Feature-Based Tagging

The Task, Again

- Recall:
  - tagging ~ morphological disambiguation
  - tagset $V_T \subset (C_1,C_2,...C_n)$
    - C_i - morphological categories, such as POS, NUMBER, CASE, PERSON, TENSE, GENDER, ...
  - mapping $w \rightarrow \{t \in V_T\}$ exists
    - restriction of Morphological Analysis: $A^+ \rightarrow 2^{(L,C1,C2,...,Cn)}$

      where A is the language alphabet, L is the set of lemmas
  - extension to punctuation, sentence boundaries (treated as words)

Feature Selection Problems

- Main problem with Maximum Entropy [tagging]:
  - Feature Selection (if number of possible features is in the hundreds of thousands or millions)
  - No good way
    - best so far: Berger & DP's greedy algorithm
    - heuristics (cutoff based: ignore low-count features)
- Goal:
  - few but "good" features ("good" ~ high predictive power ~ leading to low final cross entropy)

Feature-based Tagging

- Idea:
  - save on computing the weights (λ_i)
    - are they really so important?
  - concentrate on feature selection
- Criterion (training):
  - error rate (~ accuracy; borrows from Brill's tagger)
- Model form (probabilistic - same as for Maximum Entropy):

$$p(y|x) = (1/Z(x))\, e^{\sum_{i=1..N} \lambda_i f_i(y,x)}$$

→ Exponential (or Loglinear) Model

Feature Weight (Lambda) Approximation

- Let Y be the sample space from which we predict (tags in our case), and $f_i(y,x)$ a b.v. feature
- Define a "batch of features" and a "context feature":

  $B(x) = \{f_i;$ all f_i's share the same context x$\}$

  $f_{B(x)}(x') = 1 \Leftrightarrow_{df}\ x \subset x'$ (x is part of x')

  - in other words, holds wherever a context x is found
- Example:

  $f_1(y,x) = 1 \Leftrightarrow_{df}$ y=JJ, left tag = JJ

  $f_2(y,x) = 1 \Leftrightarrow_{df}$ y=NN, left tag = JJ

  B(left tag = JJ) = $\{f_1, f_2\}$ (but not, say, [y=JJ, left tag = DT])

Estimation

- Compute:

  $p(y|B(x)) = (1/Z(B(x)))\ \Sigma_{d=1..|T|}\delta(y_d,y)f_{B(x)}(x_d)$

  - frequency of y relative to all places where any of B(x) features holds for some y; Z(B(x)) is the natural normalization factor

  □ $Z(B(x)) = \Sigma_{d=1..|T|} f_{B(x)}(x_d)$

  "compare" to uniform distribution:

  □ $\alpha(y,B(x)) = p(y|B(X)) / (1 / |Y|)$

  $\alpha(y,B(x)) > 1$ for $p(y|B(x))$ better than uniform; and vice versa

- If $f_i(y,x)$ holds for exactly one y (in a given context x),

  then we have 1:1 relation between $\alpha(y,B(x))$ and $f_i(y,x)$ from B(x)

  and $\lambda_i = \log(\alpha(y,B(x)))$

NB: works in constant time independent of λ_j, $j \neq i$

What we got

- Substitute:

$$p(y|x) = (1/Z(x))\, e^{\sum_{i=1..N}\lambda_i f_i(y,x)} =$$

$$= (1/Z(x)) \prod_{i=1..N} \alpha(y,B(x))^{f_i(y,x)}$$

$$= (1/Z(x)) \prod_{i=1..N} (|Y|\ p(y|B(x)))^{f_i(y,x)}$$

$$= (1/Z'(x)) \prod_{i=1..N} (p(y|B(x)))^{f_i(y,x)}$$

$$= (1/Z'(x)) \prod_{B(x');\, x' \subset x} p(y|B(x'))$$

... Naive Bayes (independence assumption)

The Reality

- take advantage of the exponential form of the model (do ***<u>not</u>*** reduce it completely to naive Bayes):
  - vary $\alpha(y,B(x))$ up and down a bit (quickly)
    - captures dependence among features
  - recompute using "true" Maximum Entropy
    - the ultimate solution
  - combine feature batches into one, with new $\alpha(y,B(x'))$
    - getting very specific features

