Assignment 3[*]

CMSC 473/673 — Introduction to Natural Language Processing

Due Wednesday November 22nd, 2017, 11:59 AM

Item	**Summary**
Assigned	Wednesday October 25th, 2017
Due	Wednesday November 22nd, 2017
Topic	Hidden Markov Models
Points	170

In this assignment you will understand and gain experience in both implementing and using Hidden Markov Models (HMM).

As with the previous assignments, you are to *complete* this assignment on your own: that is, the code and writeup you submit must be entirely your own. However, you may discuss the assignment at a high level with other students or on the discussion board. Note at the top of your assignment who you discussed this with or what resources you used (beyond course staff, any course materials, or public Piazza discussions).

The following table gives the overall point breakdown for this assignment.

Question	1	2	3	4	5
Points	20	30	30	30	60 (+20 EC)

As before, the first page provides a **task list**. This task list captures the questions, and details (without other explanatory text) the tasks you are to do and what your completed assignment should answer. Following the task list are the **full questions**. The full questions do *not* require you to answer additional questions.

The task list enumerates what you must do, but it does not necessarily specify *how*—that's where the full questions come in. Therefore, you **should still read and reference** the full questions.

What To Turn In Turn in a writeup in PDF format that answer the questions; turn in all requested code necessary to replicate your results. Be sure to include specific instructions on how to build (compile) your code. Answers to the following questions should be long-form. Provide any necessary analyses and discussion of your results.

How To Submit Submit the assignment using the `submit` utility on GL. The course id is `cs473_ferraro`. This assignment's id is `a3`.

Please run your code on GL before you submit it. In particular, make sure that your code does not use any local *copies* of data sets.

[*]Thanks to Chi Zhang, who co-wrote this assignment.

Task List

1. Analyze some part of speech data (training, development, or both):
    (a) How many different part of speech tag types are defined in the data files themselves? It's sufficient to look at the training set; all tags defined in dev or test appear at least once in training.
    (b) What are the ten most common tuples centered around the word "to", using one word to the left and one word to the right? You can lowercase the words with the `--lowercase` flag.
    (c) What are all of the ways that the word "to" is defined? Does this surprise you?
    (d) Summarize some of the syntactic patterns surrounding the word "to."
    (e) Find a word ω that is tagged with both the RP tag and some other tag X; compare, contrast and summarize how ω as an RP is used vs. as an X.
2. Play around with the ice cream spreadsheet. Turn in your responses for the six questions (a)-(f).
3. Read and write a half page summary[1] and review of Merialdo (1994).
4. Implement and evaluate a supervised HMM Part-of-speech tagger using the Viterbi algorithm. Submit your code and a brief report describing:
    - Your analysis of the evaluation results, during both development and testing.
    - Any internal improvements you made during development.
    - Additional ideas to improve the result (you do *not* have to implement them).
    - What are the top 5 POS tags that are most commonly identified correctly, and what are the 5 POS tags that are most commonly identified *incorrectly*?
5. Develop a semi-supervised POS HMM tagger. You should do this by implementing, evaluating, and analyzing one of the following options:
    - "Regular" ("soft") EM, evaluated with both Viterbi and posterior decoding;
    - Viterbi ("hard") EM, evaluated with both Viterbi and posterior decoding;
    - Both "regular" ("soft") and Viterbi ("hard") EM, evaluated with Viterbi decoding;
    - Both "regular" ("soft") and Viterbi ("hard") EM, evaluated with posterior decoding;

    For **20 points of extra credit**, you may implement *both* hard and soft EM, evaluated with *both* Viterbi and posterior decoding. Turn in your code and a report containing:
    - Descriptions, analyses, and take-aways of internal development experiments.
    - The average probability of the sequences in the test set given your semi-supervised model(s).
    - The per-token accuracy and per-sentence accuracy of your decoded tag sequences for the test data set. Try to explain what happened, and why.
    - Ideas to improve the algorithms. You do not have to implement these, but identify where your systems fail and ways you might correct for those failures.

[1] Single spaced, regular font, and one column is fine.

Full Questions

1. (**20 points**) In this assignment you will experiment with Hidden Markov Models, with a special focus on part of speeech tagging. The training, development, and test data are available in the directory

   `/afs/umbc.edu/users/f/e/ferraro/pub/473-f17/a3/data`.

   The JSON files follow the same format as for assignment 2 (except the "lang" key has been removed—everything's English for this assignment).

   You can also use the counting code in

   `/afs/umbc.edu/users/f/e/ferraro/pub/473-f17/a3/code/lexicalized_spans.py`.

