Homework #7

RELEASE DATE: 12/13/2012

DUE DATE: 12/27/2012, BEFORE THE END OF CLASS

QUESTIONS ABOUT HOMEWORK MATERIALS ARE WELCOMED ON THE FORUM.

Unless granted by the instructor in advance, you must turn in a printed/written copy of your solutions (without the source code) for all problems. For problems marked with (*), please follow the guidelines on the course website and upload your source code to designated places.

Any form of cheating, lying, or plagiarism will not be tolerated. Students can get zero scores and/or fail the class and/or be kicked out of school and/or receive other punishments for those kinds of misconducts.

Discussions on course materials and homework solutions are encouraged. But you should write the final solutions alone and understand them fully. Books, notes, and Internet resources can be consulted, but not copied from.

Since everyone needs to write the final solutions alone, there is absolutely no need to lend your homework solutions and/or source codes to your classmates at any time. In order to maximize the level of fairness in this class, lending and borrowing homework solutions are both regarded as dishonest behaviors and will be punished according to the honesty policy.

You should write your solutions in English with the common math notations introduced in class or in the problems. We do not accept solutions written in any other languages.

7.1 Power of Neural Networks

In this problem, we will work with Neural Networks using $\text{sign}(\cdot)$ instead of $\tanh(\cdot)$ as the activation functions. In addition, we will take $+1$ to mean logic TRUE, and -1 to mean logic FALSE. The Caltech lecture (HW 6.5) briefly mentioned that Neural Networks can implement any Boolean function.

(1) (10%) Show a neural network that implements

$$AND\Big((x)_1, NOT\big((x)_2\big), OR\Big((x)_3, (x)_4, NOT\big((x)_1\big)\Big)\Big).$$

(2) (10%) Show a 3-4-1 neural network that implements $XOR\big((x)_1, (x)_2, (x)_3\big)$.

(3) (10%) Show a 3-3-1 neural network that implements $XOR\big((x)_1, (x)_2, (x)_3\big)$.

7.2 Neural Network Training

(1) (10%) For a neural network with at least one hidden layer and $\tanh(\cdot)$ as the activation functions on all the non-input nodes, what is the gradient (with respect to the weights) when all the weights are set to zero?

7.3 Kernel from Decision Stumps

When talking about non-uniform voting in aggregation, we mentioned that $\boldsymbol{\alpha}$ can be viewed as a weight vector learned from any linear algorithm coupled with the following transform:

$$\boldsymbol{\phi}(\mathbf{x}) = \Big(h_1(\mathbf{x}), h_2(\mathbf{x}), \cdots, h_T(\mathbf{x})\Big).$$

When studying kernel methods, we mentioned that the kernel is simply a computational short-cut for the inner product $\boldsymbol{\phi}(\mathbf{x})^T\boldsymbol{\phi}(\mathbf{x}')$. In this problem, we mix the two topics together using the decision stumps as our $h(\mathbf{x})$.

(1) (10%) Assume that the input vectors contain only integers between (including) $-B$ and B.

$$h_{s,i,\theta}(\mathbf{x}) = \text{sign}\Big(s \cdot \mathbf{x}[i] - \theta\Big).$$

Two decision stumps $h^{(1)}$ and $h^{(2)}$ are defined as the *same* if $h^{(1)}(\mathbf{x}) = h^{(2)}(\mathbf{x})$ for every $\mathbf{x} \in \mathcal{X}$. Two decision stumps are different if they are not the same. Argue that there are only finitely-many different decision stumps for $\mathcal{X}$ and list all of them for the case of $d = 2$ and $B = 4$.

(2) (10%) Let $\mathcal{H} = \{$ all different decision stumps for $\mathcal{X}$ $\}$. Since $\mathcal{H}$ is finite, we can enumerate each hypothesis $h \in \mathcal{H}$ by some index t. Define

$$\boldsymbol{\phi}_{ds}(\mathbf{x}) = \Big(h_1(\mathbf{x}), h_2(\mathbf{x}), \cdots, h_t(\mathbf{x}), \cdots, h_{|\mathcal{H}|}(\mathbf{x})\Big).$$

Derive a simple equation that evaluates $K_{ds}(\mathbf{x}, \mathbf{x}') = \boldsymbol{\phi}_{ds}(\mathbf{x})^T \boldsymbol{\phi}_{ds}(\mathbf{x}')$ efficiently.

