CSC2515 – Assignment #3

Due: Nov.29, 2005, 2pm at the **START** of class
Worth: 18%
Late assignments not accepted.

1 Aggregate Markov Models

In this assignment you will experiment with training the Aggregate Markov Model (AGMM) of Saul and Pereira on bigram word counts taken from a large corpus of English text. The AGMM is a mixture model which uses a single discrete latent variable z to model the conditional probability that a word w_1 is followed by a word w_2 in a typical English sentence. The model equations are very simple:

$$p_{ij} = P(w_2 = j|w_1 = i) = \sum_{c=1}^{C} p(w_2 = j|z = c)p(z = c|w_1 = i) = \sum_{c=1}^{C} r_{jc}q_{ci}$$

The parameters are the singly stochastic matrices $\mathbf{Q}$ and $\mathbf{R}$. Each column $\mathbf{R}(:, c)$ sums to unity and represents the distribution over words given that we are in cluster c; each column $\mathbf{Q}(:, i)$ sums to unity and represents the distribution over clusters given that we are have just seen word i. The model is trained to maximize the likelihood of a set of English sentences, under the very simple assumption that word sequences are generated under the Markov model:

$$P(w_1, w_2, w_3, \ldots, w_T) = P(w_1)\prod_{t=2}^{T} P(w_t|w_{t-1})$$

We assume that the first word, w_1 is always the special "beginning of sentence" word (BOS), so that $P(w_1) = 1$.

If we also assume that sentences are independent and identically distributed, this means the log likelihood of a set of sentences depends only on the observed counts $N(w_1, w_2)$, i.e. the total number of times w_2 was observed to follow w_1 in the training corpus:

$$\ell = \sum_{w_1,w_2} N(w_1, w_2)\ln P(w_2|w_1) = \sum_{ij} N_{ij}\ln p_{ij}$$

This likelihood can be conveniently (locally) maximized using the EM algorithm.
The updates for the E-step and M-step are given below:

$$\textbf{E-step}: \quad P(c|w_1, w_2) = \frac{P(w_2|c)P(c|w_1)}{\sum_{c'} P(w_2|c')P(c'|w_1)}$$

$$\textbf{M-step}: \quad P(c|w_1) \leftarrow \frac{\sum_w N(w_1, w)P(c|w_1, w)}{\sum_{wc'} N(w_1, w)P(c'|w_1, w)}$$

$$\textbf{M-step}: \quad P(w_2|c) \leftarrow \frac{\sum_w N(w, w_2)P(c|w, w_2)}{\sum_{ww'} N(w, w')P(c'|w, w')}$$

2 Derivations (5%)

Using the class notes, Saul and Pereira's paper, and what you know about the EM algorithm from your readings, do the following derivations and hand in your work (please don't hand in more than about one page for each derivation, hopefully less):

- By assuming that each sentence is modeled using a simple bigram Markovian model and assuming that all sentences are distributed IID, derive the log likelihood equation (eq.5 in the paper and on the previous page). Assume that the first word in every sentence is a special BOS word so we don't have to worry about modeling that probability. [Hint: try expressing the probability in the form of a product of a bunch of things raised to some powers which are either zero or one.]
- Derive the M-step updates (eqs. 3,4 in the paper and on the previous page), by (a) computing the expected complete log-likelihood and (b) taking its derivative with respect to the columns of **Q** and **R**. [Hint: don't forget to enforce that each column of **Q** and **R** must sum to unity, e.g. by using Lagrange multipliers.]

3 Data (3%)

On the course website you can download two data files, one with training data and one with testing data. Each file contains a sparse matrix whose ij^{th} entry is the number of times the bigram w_i, w_j occurred in the training corpus. The mapping from word numbers to actual words is given in the file `wordlist`. Word 1 is the "unknown word", word 2 is the BOS marker and word 3 is the "end of sentence" (EOS) marker. Every sentence begins with BOS and ends with EOS.

To get you thinking about the data a bit and learning how to explore it, answer these seven easy questions:

- How many sentences of data were used to make the training set? The test set? (Give exact answers.)
- How many words in total (including unknown words but not counting BOS or EOS markers) were used to make the training set? The test set? (Give exact answers.)
- What is the average sentence length (to the nearest word) in the training/testing set.
- What fraction of the words in the training/testing set are the "unknown" word?
- How many word pairs w_i, w_j have zero counts in the training set but nonzero counts in the test set?
- How many word pairs w_i, w_j have zero counts in the test set but nonzero counts in the training set?
- What is the 31^{st} most common word (not counting BOS and EOS) in the training data? In the test data?

4 Experiments (10%)

4.1 Unigrams

To get you started, first fit some a unigram model to the data. Assume that the first word is always BOS, so don't compute any probabilities for the BOS token. For all the other words, compute a maximum likelihood unigram (0-order model) using the training data and also a unigram model estimated using plus-one smoothing.