   This script finds occurrences of words (or parts of speech) within a left-right context window. For example, if you want to find the two most common tuples involving any particular word to the left and right of a progressive verb (`VBG`), use the command

   ```
   $ python lexicalized_spans.py train.json --pos VBG --store both \
     --most-common 2 --left 1 --right 1
   1345 unique tuples matching * word, VBG POS
   34 times: ,/, according/VBG to/TO
   8 times: <BOS>/### According/VBG to/TO
   ```

   If you only want to store the words, set the `--store` flag to `word`

   ```
   $ python lexicalized_spans.py train.json --pos VBG \
     --most-common 2 --left 1 --right 1 --store word
   1345 unique tuples matching * word, VBG POS
   34 times: , according to
   8 times: <BOS> According to
   ```

   If you only want to store the part of speech tags, set the `--store` flag to `pos`

   ```
   $ python lexicalized_spans.py train.json --pos VBG \
     --most-common 2  --left 1 --right 1 --store pos
   280 unique tuples matching * word, VBG POS
   65 times: IN VBG DT
   47 times: , VBG DT
   ```

   (**Note** that the `###` part of speech is defined separately; it is not an official part of the dataset.)

   In addition to filtering by POS (`--pos`) you can instead filter by word (`--word`); you can also filter by both. Set both `--left` and `--right` to 0 to get back a list of unigrams, ranked by overall count.

   You can return the entire ranked list by setting `--most-common` to `-1` (default is `10`).

   For this question, you can answer using **either training or dev**; do NOT use test.

   (a) How many different part of speech tag types are defined in the data files themselves? It's sufficient to look at the training set; all tags defined in dev or test appear at least once in training.

   (b) What are the ten most common tuples centered around the word "to", using one word to the left and one word to the right? You can lowercase the words with the `--lowercase` flag. (Your exact results may change from run to run, as ties among equal count tuples are broken arbitrarily.)

(c) Recall that prepositions include words such as "from," "for," and "with." What are all of the ways that the word "to" is defined? Does this surprise you?

(d) Summarize some of the syntactic patterns surrounding the word "to."

(e) Finally, let's look at the RP tag. First, find a word ω that is tagged with both the RP tag and some other tag X. Compare, contrast and summarize how ω as an RP is used vs. as an X. This word can be tagged with other parts of speech as well, but you only must focus on RP vs. X.

2. (**30 points**) In this question, you will get familiar with HMM using the ice cream spreadsheet created by Jason Eisner from `http://www.cs.jhu.edu/~jason/465/hw-hmm/lect24-hmm.xls`. You will answer questions related to different initial parameters (the red cells) and try to find out how the initial transition probabilities, emission probabilities and observations affect the end results. This will give you a change to play around with a working implementation of EM, and to see what happens from iteration to iteration.

   Turn in your responses for the following questions. In answering the following, **always** consider how the various graphs, reconstructed weather (per day and pair of days), and final perplexity can change.

   (a) What will happen if you strongly believe that one day's weather should not be the same as the day after it? This corresponds to $p(H|C)$ and $p(C|H)$ being high.

   (b) What will happen if you strongly believe that one day's weather has nothing to do with the day after it, i.e., $p(H|C) = p(C|H) = p(C|C) = p(H|H)$?

   (c) What will happen if you strongly believe that the colder the weather gets the more ice cream Jason wants to eat, e.g., $p(1|C) = 0.1$ and $p(2|C) = 0.2$ while $p(1|H) = 0.7$ and $p(2|H) = 0.2$?

   (d) What will happen if you strongly believe Jason just wants to eat ice cream and it has nothing to do with weather, i.e., $p(1|C) = p(2|C) = p(1|H) = p(2|H) = 0$?

   (e) What will happen if there are actually some additional, unmodeled factors that affect the number of ice cream Jason eats? For example, all of a sudden ice cream gets too expensive. To be specific, consider the case where

   - For the first 11 days he can eat whatever amount of ice cream he want.
   - Then for the next 11 days things become tight and he can only buy at most 2 ice creams per day.
   - And finally he's starting to run out of money, so he can only afford 1 ice cream per day.

   (f) Try out the new sequence of ice creams eaten: [2, 2, 1, 3, 1, 2, 2, 1, 3, 3, 1, 2, 1, 2, 3, 3, 3, 3, 1, 1, 2, 2, 2, 2, 3, 3, 3, 1, 3, 2, 1, 2, 1, 3] Try different initial probabilities to estimate the actual probabilities behind that observation and submit the weather you predicted. Explain why you think that is the distribution.

   - How will initial reconstruction of the weather (the leftmost graph) change?
   - How will the final graph after 10 iterations change?
   - What is the $p(1|H)$ after 10 iterations? Explain why this happened and what happened in each re-estimation step.

3. (**30 points**) Read and write a half page summary[2] and review of Merialdo (1994). It is available at `http://aclweb.org/anthology/J94-2001`.

[2] Single spaced, regular font, and one column is fine.

In addition to discussing the basic methodology and findings of this paper, identify findings you found interesting, surprising, or confusing. What is the overall takeaway (for you) from this paper?

