6.891 Machine learning and neural networks

Problem set 3

Deadline: November 2, in class

Note: You will need to use MATLAB in this problem set. Information about Matlab can be found on the course web site[1]. Data files for the problems will be made available via the course Athena locker: `/mit/6.891`.

Reading: Lecture notes 1-11, Chapters 3.8, 5.1-5.5, 5.11, 9.5 and C. Burges' tutorial (pages 1-8).

Some of the problems have been marked as optional. This means that we want you to submit solutions to only one of the optional problems. We encourage everyone to at least read through all the questions, however. You are expected to understand the solutions to all these problems even if you might not have the background to solve them on your own at this point.

Please try not to exceed 2-3 sentences for any explanations.

Problem 1: choose I or II

This question is about AdaBoost, the simple boosting algorithm presented in the lectures. We assume that there exists a method that can produce a weak learner or a component hypothesis $h(\mathbf{x})$ that does a bit better than random guess on any weighted training set $D = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$. We denote the weights on the examples at the k^{th} boosting iteration as $p_k(1), \ldots, p_k(n)$, $\sum_{i=1}^n p_k(i) = 1$. All the hypotheses $h(\mathbf{x})$ produce binary ± 1 outputs.

Starting with equal weights $p_1(i) = 1/n$, AdaBoost generates a sequence of hypotheses $h_1(\mathbf{x}), \ldots, h_m(\mathbf{x})$ which are trained using different example weights. After having generated a hypothesis $h_k(\mathbf{x})$ at the k^{th} iteration, the weights $p_k(i)$ are updated as follows

$$p_{k+1}(i) = c \cdot p_k(i) \, \exp(\, -\alpha_k y_i h_k(\mathbf{x}_i)\,), \;\; i = 1, \ldots, n \tag{1}$$

where c is the normalization constant ensuring that $\sum_{i=1}^n p_{k+1}(i) = 1$ and $\alpha_k = 1/2 \cdot \log[(1-\epsilon_k)/\epsilon_k]$ (note 1/2 in front of the log; there was a mistake in the lecture slides). ϵ_k here is

[1] http://www.ai.mit.edu/courses/6.891

the weighted training error that $h_k(\mathbf{x})$ achieves:

$$\epsilon_k = \sum_{i=1}^{n} p_k(i) \, [[h_k(\mathbf{x}_i) \neq y_i]] \tag{2}$$

where we define $[[\cdot]]$ to be one if the logical argument is true and zero otherwise.

Problem 1, option I (more theoretical)

a) Show that the choice of α_k above quarantees that the hypothesis $h_k(\mathbf{x})$ just generated has a weighted error 0.5 relative to the updated weights $p_{k+1}(i)$. Equivalently, you can show that this choice of α_k ensures that the (weighted) labels become decorrelated with the outputs:

$$\sum_{i=1}^{n} p_{k+1}(i) \, y_i \, h_k(\mathbf{x}_i) = 0 \tag{3}$$

b) What relevance if any does this have for the quality of the hypothesis generated?

c) Relative entropy or Kullback-Leibler divergence is a measure of distance between two distributions. So, for example, we can evaluate the distance between the different example weightings $p_{k+1}(i)$ and $p_k(i)$ as

$$D(p_{k+1} \| p_k) = \sum_{i=1}^{n} p_{k+1}(i) \log \frac{p_{k+1}(i)}{p_k(i)} \tag{4}$$

(this is not symmetric so it's not a distance in a strict sense but satisfies many of the properties for distances)

Now, suppose we want to find the weights $p_{k+1}(i)$ such that they are closest to $p_k(i)$ in the above relative entropy sense but still satisfy the decorralation condition

$$\sum_{i=1}^{n} p_{k+1}(i) y_i h_k(\mathbf{x}_i) = 0 \tag{5}$$

Solve: $\min D(p_{k+1} \| p_k)$ with respect to the positive weights $p_{k+1}(i)$ summing to one subject to the above decorrelation contraint. Use Lagrange multipliers to get the solution.

d) How does α_k defined above relate to the Lagrange multiplier from part c)?

