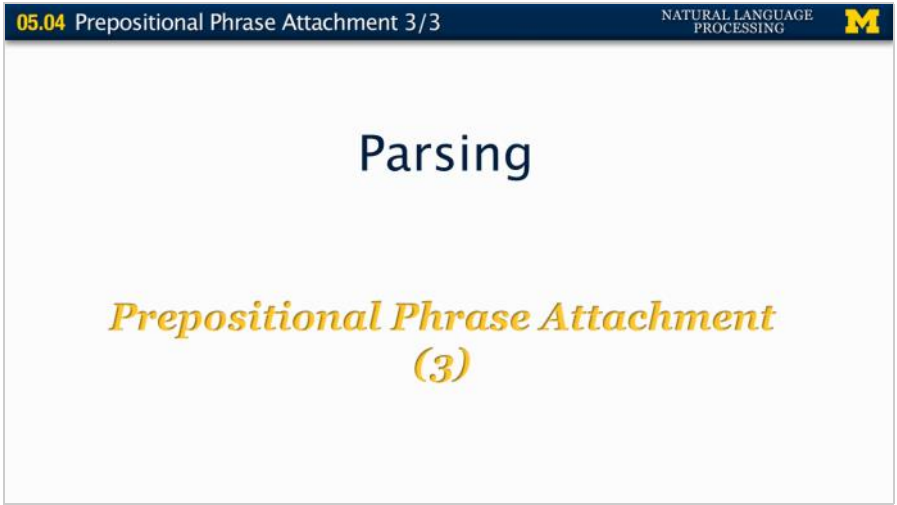
- Overall, we have 5,527 + 2,172 + 4,837 = 12,536 correct decisions, or an accuracy of 12,536/20,801 = 60%.
- Clearly Algorithm 2 outperforms Algorithm 1 on the training data.
- This is not surprising, since its expected accuracy should be no less than the worst expected accuracy of its rules (that of rule 3) and it is likely to be higher than that lowest number because of the priority given to the easier to classify cases (rules 1 and 2).
- More complicated algorithms can look at additional features, e.g., the nouns or the verb, or some combination thereof. For example, the verb "dropped" appears 75/81 times (93%) along with high attachment and only 7% with low attachment.

NLP

simple algorithm
ample training data
=> the performance on training & test set must be the same.

simple algorithm
ample training data
=> the performance on training & test set must be the same.

5.4



Algorithm 2a

If the preposition is "of", label the tuple as "low".
Else
  If the preposition is "to", label the tuple as "high".
  Else
    label the tuple as "high"

Some Observations

- First, even though the expected performance of rule 3 was 52%, its actual performance on the training set dropped to 39% after rules 1 and 2 were applied.
- In other words, these rules used up some of the information hidden in the data ahead of rule 3 and left it less useful information to rely upon.
- Even more, one can see that a better decision would have been to replace rule 3 with its exact opposite, label everything left at this stage as "high", which would have boosted the combined performance.
- Algorithm 2a would achieve 5,527 + 2,172 + 7,714 = 15,413 correct decisions for an overall accuracy of 74% on the training set.
- Second, one cannot help but notice that Algorithms 2 and 2a each have only three rules. We can imagine a classifier with 20,801 rules, one per training example, each rule of the form "if the preposition is "of" and the nouns are such and such and the verbs are such and such, then classify the data point as the actual class observed in the training set".

Some Observations

- Third, we so far reported performance on the training set.
- Can we project the performance on the training set to the test set?
- Let's start with Algorithms 1 and 3.
- Algorithm 1 labels everything as low attachment. It achieved 52% on the training set. We expect its performance on the test set to be similar. In fact it is 59% (1,826/3,097).
- This clearly demonstrates the variability of text across subsets of the data.
- In this case, this variability favors Algorithm 1 since its performance actually goes up when moving to the test set. In other cases (e.g., if we had swapped the training and test sets), its performance would have gone down.
- On average though, its performance on the test data is expected to vary around the performance on the training data.

Algorithm 3

If the preposition is "on" and the verb is "casting" and the first noun is "cloud" and the second noun is "economy", label the phrase as "high".
Else
If the preposition is "of" and the verb is "opened" and the first noun is "can" and the second noun is "worms", label the phrase as "low".
Else
… 20799 more rules …

Some Observations 1/4

- Now, let's consider Algorithm 3. It achieved a very high performance on the training data (way above the "upper bound" achieved by humans).
- However, we will now see the meaning of the word *overfitting* in action.
- Algorithm 3 was so specific to the training data that most of the rules it learned don't apply at all in the test set.
- Only 117 combinations (out of 3032) of words in the test set match a combination previously seen in the training set.
- In other words, Algorithm 3 learned a lot of good rules, but it failed to learn many more. In fact, its accuracy on the test data is only around 4%.
- An alternative to Algorithm 3 would be to combine it with a default rule (just like rule 3 in Algorithm 2) that labels everything that Algorithm 3 missed as noun attachment.
- Unfortunately, even this algorithm (let's call it Algorithm 3a) would only achieve a performance slightly above the baseline (Algorithm 1) of 59% on the test data.
- The lesson to learn here is that, on unseen data, a simple algorithm (Algorithm 1) is much better than a really complicated one that overfits (Algorithm 3). Also, the combination of the two (overfitting + baseline) just barely outperforms the baseline itself and is nowhere close to competitive.

so specific to the training data
⇒ hardly apply to the test set
⇒ only 4% accuracy on the test data.

Some Observations 2/4

- Clearly this algorithm (Algorithm 3) would achieve close to 100% accuracy on the training set.
- Why "close to 100%" and not "100%"?
- It turns out that the training set there are mutually inconsistent labels for the same data point.
- For example, "won verdict in case" appears once as high and once as low attachment.
- There are a total of 56 such "discrepancies" in the training set.
- Some of them are caused by inconsistent annotators whereas others would require more context (e.g., the entire paragraph or document) to be correctly disambiguated.

why no 100% accuracy? Problem with training set
training set can be inconsistent

Some Observations 3/4

- Next, let's see how algorithms 2 and 2a will fare on the test set.
- First, let's look at Algorithm 2.
- There are 3097 items to classify in the test set.
- Rule 1 correctly classifies 918 out of 926 instances of "of" (99% accuracy) while rule 2 gets 70% accuracy (234/332 correctly classified).
- Rule 3 achieves 810/1,839 = 44%.
- Overall the accuracy of Algorithm 2 on the test set is 63% (1,962/3,097).
- Again on the test data, Algorithm 2a outperforms Algorithm 2. Its Rule 3 gets 1,029/1,839 = 56% accuracy and the overall accuracy of Algorithm 2a on the test set is 70% (2,181/3,097).

Some Observations 4/4

- Let's now summarize the performance of the five algorithms that we have looked at so far.

Algorithm	Number of rules	Training set accuracy	Test set accuracy
Algorithm 1: default	1	52%	59%
Algorithm 2: of/to + default	3	60%	63%
Algorithm 2a: of/to + better default	3	74%	70%
Algorithm 3: memorize everything	20801	near 100%	4%
Algorithm 3a: memorize + default	20802	near 100%	62%

What's Next?

