Decision Trees & k-NN
CS 446/546

Outline

- Overview of Decision Trees
- Regression Trees
- Pruning
- Classification Trees
- Bagging, Random Forests & Boosting
- k-NN (*k-nearest neighbors*)

Overview

- **Tree-based methods** are frequently used in ML for both regression and classification tasks.

- Recall that for a *parametric model*, we define the model over the entire input space and learn a finite, fixed number of parameters using the training data. Then we use the same model and parameter set for any test input.

- Conversely, for *non-parametric models*, we divide the input space into local regions, defined by a distance measure (e.g. Euclidean distance), and for each input, the corresponding local model computed from the training data in that region is used. In this way, we say the model complexity of a non-parametric model scales with the size of the data.

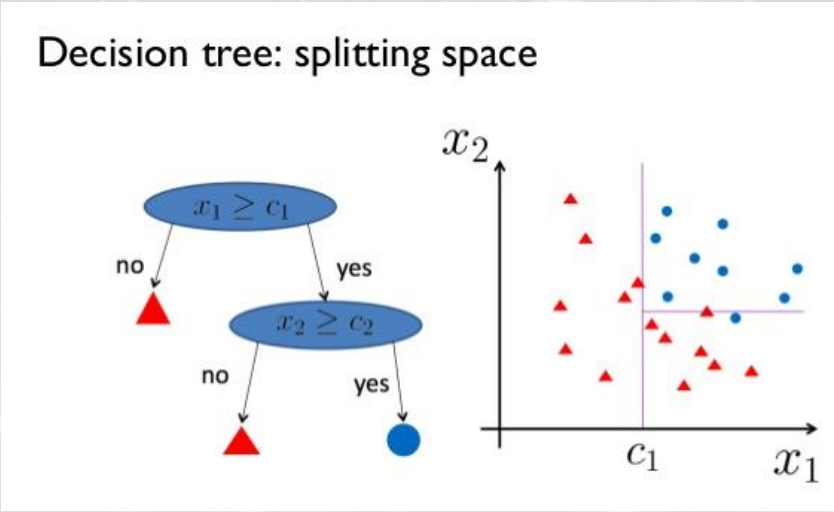
- A decision tree is a hierarchical, non-parametric model for supervised learning.

Overview

- "Learning a tree" entails segmenting/stratifying the predictor space into a number of simple regions.

- In order to make a prediction for a given observation, we typically use the mean or mode of the training observations in the region of the predictor space to which it belongs.

- A learning algorithm for a tree equates to defining the **splitting rules** used to segment the predictor space – these models are accordingly referred to as *decision trees*.



Overview

- Tree-based models have the advantage that they are simple to implement and interpretable; however, they are typically not competitive with the best supervised learning approaches.

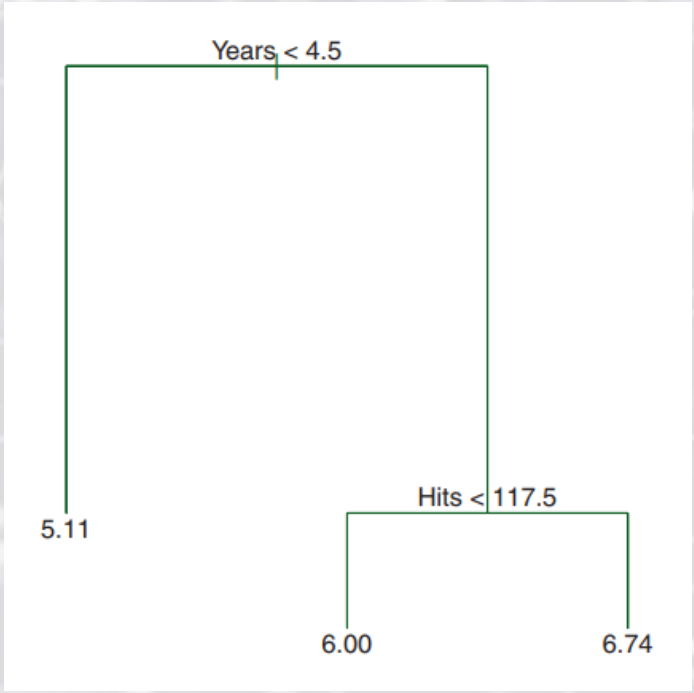
- In light of this, trees are often used in conjunction with **bagging** and **boosting**. In each case, multiple trees (i.e. *weak learners*) are combined as an *ensemble* to generate a single prediction.

- Combining a large number of trees can often result in dramatic improvement in prediction accuracy – although such improvements come at the cost of interpretability.

Regression Trees

- Let's motivate regression trees with a simple example.* Suppose that we want to predict a baseball player's (log-scale) **salary** based on the following predictor variables: *years* (number of years in MLB) and *hits* (number of hits in previous year).

Here's a how a regression tree might fit the data:



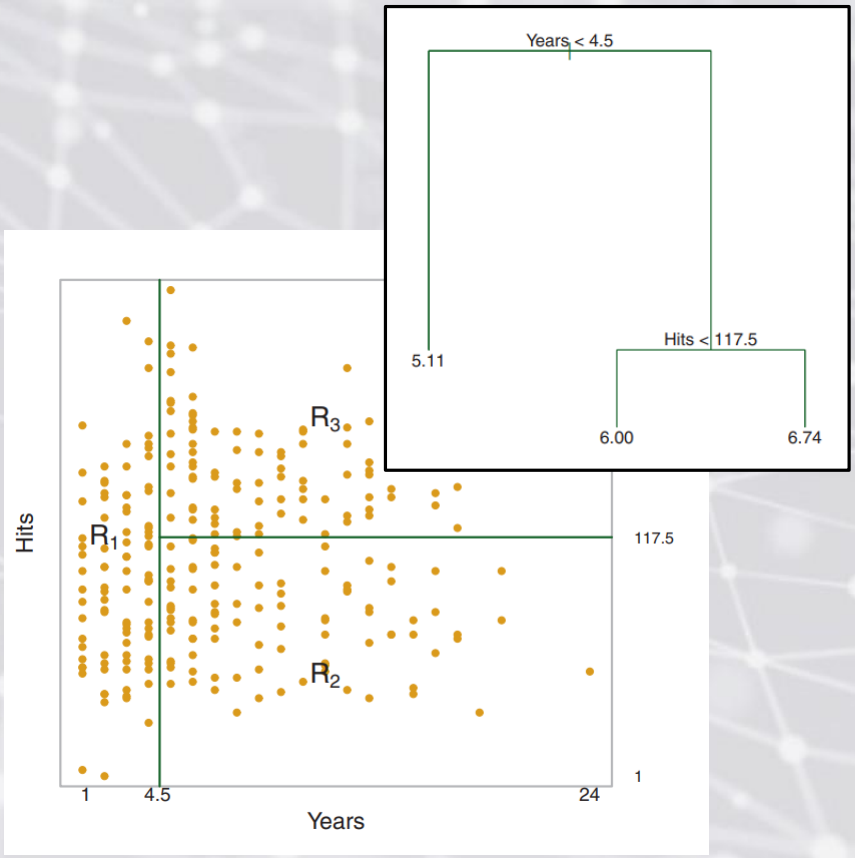
Regression tree for *fitting log salary* (as a function of Years and Hits). At a given interval node, the label (of the form $X_j < t_k$) indicates the left-hand branch emanating from that slight, and the right-hand branch corresponds to $X_j > t_k$. For instance, the split at the top of the tree results in two large branches. The lefft-hand branch corresponds to *Years* < 4.5 and right-hand branch corresponds to *Years* ≥4.5. The tree has two internal nodes and three terminal nodes (*leaves*). The number in each leaf is the mean of the response for the observations that fall there.

*Example from James, *et al.*, *Intro to Statistical Learning*, Springer (2014).

Regression Trees

- Overall, the tree stratifies or segments players into three discrete regions of the predictor space: (1) players how have played less than 4.5 years, (2) players who have played at least 4.5 years & had less than 117 hits, and (3) players who have played at least 4.5 years & had more than 117 hits.

- The leaves of the tree correspond with the regions R1, R2, and R3 in the predictor space.

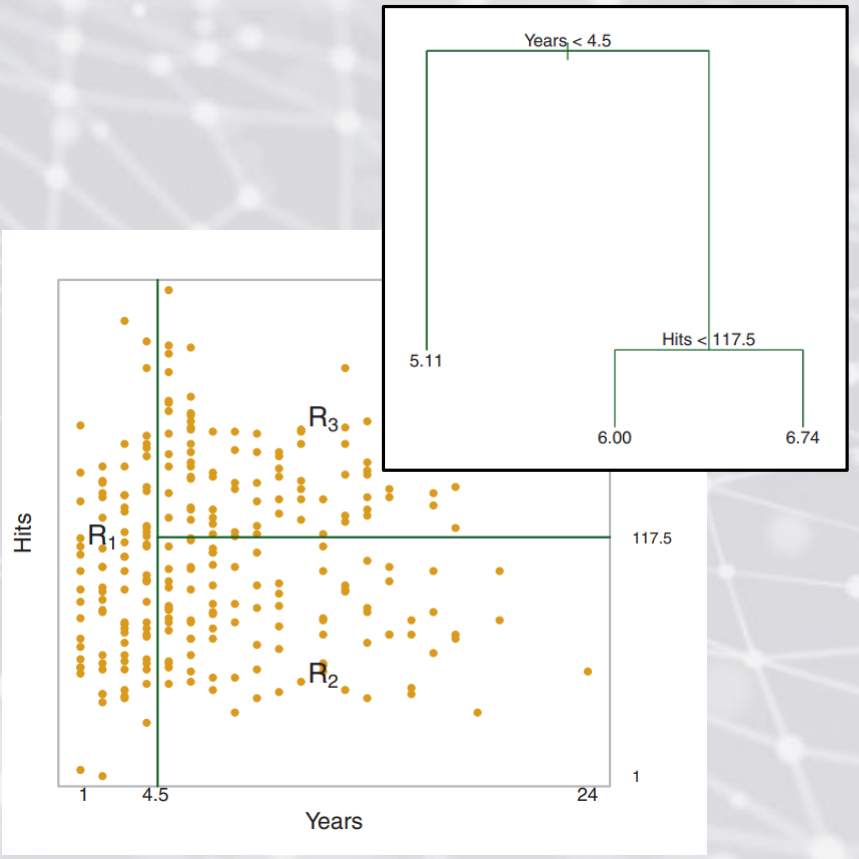


Regression Trees

- Overall, the tree stratifies or segments players into three discrete regions of the predictor space: (1) players how have played less than 4.5 years, (2) players who have played at least 4.5 years & had less than 117 hits, and (3) players who have played at least 4.5 years & had more than 117 hits.

- The leaves of the tree correspond with the regions R1, R2, and R3 in the predictor space.

It is frequently possible to interpret a tree structure hierarchically, in which case one might consider *Years* as the most important factor in determining salary (as this is feature is the basis for the first "decision" in the tree). Furthermore, based on the tree, we can infer that players with less experience earn a lower salary than more experienced players (this is intuitively clear). Given that a player is less experienced, the number of hits that he made in the previous year seems to play little role in his salary; however, hits are positively associated with salary (as should also be intuitively clear). The graphical representation of a decision tree lends this type of a model a natural interpretative framework that is often absent in other models (e.g. regression, NNs).



Regression Trees

- There are (2) basic steps required for building a regression tree:

(1) We divide the **predictor space** (i.e. the set of all possible values for the regression variables $X_1, X_2, \ldots, X_p$) into *J* distinct and non-overlapping regions, $R_1, R_2, \ldots, R_J$.

(2) For every observation that falls into region R_j, we make the same prediction, which is simply the mean of the response values (y) for the training observations in R_j.

Regression Trees

(1) We divide the **predictor space** (i.e. the set of all possible values for the regression variables $X_1, X_2, \ldots, X_p$) into *J* distinct and non-overlapping regions, $R_1, R_2, \ldots, R_J$.

- As usual, we frame our task as an optimization problem. For simplicity and ease of interpretation, we'll use high-dimensional *rectangles* to describe our regions (certainly other shapes can be used, e.g. spheres, polytopes).

- The goal in step (1) is to find boxes $R_1, \ldots, R_J$ that minimize the RSS (*residual sum square error*):

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

where $\hat{y}_{Rj}$ denotes the mean response for the training observations within the jth box.

Regression Trees

- The goal in step (1) is to find boxes $R_1,\ldots,R_J$ that minimize the RSS (*residual sum square error*):

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

(*) Unfortunately, it is computationally infeasible (NP-hard)to consider every possible partition of the feature space. For this reason, one can adopt a *top-down*, *greedy* approach known as **recursive binary splitting**.

- Top-down here refers to the fact that we begin at the top of the decision tree and perform successive splits; greedy connotes the fact that we make the (myopically) best split at that particular step (instead of looking ahead and picking a split based on future steps).

Regression Trees

- The goal in step (1) is to find boxes $R_1,\ldots,R_J$ that minimize the RSS (*residual sum square error*):

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

- In order to perform recursive binary splitting, we first select the predictor X_j and the cut-point s such that splitting the predictor space into the regions $\{X|X_j<s\}$ and $\{X|X_j\geq s\}$ leads to the greatest possible reduction in RSS.

Thus we consider all predictors $X_1,\ldots,X_p$, and all possible values of the cut-point s for each of the predictors, and then choose the predictor and cut-point such that the resulting tree has the smallest RSS.

Regression Trees

- The goal in step (1) is to find boxes $R_1,\ldots,R_J$ that minimize the RSS (*residual sum square error*):

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

- In order to perform recursive binary splitting, we first select the predictor X_j and the cut-point s such that splitting the predictor space into the regions $\{X\,|\,X_j < s\}$ and $\{X\,|\,X_j \geq s\}$ leads to the greatest possible reduction in RSS.

