Combining Multiple Learners Part b

Ethem Chp. 15
Hastie Chp. 8
Haykin Chp. 7, pp. 351-370

Overview

- Introduction
  - Rationale
- Combination Methods
  - Static Structures
    - Ensemble averaging (Voting…)
    - Bagging
    - Boosting
    - Error Correcting Output Codes
  - Dynamic structures
    - Mixture of Experts
    - Hierarchical Mixture of Experts

Ensemble Methods > Boosting

- In Bagging, generating *complementary* base-learners is left to chance and unstability of the learning method

- Kearns and Valient (1988) posed the question, **"Can a set of weak learners create a single strong learner?"**
  - Weak learner: the learner is required to perform only **slightly better than random**
  - Strong learner: arbitrary accuracy with high probability

- Schapire (1990) and Freund (1991) gave the first constructive proof.
  - Try to generate complementary weak base-learners by training the next learner on the mistakes of the previous ones
  - Convert a weak learning model to a strong learning model by "boosting" it

Boosting

- Motivation:
  - In Bagging, generating *complementary* base-learners is left to chance and **unstability** of the learning method

- Idea:
  - Focus later classifiers on examples that were **misclassified** by earlier classifiers
  - Weight the predictions of the classifiers with their error

Illustration of Bagging and Boosting

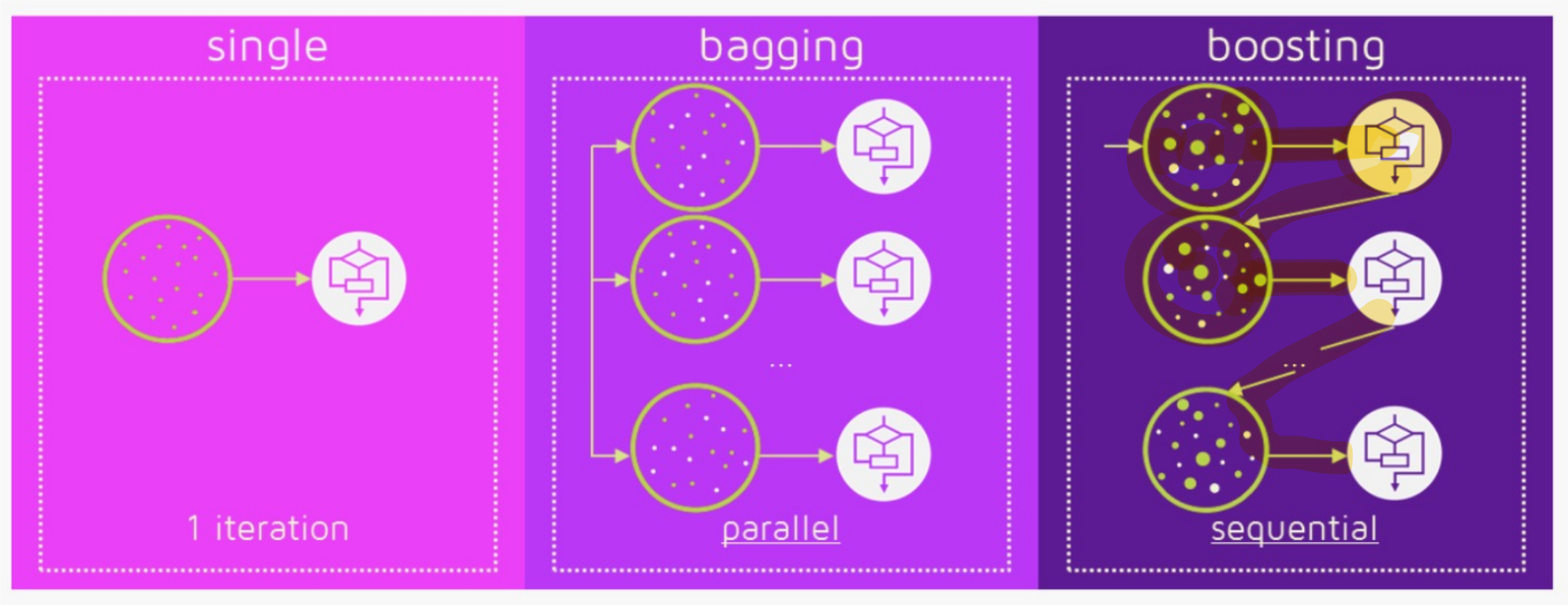


Fig 2. Bagging (independent models) & Boosting (sequential models). **Reference:** **https://quantdare.com/what-is-the-difference-between-bagging-and-boosting/**

Boosting – General Approach

take a training set D, of size N

do M times // M base learners

train a network on D

find all examples in D that the network gets wrong

emphasize those patterns, de-emphasize the others, in a new dataset D2

set D=D2

loop

output is average/vote from all machines trained

General method *– different types in literature, by filtering, sub-sampling or re-weighting, see Haykin Ch.7 for details if you are interested.*

AdaBoost (ADAptive BOOSTing)

- Modify the probabilities of drawing an instance x^t for a classifier j, based on the probability of error of ε_j
  - For the next classifier:
    - if pattern x^t is NOT correctly classified, its probability of being selected increases

$$\beta_j = \frac{\varepsilon}{1-\varepsilon_j}$$

$$p_{j+1}^t \leftarrow \beta_j p_j^t$$

AdaBoost (ADAptive BOOSTing)

- All learners must have error less than ½
  - if not, stop training
  - otherwise the problem gets more difficult for next classifier)
- Final prediction is the weighted output of all L base classifiers

$$y_i = \sum_{j=1}^{L} \left(\log \frac{1}{\beta_j} \right) d_{ji}(x)$$

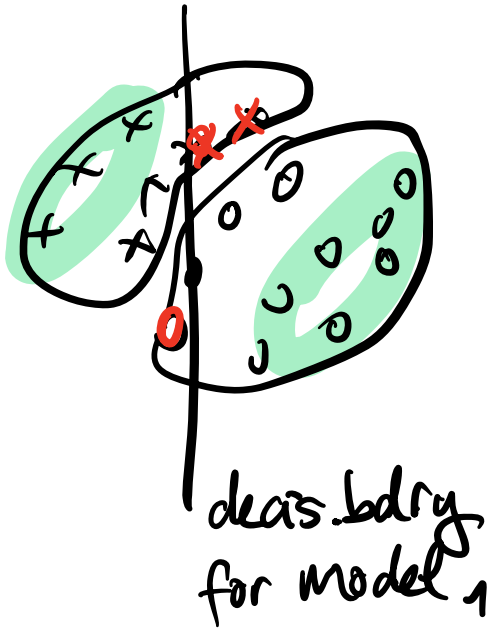
AdaBoost

Generate a sequence of base-learners **each focusing on previous one's errors**

(Freund and Schapire, 1996)

βt is the ratio of the weak learner's error rate to its accuracy.