- So far, so good. We have been able to go from 59% test set accuracy to 70% with two simple rules.
- What additional sources of information can we use to improve the algorithm?
- Here are some ideas:
  - use a few more *good* word features (e.g., more prepositions, perhaps some verb and nouns)
  - use clever ways to deal with missing information
  - use lexical semantic information (e.g., synonyms)
  - use additional context beyond the four feature types used so far.

ideas improving performance

PREP		
about	67	132
as	380	94
at	552	136
for	1136	1044
in	2251	1577
of	73	5534
on	666	550
to	2182	517
with	698	340
VERB		
bring	58	18
buy	217	126
cut	57	34
drop	84	15
follow	15	91
include	22	221
put	178	37

NOUN1		
company	61	33
director	6	51
increase	17	57
loan	23	9
rate	72	68
NOUN2		
asset	5	49
bank	33	31
board	22	23
client	23	9
company	68	204
day	57	14
year	427	106

05.04 Prepositional Phrase Attachment 3/3 NATURAL LANGUAGE PROCESSING

What's Next?

- Let's first consider a combination of the first two ideas above: looking for ways to use all possible information that can be extracted from the training data.
- This is the approach that was used by Collins and Brooks (1995).
- Their method was based on a principle called backoff which is somewhat of a combination of all the algorithms used so far (e.g., Algorithms 1, 2, and 3).
- Backoff allows us to use the most specific evidence from the training data, when available but then make reasonable approximations for the missing evidence.

05.04 Prepositional Phrase Attachment 3/3 NATURAL LANGUAGE PROCESSING

Collins and Brooks

- So what do we do? Collins and Brooks used the following algorithm.
- If a 4-tuple is available, use it.
- If not, combine the evidence from the triples that form the 4-tuple (looking only at the triples that include the preposition).
- If that is not available, look at the pairs, then the singletons, and finally use a default class.
- A 4-tuple is just a set of 4 features in a particular order, e.g., (verb, noun1, preposition, noun2).
- The matching term for 3 features is a triple; for 2 features it is a pair; and for 1 feature, the word singleton is used.

If the denominator of the next formula is 0, then use the classification for the 4-tuple

$$\hat{p}_4(H|v,n_1,p,n_2) = \frac{f(H,v,n_1,p,n_2)}{f(v,n_1,p,n_2)}$$

Else

If the denominator of the next formula > 0, then use the following estimate:

$$\hat{p}_3(H|v,n_1,p,n_2) = \frac{f(H,v,n_1,p) + f(H,v,p,n_2) + f(H,n_1,p,n_2)}{f(v,n_1,p) + f(v,p,n_2) + f(n_1,p,n_2)}$$

Else

If the denominator of the next formula > 0, then use the following estimate:

$$\hat{p}_2(H|v,n_1,p,n_2) = \frac{f(H,v,p) + f(H,p,n_2) + f(H,n_1,p)}{f(v,p) + f(p,n_2) + f(n_1,p)}$$

Else

If the denominator of the next formula > 0, then use the following estimate:

$$\hat{p}_1(H|v,n_1,p,n_2) = \frac{f(H,p)}{f(p)}$$

Else label as "low" (default):

$$\hat{p}_0(H|v,n_1,p,n_2) = 0$$

What's Next?

- The idea behind Algorithm 3 was quite reasonable – assume that if the same object appear again (as defined by the same set of four features), it will likely have the same tag.
- The problem with this approach is that there is not enough data in the training set to learn the likely classes of all possible combinations of features.
- Let's do the math. To cover all the data points in the test set, we'd need information in the training set for a total of 102,998,280,840 combinations (more than 100 Billion combinations)!
- How did we arrive at this number? It is simply the product of the numbers 1123, 1295, 52, and 1362, which are, respectively, the numbers of distinct verbs, noun1s, prepositions, and noun2s in the test set.
- It is impossible to label so much data and even if it could be done, there would be billions more combinations needed to cover a new test set.

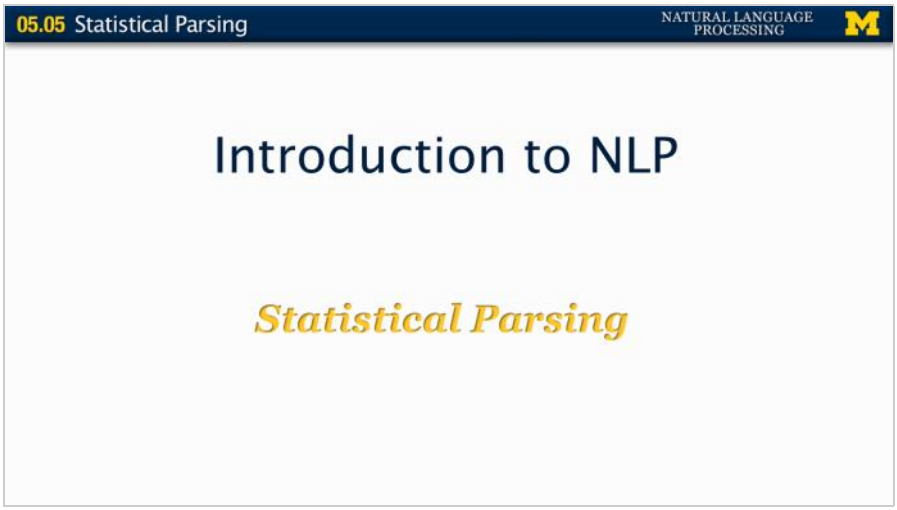
Other Methods

- Zhao and Lin 2004 – nearest neighbors
- Find most similar examples – 86.5% best accuracy
- Similar to Zavrel, Daelemans, and Veenstra 1997 – memory-based learning
- Boosting – Abney et al. 1999
- Semantics – Stetina and Nagao 1997
- Graph-based method – Toutanova et al. 2004

Comparative Results

Algorithm	Test set accuracy
Algorithm 1: default	59%
Algorithm 2a: of/to + better default	70%
Best class per preposition	72%
Collins and Brooks	84%
K-nearest neighbors	80%
TUMBL	82%
Human average (4-tuples only)	88%
Human average (whole sentence)	93%

5.5



probability associated with each rules

Need For Probabilistic Parsing

- Time flies like an arrow
  - Many parses
  - Some (clearly) more likely than others
  - Need for a probabilistic ranking method

probs. associated with each of the rules

Probabilistic Context-free Grammars

- Just like (deterministic) CFG, a 4-tuple (N,Σ,R,S)
  - N: non-terminal symbols np . PP .
  - Σ: terminal symbols (disjoint from N) "cat", "is"
  - R: rules (A → β) [p] → $\sum Prob = 1$
    - β is a string from (Σ ∪ N)*
    - p is the probability P(β|A)
  - S: start symbol (from N)

Example

```
S -> NP VP
NP -> DT N | NP PP
PP -> PRP NP
VP -> V NP | VP PP
DT -> 'a' | 'the'
N -> 'child' | 'cake' | 'fork'
PRP -> 'with' | 'to'
V -> 'saw' | 'ate'
```

Example

```
S -> NP VP
NP -> DT N
NP -> NP PP
PP -> PRP NP
VP -> V NP
VP -> VP PP
DT -> 'a'
DT -> 'the'
N -> 'child'
N -> 'cake'
N -> 'fork'
PRP -> 'with'
PRP -> 'to'
V -> 'saw'
V -> 'ate'
```


Example

```
S -> NP VP
NP -> DT N
NP -> NP PP
PP -> PRP NP
VP -> V NP
VP -> VP PP
DT -> 'a'
DT -> 'the'
N -> 'child'
N -> 'cake'
N -> 'fork'
PRP -> 'with'
PRP -> 'to'
V -> 'saw'
V -> 'ate'
```