Thus we consider all predictors $X_1,\ldots, X_p$, and all possible values of the cut-point s for each of the predictors, and then choose the predictor and cut-point such that the resulting tree has the smallest RSS.

In summary: for any *j* (predictor index) and *s* (cut-point), we define the pair of half-planes:

$$R_1(j,s)=\{X\,|\,X_j<s\}\ \ \&\ \ R_2(j,s)=\{X\,|\,X_j\geq s\}$$

And we seek the value of *j* and *s* that minimize the equation: $\sum_{i:x_i\in R_1(j,s)}\left(y_i-\hat{y}_{R_1}\right)^2+\sum_{i:x_i\in R_2(j,s)}\left(y_i-\hat{y}_{R_2}\right)^2$

where $\hat{y}_{R1}$ is the mean response for the training observations in R_1(j,s), and $\hat{y}_{R2}$ is the mean response for the training observations in R_2(j,s); finding these values can be done relatively quickly (unless then number of predictors is very large).

(*) Note: the notation $\{X\,|\,X_j < s\}$ indicates the region of the predictor space for which X_j takes on values less than s.

Regression Trees

In summary: for any *j* (predictor index) and *s* (cut-point), we define the pair of half-planes:

$$R_1(j,s) = \{X \mid X_j < s\} \quad \& \quad R_2(j,s) = \{X \mid X_j \geq s\}$$

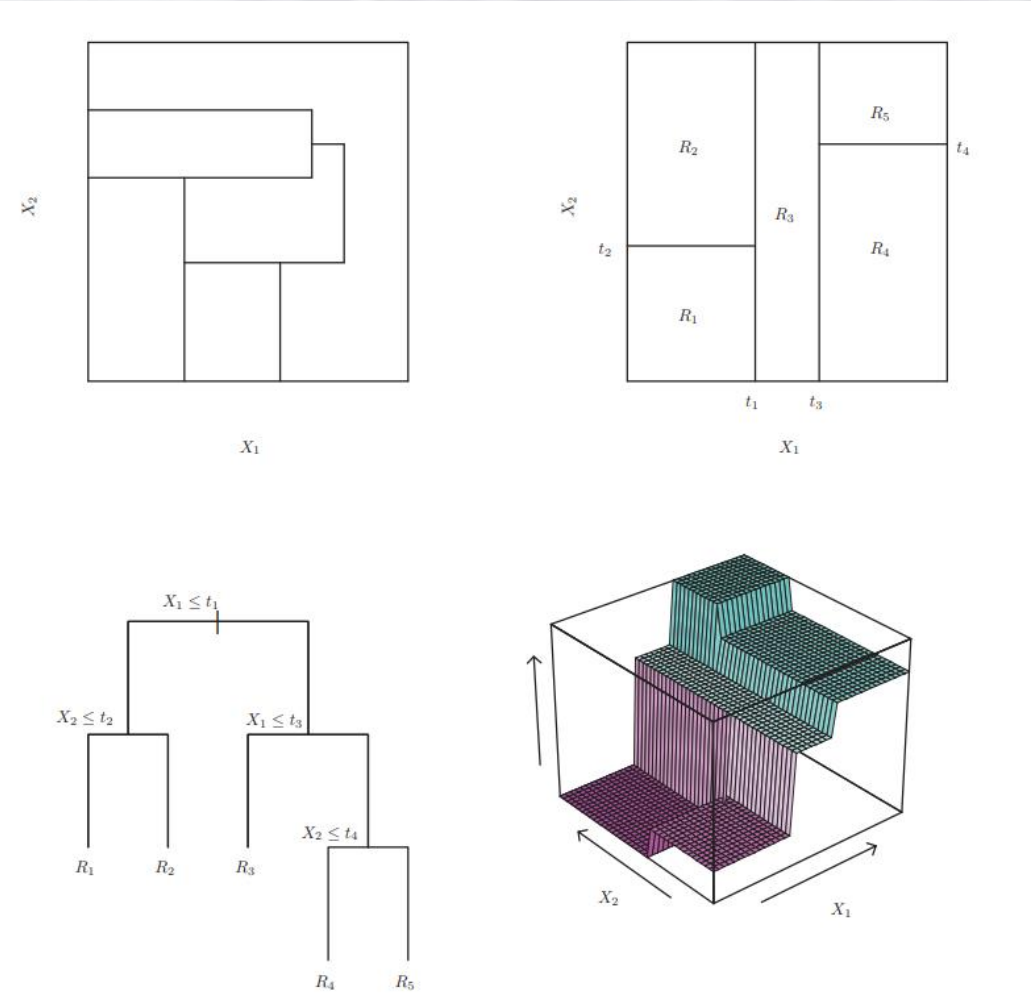
And we seek the value of *j* and *s* that minimize the equation: $\sum_{i:x_i \in R_1(j,s)} \left(y_i - \hat{y}_{R_1}\right)^2 + \sum_{i:x_i \in R_2(j,s)} \left(y_i - \hat{y}_{R_2}\right)^2$

where $\hat{y}_{R1}$ is the mean response for the training observations in R_1(j,s), and $\hat{y}_{R2}$ is the mean response for the training observations in R_2(j,s); finding these values can be done relatively quickly (unless then number of predictors is very large).

(*) Next, we repeat this process, looking for the best predictor and best cut-point in order to split the data further so as to minimize the RSS within each of the resulting regions. However, this time, instead of splitting the entire predictor space, we instead split one of the two previously identified regions (this generates three regions, then four, and so on).

- The process continues until we reach a stopping condition (e.g. stop when each leaf has fewer than some fixed number of instances); **prediction** corresponds with the mean of the training instances in the region to which the test datum belongs.

Regression Trees



Top Left: A partition of a two-dimensional feature space that could *not* result from a recursive binary splitting (why?). Top Right: The output of recursive binary splitting on two-dimensional example. Bottom Left: A tree corresponding to the partition in the top right panel. Bottom Right: A perspective plot of the prediction surface corresponding to that tree.

Regression Trees: Pruning

- Recursive binary splitting (as well as other related tree learning algorithms) are prone to overfitting, leading to a poor test set performance.

To reduce the complexity of a decision tree by **regularization** (*cf.*, *ridge regression*, *lasso*).

Regularization can be achieved through a variety of means:

(*) One possible approach is to build a tree only so long as the decrease in the RSS due to each split exceeds some (high) threshold – this strategy will result in a smaller tree (and potentially reduce model variance at the cost of a little bias) – however this strategy is very short-sighted, since a "weak" split (i.e. achieving a low RSS reduction) could be followed by a more significant split.

(*) A second, and generally superior method, is to apply **pruning**. First we grow a very large tree T_0 (say, using recursive binary splitting), and then prune it back in order to obtain a subtree.

Regression Trees: Pruning

(*) Apply **pruning:** First we grow a very large tree T_0 (say, using recursive binary splitting), and then prune it back in order to obtain a subtree.

Q: How to prune T_0? Note that using cross-validation (CV) isn't feasible in this case, as we would need to apply CV for a large number of possible subtrees.

Regression Trees: Pruning

(*) Apply **pruning:** First we grow a very large tree T_0 (say, using recursive binary splitting), and then prune it back in order to obtain a subtree.

Q: How to prune T_0? Note that using cross-validation (CV) isn't feasible in this case, as we would need to apply CV for a large number of possible subtrees.

- **Cost complexity pruning** is more practical: rather than consider all possible subtrees, we consider a sequence of trees indexed by a tuning parameter α. For each value of α there corresponds a subtree $T \subset T_0$ such that:

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

Regularization/penalty term

Is as small as possible. Here $|T|$ denotes the number of leaves in the tree T, R_m is the rectangle (in the predictor space) corresponding to the mth leaf, and $\hat{y}_{Rm}$ is the predicted response associated with rectangle R_m.

- As is common with regularization techniques, the tuning parameter α controls the trade-off between model fit and complexity.

Regression Trees: Pruning

- **Cost complexity pruning:**

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

- In practice, as α is increase from zero, branches of the decision tree are pruned in a nested and predictable way, so obtaining the whole sequence of subtrees as a function of α is easy.
- The value of α can be selected using a validation set or with CV. We then return to the full data set and obtain the subtree corresponding to α; this **regression tree algorithm** with cost complexity pruning is summarized:

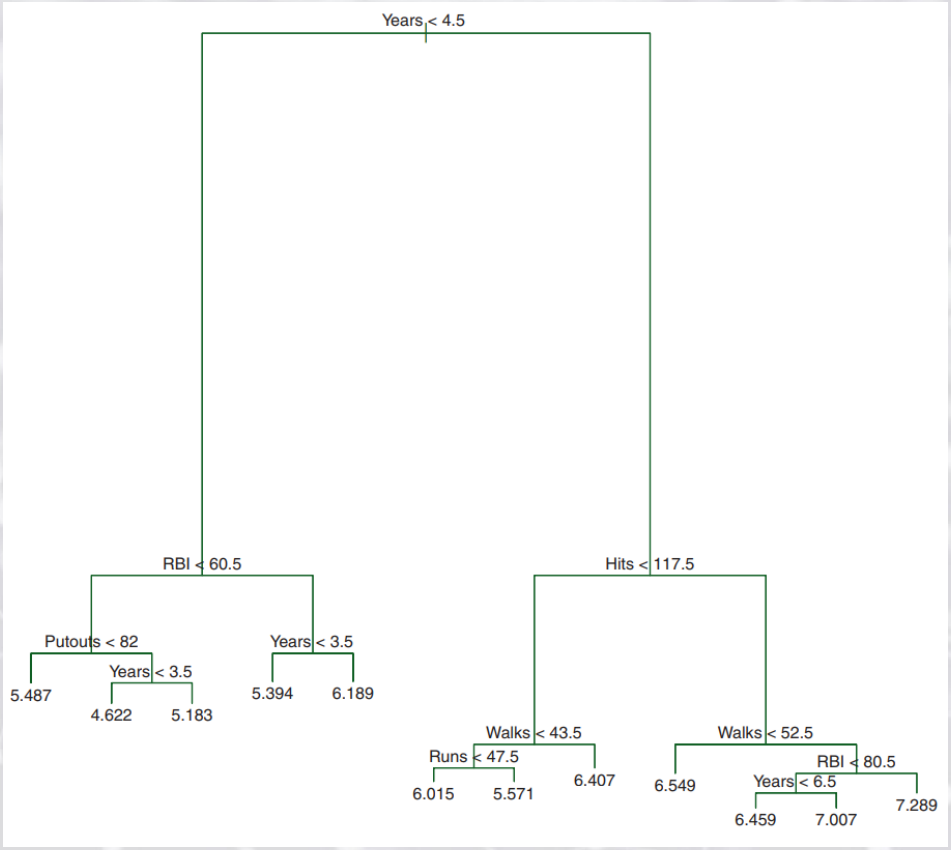
1. Use recursive binary splitting to grow a large tree on the training data, stopping only when each terminal node has fewer than some minimum number of observations.
2. Apply cost complexity pruning to the large tree in order to obtain a sequence of best subtrees, as a function of α.
3. Use K-fold cross-validation to choose α. That is, divide the training observations into K folds. For each $k = 1, \ldots, K$:
   (a) Repeat Steps 1 and 2 on all but the kth fold of the training data.
   (b) Evaluate the mean squared prediction error on the data in the left-out kth fold, as a function of α.
   Average the results for each value of α, and pick α to minimize the average error.
4. Return the subtree from Step 2 that corresponds to the chosen value of α.

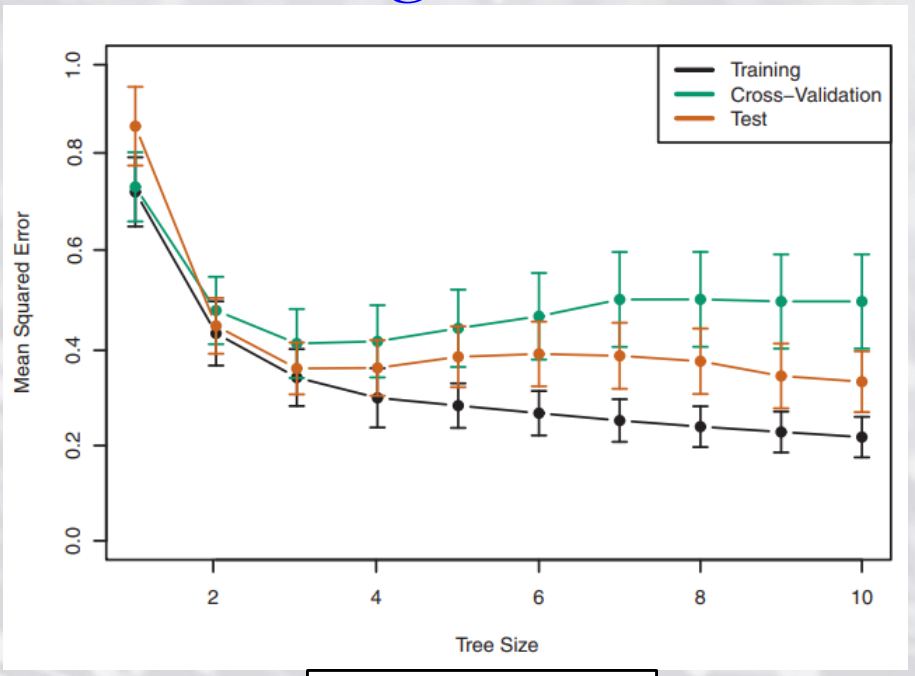
Gives us T_0

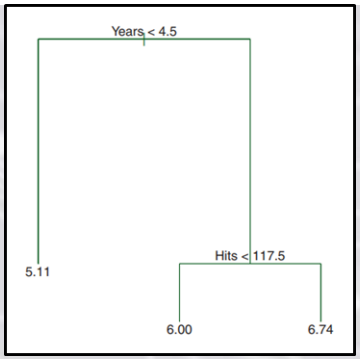
$\text{minimize} \sum_{m=1}^{|T|} \sum_{i:x_i \in R_m} (y_i - \hat{y}_{Rm})^2 + \alpha |T|$

K-fold CV

Regression Trees: Pruning







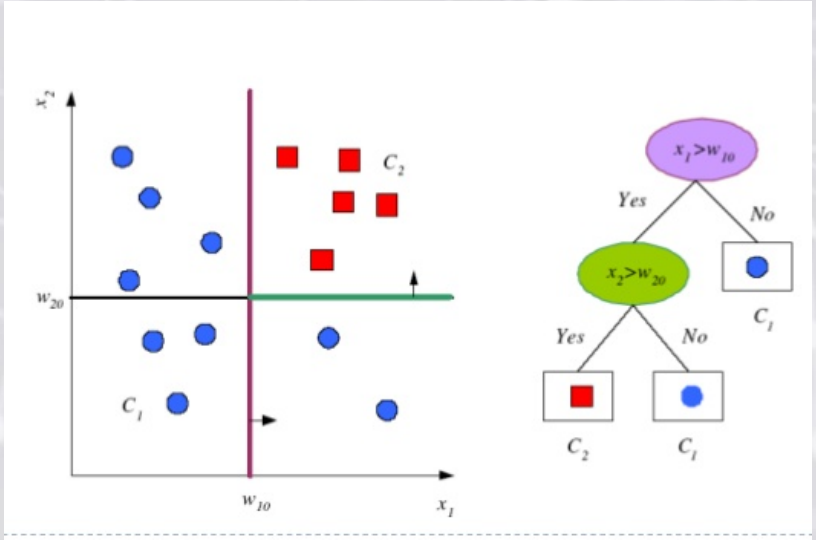
- Left: "Unpruned" regression tree (T_0) for salary response variable with predictors: years, hits, RBI, putouts, runs, and walks. Top Right: Regression tree analysis showing training, CV and test MSE as a function of number of leaves in pruned tree. The minimum CV error occurs at a tree size of three. Bottom Right: pruned tree.

