MATH 342W / 642 / RM 742 Spring 2024 HW #5

Mohammed Z Hasan

Thursday 16th May, 2024

Problem 1

These are some questions related to probability estimation modeling and asymmetric cost modeling.

(a) [easy] Why is logistic regression an example of a "generalized linear model" (glm)?

Logistic regression is considered a type of generalized linear model (GLM) because it extends the linear regression framework to model binary response variables by using a link function. In logistic regression, the response variable Y is binary, and the model predicts the probability that $Y = 1$ given a set of predictors X.

(b) [easy] What is $\mathcal{H}_{pr}$ for the probability estimation algorithm that employs the linear model in the covariates with logistic link function?

$$H_{\text{pr}} = \left\{ p(x) = \frac{1}{1 + e^{-\beta^T x}} : \beta \in \mathbb{R}^d \right\}$$

where β represents the vector of coefficients, x the vector of covariates, and d the number of covariates.

(c) [easy] If logistic regression predicts 3.1415 for a new $\boldsymbol{x}_*$, what is the probability estimate that $y = 1$ for this $\boldsymbol{x}_*$?

To find the probability estimate $p(y = 1 \mid x^*)$, we apply the logistic function:

$$p(y = 1 \mid x^*) = \frac{1}{1 + e^{-\beta^T x^*}} = \frac{1}{1 + e^{-3.1415}}$$

Let's calculate the specific probability:

$$p(y = 1 \mid x^*) = \frac{1}{1 + e^{-3.1415}}$$

(d) [harder] What is $\mathcal{H}_{pr}$ for the probability estimation algorithm that employs the linear model in the covariates with cloglog link function?

The *cloglog* is used typically for modeling count data where events happen independently with a constant known rate. It's another example of a generalized linear model (GLM). The cloglog function is defined as:

$$\log(-\log(1-p)) = \beta^T x$$

where p is the probability of $Y = 1$ given x, and β is the vector of coefficients.

The hypothesis space H_{pr} for this model includes all functions of the form:

$$H_{\text{pr}} = \{p(x) = 1 - \exp\left(() - \exp\left(()\, \beta^T x\right)\right) : \beta \in \mathbb{R}^d\}$$

(e) [difficult] Generalize linear probability estimation to the case where $\mathcal{Y} = \{C_1, C_2, C_3\}$. Use the logistic link function like in logistic regression. Write down the objective function that you would numerically maximize. This objective function is one that is argmax'd over the parameters (you define what these parameters are — that is part of the question).

Once you get the answer you can see how this easily goes to $K > 3$ response categories. The algorithm for general K is known as "multinomial logistic regression", "polytomous LR", "multiclass LR", "softmax regression", "multinomial logit" (mlogit), the "maximum entropy" (MaxEnt) classifier, and the "conditional maximum entropy model". You can inflate your resume with lots of jazz by doing this one question!

Let $\mathbf{X}$ be the feature vector for an observation. The probability of an observation belonging to class C_k is modeled as:

$$P(Y = C_k \mid \mathbf{X}) = \frac{\exp\left(()\, \boldsymbol{\beta}_k^\top \mathbf{X}\right)}{\sum_{j=1}^{3} \exp\left(()\, \boldsymbol{\beta}_j^\top \mathbf{X}\right)}$$

Objective Function:

