

**Question 1.**

**part a.** Assume that we have a procedure PARALLEL-SORT that sorts a series of parallel arrays by the value in the first array. Our two other “helper” functions are

```

COMPRESSION-DATA(X, Y, W)
1  neg ← 0
2  pos ← 0
3  for i ← 2 to X.length
4  do if X[i] ≠ X[i - 1]
5     then append X[i - 1] to X'
6         append neg to W-
7         append pos to W+
8         neg ← 0
9         pos ← 0
10  if Y[1] < 0
11     then neg ← neg + W[i]
12     else pos ← pos + W[i]
13  append X[i - 1] to X'
14  append neg to W-
15  append pos to W+
16  return (X', W-, W+)

PICK-SPLIT(X, Y, W)
1  (X, Y, W) ← PARALLEL-SORT(X, Y, W)
2  (X, W-, W+) ← COMPRESSION-DATA(X, Y, W)
3  thresh ← X[1] - 1.0
4  negleft ← 0
5  posleft ← 0
6  negright ← sum of W-
7  posright ← sum of W+
8  if negright > posright
9     then sign = -1
10         loss = posright
11     else sign = +1
12         loss = negright
13  for i ← 2 to X.length - 1
14  do posleft ← posleft + W + [i]
15     negleft ← negleft + W - [i]
16     posright ← posright - W + [i]
17     negright ← negright - W - [i]
18  if negleft + posright < loss
19     then thresh = (X[i] + X[i])/2
20         loss = negleft + posright
21         sign = -1
22  if posleft + negright < loss
23     then thresh = (X[i] + X[i])/2
24         loss = posleft + negright
25         sign = +1
26  return (loss, thresh, sign)

```

Which leads to a relatively straight-forward main loop:

```

TRAIN-SIMPLE-CLASSIFIER(X, Y, W)
1  (loss, thresh, sign) ← PICK-SPLIT(X.1, Y, W)
2  d = 0
3  for cd ← 2 to # dimensions
4  do (closs, cthresh, csign) ← PICK-SPLIT(X.cd, Y, W);
5     if closs < loss
6         then loss ← closs
7             thresh = ctthresh
8             sign = csign
9             d = cd
10
11
12  return (d, thresh, sign)

```

COMPRESS-DATA runs in  $O(m)$  time. PARALLEL-SORT runs in  $O(m \log m)$  time. Therefore PICK-SPLIT runs in  $O(m \log m)$  time. (The sort call takes the longest). TRAIN-SIMPLE-CLASSIFIER calls PICK-SPLIT  $d$  times. So, it runs on  $O(dm \log m)$  time.

**part b.**

	0	1	2	3	4	5	6	7	8	9
0		0.9889	0.9746	0.9501	0.9582	0.9500	0.9833	0.9944	0.9830	0.9832
1	0.9889		0.8802	0.9425	0.8457	0.9176	0.9366	0.9335	0.8146	0.8536
2	0.9746	0.8802		0.9000	0.9525	0.8997	0.9637	0.9860	0.8291	0.9328
3	0.9501	0.9425	0.9000		0.9643	0.9041	0.9643	0.9530	0.8543	0.8237
4	0.9582	0.8457	0.9525	0.9643		0.9421	0.9696	0.8500	0.8479	0.9280
5	0.9500	0.9176	0.8997	0.9041	0.9421		0.9642	0.8283	0.8652	0.9669
6	0.9833	0.9366	0.9637	0.9643	0.9696	0.9642		0.9694	0.9775	0.9861
7	0.9944	0.9335	0.9860	0.9530	0.8500	0.8283	0.9694		0.9235	0.9248
8	0.9830	0.8146	0.8291	0.8543	0.8479	0.8652	0.9775	0.9235		0.9011
9	0.9832	0.8536	0.9328	0.8237	0.9280	0.9669	0.9861	0.9248	0.9011	

**part c.**

Testing accuracy is 77.74%.

