Machine Learning HW1 - Theory

1. **Question:** Consider running the Perceptron algorithm on a training set S arranged in a certain order. Now suppose we run it with the same initial weights and on the same training set but in a different order, S'. Does Perceptron make the same number of mistakes? Does it end up with the same final weights? If so, prove it. If not, give a counterexample, i.e. an S and S' where order matters.

Answer: Intuitively it should not since, depending on the value of x, the update may save multiple mistakes down the road. In other words, the update function is not associative. Let's look at an example S and S' and run it through Perceptron.

$$
\begin{aligned}
S = \{ & \\
& (x_0 = (1\ 1\ 1), y_0 = 1), \\
& (x_1 = (1\ 0\ 0), y_1 = 1), \\
& (x_2 = (0\ 0\ 1), y_2 = -1), \\
\} & \\
w^0 &= (000) \\
y_0\langle w^0, x_0\rangle &= (1)\langle(0\ 0\ 0), (1\ 1\ 1)\rangle = 0 \le 0 \Rightarrow w^1 = (0\ 0\ 0) + (1)(1\ 1\ 1) = (1\ 1\ 1) \\
y_1\langle w^1, x_1\rangle &= (1)\langle(1\ 1\ 1), (1\ 0\ 0)\rangle = 1 > 0 \\
y_2\langle w^1, x_2\rangle &= (-1)\langle(1\ 1\ 1), (0\ 0\ 1)\rangle = -1 \le 0 \Rightarrow w^2 = (1\ 1\ 1) + (-1)(0\ 0\ 1) = (1\ 1\ 0) \\
y_0\langle w^2, x_0\rangle &= (1)\langle(1\ 1\ 0), (1\ 1\ 1)\rangle = 2 > 0 \\
y_1\langle w^2, x_1\rangle &= (1)\langle(1\ 1\ 0), (1\ 0\ 0)\rangle = 1 > 0 \\
y_2\langle w^2, x_2\rangle &= (-1)\langle(1\ 1\ 0), (0\ 0\ 1)\rangle = 0 \le 0 \Rightarrow w^3 = (1\ 1\ 0) + (-1)(0\ 0\ 1) = (1\ 1\ -1) \\
y_0\langle w^3, x_0\rangle &= (1)\langle(1\ 1\ -1), (1\ 1\ 1)\rangle = 1 > 0 \\
y_1\langle w^3, x_1\rangle &= (1)\langle(1\ 1\ -1), (1\ 0\ 0)\rangle = 1 > 0 \\
y_2\langle w^3, x_2\rangle &= (-1)\langle(1\ 1\ -1), (0\ 0\ 1)\rangle = 1 > 0
\end{aligned}
$$

So S gave us 3 mistakes and finished at $w^3 = (1\ 1\ -1)$.

$$
\begin{aligned}
&S' = \{\\
&\quad (x_0 = (0\ 0\ 1), y_0 = -1),\\
&\quad (x_1 = (1\ 0\ 0), y_1 = 1),\\
&\quad (x_2 = (1\ 1\ 1), y_2 = 1),\\
&\}\\
&w^0 = (000)\\
y_0\langle w^0, x_0\rangle &= (-1)\langle(0\ 0\ 0), (0\ 0\ 1)\rangle = 0 \leq 0 \Rightarrow w^1 = (0\ 0\ 0) + (-1)(0\ 0\ 1) = (0\ 0\ -1)\\
y_1\langle w^1, x_1\rangle &= (1)\langle(0\ 0\ -1), (1\ 0\ 0)\rangle = 0 \leq 0 \Rightarrow w^2 = (0\ 0\ -1) + (1)(1\ 0\ 0) = (1\ 0\ -1)\\
y_2\langle w^2, x_2\rangle &= (1)\langle(1\ 0\ -1), (1\ 1\ 1)\rangle = 0 \leq 0 \Rightarrow w^3 = (1\ 0\ -1) + (1)(1\ 1\ 1) = (2\ 1\ 0)\\
y_0\langle w^3, x_0\rangle &= (-1)\langle(2\ 1\ 0), (0\ 0\ 1)\rangle = 0 \leq 0 \Rightarrow w^4 = (2\ 1\ 0) + (-1)(0\ 0\ 1) = (2\ 1\ -1)\\
y_1\langle w^4, x_1\rangle &= (1)\langle(2\ 1\ -1), (1\ 0\ 0)\rangle = 2 > 0\\
y_2\langle w^4, x_2\rangle &= (1)\langle(2\ 1\ -1), (1\ 1\ 1)\rangle = 2 > 0\\
y_0\langle w^4, x_0\rangle &= (-1)\langle(2\ 1\ -1), (0\ 0\ 1)\rangle = 1 > 0\\
y_1\langle w^4, x_1\rangle &= (1)\langle(2\ 1\ -1), (1\ 0\ 0)\rangle = 2 > 0\\
y_2\langle w^4, x_2\rangle &= (1)\langle(2\ 1\ -1), (1\ 1\ 1)\rangle = 2 > 0
\end{aligned}
$$

