

**Exercise 2.1**

i) For each starting location in  $1, \dots, k$ , sum the number of intervals starting from it. This results in

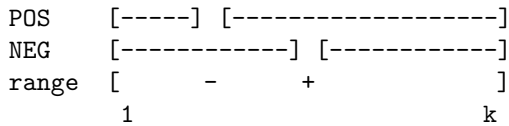
$$k + (k - 1) + (k - 2) + \dots + 1 = \sum_{i=1}^k i = k * (k + 1) / 2 \approx O(k^2),$$

the well-known arithmetic series. The result can also be seen as the number of all unique pairs  $k(k - 1) / 2$  incremented with  $k$  more cases where each number is *paired with itself*.

ii) First, note that we can not represent and update the intervals explicitly by brute force, we don't have enough time for that. One attempt is as follows.

1. Create two "abstract" intervals *POS* and *NEG* that are both initially  $[1, k]$ .
2. Interpret that each of the two intervals covers all of the possible subintervals in its range.
3. On receiving a new example  $x \in [1, k]$ , find out if it belongs to *POS*, *NEG* or both. This can be done in  $O(k)$ . For intervals  $x$  belongs to (maximally two), calculate how many of its subintervals overlap the location. If the interval is  $[a, b]$ , the overlap can be calculated atleast in time  $O(b - a) \leq O(k)$  by traversal.
4. Vote the class that got more weight
5. If there was a nonconforming "abstract" interval  $[a, b]$  covering  $x$ , split it to  $[a, x - 1]$  and  $[x + 1, b]$  with pathological cases handled by common sense. In subsequent rounds, *POS* and *NEG* are sets of "abstract" intervals, but neither can have more than  $O(k)$  components and in  $O()$  sense the running time is not affected.
6. Read next example, repeat from 3.

Visualization of the algorithm state after receiving one positive and one negative example from the range  $[1, k]$ :



This algorithm, although not the cleanest one possible, is relatively easy to understand and can operate in  $O(k)$ . It is a halving algorithm, as each time an example is received, all nonconforming intervals are removed. Unfortunately, the hypothesis class supported by this algorithm (set of intervals) is larger than the one asked for in the exercise. A more conformant solution delivering just one  $[a, b]$  interval would start with a  $h = [1, k]$  interval and truncate it each time a nonconforming example is seen; here we would need to upkeep a couple of arrays to calculate the voting strengths. That would be  $O(k)$  algorithm as well. Details omitted. □

**Exercise 2.2**

Jensen: if  $f$  is convex, and  $x$  is a random variable,

$$Ef(x) \geq f(Ex), \text{ or, } \sum p_i f(x_i) \geq f(\sum p_i x_i)$$

For this exercise, lets take  $f(x) = -\ln x$ . We should know from function analysis that if  $f$  is twice differentiable and  $f''(x) \geq 0$ ,  $f$  is convex at  $x$ . As  $\frac{\delta}{\delta^2 x} - \ln x = \frac{1}{x^2} \geq 0$  everywhere,  $-\ln x$  is convex.

a) Let  $p_i = 1/n$ . Start from an expression where Jensen holds and crank,

$$-\sum 1/n \ln a_i \geq -\ln(\sum 1/n a_i) \tag{3}$$

$$\sum 1/n \ln a_i \leq \ln(1/n \sum a_i) \tag{4}$$

$$\exp(1/n \sum \ln a_i) \leq \exp(\ln(1/n \sum a_i)) \tag{5}$$

$$\prod \exp(1/n \ln a_i) \leq \exp(\ln(1/n \sum a_i)) \tag{6}$$

$$(\prod \ln a_i)^{1/n} \leq 1/n \sum a_i. \tag{7}$$

b) Let  $d_{kl}(p, q) = \sum p_i \ln p_i/q_i$ , the *Kullback-Leibler divergence*. Is  $d_{kl}(p, q) \geq 0, \forall q, p$ ?

Apply Jensen for concave  $\ln(x)$  (just flip the inequality direction).

$$-\sum p_i \ln p_i/q_i = \sum p_i \ln q_i/p_i \tag{8}$$

$$\leq \ln \sum p_i q_i/p_i \tag{9}$$

$$= \ln \sum q_i = \ln 1 = 0 \tag{10}$$

Hence,  $\sum p_i \ln p_i/q_i \geq 0$ .

