Problems with approaches we’ve seen so far

Naive Bayes classifiers for text categorization, WSD, etc:

- independence assumptions between features can be problematic

PCFGs for parsing:

- context-free (structural) independence assumptions can be problematic
- difficult to model lexical effects (e.g., on PP attachment)

For parsing, more complex grammars can help

- parent annotation or otherwise splitting non-terminals (see J&M 14.5)
- lexicalized rules (see J&M 14.6-14.6.1)
- but grammars get huge, parsers get slow, need very carefully designed smoothing... and we still can’t handle, e.g., unbounded dependencies.
- maybe try something completely different?

A different approach to modeling

- so far, all our models have been generative
- **discriminative** models can address some of the above issues (although they will introduce others)
Generative probabilistic models

- Model the joint probability $P(\vec{x}, \vec{y})$
  - $\vec{x}$: the observed variables
  - $\vec{y}$: the latent variables

<table>
<thead>
<tr>
<th>Model</th>
<th>$\vec{x}$</th>
<th>$\vec{y}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Naive Bayes</td>
<td>features</td>
<td>classes</td>
</tr>
<tr>
<td>HMM</td>
<td>words</td>
<td>tags</td>
</tr>
<tr>
<td>PCFG</td>
<td>words</td>
<td>tree</td>
</tr>
</tbody>
</table>

Generative models have a “generative story”

- a probabilistic process that describes how the data were created
  - Multiplying probabilities of each step gives us $P(\vec{x}, \vec{y})$.

- Naive Bayes: For each item $i$ to be classified,
  - Generate its class $c_i$
  - Generate its features $f_{i1} \ldots f_{in}$ conditioned on $c_i$

Result:

$$P(\vec{c}, \vec{f}) = \prod_i P(c_i) \prod_j P(f_{ij}|c_i)$$

Other generative stories

- HMM: For each position $i$ in sentence,
  - Generate its tag $t_i$ conditioned on previous tag $t_{i-1}$
  - Generate its word $w_i$ conditioned on $t_i$

- PCFG:
  - Starting from S node, recursively generate children for each phrasal category $c_i$ conditioned on $c_i$, until all unexpanded nodes are pre-terminals (tags).
  - For each pre-terminal $t_i$, generate a word $w_i$ conditioned on $t_i$. 
Inference in generative models

- At test time, given only $\vec{x}$, infer $\vec{y}$ using Bayes’ rule:
  $$P(\vec{y}|\vec{x}) = \frac{P(\vec{x}|\vec{y})P(\vec{y})}{P(\vec{x})}$$

- So, notice we actually model $P(\vec{x}, \vec{y})$ as $P(\vec{x}|\vec{y})P(\vec{y})$.
  - You can confirm this for each of the previous models.

Discriminative probabilistic models

- Model $P(\vec{y}|\vec{x})$ directly
- No model of $P(\vec{x}, \vec{y})$
- No generative story
- No Bayes’ rule

Discriminative models more broadly

- Trained to discriminate correct vs. wrong values of $\vec{y}$, given input $\vec{x}$.
- Need not be probabilistic.
- Examples: support vector machines, artificial neural networks, decision trees, nearest neighbor methods.
- Here, we consider only one method: Maximum Entropy (MaxEnt) models.

MaxEnt classifiers

- Used widely in many different fields, under many different names
- Most commonly, multinomial logistic regression
  - multinomial if more than two possible classes
  - otherwise (or if lazy) just logistic regression
- Also: log-linear model, single neuron classifier, etc …
WSD as example classification task

- disambiguate three senses of the target word plant
- \( \vec{x} \) are the words and POS tags in the document the target word occurs in
- \( y \) is the latent sense. Assume three possibilities:

<table>
<thead>
<tr>
<th>y</th>
<th>Sense</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Noun: a member of the plant kingdom</td>
</tr>
<tr>
<td>2</td>
<td>Verb: to place in the ground</td>
</tr>
<tr>
<td>3</td>
<td>Noun: a factory</td>
</tr>
</tbody>
</table>

Defining a MaxEnt model

- Define features \( f_i(\vec{x}, y) \) that depend on both observed and latent variables.
- Each feature \( f_i \) has a real-valued weight \( w_i \) (learned in training).

\[
\begin{align*}
 f_1 &: \text{POS(tgt) = NN} & y = 1 \\
 f_2 &: \text{POS(tgt) = NN} & y = 2 \\
 f_3 &: \text{preceding_word(tgt) = 'chemical'} & y = 3 \\
 f_4 &: \text{doc_contains('animal')} & y = 1 \\
\end{align*}
\]

where \( \text{tgt} \) is the target word

- For senses \{1: member of plant kingdom; 2: put in ground; 3: factory\}, which weights are likely to be positive? Negative?

