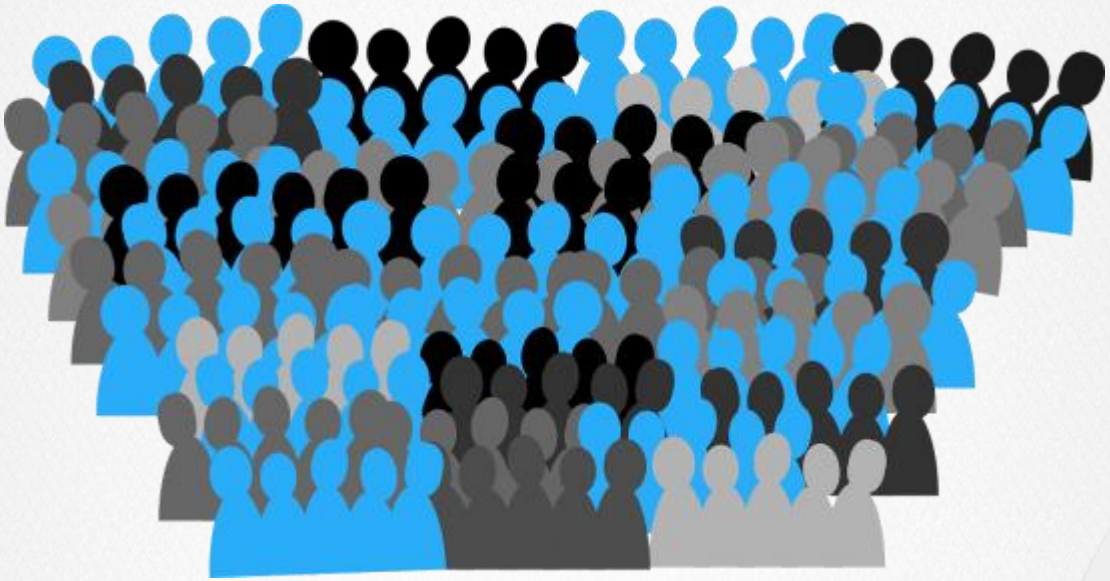
Model Improvements

Practical Machine Learning (with R)