The result can be easily extended to the case when $\mathcal{X}$ is an arbitrary box in $\mathbb{R}^d$ as well.

7.4 Power of Adaptive Boosting

The adaptive boosting (AdaBoost) algorithm, as shown in the class slides, is as follows:

- For input $\mathcal{D} = \{(\mathbf{x}_n, y_n)\}_{n=1}^N$, set $u_n = \frac{1}{N}$ for all n.
- For $t = 1, 2, \cdots, T$,
  - Learn a simple hypothesis h_t such that h_t solves

$$h_t = \underset{h \in \mathcal{H}}{\text{argmin}} \sum_{n=1}^{N} u_n \cdot [\![y_n \neq h(\mathbf{x}_n)]\!] .$$

    with the help of some base learner A_b that learns from $h \in \mathcal{H}$.
  - Compute the weighted error $\epsilon_t = \dfrac{\sum_{n=1}^N u_n \cdot [\![y_n \neq h_t(\mathbf{x}_n)]\!]}{\sum_{n=1}^N u_n}$ and the confidence

$$\alpha_t = \frac{1}{2} \ln \frac{1 - \epsilon_t}{\epsilon_t}$$

  - Change the example weights: $u_n = u_n \cdot \exp\Big(-\alpha_t y_n h_t(\mathbf{x}_n)\Big)$.
- Output: combined function $H(\mathbf{x}) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(\mathbf{x})\right)$

In this problem, we will prove that AdaBoost can reach $E_{\text{in}}(H) = 0$ if T is large enough and every hypothesis h_t satisfies $\epsilon_t \leq \epsilon < \frac{1}{2}$.

(1) (10%) Let $U^{(t-1)} = \sum_{n=1}^{N} u_n$ at the beginning of the t-th iteration. According to the AdaBoost algorithm above, for $t \geq 1$, prove that

$$U^{(t)} = \frac{1}{N} \sum_{n=1}^{N} \exp\left(-y_n \sum_{\tau=1}^{t} \alpha_\tau h_\tau(\mathbf{x}_n)\right).$$

(2) (10%) By the result in (1), prove that $E_{\text{in}}(H) \leq U^{(T)}$.

(3) (10%) According to the AdaBoost algorithm above, for $t \geq 1$, prove that $U^{(t)} = U^{(t-1)} \cdot 2\sqrt{\epsilon_t(1 - \epsilon_t)}$.

(4) (10%) Using $0 \leq \epsilon_t \leq \epsilon < \frac{1}{2}$, for $t \geq 1$, prove that $\sqrt{\epsilon_t(1-\epsilon_t)} \leq \sqrt{\epsilon(1-\epsilon)}$.

(5) (10%) Using $\epsilon < \frac{1}{2}$, prove that $\sqrt{\epsilon(1-\epsilon)} \leq \frac{1}{2}\exp\left(-2(\frac{1}{2}-\epsilon)^2\right)$.

(6) (10%) Using the results above, prove that $U^{(T)} \leq \exp\left(-2T(\frac{1}{2}-\epsilon)^2\right)$.

(7) (10%) Using the results above, argue that after $T = O(\log N)$ iterations, $E_{\text{in}}(H) = 0$.

7.5 Optimization View of Adaptive Boosting

Assume that $\mathcal{H} = \{h_\ell\}_{\ell=1}^L$ is a finite set of hypotheses. Consider an error function for an ensemble $H(\mathbf{x})$ of the form $\text{sign}\left(\sum_{\ell=1}^L \beta_\ell h_\ell(\mathbf{x})\right)$:

$$E_{\text{exp}}(H) = \frac{1}{N}\sum_{n=1}^N \exp\left(-y_n \sum_{\ell=1}^L \beta_\ell h_\ell(\mathbf{x}_n)\right).$$

Using your results in Problem 7.4(2), we can easily show that $E_{\text{exp}}(H)$ is an upper bound of $E_{\text{in}}(H)$. We now prove that AdaBoost is equivalent to a particular way of minimizing $E_{\text{exp}}(H)$.

