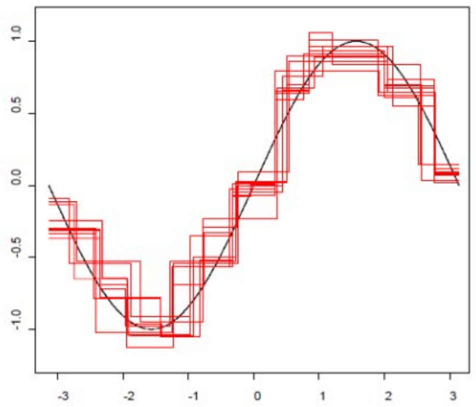
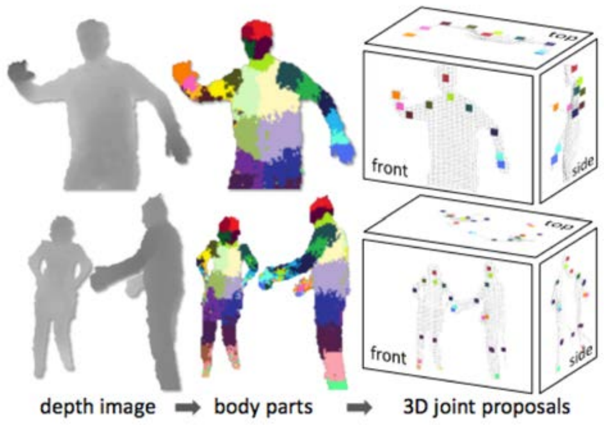
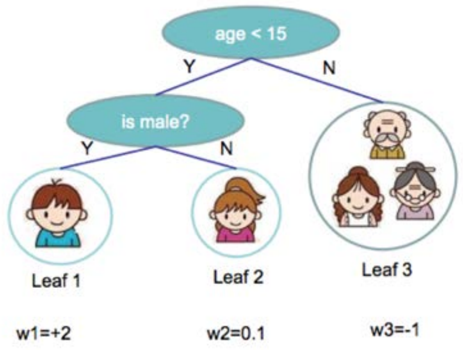
Tufts COMP 135: Introduction to Machine Learning
https://www.cs.tufts.edu/comp/135/2019s/

Ensemble Methods: Bagging and Boosting







Prof. Mike Hughes

Many slides attributable to:
Liping Liu and Roni Khardon (Tufts)
T. Q. Chen (UW),
James, Witten, Hastie, Tibshirani (ISL/ESL books)

Unit Objectives

Big idea: We can improve performance by aggregating decisions from MANY predictors

- V1: Predictors are Independently Trained
  - Using bootstrap subsample of examples: “Bagging”
  - Using random subsets of features
  - Exemplary method: Random Forest / ExtraTrees
- V2: Predictors are Sequentially Trained
  - Each successive predictor “boosts” performance
  - Exemplary method: XGBoost

Motivating Example

3 binary classifiers

- Model predictions as independent random variables
- Each one is correct 70% of the time

What is chance that majority vote is correct?

Motivating Example

3 binary classifiers

- Model predictions as independent random variables
- Each one is correct 70% of the time

What is chance that majority vote is correct?

0.784

Motivating Example

5 binary classifiers

- Model predictions as independent random variables
- Each one is correct 70% of the time

What is chance that majority vote is correct?

Motivating Example

5 binary classifiers

- Model predictions as independent random variables
- Each one is correct 70% of the time

What is chance that majority vote is correct?

0.8369...

Motivating Example

101 binary classifiers

- Model predictions as independent random variables
- Each one is correct 70% of the time

What is chance that majority vote is correct?

Motivating Example

101 binary classifiers

- Model predictions as independent random variables
- Each one is correct 70% of the time

What is chance that majority vote is correct?

>0.99…

Key Idea: Diversity

- Vary the **training data**

Bootstrap Sampling

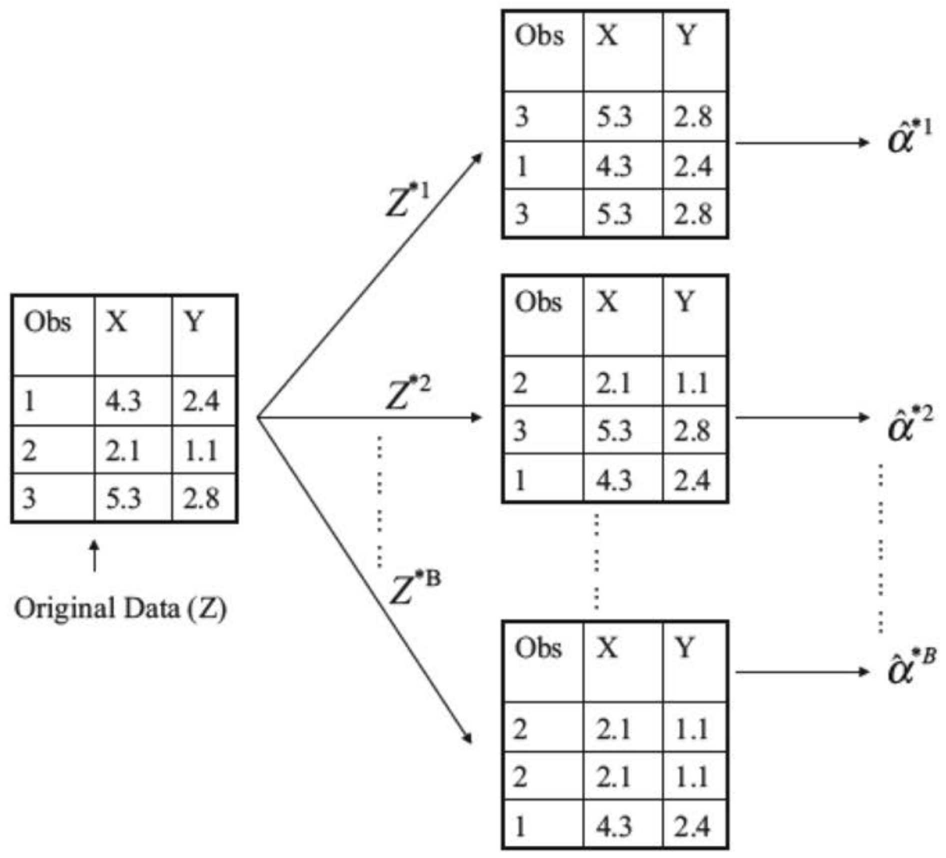
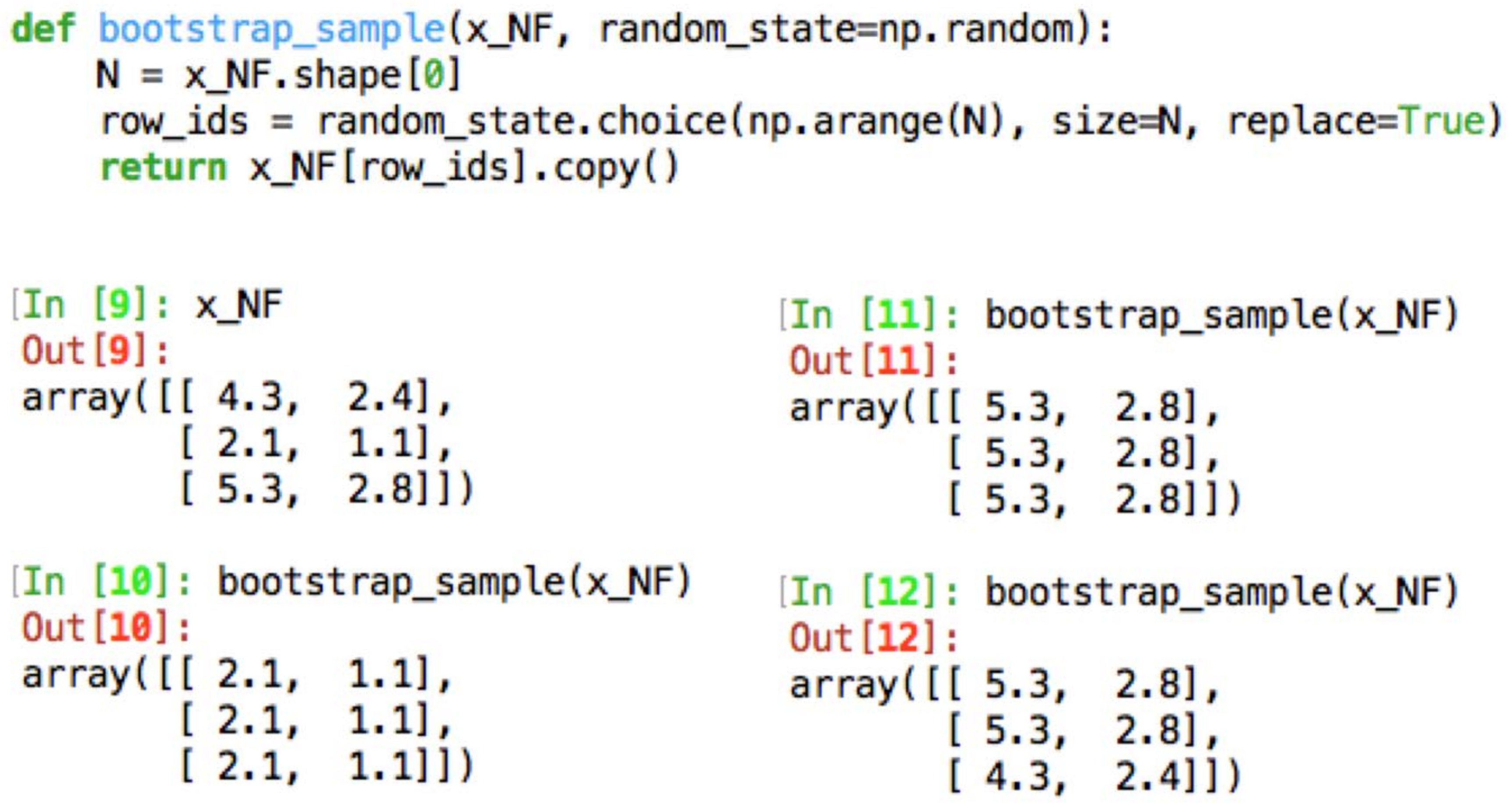


FIGURE 5.11. *A graphical illustration of the bootstrap approach on a small sample containing $n = 3$ observations. Each bootstrap data set contains n observations, sampled with replacement from the original data set. Each bootstrap data set is used to obtain an estimate of α.*

