CIS5200: Machine Learning Fall 2025

Homework 5

Release Date: October 1, 2025 *Due Date: October 20, 2025*

Overview

There are two parts to Homework 5: written and programming.

- This homework assignment accounts for 1% of the course grade.
- Collaboration: Unless otherwise specified, discussion with classmates is encouraged, but all final work must be your own. Cite collaborators and resources.
- All written homework solutions should be **handwritten** and include your name, pennkey, and an honor statement ("I affirm that I will not give or receive unauthorized assistance on this homework, and that all submitted work is my own").
- Submit your written HW0 as a single PDF file via Gradescope. Instructions for submitting the programming component to Gradescope are included in the Colab notebook.
- The deadline is **11:59 PM ET**. Late HWs will be penalized **33%** per day (with 4 unpenalized late days). We will drop the lowest HW score (which could be a zero). In general, we will not offer exceptions for being sick, having job interviews, etc. Of course, if you have extreme extenuating circumstances such as an extended illness, please reach out to the instructors on Ed.
- You must also sign your name on the provided Honor Statement in the template, affirming that you will not give or receive unauthorized assistance and that all submitted work is your own.

Introduction

Before you start the problems, let's quickly review some key concepts and share a helpful resource. If you'd like a short, intuitive explanation, you can watch this video. Below is a written walk-through of the same ideas.

1. From "surprise" to entropy

Think of a *surprise* (or *surprisal*) as a measure of how unexpected an event is.

- A rare event (low probability) is *more* surprising.
- A common event (high probability) is *less* surprising.

To formalize this we want two properties:

1. **Additivity for independent events**: if two independent events occur, their combined surprise should add.
2. **Inverse relationship with probability**: less likely $\Rightarrow$ more surprise.

The logarithm gives us exactly that. For an event s with probability $P(s)$,

$$H(s) = -\log P(s)$$

is its surprisal.

A whole probability distribution contains many possible events. To describe its overall unpredictability, we average the surprisal across all events:

$$\boxed{H(P) = -\sum_x P(x)\ \log P(x)}.$$

This average surprise is called the *entropy* of P. Entropy is high for diffuse, uncertain distributions and low when outcomes are predictable.

2. Cross-entropy: measuring surprise under the wrong belief

In practice we rarely know the true data-generating distribution P. Instead we collect a dataset (an *empirical distribution*) and train a model Q to approximate P.

Suppose we *believe* the world follows Q, but the actual data follow P. The *cross-entropy*

$$\boxed{H(P, Q) = -\sum_x P(x)\ \log Q(x)}$$

measures the average surprise when reality is P but our belief is Q.

It is the expected number of "bits of surprise" we feel on real data when our mental model is Q.

Two things contribute to that surprise:

- **Intrinsic uncertainty of** P – the randomness in the true data itself.
- **Model mismatch** – any error from using Q instead of the true P.

Notice that if $Q = P$, the cross-entropy collapses to the entropy:

$$H(P, P) = H(P).$$

3. KL divergence: isolating model mismatch

The *Kullback–Leibler (KL) divergence* extracts just the extra surprise caused by using a wrong model:

$$\boxed{\mathrm{KL}(P\|Q) = H(P, Q) - H(P)} = \sum_x P(x) \log \frac{P(x)}{Q(x)}.$$

It is always non-negative and equals zero only when $Q = P$.

You can think of KL divergence as the "penalty" or information loss when you approximate the true distribution P with the model Q.

Summary Table

Concept	**Formula**	**Intuition**
Entropy $H(P)$	$-\sum P \log P$	Average surprise if you know the true distribution
Cross-entropy $H(P, Q)$	$-\sum P \log Q$	Surprise when reality is P but you believe Q
KL divergence $\mathrm{KL}(P\|Q)$	$\sum P \log \frac{P}{Q}$	Extra surprise from believing Q instead of P

1 Written Questions

Q1 **[KL Divergence]** We revisit the Kullback–Leibler (KL) divergence as a loss for comparing two probability distributions p and q. In typical ML workflows, one distribution comes from the *data* and the other from a *model*. Training aims to align these distributions as closely as possible. Because KL is *asymmetric*, the order $(p\|q)$ versus $(q\|p)$ matters.

