INT 303 BIG DATA ANALYTICS

Lecture12: Boosting

Jia WANG

Jia.wang02@xjtlu.edu.cn

Xi'an Jiaotong-Liverpool University

西交利物浦大學

OUTLINE

- Bias/Variance Tradeoff
- Ensemble methods that minimize variance
  - Bagging
  - Random Forests
- **Ensemble methods that minimize bias**
  - **Functional Gradient Descent**
  - **Boosting**
  - **Adboost**

BAGS AND FORESTS OF TREES

- Last time we examined how the short-comings of single decision tree models can be overcome by ensemble methods - making one model out of many trees.

- We focused on the problem of training large trees, these models have low bias but high variance.

- We compensated by training an ensemble of full decision trees and then averaging their predictions - thereby reducing the variance of our final model.

Boosting

MOTIVATION FOR BOOSTING

- **Question:** Could we address the shortcomings of single decision trees models in some other way?

- For example, rather than performing variance reduction on complex trees, can we decrease the bias of simple trees - make them more expressive?

- A solution to this problem, making an expressive model from simple trees, is another class of ensemble methods called ***boosting***.

Boosting Algorithms

GRADIENT BOOSTING

- The key intuition behind boosting is that one can take an ensemble of simple models $\{T_h\}_{h \in H}$ and additively combine them into a single, more complex model.

$$T = \sum_h \lambda_h T_H$$

- Each model T_h might be a poor fit for the data, but a linear combination of the ensemble can be expressive/flexible.

- **Question:** But which models should we include in our ensemble? What should the coefficients or weights in the linear combination be?

GRADIENT BOOSTING: THE ALGORITHM

Gradient boosting is a method for iteratively building a complex regression model T by adding simple models. Each new simple model added to the ensemble compensates for the weaknesses of the current ensemble.

1. Fit a simple model $T^{(0)}$ on the training data

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

Set $T \leftarrow T^{(0)}$. Compute the residuals $\{r_1, \ldots, r_N\}$ for T.

2. Fit a simple model, $T^{(1)}$, to the current ***residuals***, i.e. train using

$$\{(x_1, r_1), \ldots, (x_N, r_N)\}$$

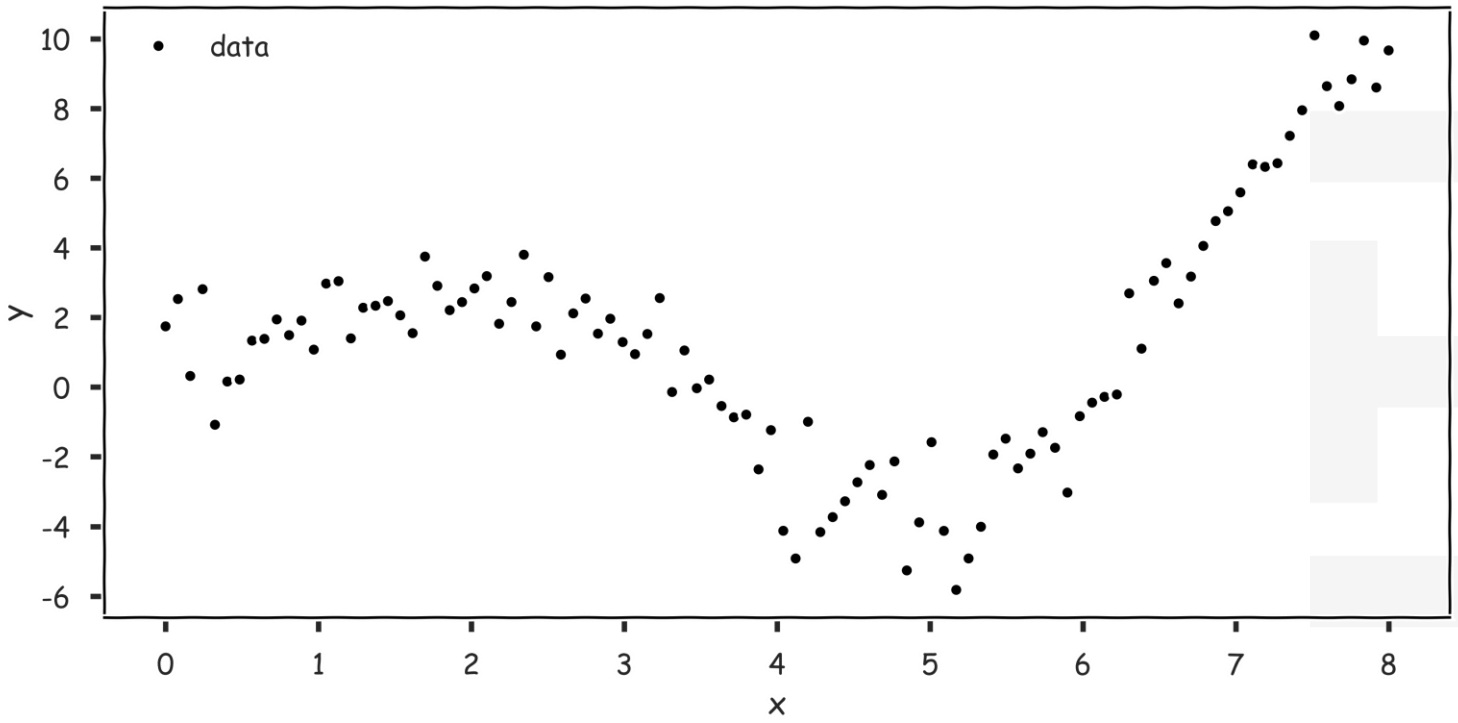
3. Set $T \leftarrow T + \lambda T^{(1)}$

4. Compute residuals, set $r_n \leftarrow r_n - \lambda T^i(x_n), \ n = 1, \ldots, N$

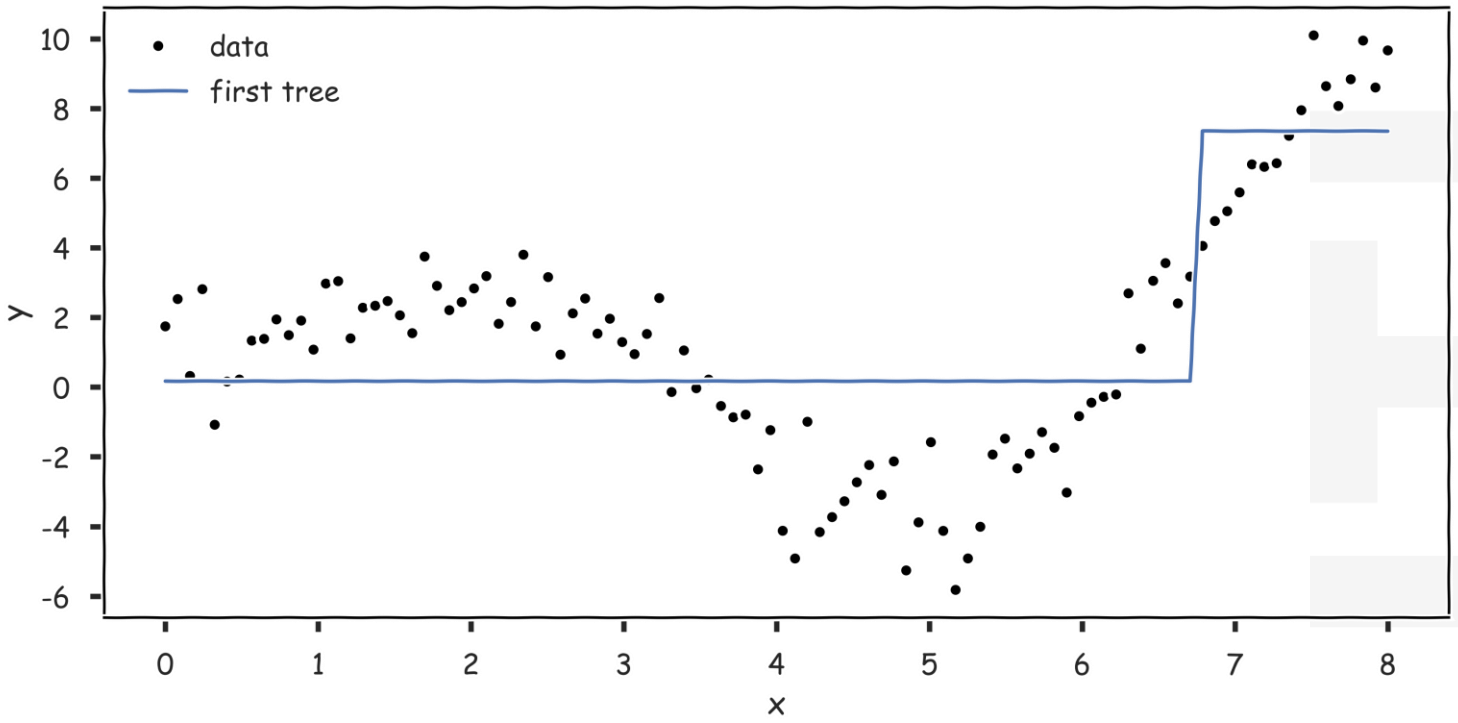
5. Repeat steps 2-4 until ***stopping*** condition met.

where λ is a constant called the ***learning rate***.

