6.867 Machine Learning

Problem Set 5

Due date: Friday November 19, 2004

Please address all questions and comments about this problem set to `6867-staff@csail.mit.edu`. You will need to use MATLAB for some of the problems but most of the code is provided. If you are not familiar with MATLAB, please consult

`http://www.ai.mit.edu/courses/6.867/matlab.html`

and the links therein.

Problem 1: Model selection

In this problem we illustrate Bayesian model selection on probabilistic suffix trees (PST's). Recall from Problem Set 4 that a PST is a classifier represented by a rooted tree, in which each edge tests for a specific value of a feature, and each node contains a probability distribution over possible class labels. To classify a datapoint represented by a sequence of features, $\mathbf{x} = (x^1, x^2, x^3, \dots)$, we follow from the root the longest path of matching consecutive features, and assign to $\mathbf{x}$ a class according to the distribution of the last node reached.

Our task is to estimate a PST for a binary classification task ($y \in \{0, 1\}$) where each feature can take values in $\{0, 1, 2\}$ from a set of training points $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_n, y_n)$. This is a model selection problem because apart from estimating the probability distributions at each node, we need to decide the optimal structure of the tree supported by the data.

Consider a generic PST with K nodes, and let $p_1, p_2, \dots, p_K$ be the probability of $y = 1$ associated with each node. The structure of the tree and edge feature values represent the model, while the probabilities $\boldsymbol{\theta} = (p_1, p_2, \dots, p_K) \in [0, 1]^K$ constitute its parameters.

One way to compare models is via the Bayesian score (or marginal likelihood). We place a prior distribution over the model parameters, in this case the "uninformative" prior $P(\boldsymbol{\theta}) = 1$ for all $\boldsymbol{\theta} \in [0, 1]^K$, and integrate out the parameters:

$$P(y_1, \dots, y_n | \mathbf{x}_1, \dots, \mathbf{x}_n, \text{PST}) = \int_{[0,1]^K} \left[\prod_{i=1}^{n} P(y_i | \mathbf{x}_i, \boldsymbol{\theta}) \right] P(\boldsymbol{\theta}) d\boldsymbol{\theta} \tag{1}$$

The Bayesian score does not depend on any specific setting of the parameters since they are integrated out, but it does depend on the prior. The prior has to divide one unit of probability mass over the parameter space. Ideally, to maximize the score, a good fraction of this probability mass is allocated to parameter values that maximize the data likelihood or $\prod_{i=1}^{n} P(y_i|\mathbf{x}_i, \boldsymbol{\theta})$. This is harder to achieve for complex models since the unit prior probability mass has to be divided over a much larger parameter space. Put another way, while finding parameter values that lead to high likelihood of the data becomes easier with more complex models, the prior probability mass allocated to those values may become vanishingly small. This highlights the complexity penalty built into the Bayesian score.

It is not always possible to evaluate the Bayesian score analytically but in our case it is. We need a bit of notation to get started:

- $n_1, n_2, \ldots, n_K$ are the number of data points from the training set classified using each of the K nodes;
- n_i^+ is the number of positive ($y = 1$) training samples ending up at node i;
- n_i^- is the number of negative samples at node i ($y = 0$).

Note that $n = \sum_{i=1}^{K} n_i$ and $n_i = n_i^+ + n_i^-$.
With this notation the likelihood of the training data can be written as:

$$\prod_{i=1}^{n} P(y_i|\mathbf{x}_i, \boldsymbol{\theta}) = \prod_{i=1}^{K} p_i^{n_i^+} (1 - p_i)^{n_i^-} \tag{2}$$

and the parameters $\hat{\boldsymbol{\theta}} = (\hat{p}_1, \hat{p}_2, \ldots, \hat{p}_K)$ that maximize the likelihood are

$$\hat{p}_i = \frac{n_i^+}{n_i} \tag{3}$$

The resulting maximum value of the log-likelihood is equal to n times the conditional mutual information score we used in Problem Set 4:

$$H(Y|X) = \sum_{i=1}^{K} \frac{n_i}{n} \left(\hat{p}_i \log \frac{1}{\hat{p}_i} + (1 - \hat{p}_i) \log \frac{1}{1 - \hat{p}_i} \right) \tag{4}$$