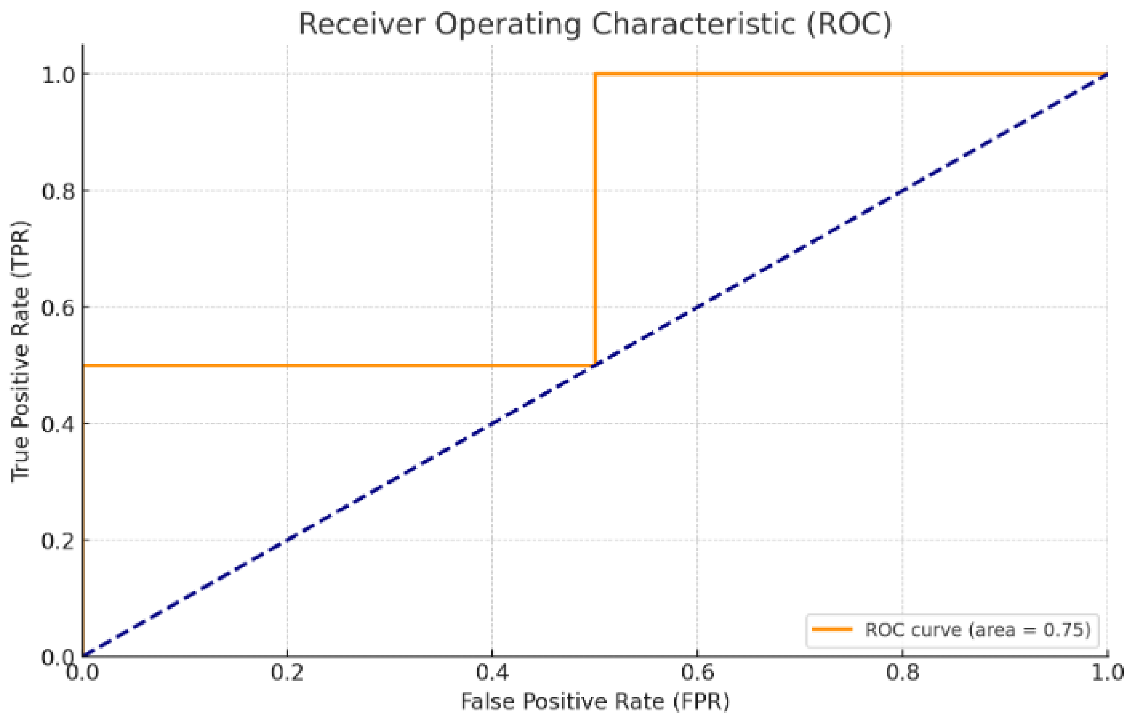
The objective is to maximize the log-likelihood of the observed data. Let $\{(\mathbf{X}_i, Y_i)\}_{i=1}^n$ be the dataset, where $Y_i \in \{C_1, C_2, C_3\}$. The log-likelihood function L is:

$$L(\boldsymbol{\beta}) = \sum_{i=1}^{n} \sum_{k=1}^{3} I(Y_i = C_k) \log P(Y_i = C_k \mid \mathbf{X}_i)$$

For K classes, the objective function remains similar:

$$L(\boldsymbol{\beta}) = \sum_{i=1}^{n} \sum_{k=1}^{K} I(Y_i = C_k) \left[\boldsymbol{\beta}_k^\top \mathbf{X}_i - \log\left(\sum_{j=1}^{K} \exp\left(()\, \boldsymbol{\beta}_j^\top \mathbf{X}_i\right)\right)\right]$$

(f) [easy] Graph a canonical ROC and label the axes. In your drawing estimate AUC. Explain very clearly what is measured by the x axis and the y axis.



X-axis (False Positive Rate - FPR):

This measures the proportion of negative samples that are incorrectly classified as positive. It is calculated as:

$$\text{FPR} = \frac{\text{False Positives}}{\text{False Positives} + \text{True Negatives}}$$

A lower FPR indicates fewer negative samples are misclassified as positive.

Y-axis (True Positive Rate - TPR):