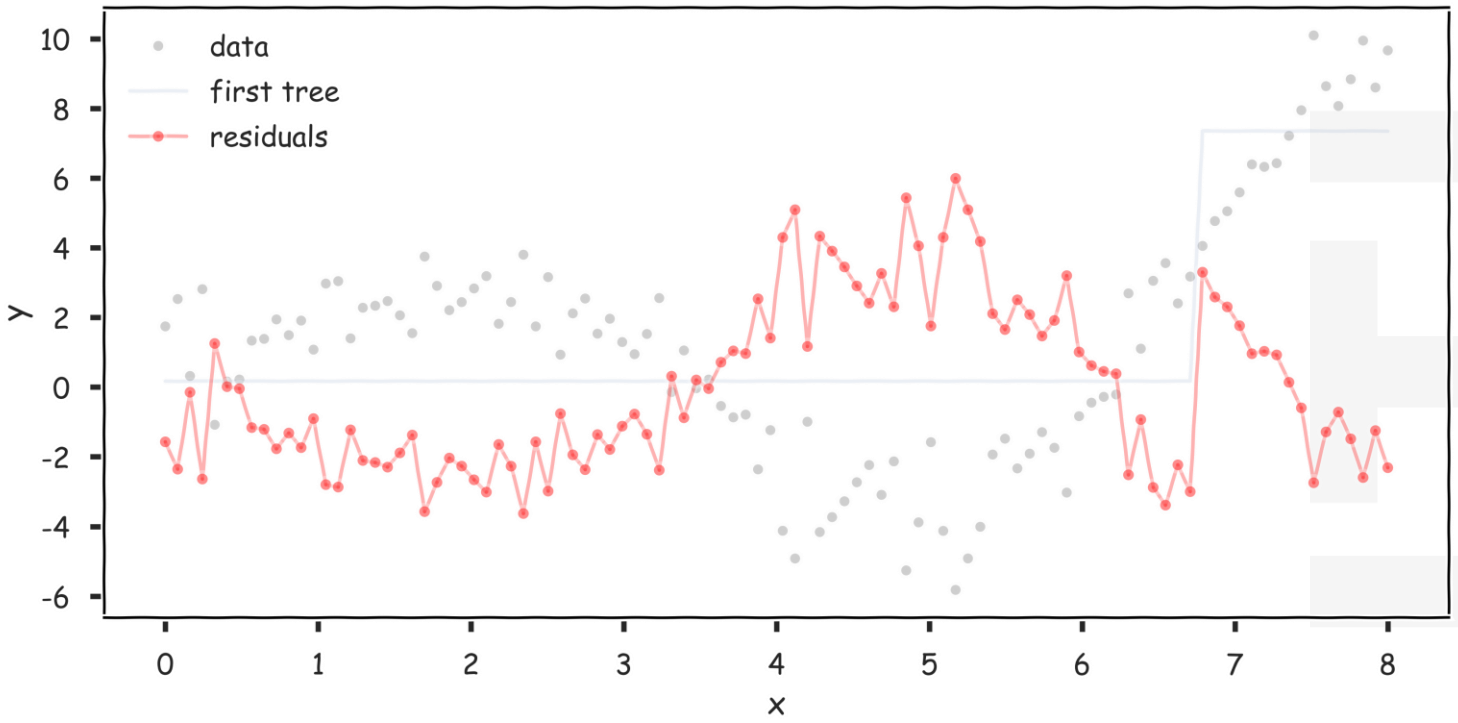
GRADIENT BOOSTING: ILLUSTRATION



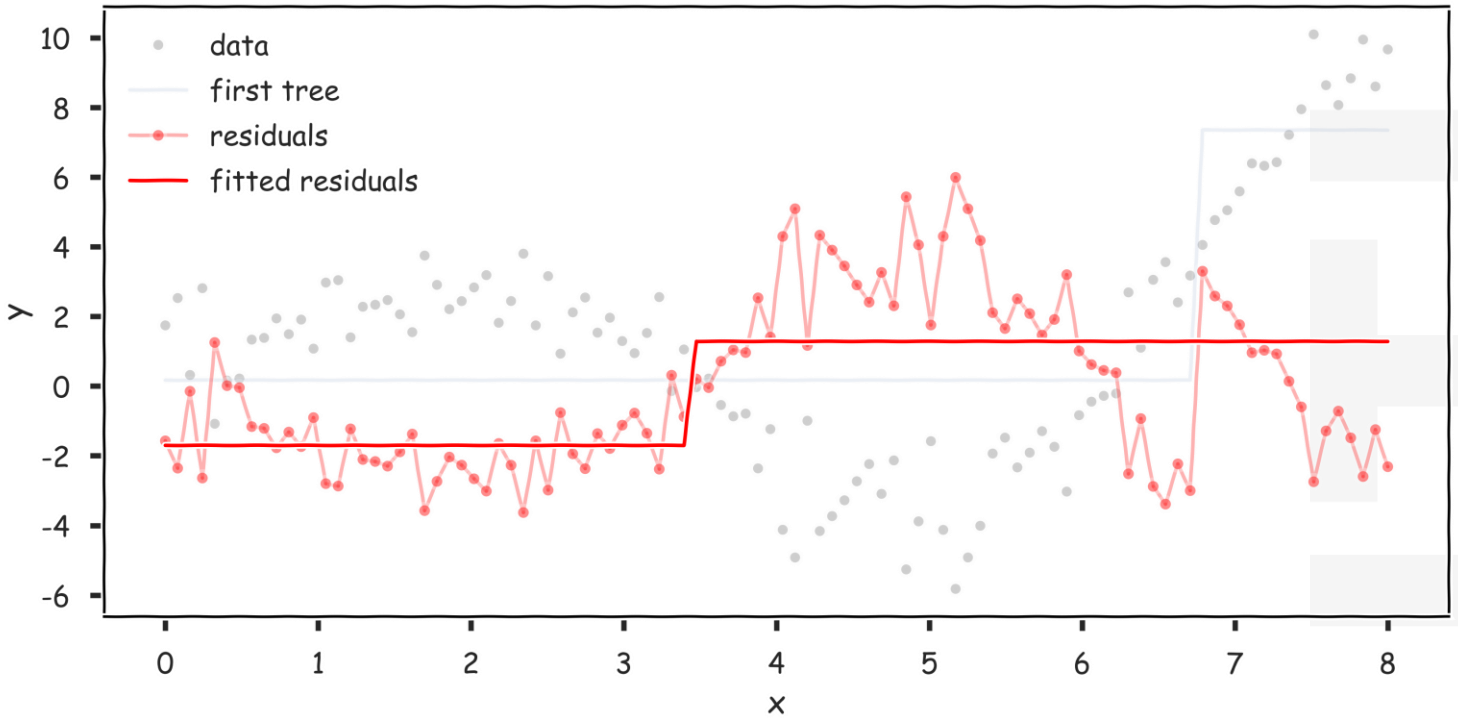
GRADIENT BOOSTING: ILLUSTRATION



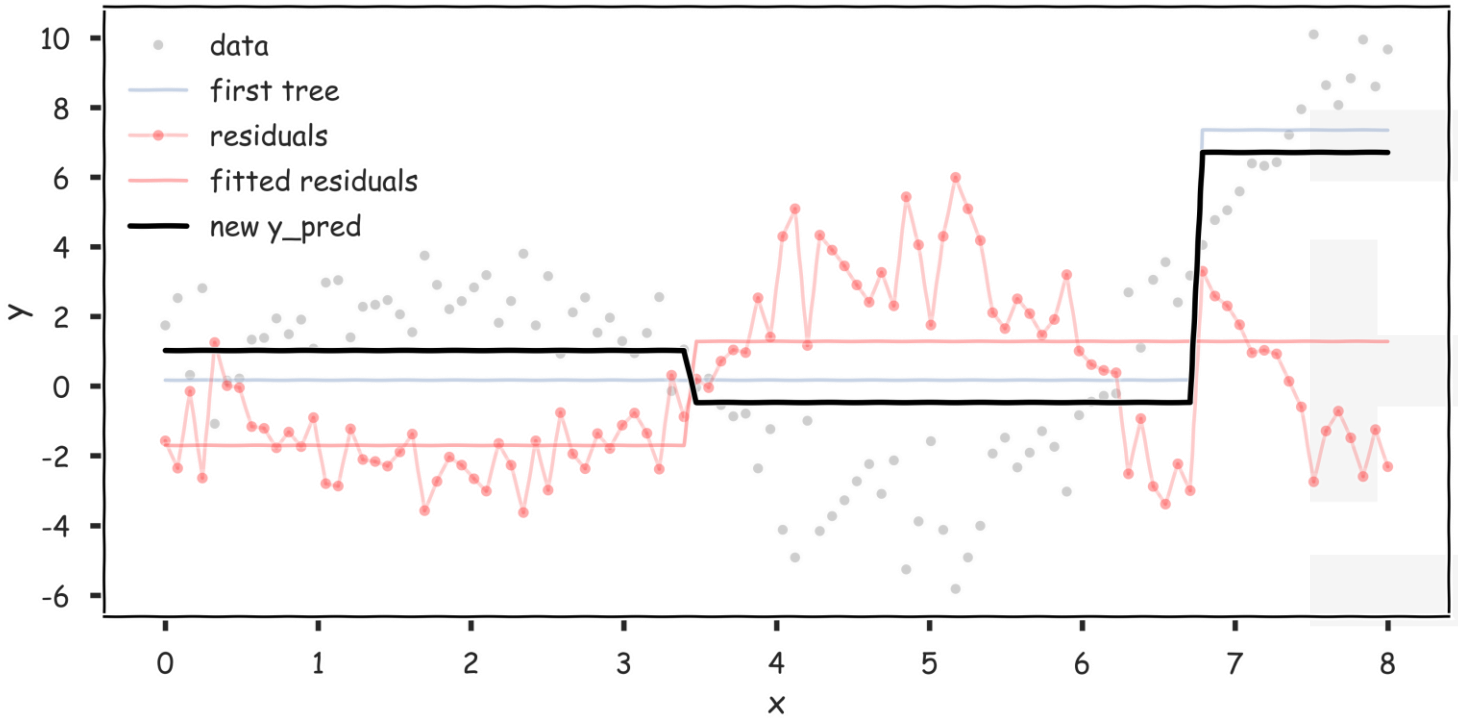
GRADIENT BOOSTING: ILLUSTRATION



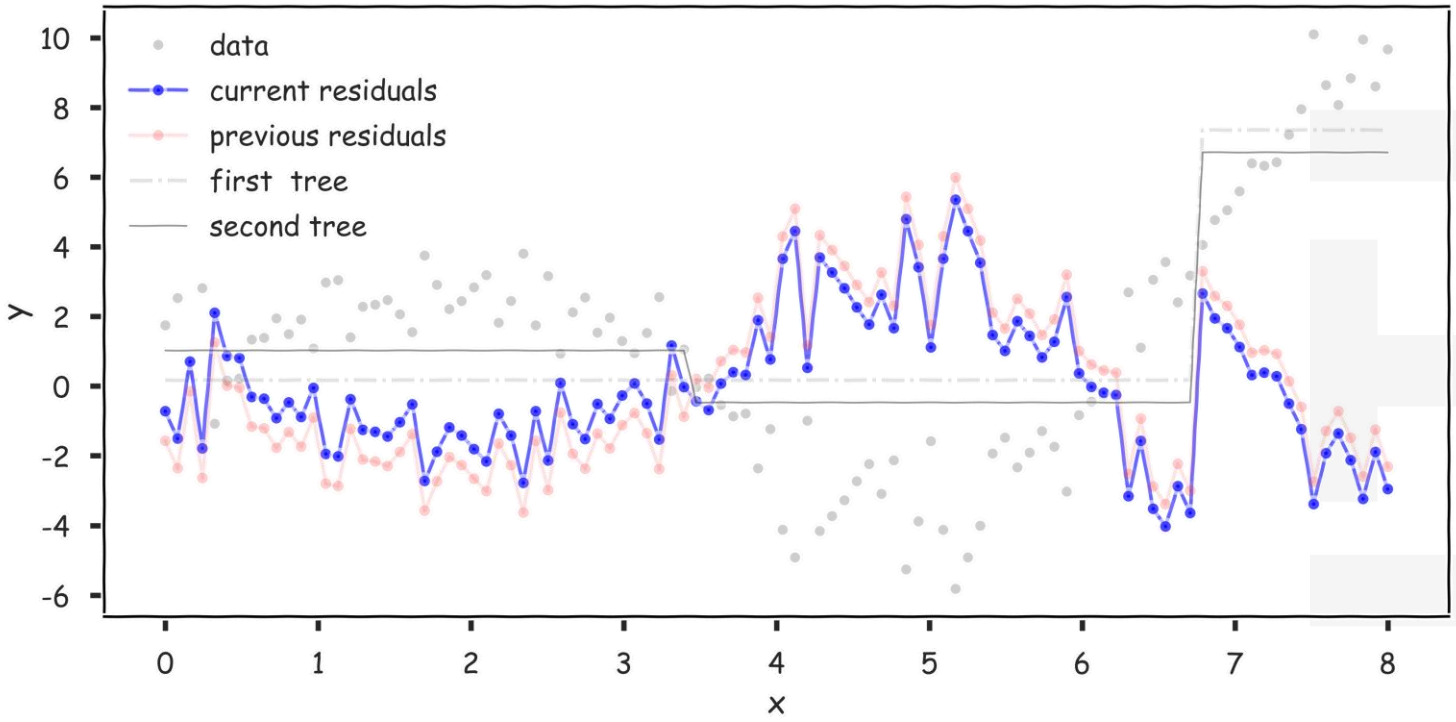
GRADIENT BOOSTING: ILLUSTRATION



GRADIENT BOOSTING: ILLUSTRATION



GRADIENT BOOSTING: ILLUSTRATION



WHY DOES GRADIENT BOOSTING WORK?

- Intuitively, each simple model $T^{(i)}$ we add to our ensemble model T, models the errors of T.
- Thus, with each addition of $T^{(i)}$, the residual is reduced

$$r_n - \lambda T^{(i)}(x_n)$$

- **Note** that gradient boosting has a tuning parameter, λ.

- In particular, how can we effectively descend through this optimization via an iterative algorithm?
- We need to formulate gradient boosting as a type of ***gradient descent***.

REVIEW: A BRIEF SKETCH OF GRADIENT DESCENT

- In optimization, when we wish to minimize a function, called the ***objective function***, over a set of variables, we compute the partial derivatives of this function with respect to the variables.

- If the partial derivatives are sufficiently simple, one can analytically find a common root - i.e. a point at which all the partial derivatives vanish; this is called a ***stationary point.***

- If the objective function has the property of being ***convex***, then the stationary point is precisely the min.

