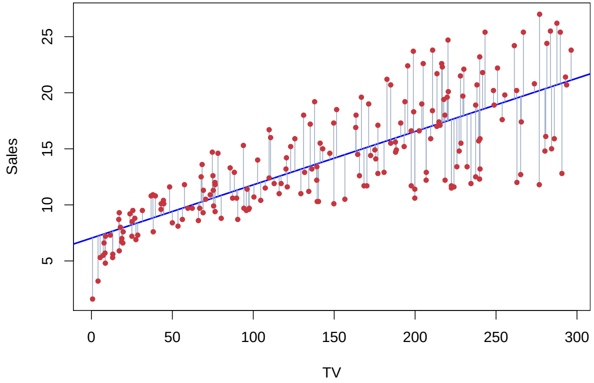
Example: advertising data



The least squares fit for the regression of `sales` onto `TV`.
In this case a linear fit captures the essence of the relationship, although it is somewhat deficient in the left of the plot.

Estimation of the parameters by least squares

- Let $\hat{y}_i = \hat{\beta}_0 + \hat{\beta}_1 x_i$ be the prediction for Y based on the ith value of X. Then $e_i = y_i - \hat{y}_i$ represents the ith *residual*
- We define the *residual sum of squares* (RSS) as

$$\mathrm{RSS} = e_1^2 + e_2^2 + \cdots + e_n^2,$$

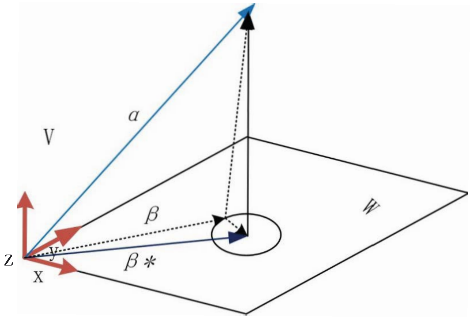
or equivalently as

$$\mathrm{RSS} = (y_1 - \hat{\beta}_0 - \hat{\beta}_1 x_1)^2 + (y_2 - \hat{\beta}_0 - \hat{\beta}_1 x_2)^2 + \ldots + (y_n - \hat{\beta}_0 - \hat{\beta}_1 x_n)^2.$$

- The least squares approach chooses $\hat{\beta}_0$ and $\hat{\beta}_1$ to minimize the RSS. The minimizing values can be shown to be

$$\hat{\beta}_1 = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^n (x_i - \bar{x})^2},$$
$$\hat{\beta}_0 = \bar{y} - \hat{\beta}_1 \bar{x},$$

where $\bar{y} \equiv \frac{1}{n} \sum_{i=1}^n y_i$ and $\bar{x} \equiv \frac{1}{n} \sum_{i=1}^n x_i$ are the sample means.



$$
\begin{aligned}
\nabla_\theta J(\theta) &= \nabla_\theta \frac{1}{2}(X\theta - Y)^T(X\theta - Y) \\
&= \frac{1}{2}\nabla_\theta\,(\theta^T X^T - Y^T)(X\theta - Y) \\
&= \frac{1}{2}\nabla_\theta\,(\theta^T X^T X\theta - \theta^T X^T Y - Y^T X\theta + Y^T Y) \\
&= \frac{1}{2}(2X^T X\theta - 2X^T Y) \\
&= X^T X\theta - X^T Y
\end{aligned}
$$

令 $X^T X\theta = X^T Y$ 因此 $\theta = (X^T X)^{-1} X^T Y$

Results for the advertising data

	Coefficient	Std. Error	t-statistic	p-value
`Intercept`	7.0325	0.4578	15.36	< 0.0001
`TV`	0.0475	0.0027	17.67	< 0.0001

上节课回顾

Assessing the Overall Accuracy of the Model

- We compute the *Residual Standard Error*

$$\mathrm{RSE} = \sqrt{\frac{1}{n-2}\mathrm{RSS}} = \sqrt{\frac{1}{n-2}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2},$$

  where the *residual sum-of-squares* is $RSS = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2$.
- *R-squared* or fraction of variance explained is

$$R^2 = \frac{\mathrm{TSS} - \mathrm{RSS}}{\mathrm{TSS}} = 1 - \frac{\mathrm{RSS}}{\mathrm{TSS}}$$

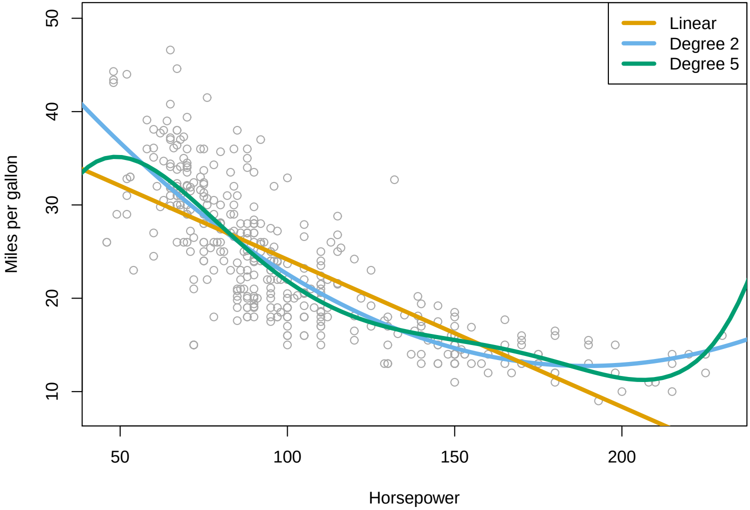
  where $\mathrm{TSS} = \sum_{i=1}^{n}(y_i - \bar{y})^2$ is the *total sum of squares*.
- It can be shown that in this simple linear regression setting that $R^2 = r^2$, where r is the correlation between X and Y:

$$r = \frac{\sum_{i=1}^{n}(x_i - \overline{x})(y_i - \overline{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \overline{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \overline{y})^2}}.$$

上节课回顾

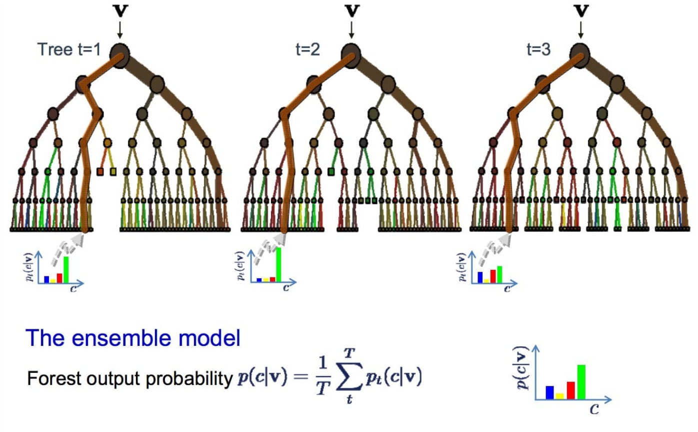
Non-linear effects of predictors

polynomial regression on `Auto` data



Ch 2.2 决策树

例如, 随机森林



The ensemble model

Forest output probability $p(c|\mathbf{v}) = \frac{1}{T}\sum_{t}^{T} p_t(c|\mathbf{v})$

Tree-based Methods

- Here we describe *tree-based* methods for regression and classification.
- These involve *stratifying* or *segmenting* the predictor space into a number of simple regions.
- Since the set of splitting rules used to segment the predictor space can be summarized in a tree, these types of approaches are known as *decision-tree* methods.

Pros and Cons

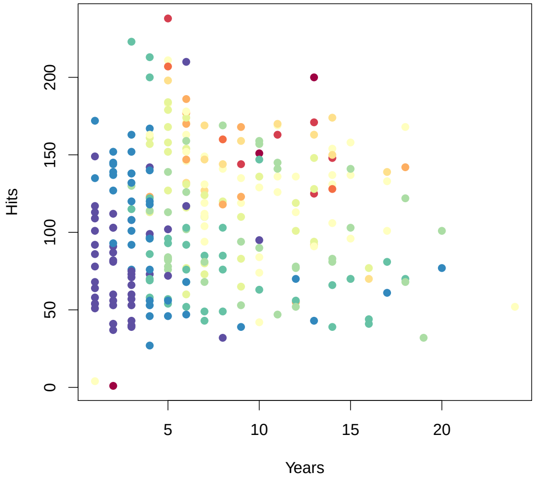
- Tree-based methods are simple and useful for interpretation.
- However they typically are not competitive with the best supervised learning approaches in terms of prediction accuracy.
- Hence we also discuss *bagging*, *random forests*, and *boosting*. These methods grow multiple trees which are then combined to yield a single consensus prediction.
- Combining a large number of trees can often result in dramatic improvements in prediction accuracy, at the expense of some loss interpretation.

The Basics of Decision Trees

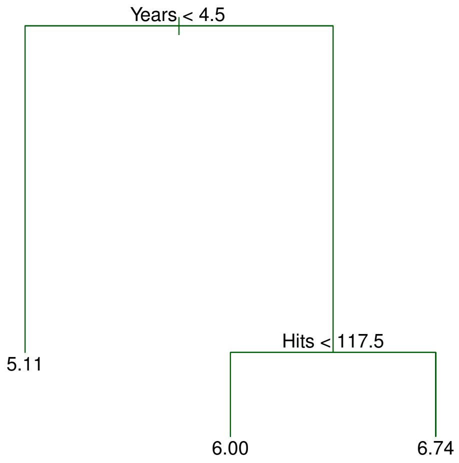
- Decision trees can be applied to both regression and classification problems.
- We first consider regression problems, and then move on to classification.