Example

```
S -> NP VP      [p0=1]
NP -> DT N      [p1]
NP -> NP PP     [p2]
PP -> PRP NP    [p3=1]
VP -> V NP      [p4]
VP -> VP PP     [p5]
DT -> 'a'       [p6]
DT -> 'the'     [p7]
N -> 'child'    [p8]
N -> 'cake'     [p9]
N -> 'fork'     [p10]
PRP -> 'with'   [p11]
PRP -> 'to'     [p12]
V -> 'saw'      [p13]
V -> 'ate'      [p14]
```

$p_1 + p_2 = 1$

$p_4 + p_5 = 1$

Probability Of A Parse Tree

- The probability of a parse tree *t* given all *n* productions used to build it:

$$p(t) = \prod_{i=1}^{n} p(\alpha_i \rightarrow \beta_i)$$

- The most likely parse is determined as follows:

$$\arg\max_{t \in T(s)} p(t)$$

- The probability of a sentence is the sum of the probabilities of all of its parses

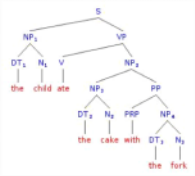
Example

```
S -> NP VP     [p0=1]
NP -> DT N     [p1]
NP -> NP PP    [p2]
PP -> PRP NP   [p3=1]
VP -> V NP     [p4]
VP -> VP PP    [p5]
DT -> 'a'      [p6]
DT -> 'the'    [p7]
N -> 'child'   [p8]
N -> 'cake'    [p9]
N -> 'fork'    [p10]
PRP -> 'with'  [p11]
PRP -> 'to'    [p12]
V -> 'saw'     [p13]
V -> 'ate'     [p14]
```


Example

```
S -> NP VP     [p0=1]
NP -> DT N     [p1]
NP -> NP PP    [p2]
PP -> PRP NP   [p3=1]
VP -> V NP     [p4]
VP -> VP PP    [p5]
DT -> 'a'      [p6]
DT -> 'the'    [p7]
N -> 'child'   [p8]
N -> 'cake'    [p9]
N -> 'fork'    [p10]
PRP -> 'with'  [p11]
PRP -> 'to'    [p12]
V -> 'saw'     [p13]
V -> 'ate'     [p14]
```

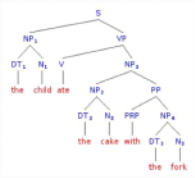
t_1



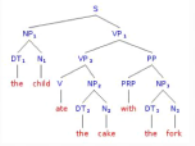
Example

```
S -> NP VP      [p0=1]
NP -> DT N      [p1]
NP -> NP PP     [p2]
PP -> PRP NP    [p3=1]
VP -> V NP      [p4]
VP -> VP PP     [p5]
DT -> 'a'       [p6]
DT -> 'the'     [p7]
N -> 'child'    [p8]
N -> 'cake'     [p9]
N -> 'fork'     [p10]
PRP -> 'with'   [p11]
PRP -> 'to'     [p12]
V -> 'saw'      [p13]
V -> 'ate'      [p14]
```

t_1



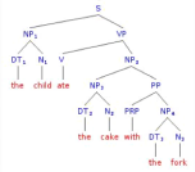
t_2



Example

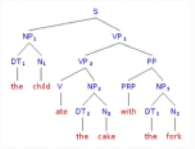
```
S -> NP VP      [p0=1]
NP -> DT N      [p1]
NP -> NP PP     [p2]
PP -> PRP NP    [p3=1]
VP -> V NP      [p4]
VP -> VP PP     [p5]
DT -> 'a'       [p6]
DT -> 'the'     [p7]
N -> 'child'    [p8]
N -> 'cake'     [p9]
N -> 'fork'     [p10]
PRP -> 'with'   [p11]
PRP -> 'to'     [p12]
V -> 'saw'      [p13]
V -> 'ate'      [p14]
```

t_1



$$p(t_1) = p_0 p_1 p_4 p_7 p_8 p_{14} p_2 p_1 p_3 p_7 p_9 p_{11} p_1 p_7 p_{10}$$

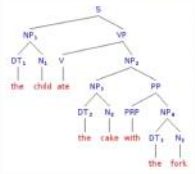
t_2



Example

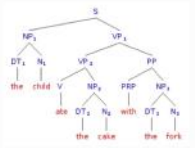
```
S -> NP VP      [p0=1]
NP -> DT N      [p1]
NP -> NP PP     [p2]
PP -> PRP NP    [p3=1]
VP -> V NP      [p4]
VP -> VP PP     [p5]
DT -> 'a'       [p6]
DT -> 'the'     [p7]
N -> 'child'    [p8]
N -> 'cake'     [p9]
N -> 'fork'     [p10]
PRP -> 'with'   [p11]
PRP -> 'to'     [p12]
V -> 'saw'      [p13]
V -> 'ate'      [p14]
```

t_1



$$p(t_1) = p_0 p_1 p_4 p_7 p_8 p_{14} p_2 p_1 p_3 p_7 p_9 p_{11} p_1 p_7 p_{10}$$

t_2

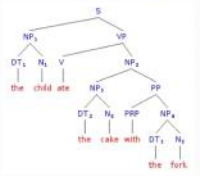


$$p(t_2) = p_0 p_1 p_5 p_7 p_8 p_4 p_3 p_{14} p_1 p_{11} p_1 p_7 p_9 p_7 p_{10}$$

Example

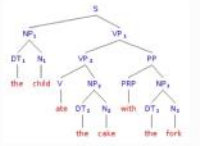
```
S -> NP VP      [p0=1]
NP -> DT N      [p1]
NP -> NP PP     [p2]
PP -> PRP NP    [p3=1]
VP -> V NP      [p4]
VP -> VP PP     [p5]
DT -> 'a'       [p6]
DT -> 'the'     [p7]
N -> 'child'    [p8]
N -> 'cake'     [p9]
N -> 'fork'     [p10]
PRP -> 'with'   [p11]
PRP -> 'to'     [p12]
V -> 'saw'      [p13]
V -> 'ate'      [p14]
```

t_1



$p(t_1) = p_0 p_1 p_4 p_7 p_8 p_{14} p_2 p_1 p_3 p_7 p_8 p_{11} p_1 p_7 p_{10}$

t_2



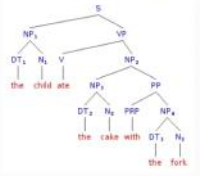
$p(t_2) = p_0 p_1 p_5 p_7 p_8 p_4 p_3 p_{14} p_1 p_{11} p_1 p_7 p_9 p_7 p_{10}$

