AdaBoost

CS109OA Introduction to Data Science

Pavlos Protopapas, Kevin Rader, and Chris Gumb

Photo: Jenni Arakaki
Interlaken, Switzerland

Lecture Outline

- AdaBoost
- Mathematical Formulation - AdaBoost
- Final Thoughts on Boosting

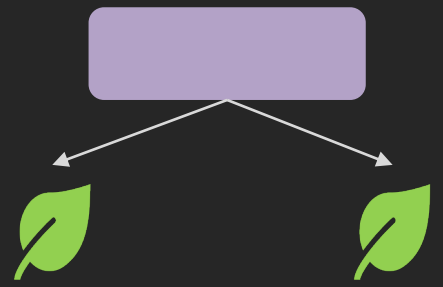
Lecture Outline

- **AdaBoost**
- Mathematical Formulation - AdaBoost
- Final Thoughts on Boosting

AdaBoost

There are two main ideas in AdaBoost:

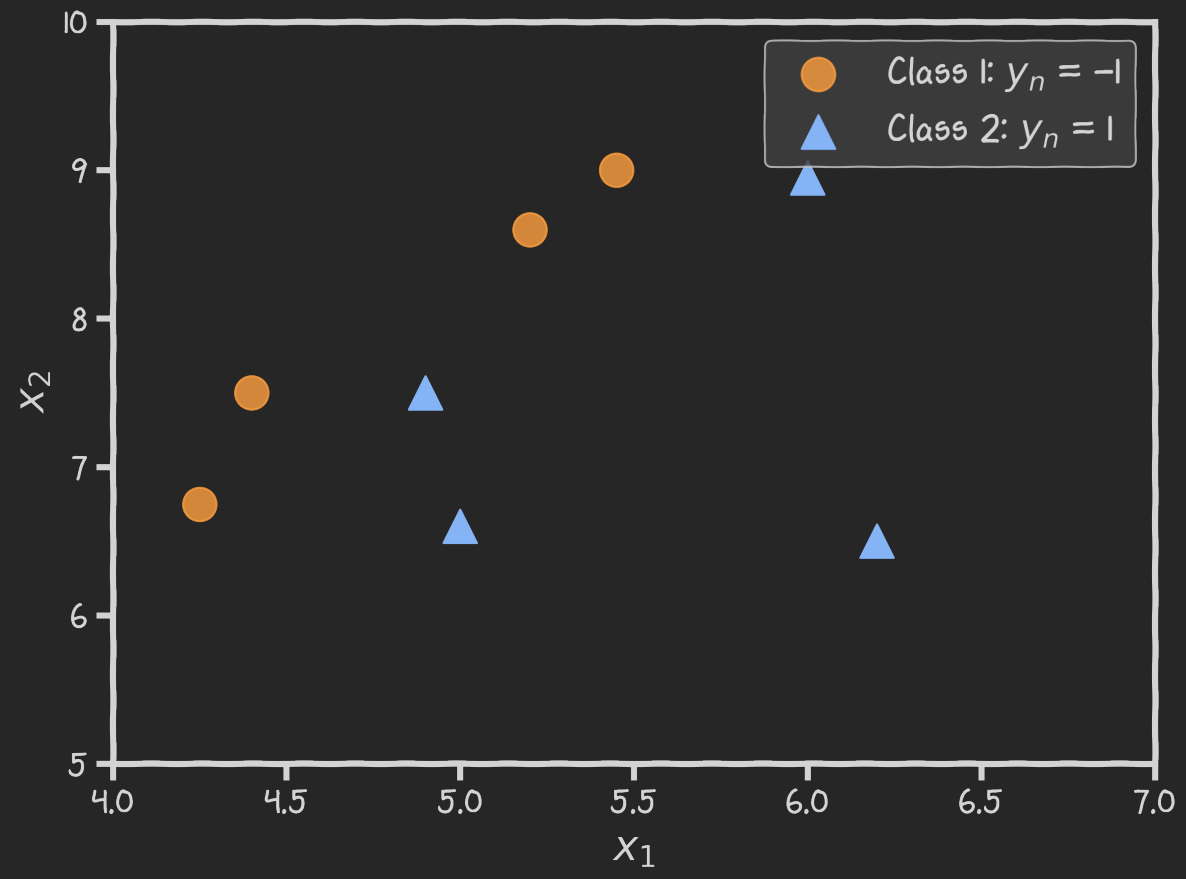
- Idea #1: **Iteratively** build a complex model T by combining several **weak learners**.
  For trees, a weak learner is a tree with 1 node with 2 leaves. We call this a **stump.**



- Idea #2: Each new stump added to the ensemble model T learns from the **mistakes** of the **current model T**. This is done by assigning higher weights to the observations that the current model incorrectly classifies.

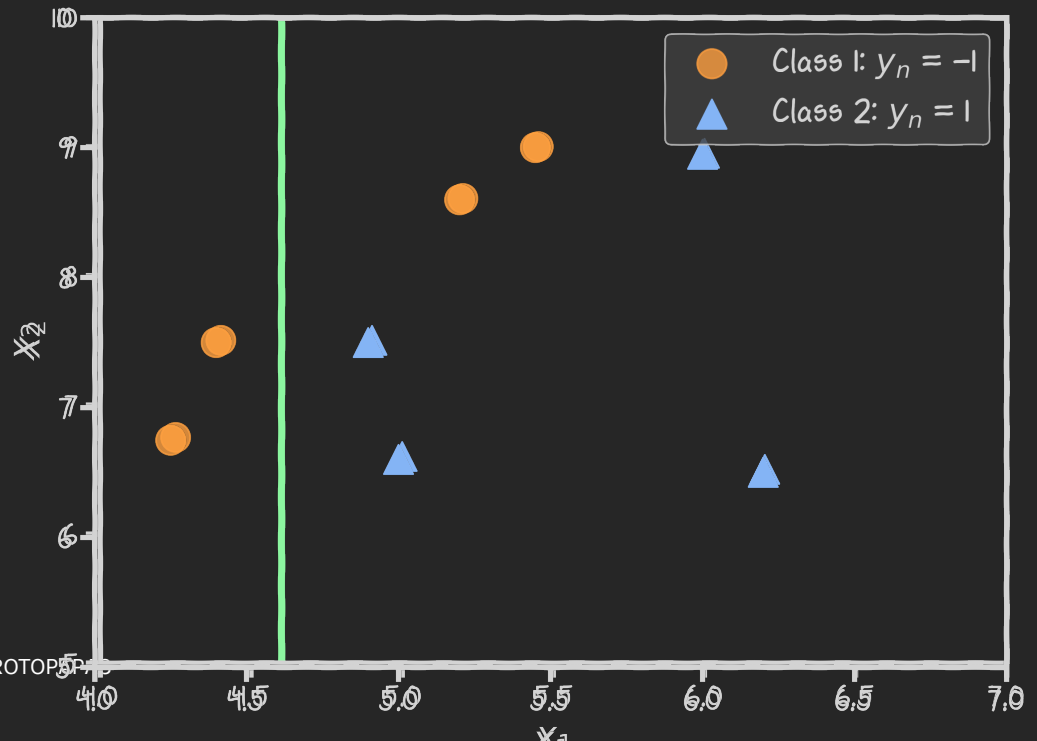
AdaBoost

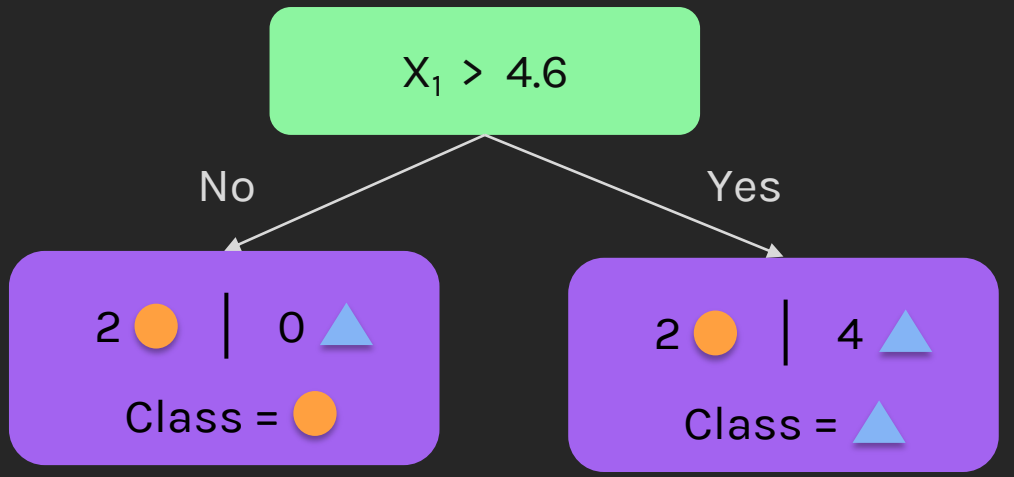
Consider the following dataset:



AdaBoost

Step 1: Fit a stump $S^{(0)}$ on the dataset.

