ARTIFICIAL INTELLIGENCE NEEDS REAL INTELLIGENCE

Ensemble Learning II

Professor: Tianyuan ZHANG
tianyuan.zhang@kedgebs.com

KEDGE BUSINESS SCHOOL

Recap of Previous Session

- **Ensemble learning**
  - A machine learning paradigm where multiple models (often called "**weak learners**") are trained to solve the same problem and combined to get better results.
- Rationale
  - No single model can capture all the patterns in the data perfectly.
  - Leverage the strength and compensate for the weaknesses of individual models.
  - A group of "weak learners" can come together to form a "strong learner".
  - **Wisdom of the crowd**

Recap of Previous Session

- Two questions to answer when designing ensemble learning methods
  1. How to generate multiple base models (estimators)?
  2. How to integrate / combine their predictions to make final prediction?
- Different decisions lead to different **types** of ensemble learning methods
  - Bagging
  - Boosting
  - Stacking

Recap of Previous Session

- **Bootstrap Aggregating → Bagging**
- An ensemble learning technique
  - **Parallel model training**
    - Multiple models / estimators are trained simultaneously on different subsets of the original training dataset.
  - **Bootstrap sampling**
    - Each subset of training data is generated by randomly sampling with replacement.
  - **Aggregation of predictions**
    - Regression: Averaging the predictions
    - Classification: Majority voting

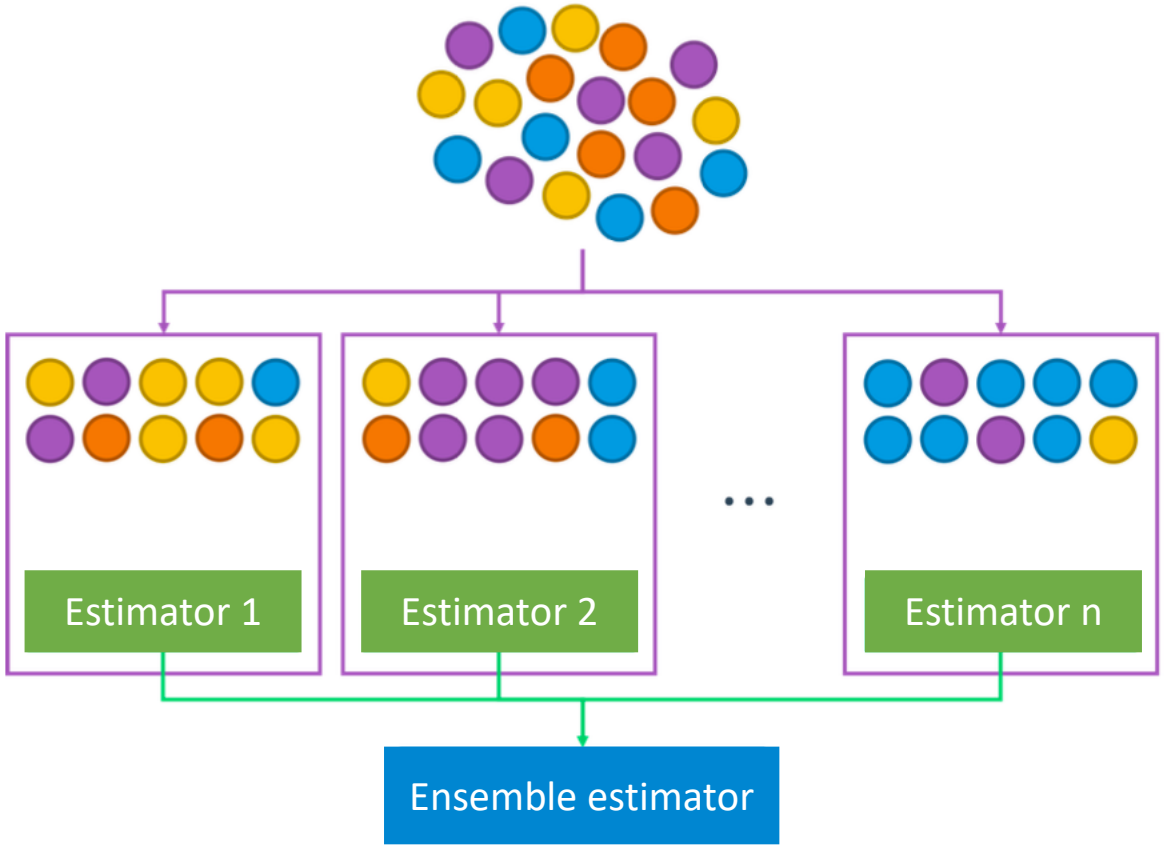
Recap of Previous Session

- **Bootstrap sampling** in the bagging method
  - A statistical resampling where random samples of the training dataset are taken **with replacement.**

Image source: Slides from Sebastian Raschka @ University of Wisconsin-Madison

Recap of Previous Session

- **Bagging**



Original dataset

↓

Bootstrap sampling

↓

Parallel training

↓

Aggregation of predictions

Image source: https://en.wikipedia.org/wiki/Bootstrap_aggregating

Recap of Previous Session

- **Bias**
  - The error from erroneous (often overly simplistic) assumptions in the algorithm
  - High bias → Under-fitting
- **Variance**
  - The error from erroneous (often overly simplistic) assumptions in the algorithm
  - High variance → Over-fitting

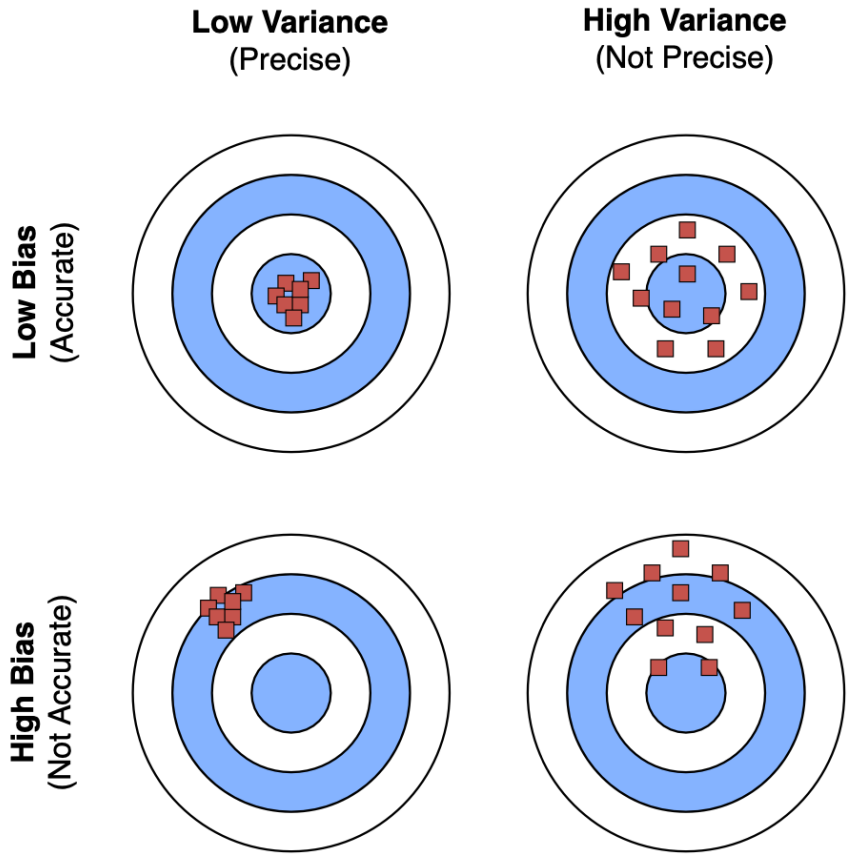


Image source: Slides from Sebastian Raschka @ University of Wisconsin-Madison

Recap of Previous Session

- **Bias-Variance trade-off**
  - For ensemble learning – Bagging
    - Generate multiple base estimators with
      - Low bias
      - High variance
    - Average the predictions to reduce the high variance and keep the low bias

