INFO 251: Applied Machine Learning

# Trees and Forests


Thanks to machine-learning algorithms, the robot apocalypse was short-lived.

# Key Concepts (last lecture)

- Churn prediction
- Decision boundaries
- Hyper-rectangles
- Splitting
- Information gain
- Recursive tree building
- Overfitting trees
- Pruning trees

# Course Outline

- Causal Inference and Research Design
  - Experimental methods
  - Non-experiment methods
- Machine Learning
  - Design of Machine Learning Experiments
  - Linear Models and Gradient Descent
  - **Non-linear models**
  - Neural models
  - Unsupervised Learning
  - Practicalities, Fairness, Bias
- Special topics

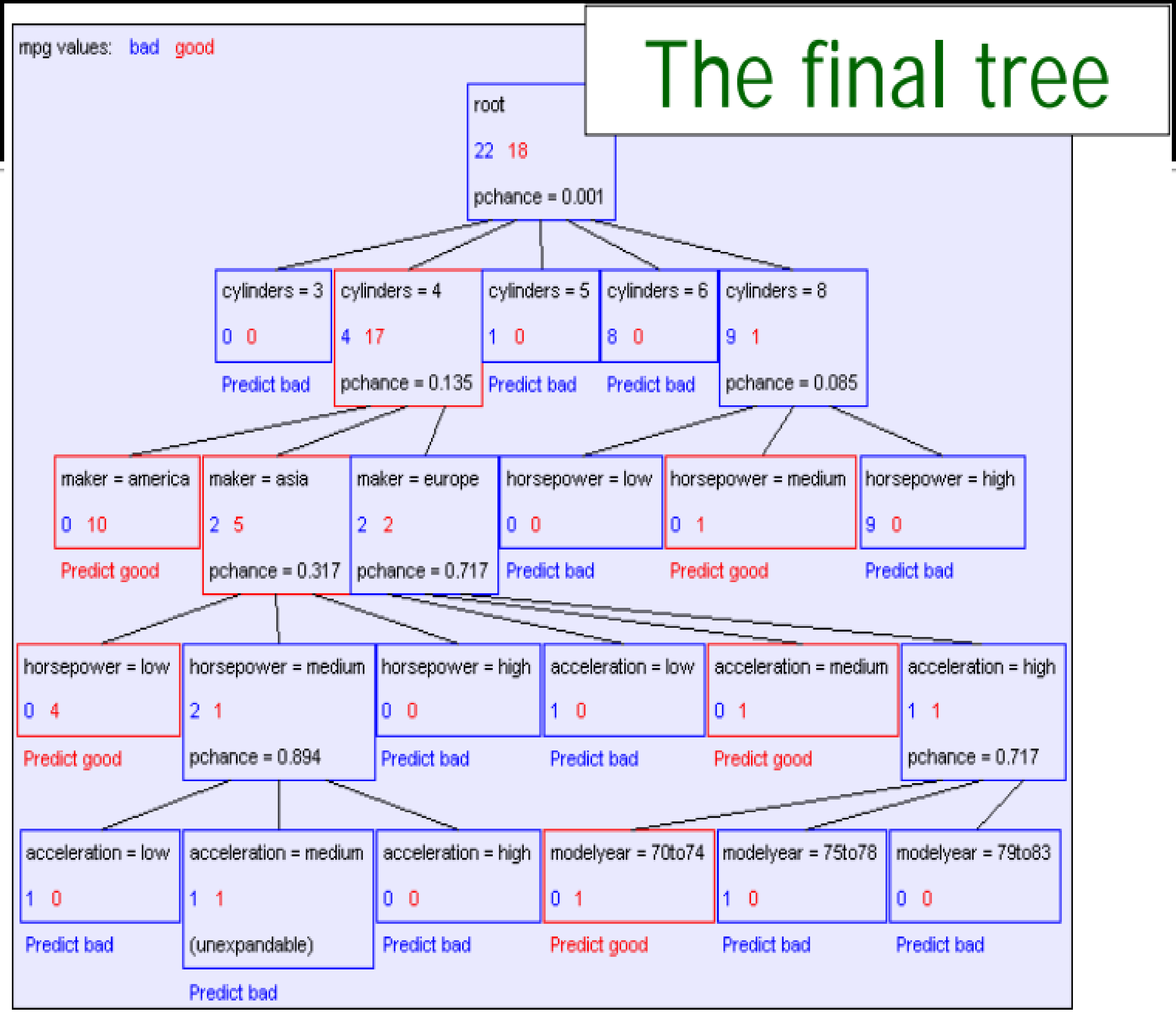
# Outline

- Decision Trees: Loose ends
  - **Overfitting and Pruning**
  - Extensions
- Regression Trees
- Random Forests
- Feature Importance

# Recap: Decision Tree algorithm

```
GrowTree(S):
    if y==0 for all <x,y> in S:
        return new leaf(0)
    else if y==1 for all <x,y> in S:
        return new leaf(1)
    else:
        xj = max_info_gain(S)
        S0 = all <x,y> in S with xj==0
        S1 = all <x,y> in S with xj==1
        return new node(xj, GrowTree(S0), GrowTree(S1))
```

# Overfitting



# Overfitting

- Overfitting strikes again:

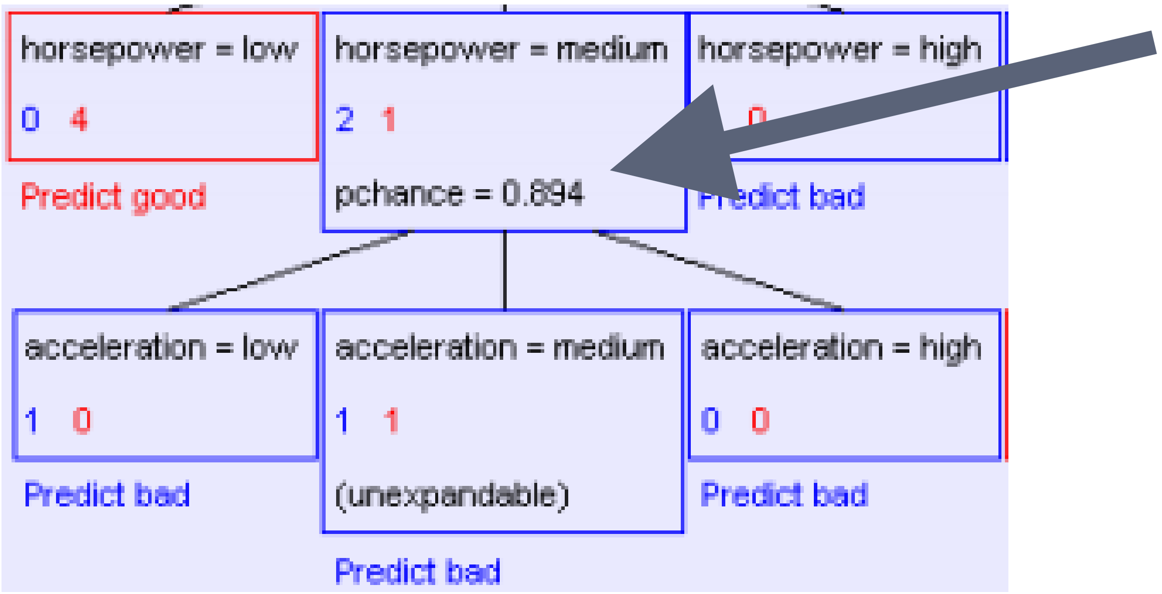
| | Num Errors | Set Size | Percent Wrong |
|---|---|---|---|
| Training Set | 1 | 40 | 2.50 |
| Test Set | 74 | 352 | 21.02 |

