UNIVERSITY OF SAN FRANCISCO

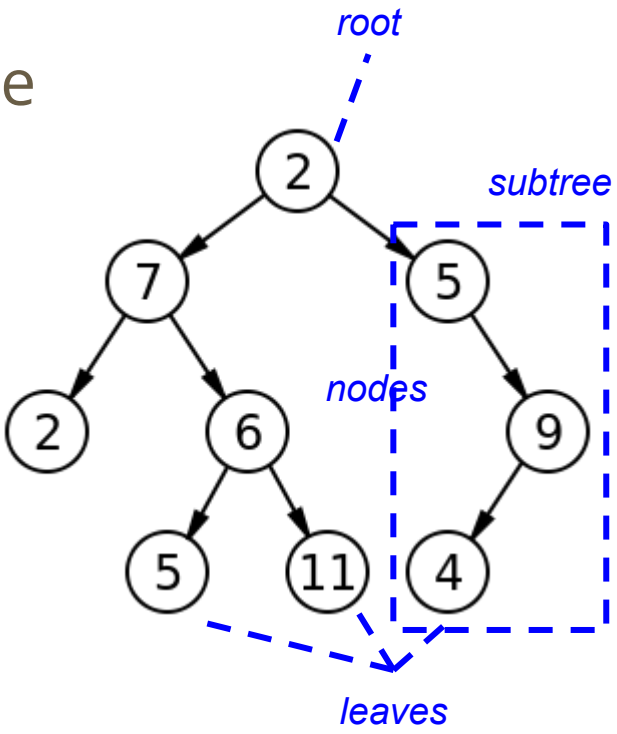
Master of Science in Analytics

Tree-Based Methods

Machine Learning 1

A tree in computer science

- An abstract data type / data structure
  - May be implemented as arrays (lists), etc.
  - Nodes have at most 1 parent
  - No “cycles”
- Uses
  - Data storage
  - Indexing (eg. in databases)
- Trees are recursive nature of trees
  - Nodes links from parent, to children
  - Algorithms on trees are generally recursive

