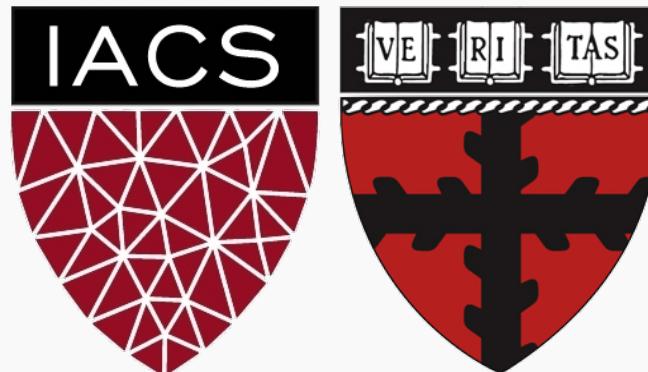


AdaBoost

CS109A Introduction to Data Science
Pavlos Protopapas, Kevin Rader and Chris Tanner





PPppBoost

Motivation for AdaBoost

Using the language of gradient descent also allows us to connect gradient boosting for regression to a boosting algorithm often used for **classification**, AdaBoost.

In classification, we typically want to minimize the classification error:

$$\text{Error} = \frac{1}{N} \sum_{n=1}^N \mathbb{1}(y_n \neq \hat{y}_n), \quad \mathbb{1}(y_n \neq \hat{y}_n) = \begin{cases} 0, & y_n = \hat{y}_n \\ 1, & y_n \neq \hat{y}_n \end{cases}$$

Naively, we can try to minimize Error via **gradient descent**, just like we did for MSE in gradient boosting.

Unfortunately, Error is not differentiable 😞

Motivation for AdaBoost (cont.)

Our solution: we replace the Error function with a differentiable function that is a good indicator of classification error.

The function we choose is called **exponential loss**:

$$\text{ExpLoss} = \frac{1}{N} \sum_{n=1}^N \exp(-y_n \hat{y}_n), \quad y_n \in \{-1, 1\}$$

Exponential loss is differentiable with respect to \hat{y}_n and it is an upper bound of Error.

Gradient Descent with Exponential Loss

We first compute the gradient for ExpLoss:

$$\nabla_{\hat{y}} \text{Exp} = [-y_1 \exp(-y_1 \hat{y}_1), \dots, -y_N \exp(-y_N \hat{y}_N)]$$

It's easier to decompose each $y_n \exp(-y_n \hat{y}_n)$ as $w_n y_n$, where

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

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

$$\nabla \text{Exp} = [-w_1 y_1, \dots, -w_N y_N]$$

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.



Gradient Descent with Exponential Loss

The update step in the gradient descent is

$$\hat{y}_n \leftarrow \hat{y}_n + \lambda \hat{r}_n(x_n), \quad n = 1, \dots, N$$

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

This means training $T^{(i)}$ 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.**



AdaBoost

With a minor adjustment to the exponential loss function, we have the algorithm for **gradient descent**:

1. Choose an initial distribution over the training data, $w_n = 1/N$.
2. At the i^{th} step, fit a simple classifier $T^{(i)}$ on weighted training data
$$\{(x_1, w_1 y_1), \dots, (x_N, w_N y_N)\}$$
3. Update the weights:

$$w_n \leftarrow \frac{w_n \exp(-\lambda^{(i)} y_n T^{(i)}(x_n))}{Z}$$

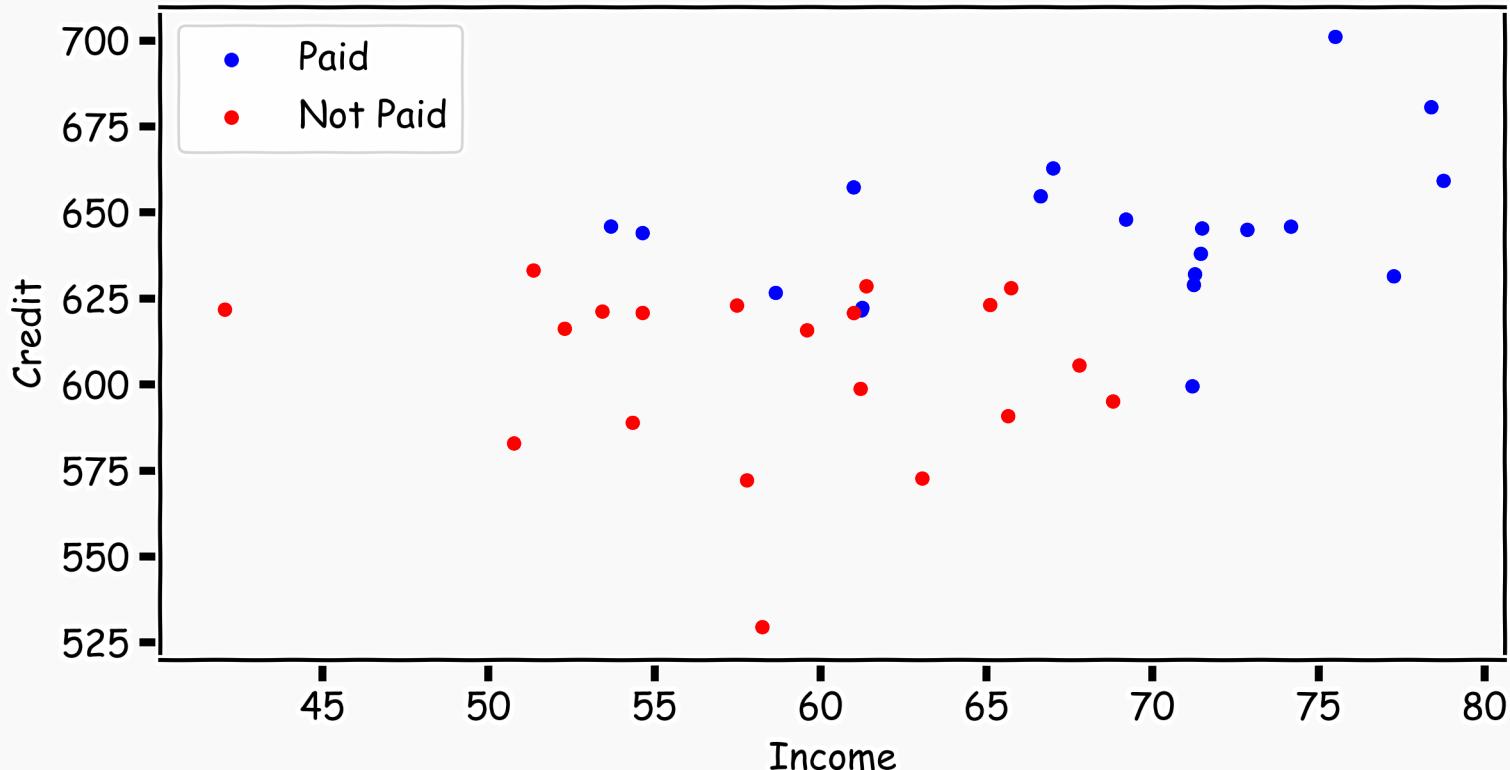
where Z is the normalizing constant for the collection of updated weights

4. Update $T: T \leftarrow T + \lambda^{(i)} T^{(i)}$

where λ is the learning rate.

