TTIC 31210: Advanced Natural Language Processing
Assignment 5: Nonparametric Bayesian Segmentation (60 points)

Instructor: Kevin Gimpel
Assigned: Friday, May 31, 2019
Due: 7:00pm, Friday, June 14, 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 "_hw5", 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

In this assignment, you will build an unsupervised segmenter for English text from which spaces have been removed. You will evaluate your model by seeing how accurately it can recover the spaces in the original text. You will use nonparametric Bayesian priors and collapsed Gibbs sampling for inference.

Data

You will use sentences from the Children's Book Test dataset (Hill et al., 2016), which contains text from books from Project Gutenberg. We have prepared the following files, including one with the sentences with spaces removed and one containing the ground truth segmentation boundaries (space locations):

- `cbt-characters.txt`: 10,000 sentences of text, lowercased, with one sentence per line.
- `cbt-boundaries.txt`: the corresponding ground truth segmentation boundaries. This file has the same number of characters as the above file, but characters are replaced with binary values indicating whether or not there exists a boundary *after* the corresponding character.
- `cbt-original.txt`: the original text from which spaces were removed (can be used for one of the optional components).

For example, a pair of lines from `cbt-characters.txt` and `cbt-boundaries.txt` is below:

```
saidtheking.
000100100011
```

The binary values indicate that in the original data, there are word boundaries after the `d`, `e`, `g`, and period (`.`). Note: there is always a boundary after the final character; i.e., every line in `cbt-boundaries.txt` ends in `1`. Since this is unsupervised learning, you should only use the `cbt-boundaries.txt` file for evaluation.

A Probabilistic Model for Segmentation

We now define a latent-variable probabilistic model for segmenting a character stream. We use X to denote a character sequence comprising a line in the `cbt-characters.txt` file and we use Z to denote the latent segmentation of X into segments. The type of Z is a partition of a length-$|X|$ sequence into adjacent ranges.

We will use lowercase letters to denote values of random variables, so x is a value of X. Empty character sequences x are not permitted. We use x_r to denote the subsequence of x corresponding to the range $r = (r_1, r_2)$, where r_1 is the start of the range and r_2 is the end of the range.

Our model will generate the character sequence x one segment at a time using a binomial model for the number of segments. The joint probability of X and Z is parameterized using parameter values θ and γ and written as follows:

$$p(X = x, Z = z \mid \theta, \gamma) = (1-\gamma)^{|z|-1}\gamma \prod_{k=1}^{|z|} p(x_{z_k} \mid \theta) \tag{1}$$

where γ is the probability of terminating the sequence of segments and therefore $1-\gamma$ is the probability of generating another segment. The terms before the product account for $|z|-1$ decisions to continue generating segments and the final decision to stop generating segments. Then, the product over segment indices k accounts for the probability of generating each segment x_{z_k} from a categorical distribution parameterized by θ. Rather than learn or infer γ, we will treat it as a tunable hyperparameter and simply fix it to a value in the interval $(0, 1)$.

Since γ is fixed, the only parameters to be learned are those of the categorical distribution over segments, $p(x_{z_k} \mid \theta)$. We could learn θ by maximizing log marginal likelihood. However, this would lead to a degenerate solution unless we heavily constrain the "support" of $p(x_{z_k} \mid \theta)$, i.e., the segments that can have nonzero probability. If left unconstrained, the maximum likelihood solution will lead to trivial segmentations that only have a single segment for each sentence in the data.

Nonparametric Priors

Instead of learning θ directly, we will let θ be drawn from a Dirichlet process prior and integrate out θ. This choice permits us to have unbounded support for the posterior distribution over segments while also displaying a "rich-get-richer" property that favors a relatively small subset of possible segments.

More formally, let $\theta \sim \mathrm{DP}(G_0, s)$ where G_0 is the **base distribution** and s is the **strength** parameter of the Dirichlet process prior. Where y is a character sequence of length $|y|$, we define the probability function for the base distribution G_0 as follows:

$$G_0(y) = (1-\beta)^{|y|-1}\beta \prod_{i=1}^{|y|} p_{\text{char}}(y_i)$$

where y_i is the ith character in y, β is the stopping probability, and p_{char} is a distribution over characters. The terms before the product account for $|y|-1$ decisions to continue generating characters and the final decision to stop generating characters. The product accounts for the probability of generating each character from p_{char}. The simple base distribution above defines a distribution over an infinite set of character sequences.

