EN.601.475/675 Machine Learning

Homework 3: Due Wednesday, October 15th, 11:59pm

Submission Instructions: You will upload your solutions to the analytical portion and the programming portion separately to Gradescope.

- **Analytical Part (this file)**: You should use LaTeX to type up your solutions (e.g., via Overleaf or similar), but please ensure that **each part (e.g., a, b, etc) begins on a new page** (to facilitate grading in Gradescope). You will upload the PDF to the assignment "Homework 3: Analytical" on Gradescope.
- **Programming Part (see Canvas)**: You should upload your `.ipynb` file directly to Gradescope under the assignment "Homework 3: Programming".

Note that you must upload **both** parts to their respective assignments on Gradescope.

Collaboration Policy: You may work with other students on the homework assignments, but you must write up and submit your own work independently, and you must acknowledge your collaborators on the first page of your homework submission.

Late Submission Policy: You have up to 48 hours of "slack" for late submissions of homework assignments, across **all** homework assignments. Once you have exhausted your budget, we will deduct points for late submissions. However, **no individual assignment may be submitted more than 48 hours late**, as we will release solutions at this time.

Use of Generative AI: You are **not permitted** to use generative AI tools (e.g., ChatGPT) for problems on this problem set.

Name: Rashed Aldulijan

JHU Email: ralduli1@jh.edu

Collaborators (or "None"): None

The analytical portion (this file) has 4 questions, for a total of 65 points.

Part 1: Analytical Questions

Question 1 ..... *15 points*

In this question, we will apply the algorithm used to train AdaBoost on a binary classification problem with "decision stumps" (single split decision trees) as weak learners. The labels are $y \in \{+1, -1\}$. The three decision stumps we will use are defined as follows, using the indicator notation:

$$h_1(x) = \mathbf{1}\{\texttt{free} = 1\}, \quad h_2(x) = \mathbf{1}\{\texttt{money} = 1\}, \quad h_3(x) = \mathbf{1}\{\texttt{win} = 1\},$$

where (for the purposes of this problem) $\mathbf{1}\{\cdot\}$ returns $+1$ if the condition is true and -1 otherwise. We have the following dataset of $n = 8$ emails, where the columns represent specific words found in an email, and the $+1$ label represents "spam," -1 label represents "not spam":

i	`free`	`money`	`win`	y
1	1	0	0	+1
2	0	1	0	+1
3	0	0	1	−1
4	1	1	0	+1
5	0	0	0	−1
6	0	1	1	+1
7	1	0	1	+1
8	0	0	1	+1

Weights initialize as $D_1(i) = 1/8$. Recall that at round t we have:

$$\varepsilon_t = \Pr_{i \sim D_t}[h_t(x_i) \neq y_i], \quad \alpha_t = \tfrac{1}{2} \ln \frac{1 - \varepsilon_t}{\varepsilon_t} \quad D_{t+1}(i) \propto D_t(i) \exp(-\alpha_t y_i h_t(x_i)).$$

(a) (5 points) For each weak learner h_j, compute $\varepsilon_1(h_j) = \Pr_{i \sim D_1}[h_j(x_i) \neq y_i]$. Which of the three is the best weak learner? Using the best weak learner, compute α_1 and D_2.

Solution: First, compute the error rates for each weak learner:

$$\varepsilon_1(h_1) = \Pr_{i \sim D_1}[h_1(x_i) \neq y_i] = \frac{2}{8} = 0.375,$$

$$\varepsilon_1(h_2) = \Pr_{i \sim D_1}[h_2(x_i) \neq y_i] = \frac{1}{8} = 0.375,$$

$$\varepsilon_1(h_3) = \Pr_{i \sim D_1}[h_3(x_i) \neq y_i] = \frac{3}{8} = 0.5.$$

Both h_1 and h_2 have the lowest error rate of 0.375. We can choose either as the best weak learner. Let's choose h_1.

$$\alpha_1 = \frac{1}{2} \ln \frac{1 - 0.375}{0.375} = \frac{1}{2} \ln \frac{0.625}{0.375} = \frac{1}{2} \ln \frac{5}{3} \approx 0.2554.$$

Let's compute $D_2(i)$ for each i:

$$D_2(i) \propto 1/8 \cdot \exp(-\alpha_1) = 0.09.$$

where i is classified correctly by h_1 (i.e., $i = 1, 3, 4, 5, 7$).

$$D_2(i) \propto 1/8 \cdot \exp(\alpha_1) = 0.1613.$$

where i is misclassified by h_1 (i.e., $i = 2, 6, 8$).