05.05 Statistical Parsing NATURAL LANGUAGE PROCESSING

Example

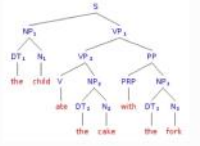
S -> NP VP [p0=1]
NP -> DT N [p1]
NP -> NP PP [p2]
PP -> PRP NP [p3=1]
VP -> V NP [p4]
VP -> VP PP [p5]
DT -> 'a' [p6]
DT -> 'the' [p7]
N -> 'child' [p8]
N -> 'cake' [p9]
N -> 'fork' [p10]
PRP -> 'with' [p11]
PRP -> 'to' [p12]
V -> 'saw' [p13]
V -> 'ate' [p14]

t_1



$p(t_1) = p_0 p_1 p_4 p_7 p_8 p_{14} p_2 p_1 p_3 p_7 p_8 p_{11} p_1 p_7 p_{10}$

t_2



$p(t_2) = p_0 p_1 p_5 p_7 p_8 p_4 p_3 p_{14} p_1 p_{11} p_1 p_7 p_9 p_7 p_{10}$

compare the prob to see which one is more likely

05.05 Statistical Parsing NATURAL LANGUAGE PROCESSING

Main Tasks With Pcfgs

- Given a grammar G and a sentence s, let T(s) be all parse trees that correspond to s
- Task 1
  - find which tree t among T(s) maximizes the probability p(t)
- Task 2
  - find the probability of the sentence p(s) as the sum of all possible tree probabilities p(t)

Probabilistic Parsing Methods

- Probabilistic Earley algorithm
  - Top-down parser with a dynamic programming table
- Probabilistic Cocke-Kasami-Younger (CKY) algorithm
  - Bottom-up parser with a dynamic programming table

Probabilistic Grammars

- Probabilities can be learned from a training corpus (Treebank)
- Intuitive meaning
  - Parse #1 is twice as probable as parse #2
- Possible to do reranking
- Possible to combine with other stages
  - E.g., speech recognition, translation

Maximum Likelihood Estimates

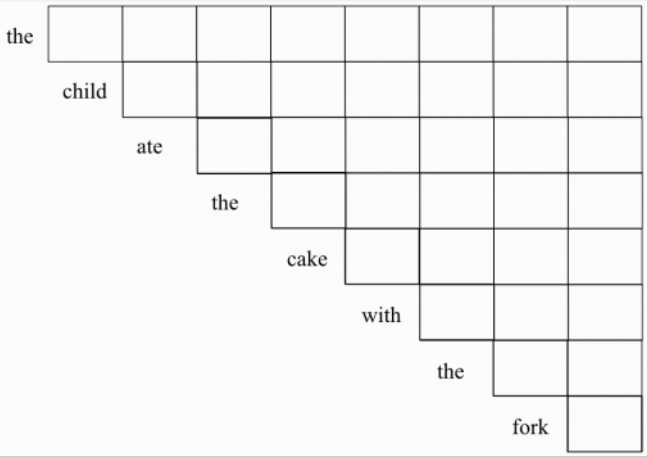
- Use the parsed training set for getting the counts
  - $P_{ML}(\alpha \rightarrow \beta) = Count\,(\alpha \rightarrow \beta) / Count(\alpha)$
- Example:
  - $P_{ML}(S \rightarrow NP\ VP) = Count\,(S \rightarrow NP\ VP) / Count(S)$

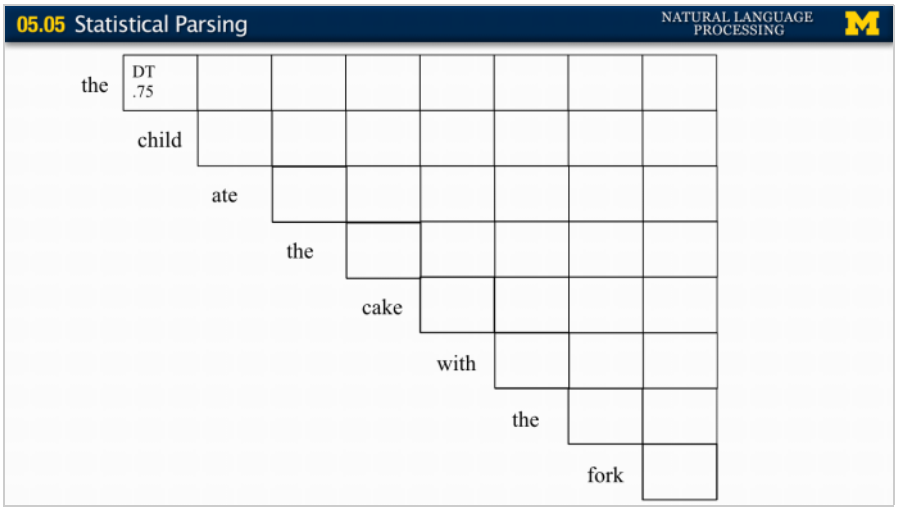
Grammar		Lexicon
S → NP VP	[.80]	*Det → that* [.10] | *a* [.30] | *the* [.60]
S → Aux NP VP	[.15]	*Noun → book* [.10] | *flight* [.30]
S → VP	[.05]	| *meal* [.15] | *money* [.05]
NP → Pronoun	[.35]	| *flights* [.40] | *dinner* [.10]
NP → Proper-Noun	[.30]	*Verb → book* [.30] | *include* [.30]
NP → Det Nominal	[.20]	| *prefer*; [.40]
NP → Nominal	[.15]	*Pronoun → I* [.40] | *she* [.05]
Nominal → Noun	[.75]	| *me* [.15] | *you* [.40]
Nominal → Nominal Noun	[.20]	*Proper-Noun → Houston* [.60]
Nominal → Nominal PP	[.05]	| *NWA* [.40]
VP → Verb	[.35]	*Aux → does* [.60] | *can* [40]
VP → Verb NP	[.20]	*Preposition → from* [.30] | *to* [.30]
VP → Verb NP PP	[.10]	| *on* [.20] | *near* [.15]
VP → Verb PP	[.15]	| *through* [.05]
VP → Verb NP NP	[.05]	
VP → VP PP	[.15]	
PP → Preposition NP	[1.0]	

