6.867 Machine Learning

Problem Set 5

Due Friday 11/21

Please address all questions and comments about this problem set to `6.867-staff@ai.mit.edu`.

Problem 1: Description length and bayesian score

We explore here briefly a model selection approach based on *Bayesian score*. Suppose we wish to select among alternative models $M_1, M_2, \ldots, M_k, \ldots$. Each model comes with a parameterized distribution over labels $P(y|\mathbf{x}, \theta_k)$, where θ_k denotes the set of parameters associated with model M_k, chosen from the set of possible parameters Θ_k. For example, if M_1 is a linear classifier and $\mathbf{x} \in \mathcal{R}^d$, then $\theta_1 = [\theta_{10}, \ldots, \theta_{1d}]'$ and $\Theta_1 = \mathcal{R}^{d+1}$.

Each model M_k is also associated with a specific prior distribution over the parameters, $P(\theta_k)$. When the parameter space is compact, for example

$$\Theta_k = \{\theta_k : |\theta_{kj}| \leq c,\ j = 0, \ldots, d\} \tag{1}$$

then we can choose the prior distribution to be uninformative in the sense that there's no a priori bias towards one setting or another. The prior in this case is simply a constant density

$$P(\theta_k) = \frac{1}{V}, \quad \text{where } V = \int_{\Theta_k} 1 \, d\theta_k \tag{2}$$

When the parameter space is not compact we would have to define a (nearly) uninformative prior in other ways. It is strictly speaking a bit questionable to call any prior "uninformative" but we will do so regardless. The nature of the prior won't be relevant to our initial discussion here.

The bayesian model is "trained" by updating the distribution over the parameters based on the observations. Let us consider each training example-label pair in turn. The probability that we assign to the first label y_1 given $\mathbf{x}_1$ is calculated based on the original prior:

$$P(y_1|\mathbf{x}_1, M_k) = \int_{\Theta_k} P(y_1|\mathbf{x}_1, \theta_k) P(\theta_k) d\theta_k \tag{3}$$

In other words, we predict with the average probability of y_1 given $\mathbf{x}_1$, averaged relative to the prior. Note that this probability still depends on the model since both the parameter

space and the prior are properties of the model M_k. In terms of description length, sending the label y_1 now costs us

$$\mathrm{DL}(1) = -\log_2 P(y_1|\mathbf{x}_1, M_k) \tag{4}$$

bits. Once we have sent the first bit we can use its label to refine the distribution over the parameters (the receiver could in principle repeat our actions once they already have y_1). The distribution over the parameters (posterior) after seing the first label is

$$P(\theta_k|D_1) = \frac{P(y_1|\mathbf{x}_1, \theta_k)P(\theta_k)}{\int_{\Theta_k} P(y_1|\mathbf{x}_1, \theta'_k)P(\theta'_k)d\theta'_k} = \frac{P(y_1|\mathbf{x}_1, \theta_k)P(\theta_k)}{P(y_1|\mathbf{x}_1, M_k)} \tag{5}$$

where $D_1 = \{\mathbf{x}_1, y_1\}$.

We predict the second label analogously, now using $P(\theta_k|D_1)$ in place of the prior:

$$\begin{aligned}
P(y_2|\mathbf{x}_2, D_1, M_k) &= \int_{\Theta_k} P(y_2|\mathbf{x}_2, \theta_k)P(\theta_k|D_1)d\theta_k & (6)\\
&= \int_{\Theta_k} P(y_2|\mathbf{x}_2, \theta_k)\frac{P(y_1|\mathbf{x}_1, \theta_k)P(\theta_k)}{P(y_1|\mathbf{x}_1, M_k)}d\theta_k & (7)\\
&= \frac{1}{P(y_1|\mathbf{x}_1, M_k)} \cdot \int_{\Theta_k} \Big[P(y_2|\mathbf{x}_2, \theta_k)P(y_1|\mathbf{x}_1, \theta_k)\Big]P(\theta_k)d\theta_k & (8)
\end{aligned}$$

