Exercises on “Sequence Labeling”

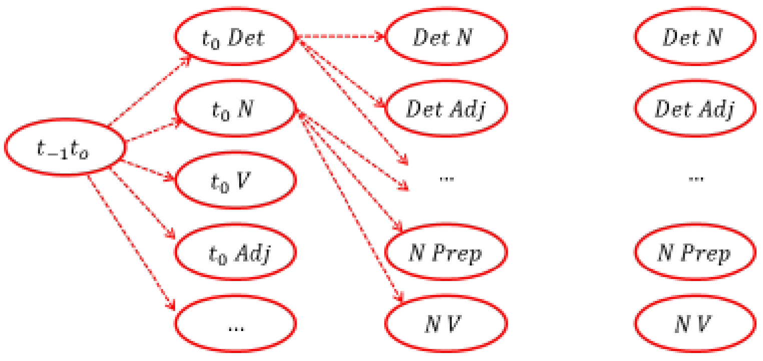
Ion Androutsopoulos, 2017–18 (last revision 22/2/18)

Submit as a group of 3–4 members a report (max. 5 pages, PDF format) for exercise 3 or exercise 4 (only one exercise per group). Include in your report all the required information, especially experimental results. Do not include code in the report, but include a link to a shared folder or repository (e.g. in Dropbox, GitHub, Bitbucket) containing your code.

1. We wish to use a second order HMM to construct a POS tagger. Assume that $P(t_i \mid t_1^{i-1}) \approx P(t_i \mid t_{i-1}, t_{i-2})$ and $P(w_i \mid w_1^{i-1}, t_1^i) \approx P(w_i \mid t_i)$, where w are the words, t are the tags, w_{-1}, w_0 are two pseudo-words at the beginning of each word sequence, and t_{-1}, t_0 are the pseudo-tags of w_{-1}, w_0.

(i) Draw a diagram like the one of slide 15 to show the lattice of the Viterbi decoder in this case. Assume that each node of the lattice has a label $t_{i-1}t_i$, where t_i is the tag of the current word and t_{i-1} is the tag of the previous word (see also slide 18).

Answer:



(ii) Write the formulae to compute $V_i(t_{i-1}t_i)$ at each node $t_{i-1}t_i$ of each j-th column of the lattice. Show in detail how the formulae are derived (as in slides 15 and 16).

Answer:

$$\max_{t_1^k} \prod_{i=1}^{k} P(t_i|t_{i-2}t_{i-1}) \cdot P(w_i|t_i) =$$

$$\max_{t_{k-1}t_k} \max_{t_1^{k-2}} \prod_{i=1}^{k} P(t_i|t_{i-2}t_{i-1}) \cdot P(w_i|t_i) = \max_{t_{k-1}t_k} V_k(t_{k-1}t_k)$$

where we set:

$$V_k(t_{k-1}t_k) = \max_{t_1^{k-2}} \prod_{i=1}^{k} P(t_i|t_{i-2}t_{i-1}) \cdot P(w_i|t_i)$$

We observe that:

$$V_k(t_{k-1}t_k) =$$

$$P(w_k|t_k) \cdot \max_{t_{k-2}} P(t_\kappa|t_{k-2}t_{k-1}) \cdot \max_{t_1^{k-3}} \prod_{i=1}^{k-1} P(t_i|t_{i-2}t_{i-1}) \cdot P(w_i|t_i) =$$
$$P(w_k|t_k) \cdot \max_{t_{k-2}} P(t_\kappa|t_{k-2}t_{k-1}) \cdot V_{k-1}(t_{k-2}t_{k-1})$$

where we set:

$$V_{k-1}(t_{k-2}t_{k-1}) = \max_{t_1^{k-3}} \prod_{i=1}^{k-1} P(t_i|t_{i-2}t_{i-1}) \cdot P(w_i|t_i)$$

Similarly, for $j = 2, \dots, k$:

$$V_j\left(t_{j-1}t_j\right) = P\left(w_j|t_j\right) \cdot \max_{t_{j-2}} P\left(t_j|t_{j-2}t_{j-1}\right) \cdot V_{j-1}\left(t_{j-2}t_{j-1}\right)$$

Using the previous recursive formula, we can compute $V_j\left(t_{j-1}t_j\right)$ at each node $t_{j-1}t_j$ of each column of the lattice that corresponds to step j, having first computed the values $V_1(t_0t_1)$ of the first column as follows:

$$V_1(t_0t_1) = P(t_1|t_{-1}t_0) \cdot P(w_1|t_1)$$

2. Modify the MEMM of slides 28–31, so that each tag will depend on the tags of the *two previous* words and the feature vector of the current word.

(i) Draw the new lattice of the Viterbi decoder (slide 31).

(ii) Derive the new formulae to compute $V_j\left(t_{j-1}t_j\right)$ at each node $t_{j-1}t_j$ of each each j-th column of the lattice (slides 28–30).

3. We wish to use a Viterbi decoder to find the most probable correct words $\hat{t}_1^k$ in slide 32 (generalization for type-2 errors) of Part 1 (n-gram language models, spelling correction, and text normalization). Use a bigram (or optionally trigram) language model.

(i) Draw the lattice of the Viterbi decoder in this case. What would the nodes of the lattice in each column stand for?

(ii) Write down the formulae to compute $V_j\left(t_j\right)$ (or $V_j\left(t_{j-1}t_j\right)$, if you use a trigram language model) at each node of each column of the lattice. Explain how you obtained the formulae.

(iii) Extend the code you wrote for exercise 4 (language model) of Part 1 to develop a context-sensitive spelling corrector (for both type-1 and type-2 errors) that uses Levenshtein distance, a bigram (or optionally trigram) language model, and a Viterbi decoder. Train the language model of your spelling corrector as in exercise 4 of Part 1. You may use an existing implementation of Levenshtein distance.

(iv) Introduce random spelling errors in the test dataset that you used in exercise 4 of Part 1, by randomly replacing characters and/or entire words by randomly selected ones. Report in detail how you constructed the new test dataset (that contains spelling errors) and provide appropriate statistics (e.g., number of characters and tokens in the new test dataset, percentage of wrong characters and tokens). Devise appropriate evaluation measures and baselines. Use them to evaluate your context-sensitive spelling corrector on the new test dataset. Report your evaluation measures, baselines, and evaluation results.

4. Develop a POS tagger for one of the languages of the Universal Dependencies treebanks (http://universaldependencies.org/), using an HMM, MEMM, or CRF model (see optional material on CRFs). Consider only the words, sentences, and POS tags of the treebanks (not the dependencies or other annotations). You may use existing HMM, MEMM, or CRF implementations or your own implementations. You may use (in MEMM or CRF) any types of word features you prefer. Draw learning curves with appropriate measures (e.g., overall accuracy, possibly also F1 separately per POS). Include experimental results of appropriate baselines (e.g., always tagging each word with the most frequent tag it had in the training set). Make sure that you use separate training and test data. Tune the feature set and hyper-parameters on a held-out part of the training data or using a cross-validation on the training data. Document clearly in a short report (max. 5 pages) how your system works (e.g., what algorithms it uses, examples of input/output), its experimental results (e.g., learning curves), and the most frequent types of mistakes it makes.

5. (i) What is the time complexity of a brute-force decoder in the HMM POS-tagger of slide 14, if the tagset contains T tags and the input sentence consists of k words? (ii) What is the time complexity when using a Viterbi decoder?

6. For some words (e.g., prepositions, the) there is (almost always) only one possible tag. How could the lattice of slide 17 and the Viterbi decoding formulae be modified to take advantage of this observation?