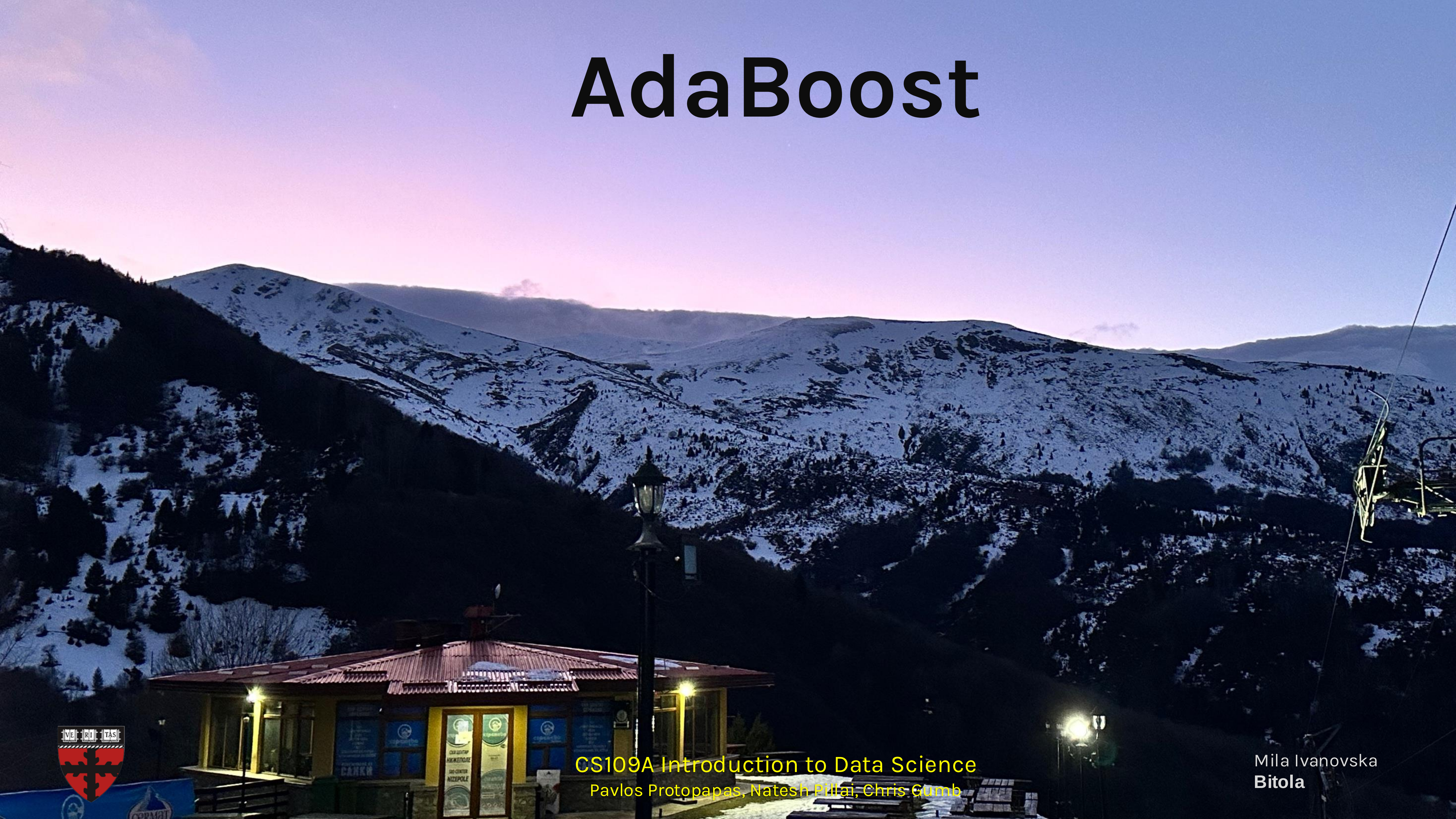
AdaBoost



CS109A Introduction to Data Science

Pavlos Protopapas, Natesh Pillai, Chris Gumb

Mila Ivanovska
Bitola

Lecture Outline

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

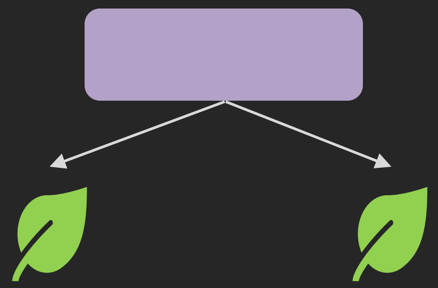
Lecture Outline

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

AdaBoost

There are two main ideas in AdaBoost:

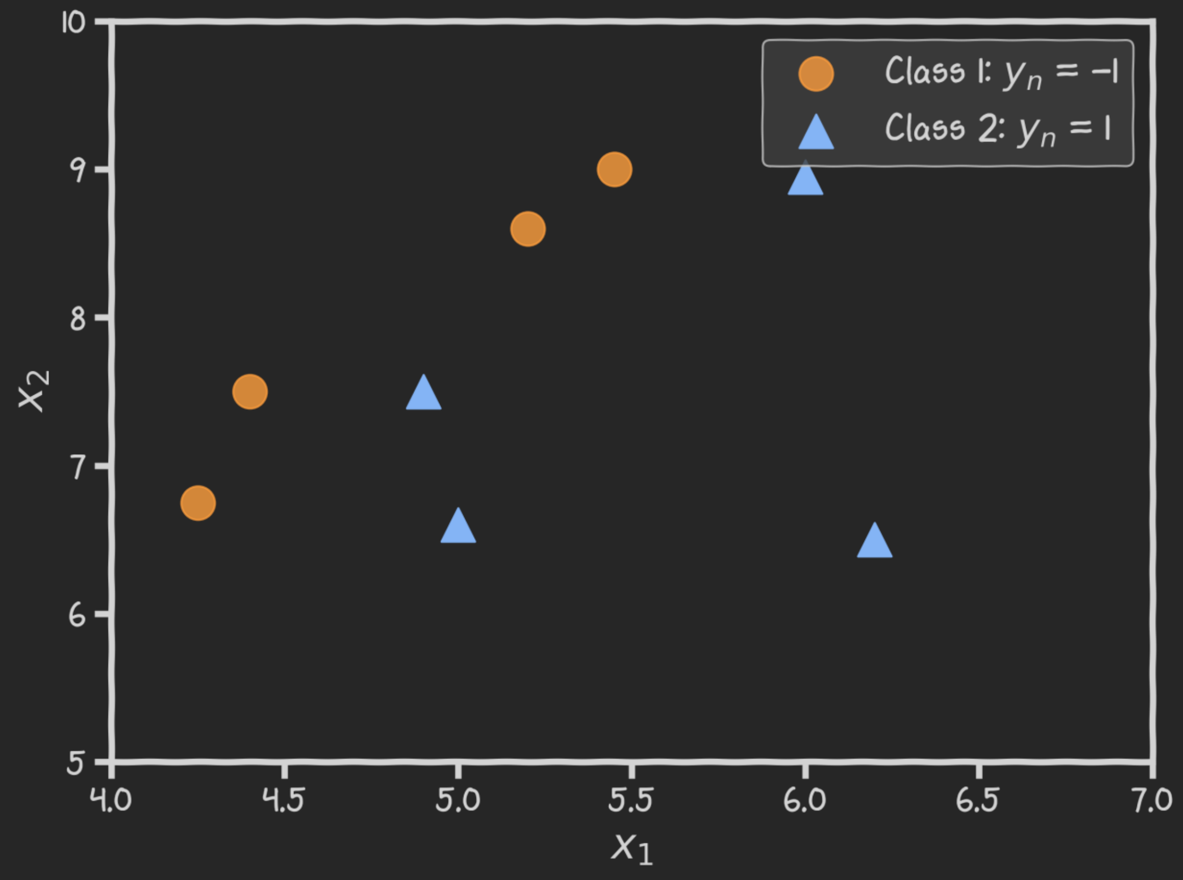
- Idea #1: **Iteratively** build a complex model T by combining several **weak learners**.
  For trees, a weak learner is a tree with 1 node with 2 leaves. We call this a **stump**.



