Data Intensive Linguistics — Lecture 8
Tagging (III): Maximum Entropy Models

Philipp Koehn

2 February 2006

School of informatics

POS tagging tools

- Three commonly used, freely available tools for tagging:
  - **TnT** by Thorsten Brants (2000): Hidden Markov Model
    `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
    `ftp://ftp.cis.upenn.edu/pub/adwait/jmx/jmx.tar.gz`
- All have similar performance (∼96% on Penn Treebank English)

Probabilities vs. rules

- We examined two supervised learning methods for the tagging task
- 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 $\Rightarrow$ **maximum entropy models**
  - a large number of features can be defined
  - features are weighted by their importance

Features

- Each tagging decision for a word occurs in a specific context
- For tagging, we consider as context the **history** h_i
  - the word itself
  - morphological properties of the word
  - other words surrounding the word
  - previous tags
- We can define a feature f_j that allows us to learn how well a specific aspect of histories h_i is associated with a tag t_i

Features (2)

- We observe in the data patterns such as:

the word like *has in 50% of the cases the tag* VB

- Previously, in HMM models, this led us to introduce probabilities (as part of the tag sequence model) such as

$$p(VB|like) = 0.5$$

Features (3)

- In a maximum entropy model, this information is captured by a **feature**

$$f_j(h_i, t_i) = \begin{cases} 1 & \text{if } w_i = like \text{ and } t_i = VB \\ 0 & \text{otherwise} \end{cases}$$

- The importance of a feature f_j is defined by a **parameter** λ_j

Features (4)

- Features may consider morphology

$$f_j(h_i, t_i) = \begin{cases} 1 & \text{if suffix}(w_i) = \text{"ing" and } t_i = VB \\ 0 & \text{otherwise} \end{cases}$$

- Features may consider tag sequences

$$f_j(h_i, t_i) = \begin{cases} 1 & \text{if } t_{i-2} = DET \text{ and } t_{i-1} = NN \text{ and } t_i = VB \\ 0 & \text{otherwise} \end{cases}$$

Features in Ratnaparkhi [1996]

frequent w_i	$w_i = X$
rare w_i	X is prefix of w_i, $\vert X\vert \leq 4$
	X is suffix of w_i, $\vert X\vert \leq 4$
	w_i contains a number
	w_i contains uppercase character
	w_i contains hyphen
all w_i	$t_{i-1} = X$
	$t_{i-2}t_{i-1} = XY$
	$w_{i-1} = X$
	$w_{i-2} = X$
	$w_{i+1} = X$
	$w_{i+2} = X$

Log-linear model

- Features f_j and parameters λ_j are used to compute the probability $p(h_i, t_i)$:

$$p(h_i, t_i) = \prod_{f_j} \lambda_j^{f_j(h_i,t_i)}$$

- These types of models are called **log-linear models**, since they can be reformulated into

$$\log p(h_i, t_i) = \sum_{f_j} f_j(h_i, t_i)\ \log \lambda_j$$

- There are many learning methods for these models, maximum entropy is just one of them

Conditional probabilities

- We defined a model $p(h_i, t_i)$ for the *joint probability distribution* for a history h_i and a tag t_i

- *Conditional probabilities* can be computed straight-forward by

$$p(h_i|t_i) = \frac{p(h_i, t_i)}{\sum_{i'} p(h_i, t_{i'})}$$

Tagging a sequence

- We want to tag a sequence $w_1, ..., w_n$
- This can be decomposed into:

$$p(t_1, ..., t_n | w_1, ..., w_n) = \prod_{i=1}^{n} p(t_i | h_i)$$

- The history h_i consist of all words $w_1, ..., w_n$ and previous tags $t_1, ..., t_{i-1}$
- We cannot use Viterbi search $\Rightarrow$ **heuristic beam search** is used (more on beam search in a future lecture on machine translation)

Questions for training

- **Feature selection**
  - given the large number of possible features, which ones will be part of the model?
  - we do not want redundant features
  - we do not want unreliable and rarely occurring features (avoid overfitting)

- Parameter values λ_j
  - λ_j are positive real numbered values
  - how do we set them?

Feature selection

- Feature selection in Ratnaparkhi [1996]
  - Feature has to occur 10 times in the training data
- Other feature selection methods
  - use features with high mutual information
  - add feature that reduces training error most, retrain

Setting the parameter values λ_j: Goals

- The **empirical expectation** of a feature f_j occurring in the training data is defined by

$$\tilde{E}(f_j) = \frac{1}{n} \sum_{i=1}^{n} f_j(h_i, t_i)$$

- The **model expectation** of that feature occurring is

$$E(f_j) = \sum_{h,t} p(h,t) f_j(h,t)$$

- We require that $\tilde{E}(f_j) = E(f_j)$

Empirical expectation

- Consider the feature

$$f_j(h_i, t_i) = \begin{cases} 1 & \text{if } w_i = like \text{ and } t_i = VB \\ 0 & \text{otherwise} \end{cases}$$

- Computing the empirical expectation $\tilde{E}(f_j)$:
  - if there are 10,000 words (and tags) in the training data
  - ... and the word *like* occurs with the tag *VB* 20 times
  - ... then

$$\tilde{E}(f_j) = \frac{1}{n} \sum_{i=1}^{n} f_j(h_i, t_i) = \frac{1}{10000} \sum_{i=1}^{10000} f_j(h_i, t_i) = \frac{20}{10000} = 0.002$$

Model expectation

- We defined the model expectation of a feature occurring as

$$E(f_j) = \sum_{h,t} p(h,t) f_j(h,t)$$

- Practically, we cannot sum over all possible histories h and tags t
- Instead, we compute the model expectation of the feature on the training data:

$$E(f_j) \approx \frac{1}{n} \sum_{i=1}^{n} p(t|h_i)\ f_j(h_i,t)$$

Note: theoretically we have to sum over all t, but $f_j(h_i,t) = 0$ for all but one t

Goals of maximum entropy training

- *Recap:* we require that $\tilde{E}(f_j) = E(f_j)$, or

$$\frac{1}{n}\sum_{i=1}^{n} f_j(h_i, t_i) = \frac{1}{n}\sum_{i=1}^{n} p(t|h_i)\ f_j(h_i, t)$$

- Otherwise we want *maximum entropy*, i.e. we do not want to introduce any additional order into the model (**Occam's razor**: simplest model is best)

- *Entropy:*

$$H(p) = \sum_{h,t} p(h,t)\ \log p(h,t)$$

Improved Iterative Scaling [Berger, 1993]

Input: Feature functions $f_1, ..., f_m$, empirical distribution $\tilde{p}(x,y)$
Output: Optimal parameter values $\lambda_1, ..., \lambda_m$

1. Start with $\lambda_i = 0$ for all $i \in \{1, 2, ..., n\}$

2. Do for each $i \in \{1, 2, ..., n\}$:

   a. $\Delta\lambda_i = \frac{1}{C} \log \frac{\tilde{E}(f_i)}{E(f_i)}$
   b. Update $\lambda_i \leftarrow \lambda_i + \Delta\lambda_i$

3. Go to step 2 if not all the λ_i have converged

Note: This algorithm requires that $\forall t, h : \sum_i f_i(t,h) = C$, which can be ensured with an additional filler feature