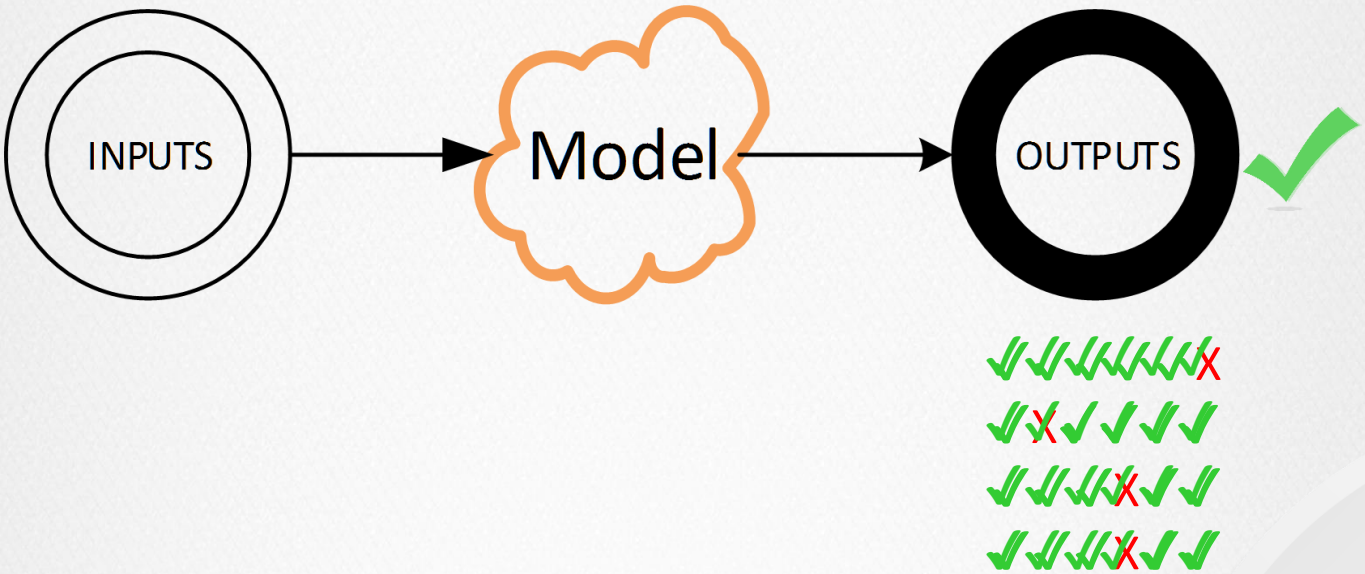
UC Berkeley

2 Big Ideas

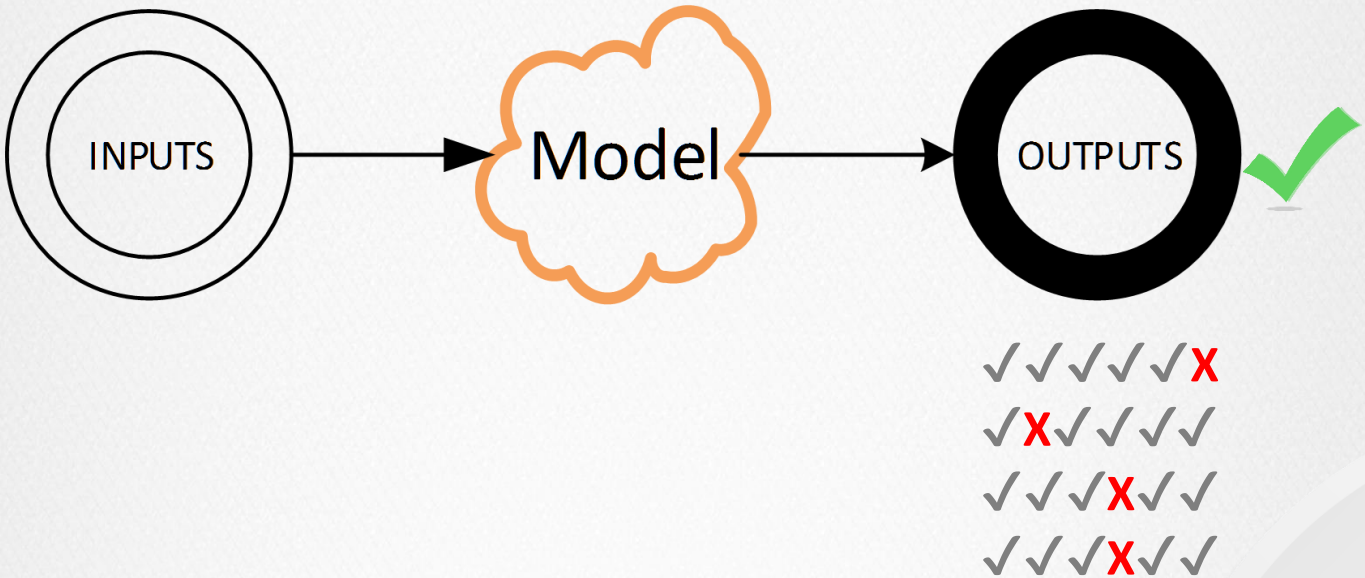
Idea 1



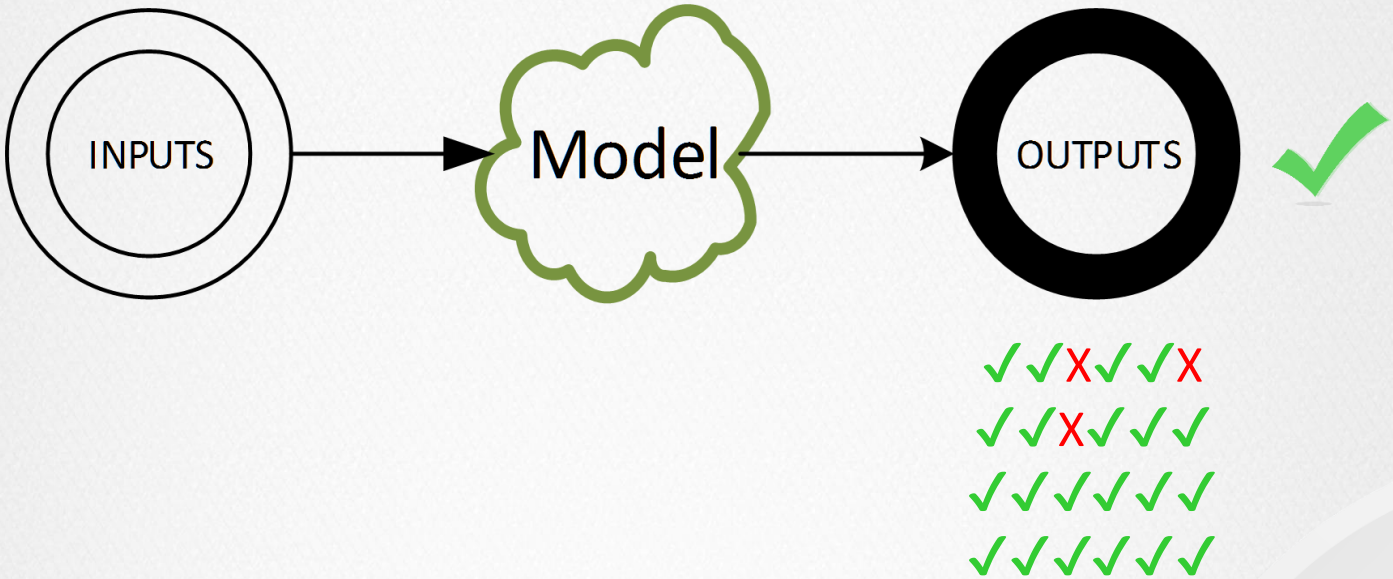
Wisdom of the Crowds



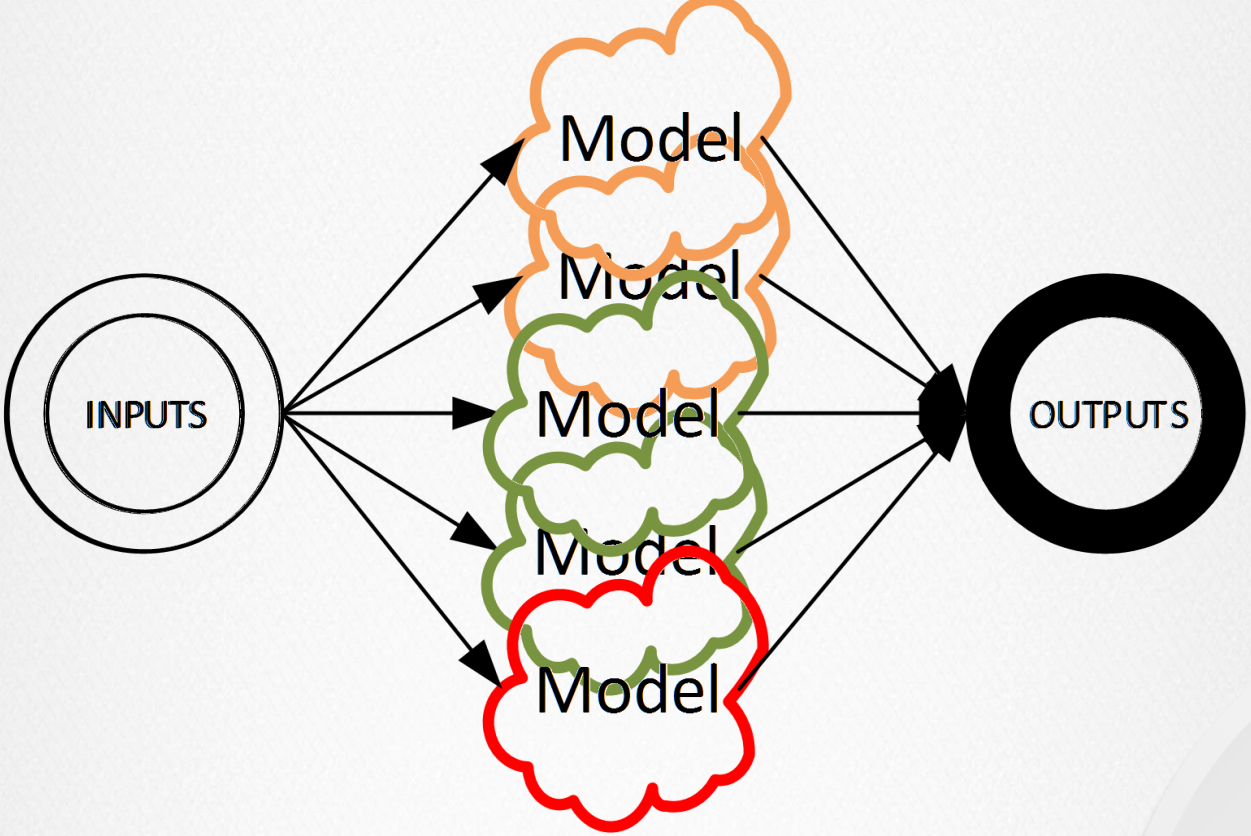
A model fits data well (generally) …



but may provide **poor** estimates for certain observations.



Another ***model*** may fit these observations well.



So why not use 2 models?