Correct if $y^t = r^t$



Training:

For all $\{x^t, r^t\}_{t=1}^N \in \mathcal{X}$, initialize $p_1^t = 1/N$

For all base-learners $j = 1, \ldots, L$

Randomly draw $\mathcal{X}_j$ from $\mathcal{X}$ with probabilities p_j^t

Train d_j using $\mathcal{X}_j$

For each (x^t, r^t), calculate $y_j^t \leftarrow d_j(x^t)$

Calculate error rate: $\epsilon_j \leftarrow \sum_t p_j^t \cdot 1(y_j^t \neq r^t)$

1 if pred ≠ target

If $\epsilon_j > 1/2$, then $L \leftarrow j - 1$; stop

$\beta_j \leftarrow \epsilon_j / (1 - \epsilon_j)$

For each (x^t, r^t), decrease probabilities if correct:

If $y_j^t = r^t$ $p_{j+1}^t \leftarrow \beta_j p_j^t$ Else $p_{j+1}^t \leftarrow p_j^t$

Normalize probabilities:

$$Z_j \leftarrow \sum_t p_{j+1}^t; \quad p_{j+1}^t \leftarrow p_{j+1}^t / Z_j$$

Testing:

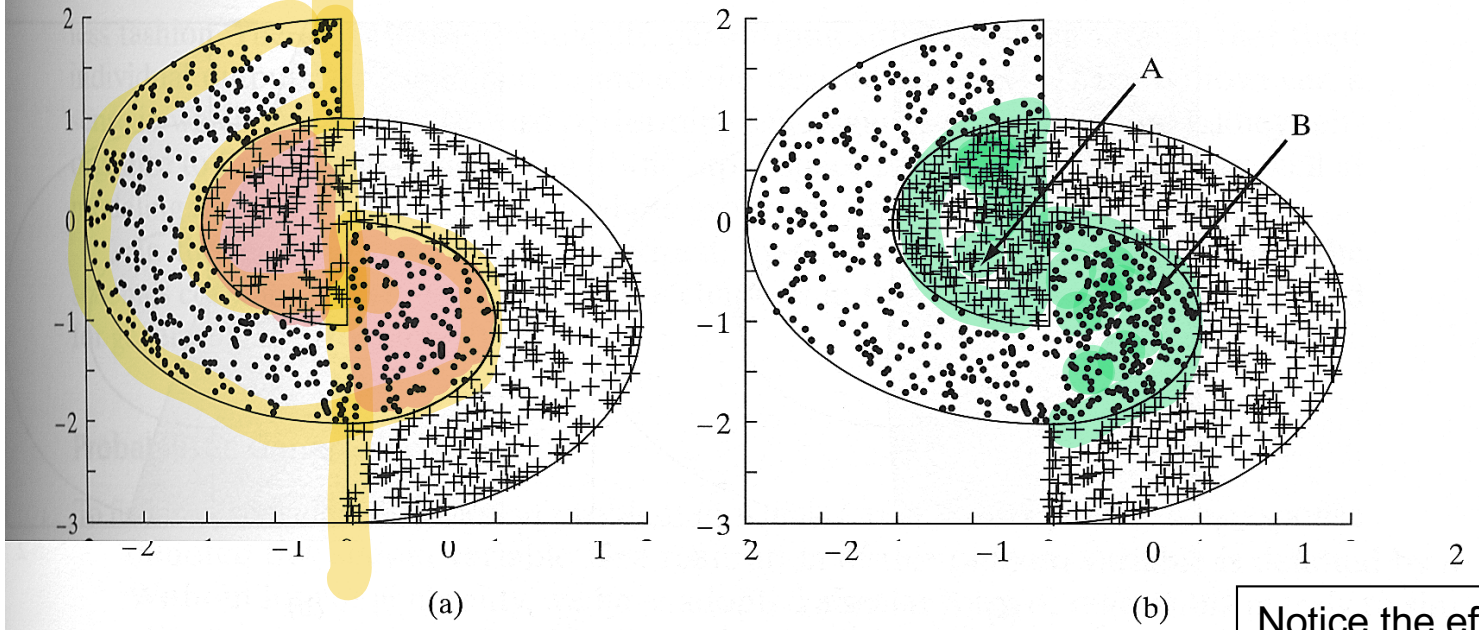
Given x, calculate $d_j(x), j = 1, \ldots, L$

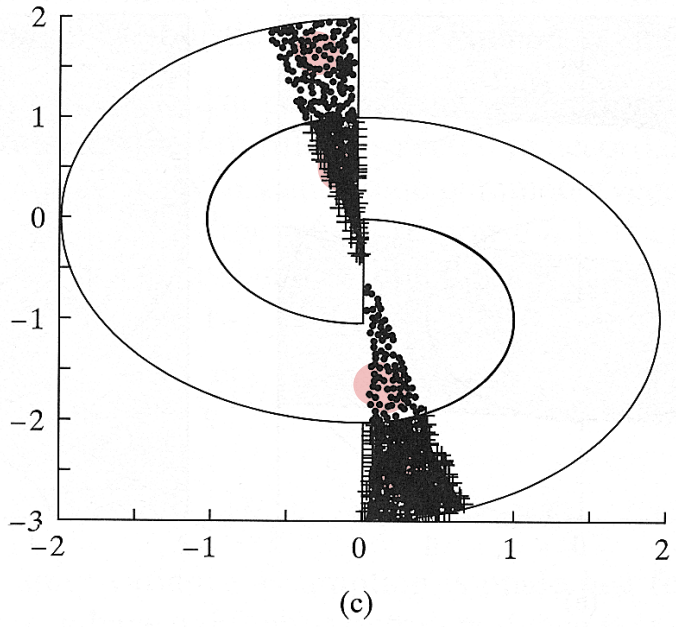
Calculate class outputs, $i = 1, \ldots, K$:

$$y_i = \sum_{j=1}^{L} \left(\log \frac{1}{\beta_j} \right) d_{ji}(x)$$

No need to memorize the algorithm, but you should be able to explain each line of code if code is given.