1. (*10 points*) Compute the Bayesian model selection score in terms of n_i, n_i^+ and n_i^- by performing the integration in (1). You may use the following integral:

$$\int_0^1 p^k (1 - p)^l dp = \frac{k!\, l!}{(k + l + 1)!}$$

The BIC score (Bayesian Information Criterion), is a large-sample approximation to the Bayesian Model selection score that is much easier to compute in practice. It takes the form of a log-likelihood penalized by $\frac{d}{2}\log n$, where d is the *effective* number of parameters:

$$\sum_{i=1}^{n} \log P(y_i|\mathbf{x}_i, \hat{\boldsymbol{\theta}}) - \frac{d}{2}\log n \tag{5}$$

In our PST d is the number of *incomplete* nodes (nodes that do not have all features $0, 1, 2$ as children). This is because if a node is fully expanded no datapoint will be assigned to it, and its probability distribution will never be used in classification.

2. (*5 points*) Express the BIC score as a function of the counts n_i^+, n_i^-, n_i, n.

We have provided you with partial MATLAB code to test the Bayesian and BIC scores in practice. The data represent the same vowel prediction task as in Problem Set 4. You can load the data with '`load hw4_p1.mat`', giving you `data`.

3. (*10 points*) The Matlab functions `log_bayesian_model_score.m` and `bic_model_score.m` need to be completed to compute the **logarithm** of the Bayesian score as derived in Part 2., and the BIC score you derived in Part 4, respectively. The functions receive as parameters the vectors of n_i^+'s and n_i^-'s for the incomplete nodes only. To test the completed functions train and plot the following two PST's:

```
plot_pst(build_pst(data, @log_bayesian_model_score))
plot_pst(build_pst(data, @bic_model_score))
```

   (For your reference, the PST's are trained by greedily extending them from the root edge by edge until no edge can be added without decreasing the model score.)

   Hand in the printouts of `log_bayesian_model_score.m` and `bic_model_score.m` and of the two plots. Note the the BIC tree is a subset of the Bayesian tree.

The command `generate_plot.m` compares the PST's in Figure 1 by their log-Bayesian model score and BIC model score. Specifically, for each of the two scores we compute the difference between the score of the more complex and the score of the simpler model. We would select the more complex model if and only if this difference is positive.

For a given number of samples in the range $[0, 3000]$, `generate_plot.m` computes the average difference produced by each of the two scores on 500 random selections of the sample. It then plots the average difference divided by the number of samples (because both the Bayesian score and the BIC score increase asymptotically linearly with the number of samples).

4. (*10 points*) Run `generate_plot` and hand in the printout of the produced figure (it may take a couple of minutes to run). Briefly explain the following observations in terms of their significance to model selection:

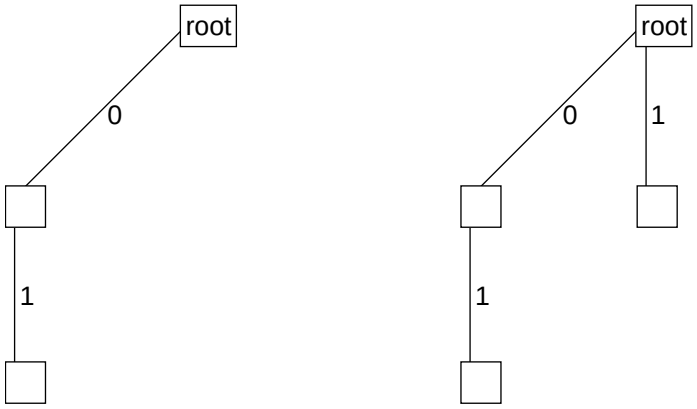

Figure 1: Models compared in Problem 1 Part 4

- each curve is negative for small sample size
- the curve generated by the Bayesian score becomes positive for smaller sample size than the curve generated by the BIC score
- at large sample size the two curves converge to each other
- at large sample size the curves become horizontal

Problem 2: EM algorithm

In this problem, we will study the use of the Expectation Maximisation (EM) algorithm for solving estimation problem involving hidden variables.

