CS 475-675 Machine Learning: Midterm 1

Fall 2021

150 points.

Name (print): Chang Yan

JHED: cyan13

If you think a question is unclear or multiple answers are reasonable, please write a brief explanation of your answer, to be safe. Also, show your work if you want wrong answers to have a chance at some credit: it lets us see how much you understood.

This exam is open-book: permitted materials include textbooks, personal notes, lecture material, recitation material, past assignments, the course Piazza, and scholarly articles and papers. Other materials are otherwise not permitted. It is not permitted to discuss or share questions or solutions of this exam with any person, via any form of communication, other than the course instructors. It is not permitted to solicit or use any solutions to past exams for this course.

Declaration:

I have neither given nor received any unauthorized aid on this exam. In particular, I have not spoken to any other student about any part of this exam. The work contained herein is wholly my own. I understand that violation of these rules, including using an unauthorized aid, copying from another person, or discussing this exam with another person in any way, may result in my receiving a 0 on this exam.

Chang Yan	2021/10/29
Signature	Date

Good luck!

True/False (50 points)

For each question, circle (or otherwise clearly indicate) either True or False. If False, explain why.
2 points for correct True/False answer, -2 points for incorrect True/False answer, 3 points for a correct explanation, 0 points for an incorrect explanation.

1) Minimizing the exponential loss is a good way of creating a classifier robust to outliers.
True False
Explanation:
My answer: False. Explanation: An exponential loss grows exponentially with the error, so when we have outliers, they would case the exponential loss to be extremely high and the classifier will try hard to fit those outliers. Thus, minimizing exponential loss is not robust to outliers.

2) Minimising the 0-1 loss on the training data can never achieve good out of sample prediction performance.
True False
Explanation:
My answer: False. Explanation: If our underlying distributions of the two classes are completely separated (i.e. there is no or only few overlap), and our sample is a good representation of the underlying distributions, we can still get a good out of sample prediction performance using the 0-1 loss.

3) Naive Bayes is a linear classifier.
True False
Explanation:
My answer: True. Explanation: Naive Bayes classifier has linear decision boundary after some (class pair specific) single feature transformations.

4) Decreasing the bias of a predictor must necessarily increase its variance.
True False
Explanation:
My answer: True. Explanation: bias and variance of an estimator are complementary to each other, as we know from the bias/variance decomposition that $Err(x_0)$ = variance + squared bias, when minimizing the error there is always a trade-off: decreasing bias will always increasing variance.

5) $\mathbb{E}[\mathbb{E}[A \mid B]] = \mathbb{E}[A]$.
True False
Explanation:
My answer: True. Explanation:

$$\begin{aligned}
\mathbb{E}[\mathbb{E}[A \mid B]] &= \int \mathbb{E}(A \mid B = b) f_B(b) db \\
&= \int \int a f_{A|B}(a \mid b) da f_B(b) db \\
&= \int \int a f_{A|B}(a \mid b) f_B(b) da db \\
&= \int \int a f_{A,B}(a, b) da db \\
&= \int a \int f_{A,B}(a, b) db da \\
&= \int a f_A(a) da \\
&= \mathbb{E}[A]
\end{aligned}$$

6) Rosenblatt's Perceptron algorithm can be modified to solve multiclass classification problems, provided any pair of classes is linearly separable.
True False
Explanation:
My answer: True. Explanation: We just need to do one-hot encode for each of the N classes (creating N label sets for the N classes, in each row set the label of this class = 1 and other classes = -1) and train N different predictors $\hat{f}_k(\vec{x}_i)$ for each class in data. We know each pair of classes is linearly separable, so each single class verses all other classes combined should also be separable. If the separation is not linear, we can use kernel trick or other high dimensional transformations to find a hyperplane to separate them, and still use Rosenblatt's Perceptron algorithm to find each hyperplanes. Then in prediction we pick the class with biggest $\hat{f}_k(\vec{x}_i)$.

7) Naive Bayes is a special case of a Markov random field.
True False
Explanation:
My answer: True. Explanation: Naive Bayes can both be represented by a directional graph (Bayesian Networks) and Undirectional graph (MRF). We can construct a MRF that represents a naive bayes. We can just put the node Y in the middle, and all X_i are connected to the node Y. There is no other connections. In this MRF, we can see we have $X_i \perp\!\!\!\perp X_j \mid Y$ for all i, j. Note that this is the sample independence relationships as in a Naive Bayes. So the joint probability is simply $p(\vec{X}, Y) = p(Y) \sum p(X_i \mid Y)$.This is also the same as the joint probability of Naive Bayes in a directional graph, so it is true that Naive Bayes can be represented using MRF.

