COS 402: Artificial Intelligence

*— written exercises for R&N **second** edition —*

Written exercises W6 Fall 2011
Machine learning Due: Wednesday, January 11

Important note: This version of the written exercises was prepared for those using the ***second*** edition of R&N, and all references below are therefore to that edition. (If you are using the other edition, you need to obtain the other version of these written exercises.)

Approximate point values are given in brackets. Be sure to show your work and justify all of your answers. See the course home page for information on when and where to submit written exercises, and grading criteria.

1. (15) In class, we looked at the following dataset:

x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	y
1	1	0	0	0	1	0	1	1
1	0	0	0	0	0	0	0	0
0	0	0	1	1	1	1	0	0
0	1	0	0	1	0	0	0	0
1	0	0	0	0	1	0	1	0
0	1	1	0	1	1	0	1	1
1	1	0	1	0	1	0	1	1
0	1	0	1	0	1	0	0	1
0	0	0	0	0	0	1	1	0

In this formulation, there are eight attributes (or features or dimensions), $x_1, \ldots, x_8$, each taking the values 0 or 1. The label (or class) is given in the last column denoted y; it also takes the values 0 or 1. In class, it was noticed that the label y is 1 if and only if x_2 and x_6 are both equal to 1. Since attributes and labels are $\{0, 1\}$-valued, we can write this rule succinctly as $y = x_2 x_6$. In general, such a product of any number of attributes is called a *monomial*. (This includes the "empty" monomial, which, being a product of no variables, is always equal to 1.)

Throughout this problem, you can assume that the attributes and labels are all $\{0, 1\}$-valued. Also, let n be the number of attributes (for instance, $n = 8$ in the example above).

a. Describe a simple algorithm that, given a dataset, will efficiently (in polynomial time) find a monomial consistent with it, assuming that one exists.

b. What is the total number of monomials that can be defined on n attributes?

c. Suppose you applied your algorithm to the dataset above, and that a consistent monomial was found. Use the bound derived in class (or the results in R&N) to compute an upper bound on the generalization error of this monomial. ("Generalization error" is the same as what R&N calls simply the "error" or "error rate" in Section 18.5.) Derive a bound that holds with 95% confidence (so that $\delta = 0.05$).

d. Continuing the last question in which your algorithm is applied to data with $n = 8$ attributes, how many training examples would be needed to be sure the generalization error of a consistent monomial is at most 10% with 95% confidence?

2. (32) Consider the following dataset consisting of five training examples followed by three test examples:

x_1	x_2	x_3	y
training			
$-$	$+$	$+$	$-$
$+$	$+$	$+$	$+$
$-$	$+$	$-$	$+$
$-$	$-$	$+$	$-$
$+$	$+$	$-$	$+$
test			
$+$	$-$	$-$	?
$-$	$-$	$-$	?
$+$	$-$	$+$	?

There are three attributes (or features or dimensions), x_1, x_2 and x_3, taking the values $+$ and $-$. The label (or class) is given in the last column denoted y; it also takes the two values $+$ and $-$.

Simulate each of the following four learning algorithms on this dataset. In each case, show the final hypothesis that is induced, and show how it was computed. Also, say what its prediction would be on the three test examples.

For parts b, c and d, be sure to see the errata for R&N Chapters 18 and 20 below.

a. The *decision tree algorithm* discussed in class and R&N. For this algorithm, use the information gain (entropy) impurity measure as a criterion for choosing an attribute to split on. Grow your tree until all nodes are pure, but do not attempt to prune the tree.

b. *AdaBoost*. For this algorithm, you should interpret label values of $+$ and $-$ as the real numbers $+1$ and -1. Use decision stumps as weak hypotheses, and assume that the weak learner always computes the decision stump with minimum error on the training set weighted by D_t. (Recall that a decision stump is a one-level decision tree; see R&N p. 666.) Run your boosting algorithm for three rounds.

c. *Support vector machines*. For this algorithm, you should interpret both label and attribute values of $+$ and $-$ as the real numbers $+1$ and -1. Also, you can use the additional information that the first three examples are support vectors, but the others are not, so that α_4 and α_5 are both zero in R&N Eq. (20.17). This means that you can maximize this equation over α_1, α_2 and α_3 using calculus. (Note that if any of these variables turn out to be negative, there's a problem.) When you have found a solution vector $\mathbf{w}$, check it by showing that, for each labeled training example $(\mathbf{x}, y)$, we have $y(\mathbf{w} \cdot \mathbf{x}) \geq 1$, and that equality holds for the support vectors, i.e., the first three examples. (The notation here is as in class and R&N.) You do not need to use a "kernel," just a regular inner product, as in Eqs. (20.17) and (20.18).

d. *Neural networks*. For this algorithm, use a single-layer neural net consisting of just a single perceptron at the output, no hidden layers, and the three features at the input level. Attribute values of $+$ and $-$ should be interpreted as the real numbers $+1$ and -1, while label values of $+$ and $-$ should be interpreted as 1 and 0. You can disregard the "bias weight" (denoted $W_{0,i}$ in R&N Figure 20.15), i.e., assume it is fixed to be zero. Assume that the neural net is trained for a single "epoch" that runs through the training data once in the order given. Use a learning rate of $\alpha = 0.1$, and start with all weights equal to zero. For g, use the standard sigmoid function given in Figure 20.16(b).

Errata for R&N Chapters 18 and 20

There are a few important errors in R&N.

First of all, in Figure 18.10, the second to last line is written ambiguously. It should read:

$$\mathbf{z}[m] \leftarrow \log[(1 - \mathit{error})/\mathit{error}].$$

(Actually, however, I would encourage you to use the pseudocode and notation for AdaBoost given in class and as a handout on the "Schedule & Readings" webpage.)

Secondly, the equation second from the bottom on page 741 that now reads:

$$= \mathit{Err} \times \frac{\partial}{\partial W_j} g\left(y - \sum_{j=0}^{n} W_j x_j\right)$$

should instead read:

$$= \mathit{Err} \times \frac{\partial}{\partial W_j} \left(y - g\left(\sum_{j=0}^{n} W_j x_j\right)\right).$$

(Note: this error was corrected in some later printings of the book.)

Finally, the paragraph describing SVM's at the very bottom of page 749 continuing at the top of 751 is not quite correct, but some explanation is required to describe what the problem is. In class, we implicitly required the hyperplane sought by the SVM algorithm to pass through the origin. This resulted in a hypothesis of the form

$$\mathrm{sign}(\mathbf{w} \cdot \mathbf{x}).$$

In other treatments of SVM's, however, the hyperplane is often *not* required to pass through the origin. Thus, the computed hypothesis has the form

$$\mathrm{sign}(\mathbf{w} \cdot \mathbf{x} + b),$$

so that the hyperplane is defined both by the vector $\mathbf{w}$ and the scalar b.

The treatment in R&N is not quite correct for either of these cases. For the through-the-origin case, their treatment would be correct if the constraint $\sum_i \alpha_i y_i = 0$ were omitted. With the omission of this constraint, their treatment is the same as was presented in class. For the not-through-the-origin case, the treatment in R&N would be correct if Eq. (20.18) were replaced by

$$h(\mathbf{x}) = \mathrm{sign}\left(\sum_i \alpha_i y_i (\mathbf{x} \cdot \mathbf{x}_i) + b\right),$$

for some b that can be written in terms of the other variables (details omitted).

For this class (including part 2c above), we will only consider the through-the-origin case.