Example from Jurafsky and Martin

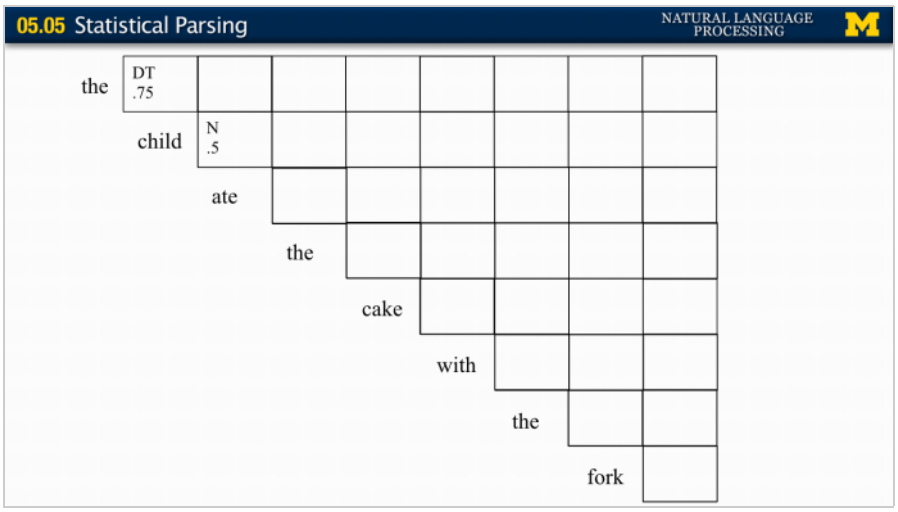
Example

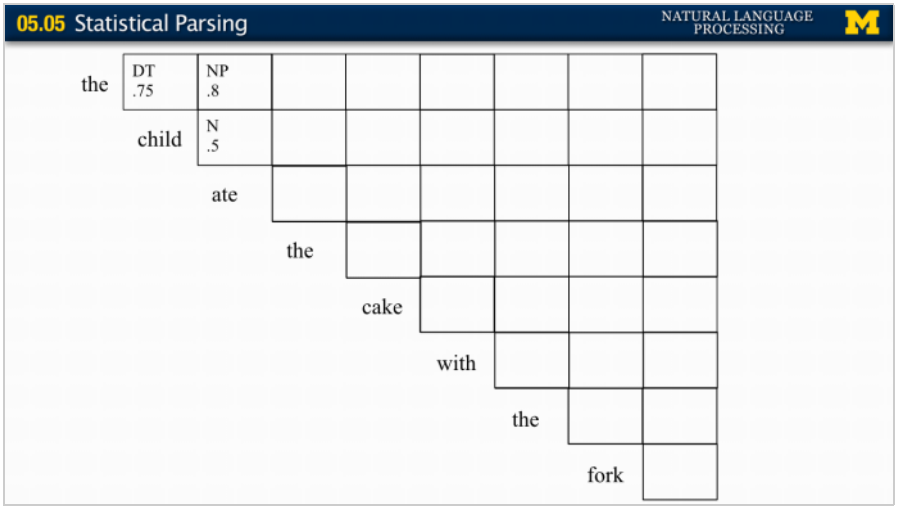
```
S -> NP VP      [p0=1]
NP -> DT N      [p1=.8]
NP -> NP PP     [p2=.2]
PP -> PRP NP    [p3=1]
VP -> V NP      [p4=.7]
VP -> VP PP     [p5=.3]
DT -> 'a'       [p6=.25]
DT -> 'the'     [p7=.75]
N -> 'child'    [p8=.5]
N -> 'cake'     [p9=.3]
N -> 'fork'     [p10=.2]
PRP -> 'with'   [p11=.1]
PRP -> 'to'     [p12=.9]
V -> 'saw'      [p13=.4]
V -> 'ate'      [p14=.6]
```

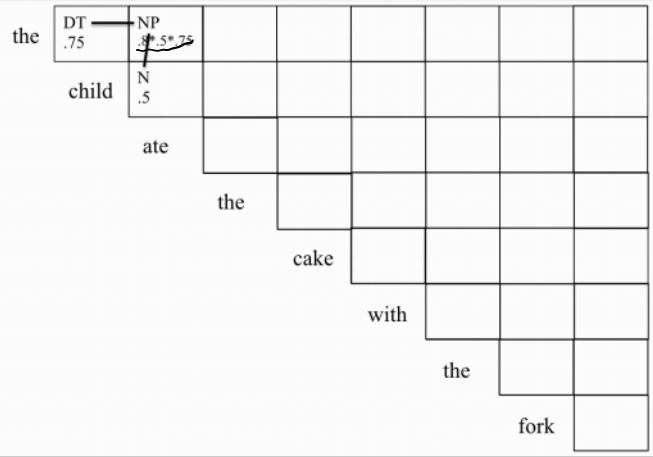





Probabilistic Version of CKY Algorithm







Question

- How, on your own, could you compute the probability of the entire sentence using Probabilistic CKY?
- Don't forget that there may be multiple parses, so you will need to add the corresponding probabilities.

NLP

Lexicalized Parsing

5.6

NLP

Introduction to NLP

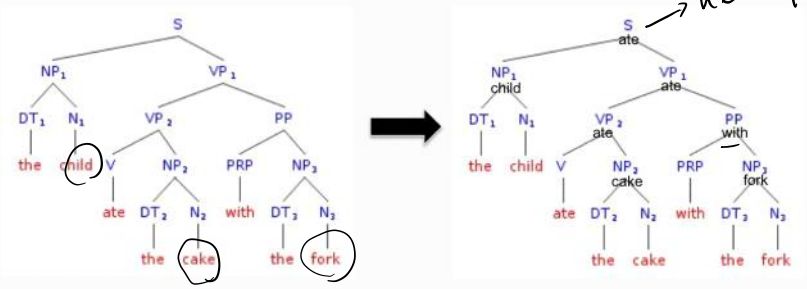
Lexicalized parsing

Limitations of PCFGs

- The probabilities don't depend on the specific words
  - E.g., *give* someone something (2 arguments) vs. *see* something (1 argument)
- It is not possible to disambiguate sentences based on semantic information
  - E.g., eat pizza with *pepperoni* vs. eat pizza with *fork*
- Lexicalized grammars – idea
  - Use the head of a phrase as an additional source of information
  - VP[ate] -> V[ate]

not refer to specific words.

Lexicalization Example



→ head of the sentence is the head of its noun phrase

Collins Parser (1999) 1/2

- Generative, lexicalized model
- Types of rules
  - LHS → $L_n L_{n-1} \ldots L_1 H R_1 \ldots R_{m-1} R_m$
  - H gets generated first
  - L gets generated next
  - R gets generated last

Collins Parser (1999) 2/2

- Maximum likelihood estimates

$$P_{ML}(\text{PPof-IN} \mid \text{VPthink-VB}) = \frac{\text{Count}(\text{PPof-IN right of the head VPthink-VB})}{\text{Count}(\text{symbols right of the head VPthink-VB})}$$

- Smoothing

$$\text{smoothedP}(\text{PPof-IN} \mid \text{VPthink-VB}) = \lambda_1 P(\text{PPof-IN} \mid \text{VPthink-VB}) + \lambda_2 P(\text{PPof-IN} \mid \text{VP-VB}) + (1-\lambda_1-\lambda_2) P(\text{PPof-IN} \mid \text{VP})$$

Smoothing

Issues With Lexicalized Grammars

- Sparseness of training data
  - Many probabilities are difficult to estimate from the Penn Treebank
- Combinatorial explosion
  - Need for parameterization

Discriminative Reranking

- A parser may return many parses of a sentence, with small differences in probabilities
- The top returned parse may not necessarily be the best because the PCFG may be deficient
- Other considerations may need to be taken into account
  - parse tree depth
  - left attachment vs. right attachment
  - discourse structure
- Can you think of others features that may affect the reranking?

Answer

- Considerations that may affect the reranking
  - parse tree depth
  - left attachment vs. right attachment
  - discourse structure
- Can you think of others?
  - consistency across sentences
  - or other stages of the NLU pipeline

Discriminative Reranking

- n-best list
  - Get the parser to produce a list of n-best parses (where n can be in the thousands)
- reranking
  - Train a discriminative classifier to rerank these parses based on external information such as a bigram probability score or the amount of right branching in the tree

Sample Performances

- F1 (sentences <= 40 words)
  - Charniak (2000) - 90.1%
  - Charniak and Johnson (2005) - 92% (discriminative reranking)

NLP

Dependency parsing is the current fashion

NLP

Introduction to NLP

Dependency Parsing

Dependency Structure

- blue
  - modifier, dependent, child, subordinate
- house
  - head, governor, parent, regent

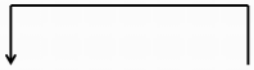
Dependency Structure

most important word in sentence

Unionized workers are usually better paid than their non-union counterparts.
1 2 3 4 5 6 7 8 9 10

05.07 Dependency Parsing NATURAL LANGUAGE PROCESSING

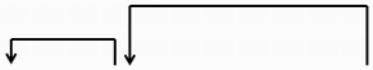
Dependency Structure



Unionized workers are usually better paid than their non-union counterparts.
1 2 3 4 5 6 7 8 9 10