Low bias

High variance

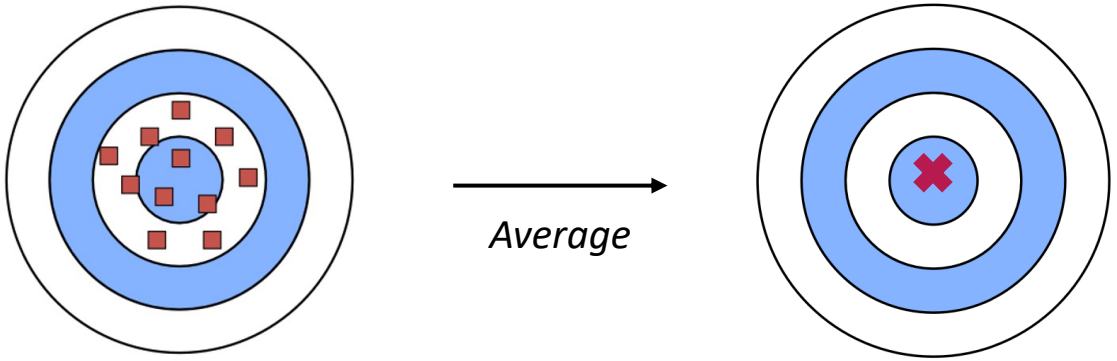


Image source: https://en.wikipedia.org/wiki/Bias–variance_tradeoff

Recap of Previous Session

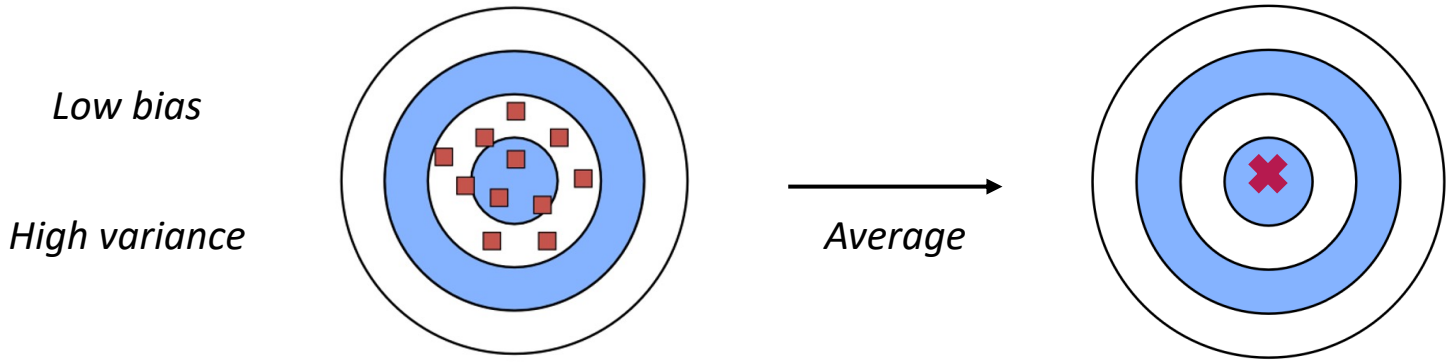
- **Random Forests**
  - A bagging method that uses decision tree as the weak learner
  - Why decision tree?
    - Prone to over-fitting
    - High variance, low bias
  - One more trick
    - When training each decision tree, use a random subset of features instead of all features.
      - Make different base estimators less correlated, more independent
  - Byproduct
    - Feature importance

Outline

- **Introduction to Boosting**
- Adaptive Boosting
- Gradient Boosting
- Introduction to Stacking

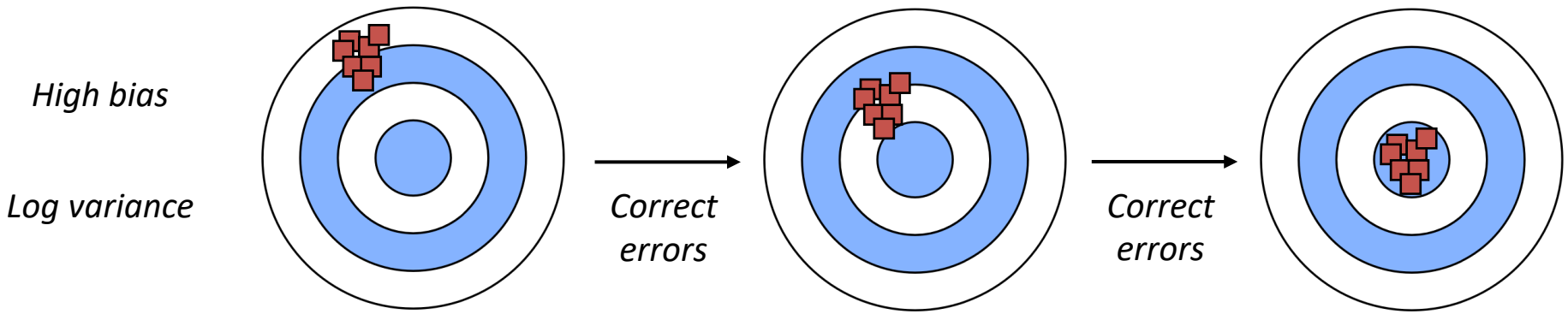
Introduction to Boosting

- Rationale behind **bagging**
  - Generate multiple base estimators with
    - **Low bias**
    - **High variance**
  - Average the predictions to reduce the high variance and keep the low bias



Introduction to Boosting

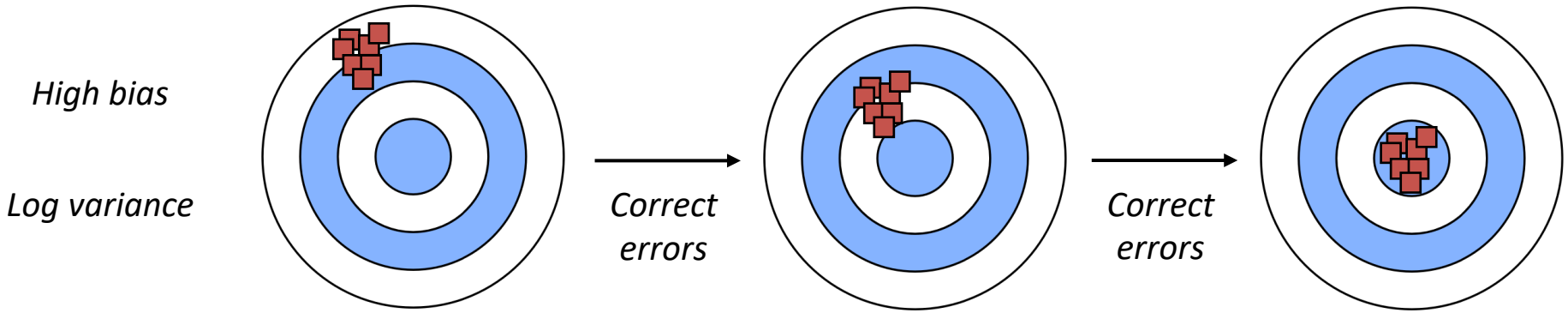
- Rationale behind **boosting**
  - Generate a base estimator with
    - **High bias**
    - **Low variance**
  - Correct the previous prediction errors to reduce the high bias



Introduction to Boosting

- Rationale behind **boosting**
  - Generate multiple base estimators with
    - **High bias**
    - **Low variance**

    → *Decrease model complexity* → *Under-fitted models* → *Shallow decision tree* → *Decision stump (depth = 1)*
  - Correct the previous prediction errors to reduce the high bias

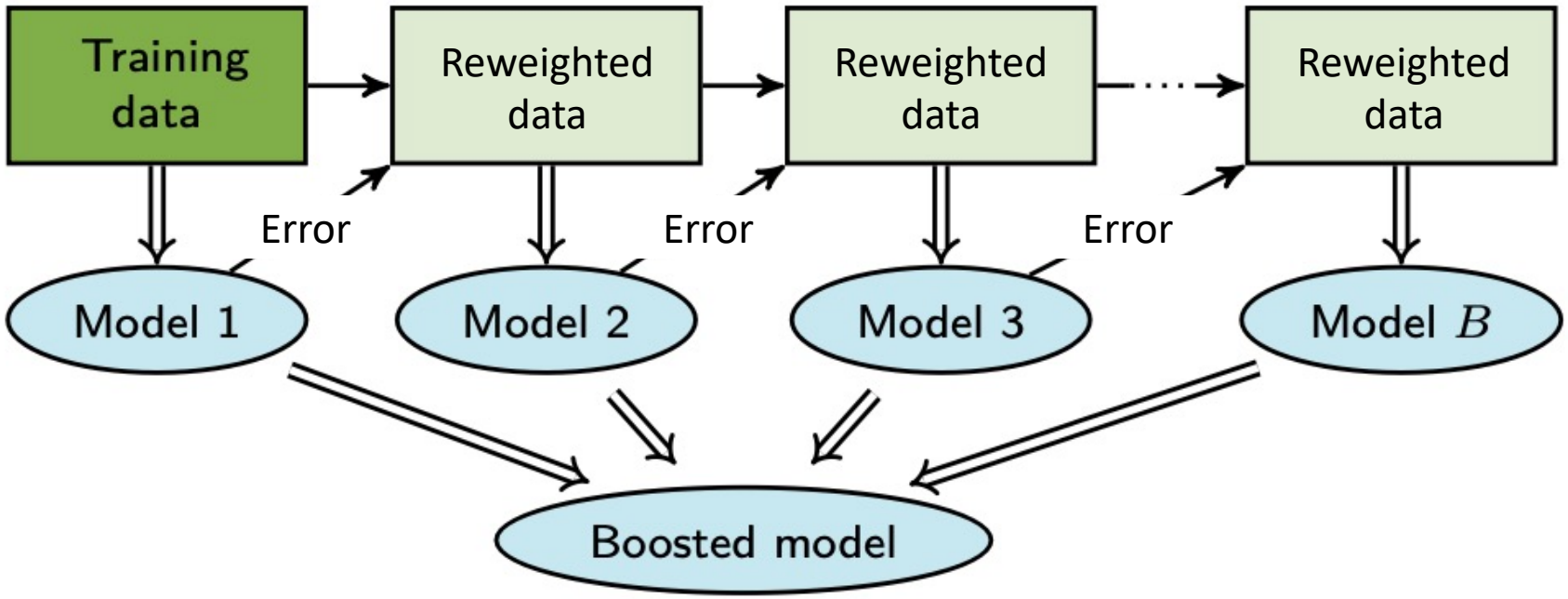


Introduction to Boosting

- **Boosting**
  - A type of ensemble learning methods that boosts weak learners to strong learners
  - Key features
    - **Sequential model training**
      - Each model in the sequence is trained on the dataset with a focus on the instances that previous models mis-estimated.
    - **Error correction focus**
      - The algorithm identifies the errors or mispredictions of the previous model and increases their weight, making them more significant in the training of the next model.
    - **Cumulative learning**
      - Unlike parallel ensemble methods, boosting involves cumulative learning where each model builds on the knowledge of its predecessors.