From Haykin





Notice the effect of emphasizing the error zone of the 1st classifier

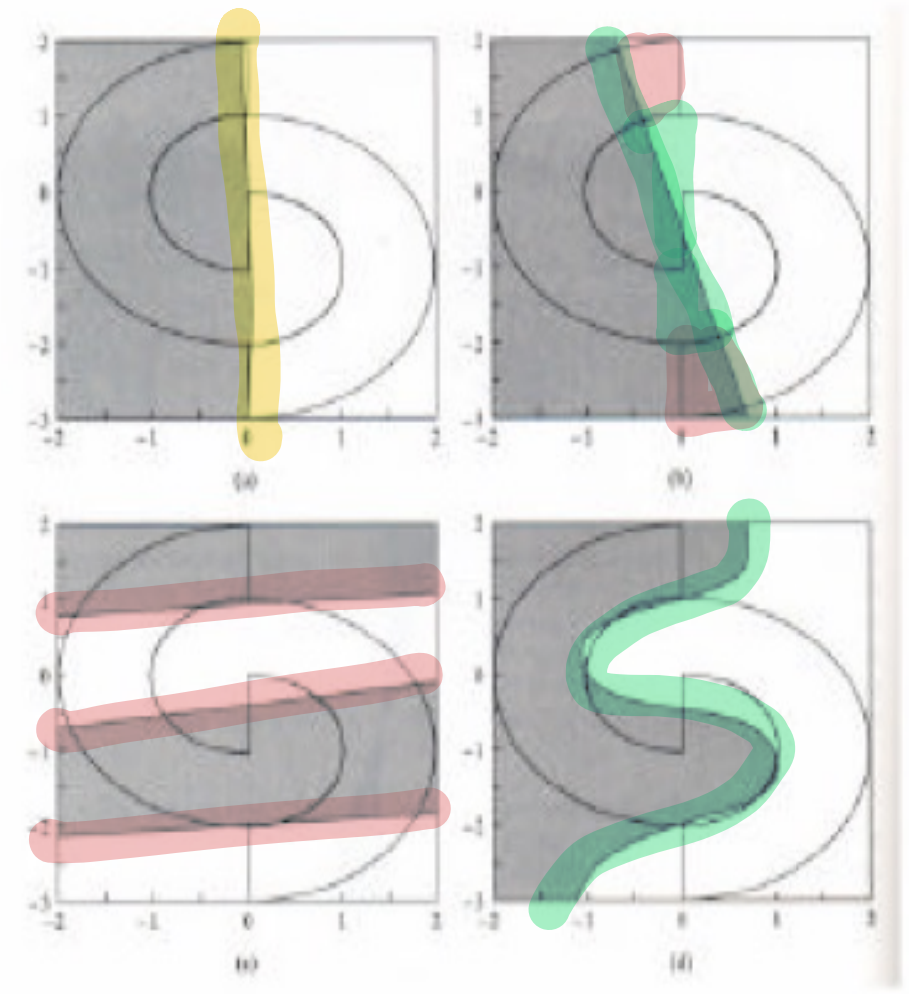
FIGURE 7.6 Scatter plots for expert training in computer experiment on boosting: (a) Expert 1. (b) Expert 2. (c) Expert 3.

Three 2-5-2 MLP networks

Accuracies:

Expert 1 : 75.15 percent
Expert 2 : 71.44 percent
Expert 3 : 68.90 percent

Committee Machine:
91.79% correct



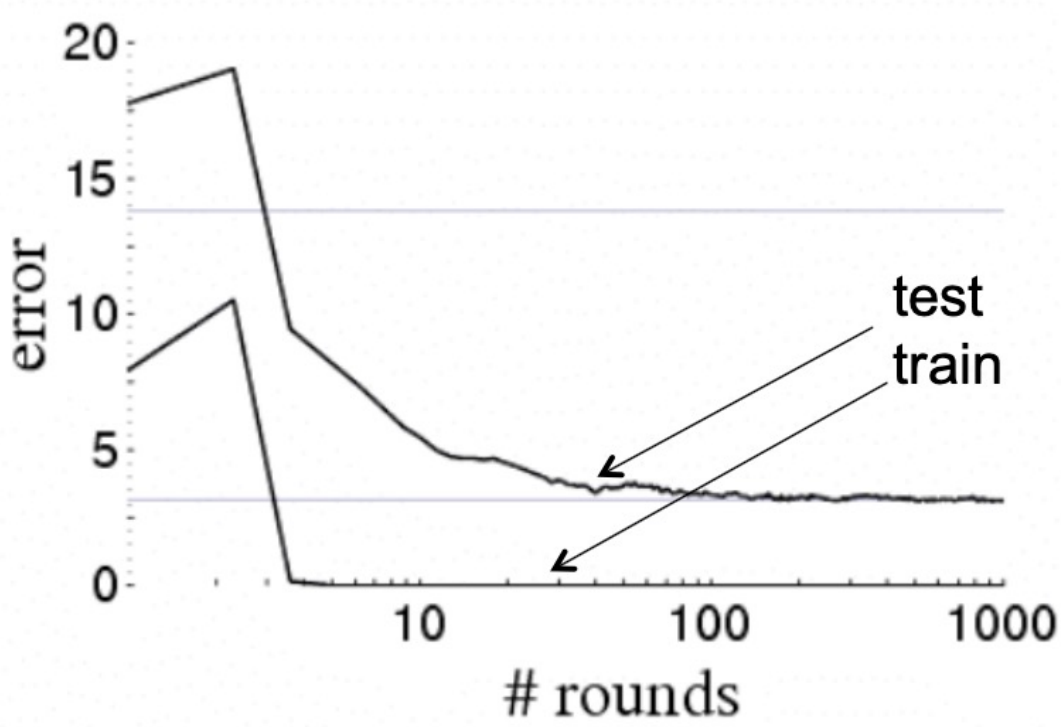
(c)

FIGURE 7.7 Decision boundaries formed by the differen experiment. (a) Expert 1. (b) Expert 2. (c) Expert 3. (d) Entire c

Adaboost

- Training error falls in each boosting iteration
- Generalization error also *tends* to fall
  - Improved generalization performance over 22 benchmark problems, equal accuracy in one, worse accuracy in 4 problems [Shapire 1996].
  - Shapire et al. explain the success of AdaBoost due to its property of increasing the margin, with the analysis involving the confidence of the individual classifiers [Shapire 1998].