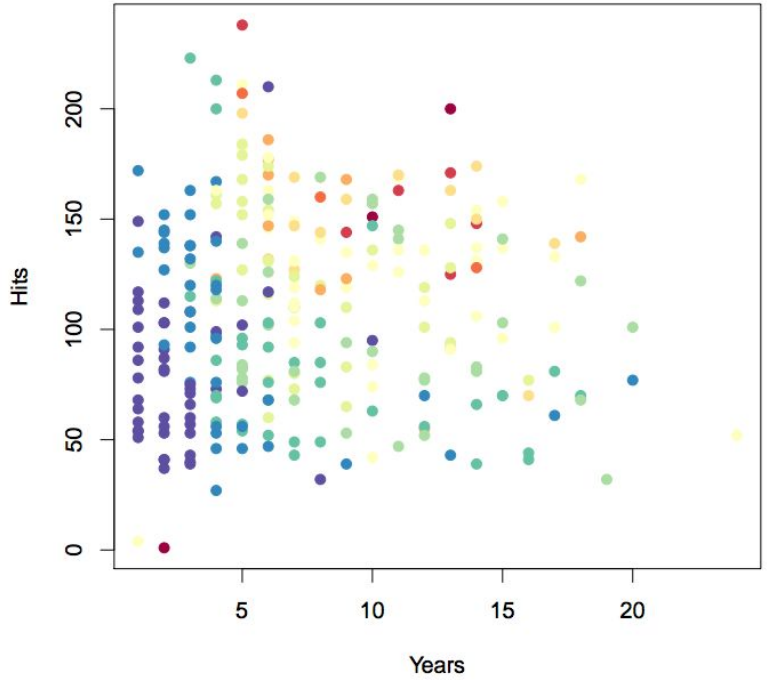


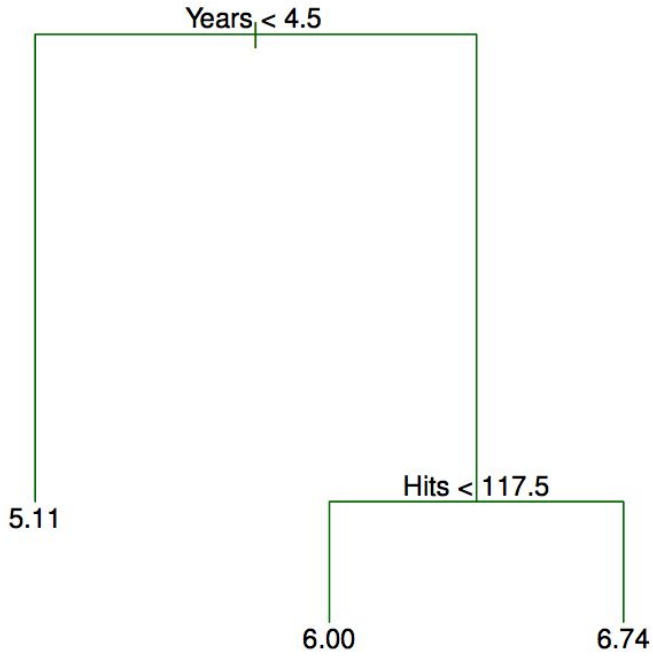
A tree in data science

- Segments the predictor space into smaller regions
- Collectively known as decision trees
  - Regression trees
  - Classification trees
- Arguments in favour & against (pros & cons)
  - Great for interpretation
  - Mediocre individual performance, but used in confederation for better performance
  - Potential performance improvements with: *bagging*, *random forests* or *boosting*

Regression trees

- Example: salary for baseball players
  - Task: predict a salary based on: hits & years of experience
  - Salary coded for low (blue / green) to high (yellow / red)



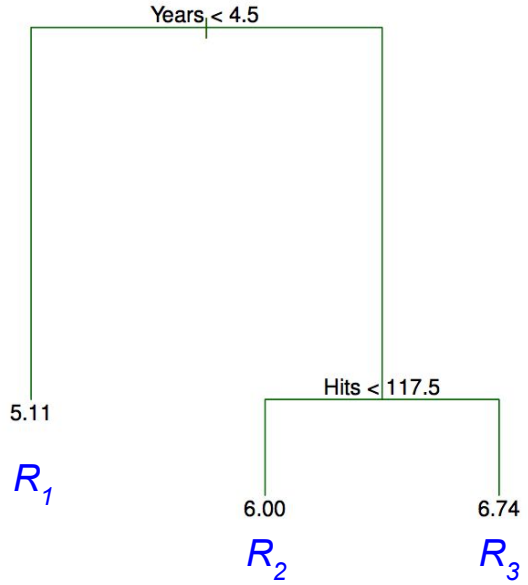


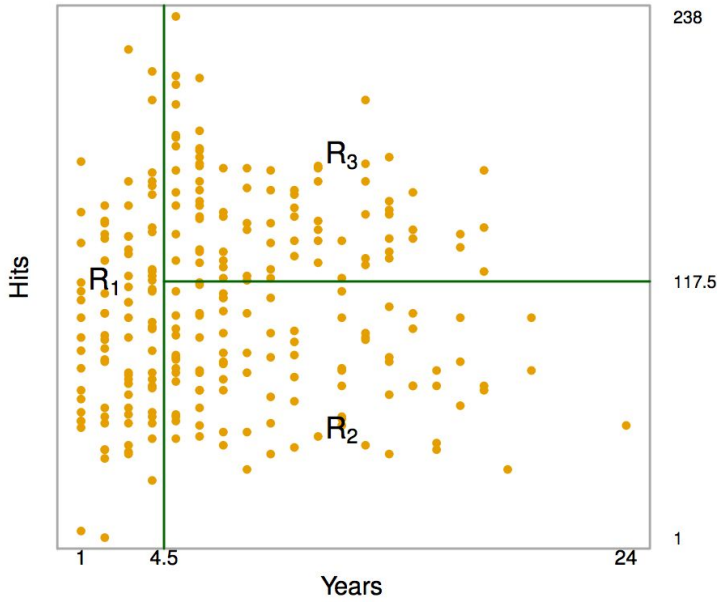
- Data (left) results in tree (right)

