TTIC 31210: Advanced Natural Language Processing
Assignment 4: Gibbs Sampling for HMMs (60 points)

Instructor: Kevin Gimpel
Assigned: Thursday, May 16, 2019
Due: 7:00 pm, Monday, June 3, 2019
Submission: email to `kgimpel@ttic.edu`

Submission Instructions

Package your report and code in a single zip file or tarball, name the file with your name followed by "_hw4", and email the file to `kgimpel@ttic.edu` by 7:00 pm on the due date. For the report, use pdf format or a Jupyter Notebook. Please do not use plain text files.

Collaboration Policy

You are welcome to discuss assignments with others in the course, but solutions and code should be written individually. You may use software libraries for helper code, but you must write the code for the inference algorithms yourself.

Overview

This assignment is a continuation of Assignment 3. You will use the same HMM model and data (and you are encouraged to reuse your codebase from Assignment 3). You will derive and implement a Gibbs sampler for sampling from the posterior distribution over the hidden variables given the input. You will use this sampler for various inference tasks, including both traditional argmax inference as well as minimum Bayes risk inference.

Data

You will be using the same manually-annotated English data as in Assignment 3. To review, the data available from the course web page contains the following two files:

- `en_ewt.train`: training data; contains 12,543 annotated sentences.
- `en_ewt.dev`: development data (DEV); contains 2,002 annotated sentences.

Each non-blank line corresponds to a single token in a sentence and follows the format "word[TAB]tag". Blank lines separate sentences.

Hidden Markov Models for Tagging

We will use a hidden Markov model (HMM) for part-of-speech tagging. We will use Y_t to denote the random variable corresponding to the label (tag) at position t. We will use X_t to denote the random variable corresponding to the symbol (word) at position t. A typical HMM uses the following conditional independence assumptions:

$$\text{for } k > 1 : Y_t \perp\!\!\!\perp Y_{t-k} \mid Y_{t-1}$$
$$\text{for } k > 1 : Y_t \perp\!\!\!\perp Y_{t+k} \mid Y_{t+1}$$
$$\text{for } k \neq 0 : X_t \perp\!\!\!\perp Y_{t+k} \mid Y_t$$

We will use $\boldsymbol{Y}$ to denote the full sequence of Y random variables where individual random variables are indexed as Y_t. We will use lowercase letters to denote values of random variables, i.e., y is a value that Y_t can take on and $\boldsymbol{y}$ is a value that $\boldsymbol{Y}$ can take on. We use analogous notation for the X random variables. The set of possible labels (tags) will be denoted $\mathcal{L}$ and the set of possible symbols (words) will be denoted $\mathcal{V}$.

Using these conditional independence assumptions, the probability of a length-T sequence of labels and symbols is:

$$P(\boldsymbol{X} = \boldsymbol{x}, \boldsymbol{Y} = \boldsymbol{y}) = P(\texttt{</s>} \mid Y_T = y_T) \prod_{t=1}^{T} P(Y_t = y_t \mid Y_{t-1} = y_{t-1}) P(X_t = x_t \mid Y_t = y_t)$$

where </s> is the end-of-sequence label and where we fix $Y_0 =$ <s>, the start-of-sequence label.

We parameterize the probability distributions in the above equation as follows. The parameters of the $P(Y_t \mid Y_{t-1})$ distributions are called the **transition probabilities** and we denote the probability of transitioning from tag y' to tag y as $p_{\boldsymbol{\tau}}(y \mid y')$. The parameters of the $P(X_t \mid Y_t)$ distributions are called the **emission probabilities** and we denote the probability of emitting word x from tag y as $p_{\boldsymbol{\eta}}(x \mid y)$. So, we write

$$P(\boldsymbol{X} = \boldsymbol{x}, \boldsymbol{Y} = \boldsymbol{y}) = p_{\boldsymbol{\tau}}(\texttt{</s>} \mid y_T) \prod_{t=1}^{T} p_{\boldsymbol{\tau}}(y_t \mid y_{t-1}) p_{\boldsymbol{\eta}}(x_t \mid y_t) \tag{1}$$

Use your code from the last assignment to estimate the HMM parameters with add-λ smoothing (using the prescribed λ values).

1. Gibbs Sampling for HMMs (40 points)

For supervised HMMs, there is no inference required during learning. However, to use the model to tag a new sentence $\boldsymbol{x}$, we need to solve the following argmax inference problem:

$$\hat{\boldsymbol{y}} = \operatorname*{argmax}_{\boldsymbol{y}} \; P(\boldsymbol{Y} = \boldsymbol{y} \mid \boldsymbol{X} = \boldsymbol{x}) \tag{2}$$

Typically, the Viterbi algorithm is used to exactly solve this test-time inference problem, as you implemented in Assignment 3.