**Exercise 2.3**

a) We have the estimate of theorem 2.9 of the slides,

$$L(WA) \leq c \ln W_1 - c \ln W_{T+1},$$

where  $W_t = \sum_i W_{t,i}$ . Now we can bound the second term on the right more efficiently. Let  $I$  be the set of those  $k$  experts with losses bounded by  $M$ .

$$-\ln W_{T+1} \leq -\ln \sum_{i \in I} w_{T+1,i} \tag{11}$$

$$\leq \min_{i \in I} (-\ln k w_{1,i} \exp(-\eta L(\epsilon_i))) \tag{12}$$

$$= -\ln k - \min_{i \in I} (-\ln w_{1,i} \exp(-\eta L(\epsilon_i))) \tag{13}$$

$$= -\ln k + \min_{i \in I} \eta L(\epsilon_i) \tag{14}$$

plugging this to the  $L(WA)$  bound, we get

$$L(WA) \leq c \ln W_1 - c \ln k + c \min_{i \in I} \eta L(\epsilon_i) \quad (15)$$

$$= c \ln w_1/k + c \min_{i \in I} \eta L(\epsilon_i) \quad (16)$$

$$\leq c\eta M + c \ln N/k. \quad (17)$$

Explanation: First we applied the definition of  $W_T$ , then reasoned that for decreasing  $-\ln(x)$ , less  $x$  is *more*. Next, we used the update rule of the algorithm to expand  $w_{T+1,i}$  and used the min expert value  $k$  times. With some ln cranking, we got the estimate for the second term that we subsequently plugged in.

We can see from the bound that it can be beneficial to have loss-bounded, preferably good experts.

**b)** Just switch the  $w_i$  to  $p_i$  and do quite similarly to a).

$$-\ln W_{T+1} \leq \min_i (-\ln w_{1,i} \exp(-\eta L(\epsilon_i))) \quad (18)$$

$$= \min_i (-\ln p_i - \ln \exp(-\eta L(\epsilon_i))) \quad (19)$$

$$= \min_i (-\ln p_i + \eta L(\epsilon_i)). \quad (20)$$

Again, plug to the loss bound to get

$$L(WA) \leq c \ln W_1 + c(\min_i (-\ln p_i + \eta L(\epsilon_i))) \quad (21)$$

$$= 0 + \min_i (c \ln 1/p_i + c\eta L(\epsilon_i)), \quad (22)$$

as the initial potential  $W_1 = 1$ . We can see from the bound that it benefits the algorithm if we can give it a vector of good starting weights. These weights could then be interpreted as our prior beliefs of the expert performances.

**Exercise 2.4**

Start by simplifying the log loss

$$L_{\log}(y, \hat{y}) = \frac{1-y}{2} \ln \frac{1-y}{1-\hat{y}} + \frac{1+y}{2} \ln \frac{1+y}{1+\hat{y}}$$

for the two cases  $\{-1, 1\}$ ,

$$L(-1, \hat{y}) = -\ln \frac{1-\hat{y}}{2} \quad (23)$$

$$L(1, \hat{y}) = \ln \frac{1+\hat{y}}{2} \quad (24)$$

which allows us to simplify the exp terms (with  $\eta = 1$ ) as

$$\exp(-L(-1, \hat{y})) = \exp\left(\ln \frac{1-\hat{y}}{2}\right) = \frac{1-\hat{y}}{2} \quad (25)$$

$$\exp(-L(1, \hat{y})) = \exp\left(\ln \frac{1+\hat{y}}{2}\right) = \frac{1+\hat{y}}{2} \quad (26)$$

Now,

$$\ln \frac{W_t}{W_{t+1}} = -\ln \frac{W_{t+1}}{W_t} = -\ln \frac{\sum_i w_{t+1,i}}{W_t} \quad (27)$$

$$= -\ln \sum_i \frac{w_{t,i} \exp(-L(y_t, x_{t,i}))}{W_t} \quad (28)$$

$$= -\ln \sum_i v_{t,i} \exp(-L(y_t, x_{t,i})), \quad (29)$$

where the main idea was to apply the update rule of the WA algorithm. Finally, inputting the exponentiated losses derived earlier, we can get the log losses, shown below for the case  $y_t = 1$ .

$$y_t = -1 : -\ln \sum_i v_{t,i} \frac{1-x_{t,i}}{2} = -\ln\left(\frac{\sum_i v_{t,i} - \mathbf{v}_t \cdot \mathbf{x}_t}{2}\right) \quad (30)$$

$$= -\ln \frac{1 - \mathbf{v}_t \cdot \mathbf{x}_t}{2} \quad (31)$$

$$= L_{\log}(-1, \mathbf{v}_t \cdot \mathbf{x}_t) \quad (32)$$

The other case is handled similarly.  $\square$

### Exercise 2.5

Tedious, brute-force differentiations, such as in this exercise, might be better done on a computer, if you already know how to differentiate. If you don't, this exercise might not be the best place to start learning it.