Log of salary

Alternative tree representations

- Tree on left is classic CS tree; includes outcomes
- Tree on right is common in ML/ data science
- Interpretation:
  - Years is most important feature
  - Among experienced players, hits is most important feature





Goal

- Notice division of feature space
  - In our 2D example: recursive division of features into rectangles
  - In multiple dimensions: high-dimensional rectangles, "*boxes*"
  - In theory, could use any theoretical shape, but not practice
- Overall goal
  - Find boxes R_1, ..., R_J that minimize RSS
  - (Our RSS formula remains unchanged except as it applies to boxes)

$$\sum_{j=1}^{J} \sum_{i \in R_j} (y_i - \hat{y}_{R_j})^2$$

  - ... but in practice we cannot consider every possible partition

Recursive binary splitting

- An algorithm for building decision trees
  - Top-down: starts at root of tree (all data) and splits
  - Greedy: at each step, it only makes the best split at that time
  - Caution: greedy algorithms sometimes produce results which are not globally optimal (eg. 0|1 knapsack problem)
- Description
  - Select predictor X_j and cutpoint s so that $\{X|X_j < s\}$ and $\{X|X_j \geq s\}$ leads to greatest reduction in RSS
  - Repeat above on remaining boxes
  - Continue the above until some stopping criterion is satisfied (eg. no region has more than 5 observations)
- Making predictions after the tree is built
  - Determine region based on features
  - Use mean of training observations in the region

Problems with performance

- Performance vs. tree size
  - Recursive binary splitting may produce good predictions on training & poor performance on testing
  - Sometimes smaller trees (fewer splits) have lower variance
- Solution # 1
  - Grow the tree while RSS decrease exceeds some high threshold
  - Bad idea for a greedy algorithm
  - But worthless splits early on may give excellent splits later
- Solution # 2
  - Grow a (very) large tree
  - Prune the tree to get a subtree