Setup & notation Let X be a *discrete* random variable. Consider two discrete distributions $P(x)$ and $Q(x)$, given as vectors of frequencies for each outcome x. We define

$$Z_p \;=\; \sum_x P(x), \qquad p(x) \;=\; \frac{P(x)}{Z_p} \quad \big(\text{optionally } Z_q = \textstyle\sum_x Q(x),\; q(x) = Q(x)/Z_q\big).$$

The KL divergence is

$$\mathrm{KL}(p\|q) \;=\; \sum_x p(x)\,\log\frac{p(x)}{q(x)}.$$

We use a parametric model $q_W(x)$ with weights W; thus $q = q_W$.

1. Formulate $\mathrm{KL}(q_W\|p)$ as an optimization problem in terms of $P(x)$, q_W, and Z_p (with $Z_p = \sum_x P(x)$). *Write it explicitly as an* $\arg\min$ *over* W*:*

   $$\arg\min_W \;\cdots$$

   Then algebraically simplify the objective so that any constants (with respect to W) are grouped/eliminated.

2. *Follow the instruction in (1)*, but now derive the $\arg\min$ formulation for $\mathrm{KL}(p\|q_W)$ in terms of $P(x)$, q_W, and Z_p, and simplify similarly.

3. Indicate whether $\mathrm{KL}(q_W\|p)$ or $\mathrm{KL}(p\|q_W)$ is generally easier to compute in practice, and *justify your choice. Hint:* Compare your objectives from (1)–(2) and consider which terms are straightforward to estimate from data (e.g., empirical sums under p) versus those that require expectations under the model q_W.

4. Examine your $\arg\min$ expression from (2) for $\mathrm{KL}(p\|q_W)$. *Do you recognize its resemblance to a standard loss used in supervised classification (cross-entropy / negative log-likelihood)?* Explain the connection precisely and state whether your objective differs from cross-entropy by any constant term independent of W.

Part 1: Ensembles

Q1 [*Concept Check*] **[AdaBoost]** AdaBoost is a seminal boosting algorithm that combines weak learners (often decision stumps) into a strong classifier. Its core innovation is adaptively reweighting the training dataset at each iteration, focusing more on instances that were previously misclassified. The weight update rule and the final aggregation method are key to understanding its mechanics.

Explain the role of the coefficient α_t in the AdaBoost algorithm. Specifically, answer the following:

1. Write the formula for α_t and describe how it is a function of the weighted error ϵ_t of the weak learner at step t.

2. Intuitively, why does this formula make sense? What does a large α_t value imply about the performance of the t-th weak learner?

3. How does α_t influence the final aggregated classifier $H(\mathbf{x}) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(\mathbf{x})\right)$?

Q2 [*Application / Real-World Motivation*] **[Gradient Tree Boosting]**
While bagging methods like Random Forests are excellent for high-variance models, boosting methods like Gradient Boosting Machines (GBM), particularly with decision trees as weak learners, excel at reducing both bias and variance. They are dominant in tabular data competitions (e.g., Kaggle) due to their predictive power. A key consideration in real-world applications is managing complexity to prevent overfitting, which is controlled by parameters like the learning rate, tree depth, and number of estimators.

You are a data scientist building a fraud detection system for credit card transactions. The dataset is highly imbalanced (99% legitimate, 1% fraudulent). You decide to use a Gradient Tree Boosting model (e.g., XGBoost or LightGBM).

1. Why is an ensemble method like Gradient Boosting potentially better for this task than a single deep decision tree?

2. Describe two specific techniques or parameters you would use within the Gradient Boosting framework to address the class imbalance problem.

3. Compared to a Random Forest model, what is a potential advantage and a potential disadvantage of using Gradient Boosting for this task, considering model performance and operational constraints?