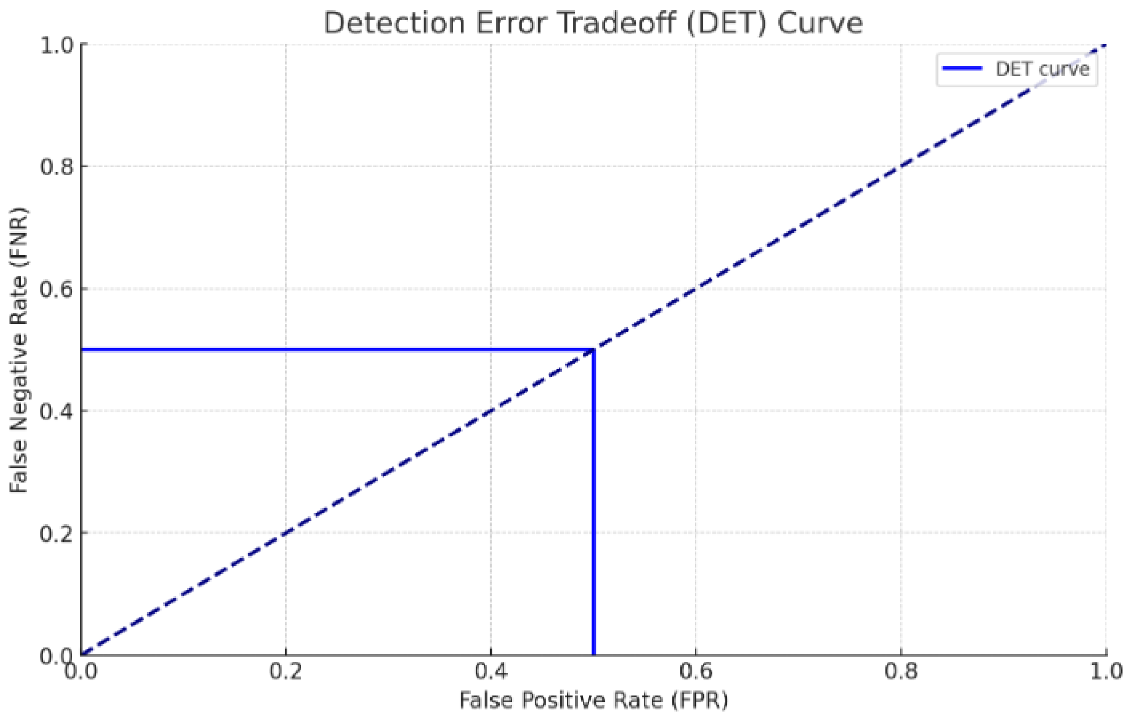
This measures the proportion of positive samples that are correctly classified as positive. It is also known as sensitivity or recall. It is calculated as:

$$\text{TPR} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

(g) [easy] Pick one point on your ROC curve from the previous question. Explain a situation why you would employ this model.

Im choosing: TPR = 0.5 and FPR = 0.2. Employing this model in a medical screening context is justified because it achieves a balance between correctly identifying a substantial number of true positives and keeping the false positive rate at a manageable level.

(h) [harder] Graph a canonical DET curve and label the axes. Explain very clearly what is measured by the x axis and the y axis. Make sure the DET curve's intersections with the axes is correct.



X-axis (False Positive Rate - FPR):

This measures the proportion of negative samples that are incorrectly classified as positive. It is calculated as:

$$\text{FPR} = \frac{\text{False Positives}}{\text{False Positives} + \text{True Negatives}}$$

A lower FPR indicates fewer negative samples are misclassified as positive.

Y-axis (False Negative Rate - FNR):

This measures the proportion of positive samples that are incorrectly classified as negative. It is calculated as:

$$\text{FNR} = \frac{\text{False Negatives}}{\text{False Negatives} + \text{True Positives}}$$

(i) [easy] Pick one point on your DET curve from the previous question. Explain a situation why you would employ this model.

I chose FPR $= 0.3$ and FNR $= 0.2$. Using this model for security screening at an airport makes sense because it strikes a good balance: it catches most potential threats (low FNR) while keeping the number of unnecessary additional screenings (moderate FPR) manageable.

(j) [difficult] [MA] The line of random guessing on the ROC curve is the diagonal line with slope one extending from the origin. What is the corresponding line of random guessing in the DET curve? This is not easy...

Problem 2

These are some questions related to bias-variance decomposition. Assume the two assumptions from the notes about the random variable model that produces the δ values, the error due to ignorance.

(a) [easy] Write down (do not derive) the decomposition of MSE for a given $\boldsymbol{x}_*$ where $\mathbb{D}$ is assumed fixed but the response associated with $\boldsymbol{x}_*$ is assumed random.

