Tree-Based Methods for Regression & Classification

CS 530
Chapman

Spring 2021

1

Take 🏠 message for rest of course: Tree-based methods

- We start with a simple, though highly non-linear, learning rule—decision-trees—that often does not predict very well, and discuss general-purpose methods to improve it.
- We encounter another example of bootstrapping (bagging) and see the importance of decorrelation in bootstrapping (random forests)
- We introduced the boosting technique: combining weak learners that work serially on each other's residuals to form a strong learning rule.

2

TABLE OF CONTENTS

3

Regression trees

DECISION TREES

4

Intro to Tree-Based Methods

Tree-based methods involve segmenting the feature space of our training data (a.k.a. the *predictor space*) into rectangular regions, or "boxes". This set of splitting decisions can be represented as a decision tree.

We can use decision trees for

- continuous response variables (*regression trees*).
- categorical response variables (*classification trees*).

5

Intro to Tree-Based Methods

To illustrate this, imagine we have two explanatory variables x_1 and x_2.

- Find a split along either x_1 or x_2 that provides the greatest "distinction" between the sets on either side of the split. Call that split t_1.

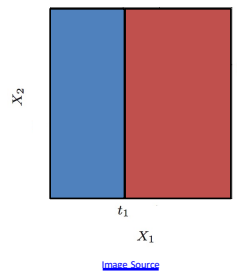


Image Source

6

Intro to Tree-Based Methods

To illustrate this, imagine we have two explanatory variables x_1 and x_2.

- Find a split along either x_1 or x_2 that provides the greatest "distinction" between the sets on either side of the split. Call that split t_1.
- Consider splits along x_1 and x_2 for the two regions that were created by the split along t_1. Pick the split that best divides the data in one of the regions. Call that split t_2.

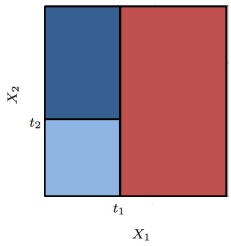


Image Source

7

Intro to Tree-Based Methods

To illustrate this, imagine we have two explanatory variables x_1 and x_2.

- Find a split along either x_1 or x_2 that provides the greatest "distinction" between the sets on either side of the split. Call that split t_1.
- Consider splits along x_1 and x_2 for the two regions that were created by the split along t_1. Pick the split that best divides the data in one of the regions. Call that split t_2.
- Repeat as many times as desired or as required.

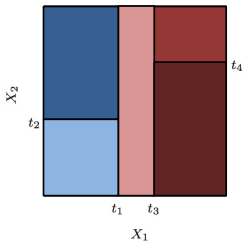


Image Source

8

Intro to Tree-Based Methods

This set of splits can be represented with a decision tree, as below:

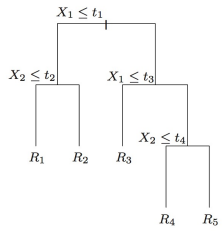


Image Source

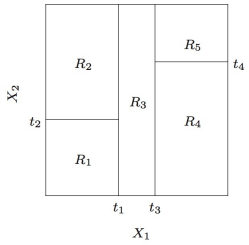


Image Source

9

Intro to Tree-Based Methods

The resulting 5 regions $R_1, \ldots, R_5$ from the 4 splits are each associated with a prediction. Their grouping should (generally) be such that neighboring regions are more similar to each other than regions farther away.

This process is called *recursive binary splitting*, and the nodes at the bottom of the tree (the leaves, labeled $R_1, \ldots, R_5$) are the terminal nodes of the tree. Note, n splits result in $n + 1$ terminal nodes.

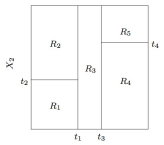
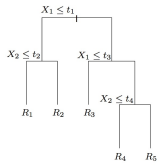


Image Source



10

Regression Trees

Consider the following toy example to clarify how regression trees work.

We want to model

$y \leftarrow$ income

based on

$x_1 \leftarrow$ years of school (after high school)

$x_2 \leftarrow$ years of relevant work experience

A linear regression model might be

$$y = b_0 + b_1 x_1 + b_2 x_2.$$

11

Regression Trees

Consider the following toy example to clarify how regression trees work.