Q3 [*Short Proof/Reasoning*] **[Why Ensembles are Good (Variance Reduction in Bagging]**
Bagging (Bootstrap Aggregating) reduces the variance of a base estimator. The core idea is to train multiple models on bootstrapped samples of the training data and average their predictions. For an unstable estimator (high variance, low bias) like a deep decision tree, this averaging process stabilizes the predictions.
Assume we have T independent and identically distributed (i.i.d.) random variables, $Z_1, Z_2, ..., Z_T$, each with variance σ^2. The variance of their average, $\bar{Z} = \frac{1}{T}\sum_{t=1}^{T} Z_t$, is σ^2/T.
In bagging, however, the base learners h_t trained on different bootstrap samples are *not* independent; they are positively correlated because they are all trained on data drawn from the same underlying distribution. Let ρ be the correlation between any two learners, and let each learner have variance σ^2.
Show that the variance of the bagged ensemble $\bar{h} = \frac{1}{T}\sum_{t=1}^{T} h_t$ is:

$$\rho\sigma^2 + \frac{1-\rho}{T}\sigma^2$$

Explain why this formula demonstrates the variance-reduction effect of bagging and its limit.

Q4 [*Proof / Multi-Step Problem*] **[Boosting (AdaBoost) - Minimizing Exponential Loss]**
The AdaBoost algorithm can be derived from the perspective of forward stagewise additive modeling, where the goal is to minimize an exponential loss function: $L(y, F(\mathbf{x})) = \exp(-yF(\mathbf{x}))$, where $y \in \{-1, +1\}$ and $F(\mathbf{x})$ is the additive model.
Let $F(\mathbf{x}) = \sum_{t=1}^{T} \alpha_t h_t(\mathbf{x})$ be an ensemble of weak classifiers $h_t(\mathbf{x}) \in \{-1, +1\}$. We want to minimize the exponential loss over the training data: $J(F) = \sum_{i=1}^{N} \exp(-y_i F(\mathbf{x}_i))$.
Assume we have already built the ensemble up to stage $t-1$, obtaining $F_{t-1}(\mathbf{x})$. We now want to find the next weak learner h_t and its weight α_t to add to the model: $F_t(\mathbf{x}) = F_{t-1}(\mathbf{x}) + \alpha_t h_t(\mathbf{x})$.

1. Show that the loss at stage t can be written as:

$$J(F_t) = \sum_{i=1}^{N} w_i^{(t)} \exp(-y_i \alpha_t h_t(\mathbf{x}_i))$$

   where $w_i^{(t)} = \exp(-y_i F_{t-1}(\mathbf{x}_i))$ is a weight that is constant with respect to α_t and h_t.

2. Split the sum in part (a) into two sums: one over the correctly classified instances ($y_i = h_t(\mathbf{x}_i)$) and one over the misclassified instances ($y_i \neq h_t(\mathbf{x}_i)$). Use this to show that:

$$J(F_t) = e^{-\alpha_t} \sum_{y_i = h_t(\mathbf{x}_i)} w_i^{(t)} + e^{\alpha_t} \sum_{y_i \neq h_t(\mathbf{x}_i)} w_i^{(t)}$$

3. Let $\epsilon_t = \frac{\sum_{y_i \neq h_t(\mathbf{x}_i)} w_i^{(t)}}{\sum_{i=1}^{N} w_i^{(t)}}$ be the weighted error of h_t. Show that minimizing $J(F_t)$ with respect to α_t (for a fixed h_t) leads to the optimal weight:

$$\alpha_t = \frac{1}{2} \ln\left(\frac{1 - \epsilon_t}{\epsilon_t}\right)$$

   (Hint: Differentiate the expression from part (b) with respect to α_t and set the derivative to zero.)

Q5 [*More Challenging / Cross-Concept*] **[Random Forests (Complexity Control) & Boosting (Gradient Tree Boosting)]**
Both Random Forests and Gradient Boosting use decision trees as base learners but control model complexity in fundamentally different ways. Random Forests are a *bagging* method that reduces variance by averaging many low-bias, high-variance trees. Gradient Boosting is a *boosting* method that reduces bias by sequentially adding small, high-bias, low-variance trees to correct errors. The complexity of the final model is controlled by different sets of hyperparameters.
Compare and contrast the role of the "number of trees" (T) hyperparameter in Random Forests versus Gradient Tree Boosting.