Baseball salary data: how would you stratify it?

Salary is color-coded from low (blue, green) to high (yellow,red)



Decision tree for these data



Details of previous figure

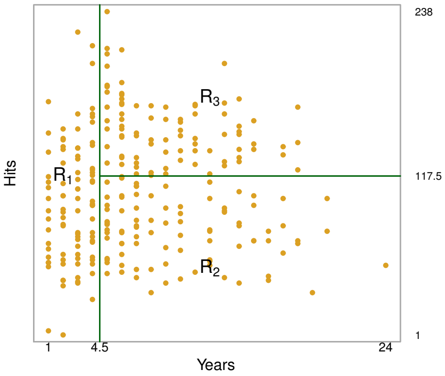
- For the Hitters data, a regression tree for predicting the log salary of a baseball player, based on the number of years that he has played in the major leagues and the number of hits that he made in the previous year.

- At a given internal node, the label (of the form $X_j < t_k$) indicates the left-hand branch emanating from that split, and the right-hand branch corresponds to $X_j \geq t_k$. For instance, the split at the top of the tree results in two large branches. The left-hand branch corresponds to `Years`<4.5, and the right-hand branch corresponds to `Years`>=4.5.

- The tree has two internal nodes and three terminal nodes, or leaves. The number in each leaf is the mean of the response for the observations that fall there.

Results

- Overall, the tree stratifies or segments the players into three regions of predictor space: $R_1 = \{X \mid$ `Years` $< 4.5\}$, $R_2 = \{X \mid$ `Years` $>= 4.5$, `Hits` $< 117.5\}$, and $R_3 = \{X \mid$ `Years` $>= 4.5$, `Hits` $>= 117.5\}$.



Terminology for Trees

- In keeping with the *tree* analogy, the regions R_1, R_2, and R_3 are known as *terminal nodes*
- Decision trees are typically drawn *upside down*, in the sense that the leaves are at the bottom of the tree.
- The points along the tree where the predictor space is split are referred to as *internal nodes*
- In the hitters tree, the two internal nodes are indicated by the text `Years`<4.5 and `Hits`<117.5.

Interpretation of Results

- `Years` is the most important factor in determining `Salary`, and players with less experience earn lower salaries than more experienced players.
- 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`.
- But among players who have been in the major leagues for five or more years, the number of `Hits` made in the previous year does affect `Salary`, and players who made more `Hits` last year tend to have higher salaries.
- Surely an over-simplification, but compared to a regression model, it is easy to display, interpret and explain

Details of the tree-building process

1. We divide the predictor space — that is, the set of possible values for $X_1, X_2, \ldots, X_p$ — into J distinct and non-overlapping regions, $R_1, R_2, \ldots, R_J$.
2. For every observation that falls into the region R_j, we make the same prediction, which is simply the mean of the response values for the training observations in R_j.

More details of the tree-building process

- In theory, the regions could have any shape. However, we choose to divide the predictor space into high-dimensional rectangles, or *boxes*, for simplicity and for ease of interpretation of the resulting predictive model.
- The goal is to find boxes $R_1, \ldots, R_J$ that minimize the RSS, given by

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

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

More details of the tree-building process

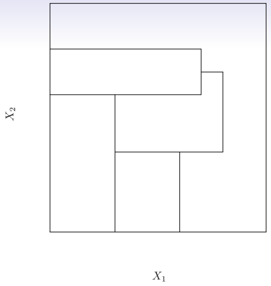
- Unfortunately, it is computationally infeasible to consider every possible partition of the feature space into J boxes.
- For this reason, we take a *top-down*, *greedy* approach that is known as recursive binary splitting.
- The approach is *top-down* because it begins at the top of the tree and then successively splits the predictor space; each split is indicated via two new branches further down on the tree.
- It is *greedy* because at each step of the tree-building process, the *best* split is made at that particular step, rather than looking ahead and picking a split that will lead to a better tree in some future step.

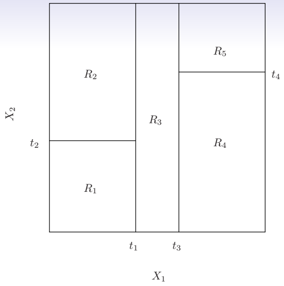
Details— Continued

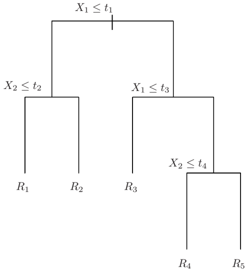
- We first select the predictor X_j and the cutpoint s such that splitting the predictor space into the regions $\{X|X_j < s\}$ and $\{X|X_j \geq s\}$ leads to the greatest possible reduction in RSS.
- Next, we repeat the process, looking for the best predictor and best cutpoint in order to split the 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 split one of the two previously identified regions. We now have three regions.
- Again, we look to split one of these three regions further, so as to minimize the RSS. The process continues until a stopping criterion is reached; for instance, we may continue until no region contains more than five observations.

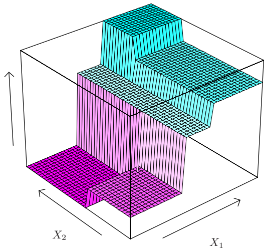
Predictions

- We predict the response for a given test observation using the mean of the training observations in the region to which that test observation belongs.
- A five-region example of this approach is shown in the next slide.


X_2
X_1


R_2
R_3
R_5
t_4
R_4
X_2
t_2
R_1
t_1
t_3
X_1


$X_1 \leq t_1$
$X_2 \leq t_2$
$X_1 \leq t_3$
$X_2 \leq t_4$
R_1
R_2
R_3
R_4
R_5


X_2
X_1

Details of previous figure

Top Left: A partition of two-dimensional feature space that could not result from recursive binary splitting.

Top Right: The output of recursive binary splitting on a two-dimensional example.

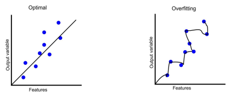
Bottom Left: A tree corresponding to the partition in the top right panel.

Bottom Right: A perspective plot of the prediction surface corresponding to that tree.

Pruning a tree

- The process described above may produce good predictions on the training set, but is likely to *overfit* the data, leading to poor test set performance. *Why?*

Pruning a tree



- The process described above may produce good predictions on the training set, but is likely to *overfit* the data, leading to poor test set performance. *Why?*
- A smaller tree with fewer splits (that is, fewer regions $R_1, \ldots, R_J$) might lead to lower variance and better interpretation at the cost of a little bias.
- One possible alternative to the process described above is to grow the tree only so long as the decrease in the RSS due to each split exceeds some (high) threshold.
- This strategy will result in smaller trees, but is too *short-sighted:* a seemingly worthless split early on in the tree might be followed by a very good split — that is, a split that leads to a large reduction in RSS later on.

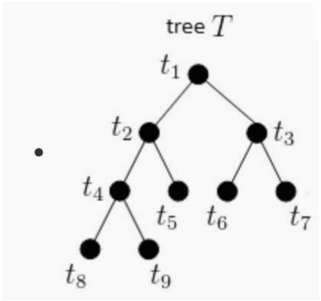
Pruning a tree— continued

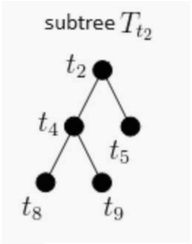
- A better strategy is to grow a very large tree T_0, and then *prune* it back in order to obtain a *subtree*
- *Cost complexity pruning* — also known as *weakest link pruning* — is used to do this
- we consider a sequence of trees indexed by a nonnegative tuning parameter α. For each value of α there corresponds a subtree $T \subset T_0$ such that

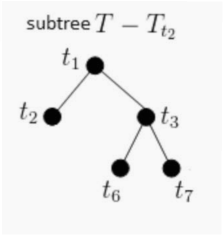
$$\sum_{m=1}^{|T|} \sum_{i:\, x_i \in R_m} (y_i - \hat{y}_{R_m})^2 + \alpha |T|$$

is as small as possible. Here $|T|$ indicates the number of terminal nodes of the tree T, R_m is the rectangle (i.e. the subset of predictor space) corresponding to the mth terminal node, and $\hat{y}_{R_m}$ is the mean of the training observations in R_m.

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







- Initialization:
  - let T^1 be the tree obtained with $\alpha^1 = 0$
  - by minimizing $R(T)$

- Step 1
  - select node $t \in T^1$ that minimizes
    - $g_1(t) = \dfrac{R(t) - R(T_t^1)}{|f(T_t^1)| - 1}$
  - let t_1 be this node
  - let $\alpha^2 = g_1(t_1)$ and $T^2 = T^1 - T_{t_1}^1$

- step i
  - select node $t \in T^i$ that minimizes
    - $g_i(t) = \dfrac{R(t) - R(T_t^i)}{|f(T_t^i)| - 1}$
  - let t_i be this node
  - let $\alpha^{i+1} = g_i(t_i)$ and $T^{i+1} = T^i - T_{t_i}^i$

Output:

- a sequence of trees $T^1 \supseteq T^2 \supseteq \ldots \supseteq T^k \supseteq \ldots \supseteq \{root\}$
- a sequence of parameters $\alpha^1 \leqslant \alpha^2 \leqslant \ldots \leqslant \alpha^k \leqslant \ldots$

Choosing the best subtree

- The tuning parameter α controls a trade-off between the subtree's complexity and its fit to the training data.
- We select an optimal value $\hat{\alpha}$ using cross-validation.
- We then return to the full data set and obtain the subtree corresponding to $\hat{\alpha}$.