We want to model

$y \leftarrow$ income

based on

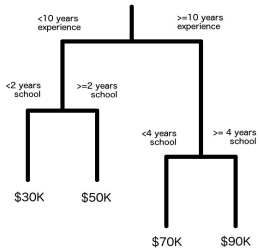
$x_1 \leftarrow$ years of school (after high school)

$x_2 \leftarrow$ years of relevant work experience

A linear regression model might be

$$y = b_0 + b_1 x_1 + b_2 x_2.$$

But a tree-based model might be the one on the right.

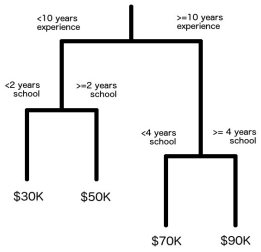


12

Regression Trees

The regression tree model:

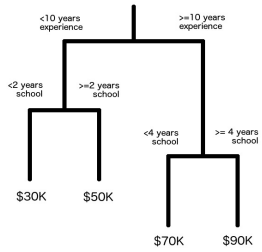
1. Looks at both explanatory variables (years of school & years of experience) to find a split that creates the clearest distinction in the predictor variable (income).
2. Splits first on whether a person has had at least 10 years work experience.
3. Splits next on whether a person has had 2 years of college (for those with <10 years experience)
4. Finally splits on whether a person has had >=4 years of college (for those >=10 years experience)



13

Regression Trees

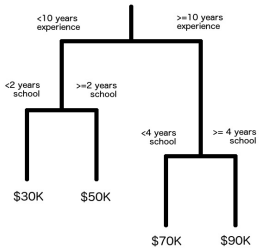
The terminal nodes with income values ($30K, $50K, etc.) give us the *average* income for the data points in that region.



14

Regression Trees

How do regression trees decide where to split the data? Where should the forks in the tree be?



15

Regression Trees

For linear regression, the hyperplane that best fit the data was found through minimizing the *residual sum of squares* (RSS)—the distance between the model and the data:

$$\text{RSS} = \sum (y_i - \hat{y}_i)^2$$

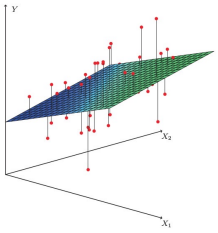


Image Source

16

Regression Trees

With regression trees, instead of trying to fit the data to a single hyperplane, the model is a step function with a different predicted y value associated to each region in feature space.

In the diagram on the right, the height of each plateau equals the y value predicted for that associated region of feature space.

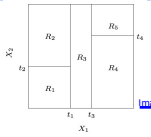


Image Source

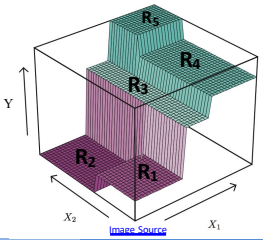


Image Source

17

Regression Trees

Similarly to linear regression, the goal is to minimize the RSS:

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

where $\hat{y}_{R_j}$ is the prediction for region R_j, J is the number of terminal nodes.

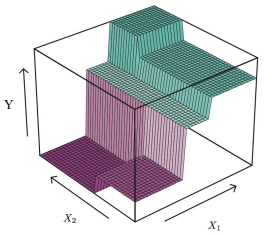


Image Source

18

Regression Trees

Testing the RSS for every possible partition of the feature space is typically computationally intractable. How do we decide where to split then?

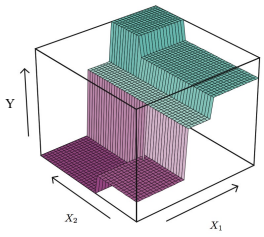


Image Source

19

Regression Trees

- We use a top-down approach that starts with the entire feature space and successively splits portions into two.
- We use a greedy approach that makes the best split for any predictor at that step, rather than looking ahead to see if a less-than-optimal split now might yield better predictive power later.
- Each split tries to minimize the RSS as much as possible at that step.