- How to deal with overfitting in Regression?
  - Regularization
- K-Nearest Neighbors?
  - Increasing K
- Naïve Bayes?
  - Smoothing

# Overfitting in Decision Trees

- Three common solutions:
  1. Stop growing tree when split is not statistically significant
  2. Grow tree, then prune afterwards
  3. Set maximum depth

# Example: Over-splitting




- Should we really split here?
- Only 3 relevant training examples
- The resulting distributions are likely due to chance

- **One solution**: compute the value/ significance of each split
  - For instance, a chi-squared test, or info gain
- Only split if value exceeds some threshold

$$\chi^2 = \sum_{i=1}^{n} \frac{(O_i - E_i)^2}{E_i}$$

# Pruning

- Build the full decision tree
- Starting with the deepest nodes, delete splits where value of split does not exceed some threshold T
- Continue upward until no more prunable nodes

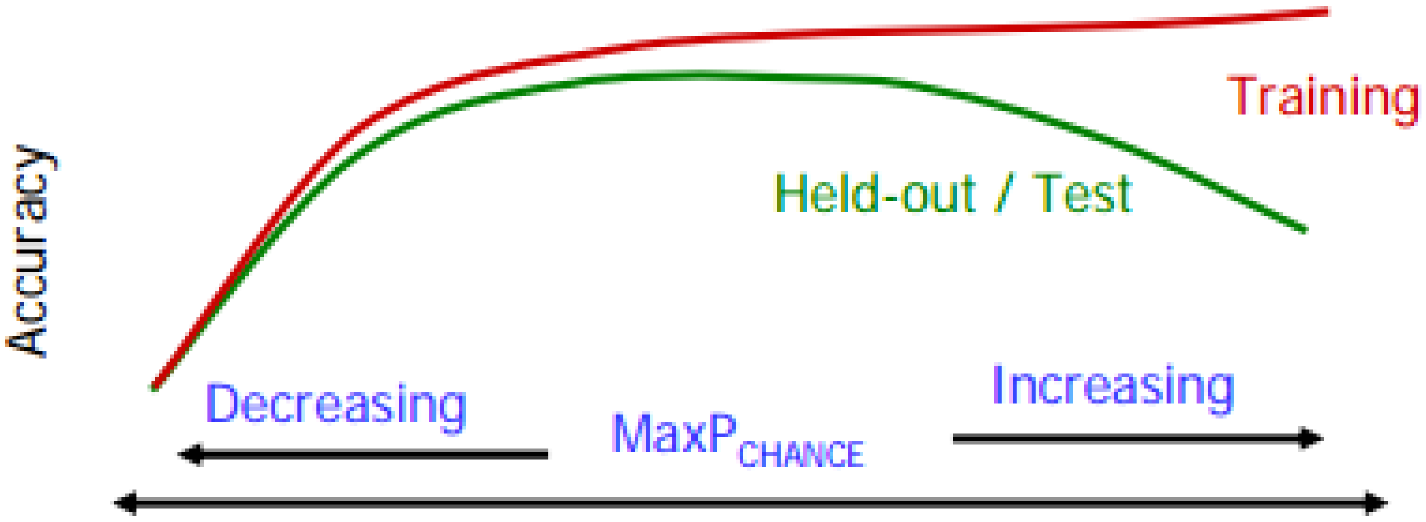
| | Num Errors | Set Size | Percent Wrong |
|---|---|---|---|
| Training Set | 1 | 40 | 2.50 |
| Test Set | 74 | 352 | 21.02 |

→

| | Num Errors | Set Size | Percent Wrong |
|---|---|---|---|
| Training Set | 5 | 40 | 12.50 |
| Test Set | 56 | 352 | 15.91 |

# Regularization

- T is a regularization (hyper-)parameter
  - How to determine value?
- Cross-Validation!

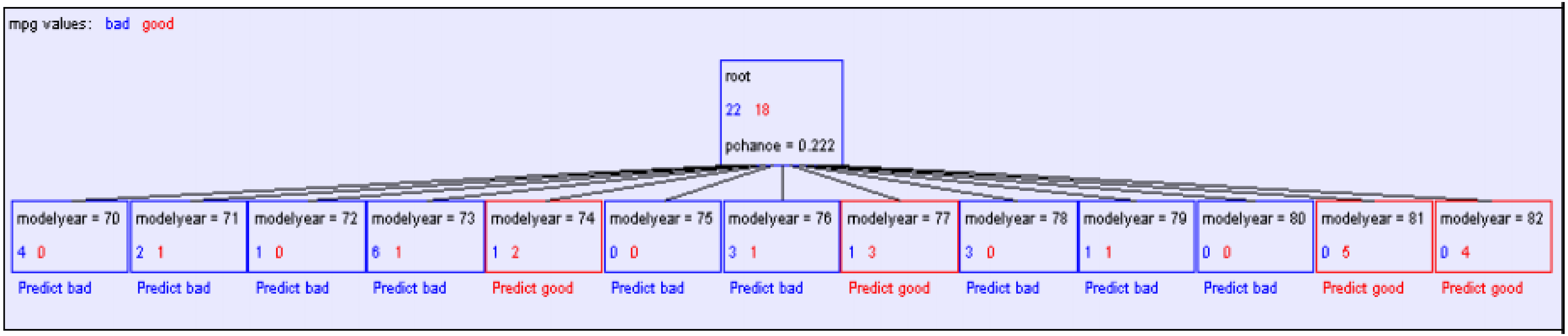


# Outline

- Decision Trees: Loose ends
  - Overfitting and Pruning
  - **Extensions**
- Regression Trees
- Random Forests
- Feature Importance

# Multi-valued features

- Features with many discrete values:
  - Splits with many children (comparing Info Gain?)
  - Can produce degenerate cases
  - Common solution: One vs. all other values