When the dataset D is fixed and only the response associated with x^* is assumed random, the Mean Squared Error (MSE) for a prediction $\hat{y}$ at x^* can be decomposed as follows:

$$\text{MSE}(\hat{y}) = \text{Bias}^2(\hat{y}) + \text{Variance}(\hat{y}) + \text{Irreducible Error}$$

However, under the assumption that D is fixed, the decomposition simplifies to:

$$\text{MSE}(\hat{y}) = \text{Bias}^2(\hat{y}) + \text{Irreducible Error}$$

(b) [easy] Write down (do not derive) the decomposition of MSE for a given $\boldsymbol{x}_*$ where the responses in $\mathbb{D}$ is random but the $\boldsymbol{X}$ matrix is assumed fixed and the response associated with $\boldsymbol{x}_*$ is assumed random like previously.

When the responses in D are random (but the features X are fixed), the Mean Squared Error (MSE) decomposition becomes:

$$\text{MSE}(\hat{y}) = \text{Bias}^2(\hat{y}) + \text{Variance}(\hat{y}) + \text{Irreducible Error}$$

Here, the variance term emerges due to the randomness in the responses of D, which affects the stability of the estimator $\hat{y}$.

(c) [easy] Write down (do not derive) the decomposition of MSE for general predictions of a phenomenon where all quantities are considered random.

When all components, including features in D, responses in D, and x^* are random, the Mean Squared Error (MSE) decomposition is fully expressed as:

$$\text{MSE}(\hat{y}) = \text{Bias}^2(\hat{y}) + \text{Variance}(\hat{y}) + \text{Irreducible Error}$$

This reflects the influence of variability in all components on the prediction error.

(d) [difficult] Why is it in (a) there is only a "bias" but no "variance" term? Why did the additional source of randomness in (b) spawn the variance term, a new source of error?

In (a), since D (the training data) is fixed, the model training will always produce the same set of parameters or predictions for any fixed x^*. Therefore, there is no variability in $\hat{y}$ due to D, leading to a zero variance term. The bias term arises because the fixed model may systematically deviate from the true response values.

When additional randomness is introduced in (b) (random responses in D), each training of the model can lead to different parameters or predictions due to different responses. This introduces variability in $\hat{y}$, hence a variance term appears.

(e) [harder] A high bias / low variance algorithm is underfit or overfit?

A high bias and low variance algorithm is typically underfit. This means the model is too simple and does not capture the underlying pattern in the data well, leading to systematic errors in predictions across different training sets.

(f) [harder] A low bias / high variance algorithm is underfit or overfit?

A low bias and high variance algorithm is typically overfit. This indicates that the model is too complex and fits the noise or random fluctuations in the training data rather than the true underlying pattern. This results in poor generalization to new data.

(g) [harder] Explain why bagging reduces MSE for “free” regardless of the algorithm employed.

Bagging is a powerful ensemble technique that reduces the variance part of the error without increasing the bias, effectively reducing the Mean Squared Error (MSE). Here's why bagging achieves this "for free" across different algorithms:

Variance Reduction: By averaging multiple estimates from subsets of the data (or bootstrap samples), bagging reduces the variability of the prediction. Each individual model might have high variance if trained on its subset of data, but aggregating their predictions (e.g., by averaging for regression tasks) can cancel out some of the individual variabilities, leading to a more stable and accurate prediction.

Bias Maintenance: Each model in bagging is trained on a bootstrap sample, which is as complex as the training set. Since the complexity is maintained, the bias of each individual model is roughly the same as the bias of a model trained on the entire dataset. Therefore, the aggregation of models does not increase the bias.

Model Independence: The effectiveness of bagging increases as the predictions from the individual models are more uncorrelated. By using random subsets of the training data, each model is slightly different, which helps in achieving uncorrelated predictions, especially when the base model is unstable or highly sensitive to changes in the input data (like decision trees).