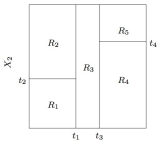


Image Source

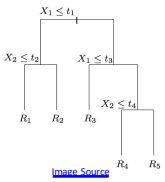
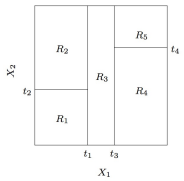


Image Source

20

Algorithm for building regression trees

1. We divide the features space into J non-overlapping regions $R_1, \cdots, R_J$.
2. For each observation in R_j we predict the same value, $\hat{y}_{R_j}$, the mean response value of the training observations in R_j.



21

How do we construct the regions?

- We seek regions $1, \ldots, J$ that minimize

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

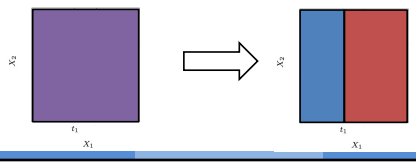
- Assume those regions are boxes (rectangular)
- Considering every possible split for every possible box combinations is computationally infeasible
  - We use recursive greedy binary splitting instead

22

Recursive binary splitting

- We start with whole feature space as one region
- We then find j and s such that splitting feature space into
  - $R_1(j,s) = \{X | X_j < s\}$ and
  - $R_2(j,s) = \{X | X_j \geq s\}$

  lead to greatest possible reduction in the RSS.



23

Recursive binary splitting

- So, we seek j and s minimize

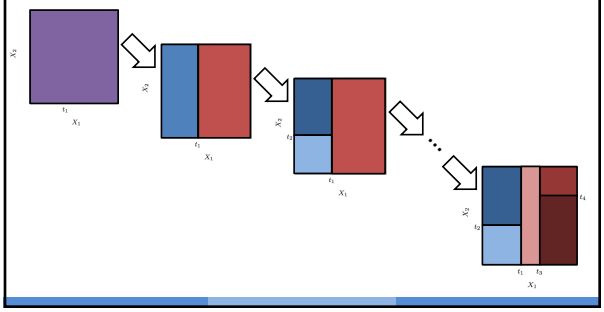
$$\sum_{i: x_i \in R_1(j,s)} \left(y_i - \hat{y}_{R_1}\right)^2 + \sum_{i: x_i \in R_2(j,s)} \left(y_i - \hat{y}_{R_2}\right)^2$$

  - Here $\hat{y}_{R_j}$ is the mean response for training-set observations in $R_j(j,s)$.

24

Recursive binary splitting

- We then continue this recursively for every region of space



25

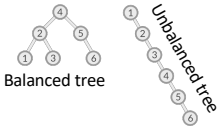
Recursive binary splitting

- This is a greedy algorithm. It makes the best decision possible at every step—the split that minimizes the RSS in a particular step.
- It does not look ahead and seek splits that would lead to better trees (smallest RSS) overall

26

Time complexity of decision trees

- What is the time complexity of decision trees?
  - Decision trees calculate splits on each of p features in each of n samples in every non-leaf node.
  - For perfectly balanced trees, tree depth is $O(\log n)$; for perfectly unbalanced trees it is is $O(n)$.
  - So, answer is between $O(pn^2)$ for unbalanced trees and $O(pn \log n)$ for balanced trees.



27

Tree pruning

DECISION TREES

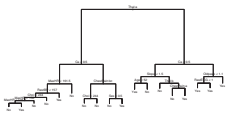
28

Tree pruning

The more branches we add to the regression tree, the better it will fit our training data. But, if we keep splitting until each data point in the training set gets its own terminal node, it will likely lead to overfitting.

Note: as per our (greedy) algorithm, we do not know how much the next split will reduce the RSS. So, we cannot stop once a specific RSS decrease in a split is below some threshold. Without a clear stopping criterion, overfitting is likely.

To avoid overfitting—and keep open the option to potentially find "good future splits"—one approach is tree pruning. We grow a very large tree and prune to get a good subtree.



29

Tree pruning

Pruning is a technique that reduces the size of decision-trees by removing sections of the tree that provide little power to classify instances. Pruning reduces the complexity of the final classifier, and hence improves predictive accuracy by reducing overfitting (i.e., reducing variability).

