

CS 294 - Practical Machine Learning - Homework 4

This assignment is due Tuesday, April 8, 2008. For questions about Part I, contact Erik Sudderth; for Part II, contact Fei Sha.

Part I: Hidden Markov Models

Preliminaries This part of the assignment will be done entirely in R. You will need to use the package `hmm.discnp`, and source the auxiliary functions provided in `hw4-hmmFuncs.R`. The script `hw4-hmmScript.R` provides partial solutions, and comments indicating where additional code is needed. *WARNING:* Training HMMs via the EM algorithm may take a few hours of computation time, so start early!

Problem 1: EM Training & Model Order Selection We begin by learning hidden Markov models (HMMs) which describe the statistics of English text. In this application, each discrete “time” point corresponds to a single letter. For training, we use a chapter from Lewis Carroll’s *Alice’s Adventures in Wonderland*, available in `aliceTrainRaw.txt`. To simplify the modeling task, we first converted letters to lower-case and removed all punctuation. The resulting text, stored in `aliceTrain.txt`, is a sequence composed of 27 distinct characters (26 letters, as well as whitespace encoded via an underscore ‘_’).

- (a) The method `hmm`, provided by the R package `hmm.discnp`, is an implementation of the EM algorithm for ML parameter estimation in HMMs. Use this package to learn HMMs with different hidden state dimensions (for example, try models with $N = 1, 3, 5, 10, 15, 20, 30$ states). Note that the model with $N = 1$ assumes that characters are independent. For each of these models, compute the log-likelihood which it assigns to the training sequence. Save these models for later sections.

In many applications of HMMs, there is insufficient data to select the model order via cross-validation. In these situations, the state dimension is often selected via either the *Akaike information criterion (AIC)* or *Bayesian information criterion (BIC)*. Let $y = (y_1, \dots, y_T)$ denote the observed training sequence, $x = (x_1, \dots, x_T)$ a hidden state sequence, and $\hat{\theta}_N$ an ML estimate of the parameters for an HMM with N states:

$$\hat{\theta}_N = \arg \max_{\theta_N} p(y | \theta_N) = \arg \max_{\theta_N} \sum_x p(y | x, \theta_N) p(x | \theta_N) \quad x_t \in \{1, \dots, N\}$$

For this model, the AIC and BIC take the following form:

$$\begin{aligned} \text{AIC}_N &= \log p(y \mid \hat{\theta}_N) - d(N) \\ \text{BIC}_N &= \log p(y \mid \hat{\theta}_N) - \frac{1}{2}d(N) \log(T) \end{aligned}$$

Here, $d(N)$ is the *number* of parameters for an HMM with N states. The “best” model is then the one for which AIC_N or BIC_N is largest.

- (b) Derive a formula for the number of parameters d in an HMM with N hidden states, and observations taking one of M discrete values. Remember to account for normalization constraints (for example, a discrete distribution on 4 events has only 3 degrees of freedom, since the probabilities of these events must sum to one). Plot the training log-likelihood $\log p(y \mid \hat{\theta}_N)$, AIC_N , and BIC_N versus N for the HMMs learned in part (a). Which criterion favors simpler models?
- (c) To test our learned HMMs, we use the text from a different chapter of *Alice’s Adventures in Wonderland*, available in `aliceTest.txt`. Using `loglike.hmm`, evaluate the test chapter’s log-likelihood with respect to each HMM learned in part (a). Plot these test log-likelihoods versus N . Which model selection criterion better predicted test performance?
- (d) Using the method `sampleText`, generate a random 500-character sequence from three different HMMs: the model with no temporal dependence ($N = 1$), the model with the highest BIC_N , and the model with the highest AIC_N . Compare and contrast these sequences. What aspects of English text do they capture? What do they miss?

Problem 2: Filling in Missing Letters In addition to computing likelihoods, HMMs lead to an efficient forward-backward algorithm which estimates the posterior probabilities of unobserved state sequences. This problem uses this method to estimate the identities of characters which have been *erased* from a text document.

Let $x_t \in \{1, \dots, N\}$ denote the hidden state at position t , and y_t the “true” character at position t of some document. Suppose that instead of observing y , we observe an alternative sequence z in which some letters have been erased. We assume that each letter is independently erased with probability ϵ , so that

$$\Pr[z_t = y_t \mid y_t] = 1 - \epsilon \qquad \Pr[z_t = * \mid y_t] = \epsilon$$

where ‘*’ is a special erasure symbol. Figure 1 shows a graphical model describing this generative process. Note that we never observe an “incorrect” letter; z_t is always either identical to y_t , or the erasure symbol ‘*’.

- (a) Starting with the test sequence from `aliceTest.txt`, generate a “noisy” text sequence by randomly erasing letters with probability $\epsilon = 0.2$. The `perturbText` method provides an easy way to do this. Print out the first 500 characters of the noisy sequence.