1. In a Random Forest, what happens to the training and test error as T becomes very large? Explain this behavior using the bias-variance decomposition.

2. In Gradient Boosting, what happens to the training error as T becomes very large? What risk is associated with using a very large T in Gradient Boosting, and how is this typically mitigated?

3. The "depth of the base trees" is another critical hyperparameter. How does the ideal tree depth differ between a standard Random Forest and a standard Gradient Boosting model? Justify your answer based on the fundamental goals of each ensemble method.

Name: YiFan Cai

PennKey: cai03

Honor Statement: By signing my name below, I affirm that I will not give or receive unauthorized assistance on this homework, and that all submitted work is my own.

Signature: YiFan Cai Date: 2025/10/14

Note: This document is a read-only file. To create an editable version click on Menu in the top left corner of your screen and choose the Copy Project option.

1 Written Questions

Q1 **[KL Divergence]** We revisit the Kullback–Leibler (KL) divergence as a loss for comparing two probability distributions p and q. In typical ML workflows, one distribution comes from the *data* and the other from a *model*. Training aims to align these distributions as closely as possible. Because KL is *asymmetric*, the order $(p\|q)$ versus $(q\|p)$ matters.

Setup & notation Let X be a *discrete* random variable. Consider two discrete distributions $P(x)$ and $Q(x)$, given as vectors of frequencies for each outcome x. We define

$$Z_p = \sum_x P(x), \qquad p(x) = \frac{P(x)}{Z_p} \quad \text{(optionally } Z_q = \textstyle\sum_x Q(x),\ q(x) = Q(x)/Z_q\text{)}.$$

The KL divergence is

$$\mathrm{KL}(p\|q) = \sum_x p(x) \log \frac{p(x)}{q(x)}.$$

We use a parametric model $q_W(x)$ with weights W; thus $q = q_W$.

1. Formulate $\mathrm{KL}(q_W\|p)$ as an optimization problem in terms of $P(x)$, q_W, and Z_p (with $Z_p = \sum_x P(x)$). *Write it explicitly as an* arg min *over* W:

   $$\arg\min_W \ \cdots$$

   Then algebraically simplify the objective so that any constants (with respect to W) are grouped/eliminated.

2. *Follow the instruction in (1),* but now derive the arg min formulation for $\mathrm{KL}(p\|q_W)$ in terms of $P(x)$, q_W, and Z_p, and simplify similarly.

3. Indicate whether $\mathrm{KL}(q_W\|p)$ or $\mathrm{KL}(p\|q_W)$ is generally easier to compute in practice, and *justify your choice. Hint:* Compare your objectives from (1)–(2) and consider which terms are straightforward to estimate from data (e.g., empirical sums under p) versus those that require expectations under the model q_W.

4. Examine your arg min expression from (2) for $\mathrm{KL}(p\|q_W)$. *Do you recognize its resemblance to a standard loss used in supervised classification (cross-entropy / negative log-likelihood)?* Explain the connection precisely and state whether your objective differs from cross-entropy by any constant term independent of W.

1. $KL(q_w|p) = \sum_x q_w(x) \log \frac{q_w(x)}{p(x)}$

$= \sum_x q_w(x) \cdot (\log q_w(x) - \log p(x) + \log Z_p)$

$= \sum_x q_w(x) \cdot \log q_w(x) - \sum_x q_w(x) \cdot \log p(x) + \log Z_p$

$\Rightarrow \arg\min_w KL(q_w|p) = \arg\min_w \sum_x q_w(x) \log q_w(x) - \sum_x q_w(x) \cdot \log p(x)$

2. $KL(p\|q_w) = \sum_x p(x) \cdot \log \frac{p(x)}{q_w(x)} = \sum_x p(x) \cdot \log p(x) - \sum_x p(x) \cdot \log q_w(x)$

$= \left[\sum_x p(x) \cdot \log p(x) - \sum_x p(x) \cdot \log Z_p - \sum_x p(x) \cdot \log q_w(x)\right] / Z_p$

$\arg\min_w KL(p\|q_w) = \arg\min_w -\sum_x p(x) \cdot \log q_w(x)$

3. $KL(p\|q_w)$ is easier since it just use $p(x)$ as the sum weight which is from the training set easily. But for the other one, the weight $q_w(x)$ is hard to get, since it is from the model and still under training.