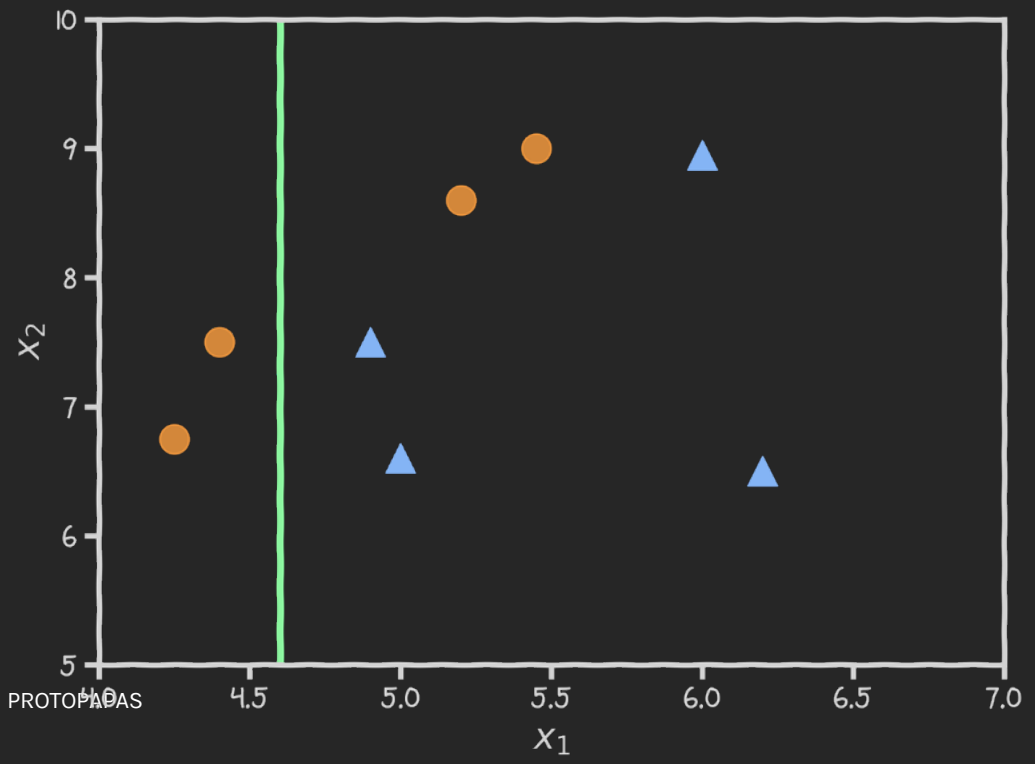




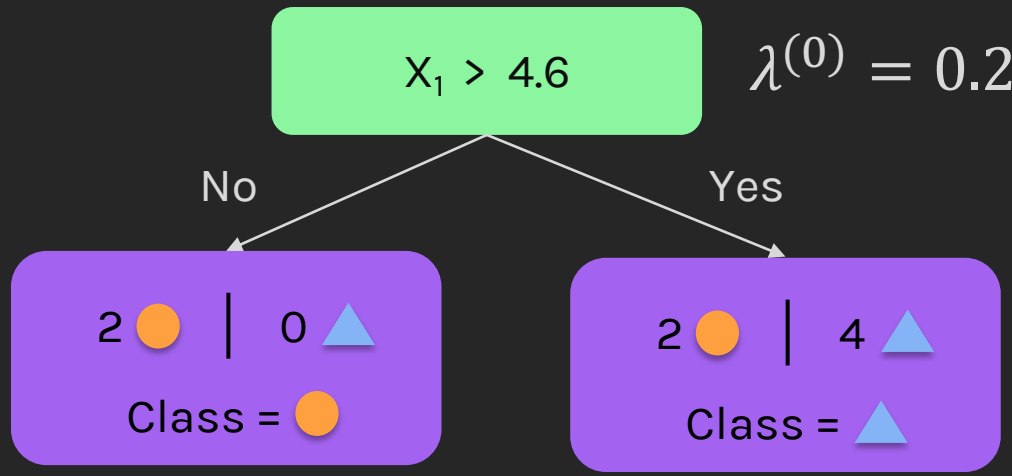
AdaBoost

Step 2: Now that we have the first weak learner, we will assign the stump a scaling factor, $\lambda^{(0)}$; The scaling factor indicates how much the stump **contributes to the entire ensemble**.

We assign λ to each successive stump because it offers some flexibility, and we can give more importance to stumps that perform better.



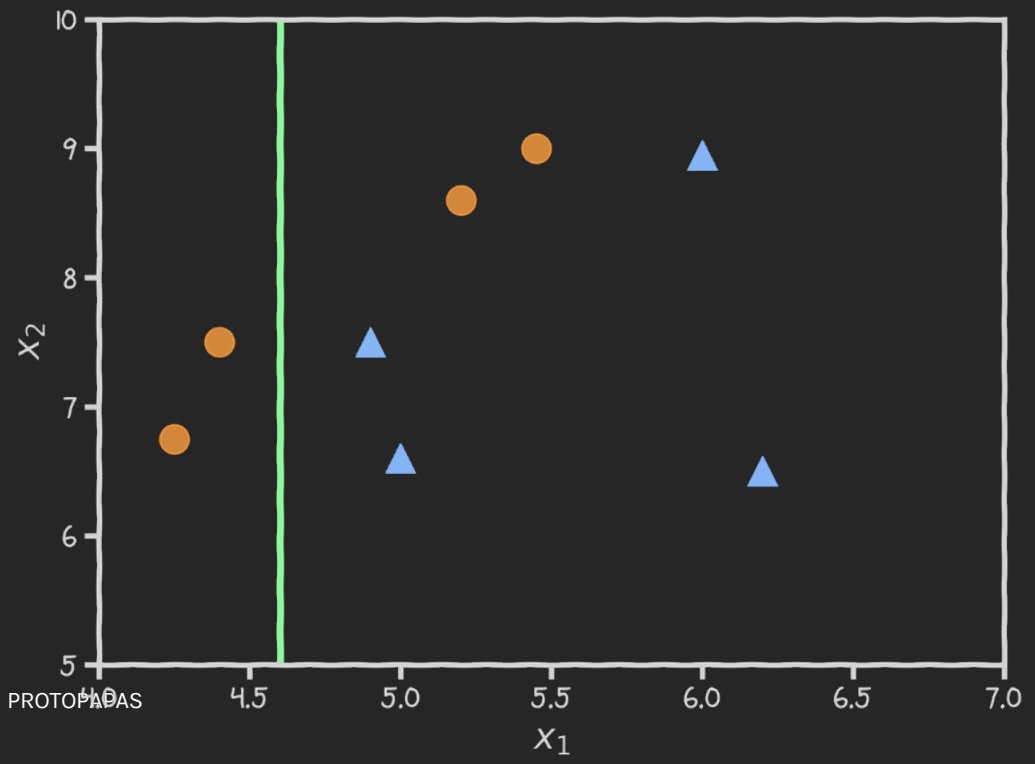
Let this model's scaling factor λ be 0.25.

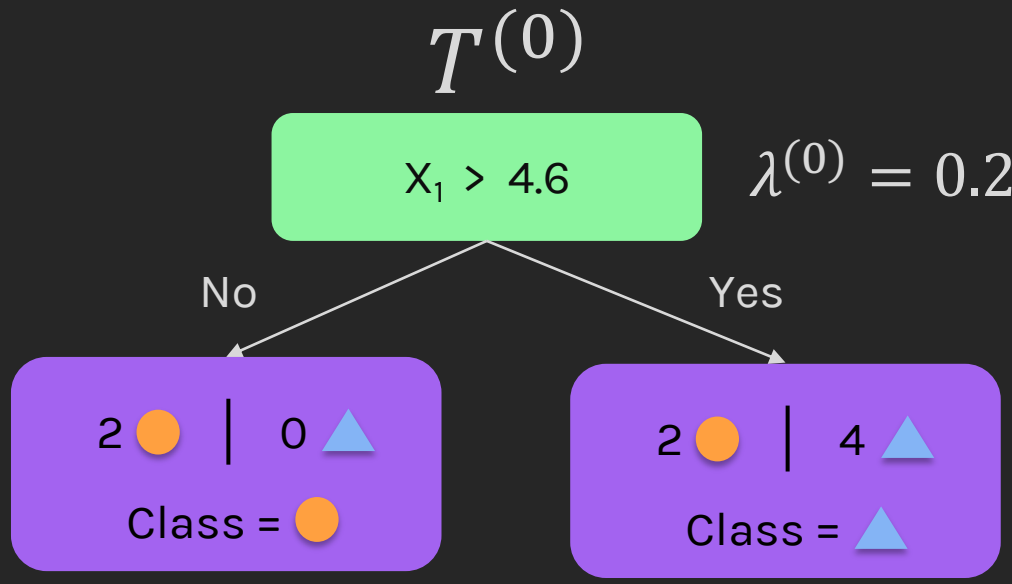


AdaBoost

Step 3: Construct the **ensemble model** $T^{(0)}$ using:

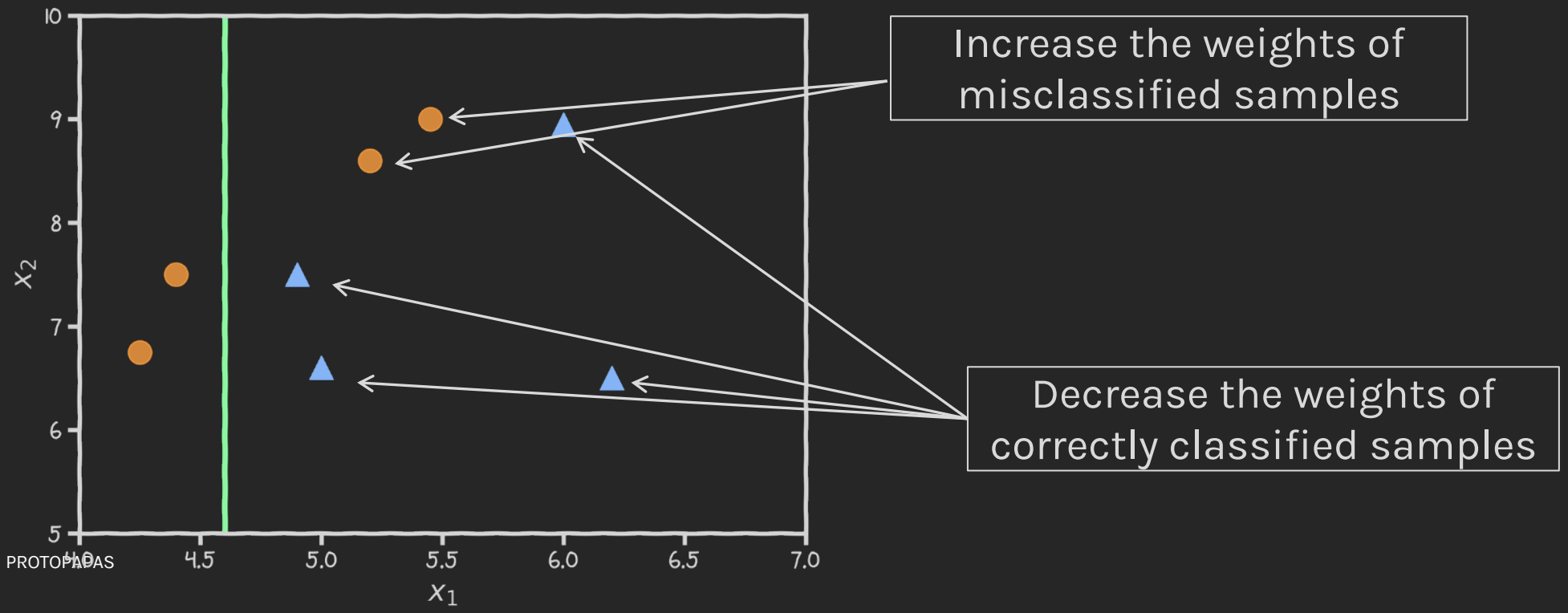
$$T^{(i)} \leftarrow \begin{cases} \lambda^{(i)} S^{(i)} & i = 0 \\ T^{(i-1)} + \lambda^{(i)} S^{(i)} & i = 1, 2, \ldots \end{cases}$$





AdaBoost

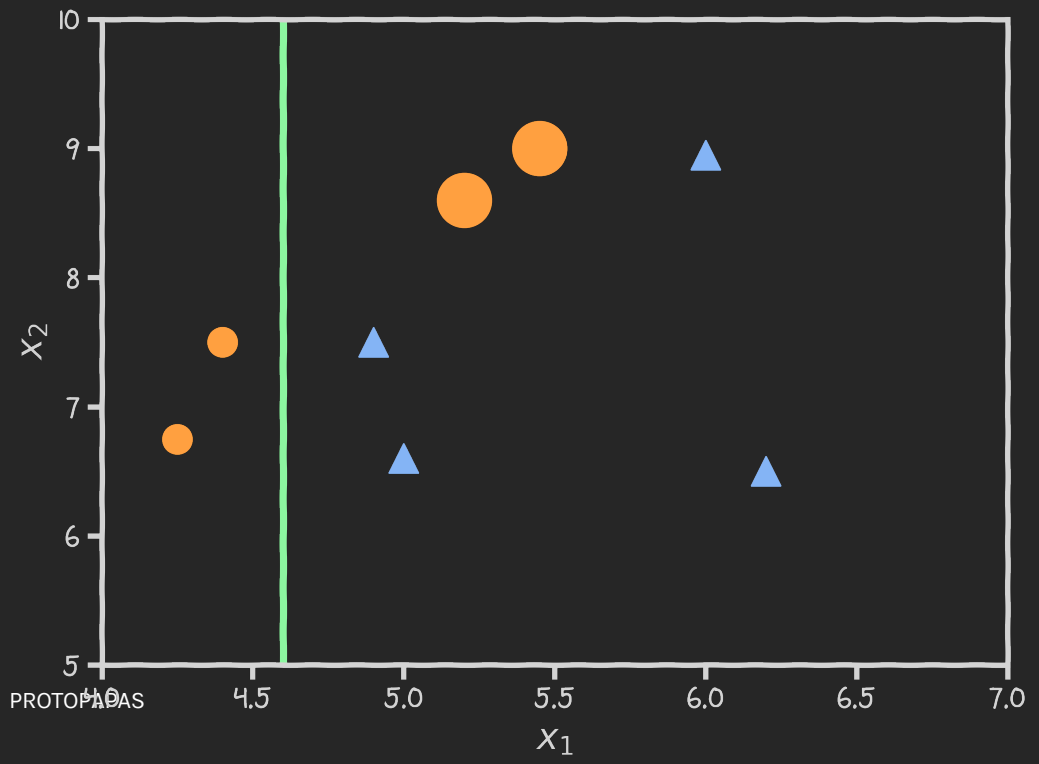
Step 4: Adjust the weights assigned to each data point to ensure the next stump focuses on the points **misclassified by the current model**.



AdaBoost

Step 4: Adjust the weights assigned to each data point to ensure the next stump focuses on the points **misclassified by the current model**.

$$w_n^{(1)} \leftarrow \frac{w_n^{(0)} e^{-\lambda\, y_n\, S^{(0)}(x_n)}}{Z} = \frac{w_n^{(1')}}{Z}$$



AdaBoost

Yes, the labels are -1,+1, not 0,1.
This choice makes the math cleaner
we'll see why soon.

ground truth -1 and 1

prediction from current stump

$$w_n^{(1)} \leftarrow \frac{w_n^{(0)} e^{-\lambda y_n s^{(0)}(x_n)}}{Z} = \frac{w_n^{(1')}}{Z}$$

new weights

previous weights

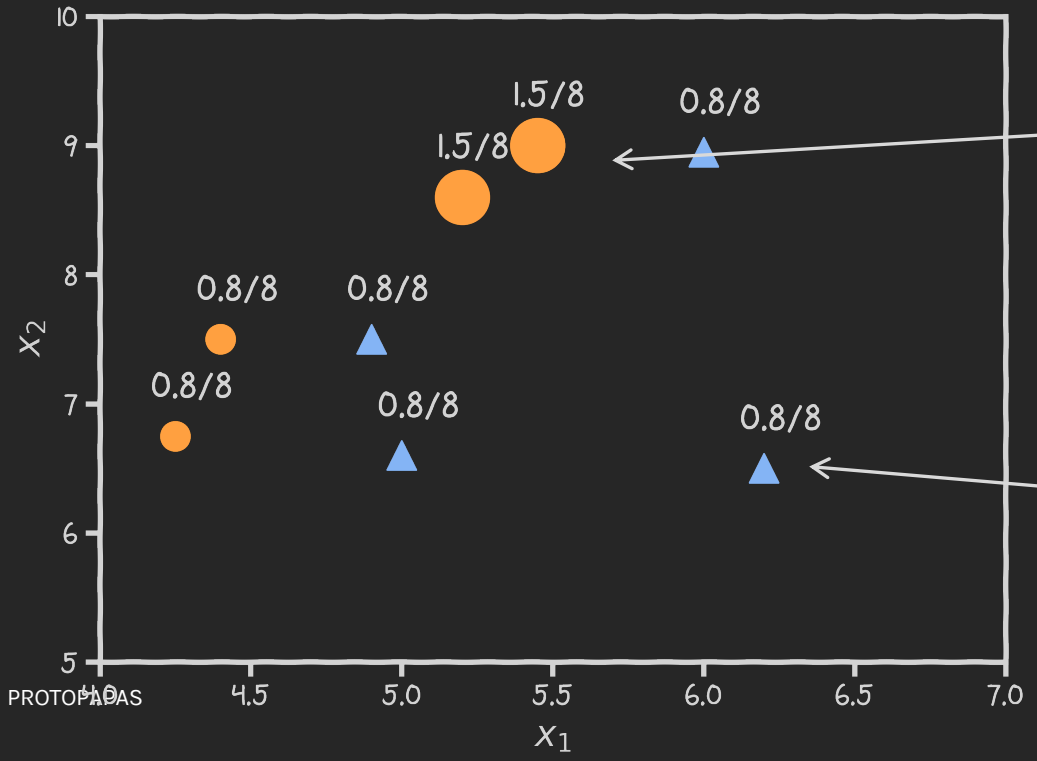
normalizes the new weights

$$Z = \sum_n^N w_n^{(i)} e^{-\lambda^{(i)} y_n S^{(i)}(x_n)}$$

AdaBoost

Step 4: Adjust the weights assigned to each data point to ensure the next stump focuses on the points **misclassified by the ensemble**.

$$w_n^{(1)} \leftarrow \frac{w_n^{(0)} e^{-\lambda\, y_n\, S^{(0)}(x_n)}}{Z} = \frac{w_n^{(1')}}{Z}$$



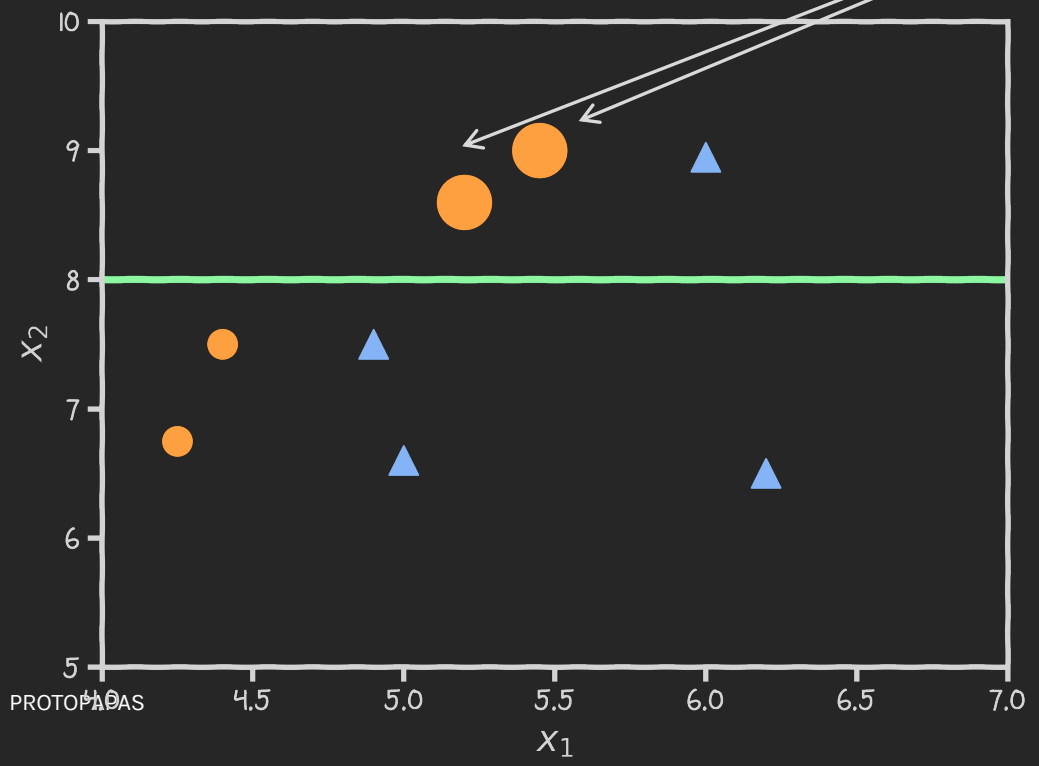
$$w^{(1)} \leftarrow \frac{w^{(1')}}{Z} \approx \frac{1.5}{8}$$