Introduction to Boosting

- **Boosting**



- An **iterative** process, where the training set is **reweighted** in each iteration

Introduction to Boosting

- Bagging vs. Boosting

Bagging	Boosting
Learn base estimators in parallel	Learn base estimators sequentially
Use bootstrapped sets as training set	Use reweighted sets as training set
Reduce variance (deep trees as base models)	Reduce bias (shallow trees as base models)
Too many base estimators won't lead to overfit	Too many base estimators will lead to overfit

Image source: Slides from Thomas Schön @ Uppsala Universitet

Outline

- Introduction to Boosting
- **Adaptive Boosting**
- Gradient Boosting
- Introduction to Stacking

Adaptive Boosting

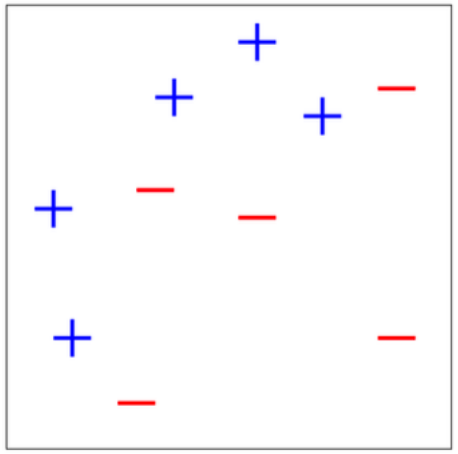
- **AdaBoost**
  - Obtain different base models by **reweighting** the training data at each iteration
    - Reduce bias by focusing on the 'hard' training instances
    - Increase weights of instances misestimated by the previous estimator and vice versa
  - Base estimators should be simple so that bias is high, and variance is low
    - Different instance weights can lead to different base estimators
    - Underfitting model: **Decision stump** (or very shallow tree)
  - AdaBoost also assigns each base estimator a unique weight to aggregate their predictions
    - Classification: weighted voting
    - Regression: weighted median

Adaptive Boosting

- **AdaBoost procedure**
  1. Assign initial weights $w_i^1 = 1/n$ to all training data points $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$
  2. For iteration $b = 1$ to B:
     1. Train a weak learner $\hat{y}^{(b)}(\mathbf{x})$ on the weighted training data $\left\{\left(\mathbf{x}_i, y_i, w_i^b\right)\right\}_{i=1}^n$
     2. Calculate the weight $\alpha^{(b)}$ of the weak learner $\hat{y}^{(b)}(\mathbf{x})$ based on its error rate $\varepsilon^{(b)}$
     3. Update the weights $\left\{w_i^{b+1}\right\}_{i=1}^n$ from $\left\{w_i^b\right\}_{i=1}^n$
        1. Increase weights for all data points misestimated by $\hat{y}^{(b)}(\mathbf{x})$
        2. Decrease weights for all data points correctly estimated by $\hat{y}^{(b)}(\mathbf{x})$
  3. Aggregate the predictions of the B weaker learners $\left\{\hat{y}^{(1)}(\mathbf{x}), \ldots, \hat{y}^{(B)}(\mathbf{x})\right\}$ to obtain the prediction of the ensemble $\hat{y}_{\text{ensemble}}^B(\mathbf{x})$

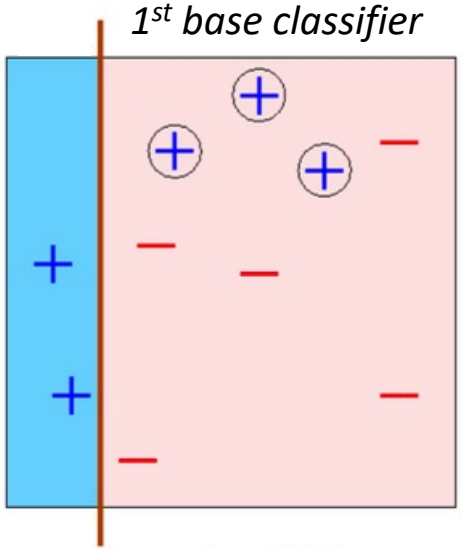
Adaptive Boosting

- A binary classification example
  - Iteration 1



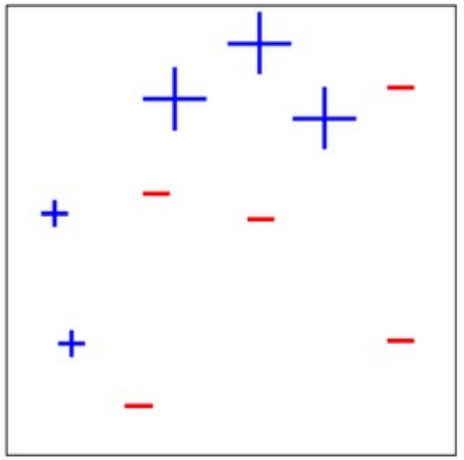
Initial train data

1st base classifier



$\varepsilon^{(1)} = 0.30$

$\alpha^{(1)} = 0.42$

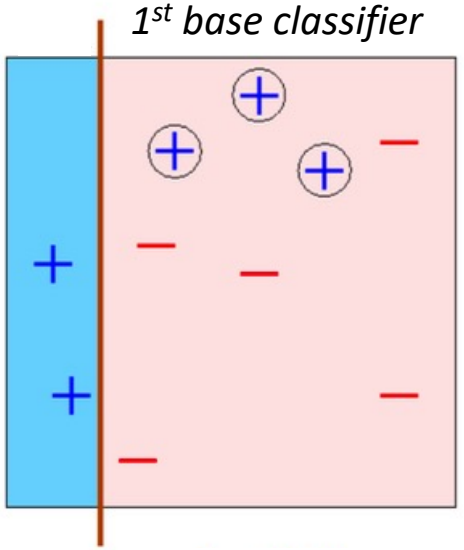


Reweighted train data

Adaptive Boosting

- A binary classification example
  - Iteration 2

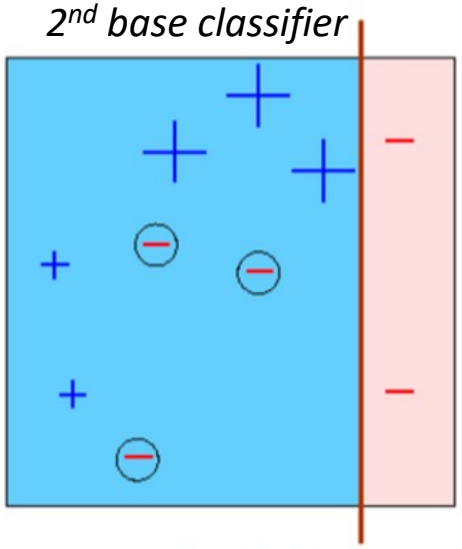
1st base classifier



$\varepsilon^{(1)} = 0.30$

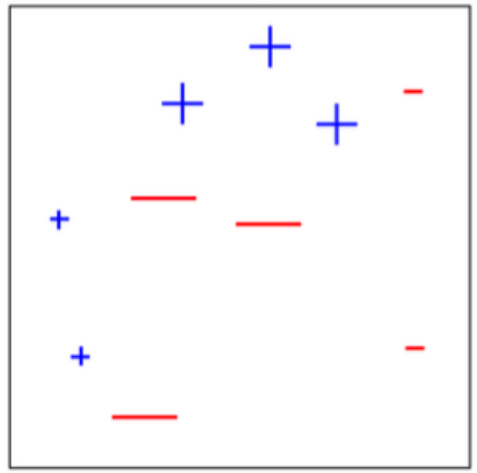
$\alpha^{(1)} = 0.42$

2nd base classifier



$\varepsilon^{(2)} = 0.21$

$\alpha^{(2)} = 0.65$

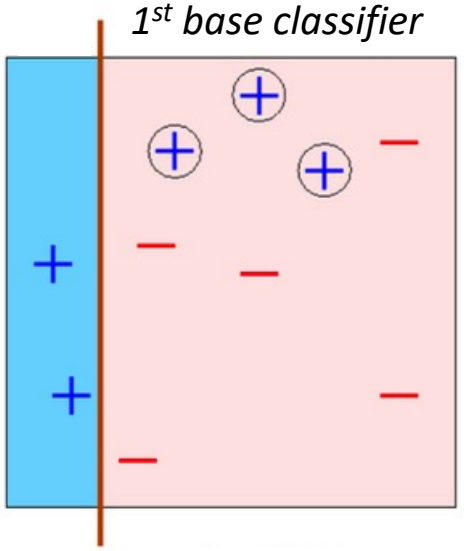


Reweighted train data

Adaptive Boosting

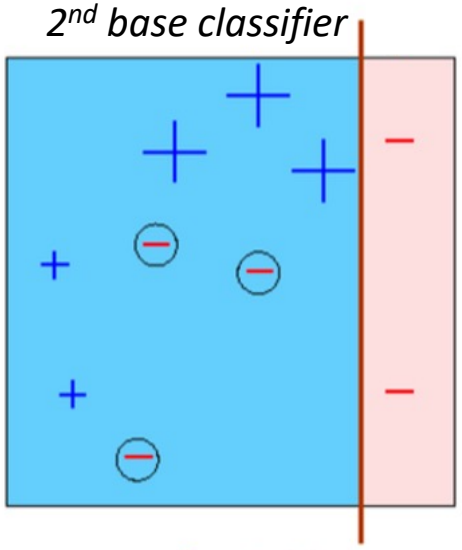
- A binary classification example
  - Iteration 3