Classification Trees

- Classification trees are used to predict categorical responses (rather than a quantitative response).

- Recall that for regression trees, the predicted response for an observation is the mean/mode response of the training observations that belong to the same terminal node. In contrast, for a classification tree, we predict that each observation belongs to the most commonly occurring class of training observations in the region to which it belong.

- In interpreting the results of a classification tree, we are often interested not only in the class prediction, but also the *class proportions* among the training observations that fall into that region.



Classification Trees

- Growing a classification tree is similar to growing a regression tree. One possibility is to minimize the analogue of RSS (or penalized RSS), by using the *classification error rate*.

- If we define model prediction based on the *most commonly occurring class* of training observations in a given region, the classification error rate is simply the fraction of the training observations that do **not** belong to the most common class:

$$Error = 1 - \max_k \left(\hat{p}_{mk} \right)$$

where $\hat{p}_{mk}$ represents the proportion of training observations in the mth region that are from the kth class.

- Despite the intuitive appeal of using classification error rate to grow a classification tree, in practice, classification error tends not to be sufficiently sensitive for tree-growing. Practitioners instead tend to favor (2) alternative measures for learning classification trees.

Classification Trees: Gini Index & Cross-Entropy

(1) The **Gini index** is defined by:

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

where, again, $\hat{p}_{mk}$ represents the proportion of training observations in the mth region that are from the kth class.

The Gini index measures the total variance across the K classes; it takes on a small value if all of the $\hat{p}_{mk}$'s are close to zero or one. For this reason, the Gini index is known as an **purity measure**, as a small value indicates that a node contains predominantly observations from a single class.

Classification Trees: Gini Index & Cross-Entropy

(1) The **Gini index** is defined by:

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

where, again, $\hat{p}_{mk}$ represents the proportion of training observations in the mth region that are from the kth class.

The Gini index measures the total variance across the K classes; it takes on a small value if all of the $\hat{p}_{mk}$'s are close to zero or one. For this reason, the Gini index is known as an **purity measure**, as a small value indicates that a node contains predominantly observations from a single class.

Example. Consider a two-class problem with 400 cases in each class. Suppose one split created the nodes (300,100) and (100,300), while the other created the nodes (200,400) and (200,0). Both splits produce a misclassification rate of 0.25.

The Gini index for case 1: (300/400)(1-300/400)+(300/400)(1-300/400)=0.375

The Gini index for case 2: (400/600)(1-400/600)+(200/200)(1-200/200)=0.222

(*) The smaller Gini index value for case 2 indicates that this split would be favored (intuitively the split is preferable since of the nodes is **pure**, i.e. it only contains one class).

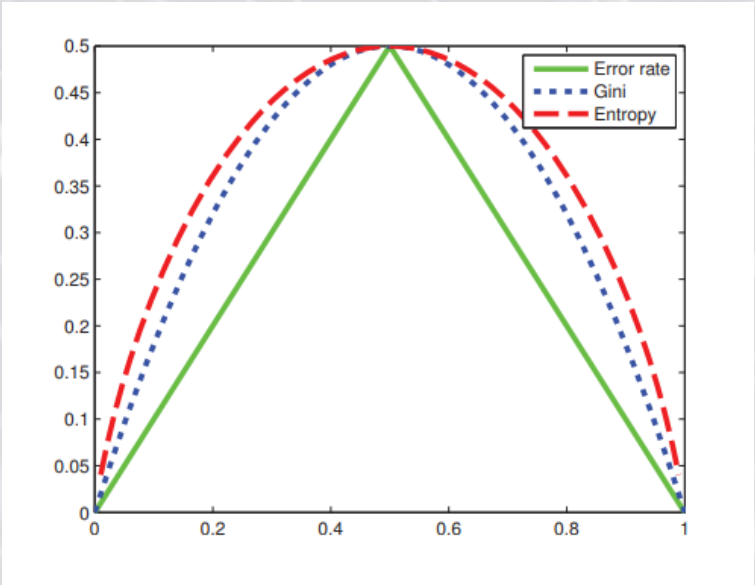
Classification Trees: Gini Index & Cross-Entropy

(2) An alternative to the Gini index is **cross-entropy** defined by:

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

Since $0 \leq \hat{p}_{mk} \leq 1$, it follows that $0 \leq -\hat{p}_{mk} \log \hat{p}_{mk}$. It is not hard to show that the cross-entropy will take on a value near zero if the $\hat{p}_{mk}$'s are all close to zero or one. Thus, like Gini index, the cross-entropy will take on a small value if the mth node is pure. Minimizing cross-entropy is equivalent to maximizing information gain (between the input variables and class label).

In fact, it turns out that the Gini index and cross-entropy are quite similar numerically.



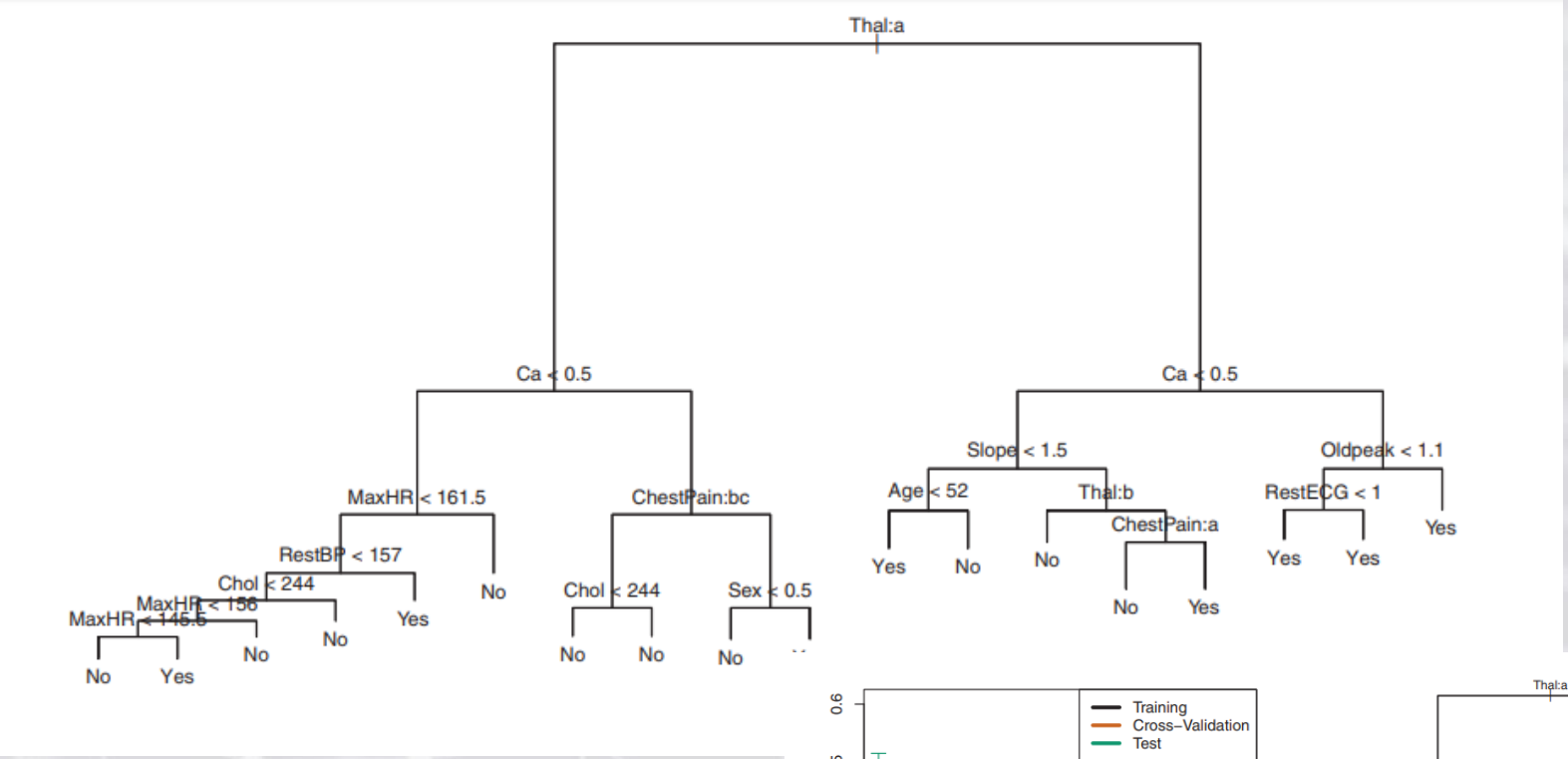
Node impurity measures for binary classification. The horizontal axis corresponds to p, the probability of class 1; the vertical axis corresponds with the error/gini index/entropy measures.

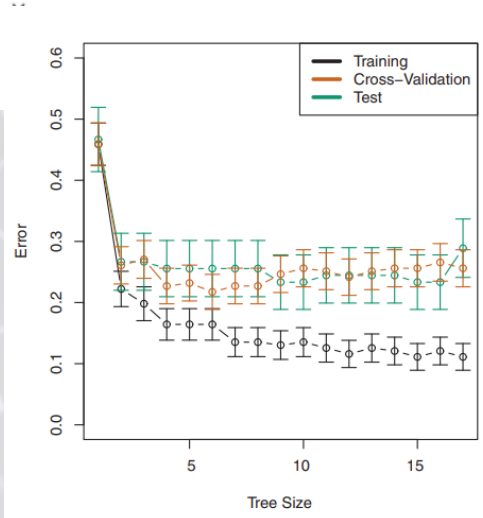
Classification Trees: Gini Index & Cross-Entropy

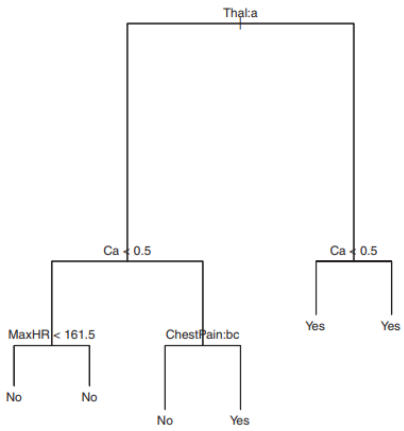
• When building a classification tree, either the Gini index or the cross-entropy are typically used to evaluate the quality of a particular split, since these two approaches are sensitive to node purity.

(*) Note that any of the (3) aforementioned approaches: classification error rate, Gini index or cross-entropy can be used for pruning the three (often classification error rate is preferable if predication accuracy is paramount).