In your initial implementation, simply use a uniform distribution over characters for p_{char}. You will also experiment with using a unigram character distribution estimated from the training set. While the base distribution parameters could be learned or inferred, for this assignment you should just keep them fixed. That is, you should fix or estimate p_{char} before inference and keep it fixed during inference. Also, you should treat β like a hyperparameter to tune (like γ) and experiment with various fixed values.

Note that we could replace the simple decomposition above with a character n-gram model that generates each character conditioned on the previous $n-1$ characters, or a character LSTM that decomposes into a product over all positions, or other similar sorts of models. One of the optional components of this assignment involves implementing a bigram character model for this base distribution.

Inference via Collapsed Gibbs Sampling

We assume our dataset has N sentences, each of which is a value for the random variable X, so we denote this set of random variable values by $\{x^{(j)}\}_{j=1}^{N}$. We use $\{Z^{(j)}\}_{j=1}^{N}$ to denote the set of latent segmentation random variables corresponding to these N sentences. We will use collapsed Gibbs sampling to obtain samples from the collapsed posterior distribution $p(\{Z^{(j)}\}_{j=1}^{N} \mid \{x^{(j)}\}_{j=1}^{N}, \gamma, G_0, \beta)$ where θ has been integrated out. Our sampler will follow the procedure of Goldwater et al. (2009). See Appendix A.1 in their paper for more background and derivations, though since they also integrate out γ, their sampling formulas are more complex than ours.

We will shift to different notation for purposes of defining the collapsed Gibbs sampler. Rather than sample new values for the latent segmentation variable Z explicitly, we will instead work with a different view of the latent segmentation. In particular, we introduce a boundary vector b that is derived from the latent segmentation variable value z for a sentence x. The boundary vector b has length $|x|$ and each $b_i \in \{0,1\}$ indicates whether a segment boundary appears immediately after position i in the segmentation z. The final value, $b_{|x|}$, is always fixed to 1, since the end of a sentence is always the end of a segment. The start of a sentence is always the start of a segment, though this fact is not explicitly encoded in the boundary vector. These binary boundary vectors have the same type as the lines in the `cbt-boundaries.txt` file.

The **state** of the sampler consists of values b for all sentences x in the dataset. The sampler will iterate through every possible boundary position b_i in the data (except for the final one in each sentence, which is always fixed to 1) and sample a new value for b_i from the set $\{0,1\}$. If $b_i = 0$, then a single segment crosses the potential boundary b_i and we denote this "full" segment by y_{full}. If $b_i = 1$, then a boundary divides y_{full} into two segments, which we denote y_{prev} and y_{next}. The starting positions of y_{full} and y_{prev}, and the ending positions of y_{full} and y_{next}, are determined based on the other boundary values in the current state of the sampler.

Before specifying the probabilities of the two values for a boundary b_i for our sampler, we first define notation to handle the two cases for including the stopping probability for a segment y:

$$t(y) = \begin{cases} \gamma & \text{if } y \text{ ends at the end of its sentence} \\ 1-\gamma & \text{else} \end{cases}$$

That is, $t(y)$ returns the stopping probability if y is the last segment in the sentence, and one minus the stopping probability otherwise.

Now, the probability of choosing the value 0 is given as follows:

$$p(b_i = 0 \mid b_{-i}, \ldots) \propto \frac{n^{(-i)}_{y_{\text{full}}} + sG_0(y_{\text{full}})}{n^{(-i)} + s} \times t(y_{\text{full}})$$

where b_{-i} denotes values for all boundaries other than b_i, "..." stands for all other variables being conditioned on in the collapsed posterior, $n^{(-i)}_{y_{\text{full}}}$ refers to the number of times the segment y_{full} appears in the state of the sampler aside from the segments influenced by boundary b_i, and $n^{(-i)}$ is the total number of segments in that same set (namely, the segments in the state of the sampler aside from the segments influenced by b_i). The segments influenced by b_i are only y_{full}, y_{prev}, and y_{next}, so the counts are computed over all other segments in the state.