- 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.

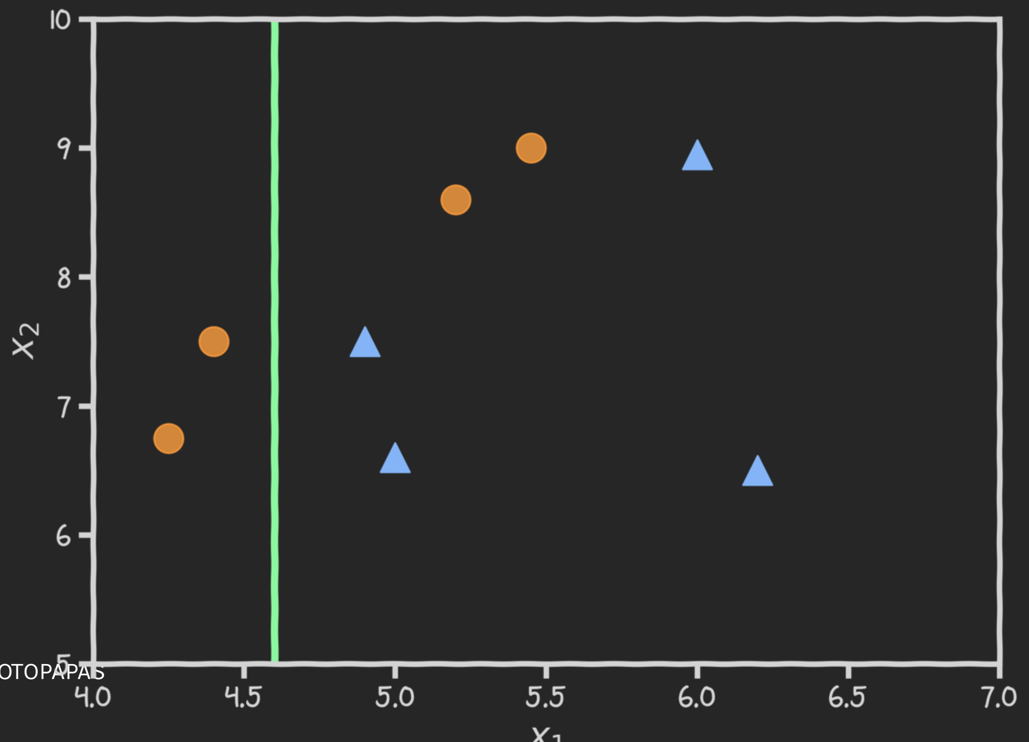
AdaBoost

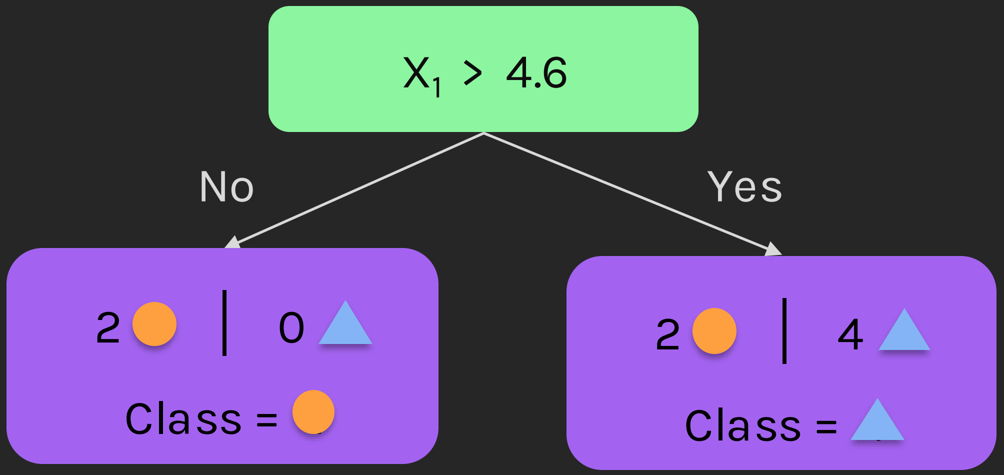
Consider the following dataset:



AdaBoost

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

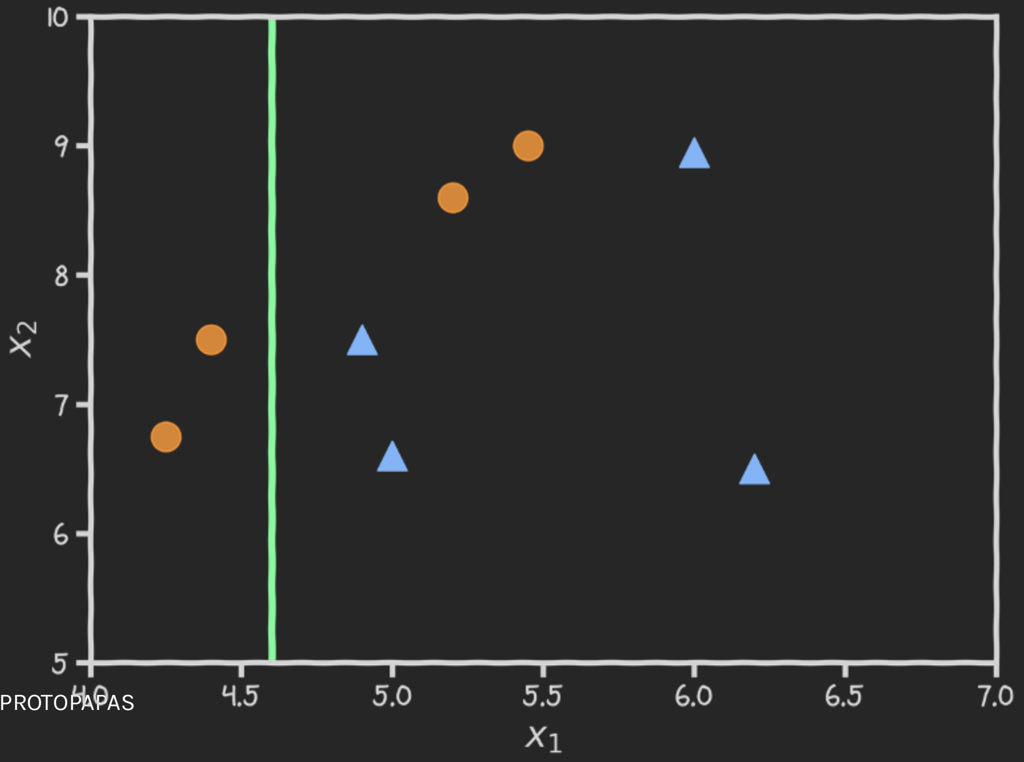




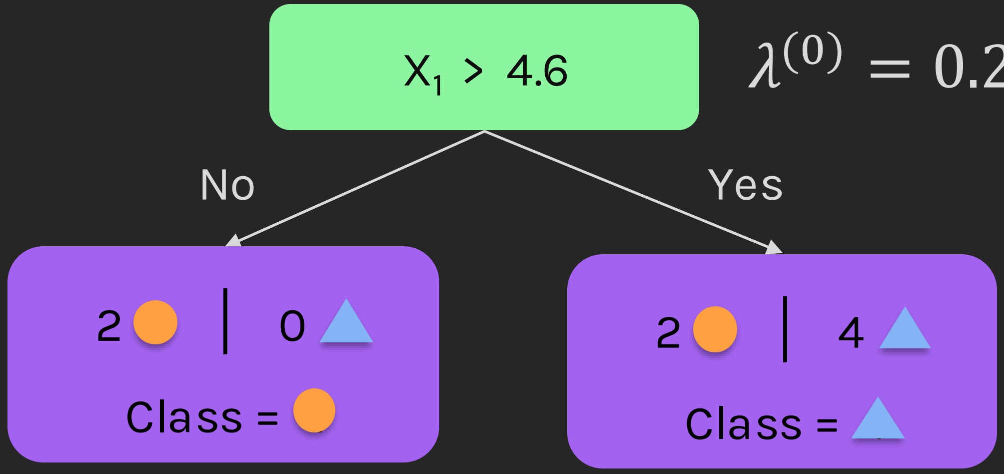
AdaBoost

Step 3: 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.