(h) [harder] Explain why RF reduces MSE atop bagging M trees and specifically mention the target that it attacks in the MSE decomposition formula and why it's able to reduce that target.

Random Forest is an extension of bagging where each decision tree in the ensemble is constructed by using a random subset of features at each split. This adds an extra layer of variance reduction atop bagging. Here's how RF specifically targets components of MSE:

Additional Variance Reduction: In bagging, trees are still correlated because they tend to use the best predictors at the top splits, which leads to similar tree structures and correlated errors. Random forests mitigate this by forcing each tree to consider different subsets of features for splitting at each node. This reduces the correlation between the trees even further compared to standard bagging, thus reducing the variance component of MSE even more.

Target in MSE Decomposition: The primary target of RF in MSE decomposition is the variance component. By reducing the correlation between the trees, RF effectively lowers the variance without affecting the bias significantly.

(i) [difficult] When can RF lose to bagging M trees? Hint: think hyperparameter choice.

Random Forest might underperform compared to bagging under certain conditions related to hyperparameter settings:

Feature Subset Size: In RF, the size of the feature subset (max_features) chosen at each split is crucial. If this parameter is set too high (close to the total number of features), RF will behave more like bagging, losing its advantage of de-correlation. Conversely, if it is set too low, the trees might become too weak, unable to capture sufficient complexity, leading to higher bias.

Depth of Trees: If the trees in RF are allowed to grow very deep, they might overfit, especially if max_features is large. On the other hand, overly shallow trees might not capture enough complexity, leading to underfitting.

Number of Trees (M): While more trees generally improve performance by reducing variance, after a certain point, the benefits can plateau while computation costs continue to rise. Additionally, if the number of trees is not sufficient, RF may not adequately explore the feature space, resulting in higher bias or variance than bagging, which could potentially perform better with fewer, but fully diversified trees.

In summary, the performance of Random Forest relative to bagging depends critically on the choice of hyperparameters, and poor choices can lead to scenarios where bagging might outperform RF.

Problem 3

These are some questions related to missingness.

(a) [easy] [MA] What are the three missing data mechanisms? Provide an example when each occurs (i.e., a real world situation). We didn't really cover this in class so I'm making it a MA question only. This concept will NOT be on the exam.

(b) [easy] Why is listwise-deletion a *terrible* idea to employ in your $\mathbb{D}$ when doing supervised learning?

Listwise deletion involves discarding any records in the dataset D that contain missing values. This method can severely bias the analysis if the missing data are not missing at random:

- Reduced Sample Size: It reduces the effective sample size, which can lead to loss of power in statistical analyses.
- Bias in Estimates: If the missingness is related to the value of the missing data, listwise deletion can lead to biased estimates because the remaining data are not representative of the entire population.

(c) [easy] Why is it good practice to augment $\mathbb{D}$ to include missingness dummies? In other words, why would this increase oos predictive accuracy?

Creating dummy variables for missing data can be beneficial:

- Captures Information about Missingness: These dummies indicate whether the value was missing and can help the model learn if the missingness itself is predictive of the outcome. This is particularly useful if the missingness pattern holds information about the target variable.
- Improves Out-of-Sample (OOS) Predictive Accuracy: By incorporating these dummies, models can handle missing values more effectively and make better predictions when faced with missing data in new, unseen data sets.

(d) [easy] To impute missing values in $\mathbb{D}$, what is a good default strategy and why?

A good default strategy for imputation is multiple imputation:

- Robustness: It acknowledges the uncertainty about the true value to be imputed by creating several different plausible imputations. This results in multiple datasets, each analyzed in the usual way, and the results are combined.
- Flexibility and Efficiency: Multiple imputation is generally better than single imputation methods like mean imputation because it preserves the statistical properties of the data such as variance and covariance.
- Widely Applicable: It can be used in a variety of missing data scenarios and is effective across different types of data (MAR, MCAR, and potentially NMAR if modeled correctly).