1st base classifier



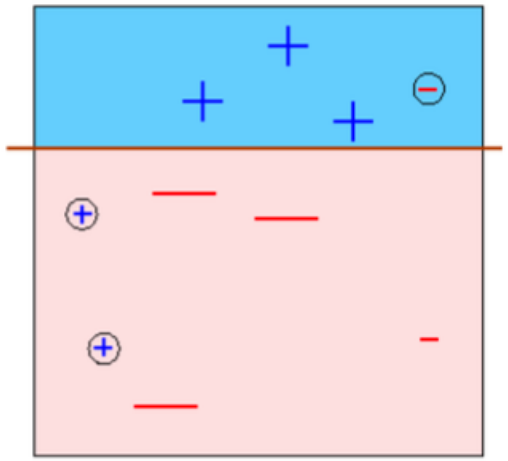
$\varepsilon^{(1)} = 0.30$
$\alpha^{(1)} = 0.42$

2nd base classifier



$\varepsilon^{(2)} = 0.21$
$\alpha^{(2)} = 0.65$

3rd base classifier

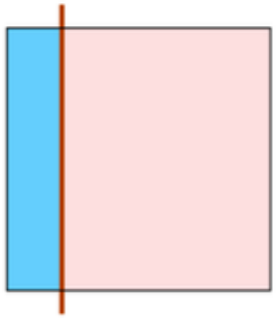


$\varepsilon^{(3)} = 0.14$
$\alpha^{(3)} = 0.92$

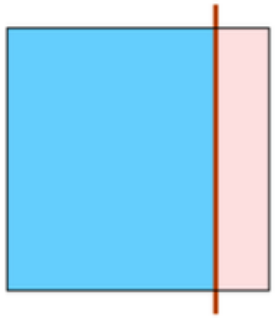
Adaptive Boosting

- A binary classification example
  - Final classifier
    - Weighted voting

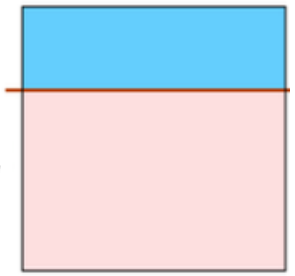
$$\hat{y}^3_{ensemble} = \mathbf{sign}\,(\,0.42$$



$$+0.65$$

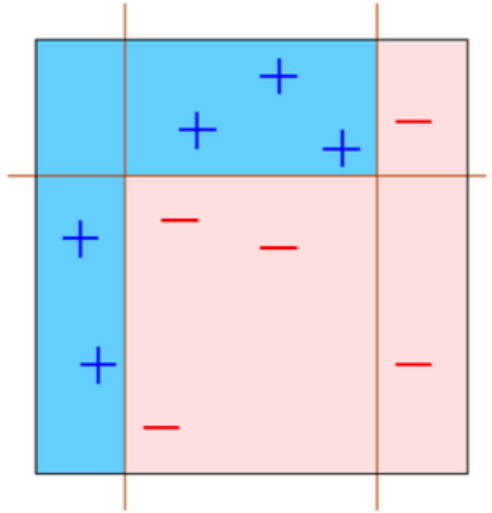


$$+0.92$$



$$)$$

$$=$$



Adaptive Boosting

- **AdaBoost procedure**
  1. Assign initial weights $w_i^1 = 1/n$ to all training data points $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$
  2. For iteration $b = 1$ to B:
     1. Train a weak learner $\hat{y}^{(b)}(\mathbf{x})$ on the weighted training data $\left\{\left(\mathbf{x}_i, y_i, w_i^b\right)\right\}_{i=1}^n$
     2. Calculate the **weight $\alpha^{(b)}$** of the weak learner $\hat{y}^{(b)}(\mathbf{x})$ based on its **error rate $\varepsilon^{(b)}$**
     3. Update the **weights $\left\{w_i^{b+1}\right\}_{i=1}^n$** from $\left\{w_i^b\right\}_{i=1}^n$
        1. Increase weights for all data points misestimated by $\hat{y}^{(b)}(\mathbf{x})$
        2. Decrease weights for all data points correctly estimated by $\hat{y}^{(b)}(\mathbf{x})$
  3. Aggregate the predictions of the B weaker learners $\left\{\hat{y}^{(1)}(\mathbf{x}), \ldots, \hat{y}^{(B)}(\mathbf{x})\right\}$ to obtain the prediction of the ensemble $\hat{y}_{\text{ensemble}}^B(\mathbf{x})$

Adaptive Boosting

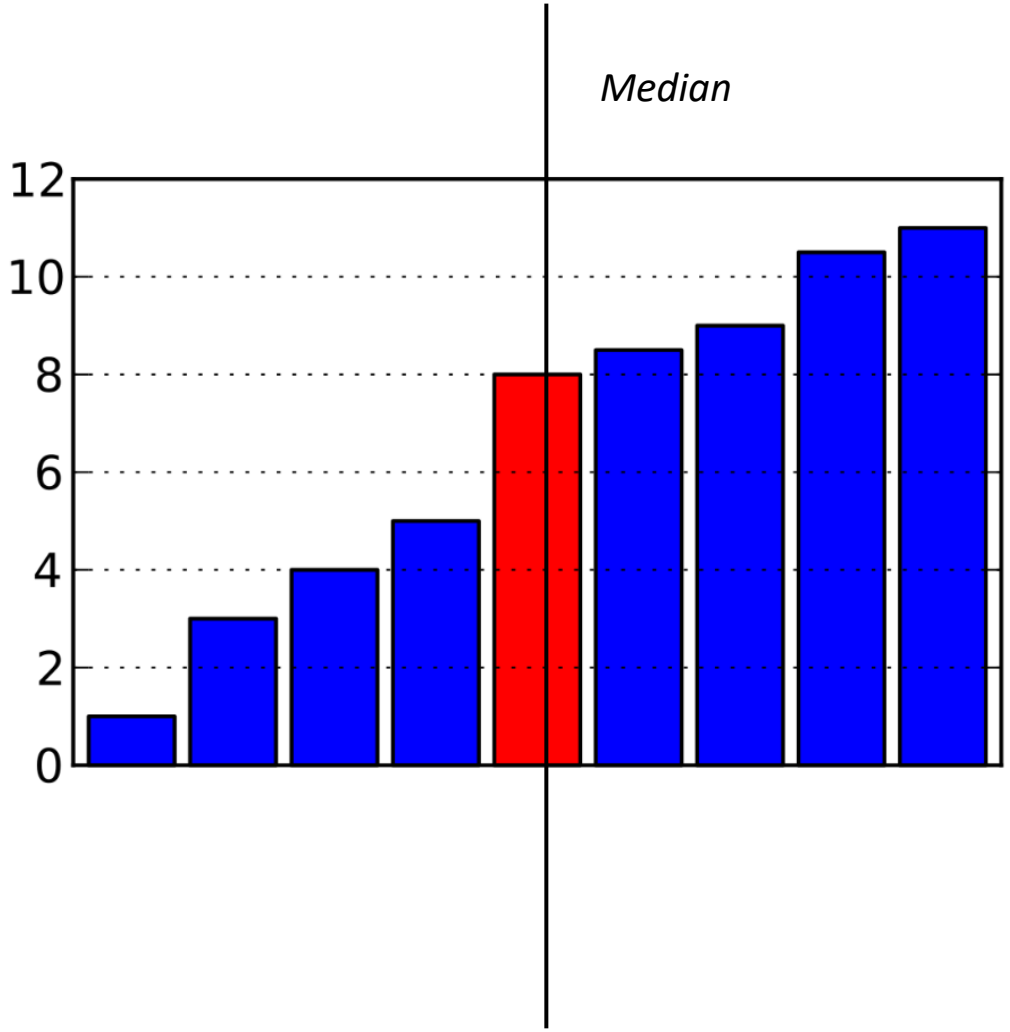
- **How to calculate error rate and update weights?**
  - For binary classification
    - **Error rate:** Compute based on whether the predicted class is correct or not
      - $\varepsilon^{(b)} = \sum_{i=1}^{n} w_i^b \times \begin{cases} 0 & \text{if } \hat{y}^{(b)}(\mathbf{x}_i) = y_i \\ 1 & \text{if } \hat{y}^{(b)}(\mathbf{x}_i) \neq y_i \end{cases}$
    - **Weight of base estimator:** Small if error rate is large and vice versa
      - $\alpha^{(b)} = \frac{1}{2}\log\left[\left(1 - \varepsilon^{(b)}\right)/\varepsilon^{(b)}\right]$
    - **Weight of each instance**: greater if the predicted class is wrong and vice versa
      - $w_i^{b+1} = w_i^b \times \begin{cases} \mathrm{e}^{-\alpha^{(b)}} & \text{if } \hat{y}^{(b)}(\mathbf{x}_i) = y_i \\ \mathrm{e}^{\alpha^{(b)}} & \text{if } \hat{y}^{(b)}(\mathbf{x}_i) \neq y_i \end{cases}$

Adaptive Boosting

- **How to calculate error rate and update weights?**
  - For binary classification
    - Yoav Freund and Robert E Schapire. “A decision-theoretic generalization of on-line learning and an application to boosting”. In: Journal of computer and system sciences 55.1 (1997), pp. 119–139.
  - For multi-class classification
    - Trevor Hastie et al. “Multi-class adaboost”. In: Statistics and its Interface 2.3 (2009), pp. 349–360.
  - For regression
    - H. Drucker, “Improving Regressors using Boosting Techniques”, 1997.

