CS446: Pattern Recognition and Machine Learning **Spring 2005**

Midterm Exam

March 18, 2005

This is a closed book exam. Everything you need in order to solve the problems is supplied in the body of this exam.

The exam ends at 1:45 pm. It contains 5 problems. You have 75 minutes to earn a total of 100 points. Answer each question in the space provided.

If you need more room, write on the back side of the paper and indicate that you have done so.

Besides having the correct answer, being concise and clear is very important. For full credit, you must show your work and explain your answers.

Good Luck!

Name:

Problem 1 (20 points):	
Problem 2 (20 points):	
Problem 3 (20 points):	
Problem 4 (20 points):	
Problem 5 (20 points):	
Total (100 points):	

Problem 1 [Decision Trees - 20 points]

You are given a collection of data points S describing 8 students furiously studying for an exam. Each student has three observable attributes, and each is either *focused* on their studies or not. The *label* is IsFocused.

#	AmountOfSleep	IsInterested	HasBreakPlans	IsFocused
1	Not Enough	Yes	No	Yes
2	Too Much	Yes	No	Yes
3	None	Yes	No	No
4	Not Enough	Yes	No	Yes
5	None	No	Yes	No
6	Too Much	No	Yes	No
7	Not Enough	No	No	No
8	Not Enough	Yes	No	Yes

(a) [5] What is the entropy of the data?

Solution:
The entropy of the data is 1.

What is the entropy of the attribute *AmountOfSleep*?

Solution:

$$Entropy(AmountOfSleep) = -\frac{1}{4}\log_2\frac{1}{4} - \frac{1}{4}\log_2\frac{1}{4} - \frac{1}{2}\log_2\frac{1}{2} = \frac{3}{2}$$

A number of students interpreted this as the entropy of the split on *AmountOfSleep*, which wasn't asked for and somewhat more cumbersome to compute. However, that answer was also accepted.

(b) [5] Compute Gain(S, *HasBreakPlans*). (If needed, see the back page for formulas.)

Solution:

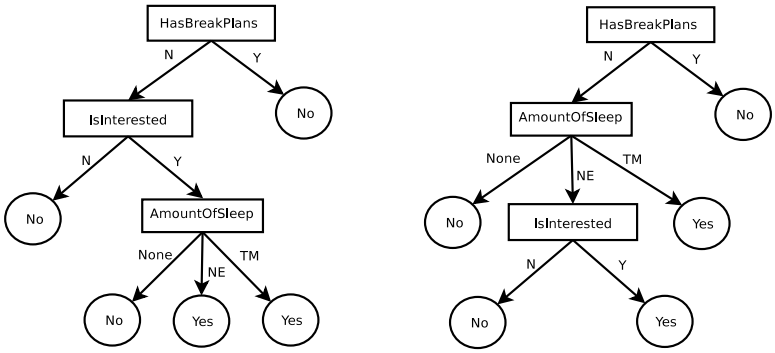
$$Entropy(AmountOfSleep) = 1 - \frac{1}{4}\cdot 0 - \frac{3}{4}\left(-\frac{1}{3}\log_2\frac{1}{3} - \frac{2}{3}\log_2\frac{2}{3}\right)$$

(c) [5] You are given that:
Gain(S, *AmountOfSleep*) = .344,
Gain(S, *IsInterested*) = .2049,
Gain(S,*HasBreakPlans*) = .5171
Use it to choose the appropriate root node.

Then, build the rest of the tree in a way that is *consistent* with the given data. Please note that beyond the root node, you don't need to use any specific algorithm; just make sure your tree is consistent with the data. Draw the tree below.

Solution:

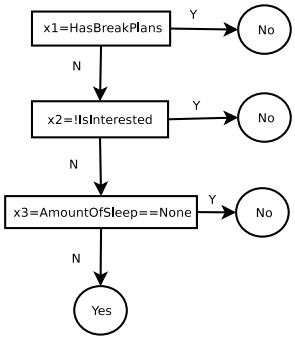
There are two possible trees you could have drawn:



(d) [5] Write a linear threshold function that produces the same prediction for the label *IsFocused* as the decision tree you drew above. Before doing it, decide and write down the features of your linear threshold function.

Solution:

You can convert either of the above trees into a 1-DL, and use the same method for constructing a linear threshold function as you saw in the homework. Example of the left tree:



The corresponding linear threshold function is:

$$-1 \cdot x_1 - \frac{1}{2} \cdot x_2 - \frac{1}{4} \cdot x_3 + \frac{1}{8} > 0$$

A number of students eyeballed it, which was fine as long as the function worked.

Problem 2 [Boosting - 20 points]
In this problem you will run AdaBoost. [**Please consult the formula sheet for the definitions of α and D**]

Suppose you have two weak learners, w_0 and w_1, and a set of 17 points. Your initial distribution D_0 is the uniform distribution.