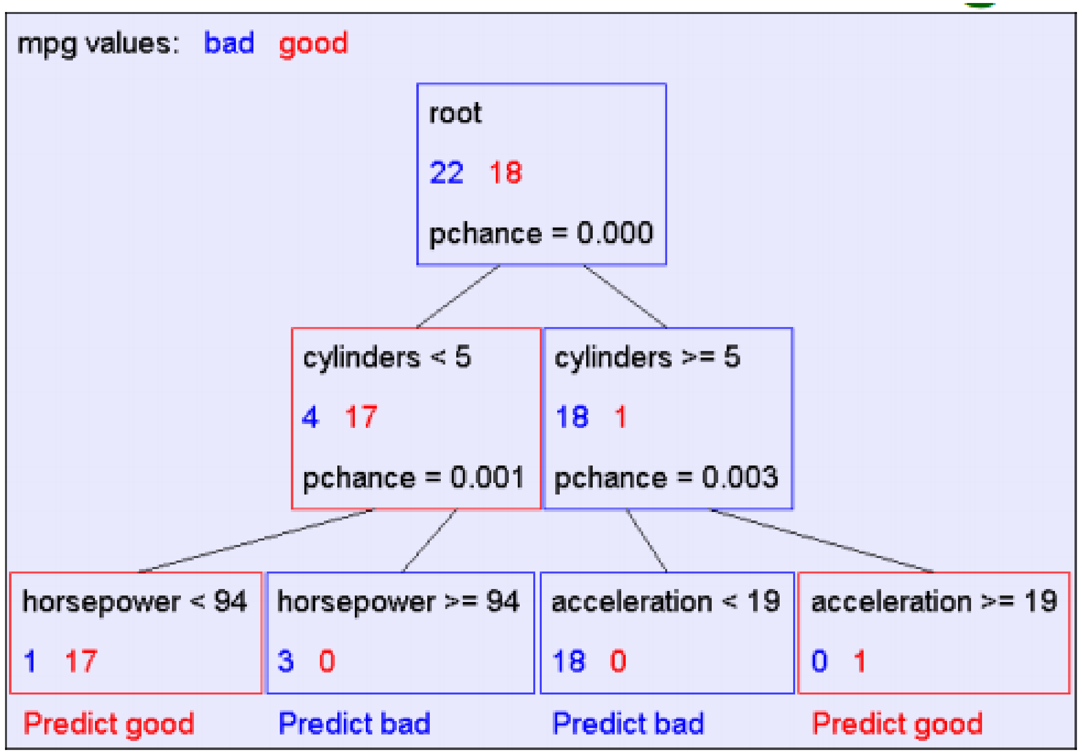


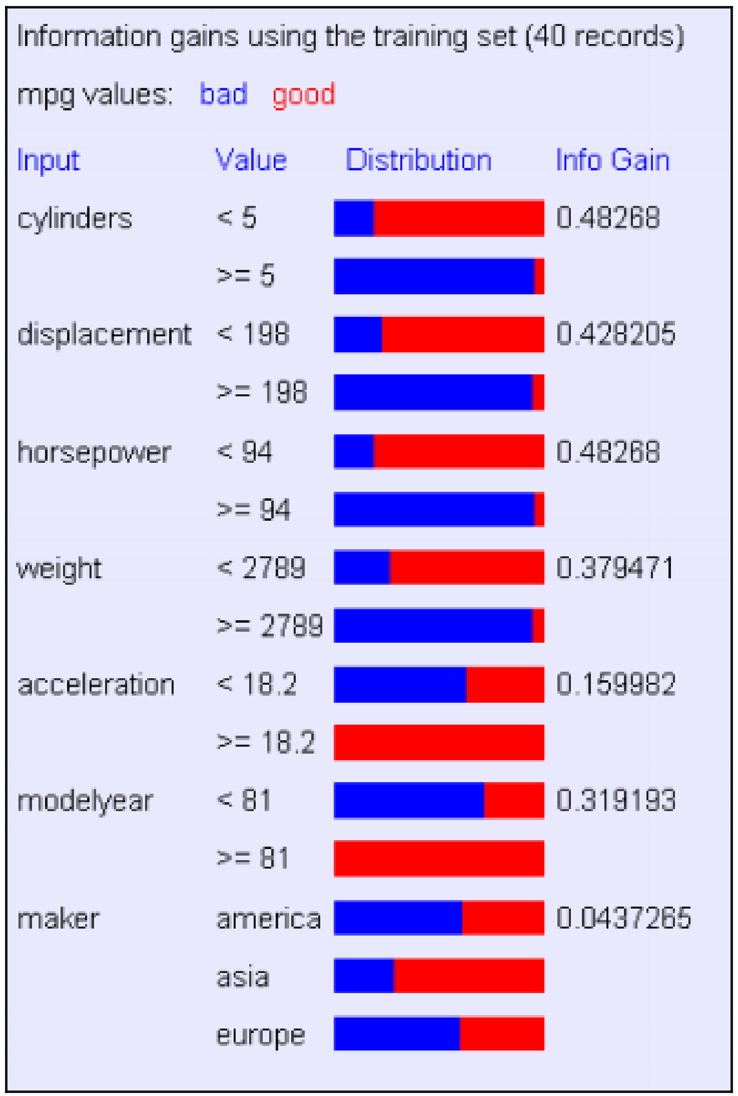
# Continuous features

- Continuous features
  - Common solution: Bucket or threshold values
  - E.g., model years <1970, 1970-1980, >1980
- How to choose the buckets/thresholds?
  - Sort instances based on value of an attribute (e.g. year)
  - Identify adjacent examples that differ in their label
    - This generates a set of candidate thresholds splitting thresholds for that feature
  - Use information gain to decide appropriate threshold

# Thresholded splits

- Bucketing example:
  - Creates deeper, denser tree (for same value of T)

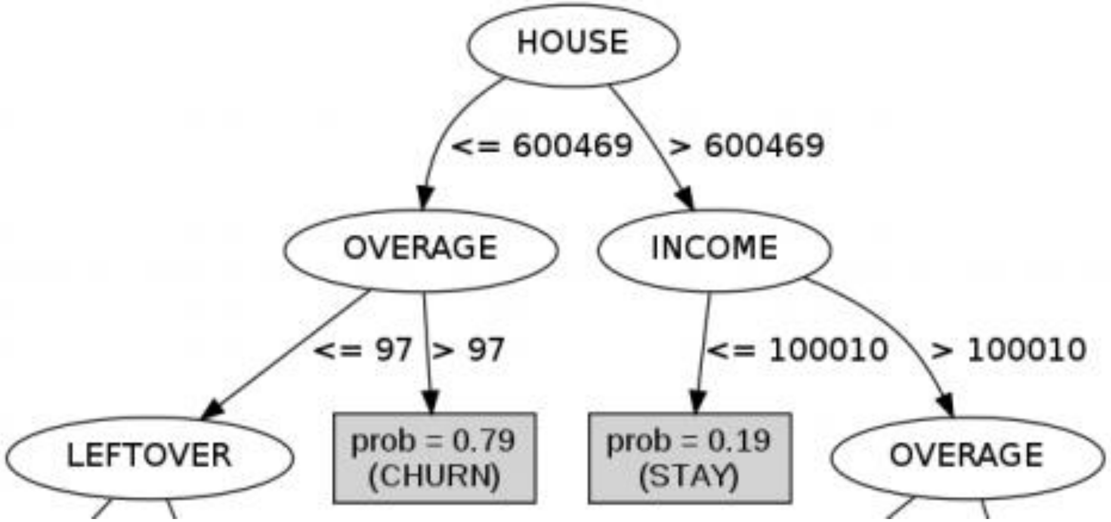



| | Num Errors | Set Size | Percent Wrong |
|---|---|---|---|
| Training Set | 1 | 40 | 2.50 |
| Test Set | 53 | 352 | 15.06 |