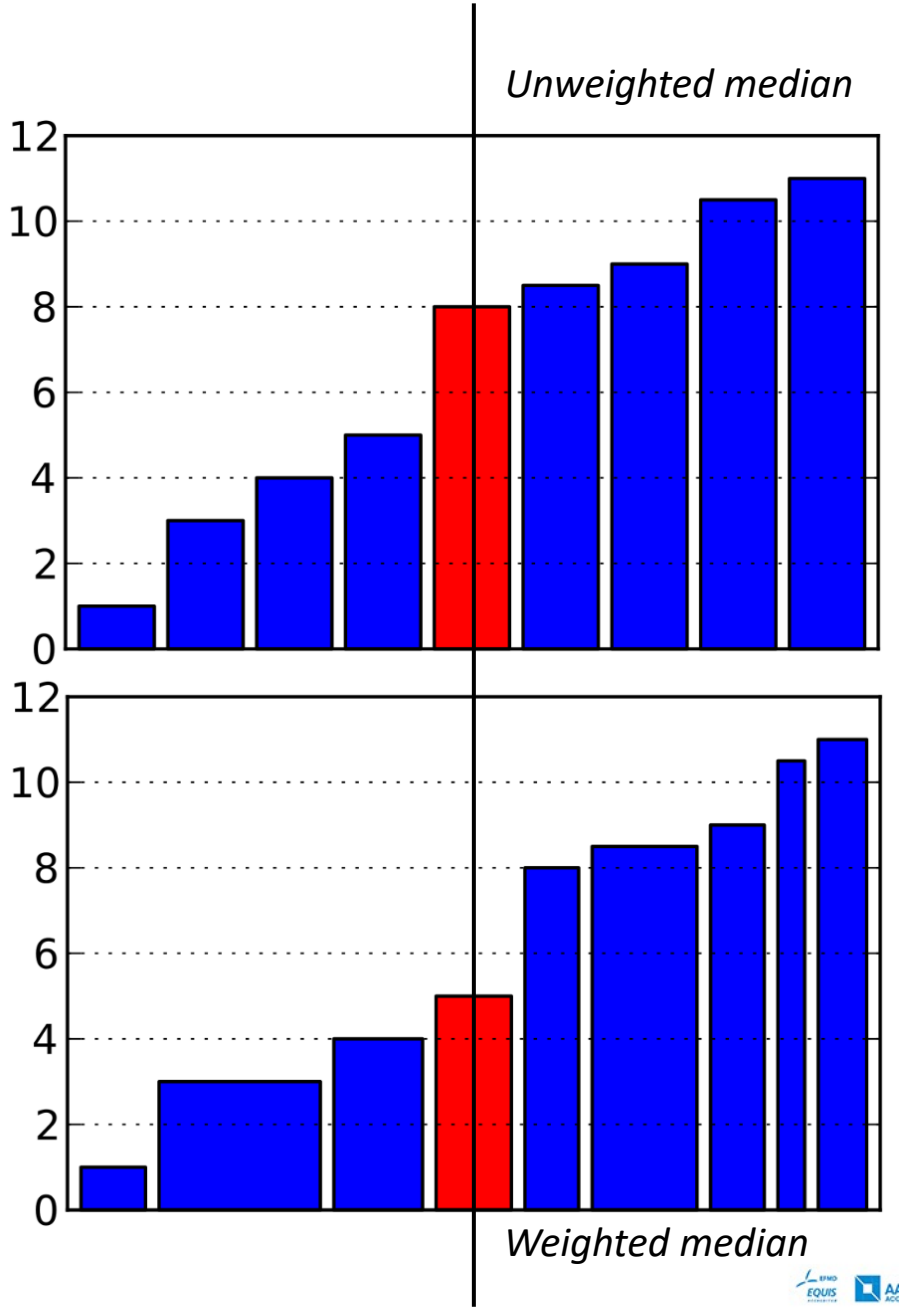
Adaptive Boosting

- **How to calculate the final prediction?**
  - For classification
    - Weighted voting
  - For regression
    - Weighted median
      - Arrange the predicted values from small to large
      - Visualize them as a bar chart, each bar is a prediction
      - The red bar represent the median



Adaptive Boosting

- **How to calculate the final prediction?**
  - For regression
    - Weighted median
      - We can regard the width of the bar as the weight of the prediction
      - Applying weights to predictions, then the median changes from 8 to 5
      - Here, weight is calculated based on error rate, thus reflects the reliability of the prediction.
      - The sum of weights (reliability) is equal on both sides.



Adaptive Boosting

- Implementation in scikit-learn
  - sklearn.ensemble.AdaBoostClassifier
  - sklearn.ensemble.AdaBoostRegressor

Adaptive Boosting

- Key hyper-parameter
  - max_depth of base trees
    - default = 1
    - or small number
  - **n_estimators : *int, default=50***

    The maximum number of estimators at which boosting is terminated. In case of perfect fit, the learning procedure is stopped early. Values must be in the range `[1, inf)`.

  - **learning_rate : *float, default=1.0***

    Weight applied to each classifier at each boosting iteration. A higher learning rate increases the contribution of each classifier. There is a trade-off between the `learning_rate` and `n_estimators` parameters. Values must be in the range `(0.0, inf)`.

Outline

- Introduction to Boosting
- Adaptive Boosting
- **Gradient Boosting**
- Introduction to Stacking

Gradient Boosting

- A variant of boosting method
- Similar to AdaBoost
  - Gradient Boosting is a **sequential** algorithm
- Unlike AdaBoost
  - Gradient Boosting **doesn't reweight the training data**

Gradient Boosting

- Key features
  - A sequence of models, **each fixing the remaining mistakes of the previous ones**
    - Each iteration, the task is to predict the residual error of the ensemble directly
  - **Additive model**, predictions at iteration B are **sum** of base model predictions
    - **Learning rate** (or weight) α are used to calculate the sum
    - $\hat{y}^{B}_{\text{ensemble}}(\mathbf{x}_i) = \hat{y}^{(1)}(\mathbf{x}_i) + \sum_{b=1}^{B} \alpha\ \hat{y}^{(b)}(\mathbf{x}_i) = \hat{y}^{B-1}_{\text{ensemble}}(\mathbf{x}_i) + \alpha\ \hat{y}^{(B)}(\mathbf{x}_i)$
    - A small learning rate indicates the ensemble corrects the remaining mistakes slowly and conservatively

Gradient Boosting

- **Gradient boosting procedure (regression)**
  1. Train the 1st base model $\hat{y}^{(1)}(\mathbf{x})$ on the training data points $\{(\mathbf{x}_i, y_i)\}_{i=1}^{n}$
  2. For iteration $b = 2$ to B:
     1. Calculate the residual of the current ensemble prediction for every $(\mathbf{x}_i, y_i)$
        1. Residual $r_i = y_i - \hat{y}_{ensemble}^{b-1}(\mathbf{x}_i)$
     2. Train a weak learner $\hat{y}^{(b)}(\mathbf{x})$ to predict the remaining residuals $\{(\mathbf{x}_i, r_i)\}_{i=1}^{n}$
     3. Update the ensemble prediction for every $(\mathbf{x}_i, y_i)$
        1. Ensemble prediction $\hat{y}_{ensemble}^{b}(\mathbf{x}_i) = \hat{y}_{ensemble}^{b-1}(\mathbf{x}_i) + \alpha\, \hat{y}^{(b)}(\mathbf{x}_i)$
  3. Early stopping (optional)
     1. Stop when performance on the validation set doesn't improve for a pre-defined number of iterations