4. cross-entropy: $H(p, q_w) = -\sum_x p(x) \log q_w(x)$

$\Rightarrow H(p, q_w) = -\frac{1}{Z_p} \sum_x P(x) \log q_w(x)$

thus, the differ is just a $1/Z_p$

This is because $KL(p\|q_w) = H(p, q_w) - H(p)$

and $H(p)$ is not related to w.

thus optimizing $KL(p\|q_w)$ is same with optimizing $H(p, q_w)$

Part 1: Ensembles

Q1 [*Concept Check*] **[AdaBoost]** AdaBoost is a seminal boosting algorithm that combines weak learners (often decision stumps) into a strong classifier. Its core innovation is adaptively reweighting the training dataset at each iteration, focusing more on instances that were previously misclassified. The weight update rule and the final aggregation method are key to understanding its mechanics.

Explain the role of the coefficient α_t in the AdaBoost algorithm. Specifically, answer the following:

1. Write the formula for α_t and describe how it is a function of the weighted error ϵ_t of the weak learner at step t.
2. Intuitively, why does this formula make sense? What does a large α_t value imply about the performance of the t-th weak learner?
3. How does α_t influence the final aggregated classifier $H(\mathbf{x}) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(\mathbf{x})\right)$?

1. $\epsilon_t = \sum_{i}^{N} w_i^t \, I(h_t(x_i) \neq y_i)$

$\alpha_t = \frac{1}{2} \ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$

we can compare ϵ_t with 0.5, to judge whether it is better than guess randomly.

2. we want. $\epsilon < 0.5$ be a good predictor, $\epsilon > 0.5$ be a bad one, α_t is a weight, we let good predictor be more important.

3. $H(x) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$

every learner have a vote, and use weight α_t, important and accurate voter will be stronger, and be considered more.

Q2 [*Application / Real-World Motivation*] **[Gradient Tree Boosting]**
While bagging methods like Random Forests are excellent for high-variance models, boosting methods like Gradient Boosting Machines (GBM), particularly with decision trees as weak learners, excel at reducing both bias and variance. They are dominant in tabular data competitions (e.g., Kaggle) due to their predictive power. A key consideration in real-world applications is managing complexity to prevent overfitting, which is controlled by parameters like the learning rate, tree depth, and number of estimators.

You are a data scientist building a fraud detection system for credit card transactions. The dataset is highly imbalanced (99% legitimate, 1% fraudulent). You decide to use a Gradient Tree Boosting model (e.g., XGBoost or LightGBM).

1. Why is an ensemble method like Gradient Boosting potentially better for this task than a single deep decision tree?
2. Describe two specific techniques or parameters you would use within the Gradient Boosting framework to address the class imbalance problem.
3. Compared to a Random Forest model, what is a potential advantage and a potential disadvantage of using Gradient Boosting for this task, considering model performance and operational constraints?

1. a single deep decision tree is a high variance model that can easily overfit. especially in noisy or imbalanced datasets.
Gradient boosting: builds many shallow trees sequentially, each tree corrects the errors of previous one. It also reduce both bias and var.

2. Since fraud samples are rare. we can assign higher weight to fraud samples, thus let model to pay more attentions to them
we should design a appropriate evaluate metric or objective func so that we can punish heavily if model can't identify a fraud sample.

3. ✓ with low bias and var, it can have higher predictive accuracy on detecting rare porterns like fraud. It focuses on hard-to-classify examples through sequential boosting
✗ require more hyperparameter tuning. Training is sequential so slower and less parallelizable.

