Problem 1

In this exercise, the target label $y$ belongs to the set $\{0, 1\}$.

**Absolute loss.** Let $p \in [0, 1]$. The absolute loss is defined as $L_{\text{abs}}(y, p) = |y - p|$. Because $p \in [0, 1]$, it holds $L_{\text{abs}}(0, p) = p$ and $L_{\text{abs}}(1, p) = 1 - p$. Assume that next label is 1 with probability $q$ and 0 with probability $1 - q$. The expected loss is then as a function of our prediction $p$

$$f(p) = (1 - q)L_{\text{abs}}(0, p) + qL_{\text{abs}}(1, p) = (1 - q)p + q(1 - p) = (1 - 2q)p + q.$$  

$f$ is a linear function that has its minimum when $p = 0$ or $p = 1$. Because $f(0) < f(1) \iff q < 1 - q \iff q < 1/2$,
the Bayes optimal prediction is as a function of $q$

$$\hat{y}(q) = \begin{cases} 0 & \text{if } q \in [0, 1/2[ \\ 1 & \text{if } q \in [1/2, 1] \end{cases}.$$  

When $q = 1/2$, the loss is always $f(p) = 1/2$, no matter which $p$ we choose.

**Logarithmic loss.** Let $p \in ]0, 1[$. The logarithmic loss is

$$L_{\text{log}}(y, p) = \begin{cases} -\ln(1 - p) & \text{if } y = 0 \\ -\ln p & \text{if } y = 1 \end{cases}.$$  

In this case, the expected loss is

$$f(p) = -(1 - q)\ln(1 - p) - q\ln p.$$  

We search the point where $f'(p) = 0$ as follows:

$$f'(p) = \frac{1 - q - q}{1 - p} = \frac{p(1 - q) - q(1 - p)}{p(1 - p)} = \frac{p - q}{p(1 - p)} = 0 \Rightarrow p = q.$$  

Because $f'(p) < 0$ if $p < q$, and $f'(p) > 0$ if $p > q$, the expected loss $f(p)$ is minimized by choosing $\hat{y}(q) = p$. Then the expected loss is $f(q) = -(1 - q)\ln(1 - q) - q\ln q$ (which is the entropy of the target label in nats).
Squared loss. Let $p \in [0, 1]$. The definition of the squared loss is $L_{sq}(y, p) = (y - p)^2$, and the expected loss thus

$$f(p) = (1 - q)p^2 + q(1 - p)^2$$

$$= (1 - q)p^2 + q(1 - 2p + p^2)$$

$$= p^2 - 2q \cdot p + q.$$  

Setting its derivative to zero yields

$$f'(p) = 2(p - q) = 0 \implies p = q.$$  

The value $f(p)$ is minimized at $p = q$ because $f'(p) < 0$ if $p < q$, and $f'(p) > 0$ if $p > q$. Similarly as in the case of the logarithmic loss, the optimal Bayes prediction is $\hat{y}(q) = p$. (We see now that $f(q) = q(1 - q)$, which is the variance of the Bernoulli$(q)$ distributed target label.)

Problem 2

Instead of thinking about binary classifiers over the set $X = \{a, a+1, \ldots, b-1, b\}$, it is easier here to consider subsets of $X$. (Following the terminology on page 15 of lectures, we might call them concepts.) Let $C$ be the set of all intervals that are subsets of $X$. There are $b - a + 1$ potential end points for intervals. There are three kinds of intervals:

1. the empty set (we consider it an interval corresponding to $h_{p,q}$ for any $p > q$; removing it from our class of intervals would not change anything essential in what follows)
2. intervals consisting of just one point of $X$
3. intervals with two distinct end point from $X$.

Therefore,

$$|C| = 1 + (b - a + 1) + \frac{(b - a + 1)(b - a)}{2}.$$  

Consider now the Halving Algorithm. For $S_+, S_- \subseteq X$, let $V(S_+, S_-)$ be the set of intervals in $C$ that contain all the elements of $S_+$ but none of the elements of $S_-$. Notice that for any $x \in X$ we have

$$V(S_+, S_-) = V(S_+ \cup \{x\}, S_-) \cup V(S_+, S_- \cup \{x\}).$$  

Specifically, $V(S_+ \cup \{x\}, S_-)$ consists of those intervals $c \in V(S_+, S_-)$ that include the point $x$, and $V(S_+, S_- \cup \{x\})$ consists of those that don’t include $x$.