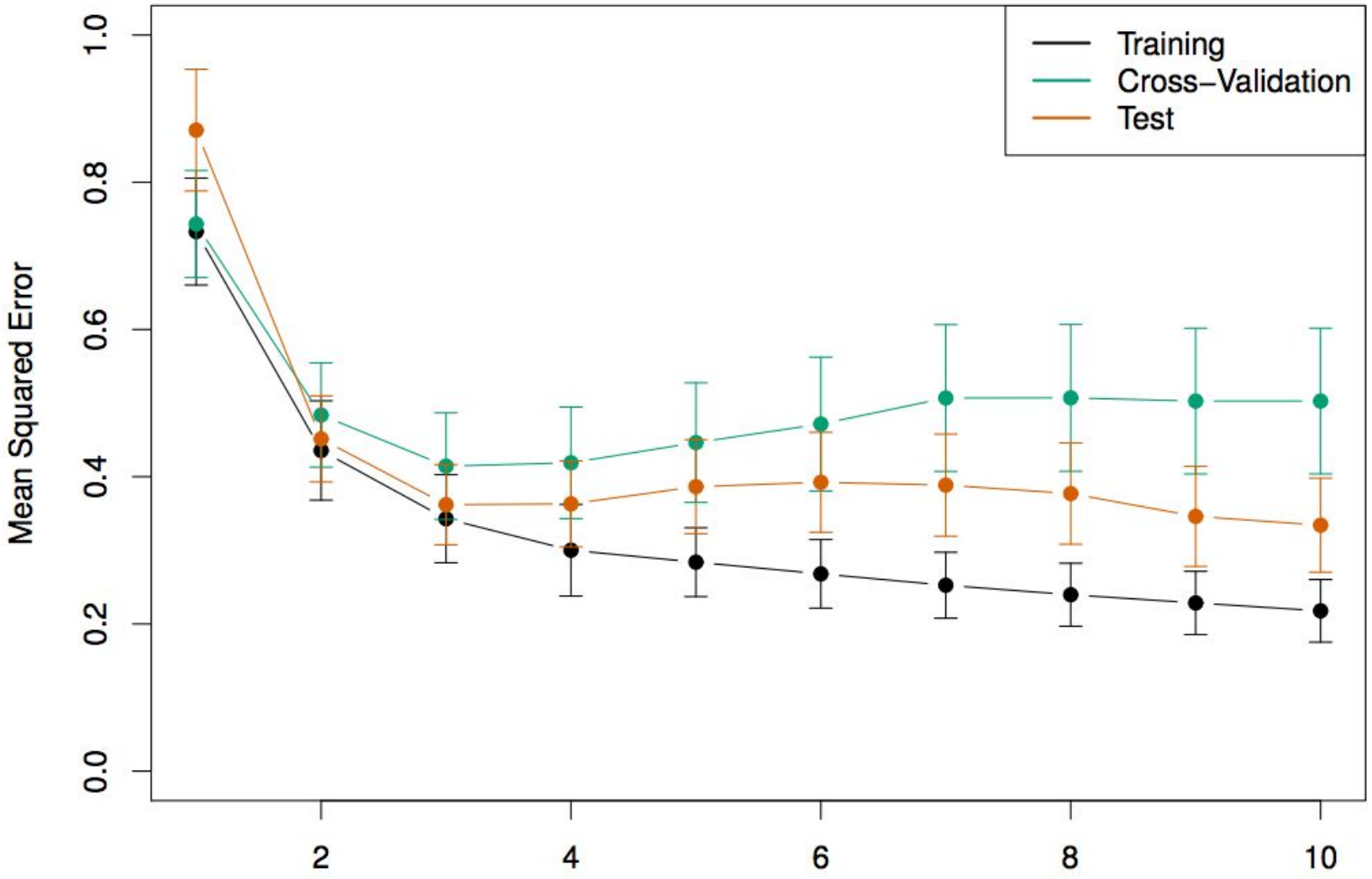
Cost complexity pruning

- An algorithm for tree pruning
  - Also known as *weakest link pruning*
  - Depends on non-negative tuning parameter, α — trade-off between complexity of subtree and fit to training observations
- Driven by equation

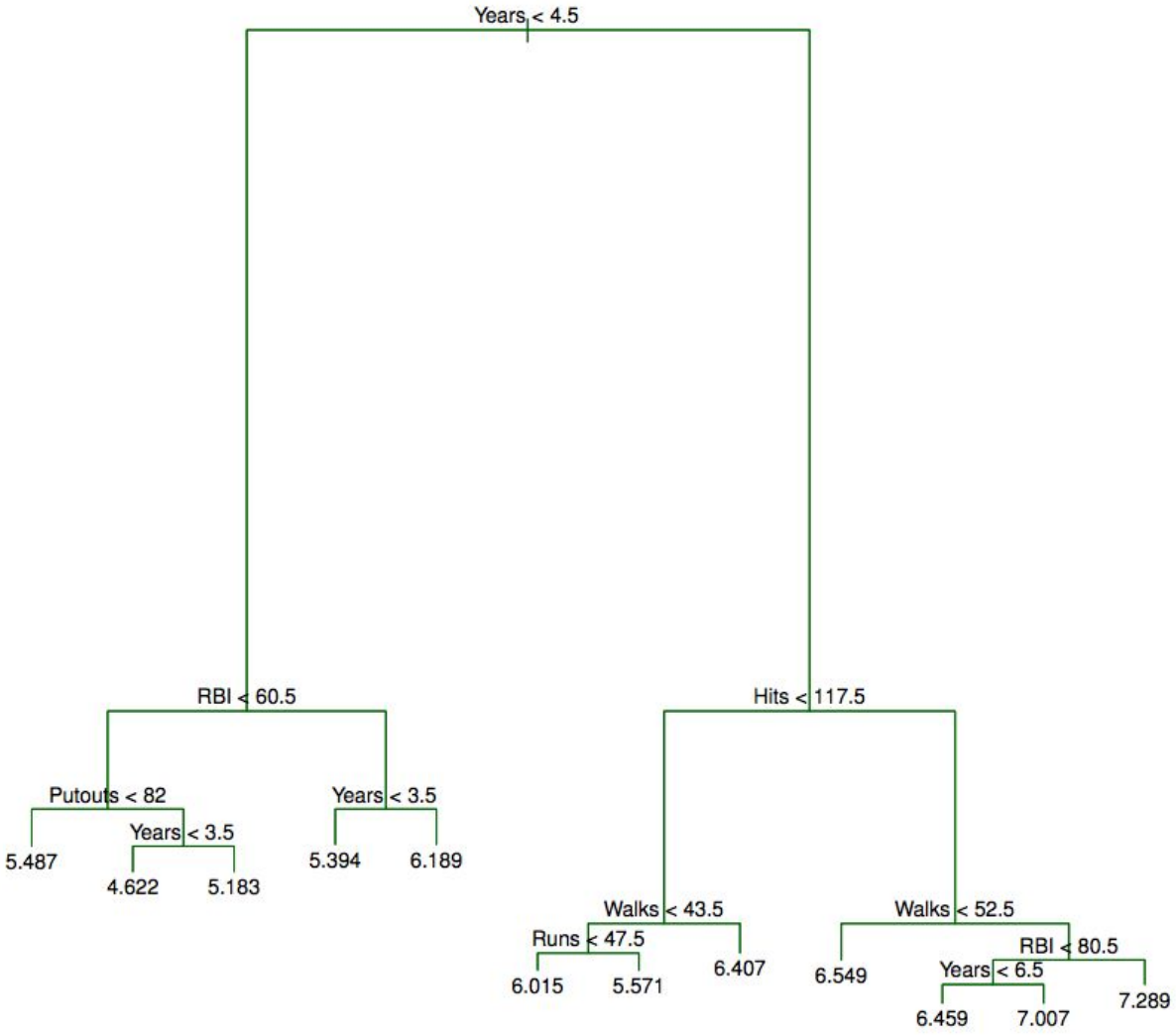
$$\sum_{m=1}^{|T|} \sum_{i:\ x_i \in R_m} (y_i - \hat{y}_{R_m})^2 + \alpha |T|$$