Sending this label will cost us

$$\begin{aligned}
\mathrm{DL}(2|1) &= -\log_2 P(y_2|\mathbf{x}_2, D_1, M_k) & (9)\\
&= \log_2 P(y_1|\mathbf{x}_1, M_k) - \log_2 \int_{\Theta_k} \Big[P(y_2|\mathbf{x}_2, \theta_k)P(y_1|\mathbf{x}_1, \theta_k)\Big]P(\theta_k)d\theta_k & (10)
\end{aligned}$$

bits. Let's see how we are doing and add the cost of sending the first two labels:

$$\begin{aligned}
\mathrm{DL}(1,2) &= \mathrm{DL}(1) + \mathrm{DL}(2|1) & (11)\\
&= -\log_2 \int_{\Theta_k} \Big[P(y_2|\mathbf{x}_2, \theta_k)P(y_1|\mathbf{x}_1, \theta_k)\Big]P(\theta_k)d\theta_k & (12)
\end{aligned}$$

This suggests that if we continue in the same manner the cost of sending all the n labels will be exactly

$$\mathrm{DL}(1, \ldots, n) = -\log_2 \int_{\Theta_k} \Big[\prod_{i=1}^{n} P(y_i|\mathbf{x}_i, \theta_k)\Big]P(\theta_k)d\theta_k \tag{13}$$

bits. There are two important things to note about this result. First, it does not depend on the order in which we have chosen to send the labels. This property where the order of the samples does not change the probability is referred to as *exchangeability*. Second, it seems we'd get the same result if we encoded the labels using the following overall conditional distribution:

$$P(y_1, \ldots, y_n|\mathbf{x}_1, \ldots, \mathbf{x}_n, M_k) = \int_{\Theta_k} \Big[\prod_{i=1}^{n} P(y_i|\mathbf{x}_i, \theta_k)\Big]P(\theta_k)d\theta_k \tag{14}$$

Let's understand this probability in a bit more detail. First, the labels are not independent of each other according to this conditional distribution. We did, after all, adjust the distribution over the parameters after each training example and label in the sequential procedure; labels later on were predicted with the knowledge of the preceding ones. So why is it that the above conditional distribution over all the labels together does not appear to be "trained" at all but merely averaged relative to the prior? This is because our sequential procedure was a fair way of evaluating the Bayesian method; we only conditioned predictions on information already available. Indeed, unlike with the maximum likelihood criterion, no other complexity penalty is necessary for evaluating the specific model. The log-probability

$$\log \int_{\Theta_k} \Big[\prod_{i=1}^{n} P(y_i|\mathbf{x}_i, \theta_k) \Big] P(\theta_k) d\theta_k \tag{15}$$

is known as the *Bayesian score* for model M_k. Note that the description length and the Bayesian score are one and the same (only differ in terms of the overall sign and typically use a different base logarithm). We will adopt the Bayesian score in the remainder of this problem.

The problem: In this problem we will use the Bayesian score for model selection. For simplicity we assume that the labels are binary (0/1) and $x \in [0, 1]$. The inputs x are drawn uniformly at random from the interval $[0, 1]$. We define the set of possible models as follows. For any $k = 0, 1, \ldots$ (indexing model M_k) we divide the segment $[0, 1]$ into 2^k equal size regions. Within the j^{th} region $S_{j;k}$ we set the overall probability of 0/1 labels:

$$P(y = 1|x, \theta_k) = \theta_{j;k}, \quad \text{when} \quad x \in S_{j;k} \tag{16}$$

where $\theta_k = \{\theta_{j;k},\ j = 1, \ldots, 2^k\}$. The parameters across the regions can be set independently. We use an uninformative prior over the parameters, as defined earlier. Note that model M_{k+1} can be seen as a refinement of M_k for any k.

(1-1) (2pts) Write explicitly the uninformative prior over the parameters $\theta_k = \{\theta_{j;k},\ j = 1, \ldots, 2^k\}$ for model M_k.