A mixture of m Gaussians model, for example, is a simple hidden or latent variable model

$$p(\mathbf{x};\theta) \;=\; \sum_{j=1}^{m} p_j N(\mathbf{x};\mu_j,\mathbf{\Sigma}_j) \tag{6}$$

The distinction between hidden and observed variables depends on the data we expect to see in estimating these models. In a typical setting, we observe only $\mathbf{x}$ samples and the corresponding choices of the mixture components remain hidden. Thus $\mathbf{x}$ is an observed (vector valued) variable, while j is hidden.

A direct optimization of latent variable models is often difficult since the likelihood of the data involves summations over the values of the hidden variables. Specifically, in the log-likelihood of the data for a mixture of m Gaussians

$$l(\mathbf{x}_1,\ldots,\mathbf{x}_n;\theta) \;=\; \log\prod_{i=1}^{n} p(\mathbf{x}_i;\theta) = \sum_{i=1}^{n}\log\sum_{j=1}^{m} p_j N(\mathbf{x}_i;\mu_j,\mathbf{\Sigma}_j) \tag{7}$$

the summation over the mixture components appear inside the logarithm, coupling the means and covariances of the Gaussian components.

Suppose for a moment that we also observed the selections of the mixture components, $j_1, \ldots, j_n$, in addition to $\mathbf{x}_1, \ldots, \mathbf{x}_n$. In this case we could write down the complete log-likelihood of the data

$$l_c(\mathbf{x}_1, \ldots, \mathbf{x}_n, j_1, \ldots, j_n; \theta) = \sum_{i=1}^{n} \log \left(p_{j_i} N(\mathbf{x}_i; \mu_{j_i}, \mathbf{\Sigma}_{j_i}) \right) \tag{8}$$

where each sample $(\mathbf{x}_i, j_i)$ contains a value assignment to all the variables in the model. The means and covariances of the Gaussians, along with the mixing proportions, could now be estimated independently of each other.

Given only $\mathbf{x}_1, \ldots, \mathbf{x}_n$, we can nevertheless use the current setting of the model parameters, θ^k, to infer what the mixture selections would have been. In other words, we can evaluate the expected complete log-likelihood:

$$\begin{aligned} El_c(\theta; \theta^k) &= E\left\{ l_c(\mathbf{x}_1, \ldots, \mathbf{x}_n, j_1, \ldots, j_n; \theta) \,\middle|\, \mathbf{x}_1, \ldots, \mathbf{x}_n, \theta^k \right\} & (9) \\ &= \sum_{i=1}^{n} E\left\{ \log \left(p_{j_i} N(\mathbf{x}_i; \mu_{j_i}, \mathbf{\Sigma}_{j_i}) \right) \,\middle|\, \mathbf{x}_i, \theta^k \right\} & (10) \\ &= \sum_{i=1}^{n} \sum_{j=1}^{m} P(j|\mathbf{x}_i, \theta^k) \log \left(p_j N(\mathbf{x}_i; \mu_j, \mathbf{\Sigma}_j) \right) & (11) \end{aligned}$$

where the expectations are over $j_1, \ldots, j_n$ given θ^k and $\mathbf{x}_1, \ldots, \mathbf{x}_n$. $El_c(\theta; \theta^k)$ should be viewed as a function of possible new settings of parameters

$$\theta = \{p_1, \ldots, p_m, \mu_1, \ldots, \mu_m, \Sigma_1, \ldots, \Sigma_m\} \tag{12}$$

for a fixed θ^k and $\mathbf{x}_1, \ldots, \mathbf{x}_n$ (which we have suppressed in the notation). Evaluating the expected complete log-likelihood for a hidden variable model as a function of possible new parameters θ defines the E-step of the EM algorithm.

The M-step of the EM algorithm corresponds simply to finding θ^{k+1} that maximize the expected complete log-likelihood:

$$\theta^{k+1} = \arg\max_{\theta} \; El_c(\theta; \theta^k) \tag{13}$$

This estimation step is again easy, as if we had complete (but weighted) data.

More generally, we don't actually need to solve for the parameters θ^{k+1} that exactly maximize the expected complete log-likelihood in the M-step. To guarantee that the log-likelihood of the observed data increases after every M-step, it suffices to find any θ^{k+1} for which

$$El_c(\theta^{k+1}; \theta^k) > El_c(\theta^k; \theta^k) \tag{14}$$

When we settle for a suboptimal θ^{k+1}, with the above constraint, the M-step is known as a generalized M-step.