- Test set error decreases even after training error is zero



Overview

- Introduction
  - Rationale
- Combination Methods
  - Static Structures
    - Ensemble averaging (Voting)
    - Bagging
    - Boosting
    - Error Correcting Output Codes
  - Dynamic structures
    - Mixture of Experts
    - Cascading
    - Stacking

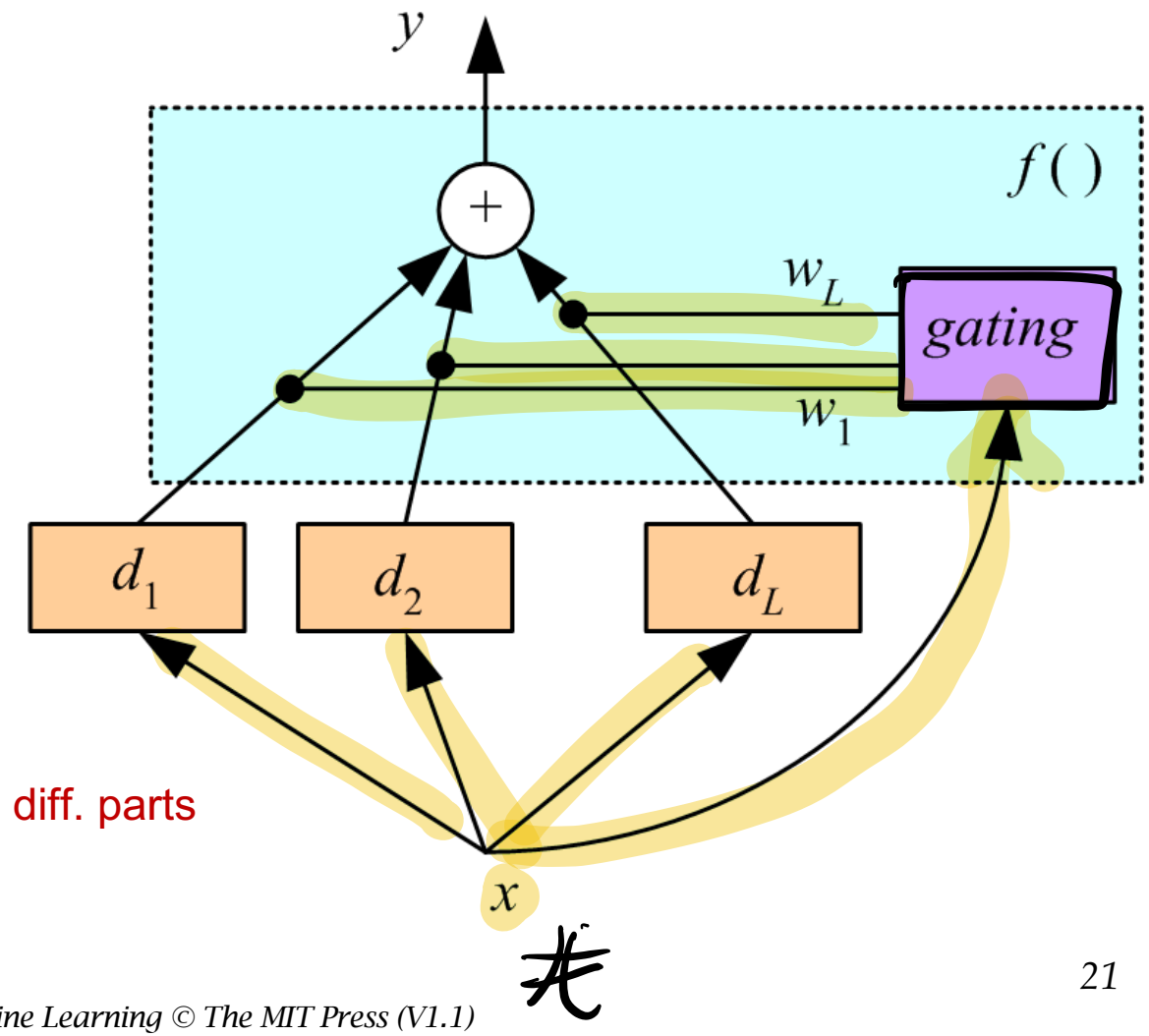
Dynamic Methods > Mixtures of Experts

Voting where weights are input-dependent (gating) – not constant

$$y = \sum_{j=1}^{L} w_j d_j$$

(Jacobs et al., 1991)