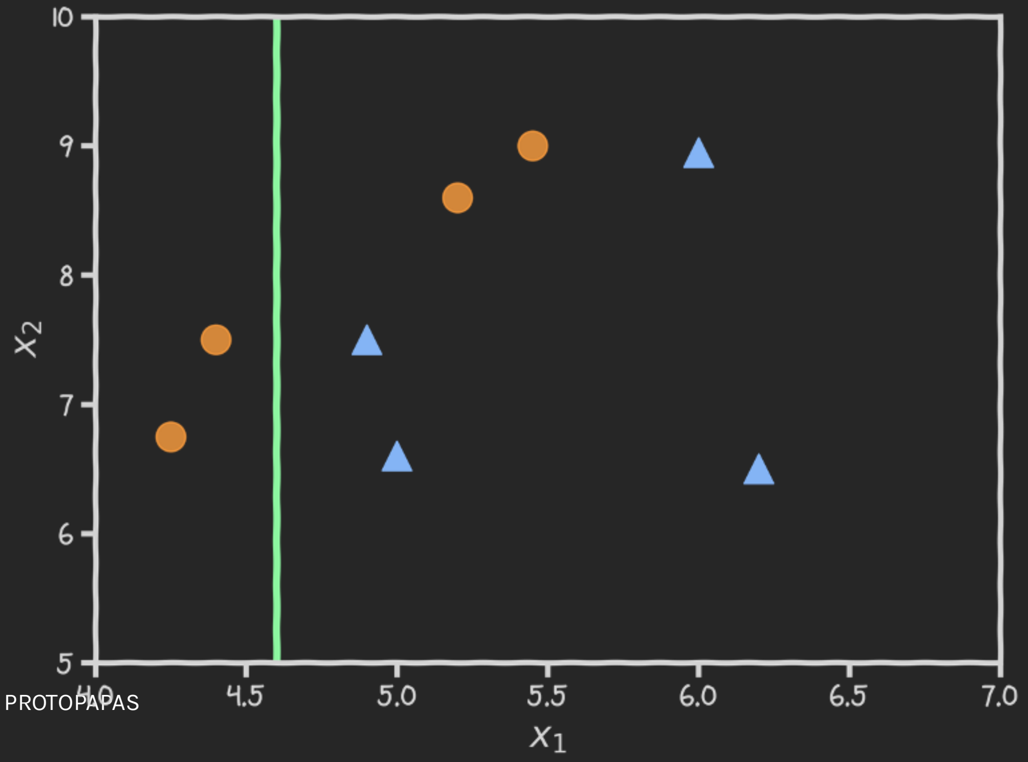
$$\lambda^{(0)} = 0.25$$

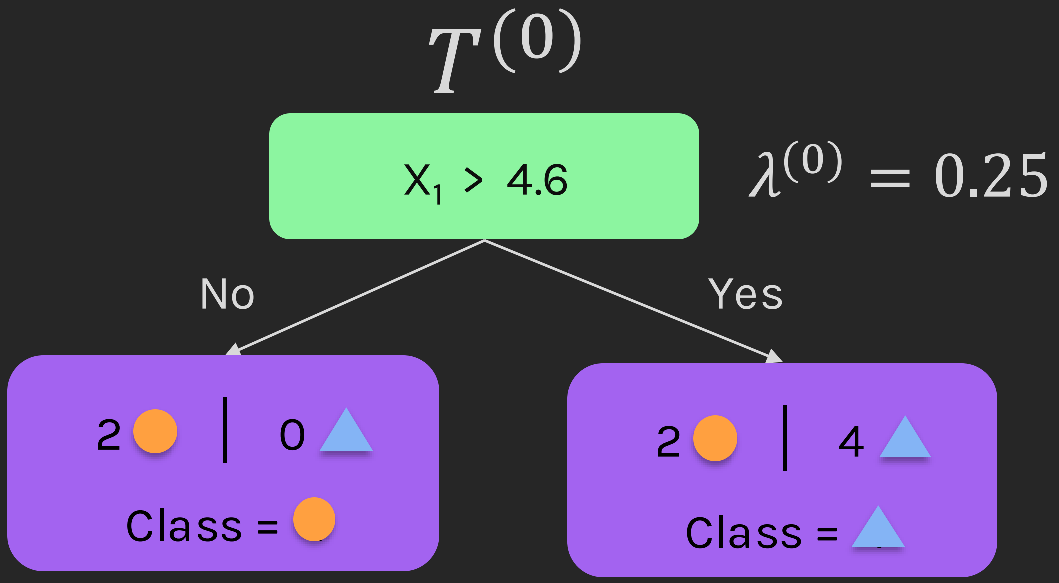
$$Error\ (\epsilon) = \frac{2}{8}$$

AdaBoost

Step 4: Construct the **ensemble model** $T^{(0)}$ using:

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

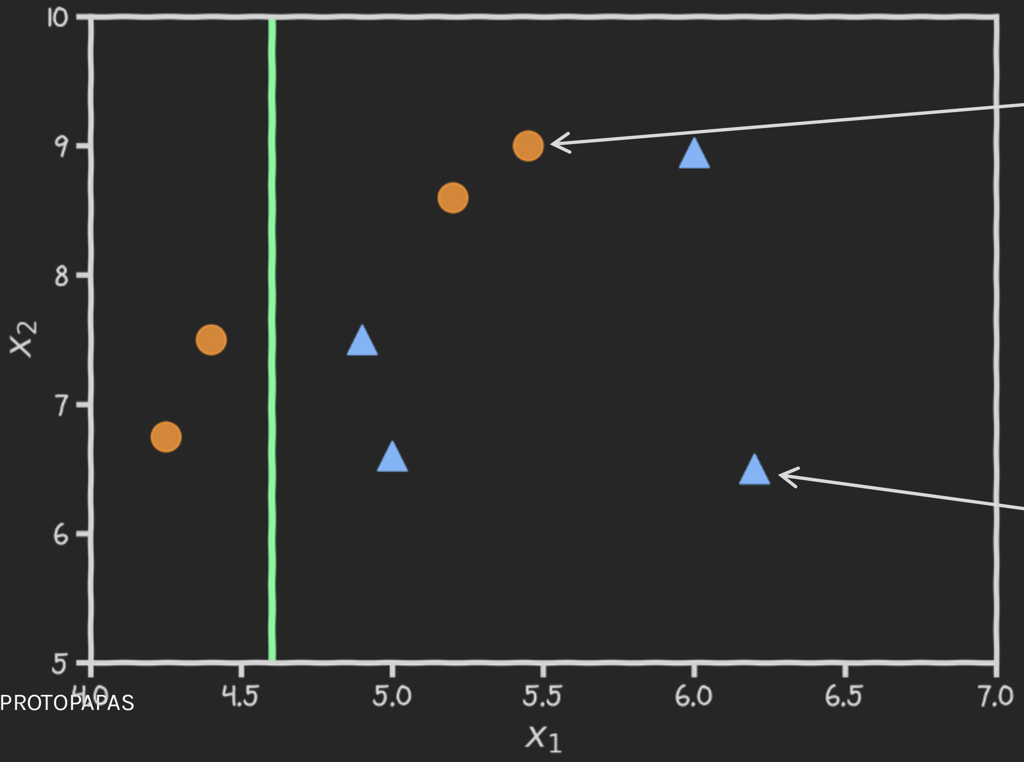




$$Error\ (\epsilon) = \frac{2}{8}$$

AdaBoost

Step 5: Adjust the weights assigned to each data point to ensure the next stump focuses on the points **misclassified by the current model**.