Problem 1, option II (more practical)

We have provided you with MATLAB code that finds and evaluates decision stumps. These are the hypothesis that the boosting algorithm assumes we can generate. We ask you to use this code to generate a short matlab script that "boosts" these decision stumps. We have provided you with skeletons "boost_digit.m", "boost.m", "eval_boost.m", "find_stump.m", "eval_stump.m".

a) Run the boosting algorithm on the digits example (see "boost_digit.m"). Try running it 1, 5, 10, and 20 iterations. Plot the training and test errors corresponding to each case. Also identify the pixels that the decision stumps are concentrating on. Do you expect that such "features" your boosting algorithm finds here are similar to those that the simple filtering method discussed in the previous problem set would find?

b) Are the resulting training and test errors what you would expect them to be? Why or why not?

c) Now limit the coefficients α_k to be in the interval $[0, 1]$. In other words, set $\alpha_k = min(1, \log[(1-\epsilon_k)/\epsilon_k)]$. Redo the experiment in part a) and again plot the test errors.

d) Discuss briefly why the additional constraint on α_k might help or hurt the test performance.

Problem 2

a) Show that the leave-one-out cross-validation estimate of the generalization error of a learning method is unbiased in the sense that taking the expected value of this estimate with respect to the probability of selecting the training set yields the true expected error.

b) Show that the leave-one-out cross-validation estimate for a suppoer vector machine is upper bounded by the number of support vectors.

c) Combine a) and b) to a result that says that the expected error of a support vector machine is upper bounded by the expected number of support vectors.

d) Part c) should give you some (but not strong) reason to think that the number of support vectors is an indication of the generalization error. Here we ask you to test whether this is at all true in practice. We have provided you with simple matlab code for SVMs ("train_svm.m" and "eval_svm.m"). "polykernel.m" and "gausskernel.m" permit you to use different types of kernels, including

$$K(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i^T \mathbf{x}_j)^p, \ \ p = 1, 2, \ldots \tag{6}$$

as well as

$$K(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\frac{1}{2\sigma^2}\|\mathbf{x}_i - \mathbf{x}_j\|^2\right) \tag{7}$$

where σ^2 is a positive kernel width parameter.

For polynomial kernels of degree $p \in \{1, 2, 4, 8\}$ and radial basis (Gaussian) kernels with width parameter $\sigma^2 \in \{1/2, 1/4, 1/9, 1/16\}$, evaluate the test error as a function of the resulting numbers of support vectors.

To ensure that the digits are normalized to one (with both positive and negative components) and that the labels are ± 1, please use the initialization script "preproc.m" (see below). Both "polykernel.m" and "gausskernel.m" assume that the digits have been normalized in this way.

```
% preproc.m

load digit_x.dat;
load digit_y.dat;
[n,m] = size(digit_x);

digit_y = 2*digit_y-1;
digit_x = (2*digit_x-1)/sqrt(m);

load digit_x_test.dat;
load digit_y_test.dat;

digit_y_test = 2*digit_y_test-1;
digit_x_test = (2*digit_x_test-1)/sqrt(m);
```

Note that you cannot simply evaluate the number of support vectors by finding the number of non-zero coefficients α_i. This is because the quadratic programming package finds the solution upto some specified accuracy. Count instead the number of α_i's that are larger than say $0.001/\max(\alpha_1, \ldots, \alpha_n)$.

e) Another indicator of generalization is the margin by which we can separate the training examples. Here we ask you to check how well the size of the margin correlates with the test performance.

Note first that $K(\mathbf{x}, \mathbf{x}) = 1$ for all (normalized digits) $\mathbf{x}$ for all the kernels discussed above. This means that the feature vectors are also normalized to one. So the size of the margin that we attain in the feature space is a fair indicator of separation. We have to have the feature vectors normalized since scaling clearly affects how widely they are spaced.

Now, to evaluate the margin, recall that SVM finds $\mathbf{w}$ in the feature space such that

$$y_i(\mathbf{w}^T\mathbf{x}_i + w_0) - 1 \geq 0. \tag{8}$$

and for all support vectors $i \in SV$ (assuming that the examples are separable)

$$y_i(\mathbf{w}^T\mathbf{x}_i + w_0) - 1 = 0 \tag{9}$$

So, the examples seem to be separated from the boundary by 1. But this is scaled by $\|\mathbf{w}\|$. The margin or the orthogonal distance from the boundary is therefore

$$\text{Margin} = \frac{1}{\|\mathbf{w}\|} \tag{10}$$

Using the form of the SVM solution for $\mathbf{w}$, express the margin in terms of the kernel function, $\{\alpha_i\}$, and $\{y_i\}$.

f) Evaluate the test errors as a function of the margin attained for the training examples. Provide a brief explanation for the results.