Problem 4

These are some questions related to gradient boosting. The final gradient boosted model after M iterations is denoted G_M which can be written in a number of equivalent ways (see below). The g_t's denote constituent models and the G_t's denote partial sums of the constituent models up to iteration number t. The constituent models are "steps in functional steps" which have a step size η and a direction component denoted $\tilde{g}_t$. The directional component is the base learner $\mathcal{A}$ fit to the negative gradient of the objective function L which measures how close the current predictions are to the real values of the responses:

$$\begin{aligned} G_M &= G_{M-1} + g_M \\ &= g_0 + g_1 + \ldots + g_M \\ &= g_0 + \eta\tilde{g}_1 + \ldots + \eta\tilde{g}_M \\ &= g_0 + \eta\mathcal{A}\left(\langle \boldsymbol{X}, -\nabla L(\boldsymbol{y}, \hat{\boldsymbol{y}}_1)\rangle, \mathcal{H}\right) + \ldots + \eta\mathcal{A}\left(\langle \boldsymbol{X}, -\nabla L(\boldsymbol{y}, \hat{\boldsymbol{y}}_M)\rangle, \mathcal{H}\right) \\ &= g_0 + \eta\mathcal{A}\left(\langle \boldsymbol{X}, -\nabla L(\boldsymbol{y}, g_1(\boldsymbol{X}))\rangle, \mathcal{H}\right) + \ldots + \eta\mathcal{A}\left(\langle \boldsymbol{X}, -\nabla L(\boldsymbol{y}, g_M(\boldsymbol{X}))\rangle, \mathcal{H}\right) \end{aligned}$$

(a) [easy] From a perspective of only multivariable calculus, explain gradient descent and why it's a good idea to find the minimum inputs for an objective function L (in English).

Gradient descent is an optimization algorithm used to minimize the value of a function. It works by iteratively moving towards the minimum value of the function. The key idea is to adjust the input variables in the direction opposite to the gradient of the function at the current point.

It's a Good Idea because:

- Efficiency: Gradient descent is computationally efficient, especially for functions with many variables. It doesn't require computing the function's minimum directly, which would not work for complex functions.
- Convergence: By taking small steps in the direction of the negative gradient, it ensures that we are moving towards the function's minimum, as the gradient indicates the direction of the steepest ascent, and the negative gradient points towards the steepest descent. We call this steepest part the global minima.

(b) [easy] Write the mathematical steps of gradient boosting for supervised learning below. Use L for the objective function to keep the procedure general. Use notation found in the problem header.

Start with a simple model, often the mean of the target values for regression or the log-odds for binary classification.

$$g_0(x) = \text{initial model}$$

For Each Iteration $t = 1$ to M:

(a) Compute Residuals: Calculate the negative gradient (residuals) of the loss function with respect to the current model's predictions.

$$r_t = -\nabla L(y, \hat{y}_t)$$

where $\hat{y}_t = G_{t-1}(x)$.

(b) Fit a Base Learner: Train a base learner A to the residuals r_t using the input features X.

$$\tilde{g}_t = A(\langle X, r_t \rangle, H)$$

(c) Update the Model: Add the scaled base learner to the current model.

$$G_t = G_{t-1} + \eta \tilde{g}_t$$

Final Model:

After M iterations, the final model is:

$$G_M = G_0 + \sum_{t=1}^{M} \eta \tilde{g}_t$$

(c) [easy] For regression, what is $g_0(\boldsymbol{x})$?

For regression, the initial model $g_0(x)$ is typically the mean of the target values:

$$g_0(x) = \bar{y} = \frac{1}{n} \sum_{i=1}^{n} y_i$$

(d) [easy] For probability estimation for binary response, what is $g_0(\boldsymbol{x})$?

For binary classification, the initial model $g_0(x)$ is typically the log-odds of the positive class probability:

$$g_0(x) = \log \left(\frac{p}{1-p} \right)$$

where p is the proportion of positive examples in the training data.

(e) [harder] What are all the hyperparameters of gradient boosting? There are more than just two.

- Learning Rate (η): Controls the contribution of each base learner.
- Max Depth: The maximum depth of each base learner (tree).
- Loss Function: The function to be minimized (e.g., mean squared error for regression, log-loss for classification).

