Recap: dependency parsing

Last time we said:

- Transition-based dependency parser uses a shift-reduce algorithm to efficiently parse sentences, producing projective structures.
- Can either be greedy or use beam search.
- Either way, we need a model to tell us $P(\text{action}|\text{configuration})$.
- Any classifier could do, but we’ll use logistic regression.

Today’s lecture

- A simpler example to illustrate logistic regression.
- How would we apply it to dependency parsing?
- How can we use it for reranking?

Example task: word sense disambiguation

Remember, logistic regression can be used for any classification task. The following slides use an example from lexical semantics:

- Given a word with different meanings (senses), can we classify which sense is intended?

I visited the Ford plant yesterday.
The farmers plant soybeans in spring.
This plant produced three kilos of berries.
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:
  
  \[
  y = \begin{array}{l}
  \text{sense} \\
  1 \quad \text{Noun: a member of the plant kingdom} \\
  2 \quad \text{Verb: to place in the ground} \\
  3 \quad \text{Noun: a factory}
  \end{array}
  \]

- We want to build a model of \( P(y|\vec{x}) \).

Defining a MaxEnt model: intuition

- Start by defining a set of features that we think are likely to help discriminate the classes. E.g.,
  - the POS of the target word
  - the words immediately preceding and following it
  - other words that occur in the document

- During training, the model will learn how much each feature contributes to the final decision.

Multinomial logistic regression

- Plain logistic regression is a binary classifier: yes/no.

- If we want to discriminate multiple classes, we need to use multinomial logistic regression.

- JM3 goes through binary version first, then multinomial. Here I will do the more general case only.

- Historically, these models were often called Maximum Entropy (MaxEnt) models in NLP.

Defining a MaxEnt model

- Features \( f_i(\vec{x}, y) \) depend on both observed and latent variables. E.g., if \( \text{tgt} \) is the target word:
  
  \[
  \begin{align*}
  f_1 : & \quad \text{POS(tgt) = NN} \land y = 1 \\
  f_2 : & \quad \text{POS(tgt) = NN} \land y = 2 \\
  f_3 : & \quad \text{preceding_word(tgt) = ‘chemical’} \land y = 3 \\
  f_4 : & \quad \text{doc\_contains(‘animal’)} \land y = 1
  \end{align*}
  \]
Defining a MaxEnt model

- Features $f_i(\vec{x}, y)$ depend on both observed and latent variables. E.g., if $tgt$ is the target word:

- Each feature $f_i$ has a real-valued weight $w_i$ (learned in training).

- $P(y | \vec{x})$ is a monotonic function of $\vec{w} \cdot \vec{f}$ (that is, $\sum_i w_i f_i(\vec{x}, y)$).

Example of features and weights

- Let’s look at just two features from the plant disambiguation example:

- Our classes are:
  {1: member of plant kingdom; 2: put in ground; 3: factory}

- Our example doc ($\vec{x}$):
  [... animal/NN ... chemical/JJ plant/NN ...]

Two cases to consider

- Computing $P(y = 1|\vec{x})$:
  - Here, $f_1 = 1$ and $f_2 = 0$.
  - We would expect the probability to be relatively high.
  - Can be achieved by having a positive value for $w_1$.
  - Since $f_2 = 0$, its weight has no effect on the final probability.

- Computing $P(y = 2|\vec{x})$:
Two cases to consider

- Computing \( P(y = 1|\vec{x}) \):
  - Here, \( f_1 = 1 \) and \( f_2 = 0 \).
  - We would expect the probability to be relatively high.
  - Can be achieved by having a positive value for \( w_1 \).
  - Since \( f_2 = 0 \), its weight has no effect on the final probability.

- Computing \( P(y = 2|\vec{x}) \):
  - Here, \( f_1 = 0 \) and \( f_2 = 1 \).
  - We would expect the probability to be close to zero, because sense 2 is a verb sense, and here we have a noun.
  - Can be achieved by having a large negative value for \( w_2 \).
  - By doing so, \( f_2 \) says: “If I am active, do not choose sense 2!”.

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
  - \( \exp(x) = e^x \) (the monotonic function)
  - \( \sum_i w_i f_i \) is the dot product of \( \vec{w} \) and \( \vec{f} \), also written \( \vec{w} \cdot \vec{f} \).
  - The normalization constant \( Z = \sum_{y'} \exp(\sum_i w_i f_i(\vec{x}, y')) \)

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

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

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

Feature weights: intuition

- \( P(y|\vec{x}) \) increases when \( \sum_i w_i f_i(\vec{x}, y) \) increases.

- Using the same binary features of the target word \( tgt \):
  \[
  f_1 : \quad \text{POS}(tgt) = \text{NN} \& y = 1
  
  f_2 : \quad \text{POS}(tgt) = \text{NN} \& y = 2
  
  f_3 : \quad \text{preceding_word}(tgt) = \text{‘chemical’} \& y = 3
  
  f_4 : \quad \text{doc.contains}(\text{‘animal’}) \& y = 1
  
- 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
  
  \[
  \begin{align*}
  \text{POS}(\text{tgt}) &= t \text{ & } y \\
  \text{preceding word}(\text{tgt}) &= w \text{ & } y \\
  \text{doc contains}(w) &= y
  \end{align*}
  \]
  - 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

Features for dependency parsing

- We want the model to tell us \( P(\text{action}|\text{configuration}) \).
- So \( y \) is the action, and \( \vec{x} \) is the configuration.
- Features are various combinations of words/tags from stack/input:

<table>
<thead>
<tr>
<th>Source</th>
<th>Feature templates</th>
</tr>
</thead>
<tbody>
<tr>
<td>One word</td>
<td>( s_1 \cdot w )</td>
</tr>
<tr>
<td>( s_2 \cdot w )</td>
<td>( s_2 \cdot f )</td>
</tr>
<tr>
<td>( b_1 \cdot w )</td>
<td>( b_1 \cdot f )</td>
</tr>
</tbody>
</table>

  | Two word | \( s_1 \cdot w \circ s_2 \cdot w \) | \( s_1 \cdot f \circ s_2 \cdot f \) | \( s_1 \cdot \text{WT} \circ b_1 \cdot \text{WT} \) |
  | \( s_1 \cdot f \circ s_2 \cdot \text{WT} \) | \( s_1 \cdot w \circ s_2 \cdot w \circ s_2 \cdot f \) | \( s_1 \cdot w \circ s_1 \cdot f \circ s_2 \cdot f \) |
  | \( s_1 \cdot w \circ s_1 \cdot f \circ s_2 \cdot f \) | \( s_1 \cdot w \circ s_1 \cdot f \circ s_2 \cdot \text{WT} \) |

  Figure 1.2: Standard feature templates for training transition-based dependency parsers. In the template specifications, \( s_e \) refers to a location on the stack, \( b_e \) refers to a location in the word buffer, \( w \) refers to the wordform of the input, and \( t \) refers to the part of speech of the input.

MaxEnt for n-best re-ranking

- So far, we’ve used logistic regression for classification.
  - Fixed set of classes, same for all inputs.

- Word sense disambiguation:
  - Input Possible outputs
    - word in doc1 sense 1, sense 2, sense 3
    - word in doc2 sense 1, sense 2, sense 3

- Dependency parsing:
  - Input Possible outputs
    - parser config1 action 1, \ldots action \( n \)
    - parser config2 action 1, \ldots action \( n \)

- We can also use MaxEnt for reranking an \( n \)-best list.

  Example scenario (Charniak and Johnson, 2005):
  - Use a generative parsing model \( M \) with beam search to produce a list of the top \( n \) parses for each sentence. (= most probable according to \( M \))
  - Use a MaxEnt model \( M' \) to re-rank those \( n \) parses, then pick the most probable according to \( M' \).
Why do it this way?

Why two stages?

• Generative models typically faster to train and run, but can’t use arbitrary features.

• In NLP, MaxEnt models may have so many features that extracting them from each example can be time-consuming, and training is even worse (see next lecture).

Why are the features a function of both inputs and outputs?

• Because for re-ranking this matters: the outputs may not be pre-defined.

MaxEnt for n-best re-ranking

• In reranking scenario, the options depend on the input. E.g., parsing, with \( n = 2 \):
  – Input: healthy dogs and cats
  – Possible outputs:

MaxEnt for constituency 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 \( y \).

• Note these are no longer binary features.
Global features

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

\[ \begin{aligned}
\text{NP} & \quad \text{vs} \quad \text{NP} \\
\text{JJ} & \quad \text{NP} \\
\text{healthy} & \quad \text{NP} \\
\text{dogs} & \quad \text{CC} \quad \text{NP} \\
\text{and} & \quad \text{NP} \\
\text{cats} & \quad \text{JJ} \\
\text{healthy} & \quad \text{NP} \\
\text{dogs} & \quad \text{CC} \quad \text{NP} \\
\text{and} & \quad \text{NP} \\
\text{cats} & \quad \text{JJ} \\
\end{aligned} \]

Features for parsing

- Altogether, Charniak and Johnson (2005) use 13 feature templates
  - with a total of 1,148,697 features
  - and that is after removing features occurring less than five times

- One important feature not mentioned earlier: the log prob of the parse under the generative model!

- So, how does it do?

Parser performance

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

\[ \begin{aligned}
\text{standard PCFG} & \quad 80\% \\
\text{lexicalized PCFG (Charniak, 2000)} & \quad 89.7\% \\
\text{re-ranked LPCFG (Charniak and Johnson, 2005)} & \quad 91.0\% \\
\end{aligned} \]

\[ \text{Figure from (Charniak, 1996): assumes POS tags as input} \]

1
Parser performance

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

- Recent WSJ parser is $93.8\%$, combining NNets and ideas from parsing, language modelling (Choe and Charniak, 2016)

- But as discussed earlier, other languages/domains are still much worse.

Evaluating during development

Whenever we have a multistep system, worth asking: where should I put my effort to improve the system?

- If my first stage (generative model) is terrible, then $n$ needs to be very large to ensure it includes the correct parse.
- Worst case: if computation is limited ($n$ is small), maybe the correct parse isn’t there at all.
- Then it doesn’t matter how good my second stage is, I won’t get the right answer.

Another use of oracles

Can be useful to compute oracle performance on the first stage.

- Oracle always chooses the correct parse if it is available.
- Difference between oracle and real system = how much better it could get by improving the 2nd stage model.
- If oracle performance is very low, need to increase $n$ or improve the first stage model.

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 use arbitrary local/global features, including correlated ones
  - much slower to train; so may use $n$-best list
- Both kinds of model are used for a variety of NLP tasks.
  - Classification, parsing (discussed here)
  - Sequence models: e.g., taggers that use spelling features.
Overall summary

- We've seen examples of applying MaxEnt to WSD, dependency parsing, and n-best reranking for constituency parsing.
- Discussed model structure and the role of features.
- Next time: training the model and relationship to neural nets.

References