(b) (5 points) Explain how AdaBoost's reweighting helps when rare positive (+1) emails are initially misclassified by obvious word stumps like h1 (free).

Solution: In this example, we had weight of 0.1613 for the misclassified spam emails, which is higher than the initial weight of 0.125. This increase in weight means that in the next round of boosting, the algorithm will pay more attention to these misclassified positive emails. As a result, it is more likely to select a weak learner that can correctly classify these rare positive emails, which improves the overall performance of the ensemble model.

(c) (5 points) Explain how AdaBoost can incorporate a learner that does worse than random, in other words a learner for which $\varepsilon_t > 0.5$.

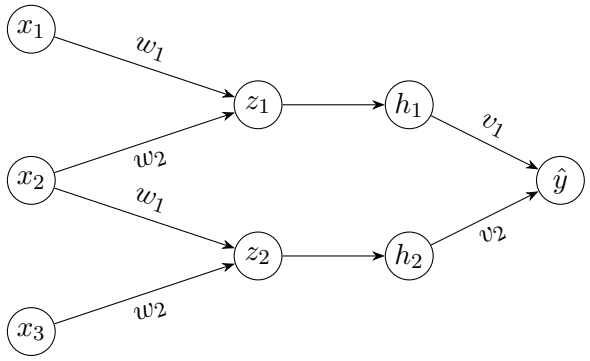
Solution:

$$H_t(x) = \text{sign}\left(\sum_{s=1}^{t} \alpha_s h_s(x)\right).$$

If a weak learner h_t has an error rate $\varepsilon_t > 0.5$, which means α_t is small. AdaBoost incorporate these weak learner but it gives it smaller α as the error get larger, so it has less influence on the final prediction. This way, AdaBoost can still benefit from the weak learner's contribution, even if it performs worse than random guessing.

Question 2 ..... *15 points*

Consider the following mini convolutional neural net, where (x_1, x_2, x_3) is the input, followed by a convolution layer with a filter (w_1, w_2), a ReLU layer, and a fully connected layer with weight (v_1, v_2).



More concretely, the computation is specified by

$$z_1 = x_1 w_1 + x_2 w_2, \quad z_2 = x_2 w_1 + x_3 w_2,$$

$$h_1 = \max\{0, z_1\}, \quad h_2 = \max\{0, z_2\},$$

$$\hat{y} = h_1 v_1 + h_2 v_2.$$

We assume the true labels y are -1 or 1, so that $(x, y) \in \mathbb{R}^3 \times \{-1, +1\}$.

The logistic loss CNN can then be written as follows

$$\ell(y, \hat{y}) = \ln(1 + \exp(-y\hat{y})),$$

which is a function of the parameters of the network: w_1, w_2, v_1, v_2.

(a) (5 points) Write down $\frac{\partial \ell}{\partial v_1}$ and $\frac{\partial \ell}{\partial v_2}$ (show the intermediate steps that use chain rule). You can use the sigmoid function $\sigma(z) = \frac{1}{1+e^{-z}}$ to simplify your notation.

Solution:

$$\frac{\partial \ell}{\partial \hat{y}} = -\frac{y}{1 + e^{y\hat{y}}} = -y\sigma(-y\hat{y}).$$

$$\frac{\partial \hat{y}}{\partial v_1} = h_1, \quad \frac{\partial \hat{y}}{\partial v_2} = h_2.$$

$$\frac{\partial \ell}{\partial v_1} = \frac{\partial \ell}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial v_1} = \boxed{-y\sigma(-y\hat{y})h_1},$$

$$\frac{\partial \ell}{\partial v_2} = \frac{\partial \ell}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial v_2} = \boxed{-y\sigma(-y\hat{y})h_2}.$$

(b) (5 points) Write down $\frac{\partial \ell}{\partial w_1}$ and $\frac{\partial \ell}{\partial w_2}$ (show the intermediate steps that use chain rule). The derivative of the ReLU function is $H(a) = \mathbf{1}[a > 0]$, which you can use directly in your answer. In particular (by convention), we take the gradient of the ReLU function to be zero when $a = 0$.[1]

Solution:

$$\frac{\partial \ell}{\partial w_1} = \frac{\partial \ell}{\partial \hat{y}} \cdot \left(\frac{\partial \hat{y}}{\partial h_1} \cdot \frac{\partial h_1}{\partial z_1} \cdot \frac{\partial z_1}{\partial w_1} + \frac{\partial \hat{y}}{\partial h_2} \cdot \frac{\partial h_2}{\partial z_2} \cdot \frac{\partial z_2}{\partial w_1} \right).$$