Summary: tree algorithm

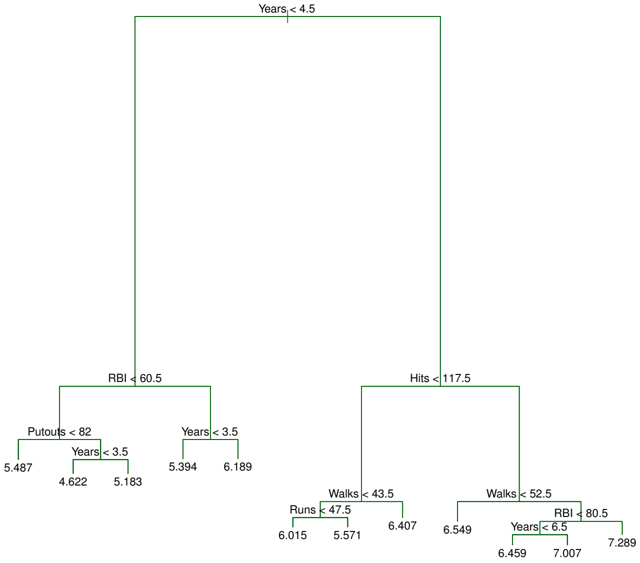
1. Use recursive binary splitting to grow a large tree on the training data, stopping only when each terminal node has fewer than some minimum number of observations.
2. Apply cost complexity pruning to the large tree in order to obtain a sequence of best subtrees, as a function of α.
3. Use K-fold cross-validation to choose α. For each $k = 1, \ldots, K$:
   3.1 Repeat Steps 1 and 2 on the $\frac{K-1}{K}$th fraction of the training data, excluding the kth fold.
   3.2 Evaluate the mean squared prediction error on the data in the left-out kth fold, as a function of α.

   Average the results, and pick α to minimize the average error.
4. Return the subtree from Step 2 that corresponds to the chosen value of α.

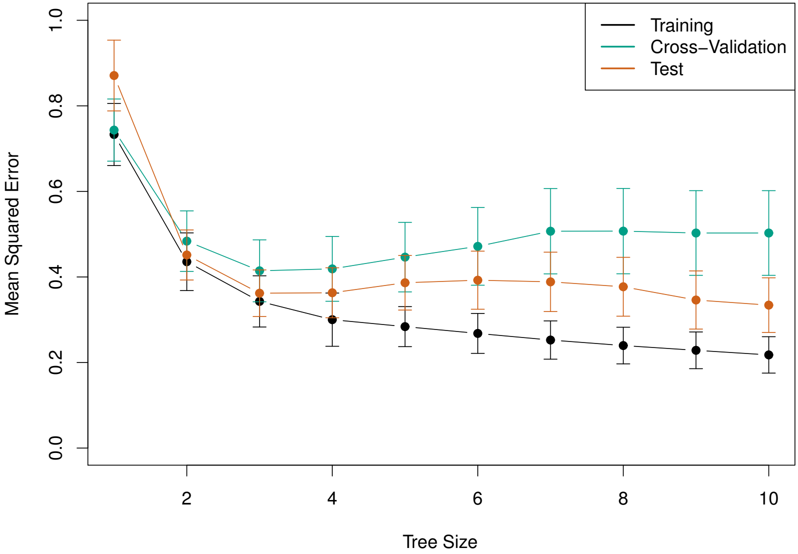
Baseball example continued

- First, we randomly divided the data set in half, yielding 132 observations in the training set and 131 observations in the test set.
- We then built a large regression tree on the training data and varied α in in order to create subtrees with different numbers of terminal nodes.
- Finally, we performed six-fold cross-validation in order to estimate the cross-validated MSE of the trees as a function of α.

Baseball example continued



Baseball example continued



Classification Trees

- Very similar to a regression tree, except that it is used to predict a qualitative response rather than a quantitative one.
- 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 belongs.

Details of classification trees

- Just as in the regression setting, we use recursive binary splitting to grow a classification tree.
- In the classification setting, RSS cannot be used as a criterion for making the binary splits
- A natural alternative to RSS is the *classification error rate*. this is simply the fraction of the training observations in that region that do not belong to the most common class:

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

  Here $\hat{p}_{mk}$ represents the proportion of training observations in the mth region that are from the kth class.
- However classification error is not sufficiently sensitive for tree-growing, and in practice two other measures are preferable.

Gini index and Deviance

- The *Gini index* is defined by

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

  a measure of total variance across the K classes. The Gini index 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 referred to as a measure of node *purity* — a small value indicates that a node contains predominantly observations from a single class.

Gini index and Deviance

- The *Gini index* is defined by

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

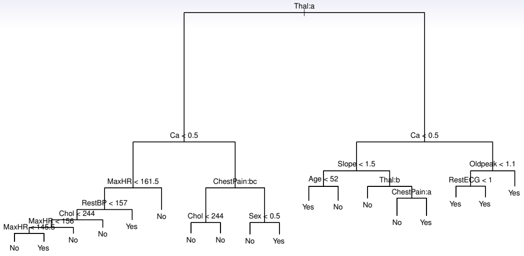
  a measure of total variance across the K classes. The Gini index takes on a small value if all of the $\hat{p}_{mk}$'s are close to zero or one.
- For this reason the Gini index is referred to as a measure of node *purity* — a small value indicates that a node contains predominantly observations from a single class.
- An alternative to the Gini index is *cross-entropy*, given by

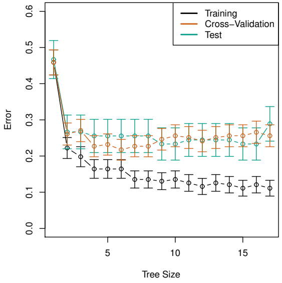
$$D = -\sum_{k=1}^{K} \hat{p}_{mk} \log \hat{p}_{mk}.$$

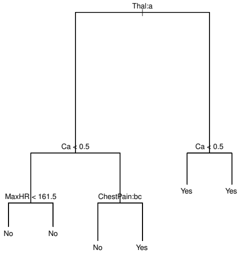
- It turns out that the Gini index and the cross-entropy are very similar numerically.

Example: heart data

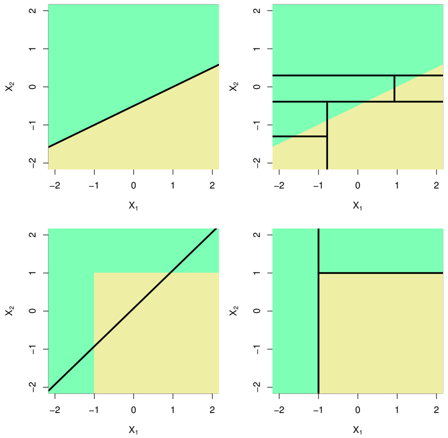
- These data contain a binary outcome `HD` for 303 patients who presented with chest pain.
- An outcome value of `Yes` indicates the presence of heart disease based on an angiographic test, while `No` means no heart disease.
- There are 13 predictors including `Age`, `Sex`, `Chol` (a cholesterol measurement), and other heart and lung function measurements.
- Cross-validation yields a tree with six terminal nodes. See next figure.


Thal:a
Ca < 0.5
Ca < 0.5
MaxHR < 161.5
ChestPain:bc
Slope < 1.5
Oldpeak < 1.1
RestBP < 157
Age < 52
Thal:b
RestECG < 1
Chol < 244
ChestPain:a
MaxHR < 156
MaxHR < 145.5
Chol < 244
Sex < 0.5
No
Yes
No
No
Yes
No
No
No
No
Yes
Yes
No
No
No
Yes
Yes
Yes
Yes


Training
Cross−Validation
Test
Error
Tree Size
0.0
0.1
0.2
0.3
0.4
0.5
0.6
5
10
15


Thal:a
Ca < 0.5
Ca < 0.5
MaxHR < 161.5
ChestPain:bc
Yes
Yes
No
No
No
Yes

Trees Versus Linear Models



Top Row: True linear boundary; Bottom row: true non-linear boundary.

Left column: linear model; Right column: tree-based model

Advantages and Disadvantages of Trees

- ▲ Trees are very easy to explain to people. In fact, they are even easier to explain than linear regression!
- ▲ Some people believe that decision trees more closely mirror human decision-making than do the regression and classification approaches seen in previous chapters.
- ▲ Trees can be displayed graphically, and are easily interpreted even by a non-expert (especially if they are small).
- ▲ Trees can easily handle qualitative predictors without the need to create dummy variables.
- ▼ Unfortunately, trees generally do not have the same level of predictive accuracy as some of the other regression and classification approaches seen in this book.

However, by aggregating many decision trees, the predictive performance of trees can be substantially improved. We introduce these concepts next.

Bagging

- *Bootstrap aggregation*, or *bagging*, is a general-purpose procedure for reducing the variance of a statistical learning method; we introduce it here because it is particularly useful and frequently used in the context of decision trees.

- Recall that given a set of n independent observations $Z_1, \ldots, Z_n$, each with variance σ^2, the variance of the mean $\bar{Z}$ of the observations is given by σ^2/n.

