CMPT-413
Computational Linguistics

Anoop Sarkar
http://www.cs.sfu.ca/∼anoop

March 5, 2012

Prepositional Phrases

- noun attach: *I bought the shirt with pockets*
- verb attach: *I washed the shirt with soap*
- As in the case of other attachment decisions in parsing: it depends on the meaning of the entire sentence – needs world knowledge, etc.
- Maybe there is a simpler solution: we can attempt to solve it using heuristics or associations between words

Structure Based Ambiguity Resolution

- Right association: a constituent (NP or PP) tends to attach to another constituent immediately to its right (Kimball 1973)
- Minimal attachment: a constituent tends to attach to an existing non-terminal using the fewest additional syntactic nodes (Frazier 1978)
- These two principles make opposite predictions for prepositional phrase attachment
- Consider the grammar:

$$VP \rightarrow V\ NP\ PP \tag{1}$$
$$NP \rightarrow NP\ PP \tag{2}$$

for input: *I [$_{VP}$ saw [$_{NP}$ the man ... [$_{PP}$ with the telescope]*, RA predicts that the PP attaches to the NP, i.e. use rule (2), and MA predicts V attachment, i.e. use rule (1)

Structure Based Ambiguity Resolution

- Garden-paths look structural:
  The emergency crews hate most is domestic violence
- Neither MA or RA account for more than 55% of the cases in real text
- Psycholinguistic experiments using eyetracking show that humans resolve ambiguities as soon as possible in the left to right sequence using the words to disambiguate
- Garden-paths are caused by a combination of lexical and structural effects:
  The flowers delivered for the patient arrived

Ambiguity Resolution: Prepositional Phrases in English

- Learning Prepositional Phrase Attachment: Annotated Data

v	n1	p	n2	Attachment
join	board	as	director	V
is	chairman	of	N.V.	N
using	crocidolite	in	filters	V
bring	attention	to	problem	V
is	asbestos	in	products	N
making	paper	for	filters	N
including	three	with	cancer	N
⋮	⋮	⋮	⋮	⋮

Prepositional Phrase Attachment

Method	Accuracy
Always noun attachment	59.0
Most likely for each preposition	72.2
Average Human (4 head words only)	88.2
Average Human (whole sentence)	93.2

Back-off Smoothing

- Let 1 represent noun attachment.
- We want to compute probability of noun attachment: $p(1 \mid v, n1, p, n2)$.
- Probability of verb attachment is $1 - p(1 \mid v, n1, p, n2)$.

Back-off Smoothing

1. If $f(v, n1, p, n2) > 0$ and $\hat{p} \neq 0.5$

$$\hat{p}(1 \mid v, n1, p, n2) = \frac{f(1, v, n1, p, n2)}{f(v, n1, p, n2)}$$

2. Else if $f(v, n1, p) + f(v, p, n2) + f(n1, p, n2) > 0$ and $\hat{p} \neq 0.5$

$$\hat{p}(1 \mid v, n1, p, n2) = \frac{f(1, v, n1, p) + f(1, v, p, n2) + f(1, n1, p, n2)}{f(v, n1, p) + f(v, p, n2) + f(n1, p, n2)}$$

3. Else if $f(v, p) + f(n1, p) + f(p, n2) > 0$

$$\hat{p}(1 \mid v, n1, p, n2) = \frac{f(1, v, p) + f(1, n1, p) + f(1, p, n2)}{f(v, p) + f(n1, p) + f(p, n2)}$$

4. Else if $f(p) > 0$

$$\hat{p}(1 \mid v, n1, p, n2) = \frac{f(1, p)}{f(p)}$$

5. Else $\hat{p}(1 \mid v, n1, p, n2) = 1.0$

Prepositional Phrase Attachment: (Collins and Brooks 1995)

- **Results**: 84.5% accuracy
  with the use of some limited word classes for dates, numbers, etc.
- Using complex word classes taken from `WordNet` (which we shall be looking at later in this course) increases accuracy to 88% (Stetina and Nagao 1998)
- We can improve on parsing performance with Probabilistic CFGs by using the insights taken from PP attachment.
- Modify the PCFG model to be sensitive to words and other context-sensitive features of the input.
- And generalizing to other kinds of attachment problems, like coordination or deciding which constituent is an argument of a verb.