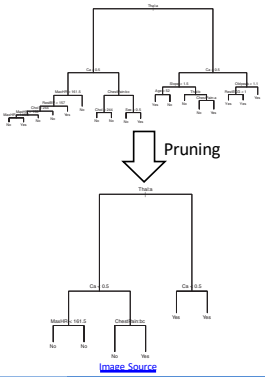


Image Source

30

Tree pruning

Here's how cost-complexity pruning (a.k.a. *weakest link pruning*) works:

- Grow a very large tree, T_0, using recursive binary splitting.
- Prune the tree by removing branches. This results in a subtree $T \subset T_0$.
- Consider the sequence of subtrees created from pruning T_0. For a tuning parameter, α, there is the optimal subtree, T^*, such that:

For some α:

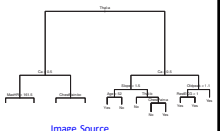


Image Source

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

31

Tree pruning

Here's how cost-complexity pruning works (continued):

$$T^* = \text{argmin}_T \underbrace{\sum_{m=1}^{|T|} \sum_{i:x_i \in R_m} (y_i - \hat{y}_{R_m})^2}_{\text{"RSS"}} + \underbrace{\alpha|T|}_{\text{Penalty}}$$

For some α:

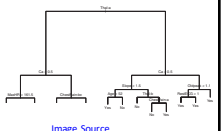


Image Source

- $|T|$ is the number of terminal nodes in the subtree T.
- R_m is the region of the predictor space associated with the m'th terminal node.
- Like Ridge or LASSO, we are penalizing both RSS and a tree-size penalty-term.

32

Tree pruning

Here's how cost-complexity pruning works (continued II):

- Use k-fold cross-validation by calculating the mean squared prediction error in the left-out training fold as a function of α. This will give us the optimal value of α.
- Pick the subtree that corresponds to the value of α that minimizes the cross-validation error.

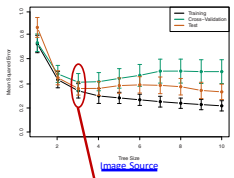
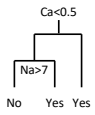


Image Source



33

Classification trees

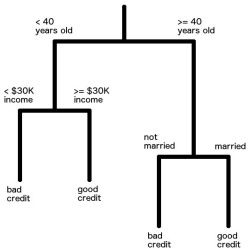
DECISION TREES

34

Classification Trees

So far, we focused on regression trees.

For *categorical* data, we can use classification trees. These are also created using binary recursive splitting. But rather than trying to yield an average value of best fit at the terminal nodes (as with regression trees), the terminal nodes give us the most common category (either majority or plurality vote) in each region of our predictor space (on the training data). The training-set agreement within each region can serve as a confidence measure.



35

Classification Trees

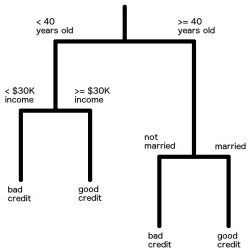
Mathematically, let us define

$$\hat{p}_{m,k} = \frac{\sum_{x_i \in R_m} I(y_i = k)}{\#\{x_i \in R_m\}},$$

the proportion of observations from class k in node m. Here $I(y_i = k)$ is the indicator function (i.e., 1 when $y_i = k$ and 0 elsewhere). We classify the observations in node m to class

$$\arg\max_k(\hat{p}_{m,k}),$$

the majority (for binary classification) or plurality (for 3 or more classes) class in node m.



36

Classification Trees

As a toy example of a classification tree, suppose we model

y ← credit (good or bad)

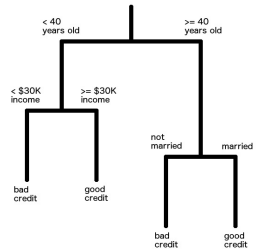
based on

x_1 ←age

x_2 ←income

x_3 ←marital status

Note, y is categorical: "good credit" or "bad credit".

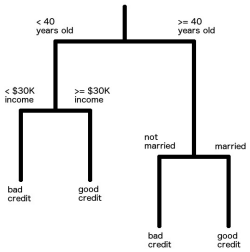


37

Classification Trees

The classification tree

1. Looks at the entire data set and considers splits along the 3 variables (age, income, marital status) and seeks the division resulting in most accuracy based on credit (output variable).
2. Splits first by age—people younger than 40 and people at least 40 years old.
3. Next splits the under-40's based on income.
4. Then splits the over-40's based on marital status.