- In other words, *averaging a set of observations reduces variance*. Of course, this is not practical because we generally do not have access to multiple training sets.

Bagging— continued

- Instead, we can bootstrap, by taking repeated samples from the (single) training data set.
- In this approach we generate B different bootstrapped training data sets. We then train our method on the bth bootstrapped training set in order to get $\hat{f}^{*b}(x)$, the prediction at a point x. We then average all the predictions to obtain

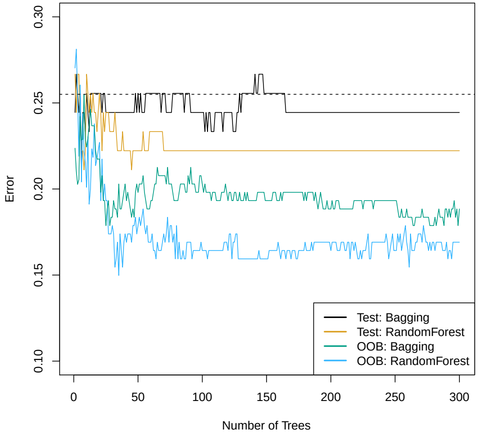
  $$\hat{f}_{\text{bag}}(x) = \frac{1}{B}\sum_{b=1}^{B}\hat{f}^{*b}(x).$$

  This is called *bagging*.

Bagging classification trees

- The above prescription applied to regression trees
- For classification trees: for each test observation, we record the class predicted by each of the B trees, and take a *majority vote*: the overall prediction is the most commonly occurring class among the B predictions.

Bagging the heart data



Details of previous figure

Bagging and random forest results for the Heart data.

- The test error (black and orange) is shown as a function of B, the number of bootstrapped training sets used.
- Random forests were applied with $m = \sqrt{p}$.
- The dashed line indicates the test error resulting from a single classification tree.
- The green and blue traces show the OOB error, which in this case is considerably lower

Out-of-Bag Error Estimation

- It turns out that there is a very straightforward way to estimate the test error of a bagged model.
- Recall that the key to bagging is that trees are repeatedly fit to bootstrapped subsets of the observations. One can show that on average, each bagged tree makes use of around two-thirds of the observations.
- The remaining one-third of the observations not used to fit a given bagged tree are referred to as the *out-of-bag* (OOB) observations.
- We can predict the response for the ith observation using each of the trees in which that observation was OOB. This will yield around $B/3$ predictions for the ith observation, which we average.
- This estimate is essentially the LOO cross-validation error for bagging, if B is large.

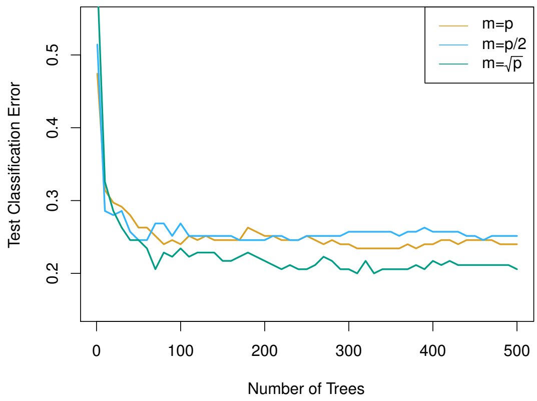
Random Forests

- *Random forests* provide an improvement over bagged trees by way of a small tweak that *decorrelates* the trees. This reduces the variance when we average the trees.
- As in bagging, we build a number of decision trees on bootstrapped training samples.
- But when building these decision trees, each time a split in a tree is considered, *a random selection of m predictors* is chosen as split candidates from the full set of p predictors. The split is allowed to use only one of those m predictors.
- A fresh selection of m predictors is taken at each split, and typically we choose $m \approx \sqrt{p}$ — that is, the number of predictors considered at each split is approximately equal to the square root of the total number of predictors (4 out of the 13 for the Heart data).

Example: gene expression data

- We applied random forests to a high-dimensional biological data set consisting of expression measurements of 4,718 genes measured on tissue samples from 349 patients.
- There are around 20,000 genes in humans, and individual genes have different levels of activity, or expression, in particular cells, tissues, and biological conditions.
- Each of the patient samples has a qualitative label with 15 different levels: either normal or one of 14 different types of cancer.
- We use random forests to predict cancer type based on the 500 genes that have the largest variance in the training set.
- We randomly divided the observations into a training and a test set, and applied random forests to the training set for three different values of the number of splitting variables m.

Results: gene expression data



Details of previous figure

- Results from random forests for the fifteen-class gene expression data set with $p = 500$ predictors.
- The test error is displayed as a function of the number of trees. Each colored line corresponds to a different value of m, the number of predictors available for splitting at each interior tree node.
- Random forests ($m < p$) lead to a slight improvement over bagging ($m = p$). A single classification tree has an error rate of 45.7%.

Boosting

- Like bagging, boosting is a general approach that can be applied to many statistical learning methods for regression or classification. We only discuss boosting for decision trees.
- Recall that bagging involves creating multiple copies of the original training data set using the bootstrap, fitting a separate decision tree to each copy, and then combining all of the trees in order to create a single predictive model.
- Notably, each tree is built on a bootstrap data set, independent of the other trees.
- Boosting works in a similar way, except that the trees are grown *sequentially*: each tree is grown using information from previously grown trees.

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:
   2.1 Fit a tree $\hat{f}^b$ with d splits ($d + 1$ terminal nodes) to the training data (X, r).
   2.2 Update $\hat{f}$ by adding in a shrunken version of the new tree:
   $$\hat{f}(x) \leftarrow \hat{f}(x) + \lambda \hat{f}^b(x).$$
   2.3 Update the residuals,
   $$r_i \leftarrow r_i - \lambda \hat{f}^b(x_i).$$
3. Output the boosted model,
$$\hat{f}(x) = \sum_{b=1}^{B} \lambda \hat{f}^b(x).$$

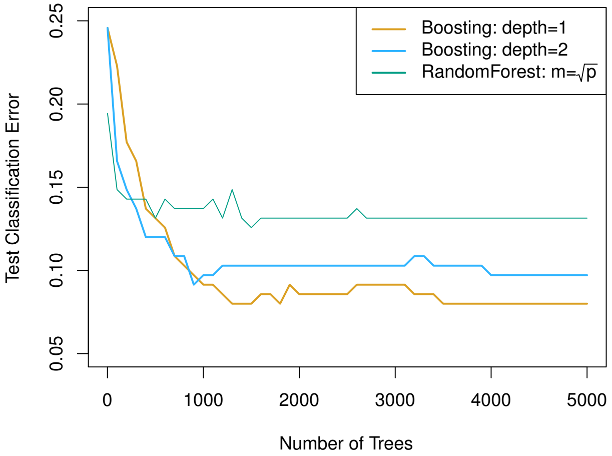
What is the idea behind this procedure?

- Unlike fitting a single large decision tree to the data, which amounts to *fitting the data hard* and potentially overfitting, the boosting approach instead *learns slowly*.
- Given the current model, we fit a decision tree to the residuals from the model. We then add this new decision tree into the fitted function in order to update the residuals.
- Each of these trees can be rather small, with just a few terminal nodes, determined by the parameter d in the algorithm.
- By fitting small trees to the residuals, we slowly improve $\hat{f}$ in areas where it does not perform well. The shrinkage parameter λ slows the process down even further, allowing more and different shaped trees to attack the residuals.

Boosting for classification

- Boosting for classification is similar in spirit to boosting for regression, but is a bit more complex. We will not go into detail here, nor do we in the text book.
- Students can learn about the details in *Elements of Statistical Learning, chapter 10.*
- The R package `gbm` (gradient boosted models) handles a variety of regression and classification problems.

Gene expression data continued



Details of previous figure

- Results from performing boosting and random forests on the fifteen-class gene expression data set in order to predict *cancer* versus *normal*.
- The test error is displayed as a function of the number of trees. For the two boosted models, $\lambda = 0.01$. Depth-1 trees slightly outperform depth-2 trees, and both outperform the random forest, although the standard errors are around 0.02, making none of these differences significant.
- The test error rate for a single tree is 24%.

Tuning parameters for boosting

1. The *number of trees* B. Unlike bagging and random forests, boosting can overfit if B is too large, although this overfitting tends to occur slowly if at all. We use cross-validation to select B.

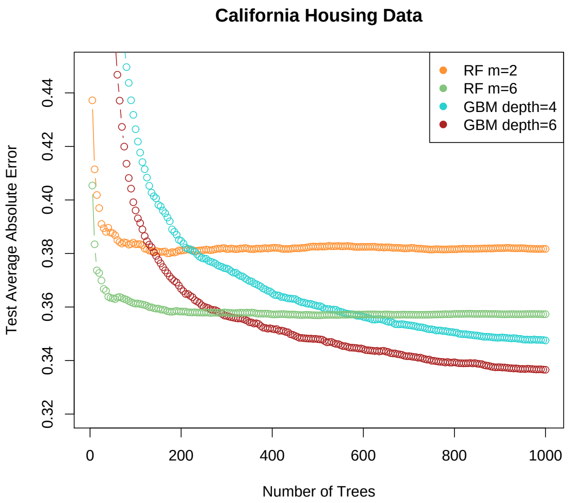
Tuning parameters for boosting

1. The *number of trees* B. Unlike bagging and random forests, boosting can overfit if B is too large, although this overfitting tends to occur slowly if at all. We use cross-validation to select B.
2. The *shrinkage parameter* λ, a small positive number. This controls the rate at which boosting learns. Typical values are 0.01 or 0.001, and the right choice can depend on the problem. Very small λ can require using a very large value of B in order to achieve good performance.