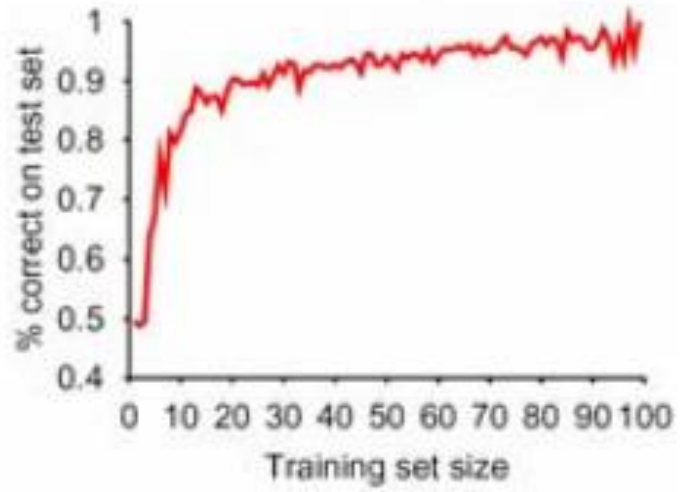
# Output probabilities

- How to do better than predicting majority class?
- Estimate probabilities from the relevant examples at each node
- Can use smoothing to improve estimates (e.g., Laplace smoothing)



# Scaling up

- More data is almost always better




- Scaling up with standard recursive algorithms can be hard
- New algorithms make single pass through data
  - E.g. Very Fast Decision Trees (VFTD)

# Outline

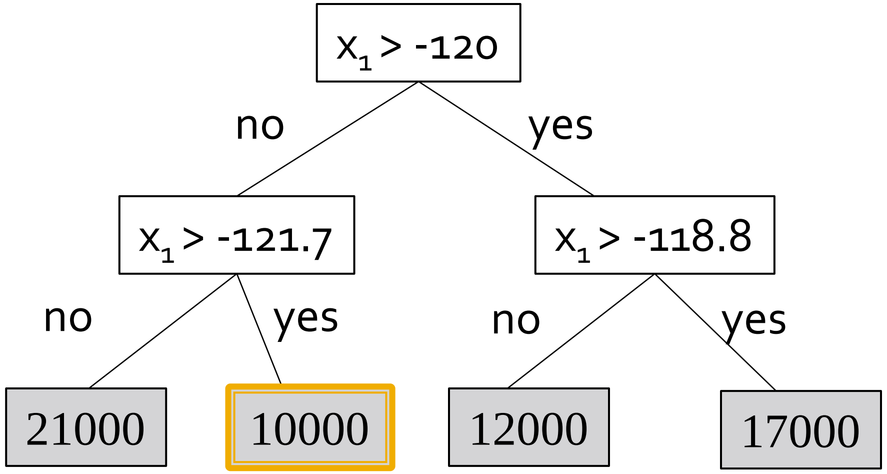
- Decision Trees: Loose ends
  - Overfitting and Pruning
  - Extensions
- **Regression Trees**
- Random Forests
- Feature Importance

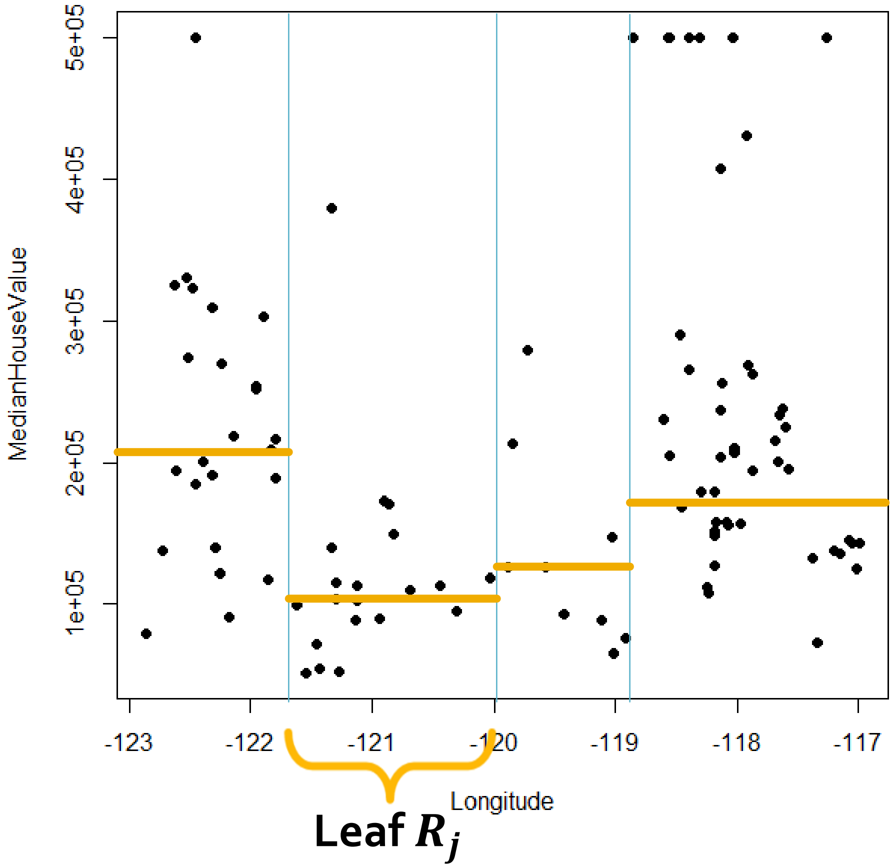
# Regression Trees

- What if output values are continuous or real-valued (i.e., not discrete)?
- Regression trees
  - Construct binary tree, minimize error in each leaf
  - Before, we counted # elements of each type in leaf
  - Now we choose predicted value that minimizes error
- Example: Predict median housing value based on a house's location (latitude, longitude)

# Regression Trees

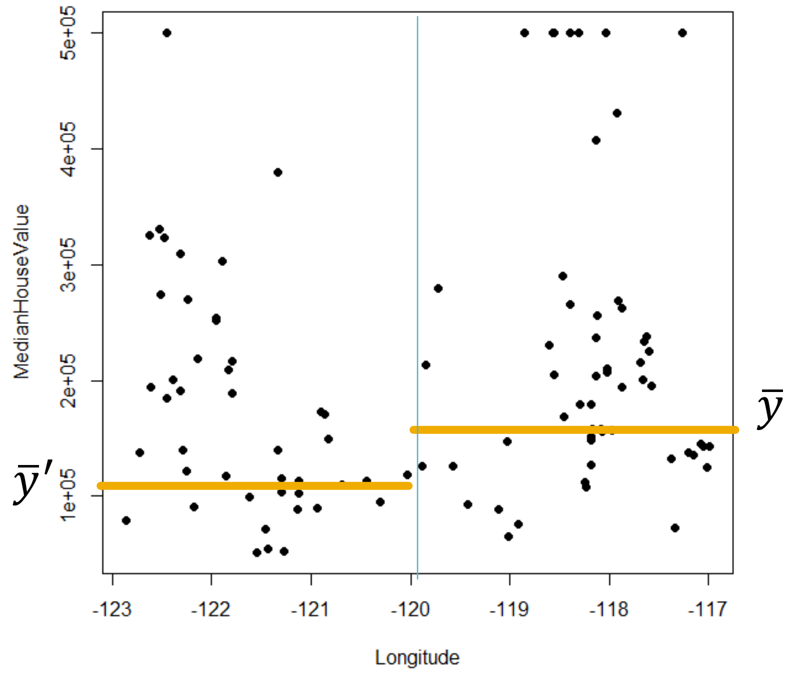
```
> head(calif[,c(1,8,9)])
  MedianHouseValue Latitude Longitude
1           452600    37.88   -122.23
2           358500    37.86   -122.22
3           352100    37.85   -122.24
4           341300    37.85   -122.25
5           342200    37.85   -122.25
6           269700    37.85   -122.25
```