We can now write the Halving Algorithm (page 28 of lectures) for our special case as

1. Initialise $S_+ = \emptyset$ and $S_- = \emptyset$.
2. Repeat as long as there are examples to be classified:
   (a) Receive point $x \in X$ to be classified.
   (b) Let $P = |V(S_+ \cup \{x\}, S_-)|$ and $M = |V(S_+, S_- \cup \{x\})|$.
   (c) If $P \geq M$, predict 1, else predict 0.
   (d) Receive the correct classification $y$.
   (e) If $y = 1$, let $S_+ \leftarrow S_+ \cup \{x\}$. Else let $S_- \leftarrow S_- \cup \{x\}$.

To simplify notations later, we change the initialisation to $S_- = \{a - 1, b + 1\}$. Notice that this does not change the results in any way, since $a - 1$ and $b + 1$ are not in any of the intervals we consider. However, it is convenient that any point in $X$ is always between two points of $S_-$. We call the elements of $S_+$ positive examples, and elements of $S_-$ negative examples.
Recall that as always with the Halving Algorithm, we are assuming that the data we are given is such that at least one hypothesis of the class is consistent, that is, $V(S_+, S_-)$ never becomes empty. In particular, if $u, v \in S_+$ and $u \leq x \leq v$, then $x \notin S_-$. Consider now determining efficiently whether $P \geq M$ holds when we are given a point $x \in X$ to classify. We split into two cases depending on whether $S_+$ is empty or not.

First, assume $S_+ = \emptyset$. In the special case that $x \in S_-$, we have $P = 0$ and, since by assumption there is always one consistent interval, $M > 0$. Therefore $P < M$ and we predict 0.

Otherwise, we can pick two elements $r, s \in S_-$ such that $r < x < s$, and there are no negative examples between $r$ and $s$. (As mentioned above, we can include $a - 1$ and $b + 1$ in $S_-$ so that this is always possible.)

Now $V(\{ x \}, S_-)$ consists of intervals $[p, q]$ with $r < p \leq x$ and $x < q < s$. Therefore, $P = (x - r)(s - x)$.

We could also calculate $M$ by considering various kinds of intervals that include none of the points in $S_- \cup \{ x \}$. However, it is easier to keep track of the value $N = |V(S_+, S_-)|$. We know that $M = N - P$, and we can recursively update $N \leftarrow N - P$ when we observe the label $y = 0$ for $x$. (If $y = 1$ we move to the case $S_+ \neq \emptyset$ and this is no longer necessary.)

Assume now that $S_+ \neq \emptyset$, and let $u = \min S_+$ and $v = \max S_+$ be the smallest and largest positive example. Further, let

$$r = \max \{ z \in S_- | z < u \}$$

be the first negative example to the left of $r$, and

$$s = \min \{ z \in S_- | z > v \}$$

be the first negative example to the right of $s$.

Now the version space $V(S_+, S_-)$ consists of intervals $[p, q]$ with $r < p \leq u$ and $v \leq q < s$. If $u \leq x \leq v$, then $x$ belongs to all such intervals, so we predict $\hat{y} = 1$. If $x \leq r$ or $x \geq s$, then $x$ belongs to none of such intervals, so we predict $\hat{y} = 0$. In both of these cases, because of the assumption that at least one hypothesis from our hypothesis class is consistent with all the examples, we know that the prediction we made is correct, and the version space does not change.