Tuning parameters for boosting

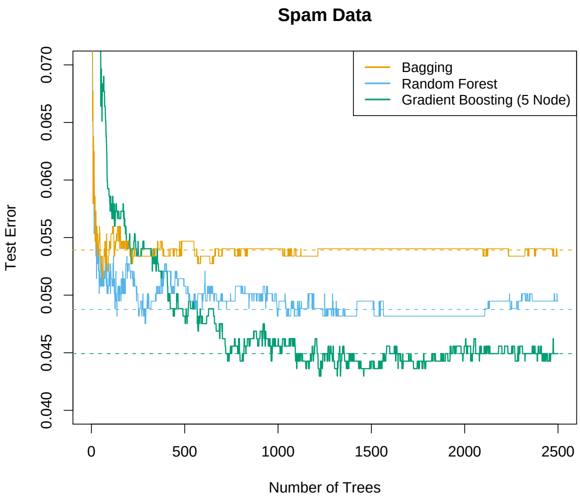
1. The *number of trees* B. Unlike bagging and random forests, boosting can overfit if B is too large, although this overfitting tends to occur slowly if at all. We use cross-validation to select B.
2. The *shrinkage parameter* λ, a small positive number. This controls the rate at which boosting learns. Typical values are 0.01 or 0.001, and the right choice can depend on the problem. Very small λ can require using a very large value of B in order to achieve good performance.
3. The *number of splits* d in each tree, which controls the complexity of the boosted ensemble. Often $d = 1$ works well, in which case each tree is a *stump*, consisting of a single split and resulting in an additive model. More generally d is the *interaction depth*, and controls the interaction order of the boosted model, since d splits can involve at most d variables.

Another regression example



from *Elements of Statistical Learning, chapter 15.*

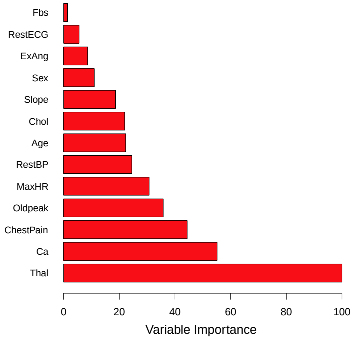
Another classification example



from *Elements of Statistical Learning, chapter 15.*

Variable importance measure

- For bagged/RF regression trees, we record the total amount that the RSS is decreased due to splits over a given predictor, averaged over all B trees. A large value indicates an important predictor.
- Similarly, for bagged/RF classification trees, we add up the total amount that the Gini index is decreased by splits over a given predictor, averaged over all B trees.



Variable importance plot for the `Heart` data

Summary

- Decision trees are simple and interpretable models for regression and classification
- However they are often not competitive with other methods in terms of prediction accuracy
- Bagging, random forests and boosting are good methods for improving the prediction accuracy of trees. They work by growing many trees on the training data and then combining the predictions of the resulting ensemble of trees.
- The latter two methods— random forests and boosting— are among the state-of-the-art methods for supervised learning. However their results can be difficult to interpret.

Bayesian Additive Regression Trees

Bayesian Additive Regression Trees

- We discuss *Bayesian additive regression trees* (BART), an ensemble method that uses decision trees as its building blocks.

Bayesian Additive Regression Trees

- We discuss *Bayesian additive regression trees* (BART), an ensemble method that uses decision trees as its building blocks.
- Recall that *bagging* and *random forests* make predictions from an average of regression trees, each of which is built using a random sample of data and/or predictors. Each tree is built separately from the others.

Bayesian Additive Regression Trees

- We discuss *Bayesian additive regression trees* (BART), an ensemble method that uses decision trees as its building blocks.
- Recall that *bagging* and *random forests* make predictions from an average of regression trees, each of which is built using a random sample of data and/or predictors. Each tree is built separately from the others.
- By contrast, *boosting* uses a weighted sum of trees, each of which is constructed by fitting a tree to the residual of the current fit. Thus, each new tree attempts to capture signal that is not yet accounted for by the current set of trees.

Bayesian Additive Regression Trees — Details

Bayesian Additive Regression Trees — Details

- BART is related to both random forests and boosting: each tree is constructed in a random manner as in bagging and random forests, and each tree tries to capture signal not yet accounted for by the current model, as in boosting.

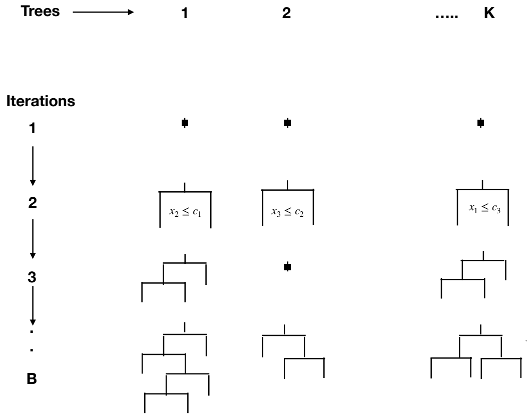
Bayesian Additive Regression Trees — Details

- BART is related to both random forests and boosting: each tree is constructed in a random manner as in bagging and random forests, and each tree tries to capture signal not yet accounted for by the current model, as in boosting.
- The main novelty in BART is the way in which new trees are generated.

Bayesian Additive Regression Trees — Details

- BART is related to both random forests and boosting: each tree is constructed in a random manner as in bagging and random forests, and each tree tries to capture signal not yet accounted for by the current model, as in boosting.
- The main novelty in BART is the way in which new trees are generated.
- BART can be applied to *regression*, *classification* and other problems; we will focus here just on regression.

BART algorithm — the idea



Perturbations based on partial residuals

Average these to get final predictions

Bayesian Additive Regression Trees — Some Notation

Bayesian Additive Regression Trees — Some Notation

- We let K denote the number of regression trees, and B the number of iterations for which the BART algorithm will be run.

Bayesian Additive Regression Trees — Some Notation

- We let K denote the number of regression trees, and B the number of iterations for which the BART algorithm will be run.
- The notation $\hat{f}_k^b(x)$ represents the prediction at x for the kth regression tree used in the bth iteration. At the end of each iteration, the K trees from that iteration will be summed, i.e. $\hat{f}^b(x) = \sum_{k=1}^{K} \hat{f}_k^b(x)$ for $b = 1, \ldots, B$.

BART iterations

BART iterations

- In the *first iteration* of the BART algorithm, all trees are initialized to have a single root node, with $\hat{f}_k^1(x) = \frac{1}{nK} \sum_{i=1}^n y_i$, the mean of the response values divided by the total number of trees. Thus,

$$\hat{f}^1(x) = \sum_{k=1}^K \hat{f}_k^1(x) = \frac{1}{n} \sum_{i=1}^n y_i$$

BART iterations

- In the *first iteration* of the BART algorithm, all trees are initialized to have a single root node, with $\hat{f}_k^1(x) = \frac{1}{nK}\sum_{i=1}^n y_i$, the mean of the response values divided by the total number of trees. Thus,

$$\hat{f}^1(x) = \sum_{k=1}^K \hat{f}_k^1(x) = \frac{1}{n}\sum_{i=1}^n y_i$$

- *In subsequent iterations*, BART updates each of the K trees, one at a time. In the bth iteration, to update the kth tree, we subtract from each response value the predictions from all but the kth tree, in order to obtain a *partial residual*

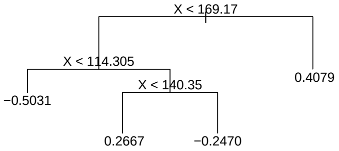
$$r_i = y_i - \sum_{k'<k} \hat{f}_{k'}^b(x_i) - \sum_{k'>k} \hat{f}_{k'}^{b-1}(x_i),\ i = 1, \ldots, n$$

New trees are chosen by perturbations

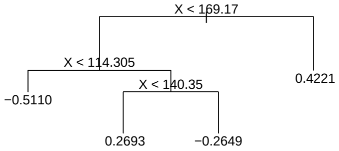
- Rather than fitting a fresh tree to this partial residual, BART randomly chooses a perturbation to the tree from the previous iteration ($\hat{f}_k^{b-1}$) from a set of possible perturbations, favoring ones that improve the fit to the partial residual.
- There are two components to this perturbation:
    1. We may change the structure of the tree by adding or pruning branches.
    2. We may change the prediction in each terminal node of the tree.

Examples of possible perturbations to a tree

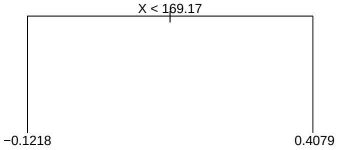
(a): $\hat{f}_k^{b-1}(X)$



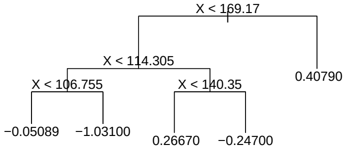
(b): Possibility #1 for $\hat{f}_k^{b}(X)$



(c): Possibility #2 for $\hat{f}_k^{b}(X)$



(d): Possibility #3 for $\hat{f}_k^{b}(X)$



What does BART Deliver?

What does BART Deliver?