Q3 [*Short Proof/Reasoning*] **[Why Ensembles are Good (Variance Reduction in Bagging]**
Bagging (Bootstrap Aggregating) reduces the variance of a base estimator. The core idea is to train multiple models on bootstrapped samples of the training data and average their predictions. For an unstable estimator (high variance, low bias) like a deep decision tree, this averaging process stabilizes the predictions.
Assume we have T independent and identically distributed (i.i.d.) random variables, $Z_1, Z_2, ..., Z_T$, each with variance σ^2. The variance of their average, $\bar{Z} = \frac{1}{T}\sum_{t=1}^{T} Z_t$, is σ^2/T.
In bagging, however, the base learners h_t trained on different bootstrap samples are *not* independent; they are positively correlated because they are all trained on data drawn from the same underlying distribution. Let ρ be the correlation between any two learners, and let each learner have variance σ^2.
Show that the variance of the bagged ensemble $\bar{h} = \frac{1}{T}\sum_{t=1}^{T} h_t$ is:

$$\rho\sigma^2 + \frac{1-\rho}{T}\sigma^2$$

Explain why this formula demonstrates the variance-reduction effect of bagging and its limit.

$$\mathrm{Var}\left(\frac{1}{T}\sum_{t=1}^{T} h_t\right) = \frac{1}{T^2}\left(\sum_{t=1}^{T}\mathrm{Var}(h_t) + \sum_{i\neq j}\mathrm{Cov}(h_i, h_j)\right)$$

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

$$= \frac{\sigma^2}{T} + \frac{T-1}{T}\rho\sigma^2 = \rho\sigma^2 + \frac{1-\rho}{T}\cdot\sigma^2$$

so, if ρ decrease, the var will decrease sharply, and the more the random variables T the better the consequence.

Q4 [*Proof / Multi-Step Problem*] **[Boosting (AdaBoost) - Minimizing Exponential Loss]**
The AdaBoost algorithm can be derived from the perspective of forward stagewise additive modeling, where the goal is to minimize an exponential loss function: $L(y, F(\mathbf{x})) = \exp(-yF(\mathbf{x}))$, where $y \in \{-1, +1\}$ and $F(\mathbf{x})$ is the additive model.
Let $F(\mathbf{x}) = \sum_{t=1}^{T} \alpha_t h_t(\mathbf{x})$ be an ensemble of weak classifiers $h_t(\mathbf{x}) \in \{-1, +1\}$. We want to minimize the exponential loss over the training data: $J(F) = \sum_{i=1}^{N} \exp(-y_i F(\mathbf{x}_i))$.
Assume we have already built the ensemble up to stage $t-1$, obtaining $F_{t-1}(\mathbf{x})$. We now want to find the next weak learner h_t and its weight α_t to add to the model: $F_t(\mathbf{x}) = F_{t-1}(\mathbf{x}) + \alpha_t h_t(\mathbf{x})$.

1. Show that the loss at stage t can be written as:

$$J(F_t) = \sum_{i=1}^{N} w_i^{(t)} \exp(-y_i \alpha_t h_t(\mathbf{x}_i))$$

where $w_i^{(t)} = \exp(-y_i F_{t-1}(\mathbf{x}_i))$ is a weight that is constant with respect to α_t and h_t.

2. Split the sum in part (a) into two sums: one over the correctly classified instances ($y_i = h_t(\mathbf{x}_i)$) and one over the misclassified instances ($y_i \neq h_t(\mathbf{x}_i)$). Use this to show that:

$$J(F_t) = e^{-\alpha_t} \sum_{y_i = h_t(\mathbf{x}_i)} w_i^{(t)} + e^{\alpha_t} \sum_{y_i \neq h_t(\mathbf{x}_i)} w_i^{(t)}$$

3. Let $\epsilon_t = \frac{\sum_{y_i \neq h_t(\mathbf{x}_i)} w_i^{(t)}}{\sum_{i=1}^{N} w_i^{(t)}}$ be the weighted error of h_t. Show that minimizing $J(F_t)$ with respect to α_t (for a fixed h_t) leads to the optimal weight:

$$\alpha_t = \frac{1}{2} \ln\left(\frac{1 - \epsilon_t}{\epsilon_t}\right)$$

(Hint: Differentiate the expression from part (b) with respect to α_t and set the derivative to zero.)