Or three?

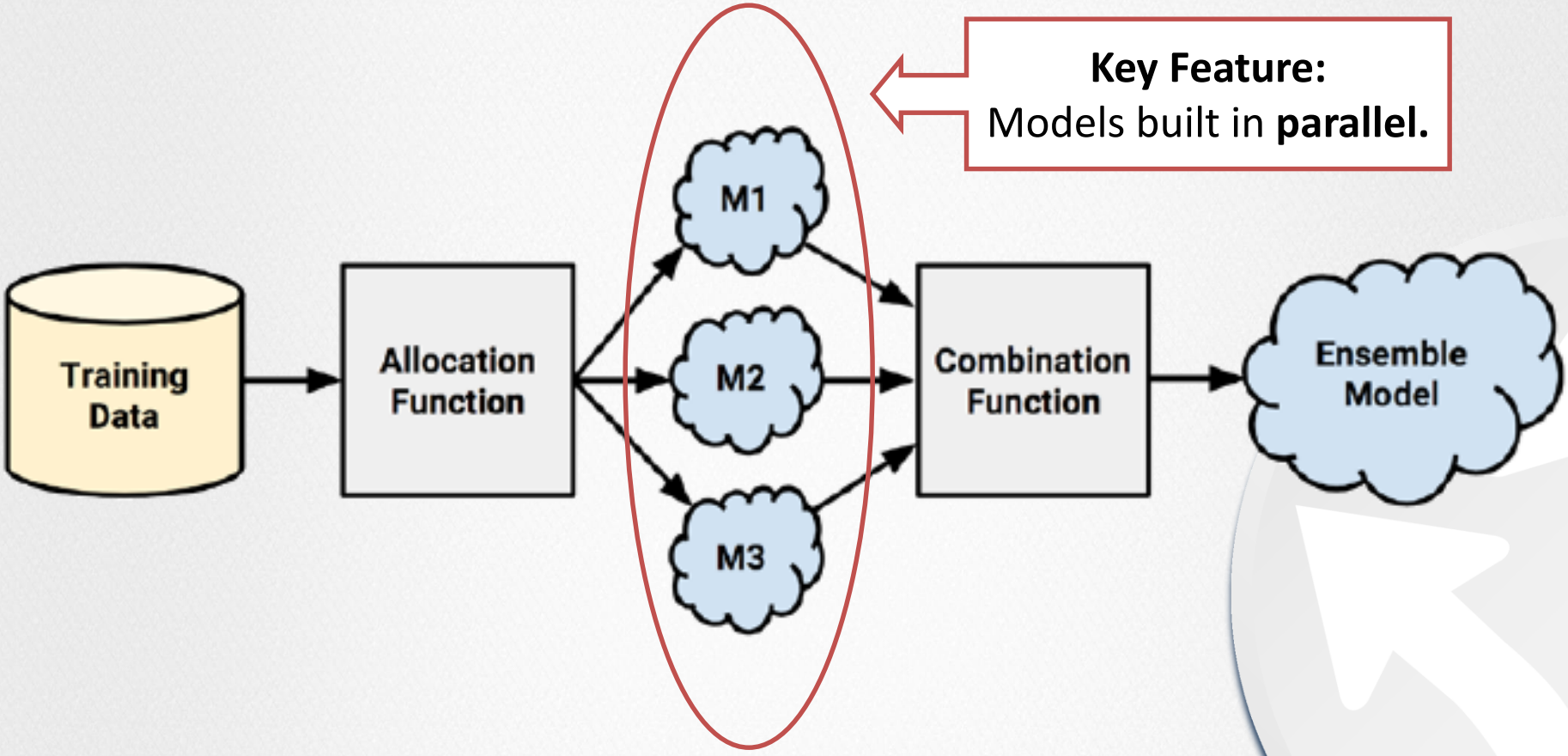
Or twenty?

Wisdom of the crowds

Combine results from multiple models (***ensembles***) to get:

- Better predictions
- Lowers variance for the same model

Ensembles



Source: *Machine Learning with R*

Allocation

- Easiest thing to do?
  - ***All*** data to each model
  - ***Split*** data randomly between models

Are other strategies?