Bootstrap Sampling in Python



```python
def bootstrap_sample(x_NF, random_state=np.random):
    N = x_NF.shape[0]
    row_ids = random_state.choice(np.arange(N), size=N, replace=True)
    return x_NF[row_ids].copy()
```

```
In [9]: x_NF
Out[9]:
array([[ 4.3,  2.4],
       [ 2.1,  1.1],
       [ 5.3,  2.8]])
```

```
In [10]: bootstrap_sample(x_NF)
Out[10]:
array([[ 2.1,  1.1],
       [ 2.1,  1.1],
       [ 2.1,  1.1]])
```

```
In [11]: bootstrap_sample(x_NF)
Out[11]:
array([[ 5.3,  2.8],
       [ 5.3,  2.8],
       [ 5.3,  2.8]])
```

```
In [12]: bootstrap_sample(x_NF)
Out[12]:
array([[ 5.3,  2.8],
       [ 5.3,  2.8],
       [ 4.3,  2.4]])
```

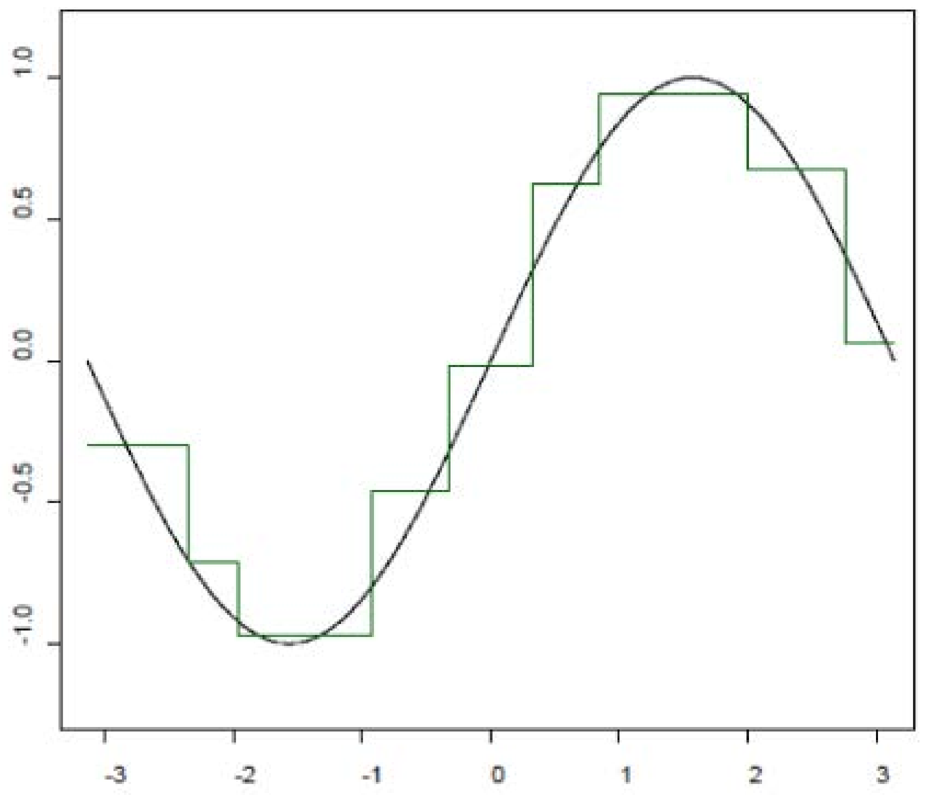
Bootstrap Aggregation: BAgg-ing

- Draw B “replicas” of training set
  - Use bootstrap sampling with replacement

- Make prediction by **averaging**

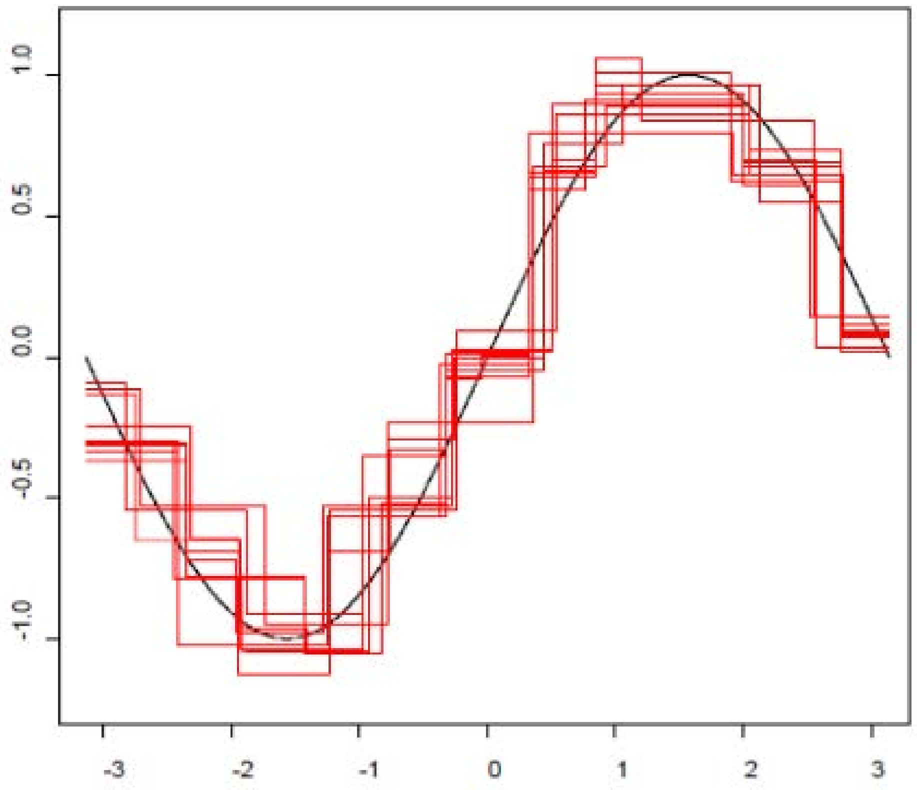
$$\hat{f}_{\text{bag}}(x) = \frac{1}{B} \sum_{b=1}^{B} \hat{f}^{*b}(x)$$

Regression Example: 1 tree



Images are taken from Adele Cutler's slides

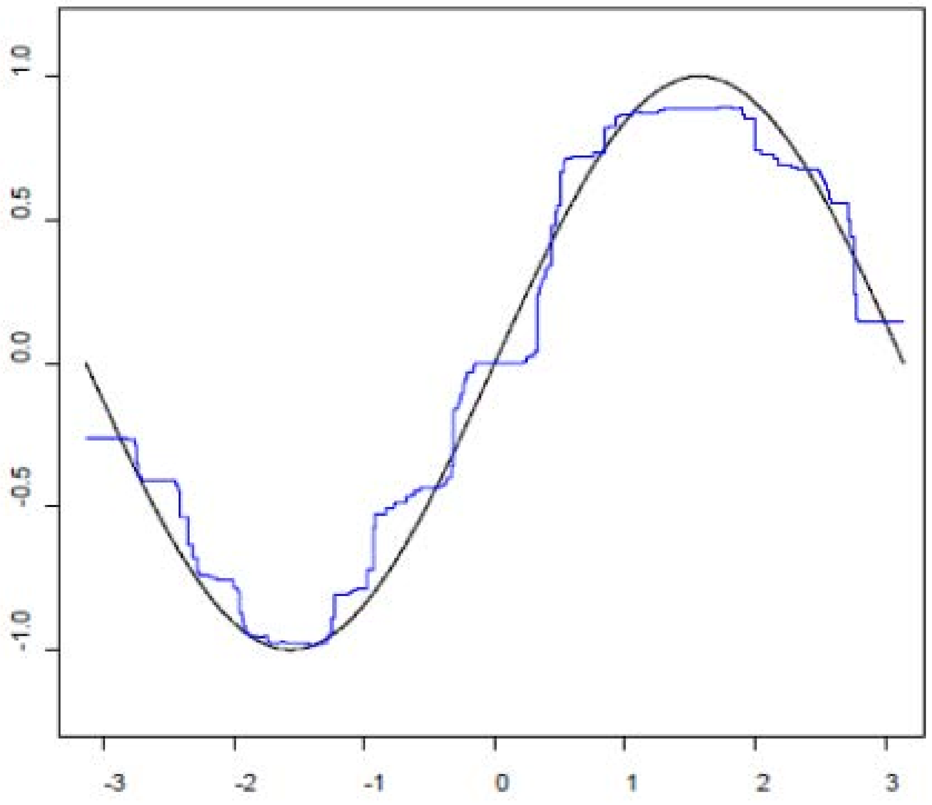
Regression Example: 10 trees



The solid black line is the ground-truth,
Red lines are predictions of single regression trees

Images are taken from Adele Cutler's slides

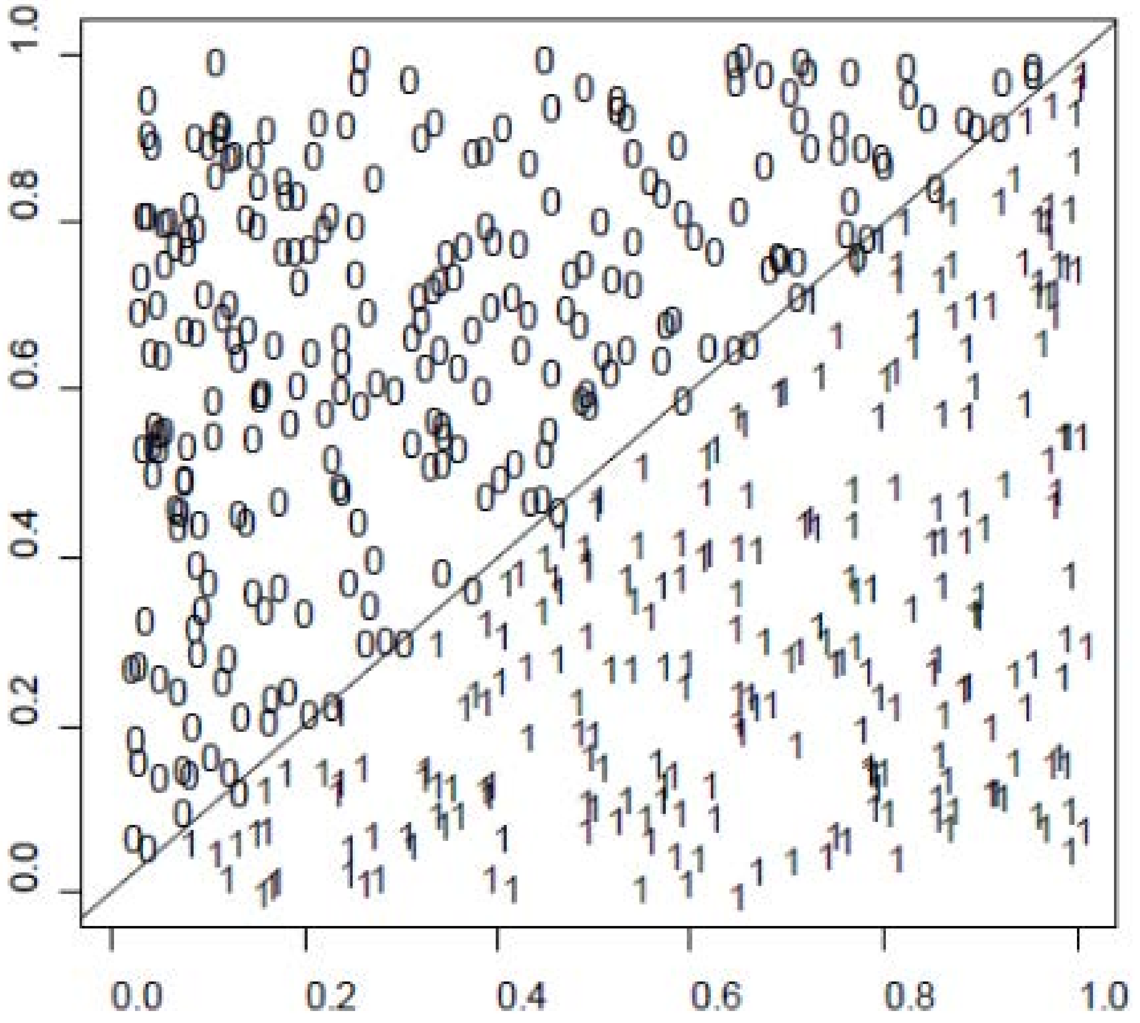
Regression Average of 10 trees



The solid black line is the ground-truth,
The blue line is the prediction of the average of 10 regression trees