In general, experts or gating can be non-linear



Base learners become experts in diff. parts of the input space

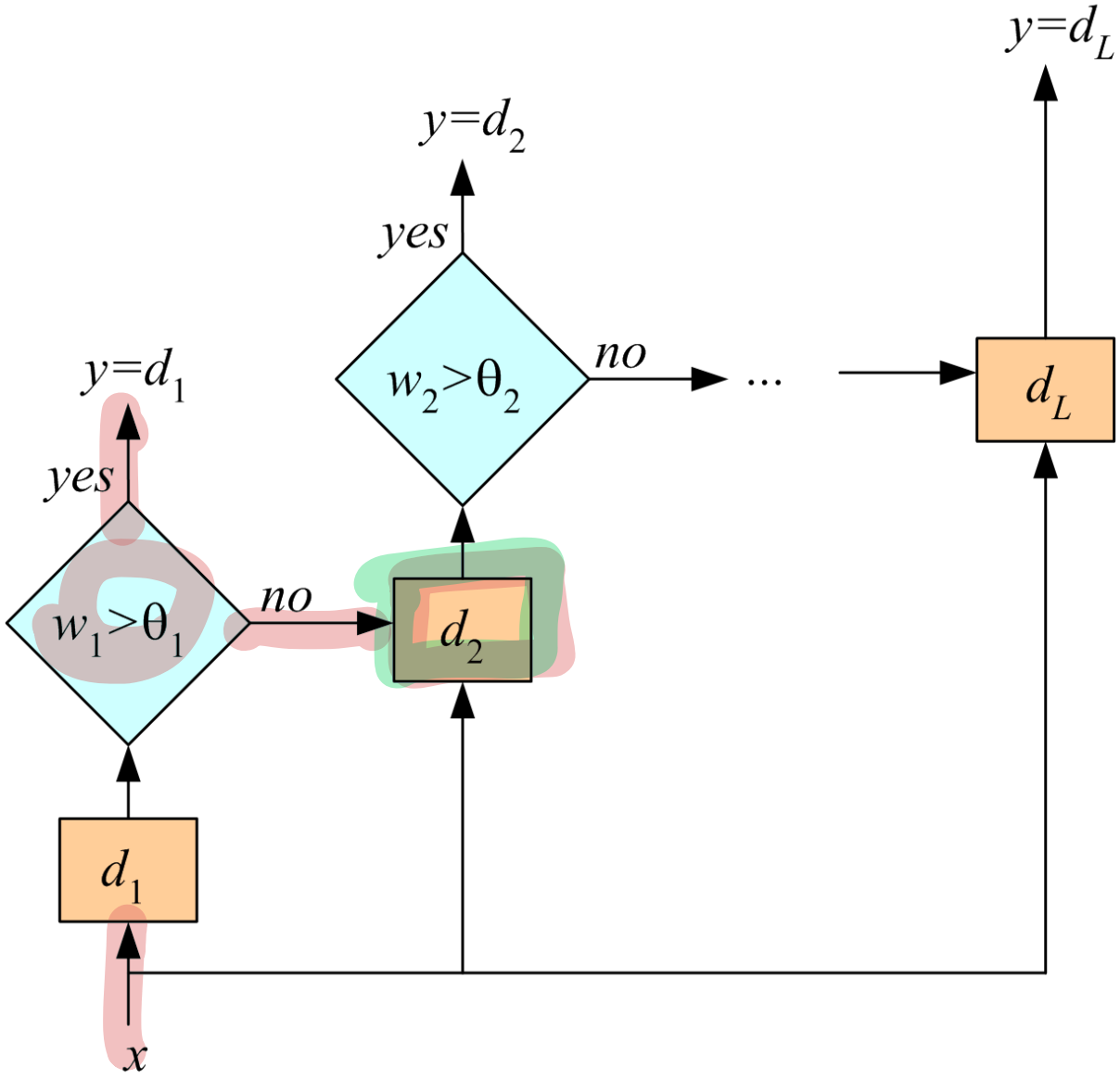
Dynamic Methods > Cascading

Cascade learners in order of complexity

Use d_j only if preceding ones are not confident

Training must be done on samples for which the previous learner is not confident

Note the difference compared to boosting



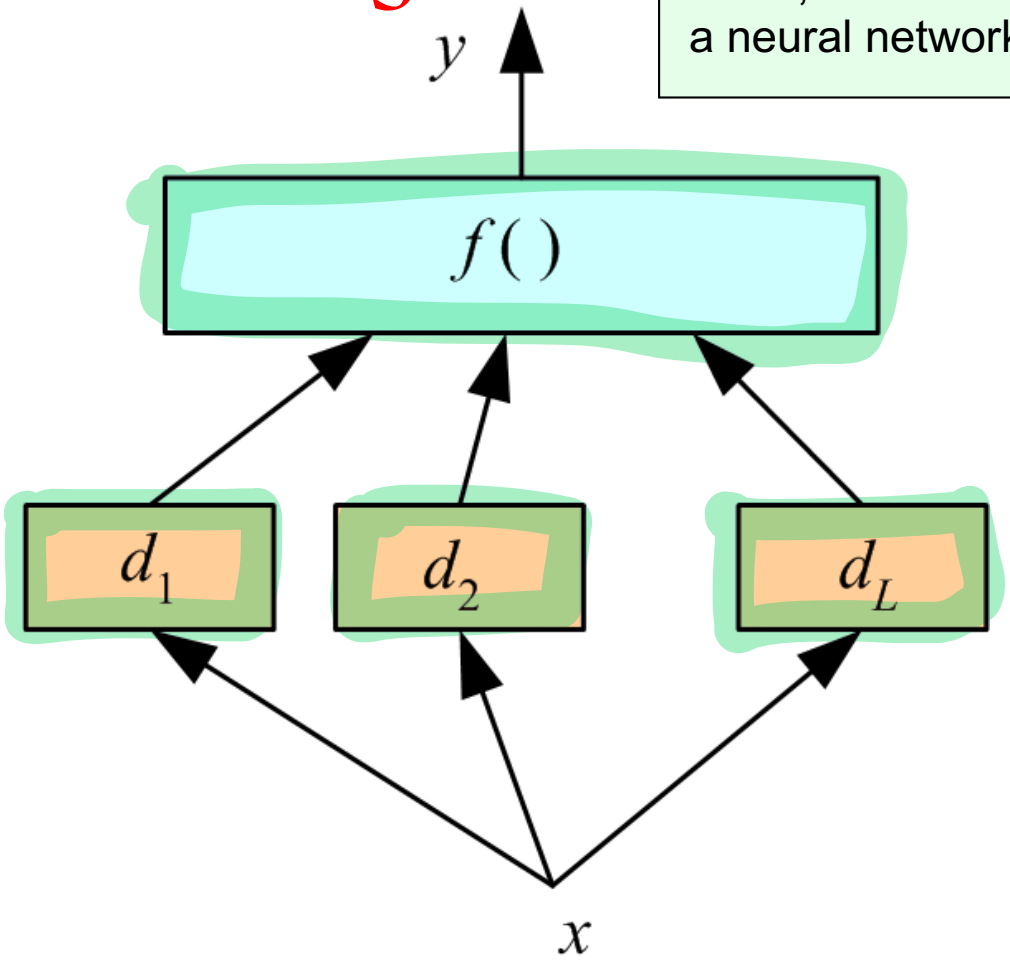
Dynamic Methods > Cascading

- Cascading assumes that the classes can be explained by small numbers of “rules” in increasing complexity, and a small set of exceptions not covered by the rules

Dynamic Methods > Stacking

f need not be linear, it can be a neural network

- Wolpert 1992



We cannot train f() on the training data; combiner should learn how the base-learners make errors.

