CS114 (Spring 2018) Homework 4
Part-of-speech Tagging with Hidden Markov Models

Due March 20, 2018

You are given `pos_tagger.py`, and `brown.zip`, the Brown corpus (of part-of-speech tagged sentences). Sentences are separated into a training set (≈80% of the data) and a development set (≈10% of the data). A testing set (≈10% of the data) has been held out and is not given to you. Each folder (`train`, `dev`) contains a number of documents, each of which contains a number of tokenized, tagged sentences in the following format: `[<word>/<tag>]`. For example:

```
The/at Fulton/np-tl County/nn-tl Grand/jj-tl Jury/nn-tl
said/vbd Friday/nr an/at investigation/nn of/in Atlanta's/np$
recent/jj primary/nn election/nn produced/vbd ``/`` no/at
evidence/nn ''/'' that/cs any/dti irregularities/nns took/vbd
place/nn ./.
```

Assignment: Supervised HMM

Your task is to implement a supervised hidden Markov model to perform part-of-speech tagging. Specifically, in `pos_tagger.py`, you should fill in the following functions:

- `train(self, train_set)`: This function should, given a folder of training documents, fill in `self.transition` and `self.emission`. Recall that:
  - Transition probabilities for a pair of POSs (p, s):
    $$P(s|p) = \frac{count(POS_{previous\_word} = p, POS_{word} = s)}{count(POS_{previous\_word} = p)}$$
  - Emission probabilities for a pair of $(POS = s, word = w)$:
    $$P(w|s) = \frac{count(POS_{word} = s, word = w)}{count(POS_{word} = s)}$$

You can choose which data structures you want to use for `self.transition` and `self.emission`. You may want to reuse the bigram code (either your own, or the sample code) from Homework 2. Be sure to account for `<S>` and `</S>` (as POS tags), and `UNK` (as a word). You should also use log-probabilities, as in Homework 3. Finally, you should also implement smoothing: feel free to use any smoothing method that you have learned.

- `viterbi(self, sentence)`: Implement the Viterbi algorithm! Specifically, for each development (or testing) sentence, you should construct and fill in two trellises, `v` (for `viterbi`) and `backpointer`, according to the pseudo-code given in the book. Note that $a \in A$ are elements of the transition matrix and $b(o) \in B$ are elements of the emission matrix.

function VITERBI(*observations* of len T, *state-graph* of len N) **returns** *best-path*

create a path probability matrix *viterbi[N+2,T]*
for each state s **from** 1 **to** N **do** ; initialization step
  $viterbi[s,1] \leftarrow a_{0,s} * b_s(o_1)$
  $backpointer[s,1] \leftarrow 0$
for each time step t **from** 2 **to** T **do** ; recursion step
  for each state s **from** 1 **to** N **do**
    $viterbi[s,t] \leftarrow \max_{s'=1}^{N} viterbi[s',t-1] * a_{s',s} * b_s(o_t)$
    $backpointer[s,t] \leftarrow \operatorname{argmax}_{s'=1}^{N} viterbi[s',t-1] * a_{s',s} * b_s(o_t)$
$viterbi[q_F,T] \leftarrow \max_{s=1}^{N} viterbi[s,T] * a_{s,q_F}$; termination step
$backpointer[q_F,T] \leftarrow \operatorname{argmax}_{s=1}^{N} viterbi[s,T] * a_{s,q_F}$; termination step
return the backtrace path by following backpointers to states back in time from $backpointer[q_F,T]$

Figure 9.11 Viterbi algorithm for finding optimal sequence of hidden states. Given an observation sequence and an HMM $\lambda = (A,B)$, the algorithm returns the state path through the HMM that assigns maximum likelihood to the observation sequence. Note that states 0 and q_F are non-emitting.

Again, you can choose which data structures you want to use for `v` and `backpointer` (this time, a nested dictionary is highly recommended). Remember that when working in log-space, you should add the logs, rather than multiply the probabilities. Return the backtrace path.

- `test(self, dev_set)`: This function should, given a folder of development (or testing) documents, return a dictionary of results such that:
  - `results[sentence_id]['correct']` = correct sequence of POS tags
  - `results[sentence_id]['predicted']` = predicted sequence of POS tags (hint: call the `viterbi` function)

  You can assign each sentence a `sentence_id` however you want. Your sequences of POS tags can just be lists, e.g. `['at', 'np-tl', etc.]`. Do not include `<S>` or `</S>`.

- `evaluate(self, results)`: This function should return the overall accuracy (# of words correctly tagged / total # of words). You can refer back to your code from Homework 3 if you want. You don't have to calculate precision, recall, or F1 score.

After you fit your POS tagger on the training data, use the dev set for any hyperparameter tuning (for example, α if you use add-α (Lidstone) smoothing, or λ for interpolation). Simple add-one smoothing is also fine. Report your accuracy on the dev set.

Extra Credit: Unsupervised HMM

This part of the assignment is not mandatory but you will be awarded extra credit if you finish it.

You are given a list of sentences without any part-of-speech tags:

```
He saw a cat.
A cat saw him.
He saw a dog.
A dog saw him.
He chased the cat.
The cat chased him.
He chased the dog.
The dog chased him.
```

Your task is to implement the Baum-Welch algorithm (also known as the forward-backward algorithm) to perform unsupervised part-of-speech tagging. Assume that there are four POS tags (you can call them 1, 2, 3, 4).

Use the pseudo-code in the book to guide your solution. Random intitialization of A and B is probably fine, and the definition of "convergence" is left up to you.

function FORWARD-BACKWARD(*observations* of len T, *output vocabulary* V, *hidden state set* Q) **returns** *HMM=(A,B)*

initialize A and B
iterate until convergence
E-step

$$\gamma_t(j) = \frac{\alpha_t(j)\beta_t(j)}{\alpha_T(q_F)} \quad \forall\, t \text{ and } j$$

$$\xi_t(i,j) = \frac{\alpha_t(i)\, a_{ij} b_j(o_{t+1})\beta_{t+1}(j)}{\alpha_T(q_F)} \quad \forall\, t,\ i, \text{ and } j$$

M-step

$$\hat{a}_{ij} = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1}\sum_{k=1}^{N} \xi_t(i,k)}$$

$$\hat{b}_j(v_k) = \frac{\sum_{t=1 s.t.\ O_t=v_k}^{T} \gamma_t(j)}{\sum_{t=1}^{T} \gamma_t(j)}$$

return A, B

Figure 9.16 The forward-backward algorithm.

After you have trained your model, test it on the list of sentences (using the Viterbi algorithm from before). Report and discuss your model parameters and the tags for each sentence.

Up to 20% will be awarded as extra credit for your unsupervised HMM.

Write-up

You should also prepare a (very short!) write-up that includes the following (which will count toward your grade):

- Your evaluation results on the development set
- If you tried the unsupervised HMM:
  - Your model parameters (give tables A and B). Is this what you expected?
  - Tags for each sentence

Please also include the following (which will *not* count toward your grade):

- (About) how many hours you spent working on this assignment
- Any parts of the assignment you found particularly easy or difficult
- Any other comments on the assignment you would like to make

Submission Instructions

Zip up your write-up (PDF or plain-text format, please!), `pos_tagger.py`, and any files you used for the unsupervised HMM, if you tried it. Do not include `brown.zip`.