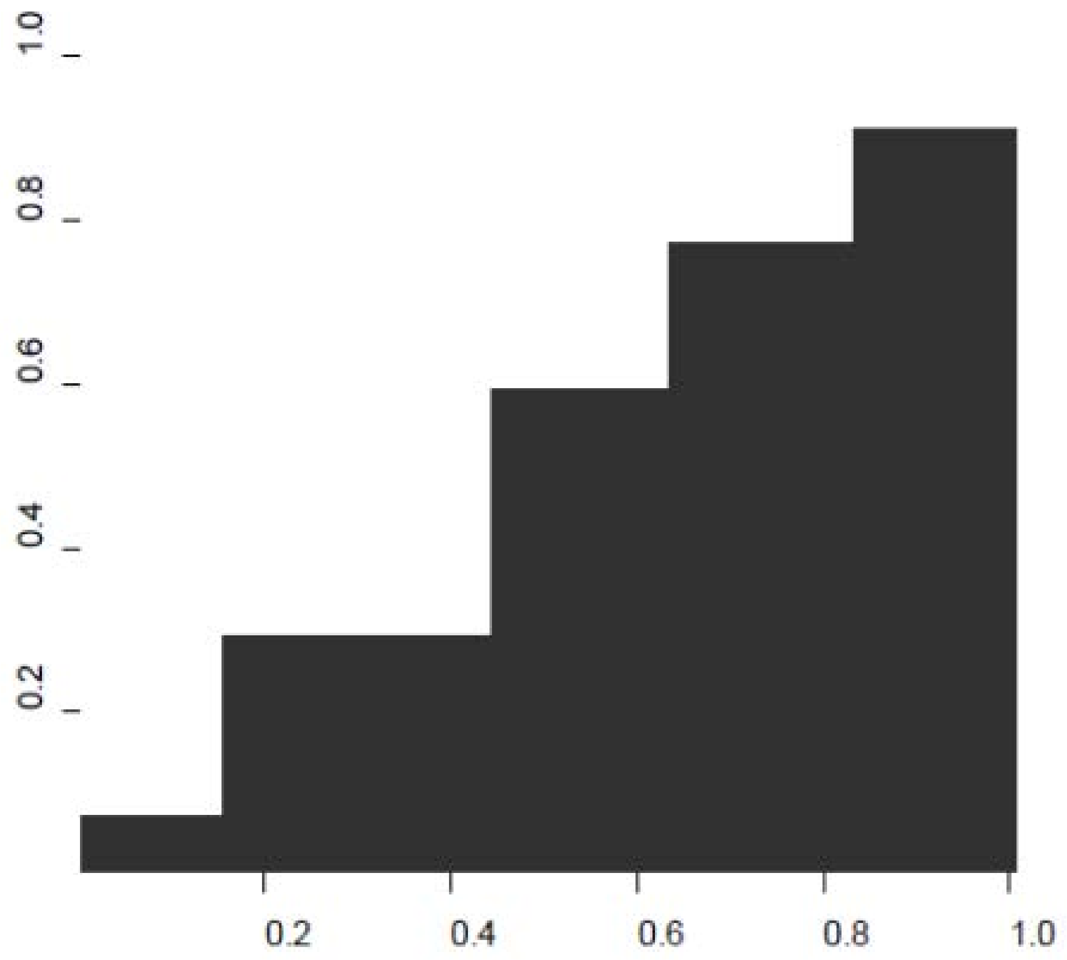
Images are taken from Adele Cutler's slides

Binary Classification



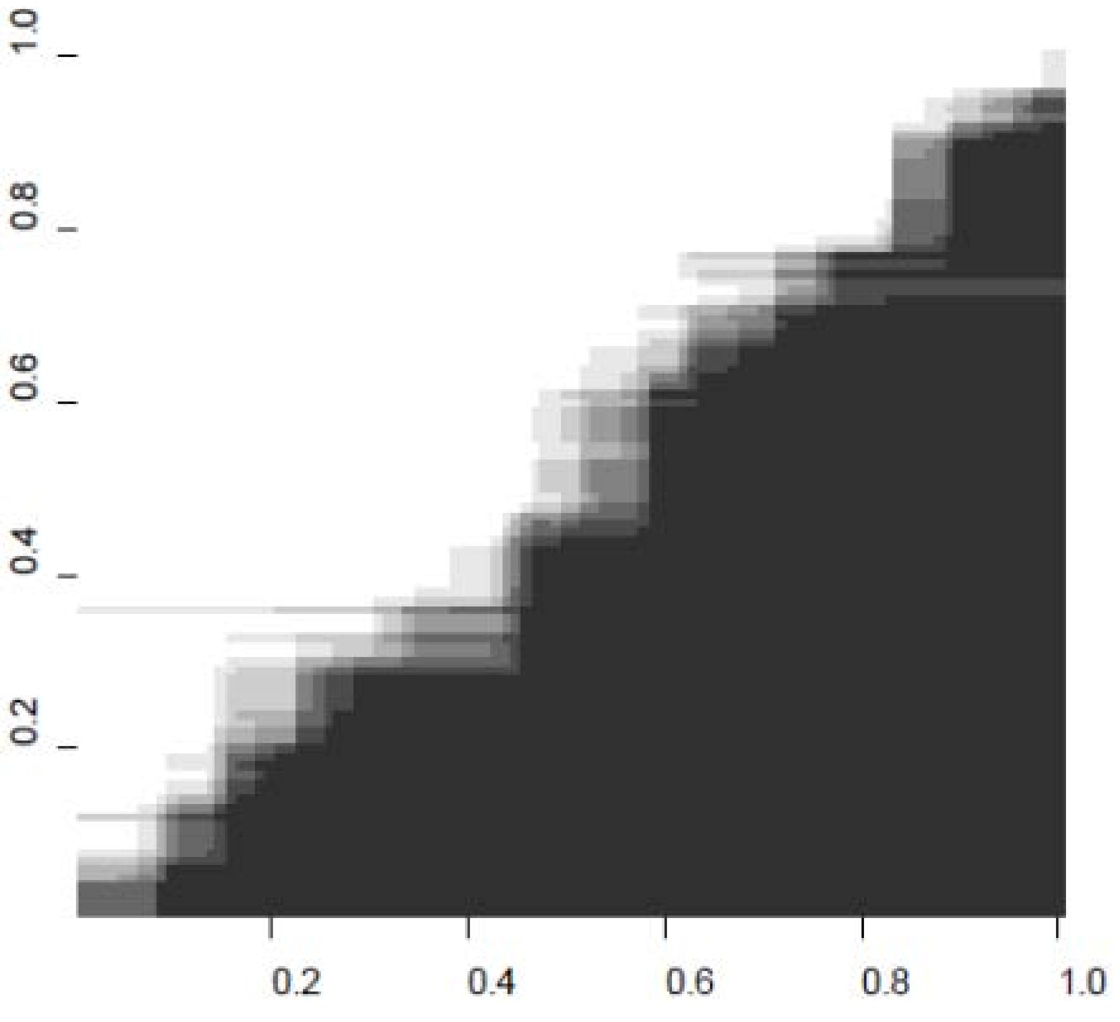
Images are taken from Adele Cutler's slides

Decision Boundary: 1 tree



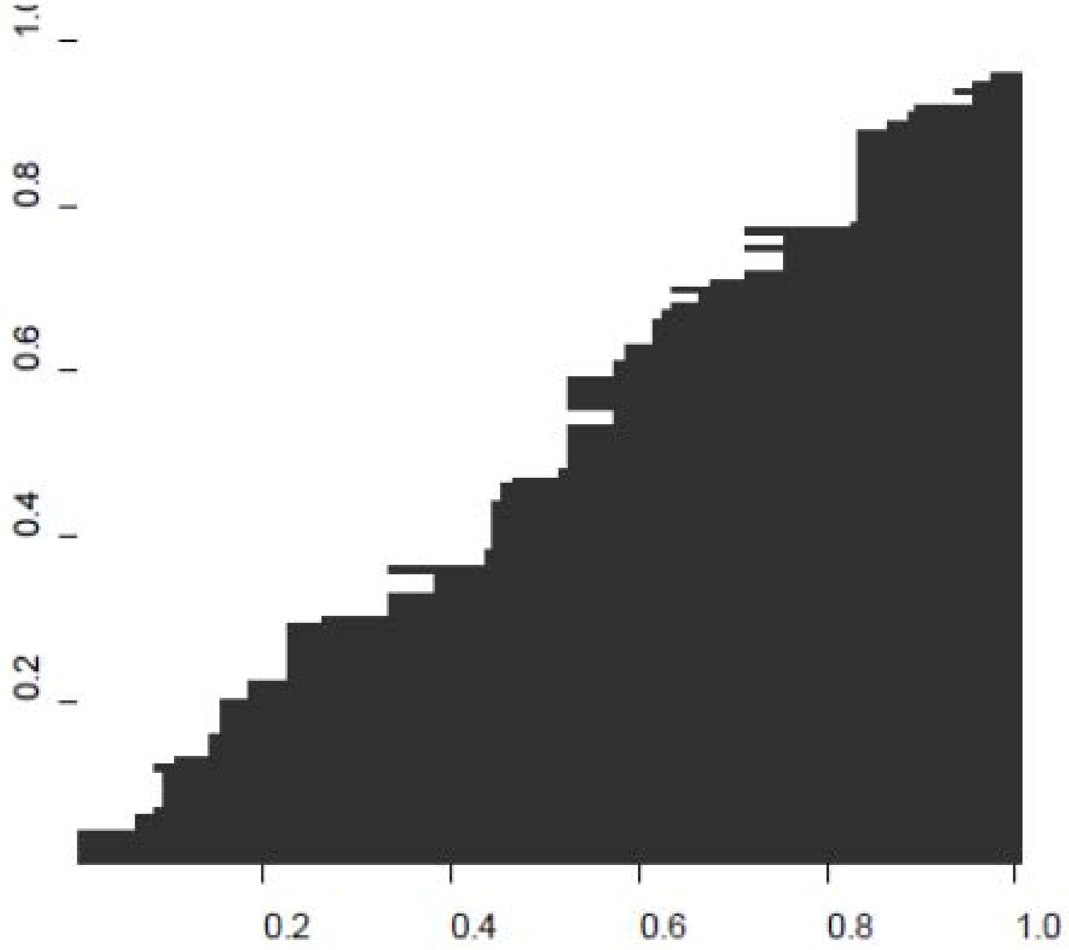
Images are taken from Adele Cutler's slides

Decision boundary: 25 trees



Images are taken from Adele Cutler's slides

Average over 25 trees



Images are taken from Adele Cutler's slides

Variance of averages

- Given B independent observations

$$z_1, z_2, \dots z_B$$

  - Each one has variance v

- Compute the mean of the B observations

$$\bar{z} = \frac{1}{B} \sum_{b=1}^{B} z_b$$

  - What is variance of this estimator?