REVIEW: A BRIEF SKETCH OF GRADIENT DESCENT THE ALGORITHM

- In practice, our objective functions are complicated and analytically find the stationary point is intractable.
- Instead, we use an iterative method called ***gradient descent***:

1. Initialize the variables at any value:

$$x = [x_1, ..., x_J]$$

2. Take the gradient of the objective function at the current variable values:

$$\nabla f(x) = \left[\frac{\partial f}{\partial x_1}(x), \ldots, \frac{\partial f}{\partial x_J}(x)\right]$$

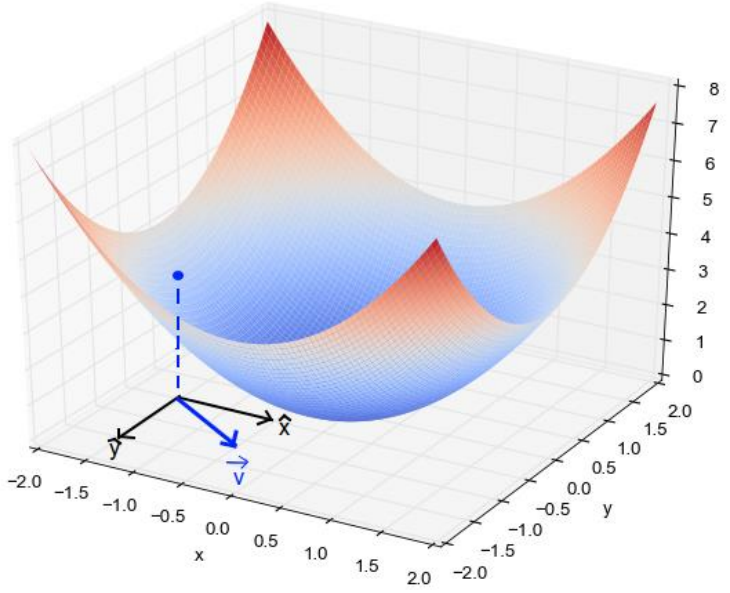
3. Adjust the variables values by some negative multiple of the gradient:

- The factor λ is often called the learning rate.

$$x \leftarrow x - \lambda \nabla f(x)$$

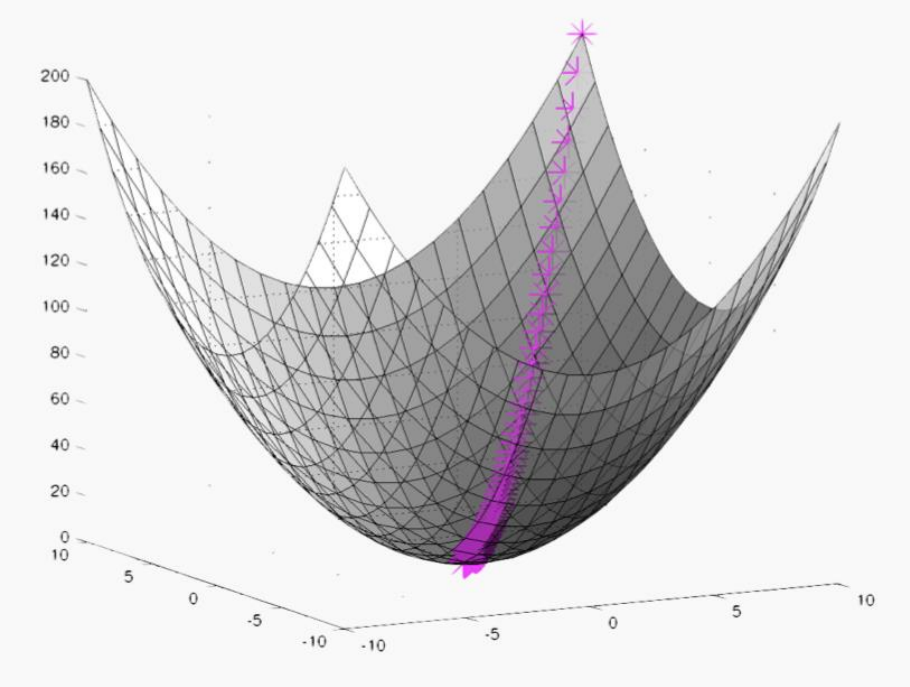
WHY DOES GRADIENT DESCENT WORK?

- **Claim:** If the function is convex, this iterative methods will eventually move x close enough to the minimum, for an appropriate choice of λ.
- **Why does this work?** Recall, that as a vector, the gradient at at point gives the direction for the greatest possible rate of increase.



WHY DOES GRADIENT DESCENT WORK?

- Subtracting a λ multiple of the gradient from x, moves x in the ***opposite*** direction of the gradient (hence towards the steepest decline) by a step of size λ.
- If *f* is convex, and we keep taking steps descending on the graph of *f*, we will eventually reach the minimum.



GRADIENT BOOSTING AS GRADIENT DESCENT

- Often in regression, our objective is to minimize the MSE

$$\text{MSE}(\hat{y}_1, \ldots, \hat{y}_N) = \frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2$$

- Treating this as an optimization problem, we can try to directly minimize the MSE with respect to the predictions

$$\begin{aligned} \nabla \text{MSE} &= \left[\frac{\partial \text{MSE}}{\partial \hat{y}_1}, \ldots, \frac{\partial \text{MSE}}{\partial \hat{y}_N} \right] \\ &= -2 \left[y_1 - \hat{y}_1, \ldots, y_N - \hat{y}_N \right] \\ &= -2 \left[r_1, \ldots, r_N \right] \end{aligned}$$

- The update step for gradient descent would look like

$$\hat{y}_n \leftarrow \hat{y}_n + \lambda r_n, \quad n = 1, \ldots, N$$

GRADIENT BOOSTING AS GRADIENT DESCENT (CONT.)

- The solution is to change the update step in gradient descent. Instead of using the gradient.
- The residuals - we use an ***approximation*** of the gradient that depends on the predictors:

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

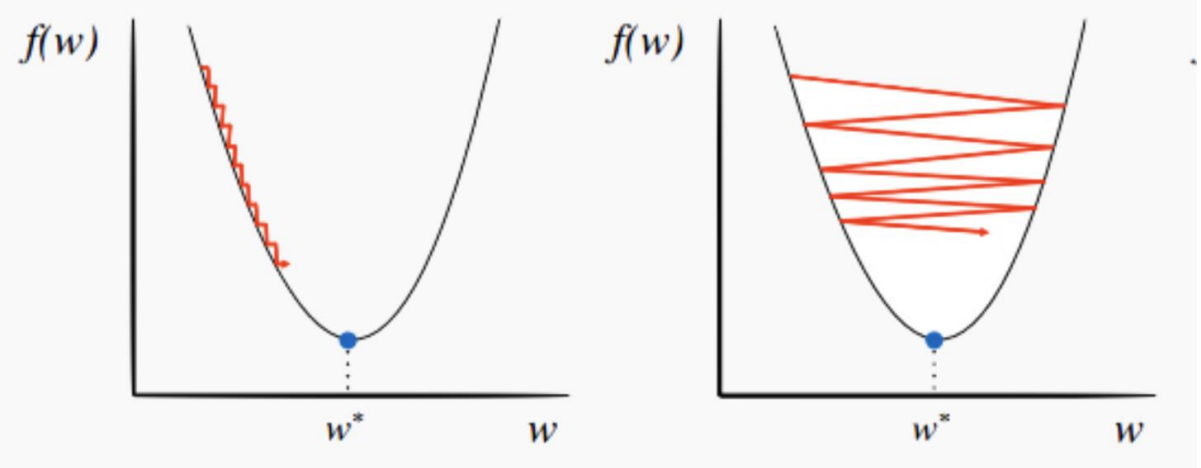
- In gradient boosting, we use a simple model to approximate the residuals, $\hat{r}_n(x_n)$, in each iteration.
- **Motto:** gradient boosting is a form of gradient descent with the MSE as the objective function.
- **Technical note:** note that gradient boosting is descending in a space of models.

CHOOSING A LEARNING RATE

- Under ideal conditions, gradient descent iteratively approximates and converges to the optimum.
- ***When do we terminate gradient descent?***
  - We can limit the number of iterations in the descent. But for an arbitrary choice of maximum iterations, we cannot guarantee that we are sufficiently close to the optimum in the end.
  - If the descent is stopped when the updates are sufficiently small (e.g. the residuals of T are small), we encounter a new problem: the algorithm may never terminate!
- Both problems have to do with the magnitude of the learning rate, λ.

CHOOSING A LEARNING RATE

- For a constant learning rate, λ, if λ is too small, it takes too many iterations to reach the optimum.



- If λ is too large, the algorithm may 'bounce' around the optimum and never get sufficiently close.