8) If $A \perp\!\!\!\perp C$ then either $A \perp\!\!\!\perp B$ or $B \perp\!\!\!\perp C$.
True False
Explanation:
My answer: False. Explanation: We can find a random variable B that is dependent to both A and C when A and C are independent. For example, if A describes if event A happens, and C describes if event C happens. Lest event A and C be independent events, so A and C are independent random variables. We can let B be a random variable describing the situation that both event A and event C happens. We can see clearly that A and B are dependent, and B and C are dependent. However, A and C are independent. This situation is possible, so the statement above is false.

9) A support vector machine applied to a linearly separable dataset must have at least 2 support vectors.
True False
Explanation:
My answer: True. Eexplanation: We need at least one vector for the boundary at one side and at least one another vector for the boundary at the another side, and the minimum distance between them is the width. Thus, at least two is needed.

10) Logistic regression is a special case of the projection pursuit model with $M = 1$.
True False
Explanation:
My answer: True. Eexplanation: logistic regression only has one $g(\vec{t})$ which is the sigmoid function, and we know projection pursuit models are combination of M number of unrestricted $g_m(\vec{t})$ s. Thus, logistic regression fullfills this requirement and it only has one $g(\vec{t})$, so in this case we have M = 1, and it is a projection pursuit model.

Multiple Part Questions (100 points)

11) Probability And Likelihood (25 points)

(i) Show that if $A \perp\!\!\!\perp B \cup D \mid C$ then $A \perp\!\!\!\perp D \mid C$. You may want to use the fact that if $X \perp\!\!\!\perp Y \mid Z$ then $p(X, Y \mid Z) = p(X \mid Z)p(Y \mid Z)$.

Answer: We know $A \perp\!\!\!\perp B \cup D \mid C$ can also be written as $A \perp\!\!\!\perp B, D \mid C$, and apply the independence relationship we have:

$$p(A, B, D \mid C) = p(A \mid C)p(B, D \mid C)$$

Then we can sum over all values of B, we will get:

$$\sum_B p(A, B, D \mid C) = \sum_B p(A \mid C)p(B, D \mid C)$$
$$\sum_B p(A, B, D \mid C) = p(A \mid C) \sum_B p(B, D \mid C)$$
$$p(A, D \mid C) = p(A \mid C)p(D \mid C)$$

Using the theorem of independence, the above equation tells us that: $A \perp\!\!\!\perp D \mid C$.

(ii) Consider the following density function (for non-negative x): $f(x; \lambda) = \lambda \exp\{-\lambda x\}$. Derive the maximum likelihood estimate of its parameter λ.
Answer:

$$\begin{aligned} L_{[D]}(\lambda) &= \prod_{i=1}^{n} p(x_i, y_i; \lambda) \\ &= \prod_{i=1}^{n} \lambda \exp\{-\lambda x_i\} \\ &= \lambda^n \prod_{i=1}^{n} e^{-\lambda x_i} \end{aligned}$$

Take the log-likelihood:

$$log L_{[D]}(\lambda) = log \lambda^n + \sum_{i=1}^{n} (-\lambda x_i)$$

Then take derivative with respect to λ and set it to 0

$$\frac{\partial log L_{[D]}(\lambda)}{\partial \lambda} = \frac{n}{\lambda} - \sum_{i=1}^{n} x_i = 0$$
$$\frac{n}{\lambda} = \sum_{i=1}^{n} x_i$$
$$\lambda = \frac{n}{\sum_{i=1}^{n} x_i}$$

So the MLE estimation of λ is $\frac{n}{\sum_{i=1}^{n} x_i}$ where n is number of samples

(iii) What function of x does λ correspond to?
Answer: if we have $X = x_i$ for all i (perfect approximation), we have:

$$\lambda = \frac{n}{\sum_{i=1}^{n} X} = \frac{n}{nX} = \frac{1}{X}$$

So the function λ correspond to is $\frac{1}{X}$.