So S' gave us 4 mistakes and finished at $w^4 = (21 - 1)$, both a different result and a different number of mistakes.

2. **Question:** Explain the explicit relationship between the SGD algorithm with hinge loss and the Perceptron algorithm.

Answer: Let's look at the SGD algorithm update.

$$w_{new} = w_{old} - \eta \nabla \phi(y^i w^T x^i)$$

$\phi(z) = max(0, -z)$ is piecewise differentiable. For simplicity, let's assume the derivative at 0 to be equal to the $-z$ part.

$$
\begin{aligned}
\frac{\delta \phi(z = y^i w^T x^i)}{\delta w_j} &= \begin{cases} 0 \text{ if } y^i w^T x^i > 0 \\ \frac{\delta - y^i w^T x^i}{\delta w_j} = -y^i x^i_j \text{ if } y^i w^T x^i \leq 0 \end{cases} \iff \\
\nabla \phi(y^i w^T x^i) &= \begin{cases} 0 \text{ if } y^i w^T x^i > 0 \\ -y^i x^i \text{ if } y^i w^T x^i \leq 0 \end{cases}
\end{aligned}
$$

If we pick $\eta = 1$ then

$$w_{new} = \begin{cases} w_{old} \text{ if } y^i w^T x^i > 0 \\ w_{old} + y^i x^i \text{ if } y^i w^T x^i \leq 0 \end{cases}$$

Notice that that's the same update step as the Perceptron algorithm! Thus the Perceptron update is a special case of SGD, equivalent to SGD paired with the hinge loss function and $\eta = 1$. The Perceptron algorithm assumes separability of the training set S; thus it stops once w converges. SGD could stop before convergence, but we are guaranteed convergence because of separability so it will eventually reach that point. Of course, the ordering of the set matters in the results we reach, so the results might not be the same since SGD processes S in a random order, but the outcome will be the same.

3.

a) **Question:** Show formally that for any distribution on $\mathbb{R}$ (assume finite support, for simplicity; i.e., assume the distribution is bounded within $[-B, B]$ for some large B) and any unknown labeling function $c \in \mathcal{C}$ that is a 3-piece classifier, there exists a decision stump $h \in \mathcal{H}$ that has error at most 1/3, i.e. $\mathbb{P}[h(x) \neq c(x)] \leq 1/3$.

Answer: Take $\forall c_{\theta_1,\theta_2} \in C$. The function is defined as

$$c(x) = \begin{cases} -b \text{ if } x \in L = [-B, \theta_1) \\ b \text{ if } x \in C = [\theta_1, \theta_2] \\ -b \text{ if } x \in R = (\theta_2, B] \end{cases}$$

Let's look at the probabilities that a point is in each set:

$$\mathbb{P}[x \in L] + \mathbb{P}[x \in C] + \mathbb{P}[x \in R] = 1$$

From this we can say that at least one of these probabilities has to be $\leq 1/3$; if they all were $> 1/3$ then the sum would be > 1 (contradiction).

Assume that $\mathbb{P}[x \in L] \leq 1/3$. Then take

$$h(x) = \begin{cases} b \text{ if } x \in L \cup C = [-B, \theta_2] \\ -b \text{ if } x \in R = (\theta_2, B] \end{cases}$$

Then it's easy to see that $\mathbb{P}[h(x) \neq c(x)] = \mathbb{P}[x \in L] =\leq 1/3$ since $h(x)$ agrees with $c(x)$ on C and R.

Similarly, assume $\mathbb{P}[x \in R] \leq 1/3$. Then take

$$h(x) = \begin{cases} -b \text{ if } x \in L = [-B, \theta_1) \\ b \text{ if } x \in C \cup R = [\theta_1, B] \end{cases}$$

Then it's easy to see that $\mathbb{P}[h(x) \neq c(x)] = \mathbb{P}[x \in R] =\leq 1/3$ since $h(x)$ agrees with $c(x)$ on C and L.