38

Classification Trees

Regression trees decided where to split by minimizing RSS. How should we decide where to split the data for a classification tree?

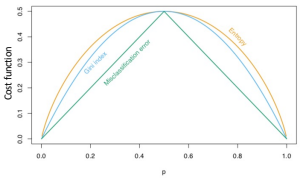
39

Classification Trees

Regression trees decided where to split by minimizing RSS. How should we decide where to split the data for a classification tree?
We need some measure of the accuracy of the prediction in the resulting regions.

The most used cost functions are:

- (Misclassification error)
- Gini index (a.k.a. Gini impurity), measuring classification error rate
- Cross-entropy



Cost functions for $K = 2$. (Cross entropy scaled to pass through (0.5,0.5).

40

Classification Trees Cost Function: Misclassification error

The misclassification error, E_m, at terminal node m is defined as

$$E_m = \frac{\sum_{x_i \in R_m} I(y_i \neq k)}{\#\{x_i \in R_m\}} = 1 - \hat{p}_{m,k}.$$

$\hat{p}_{m,k}$ ← proportion of training observations of class k at node m

This measures the proportion of misclassified samples for class k at node m.

More of a pedagogical measure; not often used for decision trees in practice (not sensitive enough).

41

Classification Trees Cost Function: Gini Index

The Gini index, G_m, at terminal node m is defined as

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

K ← total number of classes

$\hat{p}_{mk}$ ← proportion of training observations of class k at node m

The Gini index gives us a measure of variance across the K classes. It is also a measure of "node purity" at each node, because it is small when $\hat{p}_{mk}$ is close to 0 or 1.

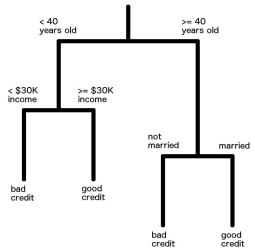
The Gini index for the entire model is usually a weighted average of G_m values for each terminal node.

42

Classification Trees Cost Function: Gini Index

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

Better classification trees tend to have lower Gini indexes. For instance, if the tree on the right fit the data perfectly, then every $\hat{p}_{mk}$ would be either 0 or 1; since the only people who fell into each terminal node would either all have good credit, or all have bad credit.



43

Classification Trees Cost Function: Cross-Entropy

Another, measure of classification error, which stems from information theory, is cross-entropy. The cross-entropy D_m at node m is:

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

K ← total number of classes

$\hat{p}_{mk}$ ← proportion of training observations of class k at node m

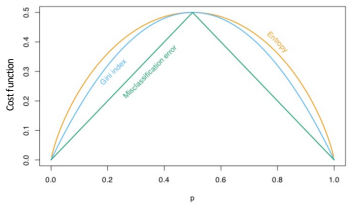
Note, $\hat{p}_{mk}$ is always between 0 and 1. So $-\hat{p}_{mk} \log \hat{p}_{mk} \geq 0 \; \forall k, m$. Lower cross-entropy again occurs when $\hat{p}_{mk}$ is close to 0 or 1 and indicates better model fit. Cross-entropy values tend to match up well with Gini index values.

44

Comparing Classification Trees Cost Functions

Comparing the most used cost functions for $K = 2$. (Cross entropy scaled to pass through $(0.5,0.5)$):

- (Misclassification error)
- Gini index (a.k.a. Gini impurity), measuring classification error rate
- Cross-entropy



45

Trees vs. Linear Models

When is it better to use a tree, and when is it better to use a linear model?

When the relation between the explanatory variables and the response variable is more similar to a linear boundary, a linear model will typically work well (top left), while a tree will not (bottom left).

When the relation between the explanatory variables and the response variable is nonlinear and complex, decision trees (bottom right) typically outperform linear models (top right).