	0	1	2	3	4	5	6	7	8	9
0	171	0	0	0	2	2	0	0	3	0
1	1	137	9	6	1	3	2	0	15	8
2	1	3	129	10	0	1	5	0	26	2
3	2	6	6	129	0	4	0	3	21	12
4	2	20	0	0	142	5	2	6	4	0
5	0	0	0	6	1	160	11	0	3	1
6	1	2	0	1	6	0	169	0	2	0
7	0	3	0	8	8	18	1	128	11	2
8	1	17	9	7	11	13	0	1	114	1
9	0	3	1	24	9	16	0	1	8	118

**part d.**

The exponential loss (or risk) is

$$\begin{aligned} L &= \sum_i e^{-m_i} \\ &= \sum_i e^{-y_i s(x_i) - y_i \alpha' g'(x_i)} \\ &= \sum_i l_i e^{-\alpha' y_i g'(x_i)} \end{aligned}$$

if we define  $l_i$  to be  $e^{-y_i s(x_i)}$  (the exponential loss for point  $i$  before adding this new classifier). Differentiating with respect to  $\alpha'$  yields

$$\begin{aligned} -\frac{\partial L}{\partial \alpha'} &= \sum_i l_i y_i g'(x_i) e^{-\alpha' y_i g'(x_i)} \\ &= \sum_{i \in G_+} l_i e^{-\alpha'} - \sum_{i \in G_-} l_i e^{\alpha'} \end{aligned}$$

where  $G_+$  is the set of indexes of the points  $g'$  correctly classifies and  $G_-$  are the rest of the indexes. (Note that for  $i \in G_+$ ,  $y_i g'(x_i) = +1$ .) We can take this expression and set it equal to zero to find the minimum. If we let  $L_+ = \sum_{i \in G_+} l_i$  and let  $L_-$  be similarly defined the result is

$$\begin{aligned} L_- e^{\alpha'} &= L_+ e^{-\alpha'} \\ e^{2\alpha'} &= \frac{L_+}{L_-} \\ \alpha' &= \frac{1}{2} \ln \frac{L_+}{L_-} . \end{aligned}$$

**part e.**

First, let  $L = L_+ + L_-$ .  $L$  is a constant that does not depend on  $g'$  (it is the total margin of all points, regardless of how  $g'$  classifies them). Then we can derive that

$$\begin{aligned} \arg \min_{g'} \sum_i l_i e^{-\alpha' y_i g'(x_i)} &= \arg \min_{g'} L_+ e^{-\alpha'} + L_- e^{\alpha'} \\ &= \arg \min_{g'} L_+ \sqrt{\frac{L_-}{L_+}} + L_- \sqrt{\frac{L_+}{L_-}} \\ &= \arg \min_{g'} 2\sqrt{L_+ L_-} \\ &= \arg \min_{g'} L_+ L_- \\ &= \arg \min_{g'} (L - L_-) L_- \\ &= \arg \min_{g'} L_- \end{aligned}$$

The last line is slightly tricky.  $L_-(L-L_-)$  is a quadratic function of  $L_-$  with a maximum at  $L_- = L/2$  and symmetric minima at  $L_- = 0$  and  $L_- = L$ . Any  $g'$  which results in  $L_- > L/2$  will result in an  $\alpha' < 0$ . We can flip the sign of the  $g'$  and the sign of  $\alpha'$  and end up with an equivalent  $g'$  and  $\alpha'$  such that the later is positive and  $L_-$  for the former is less than  $L/2$ . Therefore, we do not need to consider the case where  $L_- > L/2$ . Therefore, the smaller  $L_-$ , the smaller the value of  $(L - L_-)L_-$  becomes.

But notice that

$$L_- = \sum_{i \in G_-} l_i$$

which is exactly the empirical error (the sum over the incorrect data examples) where each data sample is weighted by  $l_i$ . So, our weighted empirical risk minimization problem is one in which each data point is weighted by  $l_i = e^{-y_i s(x_i)}$ .

**part f.**

So, we will construct an algorithm to keep adding new simple classifiers and weights to our “complex” classifier until the exponential risk stop improving. We will never get a new  $\alpha' < 0$  because we could do better by switching the sign of our classifier. Therefore, all  $\alpha$  values will be non-negative. If our new  $g'$  has weighted empirical risk of 0.5 (or half of the total weight if the weight doesn't sum to 1), then the resulting  $\alpha'$  will be 0. That means that  $s(x)$  will not change and so the next loop through the algorithm will have the same result. Hence, from this point forward, nothing new will happen. So, we can terminate whenever the empirical risk of our new classifier is greater than or equal to 0.5.

