DECISION PATTERNS

Model Improvements

Practical Machine Learning (with R)

UC Berkeley

Spring 2016

Topics

- Administrativa
  - Role Call
  - Assignments due to github
- Review/Expectations
  - Readings
  - Previous Lecture
- New Topics

REVIEW AND EXPECTATIONS

Resampling Methods

- Get more accurate estimation of a statistic/value by resampling methods

→ Improves function estimate

Model Performance Evaluation

- Understand model performance metrics
- How they are calculated …

Classification

- Metrics: Accuracy, Kappa
- Graphical
  - `caret::confusionMatrix`
  - `pROC::ROC`

Regression

- RMSE, MAE, etc.
- Graphical
  - Histogram of errors
  - $y \; vs \; \hat{y}$ scatterplot

Decision Trees

- All about splitting
  - Different variants have different rules for evaluating the splits
- Tree = Ruleset = Partition of Space
  - Node = Rule = "box" (contiguous region of space)

Tree Variants

- There are many tree variants

- Tweaks
  - change how splits are determined
  - when to stop growing the tree
  - how the node value is determined

RULES

- As derived from trees often have repeated conditions

M5rules, C4.5Rules

```
NumCarbon > 3.777 &
SurfaceArea1 > 0.978 &
SurfaceArea1 > 8.404 &
FP009 <= 0.5 &
FP075 <= 0.5 &
NumRotBonds > 1.498 &
NumRotBonds > 1.701
```

Rules and their conditions live on their own, conditions can be adjusted to help bias-variance trade-off

Missing Data

- Missing values in predictors are common
- A split determines which observations go to the LHS and RHS. How to Handle `NA`s?
- `NA_Categorical`
  - Treat as separate category
- `NA` (in general)
  - Use **Surrogate Splits**

Surrogate Splits

- Tree is built ignoring missing data
  - Any record with incomplete data (response or predictor) is rejected -or-
  - Missing data is not used for split-point determinations
- Variables are often *collinear* → splits are similar and send variables down the same path.
  - Choose a surrogate (variable) split that best approximates the chosen split (accuracy)
  - Very often this is also a good split

Review and Expectation

Understand **Recursive Partitioning** :

- intrinsically how recursive partitioning models work; how splits are determined; what splitting accomplishes
- how model tuning parameters control for bias variance trade-offs
- what `cp` is and how to use it to prune trees to the proper size

Tree Method Advantages

- List em

Review and Expectation

Use **`rpart`** and/or ctree

- Build Recursive Partitioning Models using `rpart`
- Prune trees to statistically relevant size
- Plot `rpart` trees using
  - `party::as.party`
  - `map.tree::drawtree`

Understand Caret

- Use the **caret** package and the `train` function to build models
- Understand the difference between using `caret` and building models manually. What `caret` provides.
- Control how models are built using the `train & trainControl` functions
- How to extract the final model
- How to plot the tuning parameters, performance curves

QUESTIONS

IMPROVING MODELS

Tree Disadvantages

- List em

Tree Disadvantages

- Model instability (sensitive to data)
  - Derives from each subsequent split is dependent on prior splits
- Less than optimal predictive performance
  - Rectangular regions
- Limited number of outcome values <= number of terminal nodes
- Intra-node high and low values not fit well
- Selection bias toward predictors with higher number of distinct values
- Tuning parameter, C_p
- Splits of correlated variables ambiguous
- Treatment of missing values