We already have $\frac{\partial \ell}{\partial \hat{y}} = -y\sigma(-y\hat{y})$.

$$\frac{\partial \hat{y}}{\partial h_1} = v_1, \quad \frac{\partial h_1}{\partial z_1} = H(z_1), \quad \frac{\partial z_1}{\partial w_1} = x_1,$$

$$\frac{\partial \hat{y}}{\partial h_2} = v_2, \quad \frac{\partial h_2}{\partial z_2} = H(z_2), \quad \frac{\partial z_2}{\partial w_1} = x_2.$$

$$\frac{\partial \ell}{\partial w_1} = -y\sigma(-y\hat{y})(v_1 H(z_1)x_1 + v_2 H(z_2)x_2) = \boxed{-y\sigma(-y\hat{y})(v_1 H(z_1)x_1 + v_2 H(z_2)x_2)}.$$

Similarly,

$$\frac{\partial \ell}{\partial w_2} = -y\sigma(-y\hat{y})(v_1 H(z_1)x_2 + v_2 H(z_2)x_3) = \boxed{-y\sigma(-y\hat{y})(v_1 H(z_1)x_2 + v_2 H(z_2)x_3)}.$$

[1] While the ReLU function is not differentiable at $a = 0$, zero is a *subgradient*, a generalization of the gradient to convex non-differentiable functions.

(c) (3 points) We often initialize the weights of neural networks randomly. Suppose we ignored this practice, and instead we initialized all of the parameters v_1, v_2, w_1, w_2 as zero. What is the value of $z_1, z_2, h_1, h_2, \hat{y}$ with this initialization? What happens to the gradient in this case? In conclusion, why is this a poor choice of initialization?

Solution: all gradients are zero, the weights will not be updated during training. $\hat{y}$ will be zero, and the loss will always be 2. This means the model will not learn from the data, making this a poor choice of initialization.

(d) (2 points) Using the derivations above, fill in the missing details of the backpropagation algorithm below that is used to train this mini CNN.

Algorithm 1 Backpropagation for the above mini CNN

Input: A training set $(x_1, y_1), \ldots, (x_n, y_n)$
Initialize: set w_1, w_2, v_1, v_2 randomly
Randomly pick an example (x_i, y_i)
Forward propagation:

Solution: compute

$$z_1 = x_1 w_1 + x_2 w_2, \quad z_2 = x_2 w_1 + x_3 w_2,$$

$$h_1 = \max(0, z_1), \quad h_2 = \max(0, z_2), \quad \hat{y} = v_1 h_1 + v_2 h_2$$

Backward propagation:

Solution:
compute

$$g_{w_1} = -y\sigma(-y\hat{y})(v_1 H(z_1) x_1 + v_2 H(z_2) x_2),$$

$$g_{w_2} = -y\sigma(-y\hat{y})(v_1 H(z_1) x_2 + v_2 H(z_2) x_3),$$

$$g_{v_1} = -y\sigma(-y\hat{y}) h_1, \qquad g_{v_2} = -y\sigma(-y\hat{y}) h_2.$$

Output: return $g_{w_1}, g_{w_2}, g_{v_1}, g_{v_2}$

Question 3 ..... *15 points*

Dropout Regularization and Variance

One approach to regularization in a neural network model is dropout. During training, dropout is applied independently to each unit (either inputs x_i, or hidden units h_{li}), setting them to zero with probability p, and otherwise scaling them by $1/(1-p)$. This is only applied during model training: during final model evaluation, the units are unchanged.

(a) (5 points) We will consider a simple linear model $\hat{y} = w^\top x$, with dropout applied to the inputs x. To start, derive the expectation and covariance matrix of the effective input vector $\tilde{x}$ when dropout is applied. Treat the inputs x as fixed, so that dropout is the only source of randomness (and hence the only source of randomness to consider in computing expectation and covariance).