Classification Trees: Gini Index & Cross-Entropy





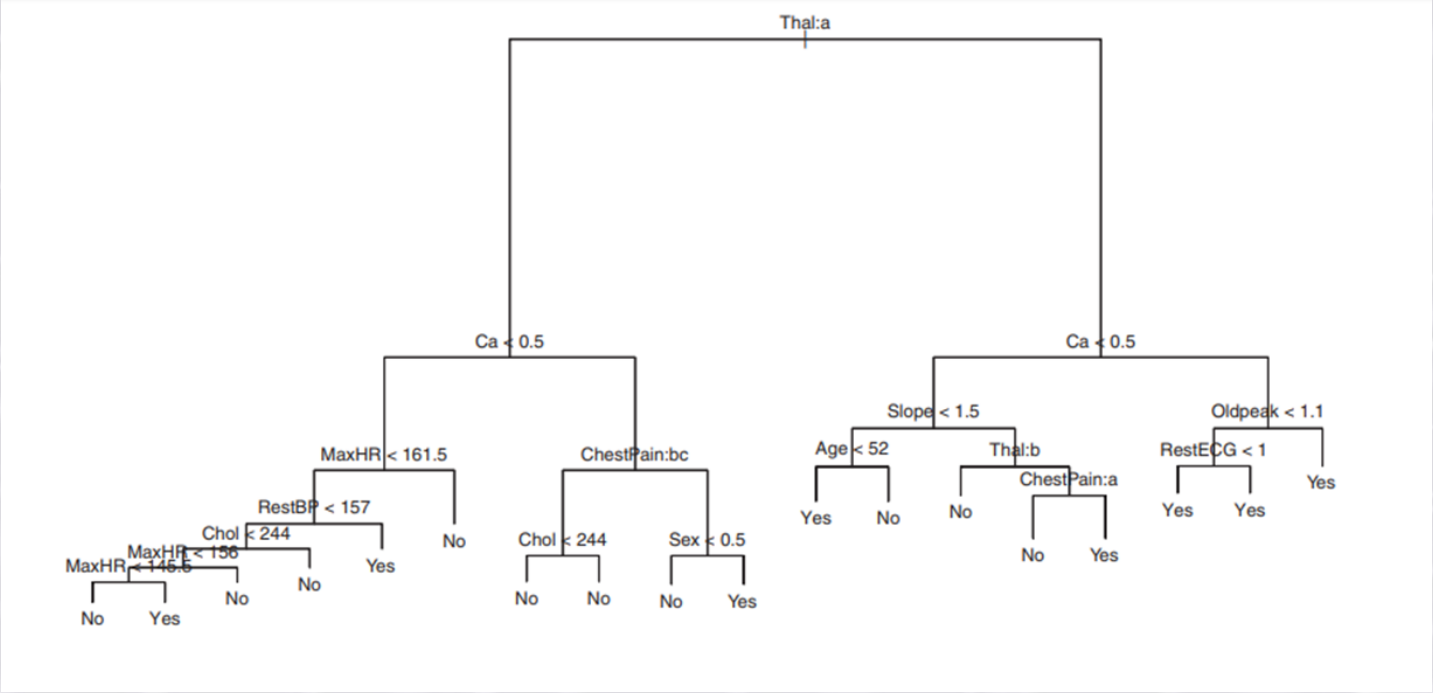


Top: Data containing binary outcome for 303 patients who present with chest pain; outcome Yes indicates hear disease presence; there are 13 predictors (Age, Sex, Chol (cholesterol measurement), etc. Unpruned tree shown. Middle: CV error, training, test error as function of tree size. Right: Pruned tree corresponding with minimal CV error.

Classification Trees

• So far, we have only considered **predictor variables on continuous values**. However, decision trees can be constructed even in the presence of categorical predictor variables (see heart disease data shown – *Sex* is, for instance, a categorical variable).

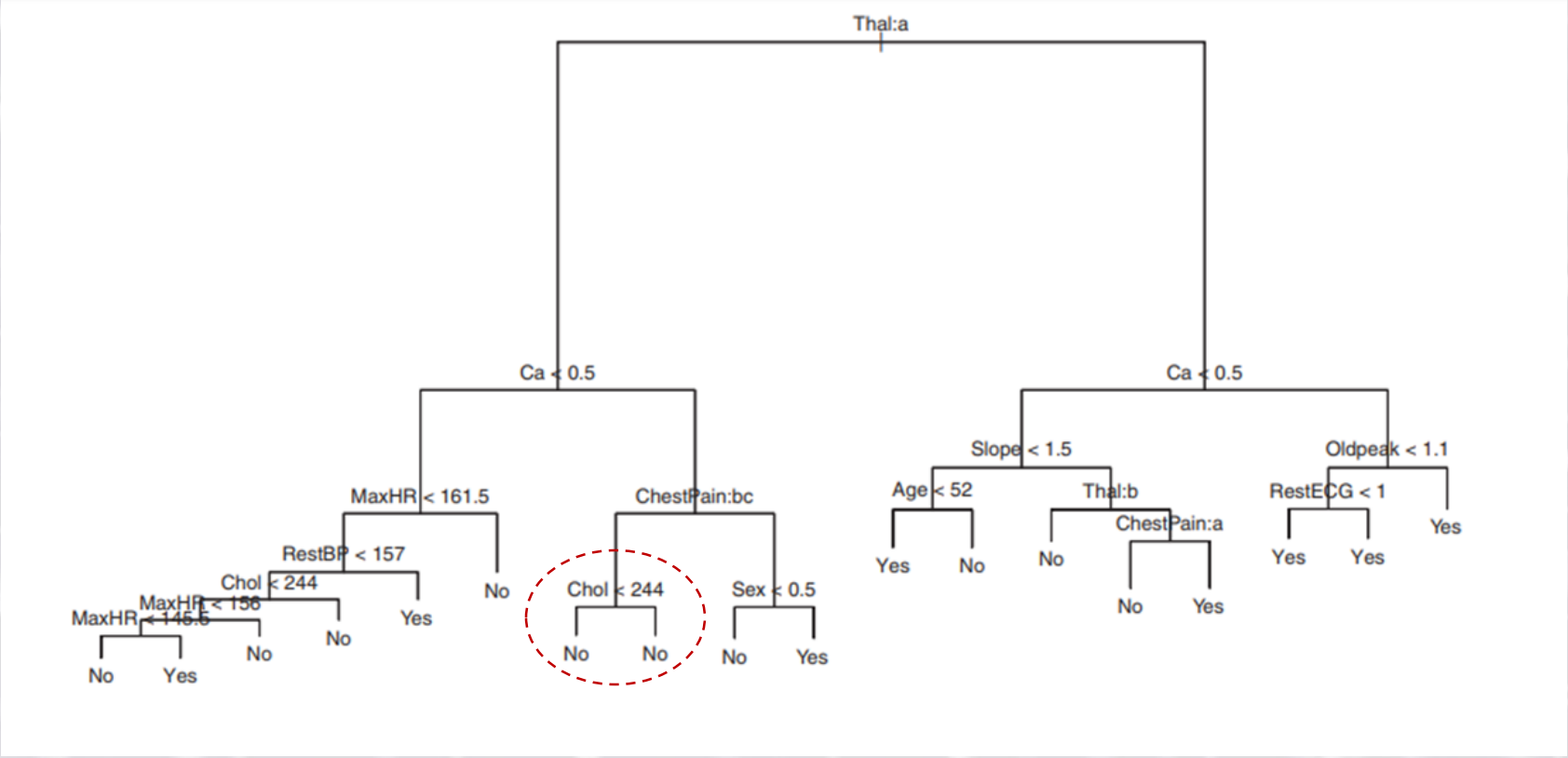
• In this case, a split on a categorical variables amounts to assigning some of the category values to one branch and assigning the remaining to the other branch, e.g., *Thal* (Thalium stress test), is split according to normal (left branch) and fixed or reversible defects (right branch).



Classification Trees

• Notice that some of the splits in the tree shown yield two leaves with the same predicted value (e.g. Chol < 244 => No, No).

Why was the split performed? It yields a larger node purity.



Classification Trees

• Thus far all attributes, categorical or quantitative, and all split positions, one can calculate the impurity (e.g. entropy, Gini index, etc.) and choose the one that corresponds with the minimum entropy, etc. Then tree construction continues recursively and in parallel for all the branches that are not pure until they are pure.

• This procedure is the basis for the well-known *classification and regression tree algorithm* (CART, 1993).

```
GenerateTree(X)
    If NodeEntropy(X)< θ_I /* equation 9.3 */
      Create leaf labelled by majority class in X
      Return
    i ← SplitAttribute(X)
    For each branch of x_i
      Find X_i falling in branch
      GenerateTree(X_i)

SplitAttribute(X)
    MinEnt← MAX
    For all attributes i = 1,...,d
        If x_i is discrete with n values
          Split X into X_1,...,X_n by x_i
          e ← SplitEntropy(X_1,...,X_n) /* equation 9.8 */
          If e<MinEnt MinEnt ← e; bestf ← i
        Else /* x_i is numeric */
          For all possible splits
              Split X into X_1, X_2 on x_i
              e←SplitEntropy(X_1, X_2)
              If e<MinEnt MinEnt ← e; bestf ← i
    Return bestf
```

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

Trees: Comparison with Linear Regression Models

- Regression and classification trees different significantly from classical linear regression models – in particular, as we have mentioned, classical regression models are parametric, whereas decision trees are nonparametric (their complexity is proportional to the dataset size).

- Moreover, consider a generic linear regression model: $f(X) = \beta_0 + \sum_{j=1}^{p} X_j \beta_j$

- Whereas a regression tree assumes a model of the form:

$$f(X) = \sum_{m=1}^{M} c_m 1_{(X \in R_m)}$$

where $R_1,\ldots,R_m$ represent the partition of the feature space, and 1 is the **indicator function**.

Trees: Comparison with Linear Regression Models

$$f(X) = \beta_0 + \sum_{j=1}^{p} X_j \beta_j$$

Linear Regression Model

$$f(X) = \sum_{m=1}^{M} c_m 1_{(X \in R_m)}$$

Regression Tree Model

Q: Which is the better model?

Trees: Comparison with Linear Regression Models

$$f(X) = \beta_0 + \sum_{j=1}^{p} X_j \beta_j$$

Linear Regression Model

$$f(X) = \sum_{m=1}^{M} c_m 1_{(X \in R_m)}$$

Regression Tree Model

Q: Which is the better model?

Of course it depends! (recall the *no free lunch Theorem*)

(*) If the relationship between the features and response variable is well-approximated by a linear model, then linear regression is a strong candidate.

Q: How can we determine the *goodness of fit* of a linear regression model?

Trees: Comparison with Linear Regression Models

$$f(X) = \beta_0 + \sum_{j=1}^{p} X_j \beta_j$$

Linear Regression Model

$$f(X) = \sum_{m=1}^{M} c_m 1_{(X \in R_m)}$$

Regression Tree Model

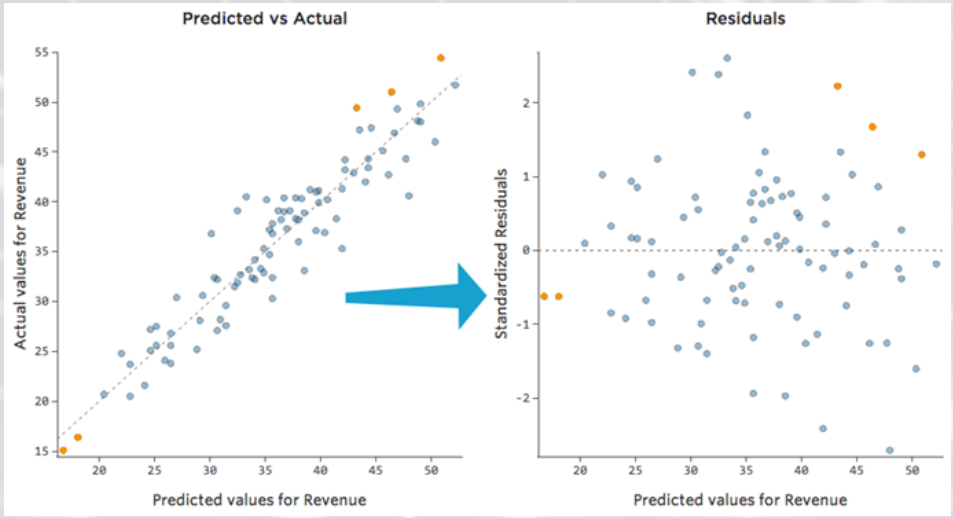
Q: Which is the better model?

Of course it depends! (recall the *no free lunch Theorem*)

(*) If the relationship between the features and response variable is well-approximated by a linear model, then linear regression is a strong candidate.

Q: How can we determine the *goodness of fit* of a linear regression model?

A: Many different ways – compare MSE (typically not the best way), *adjusted* R^2, *BIC* (Bayesian information criterion), perform **hypothesis tests** (e.g. F-test), check residual scatter plot.



Trees: Comparison with Linear Regression Models

$$f(X) = \beta_0 + \sum_{j=1}^{p} X_j \beta_j$$

Linear Regression Model

$$f(X) = \sum_{m=1}^{M} c_m 1_{(X \in R_m)}$$

Regression Tree Model

Q: Which is the better model?

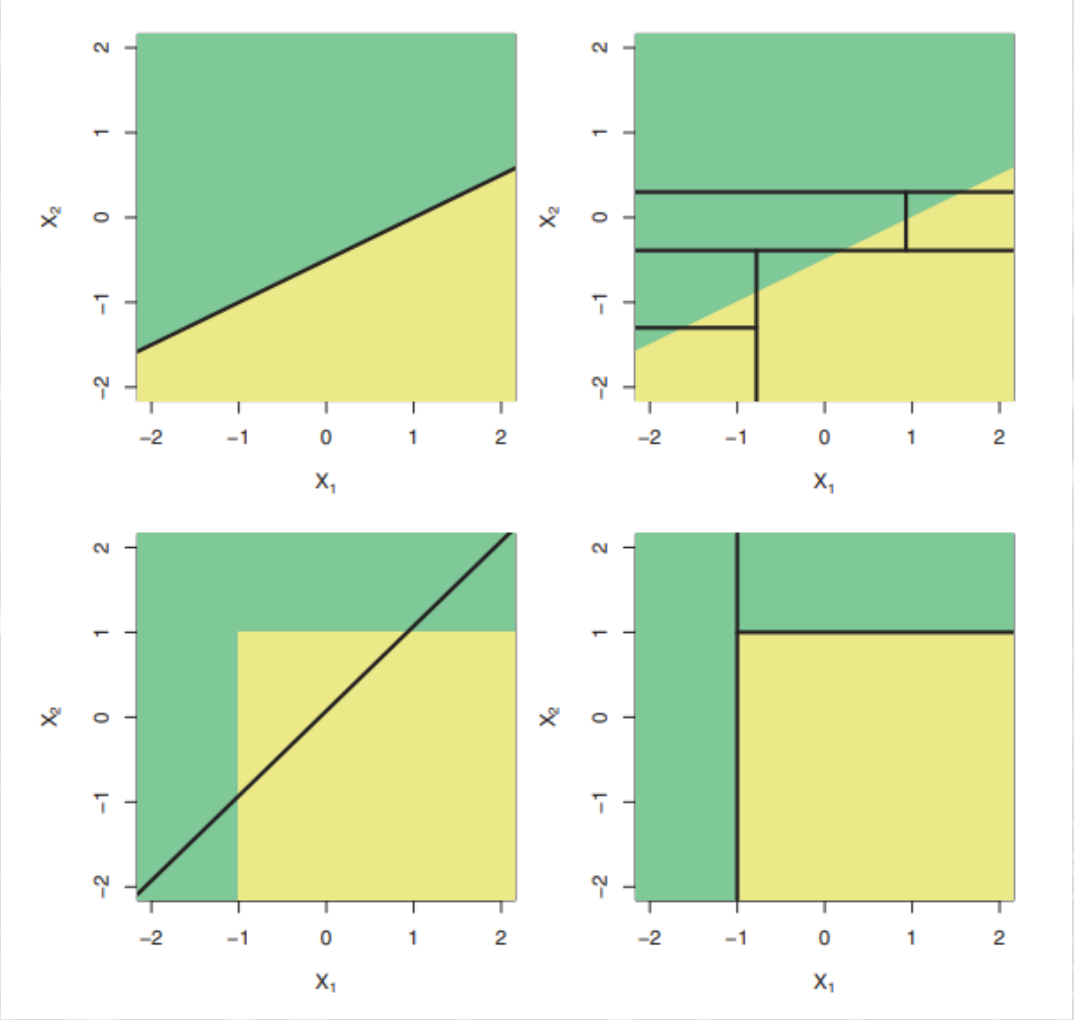
Of course it depends! (recall the *no free lunch Theorem*)

• If the relationship between the features and response variable is well-approximated by a linear model, then linear regression is a strong candidate.