# Regression Trees

- How to choose split point?
  - Idea: Minimize prediction error
- In 1-dimension: choose *s* to minimize
  - $\min_{\bar{y}} \sum_{i:x_i>s} (\bar{y} - y_i)^2 + \min_{\bar{y}'} \sum_{i:x_i \leq s} (\bar{y}' - y_i)^2$
  - Consider finite splits (e.g. *s* between data)

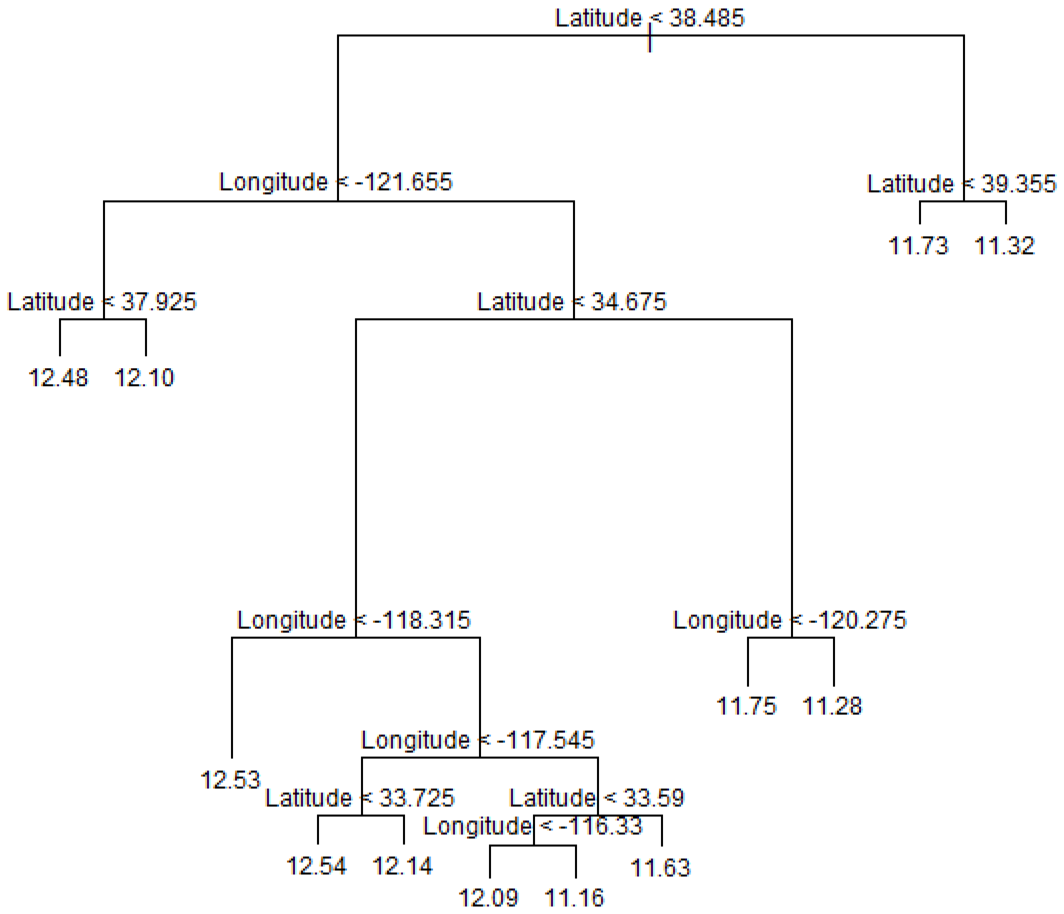


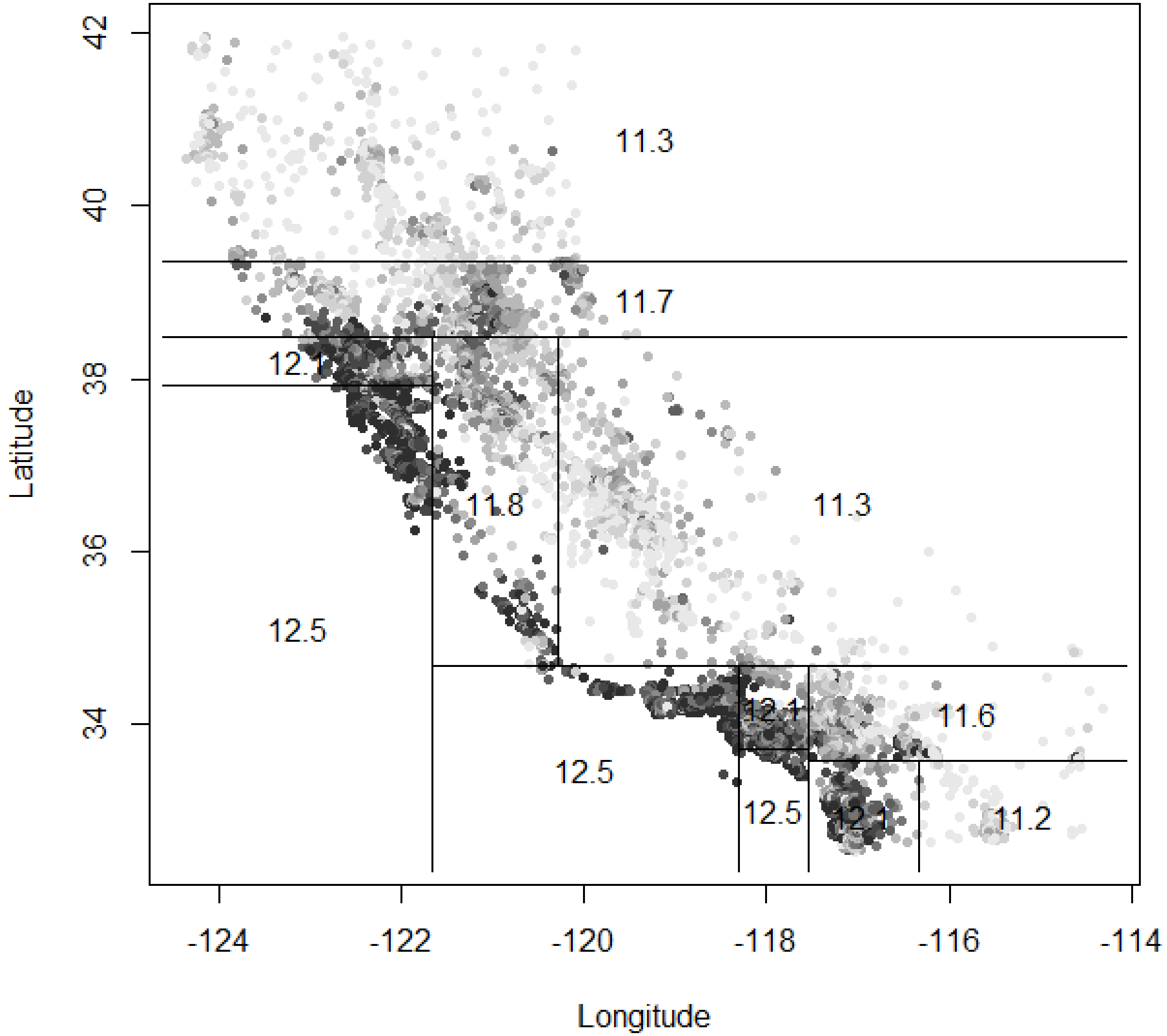

- This intuition generalizes trivially to *D* dimensions

# Regression Trees: Recursive Algorithm

1. Start with a single node ($c_j$) containing all points.
   1. Calculate predicted value $\bar{y}_{c_j} = \frac{1}{n}\sum_{i \in c_j} y_i$
   2. Calculate total error: $J = \sum_{c_j}\sum_{i \in c_j}\left(\bar{y}_{c_j} - y_i\right)^2$
2. If all points in the node have the identical features, stop.
   - Otherwise, search all binary splits of all variables for split that most reduces $J$
   - Stop if $J$ decreases less than $\delta$ or if nodes are close to empty
   - Otherwise, make that split, creating two new nodes
3. Recurse on each new node