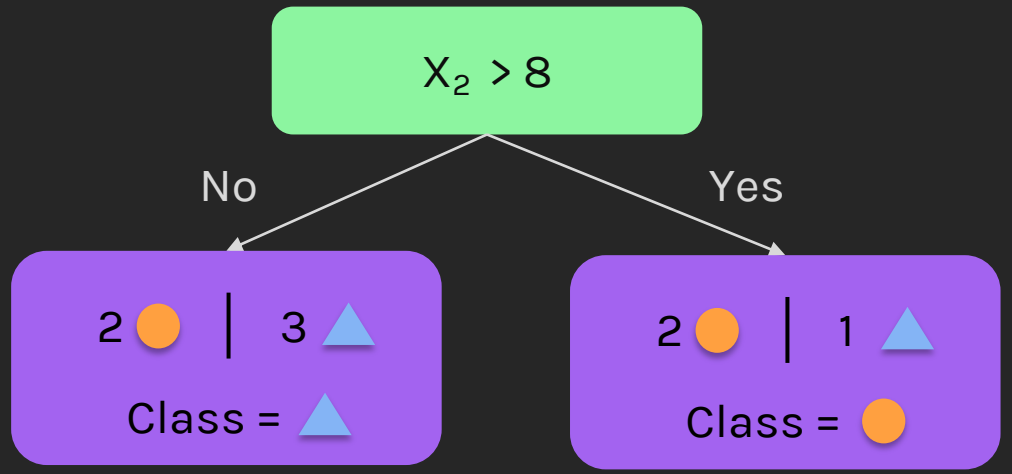
$$w^{(1)} \leftarrow \frac{w^{(1')}}{Z} \approx \frac{0.8}{8}$$

AdaBoost

Step 5: Create another stump $S^{(1)}$ on the re-weighted data.

Notice that the $S^{(1)}$ has correctly classified the data points that $T^{(0)}$ misclassified.



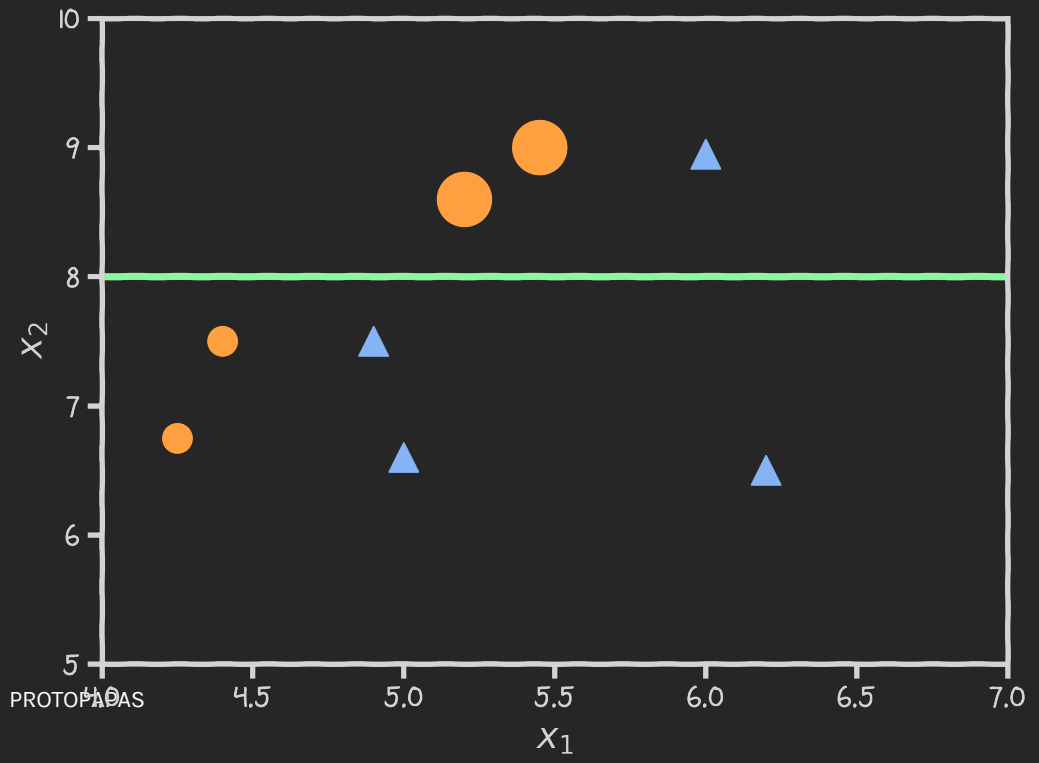


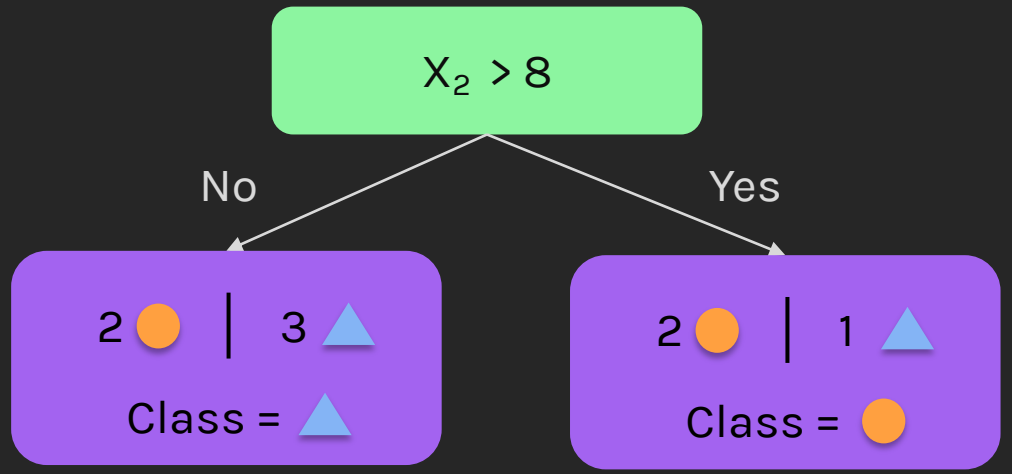
AdaBoost

The Greek column symbol 𝕀 just means: 1 when the stump is wrong, 0 when it's right.

Step 6: With the new weights, calculate the total error in the stump using:

$$\epsilon^{(1)} = \sum_{n=1}^{N} w_n^{(1)} \mathbb{I}\left(S^{(1)}(x_n) \neq y_n\right)$$