Solution:

$$z_i = \begin{cases} 0 & \text{with probability } p \\ \frac{1}{1-p} & \text{with probability } 1-p \end{cases}$$

Expectation:

$$\mathbb{E}[\tilde{x}_i] = E[x_i z_i] = x_i E[z_i] = x_i \cdot (p \cdot 0 + (1-p) \cdot \frac{1}{1-p}) = x_i$$

$$\mathbb{E}[\tilde{x}] = x$$

Covariance: For $i \neq j$, dropout is independent, so

$$\text{Cov}[\tilde{x}_i, \tilde{x}_j] = 0$$

$$\text{Var}[\tilde{x}_i] = \text{Var}[x_i z_i] = x_i^2 \text{Var}[z_i]$$

$$\text{Var}[z_i] = E[z_i^2] - (E[z_i])^2 = (p \cdot 0^2 + (1-p) \cdot (\frac{1}{1-p})^2) - 1^2 = \frac{p}{1-p}$$

$$\text{Var}[\tilde{x}_i] = x_i^2 \frac{p}{1-p}$$

Thus, the covariance matrix is just the diagonal with entries:

$$\text{Cov}[\tilde{x}] = \text{diag}\left(x_1^2 \frac{p}{1-p}, \ldots, x_d^2 \frac{p}{1-p} \right)$$

This tells us that the variance of each input feature is scaled by its squared value and the dropout probability.

(b) (7 points) Again, treating the data y, x and the weights w as fixed, show that the expected squared loss under dropout (i.e., during training) can be written as the original squared loss plus an extra penalty on the weights.

Solution:

$$\mathbb{E}[(y - w^\top \tilde{x})^2] = \mathbb{E}[y^2 - 2yw^\top \tilde{x} + (w^\top \tilde{x})^2]$$

Since y and w are fixed, we have:

$$= y^2 - 2yw^\top \mathbb{E}[\tilde{x}] + w^\top \mathbb{E}[\tilde{x}\tilde{x}^\top]w$$

We already computed $\mathbb{E}[\tilde{x}] = x$. Now, we need to compute $\mathbb{E}[\tilde{x}\tilde{x}^\top]$:

$$\mathbb{E}[\tilde{x}\tilde{x}^\top] = \text{Cov}[\tilde{x}] + \mathbb{E}[\tilde{x}]\mathbb{E}[\tilde{x}]^\top = \text{diag}\left(x_1^2 \frac{p}{1-p}, \ldots, x_d^2 \frac{p}{1-p}\right) + xx^\top$$

So,

$$w^\top \mathbb{E}[\tilde{x}\tilde{x}^\top]w = w^\top xx^\top w + w^\top \text{diag}\left(x_1^2 \frac{p}{1-p}, \ldots, x_d^2 \frac{p}{1-p}\right) w$$

The first term is just $(w^\top x)^2$, and the second term can be written as:

$$= \sum_{i=1}^{d} (w_i^2 x_i^2) \frac{p}{1-p}$$

Putting it all together:

$$= y^2 - 2yw^\top x + (w^\top x)^2 + \sum_{i=1}^{d} (w_i^2 x_i^2) \frac{p}{1-p}$$

$$= (y - w^\top x)^2 + \sum_{i=1}^{d} (w_i^2 x_i^2) \frac{p}{1-p}$$

(c) (3 points) Interpret the penalty in (b). How is it related to L_2 (ridge) regularization? When does it differ?

Solution: let's assume that $\lambda = \frac{p}{1-p}$, then the penalty term can be written as:

$$\lambda \sum_{i=1}^{d} w_i^2 x_i^2$$

This is similar to L_2 regularization, which penalizes the sum of the squares of the weights. However, in this case, the penalty is weighted by the square of the input features x_i^2. This means that features with larger values will contribute more to the penalty, effectively encouraging the model to reduce the weights associated with those features more than those with smaller values. This differs from standard L_2 regularization, which treats all weights equally regardless of the input feature values.

Question 4 ..... *20 points*

Bagging (Bootstrap Aggregating) is a technique designed to reduce the variance of a learning algorithm. Consider a regression setting where the true function output is $f(\mathbf{x})$, and a single base learner trained on a specific bootstrap sample predicts $\hat{f}(\mathbf{x})$.

Assume the following properties for the distribution of base learner predictions $\hat{f}(\mathbf{x})$:

- **Unbiasedness:** The expected prediction equals the true function output: $E[\hat{f}(\mathbf{x})] = f(\mathbf{x})$.
- **Variance:** The variance of the prediction is $\text{Var}[\hat{f}(\mathbf{x})] = \sigma^2$.
- **Covariance:** The covariance between the predictions of any two *different* base learners, i and j, is $\text{Cov}[\hat{f}_i(\mathbf{x}), \hat{f}_j(\mathbf{x})] = \rho\sigma^2$, where $\rho \in [0, 1]$ is the correlation coefficient.