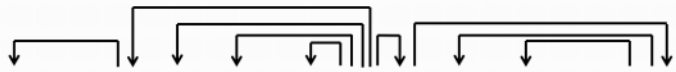
05.07 Dependency Parsing NATURAL LANGUAGE PROCESSING

Dependency Structure



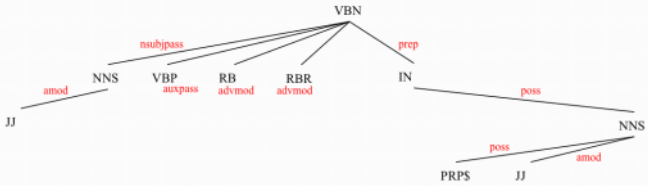
Unionized workers are usually better paid than their non-union counterparts.
1 2 3 4 5 6 7 8 9 10

Dependency Structure



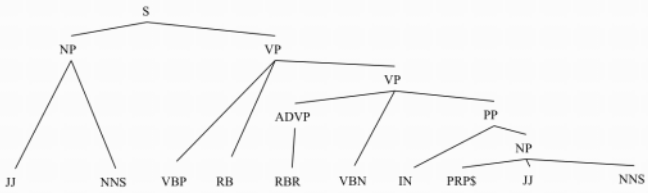
Unionized workers are usually better paid than their non-union counterparts.
1 2 3 4 5 6 7 8 9 10

Other Notations



Unionized workers are usually better paid than their non-union counterparts.
1 2 3 4 5 6 7 8 9 10

Phrase Structure



Unionized workers are usually better paid than their non-union counterparts.
1 2 3 4 5 6 7 8 9 10

Dependency Grammars

- Characteristics
  - Lexical/syntactic dependencies between words
  - The top-level predicate of a sentence is the root
  - Simpler to parse than context-free grammars
  - Particularly useful for free word order languages

How To Identify The Heads

- H=head, M=modifier
  - H determines the syntactic category of the construct
  - H determines the semantic category of the construct
  - H is required; M may be skipped
  - Fixed linear position of M with respect to H

Head Rules From Collins

Collins' PhD dissertation

Parent Non-terminal	Direction	Priority List
ADJP	Left	NNS QP NN $ ADVP JJ VBN VBG ADJP JJR NP JJS DT FW RBR RBS SBAR RB
ADVP	Right	RB RBR RBS FW ADVP TO CD JJR JJ IN NP JJS NN
CONJP	Right	CC RB IN
FRAG	Right	
INTJ	Left	
LST	Right	LS :
NAC	Left	NN NNS NNP NNPS NP NAC EX $ CD QP PRP VBG JJ JJS JJR ADJP FW
PP	Right	IN TO VBG VBN RP FW
PRN	Left	
PRT	Right	RP
QP	Left	$ IN NNS NN JJ RB DT CD NCD QP JJR JJS
RRC	Right	VP NP ADVP ADJP PP
S	Left	TO IN VP S SBAR ADJP UCP NP
SBAR	Left	WHNP WHPP WHADVP WHADJP IN DT S SQ SINV SBAR FRAG
SBARQ	Left	SQ S SINV SBARQ FRAG
SINV	Left	VBZ VBD VBP VB MD VP S SINV ADJP NP
SQ	Left	VBZ VBD VBP VB MD VP SQ
UCP	Right	
VP	Left	TO VBD VBN MD VBZ VB VBG VBP VP ADJP NN NNS NP
WHADJP	Left	CC WRB JJ ADJP
WHADVP	Right	CC WRB
WHNP	Left	WDT WP WP$ WHADJP WHPP WHNP
WHPP	Right	IN TO FW

Table A.1: The head-rules used by the parser. *Parent* is the non-terminal on the left-hand-side of a rule. *Direction* specifies whether search starts from the left or right end of the rule. *Priority* gives a priority ranking, with priority decreasing when moving down the list.

Techniques (1)

- Dynamic programming
  - CKY – similar to lexicalized PCFG, cubic complexity (Eisner 96)

Cubic complexity.

predicate.

Techniques (2)

- Constraint-based methods
  - Maruyama 1990, Karlsson 1990
  - Example
    - word(pos(x)) = DET ⇒(label(X) = NMOD, word(mod(x)) = NN, pos(x) < mod(x))
    - A determiner (DET) modifies a noun (NN) on the right with the label NMOD.
  - NP complete problem; heuristics needed
- Constraint graph
  - For initial constraint graph using a core grammar: nodes, domains, constraints
  - Find an assignment that doesn't contradict any constraints. If more than one assignment exists, add more constraints.

problematic.

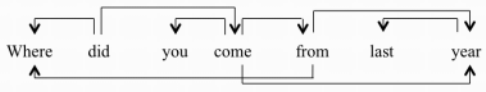
Techniques (3)

- Deterministic parsing
  - Covington 2001
  - MaltParser by Nivre
    - shift/reduce as in a shift/reduce parser
    - reduce creates dependencies with the head on either the left or the right
- Graph-based methods
  - Maximum spanning trees (MST)
    - MST Parser by McDonald et al. 2005

Techniques based on graph

Non-projectivity

Output of (the non-projective) MSTParser



Output of Stanford parser

advmod(come-4, Where-1)
aux(come-4, did-2)
nsubj(come-4, you-3)
root(ROOT-0, come-4)
prep(come-4, from-5)
amod(year-7, last-6)
pobj(from-5, year-7)

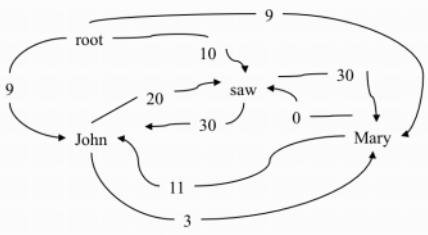
1	Where	Where	WRB	WRB	-	2	SBJ	-	-
2	did	did	VBD	VBD	-	0	ROOT	-	-
3	you	you	PRP	PRP	-	4	SBJ	-	-
4	come	come	VBP	VBP	-	2	VC	-	-
5	from	from	IN	IN	-	4	DIR	-	-
6	last	last	JJ	JJ	-	7	NMOD	-	-
7	year	year	NN	NN	-	5	PMOD	-	-
8	?	?	.	.	-	2	P	-	-

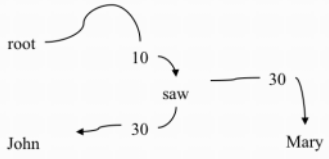
Dependency Parsing

- Background
  - McDonald et al. 2005
- Projectivity
  - English dependency trees are mostly projective (can be drawn without crossing dependencies).
  - Other languages are not.
- Idea
  - Dependency parsing is equivalent to search for a maximum spanning tree in a directed graph.
  - Chu and Liu (1965) and Edmonds (1967) give an efficient algorithm for finding MST for directed graphs.

MST Parser Example

- Consider the sentence "John saw Mary"
- The Chu-Liu-Edmonds algorithm gives the MST on the right hand side (right). This is in general a non-projective tree.