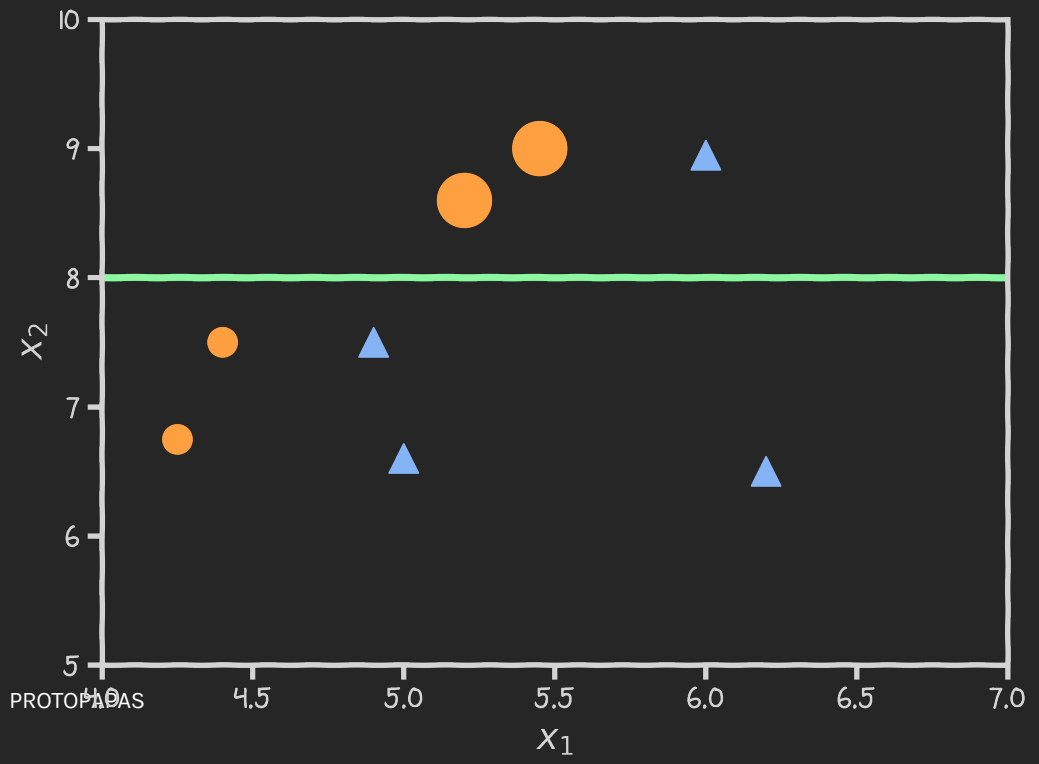


$$Error\ (\epsilon) = \frac{0.8}{8} + \frac{0.8}{8} + \frac{0.8}{8} = 0.3$$

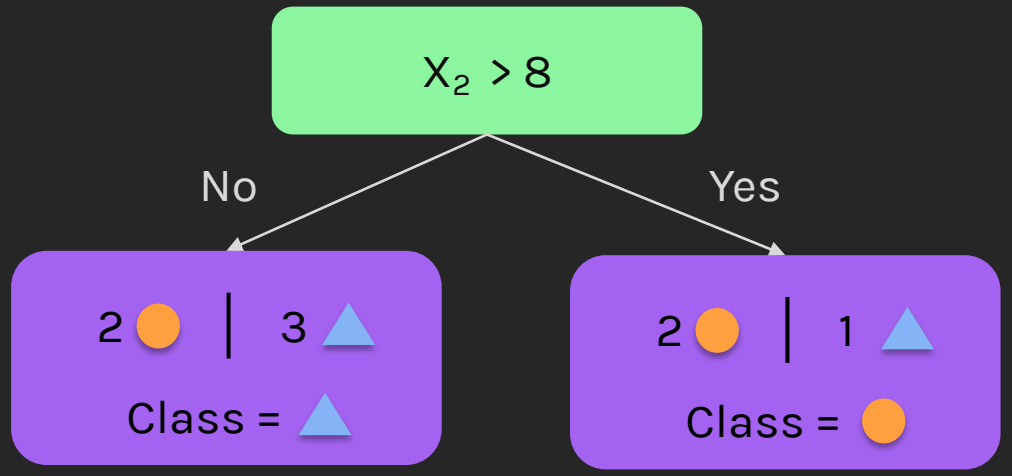
AdaBoost

Step 7: Assign the stump a scale, $\lambda^{(1)}$, that indicates how much it **contributes to the entire ensemble.**

Let this STUMP'S weight $\lambda^{(1)}$ be 0.20.



$\lambda^{(1)} = 0.20$

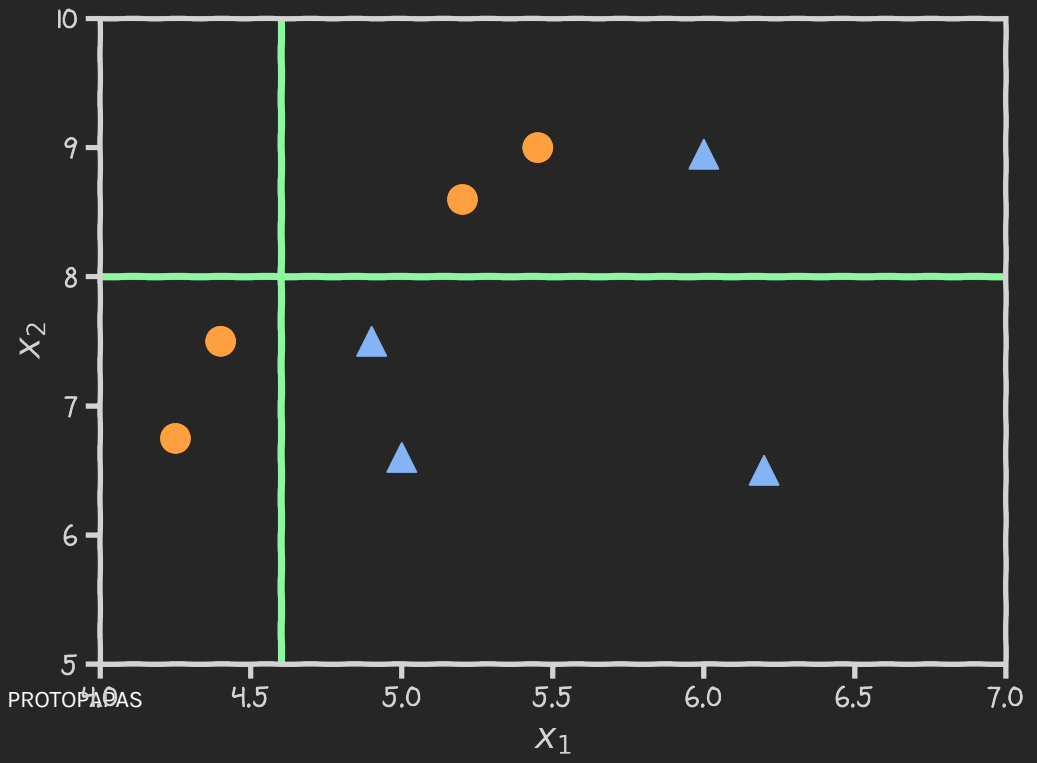


$Error\ (\epsilon) = 0.3$

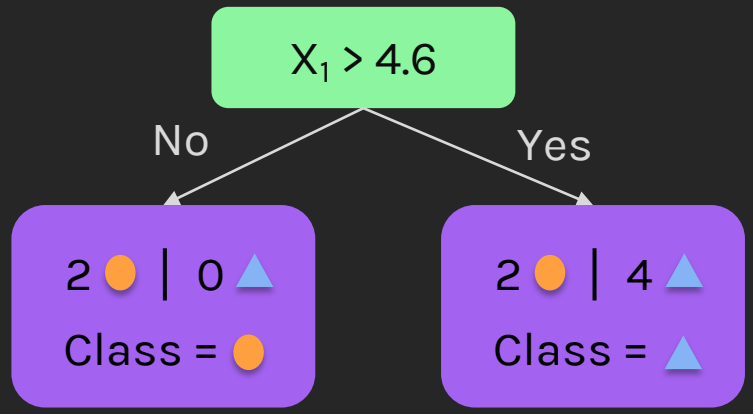
AdaBoost

Step 8: Construct the **ensemble model** $T^{(1)}$ using:

$$T^{(i)} \leftarrow \begin{cases} \lambda^{(i)} S^{(i)} & i = 0 \\ T^{(i-1)} + \lambda^{(i)} S^{(i)} & i = 1, 2, \ldots \end{cases}$$

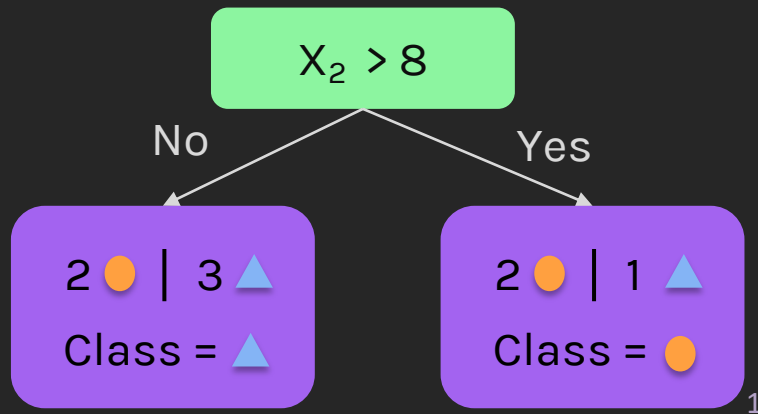


0.25 *



+

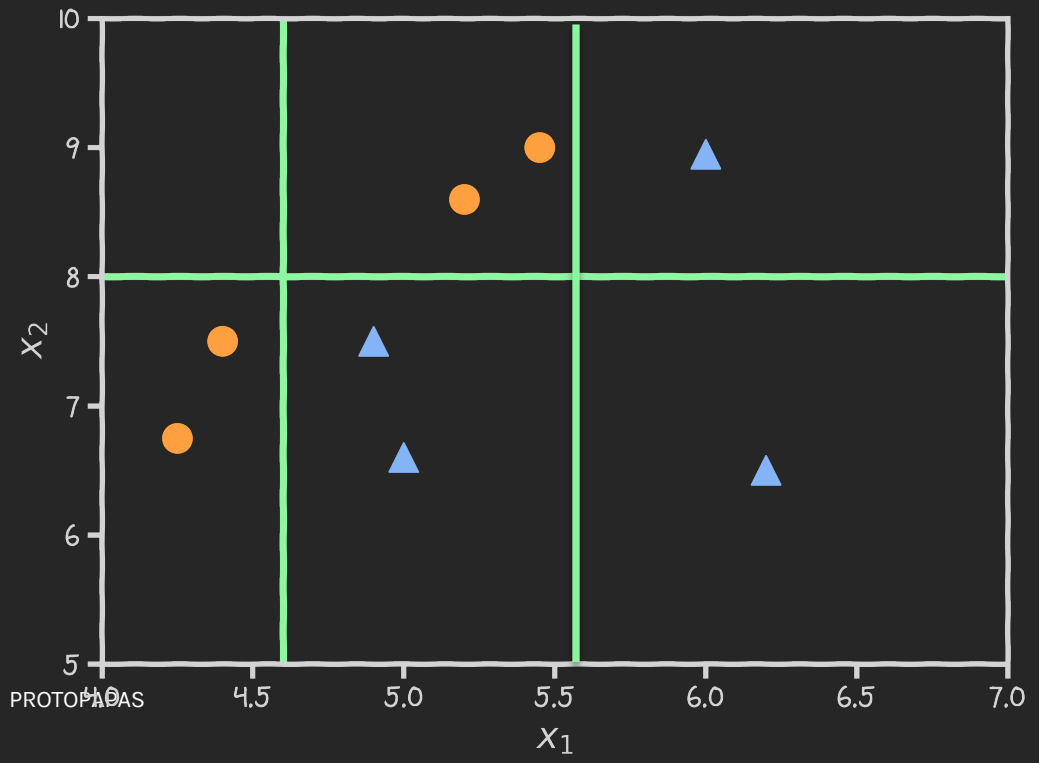
0.20 *



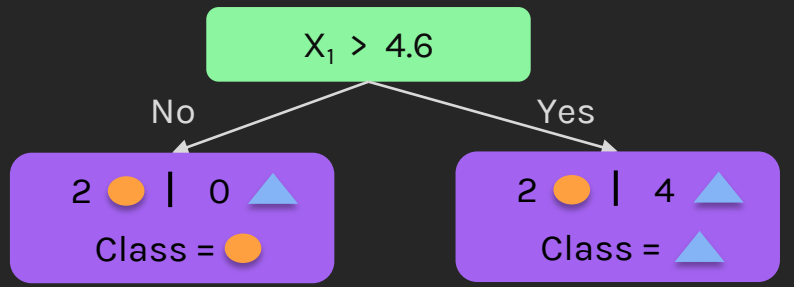
AdaBoost

Repeating the same process again, we get:

Thus, the ensemble keeps growing in terms of the number of stumps being used.

