6.867 Machine Learning

Problem set 5

Due Thursday, November 14, in class

What and how to turn in?

Turn in short written answers to the questions explicitly stated, and when requested to explain or prove. Do **not** turn in answers when requested to "think", "consider", "try" or "experiment" (except when specifically instructed). You may turn in answers to questions marked "optional"— they will be read and corrected, but a grade will not be recorded for them.

Turn in all MATLAB code explicitly requested, or that you used to calculate requested values. It should be clear exactly what command was used to get the answer to each question.

To help the graders (including yourself...), please be neat, answer the questions briefly, and in the order they are stated. Staple each "Problem" separately, and be sure to write your name on the top of every page.

Problem 1: model selection

Minimum description length principle is a bit unwieldy to follow in practice. This is because we would have to find discretized estimates for the parameters, find the right resolution at which to discretize them and so on. A simple alternative is the following asymptotic approximation:

$$\mathrm{DL} \approx -\left(\log p(\mathrm{data}|\hat{\theta}) - \frac{d}{2}\log(n) \right) \tag{1}$$

where d is the dimension of the parameter space (the number of parameters) and $\hat{\theta}$ is the maximum likelihood parameter estimate. This approximation is valid for large n. We have taken the minus sign outside the terms so as to deal with log-probabilities rather than bits (negative log-probabilities). The criterion inside the brackets is also known as the *Bayesian Information Criterion* or BIC for short. We will use the BIC criterion here and select the

model – a family of distributions – with the maximum value of

$$\log p(\text{data}|\hat{\theta}) - \frac{d}{2}\log(n) \tag{2}$$

Since this is justified as an asymptotic criterion, it is useful to see whether it is at all appropriate when n is not very large. We'll try to verify this in the context of simple Gaussians.

1. [**10pt**] Let's start by deriving the log-likelihood of data for a Gaussian with maximum likelihood parameters. By ignoring constant terms – terms not depending on the mean or the covariance – we get

$$\begin{aligned}
\log p(\text{data}|\theta) &= \sum_{i=1}^{n} \log p(\mathbf{x}_i|\mu, \Sigma) && (3)\\
&= \sum_{i=1}^{n} \left(-\frac{1}{2}(\mathbf{x}_i - \mu)^T \Sigma^{-1} (\mathbf{x}_i - \mu) - \frac{1}{2}\log(|\Sigma|) \right) && (4)\\
&= -\frac{1}{2}\sum_{i=1}^{n} (\mathbf{x}_i - \mu)^T \Sigma^{-1} (\mathbf{x}_i - \mu) - n \cdot \frac{1}{2}\log(|\Sigma|) && (5)\\
&= -\frac{1}{2} n \cdot \text{Trace}\left(\frac{1}{n}\sum_{i=1}^{n} (\mathbf{x}_i - \mu)^T \Sigma^{-1} (\mathbf{x}_i - \mu) \right) - n \cdot \frac{1}{2}\log(|\Sigma|) && (6)
\end{aligned}$$

   where the trace of a square matrix A is the sum of its diagonal components. In other words, $Trace(A) = \sum_{j=1}^{d} A_{jj}$. The nice property about trace is that $Trace(AB) = Trace(BA)$ whenever BA makes sense as a matrix product. We were able to introduce trace above since $(\mathbf{x}_i - \mu)^T \Sigma^{-1} (\mathbf{x}_i - \mu)$ is a scalar (or 1 by 1 matrix). By inserting maximum likelihood parameter estimates for the mean and the covariance, simplify the above expression to make it only a function of n, the dimension d of $\mathbf{x}$, and the log-determinant of the covariance matrix. The result is $\log p(\text{data}|\hat{\theta})$ that we need for BIC.

2. [**10pt**] Your result holds even if we restrict the covariance matrix to be diagonal. For a Gaussian, this restriction means that we assume that the components of $\mathbf{x}$ are independent. The maximum likelihood estimate of the diagonal covariance matrix will be diagonal, of course. Write down the expressions for the BIC scores for the two models: model 1 with full covariance and model 0 with diagonal covariance.

   Let's explore a little how well the criterion works with small sample sizes. Specifically, we wish to test whether the complexity penalty is appropriate. We will generate data from a Gaussian that conforms to the simpler diagonal covariance and see whether the criterion allows us to select the simpler alternative. The more complex model

(full covariance) surely provides a better fit in terms of the data likelihood. The complexity penalty then has to be large enough to overcome this bias.

Even if we test the criterion only one way, there are still two possible variations we can perform in evaluating the criterion. One is the sample size, the other is the dimensionality of the examples. We assume that the true data is generated from a Gaussian with zero mean and identity covariance (the exact values of the diagonal entries of the true covariance matrix are secondary).