Search for Features

- Essentially, a way to get rid of unimportant features:
  - start with a pool of features extracted from full data
  - remove infrequent features (small threshold, < 2)
  - organize the pool into batches of features
- Selection from the pool P:
  - start with empty S (set of selected features)
  - try all features from the pool, compute $\alpha(y,B(x))$, compute error rate over training data.
  - add the best feature batch permanently; stop when no correction made [complexity: |P| x |S| x |T|]

Adding Features in Blocks, Avoiding the Search for the Best

- Still slow; solution: add ***<u>ten (5,20)</u>*** best features at a time, assuming they are independent (i.e., the next best feature would change the error rate the same way as if no intervening addition of a feature is made).
- Still slow [(|P| x |S| x |T|)/10, or 5, or 20]; solution:
- Add ***<u>all</u>*** features improving the error rate by a certain threshold; then gradually lower the threshold down to the desired value; complexity [|P| x log|S| x |T|] if

  $\text{threshold}^{(n+1)} = \text{threshold}^{(n)} / k, k > 1$ (e.g. $k = 2$)

Types of Features

- Position:
  - current
  - previous, next
  - defined by the closest word with certain major POS
- Content:
  - word (w), tag(t) - left only, “Ambiguity Class” (AC) of a subtag (POS, NUMBER, GENDER, CASE, ...)
- Any combination of position and content
- Up to three combinations of (position,content)

Ambiguity Classes (AC)

- Also called “pseudowords” (MS, for word sense disambiguationi task), here: “pseudotags”
- AC (for tagging) is a set of tags (used as an indivisible token).
  - Typically, these are the tags assigned by a morphology to a given word:
    - MA(books) [restricted to tags] = { NNS, VBZ }:
      AC = NNS_VBZ
- Advantage: deterministic

  → looking at the ACs (and words, as before) to the ***right*** allowed

Subtags

- Inflective languages: too many tags $\rightarrow$ data sparseness
- Make use of separate categories (remember morphology):
  - tagset $V_T \subset (C_1, C_2, \ldots C_n)$
    - C_i - morphological categories, such as POS, NUMBER, CASE, PERSON, TENSE, GENDER, ...
- Predict (and use for context) the individual categories
- Example feature:
  - previous word is a noun, and current CASE subtag is genitive
- Use separate ACs for subtags, too (AC_{POS} = N_V)

Combining Subtags

- Apply the separate prediction (POS, NUMBER) to
  - MA(books) = { (Noun, Pl), (VerbPres, Sg)}
- Now what if the best subtags are
  - Noun for POS
  - Sg for NUMBER
    - (Noun, Sg) is ***not*** possible for books
- Allow only possible combinations (based on MA)
- Use independence assumption (Tag = $(C_1, C_2, \ldots, C_n)$):

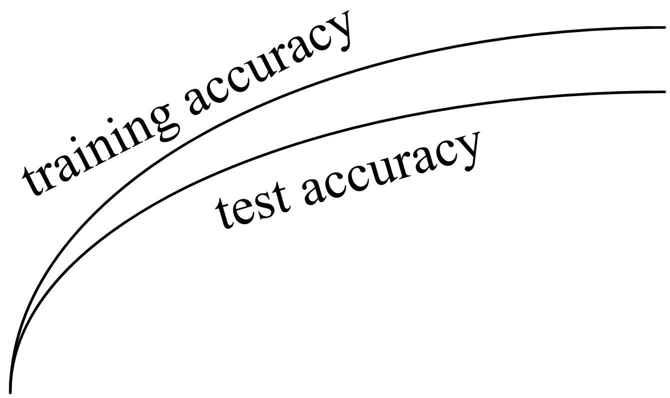
$$\text{(best) Tag} = \operatorname{argmax}_{\text{Tag} \in \text{MA}(w)} \prod_{i=1..|\text{Categories}|} p(C_i|w,x)$$

Smoothing

- Not needed in general (as usual for exponential models)
  - however, some basic smoothing has an advantage of not learning unnecessary features at the beginning
  - very coarse: based on ambiguity classes
    - assign the most probable tag for each AC, using MLE
    - e.g. NNS for AC = NNS_VBZ
  - last resort smoothing: unigram tag probability
  - can be even parametrized from the outside
  - also, needed during training

Overtraining

- Does not appear in general
  - usual for exponential models
  - does appear in relation to the training curve:



  - but does not go down until very late in the training (singletons do cause overtraining)