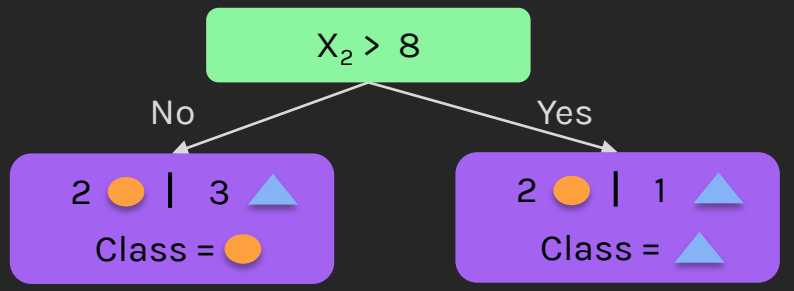


0.25 *



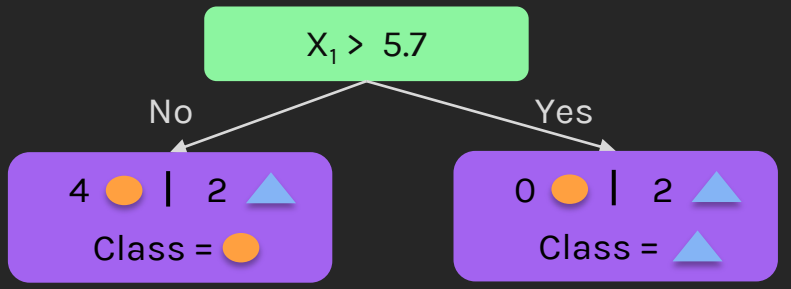
+

0.20 *



+

0.42 *



AdaBoost: SUMMARY

Given n, training data points, set $w_n^{(0)} = 1/N$

For $i = 0 \ldots$ until stopping condition is met:

- Train a weak learner $S^{(i)}$ using weights $w_n^{(i)}$.
- Calculate the total error, $\epsilon^{(i)}$, of the weak learner.
- Calculate the importance of each model $\lambda^{(i)}$.
- Combine all stumps into the new ensemble model, $T^{(i)}$.
- Update the sample weights so misclassified points receive higher weight and correctly classified points receive lower weight.

AdaBoost

Step 1: Given training data $\{(x_1, y_1), \ldots, (x_N, y_N)\}$, choose an initial distribution $w_n^{(0)} = 1/N$.

For $i = 0 \ldots$ until stopping condition is met:

Step 2: Train a weak learner $S^{(i)}$ using weights $w_n^{(i)}$.

Step 3: Calculate the total error of the weak learner using:

$$\epsilon^{(i)} = \sum_{n=1}^{N} w_n^{(i)} \, \mathbb{I}\left(y_n \neq S^{(i)}(x_n)\right)$$

Step 4: Calculate the importance of each model $\lambda^{(i)}$.

Step 5: Construct the ensemble model using:

$$T^{(i)} \leftarrow \begin{cases} \lambda^{(i)} S^{(i)} & i = 0 \\ T^{(i-1)} + \lambda^{(i)} S^{(i)} & i = 1, 2, \ldots \end{cases}$$

Step 6: Update the sample weights so that misclassified points receive higher weight and correctly classified points receive lower weight:

$$w_n^{(i+1)} \leftarrow \frac{w_n^{(i)} \, e^{-\lambda^{(i)} \, y_n \, S^{(i)}(x_n)}}{Z}$$

Final model: $T^{(i)}(x) = sign\left[\sum_{i=1}^{T} \lambda^{(i)} \, S^{(i)}(x)\right]$

AdaBoost

Step 1: Given training data $\{(x_1, y_1), \ldots, (x_N, y_N)\}$, choose an initial distribution $w_n^{(0)} = 1/N$.

For $i = 0 \ldots$ until stopping condition is met:

Step 2: Train a weak learner $S^{(i)}$ using weights $w_n^{(i)}$.

Step 3: Calculate the total error of the weak learner using:

$$\epsilon^{(i)} = \sum_{n=1}^{N} w_n^{(i)} \mathbb{I}\left(y_n \neq S^{(i)}(x_n)\right)$$

Step 4: Calculate the importance of each model $\lambda^{(i)}$.

Step 5: Construct the ensemble model

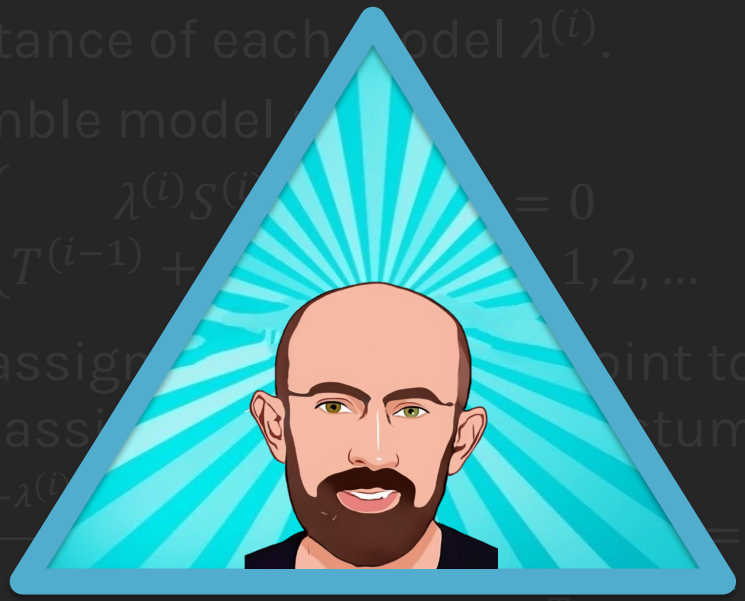
$$T^{(i)} \leftarrow \begin{cases} \lambda^{(i)} S^{(i)} & = 0 \\ T^{(i-1)} + & 1, 2, \ldots \end{cases}$$

Step 6: Adjust the weights assign int to ensure the next stump focuses on the points misclassi stump

$$w_n^{(i+1)} \leftarrow \frac{w_n^{(i)} e^{-\lambda^{(i)}}}{} \qquad = sign\left[T^{(i-1)}(x) + \lambda^{(i)} S^{(i)}(x)\right]$$

Final model: $T^{(i)}(x) = sign\left[\sum_{i=1}^{T} \lambda^{(i)} S^{(i)}(x)\right]$

GREAT PROFESSORS always KNOW your next questions!



AdaBoost

Step 1: Given training data $\{(x_1, y_1), \ldots, (x_N, y_N)\}$, choose an initial distribution $w_n^{(0)} = 1/N$.

For $i = 0 \ldots$ until stopping condition is met:

Step 2: Train a weak learner $S^{(i)}$ using weights $w_n^{(i)}$.

Step 3: Calculate the total error of the weak learner using:

$$\epsilon^{(i)} = \sum_{n=1}^{N} w_n^{(i)} \, \mathbb{I}\left(y_n \neq S^{(i)}(x_n)\right)$$

Step 4: Calculate the importance of each model $\lambda^{(i)}$.

Step 5: Construct the ensemble model using:

$$T^{(i)} \leftarrow \begin{cases} \lambda^{(i)} S^{(i)} & i = 0 \\ T^{(i-1)} + \lambda^{(i)} S^{(i)} & i = 1, 2, \ldots \end{cases}$$

Step 6: Adjust the weights assigned to each data point to ensure the next stump focuses on the points misclassified by the previous stump

$$w_n^{(i+1)} \leftarrow \frac{w_n^{(i)} \, e^{-\lambda^{(i)} \, y_n \, S^{(i)}(x_n)}}{Z},$$

Final model: $T^{(i)}(x) = sign\left[\sum_{i=1}^{T} \lambda^{(i)} \, S^{(i)}(x)\right]$

AdaBoost

- **How do I create a stump?**
  In AdaBoost the stumps are created using simple decision trees with `max_depth = 1`.

- **How to use the normalized weights to make the stump?**

  There are two options:

  A. Create a new dataset of same size of the original dataset with repetition based on the newly updated sample weight.

  B. Use a weighted version of the Gini index.

- **How do I calculate the scaling factor $\lambda^{(i)}$ of each stump?**
  Next few slides…


PPPP BOOSTING

AdaBoost

Recall in gradient boosting for regression, we minimize the MSE loss.