• Otherwise, if there is instead a highly non-linear and complex relationship between the features and response, then the decision tree may outperform classical approaches.

• Of course, a regression tree may also be preferred due to its interpretability and potential for **knowledge extraction** (we explore this notion next).

Trees: Comparison with Linear Regression Models



Top Row: A 2D classification example for which the true decision boundary is linear – the classical regression model outperforms the decision tree. Bottom Row: The true decision boundary is non-linear here; the decision trees outperforms classical regression.

Rule Extraction from Trees

• A decision tree performs its own *hierarchical* feature extraction (indeed, this is one of their potential appeals) – in the sense that features closer to the root are considered to be of more global importance.

• It is possible to use a decision tree explicitly for feature extraction: we build a tree and then only retain those features used be the tree.

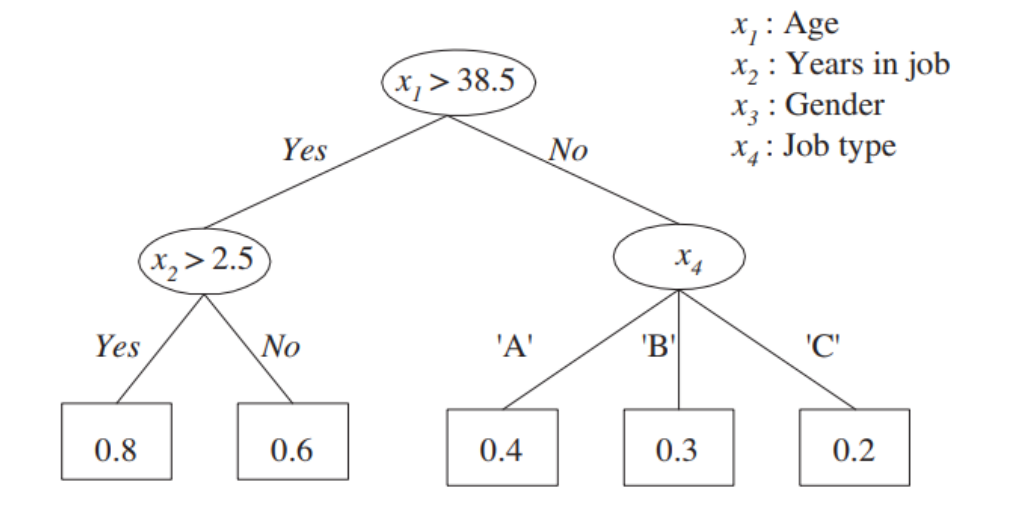
(*) As noted, a major advantage of decision trees is their **interpretability**. The decision nodes convey conditions that are simple to understand. As such, each path that traverses the tree can be interpreted as a conjunction of tests or rules, which all need to be satisfied to reach a particular leaf.

(*) Tree-path traversal searches are common in agent-based modeling application in A.I. more generally; these paths can be expressed as **IF-THEN rules**.

Rule Extraction from Trees

(*) Tree-path traversal searches are common in agent-based modeling application in A.I. more generally; these paths can be expressed as IF-THEN rules.

For example: Here is an invented decision tree, where each path from root to leaf can be written down as a *conjunctive rule*, comprising conditions defined by the decision nodes along the path.



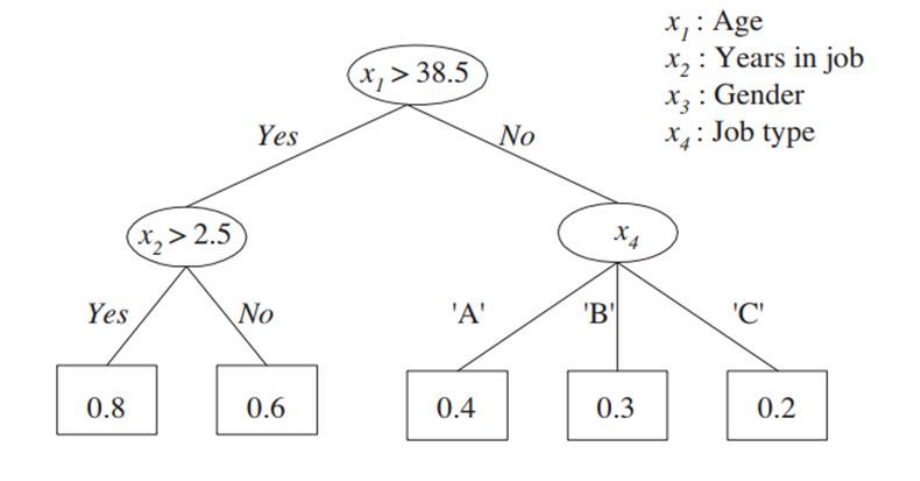
Here are the corresponding rules:

R1: IF (age>38.5) AND (years-in-job>2.5) THEN y =0.8
R2: IF (age>38.5) AND (years-in-job≤2.5) THEN y =0.6
R3: IF (age≤38.5) AND (job-type='A') THEN y =0.4
R4: IF (age≤38.5) AND (job-type='B') THEN y =0.3
R5: IF (age≤38.5) AND (job-type='C') THEN y =0.2

Rule Extraction from Trees

(*) A rule base likes this facilitates knowledge extraction and can be directly used for inference (without need for the tree data structure itself).

• Having a rule set additionally enables domain experts to verify the validity of the model; the rules represent a structural compression of the data according to the most important features and split positions.



• For instance, one can see that with respect to the response (y), people who are thity-eight years old or less are different from people who are thirty-nine years old or more; moreover, among the latter group, it is the job type that makes them different, whereas with the former group it is the number of years in the job that is the most discriminating characteristic.

• For classification trees it is possible that one or more leaf has the same label, in which case multiple conjunctive expressions can be combined as a *disjunction* (OR).

Advantages and Disadvantages of Trees

- Trees are much more interpretable than traditional regression models.

- Trees can handle categorical variables easily); commonly, regression models require the inclusion of numerous "dummy variables" to accommodate categorical variables.

- Trees admit of rule extraction/knowledge extraction techniques; they naturally perform feature extraction on the data.

- Some argue that trees more closely resemble logical/rule-based human decision-making than traditional regression models.

- Unfortunately, trees generally do not have the same level of predictive accuracy as other regression and classification approaches – nevertheless this shortcoming can often be alleviated through **bagging, random forests** and **boosting** (although this comes at the cost of interpretability).

Decision Trees: Bagging

- Decision trees tend to suffer from high variance. This means that in general, prediction values vary widely. In contrast, a low variance model will yield similar results if applied repeatedly to distinct data sets.

- Linear regression models tend to have usually have low variance if the ration of data points (n) to number of predictor variables/complexity (p) is moderately large – i.e. in the case of an *overdetermined system*.

- Recall that *bootstrapping* in statistics consists of performing random samples with replacement.

- **Bagging** (i.e. *bootstrap aggregation*) is a general-purpose procedure for reducing the variance of a statistical learning method; it is often used to improve the performance of decision trees.

Decision Trees: Bagging

- Recall that the *Central Limit Theorem* (CLT) tells us that when we aggregate independent data (for a sufficiently large sample) the variance reduces in proportion to the sample size (σ^2/n). This is in fact a very useful "rule" in ML: *averaging a set of observations reduces variance*.

- This should remind us of the benefits *ensemble learning*, in which we build a set of prediction models: $f^1(x)$, $f^2(x)$,…,$f^B(x)$, and average them in order to obtain a single, low-variance model given by:

$$f_{avg}(x) = \frac{1}{B}\sum_{b=1}^{B} f^b(x)$$

Decision Trees: Bagging

- With ensemble learning in general, it is often the case that we partition the training data into smaller subsets and learn a model for each subset. This is however not always possible when we don't have access to multiple training data set.

- Alternatively, we can bootstrap by taking repeated samples from the training data set. In this way we generate B different bootstrapped training data sets, and then train our method on the *b*th bootstrapped set to get $f^{*b}(x)$. Finally, we average the predictions as before:

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

(*) This method is called **bagging**. To apply bagging to regression trees, we construct B separate regression trees using B bootstrapped training sets, and average the resulting predictions. Note that the trees are **not pruned** – so each has high variance and low bias, but, again the averaging mechanism has the effect of reducing the overall model variance.

(*) Bagging has been shown to be very effective by combining hundreds/thousands of trees in total.

Decision Trees: Bagging

Q: How do we perform bagging with classification (i.e. when the output is categorical)?

(*) One straightforward approach is to simply apply bootstrapping as before and to use the "majority vote" rule when generating the overall prediction.

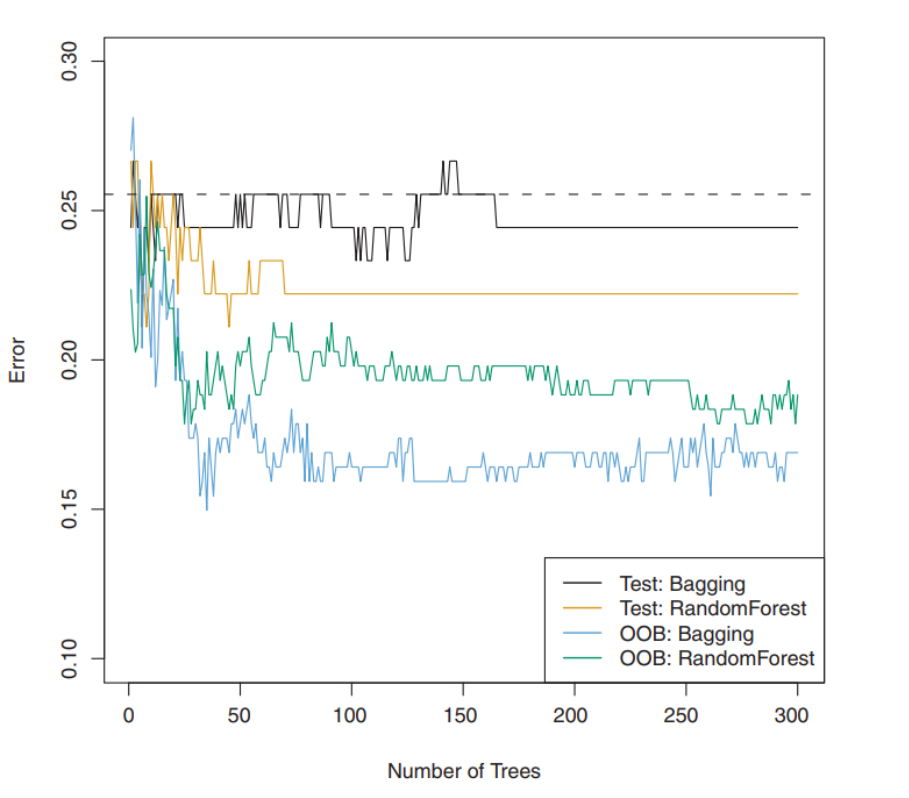
• The figure on the next slide displays results from bagging trees on the previously mentioned data related to heart disease; the test error rate is shown as a function of the number of trees constructed using bootstrapped training data sets.

• As evidenced by these results, the number of trees B is, in fact, not a critical parameter with bagging, in the sense that a large B does not necessarily yield overfitting. In practice, B can be tuned through cross-validation using out-of-bag error estimation (explained next).

Decision Trees: Bagging

• Test error can be estimated for a bagged model without using cross-validation. Because we used bagging, each bagged tree only uses approximately 2/3 of the total data for training. We can therefore estimate the test error using OOB (out-of-bag) observations for each element x_i in the training data set (we just average/take majority vote) for the trees that were not trained on datum xi.

• The resulting OOB error is therefore a valid estimate of the test error for the bagged model, since the response for each observation is predicted using only the trees that were not fir using that observation. Note that OOB is particularly useful in lieu of cross-validation for large data sets.



Decision Trees: Bagging

• As mentioned, bagging can improve the accuracy of a decision tree – but it often comes at the cost of interpretability.

• Although the collection of bagged trees is much more difficult to interpret than a single tree, one can obtain an overall summary of the importance of each predictors using the *RSS* (for regression tress) or *Gini index* (for classification trees).