Gradient Boosting

- A regression example

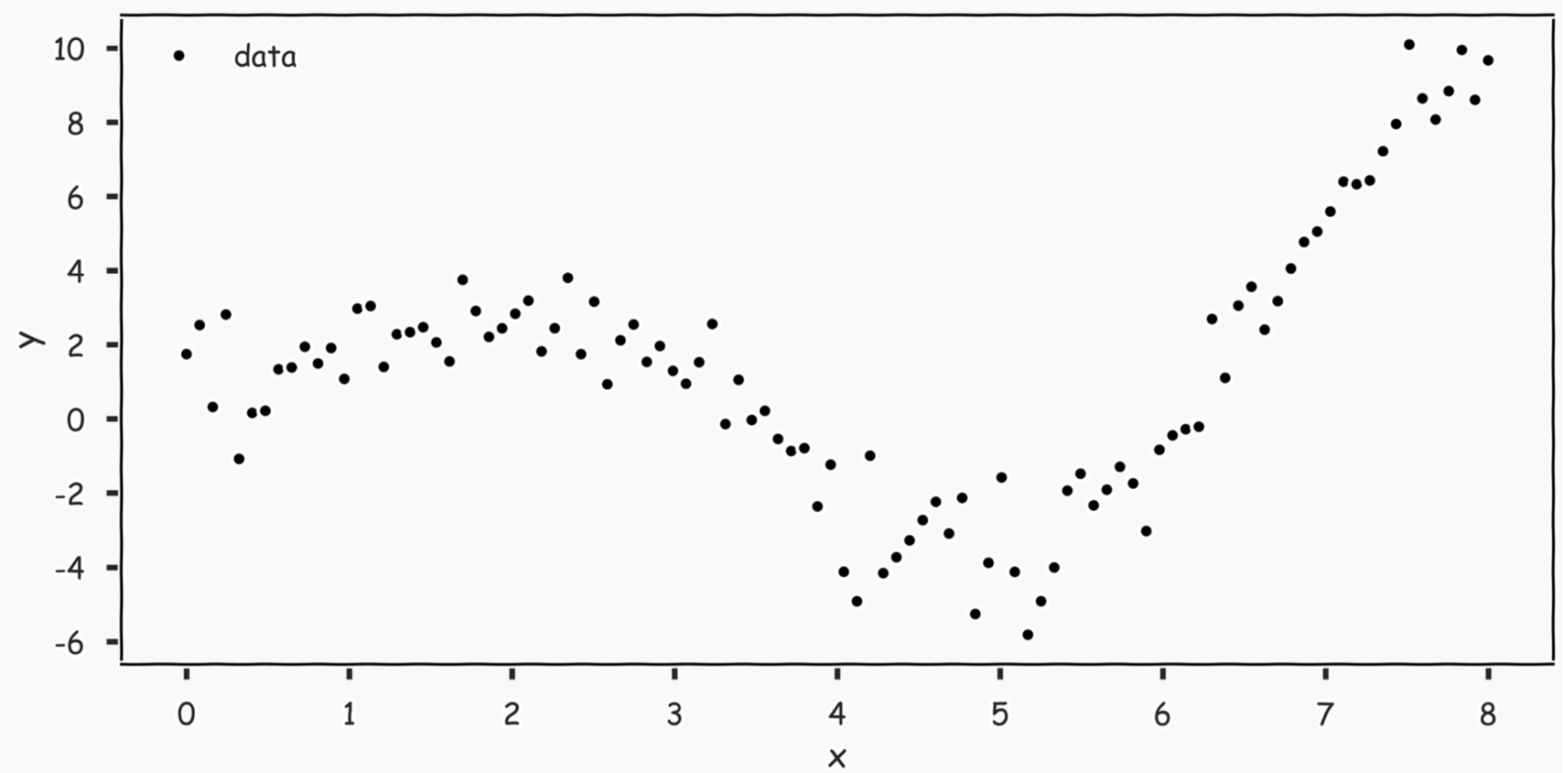


Image source: Slides from Pavlos Protopapas & Kevin Rader @ IACS

Gradient Boosting

- A regression example

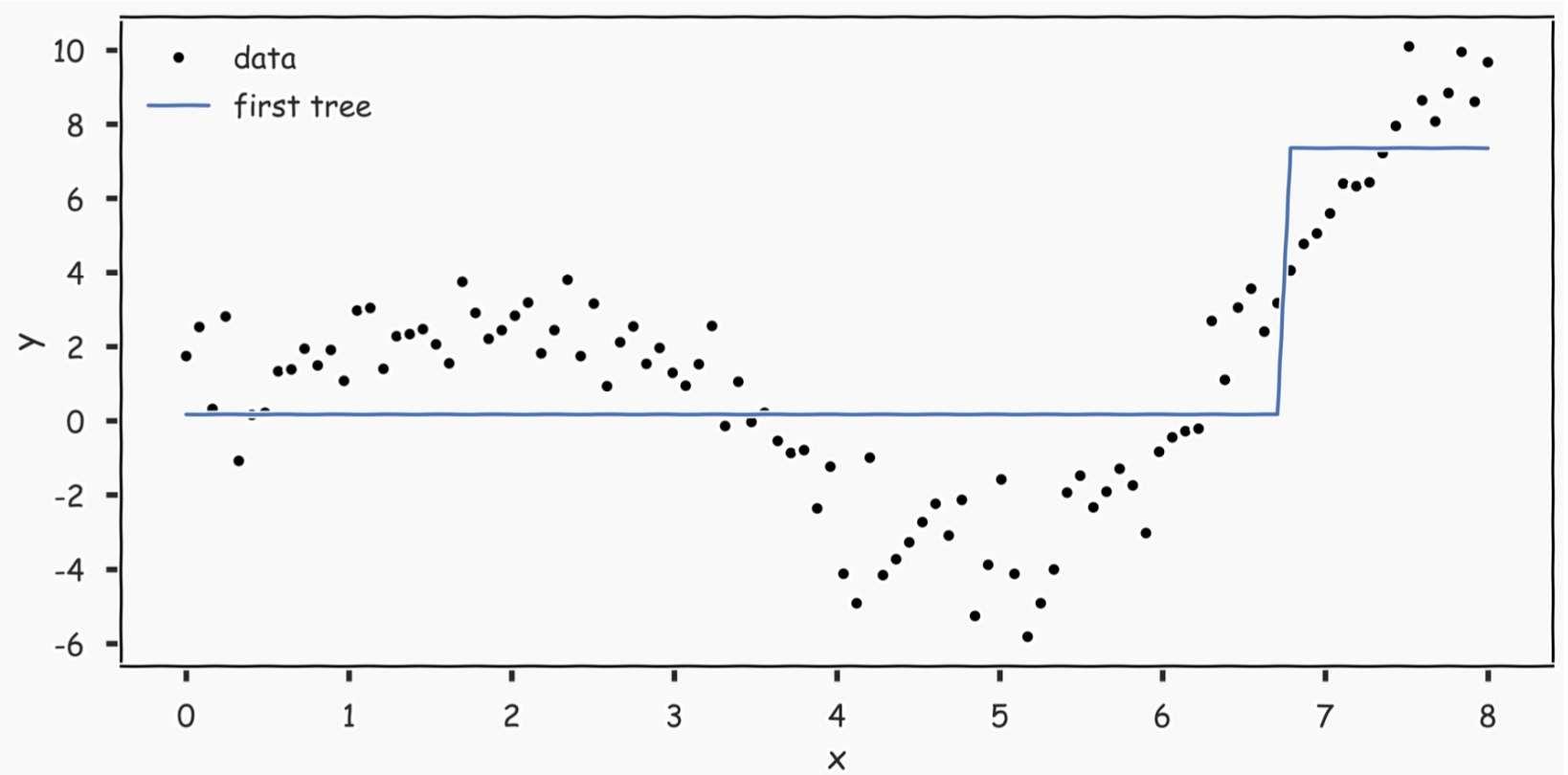


Image source: Slides from Pavlos Protopapas & Kevin Rader @ IACS

Gradient Boosting

- A regression example

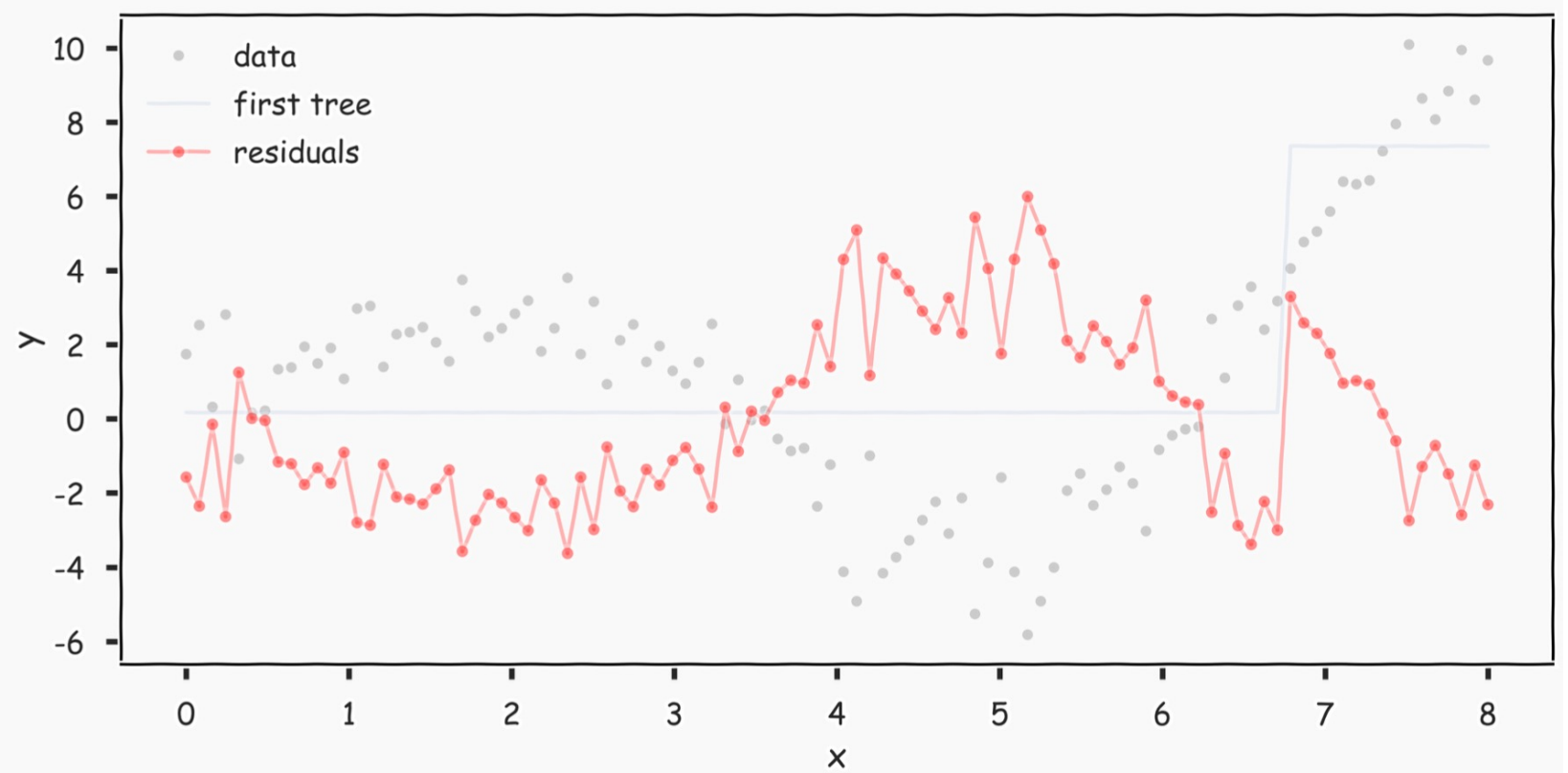


Image source: Slides from Pavlos Protopapas & Kevin Rader @ IACS

Gradient Boosting

- A regression example

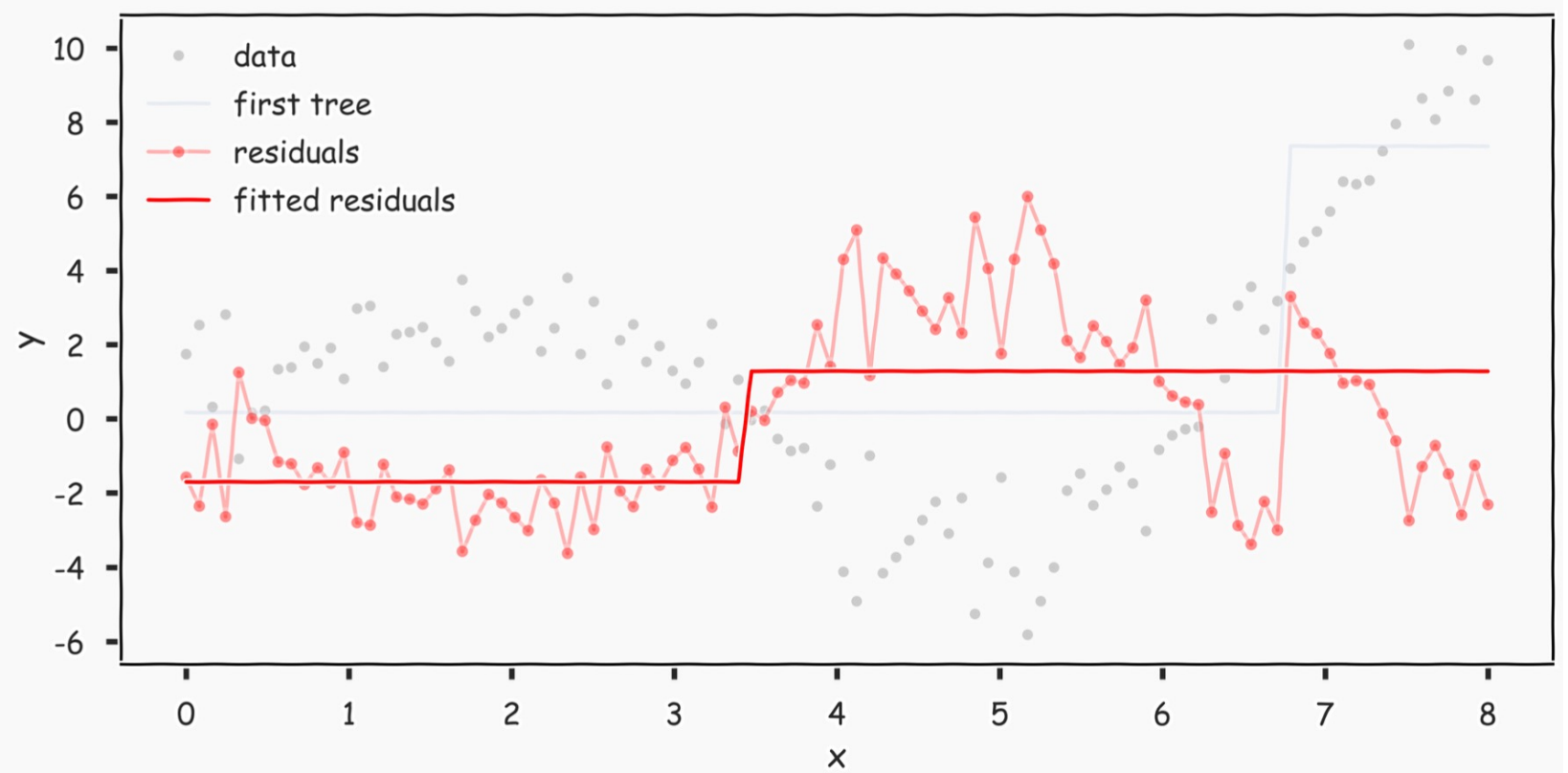


Image source: Slides from Pavlos Protopapas & Kevin Rader @ IACS

Gradient Boosting

- A regression example

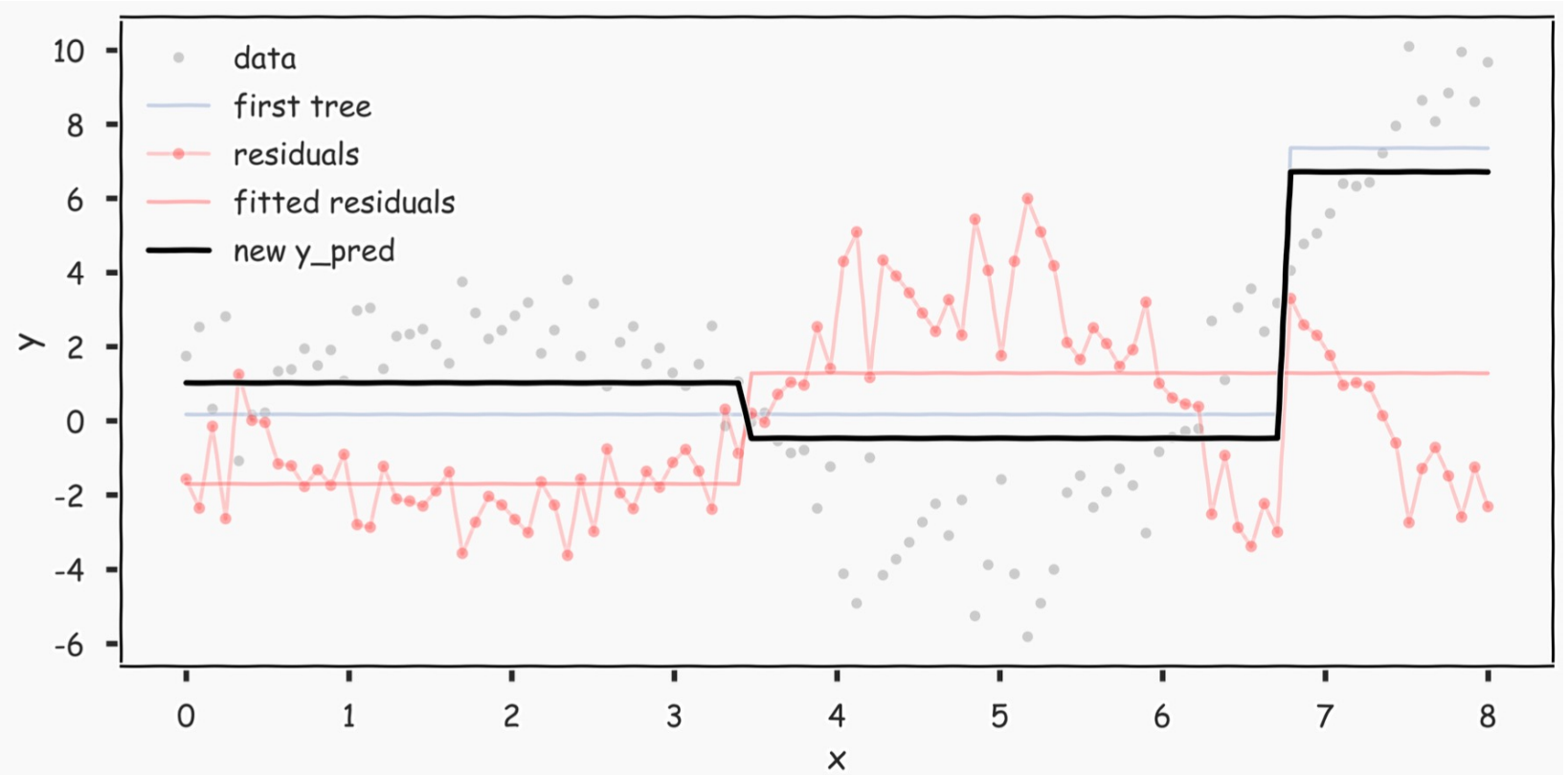


Image source: Slides from Pavlos Protopapas & Kevin Rader @ IACS

Gradient Boosting

- A regression example

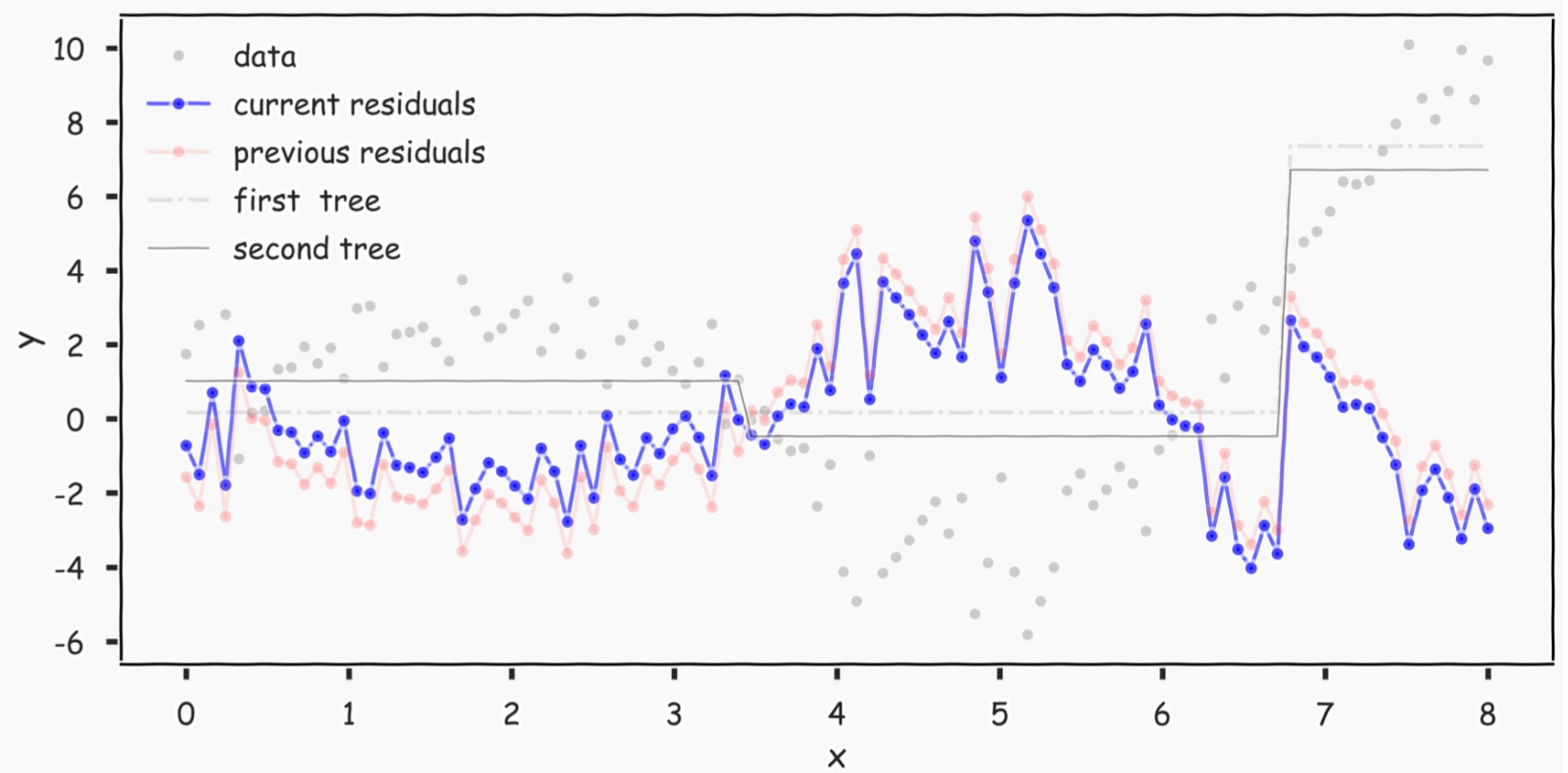


Image source: Slides from Pavlos Protopapas & Kevin Rader @ IACS

Gradient Boosting

- **For regression**
  - The 1st base model $\hat{y}^{(1)}(\mathbf{x})$ predict the target variable $\{y_i\}_{i=1}^{n}$
  - The residual is the difference between the target value and predicted value
    - Residual $r_i = y_i - \hat{y}_{ensemble}^{b-1}(\mathbf{x}_i)$
- **For classification**
  - The 1st base model $\hat{y}^{(1)}(\mathbf{x})$ predict the probability $\{p_i\}_{i=1}^{n}$ of the positive class
  - The residual is the difference between 1 and predicted probability
    - Residual $r_i = 1 - p_i$

Gradient Boosting

- Implementation in scikit-learn
  - sklearn.ensemble.GradientBoostingClassifier
  - sklearn.ensemble.GradientBoostingRegressor
  - sklearn.ensemble.HistGradientBoostingClassifier
  - sklearn.ensemble.HistGradientBoostingRegressor
- The most popular implementation is XGBoost
  - Faster version of gradient boosting