CHOOSING A LEARNING RATE

Choosing λ:

- If λ is a constant, then it should be tuned through cross validation.
- For better results, use a variable λ. That is, let the value of λ depend on the gradient

$$\lambda = h(\|\nabla f(x)\|),$$

- where $\|\nabla f(x)\|$ is the magnitude of the gradient, $\nabla f(x)$.
  So
  - around the optimum, when the gradient is small, λ should be small
  - far from the optimum, when the gradient is large, λ should be larger

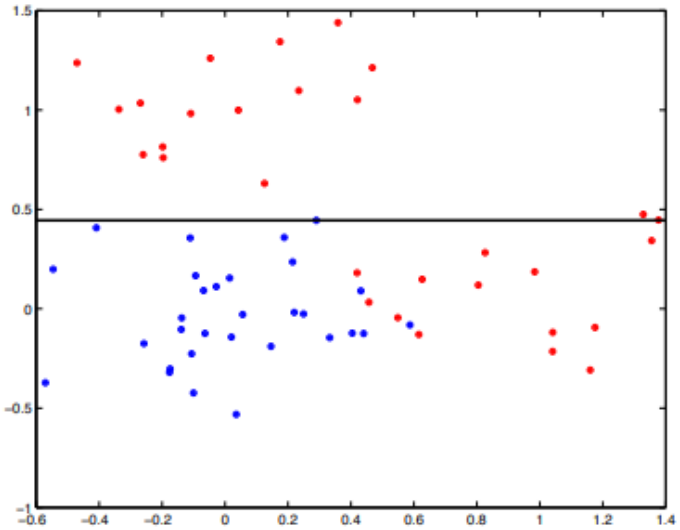
AdaBoost

COMPONENTS : DECISION STUMPS

• Consider the following simple family of component classifiers generating ± 1 labels:

$$h(\mathbf{x};\theta) = \text{sign}(\, w_1\, x_k - w_0\,)$$

where θ = {k, w1, w0}. These are called decision stumps. Each decision stump pays attention to only a single component of the input vector.



MOTIVATION FOR ADABOOST

- Using the language of gradient descent also allow 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 with respect to the predictions, $\hat{y}_n$ ☹

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***

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

- 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:

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

- 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

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

- 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

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

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

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

GRADIENT DESCENT WITH EXPONENTIAL LOSS

- We need to define a loss function for the combination so we can determine which new component h(x; θ) to add and how many votes it should receive

$$h_m(\mathbf{x}) = \alpha_1 h(\mathbf{x}; \theta_1) + \ldots + \alpha_m h(\mathbf{x}; \theta_m)$$

- While there are many options for the loss function we consider here only a simple exponential loss:

$$\exp\{ -y\, h_m(\mathbf{x}) \}$$

GRADIENT DESCENT WITH EXPONENTIAL LOSS

- Consider adding the m^{th} component:

$$\sum_{i=1}^{n} \exp\{ -y_i[h_{m-1}(\mathbf{x}_i) + \alpha_m h(\mathbf{x}_i; \theta_m)] \}$$

$$= \sum_{i=1}^{n} \exp\{ -y_i h_{m-1}(\mathbf{x}_i) - y_i \alpha_m h(\mathbf{x}_i; \theta_m) \}$$

GRADIENT DESCENT WITH EXPONENTIAL LOSS

- Consider adding the m^{th} component:

$$\sum_{i=1}^{n} \exp\{ -y_i[h_{m-1}(\mathbf{x}_i) + \alpha_m h(\mathbf{x}_i;\theta_m)] \}$$

$$= \sum_{i=1}^{n} \exp\{ -y_i h_{m-1}(\mathbf{x}_i) - y_i \alpha_m h(\mathbf{x}_i;\theta_m) \}$$

$$= \sum_{i=1}^{n} \underbrace{\exp\{ -y_i h_{m-1}(\mathbf{x}_i)\}}_{\text{fixed at stage } m} \exp\{-y_i \alpha_m h(\mathbf{x}_i;\theta_m) \}$$

$$= \sum_{i=1}^{n} W_i^{(m-1)} \exp\{-y_i \alpha_m h(\mathbf{x}_i;\theta_m) \}$$

GRADIENT DESCENT WITH EXPONENTIAL LOSS

- To increase modularity we'd like to further decouple the optimization of $h(\mathbf{x};\theta_m)$ from the associated votes α_m
- To this end we select $h(\mathbf{x};\theta_m)$ that optimizes the rate at which the loss would decrease as a function of α_m

$$\frac{\partial}{\partial \alpha_m}\Big|_{\alpha_m=0} \sum_{i=1}^{n} W_i^{(m-1)} \exp\{-y_i \alpha_m h(\mathbf{x}_i;\theta_m)\} =$$

$$\left[\sum_{i=1}^{n} W_i^{(m-1)} \exp\{-y_i \alpha_m h(\mathbf{x}_i;\theta_m)\} \cdot \big(-y_i h(\mathbf{x}_i;\theta_m)\big)\right]_{\alpha_m=0}$$

$$= \left[\sum_{i=1}^{n} W_i^{(m-1)} \big(-y_i h(\mathbf{x}_i;\theta_m)\big)\right]$$

GRADIENT DESCENT WITH EXPONENTIAL LOSS

- We find $h(\mathbf{x}; \hat{\theta}_m)$ that minimizes

$$-\sum_{i=1}^{n} \tilde{W}_i^{(m-1)} y_i h(\mathbf{x}_i; \theta_m)$$

  where $\sum_{i=1}^{n} \tilde{W}_i^{(m-1)} = 1$.

- α_m is subsequently chosen to minimize

$$\sum_{i=1}^{n} \tilde{W}_i^{(m-1)} \exp\{-y_i \alpha_m h(\mathbf{x}_i; \hat{\theta}_m)\}$$

Given: $(x_1, y_1), \ldots, (x_m, y_m)$ where $x_i \in \mathcal{X}$, $y_i \in \{-1, +1\}$.

Initialize $D_1(i) = 1/m$ for $i = 1, \ldots, m$. ← Initial Distribution of Data

For $t = 1, \ldots, T$:

- Train weak learner using distribution D_t.
- Get weak hypothesis $h_t : \mathcal{X} \to \{-1, +1\}$. ← Train model
- Aim: select h_t with low weighted error:

$$\varepsilon_t = \Pr_{i \sim D_t}\left[h_t(x_i) \neq y_i\right].$$ ← Error of model

- Choose $\alpha_t = \frac{1}{2} \ln\left(\frac{1 - \varepsilon_t}{\varepsilon_t}\right)$. ← Coefficient of model
- Update, for $i = 1, \ldots, m$:

$$D_{t+1}(i) = \frac{D_t(i) \exp(-\alpha_t y_i h_t(x_i))}{Z_t}$$ ← Update Distribution

where Z_t is a normalization factor (chosen so that D_{t+1} will be a distribution).

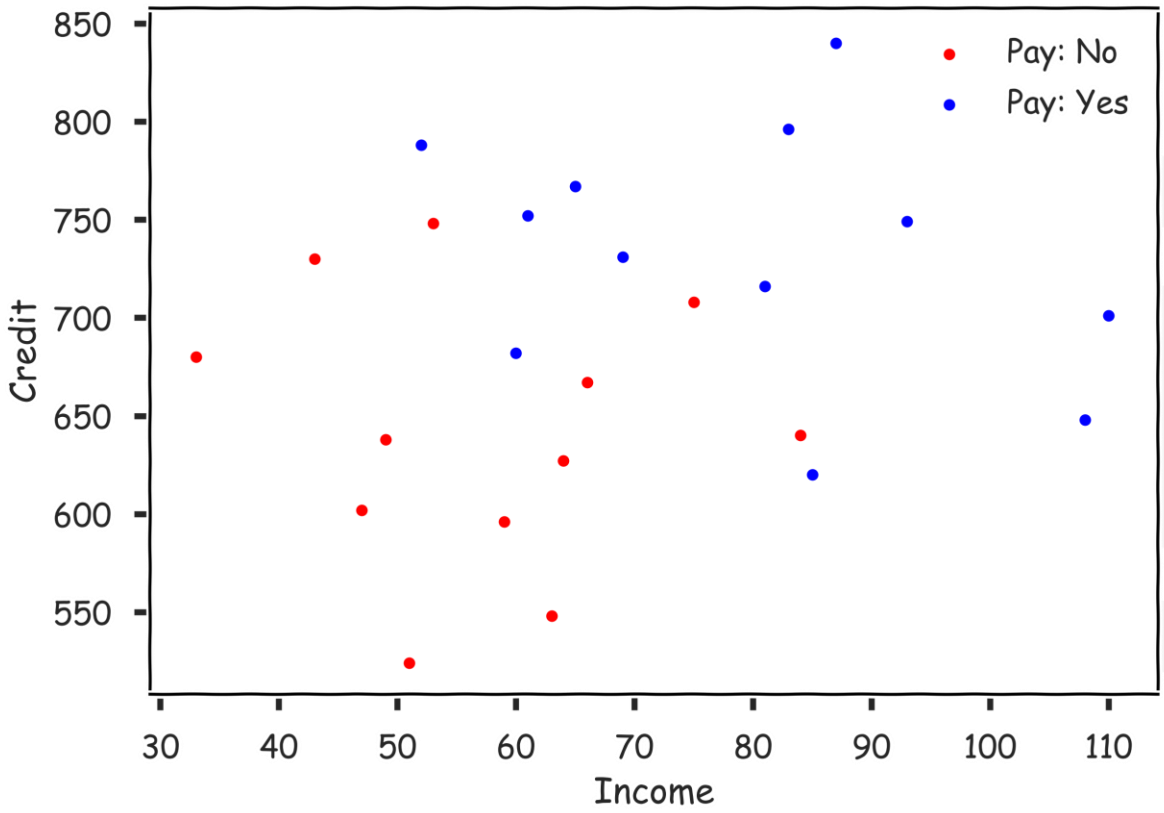
Output the final hypothesis:

$$H(x) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right).$$ ← Final average