(a) [12] You find that w_0 makes *one* mistake while w_1 makes *four* mistakes.

- Choose which learner w_0 or w_1 will AdaBoost use in the first round.

  Solution:
  $\epsilon_0 = \frac{1}{17} < \epsilon_1 = \frac{4}{17}$ Therefore, take w_0.

- Compute the weight α_0

  Solution:

$$\alpha_o = \frac{1}{2}\log_2\frac{1-\epsilon}{\epsilon} = \frac{1}{2}\log_2\frac{1-\frac{1}{17}}{\frac{1}{17}} = 2$$

- Compute the distribution D_1. Specify the weight of D_1 on examples on which h_0 made a mistake and the weight on examples on which h_0 was correct.

  Solution:

$$D_0(i) = \frac{1}{17}$$

$$D_1(i) = \begin{cases} \frac{D_0}{Z_0}2^{-\alpha_0} & \text{if } h_0(x_i) = y_i \\ \frac{D_0}{Z_0}2^{\alpha_0} & \text{if } h_0(x_i) \neq y_i \end{cases}$$

$$D_1(i) = \begin{cases} \frac{\frac{1}{17}}{Z_0}2^{-2} = \frac{1}{4\cdot 17\cdot Z_0} & \text{if } h_0(x_i) = y_i \\ \frac{\frac{1}{17}}{Z_0}2^{2} = \frac{4}{17\cdot Z_0} & \text{if } h_0(x_i) \neq y_i \end{cases}$$

$$\frac{16}{4\cdot 17 Z_0} + \frac{4}{17 Z_0} = 1;\ Z_0 = \frac{8}{17}$$

$$D_1(i) = \begin{cases} \frac{1}{32} & \text{if } h_0(x_i) = y_i; \\ \frac{1}{2} & \text{if } h_0(x_i) \neq y_i. \end{cases}$$

(b) [8] Write (and explain your notation)

(1) an expression for the error of an individual AdaBoost weak learner in the ith round.

Solution:
$\epsilon_i = \sum_{x_j | h_i(x_j) \neq y_j} D_i(x_j)$
This is just the first computation you did in part (a).

(2) an expression for the global error of the AdaBoost hypothesis.

Solution:
The expression for global error is:

$$\text{trainingError}(H_{final}) = \frac{1}{m} \sum_i y_i \neq H_{final}(x_i)$$

How does the global error of the hypothesis behave in later rounds of Adaboost?

Solution:
In lecture, it was shown that $\text{trainingError}(H_{final}) \leq \prod_t Z_t$ where $Z_t = 2\sqrt{\epsilon_t(1-\epsilon_t)} < 1$. Thus, as the number of rounds increase, the global error decreases.

How does the weighted training error of the individual weak learners behave in later rounds?

Solution:
Since AdaBoost places greater weight on the difficult examples (the ones which are repeatedly misclassified), the weighted error of later rounds will tend to increase.

Problem 3 [Winnow; VC Dimension]

We would like to learn a function in the class of 3-*term DNF formulas over* $\{x_1, x_2, \ldots x_{100}\}$ using the Winnow algorithm. Recall that the class of 3-term DNF consists of DNF formulae that have three terms, where each one is a conjunction of any number of variables. For example:

$$f = (x_1 \wedge x_3 \wedge x_5 \wedge x_7 \wedge \ldots \wedge x_{99}) \vee (x_2) \vee (x_4).$$

(a) [4] Suggest an *efficient strategy* to use Winnow in order to learn functions in this class. Specifically, what will be the instance space over which you will run Winnow?

Solution:

The class of k-term-DNFs for $k \geq 2$ are not efficiently PAC learnable, unless $P = NP$, as shown by Kearns, Li, Pitt, and Valiant (1987). However, we *have* seen an algorithm for learning k-CNFs, which are a superset of k-term-DNFs. We can efficiently learn the concept by doing our learning in the space of 3-CNFS.

Note, strategies that involved the space of all monomials were not acceptable, since that would blow up the feature space exponentially. Also, unlike perceptron, one cannot use kernel methods with winnow.

(b) [4] For the function f above, provide a possible hypothesis that Winnow might produce given data that is consistent with this function. Make sure that the hypothesis you propose is consistent with the function f.

Solution:

Converting the above concept into a 3-CNF yields:

$$f' = \bigwedge_{i_{odd}} (x_i \vee x_2 \vee x_4)$$

Thus, we will have a weight vector with $\begin{pmatrix} 100 \\ 3 \end{pmatrix}$ elements, with the weights corresponding to the above 50 literals summing to the threshold, and the others zero (or non-zero, but insignificant).