- Considerations:
  - $|T|$ is number of leaves in the (sub-)tree T
  - R_m is the rectangle of m^{th} leaf node
  - Select optimal α through cross-validation

Back to baseball - pre-pruned tree



Back to baseball - pre-pruned tree



Putting it all together

- Grow the tree
  - Use recursive binary splitting to grow a large tree
  - Constructed from training data
  - Stopping when leaves have no more than X observations
- Prune the tree
  - Use cost complexity pruning to get the best set of subtrees
  - Based on α
- Choose the best α
  - Use k-fold cross-validation
  - Average the results and pick α to minimize average (training) error
- Best subtree corresponds to pruned subtree for α

Implementation in sklearn (1)

- Choose the best value for max_depth
  - Depth ~= height: max(nodes in path from root to leaves)
  - Use k-fold cross validation
- Depend on sklearn for implementation details
  - Grow the tree
  - Prune the tree
- Test the tree using “predict”

Implementation in sklearn (2)

1) Import data
    a) If necessary, split data into train, test sets
2) Coerce training data into: train_X and train_Y
3) Coerce test data into: test_X and test_Y

```
# 3) See the documentation for full set of options

from sklearn.tree import DecisionTreeRegressor

depth = 5    # Note that depth, not alpha, is the hyperparameter
regressor = DecisionTreeRegressor(max_depth=depth)
regressor.fit(X, Y)
predicted_y = regressor.predict(test_x)
```

Lab: regression trees

- Data
  - Get BlogFeedback data via UCI: https://archive.ics.uci.edu/ml/datasets/BlogFeedback
  - Test: start with blogData_test-2012.03.31.01_00.csv
  - Final column is response (blog feedback)
- Task
  - Construct one or more models to predict blog feedback
  - Find ideal tree depth via k-fold cross validation (eg. with depth = 5, MSE = 669.67; this is not ideal)
  - Compare results to linear regression
- Advanced Tasks
  - Show your resulting tree — eg. using GraphViz
  - Compare these results to ridge regression, the lasso
  - Add all of the test data (not just 2012-03-31)

Classification trees

- Similar to regression trees; differences
  - Response is qualitative
  - Leaves (regions) predict most commonly occurring class of training observations
- Grow the tree using recursive binary splitting
  - Cannot use RSS for splitting criterion. Why?
  - Criteria for classification trees: classification error rate, gini index, cross-entropy

