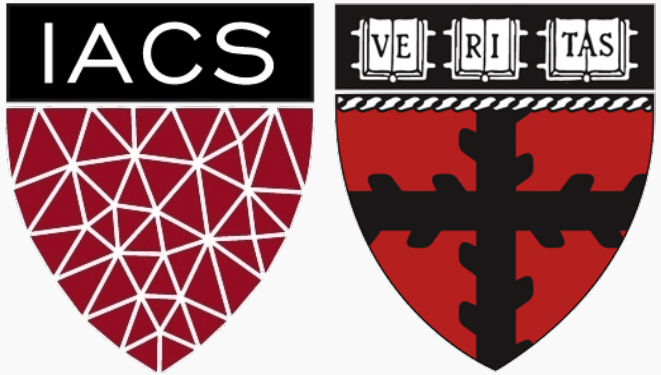
AdaBoost

CS109A Introduction to Data Science

Pavlos Protopapas, Natesh Pillai



- Do we recombine our residual predictions at the end of the boosting process to come up with a final model prediction or do we iteratively combine residual predictions to determine the next set of residuals?
- When would gradient boosting not be suitable for an application?
- Can you combine bootstrapping and boosting?
- How do we choose the appropriate lambda parameters?
- Why does boosting face the issue of overfitting when tree number increases while on the other hand random forest doesn't face this issue?
- In boosting, how can we make sure that the different trees are not highly correlated? Is there a mechanism similar to random forest?
- In boosting, if one has extreme outliers, can it ruin the boosting model since it is trained on residuals?

Nobody:

Random forest ensembles:

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:***

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

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:

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

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

$$\nabla_{\hat{y}}\,\text{Exp} = \left[-w_1 y_1, \ldots, -w_N y_N\right]$$

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

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

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

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

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

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

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

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

3. Update the weights:

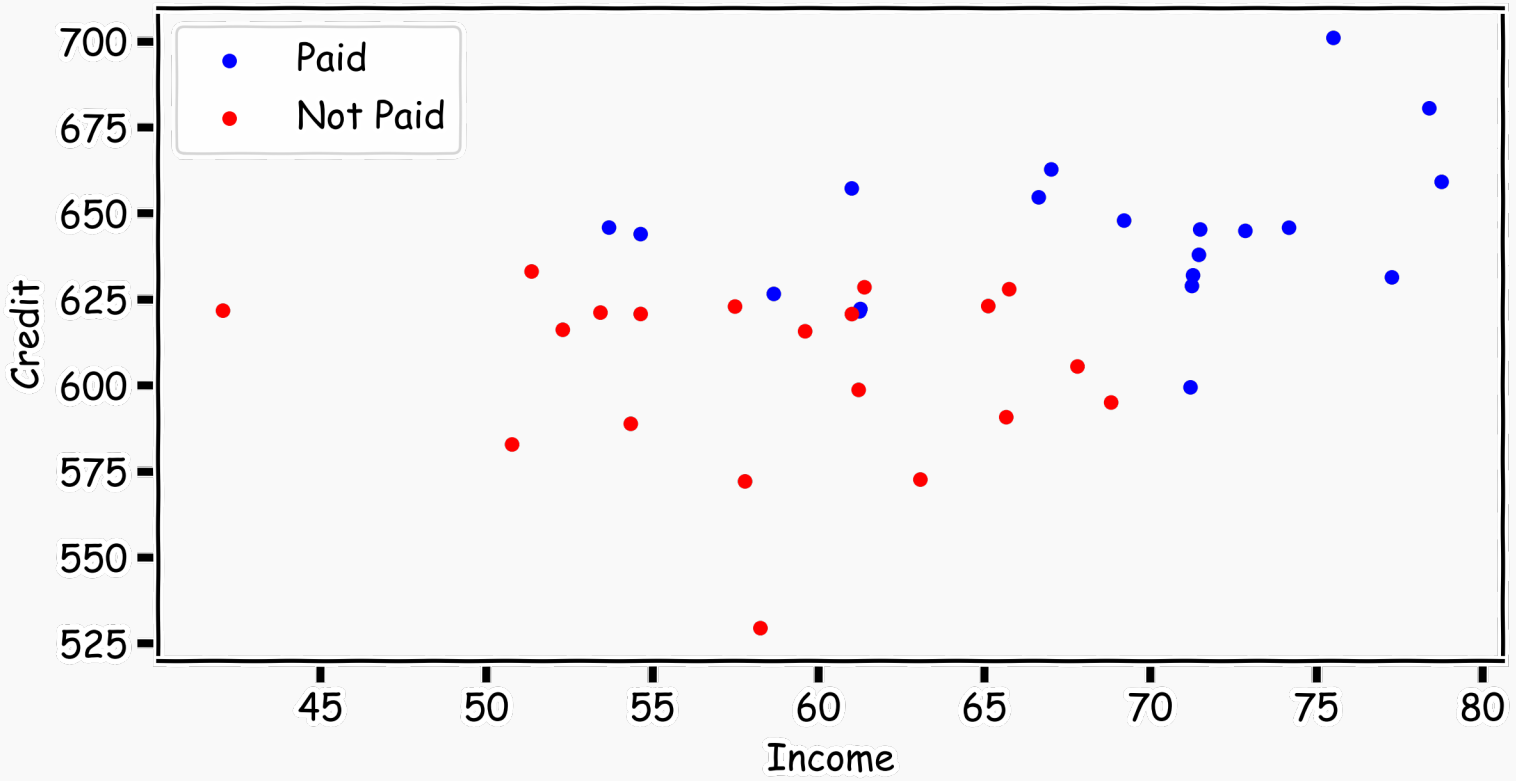
$$w_n \leftarrow \frac{w_n \exp(-\lambda^{(i)} y_n T^{(i)}(x_n))}{Z}$$