The full Bibtex citation is

```
@article{merialdo1994tagging,
  title={Tagging English text with a probabilistic model},
  author={Merialdo, Bernard},
  journal={Computational linguistics},
  volume={20},
  number={2},
  pages={155--171},
  year={1994},
  publisher={MIT Press}
}
```

4. (**30 points**) In this question, you will implement a **supervised** HMM Part-of-speech tagger using Viterbi decoding. Using the training data to train, the development data to perform internal comparisons, and then perform a final evaluation where you use the training set to train and the test set to evaluate. Remember that you will need to implement this using log probabilities, and to include beginning and ending tags and observations on each sentence.

   Internal Development You should define the word vocabulary from the training set; note, your vocabulary could *exclude* words from the training set. How you smooth (either the transition or the emission probabilities) or define the vocabulary is up to you.

   Evaluation You should evaluate on both per-token **accuracy** (how many tokens were tagged correctly), *and* on per-sentence accuracy (how many sentences were tagged *completely* correctly). Per-token accuracy (as compared to say, macro F_1) is a legacy convention for POS tagging.

   Submit your code and a brief report describing:

   - Your analysis of the evaluation results, during both development and testing.
   - Any internal improvements you made during development.
   - Additional ideas to improve the result (you do *not* have to implement them).
   - What are the top 5 POS tags that are most commonly identified correctly, and what are the 5 POS tags that are most commonly identified *incorrectly*? You can use the `sklearn.metrics.{recall,precision,f1}_score` functions with `average = 'None'`, if you want (but there are other ways that are no more difficult to do the same analysis).

   Resources You can find descriptions of the general procedure in slide deck 13, starting on pages 46-55, and in §10.4 of 3SLP.

   If you are curious, you can find the state-of-the-art for POS tagging on the ACL's website. It was updated in 2016, but some new methods using RNN/LSTM claim to have better performance.

5. (**60 points**) In this question you will implement a semi-supervised HMM, as described in deck 14 (pages 35-45).

   Algorithm and Decoding You **must** implement one of the following options:

- "Regular" ("soft") EM, evaluated with both Viterbi and posterior decoding;
- Viterbi ("hard") EM, evaluated with both Viterbi and posterior decoding;
- Both "regular" ("soft") and Viterbi ("hard") EM, evaluated with Viterbi decoding;
- Both "regular" ("soft") and Viterbi ("hard") EM, evaluated with posterior decoding;

For **20 points of extra credit**, you may implement *both* hard and soft EM, evaluated with *both* Viterbi and posterior decoding. "Regular" EM is what we covered in class; Viterbi EM is described below.

You should run each EM for *at least* 10 iterations. This means you should make at least 10 complete passes over the training and raw data (see below). At the end of each iteration, track both the evaluation perplexity and accuracy of the decodings you're using (Viterbi, posterior, or both).

Data In addition to the training, development, and test data you've already used, you should also make use of unlabeled (with respect to POS tags) data; see the `raw.json` file. The number of sentences in the raw data is roughly the same as in the labeled data. The data structure is the same except there is no POS field.

Your vocabulary should be defined as some subset from the training and raw data; this will require some smoothing. How you combine these two sets to form the vocabulary is up to you.

What to Turn In In addition to the code you must also provide a report containing:

- Descriptions, analyses, and take-aways of internal development experiments.
- The average probability of the sequences in the test set given your semi-supervised model(s).
- The per-token accuracy and per-sentence accuracy of your decoded tag sequences for the test data set. Try to explain what happened, and why.
- Ideas to improve the algorithms. You do not have to implement these, but identify where your systems fail and ways you might correct for those failures.

What is Viterbi EM? In Viterbi EM, you obtain counts for each unlabeled sequence by using the Viterbi (most-likely tag sequence) decoding. This is in contrast to the "regular" EM we covered in class, where "new" counts were based on posterior probabilities.

For example, if you have an input sentence with a Viterbi POS sequence

```
Chris/NNP ate/VBN cake/NN ./.
```

then, based on this sentence alone, you would update the `NNP` to `VBN` transition count, but not any other transition starting from `NNP`. This contrasts with soft EM, where you update the transition counts according to the posterior distribution, e.g., $p(z_1 = \texttt{NNP}, z_2 = \texttt{VBN} \mid \texttt{Chris ate cake .})$.

Implementation Hints You will need to implement this using log probabilities. This requires you to initialize your forward α, backward β, and Viterbi γ tables correctly. You will also need to perform a (log)sum of log probabilities; the `logsumexp` function in `scipy` will be helpful.

Remember to include beginning and ending tags and observations on each sentence.

Make sure not to change the transition and emission probabilities while you're in an EM iteration! You will want to keep three versions of count tables: a "current" one, containing the counts needed to get the current transition and emission probabilities; an "original" or "gold" one, containing the gold counts from the training data; and a "new" one, containing the counts (expected or Viterbi) from the currrent EM iteration. Update the "current" ones only once you've run a complete EM cycle over your entire dataset.