1. Consider soft binary classification with some appropriate loss function $L$. Suppose that for some reason we believe that the next label $y$ is $1$ with probability $q$, and $0$ with probability $1 - q$. Then a natural prediction would be $p$ which minimises the expected loss

$$(1 - q)L(0, p) + qL(1, p).$$

Such a value $p$ is called the Bayes optimal prediction.

Determine the Bayes optimal prediction, as a function of $q$, for absolute loss $L_{abs}$, logarithmic loss $L_{log}$ and square loss $L_{sq}$.

2. Let $X = \{a, a + 1, \ldots, b - 1, b\}$ for some integers $a$ and $b$. Let $H : X \rightarrow \{0, 1\}$ consist of all functions $h_{p,q}$ where

$$h_{p,q}(x) = \begin{cases} 1 & \text{if } p \leq x \leq q \\ 0 & \text{otherwise} \end{cases}$$

(i.e., $H$ consists of all the indicator functions of intervals restricted to $X$).

Give a computationally efficient implementation to the Halving Algorithm for this hypothesis class. In other words, explain how you can keep track of the version space and perform the necessary voting. You should get the running time at least down to $O(b - a)$ per trial. By using efficient data structures, a logarithmic time should also be possible.

Give your solution as a fairly high-level pseudocode, but explain the data structures in sufficient detail so that the running times are justified.

**Remark:** On this course we do not often analyse the running times of algorithms. The Halving Algorithm (and its relatives like Weighted Majority) are an exception. Because the mistake bound depends only logarithmically on $|H|$, it allows quite large hypothesis classes. Then the naive $O(|H|)$ implementation is impractical, and it becomes interesting to find ways of improving the computational efficiency for specific hypothesis classes by taking into account their combinatorial structure.

3. Consider the Weighted Majority algorithm with some $\beta > 0$, and let $\eta$ and $c$ be as in Theorem 1.5. In this problem you are asked to generalise the proof of Theorem 1.5.

(a) Show that if for some $M \geq 0$ there are $k$ different hypotheses $h_i$ that all satisfy $L_{0,1}^{\leq T}(S, h_i) \leq M$, then

$$L_{0,1}^{\leq T}(S, WM) \leq c\eta M + c\ln \frac{n}{k}.$$
4. This is a small programming exercise for familiarising you with doing simple online learning in a programming environment of your choice. Popular choices include Matlab, Octave, R, and Python. You may use something else if you prefer, but if plan to you use some unusual language, please check first with the lecturer that he understands it enough to grade your work.

Consider the following setting.

- There are $T$ inputs $x_1, \ldots, x_T$, where each $x_t = (x_{t,1}, \ldots, x_{t,n}) \in \{-1, 1\}^n$ is an $n$-dimensional vector. Each component $x_{t,i}$ is 1 with probability $1/2$ and $-1$ with probability $1/2$.
- The concept class is $H = \{h_1, \ldots, h_n\}$ where $h_i$ picks the $i$th component of the input: $h_i((x_1, \ldots, x_n)) = x_i$.
- The labels $y_t \in \{-1, 1\}$ are determined by
  
  $y_t = \begin{cases} 
  x_{t,1} & \text{with probability } 1 - \nu \\
  -x_{t,1} & \text{with probability } \nu 
  \end{cases}$

  for some $\nu > 0$. In other words, $h_1$ is the target concept but there is classification noise with rate $\nu$.

Implement the Weighted Majority algorithm in this setting. Try it with a couple of values for $n$ (say $n = 100$ and $n = 1000$) and $\nu$ (say $\nu = 0.2$ and $\nu = 0.4$) and see, how the choice of learning rate $\beta$ affects the behaviour of the algorithm. In particular, making following kinds of plots may be illuminating:

- Plot the normalized weights $v_{t,i}/\sum_j w_{t,j}$ as a function of $t$; compare the “relevant” weight $v_{t,1}$ to “irrelevant” ones $v_{t,j}$, $j \neq 1$, or plot all the weights into the same figure.
- Plot the cumulative loss $\sum_{j=1}^t L_{0-1}(y_j, \hat{y}_t)$ as a function of $t$.
- Plot the cumulative mistake count with respect to the “unnoisy” labels, $\sum_{j=1}^t L_{0-1}(x_{t,1}, \hat{y}_t)$, as a function of $t$.

In general, you should see that with $\beta$ close to 0 the algorithm behaves erratically, with $\beta$ close to 1 more smoothly but also takes more time. In most cases $T = 500$ or fewer iterations should be sufficient to see what is going on.

Your answer to this problem should consist of a brief explanation of the observations you made, supported by some plots, and your program code (which doesn’t need to be polished, but some comments in the code might be useful if you do something non-obvious). Don’t include all the plots you make, just half a dozen or so representative ones. You are not expected to make a systematic study of this setup, which after all is a bit artificial. The goal is to make the Weighted Majority algorithm a bit more concrete and prepare for later exercises with other online algorithms.