In AdaBoost, we minimize a different function, we call **exponential loss:**

$$Exp\ Loss = \frac{1}{N}\sum_{n=1}^{N} e^{(-y_n \hat{y}_n)} \quad \text{where} \quad y_n \in \{-1, 1\}$$

Exponential loss is differentiable with respect to $\hat{y}_n$ and always larger than or equal to the 0–1 error. So minimizing it indirectly reduces classification error.

Choosing the Learning Rate

Unlike in the case of gradient boosting for regression, we can analytically solve for the optimal learning rate for AdaBoost, by optimizing:

$$\underset{\lambda}{\operatorname{argmin}} \frac{1}{N} \sum_{n=1}^{N} e^{(-y_n(T+\lambda^{(i)} S^{(i)}(x_n)))}$$

Doing so, we get: $\lambda^{(i)} = \frac{1}{2} \ln\left(\frac{1-\epsilon}{\epsilon}\right)$ where

$\epsilon = \sum_{n=1}^{N} w_n^{(i)} \mathbb{I}(y_n \neq T^{(i)}(x_n))$ and $w_n^{(i+1)} \leftarrow \frac{w_n^{(i)} e^{-\lambda^{(i)} y_n S^{(i)}(x_n)}}{Z}$

Lecture Outline

- AdaBoost
- **Mathematical Formulation - AdaBoost**
- Final Thoughts on Boosting

Gradient Descent with Exponential Loss

$$Exp\ Loss = \frac{1}{N}\sum_{n=1}^{N} e^{(-y_n \hat{y}_n)}$$

Similar to what we did for gradient boosting, compute the gradient for the loss w.r.t the predictions:

$$\nabla_{\hat{y}} Exp\ Loss = [-y_1 e^{(-y_1 \hat{y}_1)}, \dots, -y_n e^{(-y_n \hat{y}_n)}]$$

Set $w_n = \exp(-y_n \hat{y}_n)$. Notice that when $y_n = \hat{y}_n$, the weight w_n is small; when $y_n \neq \hat{y}_n$, the weight is larger.

$$\nabla_{\hat{y}} Exp\ Loss = [-y_1 w_1, \dots, -y_n w_n]$$

This way, we see that the gradient is just a re-weighting applied to the target values.

Gradient Descent with Exponential Loss

The update step in the gradient descent is

$$\hat{y}_n \leftarrow \hat{y}_n + \lambda w_n y_n, \qquad n = 1, \dots, N$$

Just like in gradient boosting, we approximate the gradient, $w_n y_n$, with a simple model, $S^{(i)}$, that depends on x_n.

This means training $S^{(i)}$ on a re-weighted set of target values,

$$\{(x_1, w_1 y_1), \dots, (x_N, w_N y_N)\}$$

That is, gradient descent with exponential loss, means iteratively training simple models that **focuses on the points misclassified by the previous model.**

Comparison: Gradient Boosting and AdaBoost

GRADIENT BOOST

We minimize the **MSE:**

$$MSE = \frac{1}{N}\sum_{n=1}^{N}(y_n - \hat{y}_n)^2$$

Compute the gradient for the loss w.r.t predictions:

$$\nabla_{\hat{y}} MSE = -2[r_1, \dots r_n]$$

The update step is:

$$\hat{y}_n \leftarrow \hat{y}_n + \lambda \hat{r}_n, \qquad n = 1, \dots, N$$

ADABOOST

We minimize the **exponential loss:**

$$Exp\ Loss = \frac{1}{N}\sum_{n=1}^{N} e^{(-y_n \hat{y}_n)} \quad \text{where} \quad y_n \in \{-1, 1\}$$

Compute the gradient for the loss w.r.t predictions:

$$\nabla_{\hat{y}} Exp\ Loss = [-y_1 e^{(-y_1 \hat{y}_1)}, \dots, -y_n e^{(-y_n \hat{y}_n)}]$$

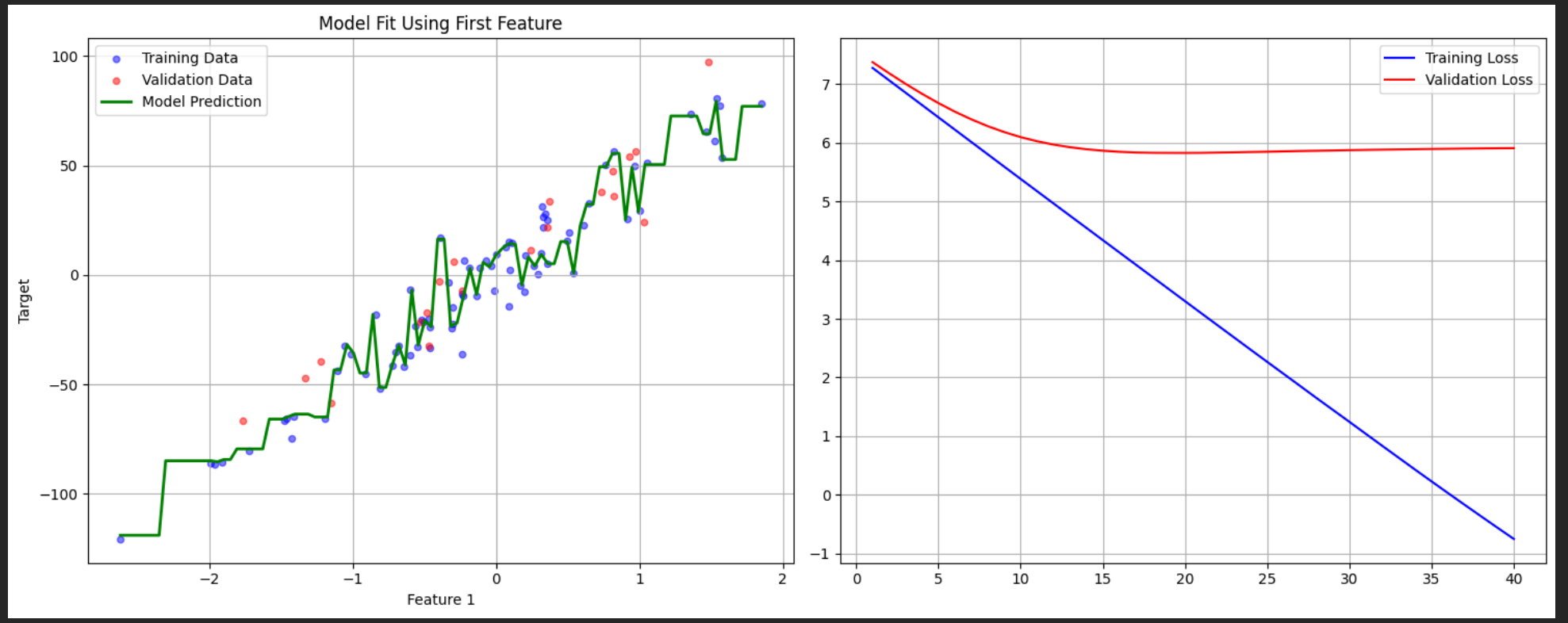
Setting $w_n = \exp(-y_n \hat{y}_n)$:

$$\nabla_{\hat{y}} Exp\ Loss = [-y_1 w_1, \dots, -y_n w_n]$$

The update step is:

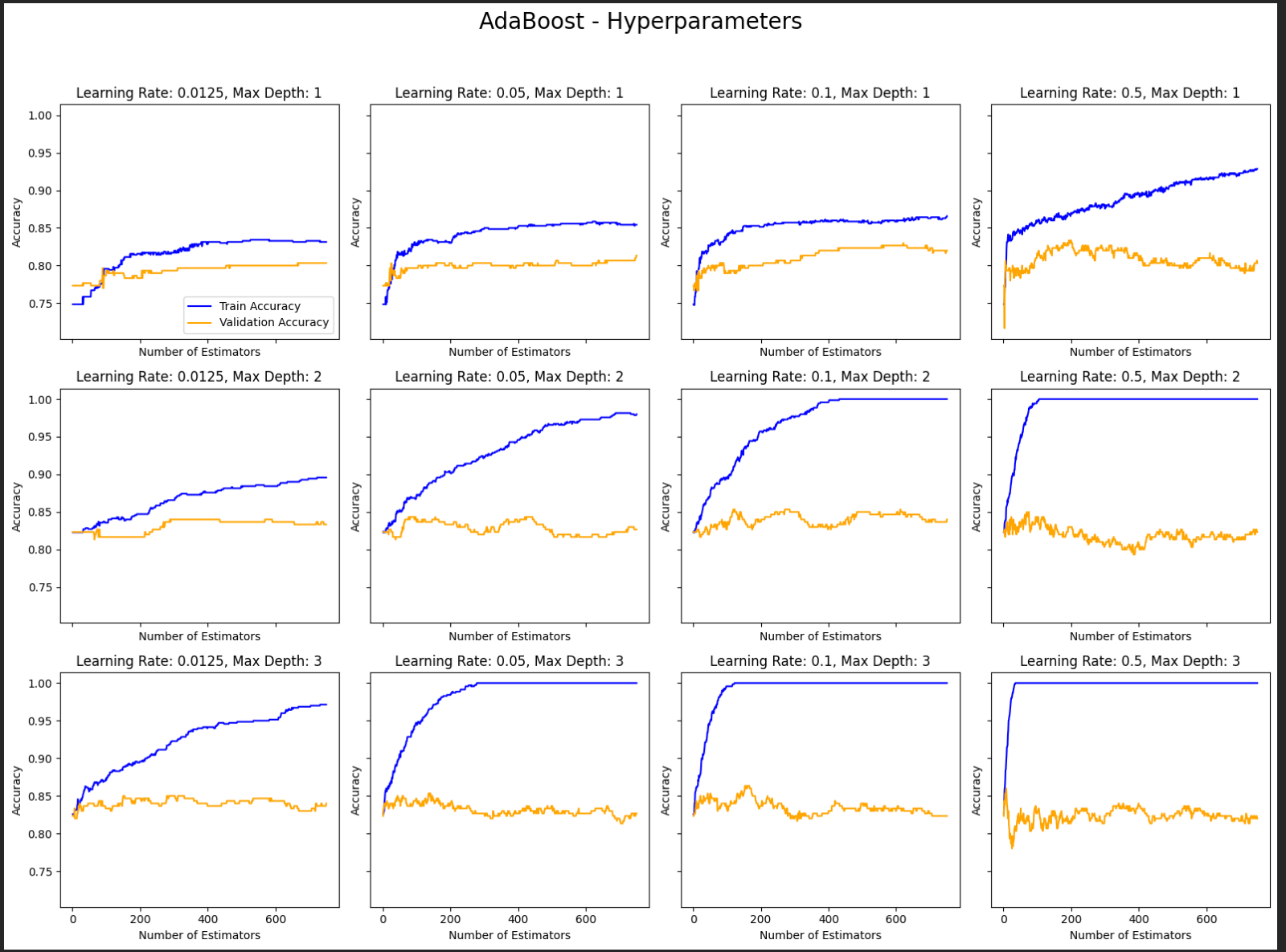
$$\hat{y}_n \leftarrow \hat{y}_n + \lambda S^{(i)} \qquad n = 1, \dots, N$$