Combination Function

- Easiest thing to do?
  - **Average** model results

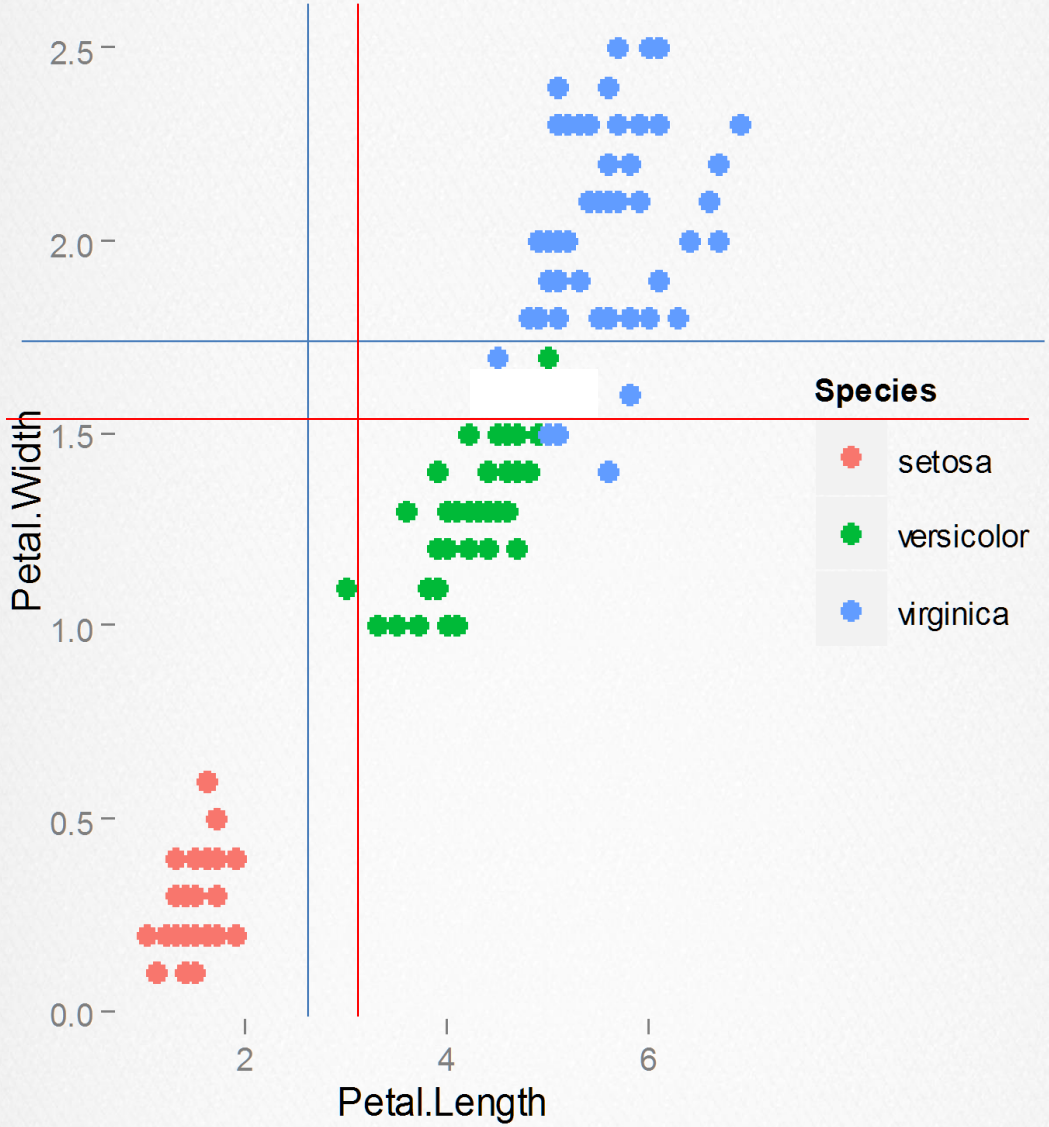
Other strategies?

Bagging Models

- Brieman:
  "Bagging is a general approach that uses bootstrapping in conjunction with any regression (or classification) model to construct an ensemble."

```
for i = 1 to m do
    Generate a bootstrap sample of the original data
    Train an unpruned tree model on this sample
end
```

$$\hat{y} = \frac{\sum_i \hat{y}_i}{m}$$


2.5
2.0
1.5
1.0
0.5
0.0
Petal.Width
2
4
6
Petal.Length
Species
setosa
versicolor
virginica

Bagging Notes

- Advantages
  - Lowers variance
  - Increases stability
  - Has less effect on lower variance models (e.g. linear models)
  - More effective with weak learners
- Disadvantages
  - Computational cost → but parallelizable
  - Reduces Interpretability

RANDOM FORESTS

Random Forest

- Bagged trees
- Consider subset of predictors at each split

```
Select the number of models to build, m
for i = 1 to m do
    Generate a bootstrap sample of the original data
    Train a tree model on this sample
    for each split do
        Randomly select k (< P) of the original predictors
        Select the best predictor among the k predictors and
        partition the data
    end
    Use typical tree model stopping criteria to determine when a
    tree is complete (but do not prune)
end
```

Tuning Parameter

m_{try} : number of predictors to use at each split

- **regression** 1/3rd of number predictors
- **classification** sqrt(number of predictors)

- Kuhn: "Starting with five values of k that are somewhat evenly spaced across the range from 2 to P".