Let $n_{j;k}$ denote the number of times x falls in region $S_{j;k}$ based on the available data $\{(x_1, y_1), \ldots, (x_n, y_n)\}$. Note that these counts depend on the model since they are specific to the regions that the model uses. We also define $n_{j;k}(y)$ as the counts of labels in these regions. Clearly, $\sum_{y=0,1} n_{j;k}(y) = n_{j;k}$.

(1-2) (5pts) Let $\mathbf{x}_{j;k}$ be the set of points that fell in region $S_{j;k}$ (there are $n_{j;k}$ of them) and $\mathbf{y}_{j;k}$ the corresponding labels (summarized by the statistics $n_{j;k}(y)$). Express the conditional probability $P(\mathbf{y}_{j;k}|\mathbf{x}_{j;k}, \theta_{j;k})$ in terms of the counts. Collect the results for all regions to get $P(y_1, \ldots, y_n|x_1, \ldots, x_n, \theta_k)$.

(1-3) (10pts) Using the counts $n_{j;k}(y)$ given above and the integral

$$\int_0^1 \theta^k (1-\theta)^l d\theta = \frac{k!l!}{(k+l+1)!} \tag{17}$$

write the expression for the bayesian score of model M_k. How do the regions without points contribute to the score?

To understand the score in a bit more detail let's consider the choice between models M_0 and M_1. The same type of choices are made (only at a finer level) when choosing between M_k and M_{k+1}. Suppose the available data consists of exactly $2n$ points, where n of them are labeled $y = 1$ and the rest $y = 0$. We are interested in comparing the scores for M_0 and M_1 based on how the $2n$ samples are divided across the two regions that M_1 cares about (M_0 only has one region, the interval $[0, 1]$).

Now, we can put $i = 0, \ldots, n$ of the points labeled $y = 1$ in the first region $[0, 1/2)$ and the rest in the second. We can similarly divide the points labeled $y = 0$ in $n + 1$ possible ways across the two regions. We are interested in understanding the difference between the scores for M_0 and M_1 for any combination of such divisions. In other words, we are exploring how we would choose between the models in response to the data.

(1-4) (10pts) We have provided you with a skeleton routine `compareBayes(n)` that plots where the scores for M_1 and M_0 match. The two axis of the plot are the fraction of $y = 1$ points that fall in the first region, and the fraction of $y = 0$ points that fall in the first region. Since our prior places no preference over either label, the plot is symmetric. You need to

a) fill in the evaluation of the scores for the two models.

b) Run the routine with $n = 10$ and $n = 50$. Interpret the regions separated in the plot. Interpret any differences in the plots corresponding to $n = 10$ and $n = 50$.

BIC (Bayesian Information Criterion) is an asymptotic approximation to the Bayesian score. In our setting it is given by

$$\log P(y_1, \ldots, y_N | x_1, \ldots, x_N, \hat{\theta}_k) - \frac{d_k}{2} \log(N) \tag{18}$$

where N is the number of data points (here $2n$), $\hat{\theta}$ is the maximum likelihood parameter estimates for the model M_k, and d_k is the number of independent parameters in the model (cf. the asymptotic MDL criterion discussed in the lectures).

(1-5) (10pts) Use the provided routine `compareBIC(n)` to produce the analogous plots for BIC. Compare the resulting plots with those of the Bayesian score. Are there any systematic differences? Does one criterion tend to select more complex models than the other?

Problem 2: EM and mixture models

In this problem you will develop variants of the expectation-maximization (EM) procedure to perform parameter estimation w.r.t. to a probabilistic model in two variables $(\mathbf{x}, y)$. Here $\mathbf{x} \in \mathcal{R}^d$ is *visible* and y is *hidden*. Our training data is comprised of just samples $(\mathbf{x}_1, \ldots, \mathbf{x}_n)$ of the visible variable $\mathbf{x}$ but we still wish to estimate a joint model over both $\mathbf{x}$ and y. One motivation for this setting goes as follows. In a typical application the full distribution over the visible variables $P(\mathbf{x})$ may be too complicated to model directly. Instead, we assume that with some additional information, call it y, the conditional distribution $P(\mathbf{x}|y)$ takes a much simpler and manageable form.