We have provided you with a MATLAB routine `n_min = bictest(d)` which takes the dimension d as an argument and returns a measure of how many examples we need before the BIC criterion consistently picks the right model. Since BIC is asymptotically correct this will happen eventually. The routine generates a large training set at random and incorporates examples from this set incrementally in the evaluation of the BIC scores. As a result we get a sequence of BIC scores which are evaluated on the basis of successively larger subsets of the training set. We then find the number of examples (for this choice of the training set) that are needed before the BIC scores remain consistent with the correct model for the remaining examples in the large set. This measure is a random variable since it is based on the randomly chosen training set. You need to call this routine repeatedly to get a better idea of the number of examples that are required for BIC to work properly.

Before using the code, however, you will have to insert the formulae for the BIC scores corresponding to the two Gaussian models (this piece of the code is missing). Recall that model 0 assumes a diagonal covariance matrix while model 1 permits a full covariance matrix.

3. [**5pt**] Plot samples of the number of points (that are needed for BIC to work properly) as a function of the dimension d. Use $d = 2, \ldots, 10$.

4. [**10pt**] Briefly explain the why the scatter plot looks the way it does.

Problem 2: monotonicity of the EM algorithm

We try to understand here why the EM algorithm works. The crucial property of the algorithm is that it monotonically increases the log-likelihood of the data. This is a very convenient property in practice. For example, if you happen to have a bug in the code it is unlikely that the log-likelihood would be monotonic until convergence. Monotonicity of the log-likelihood therefore serves as a good test of whether the code is correct. In addition, when solving large problems in practice, where even a single EM iteration may take a long time (hours), it is quite reasonable that the time is spent on actually improving the solution. It might be a bit frustrating if an iteration involving hours of computation would occasionally decrease the log-likelihood.

Suppose now that we have observed $D = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ and wish to estimate an m-component Gaussian mixture model:

$$p(\mathbf{x}|\theta) = \sum_{j=1}^{m} p_j p(\mathbf{x}|\mu_j, \Sigma_j) \tag{7}$$

where the parameters θ refer to all the parameters in the mixture model

$$\theta = \{p_1, \ldots, p_m, \mu_1, \ldots, \mu_m, \Sigma_1, \ldots, \Sigma_m\} \tag{8}$$

and

$$p(\mathbf{x}|\mu_j, \Sigma_j) = \frac{1}{(2\pi)^{d/2}|\Sigma_j|^{1/2}} \exp\{-\frac{1}{2}(\mathbf{x} - \mu_j)^T \Sigma_j^{-1} (\mathbf{x} - \mu_j)\} \tag{9}$$

where d is the dimensionality of the examples $\mathbf{x}$; its value plays no role in establishing the monotonicity property.

1. [**5pt**] Let's start by examining what the gradient of the log-likelihood looks like. The log-likelihood function is given by

$$l(\theta; D) = \sum_{i=1}^{n} \log p(\mathbf{x}_i|\theta) \tag{10}$$

Take the derivative of the log-likelihood with respect to the mean μ_k (the kth Gaussian) and set the derivative to zero. Simplify the result so that it reads like an update equation: $\mu_k = \ldots$. Express the right hand side of this equation in terms of the posterior weight of the kth component

$$\hat{n}_k = \sum_{i=1}^{n} P(k|\mathbf{x}_i, \theta) \tag{11}$$

and the corresponding weighted posterior mean of the examples

$$\frac{1}{\hat{n}_k} \sum_{i=1}^{n} P(k|\mathbf{x}_i, \theta)\mathbf{x}_i \tag{12}$$

Sure looks like an EM step.

2. [**5pt**] Let's try to interpret the previous update as a Gradient ascent. In other words, we need to find the step size ϵ so that the update

$$\mu_k \leftarrow \mu_k + \epsilon \frac{\partial}{\partial \mu_k} l(\theta; D) \tag{13}$$

gives exactly the update you just derived above (the right hand sides should match). What is the step size ϵ?

So, we have expressed the EM update as a gradient ascent learning rule with a rather large step size that changes from one iteration to another. But how could this possibly lead to a monotonically increasing log-likelihood? We need to explore this a bit further.

We will rely on the following result:

$$\sum_{j=1}^{m} q(j|i) \log \left(\frac{p_j p(\mathbf{x}_i|\mu_j, \Sigma_j)}{q(j|i)} \right) \leq \log \left(\sum_{j=1}^{m} p_j p(\mathbf{x}_i|\mu_j, \Sigma_j) \right) = \log p(\mathbf{x}_i|\theta) \tag{14}$$

which holds for any choice of $q(j|i) \geq 0$ provided that $\sum_{j=1}^{m} q(j|i) = 1$ for each i.

3. [**5pt**] Show that the inequality becomes an equality if we set $q(j|i)$ equal to the posterior probability: $q(j|i) = P(j|\mathbf{x}_i, \theta)$. This also turns out to be the only choice of $q(j|i)$ that attains the equality.