(iv) If, instead of solving for λ in closed form, we instead opted to use stochastic gradient descent to update our guess $\lambda_{(t)}$ to obtain $\lambda_{(t+1)}$, using the ith row of data x_i, and a learning parameter ρ, what would our update be?
Answer: The update is:

$$\begin{aligned}
\lambda_{(t+1)} &= \lambda_{(t)} - \rho \nabla f(x; \lambda) \\
&= \lambda_{(t)} - \rho \frac{\partial f(x; \lambda)}{\partial \lambda} \\
&= \lambda_{(t)} - \rho \frac{\partial \lambda e^{-\lambda x_i}}{\partial \lambda} \\
&= \lambda_{(t)} - \rho (e^{-\lambda_{(t)} x_i} - \lambda_{(t)} x_i e^{-\lambda_{(t)} x_i}) \\
&= \lambda_{(t)} - \rho (1 - \lambda_{(t)} x_i) e^{-\lambda_{(t)} x_i}
\end{aligned}$$

So we have update rule:

$$\lambda_{(t+1)} = \lambda_{(t)} - \rho (1 - \lambda_{(t)} x_i) exp(-\lambda_{(t)} x_i)$$

12) **Classification (25 points)**

Consider the *Slightly Less Naive Bayes* classifier on features $X_1, \ldots, X_k$ and outcome Y, defined by the following independence restrictions:

$$X_1 \perp\!\!\!\perp X_3, \ldots, X_k \mid X_2, Y$$
$$X_k \perp\!\!\!\perp X_1, \ldots, X_{k-2} \mid X_{k-1}, Y$$
$$X_i \perp\!\!\!\perp X_1, \ldots, X_{i-2}, X_{i+2}, \ldots, X_k \mid X_{i-1}, X_{i+1}, Y$$

(i) Write down the joint likelihood $\mathcal{L}_{[D]} \equiv \prod_{i=1}^{n} p(\vec{x}_i, y_i)$ for this model. You may want to think about the MRF factorization corresponding to the above independences.
Answer: From the MRF, we know this is just the Naive Bayes plus $X_1, X_2, ...X_k$ are connected like a chain. Thus, we have k-1 cliques in this case, and the probably can be written as:

$$p(\vec{x}_i, y_i) = \frac{1}{Z} \prod_{j=1}^{k-1} \phi_{X_j, X_{j+1}, Y}(x_{ij}, x_{ij+1}, y_i)$$

So the joint likelihood is:

$$\mathcal{L}_{[D]} \equiv \prod_{i=1}^{n} \frac{1}{Z} \prod_{j=1}^{k-1} \phi_{X_j, X_{j+1}, Y}(x_{ij}, x_{ij+1}, y_i)$$

(ii) Write down the predictive model $p(Y \mid \vec{X})$ for this classifier.
Answer:

$$\begin{aligned} p(Y \mid \vec{X}) &= \frac{p(\vec{X}, Y)}{\sum_y p(\vec{X}, y)} \\ &= \frac{\frac{1}{Z} \prod_{j=1}^{k-1} \phi_{X_i, X_{i+1}, Y}(X_i, X_{i+1}, Y)}{\sum_y \frac{1}{Z} \prod_{i=1}^{k-1} \phi_{X_i, X_{i+1}, Y}(X_i, X_{i+1}, y)} \end{aligned}$$

(iii) Write down the decision boundary for classes Y_j and Y_m: $\log \frac{p(Y_j|\vec{X})}{p(Y_m|\vec{X})}$. Is this a linear decision boundary in $\vec{X}$?
Answer:

$$\begin{aligned} \log \frac{p(Y_j \mid \vec{X})}{p(Y_m \mid \vec{X})} &= \log \frac{\frac{1}{Z} \prod_{i=1}^{k-1} \phi_{X_i, X_{i+1}, Y}(X_i, X_{i+1}, Y_j)}{\frac{1}{Z} \prod_{i=1}^{k-1} \phi_{X_i, X_{i+1}, Y}(X_i, X_{i+1}, Y_m)} \\ &= \sum_{i=1}^{k-1} \log \frac{\phi_{X_i, X_{i+1}, Y}(X_i, X_{i+1}, Y_j)}{\phi_{X_i, X_{i+1}, Y}(X_i, X_{i+1}, Y_m)} \end{aligned}$$

This is not a linear decision boundary in $\vec{X}$, but we can have a linear model after some (class pair specific) adjacent feature transformations g_{jm}:

$$\log \frac{p(Y_j \mid \vec{X})}{p(Y_m \mid \vec{X})} = \sum_{i=1}^{k-1} g_{jm}(X_i, X_{i+1})$$

(iv) Is the Naive Bayes model a submodel of the Slightly Less Naive Bayes model? Recall that a model is a set of distributions (that satisfy some independence restrictions in this case), and a submodel is a subset. In other words, all distributions in a submodel (subset) are also in a supermodel (superset). That is, elements of a submodel satisfy restrictions in a supermodel.