Why Bagging Works: Reduce Variance!

- Flexible learners applied to small datasets have high variance w.r.t. the data distribution
  - Small change in training set -> big change in predictions on heldout set

- Bagging decreases heldout error by decreasing the variance of predictions

- Bagging can be applied to **any** base classifiers/regressors

Another Idea for Diversity

- Vary the **features**

Random Forest

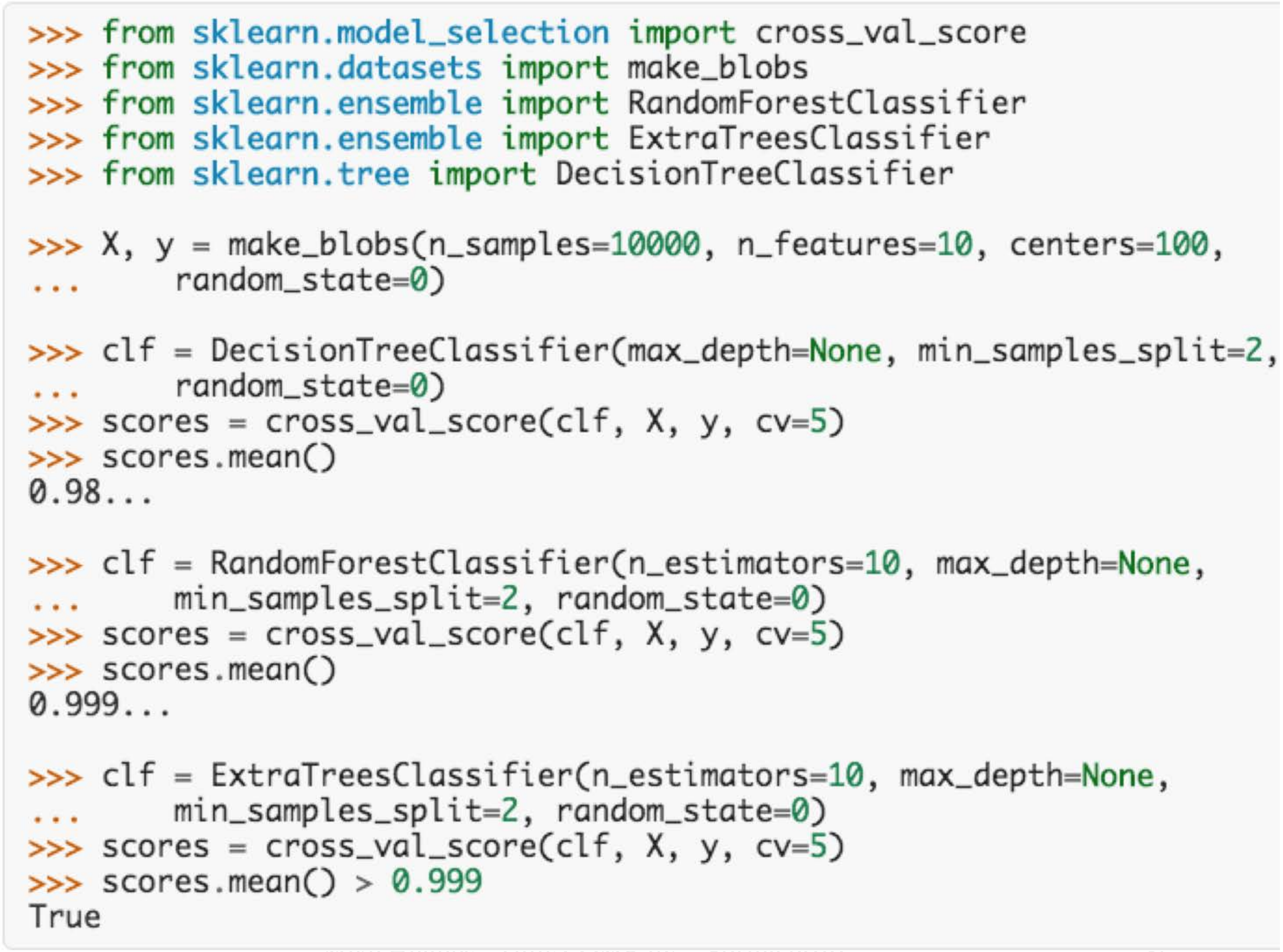
Combine example diversity AND feature diversity

```
For t = 1 to T (# trees):
   Create an **bootstrap sample** from the training set.
   Greedy train tree on **random subsample** of features

       For each node within a **maximum depth**:
              Randomly select $m$ features from $F$ features
              Find the best split among $m$ variables

Average the trees to get predictions for new data.
```

Extremely Randomized Trees aka “ExtraTrees” in sklearn

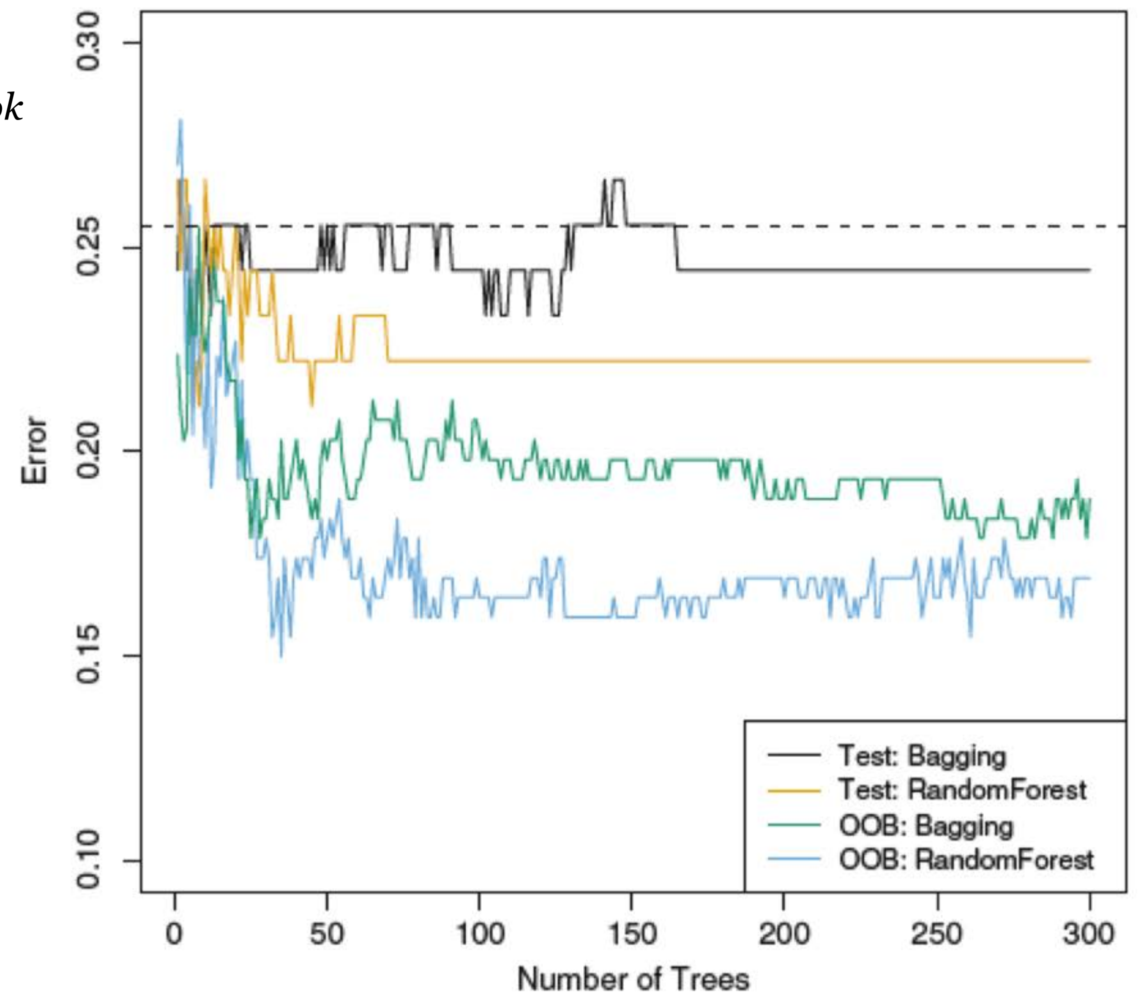
Speed and feature diversity

```
For t = 1 to T (# trees):
  Create an **bootstrap sample** from the training set.
  Greedy train tree on **random subsample** of features

    For each node within a **maximum depth**:
        Randomly select $m$ features from $F$ features
        ~~Find the best split among $m$ variables~~
        Try 1 random split at each of m variables,
            then select the best split of these
```



```
>>> from sklearn.model_selection import cross_val_score
>>> from sklearn.datasets import make_blobs
>>> from sklearn.ensemble import RandomForestClassifier
>>> from sklearn.ensemble import ExtraTreesClassifier
>>> from sklearn.tree import DecisionTreeClassifier

>>> X, y = make_blobs(n_samples=10000, n_features=10, centers=100,
...     random_state=0)

>>> clf = DecisionTreeClassifier(max_depth=None, min_samples_split=2,
...     random_state=0)
>>> scores = cross_val_score(clf, X, y, cv=5)
>>> scores.mean()
0.98...

>>> clf = RandomForestClassifier(n_estimators=10, max_depth=None,
...     min_samples_split=2, random_state=0)
>>> scores = cross_val_score(clf, X, y, cv=5)
>>> scores.mean()
0.999...

>>> clf = ExtraTreesClassifier(n_estimators=10, max_depth=None,
...     min_samples_split=2, random_state=0)
>>> scores = cross_val_score(clf, X, y, cv=5)
>>> scores.mean() > 0.999
True
```