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

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

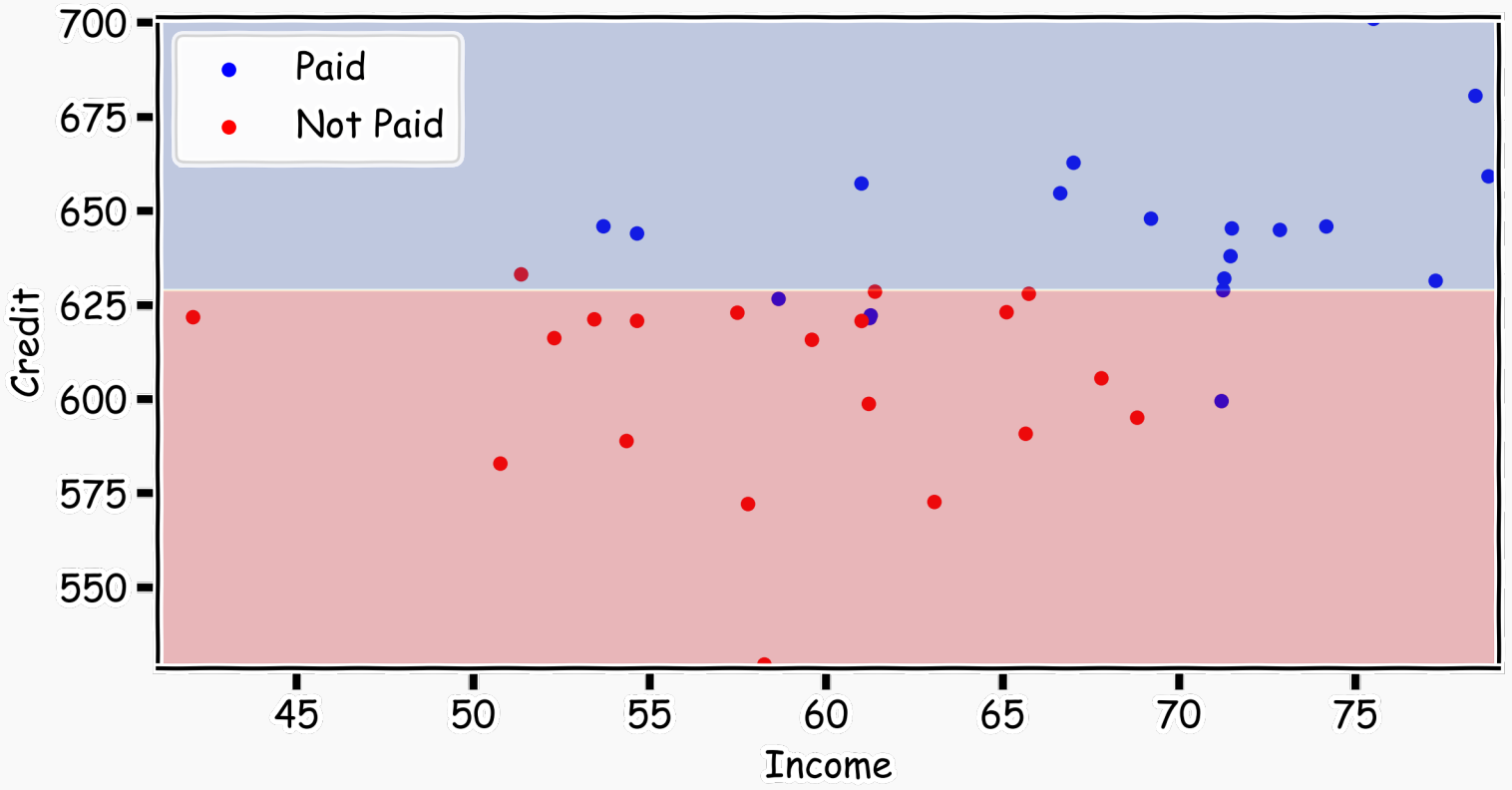
where λ is the learning rate.