Hint: think about the graphoid axioms, and how Naive Bayes independences compare to Slightly Less Naive Bayes independences.

Answer: Yes. In Naive Bayes the condition we need to satisfy is $X_i \perp\!\!\!\perp X_j \mid Y$ for and i and j. Thus, we can first apply the composition axiom multiple times to get the following three relationships:

$$X_1 \perp\!\!\!\perp X_2, X_3, \ldots, X_k \mid Y$$
$$X_k \perp\!\!\!\perp X_1, \ldots, X_{k-2}, X_{k-1} \mid Y$$
$$X_i \perp\!\!\!\perp X_1, \ldots, X_{i-2}, X_{i-1}, X_{i+1}, X_{i+2}, \ldots, X_k \mid Y$$

Then, we apply the weak union axiom to each of them and we get:

$$X_1 \perp\!\!\!\perp X_3, \ldots, X_k \mid X_2, Y$$
$$X_k \perp\!\!\!\perp X_1, \ldots, X_{k-2} \mid X_{k-1}, Y$$
$$X_i \perp\!\!\!\perp X_1, \ldots, X_{i-2}, X_{i+2}, \ldots, X_k \mid X_{i-1}, X_{i+1}, Y$$

This is exactly the restrictions for a Slightly Less Naive Bayes. Thus, we can see that the Naive Bayes satisfies all restriction of Slightly Less Naive Bayes. So, the Naive Bayes model is a submodel of the Slightly Less Naive Bayes model.

13) **Neural Network Models (25 points)**

Consider a neural network with K inputs $\vec{X}$, and output Y.

(i) Assume there are $M > 10$ hidden layers with one node each. The first hidden node V_1 uses the sigmoid activation function $\sigma(x) = 1/(1+e^{-x})$ applied to a linear combination of $\vec{X}$ given by $\vec{\beta}_0^T\vec{X}$, all subsequent hidden nodes V_m $(m = 2, \ldots, M)$ use a sigmoid activation function applied to a weighted combination of the value of the previous hidden node V_{m-1} and a bias term ($V_m = \text{sigmoid}(\beta_{m0} + \beta_{m1}V_{m-1})$), with the output Y being given by $\text{sigmoid}(\beta_{M0} + \beta_{M1}V_M)$.

Describe how the gradients with respect to $\vec{\beta}_0$ differ between layers. Justify your answer. Hint: calculate the derivative of $\sigma(x)$, and think about what will happen by chain rule of differentiation if a set of $\sigma(x)$ functions are composed.

Answer: First compute the derivative of $\sigma(x)$:

$$\begin{aligned}\frac{\partial\sigma(x)}{\partial\vec{\beta}_0} &= \frac{\partial(1+e^{-\vec{\beta}_0^T\vec{X}})^{-1}}{\partial\vec{\beta}_0} \\ &= -(1+e^{-\vec{\beta}_0^T\vec{X}})^{-2}(-\vec{X}e^{-\vec{\beta}_0^T\vec{X}}) \\ &= \frac{\vec{X}e^{-\vec{\beta}_0^T\vec{X}}}{(1+e^{-\vec{\beta}_0^T\vec{X}})^2}\end{aligned}$$

Then we compute the second layer using the chain rule: (m=2)

$$\begin{aligned}\frac{\partial\sigma(\beta_{20}+\beta_{21}\sigma(x))}{\partial\vec{\beta}_0} &= \frac{\partial\sigma(\beta_{20}+\beta_{21}\sigma(x))}{\partial\sigma(x)}\frac{\partial\sigma(x)}{\partial\vec{\beta}_0} \\ &= \frac{\partial(1+e^{-\beta_{20}-\beta_{21}\sigma(x)})^{-1}}{\partial\vec{\beta}_0}\frac{\vec{X}e^{-\vec{\beta}_0^T\vec{X}}}{(1+e^{-\vec{\beta}_0^T\vec{X}})^2} \\ &= \frac{\beta_{21}e^{-\beta_{20}-\beta_{21}\sigma(x)}}{(1+e^{-\beta_{20}-\beta_{21}\sigma(x)})^2}\frac{\vec{X}e^{-\vec{\beta}_0^T\vec{X}}}{(1+e^{-\vec{\beta}_0^T\vec{X}})^2} \\ &= \frac{\beta_{21}e^{-\beta_{20}-\beta_{21}V_1}}{(1+e^{-\beta_{20}-\beta_{21}V_1})^2}\frac{\vec{X}e^{-\vec{\beta}_0^T\vec{X}}}{(1+e^{-\vec{\beta}_0^T\vec{X}})^2}\end{aligned}$$