Problem 3: choose option I or II

Problem 3, option I (more theoretical)

Here you are to show that the EM-algorithm increases the likelihood of the training examples monotonically after each iteration.

Let y and j index mixture components and i the training examples. In this notation, for example, the log-likelihood of the observed data is given by

$$\begin{aligned}
J(\theta) &= \sum_{i=1}^{n} \log P(\mathbf{x}_i|\theta) = \sum_{i=1}^{n} \log \left[\sum_{j=1}^{m} P(\mathbf{x}_i, y_i = j|\theta) \right] && (11) \\
&= \sum_{i=1}^{n} \log \left[\sum_{j=1}^{m} P(y_i = j|\theta) P(\mathbf{x}_i|y_i = j, \theta) \right] && (12) \\
&= \sum_{i=1}^{n} \log \left[\sum_{j=1}^{m} p_j \, P(\mathbf{x}_i|\mu_j, \Sigma_j) \right] && (13)
\end{aligned}$$

where $\theta = \{p_1, \ldots, p_m, \mu_1, \ldots, \mu_m, \Sigma_1, \ldots, \Sigma_m\}$ and p_j is the prior probability of selecting component j.

a) Let $\theta = \theta_0$ be the current setting of the parameters in our mixture model. Show that performing the M-step of the EM-algorithm corresponds to maximizing

$$J(\theta|\theta_0) = \sum_{i=1}^{n} E_{\theta_0} \{ \log P(\mathbf{x}_i, y_i|\theta) \} \tag{14}$$

with respect to θ. Here the expectation is taken with respect to $P(y_i|\mathbf{x}_i, \theta_0)$ where the component indicator y_i is the random variable. It suffices to derive a single parameter update from $J(\theta|\theta_0)$ (e.g., for p_1) and show that it is indeed identical to the update in the M-step prsented in the lectures.

b) Before we move on we need an intermediate result. Show that for any distribution q_j, $j = 1, \ldots, m$, $\sum_j q_j = 1$ and positive coefficients c_j

$$\sum_{j=1}^{m} q_j \log \frac{c_j}{q_j} \leq \log \sum_{j=1}^{m} c_j \tag{15}$$

c) Use a) and b) to show that

$$J(\theta) - J(\theta_0) \geq J(\theta|\theta_0) - J(\theta_0|\theta_0) \tag{16}$$

d) Provide the summary argument for why part c) shows that EM increases the log-likelihood of the training examples monotonically.

Problem 3, option II (more practical)

Naive Bayes model is a very simple probability model that makes the tell-tale independence asumption about the components of the vector $\mathbf{x}$ given the mixture component. In other words,

$$P(\mathbf{x}, y) = P(y)P(\mathbf{x}|y) =^{*} P(y)\left[\prod_{j=1}^{d} P(x_j|y)\right] \tag{17}$$

where $y = 1, \ldots, m$ indexes the mixture components and d is the dimensionality of $\mathbf{x}$. We assume for simplicity that each component of the input vector, x_j, takes values in a discrete set $\{0, ..., r\}$ where from now on $r = 1$ (x_j are all binary variables).

a) We have given you a mixture of Gaussians code. Modify this code for the naive Bayes mixture model. The relevant files are

   "inimix.m" initializes the mixture of Gaussians model.

   "runmix.m", the overall script that initializes the mixture model and runs the em iterations.

   "em.m" performs one EM-iteration, returning the new parameters.

   "evalgauss.m" evaluates the log-likelihood that a single Gaussian assigns to an observation vector

   "evalmix.m" computes the log-likelihood of the data for a mixture of Gaussians distribution "softmax.m" computes the softmax distribution of the input.

Return the MATLAB code.

b) Suggest an appropriate initialization for these mixture models.

c) Apply your code to the digit recognition problem using 1 or 2 components for each digit. Check that the log-likelihood of the training examples increases monotonically after each iteration. Return the corresponding plots.

d) Evaluate the number of misclassified digits for the two mixture models and briefly discuss the results.