4. [**5pt**] Now let

$$\tilde{J}(Q;\theta) \quad = \quad \sum_{i=1}^{n} \sum_{j=1}^{m} q(j|i) \log \left(\frac{p_j p(\mathbf{x}_i|\mu_j, \Sigma_j)}{q(j|i)} \right) \tag{15}$$

where for simplicity we use Q to refer to all $q(j|i)$. We have simply added a summation over the the training examples. By relying on the previous result provide the argument for why

$$\tilde{J}(Q;\theta) \quad \leq \quad l(\theta; D), \quad \text{and} \quad \max_{Q} \tilde{J}(Q;\theta) = l(\theta; D) \tag{16}$$

where the maximization is taken with respect to all $q(j|i)$ (which can be chosen independently for each different i).

5. [**5pt**] We are almost there. Let θ^t be our current setting of the parameters in the mixture model. We use Q^t to refer to the corresponding setting $q^t(j|i) = P(j|\mathbf{x}_i, \theta^t)$. Finding Q^t represents the E-step of the EM algorithm, where we evaluate the posterior probabilities over the mixture components. Let's look at the M-step in our new formulation. Given Q^t, show that the parameters θ^{t+1} that we would get by maximizing $J(Q^t;\theta)$ with respect to θ correspond exactly the M-step of the EM algorithm. It suffices to show this for μ_k only (as in the case of gradient ascent above).

6. Based on our derivations above, we finally have the following chain of inequalities:

$$l(\theta^t; D) = J(Q^t;\theta^t) \overbrace{\leq J(Q^t;\theta^{t+1})}^{\text{M-step}} \overbrace{\leq J(Q^{t+1};\theta^{t+1})}^{\text{E-step}} = l(\theta^{t+1}; D) \tag{17}$$

This ensures a monotonic increase in the log-likelihood.

7. [**5pt**] Provide a brief summary argument explaining this result and how we get it on the basis of the previous ones.

It is worthwhile to note that the EM algorithm is not restricted to mixture models although we have introduced the algorithm in this context. It is applicable more generally to estimation problems where the data is incomplete. Our available data here was also incomplete since we only had $\mathbf{x}$ and not the component that $\mathbf{x}$ were generated from. More precisely, the variable specifying the mixture component was unobserved (not included in the data).

Problem 3: EM and mixture of Gaussians

In this problem we will investigate properties of EM algorithm applied to the problem of estimating mixture of Gaussians model. We will use a set of 2D data points, provided in `MOGdata.mat`. The implementation of regularized EM for mixture of Gaussians estimation is provided in `em_mix.m`

The command to run EM is

```
[param,history,ll] = em_mix(data,m,eps);
```

where the input points are given as rows of `data`, and `m` is the number of component is the estimated mixture. In the output, `param` is a cell array with `m` elements. Each element is a structure with the following fields:

`mean` - the resulting mean of the Gaussian component,

`cov` - the resulting covariance matrix of the component,

`p` - the resulting estimate of the mixing parameter.

The value of `param` is updated in every iteration of EM; the output argument `history` contains copies of these subsequent values of `param` and allows to analize our experiments. Finally, `ll` is the vector t-th element of which is the value of regularized log-likelihood over the data achieved after t iterations (i.e. the last element of `ll` is the final log-likehood of the fitted mixture of Gaussians). The EM is programmed to stop when the relative change in the log-likelihood between iteration falls below a small threshold.

1. [**10pt**] Run EM with 4 components on the data. Plot the graphs of estimated mixing probabilities for each component as a function of EM iterations. Compare the convergence speed of the mixing probabilities for the different components. Repeat this experiment at least 10 times (recall that the EM is initialized randomly, and therefore its results are a random variable), to evaluate the tendency. Turn in a typical plot (which should contain 4 lines, one for each mixing probability) and explain the observed behavior; in particular, if the parameters for some of the Gaussian components converge faster, point that out and explain why it happens.

The data in this problem was actually generated with 4 Gaussian components. However, in an unsupervised setting we can't know that and must set the number of component based on the data and the results of EM. This is a case of model selection, and we will attempt to approach it using Bayesian Information Criterion (BIC) (see the first problem).

2. [**5pt**] For `m` from 1 to 6, run EM with `m` components on the data from previous question, and compute the BIC score of each model. Repeat the experiment at least 10 times. Which model tends be selected in the majority of cases?

3. [**10pt**] Now repeat the experimental setup from the previous question with data sets provided in `MOGsmall.dat` and `MOGlarge.dat`, which have 40 and 8000 examples respectively. Those data sets were generated by the same 4-component model as `MOGdata`. Which models tend to be selected? Do the results differ from model selection in the preceding question? Explain why or why not.