We consider here generative models of the form $P(\mathbf{x}, y) = P(\mathbf{x}|y)P(y)$. There are many ways of parameterizing generative models of this type. Let $\theta \in \Theta$ denote the parameters of this model (we will discuss specific choices below). The set of possible joint distributions is then given by

$$\mathcal{M} = \{P \ : \ P(\mathbf{x}, y) = P(\mathbf{x}|y, \theta)P(y|\theta), \ \theta \in \Theta\} \tag{19}$$

The Expectation-Maximization (EM) algorithm is an iterative parameter fitting routine designed for this situation. Rather than attempting to optimize the likelihood directly, we introduce an auxiliary distribution $Q(\mathbf{x}, y)$, with the marginal distribution over $\mathbf{x}$ set to the empirical distribution of the data. That is, $Q(\mathbf{x}, y) = Q(y|\mathbf{x})Q(\mathbf{x})$ where $Q(\mathbf{x}) = \frac{1}{n}\sum_i \delta(\mathbf{x}|\mathbf{x}_i)$. We are free to choose $Q(y|\mathbf{x}_i)$ for each sample $\mathbf{x}_i$.

We then minimize the Kullback-Leibler divergence (up to a constant) between the auxiliary distribution Q and the model $P \in \mathcal{M}$.

$$\begin{aligned} J(Q, P) &= E_{(\mathbf{x},y)\sim Q}\left\{\log \frac{Q(\mathbf{x}, y)}{P(\mathbf{x}, y)}\right\} &(20)\\ &= \frac{1}{n}\sum_{i=1}^{n}\sum_{y} Q(y|\mathbf{x}_i) \log \frac{Q(y|\mathbf{x}_i)}{P(\mathbf{x}_i, y)} + \text{const.} &(21)\end{aligned}$$

This is minus the auxiliary objective defined in lecture (save for the constant). The minimization is accomplished by providing an initial guess $P^{(0)} \in \mathcal{M}$ about the joint model and then alternating the following steps:

1. *E-step.* Update the conditional part of the auxiliary distribution, i.e., update $Q(x|y)$, so as to minimize the objective.

$$Q^{(k)} = \arg\min_{Q(y|\mathbf{x})} J(Q, P^{(k)}) \tag{22}$$

   The solution to the E-step is given by selecting $Q^{(k)}(y|\mathbf{x})$ to agree with the posterior under $P^{(k)}$, i.e. by setting $Q^{(k)}(y|\mathbf{x}_i) = P^{(k)}(y|\mathbf{x}_i)$ for all y and for each sample $\mathbf{x}_1, \ldots, \mathbf{x}_n$.

2. *M-step.* Update P so as to minimize the objective.

$$P^{(k+1)} = \arg\min_{P\in\mathcal{M}} J(Q^{(k)}, P) \tag{23}$$

The solution to the M-step depends upon the model $\mathcal{M}$ but reduces to weighted maximum-likelihood calculation over this parameterized family of probability distributions.

$$P^{(k+1)} = \arg\max_{P\in\mathcal{M}} \sum_{i=1}^{n} E_{y\sim Q^{(k)}(y|\mathbf{x}_i)}\{\log P(\mathbf{x}_i, y)\} \quad (24)$$

$$= \arg\max_{P\in\mathcal{M}} \sum_{i=1}^{n} \sum_{y} Q^{(k)}(y|\mathbf{x}_i) \log P(\mathbf{x}_i, y) \quad (25)$$

Iterating these two steps for $k = 0, 1, \ldots$, EM generates a sequence of models $P^{(k)} \in \mathcal{M}$. As discussed in lecture, the log-likelihood of the data $\{\mathbf{x}_i, i = 1, \ldots, n\}$ w.r.t $P^{(k)}$ is monotonically increasing as a function of k.

