HMM Tagging

PA154 Language Modeling (6.2)

Pavel Rychlý

pary@fi.muni.cz

March 26, 2024

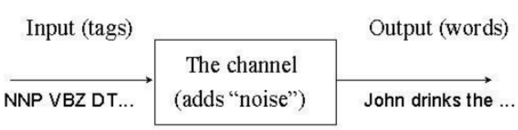
Source: Introduction to Natural Language Processing (600.465)
Jan Hajič, CS Dept., Johns Hopkins Univ.
www.cs.jhu.edu/˜hajic

Review

- Recall:
  - tagging $\sim$ morphological disambiguation
  - tagset $V_T \subset (C_1, C_2, \ldots C_n)$
    - C_i - morphological categories, such as POS, NUMBER, CASE, PERSON, TENSE, GENDER,...
  - mapping $w \rightarrow \{t \in V_T\}$ exists
    - restriction of Morphological Analysis: $A^+ \rightarrow 2^{(L,C2,C2,\ldots,Cn)}$ where A is the language alphabet, L is the set of lemmas
  - extension of punctuation, sentence boundaries (treated as words)

The Setting

- Noisy Channel setting:



- Goal (as usual): discover ”input” to the channel (T, the tag seq.) given the ”output” (W, the word sequence)
  - $p(T|W) = p(W|T)p(T)/p(W)$
  - $p(W)$ fixed (W given)... $argmax_T p(T|W) = argmax_T p(W|T)p(T)$

The Model

- Two models ($d = |W| = |T|$ word sequence length):
  - $p(W|T) = \Pi_{i=1\ldots d} p(w_i|w_1, \ldots, w_{i-1}, t_1, \ldots, t_d)$
  - $p(T) = \Pi_{i=1\ldots d} p(t_i|t_1, \ldots, t_{i-1})$
- Too much parameters (as always)
- Approximation using the following assumptions:
  - words do not depend on the context
  - tag depends on limited history: $p(t_i|t_1, \ldots, t_{i-1}) \cong p(t_i|t_{i-n+1}, \ldots, t_{i-1})$
    - n-gram tag ”language” model
  - word depends on tag only: $p(w_i|w_1, \ldots, w_{i-1}, t_1, \ldots, t_d) \cong p(w_i|t_i)$

The HMM Model Definition

- (Almost) general HMM:
  - output (words) emitted by states (not arcs)
  - states: (n-1)-tuples of tags if n-gram tag model used
  - five-tuple (S, s_0, Y, P_S, P_Y) where:
    - $S = \{s_0, s_1, \ldots, s_T\}$ is the set of states, s_0 is the initial state,
    - $Y = \{y_1, y_2, \ldots, y_y\}$ is the output alphabet (the words),
    - $P_S(s_j|s_i)$ is the set of prob. distributions of transitions -$P_S(s_j|s_i) = p(t_i|t_{i-n+1}, \ldots, t_{i-1}); s_j = (t_{i-n+2}, \ldots, t_i), s_i = (t_{i-n+1}, \ldots, t_{i-1})$
    - $P_Y(y_k|s_i)$ is the set of output (emission) probability distributions -another simplification: $P_Y(y_k|s_j)$ if s_i and s_j contain the same tag as the rightmost element: $P_Y(y_k|s_i) = p(w_i|t_i)$

Supervised Learning (Manually Annotated Data Available)

- Use MLE
  - $p(w_i|t_i) = c_{wt}(t_i, w_i)/c_t(t_i)$
  - $p(t_i|t_{i-n+1}, \ldots, t_{i-1}) = c_{tn}(t_{i-n+1}, \ldots, t_{i-1}, t_i)/c_{t(n-1)}(t_{i-n+1}, \ldots, t_{i-1})$
- Smooth(both!)
  - $p(w_i|t_i)$: ”Add 1” for all possible tag, word pairs using a predefined dictionary (thus some 0 kept!)
  - $p(t_i|t_{i-n+1}, \ldots, t_{i-1})$: linear interpolation:
    - e.g. for trigram model: $p'_\lambda(t_i|t_{i-2}, t_{i-1}) = \lambda_3 p(t_i|t_{i-2}, t_{i-1}) + \lambda_2 p(t_i|t_{i-1}) + \lambda_1 p(t_i) + \lambda_0/|V_T|$

Unsupervised Learning

- Completely unsupervised learning impossible
  - at least if we have the tagset given- how would we associate words with tags?
- Assumed (minimal) setting:
  - tagset known
  - dictionary/morph. analysis available (providing possible tags for any word)
- Use: Baum-Welch algorithm (see lecture 6.1)
  - ”tying”: output (state-emitting only, same dist. from two states with same ”final” tag)

Comments on Unsupervised Learning

- Initialization of Baum-Welch
  - is some annotated data available, use them
  - keep 0 for impossible output probabilities
- Beware of:
  - degradation of accuracy (Baum-Welch criterion: entropy, not accuracy!)
  - use heldout data for cross-checking
- Supervised almost always better

Unknown Words

- ”OOV” words (out-of-vocabulary)
  - we do not have list of possible tags for them
  - and we certainly have no output probabilities
- Solutions:
  - try all tags (uniform distribution)
  - try open-class tags (uniform, unigram distribution)
  - try to ”guess” possible tags (based on suffix/ending) - use different output distribution based on the ending (and/or other factors, such as capitalization)

Running the Tagger

- Use Viterbi
  - remember to handle unknown words
  - single-best, n-best possible
- Another option
  - assign always the best tag at each word, but consider all possibilities for previous tags (no back pointers nor a path-backpass)
  - introduces random errors, implausible sequences, but might get higher accuracy (less secondary errors)

(Tagger) Evaluation

- **A must.** Test data (S), previously unseen (in training)
  - change test data often if at all possible! (”feedback cheating”)
  - Error-rate based
- Formally:
  - Out(w) = set of output ”items” for an input ”item” w
  - True(w) = single correct output (annotation) for w
  - Errors(S) = $\sum_{i=1..|S|} \delta$ (Out(w_i) $\neq$ True(w_i))
  - Correct(S) = $\sum_{i=1..|S|} \delta$ (True(w_i) $\in$ Out(w_i))
  - Generated(S) = $\sum_{i=1..|S|} \delta|$Out(w_i)$|$

Evaluation Metrics

- Accuracy: Single output (tagging: each word gets a single tag)
  - Error rate: Err(S) = Errors(S)$/|S|$
  - Accuracy: Acc(S) $= 1-$(Errors(S)$/|S|$) $= 1-$ Err(S)
- What if multiple (or no) output?
  - Recall: R(S) = Correct(S)/$|S|$
  - Precision: P(S) = Correct(S)/Generated(S)
  - Combination: F measure: $F = 1/(\alpha/P + (1-\alpha)/R)$
    - α is a weight given to precision vs. recall; for $\alpha = .5, F = 2PR/(R+P)$