The probability of choosing the value 1 is:

$$p(b_i = 1 \mid b_{-i}, \ldots) \propto \frac{n^{(-i)}_{y_{\text{prev}}} + sG_0(y_{\text{prev}})}{n^{(-i)} + s} \times (1-\gamma) \times \frac{n^{(-i)}_{y_{\text{next}}} + \mathbb{I}[y_{\text{prev}} = y_{\text{next}}] + sG_0(y_{\text{next}})}{n^{(-i)} + 1 + s} \times t(y_{\text{next}})$$

where the indicator $\mathbb{I}[y_{\text{prev}} = y_{\text{next}}]$ is included because the conditional probability of y_{next} has y_{prev} in its conditioning context. This is also the reason why the denominator has an extra "+1". Note that instead of $t(y_{\text{prev}})$ we explicitly write out $(1-\gamma)$ because we know that y_{prev} is not the final segment because there is another segment after it, namely y_{next}.

Note that the above formulas can be simplified slightly because $t(y_{\text{full}}) = t(y_{\text{next}})$. That is, y_{full} ends at the end of its sentence if and only if y_{next} ends at the end of its sentence. Therefore, the same term appears in both right-hand sides and will not affect the distribution after normalization. (So, you can actually leave out the $t()$ terms in your implementation because they will not affect the sampler, but be sure to keep the $(1-\gamma)$ term!)

To implement this sampler, you will need to define data structures to represent the counts of all segments in the current state. It is common to handle the count tracking in conjunction with the sampling formulas as follows. Before you compute the above probabilities for sampling a new value for b_i, subtract 1 from the counts for the segments influenced by b_i that are consistent with the current value for b_i. After sampling the new value, add 1 to the counts of the new segments that were created. If, when sampling a new value for b_i, your sampler chooses the same value it had before, then you will be subtracting and adding 1 to the same counts, so the counts will not change. Also, keep track of the total count $n^{(-i)}$ and whenever you add or subtract 1 to the count for any segment above, add or subtract 1 to the total count $n^{(-i)}$ as well. A good way to check for bugs in your sampler and count tracking is to print an error message when any count becomes negative.

We will define an **iteration** of the Gibbs sampler as proceeding through all boundary positions in the dataset (aside from sentence-final positions) and sampling a value for each. Note that unlike Gibbs sampling for inference in HMMs, where you created a completely new Gibbs sampler for each sentence, here you will use a single sampler for the entire dataset and an iteration of the sampler involves iterating over the whole dataset.

Initialization

We have to initialize the state of the Gibbs sampler. You can experiment with different ways of doing this if you want, but as a default method, place a boundary after a character with probability γ. Ensure that there is always a boundary after the final character in each sentence.

Evaluation

The state of the sampler consists of boundary values b for all sentences in the dataset. This state corresponds to a segmentation of the dataset and can be evaluated automatically by computing the **boundary prediction accuracy (BPA)** using the `cbt-boundaries.txt` file as the gold standard. BPA is computed simply by checking how many of the predicted and gold boundaries match, *excluding the final boundary in each sentence* (because it is always fixed to 1 and therefore always correct). You should print the BPA for the initial state of your Gibbs sampler and after every iteration. The BPA for an iteration is computed using only the state at the end of that iteration.

It is often helpful to print additional diagnostics during each iteration of Gibbs sampling. In addition to reporting BPA, record and print the **number of boundary changes (NBC)** that occurred. That is, the NBC for an iteration is the number of b_i variables that changed values during that iteration. The NBC should reduce over time and stabilize at a value that is consistent across iterations. It should not go to zero (unless you are using aggressive temperatures or annealing strategies). Also, print the number of unique segments, i.e., the number of segment types, in your segmented corpus. Note that this is different from $n^{(-i)}$, which represents a count of segment tokens, not types.

1. Gibbs Sampling Implementation (20 points)

Implement the model and collapsed Gibbs sampling procedure described above. To get started with the implementation, compute and report the BPA of the unsegmented corpus. This entails reading in the data and computing BPA. (Hint: I found this to be ~~73.5602%~~ 680060/928089 = 73.2753%.) Using my implementation, running the Gibbs sampler takes 2-3 seconds per iteration. For p_{char}, use a uniform distribution over the set of characters observed in the data. Submit your code.