Some other studies

- **Toutanova, Manning, and Ng, 2004**:
  use sophisticated smoothing model for PP attachment
  86.18% with words & stems; with word classes: 87.54%
- **Merlo, Crocker and Berthouzoz, 1997**:
  test on multiple PPs, generalize disambiguation of 1 PP to 2-3 PPs
  14 structures possible for 3PPs assuming a single verb: all 14 are attested in the Treebank
  same model as CB95; but generalized to dealing with upto 3PPs
  1PP: 84.3%   2PP: 69.6%   3PP: 43.6%
  Note that this is still not the real problem faced in parsing natural language

Probability Models

- $p(x, y)$: x = input, y = labels
- Pick best prob distribution $p(x, y)$ to fit the data
- Max likelihood of the data *according to the prob model* equivalent to minimizing entropy

Probability Models

- Max likelihood of the data *according to the prob model*
- Equivalent to picking best parameter values θ such that the data gets highest likelihood:

$$\mathrm{max}_{\theta} p(\theta \mid \mathrm{data}) = \mathrm{max}_{\theta} p(\theta) \cdot p(\mathrm{data} \mid \theta)$$

Advantages of probability models

- parameters can be estimated automatically, while scores have to twiddled by hand
- parameters can be estimated from supervised or unsupervised data
- probabilities can be used to quantify confidence in a particular state and used to compare against other probabilities in a strictly comparable setting
- modularity: $p(\textit{semantics}) \cdot p(\textit{syntax} \mid \textit{semantics}) \cdot p(\textit{morphology} \mid \textit{syntax}) \cdot p(\textit{phonology} \mid \textit{morphology}) \cdot p(\textit{sounds} \mid \textit{phonology})$

Naive Bayes Classifier

- $\mathbf{x}$ is the input that can be represented as d independent features f_j, $1 \leq j \leq d$
- y is the output classification
- $P(y \mid \mathbf{x}) = \frac{P(y) \cdot P(\mathbf{x} \mid y)}{P(\mathbf{x})}$
- $P(\mathbf{x} \mid y) = \prod_{j=1}^{d} P(f_j \mid y)$
- $P(y \mid \mathbf{x}) = P(y) \cdot \prod_{j=1}^{d} P(f_j \mid y)$

Using Naive Bayes for Document Classification

- Spam text: `Learn how to make $38.99 into a money making machine that pays ... $7,000 / month !`
- Distinguish spam text from regular email text
- Find useful features to make this distinction

Using Naive Bayes

- Useful features
  1. contains `turn $AMOUNT into`
  2. contains `$AMOUNT`
  3. contains `Learn how to`
  4. contains exclamation mark at end of sentence

Using Naive Bayes

- how many times do these features occur?
  1. contains: `turn $AMOUNT into`
     in spam text: 50
     in normal email: 2
     i.e. 25x more likely in spam
  2. contains: `$AMOUNT`
     in spam text: 90
     in normal email: 10
     i.e. 9x more likely in spam

Using Naive Bayes

- How likely is it for *both* features to occur at the same time in a spam message?
  1. contains: `turn $AMOUNT into`
  2. contains: `$AMOUNT`
- Assume we have a new feature, contains: `turn $AMOUNT into` *and* `$AMOUNT`
- The model predicts that the event that both features occur simultaneously has probability $\frac{140}{152} = 0.92$
- But Naive Bayes assumes that these features are independent and should occur with probability: $0.92 \cdot 0.9 = 0.864$

Using Naive Bayes

- Naive Bayes needs overlapping but independent features
- How can we use all of the features we want?
  1. contains `turn $AMOUNT into`
  2. contains `$AMOUNT`
  3. contains `Learn how to`
  4. contains exclamation mark at end of sentence
- how about giving each feature a weight w equal to its log probability: $w = \log p(f, y)$

Using Naive Bayes

- each feature gets a score equal to its log probability
- Assign scores to features:
  1. $w_1 = +1$ contains `turn $AMOUNT into`
  2. $w_2 = +5$ contains `$AMOUNT`
  3. $w_3 = +0.2$ contains `Learn how to`
  4. $w_4 = -2$ contains exclamation mark at end of sentence

Using Naive Bayes

- so add the scores and treat it like a log probability
- $\log p(\textit{spam} \mid \textit{feats}) = 4.2$
- but then, $p(\textit{spam} \mid \textit{feats}) = \textit{exp}(4.2) = 66.68$
- how do we compute keep arbitrary scores and still get probabilities?

Log linear model

- Let there be m features, $f_k(\mathbf{x}, y)$ for $k = 1, \ldots, m$
- Define a parameter vector $\mathbf{w} \in \mathbb{R}^m$
- Each $(\mathbf{x}, y)$ pair is mapped to score:

$$s(\mathbf{x}, y) = \sum_k w_k \cdot f_k(\mathbf{x}, y)$$

- Using inner product notation:

$$\begin{aligned} \mathbf{w} \cdot \mathbf{f}(\mathbf{x}, y) &= \sum_k w_k \cdot f_k(\mathbf{x}, y) \\ s(\mathbf{x}, y) &= \mathbf{w} \cdot \mathbf{f}(\mathbf{x}, y) \end{aligned}$$

- To get a probability from the score: Renormalize!