For this particular 3-term-DNF, one can come up with weight vector directly, something like:

$[1, 50, 1, 50, 1, 0, 1, \ldots, 1, 0] \cdot x \geq 50$

Answers of this form were acceptable, as long as it had the correct weights and threshold.

(c) [4] Give an order of magnitude bound on the number of mistakes Winnow will make when learning f.

Solution:

We know from lecture that the bound for winnow is $O(k \log n)$, where k is the number of active literals, and n is the size of the feature space. Thus we will have $O(50 \log \begin{pmatrix} 100 \\ 3 \end{pmatrix}$ mistakes.

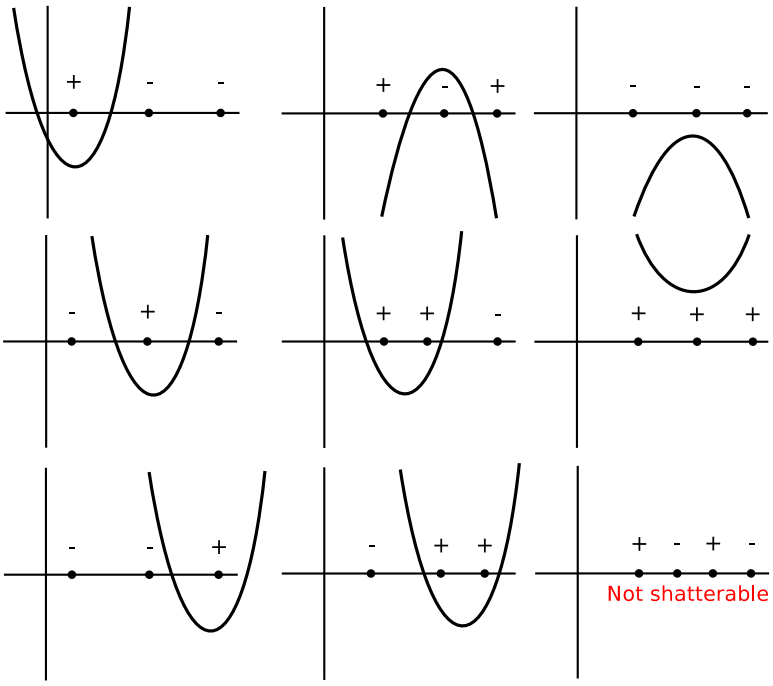
You cannot use the $k \log n$ bound for the second weight vector above though, because it belongs to a different family of functions, of which the analysis is more complicated.

(d) [8] We define a set of concepts $H = \{sgn(ax^2 + bx + c) | a, b, c \in R\}$, where $sgn(z)$ is 1 when z is positive, and 0 otherwise.

What is the VC dimension of H? Prove your claim.

Solution:

The VC dimension is 3. To prove this, we need only show that there is one configuration such that all labelings are shatterable, and that no set of 4 points are shatterable. Note that we are only dealing with points on the x axis (although the VC dimension is still 3 in two dimensions). Any alternating labeling of four points will result in a configuration that is not shatterable because quadratic functions can change signs at most twice.



Problem 4 [On-Line Learning and PAC Learning]

In this problem you will define and explain the difference between two of the learning frameworks we discussed in class and show that, in some sense, one framework is stronger than the other.

(a) [4] You are given an algorithm $\mathcal{B}$ that has a mistake bound of m_b when learning functions in the class C.

Give a precise *definition* of the above statement, that "m_b is the mistake bound of $\mathcal{B}$ when learning functions in C".

Solution:

After $\mathcal{B}$ makes m_b mistakes, $\mathcal{B}$ will have exactly learned the concept C, making no more mistakes.

(b) [4] Give a *definition* of the following statement: The class C is PAC learnable, and $m_p = m(\epsilon, \delta)$ is the number of examples required to guarantee (ϵ, δ) behavior.

Solution:

Given m_p examples which are consistent with a hypothesis h, we are guaranteed with probability $1 - \delta$ that h will have error at most ϵ.

(c) [4] Discuss the difference between m_b and m_p defined above. Specifically, address the fact that m_b and m_p count *different* things. What does each count?

Solution:

m_b counts the number of *mistakes* which must be made to guarantee that no more mistakes will be made. m_p on the otherhand, is the number of *consistent examples* which must be seen to guarantee (ϵ, δ) behavior.

(d) [8] You are given the on-line algorithm $\mathcal{B}$ from (a), with the mistake bound m_b. Design an algorithm $\mathcal{P}$ that uses $\mathcal{B}$ as a subroutine and is a PAC learning algorithm for C, with (ϵ, δ) behavior.

- Explain what is required in order to guarantee (ϵ, δ) behavior (recall (b)).

  Solution:

- Explain how you will use the algorithm $\mathcal{B}$ as a subroutine in the new algorithm, $\mathcal{P}$.