  - The XGBoost is compatible with scikit-learn
  - Official documentation

Gradient Boosting

- Key hyper-parameter

learning_rate : *float, default=0.1*

Learning rate shrinks the contribution of each tree by `learning_rate`. There is a trade-off between learning_rate and n_estimators. Values must be in the range `[0.0, inf)`.

n_estimators : *int, default=100*

The number of boosting stages to perform. Gradient boosting is fairly robust to over-fitting so a large number usually results in better performance. Values must be in the range `[1, inf)`.

max_depth : *int or None, default=3*

Maximum depth of the individual regression estimators. The maximum depth limits the number of nodes in the tree. Tune this parameter for best performance; the best value depends on the interaction of the input variables. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples. If int, values must be in the range `[1, inf)`.

Gradient Boosting

- Key hyper-parameter

n_iter_no_change : *int, default=None*

`n_iter_no_change` is used to decide if early stopping will be used to terminate training when validation score is not improving. By default it is set to None to disable early stopping. If set to a number, it will set aside `validation_fraction` size of the training data as validation and terminate training when validation score is not improving in all of the previous `n_iter_no_change` numbers of iterations. Values must be in the range `[1, inf)`. See Early stopping in Gradient Boosting.

validation_fraction : *float, default=0.1*

The proportion of training data to set aside as validation set for early stopping. Values must be in the range `(0.0, 1.0)`. Only used if `n_iter_no_change` is set to an integer.

Outline

- Introduction to Boosting
- Adaptive Boosting