Feature templates

- In practice, features are usually defined using templates

\[
\begin{align*}
\text{POS(tgt)} &= t & \text{& y} \\
\text{preceding_word(tgt)} &= w & \text{& y} \\
\text{doc_contains(w)} &= w & \text{& y} \\
\end{align*}
\]

- instantiate with all possible POSs \( t \) or words \( w \) and classes \( y \)
- usually filter out features occurring very few times
- templates can also define real-valued or integer-valued features

- NLP tasks often have a few templates, but 1000s or 10000s of features

Classification with MaxEnt

- Choose the class that has highest probability according to

\[
P(y|\vec{x}) = \frac{1}{Z} \exp \left( \sum_i w_i f_i(\vec{x}, y) \right)
\]

where

\[
\begin{align*}
\exp(x) &= e^x \\
\sum_i w_i f_i & \text{ is the dot product of vectors } \vec{w} \text{ and } \vec{f}, \text{ also written } \vec{w} \cdot \vec{f}. \\
The \text{normalization constant } Z &= \sum_{y'} \exp(\sum_i w_i f_i(\vec{x}, y'))
\end{align*}
\]
Which features are active?

- Example doc: [... animal/NN ... chemical/JJ plant/NN ...]

\[ P(y = 1|\vec{x}) \] will have \( f_1, f_4 = 1 \) and \( f_2, f_3 = 0 \)
\[ P(y = 2|\vec{x}) \] \( f_2 = 1 \) \( f_1, f_3, f_4 = 0 \)
\[ P(y = 3|\vec{x}) \] \( f_3 = 1 \) \( f_1, f_2, f_4 = 0 \)
\[ P(y = 4|\vec{x}) \] \( f_1, f_2, f_3, f_4 = 0 \)

- Notice that zero-valued features have no effect on the final probability

- Other features will be multiplied by their weights, summed, then exp.

Training the model

- Given annotated data, choose weights that make the labels most probable under the model.

- That is, given items \( x^{(1)} \ldots x^{(N)} \) with labels \( y^{(1)} \ldots y^{(N)} \), choose

\[ \hat{w} = \operatorname{argmax}_{\vec{w}} \sum_j \log P(y^{(j)}|x^{(j)}) \]

- called conditional maximum likelihood estimation (CMLE)

- Like MLE, CMLE will overfit, so we use tricks (regularization) to avoid that.

The downside to MaxEnt models

- Supervised MLE in generative models is easy: compute counts and normalize.

- Supervised CMLE in MaxEnt model not so easy
  - requires multiple iterations over the data to gradually improve weights (using gradient ascent).
  - each iteration computes \( P(y^{(j)}|x^{(j)}) \) for all \( j \) and each possible \( y^{(j)} \).
  - not too bad when there are only a handful of classes \( y \). BUT...
MaxEnt for parsing

- Now we have $y =$ parse tree, $x =$ sentence.
- Features can mirror parent-annotated/lexicalized PCFG:
  - counts of each CFG rule used in $y$
  - pairs of words in head-head dependency relations in $y$
  - each word in $x$ with its parent and grandparent categories in $y$.

Global features

- Features can also capture global structure. Ex (Charniak and Johnson, 2005):
  - length difference of coordinated conjuncts

I gave him the book about NLP
vs
I gave the book about NLP to him

Features for parsing

- Altogether, Charniak and Johnson (2005) use 13 feature templates
- with a total of 1,148,697 features
- and that is after filtering out features occurring fewer than five times
Training a MaxEnt parser (naively)

For each iteration $i$
  For each sentence $x$
    For each parse tree $y$
      extract features
      compute probabilities

Training a MaxEnt parser (naively)

For each iteration $i$ ⇐ 10s
  For each sentence $x$ ⇐ 10,000s
    For each parse tree $y$ ⇐ 1000s-10,000s
      extract features ⇐ millions
      compute probabilities

Discriminative re-ranking

- Instead of learning to discriminate best parse from all parses, discriminate from other plausible parses.
- Use generative parser to produce $n$-best list of parses.
- Train MaxEnt model to re-rank the list.
  - now only $n = 50$ parses instead of thousands.

ACK!
Parser performance

• $F_1$-measure (from precision/recall on constituents) on WSJ test set:
  
  - standard PCFG: $\sim$80%
  - lexicalized PCFG (Charniak, 2000): 89.7%
  - re-ranked LPCFG (Charniak and Johnson, 2005): 91.0%

• Fully discriminative parsers have now been developed; performance is similar.
• Best WSJ parser is now just over 92%
• But performance on other domains and (esp) other languages is much worse.

Summary of model types

• Probabilistic generative models:
  - model $P(x, y)$ by assuming a generative process
  - easy to train but independence assumptions can be problematic
• Probabilistic discriminative models:
  - model $P(y|x)$ only, have no generative process
  - can incorporate arbitrary local/global features, including correlated ones
  - much slower to train; may require pre-filtering of alternatives ($n$-best list)
• Both kinds of model are currently in use for many NLP tasks.

References