# Regression Trees: Example



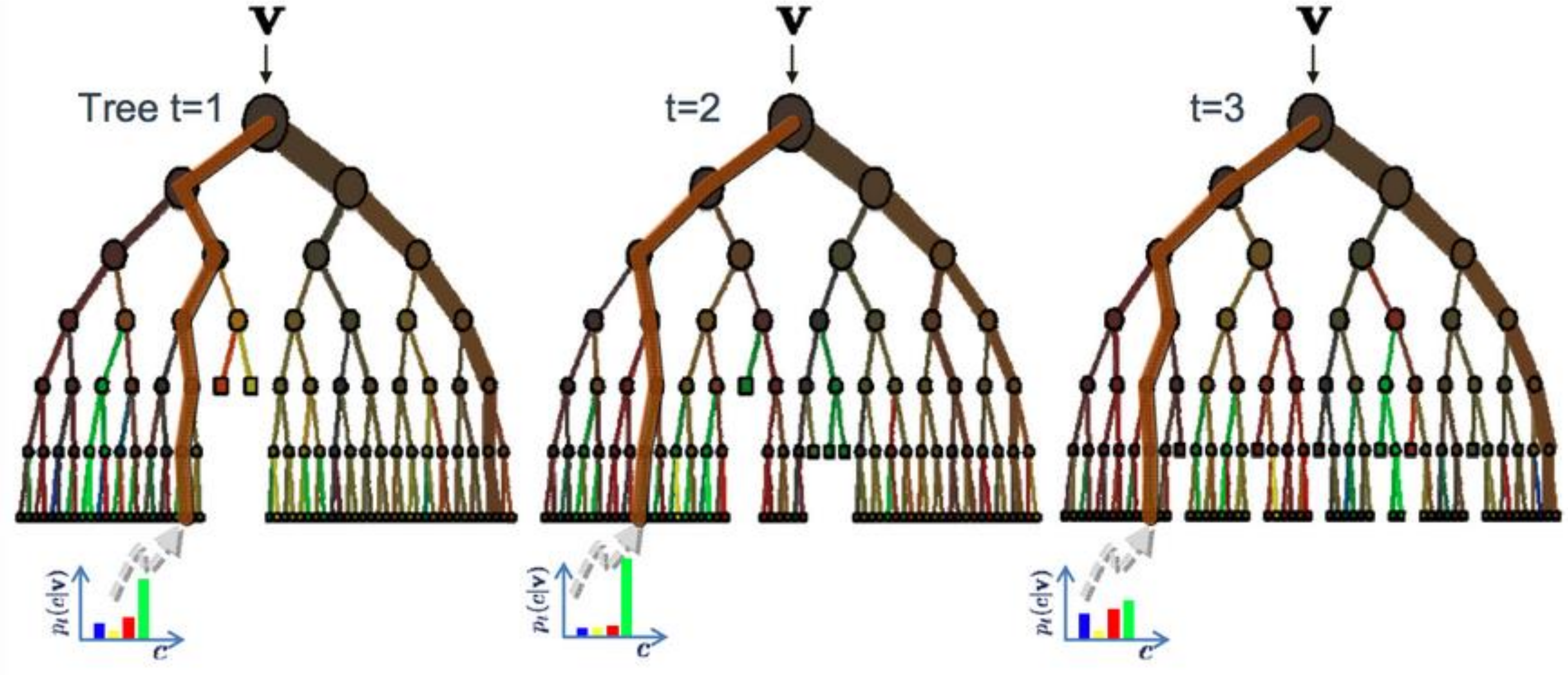



# Outline

- Decision Trees: Loose ends
  - Overfitting and Pruning
  - Extensions
- Regression Trees
- **Random Forests**
- Feature Importance

# Trees to forests

- Which classifier works best?
  - “Random forests” combine outputs of multiple classifiers



# Building a forest

- Bootstrap sample a new training set
  - with replacement
- Build a decision tree
  - Random subset of splits/features
  - Forces differentiated trees
  - No pruning! “Regularization” through forest
- Repeat until you have lots of trees
- Predict by taking a vote among the trees

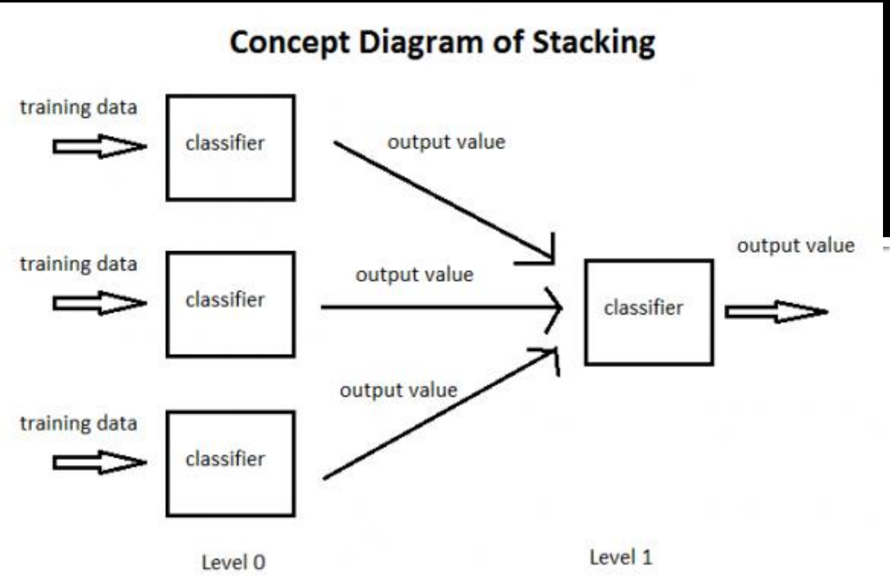
# Example: The "CART" forest

- Formally:

$$\hat{y}_i = \sum_{f_k \in \mathcal{F}} f_k(x_i)$$

- $\mathcal{F}$ is the space of regression trees
  - Each $f_k$ maps data examples $x_i$ to tree leaves
  - With CART (a common decision tree algorithm), each leaf isn't a decision value, it's a "score"
  - Scores are summed across trees