Credit: ISL textbook



Single tree

FIGURE 8.8. *Bagging and random forest results for the* Heart *data. The test error (black and orange) is shown as a function of B, the number of bootstrapped training sets used. Random forests were applied with $m = \sqrt{p}$. The dashed line indicates the test error resulting from a single classification tree. The green and blue traces show the OOB error, which in this case is considerably lower.*

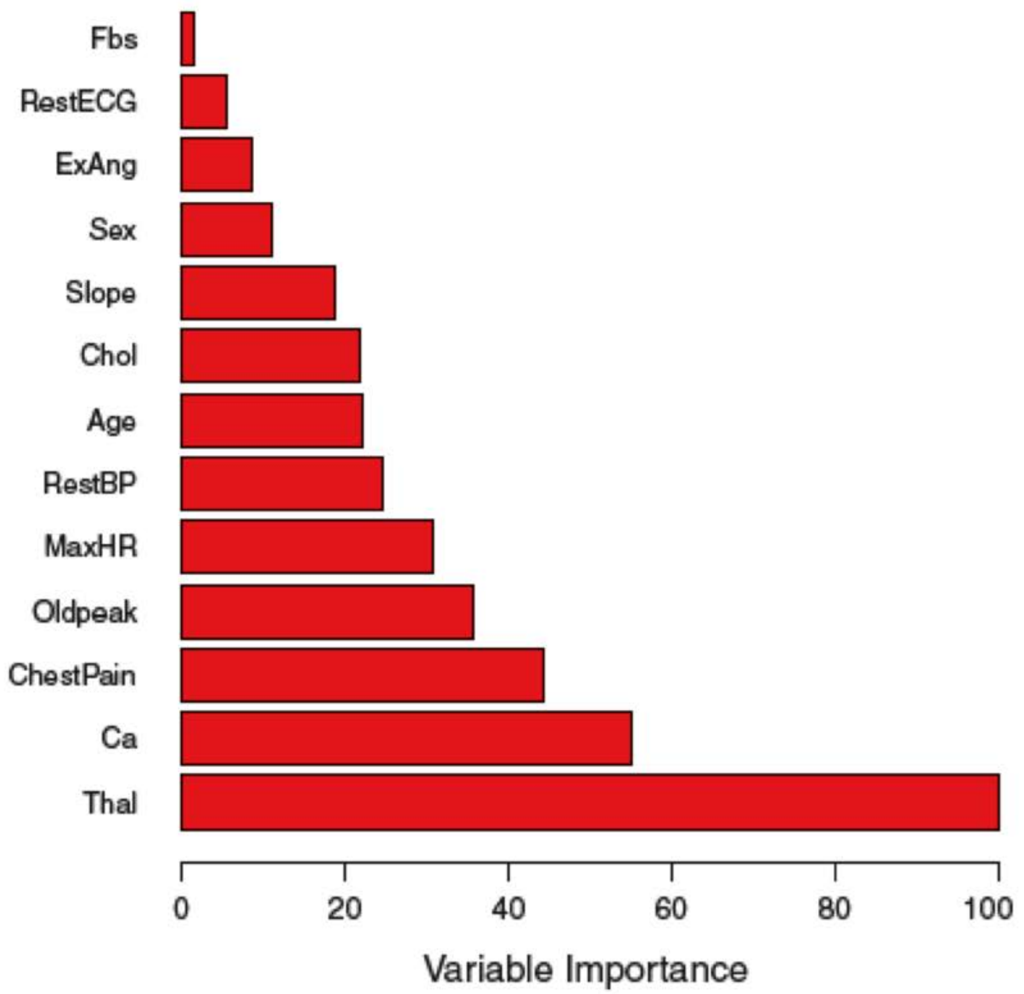
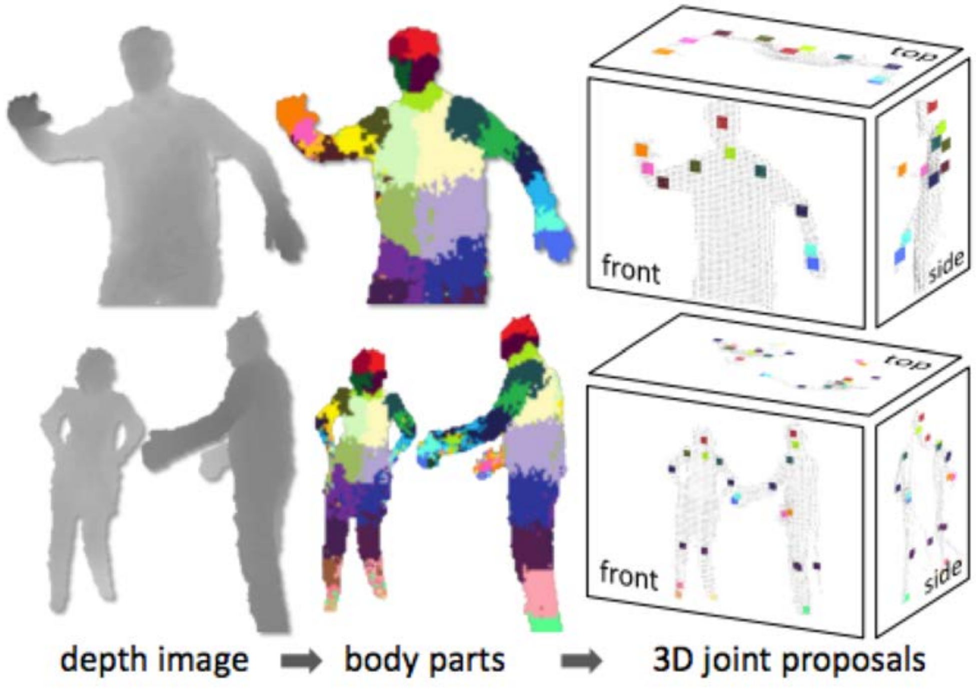


FIGURE 8.9. *A variable importance plot for the* `Heart` *data. Variable importance is computed using the mean decrease in Gini index, and expressed relative to the maximum.*

Random Forest in Industry

Real-Time Human Pose Recognition in Parts from Single Depth Images

Jamie Shotton Andrew Fitzgibbon Mat Cook Toby Sharp Mark Finocchio
Richard Moore Alex Kipman Andrew Blake
Microsoft Research Cambridge & Xbox Incubation



Microsoft Kinect RGB-D camera

Real-Time Human Pose Recognition in Parts from Single Depth Images

Jamie Shotton Andrew Fitzgibbon Mat Cook Toby Sharp Mark Finocchio
Richard Moore Alex Kipman Andrew Blake
Microsoft Research Cambridge & Xbox Incubation

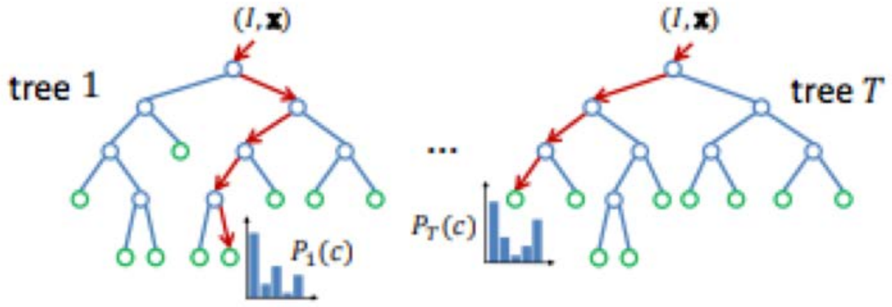


Figure 4. **Randomized Decision Forests.** A forest is an ensemble of trees. Each tree consists of split nodes (blue) and leaf nodes (green). The red arrows indicate the different paths that might be taken by different trees for a particular input.

3.3. Randomized decision forests

Randomized decision trees and forests [35, 30, 2, 8] have proven fast and effective multi-class classifiers for many tasks [20, 23, 36], and can be implemented efficiently on the GPU [34]. As illustrated in Fig. 4, a forest is an ensemble of T decision trees, each consisting of split and leaf nodes. Each split node consists of a feature f_θ and a threshold τ. To classify pixel $\mathbf{x}$ in image I, one starts at the root and repeatedly evaluates Eq. 1, branching left or right according to the comparison to threshold τ. At the leaf node reached in tree t, a learned distribution $P_t(c|I, \mathbf{x})$ over body part labels c is stored. The distributions are averaged together for all trees in the forest to give the final classification

$$P(c|I, \mathbf{x}) = \frac{1}{T} \sum_{t=1}^{T} P_t(c|I, \mathbf{x}) . \qquad (2)$$

Training. Each tree is trained on a different set of randomly synthesized images. A random subset of 2000 example pixels from each image is chosen to ensure a roughly even distribution across body parts. Each tree is trained using the following algorithm [20]:

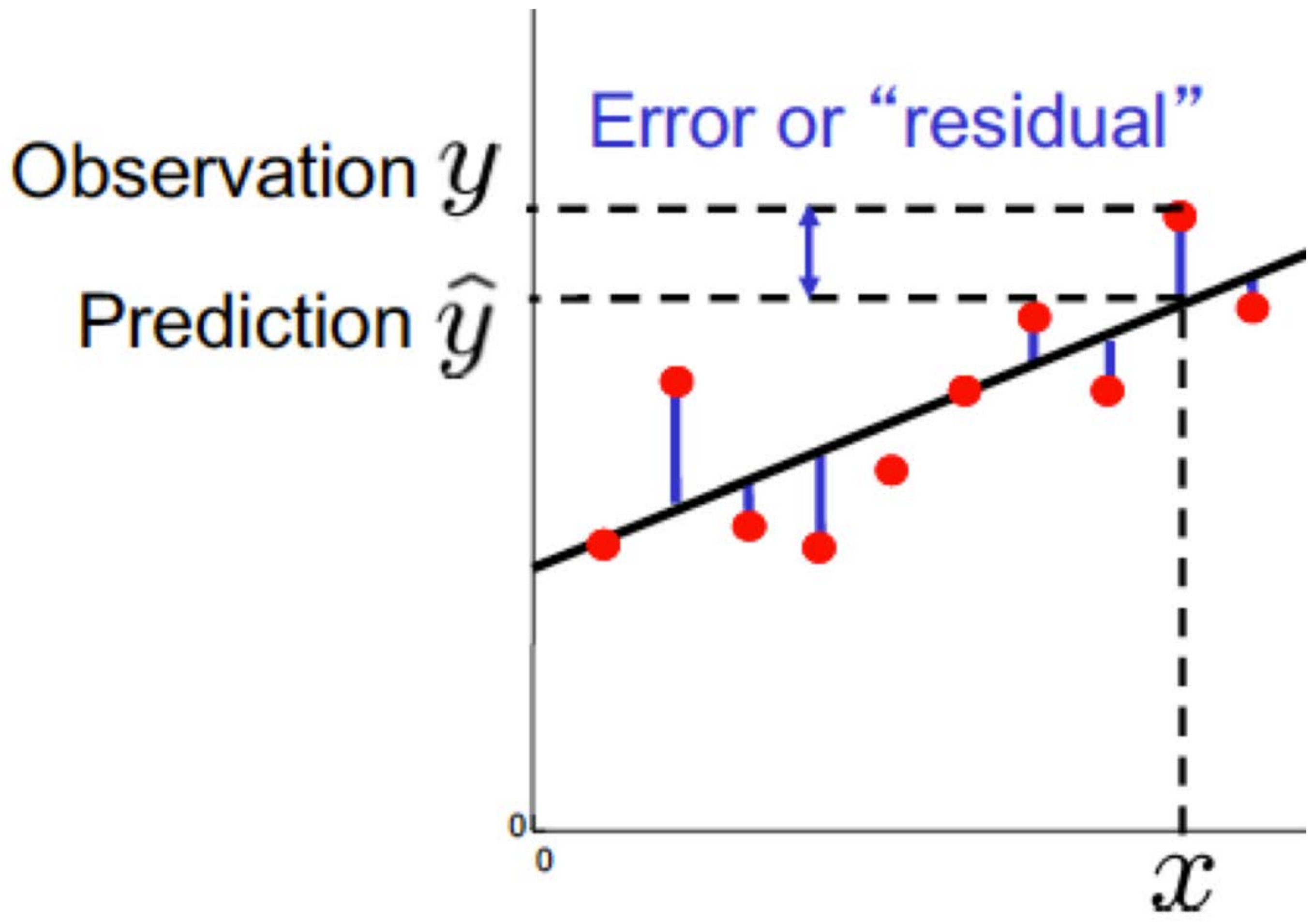
1. Randomly propose a set of splitting candidates $\phi = (\theta, \tau)$ (feature parameters θ and thresholds τ).
2. Partition the set of examples $Q = \{(I, \mathbf{x})\}$ into left and right subsets by each ϕ:

$$Q_l(\phi) = \{ (I, \mathbf{x}) \mid f_\theta(I, \mathbf{x}) < \tau \} \qquad (3)$$
$$Q_r(\phi) = Q \setminus Q_l(\phi) \qquad (4)$$

Summary of Ensemble v1: Independent predictors

- Average over independent base predictors
- Why it works: Reduce variance
- PRO
  - Often better heldout performance than base model
- CON
  - Training B separate models is expensive

Vocabulary: Residual



Ensemble Method v2: Sequentially Predict Residual

- Model f1: Trained with (x,y) pairs
  - Capture residual: r1 = y – f1

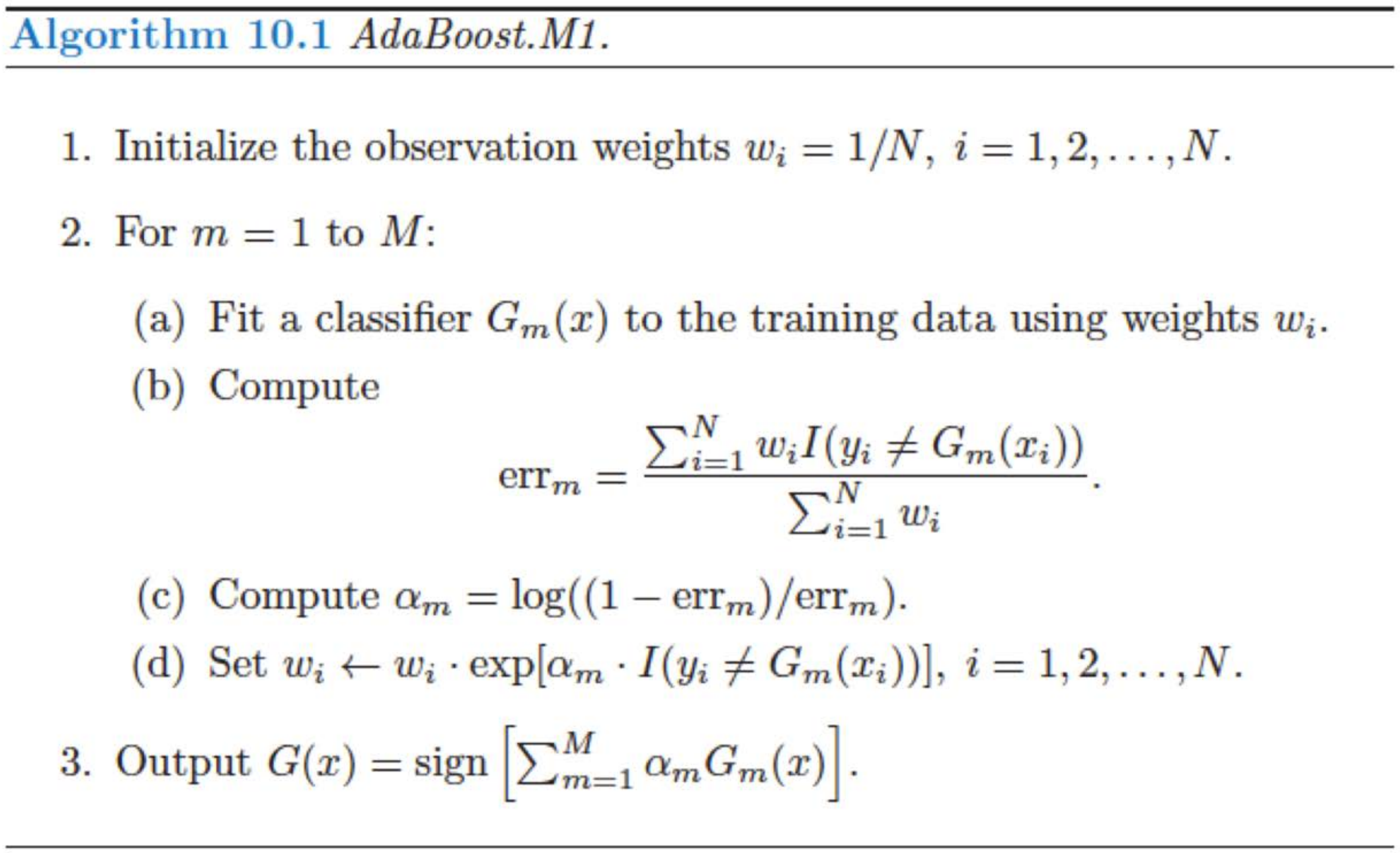
- Model f2: Trained with (x, r1) pairs
  - Capture residual: r2 = r1 – f2

- Repeat!

Combine weak learners into a powerful committee

Adaboost (Adaptive Boosting)
Reweight misclassified examples

ESL textbook



Algorithm 10.1 *AdaBoost.M1.*

```
1. Initialize the observation weights $w_i = 1/N,\ i = 1, 2, \ldots, N$.
2. For $m = 1$ to $M$:
    (a) Fit a classifier $G_m(x)$ to the training data using weights $w_i$.
    (b) Compute
            $\text{err}_m = \frac{\sum_{i=1}^N w_i I(y_i \neq G_m(x_i))}{\sum_{i=1}^N w_i}$.
    (c) Compute $\alpha_m = \log((1 - \text{err}_m)/\text{err}_m)$.
    (d) Set $w_i \leftarrow w_i \cdot \exp[\alpha_m \cdot I(y_i \neq G_m(x_i))],\ i = 1, 2, \ldots, N$.
3. Output $G(x) = \text{sign}\left[\sum_{m=1}^M \alpha_m G_m(x)\right]$.
```

Boosting with depth-1 tree "stump"

ESL textbook

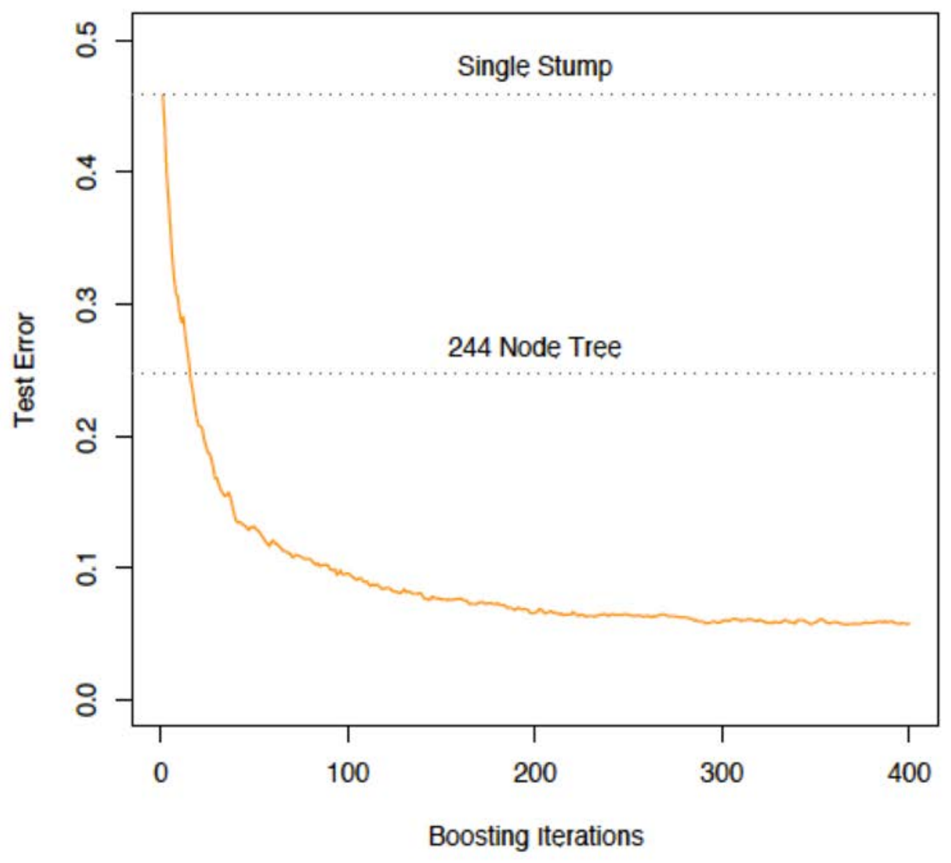
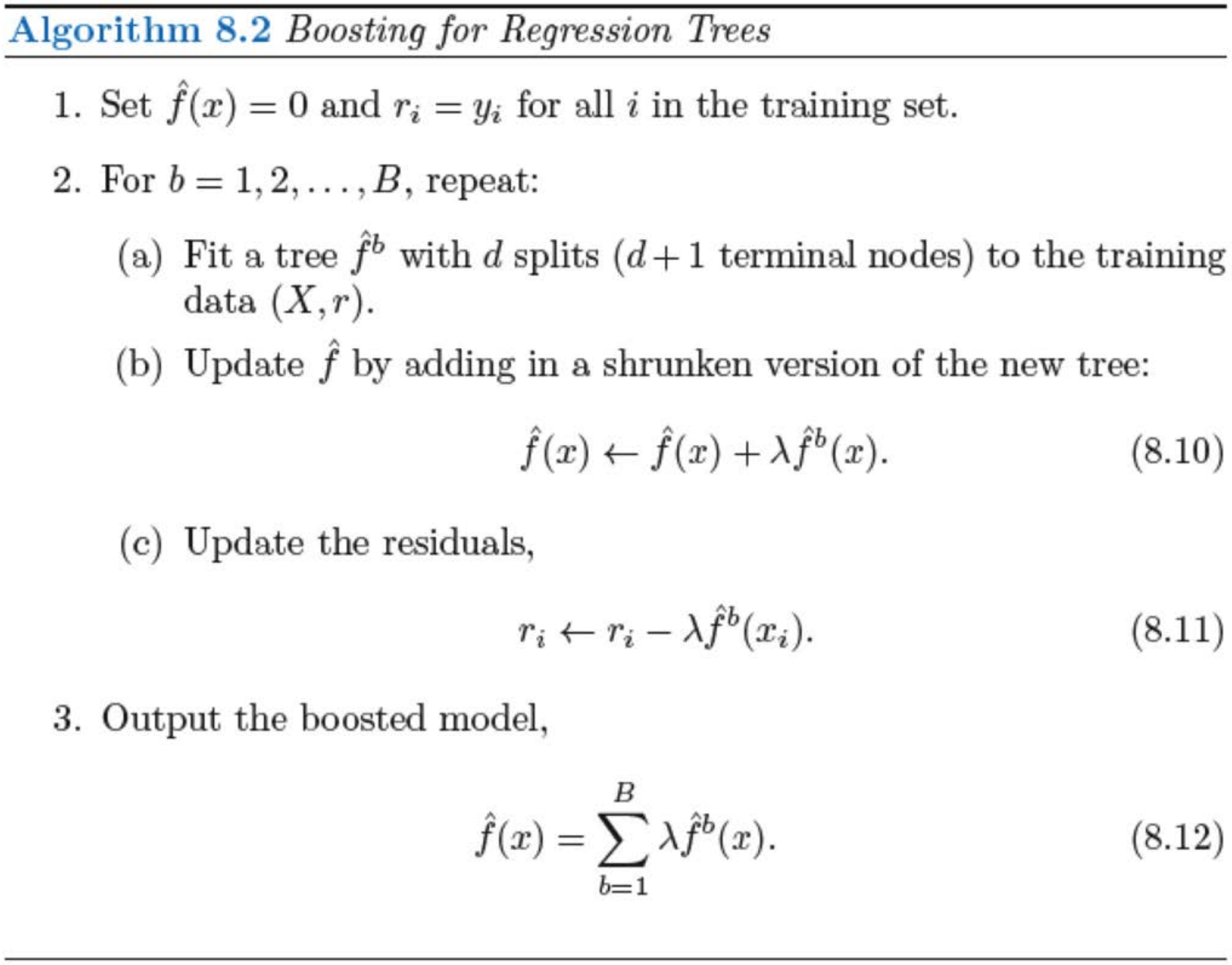