The bagging ensemble uses M base learners, and its final prediction $\hat{F}(\mathbf{x})$ is their average:

$$\hat{F}(\mathbf{x}) = \frac{1}{M}\sum_{i=1}^{M} \hat{f}_i(\mathbf{x})$$

(a) (5 points) Calculate the expected prediction $E[\hat{F}(\mathbf{x})]$ for the Bagging ensemble. Use the expected prediction to determine the bias of the ensemble, $\text{Bias}[\hat{F}(\mathbf{x})] = E[\hat{F}(\mathbf{x})] - f(\mathbf{x})$.

Solution:

$$E[\hat{F}(\mathbf{x})] = E\left[\frac{1}{M}\sum_{i=1}^{M} \hat{f}_i(\mathbf{x})\right] = \frac{1}{M}\sum_{i=1}^{M} E[\hat{f}_i(\mathbf{x})] = \frac{1}{M}\sum_{i=1}^{M} f_i(\mathbf{x}) = f(\mathbf{x})$$

Therefore, the bias of the ensemble is:

$$\text{Bias}[\hat{F}(\mathbf{x})] = E[\hat{F}(\mathbf{x})] - f(\mathbf{x}) = f(\mathbf{x}) - f(\mathbf{x}) = 0$$

(b) (10 points) Calculate the variance of the Bagging ensemble's prediction, $\text{Var}[\hat{F}(\mathbf{x})]$. Your derivation should clearly show the sum of the M variance terms and the sum of the $M(M-1)$ covariance terms. Show that the final result can be expressed in the form:

$$\text{Var}[\hat{F}(\mathbf{x})] = \frac{\sigma^2}{M} + \rho\sigma^2\left(1 - \frac{1}{M}\right)$$

Hint: $\text{Var}\left[\sum_{i=1}^{M} X_i\right] = \sum_{i=1}^{M} \text{Var}[X_i] + \sum_{i \neq j} \text{Cov}[X_i, X_j]$.

Solution:

$$\text{Var}[\hat{F}(\mathbf{x})] = \text{Var}\left[\frac{1}{M}\sum_{i=1}^{M}\hat{f}_i(\mathbf{x})\right] = \frac{1}{M^2}\text{Var}\left[\sum_{i=1}^{M}\hat{f}_i(\mathbf{x})\right]$$

$$= \frac{1}{M^2}\left(\sum_{i=1}^{M}\text{Var}[\hat{f}_i(\mathbf{x})] + \sum_{i\neq j}\text{Cov}[\hat{f}_i(\mathbf{x}), \hat{f}_j(\mathbf{x})]\right)$$

$$= \frac{1}{M^2}\left(M\sigma^2 + \frac{M!}{(M-2)!}\rho\sigma^2\right)$$

$$= \frac{1}{M^2}\left(M\sigma^2 + M(M-1)\rho\sigma^2\right)$$

$$\text{Var}[\hat{F}(\mathbf{x})] = \frac{\sigma^2}{M} + \rho\sigma^2\left(1 - \frac{1}{M}\right)$$

(c) (5 points) Analyze the ensemble's variance as the number of learners M becomes very large ($M \to \infty$).

1. If the base learners were **perfectly uncorrelated** ($\rho = 0$), what would $\text{Var}[\hat{F}(\mathbf{x})]$ converge to?
2. If the base learners were **perfectly correlated** ($\rho = 1$), what would $\text{Var}[\hat{F}(\mathbf{x})]$ converge to?

In one to two lines, explain what these limiting results reveal about *why randomness is required* for Bagging to be successful in practice.

Solution: based on my knowledge, I know that as M approaches infinity, the variance of the ensemble's prediction converges to 0, and the bias will be bigger if ρ is 0, and vise versa. Lets prove that:

$$\text{Var}[\hat{F}(\mathbf{x})] = \frac{\sigma^2}{M} + \rho\sigma^2\left(1 - \frac{1}{M}\right)$$

$$lim_{M\to\infty}\text{Var}[\hat{F}(\mathbf{x})] = lim_{M\to\infty}\left(\frac{\sigma^2}{M} + \rho\sigma^2\left(1 - \frac{1}{M}\right)\right)$$

$$= \rho\sigma^2$$

As I mentioned earlier, if the base learners were perfectly uncorrelated ($\rho = 0$), variance would converge to 0. If the base learners were perfectly correlated ($\rho = 1$), variance would converge to σ^2. This shows that randomness is required for Bagging to be successful in practice because it helps to reduce the correlation between base learners, (the wisdom of crowds).

Part 2: Programming Questions

Please see Canvas for the link to the Programming Assignment