# Ensemble methods




- Bagging = **b**ootstrap **agg**regat**ing**
  - Create artificial versions of data via bootstrap
  - 1 sample = bootstrap
  - *M* samples = bagging
- Stacking: train model (e.g. another tree, a logistic regression) on output of other models
- Boosting: Can a set of weak learners create a single strong learner? (Kearns, 1988)
  - Train a sequence of models, each emphasizes the examples misclassified by the previous model

# Adaptive Boosting (AdaBoost)

- Adaboost:

1. Initially, set a weight for each training example = 1/n
2. Train a classifier where the objective respects the weights
3. Increase the weights for misclassified examples
4. Return to 2

Bauer, E. and Kohavi, R. An empirical comparison of voting classification algorithms: Bagging, boosting and variants. *Machine Learning 36* (1999), 105–142.

# Gradient Boosting


http://blog.kaggle.com/2017/01/23/a-kaggle-master-explains-gradient-boosting/

# (Extreme) Gradient Boosting

- Start with regression tree-based model:

$$\hat{y}_i = \sum_{f_k \in \mathcal{F}} f_k(x_i)$$

- Gradient boosting loss function "fits on residuals":

$$\mathcal{L}^{(t)} = \sum_{i=1}^{n} l(y_i, \underbrace{\hat{y}_i^{(t-1)} + f_t(x_i)}_{f_t \text{ fits on residual of t-1}}) + \underbrace{\gamma T + \frac{1}{2}\lambda \|w\|^2}_{\text{Regularization penalty}}$$

  - $T$ is the number of leaves
  - $t$ indexes training iterations
  - $w$ is vector of scores on each leaf (i.e., the leaf weights)

- Optimization is similar to gradient descent
  - Relies on being able to measure how good each tree is
  - Next tree solves for the loss of prior tree

Friedman, J.H., 2001. Greedy function approximation: a gradient boosting machine. Annals of statistics 1189–1232.

# Outline

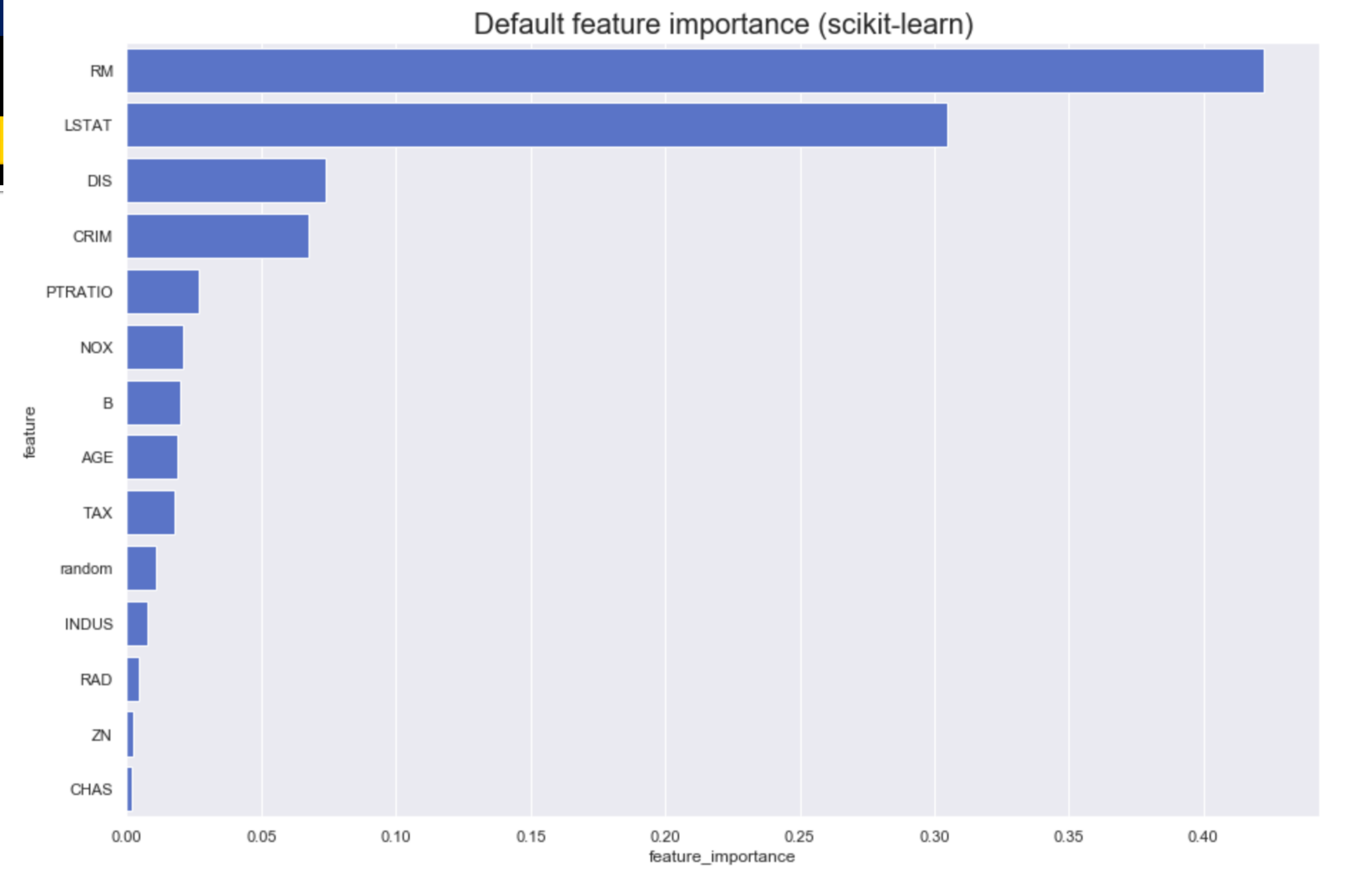
- Decision Trees: Loose ends
  - Overfitting and Pruning
  - Extensions
- Regression Trees
- Random Forests
- **Feature Importance**

