Assignment 4 - Machine Learning [*]

Anirudh Narasimhamurthy(u0941400)

April 1, 2015

1 Margins

1.1

We are asked to find w and θ which will represent f as a linear threshold function. The representation for f as a linear threshold function is given as follows:

$x_2 + x_4 + x_6 + x_{10} + x_{12} + x_{14} + x_{16} + x_{18} \geq 1$

This is because we are given a disjunction and any one of the variables need to be 1 for the label to be +.

The w and θ are given by:

$w = [0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0]$
$\theta = 0.5$

1.2

If θ=0.5, Margin γ would be maximum.

The margin is given by:

$$\gamma = \min_i \left| \frac{\mathbf{w} \cdot \mathbf{x}_i - \theta}{||\mathbf{w}||} \right|$$

For Positive examples, under the given setting only one of the relevant variables is set to 1, and six irrelevant variables are set to 1. Norm is equivalent to norm of the relevant variables. In our case we have 8 relevant variables. Therefore

γ=(1-0.5)/$\sqrt{8}$=0.5 /2.828 = 0.1767

Negative examples, under the given setting consist of none of the relevant variables and six irrelevant variables are set to 1. All others are set to 0.

$\gamma = |(0 - 0.5)|/\sqrt{8}$=0.1767

[*]CS 6350 ; Spring 2015

1.3

For positive examples, we have six relevant variables set to 1 in addition to six irreleveant variables. Therefore we would have w.x as 6.

If the hyper parameter θ is to be between 0 and 1 then choosing θ to be 0.5 would give us the optimal margin.

For Positive examples, γ=(6-0.5)/$\sqrt{8}$=1.944
For Negative examples, $\gamma = |(0 - 0.5)|/\sqrt{8}$=0.1767

If hyper paramter can be greater than 1, then having θ=3, Margin γ would be optimal.

For Positive examples γ=(6-3)/$\sqrt{8}$=1.0606
For Negative examples $\gamma = |(0 - 3)|/\sqrt{8}$=1.0606

1.4

Increasing the number of irrelevant variables seen to 10 would not have any impact on the computation of γ as the dot product of w and x would still be the same.

If θ=0.5, Margin γ would be maximum.

For Positive examples, γ=(1-0.5)/$\sqrt{8}$=0.1767
For Negative examples, γ=(0-0.5)/$\sqrt{8}$=0.1767

The perceptron mistake bound for D_1, D_2, D_3 is given by :

Mistake bound= R^2/γ^2

For D1 for positive examples we have one relevant variable and six irrelevant variables. For D2 for positive examples, we have six relevant variables and six irrelevant variables. For D3 for positive examples we have one relevant variable and 10 irrelevant variables.

Perceptron mistake bound for $D1 = sqrt(7)^2/0.1767^2 = 224.19$
Perceptron mistake bound for $D2 = sqrt(12)^2/0.1767^2 = 384.33$
Perceptron mistake bound for $D3 = sqrt(11)^2/0.1767^2 = 352.30$

1.5

Mistake bound is the number of examples that are needed to learn the model.

Ease of learning would be proportionate to the lesser number of mistakes made which we would be needed to learn.

The order would be D1, D3 and D2 in decreasing order of ease of learning.

The order is D2, D3, D1 in increasing order of ease of learning since more examples are needed to be train for D2

2 Boosting and Perceptron

2.1

From the training examples provided in the question, assume h1(x,y) as below

$$h1(x,y) = \begin{cases} +1 & \text{x} \geq 6 \\ -1 & \text{x} \leq 6 \end{cases}$$

$$h2(x,y) = \begin{cases} +1 & \text{y} \geq 9 \\ -1 & \text{y} \leq 9 \end{cases}$$

The table below shows the answers required for both parts 2.1 and 2.3 as the last column of perceptron updates corresponds to 2.3 We are given to start with (0,0). Since the threshold is 3 we start with w=(0, 0, -3). The learning rate is 1.

For computing D_t, initially all examples are equally important. We therefore detemine D_0 by using the expression

		Boosting				**Perceptron updates**
		Hypothesis 1		Hypothesis 2		
i	Label	D_0		D_1		Start with $w = (0, 0, -3)$
1	-	0.1	-	0.0625	+	No update to weight vector w
2	-	0.1	-	0.0625	-	No update to weight vector w
3	+	0.1	+	0.0625	-	$(1, -1, -2)$
4	-	0.1	-	0.0625	-	No update to weight vector w
5	-	0.1	-	0.0625	+	No update to weight vector w
6	+	0.1	+	0.0625	-	No update to weight vector w
7	+	0.1	+	0.0625	+	$(2, 0, -1)$
8	-	0.1	-	0.0625	-	No update to weight vector w
9	+	0.1	-	0.25	+	$(1, 1, 0)$
10	-	0.1	+	0.25	-	$(2, 1, 1)$

Table 1: Table for Boosting and Perceptron

2.2

To determine the final hypothesis, we have to find ϵ_0 and ϵ_1. We know ϵ_t is given by :