FIGURE 10.2. *Simulated data (10.2): test error rate for boosting with stumps, as a function of the number of iterations. Also shown are the test error rate for a single stump, and a 244-node classification tree.*

Boosting for Regression Trees

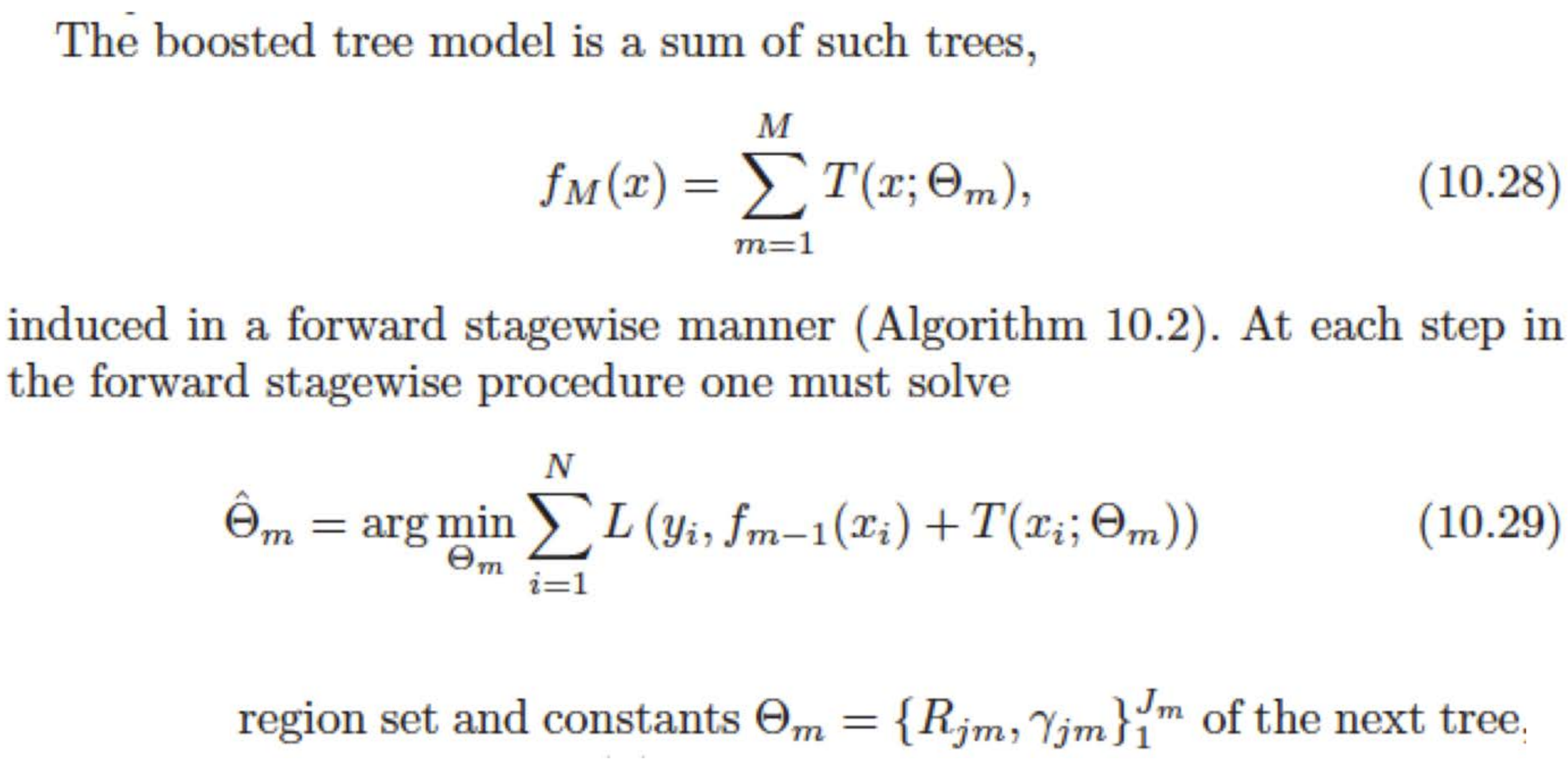
ISL textbook



Algorithm 8.2 *Boosting for Regression Trees*

```
1. Set $\hat{f}(x) = 0$ and $r_i = y_i$ for all $i$ in the training set.
2. For $b = 1, 2, \ldots, B$, repeat:
    (a) Fit a tree $\hat{f}^b$ with $d$ splits ($d + 1$ terminal nodes) to the training data $(X, r)$.
    (b) Update $\hat{f}$ by adding in a shrunken version of the new tree:
            $\hat{f}(x) \leftarrow \hat{f}(x) + \lambda \hat{f}^b(x).$        (8.10)
    (c) Update the residuals,
            $r_i \leftarrow r_i - \lambda \hat{f}^b(x_i).$        (8.11)
3. Output the boosted model,
            $\hat{f}(x) = \sum_{b=1}^{B} \lambda \hat{f}^b(x).$        (8.12)
```

Boosted Tree: Optimization



The boosted tree model is a sum of such trees,

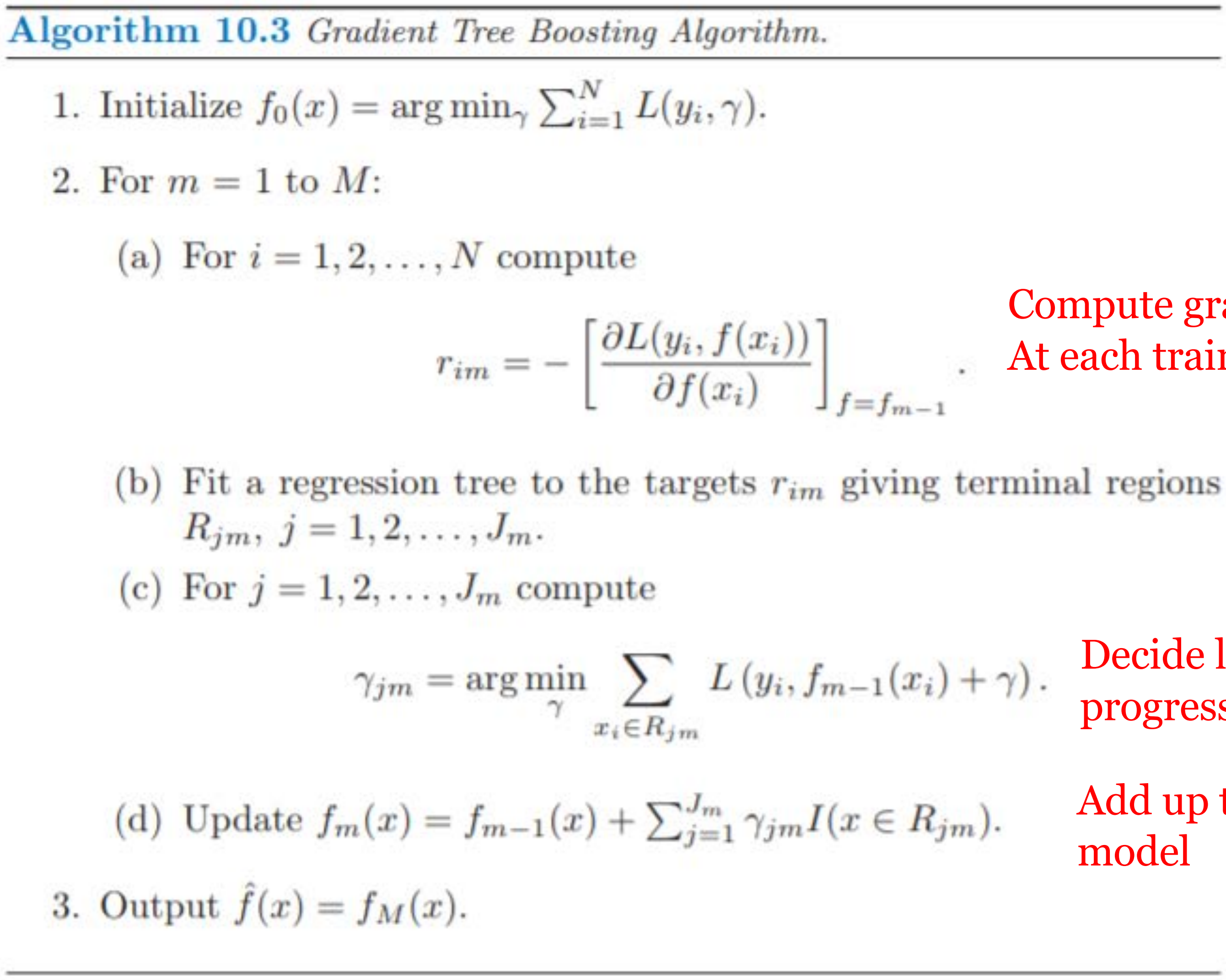
$$f_M(x) = \sum_{m=1}^{M} T(x; \Theta_m), \qquad (10.28)$$

induced in a forward stagewise manner (Algorithm 10.2). At each step in the forward stagewise procedure one must solve

$$\hat{\Theta}_m = \arg\min_{\Theta_m} \sum_{i=1}^{N} L\left(y_i, f_{m-1}(x_i) + T(x_i; \Theta_m)\right) \qquad (10.29)$$

region set and constants $\Theta_m = \{R_{jm}, \gamma_{jm}\}_1^{J_m}$ of the next tree.