(f) [easy] For regression, rederive the negative gradient of the objective function L.

For regression, the objective function is the mean squared error (MSE):

$$L(y, \hat{y}) = \frac{1}{2n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

To find the negative gradient, we compute the derivative of L with respect to $\hat{y}_i$:

$$\frac{\partial L}{\partial \hat{y}_i} = \frac{\partial}{\partial \hat{y}_i} \left(\frac{1}{2n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \right) = \frac{1}{2n} \cdot 2 \sum_{i=1}^{n} (y_i - \hat{y}_i)(-1) = -\frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)$$

So, the negative gradient is:

$$-\frac{\partial L}{\partial \hat{y}_i} = y_i - \hat{y}_i$$

(g) [easy] For probability estimation for binary response, rederive the negative gradient of the objective function L.

For binary classification, the objective function is the log-loss:

$$L(y, \hat{y}) = -\frac{1}{n} \sum_{i=1}^{n} \left[y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i) \right]$$

To find the negative gradient, we compute the derivative of L with respect to $\hat{y}_i$:

$$\frac{\partial L}{\partial \hat{y}_i} = -\frac{1}{n} \left[\frac{y_i}{\hat{y}_i} - \frac{1 - y_i}{1 - \hat{y}_i} \right]$$

So, the negative gradient is:

$$-\frac{\partial L}{\partial \hat{y}_i} = \frac{1}{n} \left[\frac{y_i}{\hat{y}_i} - \frac{1 - y_i}{1 - \hat{y}_i} \right]$$

(h) [difficult] For probability estimation for binary response scenarios, what is the unit of the output $G_M(\boldsymbol{x}_\star)$?

In probability estimation for binary response scenarios, the output $G_M(x^*)$ represents the log-odds of the positive class probability. The unit of $G_M(x^*)$ is therefore in terms of log-odds.

(i) [easy] For the base learner algorithm $\mathcal{A}$, why is it a good idea to use shallow CART (which is the recommended default)?

- Speed: Training shallow trees is computationally efficient, which speeds up the training process.

- Avoid Overfitting: Shallow trees have limited depth, which helps prevent overfitting to the training data.

(j) [difficult] For the base learner algorithm $\mathcal{A}$, why is it a bad idea to use deep CART?

Using deep CART as the base learner in gradient boosting can be problematic for several reasons:

- Overfitting: Deep trees can model complex relationships and interactions in the data, but they are also prone to overfitting. They may capture noise and outliers in the training data, leading to poor generalization to unseen data.
- High Variance: Deep trees can have high variance, meaning small changes in the training data can lead to significantly different tree structures. This instability can reduce the robustness of the ensemble model.
- Computational Cost: Training deep trees is computationally intensive, both in terms of time and memory. This can slow down the training process, especially for large datasets.

(k) [difficult] For the base learner algorithm $\mathcal{A}$, why is it a bad idea to use OLS for regression (or logistic regression for probability estimation for binary response)?

Bias-Variance Tradeoff: OLS and logistic regression are not weak learners; they aim to provide the best possible fit to the data in a single step. Boosting algorithms are designed to work with weak learners that have high bias and low variance. Strong learners like OLS and logistic regression can disrupt the bias-variance trade-off that gradient boosting tries to optimize.

(l) [difficult] If M is very, very large, what is the risk in using gradient boosting even using shallow CART as the base learner (the recommended default)?

The main risk of using gradient boosting with a very large M is overfitting. The model becomes overly complex due to the excessive number of iterations, leading it to capture noise in the training data. Consequently, this results in poor generalization performance on new data, undermining the model's effectiveness in real-world applications.

(m) [difficult] If η is very, very large but M reasonably correctly chosen, what is the risk in using gradient boosting even using shallow CART as the base learner (the recommended default)?

The risk in using gradient boosting is that there will be overfitting. A very large η means each new base learner (shallow tree) makes large updates to the model. This can cause the model to overfit the training data quickly.