Suitable computer software to attempt symbolic computation are e.g. Maple (which is commercial) and Maxima (GPL, can be downloaded from Sourceforge).

Below we show how Maple can be used to help solving this exercise.

```

> # Machine Learning spring 2005 exercise 2.5, using MAPLE
> # a) log loss
> logl := (1-y)/2*ln( (1-y)/(1-x) ) + (1+y)/2*ln( (1+y)/(1+x) );
> logld1:=diff(logl,x);
> logld2:=diff(logld1,x);
> logtmp:=simplify( (logld1^2)/logld2);
      logl := 1/2 (1 - y) ln( (1-y)/(1-x) ) + 1/2 (1 + y) ln( (1+y)/(1+x) )
      logld1 := 1/2 (1-y)/(1-x) - 1/2 (1+y)/(1+x)
      logld2 := 1/2 (1-y)/(1-x)^2 + 1/2 (1+y)/(1+x)^2
      logtmp := - (x+y)^2 / (-1-x^2+2yx)
> logt1:=simplify(eval(logtmp,y=1)); # consider y=-1,y=1 separately
> logt2:=simplify(eval(logtmp,y=-1)); # -> results in two constant
values
> logchat:=max(logt1,logt2);
      logt1 := 1
      logt2 := 1
      logchat := 1
> # b) hellinger loss
> hell := 1/2*( (sqrt(1-y) - sqrt(1-x))^2 + (sqrt(1+y) - sqrt(1+x))^2);
> helld1:=diff(hell,x);
> helld2:=diff(helld1,x);
> helltmp:=simplify( (helld1^2)/helld2);
hell := 1/2 (sqrt(1-y) - sqrt(1-x))^2 + 1/2 (sqrt(1+y) - sqrt(1+x))^2
helld1 := 1/2 (sqrt(1-y)-sqrt(1-x))/sqrt(1-x) - 1/2 (sqrt(1+y)-sqrt(1+x))/sqrt(1+x)
helld2 := 1/4 (1-x)^-1 + 1/4 (sqrt(1-y)-sqrt(1-x))/(1-x)^(3/2) + 1/4 (1+x)^-1 + 1/4 (sqrt(1+y)-sqrt(1+x))/(1+x)^(3/2)
helltmp := (sqrt(1+x) sqrt(1-x) (sqrt(1+x) sqrt(1-y) - sqrt(1-x) sqrt(1+y))^2) / (sqrt(1+x) sqrt(1-y) x - sqrt(1-x) sqrt(1+y) x + sqrt(1+x) sqrt(1-y) + sqrt(1-x) sqrt(1+y))

```

```

> hellt1:=simplify(eval(helltmp,y=1)); # consider y=-1,y=1 separately
> hellt2:=simplify(eval(helltmp,y=-1));
> hellm1:=maximize(hellt1,x=-1...1);
> hellm2:=maximize(hellt2,x=-1...1);
> hellhat:=max(hellm1,hellm2);
> # 2
      hellt1 :=  $\sqrt{2}\sqrt{1+x}$ 
      hellt2 :=  $\sqrt{1-x}\sqrt{2}$ 
      hellm1 := 2
      hellm2 := 2
      hellhat := 2
> # For comparison, square loss goes nice and clean ;)
> tmp := (y-x)^2;
> tmpd1 := diff(tmp,x);
> tmpd2 := diff(tmpd1,x);
> sqrrat := simplify( (tmpd1^2)/tmpd2);
> sqrchat := maximize(sqrrat,x=-1...1,y=-1...1);
> # 8
      tmp :=  $(-x+y)^2$ 
      tmpd1 :=  $2x-2y$ 
      tmpd2 := 2
      sqrrat :=  $2(-x+y)^2$ 
      sqrchat := 8

```

□