MaltParser (Nivre 2008)

- Very similar to shift-reduce parsing.
- It includes the following components
  - A stack
  - A buffer
  - Set of dependencies (arcs)
- The reduce operations combine an element from the stack and one from the buffer
- Arc-eager parser
  - The actions are shift, reduce, left-arc, right-arc

MaltParser Actions

$$\text{Shift} \quad \frac{[\ldots]_S \quad [w_i, \ldots]_Q}{[\ldots, w_i]_S \quad [\ldots]_Q}$$

$$\text{Reduce} \quad \frac{[\ldots, w_i]_S \quad [\ldots]_Q \quad \exists w_k : w_k \rightarrow w_i}{[\ldots]_S \quad [\ldots]_Q}$$

$$\text{Left-Arc}_r \quad \frac{[\ldots, w_i]_S \quad [w_j, \ldots]_Q \quad \neg\exists w_k : w_k \rightarrow w_i}{[\ldots]_S \quad [w_j, \ldots]_Q \quad w_i \overset{r}{\leftarrow} w_j}$$

$$\text{Right-Arc}_r \quad \frac{[\ldots, w_i]_S \quad [w_j, \ldots]_Q \quad \neg\exists w_k : w_k \rightarrow w_j}{[\ldots, w_i, w_j]_S \quad [\ldots]_Q \quad w_i \overset{r}{\rightarrow} w_j}$$

[Example from Nivre and Kuebler]

Example

- People want to be free
  - [ROOT] [People, want, to, be, free] Ø
  - Shift [ROOT, People] [want, to, be, free]
  - LA_{nsubj} [ROOT] [want, to, be, free] $A_1 = \{nsubj(want, people)\}$
  - RA_{root} [ROOT, want] [to, be, free] $A_2 = A_1 \cup \{root(ROOT, want)\}$
- The next action is chosen using a classifier
- There is no search
- The final list of arcs is returned as the dependency tree
- Very fast method

Evaluation Metric

- Labeled dependency accuracy
- # correct deps/# deps

1	Unionized	Unionized	VBN	VBN	-	2	NMOD	-	-
2	workers	workers	NNS	NNS	-	3	SBJ	-	-
3	are	are	VBP	VBP	-	0	ROOT	-	-
4	usually	usually	RB	RB	-	3	TMP	-	-
5	better	better	RBR	RBR	-	4	ADV	-	-
6	paid	paid	VBN	VBN	-	5	AMOD	-	-
7	than	than	IN	IN	-	5	AMOD	-	-
8	their	their	PRP$	PRP$	-	10	NMOD	-	-
9	non-union	non-union	JJ	JJ	-	10	NMOD	-	-
10	counterparts	counterparts	NNS	NNS	-	7	PMOD	-	-

Complexity

- Projective (CKY) $O(n^5)$
- Projective (Eisner) $O(n^3)$
- Non-projective (MST – Chu-Liu-Edmonds) $O(n^2)$
- Projective (Malt) $O(n)$

Use In Information Extraction

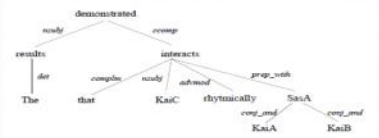


Figure 1: The dependency tree of the sentence "*The results demonstrated that KaiC interacts rhythmically with KaiA, KaiB, and SasA.*"

1. KaiC - nsubj - interacts - prep_with - SasA
2. KaiC - nsubj - interacts - prep_with - SasA - conj_and - KaiA
3. KaiC - nsubj - interacts - prep_with – SasA - conj_and - KaiB
4. SasA - conj_and - KaiA
5. SasA - conj_and - KaiB
6. KaiA – conj_and – SasA - conj_and - KaiB

1. *PROTX1* - nsubj - interacts - prep_with - *PROTX2*
2. *PROTX1* - nsubj - interacts - prep_with - *PROTX0* - conj_and - *PROTX2*
3. *PROTX1* - nsubj - interacts - prep_with – *PROTX0* - conj_and - *PROTX2*
4. *PROTX1* - conj_and - *PROTX2*
5. *PROTX1* - conj_and - *PROTX2*
6. *PROTX1* – conj_and – *PROTX0* - conj_and - *PROTX2*

[Erkan et al. 2007]

info extraction

Dependency Kernels

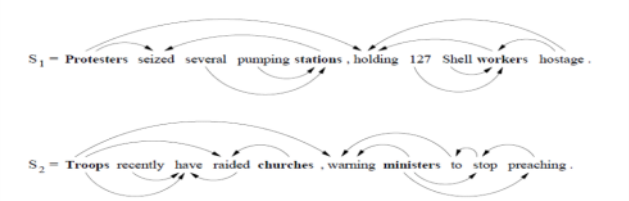


Figure 1: Sentences as dependency graphs.

Relation Instance	Shortest Path in Undirected Dependency Graph
S_1: protesters AT stations	**protesters** ⟶ seized ⟵ **stations**
S_1: workers AT stations	**workers** ⟶ holding ⟵ protesters ⟶ seized ⟵ **stations**
S_2: troops AT churches	**troops** ⟶ raided ⟵ **churches**
S_2: ministers AT churches	**ministers** ⟶ warning ⟵ troops ⟶ raided ⟵ **churches**

Table 1: Shortest Path representation of relations.

[Bunescu and Mooney 2005]

Decide the similarity of two sentences based on the similarity of their dependency structure.

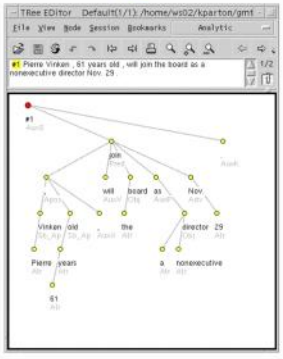
External Links

- http://ilk.uvt.nl/conll/
  - CONLL-X Shared task
- http://ufal.mff.cuni.cz/pdt2.0/
  - Prague Dependency Treebank
- http://nextens.uvt.nl/depparse-wiki/SharedTaskWebsite
- http://nextens.uvt.nl/depparse-wiki/DataOverview
- http://maltparser.org/
  - Joakim Nivre's Maltparser
- http://www.cs.ualberta.ca/~lindek/minipar.htm
  - Dekang Lin's Minipar
- http://www.link.cs.cmu.edu/link/
  - Daniel Sleator and Davy Temperley's Link parser

multiple linguage data + paper.