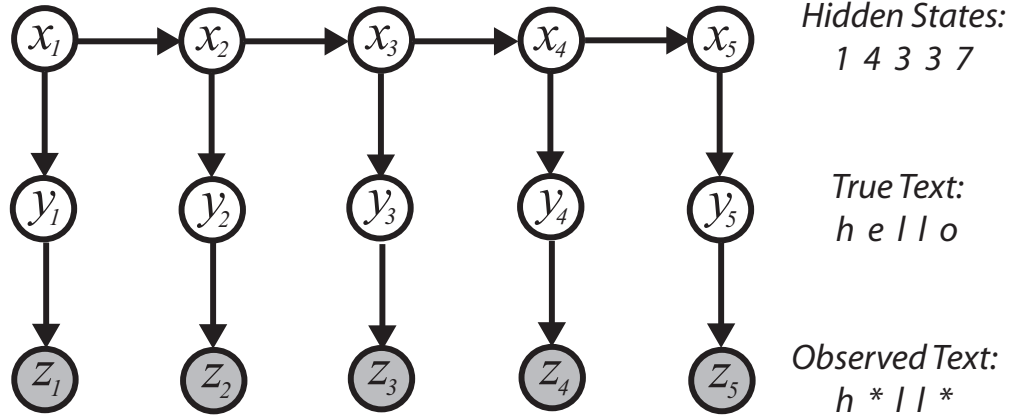


Figure 1: Graphical model illustrating an HMM with hidden states x_t which generate letters y_t . We observe a sequence z in which some of these letters have been erased.

- (b) Using the method `margprob.hmm`, compute the posterior distributions $p(x_t | z)$ for the three models from problem 1(d). To do this, exploit the fact that

$$p(z_t | x_t) = \sum_{y_t} p(z_t | y_t)p(y_t | x_t)$$

This implies that if we sum or marginalize over the possible values of the letters y_t , we recover a standard hidden Markov model in which the observations z_t are independent given the hidden state sequence x .

- (c) Suppose that we observe a letter $z_t \neq *$ at position t . Show that this implies that $y_t = z_t$ with probability one.
- (d) Suppose that we observe an erasure at position t , so that $p(z_t = * | y_t) = \epsilon$ remains *constant* as y_t is varied (since erasures provide no information about the underlying letter). Using the Markov properties of the graph in Fig. 1, and the form of the observation model, show that the posterior distribution of y_t is

$$\begin{aligned} p(y_t | z, z_t = *) &\propto p(z_t | y_t)p(y_t | z_1, \dots, z_{t-1}, z_{t+1}, \dots, z_T) \\ &\propto \sum_{x_t} p(y_t | x_t)p(x_t | z) \end{aligned}$$

where $p(x_t | z)$ is the posterior distribution of x_t given the *full* noisy sequence z .

- (e) Using the marginal distributions $p(x_t | z)$ from part (b), and the equation from part (d), determine the most likely missing letter for each erasure. Or, equivalently, implement the decision rule which minimizes the expected number of incorrect characters.
- (f) Determine the percentage of missing letters which were correctly estimated by each model. What would chance performance be for this task? Print the first 500 characters of the denoised text produced by each model, and comment on any differences.

Part II: Visualization and Nonlinear Dimensionality Reduction

Preliminaries In this part, you will use the software package [Manifold learning Matlab demo](#), accessible at <http://www.math.umn.edu/~wittman/mani>. Please download the software, which is just a single Matlab file [mani.m](#). The documentation of this software is online too at the same place where the software is downloaded.

The purpose of this assignment is just for you to experiment with several algorithms using various settings. You should include the low-dimensional embeddings computed by these algorithms in your writeup.

Note: It seems that the graphical interface that this software uses work best on a Linux machine. The sizes of many text fields might look smaller on Mac OS X.

Problem 3: Comparison linear and nonlinear projections For this problem, use “Swiss roll” from the [Examples](#) droplist. Change # Points to 1200. The data will be loaded once you click the button [Load Examples](#). Use default parameters.

- (a) **Linear methods** Run PCA and MDS. Compare the two embeddings. Is the original manifold (shown on the upper left of the window) unrolled?
- (b) **Nonlinear methods** Run LLE and ISOMAP. Compare the two embeddings. Is the manifold unrolled this time? Which one gives you the better result? Which one takes longer to finish? (On the lower left panel, the running time for different algorithms are displayed.)

Problem 4: LLE under different parameter settings Set the option [Nearest Neighbors K](#) to 4 and 16 respectively. Compare your embeddings to the embedding under default parameter $K = 8$. What has changed? Which setting gives a more faithful embedding? Now set the $K = 64$ and compute the embedding. What happens? Why this could happen? (Hint: consider the first step of LLE algorithm and the role of K there in building nearest neighbor graphs.)

Problem 5: Manifolds that have holes For this problem, use “Swiss hole” from the [Examples](#) droplist. Use 1200 data points. Change $K = 16$.

- (a) **Run LLE** Can you see the hole on the embedding?
- (b) **Run ISOMAP** Can you see the hole on the embedding? Which one do you like better, the LLE one or the ISOMAP one?
- (c) **Run Hessian LLE.** We did not cover this algorithm in class. However, Hessian LLE is provably isometrically embedding and can discover holes in manifolds. Run this algorithm. You need to change K so that it works. A good choice I found is $K = 8$. The correct embedding should be a two dimensional rectangular shape with a hole in the middle. You do not have to print out this embedding.

Note: you might need to change K accordingly. In general, LLE works better with larger K (but not too large). You should try ISOMAP with different K too. Report results that best contrast the two algorithms.

Feel free to try other examples and other algorithms and have fun!