- What is the average perplexity of the test set under the unsmoothed unigram? Under the smoothed unigram?
- What is the perplexity of the training set under the unsmoothed unigram? Under the smoothed unigram?

4.2 Bigrams

Next, fit a maximum likelihood bigram (1st-order Markov model) to the training data as well as a bigram model estimated using plus-one smoothing. Assume that the first word in every sentence is always BOS, so don't compute any probabilities of the form $p(BOS|\cdot)$. (But of course, do compute terms $p(\cdot|BOS)$.)

- What is the perplexity of the training set under the unsmoothed bigram? Under the smoothed bigram?
- What is the perplexity of the test set under the smoothed bigram? (Under the unsmoothed bigram it is infinite.)

4.3 Aggregate Markov Model

Train an Aggregate Markov Model with $C = 32$ clusters by maximizing the likelihood of the training data using the EM algorithm. Train EM for 30 iterations. Here's what to hand in:

- A plot of the training set perplexity and the test set perplexity (on the same plot but using different symbols) as a function of the number of EM iterations from 1 (not 0) up to and including 30. (I'm essentially asking you to reproduce Figures 1a and 1b from the paper in a single plot.) Add to this plot horizontal lines for the smoothed unigram and smoothed bigram test and training perplexities.
- What is the training set perplexity after you randomly initialize the parameters (as suggested below) but before you do your first iteration of EM? What about the test set perplexity?
- Generate a sample sentence from your trained model. To do this, start with $w_1 = BOS$, and sample words from $p(w_t|w_{t-1}) = \sum_c p(w_t|c)p(c|w_{t-1})$ until you sample EOS, then stop. Convert your sample into words using the provided wordlist. Generate a few samples (don't bother trying *too* many) and hand in the one you think looks most like real English.

4.4 Bonus (up to 4% extra)

- Train the AGMM with $C = 1, 2, 4, 8, 16, 32, 64, 128, 256$ clusters and report the training and test set perplexities for each model size. (Essentially, reproduce table 1 from the paper.)
- Try many random initializations and other initialization schemes than the one I suggested. For each initialization, run EM with $C = 32$ for 30 iterations. Make the same plot you made before of test and training perplexity as a function of EM iterations, but this time show (e.g. with error bars) the max, min and median training/test complexity across all your random initializations. Make an additional scatter plot of training perplexity vs. test perplexity, including all runs at all possible iterations of EM.

5 Suggestions

Here are some tips which you might find helpful:

- Initialize each element of $\mathbf{Q}$ and $\mathbf{R}$ to $0.1 + 0.9u$ where u are uniformly distributed numbers between 0 and 1. Then renormalize $\mathbf{Q}$ and $\mathbf{R}$ so their columns sum to unity.
- Since BOS appears at the beginning of every sentence and nowhere else, $p(BOS|c)$ will always be zero for all c.
- Since EOS appears at the end of every sentence and nowhere else, $p(c|EOS)$ is undefined for all c. You can either leave these parameters out of the model entirely, or include a special case in your M-step code to set them to `NaN` or something like that.
- Do all your calculations using sparse storage and computation. Obviously, you don't want to be pushing around 60K by 60K matrices in which almost all the entries are zero! In Matlab, there are very convenient sparse matrix functions for doing everything you need. In particular, you only need to compute the E-step estimates $p(c|w_1, w_2)$ for pairs (w_1, w_2) that have nonzero counts. The summations in the numerators and denominators of the M-step equations should similarly range only over nonzero elements of N_{ij}. The same goes for computing likelihood.
- If you are using a vectorized language like Matlab, don't loop over words. You can do everything easily using loops only over c. (And, if you are an obsessive hacker, you can do away with loops entirely.)
- Do some sanity checks. As I said above, $p(BOS|c)$ should always be zero for all c. $p(c|EOS)$ should never be read by any piece code. The columns of $\mathbf{Q}$ and $\mathbf{R}$ should always sum to unity. The average perplexity on the training set should decrease after every iteration of EM.
- Perplexity: the average perplexity of a model is defined as the exponential of the average log probability it assigns to a word in a sentence, given the previous word. Intuitively, you can think of perplexity as the "effective number of words that might come next". A model that knew absolutely nothing about the date would have a perplexity equal to the vocabulary size (60K in this case) and a model that was magically able to predict the next word perfectly every time would have a perplexity of zero. To compute average perplexity, first compute the average (not total) log-probability per transition (or per word in the unigram case) and then take the negative exponential. For this assignments, all your perplexities should be on the order of a few hundred words (your average log likelihoods should be between around -7 to -5), which tells you that the models you are training would need less than 10 bits per word on average to encode the sentences.