

# CS 294 - Practical Machine Learning - Homework 4

October 31, 2006

## 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:

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

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. Plot the training log-likelihood  $\log p(y | \hat{\theta}_N)$ ,  $AIC_N$ , 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: Distinguishing Forward from Backward Text** In this problem, we use HMMs to classify time series. As discussed in previous lectures, given two models with parameters  $\theta_F$  and  $\theta_B$ , respectively, we can classify an observation sequence  $y$  by thresholding the following log-likelihood ratio:

$$\log \frac{p(y_1, \dots, y_T | \theta_F)}{p(y_1, \dots, y_T | \theta_B)} \underset{<}{\overset{\geq}{>}} \eta$$

Applying this method, we will learn HMMs for “backwards” text in which the order of characters has been reversed, and use them to determine whether or not the letters of test words have been reversed.

- (a) Construct a “backward” training sequence by reversing the order of the training text from `aliceTrain.txt`. Using this sequence, train three “backward” HMMs with state dimensions  $N$  chosen to match the “forward” HMMs from problem 1(d).
- (b) Using the optional parameter of the `readText` method, break the test chapter into individual words. For each of the three state dimensions, determine the log-likelihood of each test word with respect to the forward & backward HMMs (with parameters  $\theta_F$  and  $\theta_B$ , respectively). Also determine the log-likelihoods of a reversed-order version of each test word.
- (c) Consider an algorithm which classifies words with log-likelihood ratios above some threshold as “forward”, and below that threshold as “backward”. Use the log-likelihoods from part (b) to plot three ROC curves (one for each candidate model order).
- (d) Examine some of the “forward” words which produced classification errors (i.e., words  $y$  for which  $p(y | \theta_F) < p(y | \theta_B)$ ). Do you see any patterns?

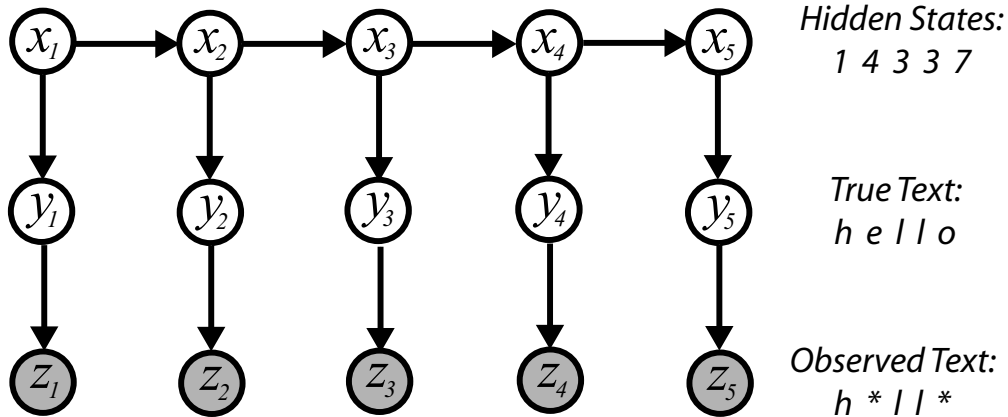


Figure 1: Graphical model illustrating an HMM with hidden states  $x_t$  which generate letters  $y_t$ . We observe a sequence  $z$  in which some of these letters have been erased.

**Problem 3: 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  $\epsilon$ , so that

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

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.
- (b) Suppose that we observe a letter  $z_t \neq *$  at position  $t$ . Show that this implies that  $y_t = z_t$  with probability one.
- (c) Suppose instead that we observe an erasure at position  $t$ , so that  $p(z_t = * | y_t)$  remains *constant* as  $y_t$  is varied. Using the Markov properties of the graph in Fig. 1, show that the posterior distribution of  $y_t$  is

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

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

- (d) 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

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

Using these distributions, determine the most likely missing letter for each erasure. Or, in other words, implement the decision rule which minimizes the expected number of incorrect characters.

- (e) 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.
- (f) *Optional:* The expression derived in part 3(c) is only correct for our special erasure model. Suppose that  $p(z_t | y_t)$  was instead some arbitrary conditional distribution. Using the Markov properties of Fig. 1 and Bayes' rule, show that

$$p(y_t | z) \propto p(z_t | y_t) \sum_{x_t} p(y_t | x_t) \cdot \frac{p(x_t | z)}{p(z_t | x_t)}$$

## Part II: Anomaly and Sequential Detection

The main purpose of this section is to study the tradeoff between the choice of threshold and quantities of interest in various anomaly detection problems.