We can see that this patterns continues for all $m > 2$ layers, and the chain rule multiplier for each layer is just:

$$\frac{\partial\sigma(\beta_{m0}+\beta_{m1}V_{m-1})}{\partial V_{m-1}} = \frac{\beta_{m1}e^{-\beta_{m0}-\beta_{m1}V_{m-1}}}{(1+e^{-\beta_{m0}-\beta_{m1}V_{m-1}})^2}$$

Thus, the gradient at node V_m can be expressed as:

$$\frac{\partial V_m}{\partial\vec{\beta}_0} = \frac{\vec{X}e^{-\vec{\beta}_0^T\vec{X}}}{(1+e^{-\vec{\beta}_0^T\vec{X}})^2}\prod_{i=2}^{m}\frac{\beta_{i1}e^{-\beta_{i0}-\beta_{i1}V_{i-1}}}{(1+e^{-\beta_{i0}-\beta_{i1}V_{i-1}})^2}$$

If we plug in $V_m = \text{sigmoid}(\beta_{m0} + \beta_{m1}V_{m-1})$, we will have:

$$\frac{\partial V_m}{\partial\vec{\beta}_0} = \frac{\vec{X}e^{-\vec{\beta}_0^T\vec{X}}}{(1+e^{-\vec{\beta}_0^T\vec{X}})^2}\prod_{i=2}^{m}\beta_{i1}V_i(1-V_i)$$

Thus, the gradient difference between V_m and V_{m-1} is by a multiple of:

$$\frac{\beta_{m1}e^{-\beta_{m0}-\beta_{m1}V_{m-1}}}{(1+e^{-\beta_{m0}-\beta_{m1}V_{m-1}})^2} = \beta_{m1}V_m(1-V_m)$$

i.e. it is just:

$$\frac{\partial V_m}{\partial \vec{\beta}_0} = \frac{\partial V_{m-1}}{\partial \vec{\beta}_0}\beta_{m1}V_m(1-V_m)$$

(ii) Now replace all $\sigma(x)$ activation functions in the model in (i) by ReLU activation functions.

How does the gradient of the output with respect to $\vec{\beta}_0$ compare with that same gradient in (i)?
Answer: Assume values are all positive now. First compute the 1st layer:

$$\begin{aligned}\frac{\partial ReLU(x)}{\partial \vec{\beta}_0} &= \frac{\partial \vec{\beta}_0^T \vec{X}}{\partial \vec{\beta}_0} \\ &= \vec{X}\end{aligned}$$

for layer 2 it is:

$$\begin{aligned}\frac{\partial ReLU(\beta_{20}+\beta_{21}ReLU(x))}{\partial \vec{\beta}_0} &= \frac{\partial ReLU(\beta_{20}+\beta_{21}ReLU(x))}{\partial ReLU(x)}\frac{\partial ReLU(x)}{\partial \vec{\beta}_0} \\ &= \beta_{21}\vec{X};\end{aligned}$$

this patterns continues for all $m > 2$ layers, and the chain rule multiplier for each layer is just:

$$\frac{\partial ReLU(\beta_{m0}+\beta_{m1}V_{m-1})}{\partial V_{m-1}} = \beta_{m1}$$

Thus, the gradient at output node V_m can be expressed as:

$$\frac{\partial V_M}{\partial \vec{\beta}_0} = \vec{X}\prod_{i=2}^{M}\beta_{i1}$$

From (i) we know the output gradient was:

$$\frac{\partial V_M}{\partial \vec{\beta}_0} = \frac{\vec{X}e^{-\vec{\beta}_0^T\vec{X}}}{(1+e^{-\vec{\beta}_0^T\vec{X}})^2}\prod_{i=2}^{M}\beta_{i1}V_i(1-V_i)$$

The difference is that for ReLU activation, the output gradient is only related to $\vec{X}$ and all the β_{m1}. For the network with sigmoid activation, the output gradient is also related to $\vec{\beta}_0$, all the β_{m0} and the current value of all nodes V_m.