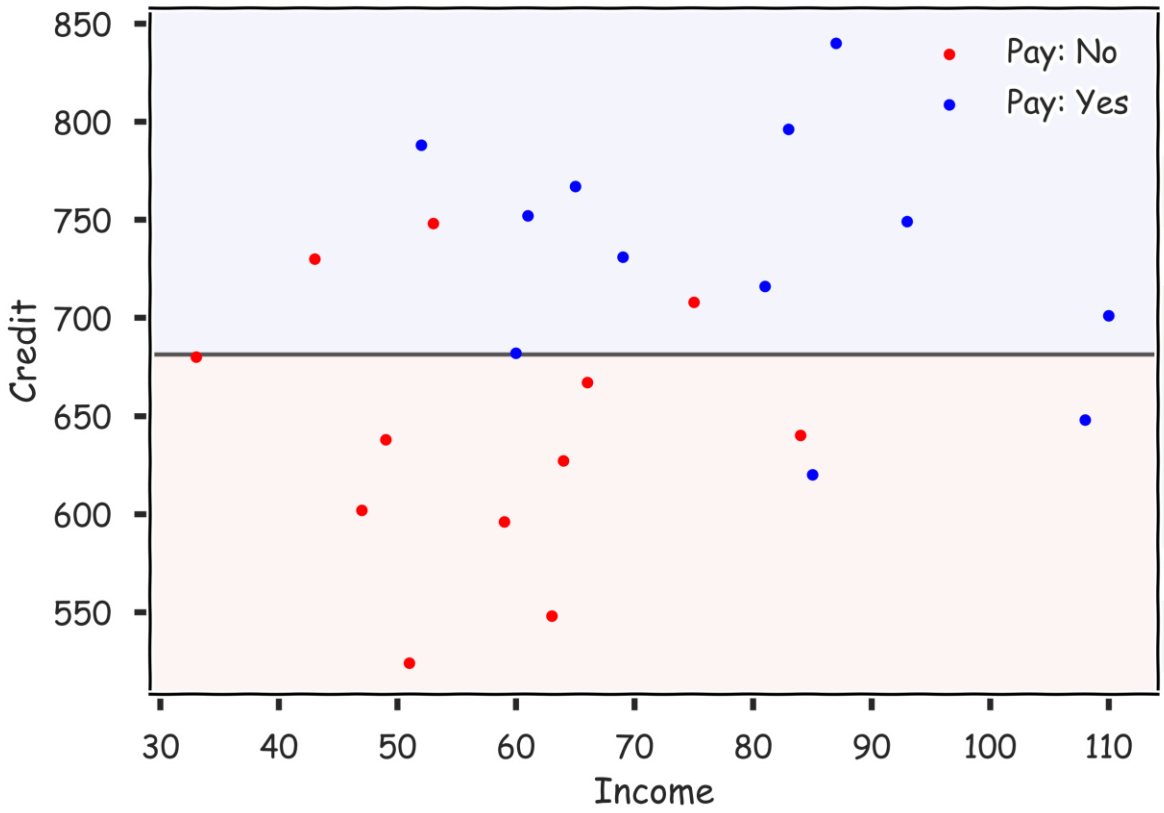
Theorem: training error drops exponentially fast