In this assignment, you will use Gibbs sampling for this argmax inference problem instead of Viterbi. Gibbs sampling defines a way to draw samples from $P(\boldsymbol{Y} \mid \boldsymbol{X} = \boldsymbol{x})$ by repeatedly sampling from $P(Y_t \mid Y_{\text{-}t} = y_{\text{-}t}, \boldsymbol{X} = \boldsymbol{x})$ where $Y_{\text{-}t}$ represents all Y random variables other than Y_t and $y_{\text{-}t}$ denotes their values. In words, this means sampling a value for a single Y_t random variable conditioned on values for all other random variables.

- **(a) (5 points)** Show that

  $$P(Y_t = y \mid Y_{-t} = y_{-t}, \boldsymbol{X} = \boldsymbol{x}) \propto P(Y_t = y \mid Y_{t-1} = y_{t-1}) P(X_t = x_t \mid Y_t = y) P(Y_{t+1} = y_{t+1} \mid Y_t = y)$$

- **(b) (5 points)** Write down the two special cases below:

  $$p(Y_1 = y \mid Y_{-1} = y_{-1}, \boldsymbol{X} = \boldsymbol{x}) \propto \; ?$$

  $$p(Y_T = y \mid Y_{-T} = y_{-T}, \boldsymbol{X} = \boldsymbol{x}) \propto \; ?$$

- **(c) (15 points)** Implement Gibbs sampling for your HMM using the above formulas. Note: the above equations use "proportional to" ($\propto$), not equals. To get probability distributions for purposes of sampling, divide by normalizing terms obtained by summing the right-hand sides over all values y.

  You have to start your Gibbs sampler by initializing the **state** of the sampler. Here, the state consists of values for the Y_t variables. Initialize by sampling a tag for each position uniformly at random from the set of tags (excluding the special start and end tags).

  Let's define an **iteration** of Gibbs sampling as the sampling of each Y_t in the sequence conditioned on all other Y_t values in the current state. So a single Gibbs sampling iteration for a sentence with T words will require T sampling steps, each time sampling from the conditional distribution of one Y_t conditioned on all others. You can simply iterate through the Y_t random variables from left to right, or you can go through them in some other order.

  Implement the ability to run your sampler for a specified number of iterations K. After K iterations, evaluate the state by comparing its tags to the ground truth tags and recording the number that match. Run a completely new Gibbs sampler for K iterations on each sentence in the development set. Compute tagging accuracy by accumulating the counts of matching tags and total tags across all sentences (using only the final state for each sentence) and dividing.

  During implementation and debugging, you can print the number of sampling steps in which a random variable actually changed its value. You should see this number decrease and then stabilize after a small number of iterations (though it should not actually shrink to zero assuming the posterior has nonzero probability mass on multiple taggings, which it typically will).

  Submit your code.

- **(d) (5 points)** Run K iterations of Gibbs sampling for each sentence in the development set, for $K \in \{2, 5, 10, 50, 100, 500, 1000\}$, then take the final sample and record its counts of correct and total tags in order to compute the tagging accuracy of the entire development set. Report the tagging accuracies for all K values. Note that for each K value, you should run a completely new sampler for each development set sentence. For each K, report the total time required and the log-probability of the sentences in DEV, as you did in Assignment 3.

- **(e) (5 points)** Since we are doing test-time inference, we are more interested in finding a setting of the variables Y_t that maximizes the posterior, rather than simply generating samples from the posterior. One way to approximate this is to sharpen each conditional distribution in the sampler by raising every probability to the power β and then renormalizing to get a distribution. For $\beta < 1$, this will flatten the distribution, and for $\beta > 1$, this will sharpen the distribution.

  The experiments in (d) used $\beta = 1$. Repeat the experiments from (d) (for all K values) but now vary β. Experiment with $\beta \in \{0.5, 2, 5\}$ and report your tagging accuracies for all K values and all β values. What trend do you find as you increase β?

- **(f) (5 points)** Repeat (e) but this time begin with $\beta = 0.1$ for each sentence and then increase β by adding 0.1 after each sampling iteration for that sentence. For the next sentence's sampler,

reset β again to 0.1 and repeat. Report your tagging accuracies for all K values considered above. Experiment with a few other schedules of varying β across iterations to see if you can get better resulting tagging accuracies.

This practice is often described as "annealing" as it is similar to methods like simulated annealing and deterministic annealing. Starting with $\beta < 1$ helps the sampler to mix faster (in order to converge to generating samples from the true posterior), then increasing β over time causes the sampler to generate samples from the sharpened posterior. As β increases, sampling from the sharpened posterior should get closer and closer to the argmax operation that the Viterbi algorithm performs.

2. Gibbs Sampling for Minimum Bayes Risk Inference (20 points)

The Viterbi algorithm solves the argmax inference problem, which is often called *maximum a posteriori* or MAP inference. However, there are other criteria that can work better in practice, especially for structured prediction.