- The output of BART is a collection of prediction models,

$$\hat{f}^b(x) = \sum_{k=1}^{K} \hat{f}_k^b(x), \text{ for } b = 1, 2, \ldots, B.$$

What does BART Deliver?

- The output of BART is a collection of prediction models,

$$\hat{f}^b(x) = \sum_{k=1}^{K} \hat{f}_k^b(x), \text{ for } b = 1, 2, \ldots, B.$$

- To obtain a single prediction, we simply take the average after some L *burn-in iterations*, $\hat{f}(x) = \frac{1}{B-L} \sum_{b=L+1}^{B} \hat{f}^b(x)$.

What does BART Deliver?

- The output of BART is a collection of prediction models,

$$\hat{f}^b(x) = \sum_{k=1}^{K} \hat{f}_k^b(x), \text{ for } b = 1, 2, \ldots, B.$$

- To obtain a single prediction, we simply take the average after some L *burn-in iterations*, $\hat{f}(x) = \frac{1}{B-L} \sum_{b=L+1}^{B} \hat{f}^b(x)$.
- The perturbation-style moves guard against overfitting since they limit how *hard* we fit the data in each iteration.

What does BART Deliver?

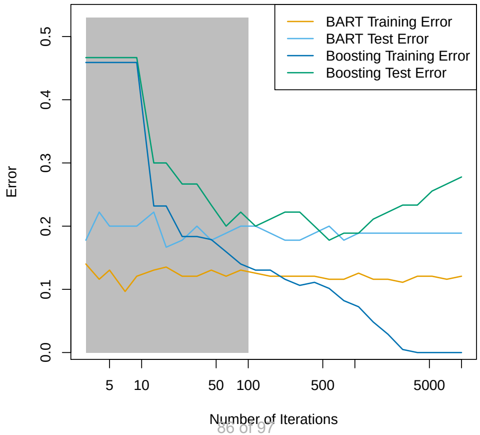
- The output of BART is a collection of prediction models,

$$\hat{f}^b(x) = \sum_{k=1}^{K} \hat{f}_k^b(x), \text{ for } b = 1, 2, \ldots, B.$$

- To obtain a single prediction, we simply take the average after some L *burn-in iterations*, $\hat{f}(x) = \frac{1}{B-L} \sum_{b=L+1}^{B} \hat{f}^b(x)$.
- The perturbation-style moves guard against overfitting since they limit how *hard* we fit the data in each iteration.
- We can also compute quantities other than the average: for instance, the *percentiles* of $f^{L+1}(x), \cdots f^B(x)$ provide a measure of uncertainty of the final prediction.

BART applied to the Heart data

$K = 200$ trees; the number of iterations is increased to $10,000$.
During the initial iterations (in gray), the test and training errors jump around a bit. After this initial burn-in period, the error rates settle down.
The tree perturbation process largely avoids overfitting.



BART is a Bayesian Method

BART is a Bayesian Method

- It turns out that the BART method can be viewed as a *Bayesian* approach to fitting an ensemble of trees: each time we randomly perturb a tree in order to fit the residuals, we are in fact drawing a new tree from a *posterior* distribution.
- Furthermore, the BART algorithm can be viewed as a *Markov chain Monte Carlo* procedure for fitting the BART model.

BART is a Bayesian Method

- It turns out that the BART method can be viewed as a *Bayesian* approach to fitting an ensemble of trees: each time we randomly perturb a tree in order to fit the residuals, we are in fact drawing a new tree from a *posterior* distribution.
- Furthermore, the BART algorithm can be viewed as a *Markov chain Monte Carlo* procedure for fitting the BART model.
- We typically choose large values for B and K, and a moderate value for L: for instance, $K = 200$, $B = 1{,}000$, and $L = 100$ are reasonable choices. BART has been shown to have impressive out-of-box performance — that is, it performs well with minimal tuning.

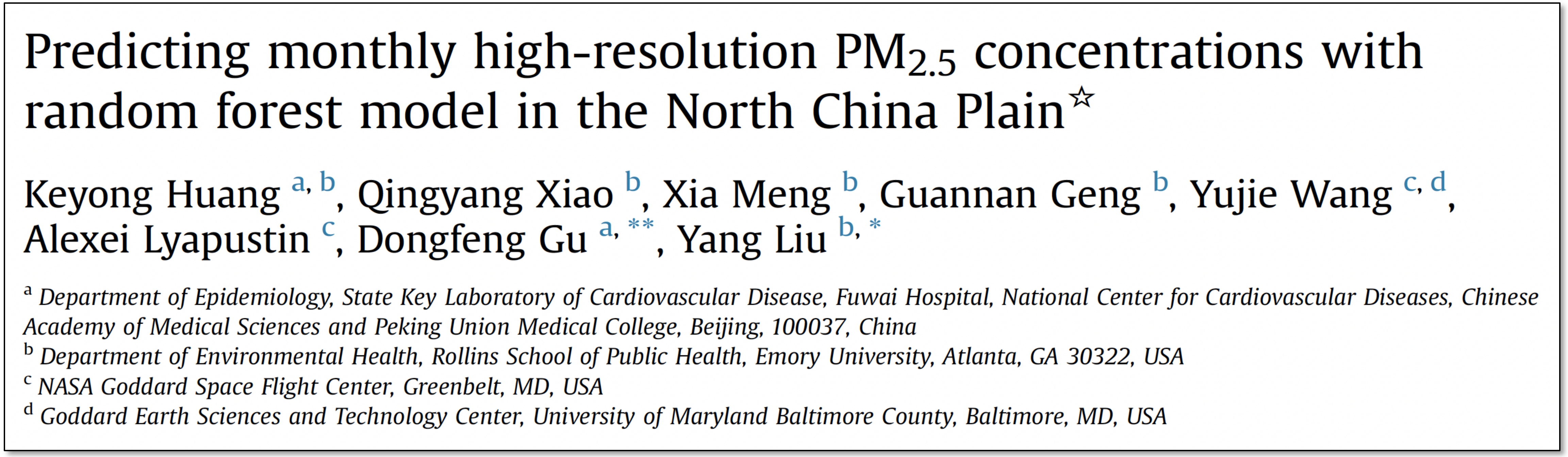
Summary of Tree Ensemble Methods

- In *bagging*, the trees are grown independently on random samples of the observations. Consequently, the trees tend to be quite similar to each other. Thus, bagging can get caught in local optima and can fail to thoroughly explore the model space.

- In *random forests*, the trees are once again grown independently on random samples of the observations. However, each split on each tree is performed using a random subset of the features, thereby decorrelating the trees, and leading to a more thorough exploration of model space relative to bagging.

- In *boosting*, we only use the original data, and do not draw any random samples. The trees are grown successively, using a "slow" learning approach: each new tree is fit to the signal that is left over from the earlier trees, and shrunken down before it is used.

- In *BART*, we once again only make use of the original data, and we grow the trees successively. However, each tree is perturbed in order to avoid local minima and achieve a more thorough exploration of the model space.



Predicting monthly high-resolution $PM_{2.5}$ concentrations with random forest model in the North China Plain☆

Keyong Huang [a, b], Qingyang Xiao [b], Xia Meng [b], Guannan Geng [b], Yujie Wang [c, d], Alexei Lyapustin [c], Dongfeng Gu [a, **], Yang Liu [b, *]

[a] *Department of Epidemiology, State Key Laboratory of Cardiovascular Disease, Fuwai Hospital, National Center for Cardiovascular Diseases, Chinese Academy of Medical Sciences and Peking Union Medical College, Beijing, 100037, China*
[b] *Department of Environmental Health, Rollins School of Public Health, Emory University, Atlanta, GA 30322, USA*
[c] *NASA Goddard Space Flight Center, Greenbelt, MD, USA*
[d] *Goddard Earth Sciences and Technology Center, University of Maryland Baltimore County, Baltimore, MD, USA*