# Feature Importance

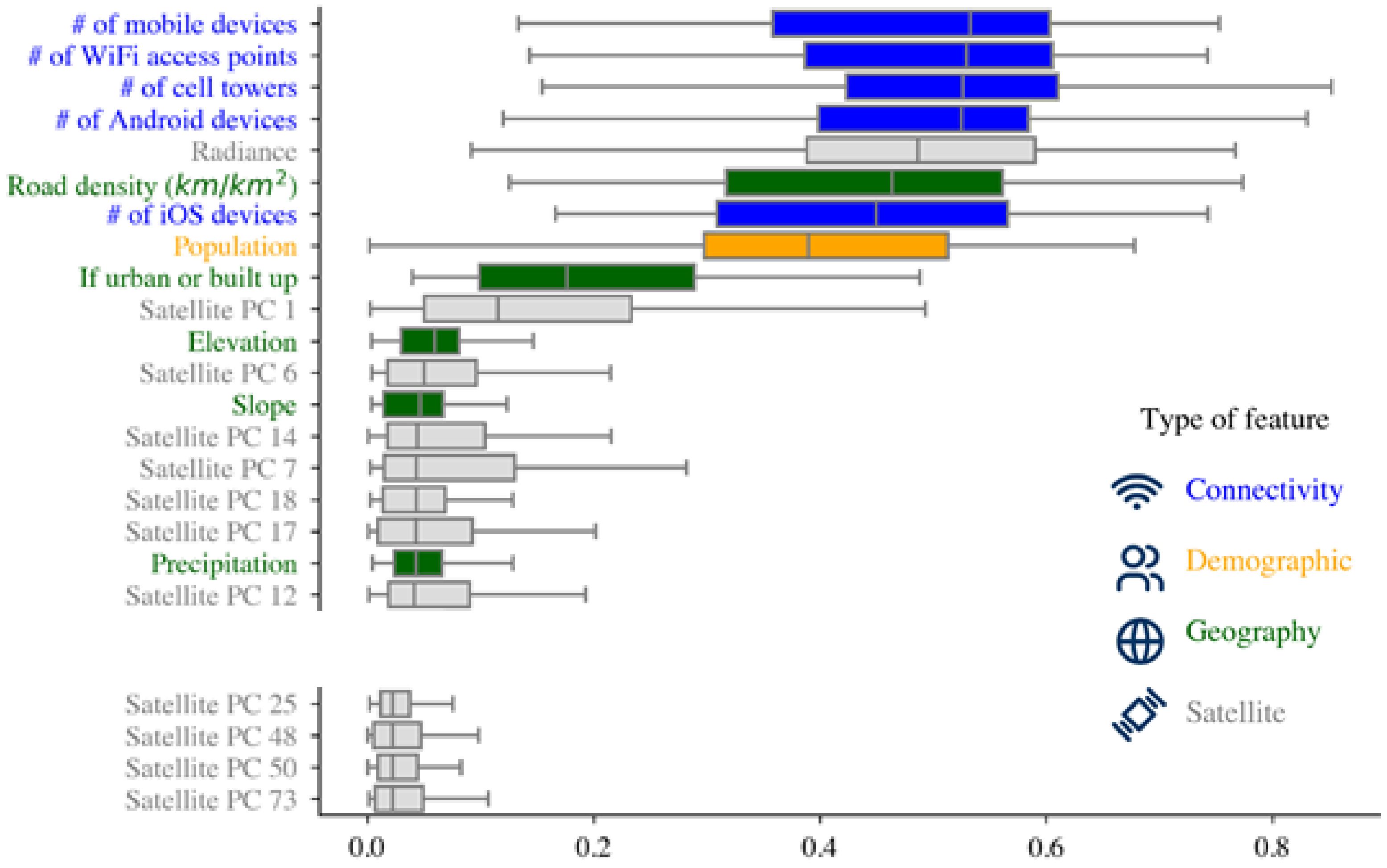
- The primary focus of random forests is prediction, rather than inference
  - A single tree is fairly interpretable, but it's hard to interpret a forest
  - This is generally true for complex, non-parametric, and non-linear models
- Nonetheless, people frequently still want to do some ex-post interpretation
  - "What features are important to the classifier?"

# Feature Importance

- Intuitively, the features that "matter":
  - Occur high in the tree (high information gain for that tree)
  - Occur frequently in the tree (if feature is non-binary)
  - Occur in many trees (if it's a forest)


Default feature importance (scikit-learn)
RM
LSTAT
DIS
CRIM
PTRATIO
NOX
B
AGE
TAX
random
INDUS
RAD
ZN
CHAS
feature
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.35
0.40
feature_importance

# Feature Importance: Example



# Feature Importance

- Formally, two common approaches:
  1. **Mean decrease impurity (aka Gini importance)**: average (across trees) decrease in weighted impurity caused by that feature

```
from sklearn.datasets import load_boston
from sklearn.ensemble import RandomForestRegressor
import numpy as np
#Load boston housing dataset as an example
boston = load_boston()
X = boston["data"]
Y = boston["target"]
names = boston["feature_names"]
rf = RandomForestRegressor()
rf.fit(X, Y)
print "Features sorted by their score:"
print sorted(zip(map(lambda x: round(x, 4), rf.feature_importances_), names),
             reverse=True)
```

```
Features sorted by their score:
[(0.5298, 'LSTAT'), (0.4116, 'RM'), (0.0252, 'DIS'), (0.0172, 'CRIM'), (0.0065, 'NOX'),
(0.0035, 'PTRATIO'), (0.0021, 'TAX'), (0.0017, 'AGE'), (0.0012, 'B'), (0.0008, 'INDUS'),
(0.0004, 'RAD'), (0.0001, 'CHAS'), (0.0, 'ZN')]
```

# Feature Importance

- Issues with impurity:
  - Biased towards features with multiple values
  - What happens when two features are closely correlated?

```
size = 10000
np.random.seed(seed=10)
X_seed = np.random.normal(0, 1, size)
X0 = X_seed + np.random.normal(0, .1, size)
X1 = X_seed + np.random.normal(0, .1, size)
X2 = X_seed + np.random.normal(0, .1, size)
X = np.array([X0, X1, X2]).T
Y = X0 + X1 + X2

rf = RandomForestRegressor(n_estimators=20, max_features=2)
rf.fit(X, Y);
print "Scores for X0, X1, X2:", map(lambda x:round (x,3),
                                    rf.feature_importances_)
```

```
Scores for X0, X1, X2: [0.278, 0.66, 0.062]
```

# Feature Importance

- Formally, two common approaches:
  1. **Mean decrease impurity**: average (across trees) decrease in weighted impurity caused by that feature
  2. **Mean decrease accuracy ("Permutation Importance")**: average (across trees) decrease in performance when a given feature is randomized
     - Not implemented in sklearn (but very easy to do so by hand - see ESLII reading)

# Feature Importance

- Issues
  - Interpret feature importances at your own risk!
  - They are informative, but rather atheoretical

# Key Concepts (this lecture)

- Overfitting and pruning
- Regression trees
- Random forests
- AdaBoost
- Gradient boosting
- Feature Importance

# Recap

- Regression
  - Parametric, fast training, linear
- Nearest Neighbors
  - Non-parametric, no training, complex decisions
- Naïve Bayes
  - Parametric, very fast training
- Decision Trees
  - Non-linear decisions, intuitive model

# For Next Class:

- Read:
  - Daume, chapters 4 and 10
- Keep working on Problem Set 4!