https://www.cs.princeton.edu/~schapire/papers/explaining-adaboost.pdf

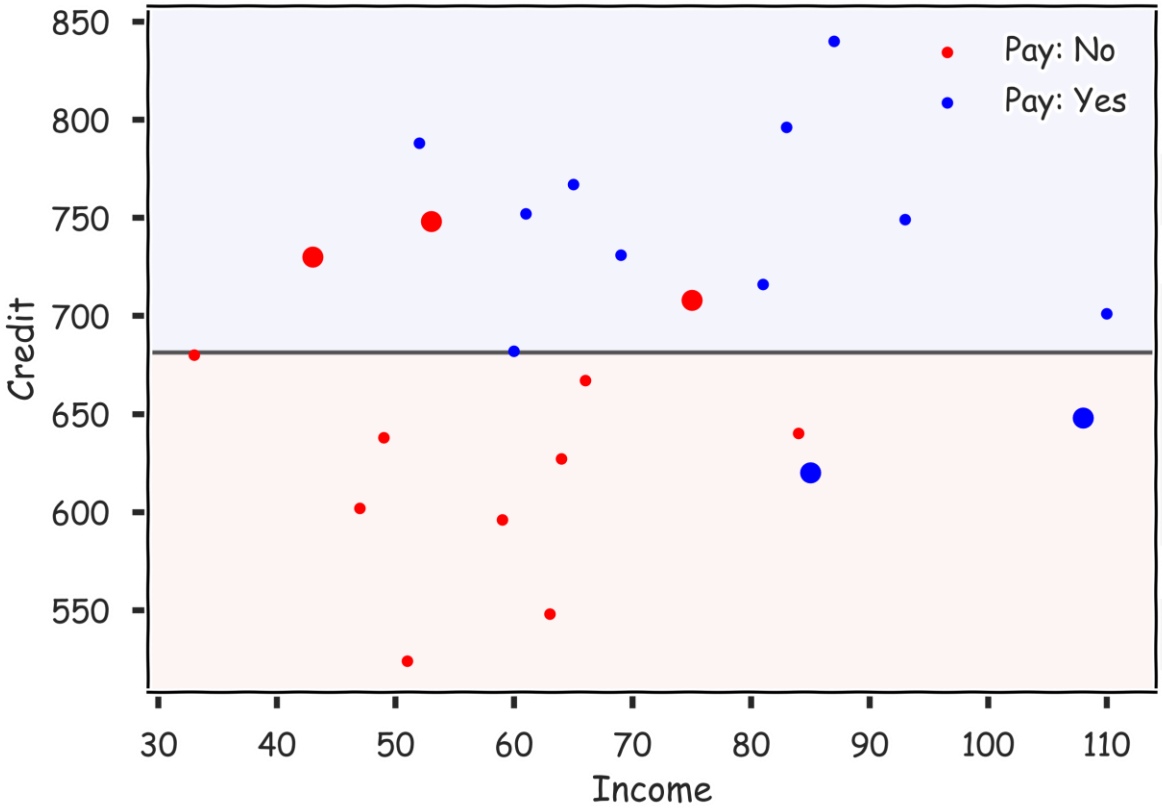
ADABOOST: ILLUSTRATION, START WITH LENDING DATA



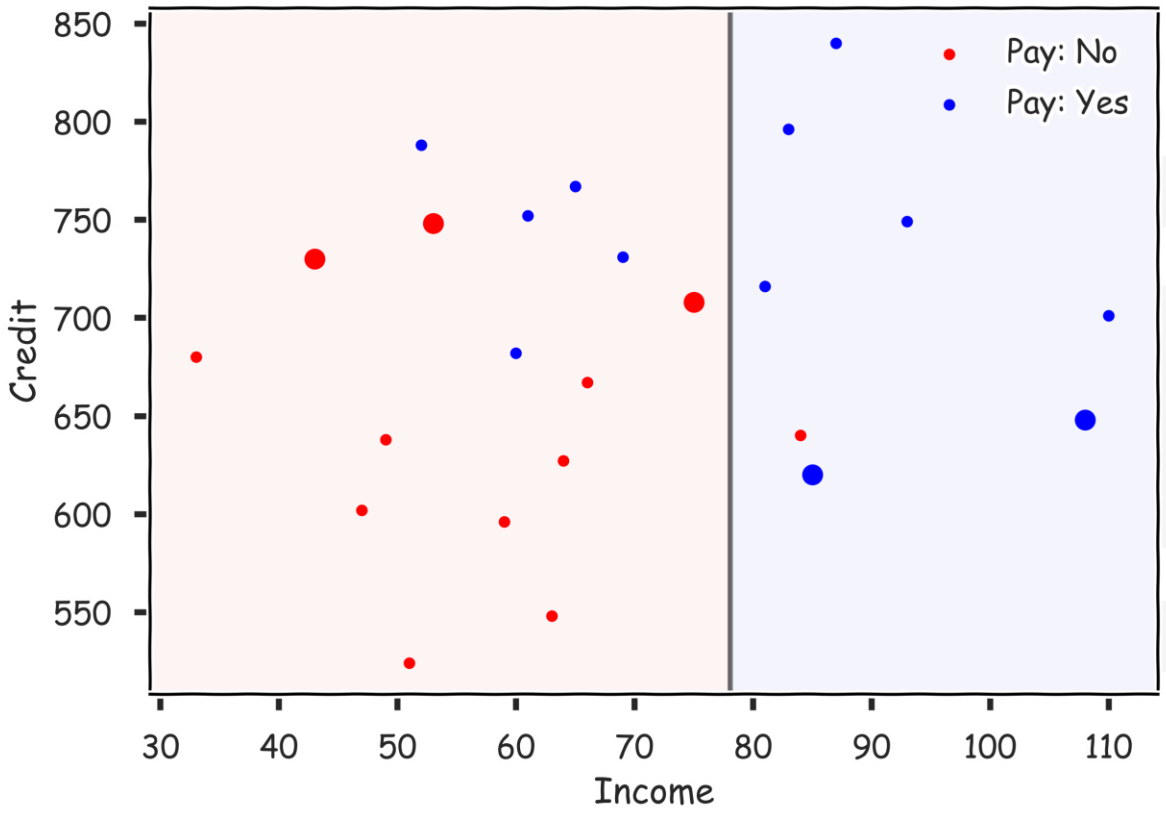
ADABOOST: ILLUSTRATION, SIMPLE DECISION TREE



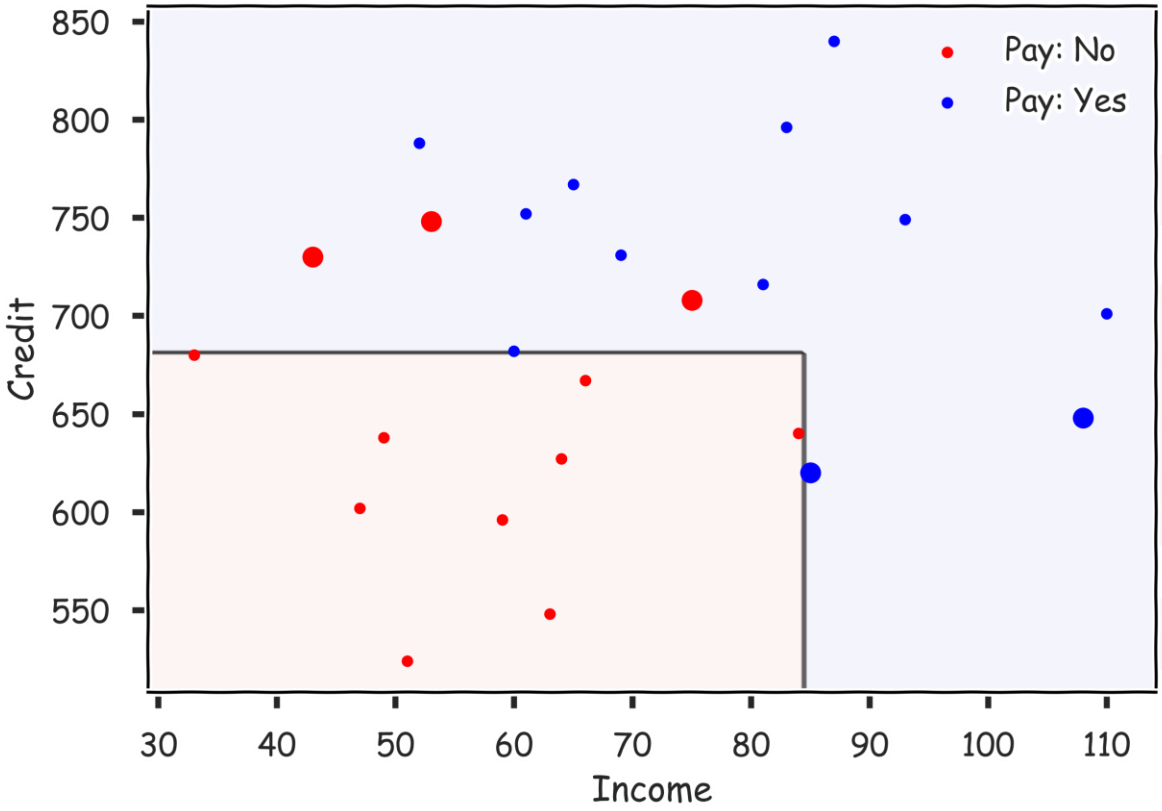
ADABOOST: ILLUSTRATION, ERRORS ARE WEIGHTED HIGHER



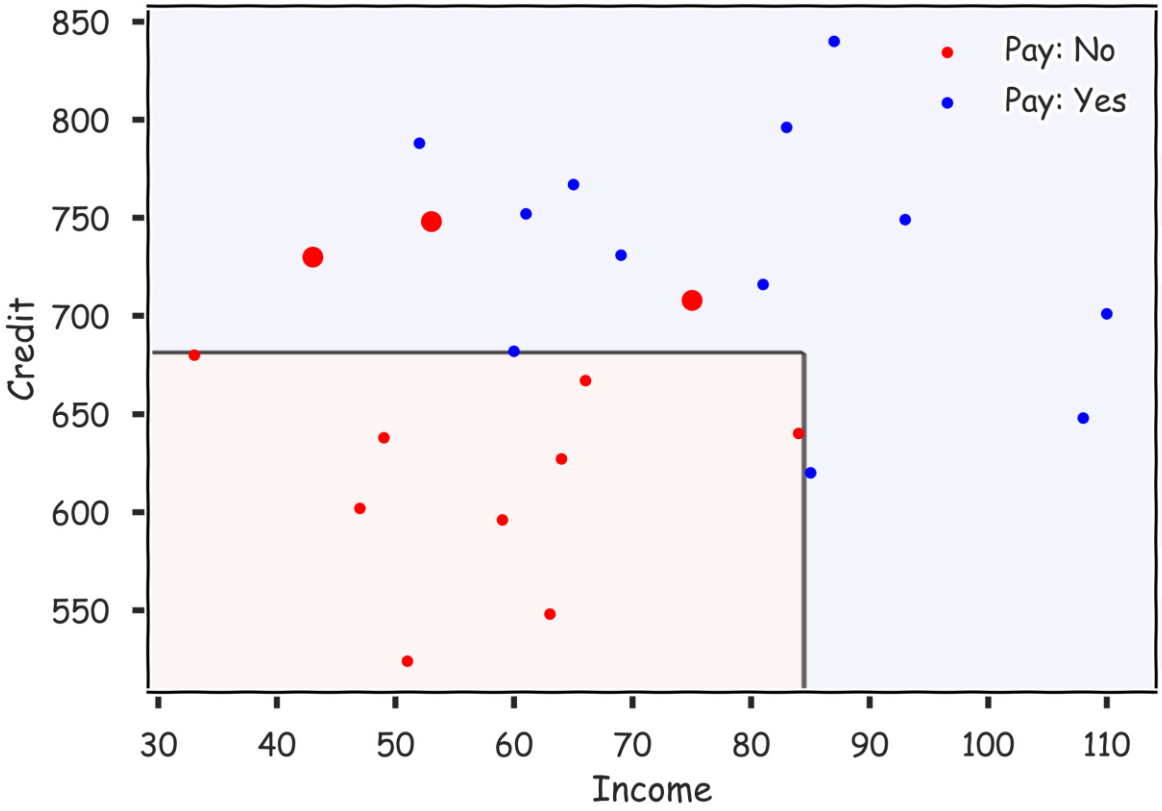
ADABOOST: ILLUSTRATION, 2ND TREE WITH WEIGHTED POINTS



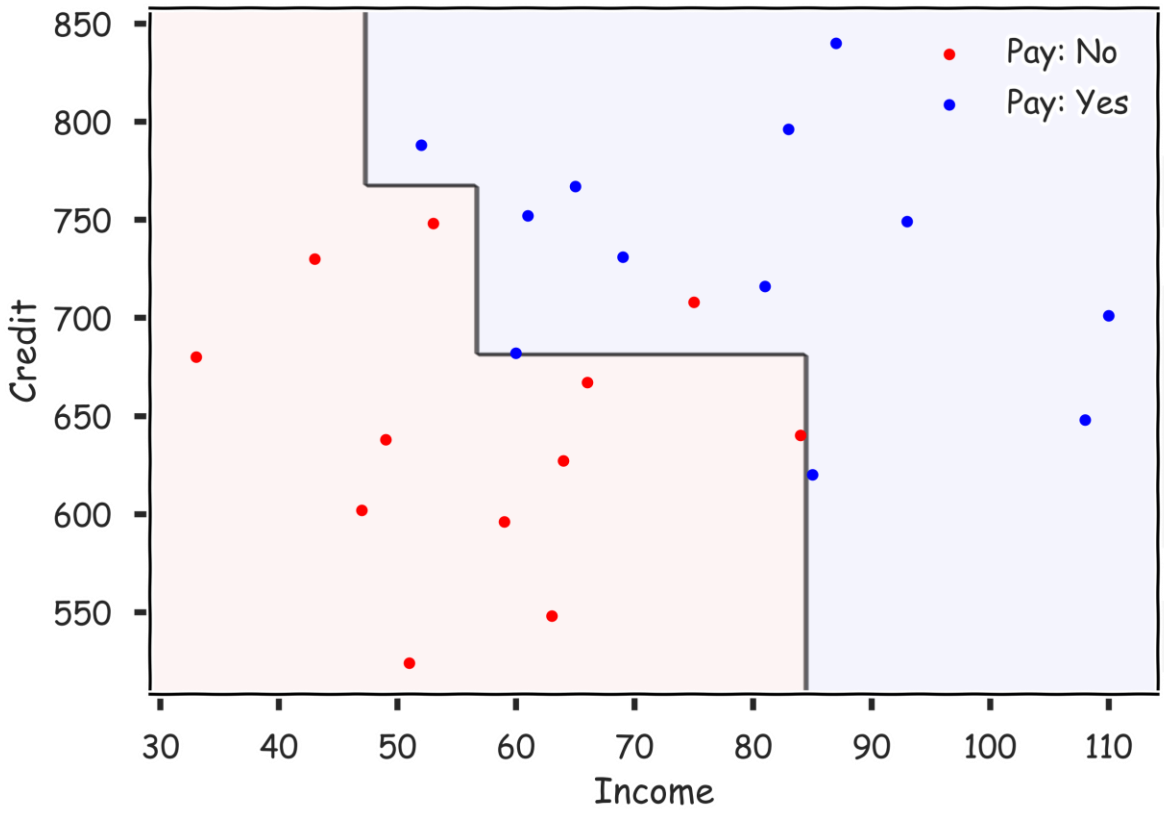
ADABOOST: ILLUSTRATION, COMBINE THE TREES



ADABOOST: ILLUSTRATION, RE-WEIGHT POINTS

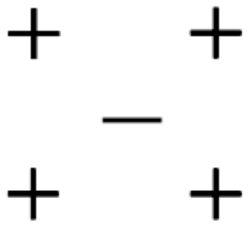


ADABOOST: ILLUSTRATION, ETC

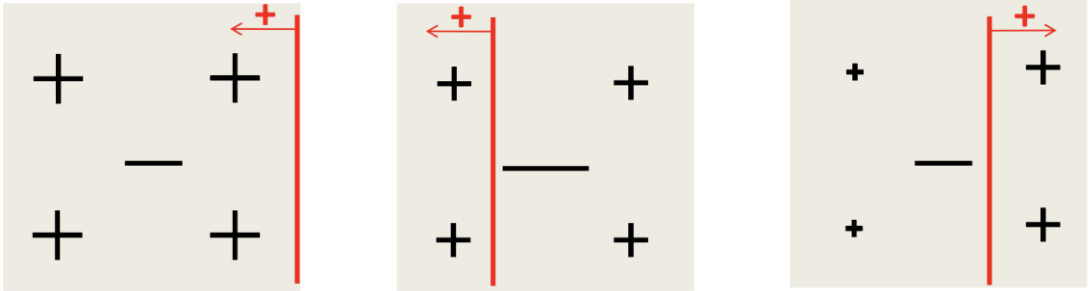


QUESTION

- Consider training a boosting classifier using decision stumps on the following dataset plot:



- How many iterations will it take to achieve zero training error? Explain the reasons.



CHOOSING THE LEARNING RAGE

- 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

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

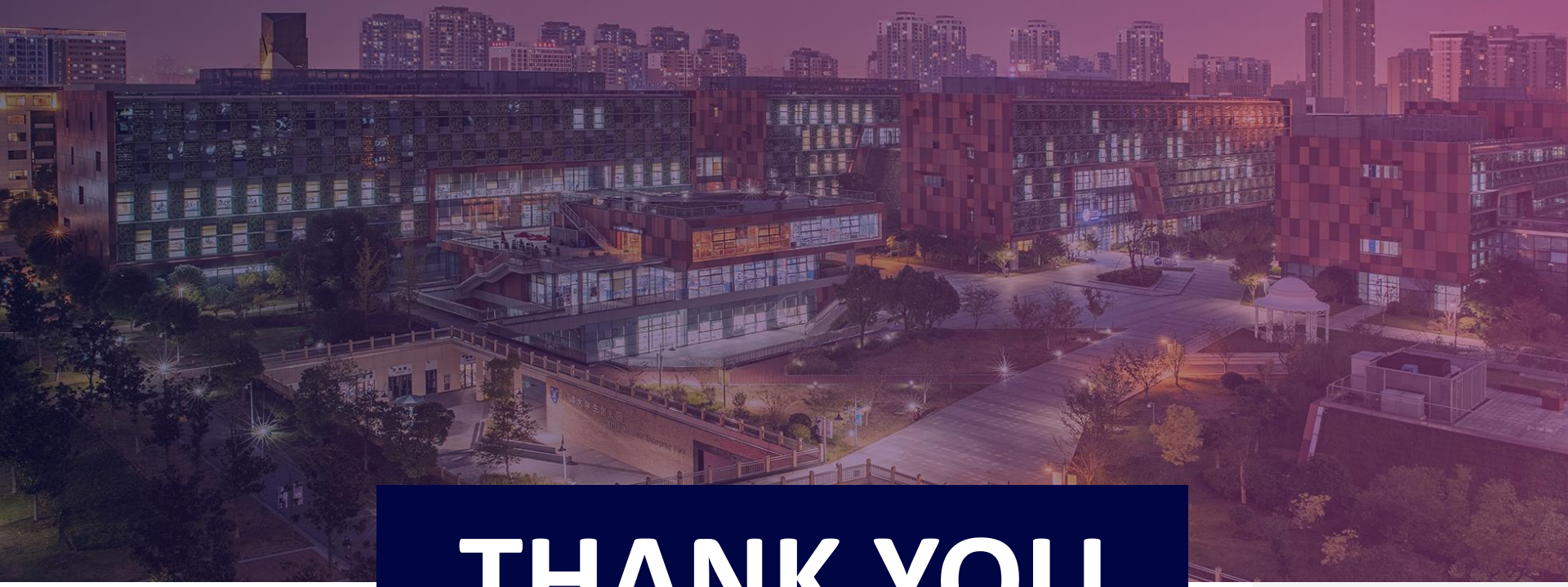
FINAL THOUGHTS ON BOOSTING

- 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

ADVANCED TOPICS

BOOSTING IN SKLEARN

- Python has boosting algorithms implemented for you:
- **`sklearn.ensemble.AdaBoostClassifier`**
- **`sklearn.ensemble.AdaBoostRegressor`**
- With arguments of **base_estimator** (what models to use), **n_estimators** (max number of models to use), **learning_rate** (λ), etc…



THANK YOU

VISIT US

WWW.XJTLU.EDU.CN

FOLLOW US

@XJTLU

Xi'an Jiaotong-Liverpool University
西交利物浦大學