AdaBoost: start with equal weights



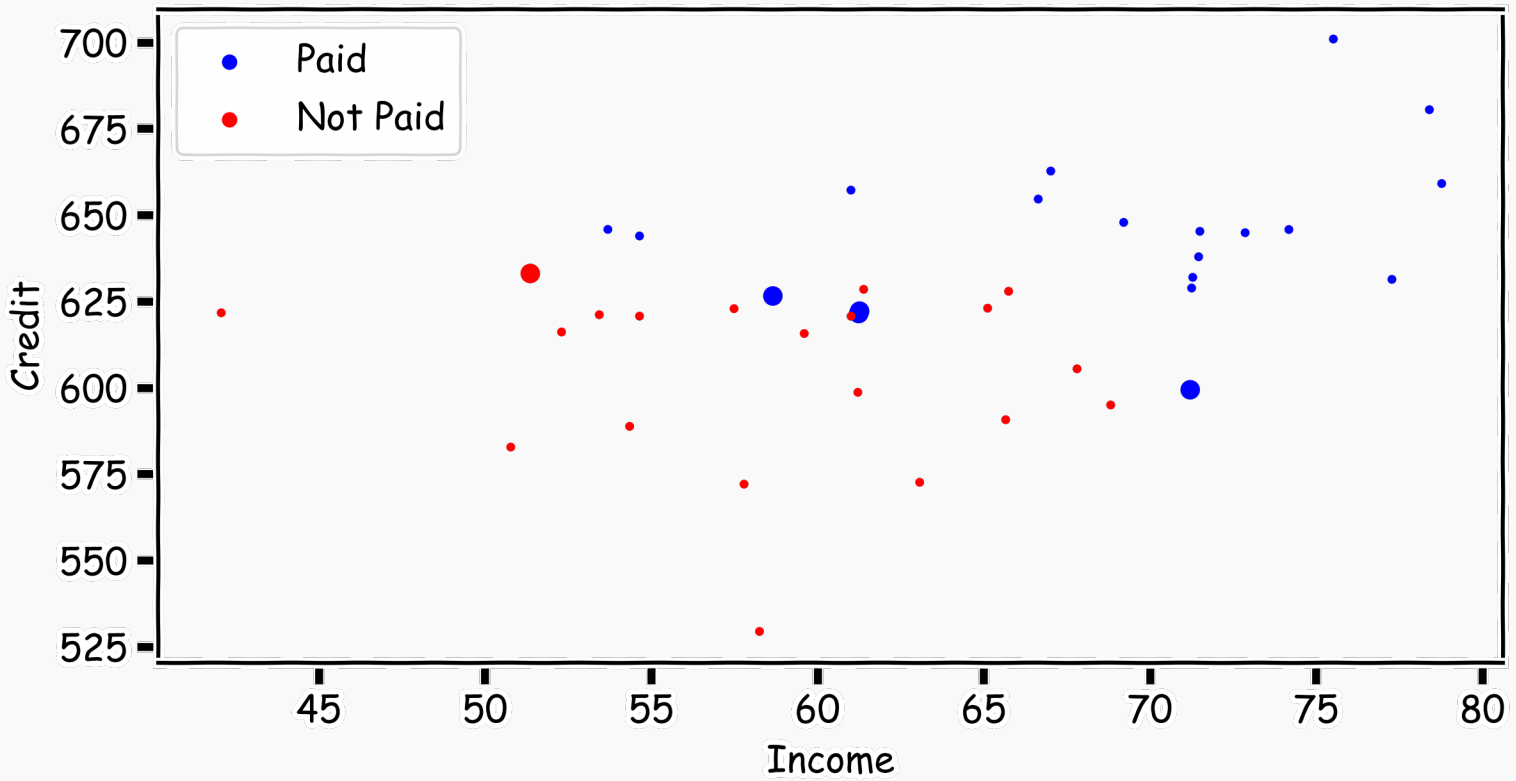
AdaBoost: fit a simple decision tree

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