1. $J(F) = \sum_{i=1}^{N} \exp(-y_i F(x_i))$

$= \sum_{i=1}^{N} \exp(-y_i [F_{t-1}(x_i) + \alpha_t h_t(x_i)])$

$= \sum_{i=1}^{N} \exp(-y_i F_{t-1}(x_i)) \cdot \exp(-y_i \alpha_t h_t(x_i))$

$= \sum_{i=1}^{N} w_i^{(t)} \cdot \exp(-y_i \alpha_t h_t(x_i))$

2. if right : $y_i = h_t(x_i) \Rightarrow y_i h_t(x_i) = 1$

if wrong : $y_i = -h_t(x_i) \Rightarrow y_i h_t(x_i) = -1$

$\Rightarrow J(F_t) = e^{-\alpha_t} \sum_{y_i = h_t(x_i)} w_i^{(t)} + e^{\alpha_t} \sum_{y_i \neq h_t(x_i)} w_i^{(t)}$

3. $J(F_t) = \left(e^{-\alpha_t}(1 - \epsilon_t) + e^{\alpha_t} \cdot \epsilon_t\right) \sum_{i=1}^{N} w_i$

$\Rightarrow J'_{\alpha_t} = \sum_{i=1}^{N} w_i \left((1 - \epsilon_t) \cdot e^{-\alpha_t} \cdot (-1) + e^{\alpha_t} \epsilon_t \cdot 1\right) = 0.$

$\Rightarrow e^{-\alpha_t} = \epsilon_t (e^{-\alpha_t} + e^{\alpha_t}) \Rightarrow e^{-\alpha_t}(1 - \epsilon_t) = \epsilon_t e^{\alpha_t}$

$\Rightarrow e^{2\alpha_t} = \frac{1 - \epsilon_t}{\epsilon_t}$

$\Rightarrow \alpha_t = \frac{1}{2} \ln \frac{1 - \epsilon_t}{\epsilon_t}$

Q5 [*More Challenging / Cross-Concept*] [**Random Forests (Complexity Control) & Boosting (Gradient Tree Boosting)**]

Both Random Forests and Gradient Boosting use decision trees as base learners but control model complexity in fundamentally different ways. Random Forests are a *bagging* method that reduces variance by averaging many low-bias, high-variance trees. Gradient Boosting is a *boosting* method that reduces bias by sequentially adding small, high-bias, low-variance trees to correct errors. The complexity of the final model is controlled by different sets of hyperparameters.

Compare and contrast the role of the "number of trees" (T) hyperparameter in Random Forests versus Gradient Tree Boosting.

1. In a Random Forest, what happens to the training and test error as T becomes very large? Explain this behavior using the bias-variance decomposition.
2. In Gradient Boosting, what happens to the training error as T becomes very large? What risk is associated with using a very large T in Gradient Boosting, and how is this typically mitigated?
3. The "depth of the base trees" is another critical hyperparameter. How does the ideal tree depth differ between a standard Random Forest and a standard Gradient Boosting model? Justify your answer based on the fundamental goals of each ensemble method.

1. when T becomes big, training error will decrease to nearly 0. while the test error will decrease at first then become stable.

As we get in Q3. Var : $\rho\sigma^2 + \frac{1-\rho}{T}\sigma^2$, $T \to \infty$ $J \to \rho\sigma^2$

2. the training error will decrease to 0.
but will overfit, which makes the test error increase
we can shrink the learning rate η. we can use a validation set to stop when T is too big!

3. Random forest use deep tree, each tree has high var and low bias, then we use many trees and average to low the var.

Gradient boosting: use shallow tree, we have low var and high bias. so we study trees sequently to gradually shrink the error.