(iii) Assume there is a single hidden layer with K nodes, each with a $ReLU(x)$ activation function, and each feature in $\vec{X}$ is connected to exactly one hidden node. In other words, each hidden node V_k $(k = 1, \ldots, K)$ is equal to $ReLU(\beta_k X_k)$. Furthermore, assume the output Y is given by $\sigma(\vec{\beta}^T(V_1, \ldots, V_K))$. Is this model equivalent to the logistic regression model? Explain.

Answer: This mode is not equivalent to the logistic regression model. We can derive its output function:

$$Y(\vec{X}) = \frac{1}{1 + exp(\sum_{i=0}^{k} \beta'_k ReLU(\beta_k X_k))}$$

This model is different than a logistic regression model because of the ReLU function here. For an original logistic model, if $\beta_k X_k < 0$, the exponentional will still evaluate to a positive value less than one when it's exponent is negative. However, with the ReLU here, if $\beta_k X_k < 0$, the ReLU will just evaluate to 0, and this node will contribute zero to the output. If all nodes are zero, the exponential will just evaluate to 1. Thus, this is not a logistic model as it is not continues, and the negative $\beta_k X_k$ values will just be set to zero.

(iv) As before, assume there is a single hidden layer with K nodes, each with a $\sigma(x)$ activation function. However, now each hidden node V_k $(k = 1, \ldots, K)$ is given by $\sigma()$ applied to a weighted combination of $\vec{X}$, where weights for each V_k are *shared*. The outcome model is given by a weighted combination of $\vec{\alpha}^T(V_1, \ldots, V_K)$. Is this model sufficiently general to be able to approximate the logistic regression model? Explain.

Answer: My initial thought is that this model is sufficiently general to be able to approximate the logistic regression model, but it actually depends on if we allow the output of a logistic model to be larger than one. If we do not allow, then this is not a good approximation because the learned α parameters can be larger than 1, making the output lager than 1. Again, we can write the output function fo this model:

$$Y(\vec{X}) = \frac{\sum_{i=0}^{k} \alpha_k}{1 + exp(-\vec{\beta^T}\vec{X})}$$

Note that if we have $\sum_{i=0}^{k} \alpha_k = 1$, this is exactly a logistic model. Even if it is not set to 1 now, it is still just one constant. As we are just approximating the logistics, and $\sum_{i=0}^{k} \alpha_k$ is learnable parameters, so the network can eventually learn it to 1 to make it a logistics model. If not, it is still a logistic model multiplied by a number, which should be an enough approximation to logistics. However, if we be strict and require that the output of a logistic model cannot be above 1 (i.e. it must be between 0-1), then this is not a good approximation as the learned α parameters may cause the output to be higher than 1.

14) **AdaBoost and Exponential Loss (25 points)**

Assume a one-dimensional dataset with $x = \langle -1, 0, 1\rangle$ and $y = \langle -1, +1, -1\rangle$. Consider three weak classifiers:

$$h_1(x) = \begin{cases} 1 & \text{if } x > \frac{1}{2} \\ -1 & \text{otherwise} \end{cases}, \qquad h_2(x) = \begin{cases} 1 & \text{if } x > -\frac{1}{2} \\ -1 & \text{otherwise} \end{cases}, \qquad h_3(x) = 1.$$

(1) Show your work for the first 2 iterations of AdaBoost. In the table below, replace the "?" in the table below with the value of the quantity at iteration t. For the ε_t column, put its error rate (remember to use the correct row weights). For the h column, put the index of the weak learner with the best error rate at the current iteration. For the α_t column, put the update parameter appropriately calculated from ε_t. In addition, calculate the distribution over examples at the second iteration $D_2(1), D_2(2), D_2(3)$. Use the natural logarithm in your calculations (log with base e).

t	h_t	α_t	ε_t	$D_t(1)$	$D_t(2)$	$D_t(3)$
1	2	0.347	0.333	$\frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{3}$
2	2	0	0.5	0.25	0.25	0.5