Overfitting in Boosting



- Boosting often overfits more slowly than other models, especially when using weak learners like stumps. But it will overfit if run for too many iterations.
- However, notice how using 40 estimators leads to overfitting on the val datasets.

Overfitting in Boosting

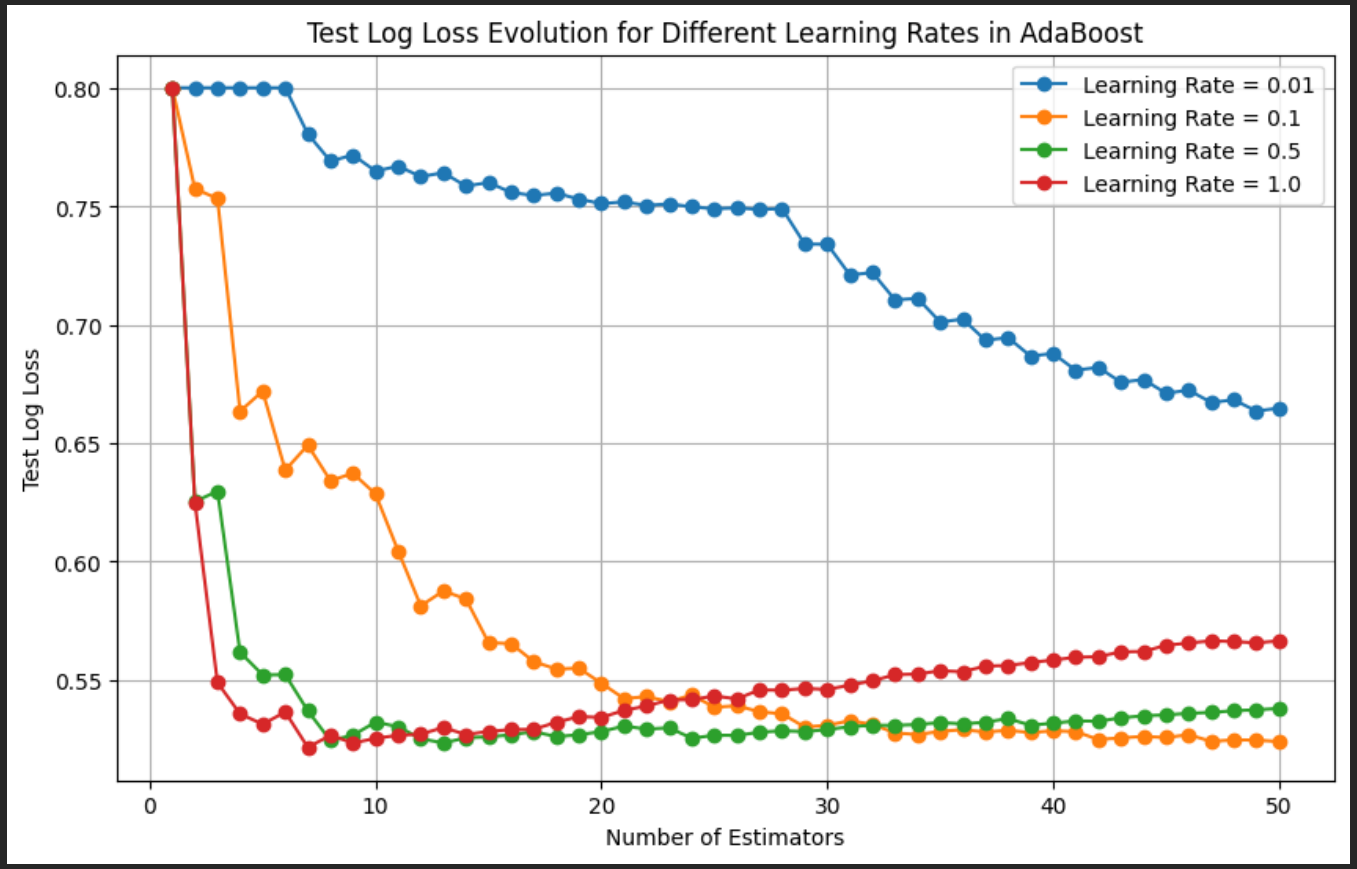


What hyperparameters can you tune?

Defaults:

- `n_estimators = 50`
- `learning_rate = [0.01-1.0]`
- `algorithm = "SAMME"`
- `random_state = None`

SAMME minimizes a multiclass exponential loss function. SAMME.R combines probabilities predicted by the stumps



Just like gradient descent, a small learning rate leads to slow learning while a large value can lead to poor generalization and higher test loss.

Lecture Outline

- AdaBoost
- Mathematical Formulation - AdaBoost
- **Final Thoughts on Boosting**

Lecture Outline

- AdaBoost
- Mathematical Formulation - AdaBoost
- **Final Thoughts on Boosting**

Final thoughts on Boosting

- **Stopping condition:** Same as gradient boosting: maximum iteration (number of stumps) *n_estimators*, minimum improvement in loss, number of iterations without improvement in the loss.
- **Overfitting**: Unlike other ensemble methods like bagging and Random Forest, boosting methods like AdaBoost will overfit if run for many iterations. Some libraries implement regularization methods which is out of the scope in this course.
- **Hyper-parameters**: All parameters associated with the stump and the stopping conditions mentioned before.

Final thoughts on Boosting

There are few implementations on boosting:

- **XGBoost**: An efficient Gradient Boosting Decision .
- **LGBM**: Light Gradient Boosted Machines. It is a library for training GBMs developed by Microsoft, and it competes with XGBoost.
- **CatBoost**: A new library for Gradient Boosting Decision Trees, offering appropriate handling of categorical features.

Everything you need to know about XGBoost (Extreme Gradient Boosting)!

- Highly optimized SoTA implementation of gradient boosting.
- **Regularization:** L1 and L2 regularization to prevent overfitting.
- **Parallelization:** Uses CPU cores for constructing trees in parallel.
- **Distributed Computing:** Scales well for large datasets using frameworks like Spark, Hadoop.
- **Cross-Validation:** Can tune hyperparameters easily using cross-validation.
- **Missing Values:** Handles missing values automatically.

Everything you need to know about XGBoost (Extreme Gradient Boosting)!

Dataset Preparation: `DMatrix`, aka XGBoost's optimized data structure.

Model: `XGBClassifier, XGBRegressor.`

Hyperparameters:

- Model Complexity: `{max_depth, min_child_weight, gamma}`
- Regularization: `{lambda, alpha}`
- Learning: `{learning_rate, n_estimators}`
- Loss Functions: `{objective, eval_metric}`

Learning Objectives: By the end of this lecture, you should be able to:

- **Explain how AdaBoost updates sample weights** to focus learning on past mistakes.
- **Describe why AdaBoost is a boosting algorithm**, and how each new stump corrects previous errors.
- **Explain how weighted decision stumps are constructed** and why weights affect the split.
- **Understand how the optimal stump weight λ is derived** from minimizing exponential loss.
- **Recognize that AdaBoost performs functional gradient descent**, where each stump approximates the negative gradient.
- **Identify when AdaBoost overfits** and how the number of estimators influences generalization.

Game time

Are decision trees parametric or non-parametric?

Pavlos Version.

A. Parametric — because they estimate parameters (splits and thresholds).

B. Parametric — because they produce a finite set of decision rules.

C. Non-parametric — because the number of parameters is not fixed in advance and grows with the data.

D. Non-parametric — because they do not assume a specific functional form for the relationship.

Game time

Are decision trees parametric or non-parametric?

Pavlos Version.

A. Parametric — because they estimate parameters (splits and thresholds).

B. Parametric — because they produce a finite set of decision rules.

C. Non-parametric — because the number of parameters is not fixed in advance and grows with the data.

D. Non-parametric — because they do not assume a specific functional form for the relationship.

Game time

Are decision trees parametric or non-parametric?

Kevin Version.

A. Parametric — because they estimate parameters (splits and thresholds).

B. Parametric — because they produce a finite set of decision rules.

C. Non-parametric — because the number of parameters is not fixed in advance and grows with the data.

D. Non-parametric — because they do not assume a specific functional form for the relationship.

Game time

Are decision trees parametric or non-parametric?

Kevin Version.

A. Parametric — because they estimate parameters (splits and thresholds).

B. Parametric — because they produce a finite set of decision rules.

C. Non-parametric — because the number of parameters is not fixed in advance and grows with the data.

D. Non-parametric — because they do not assume a specific functional form for the relationship.

The latest LLM

XGBoost

Thank you