The algorithm looks like

TRAIN-COMPLEX-CLASSIFIER( $X, Y$ )

```

1   $W \leftarrow$  array of 1.0 of the same size as  $X$ 
2   $D \leftarrow$  empty list
3   $T \leftarrow$  empty list
4   $S \leftarrow$  empty list
5  repeat
6      Normalize  $W$  (scale each element by the same amount so that it sums to 1)
7       $(d, thresh, sign) \leftarrow$  TRAIN-SIMPLE-CLASSIFIER( $X, Y, W$ )
8       $L_+ \leftarrow 0$ 
9      for  $i \leftarrow 1$  to  $X.length$ 
10     do  $G[i] \leftarrow$  EVALUATE-SIMPLE-CLASSIFIER( $X[i], d, thresh, sign$ )
11         if  $G[i] = Y[i]$ 
12             then  $L_+ \leftarrow L_+ + W[i]$ 
13
14     if  $L_+ > 0.5$ 
15         then  $\beta \leftarrow \frac{L_+}{1.0 - L_+}$ 
16             for  $i \leftarrow 1$  to  $X.length$ 
17                 do if  $G[i] = Y[i]$ 
18                     then  $W[i] \leftarrow W[i]/\beta$ 
19                 append  $\ln(\beta)$  to  $W$ 
20                 append  $d$  to  $D$ 
21                 append  $t$  to  $T$ 
22                 append  $s$  to  $S$ 
23
24     until  $L_+ \leq 0.5$ 
25 return  $(W, D, T, S)$ 

```

Note that I multiply the good examples by  $1/\beta$  instead of multiplying the good examples by  $\sqrt{1/\beta}$  and the misclassified examples by  $\sqrt{\beta}$ . I'm going to renormalize, so the two are equivalent. Additionally, I add in the new  $\alpha$  value as  $\ln(\beta/(1 - \beta))$  instead of one-half that value. The end classifier does not change.

**part g.**

Limiting ourselves to 100 rounds of boosting we get:

2-way classification:

	0	1	2	3	4	5	6	7	8	9
0		0.9972	0.9972	0.9945	0.9972	0.9917	0.9944	1.0000	1.0000	0.9944
1	0.9972		0.9916	1.0000	0.9890	0.9918	0.9862	0.9972	0.9719	0.9917
2	0.9972	0.9916		0.9944	0.9972	1.0000	0.9832	1.0000	0.9972	1.0000
3	0.9945	1.0000	0.9944		0.9973	0.9863	0.9863	0.9807	0.9692	0.9780
4	0.9972	0.9890	0.9972	0.9973		0.9972	0.9917	0.9972	0.9831	0.9917
5	0.9917	0.9918	1.0000	0.9863	0.9972		0.9945	0.9806	0.9747	0.9779
6	0.9944	0.9862	0.9832	0.9863	0.9917	0.9945		0.9972	0.9859	1.0000
7	1.0000	0.9972	1.0000	0.9807	0.9972	0.9806	0.9972		0.9915	0.9582
8	1.0000	0.9719	0.9972	0.9692	0.9831	0.9747	0.9859	0.9915		0.9661
9	0.9944	0.9917	1.0000	0.9780	0.9917	0.9779	1.0000	0.9582	0.9661	

Testing accuracy is 95.16%.

Confusion matrix:

	0	1	2	3	4	5	6	7	8	9
0	176	0	0	0	1	1	0	0	0	0
1	0	179	0	0	0	1	0	0	1	1
2	0	4	173	0	0	0	0	0	0	0
3	2	0	1	169	0	1	0	1	3	6
4	0	2	0	0	176	0	0	0	2	1
5	0	0	0	0	1	177	1	0	0	3
6	1	4	0	0	1	0	174	0	1	0
7	0	0	0	0	1	3	0	163	1	11
8	0	8	0	1	0	1	2	1	154	7
9	0	1	0	3	1	4	0	0	2	169