$$\Pr(y \mid \mathbf{x}, \mathbf{w}) = \frac{\textit{exp}\left(s(\mathbf{x}, y)\right)}{\sum_{y'} \textit{exp}\left(s(\mathbf{x}, y')\right)}$$

Log linear model

- The name 'log-linear model' comes from:

$$\log \Pr(y \mid \mathbf{x}, \mathbf{w}) = \underbrace{\mathbf{w} \cdot \mathbf{f}(\mathbf{x}, y)}_{\text{linear term}} - \underbrace{\log \sum_{y'} exp\left(\mathbf{w} \cdot \mathbf{f}(\mathbf{x}, y')\right)}_{\text{normalization term}}$$

- Once the weights are learned, we can perform predictions using these features.
- The goal: to find $\mathbf{w}$ that maximizes the log likelihood $L(\mathbf{w})$ of the labeled training set containing $(\mathbf{x}_i, y_i)$ for $i = 1 \ldots n$

$$\begin{aligned} L(\mathbf{w}) &= \sum_i \log \Pr(y_i \mid \mathbf{x}_i, \mathbf{w}) \\ &= \sum_i \mathbf{w} \cdot \mathbf{f}(\mathbf{x}_i, y_i) - \sum_i \log \sum_{y'} exp\left(\mathbf{w} \cdot \mathbf{f}(\mathbf{x}_i, y')\right) \end{aligned}$$

Log linear model

- Maximize:

$$L(\mathbf{w}) = \sum_i \mathbf{w} \cdot \mathbf{f}(\mathbf{x}_i, y_i) - \sum_i \log \sum_{y'} exp\left(\mathbf{w} \cdot \mathbf{f}(\mathbf{x}_i, y')\right)$$

- Calculate gradient:

$$\begin{aligned} &\left.\frac{dL(\mathbf{w})}{d\mathbf{w}}\right|_{\mathbf{w}} \\ &= \sum_i \mathbf{f}(\mathbf{x}_i, y_i) - \sum_i \frac{1}{\sum_{y''} exp\left(\mathbf{w} \cdot \mathbf{f}(x_i, y'')\right)} \\ &\quad \sum_{y'} \mathbf{f}(\mathbf{x}_i, y') \cdot exp\left(\mathbf{w} \cdot \mathbf{f}(\mathbf{x}_i, y')\right) \\ &= \sum_i \mathbf{f}(\mathbf{x}_i, y_i) - \sum_i \sum_{y'} \mathbf{f}(\mathbf{x}_i, y') \frac{exp\left(\mathbf{w} \cdot \mathbf{f}(\mathbf{x}_i, y')\right)}{\sum_{y''} exp\left(\mathbf{w} \cdot \mathbf{f}(x_i, y'')\right)} \\ &= \underbrace{\sum_i \mathbf{f}(\mathbf{x}_i, y_i)}_{\text{Observed counts}} - \underbrace{\sum_i \sum_{y'} \mathbf{f}(\mathbf{x}_i, y') \Pr(y' \mid \mathbf{x}_i, \mathbf{w})}_{\text{Expected counts}} \end{aligned}$$

Log linear model

- Init: $\mathbf{w}^{(0)} = \mathbf{0}$
- $t \leftarrow 0$
- Iterate until convergence:
  - Calculate: $\Delta = \frac{dL(\mathbf{w})}{d\mathbf{w}}\Big|_{\mathbf{w}=\mathbf{w}^{(t)}}$
  - Find $\beta^* = \text{argmax}_\beta L(\mathbf{w}^{(t)} + \beta\Delta)$
  - Set $\mathbf{w}^{(t+1)} \leftarrow \mathbf{w}^{(t)} + \beta^*\Delta$

Learning the weights: **w**: Generalized Iterative Scaling

$f^{\#} = max_{x,y} \sum_{j=1}^{k} f_j(x, y)$
For each iteration
    expected[1 .. # of features] $\leftarrow 0$
    For i = 1 to | training data |
        For each feature f_j
            expected[j] += $f_j(x_i, y_i) \cdot P(y_i \mid x_i)$
    For each feature f_j
        observed[j] = $f_j(x, y) \cdot \frac{c(x,y)}{|\text{training data}|}$
    For each feature f_j
        $w_j \leftarrow w_j \cdot \sqrt[f^{\#}]{\frac{\text{observed[j]}}{\text{expected[j]}}}$

cf. Goodman, NIPS '01

Maximum Entropy

- The log-linear model has an interpretation as a *maximum entropy* model.
- For observed events, maximize likelihood. For unobserved events, from all consistent models pick the one with maximum entropy.
- The maximum entropy principle: related to Occam's razor and other similar justifications for scientific inquiry
- Make the minimum possible assumptions about unseen data
- Also: Laplace's *Principle of Insufficient Reason*: when one has no information to distinguish between the probability of two events, the best strategy is to consider them equally likely