  Solution:

- As a function of m_p and m_b, provide an upper bound on the number of examples that $\mathcal{P}$ needs to use in order to guarantee (ϵ, δ) behavior.

  Solution:

empty page

Problem 5 [Kernel Perceptron and Multi Class Classification]

In this problem you will use the *Kernel Perceptron* algorithm in order to solve the multi-class classification problem, and then analyze its time complexity as a function of the number of examples available and the number of class labels.

Assume you are given a collection of m labeled examples, $\{(x_1, b_1)\}_{i=1}^{m}$, with $b_i \in \{0, 1\}$. Recall that the *Kernel perceptron* algorithm is a learning algorithm that learns perceptron via the example based representation. In the notation used in class, the hypothesis used by the algorithm is:

$$f(x) = \sum_{z \in M} S(z)K(x, z).$$

We will assume a linear kernel, $K(x, z) = x \cdot z$.

(a) [5] Write explicitly the kernel perceptron algorithm defined above. Describe its operation in the *Training* stage and in the *Evaluation (Test)* stage. Write an expression for the worst case time complexity of learning as a function of the number m of examples provided.

Solution:

```
KernelPerceptron(S)
  P ← ∅
  D ← ∅
  foreach x ∈ S
      if label(x, y) = + and Th_θ(Σ_{z∈M} S(z)K(x, z)) = 0
          P ← P ∪ x
      if label(x, y) = − and Th_θ(Σ_{z∈M} S(z)K(x, z)) = 1
          D ← D ∪ x
  return w
```

(b) [5] Consider a multi-class classification problem with k class labels $\{1, 2, \ldots k\}$. As before, we have m examples, only that now they are labeled with one of k class labels; assume that for all i, $1 \leq i \leq k$, m/k of the examples are labeled i.

Suggest a learning model in which you use the Boolean learner discussed above in order to learn a multi-class classifier. Specifically, in this part we will consider the 1-vs-*all* model, in which you learn one classifier for each class label.

- Explain how you train classifiers in this model. Specifically, explain what examples are presented to each classifier and how are they labeled.

  Solution:
  For each class c_i, partion the data with labels such that $(x \in c_i, +), (x \notin c_i, -)$. Train each f_i using the above algorithm.

- Explain how you evaluate classifiers in this model, given new data that is not labeled.

  Solution:
  For each new example x, evaluate $c = \arg\max_i f_i(x)$.

(c) [5] Consider the same multi-class classification problem as in [b] above.
Suggest a *different* learning model which uses the Boolean learner discussed above in order to learn the multi-class classification. Specifically, in this part we will consider the *all*-vs-*all* model, in which you learn one classifier for each *pair* of class labels.

- Explain how you train classifiers in this model. Specifically, explain what examples are presented to each classifier and how are they labeled.

  Solution:
  For each pair of classes c_i, c_j, partion the data such that $(x \in c_i, +), (x \in c_j, -)$. Train each f_{ij}.

- Explain how you evaluate classifiers in this model, given new data that is not labeled.

  Solution:
  Evaluate all classifiers $f_{ij}(x)$, and take a majority vote to determine the class of the example x.

(d) [5] For each of the multi-class classification in [b] and [c] above, write down the worst case time complexity of training the multi-class classifiers, as a function of the number of examples given, m, and the number of classes k.

In each case, take into account the number of Boolean classifiers used, the number of examples each of them is being trained on, and the complexity of training each Boolean classifier, derived in [a].

Derive a conclusion as to which paradigm is better for the algorithm used here, and discuss why.

Solution:

Some formulae you may need:

1. $P(A, B) = P(A|B)P(B)$
2. $Entropy(S) = -p_+ \log p_+ - p_- \log p_-$
3. $Gain(S, A) = Entropy(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} Entropy(S_v)$
4. $\log(ab) = \log(a) + \log(b)$
5. $\log(\frac{a}{b}) = \log(a) - \log(b)$
6. $\log_2(3) \approx 3/2$
7. $0 \log_2(a) = 0$
8. $(\frac{15}{16})^3 > \frac{4}{5}$
9. $M = O(min\{\frac{1}{\epsilon}(\ln |H| + \ln 1/\delta), \frac{1}{\epsilon}(VC(H) + \ln 1/\delta)\}$

AdaBoost

1. $\alpha_t = \frac{1}{2} \log_2 \frac{1-\epsilon}{\epsilon}, \quad \epsilon = \text{error}$
2. $D_{t+1}(i) = \begin{cases} \frac{D_t}{Z_t} 2^{-\alpha_t} & \text{if } h_t(x_i) = y_i \\ \frac{D_t}{Z_t} 2^{\alpha_t} & \text{if } h_t(x_i) \neq y_i \end{cases}, \quad Z_t \text{ is a normalization factor}$