Tagging as supervised learning

- Tagging is a **supervised learning problem**
  - given: some annotated data (words annotated with POS tags)
  - build model (based on **features**, i.e. representation of example)
  - predict unseen data (POS tags for words)

- Issues in supervised learning
  - there is no data like more data
  - feature engineering: how best represent the data
  - overfitting to the training data?

- There are many algorithms for supervised learning (naive Bayes, decision trees, maximum entropy, neural networks, support vector machines, ...)

Empirical Methods in Natural Language Processing
Lecture 6
Tagging (II): Transformation-Based Learning and Maximum Entropy Models

Philipp Koehn

School of Informatics

24 January 2008
One tagging method: Hidden Markov Models

- HMMs make use of two conditional probability distributions
  - tag sequence model $p(t_n|t_{n-2}, t_{n-1})$
  - tag-word prediction model $p(w_n|t_n)$

- Given these models, we can find the best sequence of tags for a sentence using the Viterbi algorithm

How good is HMM tagging?

- Labeling a sequence is very fast

- Viterbi algorithm outputs best label sequence (previous tags affect labeling of next tag), not just best tag for each word in isolation

- It is easy to get 2nd best sequence, 3rd best sequence, etc.

- But: uses only a very small window around word ($n$ previous tags)
More features

• Consider a *larger window*

<table>
<thead>
<tr>
<th>$w_{n-4}$</th>
<th>$w_{n-3}$</th>
<th>$w_{n-2}$</th>
<th>$w_{n-1}$</th>
<th>$w_n$</th>
<th>$w_{n+1}$</th>
<th>$w_{n+2}$</th>
<th>$w_{n+3}$</th>
<th>$w_{n+4}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_{n-4}$</td>
<td>$t_{n-3}$</td>
<td>$t_{n-2}$</td>
<td>$t_{n-1}$</td>
<td>$t_n$</td>
<td>$t_{n+1}$</td>
<td>$t_{n+2}$</td>
<td>$t_{n+3}$</td>
<td>$t_{n+4}$</td>
</tr>
</tbody>
</table>

• Examples for useful features
  
  – if one of the previous tags is *MD*, then *VB* is likelier than *VBP* (basic verb form instead of verb in singular present)
  
  – if next tag is *JJ*, then *RBR* is likelier than *JJR* (adverb instead of adjective)

More features (2)

• Lexical features
  
  – if one of the previous tags is *not*, then *VB* is likelier than *VBP*

• Morphological features
  
  – if word ends in *-tion* it is most likely an *NN*
  
  – if word ends in *-ly* it is most likely an adverb
Using additional features

• Using more features in a conditional probability distribution?

\[ p(t_i|w_i, f_0, ..., f_n) \]

⇒ sparse data problems
  (insufficient statistics for reliable estimation of the distribution)

• Idea: First apply HMM, then fix errors with additional features

Applying the model to training data

• We can use the HMM tagger to tag the training data

  predicted: DET JJ NN DET NN
ture tag: DET NN VB DET NN

• How can we fix these errors? Possible transformation rules:
  – change \textit{NN} to \textit{VB} if no verb in sentence
    predicted: DET JJ VB DET NN
  – change \textit{JJ} to \textit{NN} if followed by \textit{VB}
    predicted: DET NN VB DET NN
Transformation based learning

• First, **baseline tagger**
  – most frequent tag for word: \( \arg \max_t p(t|w) \)
  – Hidden Markov Model tagger

• Then apply transformations that fix the errors
  – go through the sequence word by word
  – if a feature is present in a current example,
  → apply rule (change tag)

Learning transformations

• Given: words with their true tags

• Tag sentence with baseline tagger

• Repeat
  – find transformation that minimizes error
  – apply transformation to sentence
  – add transformation to list

• Output: ordered list of transformations
Applying the learned transformations

• Given: a new sentence that we want to tag

• Tag words with baseline tagger

• For each transformation rule (in the sequence they were learned):
  – For each word (in sentence order):
    · apply transformation, if it matches

• Output: tags

Goal: minimizing error

• We need some metric to measure the error

• Here: number of wrongly assigned tags

\[
\text{error}(D, M) = 1 - \frac{\sum_{i=1}^{N} \delta(t_{i}^{\text{predicted}}, t_{i})}{N}
\]

• General considerations for error functions:
  – Some errors are more costly than others
  – Detecting cancer, if healthy vs. detecting healthy when cancer
  – Sometimes error is difficult to assess (machine translation output different from human translation may be still correct)
Overfitting

- It may be possible to fix all errors in training
- The last transformations learned may fix only one error each
- Transformations that work in training may not work elsewhere, or may even be generally harmful
- To avoid overfitting: stop early

Generative modeling vs. discriminative training

- HMMs are an example for generative modeling
  - a model $M$ is created that predicts the training data $D$
  - the model is broken up into smaller steps
  - for each step, a probability distribution is learned
  - model is optimized on $p(D|M)$, how well it predicts the data
- Transformation-based learning is an example for discriminative training
  - a method $M$ is created to predict the training data $D$
  - it is improved by reducing prediction error
  - look for features that discriminate between faulty predictions and truth
  - model is optimized on error($M,D$), also called the loss function
Probabilities vs. rules

- HMMs: probabilities allow for *graded decisions*, instead of just yes/no
- Transformation based learning: *more features* can be considered
- We would like to combine both

⇒ *Maximum Entropy models*

---

Maximum Entropy

- Each example (here: word $w$) is represented by a set of features $\{f_i\}$, here:
  - the word itself
  - morphological properties of the word
  - other words and tags surrounding the word
- The task is the classify the word into a class $c_j$ (here: the POS tag)
- How well a feature $f_i$ predicts a class $c_j$ is defined by a parameter $\alpha(f_i, c_j)$
- Maximum entropy model:

$$p(c_j|w) = \prod_{f_i \in w} \alpha(f_i, c_j)$$
Maximum Entropy training

• Feature selection
  – given the large number of possible features, which ones will be part of the model?
  – we do not want unreliable and rarely occurring features (avoid overfitting)
  – good features help us to reduce the number of classification errors

• Setting the parameter values $\alpha(f_i, c_j)$
  – $\alpha(f_i, c_j)$ are real numbered values, similar to probabilities
  – we want to ensure that the expected co-occurrence of features and classes matches between the training data and the model
  – otherwise we want to have no bias in the model (maintain maximum entropy)
  – training algorithm: generalized iterative scaling

POS tagging tools

• Three commonly used, freely available tools for tagging:
    http://www.coli.uni-saarland.de/thorsten/tnt/
  – Brill tagger by Eric Brill (1995): transformation based learning  
    http://www.cs.jhu.edu/~brill/
  – MXPOST by Adwait Ratnaparkhi (1996): maximum entropy model  

• All have similar performance ($\sim$96% on Penn Treebank English)