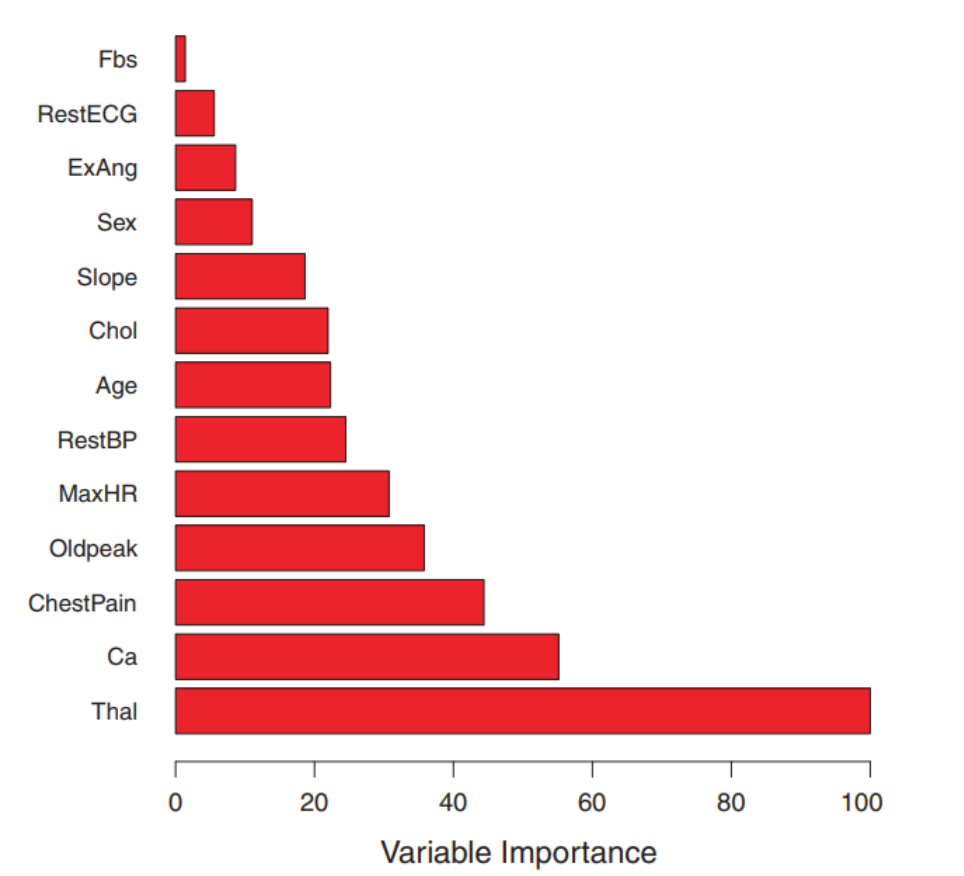
• With regression trees, we can record the total amount that the RSS is decreased due to splits over a given predictor, averaged over all B trees. The larger the value, the more important the predictor.

• Similarly, for classification trees, we can add up the total amount that the Gini index is *decreased* by splits over a given predictor, averaged over all B trees.

(*) In this way it is possible to assess **variable importance** for bagged trees.

Decision Trees: Bagging

• The graph shows ranked variables (by importance) according to the Gini index for the heart disease data set.



Decision Trees: Random Forests

• **Random forests** can improve bagging by enforcing a restriction that decorrelates bagged trees.

• Random forests use a modified tree learning algorithm that selects, at each candidate split in the learning process, a random subset of the features (this is sometimes called "feature bagging").

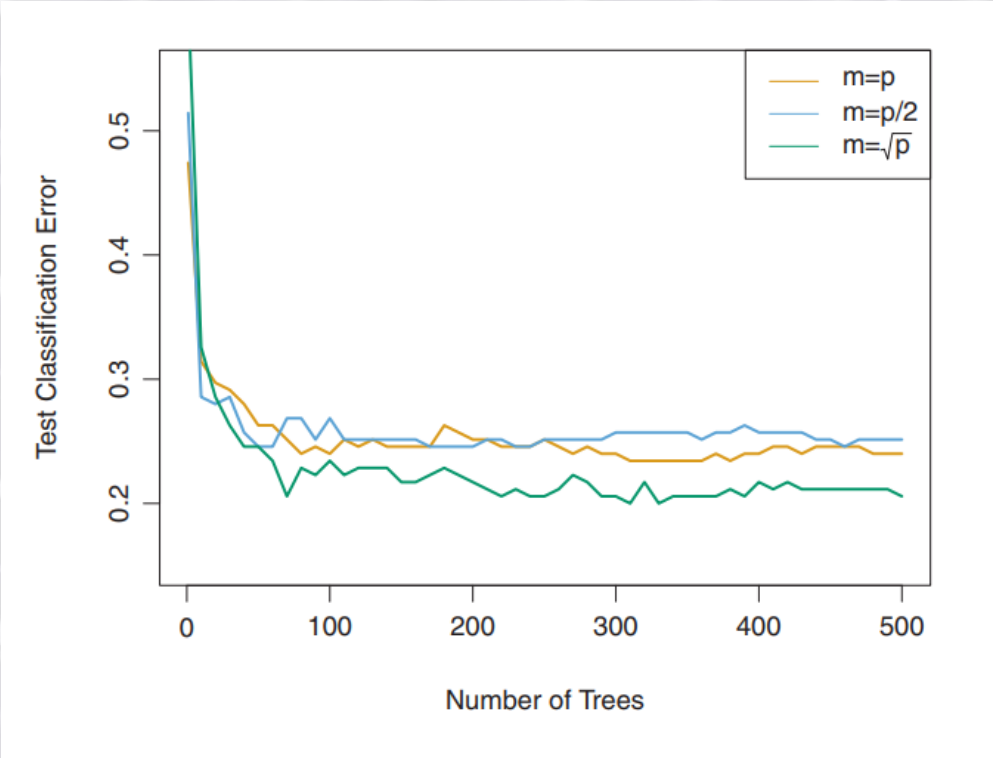
• The basic rationale for using a random subset is as follows: if a subset of the features in a data set are strongly associated with the predictor variable, then these features will be chosen in many of the bagged trees, causing the trees to be *correlated*. In the end, averaging many highly correlated models does not necessarily lead to a substantial reduction in variance.

• Random forests overcome this correlation problem by forcing each split to consider only a subset of the predictors – this allows other predictors to "have a chance."

(*) Specifically, a random forest usually uses a relatively small predictor subset size ($\sqrt{p}$), where p is the number of predictors. On the heart disease data set (shown) random forests using $\sqrt{p}$ predictors leads to a reduction in both test error and OOB error over bagging.

Decision Trees: Random Forests

• With large numbers of correlated predictors, the results can be even more dramatic. Consider a data set of high-dimensional gene expression measurements of 4,718 genes from 349 patients. Using random forests based on 500 genes with the largest variance in the training set and subsets of size $\sqrt{p}$ improved the overall classification accuracy. As with bagging, random forests tend to not overfit as B is increased.



Decision Trees: Boosting

• Like bagging, **boosting** a general approach applicable to many different ML models for regression and classification.

• With bagging, we create multiple copies of the training data via bootstrapping, fitting separate decision trees to each copy, and then combine these trees through an ensemble.

• Boosting is similar, except that **the trees are grown sequentially**, meaning that each tree is grown using information from the previously grown trees. Boosting does not involve bootstrap sampling; on the contrary, each tree is fit on a modified version of the original data set.

Decision Trees: Boosting

• Unlike fitting a single, large decision tree which can potentially overfit the data, boosting learns slowly.

• Given the current model, we fit a decision tree to the residuals from the model, *viz.*, we fit a tree using the current residuals, instead of the outcome variable (y) as the response.

• We then add this new decision tree into the fitted function in order to update the residuals. Each of these trees can be quite small, with just a few leaves, determined by the parameter d (the number of splits) in the algorithm.

• By fitting small trees to the residuals, we slowly improve $\hat{f}$ in areas where it does not perform well. Lastly, the **shrinkage parameter** λ slows the process down even further, allowing more and different shaped trees to hone in on the residuals.

(*) Note that this procedure is strongly sequential (in contrast to bagging), meaning that the construction of each tree depends on the previous trees.

Decision Trees: Boosting

• Boosting algorithm 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:

   $$\hat{f}(x) \leftarrow \hat{f}(x) + \lambda \hat{f}^b(x).$$

   (c) Update the residuals,

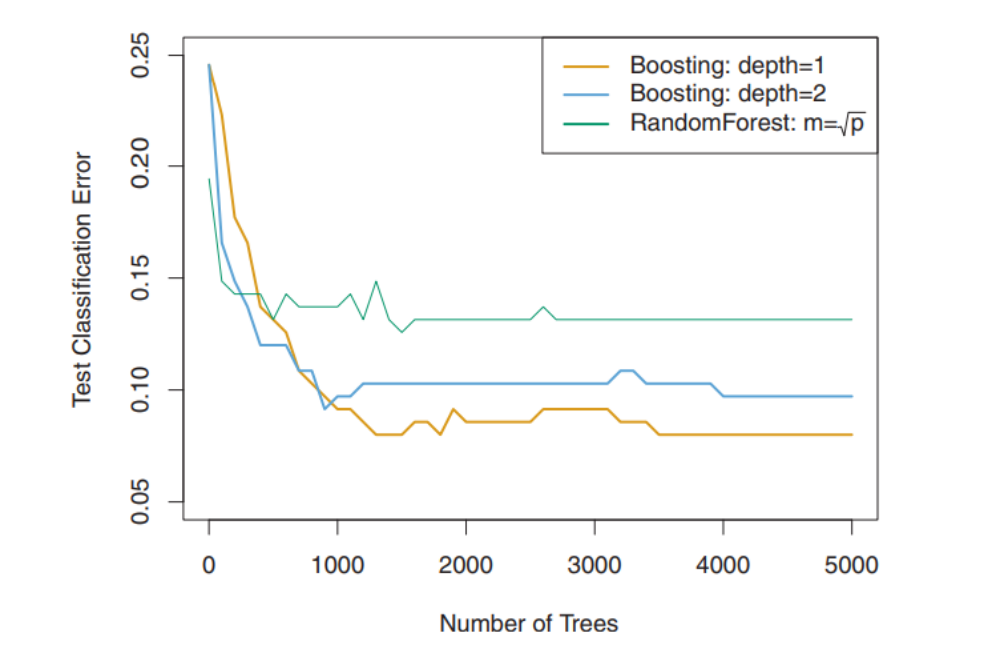
   $$r_i \leftarrow r_i - \lambda \hat{f}^b(x_i).$$

3. Output the boosted model,

$$\hat{f}(x) = \sum_{b=1}^{B} \lambda \hat{f}^b(x).$$

• Boosting has **(3) tuning parameters**: (1) the number of trees B; unlike bagging and random forests, if B is too large then boosting can overfit; (2) the shrinkage parameter λ, a small positive number; this controls the rate at which boosting learns – note that a very small value can require a very large value of B to achieve good performance; (3) the number d of splits in each tree, which controls the complexity of the boosted ensemble; when d=1 the tree is called a stump, consisting of a single split.

Decision Trees: Boosting



• Results from performing boosting and random forests on gene expression data for cancer prediction; test error is shown as a function of the number of trees.

Research Applications of Decision Trees

• Decision trees are used extensively across a broad spectrum of data mining and ML domains; in particular the **CART algorithm** (*classification and regression tree algorithm*) and **ID3 algorithm** (*iterative dicohtomiser 3*) are two of the most popular implementations of decision trees.

(*) We have previously introduced the CART algorithm; the ID3 algorithm is similar. With ID3 we employ a top-down, greedy approach; the attribute to be split is based on (2) criteria: entropy and information gain.

Research Applications of Decision Trees

Baradwaj, et al., "Mining Educational Data to Analyze Students' Performance", IJACSA, 2011.

• The authors apply knowledge discovery using the ID3 algorithm for prediction about student academic performance.

TABLE I. STUDENT RELATED VARIABLES

Variable	Description	Possible Values
PSM	Previous Semester Marks	{First > 60% Second >45 & <60% Third >36 & <45% Fail < 36%}
CTG	Class Test Grade	{Poor , Average, Good}
SEM	Seminar Performance	{Poor , Average, Good}
ASS	Assignment	{Yes, No}
GP	General Proficiency	{Yes, No}
ATT	Attendance	{Poor , Average, Good}
LW	Lab Work	{Yes, No}
ESM	End Semester Marks	{First > 60% Second >45 & <60% Third >36 & <45% Fail < 36%}

IF PSM = 'First' AND ATT = 'Good' AND CTG = 'Good' or 'Average' THEN ESM = First
IF PSM = 'First' AND CTG = 'Good' AND ATT = "Good' OR 'Average' THEN ESM = 'First'
IF PSM = 'Second' AND ATT = 'Good' AND ASS = 'Yes' THEN ESM = 'First'
IF PSM = 'Second' AND CTG = 'Average' AND LW = 'Yes' THEN ESM = 'Second'
IF PSM = 'Third' AND CTG = 'Good' OR 'Average' AND ATT = "Good' OR 'Average' THEN PSM = 'Second'
IF PSM = 'Third' AND ASS = 'No' AND ATT = 'Average' THEN PSM = 'Third'
IF PSM = 'Fail' AND CTG = 'Poor' AND ATT = 'Poor' THEN PSM = 'Fail'

Figure 3. Rule Set generated by Decision Tree

https://arxiv.org/ftp/arxiv/papers/1201/1201.3417.pdf

Research Applications of Decision Trees

Kumar, et al., "Decision Support System for Medical Diagnosis Using Data Mining ", IJCSI, 2011.

• The authors focus on applications of Medical diagnosis by learning pattern through the collected data of diabetes, hepatitis and heart diseases in order to assist physicians. In the paper, they propose the use of decision trees with ID3 algorithm and CART algorithms to classify these diseases and compare the effectiveness, correction rate among them.