Discrete EM. First, let's consider the case that both $\mathbf{x}$ and y are discrete random variables.

(2-1) (5pts) Let y be a binary (0/1) random variable and let $\mathbf{x}$ take 2^d possible values, $\mathbf{x} \in \{0, \ldots, 2^d - 1\}$. Let $P(\mathbf{x}, y) = P(\mathbf{x}|y)P(y)$ be fully-parameterized; in other words, there are no restrictions on how we can set $P(\mathbf{x}|y)$ and $P(y)$. Work out both the E- and M-steps for this simple problem. Can you explain why this simple case is perhaps not very interesting?

(2-2) (10 pts) Next, suppose we can instead decompose the output variables $\mathbf{x}$ into d binary components $\mathbf{x} = (x_1, \ldots, x_d)$ such that each of these are 0/1 binary random variables. Let's also model the components as being conditionally independent given y, that is require that $P(\mathbf{x}|y) = \prod_{k=1}^{d} P(x_k|y)$. Work out the M-step for this problem. Implement your procedure in MATLAB and run your procedure based on the data `data1` in `hw4em1.mat`. In this context, EM is essentially trying to "learn" 1-bit of information that the components $\mathbf{x}_1, \ldots, \mathbf{x}_d$ have in common.

Note that we are not looking for a general implementation of EM but a simple one tailored to this case.

(2-3) (optional) How could we generalize the above approach to refine our model? (*Hint.* Let $y \in \{1, \ldots, N\}$). Implement your proposal and employ the Bayesian information criterion (BIC) to select the order N of your model.

Gaussian Mixtures and EM. Next consider the case that the hidden variable $y \in \{1, ..., m\}$ is discrete while the visible variable $\mathbf{x} \in \mathcal{R}^d$ is continuous. In other words, we consider *mixture models* of the form

$$p(\mathbf{x}) = \sum_{j=1}^{m} p(\mathbf{x}|y = j)P(y = j) \quad (26)$$

We assume throughout that $\mathbf{x}$ is *conditionally Gaussian* in the sense that $\mathbf{x} \sim \mathcal{N}(\mu_j, \Sigma_j)$ when $y = j$.

We have provided you with the EM code for mixture of Gaussians (with visualization). The command to run is

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

where the input points are given as rows of `data`, `m` is the number of components in the estimated mixture, and `eps` determines the stopping criteria of EM: the algorithm stops when the relative change in log-likelihood falls below `eps`. 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 after every iteration of EM; the output argument `history` contains copies of these subsequent values of `param` and allows to analyze our experiments. Finally, `ll` is the vector where the t-th element is the value of the log-likelihood of the data after t iterations (i.e. the last element is the final log-likelihood of the fitted mixture of Gaussians).

To overcome any numerical problems the code involves (slight) regularization. Specifically, the M-step of the EM algorithm solves a regularized weighted log-likelihood problem, where the prior distribution over the mixing proportions is a Dirichlet and the prior over each covariance matrix is a Wishart. The equivalent sample size is set to one. The returned log-likelihood values include the regularization penalties (log-priors). See the code for details if you wish to change any of these settings.

(2-4) (10pts) Run the EM algorithm based on `data2` provided by `hw5em2.mat` with $m = 2, 3, 4, 5$ components. Select the appropriate model according to the Bayesian Information Criterion (BIC). Note that you may have to rerun the algorithm a few times (and select the model with the highest log-likelihood) for each choice of m as EM can sometimes get stuck in a local minimum. Is the model selection result sensible based on what you would expect visually? Why or why not?

(2-5) (optional) Modify the M-step of the EM code so that the covariance matrices of the Gaussian components are constrained to be equal. Rerun the model selection problem using BIC as the selection criterion. Would we select a different number of components in this case?

Problem 3: Spectral clustering

In this problem we will experiment with k-means and a spectral clustering algorithm. We have provided you with a skeleton for a k-means clustering routine, as well as two routines for data generation and plotting:

`X = mkdata(m,n)` generates random 2D data points, from `m` Gaussian distributions, with `n` points from each.