Lastly, assume $\mathbb{P}[x \in C] \leq 1/3$. Then take

$$h(x) = \begin{cases} -b \text{ if } x \in L \cup C \cup R = [-B, B] \end{cases}$$

Then it's easy to see that $\mathbb{P}[h(x) \neq c(x)] = \mathbb{P}[x \in C] =\leq 1/3$ since $h(x)$ agrees with $c(x)$ on L and R.

Therefore $\forall c(x) \exists h(x)$ with the required property.

b. **Question:** Describe a simple, efficient procedure for finding a decision stump that minimizes error with respect to a finite training set of size m. Such a procedure is called an empirical risk minimizer (ERM).

Answer:

In the vein of the previous answer, we need a procedure that estimates what θ_1 and θ_2 are and figures out the weights of the 3 different bins, so that we can choose the right classifier.

```
Initialize 3 counters: L = 0, C = 0, R = 0, theta_1 = +infty, theta_2 = -infty.

For each example [x_i, y_i] with y_i == b do:
  C += 1
  theta_1 = min(x_i, theta_1)
  theta_2 = max(x_i, theta_2)
end

For each example [x_i, y_i] with y_i == -b do:
  L += 1 if y_i == -b and x_i < theta_1
  R += 1 if y_i == -b and x_i > theta_2
end

Case min(L, R, C)
  when L return h(theta_2)
  when C return h(0) = -b
  when R return h(theta_1)
end
```

The error on the training set will be $min(L, C, R)/m$. The algorithm runs in $O(m)$ complexity.

c. **Question:** Give a short intuitive explanation for why we should expect that we can easily pick a sufficiently large m that the training error is a good approximation of the true error, i.e. why we can ensure generalization. (Your answer should relate to what we have gained in going from requiring a learner for $\mathcal{C}$ to requiring a learner for $\mathcal{H}$.) This lets us conclude that we can weakly learn $\mathcal{C}$ using $\mathcal{H}$.

Answer: We have gained simplicity! But also error! We've also gained the fact that $\mathcal{H}$ is PAC learnable.

So for h learned from training set of size $m = \frac{1}{\epsilon}\log(\frac{1}{\delta})$ we know that $err(h) \leq \epsilon$ with probability $1 - \delta$ for any $x \notin X$, where X is the "blind spot" (one of the minimal set L, C, R from above). For the blind spot the error is $\leq 1/3$, as long as A can give us the right classifier. So in this case the "bad event" Is A gave us the wrong classifier h, even though the error of $h > 1/3 + \epsilon$. This happens if we draw samples that are not representative of the actual distribution (if it was, then we'd pick the right classifier). The probability that happens is the probability that we draw more samples from X than the actual representative probability $\mathbb{P}[x \in X] \leq 1/3$.

So if k samples out of m are from X, then the chance that $k > \frac{m}{3}$, knowing that we have at most a $1/3$ chance to draw x is given to use by the binomial distribution

$\mathbb{P}[k > \frac{m}{3}] = F(\frac{2m}{3}; m, \frac{2}{3})$

which we can bound by δ for a particular size of m.

For a large enough m, then we know $\mathcal{C}$ can be weakly learned by $\mathcal{H}$ since $\mathcal{H}$ can give us a PAC learner with $err(h) \leq 1/3 + \epsilon \leq 1/2 - (1/6 - \epsilon)$ for values of m that make $\epsilon \leq 1/6$.

4. **Question:** Consider an iteration of the AdaBoost algorithm (using notation from the video lecdeeure on Boosting) where we have obtained classifer h_t . Show that with respect to the distribution D_{t+1} generated for the next iteration, h_t has accuracy exactly 1/2.

Answer: From the video lecdeeure:

$\beta = \frac{\frac{1}{2}-\gamma}{\frac{1}{2}+\gamma}$ After a step the weight of the correct points is $\left(\frac{1}{2}+\gamma\right)\beta w$. After a step the weight of all the points is $w(1-2\gamma)$.

Thus, for D_{t+1} our classifier would have accuracy

$$A = \frac{\left(\frac{1}{2}+\gamma\right)\beta w}{(1-2\gamma)w} = \frac{\frac{1}{2}-\gamma}{2(\frac{1}{2}-\gamma)} = \frac{1}{2}$$