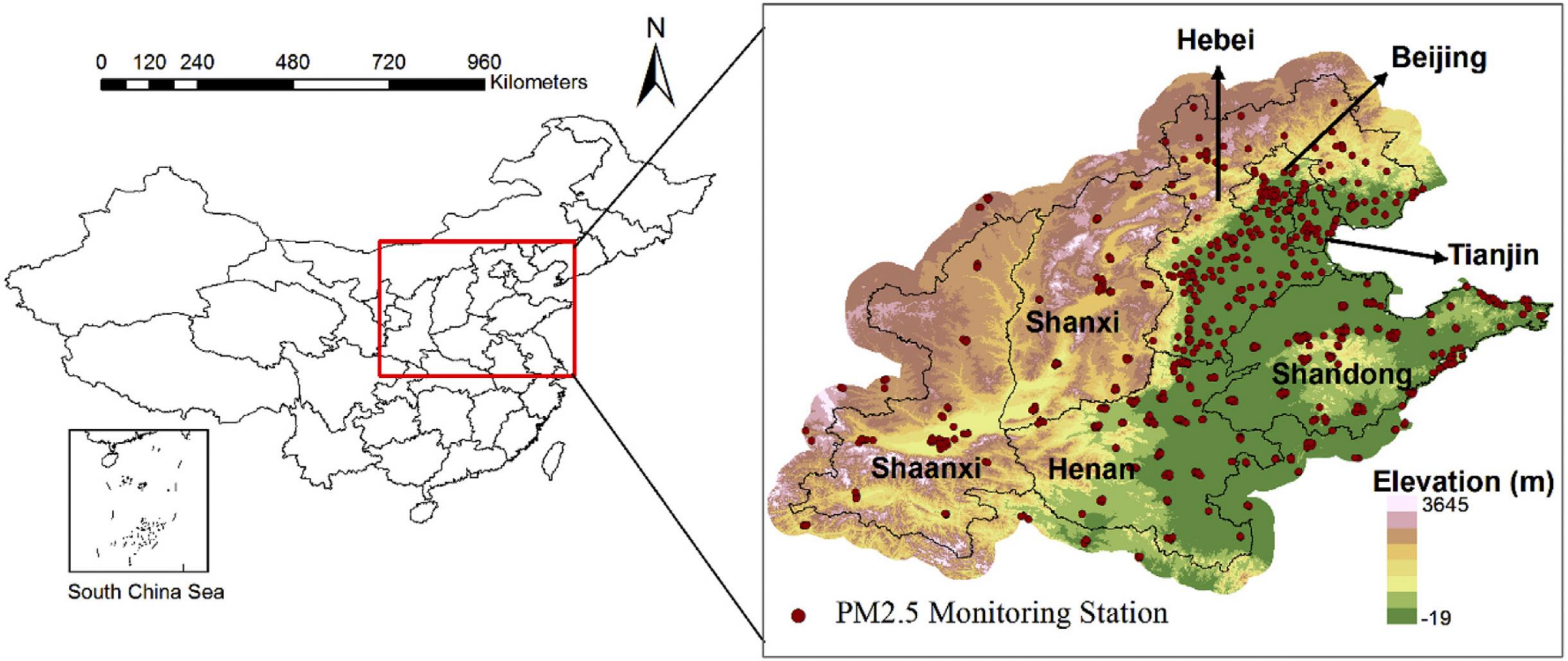


Fig. 1. Study area with a 50-km buffer, showing locations of ground $PM_{2.5}$ monitoring stations.

2 of 7

Random Forest Model

Variable
MAIAC AOD
MERRA-2 $PM_{2.5}$ Species
Surface pressure
2m air temperature
2m dew point temperature
2m relative humidity
Relative humidity under 875 hpa
Specific humidity under 875 hpa
Air temperature under 875 hpa
10m wind speed
10m uwind
10m vwind
U wind under 875 hpa
V wind under 875 hpa
Wind under 875 hpa
Surface net solar radiation
Surface net thermal radiation
Total precipitation
Evaporation
Total cloud cover
Low cloud cover
Surface albedo
Sunshine duration
Boundary layer height
Normalized difference vegetation index
Elevation
Population density
Road length
Distance to nearest road
Fraction of urban cover
Fraction of forest cover
Fraction of greenspace
Visibility
Fire spots

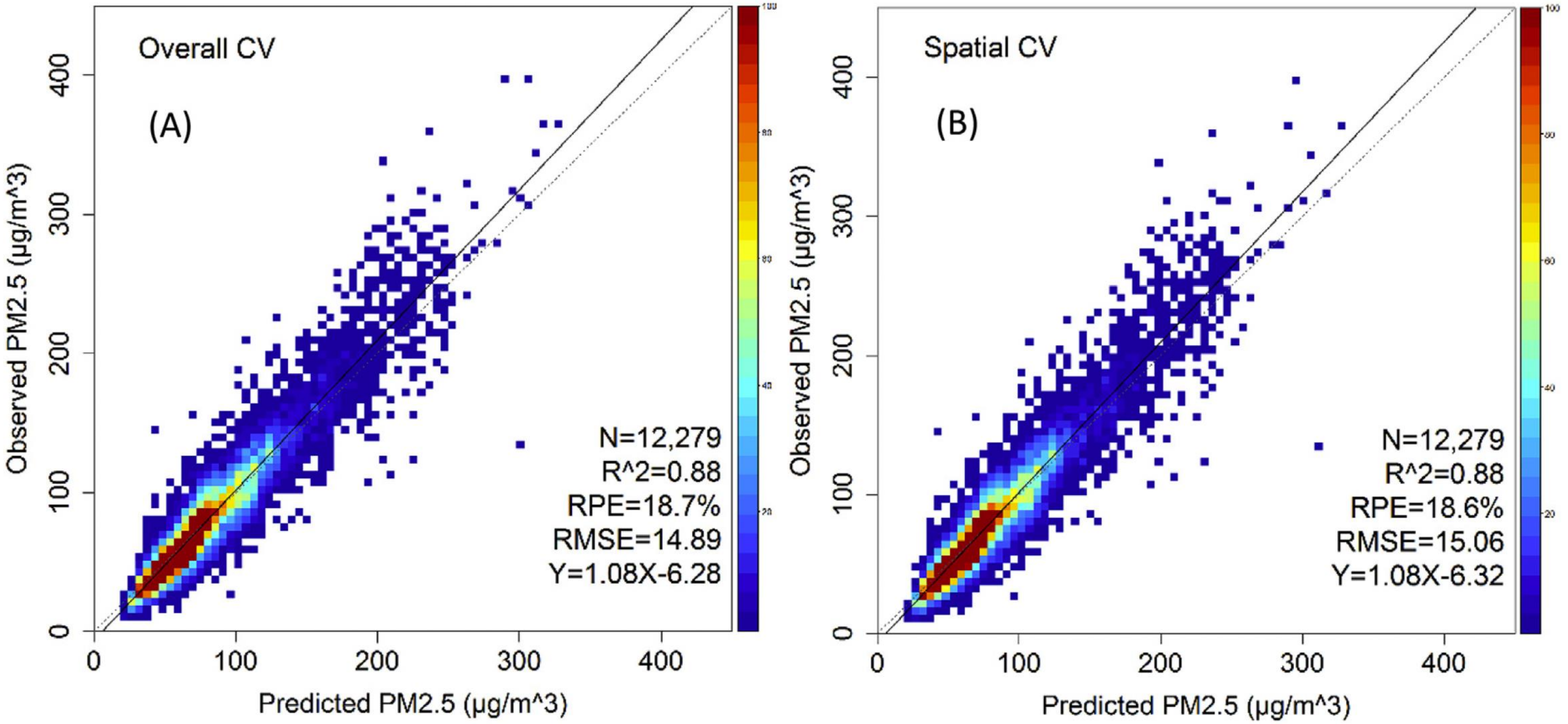


Fig. 2. 10-fold cross validation. (A) Overall cross validation; (B) Spatial cross validation.

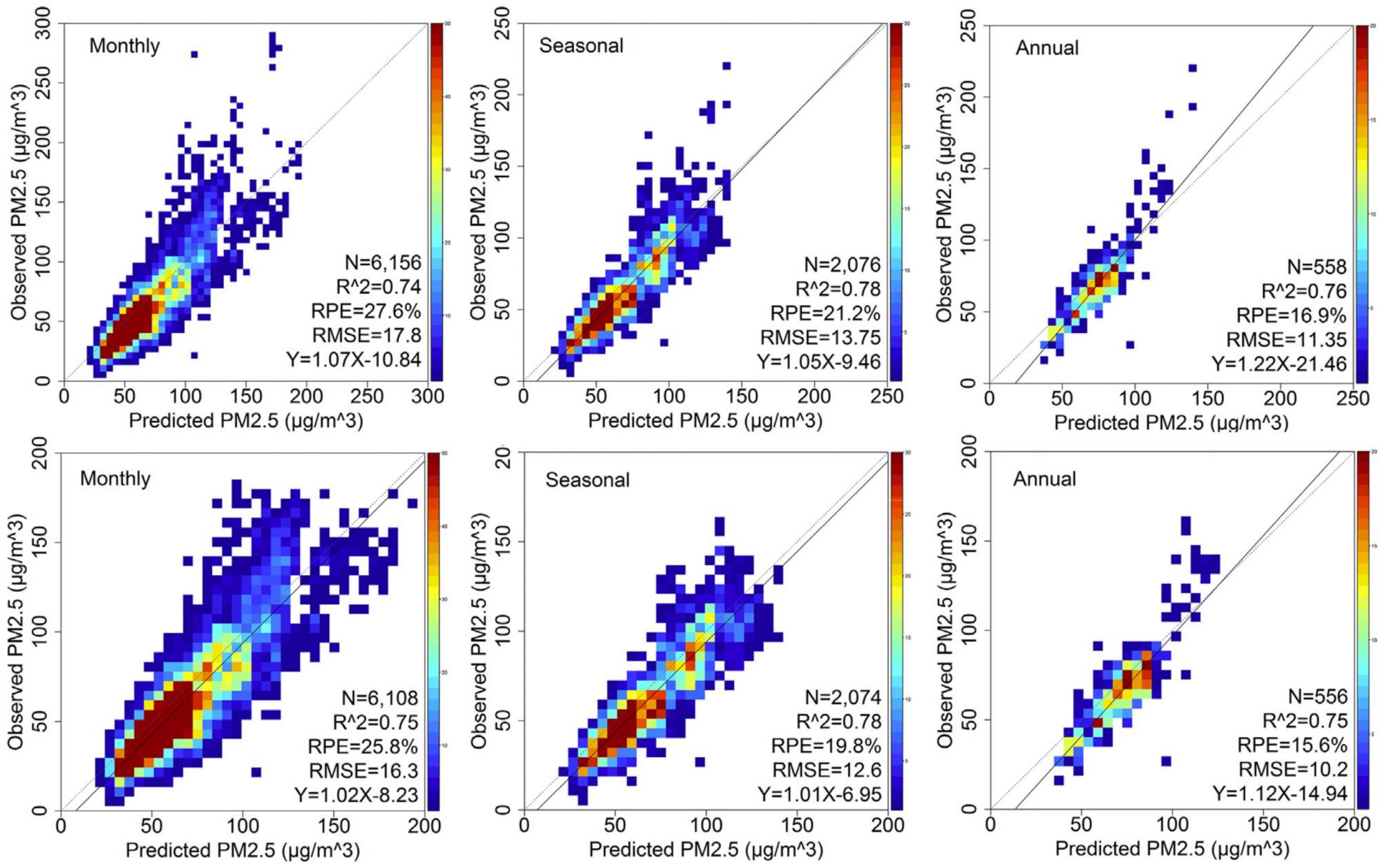


Fig. 3. Results of predicting $PM_{2.5}$ concentrations in 2016 at monthly, seasonal and annual level with models fitted from data of year 2013–2015. Upper panel: for the entire dataset; Lower panel: for dataset removing monthly $PM_{2.5}$ concentrations greater than 180 μg/m3.