2. Experimentation (20 points)

- **(a) (7 points)** Run experiments with your Gibbs sampler for various values of the hyperparameters γ, s, and β. Run at least 100 iterations of Gibbs sampling for each and report the BPA of the final iteration. Figure out which hyperparameters matter and which have very little effect. Note: we are not using a train/dev/test split for this assignment (because this is not a standard task or dataset), so simply do your tuning based on BPA on the single dataset provided.
- **(b) (7 points)** Implement the ability to use a unigram character distribution for p_{char}. Estimate the unigram distribution from the dataset. Experiment with this model using the best hyperparameter values from part (a) (or you can retune the hyperparameters if you like). Does this model improve over using a uniform character distribution?
- **(c) (3 points)** Since we are only evaluating the final sample, we only actually care about producing a sample with high probability under the collapsed posterior. So, try annealing to alter the probabilities over the sampling iterations so as to yield a final sample with high posterior probability. That is, as you did in Assignment 4, raise each probability to the power τ and then renormalize to get a distribution. For $\tau < 1$, this will flatten the distribution, and for $\tau > 1$, this will sharpen the distribution. Experiment with annealing schedules that begin with $\tau < 1$ and eventually reach $\tau = 1$ by the end of sampling (you may need more than 100 iterations of Gibbs sampling for this to work). Note that such schedules will make the posteriors flatter than they otherwise would be, which may help the sampler to "mix" faster early on during sampling. Can you find a schedule that improves over using a fixed $\tau = 1$?
- **(d) (3 points)** Experiment with annealing schedules that start with $\tau < 1$ and end with $\tau > 1$. These schedules will flatten the distributions early on to help the sampler mix faster, then sharpen the distributions later on. Can you find a schedule that works well in this setting?

3. Analysis (10 points)

Using your best setting from part 2, print the 200 segments with the largest counts from the final sampling iteration. Also, print the first 20 sentences of the segmented data from the final sample. Inspect both the segmented sentences and the top 200 segments and look for segments that indicate signs of (a) under-segmentation and (b) over-segmentation. List 10 examples of each in your report. Are they reasonable errors? You may also want to look at other sentences in the final sample to see these segments in context to gain more intuition about why they are occurring. Are there cases where you think your model is doing a better job at placing spaces than the original text (i.e., better than the cultural process of language formation and evolution)? Do you see signs of overfitting to the domain of the given dataset?

4. Your Choice (10 points)

Do one (1) of the following:

1. You previously compared uniform and unigram character distributions in the base distribution. Implement the ability to use a bigram character distribution and estimate the bigram probabilities from the training sentences just like you estimated the character unigram distribution. Experiment with using this character bigram model in the base distribution and report your findings.

2. You can readily apply your framework to text data with spaces preserved. Run your Gibbs sampler on such a dataset, namely the `cbt-original.txt` file. You will no longer be able to compute BPA, but you can look at the segmented output and the list of the most frequent segments from the final iteration. Do this, and compare these two types of outputs qualitatively to the analogous ones when using text without spaces (that you analyzed in part 3). How does the presence of spaces affect things? What appears to be essentially the same between the two settings and what differs? Try applying your framework to another text domain (with spaces preserved), analyze the output, and discuss your findings.

3. Our tuning procedure in part 2(a) used the gold standard boundaries for tuning hyperparameters based on BPA. Consider a less-supervised method for tuning and model selection based on the type/token ratio. That is, we want the type/token ratio of our segmented output to match the type/token ratio of real text (with spaces preserved). Compute and print the segment type/token ratio after every iteration of Gibbs sampling. Compute the type/token ratio of a large text corpus and use it as your "ideal" type/token ratio, then use closeness to that ideal as the tuning criterion. When using type/token ratio for tuning hyperparameters, how close are your results to the supervised tuning that you did in part 2(a)?

References

Goldwater, S., Griffiths, T. L., and Johnson, M. (2009). A Bayesian framework for word segmentation: Exploring the effects of context. *Cognition*, 112(1):21–54. [3]

Hill, F., Bordes, A., Chopra, S., and Weston, J. (2016). The Goldilocks principle: Reading children's books with explicit memory representations. In *Proceedings of International Conference on Learning Representations (ICLR)*. [1]