Advantages

- No overfitting
- More trees better (limited by computation time/power only)
- In caret, parameters are considered independently
- Because each learner is selected independently of all previous learners, Random Forests is robust to a noisy response
- Computationally efficient -- each tree built on subset of predictors at each split.
- Use any tree variants as "base learner": CART, ctree, etc

Disadvantages

- Makes Decision Trees harder to interpret
- More time/work to tune the model
  - Number of trees
  - m_{try}

Idea 2

vs

Greed is Bad

Patience is Good

Greed is Bad; Patience is Good

Optimizations/search methods generally wish to arrive at a solution **efficiently**:

- Take as few steps as possible …
- go as far as possible in each step

Greedy

Greed is Bad; Patience is Good

Patience is better:

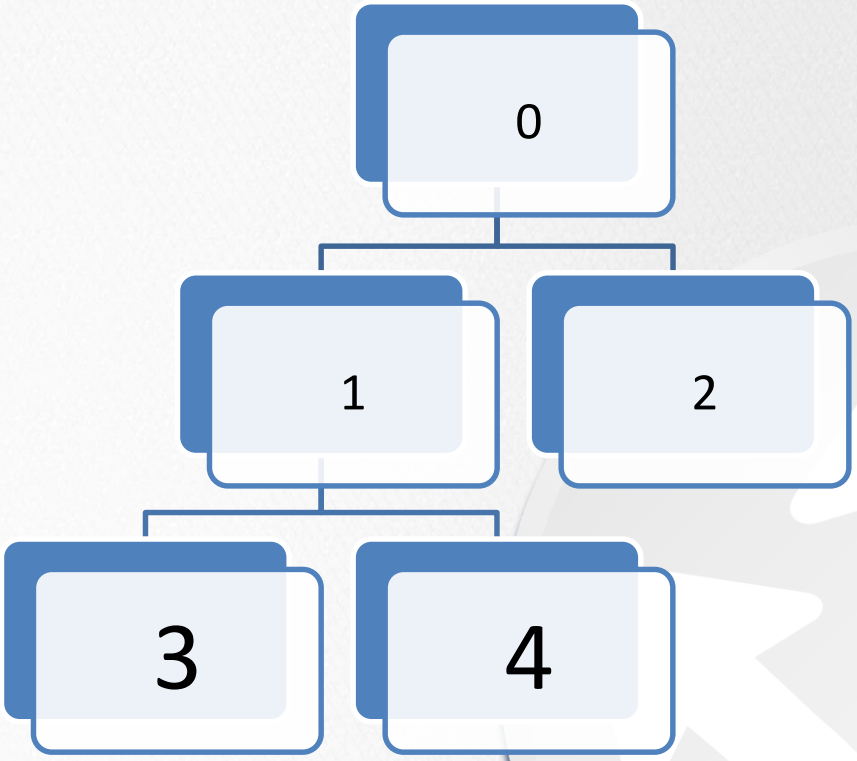
- Slowly approach your solution
- Reconsidered at each step

CART

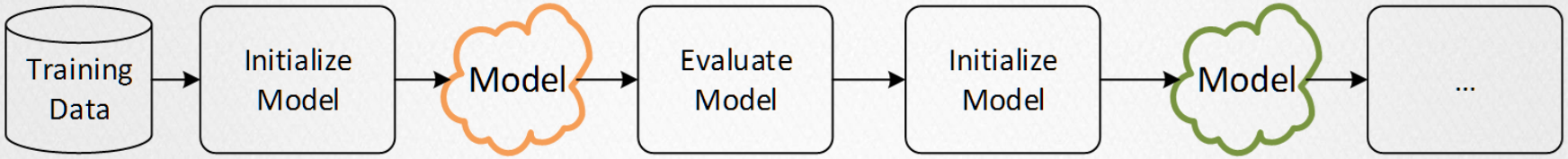
Does CART use patient strategy?

Is a single binary split a model?.

Does the model reconsiders/resets at each iteration?



Greed vs Patience



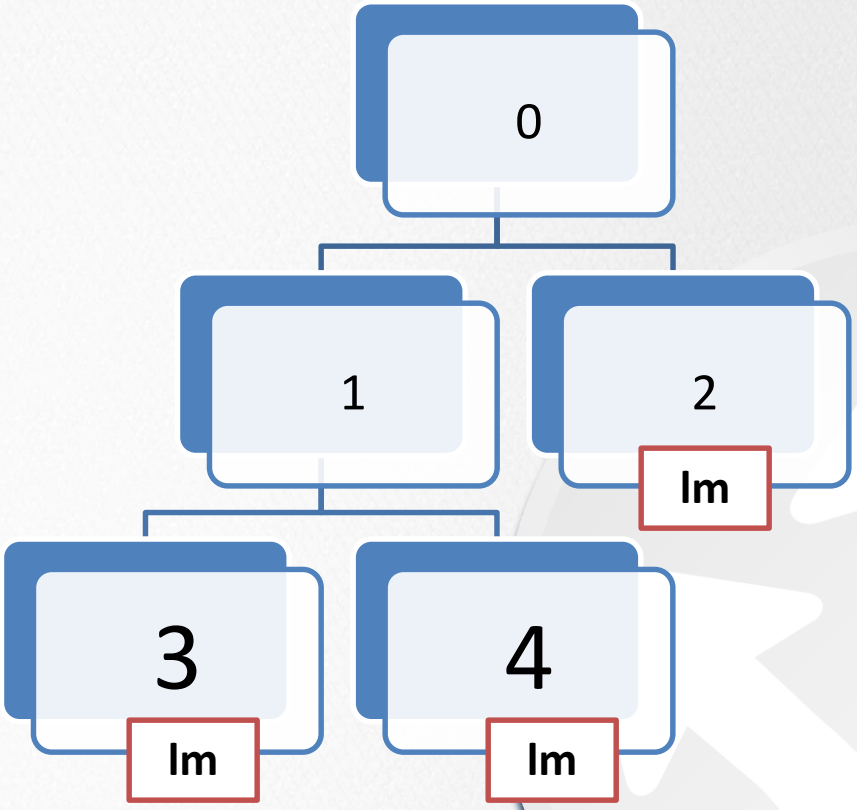
Process is:

- Sequential
- Recursive

Implication for our algorithm

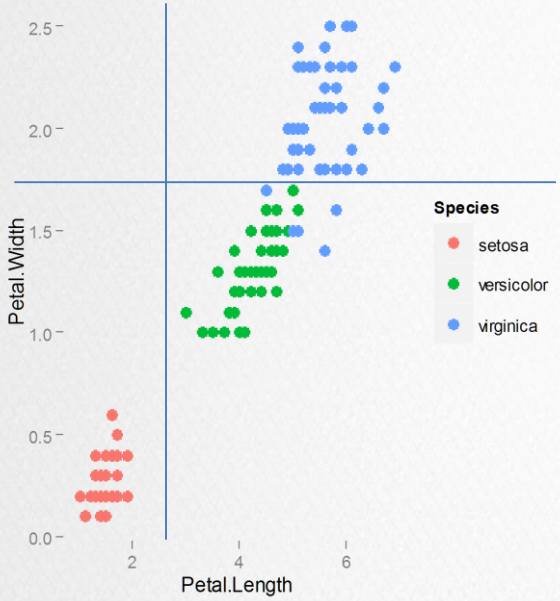
Tree Enhancement: M5

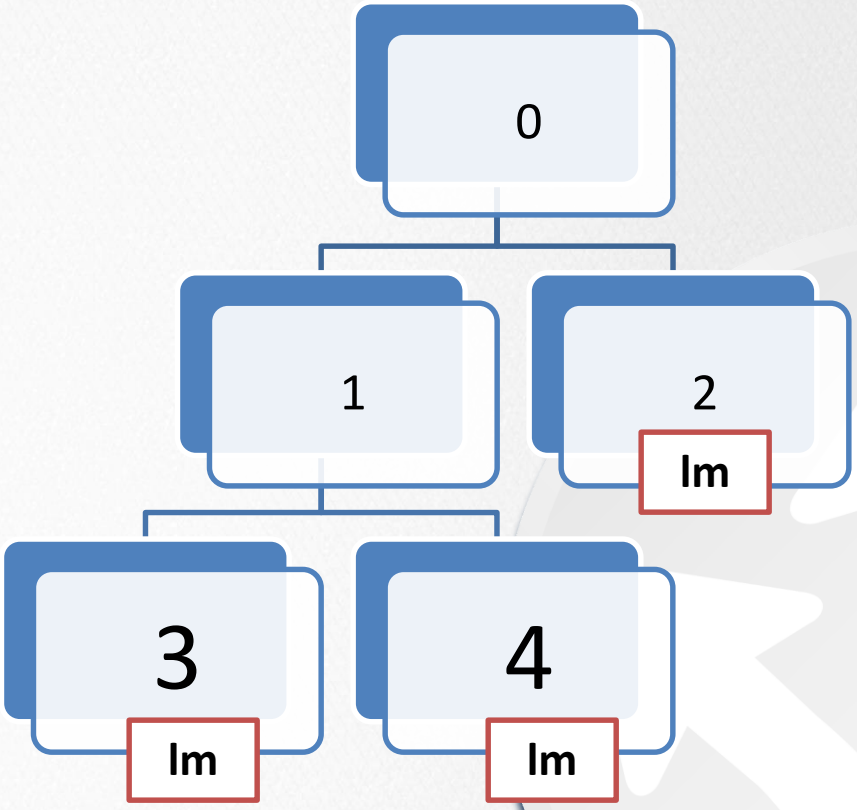
- Having one value represent the entirety of the node leaves information in the node.
- Function in the node is a simple average
- Use something better
  - **M5** put linear models in nodes of trees



M5 Tree Enhancement (cont.)

- **Greed is bad**
  - linear models are built on the residuals of the tree model.
  - Models are recursive





BOOSTING

Boosting

- Single models work;
  - Multiple models work better

- Idea is simple:
  - **Fit first** model: $\hat{y}_1 \sim f_1\ (x)$
  - **Fit** errors/residuals: $\hat{y}_2 = f_2(y - \hat{y}_1)$
    $= f_2\big(y - f_1(x)\big)$
    $= f_2\ (x)$
  - **Iterate:** $\hat{y}_i = (y - \hat{y}_{i-1}) \sim f_i\ (x)$
  - **Predict**: $\hat{y} \sim \sum_i f_i\ (x)$

Boosting
Fit successive series of models each subsequent model to the prior models residuals.

Boosting Notes

- Additive models
- Works best with "weak learners"
  - i.e. ungreedy, low bias, low variance
  - ~~Any~~ Most models with a tuning parameter can be a weak learner
  - Trees are excellent weak learners
    - Weak → "restricted depth"
- Residuals or errors define a gradient
- Interpreted as forward step-wise regression with exponential loss

BOOSTING EXAMPLES

ADABOOST (SHAPIRE/FREUND)