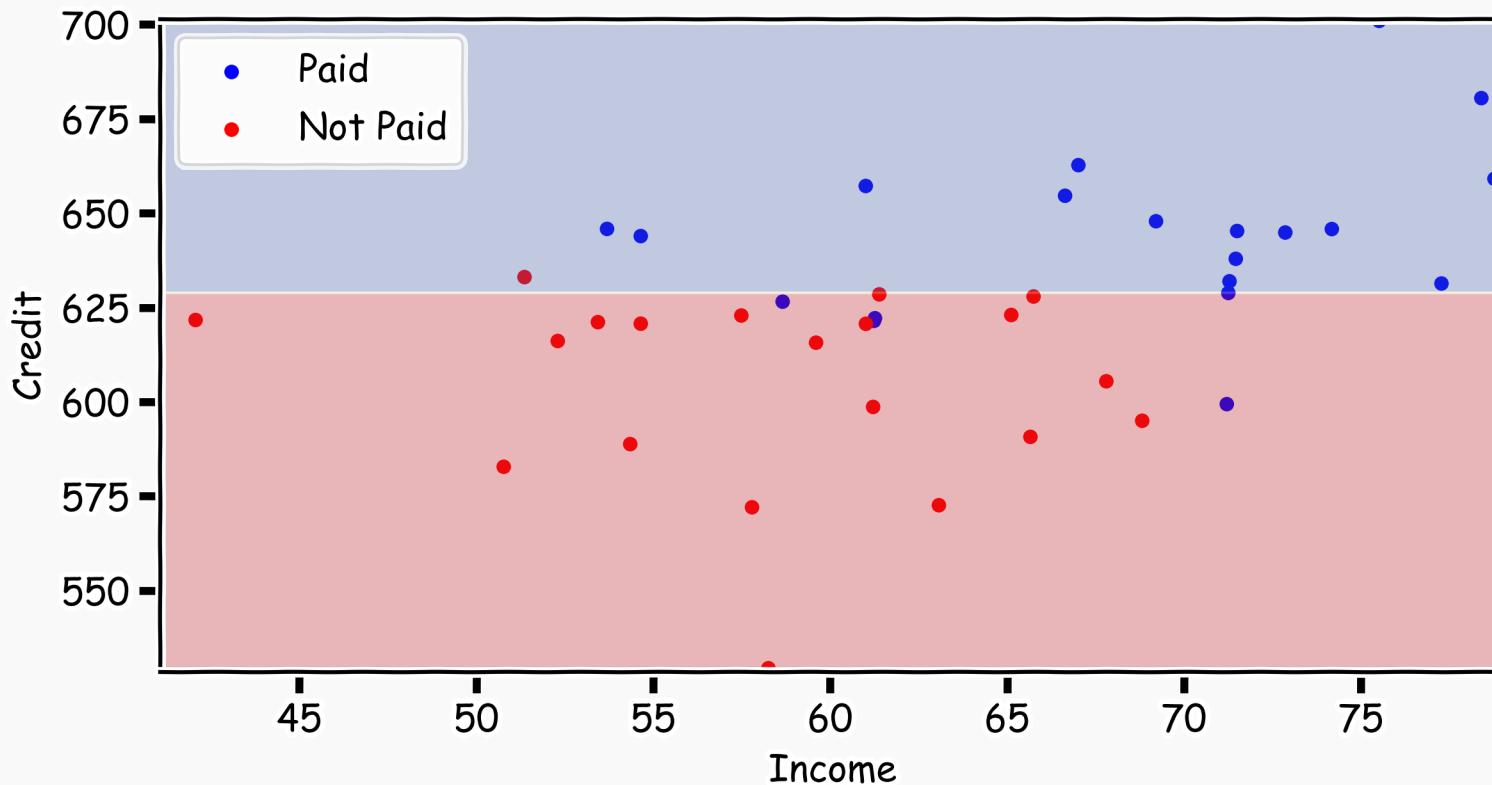


AdaBoost: start with equal weights



AdaBoost:

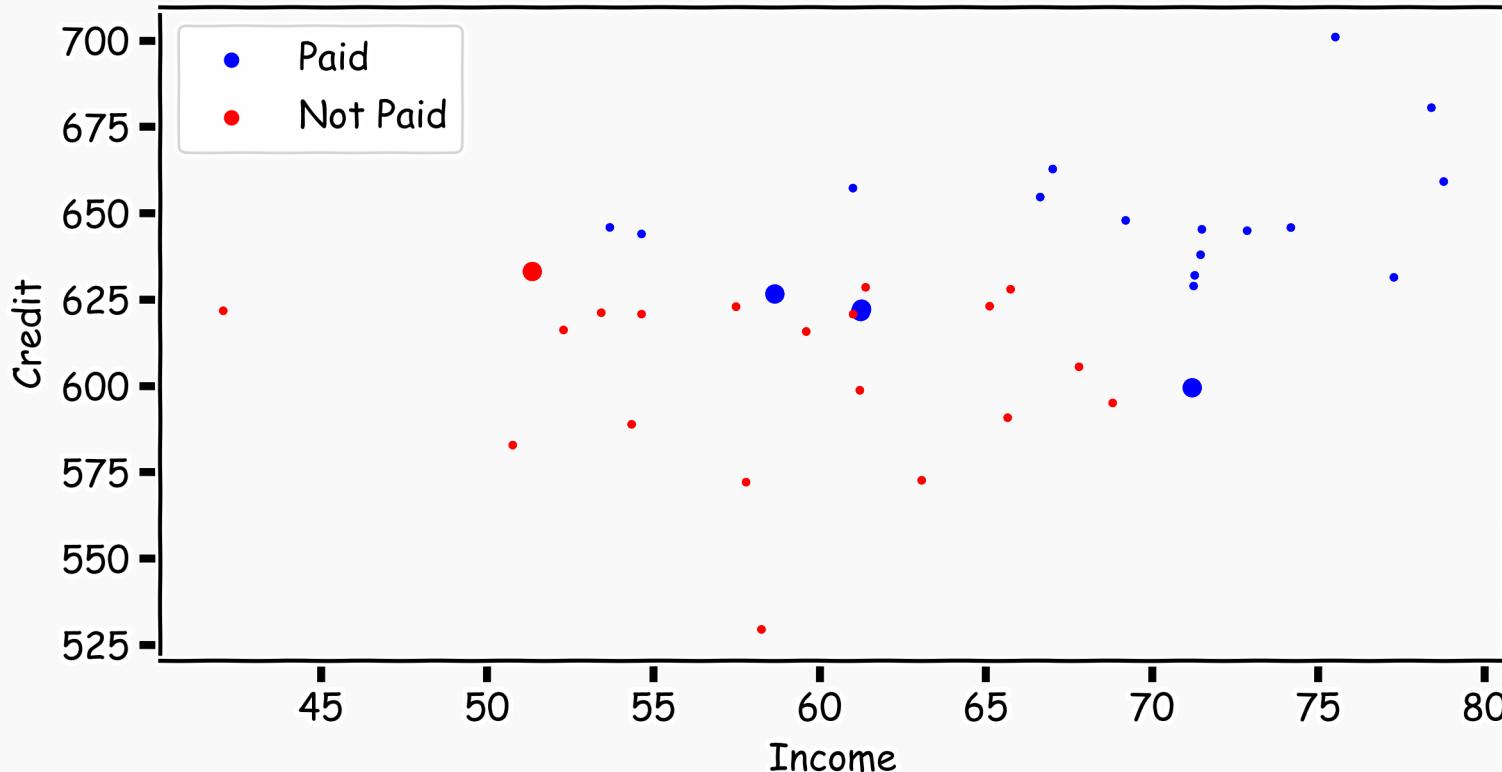
fit a simple classifier $T^{(i)}$



AdaBoost:

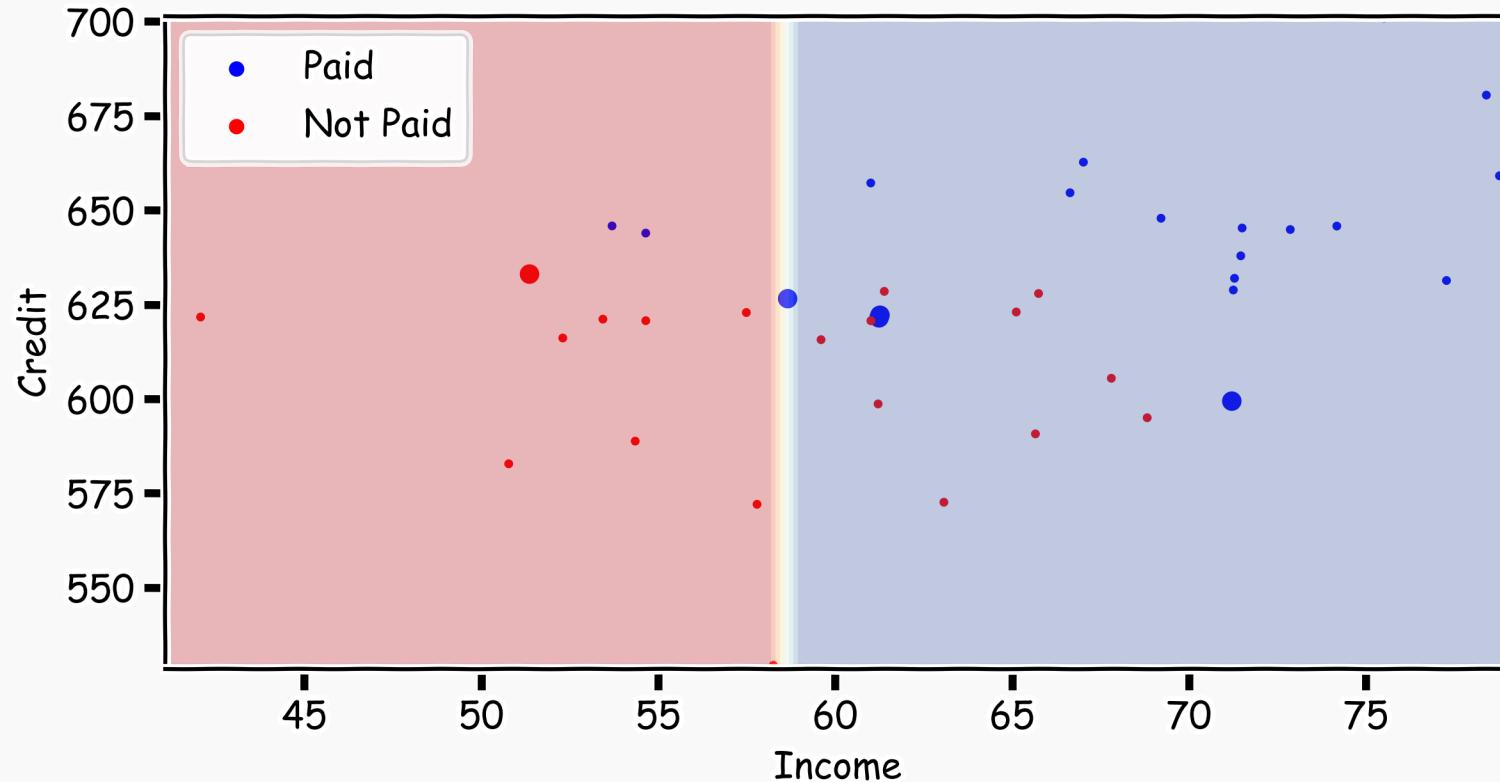
Update the weights:

$$w_n \leftarrow \frac{w_n \exp(-\lambda^{(i)} y_n T^{(i)}(x_n))}{Z}$$



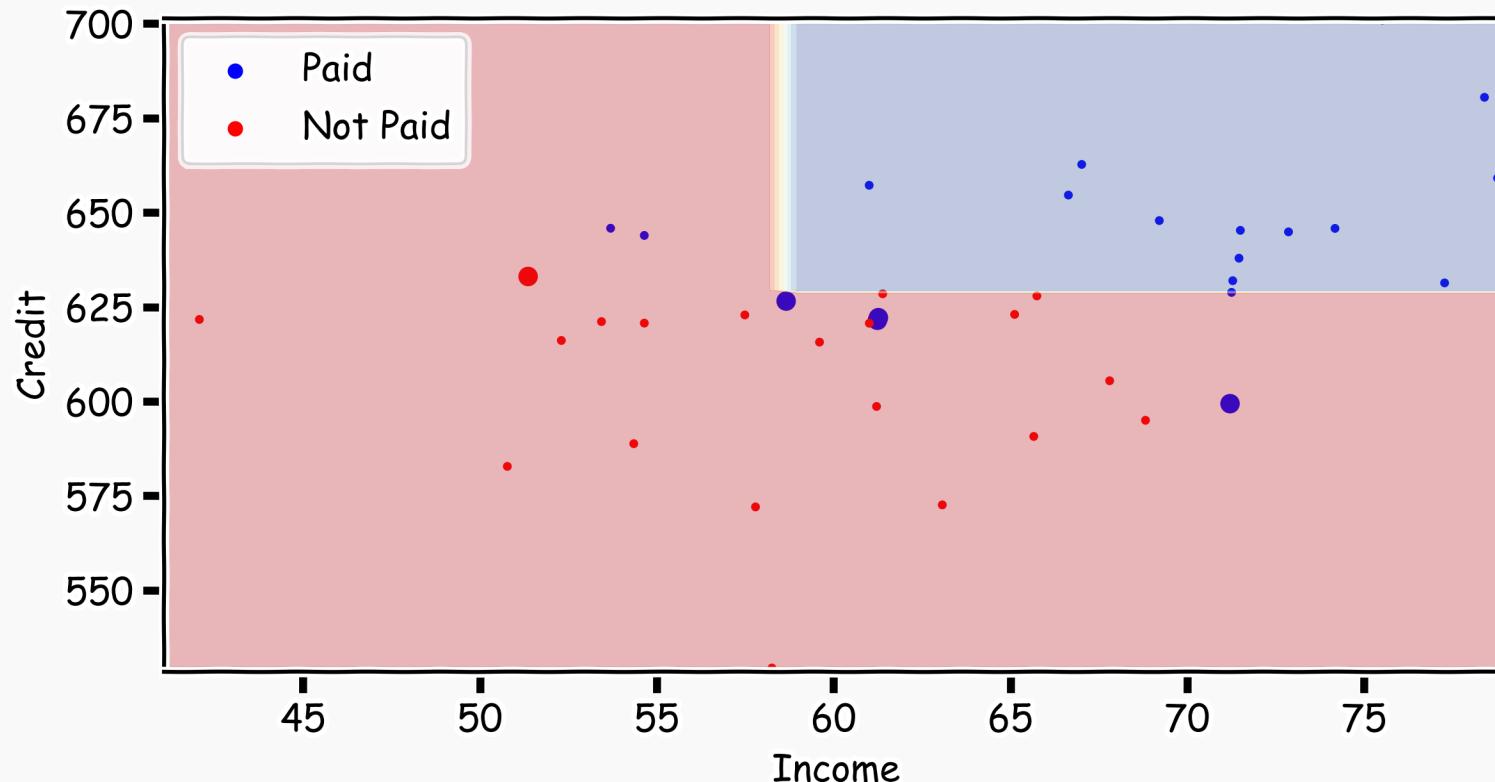
AdaBoost:

fit another simple decision tree on re-weighted data



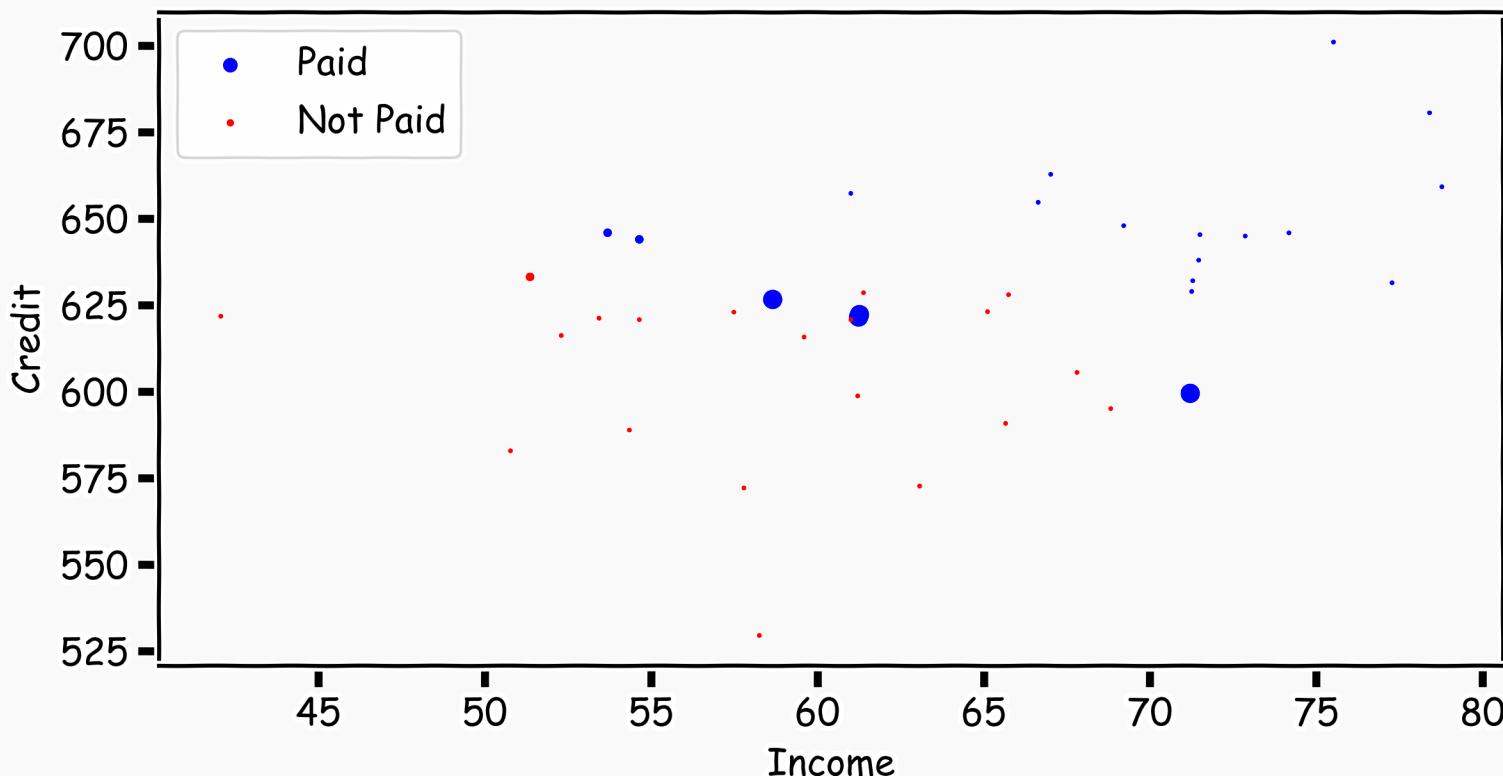
AdaBoost:

add the new model to the ensemble: $T \leftarrow T + \lambda^{(i)}T^{(i)}$



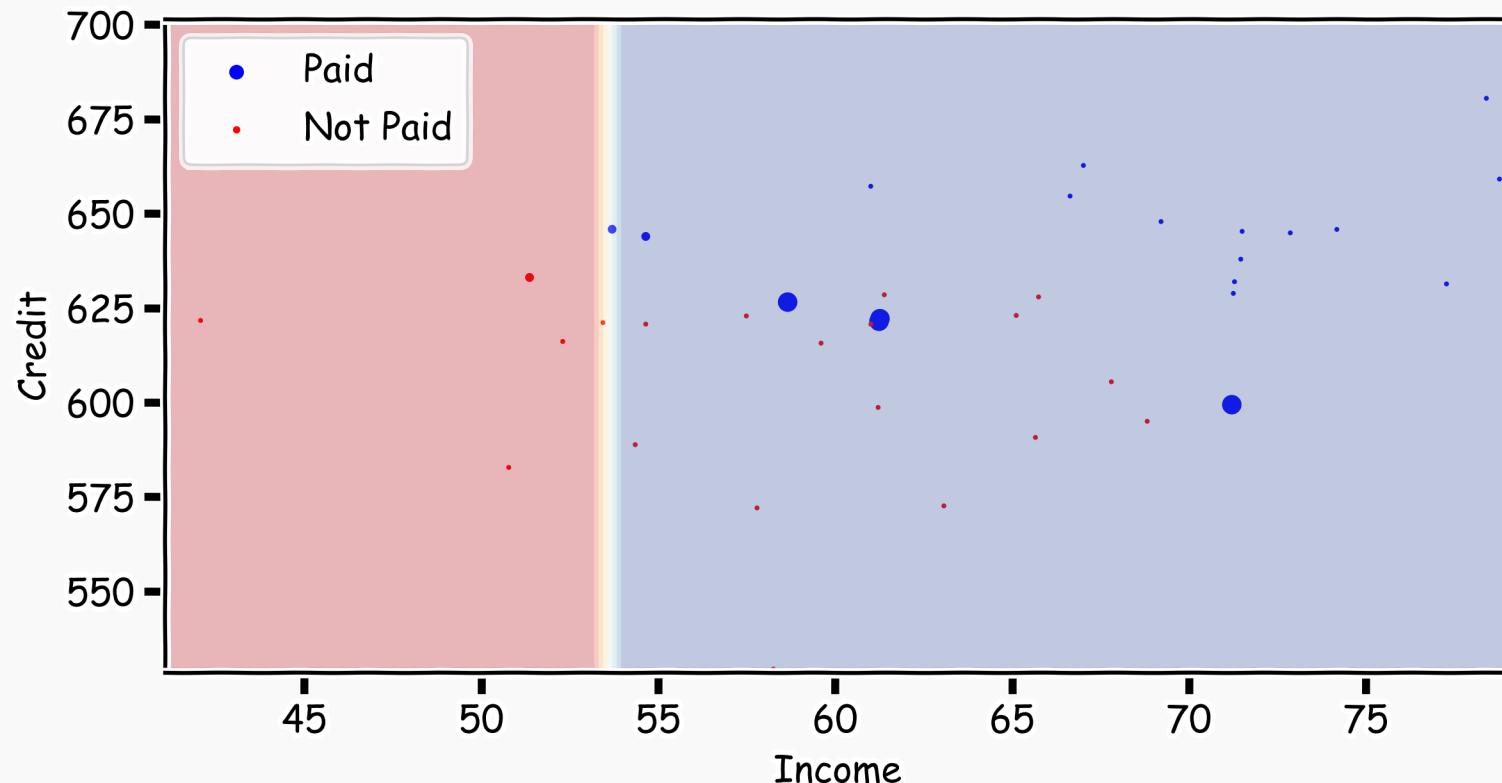
AdaBoost:

Update the weights: $w_n \leftarrow \frac{w_n \exp(-\lambda^{(i)} y_n T^{(i)}(x_n))}{Z}$