Gradient Boosting Algorithm



Algorithm 10.3 *Gradient Tree Boosting Algorithm.*

```
1. Initialize $f_0(x) = \arg\min_\gamma \sum_{i=1}^{N} L(y_i, \gamma)$.
2. For $m = 1$ to $M$:
    (a) For $i = 1, 2, \ldots, N$ compute
            $r_{im} = -\left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f=f_{m-1}}$.
    (b) Fit a regression tree to the targets $r_{im}$ giving terminal regions
        $R_{jm}, j = 1, 2, \ldots, J_m$.
    (c) For $j = 1, 2, \ldots, J_m$ compute
            $\gamma_{jm} = \arg\min_\gamma \sum_{x_i \in R_{jm}} L(y_i, f_{m-1}(x_i) + \gamma)$.
    (d) Update $f_m(x) = f_{m-1}(x) + \sum_{j=1}^{J_m} \gamma_{jm} I(x \in R_{jm})$.
3. Output $\hat{f}(x) = f_M(x)$.
```

Compute gradient
At each training example

Decide tree structure
by fitting to gradients

Decide leaf values by
progressively minimizing loss

Add up trees to get the final
model

What about regularization?

https://xgboost.readthedocs.io/

Minimization objective when adding tree *t*:

$$\text{obj}^{(t)} = \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t)}) + \sum_{i=1}^{t} \Omega(f_i)$$

$$= \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \Omega(f_t) + \text{constant}$$

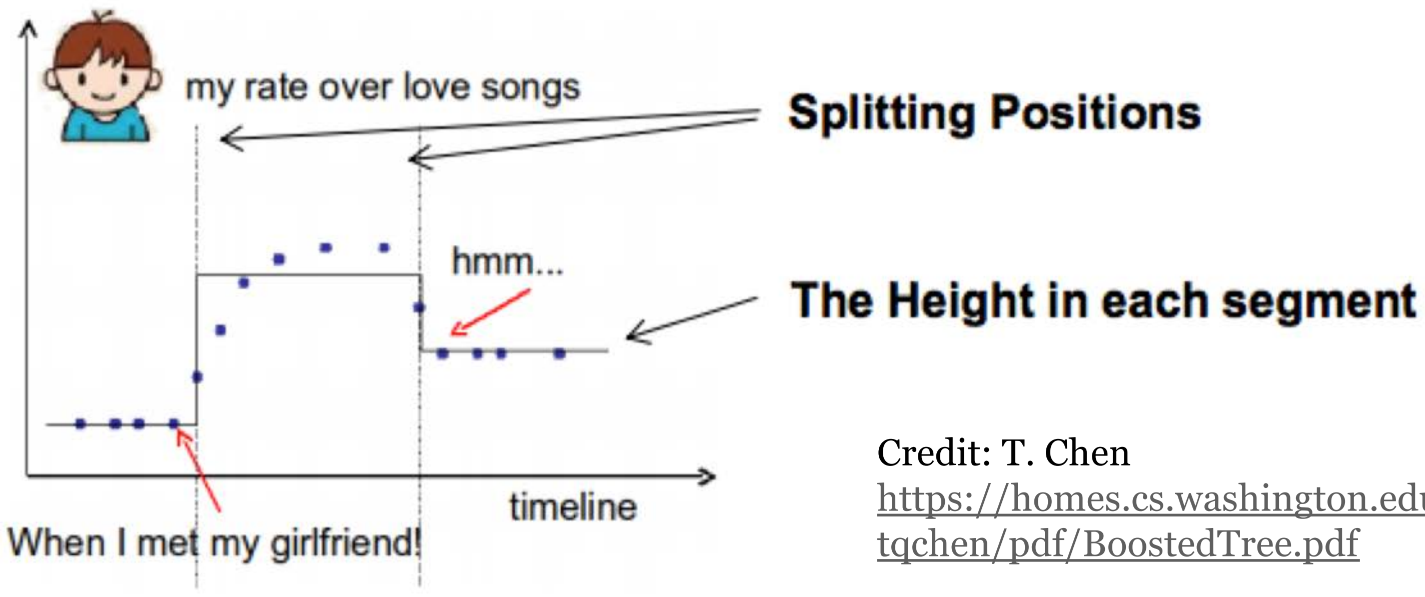
Loss function

Regularization
(limit complexity of tree t)

Regularization

We can penalize

- Number of nodes in tree
- Depth of tree
- Scalar value predicted in each region (L2 penalty)



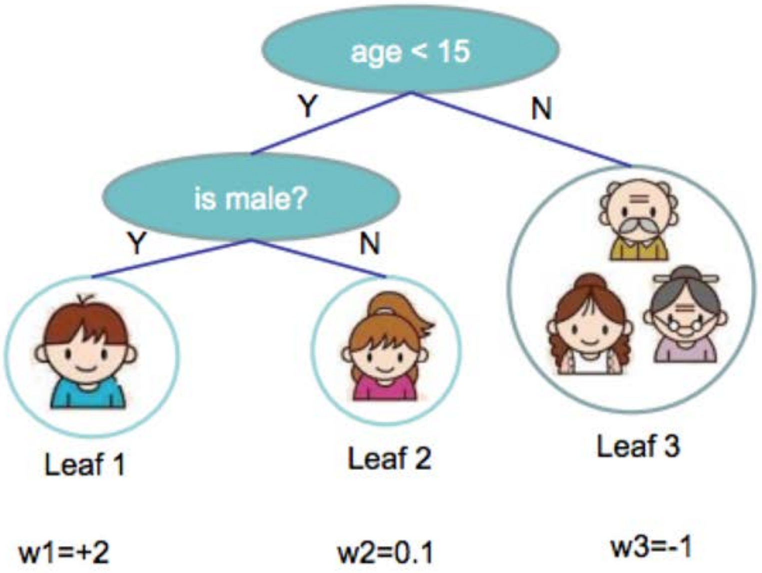
Credit: T. Chen
https://homes.cs.washington.edu/~tqchen/pdf/BoostedTree.pdf

Example Regularization Term

- Define complexity as (this is not the only possible definition)

$$\Omega(f_t) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2$$

Number of leaves **L2 norm of leaf scores**



$$\Omega = \gamma 3 + \frac{1}{2}\lambda(4 + 0.01 + 1)$$

Credit: T. Chen
https://homes.cs.washington.edu/~tqchen/pdf/BoostedTree.pdf

To Improve Gradient Boosting

Can extend gradient boosting with

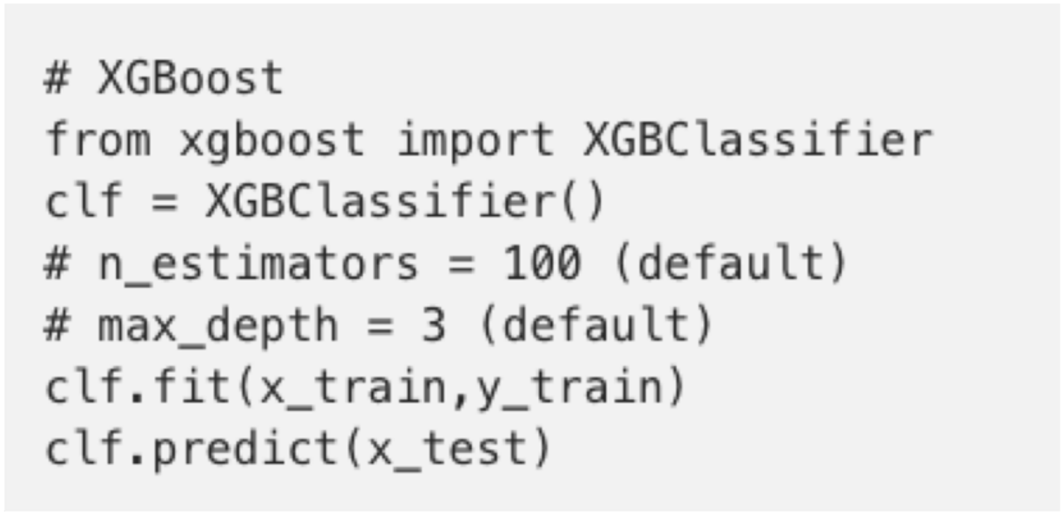
- 2nd order gradient information (Newton step)
- Penalties on tree complexity
- Very smart practical implementation

Result: **Extreme** Gradient Boosting

aka XGBoost (T. Chen & C. Guestrin)

XGBoost: **Extreme** Gradient Boosting

- Kaggle competitions in 2015
  - 29 total winning solutions to challenges published
  - 17 / 29 (58%) used XGBoost
  - 11 / 29 (37%) used deep neural networks



```
# XGBoost
from xgboost import XGBClassifier
clf = XGBClassifier()
# n_estimators = 100 (default)
# max_depth = 3 (default)
clf.fit(x_train,y_train)
clf.predict(x_test)
```

More details (beyond this class)

ESL textbook, Section 10.10

Good slide deck by T. Q. Chen (first author of XGBoost):

- https://homes.cs.washington.edu/~tqchen/pdf/BoostedTree.pdf

Summary of Boosting

PRO

- Like all tree methods, invariant to scaling of inputs (no need for careful feature normalization)
- Can be scalable

CON

- Greedy Sequential fit may not be globally optimal

IN PRACTICE

- XGBoost
  - Very popular in many benchmark competitions and industrial applications

Unit Objectives

Big idea: We can improve performance by aggregating decisions from MANY predictors

- V1: Predictors are Independently Trained
  - Using bootstrap subsample of examples: "Bagging"
  - Using random subsets of features
  - Exemplary method: Random Forest / ExtraTrees
- V2: Predictors are Sequentially Trained
  - Each successive predictor "boosts" performance
  - Exemplary method: XGBoost