Table 10: Confusion matrix of CART algorithm-heart disease dataset

TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area	Class
0.702	0.258	0.702	0.702	0.702	0.726	No
0.742	0.298	0.742	0.742	0.742	0.726	Yes

Table 11: Confusion matrix of CART algorithm- hepatitis dataset

TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area	Class
0.91	0.769	0.859	0.91	0.884	0.541	Live
0.231	0.09	0.933	0.831	0.273	0.541	Die

Table 12: Confusion matrix of CART algorithm- diabetes dataset

TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area	Class
0.534	0.132	0.884	0.934	0.6	0.727	Yes
0.868	0.466	0.776	0.868	0.82	0.727	No

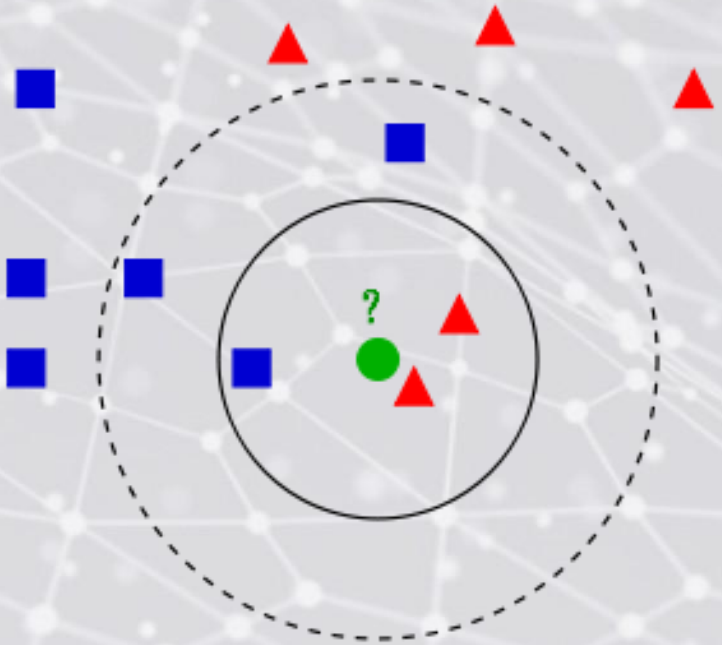
Some classification rules for diabetes datasets are as follows,

1. Age <= 28 AND Triceps skin fold thickness > 0 AND Triceps skin fold thickness <= 34 AND Age > 22 AND No.timespreg <= 3 AND Plasma gc(2) <= 127: No (61.0/7.0)
2. Plasma gc(2) <= 99 AND 2-Hour serum insulin <= 88 AND 2-Hour serum insulin <= 18 AND Triceps skin fold thickness <= 21: No (26.0/1.0)
3. Age <= 24 AND Triceps skin fold thickness > 0 AND Body MI <= 33.3: No (37.0) Diastolic blood pressure <= 40 AND Plasma gc(2) > 130: Yes (10.0)
4. Plasma gc(2) <= 107 AND Diabetespf <= 0.229 AND Diastolic blood pressure <= 80: No (23.0)
5. No.timespreg <= 6 AND Plasma gc(2) <= 112 AND Diastolic blood pressure <= 88 AND Age <= 35: No (44.0/8.0)
6. Age <= 30 AND Diastolic blood pressure > 72 AND Body MI <= 42.8: No (41.0/7.0)

https://pdfs.semanticscholar.org/505b/12958b2f718a24f92cf249d6d1fc7d224bd1.pdf

k-NN

- Like decision trees, **k-NN** (*k-nearest neighbors*) is a non-parametric model that can be used in both regression and classification settings. Despite the fact that it is an extremely simple algorithm, k-NN can nevertheless yield surprisingly strong results by learning complex partitions of the feature space.



k-NN: Classification

- Given a positive integer *k* (a hyper-parameter) and a test observation x_0, the k-NN classifier first identifies the K points in the training data that are closest to x_0, represented by N_0.

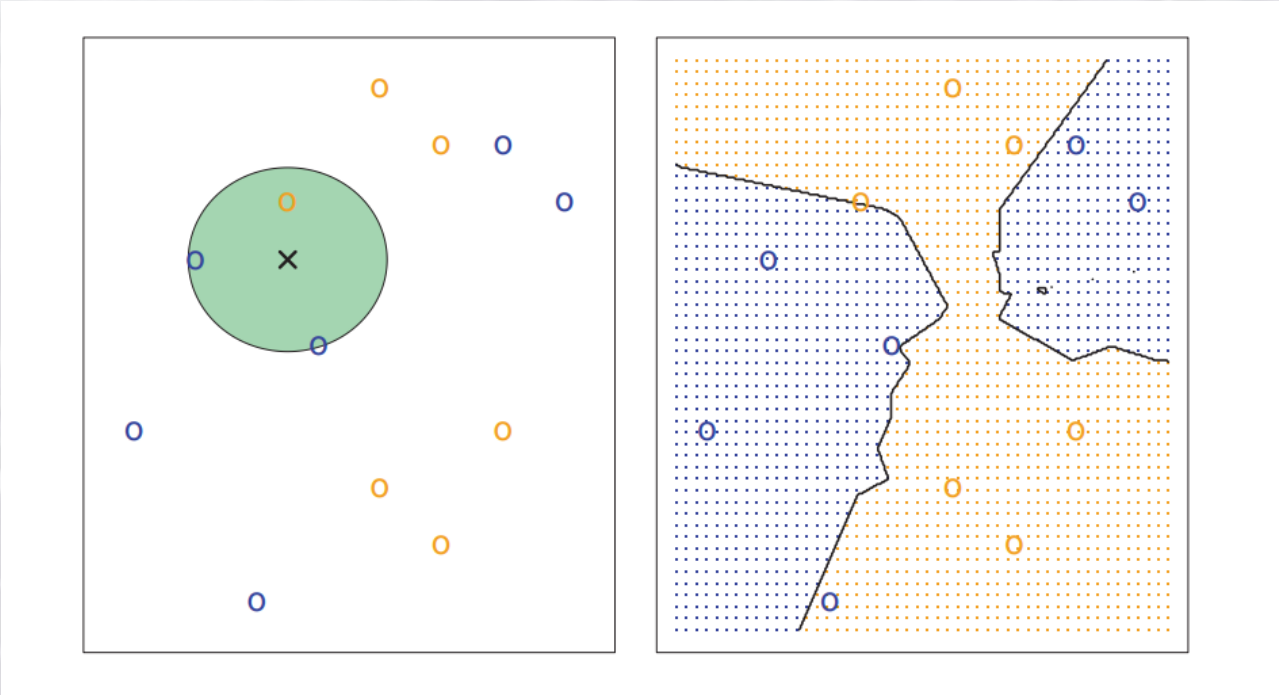
- It then estimates the conditional probability for class *j* as the fraction of points in N_0 whose response values equal *j*:

$$P(Y = j \mid X = x_0) = \frac{1}{K} \sum_{i \in N_0} I(y_i = j)$$

where ***I* is the indicator function**, so that I=1 when y_i=j.

- Finally, k-NN applies Bayes rule and classifies the test observation x_0 to the class with the largest probability.

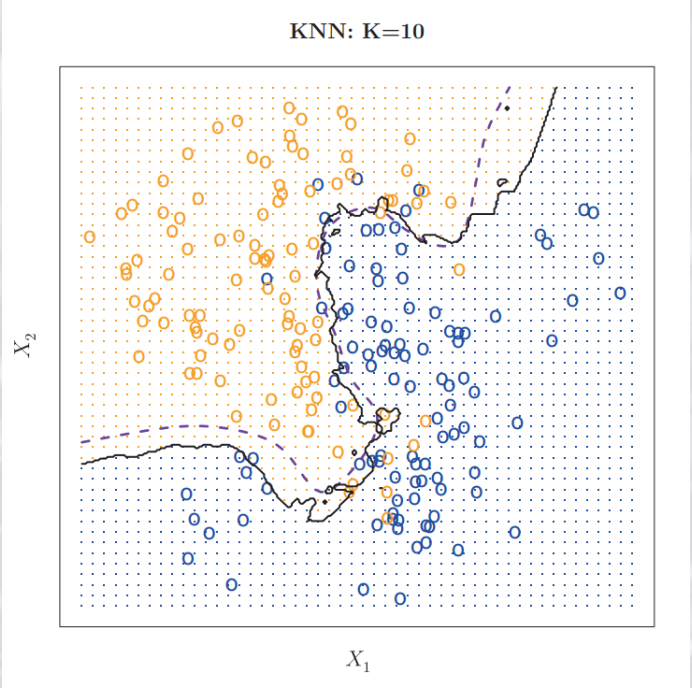
k-NN: Classification

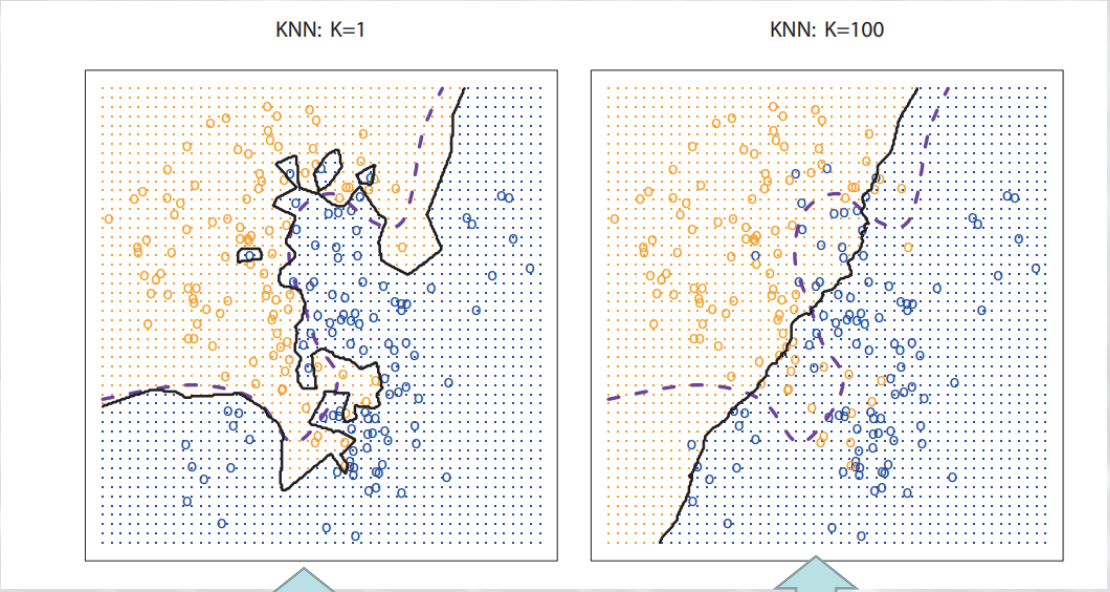


$$P(Y = j \mid X = x_0) = \frac{1}{K} \sum_{i \in N_0} I(y_i = j)$$

- The figure shows an example of k-NN with k=3. Left: a test observation at which a predicted class label is desired is shown as a black cross. The three closets points to the test observation are identified, and it is predicted that the test observation belongs to the most commonly-occurring class (blue). Right: the k-NN decision boundary for this example is shown in black. The blue/orange grid indicates the region in which the test observation will be assigned to blue/orange class; again, the non-parametric model partitions the input space into local regions – crucially relying on all of the training data to do so.

k-NN: Classification





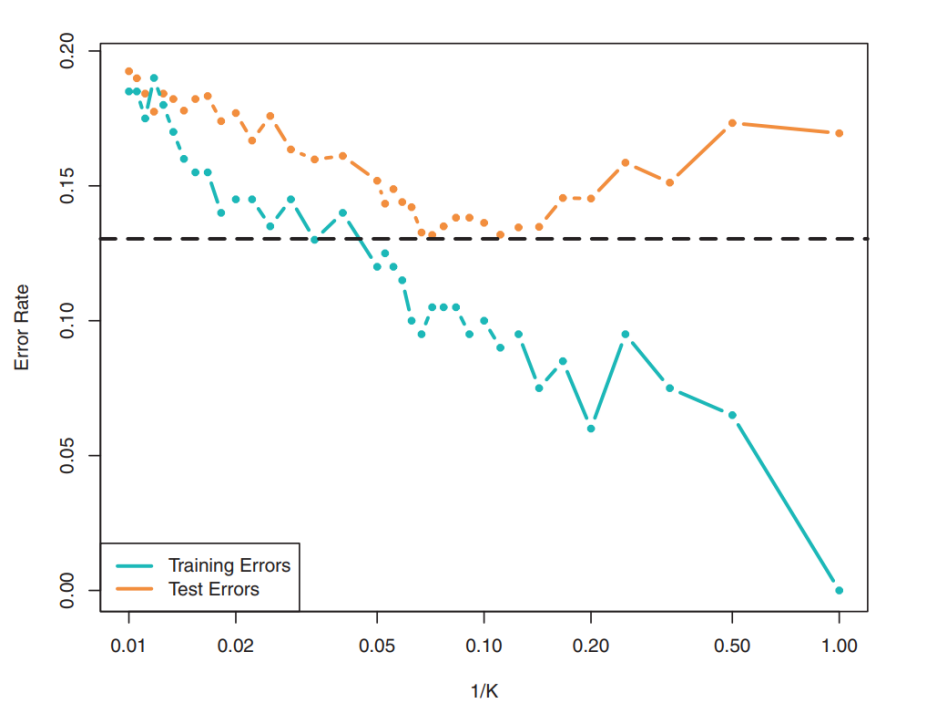
Low bias/high variance (*overfit*)

High bias/low variance (*underfit*)

- Naturally, the choice of k for k-NN can have a drastic effect on the classifier obtained. In the figures three classifiers are shown, one for which k=1, one for which k=10, and the other uses k=100.
- In the former case, the decision boundary is overly flexible; this corresponds with a low bias/high variance model. As k grows, the method becomes less flexible and produces a decision boundary that is approximately linear. The dotted line indicates the **Bayes decision boundary** (i.e. the optimal boundary). When k=10 the k-NN decision boundary closely resembles the Bayes decision boundary.

k-NN: Classification

- As is customary with ML models, the training error for a k-NN model is an *optimistic estimate* for the test error. For example when k=1, the training error might be very low (even zero), whereas the test error could still be very high due to the poor generalization ability of the model.