`X = build` allows you to create a data set of manually (graphically) entered points.

`plotclust(X,y)` plots the 2D points specified in the rows of X, and colors each point according to the cluster assignments specified in the vector y (one integer value per point).

The file `clustdata.mat` also contains three data sets `X1`, `X2` and `X3`.

(3-1) (10pts) Complete the provided skeleton for k-means clustering. The provided skeleton initializes the centers to a random subset of the points, and imposes a constant number (100) of k-means iterations.

Use your k-means implementation to cluster the three provided data sets into $k = 2, 3, 4, 5$ clusters. For each data set and each number of clusters, try running the routine several times. You can also generate additional data sets and test the behavior of k-means on them.

(3-2) (5pts) Why do different executions with the same setting for k yield different clusterings? How would you choose, in an automated way, between these various clusterings? Write a matlab routine that runs k-means five times and chooses the best of the five clusterings (it might be useful to add some calculations, or output variables, to the k-means routine).

(3-3) (5pts) Provided that you have only limited computation time, do you think it makes sense to run k-means several times separately and choose a clustering based on the criterion you suggested, instead of running k-means a single time for more iterations? Why?

(3-4) (5pts) Could the same criterion be used in order to choose between different number of clusters? Why?

Although you should be able to cluster data set `X1`, as well as most data sets generated by `mkdata`, using k-means clustering, the clustering produced for data sets `X2` and `X3` do not seem 'natural'.

(3-5) (5pts) We saw in lecture how k-means can be seen as a 'hard' version of Gaussian mixture density estimation. Suggest a clustering procedure based on Gaussian mixture density estimation that would be able to cluster data set two into natural clusters. You do not need to implement this method. (Hint: would a method based on equal-variance spherical Gaussian be able to capture the clusters?).

We will now see how spectral clustering can be used to cluster data such as `X2` and `X3`.

A Matlab implementation of the spectral clustering method presented in lecture is provided in `spectral.m`. You can call it as

```
labels = spectral(X,2,r,beta);
```

where `X` is an $n \times 2$ data matrix, with a 2D data vector in each row, `r` is the number of neighbors to use in the neighborhood graph, and `beta` is the parameter determining the exponential decay of the edge weights as a function of the distance. The default value for `r` is 3 and the default for `beta` is 1. You might want to refer to Fall 2002 Problem Set 6 (solutions available online; link from the course website) for a discussion on the effect of various settings of r and β.

Our routine only implements clustering into two clusters, as presented in lecture.

A possibly way of extending the spectral clustering framework to more than two clusters is as follows: instead of considering only the eigenvector with the second largest eigenvalue, consider the k eigenvectors $v_1, \ldots, v_k$ with the k largest eigenvalues. The ith point in the data set can now be represented by the k-dimensional vector $(v_1[i], v_2[i], \ldots, v_k[i])$. We can subsequently use k-means in order to cluster the points in this new k-dimensional representation (k-means and k-dimensional can refer to different k's).

(3-6) (5pts) Given an $n \times n$ transition probability matrix P show that the leading eigenvector (the eigenvector with largest eigenvalue) can always be taken to be the vector of all ones.

(3-7) (10pts) Modify our spectral clustering routine to cluster the points into k clusters using this method.

Run the spectral clustering routine, with $k = 2, 3, 4, 5$ on the three data sets, and possibly on other data sets you generated, and experiment with various settings of r and β. Try explaining to yourself the values of r and β that result in the 'natural' clustering, and what happens, e.g., when r is too small, or β is too small or too large.

Another advantage of spectral clustering is that all the operations can be done purely in terms of distances between the points. This means that the points to be clustered need not be points in any R^d Euclidean space, and all that is needed are the distances between the points, or even just the distances between close-by points.

(3-8) (5pts) Why is it not enough to specify only inter-point distances for k-means? What other operation needs to be performed on the input points in k-means?