Answer: Prediction of h1 on y is (-1, -1, 1), prediction of h2 on y is (-1, 1, 1), prediction of h3 on y is (1 ,1 ,1). The correctness of h1 is (true, false, false), the correctness of h2 is (ture, true, false), the correctness of h3 is (false, true, false). First round error for h1 is 0.5 - 0.5 (1/3-1/3-1/3) = 2/3, h2 is 0.5 - 0.5 (1/3+1/3-1/3) = 1/3, h3 is 0.5 - 0.5 (1/3-1/3-1/3) = 2/3. Now the best one is h2, with error 1/3. $\alpha = \frac{1}{2}log(\frac{1-\epsilon}{\epsilon}) = 0.347$. Next round weight is: D1 = 1/3*exp(-0.347*1) = 0.236, D2 = 1/3*exp(-0.347*1) = 0.236, D3 = 1/3*exp(-0.347*(-1)) = 0.472. Normalize then we have D1 = 0.25, D2 = 0.25, D3 = 0.5. Second round error for h1 is 0.5 - 0.5 (0.25-0.25-0.5) = 0.75, h2 is 0.5 - 0.5 (0.25+0.25-0.5) = 0.5, h3 is 0.5 - 0.5 (-0.25+0.25-0.5) = 0.75. The lowest one is still h2, with error 0.5, and $\alpha = \frac{1}{2}log(\frac{1-\epsilon}{\epsilon}) = 0$

(2) Without performing an explicit calculation, make an argument for what the classifier will be that AdaBoost outputs in this case after 100 iterations? Explain.
Answer: The output classifier will just be

$$f_{final}(\vec{x}) = sign(0.347 * h_2(\vec{x}))$$

This is because in t = 2, we already have $\alpha_t = 0$, meaning that the weights will not be updated. They are just multiplied by 1 as exp(0) = 1.Thus, both ϵ_t and h_t will remains the same. This means all the variable just settle here, and they will not change after t = 2. Thus, after 100 itr, it is still the same. When calculating the output, as when $t > 1$, $\alpha_t = 0$, nothing after iteration two will be counted. So, only the classifier in first iteration will have weight 0.347, and this is h_2. All following weights are just zero.

(3) In general, getting near-perfect training accuracy in machine learning leads to overfitting. However, AdaBoost can get perfect training accuracy without overfitting. Give a brief justification for why that is the case.
Answer: This is because AdaBoost is a particular greedy algorithm that minimizes the exponential loss. Exponential loss increasingly penalizes negative margins, so it is not enough to classify correctly, and the surface is forced to pay attention to the degree of misclassification. This decreases generalization error even after perfect training set accuracy is achieved.

(4) The logistic regression likelihood (with outcome labels $y \in \{1, -1\}$) is equal to $\prod_{i=1}^{n} \frac{1}{1+\exp(-y_i \vec{\beta} \vec{x}_i)}$. Show that maximizing the likelihood is equivalent to minimizing the exponential loss: $\sum_{i=1}^{n} \exp(-y_i f(\vec{x}_i))$ for some $f(.)$.
Answer: First take the log of the likelihood:

$$\begin{aligned} logL(\vec{\beta}) =& log \prod_{i=1}^{n} \frac{1}{1+\exp(-y_i \vec{\beta} \vec{x}_i)} \\ &= 0 - \sum_{i=1}^{n} log(1+\exp(-y_i \vec{\beta} \vec{x}_i)) \\ &= -\sum_{i=1}^{n} log(1+\exp(-y_i \vec{\beta} \vec{x}_i)) \end{aligned}$$

clearly, maximizing this likelihood is equivalent to minimizing:

$$\sum_{i=1}^{n} log(1+\exp(-y_i \vec{\beta} \vec{x}_i))$$

As log(x) is a monotonically increasing function, minimizing log(x) is equivalent to minimizing x. In our case here, it is equivalent to minimizing:

$$\sum_{i=1}^{n} (1+\exp(-y_i \vec{\beta} \vec{x}_i)) = n + \sum_{i=1}^{n} (\exp(-y_i \vec{\beta} \vec{x}_i))$$

As n is just a constant, this is equivalent to minimizing:

$$\sum_{i=1}^{n} (\exp(-y_i \vec{\beta} \vec{x}_i))$$

if we let $\vec{\beta} \vec{x}_i = f(\vec{x}_i)$, as $\vec{\beta} \vec{x}_i$ fulfills the requirement of "some $f(.)$", this is equivalent to minimizing:

$$\sum_{i=1}^{n} \exp(-y_i f(\vec{x}_i))$$

We can see this is exactly the form of exponential loss. Thus, we proved that maximizing the logistic regression likelihood is equivalent to minimizing exponential loss.