Binary Classification

Let one class be represented with a value of +1 and the other with a value of -1

Let each sample have the same starting weight $(1/n)$

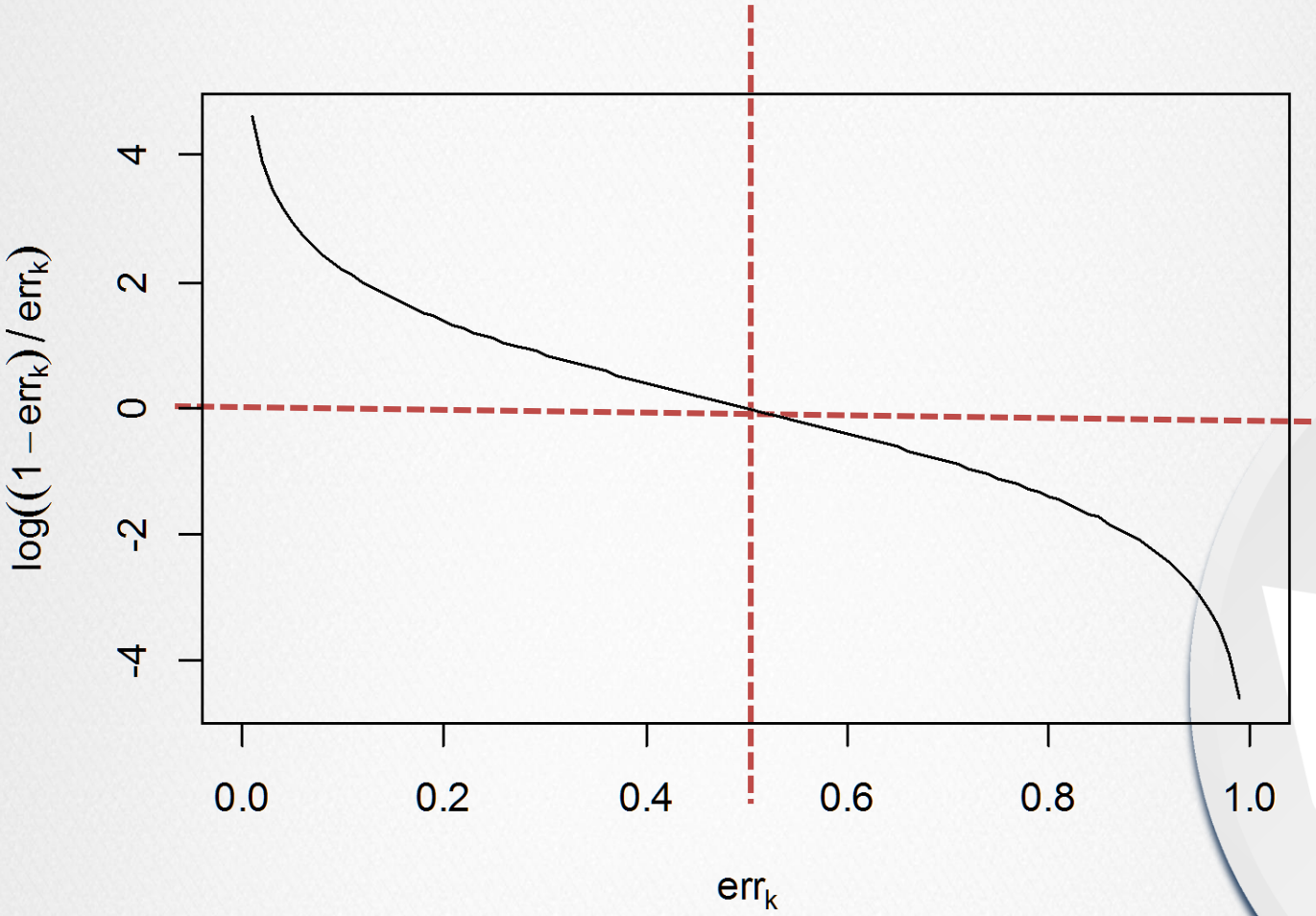
for $k = 1$ *to* K **do**

- Fit a weak classifier using the weighted samples and compute the kth model's misclassification error (err_k)
- Compute the kth stage value as $\ln\left(\left(1 - err_k\right)/err_k\right)$.
- Update the sample weights giving more weight to incorrectly predicted samples and less weight to correctly predicted samples

end

Compute the boosted classifier's prediction for each sample by multiplying the kth stage value by the kth model prediction and adding these quantities across k. If this sum is positive, then classify the sample in the +1 class, otherwise the -1 class.

Adaboost Stage Value



<---------- Model Quality

Adaboost Model Combination

- K models

Individual Effect

$$PV \mid Class = \sum_{K} SV_K * \hat{y}_K$$

Overall Model

Simple Gradient Boosting (regression)

Select tree depth, D, and number of iterations, K

Compute the average response, $\overline{y}$, and use this as the initial predicted value for each sample

Naïve Guess

for $k = 1$ *to* K **do**

  Compute the residual, the difference between the observed value and the *current* predicted value, for each sample

  Fit a regression tree of depth, D, using the residuals as the response

  Predict each sample using the regression tree fit in the previous step

  Update the predicted value of each sample by adding the previous iteration's predicted value to the predicted value generated in the previous step

end

Simple Gradient Boosting Scoring

Models are Additive

$$\hat{y} \sim \sum_{i} f_i\,(x)$$

Simple Sum of the Models
Models are Additive

Simple Gradient Boosting

Benefits

- Better predictive performance
- Lower variance

Limitations

- Less interpretability than base learner
- Can overfit … why?
  - base learners (trees) are greedy
  - model training data used to split and eval node simultaneously.

Stochastic Gradient Boosting Machines

- Proposed by Jerome Freidman (Stanford) circa 1999

M. Kuhn:
"now widely accepted as the boosting algorithm of choice among practitioners"

Simple Gradient Boosting + 2 Innovations

(to fix overfitting problem)

#1

Apply Bootstrap

Like RF

At each iteration, bootstrap training data with **bagging fraction**, *BF*.*

Additional Benefits:

lower variance, improve predication accuracy, reduce computation cost

*Typical value of *BF* is 0.5

#2

Introduce Patience/Limit Greediness

regularization/shrinkage/learning rate/decay parameter λ

Learning Rate, λ

Use Modified Residuals

$$y - \hat{y}$$ "usual" residuals

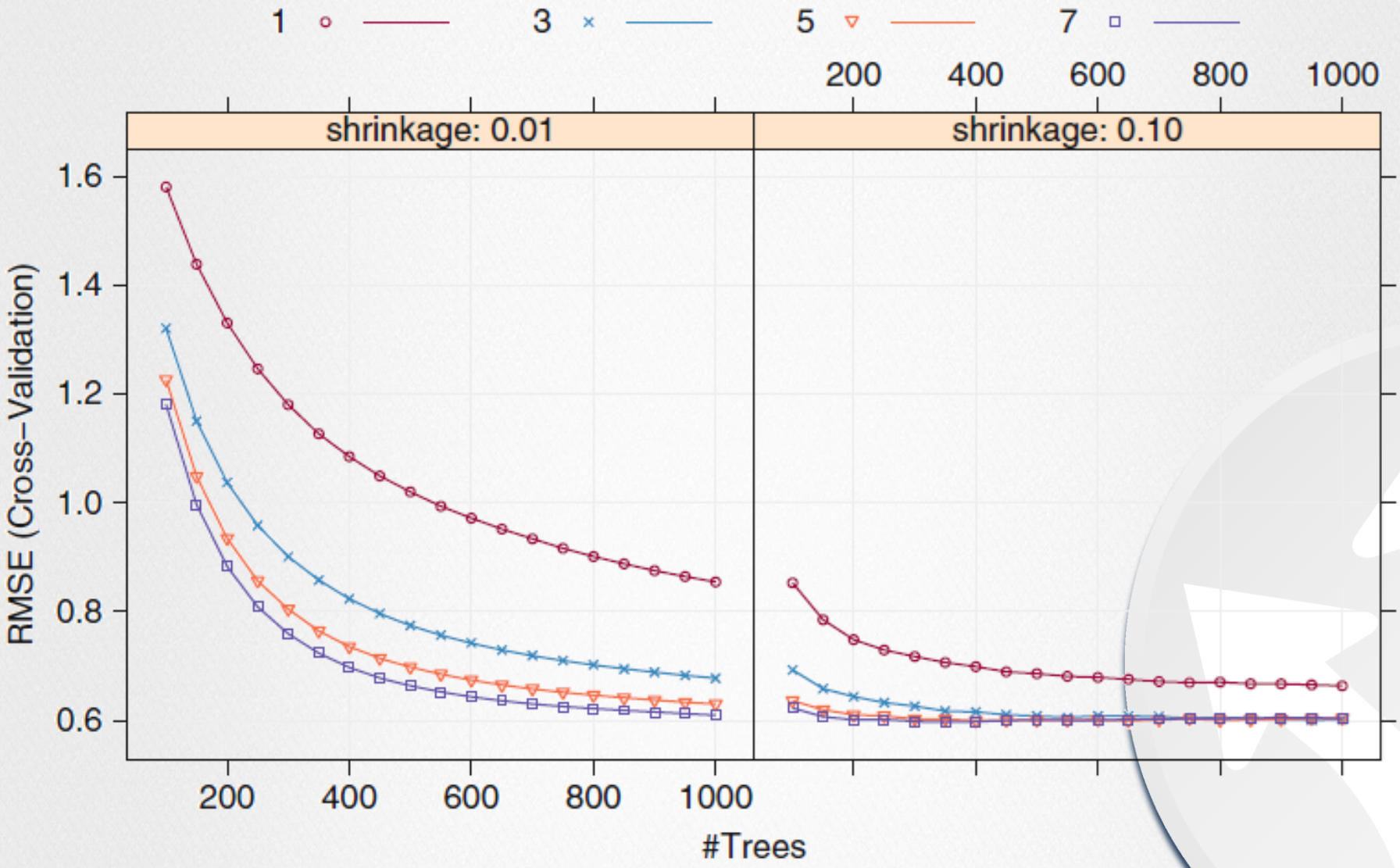
$$y - \lambda\hat{y}$$ "shrunken" residual

$0 < \lambda \leq 1$

For Example

y	$\hat{y}$	$y - \hat{y}$	$y - 0.1\hat{y}$
10	9	1	9.1
9	10	-1	8

Learning Rate leaves more *opportunity for subsequent models*


1
3
5
7
shrinkage: 0.01
shrinkage: 0.10
RMSE (Cross−Validation)
1.6
1.4
1.2
1.0
0.8
0.6
200
400
600
800
1000
#Trees

APM Fig 8.20/

Stochastic Gradient Boosting Scoring

Models are additive

$$\hat{y} \sim \lambda \sum_{i} f_i\,(x)$$

Simple sum of the models, but reduced by the learning rate

Learning Rate, λ

What are the effects of a
small (~0.01) learning rate vs. large rate?

- more opportunity left at each step
- more steps for similar performance
- increased computational costs
  - computational time, storage size
    $\lambda \sim 1/\text{computational time} \sim 1/\text{storage size}$

Stochastic Gradient Boosting Notes

- Can be applied to ***any*** learner
- Yields lower variance, better performing
- Reduced interpretability

Simple Gradient Boosting – Comparison To Random Forest

Similarities	Differences

APPENDIX

Random Forests

Strengths	Weaknesses
• An all-purpose model that performs well on most problems • Can handle noisy or missing data as well as categorical or continuous features • Selects only the most important features • Can be used on data with an extremely large number of features or examples	• Unlike a decision tree, the model is not easily interpretable • May require some work to tune the model to the data

Gradient Boosting (Classification)

Naïve Guess

Initialized all predictions to the sample log-odds: $f_i^{(0)} = \log \frac{\widehat{p}}{1-\widehat{p}}$.

for *iteration* $j = 1 \ldots M$ **do**

Cor

Ran

Tra ... residuals as the

Compute the terminal node estimates of the Pearson residuals:

$$r_i = \frac{1/n \sum_i^n (y_i - \widehat{p}_i)}{1/n \sum_i^n \widehat{p}_i(1-\widehat{p}_i)}$$

Update the current model using $f_i = f_i + \lambda f_i^{(j)}$

end

!

This is Algorithm 14.3 from APM. The caption says simple gradient boosting but is the recipe for stochastic gradient boosting.

Stochastic Gradient Boosting

- Simple Gradient Boosting can overfit (greedy)
  - Apply "regularization/shrinkage"
    - Use λ ("Learning Rate")
      Rather than add the entirety of the residuals, add a fraction of the residuals at each iteration.

$$\hat{y} \sim \lambda \sum_i f_i\,(x) \qquad 0 < \lambda \leq 1$$

    - Small values for λ (~0.01) work best
    - λ ~ 1/computational time ~ 1/storage size

- Use bagging, as well
  - Bagging Fraction: a sample of data in each loop iteration