Let us now turn to a slightly more complicated hidden variable density model. Specifically, we define a one dimensional density $p(x;\theta)$, $x \in \Re$, in terms of two hidden variables

$$p(x;\theta) = \sum_{j=1}^{m} \sum_{k=1}^{l} p_j q_k N(x;\mu_j,\sigma_k^2) \tag{15}$$

where $\theta = \{p_1,\ldots,p_m,\mu_1,\ldots,\mu_m,q_1,\ldots,q_l,\sigma_1^2,\ldots,\sigma_l^2\}$. We could view this as a simple mixture model with ml Gaussian components indexed by (j,k). However, unlike before, the parameters of the ml components cannot be set independently. For example, there are only m possible means, not ml. Alternatively, we could view this as a mixture of m non-Gaussian components, where each component distribution is a scale mixture, $p(x|j;\theta) = \sum_{k=1}^{l} q_k N(x;\mu_j,\sigma_k^2)$, combining Gaussians with different variances (scales). These m components are again not parameterized independently of each other.

As a hidden variable model, $p(x;\theta)$ in eq. (15) can be estimated from samples $x_1,\ldots,x_n$ via the EM algorithm. For this purpose, we will have to evaluate the expected complete log-likelihood (E-step) and solve for the parameters in the (generalized) M-step.

1. (*5 points*) Provide the expression for the posterior probability over the hidden variables given x_i and θ^k (current setting of the model parameters).

2. (*5 points*) Write down the portion of the expected complete log-likelihood that pertains to the new mean parameters $\mu_1,\ldots,\mu_m$ we intend to find in the M-step. The dependence on these means should be made explicit.

3. (*5 points*) Solving the M-step for the maximizing parameters θ^{k+1} would require us to jointly optimize the means $\mu_1,\ldots,\mu_m$ and the variances $\sigma_1^2,\ldots,\sigma_l^2$ in this case. We can, however, easily specify a simpler generalized M-step: solve for μ_j's given fixed σ_k^2's, and subsequently solve for σ_k^2's given the new values of μ_j's. Provide an expression for the maximizing μ_j's given fixed $\sigma_1^2,\ldots,\sigma_l^2$.

4. (*10 points*) We have provided you with MATLAB code to test the new latent variable model. Load `p2data.mat` into MATLAB to get $n = 300$ samples x drawn from distribution `trueparam` (also provided). Estimate the parameters θ of the latent variable model via the EM-algorithm, `param = em(x,m,l)`, using $m = 1,\ldots,4$, and $l = 1,2,3$. Plot the samples, estimated density, and the data generating distribution using
   ```
   plotdensity(x,param,'b'); hold on;
   plotdensity(x,trueparam,'r'); hold off;
   ```
   You may have to run the EM-algorithm for a few times to get the best results for each combination (m,l) as the initial parameters are selected at random. Provide a brief explanation why there might be a difference in the approximation quality between setting $l = 1$ (single variance parameter) and $l > 1$, even if m can take different values.

Problem 3: Spectral Clustering

In this problem, we experiment with two types of clustering methods—k-means and spectral clustering—and compare their performance on two different data sets. We have provided you with Matlab code to perform k-means clustering (`kmeans.m`) and spectral clustering (`spectral.m`), as well as two 2-dimensional data sets `X1.mat` and `X2.mat`. The function `plotclust(X,y)` will help you visualise the clusters.

1. (*10 points*) Run $k-$means clustering on both data sets, with $k = 2$. Since the initialisation of the 2 means is random, different runs of the algorithm may produce different results. Turn in a typical plot of the clusters obtained. Compare the results in each case with what you expect to be the *correct* answer (when searching for 2 clusters).

2. (*10 points*) Perform spectral clustering on both data sets, with $r = 5$ and $r = 20$ (where r is the number of nearest neighbours considered when building the graph). Turn in plots of the clusters in these cases. Compare the results in each case with what you expect to be the *correct* answer (when searching for 2 clusters), and with the result obtained from $k-$means clustering. Explain the behaviour of spectral clustering in each case.