Leave-one-out or k-fold cross validation

Learners should be as different as possible, to complement each other ideally using different learning algorithms

Stacking

Attributes			Class
x_{11}	...	x_{1n_a}	t
x_{21}	...	x_{2n_a}	f
...	...	...	...
x_{n_e1}	...	$x_{n_en_a}$	t

training set

C_1	C_2	...	C_{n_c}
t	t	...	f
f	t	...	t
...	...	...	...
f	f	...	t

predictions of the classfiers

C_1	C_2	...	C_{n_c}	Class
t	t	...	f	t
f	t	...	t	f
...	...	...	...	...
f	f	...	t	t

training set for stacking

- Uses *meta learner* instead of voting to combine predictions of base learners
- Predictions of base learners on a separate validation data are used as input for meta learner
- Base learners are usually different learning schemes

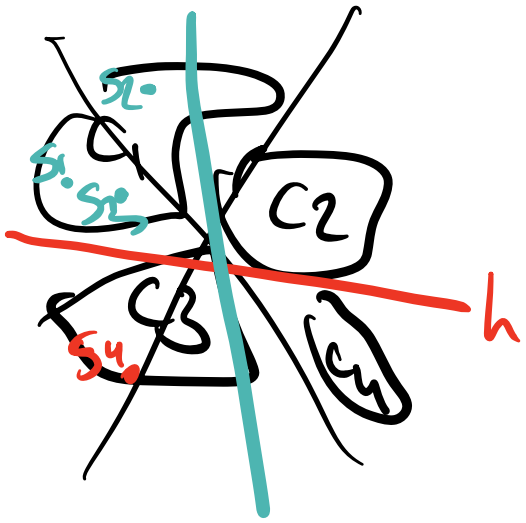
General Rules of Thumb

- Techniques manipulate either training data, architecture of learner, initial configuration, or learning algorithm. Training data is seen as most successful route; initial configuration is least successful.
- Base classifiers should exhibit low correlation
  - Understood well for regression, not so well for classification.
  - "Overproduce-and-choose" strategy for base classifiers (that perform well individually and make different mistakes)
- Uniform weighting is almost never optimal. Good strategy is to set the weighting for a component proportional to its error on a validation set.
- Unstable estimators (e.g. NNs, decision trees) benefit most from ensemble methods. Stable estimators like k-NN tend not to benefit.
- Boosting (particularly Adaboost) can be sensitive to noisy data
  - It will overfit on outliers or noisy labels.

Some Practical Advices

- If the classifier is unstable (high variance), then you may apply **bagging**.
  - Often highly flexible models (incl. DTs, NNs, NB with small/sparse data, …)
- If the classifier is stable and simple (high bias) then you may apply **boosting**.
  - Linear regression, logistic regression, SVM, small NNs…
- If you have many classes and a binary classifier then try error-correcting codes.

GRADIENT BOOSTED DECISION TREES



ECOCs -

	h_1	h_2	———	h_L
C_1	+1	+1	−1	+1 +1
C_2	+1	−1	−1	−1 −1
C_3	−1	+1		
C_4	−1	−1		

~ randomly filled matrix.

X X

+1	+1	−1	−1	+1

Train set for h_1:

$(x_1, +1)$

⋮

$(x_4, -1)$

→ each class has a codeword

– row in matrix

Gradient Boosted Decision Trees

- Gradient Boosted Decision Trees are found very successful for regression problems.

- The idea is to approximate the underlying function with successive, simple trees (weak learners), so as to improve the fit in each iteration.

Algorithm

Input: training set $\{(x_i, y_i)\}_{i=1}^{n}$, a differentiable loss function $L(y, F(x))$, number of iterations M.

Algorithm:

1. Initialize model with a constant value:

$$F_0(x) = \arg\min_{\gamma} \sum_{i=1}^{n} L(y_i, \gamma).$$

2. For m = 1 to M:

   1. Compute so-called *pseudo-residuals*:

$$r_{im} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)} \quad \text{for } i = 1, \ldots, n.$$

   2. Fit a base learner $h_m(x)$ to pseudo-residuals, i.e. train it using the training set $\{(x_i, r_{im})\}_{i=1}^{n}$.
   3. Compute multiplier γ_m by solving the following one-dimensional optimization problem:

$$\gamma_m = \arg\min_{\gamma} \sum_{i=1}^{n} L\left(y_i, F_{m-1}(x_i) + \gamma h_m(x_i)\right).$$

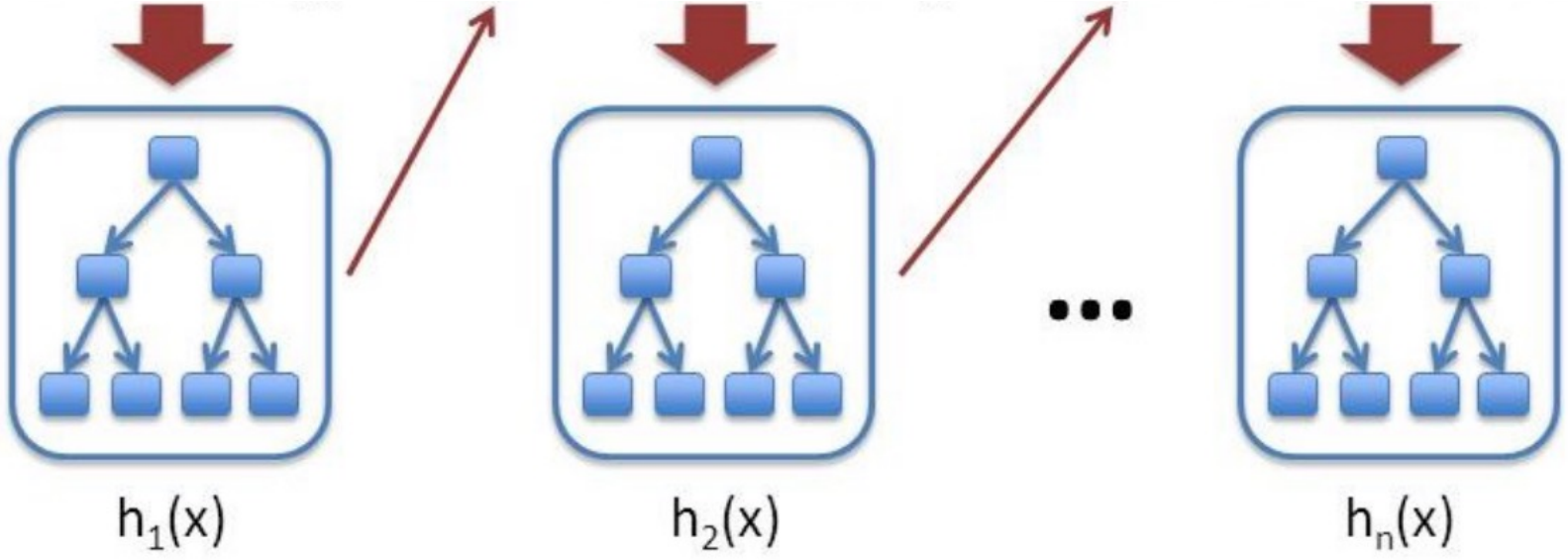
   4. Update the model:

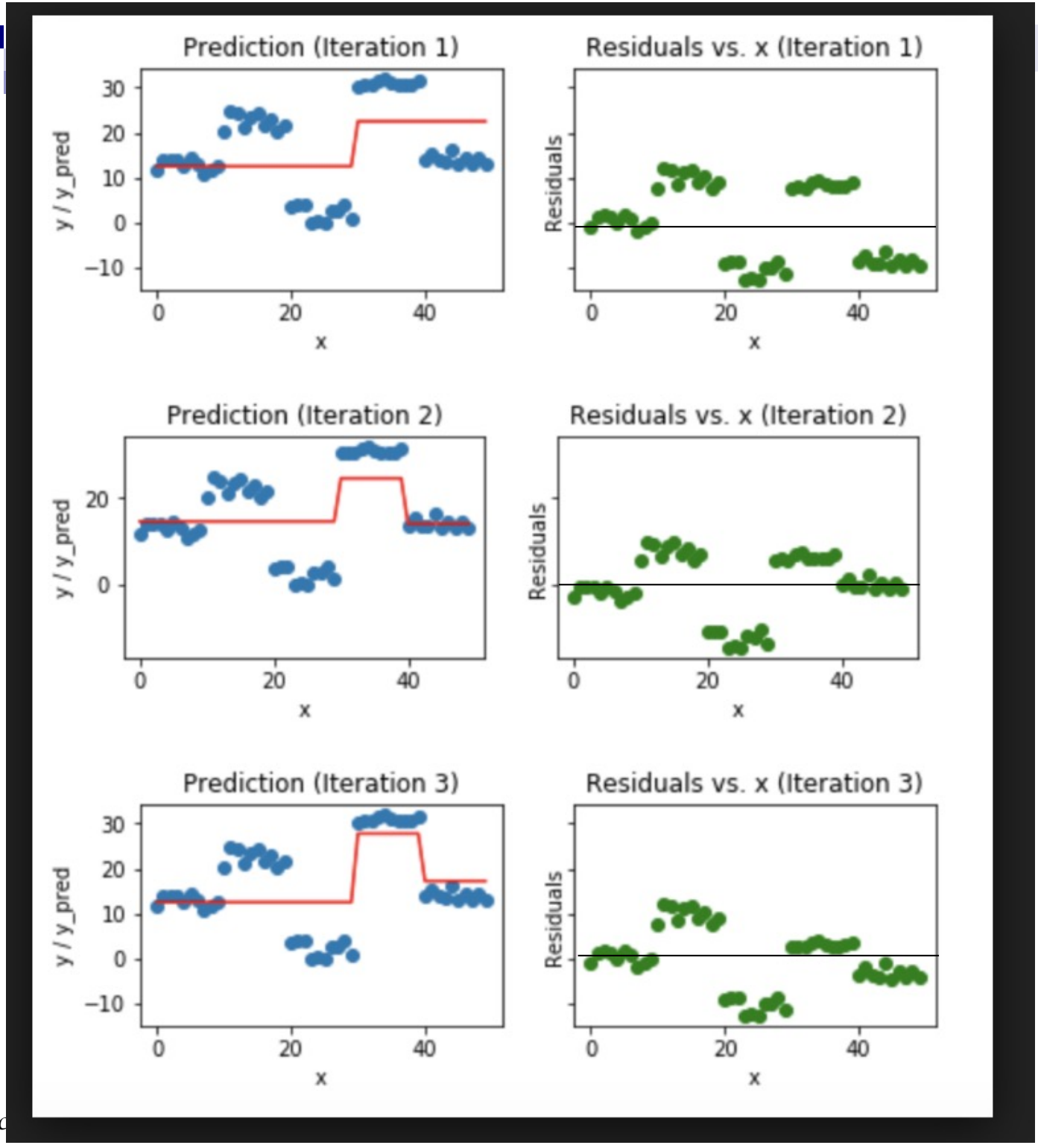
$$F_m(x) = F_{m-1}(x) + \gamma_m h_m(x).$$