**Problem 1: Anomaly Detection using PCA** In this part of the assignment, you are provided with the Abilene data set of network traffic (contained in the file `abilene.tar`). The data consists of four  $1008 \times 41$  matrices which record the amount of traffic in the Abilene network in 4 consecutive weeks. Each row of a matrix (in files `WeekX.dat`) records the amount of traffic that go through the 41 links of the network in 10-minute intervals during a period of one week time. In addition, there are files (`WeekXlabel.dat`) that contain the label of the network traffic at each interval (1 if normal, -1 if abnormal). As such, a data point  $X_t$  represents the traffic at the  $t$ -th interval.

We shall use the data for the first week to learn a simple model for the “normal” behavior of network traffic, and then test for anomalies on the data set of the second week, updating the model using the data of week 2, and test on week 3, and so on. The basic assumption of our approach is that normally the data lie in a low-dimensional subspace detectable by the PCA method. Thus, the abnormal traffic can be detected by looking at the deviation of each data point from that subspace. The detection procedure is described in three steps:

**Step 1 (PCA):** Let  $v_1, \dots, v_k$  be the top  $k$  eigenvector of the covariance matrix. Let  $V$  denote the matrix made of  $k$  columns  $v_1, \dots, v_k$ .  $\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_{41}$  denote the 41 eigenvalues.

**Step 2 (Computing the test score):** For each test data point  $X_t$  represented as a row vector, the distance from  $X_t$  to the subspace spanned by the top  $k$  eigenvector shall be used as a test score, which is calculated as  $S(X_t) = DD^T$ , where  $D = X_t(I - VV^T)$ .

**Step 3 (Thresholding):** Let  $\alpha$  be a desired false alarm rate. Raise an alarm whenever  $S(X_t) > Q_\alpha$ . Threshold  $Q_\alpha$  is computed as follows:

$$Q_\alpha = \phi_1 \left[ \frac{c_\alpha \sqrt{2\phi_2 h_0^2}}{\phi_1} + 1 + \frac{\phi_2 h_0 (h_0 - 1)}{\phi_1^2} \right]^{\frac{1}{h_0}}$$

where  $c_\alpha$  is the  $(1 - \alpha)$ -percentile of the standard normal distribution (i.e.,  $P(N(0, 1) > c_\alpha) = \alpha$ ) and  $h_0 = 1 - \frac{2\phi_1\phi_3}{3\phi_2^2}$ ,  $\phi_p = \sum_{j=k+1}^{41} \lambda_j^p$  for  $p = 1, 2, 3$ .

- Combine your PCA code with a couple of lines of code for step 2 and 3. Identify the number  $k$  of principal components that the network link traffic data lie on. Explain your choice.
- For different choices of the false alarm rate  $\alpha$ , compute the threshold  $Q_\alpha$  accordingly, and test the detection procedure on the test set. Plot the false alarm rate and compare the actually measured false alarm with the desired  $\alpha$ . Also plot the ROC curve.
- The choice of threshold above is very specific to the statistic derived from the PCA method. One can also use the following choice of threshold, which is more generally applicable: Based on the training data set, estimate the mean  $\mu$  and standard deviation  $\sigma$  of the statistic  $S(X)$ . Given a desired false alarm rate  $\alpha$ , raise an alarm whenever  $S(X) > \mu + \sigma c_\alpha$ . Repeat the same set of experiment as in part (b). Compare this to the previous choice of threshold.
- Periodically updating the model for normal behavior can help reduce both the false alarm and misdetection rate, as the data evolve over time. Suppose that you choose not to perform update, i.e., you train the (PCA) model using the data of week 1 and test on the data set of the remaining 3 weeks. Compare the ROC curve obtained by this non-updating approach to the updating one.

**Problem 2: Sequential Hypothesis Testing** This part of the assignment guides you through the proof of a key result about the relationship between the threshold and false alarm misdetection error rate (see also the lecture slides). The full proof is fairly simple and takes only a few lines, but it might be surprising to those who have not thought in terms of random stopping times before.

Suppose that we are given a sequence of data  $X_1, X_2, \dots$ , which is to be tested against two competing hypotheses:  $H_0$ : all  $X_i$  are i.i.d. by a distribution  $P_0$  with density  $f_0$ ; and  $H_1$ : all  $X_i$  are i.i.d. by a distribution  $P_1$  with density  $f_1$ . The sequential test is based on the log likelihood ratio:

$$L_n(X) = \sum_{i=1}^n \log \frac{f_1(X_i)}{f_0(X_i)}.$$

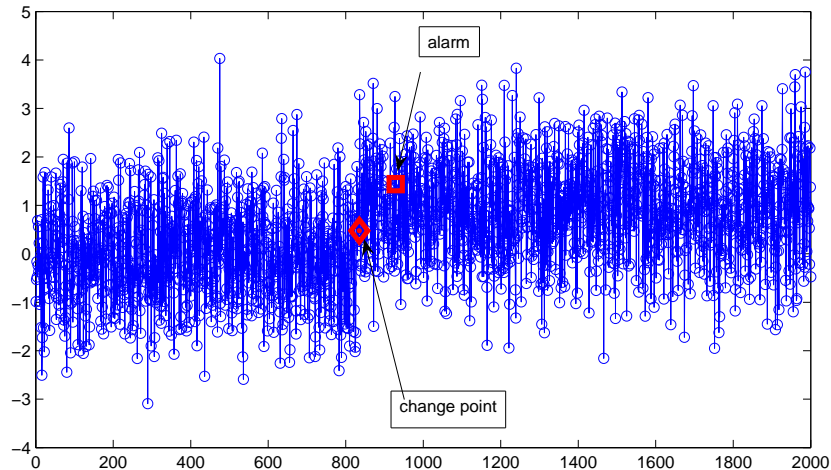


Figure 2: Illustration of the CUSUM test.

The test is as follows: Stop at the first  $n$  such that  $L_n(X) \geq b$  or  $L_n(X) \leq a$ , for some thresholds  $a < b$ . If  $L_n(X) \geq b$ , accept hypothesis  $H_1$ , otherwise accept  $H_0$ . The false alarm and misdetection rate are defined as  $\alpha = P_0(\text{accept } H_1)$  and  $\beta = P_1(\text{accept } H_0)$ , respectively.

- Assume first that there is no “overshooting” over the boundary thresholds, i.e., at the stopping time  $n$ , either  $L_n(X) = a$  or  $L_n(X) = b$ . Show that  $a = \log \frac{\beta}{1-\alpha}$  and  $b = \log \frac{1-\beta}{\alpha}$ .
- In general, the likelihood ratio statistic can overshoot. Show that  $a \geq \log \frac{\beta}{1-\alpha}$  and  $b \leq \log \frac{1-\beta}{\alpha}$ .
- Derive bounds on  $\alpha$  and  $\beta$  given thresholds  $a$  and  $b$ .

**Problem 3: Sequential Change-point Detection** This part of the assignment helps you to get familiar with the the CUSUM method for the sequential change-point detection problem (see the lecture slides for details of the procedure).

- Generate a time series of 2000 points made of an i.i.d. sequence of  $N(0,1)$  random variables followed by an i.i.d. sequence of  $N(1,1)$  random variables. The location of the change point is uniform in the interval  $[501,1500]$ .
- Apply the CUSUM test using various choices of threshold. Plot the *average delay time of detection* and the *average time between a false alarm and the true change-point* for different choices of threshold. Comment on the tradeoff. (You will have to repeat the the experiment 1000 times to collect such statistics).

Figure 2 illustrates an instance of this experiment using threshold  $b = 50$ .

## Part III: Reinforcement Learning

You will experiment with the following Reinforcement Learning demos:

*Black Jack*: <http://lslwww.epfl.ch/~anperez/BlackJack/classes/RLJavaBJ.html>

*Cat and mouse*: <http://www.cse.unsw.edu.au/~cs9417ml/RL1/applet.html>

*Robot*: <http://www.applied-mathematics.net/qlearning/qlearning.html>

*Pendulum*: <http://www.sics.se/~joakime/rlpage.php?applet=1>

- (a) Define the state-space, action-space, and rewards for Black Jack, Cat and Mouse, and Robot. How would you represent the Q function to speed up training?
- (b) Black Jack, Cat and Mouse, and Robot applets use a TD algorithm (SARSA or Q-learning). Is it possible to implement a Dynamic Programming algorithm for these tasks? What would be the pros and cons of implementing DP or a Monte Carlo algorithm?
- (c) In Cat and Mouse you can try both SARSA and Q-learning; which one works better and why?
- (d) Black Jack, Cat and Mouse, and Robot applets all have the following three parameters: i) alpha: learning rate, ii) gamma: discount factor, and iii) epsilon: greediness of action selection. Experiment with various values of these parameters; try the extreme values as well. What effect do these parameters have on performance of the algorithm? What are the optimal values? Is it beneficial to change their values during training?
- (e) The goal of Pendulum is to bring the pendulum up to vertical position and keep it there; what reward function would you use to achieve it? How would you use shaping? Discuss.
- (f) Which RL algorithm would work best for Pendulum?
- (g) There are actually 2 pendulum applets on the web site; one represents the Q function as a table, the other uses tile-coding. Compare the speed of training of both representations. How would you change the representation of the state-space and Q to improve the performance and speed of training?