Classification error rate

$$E = 1 - \max_k(\hat{p}_{mk}).$$

- p_{mk} = portion of training observations in m^{th} region from k^{th} class
- In practice, classification error rate is not sufficiently sensitive

Gini index

$$G = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk})$$

- Measures total variance among *K* classes
  - Observations from a single class —> Gini index will be 0
  - Takes a small value if p_{mk}s are close to 0 or close to 1
- Also called "*node purity*" or "*Gini impurity*"

Cross entropy

$$D = -\sum_{k=1}^{K} \hat{p}_{mk} \log \hat{p}_{mk}$$

- Also measures total variance among *K* classes
  - Observations from a single class —> cross entropy will be 0
  - Also takes a small (positive) value if p_{mk}s are close to 0 or close to 1
- Very similar results to Gini index

Implementation in sklearn

1) Import data
    a) If necessary, split data into train, test sets
2) Coerce training data into: train_X and train_Y
3) Coerce test data into: test_X and test_Y

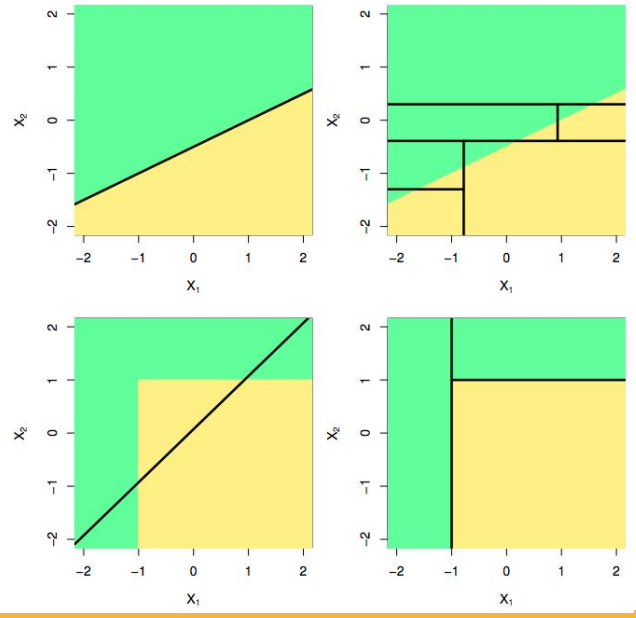
```
# 3) See the documentation for full set of options

from sklearn import tree

crit = "gini"    # Only supports "gini" or "entropy"
classifier = tree.DecisionTreeClassifier(criterion=crit)
classifier.fit(X, Y)
predicted_y = classifier.predict(test_x)
```

Lab: classification trees

- Data
  - Get BlogFeedback data via UCI: https://archive.ics.uci.edu/ml/datasets/Occupancy+Detection+
  - Test: start with datatest.txt
  - Final column is response (blog feedback)
- Task
  - Construct one or more models to predict occupancy of room {0, 1}
  - Determine ideal tree depth via cross validation
  - Determine performance vs. different criterion, splitter, etc. values
  - Compare results to logistic regression, LDA
- Advanced Tasks
  - Show your resulting tree — eg. using GraphViz
  - Compare these results to SVM (SVC)
  - Add remaining test data (datatest2.txt)

Trees vs. linear models

- Advantages of trees
  - Trees are easy to explain and interpret
  - Trees can handle qualitative predictors gracefully
- Disadvantage of trees: weak performance depends on
  - Good performance requires “boxy” relationships
  - Improve performance via:
    - Bagging
    - Random Forests
    - Boosting



Bagging (for regression trees)

- Portmanteau for "*bootstrap aggregation*"
- Recall the bootstrap
  - Theory narrative: averaging a set of observations reduces variance
  - More technically: a set of independent observations Z_1, ..., Z_n, each with variance σ^2, the variance of the mean Z of observations is σ^2/n
  - Take repeated samples from the same data set
- Generation
  - Create B different bootstrapped datasets

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

  - ... where $f^{*b}(x)$ = the prediction point at x for the bth training set

Bagging (for classification trees)