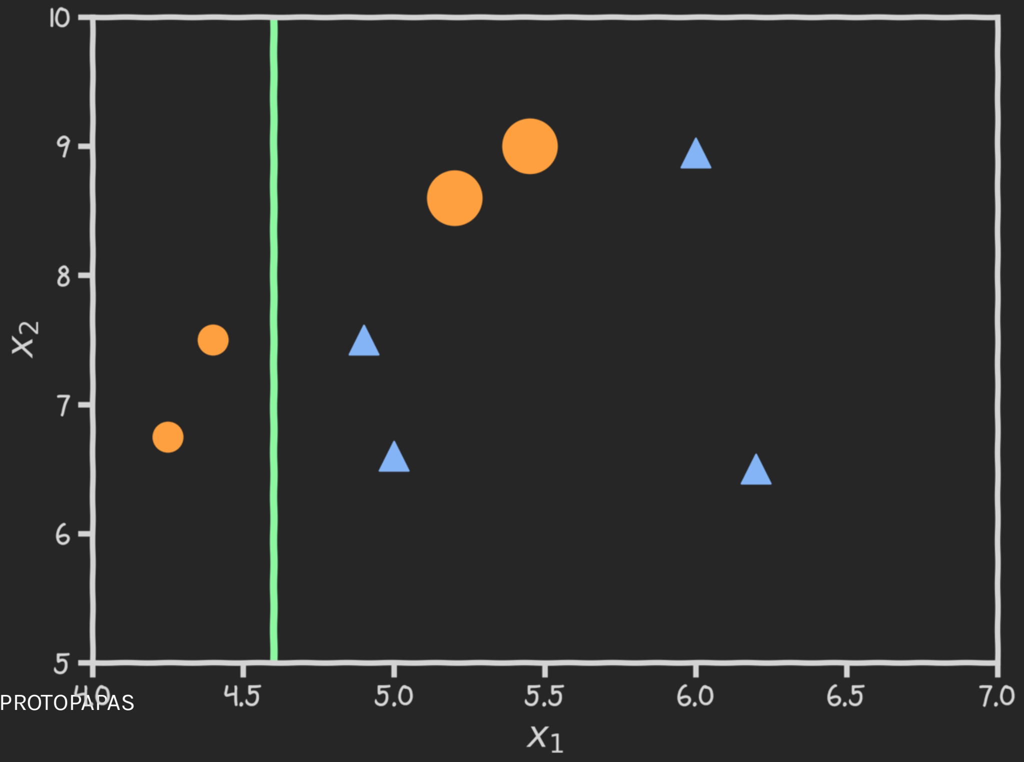
Increase the weights of misclassified samples

Decrease the weights of correctly classified samples

AdaBoost

Step 5: Adjust the weights assigned to each data point to ensure the next stump focuses on the points **misclassified by the current model**.

$$w_n^{(1)} \leftarrow \frac{w_n^{(0)} e^{-\lambda \, y_n \, T^{(0)}(x_n)}}{Z} = \frac{w_n^{(1')}}{Z}$$



AdaBoost

ground truth -1 and 1

prediction from
previous ensemble

$$w_n^{(1)} \leftarrow \frac{w_n^{(0)} e^{-\lambda\, y_n\, T^{(0)}(x_n)}}{Z} = \frac{w_n^{(1')}}{Z}$$

new weights

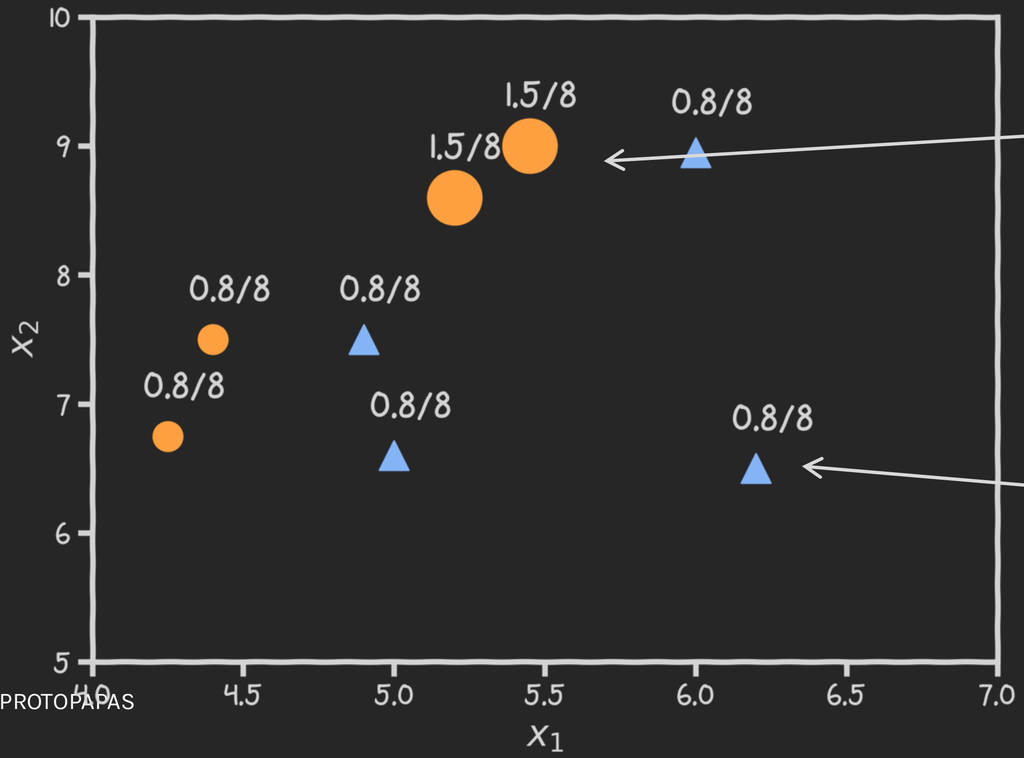
previous weights

normalizes the new weights

AdaBoost

Step 5: Adjust the weights assigned to each data point to ensure the next stump focuses on the points **misclassified by the ensemble**.

$$w_n^{(1)} \leftarrow \frac{w_n^{(0)} e^{-\lambda\, y_n\, T^{(0)}(x_n)}}{Z} = \frac{w_n^{(1')}}{Z}$$