Consider now the case $r < x < u$. The set $V(S_+ \cup \{ x \}, S_-)$ consists of intervals $[p, q]$ with $r < p \leq x$ and $u \leq q < s$. Therefore, $P = (x - r)(s - v)$. The set $V(S_+, S_- \cup \{ x \})$ consists of intervals $[p, q]$ with $x < p \leq u$ and $v \leq q < s$. Therefore, $M = (u - x)(s - v)$. We conclude that $P \geq M$ if and only if $x - r \geq u - x$, that is, $x \geq (u + r)/2$. Similarly, for the case $v < x < s$, we get $P \geq M$ if and only if $x \leq (v + s)/2$. (We see that the Halving Algorithm actually works here like the Nearest Neighbor algorithm.)

We get the following refined representation of the Halving Algorithm.

1. Initialise $S_+ = \emptyset$ and $S_- = \{ a - 1, b + 1 \}$.

2. Let $N = 1 + (b - a + 1) + \frac{(b - a + 1)(b - a)}{2}$.

3. Repeat while $S_+$ is empty:

   a) Receive input $x$.

   b) If $x \in S_-$, predict $\hat{y} = 0$ and proceed to the next input.

   c) Let $r = \max \{ z \in S_- | z < x \}$ and $s = \min \{ z \in S_- | z > x \}$.

   d) Let $P = (x - r)(s - x)$ and $M = N - P$.

   e) If $P \geq M$, predict $\hat{y} = 1$, else predict $\hat{y} = 0$.

   f) Receive the correct label $y$. If $y = 0$, update $N \leftarrow M$.

4. **Comment:** Now $S_+$ contains exactly one point $x$, and $r$ and $s$ are points in $S_-$ such that $r < x < s$ and there are no points of $S_-$ between $r$ and $s$. 

3
5. Let \( u \leftarrow x \) and \( v \leftarrow x \).

6. Repeat as long as there are points to be classified:

   (a) Receive input \( x \).

       • If \( u \leq x \leq v \), then predict \( \hat{y} = 1 \).
       • If \( x \leq r \) or \( x \geq s \), then predict \( \hat{y} = 0 \).
       • If \( r < x < u \), then if \( x \geq (r + u) / 2 \) then predict \( \hat{y} = 1 \), else predict \( \hat{y} = 0 \). Receive the correct label \( y \) and if \( y = 1 \), then update \( u \leftarrow x \), else update \( r \leftarrow x \).
       • If \( v < x < s \), then if \( x \leq (s + v) / 2 \) then predict \( \hat{y} = 1 \), else predict \( \hat{y} = 0 \). Receive the correct label \( y \) and if \( y = 1 \), then update \( v \leftarrow x \), else update \( s \leftarrow x \).

Assuming arithmetic operations take constant time, everything here is constant time per classified example except finding the points \( r \) and \( s \) in Step 3(c). This can be done in time \( O(|S_-|) \) in straightforward manner. This is already a considerable saving compared to explicitly manipulating the version space, since the size of the version space is initially \( \Theta((b-a)^2) \) and \( |S_-| \leq b-a+1 \). However, a more efficient way it to store the elements of \( S_- \) into a balanced binary search tree as they arrive. (You are expected to be familiar with balanced search trees from an earlier course in algorithms and data structures, but for this course they are not central.) Storing an element into a balanced binary search tree can be done in time \( O(\log m) \) where \( m \) is the total number of elements. Given an element \( x \) we can also in time \( O(\log m) \) find the next largest and next smallest element in the tree, compared to \( x \). Hence, everything runs in time \( O(\log |S_-|) \) per example to be classified.

In our exact implementation of the Halving Algorithm, we spend a considerable amount of work just to deal with the initial stage until the first positive example is observed. We can considerably simplify the algorithm by replacing step 3 by simply

\[ \text{While } S_+ \text{ is empty, predict } \hat{y} = 0. \]

Compared to the exact implementation, we might make one extra error (if the exact implementation succeeds in predicting the first positive example), but no more than that. However, the algorithm becomes much simpler to implement, and the computational cost is now constant per prediction. After step 3 we need to find \( r \) and \( s \), but this can easily be done in time \( O(|S_-|) \), but this is done only once, so amortised over the examples it’s still constant per example.

Finally, we may notice that this modified version of the Halving Algorithm does not necessarily need to be given the values \( a \) and \( b \) in advance, although it does need them if the first positive example does not fall between any two negative examples already seen.