- Gradient Boosting
- **Introduction to Stacking**

Introduction to Stacking

- Two questions to answer when designing ensemble learning methods
  1. **How to generate multiple base models (estimators)?**
  2. **How to integrate / combine their predictions to make final prediction?**
- Different decisions lead to different types of ensemble learning methods
  - Bagging: Bootstrap sampling training sets, majority voting / average
  - Boosting: Reweighting training samples, weighted voting / weighted median
  - **Stacking: Same training set + different algorithms, integrate by a meta-learner**

Introduction to Stacking

- Stacking
  1. Choose M different algorithms, train M different base learners $\{\hat{y}^j(\mathbf{x})\}_{j=1}^n$ on the training data set $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$
  2. Form a new training data set using the base predictions
     - New training data set $\{(\mathbf{x}'_i, y_i)\}_{i=1}^n$
     - $\mathbf{x}'_i = \{\hat{y}_i^1(\mathbf{x}_i), \dots, \hat{y}_i^M(\mathbf{x}_i)\}$
  - Train a meta-learner $\hat{Y}(\mathbf{x})$ on the new training data set $\{(\mathbf{x}'_i, y_i)\}_{i=1}^n$

Introduction to Stacking

- Stacking procedure

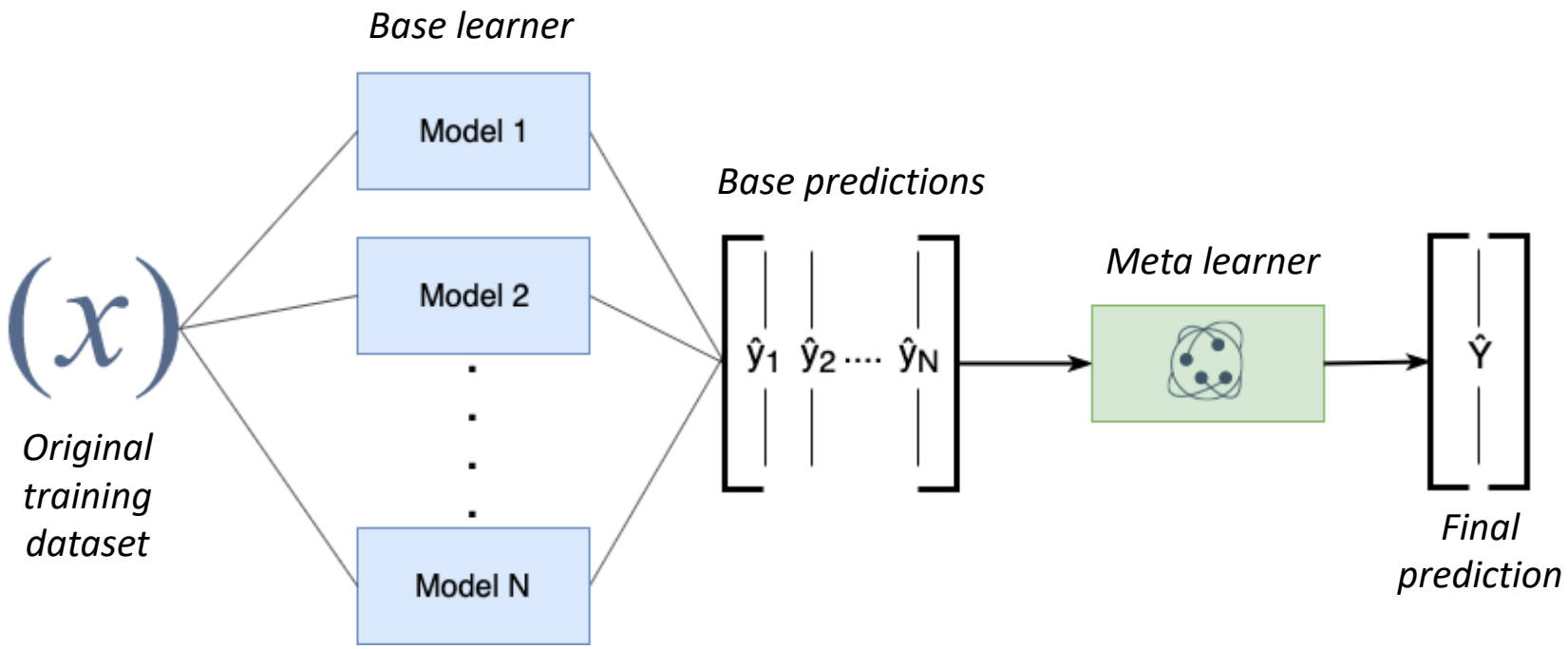


Image source: https://ml-course.github.io/master/notebooks/05%20-%20Ensemble%20Learning.html#stacking

Gradient Boosting

- Implementation in scikit-learn
  - sklearn.ensemble.StackingClassifier
    - Base learners can be any classification algorithm
    - Default meta learner is logistic regression
  - sklearn.ensemble.StackingRegressor
    - Base learners can by any regression algorithm
    - Default meta learner is RidgeCV
      - Ridge regression with cross-validation

Hands-on Exercise

- Exercise 02 Ensemble Learning II