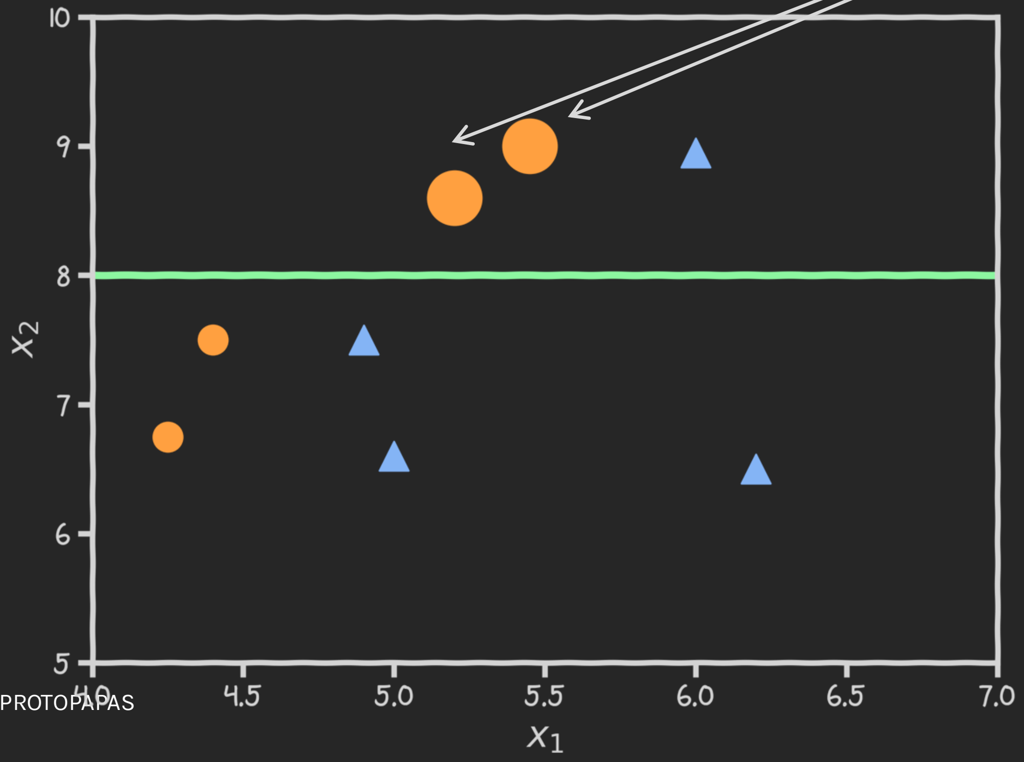
$$w^{(1)} \leftarrow \frac{w^{(1')}}{Z} \approx \frac{1.5}{8}$$

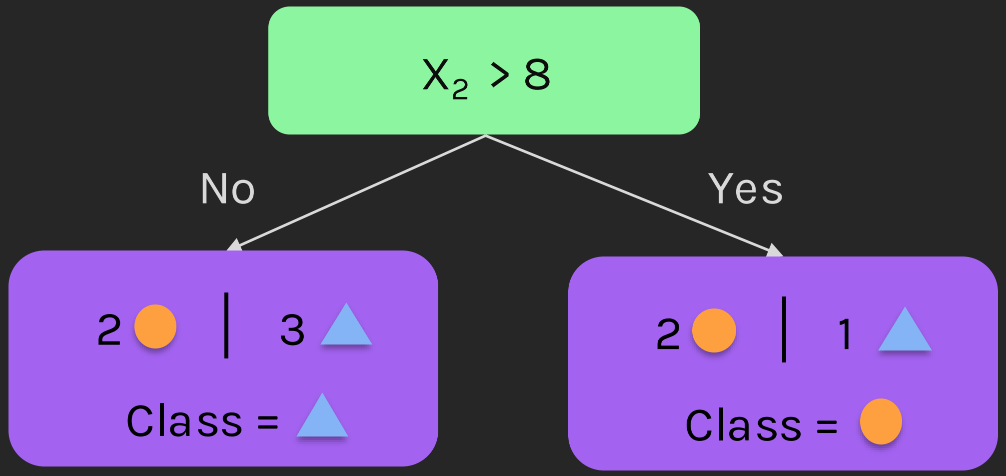
$$w^{(1)} \leftarrow \frac{w^{(1')}}{Z} \approx \frac{0.8}{8}$$

AdaBoost

Step 6: 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.

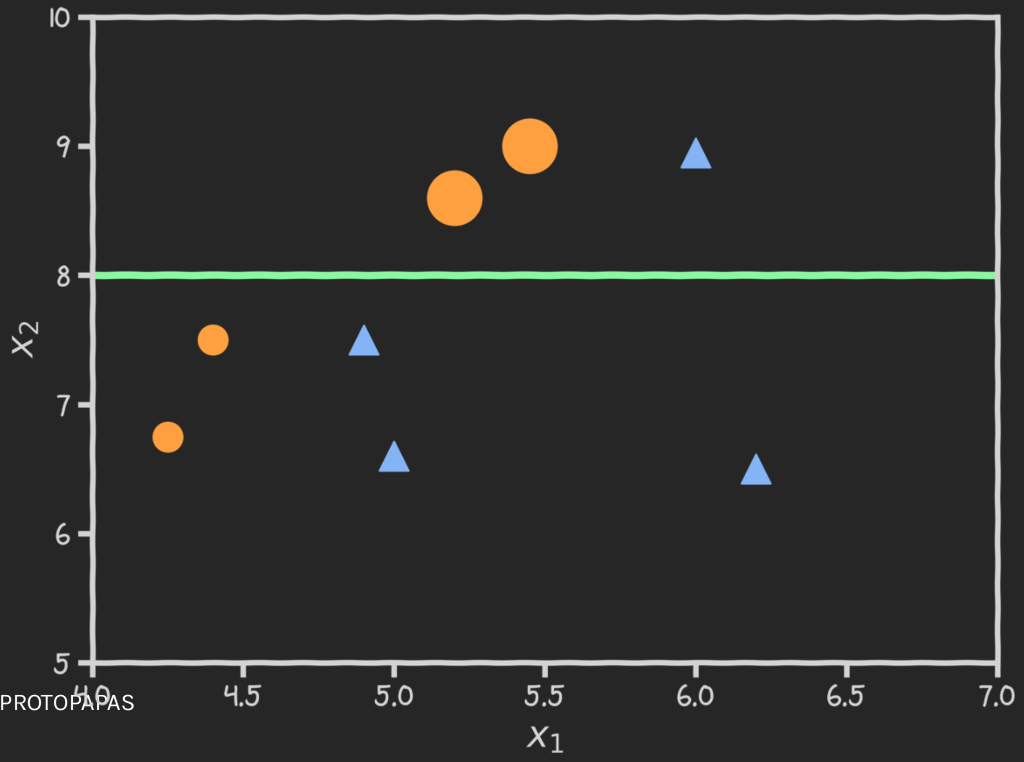


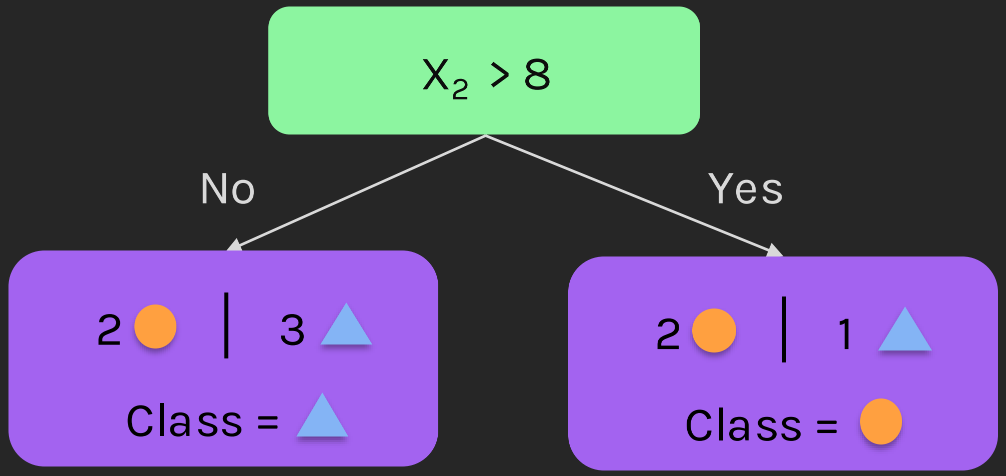


AdaBoost

Step 7: With the new weights, calculate the total error in the stump using:

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





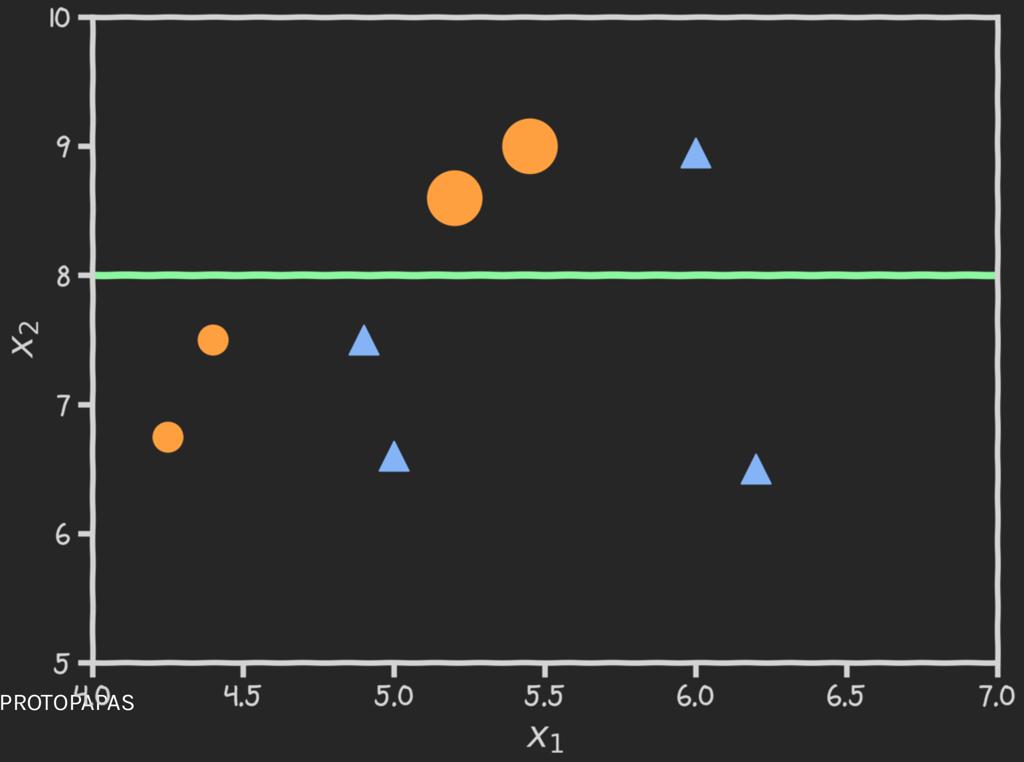
$$Error\ (\epsilon) = \frac{0.8}{8} + \frac{0.8}{8} + \frac{0.8}{8} = 0.3$$

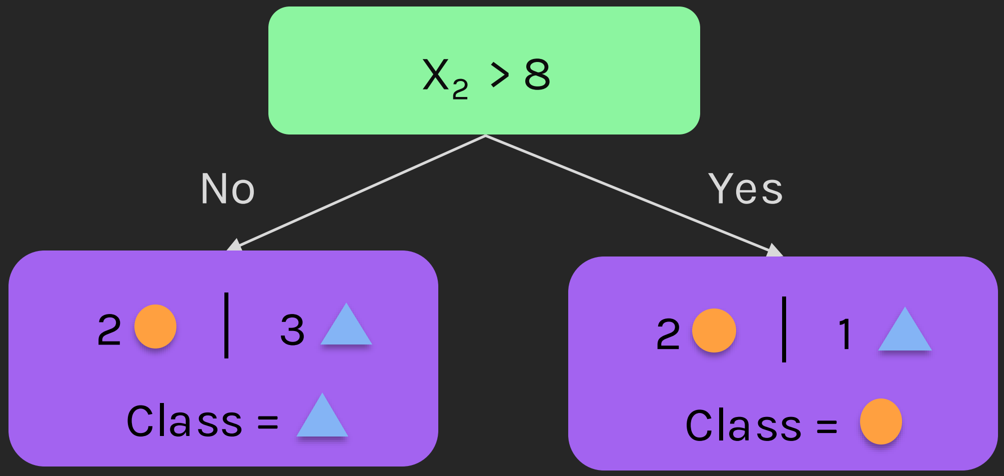
AdaBoost

Step 8: 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.

