# CS 294 - Homework 3

October 15, 2009

If you have questions, contact Alexandre Bouchard (bouchard@cs.berkeley.edu) for part 1 and Alex Simma (asimma@EECS.Berkeley.EDU) for part 2. Also check the class website for updates and due date.

## Part 1: Feature selection

This question is based around the R code in featsel.R; you shouldn't execute this script directly, but you can copy and paste the code from it into an R session (or use it as a starting point to write your own R scripts).

The idea behind this exercise is to explore how filtering performs versus Lasso regression and forward selection.

(a) Generate a dataset of size 100 with 10 features, following the instructions in featsel.R. Create a response variable that depends linearly on just two of the covariates, with some added noise (follow the guidelines in featsel.R closely). Perform filtering, using as a score the squared Pearson correlation coefficient. The Pearson correlation coefficient is a popular measure of how related two random variables are. It can be computed as

$$\mathrm{Corr}(X, Y) = \frac{\mathrm{Cov}(X, Y)}{\sqrt{\mathrm{Var}(X)}\sqrt{\mathrm{Var}(Y)}}.$$

From a finite sample of points $(X_i, Y_i)$, this is estimated as

$$\frac{\sum_i (X_i - \bar{X})(Y_i - \bar{Y})}{\sqrt{\sum_i (X - \bar{X})^2}\sqrt{\sum_i (Y - \bar{Y})^2}}.$$

It lies between $-1$ and 1. It is equal to 1 if and only if the variables are perfectly correlated, and is equal to $-1$ if and only if they are perfectly anticorrelated.

Generate random datasets at least five different times, running filtering each time. Suppose we want a model with two variables; we will discard all but the top two scoring features. Does this filtering score consistently rank the top two relevant features as the best? If not, why not? (see featsel.R for a hint)

(b) On each of the datasets you used for filtering in part (a), fit a sparse linear model using LARS. Print out the plot.lars output (just one graph will do, for one of the five runs). This output traces the weights for each feature as the regularization coefficient decreases. As it decreases, do the top two relevant features consistently receive nonzero weight before any others?

(c) Make $y$ a different linear function of several of the covariates–your choice. Pick a new dataset size–perhaps increase the ratio of features to examples. You can alter the variance of the noise term if you like by multiplying it by a constant. Redo parts (a) and (b).

(d) Generate a dataset of size 1000 with 1000 features; this dataset is a little large for LARS. Perform filtering using the squared Pearson correlation. Suppose we want a model with just two variables. Does filtering on its own consistently give us the best model? Now try filtering out all but 200 features and then using LARS. Does this combination consistently pick out the correct model? Does it run quickly?

(e) Now we'll take our dataset into YALE to try forward selection. First, output one of your generated datasets from part (a) to a file, following the instructions in featsel.R. Pick a dataset for which filtering failed to rank the relevant features highest. Name the file featsel.dat and put it in the same directory as featsel.aml and featsel.xml. Now open the experiment "featsel.xml" in YALE. Look through the operator chain; it performs forward selection using cross validation with squared error as an ob jective function, with linear regression as the internal learning module. Edit the "attributes" key in the "ExampleSource" operator to point to featsel.aml, edit the "filename" key in the "ExperimentLog" operator to specify an output file, and run the experiment. See which features were selected by clicking the ExampleSource tab in the results section. What were they?

(f) Optional: Try feature selection algorithms on a problem from your own research (or from the UCI Machine Learning Repository). For instance, compare LARS regression against $L_2$ regularized and unregularized regression. What are the number of non-zero features selected? Is there a gain of accuracy? Is regression (or classification) faster at test time?

# Part 2: Hidden Markov Models

**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 $y = (y_1, \ldots, y_T)$ denote the observed training sequence, $x = (x_1, \ldots, x_T)$ a hidden state sequence, and $\hat{\theta}_N$ an ML estimate of the parameters for an HMM with $N$ states:

$$\hat{\theta}_N = \arg\max_{\theta_N} p(y \mid \theta_N) = \arg\max_{\theta_N} \sum_x p(y \mid x, \theta_N) p(x \mid \theta_N) \qquad x_t \in \{1, \ldots, N\}$$

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

$$\mathrm{AIC}_N = \log p(y \mid \hat{\theta}_N) - d(N)$$
$$\mathrm{BIC}_N = \log p(y \mid \hat{\theta}_N) - \frac{1}{2} d(N) \log(T)$$

Here, d(N) is the *number* of parameters for an HMM with $N$ states. The "best" model is then the one for which $\text{AIC}_N$ or $\text{BIC}_N$ is largest.

(b) Without using the penalization, would you expect a model with more parameters to have a higher or lower log-likelihood than one with fewer parameters? Why? Why would we want to use the AIC or BIC, instead of just the log-likelihood to select our model?

(c) 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 $\log p(y \mid \hat{\theta}_N)$, $\text{AIC}_N$, and $\text{BIC}_N$ versus $N$ for the HMMs learned in part (a). Which criterion favors simpler models?

(d) 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 $\text{BIC}_N$, and the model with the highest $\text{AIC}_N$. 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 $x_t \in \{1, \ldots, N\}$ 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

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

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.

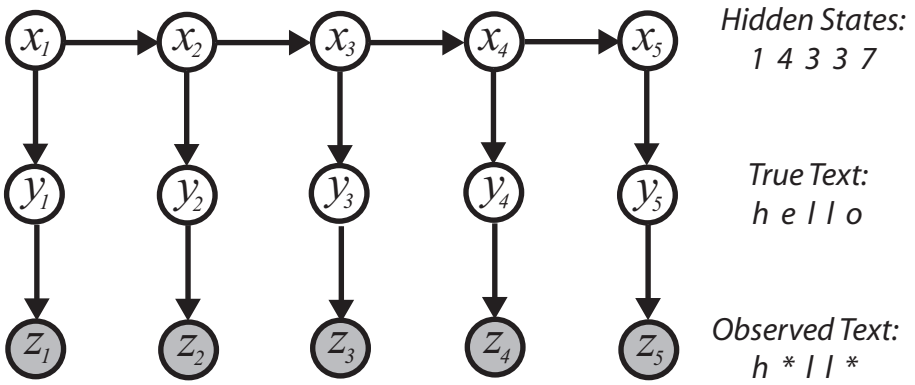


Figure 1: Graphical model illustrating an HMM with hidden states $x_t$ which generate letters $y_t$. We observe a sequence $z$ in which some of these letters have been erased.

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

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

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 $p(z_t = * \mid y_t) = \epsilon$ 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

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

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

(e) Using the marginal distributions $p(x_t \mid z)$ 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.

(g) A friend tells you that the next $n+1$st letter depends not only on the $n$th letter, but also on the $n-1$st one. So, your friend suggests that you should be fitting a second-order Hidden Markov Model, where the probability of transitioning to a hidden state depends on the last two states. Is your friend correct? Will the first order HMM not capture those dependencies? What would you expect the second-order HMM to do? (This is a qualitative question; you do not need to implement the second-order HMM.)