(1) (10%) Assume that we hope to update from $\boldsymbol{\beta}^{old}$ to $\boldsymbol{\beta}^{new}$ by changing only component i of $\boldsymbol{\beta}^{old}$. That is, for a given vector $\boldsymbol{\beta}^{old} = (\beta_1^{old}, \beta_2^{old}, \cdots, \beta_{L-1}^{old}, \beta_L^{old})$, we want to set

$$\beta_i^{new} = \beta_i^{old} + \Delta_i.$$

such that

$$\Delta_i = \underset{\Delta}{\text{argmin}} \quad \frac{1}{N}\sum_{n=1}^N \exp\left(-y_n\left(\sum_{\ell=1}^L \beta_\ell^{old} h_\ell(\mathbf{x}_n)\right) - y_n\big(\Delta \cdot h_i(\mathbf{x}_n)\big)\right).$$

Let $u_n = \frac{1}{N}\exp\left(-y_n \sum_{\ell=1}^L \beta_\ell^{old} h_\ell(\mathbf{x}_n)\right)$ and $\epsilon_i = \frac{\sum_{n=1}^N u_n \cdot [\![y_n \neq h_i(\mathbf{x}_n)]\!]}{\sum_{n=1}^N u_n}$. What is the optimal Δ_i in terms of ϵ_i?

(The result shows that the α_t in AdaBoost is the steepest descent choice for E_{exp} after getting h_t.)

(2) (10%) Suppose now that we want to pick the best i that greedily makes $E_{\text{exp}}(H)$ the smallest. That is, for a given vector $(\beta_1, \beta_2, \cdots, \beta_{L-1}, \beta_L)$, we want to solve

$$\min_{\Delta, i} \quad \frac{1}{N}\sum_{n=1}^N \exp\left(-y_n\left(\sum_{\ell=1}^L \beta_\ell^{old} h_\ell(\mathbf{x}_n)\right) - y_n\big(\Delta \cdot h_i(\mathbf{x}_n)\big)\right).$$

If all $h \in \mathcal{H}$ satisfies

$$\frac{\sum_{n=1}^N u_n \cdot [\![y_n \neq h(\mathbf{x}_n)]\!]}{\sum_{n=1}^N u_n} \leq \frac{1}{2},$$

show that the optimal

$$h_i = \underset{h \in \mathcal{H}}{\text{argmin}} \sum_{n=1}^N u_n \cdot [\![y_n \neq h(\mathbf{x}_n)]\!].$$

(The result shows that the h_t in AdaBoost is the optimal coordinate choice for E_{exp}.)

All the results can be extended to the case when $\mathcal{H}$ is of an infinite size as well. This problem shows that AdaBoost is optimizing a particular error function E_{exp} slowly (by coordinate).

7.6 Experiments with Adaptive Boosting (*)

(1) (50%) Implement the AdaBoost algorithm with decision stumps (i.e., use A_{ds} as A_b). Run the algorithm on the following set for training:

`http://www.csie.ntu.edu.tw/~htlin/course/ml12fall/hw7/hw7_train.dat`

and the following set for testing:

`http://www.csie.ntu.edu.tw/~htlin/course/ml12fall/hw7/hw7_test.dat`

Use a total of $T = 500$ iterations. Let $H_t(\mathbf{x}) = \text{sign}\left(\sum_{\tau=1}^{t} \alpha_\tau h_\tau(\mathbf{x})\right)$. Plot $E_{\text{in}}(H_t)$, $E_{\text{out}}(H_t)$ and $U^{(t)}$ (see the definition above) as functions of t on the same figure. Briefly state your findings.

7.7 Neural Networks for XOR

(1) (Bonus 10%) Following the assumptions in Problem 7.1, show a d-d-1 neural network that implements $XOR\big((x)_1, (x)_2, (x)_3, \ldots, (x)_d\big)$.