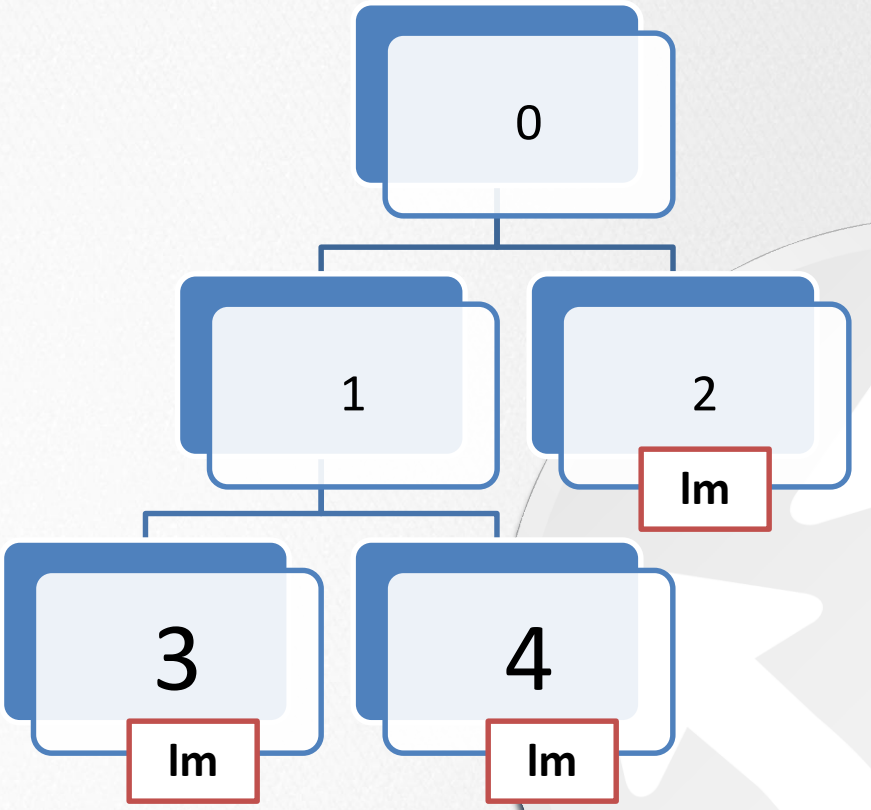
Two Big Ideas

- **Wisdom of the crowds**
  It is better to make estimates from multiple models (***ensembles***) than individual models
  - Better predictions
  - Lower variance for the same model

- **Greed is bad**. **Patience is good.**
  It is better to slowly approach your solution than arrive at an answer directly.
  - More accurate solutions

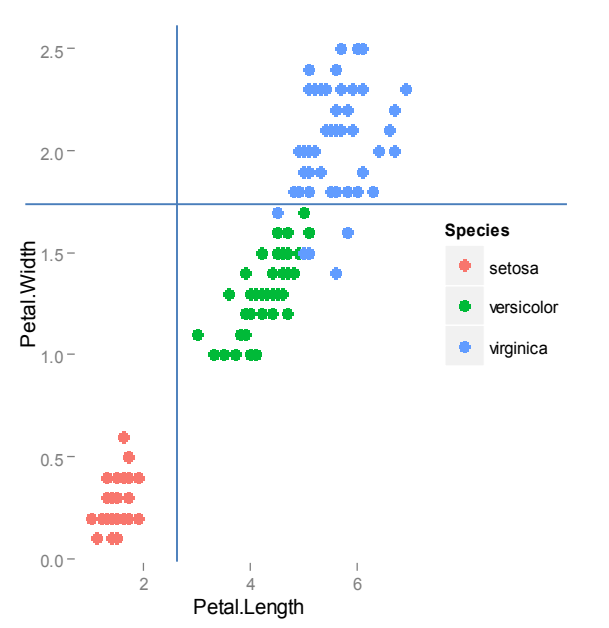
Tree Enhancement: M5

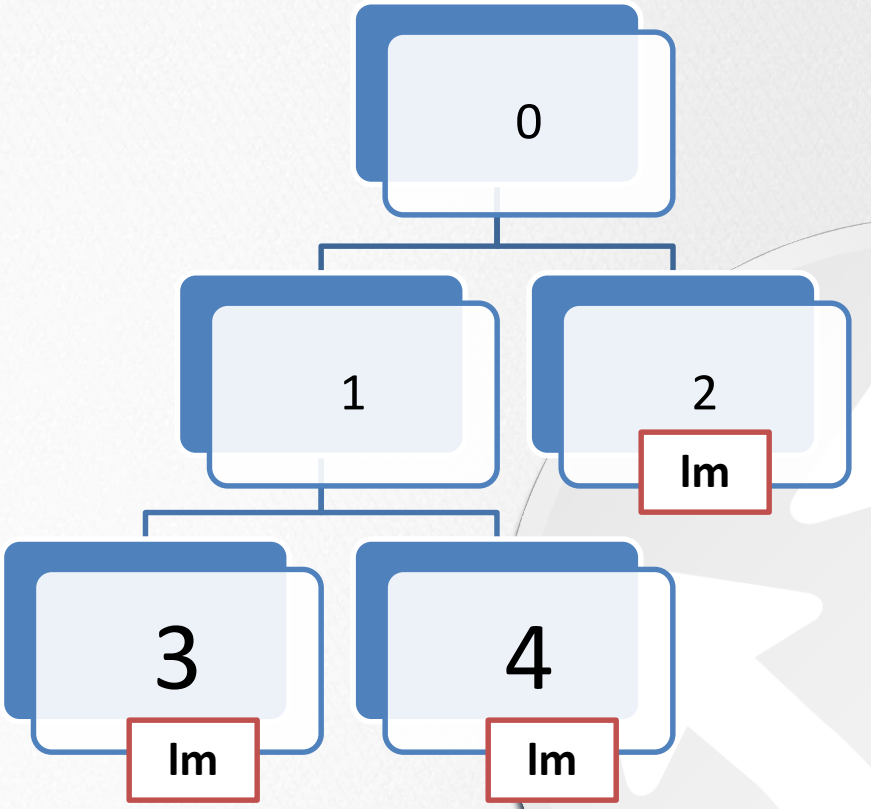
- **Wisdom of the Crowd!**
- Having one value represent the entirety of the node leaves information in the node.
- Function in the node is a simple average
- Use something better
  - **M5** put linear models in nodes of trees



M5 Tree Enhancement (cont.)