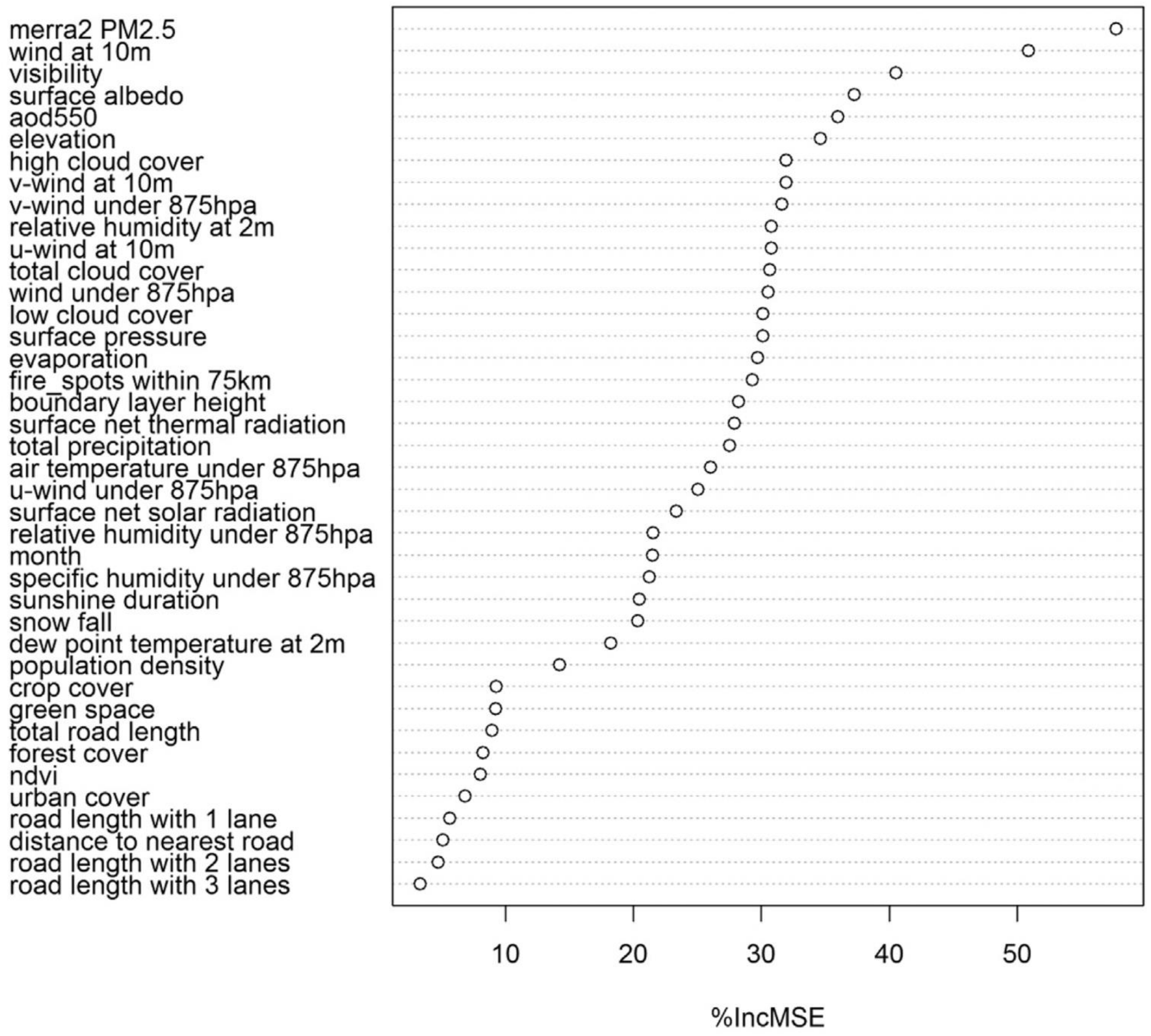


Fig. 4. Variable importance plot for the random forest model predicting the monthly $PM_{2.5}$ concentrations in North China.

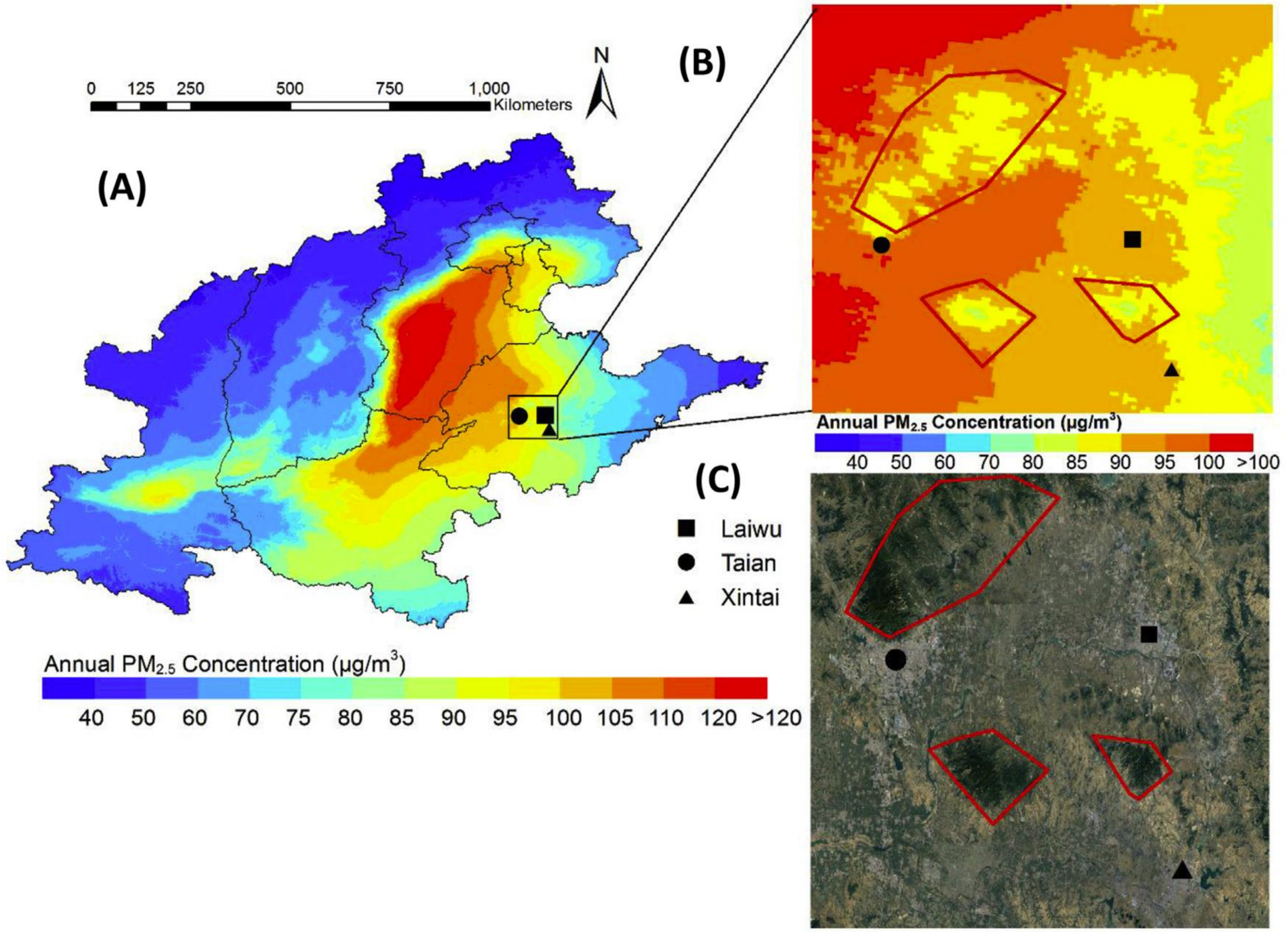


Fig. 5. $PM_{2.5}$ gradients under high resolution. A: annual $PM_{2.5}$ predictions in 2013; B: zoom in map of annual $PM_{2.5}$ predictions in Tai'an, Laiwu and Xintai City; C: satellite photo of Tai'an, Laiwu and Xintai City. Map data: Google Earth. Red polygon represented the forest cover. (For interpretation of the references to colour in this figure legend, the reader is referred to the Web version of this article.)