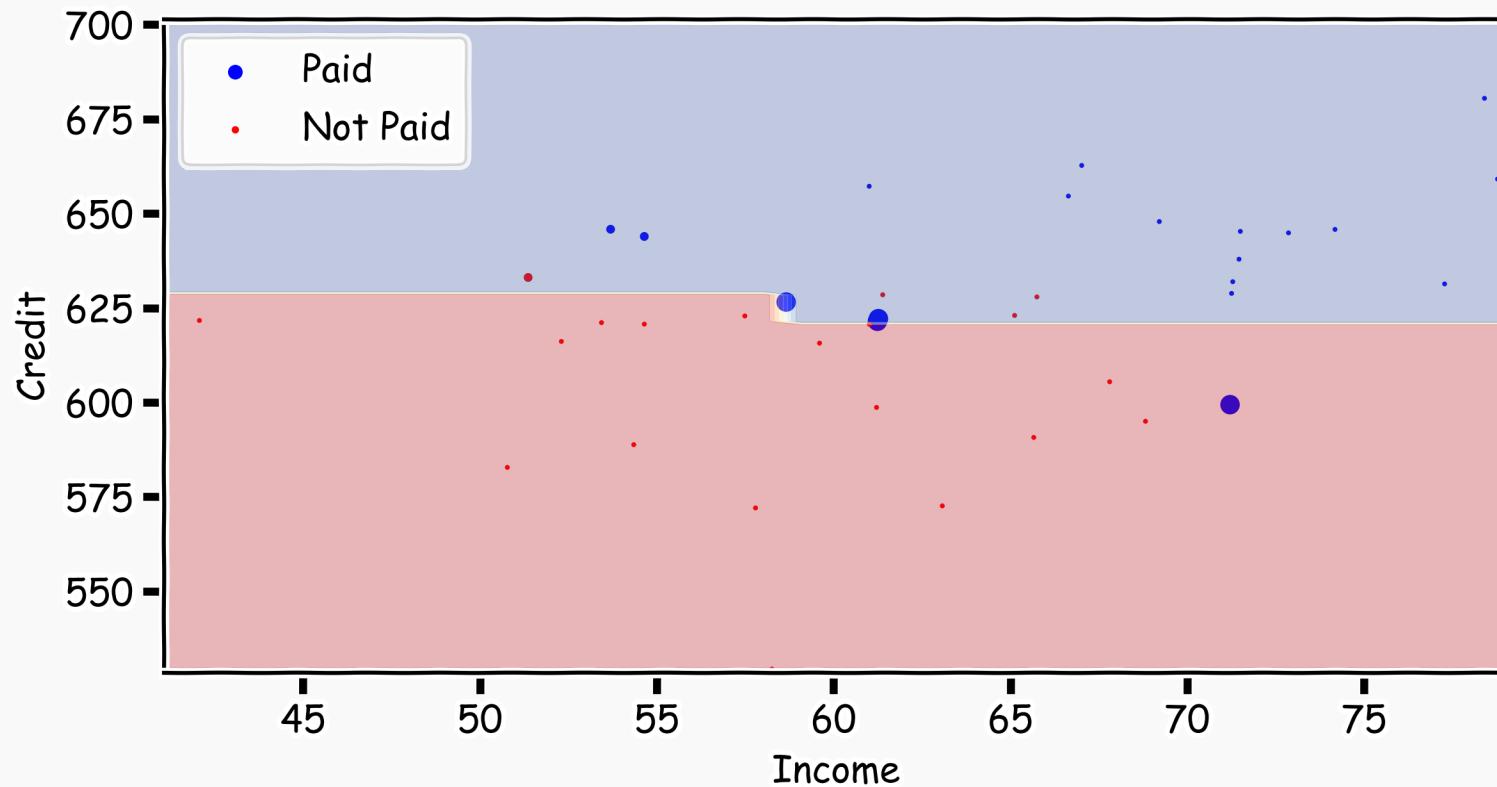
AdaBoost

fit another simple decision tree on re-weighted data



AdaBoost:

add the new model to the ensemble: $T \leftarrow T + \lambda^{(i)}T^{(i)}$



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:

$$\operatorname{argmin}_{\lambda} \frac{1}{N} \sum_{n=1}^N \exp [-y_n(T + \lambda^{(i)} T^{(i)}(x_n))]$$

Doing so, we get that

$$\lambda^{(i)} = \frac{1}{2} \ln \frac{1 - \epsilon}{\epsilon}, \quad \epsilon = \sum_{n=1}^N w_n \mathbb{1}(y_n \neq T^{(i)}(x_n))$$

Final thoughts on Boosting

There are few implementations on boosting:

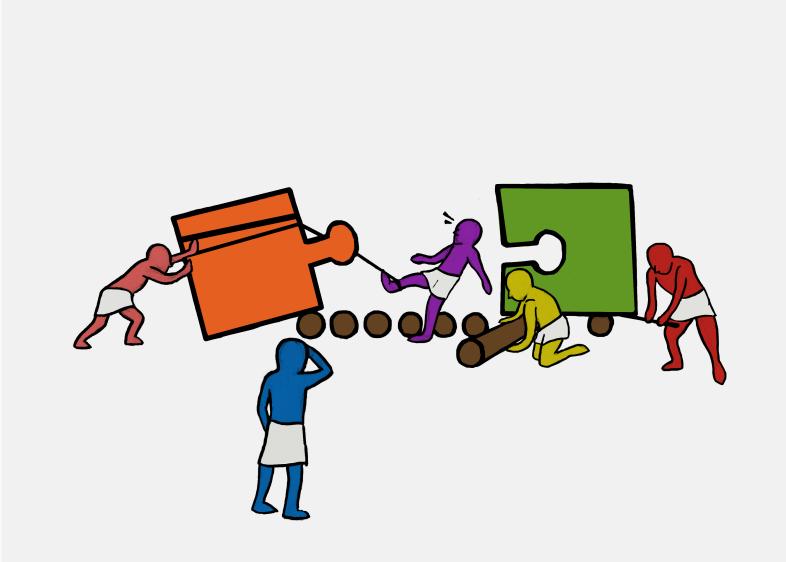
- [XGBoost](#): An efficient Gradient Boosting Decision
- [LGBM](#): Light Gradient Boosted Machines. It is a library for training GBMs developed by Microsoft, and it competes with XGBoost
- [CatBoost](#): A new library for Gradient Boosting Decision Trees, offering appropriate handling of categorical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

Final thoughts on Boosting

Increasing the number of **trees** can lead to overfitting.

Question: Why?





Today's lucky student: PAVLOS
AdaBoost in action