- The plot shows k-NN train and test errors as a function of 1/k (which can be considered a *model flexibility* parameter – or equivalently, the number of neighbors k as k decreases). Note that the test error rate graph exhibits a characteristic **U-shape**; if we choose a parameter value k corresponding with the "elbow" of this U-curve we are in the "Goldilocks zone" between underfitting and overfitting.

k-NN: Regression

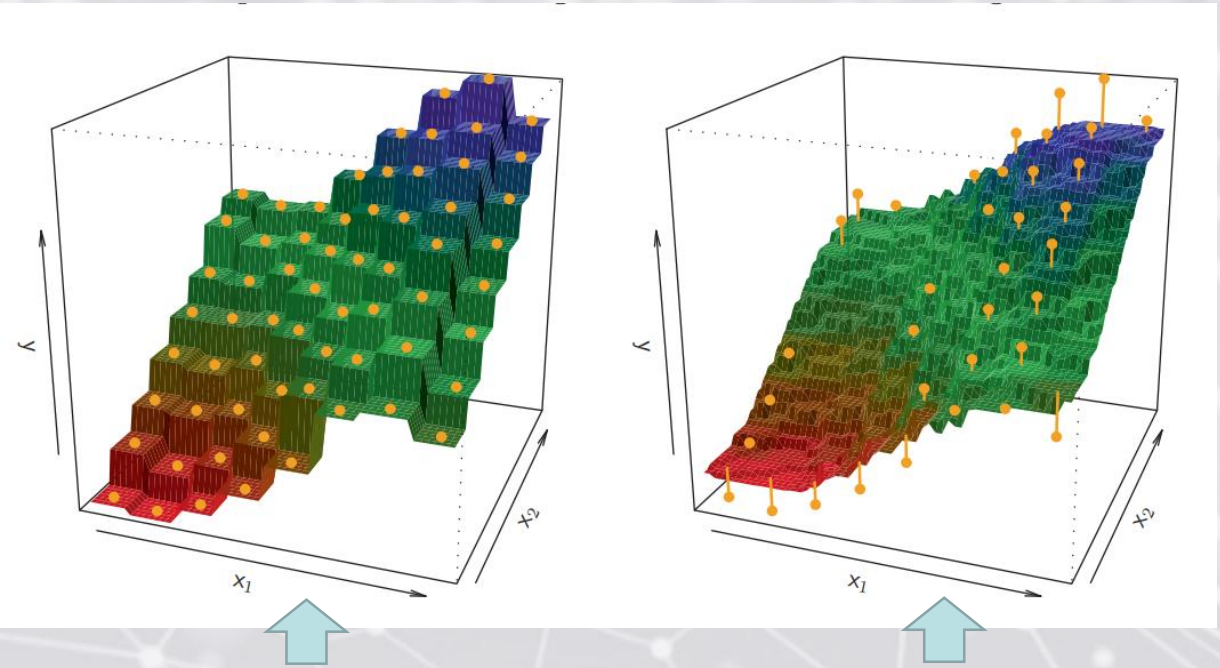
- The k-NN regression method is closely related to the k-NN classifier method – we just need to shift the reference frame from classification to regression appropriately.

- Again, given a fixed value *k* (denoting neighborhood size), and a prediction point x_0, k-NN regression first identifies the K training observations that are closest to x_0, represented by N_0. It then estimates $f(x_0)$ using the average of all the training responses in N_0:

$$f(x_0) = \frac{1}{K} \sum_{x_i \in N_0} y_i$$

k-NN: Regression

- In the figure, the fit with k=1 is shown on the left, while the right-hand panel corresponds with k=9. When k=1 the k-NN model fit perfectly interpolates the training observations, and consequently takes the form of a step-function.

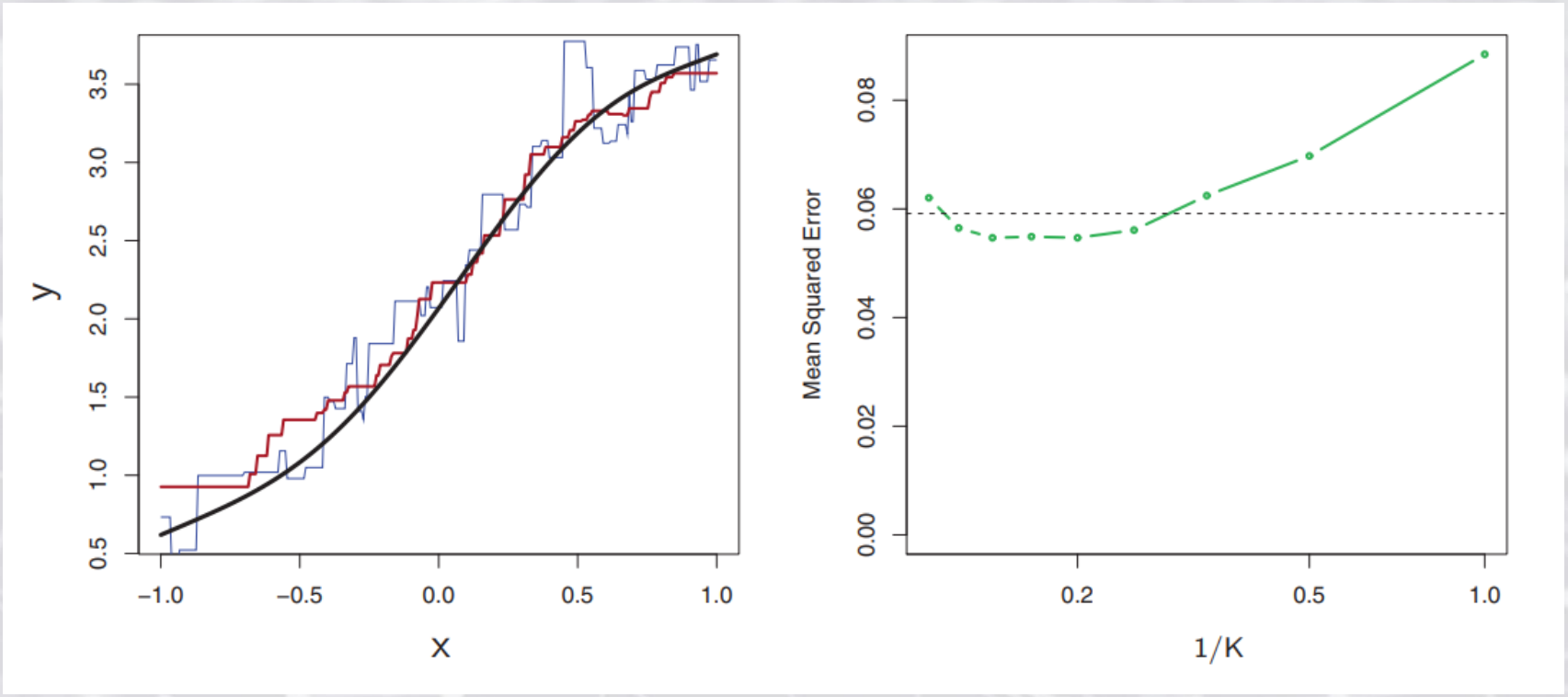


$$f(x_0) = \frac{1}{K} \sum_{x_i \in N_0} y_i$$

k=1; Low bias/high variance (*overfit*) k=9; High bias/low variance (*possible underfit*)

- When k=9, the k=NN fit is still a step function, but averaging over nine observations results in much smaller regions of constant prediction, and consequently renders a smoother fit. As in the case with k-NN for classification, a small value of k provides the most flexible fit (low bias/high variance).

k-NN: Regression



- The plot shows k-NN regression train and test errors as a function of $1/k$; the true function is shown in black; the k-NN fits with k=1 (blue) and k=9 (red) are displayed; on the right the MSE for k-NN (green) is compared with the MSE for OLS (black dotted line).

k-NN: Some Caveats

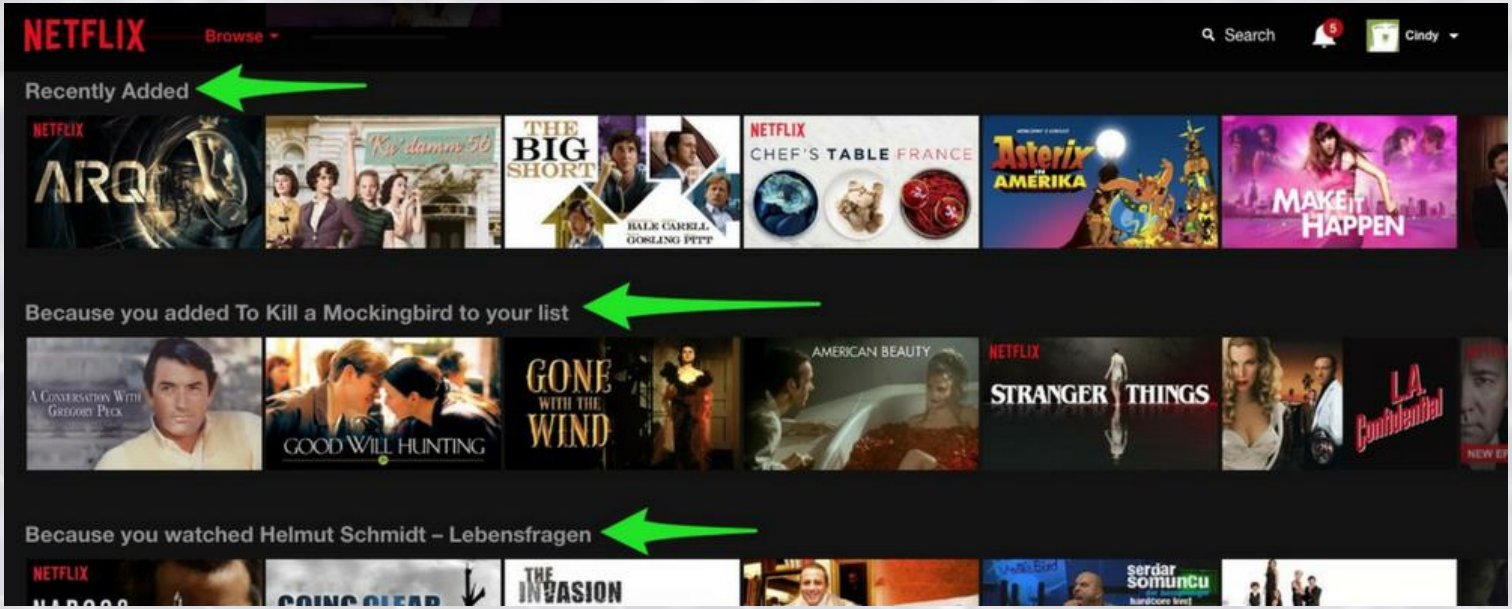
- Note that k-NN often performs poorly for high-dimensional data (due to *the curse of dimensionality*, namely, neighbors are no longer "local"). As a general rule of thumb, parametric methods tend to outperform non-parametric methods when there is a small number of observations per predictor.

- For high-dimensional (even p>10 is large for k-NN), in practice one should perform dimensionality reduction first (e.g. PCA, LDA), and then k-NN.

- In big data applications, k-NN can be used to perform **data reduction tasks** (i.e. *vectorization*); outliers can be identified as data that the k-NN classifier misclassifies, while all other points are identified as *prototype points* or *absorbed points* (i.e. points associated with a particular prototype); cf., *condensed nearest neighbor* (CNN).

- Finding the nearest neighbors in k-NN may be intractable in large dimensions, in which case practitioners customarily use an approximate nearest neighbor search algorithm.

Research applications of k-NN: Recommender Systems



- **Recommender systems** (e.g. Netflix) have become increasingly popular recently across many problem domains in AI/ML, including recommendations for experts (e.g. medicine), financial services, insurance, on-line news feeds, etc.; they comprise a subclass of *information filtering systems* that seek to predict a rating/preference based on historical data.

(*) Notably, Netflix offered a $1,000,000 prize between 2006-2009 for the best recommender system (10% more accurate than the current best), operational on a database of over 100 million movie ratings; the winning group used a vast ensemble of 107 systems.

Research applications of k-NN: Recommender Systems

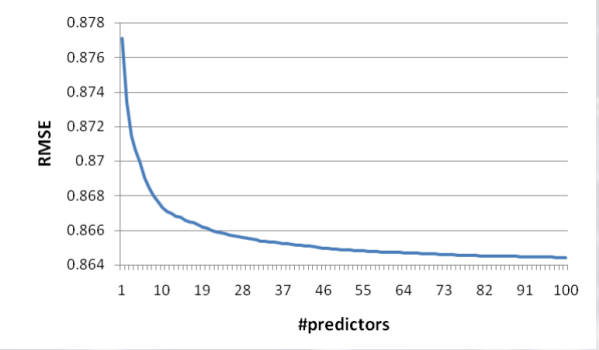
Bell, et al., "Improved Neighborhood-based Collaborative Filtering", KDD, 2007.

• The chief method used by the winners (AT&T Labs) is based on k-NN.

(*) They key innovation introduced by the authors was to enhance k-NN leading to a substantial improvement in prediction accuracy, without a meaningful increase in run-time.

• First, the authors remove certain so-called "global effects" from the data to make the different ratings more comparable, thereby improving interpolation accuracy. Second, they show how to simultaneously derive interpolation weights for all nearest neighbors. This method is very fast in practice, generating a prediction in about 0.2 milliseconds.

• On the Netflix data set this method could process the 2.8 million queries of the in 10 minutes yielding a RMSE of 0.9086; the RMSE was reduced even further using SVD-factorization at the preprocessing stage, our method can produce results with a RMSE of 0.8982.



https://www.netflixprize.com/assets/ProgressPrize2008_BellKor.pdf

Fin