$\epsilon_t = 1/2 - 1/2(\sum_{i=1}^{m} D_t(i) y_i \cdot h(x_i))$

and $\alpha_t = \frac{1}{2} \cdot ln(\frac{1-\epsilon_t}{\epsilon_t})$

In our case m=10 and we made two mistakes in the first run and hence

ϵ_0=0.2

$\alpha_0 = \frac{1}{2} \cdot ln(\frac{0.8}{0.2}) = 0.6931$

ϵ_1=0.4

$\alpha_1 = \frac{1}{2} \cdot ln(\frac{0.6}{0.4}) = 0.8109$

$\boxed{H_{final}(x,y) = sgn(0.6931 * h1(x,y) + 0.8109 * h2(x,y))}$

2.3

Final value of b=1 and the final weight vector at the end of 10 examples seen is (2,1,1) where the last term is the bias term

$\boxed{H_{final}(x,y) = sgn(2*x+1*y+1)}$

2.4

Thus from 2.2 and 2.3 we observe that the two algorithms did not converge to the same hypothesis. Hence if these two algorithms were used to predict labels for the training set, the **set of predictions would not be the same**.

This is because the Ada boost performs the prediction by down weighting the correctly predicted examples in every run and upweights the incorrectly predicted examples and so improves its prediction very well.

On the other hand, the perceptron update is slightly different as the weight vector gets updated only for an incorrectly predicted example.

3 Kernels

3.1

As per the definition provided in the class lecture slides, A function K(x,z) is a valid kernel if it corresponds to an inner product in some feature space.

To prove K(x,z) is a kernel function we can construct the gram matrix and then prove it is a kernel function if we can prove the gram matrix is positive semi definite.

Gram matrix of a set of n vectors is given by $G_{ij} = x_i^T x_j$

We are given k1(x,z) and k2(x,z) are kernels.

We know that a new kernel can be constructed from existing ones if we multiply the existing kernel by a constant 'c'. By that property,
αk1(x,z) and βk2(x,z) are also kernels.

Again by adding two kernels, we get a kernel. That is
$k'(x,x') = k_1(x,x') + k_2(x,x')$
So, $\alpha k1(x,z) + \beta k2(x,z)$ is also a kernel

Hence $K(x,z) = \alpha K_1(x,z) + \beta K_2(x,z)$ is a kernel function

3.2

We are given $x \in R^2$ and $z \in R^2$ and
$K(x,z) = (x^Tz)^3 + 9(x^Tz)^2 + 81x^Tz$

In order to prove it is a kernel, we can do as follows:

Taking x^Tz common, we have

$= x^T z[(x^T z)^2 + 9(x^T z) + 81]$

We know that $(x^T z)$ and $(x^T z)^2$ are kernel functions.
$= k1(x,z)[k2(x,z) + k3(x,z) + 81]$

We also know, multiplying or adding two kernels gives a kernel as well
$= k4(x,z)$
Therefore the given function is a valid kernel function. Hence proved.

4 Question 4

4.1

A k-decision list over the variables $x_1, ...x_n$ is an ordered sequence $L = (c_1, b_1), ...(c_l, b_l)$. The bit b is called the default value and b_i is referred to as the bit associated with condition c_i.

To prove that if a concept 'c' can be represented as a k-decision list, so can its complement $\neg c$, we can consider the following:
$c = \{(c_1, b_1), (c_2, b_2)...(c_l, b_l), b\}$

The outputs of the list can be negated to find the $\neg c$

$\neg c = \{(c_1, \neg b_1), (c_2, \neg b_2)...(c_l, \neg b_l), \neg b\}$
So, $\neg c$ can also be represented by a decision list.

4.2

Considering the case of possible conjunctions with k literals, we have n^k possible conjunctions with k literals in each of the n decision nodes.
Each of these n decision nodes can be arranged in $(n^k)!$ different ways.

From this size of our hypothesis space would be given by : $|H| = 3^{n^k} * (n^k)!$
By PAC learning taking log on both sides,
$log|H| = n^k log(3) * log(n^k)$
Hence the hypothesis space is of polynomial size complexity.
The k-decision list can be learnt by an algorithm in polynomial time since there wouldn't be any simpler explanation than a decision list to represent such hypothesis.

Thus by Occam's razor, we have proved the class of k-decision lists is PAC- learnable.

4.3

Assume the concept c as $c = (c_1, b_1), (c_2, b_2)...(c_l, b_l), b$

Going by the hint provided in the question, if we choose the weight vector as follows:
$w = [a^l * b_1, a^{l-1} * b_2, ..., 1 * b_l]$
Then the weight vector would give preference to the first decision bit b1,
y=sgn($(a^l) * c_1 * b_1 + (a^{l-1}) * c_2 * b_2 + +c_l * b_l$)
If c1 is true, the dot product wx product would have given more preference to the first decision output b1.

Hence the 1-decision lists are linearly separable functions. Hence proved.