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



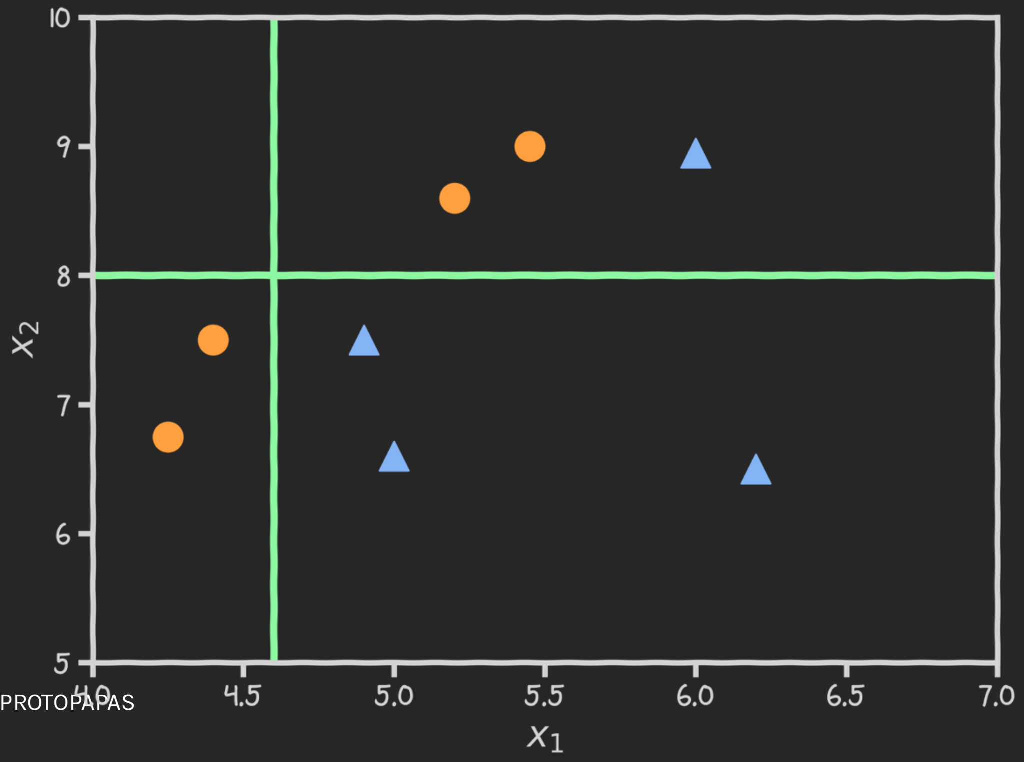


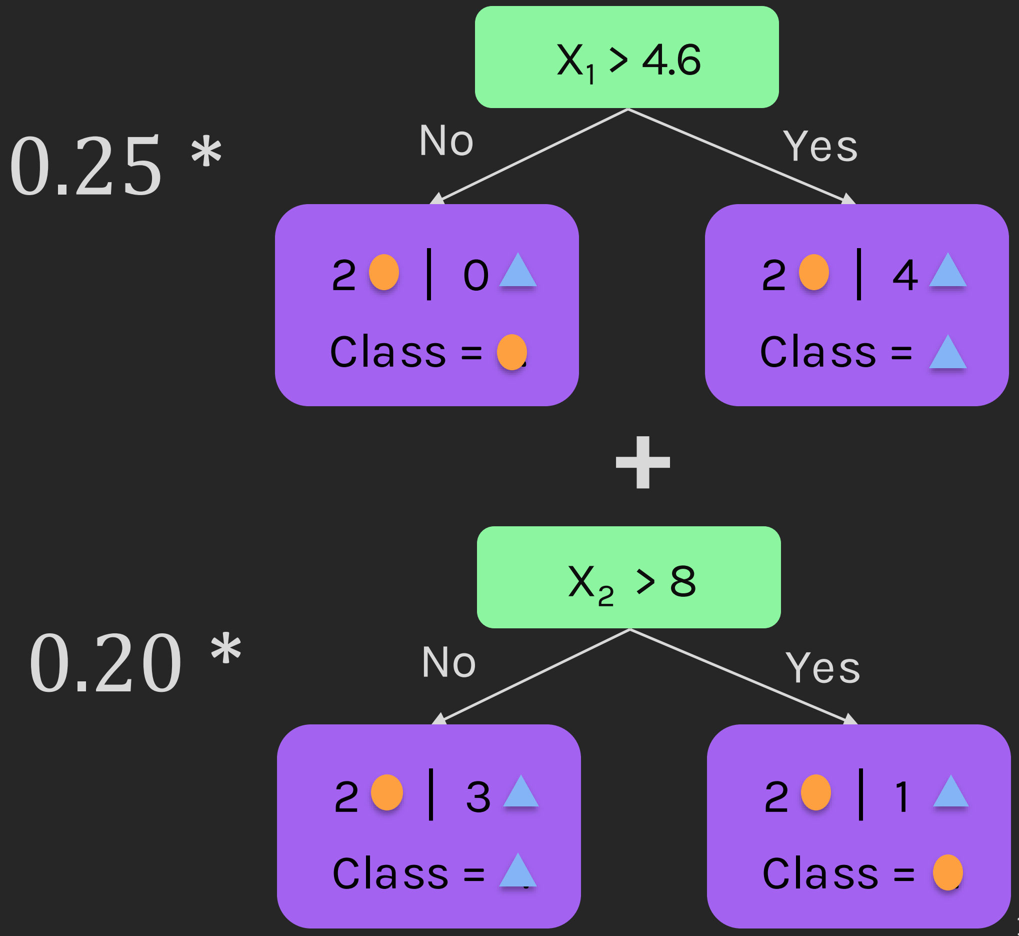
$$Error\ (\epsilon) = 0.3$$

AdaBoost

Step 9: Construct the **ensemble model** $T^{(1)}$ using:

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

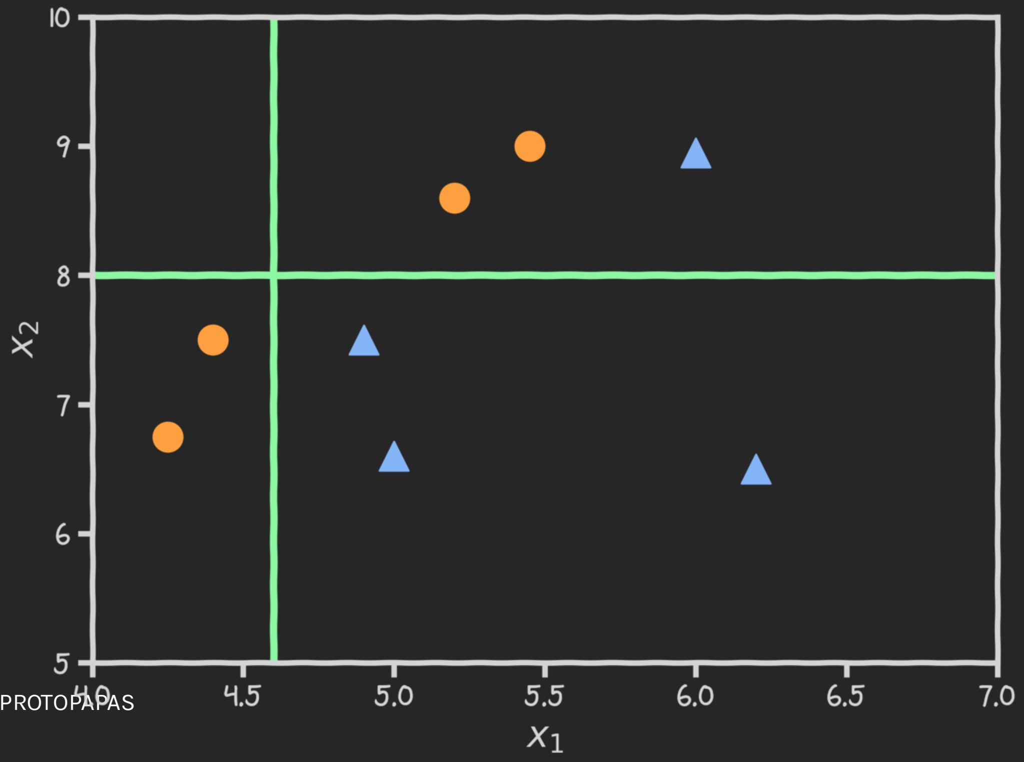


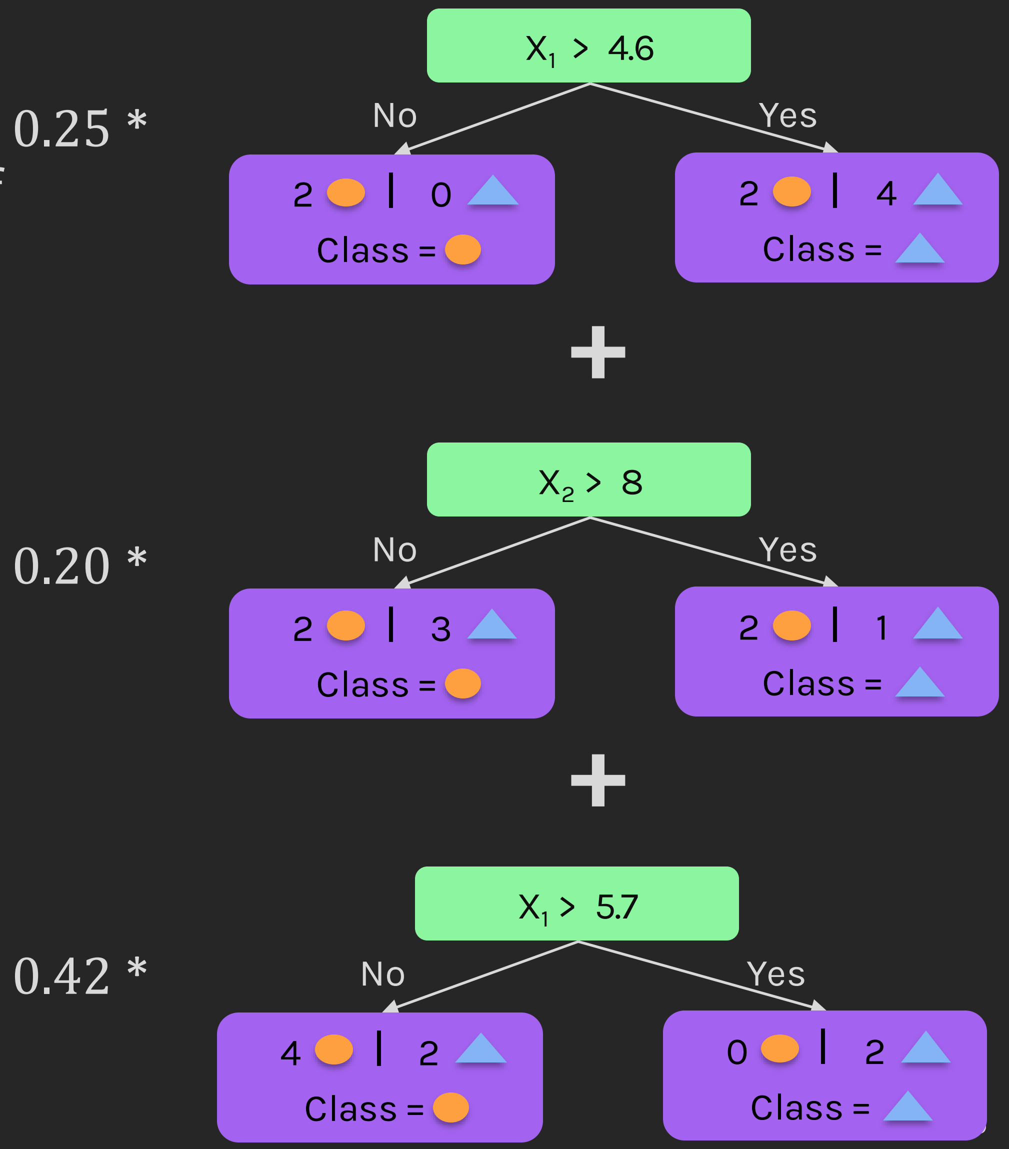


AdaBoost

Repeating the same process again, we get:

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





AdaBoost: SUMMARY

Given n, training data points, set w=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, $T^{(i)}$:
- Adjust the weights assigned to each data point to ensure the next stump focuses on the points misclassified by the previous stump

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:

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

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

$$w_n^{(i+1)} \leftarrow \frac{w_n^{(i)} \, e^{-\lambda^{(i)} \; y_n \, T^{(i)}(x_n)}}{Z}, \quad \text{where} \quad T^{(i)}(x) = sign\left[T^{(i-1)}(x) + \lambda^{(i)} S^{(i)}(x)\right]$$

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

AdaBoost

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

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

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

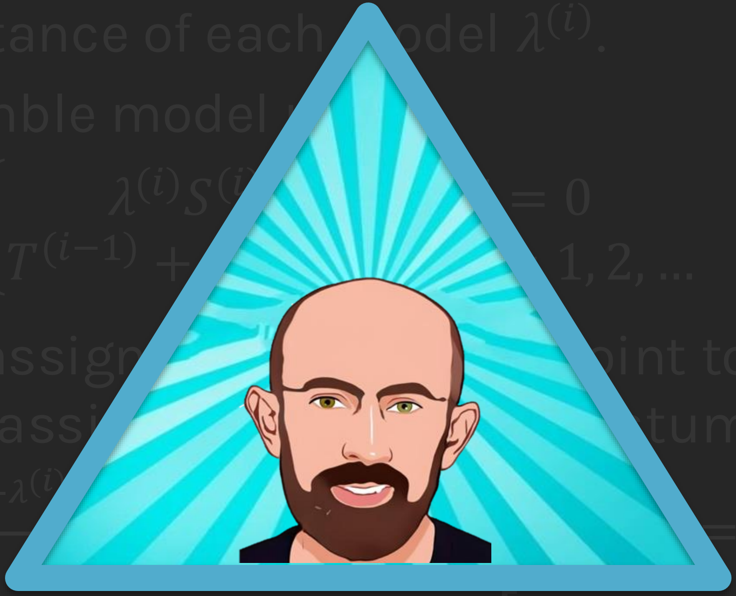
Step 3: Calculate the total error of the weak learner using:

GREAT PROFESSORS always KNOW your next questions!

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

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

Step 5: Construct the ensemble model



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

Step 6: Adjust the weights assign... oint to ensure the next stump focuses on the points misclassi... tump

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

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

AdaBoost

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

For $i = 0 \dots$ 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:

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

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

Step 5: Construct the ensemble model using:

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

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

$$w_n^{(i+1)} \leftarrow \frac{w_n^{(i)}\, e^{-\lambda^{(i)}\, y_n\, T^{(i)}(x_n)}}{Z}, \quad \text{where} \quad T^{(i)}(x) = sign\left[T^{(i-1)}(x) + \lambda^{(i)} S^{(i)}(x)\right]$$

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

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 it is an upper bound of 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:

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

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.

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

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

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

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

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

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

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**:

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

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:

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

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.

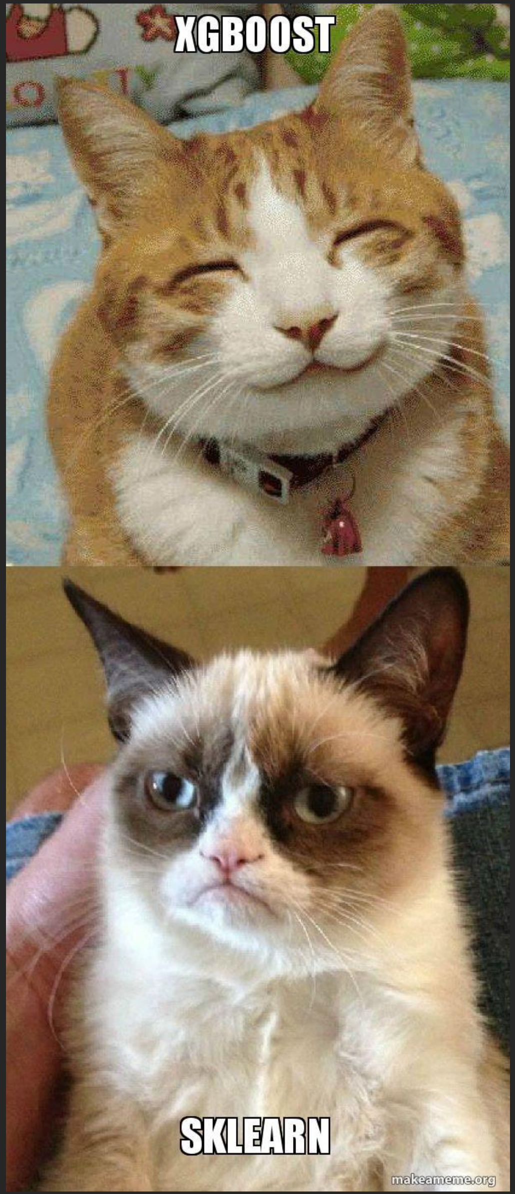
Final thoughts on Boosting

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.



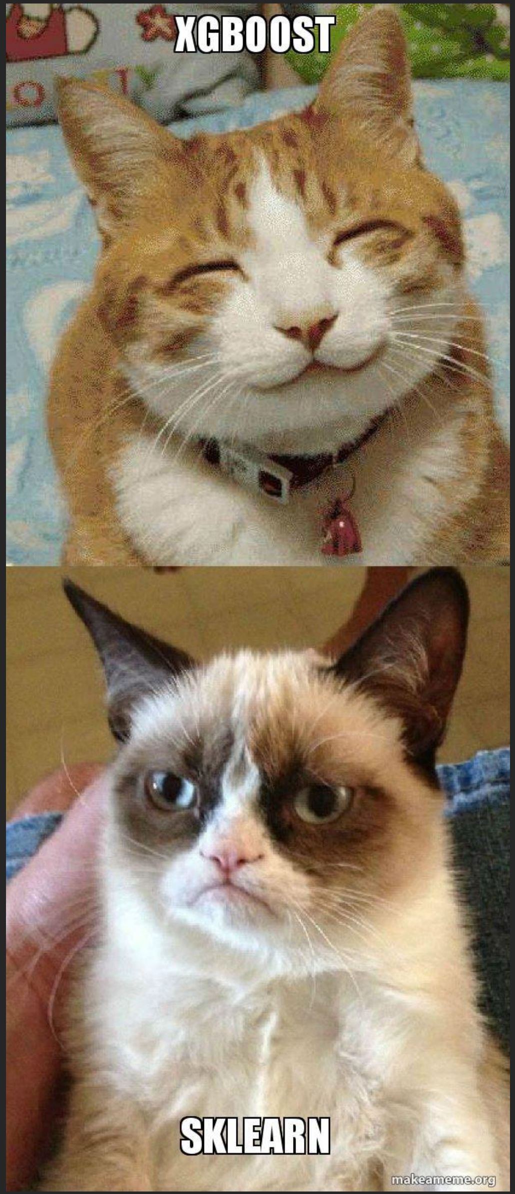
Everything you need to know about XGBoost (Extreme Gradient Boosting)!

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}`




The latest LLM
XGBoost

Thank you

Thank you