Exercise 3

a) It is stated in the proof of Theorem 1.5 that

\[
L_{0-1}(S, WM) \leq P_1 - P_{T+1} = c \ln W_1 - c \ln W_{T+1} = c \ln n - c \ln W_{T+1}.
\]  

The sum of the weights is at the end

\[
W_{T+1} = \sum_{i=1}^{n} w_{T+1,i}.
\]

The weights are always non-negative, so we can get a lower bound for the sum just by summing over those \( k \) hypotheses which we know something about in our special case. There are \( k \) different hypotheses that have made at most \( M \) mistakes. Every mistake reduces the weight by the factor \( \beta \)
in the algorithm, therefore at the smallest each of these weights is $\beta^M$. Summing them together gives the bound

$$W_{T+1} \geq k\beta^M.$$  \hspace{1cm} (2)

As $W_{T+1}$ is multiplied by a negative factor in (1) ($c = 1/(\ln 2 - \ln(1 + \beta)) > 0$ because $\beta \in [0,1]$), we can plug (2) into (1).

$$L_{0-1}(S,WM) \leq c \ln n - c \ln(k\beta^M) = c \ln n - c \ln k - cM \ln \beta = c\eta M + c \ln \frac{n}{k},$$

where $\eta = -\ln \beta$.

b) We can apply the same basic idea to the modified algorithm $WM'$:

$$L_{0-1}(S,WM') \leq P_1 - P_{T+1} = c \ln W_1 - c \ln W_{T+1}.$$  

Now at the beginning, $W_1 = \sum_{i=1}^n w_i = \sum_{i=1}^n p_i = 1$, and thus $c \ln W_1 = 0$. At the end, it holds for all $i \in \{1,2,\ldots,n\}$ that

$$W_{T+1} \geq w_{T+1,i} = p_i \beta^{L_{0-1}(S,h_i)}.$$  

For all $i \in \{1,2,\ldots,n\}$ we have the bound

$$L_{0-1}(S,WM') \leq 0 - c \ln W_{T+1} \leq -c(\ln p_i + L_{0-1}(S,h_i) \ln \beta) \leq c\eta L_{0-1}(S,h_i) + c \ln \frac{1}{p_i}.$$  

Taking the minimum over $i \in \{1,2,\ldots,n\}$ gives the tightest bound and completes the proof.

**Exercise 4**

In this exercise, only $h_1$ predicts better than e.g. random coin tosses. And even hypothesis $h_1$ makes an expected number of $\nu T$ mistakes because of the way the data is generated. Let us consider some examples of fast learning in presence of a reasonable amount of noise first. The values of the first component of the input are flipped with probability $\nu = 0.4$, the number of hypotheses is $n = 100$ and the learning rate parameter is $\beta = 0.1$. In the upmost figure, the cumulative sum of zero-one losses is plotted. If we had used solely $h_1$ for prediction right from the beginning, the expected number of mistakes would been $200$, which we obviously do not quite achieve. By carefully looking at the picture, you notice how the curve becomes less steep at around $t = 260$. That is the moment when we start predicting optimally, which is easily seen in the second plot with the cumulative sum of mistakes compared to $x_{t,1}$ components. The relative weights are plotted in the lowest figure. The learning happens rather chaotically at first when various wrong hypotheses have large relative weights for short periods.
In the next three figures, the same data is used as before, but the learning rate parameter is now \( \beta = 0.9 \). It takes much longer to find the right hypothesis, and at the beginning the weights are distributed quite evenly between the hypotheses for some time.
The last set of figures illustrates slow learning with easy data ($\nu = 0.2$). The number of hypotheses is now $n = 1000$. Even with $\beta = 0.9$ earning is relatively effective because of the small amount of noise. The relative weight of $h_1$ grows quite smoothly as a function of time. It dominates the voting from around $t = 180$ but it takes some more time before $v_{t,1}$ is close to 1.

Kuva 2: $n = 100, \nu = 0.4, \beta = 0.9, T = 500$. 
Kuva 3: $n = 1000, \nu = 0.2, \beta = 0.1, T = 500.$