wiki's of dep. parsings

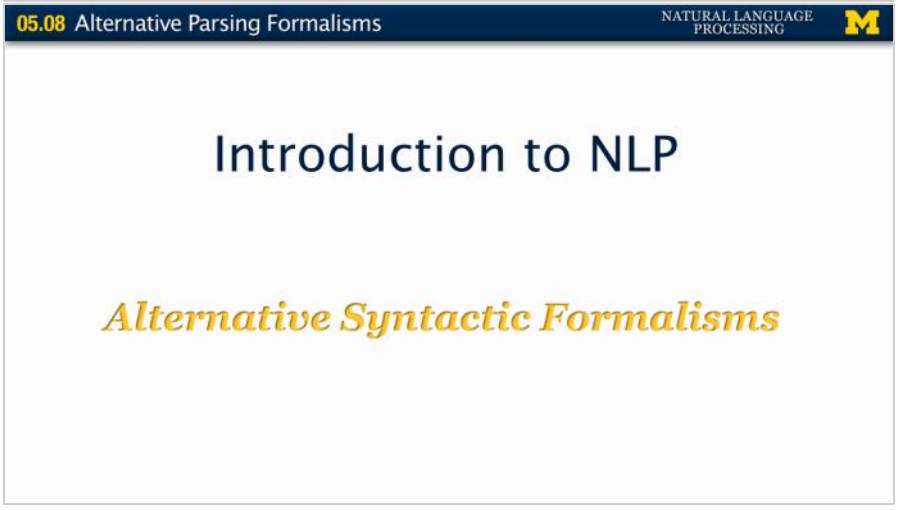
Prague Dependency Treebank Example




05.07 Dependency Parsing
NATURAL LANGUAGE PROCESSING
NLP

5.8


05.08 Alternative Parsing Formalisms
NATURAL LANGUAGE PROCESSING
NLP

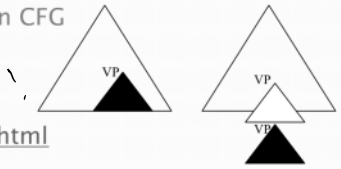

05.08 Alternative Parsing Formalisms
NATURAL LANGUAGE PROCESSING
Introduction to NLP
Alternative Syntactic Formalisms

Mildly Context-Sensitive Grammars

- Tree Substitution Grammar (TSG)
  - Terminals generate entire tree fragments
  - TSG and CFG are formally equivalent
- Tree Adjoining Grammar (TAG)
- Combinatory Categorial Grammar (CCG)

Tree Adjoining Grammar (TAG)

- Like TSG but allow adjunction
- It can generate languages like $a^n b^n c^n$ or ww (cross-serial dependencies):
  - e.g., Mary gave a book and a magazine to Chen and Mike, respectively.
- Expressive power
  - TAG is formally more powerful than CFG
  - TAG is less powerful than CSG
- Card game online!
  - http://www.ltaggame.com/
  - http://www.ltaggame.com/family.html



→ Cross-serial dependencies (book – Chen, magazine – Mike) can't be parsed by CFGs.

Combinatory Categorial Grammar (CCG)

- Complex types
  - E.g., X/Y and $X\backslash Y$
  - These take an argument of type Y and return an object of type X.
  - X/Y – means that Y should appear on the right
  - X\Y – means that Y should appear on the left
- Expressive power
  - CCGs can generate the language $a^n b^n c^n d^n$, $n>0$

I	NP
books	NP
sleep	S \ NP
enjoy	(S \ NP) / NP

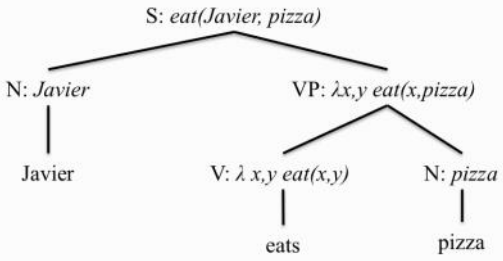
Example from Jonathan Kummerfeld, Aleka Blackwell, and Patrick Littell

Introduction to NLP

Semantic parsing

Semantic Parsing

- Associate a semantic expression with each node



All articles hottest area in recent years.

Introduction to NLP

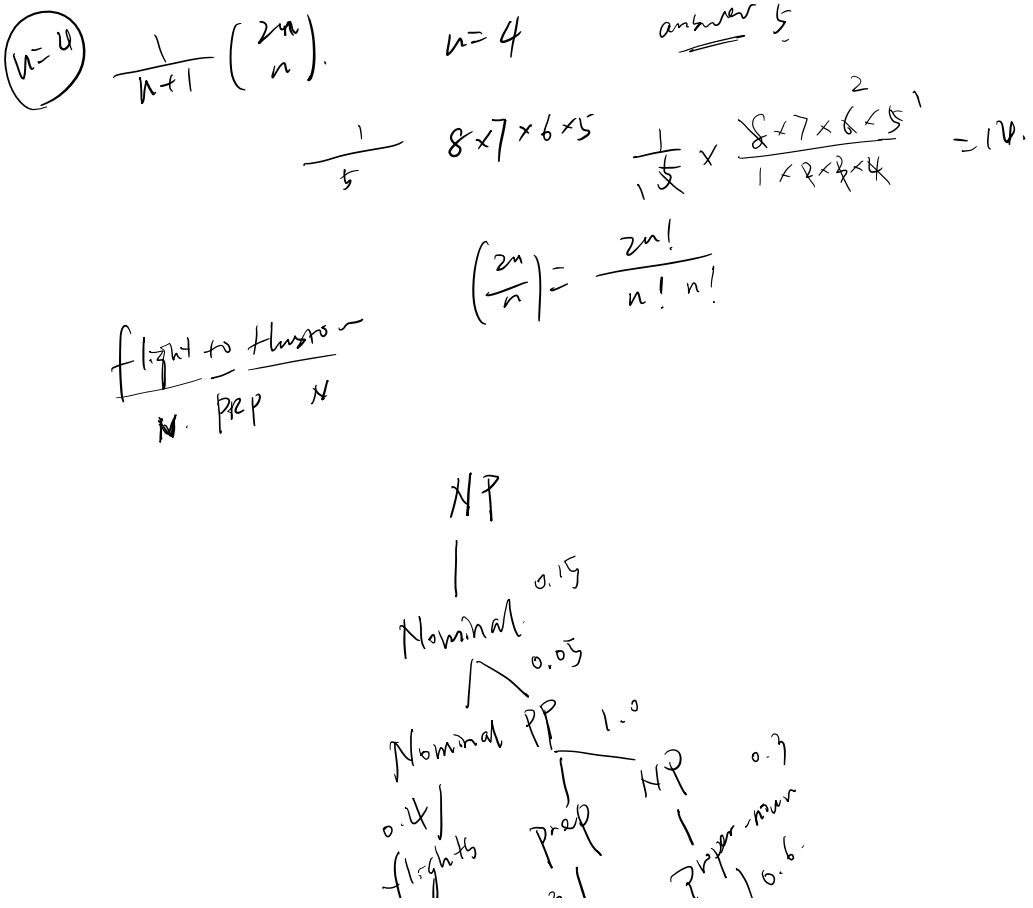
NACLO problems on parsing

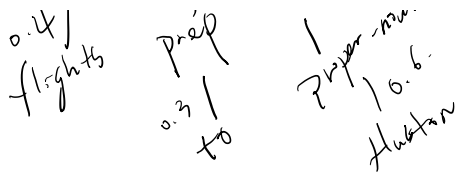
External Links

- Twodee (by Jason Eisner)
  - http://nacloweb.org/resources/problems/2013/N2013-H.pdf
- One, Two, Tree (by Noah Smith, Kevin Gimbel, and Jason Eisner)
  - http://www.nacloweb.org/resources/problems/2012/N2012-R.pdf
- CCG (by Jonathan Kummerfeld, Aleka Blackwell, and Patrick Littell)
  - http://www.nacloweb.org/resources/problems/2014/N2014-O.pdf
- Combining categories in Tok Pisin (same authors)
  - http://www.nacloweb.org/resources/problems/2014/N2014-P.pdf
- Grammar Rules (Andrea Schalley and Pat Littell)
  - http://www.nacloweb.org/resources/problems/2013/N2013-F.pdf
- Sk8 Parser (Pat Littell)
  - http://www.nacloweb.org/resources/problems/2009/N2009-G.pdf

05.08 Alternative Parsing Formalisms NATURAL LANGUAGE PROCESSING

NLP




0.4
flights
PROP
0.3
to
Houston