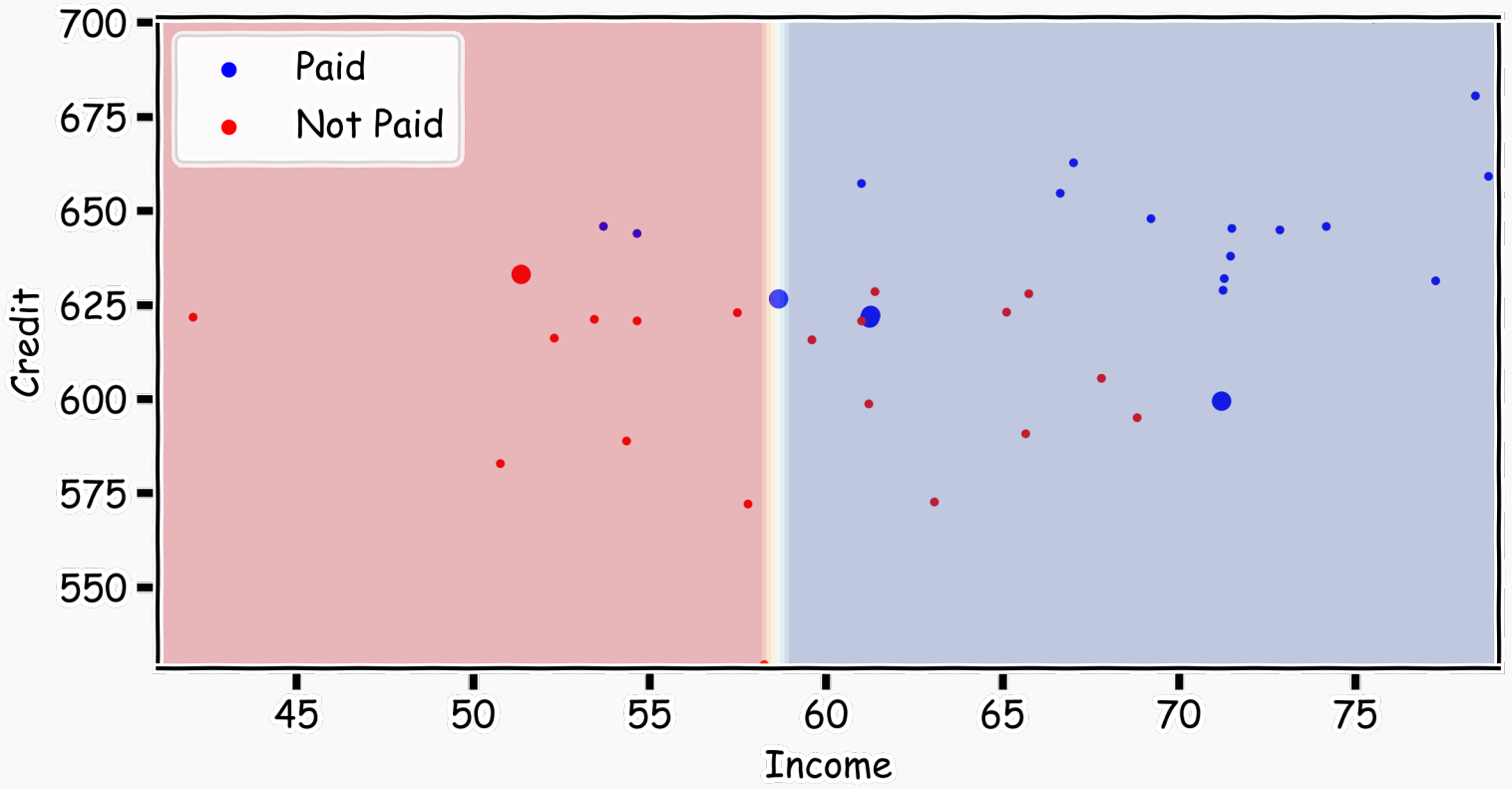
AdaBoost: update the weights

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



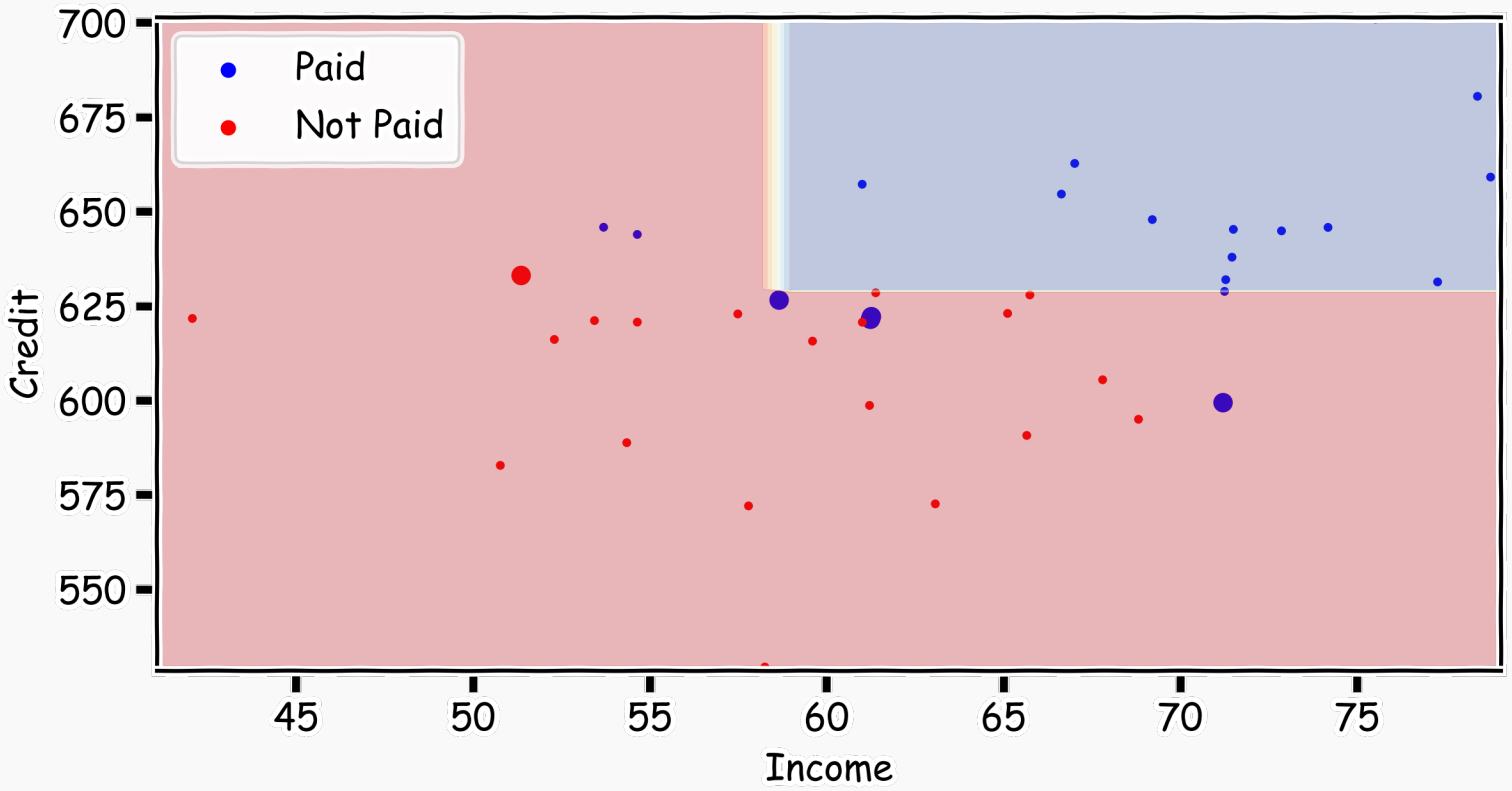
AdaBoost:

fit another simple decision tree on re-weighted data



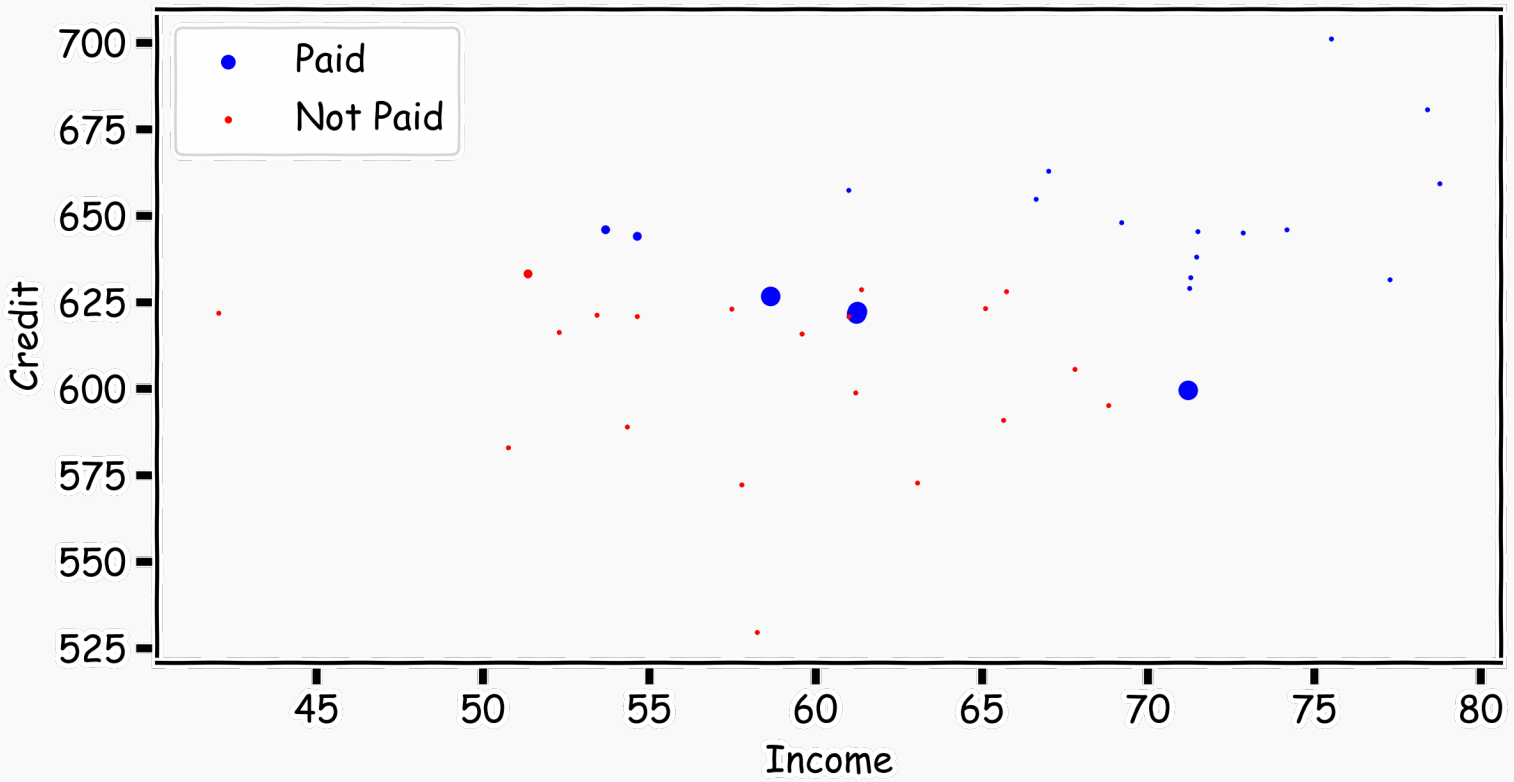
AdaBoost:

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



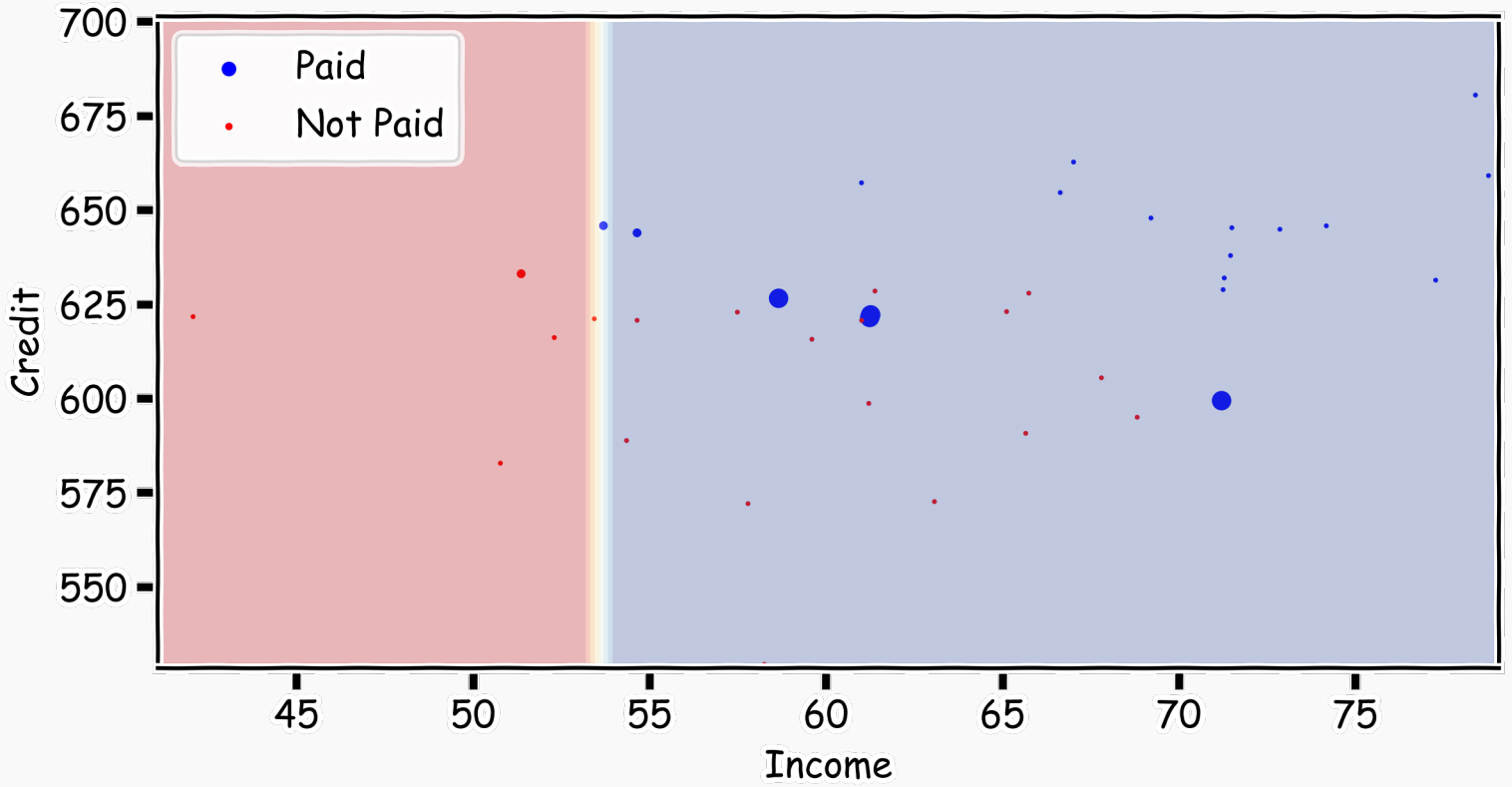
AdaBoost: update the weights

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



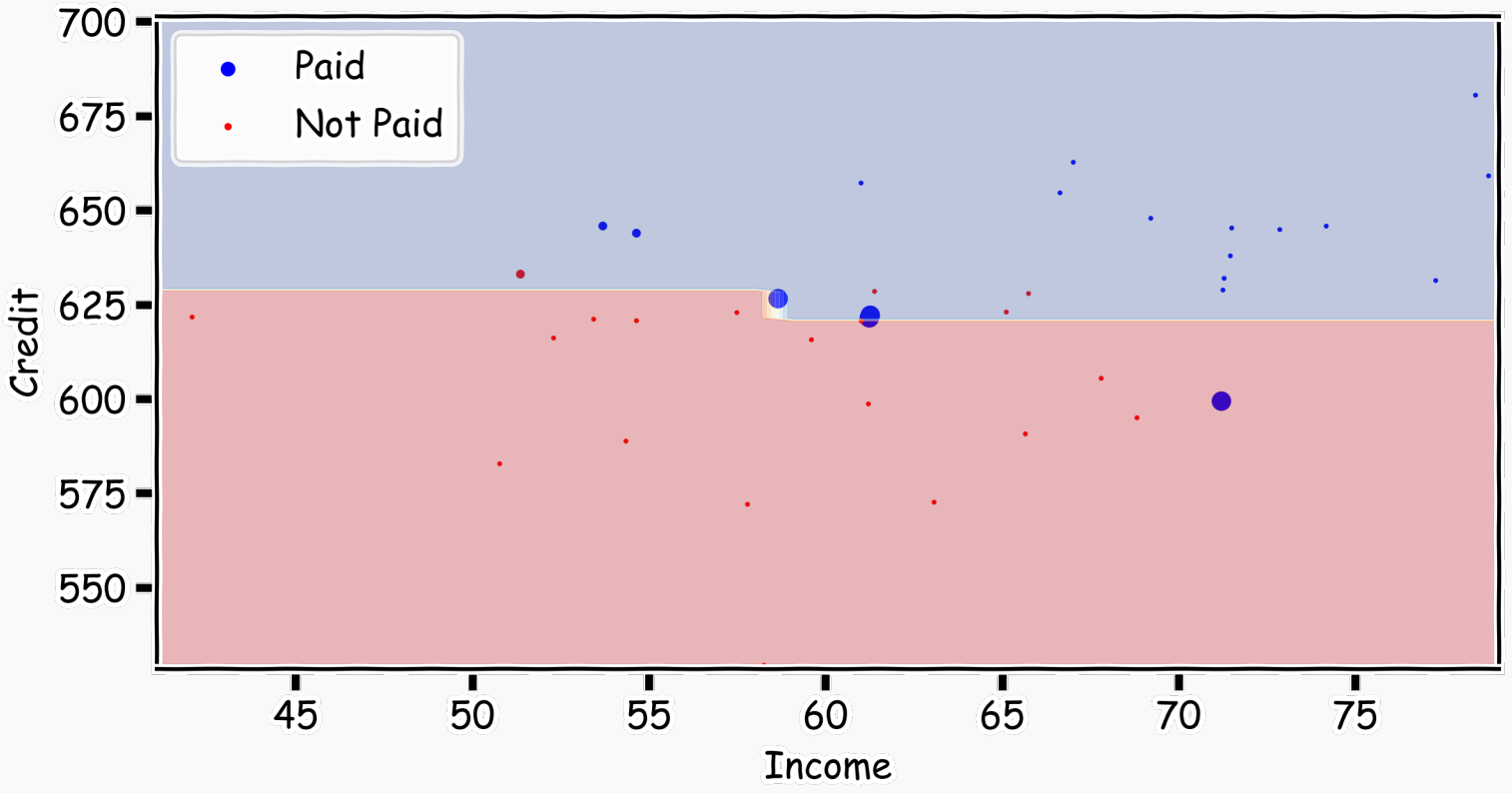
AdaBoost

fit another simple decision tree on re-weighted data



AdaBoost: add the new model to the ensemble, repeat...

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:

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

Doing so, we get that

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

Final thoughts on Boosting

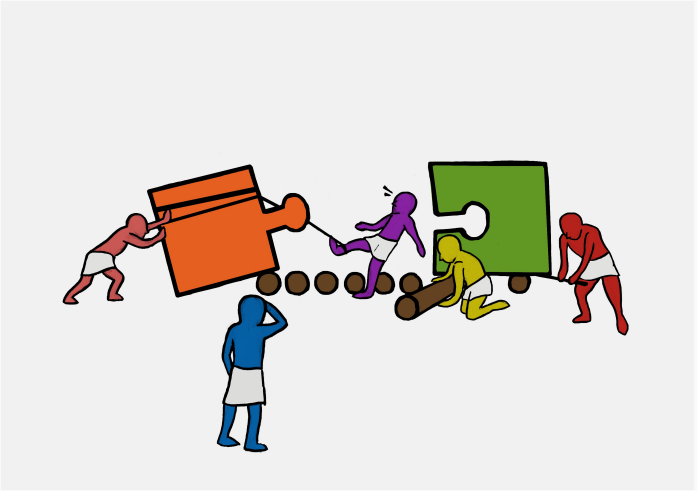
There are few implementations on boosting:

- 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

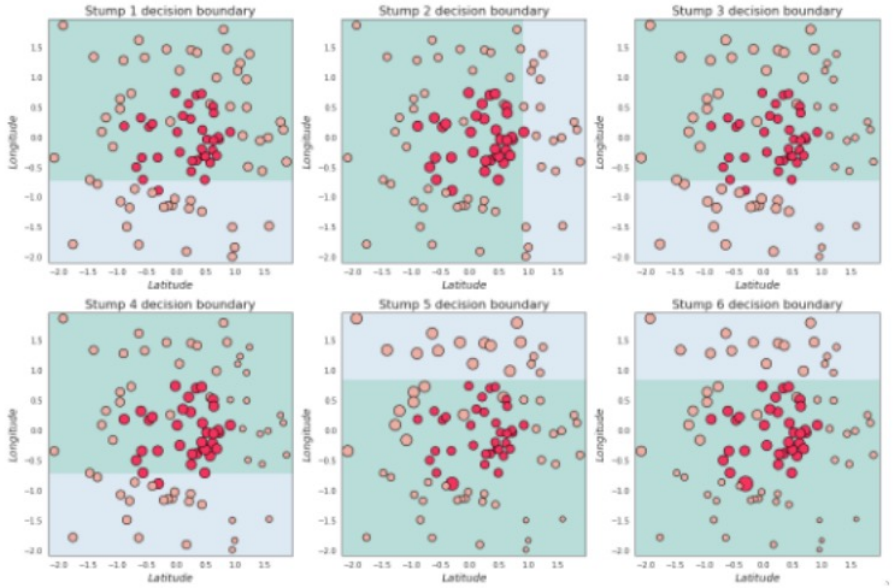
Increasing the number of ***trees*** can lead to overfitting.

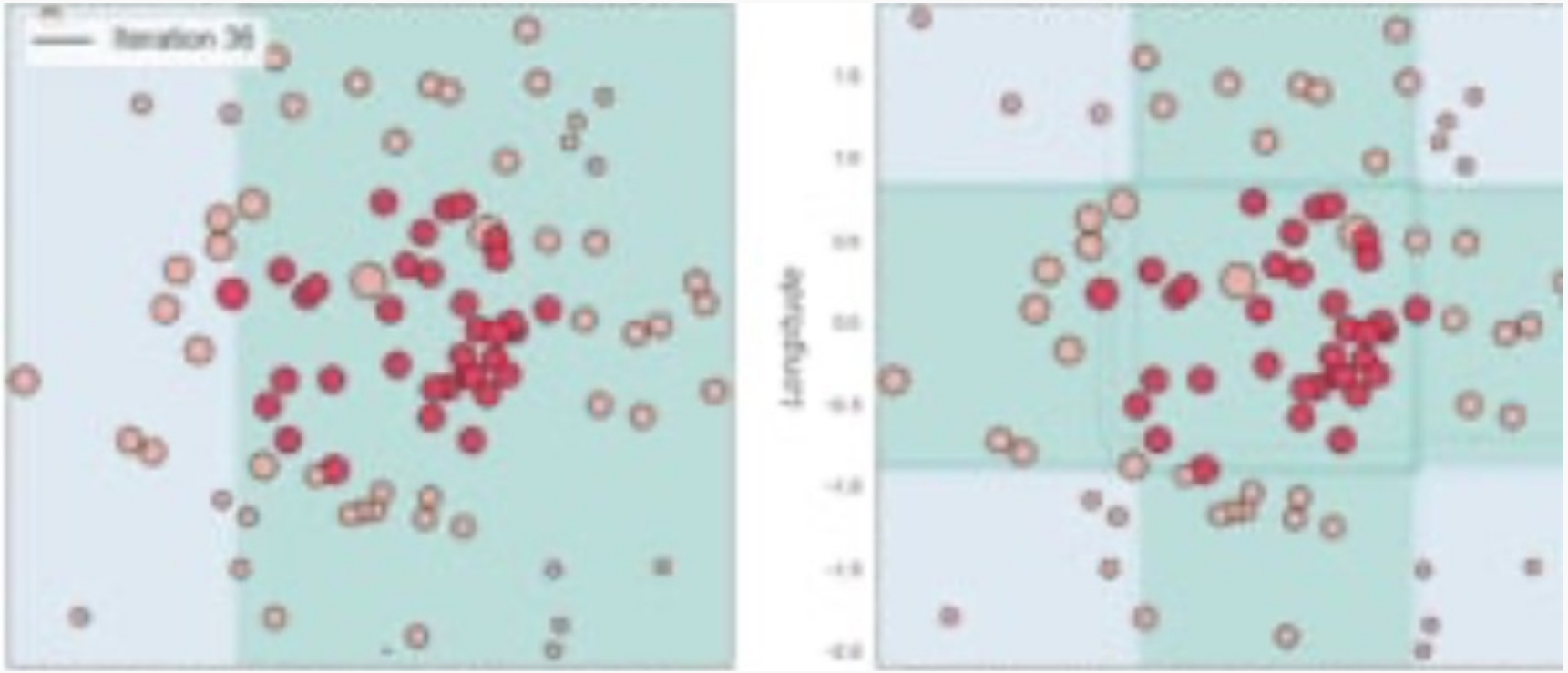
Question: Why?



🏋️‍♀️ 👨‍🏫 Exercise: Boosting Classification

The aim of this exercise to understand classification using boosting by plotting the decision boundary after each stump. Your plot may resemble the image below:




Iteration 36