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



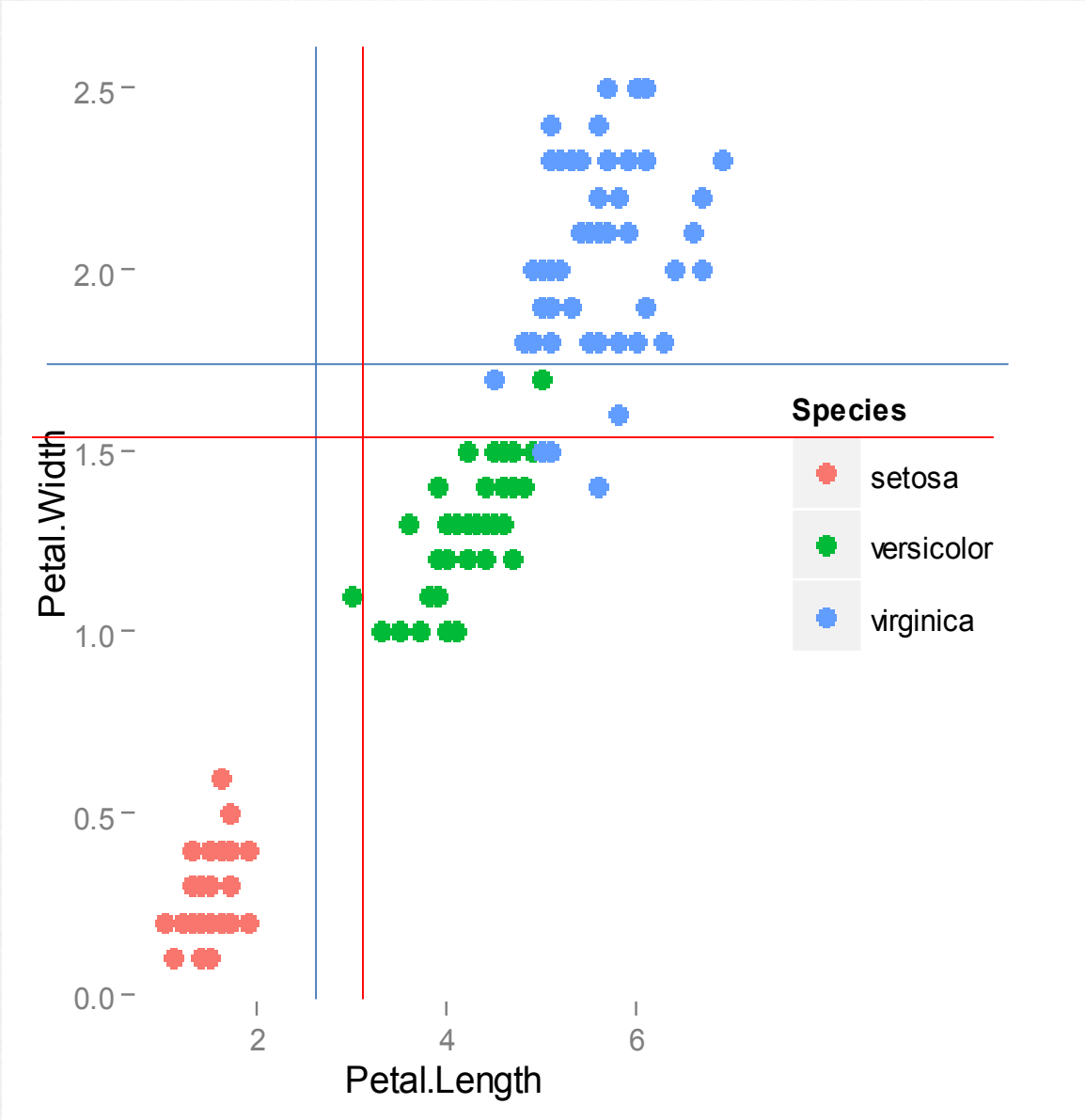


Bagging Models

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

```
1 for i = 1 to m do
2 |   Generate a bootstrap sample of the original data
3 |   Train an unpruned tree model on this sample
4 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

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

Random Forest

- **Wisdom of the Crowds:** Bagging
- **Greed is bad: consider subset of predictors at each split**

```
1  Select the number of models to build, m
2  for i = 1 to m do
3      Generate a bootstrap sample of the original data
4      Train a tree model on this sample
5      for each split do
6          Randomly select k (< P) of the original predictors
7          Select the best predictor among the k predictors and
           partition the data
8      end
9      Use typical tree model stopping criteria to determine when a
       tree is complete (but do not prune)
10 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

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 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

Simple Gradient Boosting

1. Select tree depth, D, and number of iterations, K
2. Compute the average response, $\overline{y}$, and use this as the initial predicted value for each sample
3. **for** $k = 1$ *to* K **do**
    4. Compute the residual, the difference between the observed value and the *current* predicted value, for each sample
    5. Fit a regression tree of depth, D, using the residuals as the response
    6. Predict each sample using the regression tree fit in the previous step
    7. Update the predicted value of each sample by adding the previous iteration's predicted value to the predicted value generated in the previous step
8. **end**

Simple Gradient Boosting – Comparison To Random Forest

Similarities	Differences

Stochastic Gradient Boosting

- Gradient Boosting Susceptible to Overfitting
  - 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 time

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

APPENDIX


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