One popular generalizing framework is that of minimum Bayes risk (MBR) inference. For HMMs, MBR inference has the following form:

$$\hat{\boldsymbol{y}} = \operatorname*{argmin}_{\boldsymbol{y}} \sum_{\boldsymbol{y}'} p(\boldsymbol{Y} = \boldsymbol{y}' \mid \boldsymbol{X} = \boldsymbol{x}) \operatorname{cost}(\boldsymbol{y}, \boldsymbol{y}') \tag{3}$$

MBR inference is often thought of as choosing the "consensus" output, i.e., the output that is closest on average to all likely outputs, where closeness is measured by a user-defined cost function (Kumar and Byrne, 2004). The cost function is typically chosen by the modeler to capture domain knowledge about the similarity of two outputs and/or to relate to the evaluation metric for the task.

We'll consider two cost functions. First let's define the **0-1 cost**:

$$\operatorname{cost}_{0-1}(\boldsymbol{y}, \boldsymbol{y}') = \mathbb{I}[\boldsymbol{y} \neq \boldsymbol{y}']$$

where $\mathbb{I}[f]$ returns 1 if f is true and 0 otherwise. The 0-1 cost does not give any partial credit. It returns a cost of 1 if the two label sequences $\boldsymbol{y}$ and $\boldsymbol{y}'$ differ in only one position.

The second cost function we will consider, which does give partial credit, is the **Hamming cost**:

$$\operatorname{cost}_{\text{Hamming}}(\boldsymbol{y}, \boldsymbol{y}') = \sum_{t=1}^{T} \mathbb{I}[y_t \neq y'_t]$$

This cost function counts the number of tags that don't match between the two tag sequences.

- **(a) (5 points)** Show that MBR inference (solving Eq. 3) reduces to MAP inference (solving Eq. 2) when using 0-1 cost.
- **(b) (5 points)** It can be shown that MBR with Hamming cost reduces to

$$\text{for all } t,\ \hat{y}_t = \operatorname*{argmax}_{y} P(Y_t = y \mid \boldsymbol{X} = \boldsymbol{x}) \tag{4}$$

(You do not need to prove this.) In sequence labeling models, MBR inference with Hamming cost is often called **posterior decoding**, as it chooses y_t by maximizing the posterior distribution over the random variable Y_t given the input. You could use the forward and backward dynamic

programming algorithms to efficiently compute these posterior distributions exactly. However, in this assignment you will use your Gibbs sampler to estimate these posteriors.

Note that your Gibbs sampler gives you samples from $P(\boldsymbol{Y} \mid \boldsymbol{X} = \boldsymbol{x})$. For MBR decoding, we need to calculate

$$P(Y_t = y \mid \boldsymbol{X} = \boldsymbol{x}) = \sum_{y_{\text{-}t}} P(Y_t = y, Y_{\text{-}t} = y_{\text{-}t} \mid \boldsymbol{X} = \boldsymbol{x}) \tag{5}$$

We can use the samples from the Gibbs sampler to approximate this summation and therefore to estimate each posterior probability $P(Y_t = y \mid \boldsymbol{X} = \boldsymbol{x})$. How can we use our samples for this? Well, first note that we can use our samples to estimate the posterior probability of any complete set of values $\boldsymbol{y}$ as follows:

$$P(\boldsymbol{Y} = \boldsymbol{y} \mid \boldsymbol{X} = \boldsymbol{x}) \approx \frac{1}{K} \sum_{i=1}^{K} \mathbb{I}[\tilde{\boldsymbol{y}}^{(i)} = \boldsymbol{y}] \tag{6}$$

where $\tilde{\boldsymbol{y}}^{(i)}$ denotes the values of $\boldsymbol{Y}$ in sample i. That is, we count how many samples have that set of values and divide by the total number of samples K. Derive the analogous equation representing the estimate for the posterior probability of a value y for a single random variable Y_t, by using Equations 5 and 6 and then simplifying as much as possible:

$$P(Y_t = y \mid \boldsymbol{X} = \boldsymbol{x}) \approx \ ?$$

- **(c) (10 points)** Implement MBR inference (Eq. 4) using your Gibbs sampler's samples and your estimator from part (b). Submit your code.

  Typically when researchers use multiple samples from an MCMC sampler, they throw out the first few samples (to allow for a "burn-in" period while the sampler is getting started) and also select samples that are separated across sampling iterations. E.g., they may take every 10th sample, starting from sample 100. In your implementation, you can experiment with these techniques if you wish, but be sure to first report results when using every sample that your sampler produces.

  Compare tagging accuracies across different numbers of sampling iterations, in particular when using $K \in \{2, 5, 10, 50, 100, 500, 1000\}$ and fixing $\beta = 1$. Then experiment with different fixed values of β and report how the results vary with different β values. What trend do you find when you increase β? Is it different from the trend you found above when only using the final sample? Why do you think this occurs? Report inference times but do not report log-probabilities because the goal here is not to maximize the full posterior.

References

Shankar Kumar and William Byrne. Minimum Bayes-risk decoding for statistical machine translation. In *HLT-NAACL 2004: Main Proceedings*, pages 169–176, Boston, Massachusetts, USA, May 2 - May 7 2004. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/N04-1022. [4]