3. Output $F_M(x)$.

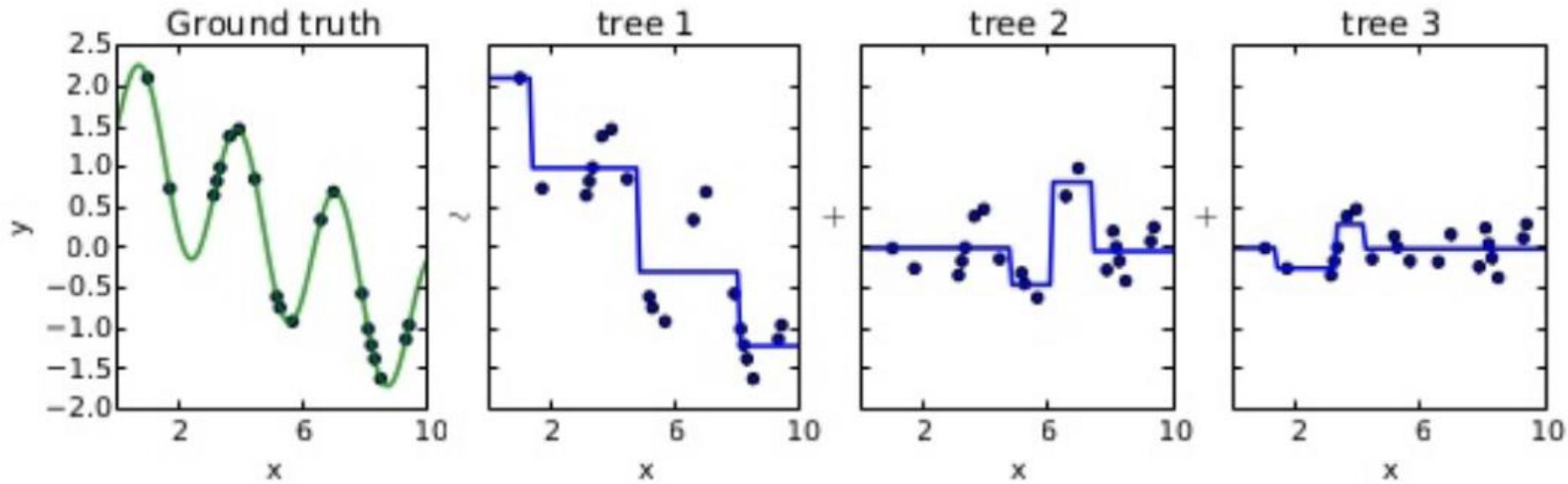
Visualization

$D_1 = \{(x_i, y_i)\}_{i=1}^{N}$ $D_2 = \{(x_i, y_i - h_1(x_i))\}_{i=1}^{N}$ $D_n = \{(x_i, y_i - h_{1:n}(x_i))\}_{i=1}^{N}$

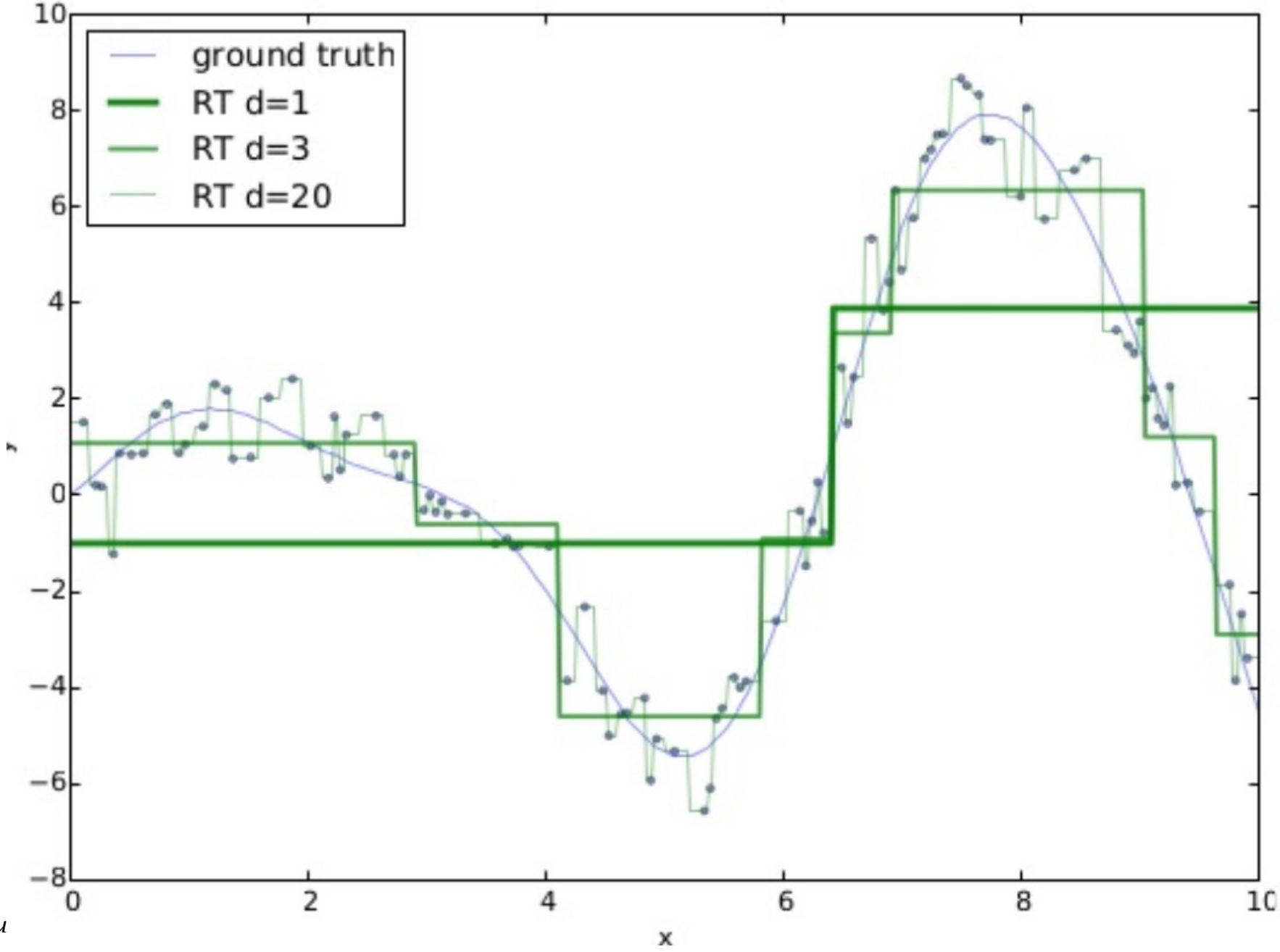



Prediction (Iteration 1)
Residuals vs. x (Iteration 1)
Prediction (Iteration 2)
Residuals vs. x (Iteration 2)
Prediction (Iteration 3)
Residuals vs. x (Iteration 3)
y / y_pred
Residuals
x

Residual fitting



https://www.slideshare.net/DataRobot/gradient-boosted-regression-trees-in-scikitlearn


ground truth
RT d=1
RT d=3
RT d=20
10
8
6
4
2
0
−2
−4
−6
−8
0
2
4
6
8
10
x

How to use it

```
>>> from sklearn.ensemble import GradientBoostingClassifier
>>> from sklearn.datasets import make_hastie_10_2
>>> X, y = make_hastie_10_2(n_samples=10000)
>>> est = GradientBoostingClassifier(n_estimators=200, max_depth=3)
>>> est.fit(X, y)
...
>>> # get predictions
>>> pred = est.predict(X)
>>> est.predict_proba(X)[0]  # class probabilities
array([ 0.67,  0.33])
```

Implementation

- Written in pure Python/Numpy (easy to extend).
- Builds on top of `sklearn.tree.DecisionTreeRegressor` (Cython).
- Custom node splitter that uses pre-sorting (better for shallow trees).

Overfitting

- Gradient Boosted Decision trees are prone to overfitting.
- When to stop?