- Change the data used to build the tree
- Construct a tree for each bootstrapped dataset
  - When testing, record the class predicted by each of the *B* trees
  - Generate the majority (most commonly occurring class among B predictions) as hypothesis for test observation
- Bagging also helps estimate test error
  - Recall: each bootstrapped dataset uses ~ ⅔rds of observations
  - Remaining observations are called OOB (*out-of-bag*) observations
  - Can predict the response for the i^{th} observation using each of the trees in which that observation was OOB
  - ~ B/3 predictions will be OOB for i^{th} observation
  - If B is large, this is roughly equivalent to leave-one-out cross-validation

Bagging implementation

1) Import data
    a) If necessary, split data into train, test sets
2) Coerce training data into: train_X and train_Y
3) Coerce test data into: test_X and test_Y

```
# 3) See the documentation for full set of options

from sklearn.ensemble import BaggingClassifier

estimators = 10    # Number of estimators, defaults to 10
classifier = BaggingClassifier(n_estimators=estimators)
classifier.fit(X, Y)
predicted_y = classifier.predict(test_x)
```

Random forests

- Change how the tree is built
  - As with bagging, we build multiple trees on B bootstrapped datasets
  - When splitting, rather than considering the full set of *p* features, pick the best from a randomly-generated *m* of the features ($m < p$)
  - Consider a new set of *m* features at each split
- Popular for classification but also works for regression
- Why does it work?
  - Combination of learning models increases classification accuracy (and randomness gives diversity to models, called *decorrelation*)
  - Averages noisy & unbiased models to create low variance
  - Some make the strong claim that random forests do not overfit or that random forests can solve any ML problem

Random Forest implementation

1) Import data
    a) If necessary, split data into train, test sets
2) Coerce training data into: train_X and train_Y
3) Coerce test data into: test_X and test_Y

```
# 3) See the documentation for full set of options

from sklearn.ensemble import RandomForestClassifier

estimators = 10    # Number of estimators, defaults to 10
classifier = RandomForestClassifier(n_estimators=estimators)
classifier.fit(X, Y)
predicted_y = classifier.predict(test_x)
```

Boosting

- Grow weak trees and depend on confederation
  - As with bagging, we build multiple trees on B bootstrapped datasets
  - Order trees sequentially (t_1, …, t_k)
  - Where a tree t_i commits an error in training, add weight to error portion of data in subsequent tree t_{i+1}
  - Final output is a linear combination of each tree's hypothesis, weighted by performance
- Example: cancer
  - Consult multiple doctors for a diagnosis
  - Combine diagnoses with weights according to previous diagnosis performance
- Boosting can be applied to classification and regression

AdaBoost

- Focus on what went wrong
  - As with bagging, we build multiple trees on B bootstrapped datasets
  - Create k trees (t_1, ..., t_k) to learn bootstrapped data
  - AdaBoost: apply weights (w_1, ..., w_N) to training data, initially 1/N
  - Iterate to convergence:
    - Add weight to misclassified observations
    - Decrease weight to correctly classified observations
- Why does it work?
  - Slow learning leads to robust learning
  - Slowly hyperfocused on misclassifications... and changes necessary to fix misclassifications
  - Boosting tends to have greater accuracy than other tree methods but may overfit

Boosting implementation

1) Import data
    a) If necessary, split data into train, test sets
2) Coerce training data into: train_X and train_Y
3) Coerce test data into: test_X and test_Y

```
# 3) See the documentation for full set of options

from sklearn.ensemble import AdaBoostClassifier

rate = 0.1    # Learning rate; default is 1.0
classifier = AdaBoostClassifier(learning_rate=rate)
classifier.fit(X, Y)
predicted_y = classifier.predict(test_x)
```

Lab: classification trees

- Data
  - Get BlogFeedback data via UCI: https://archive.ics.uci.edu/ml/datasets/Occupancy+Detection+
  - Test: start with datatest.txt
  - Final column is response (blog feedback)
- Task
  - Construct one each model each corresponding to {bagging, adaboost, random forest} to predict occupancy of room {0, 1}
  - Determine ideal learning_rate via cross validation
  - Determine performance vs. different algorithm, etc. values
  - Compare results to your previous model(s) on this dataset