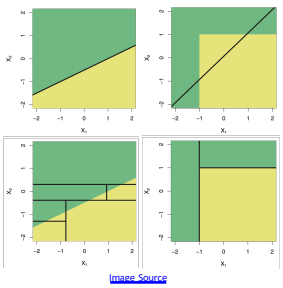


Image Source

46

Decision Tree Pros and Cons

Pros:

- Simple model to explain, understand and interpret.
- Easily handles qualitative predictors without dummy variables.
- Works well for non-linear classification.
- Same general formalism works for classification and regression

Cons:

- Often has low predictive power (due to high variance) compared with other methods.

The performance of trees can be improved using bagging, random forest, and boosting—coming up next.

47

AGGREGATION/ENSEMBLE TECHNIQUES

48

Aggregation/ensemble techniques

These techniques take an ensemble of learners and aggregate their output to a stronger learner than any of the single learners.

We are learning these techniques in the context of decision trees. But they can be used for many classification algorithms.

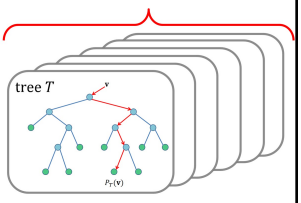


Image Source

49

Bagging

AGGREGATION/ENSEMBLE TECHNIQUES

50

Decision Tree Improvements: Bagging, Random Forests, Boosting

Bagging: Short for *bootstrap aggregation*, take several bootstrap samples of the training data, train trees on each bootstrapped set, and average the predictions (for regression) or take the majority vote (for classification).

Random Forests: Similar to bagging, but at each tree split, only considers a small, random subset of explanatory variables, not all variables.

Boosting: Builds trees sequentially using information from previously grown trees.

51

Bagging: Bootstrap Aggregation

Bagging is short for *bootstrap aggregation*.

Bootstrapping is a process of creating new data sets from the original training data by picking data points one-at-a-time, importantly *with replacement*.

Image Source

52

Bagging: Bootstrap Aggregation

For instance, bootstrapping the set of data points

$$(p_1, p_2, p_3, p_4, p_5)$$

to create 4 new data sets of the same size we might get

$$(p_2, p_1, p_1, p_3, p_5)$$
$$(p_3, p_4, p_4, p_5, p_4)$$
$$(p_3, p_1, p_2, p_4, p_1)$$
$$(p_5, p_5, p_3, p_5, p_3)$$

Again, sampling is with replacement.

Image Source

53

Bagging: Bootstrap Aggregation

Bagging procedure

- Generate B bootstrapped training data sets
- Train a decision tree on each bootstrapped set
- Define $\hat{f}^{*b}(x)$: the prediction from bootstrapped set b at point x
- Determine the overall bagging prediction to be the average of these bootstrapped predictions:
  - For regression: $\hat{f}_{bag}(x) = \frac{1}{B}\sum_{b=1}^{B}\hat{f}^{*b}(x)$
  - For classification: $\hat{f}_{bag}(x)$ is majority (or plurality) vote among $\hat{f}^{*b}(x)$

Image Source

54

Bagging: Bootstrap Aggregation

Bootstrapping reduces variance

- For n independent observations, each with variance σ^2, their average will have variance of σ^2/n (or standard deviation $\sigma/\sqrt{n}$). So, it increases prediction accuracy compared with an ordinary tree.
- If we had n independent training sets from the population, built a model on each and taken the average prediction, variance would decrease, and accuracy rise
- But that is impractical
- Instead, we bootstrap on our single, existing training set. This still reduces variance compared to a regular tree

55

Bagging: Bootstrap Aggregation

Advantages of bagging:

- Bootstrapping reduces variance—so, increases accuracy—compared with an ordinary tree
- We can (though typically do not) grow the bootstrapped trees very deep and not have to worry about pruning
- Bagging is general purpose: useful for various regression & classification techniques, but especially for trees.

56

Bagging: Bootstrap Aggregation

Estimating the test error for bagging without cross-validation:

Each bagged tree uses ~2/3 of the observations (sampled with replacement), so the remaining ~1/3 are considered out-of-bag (OOB) observations. Each observation is OOB for about 1/3 of the bagged trees, so we can average (regression) or majority vote (classification) the OOB trees for this observation to give us the OOB prediction. From this, we can find either the OOB mean squared error (MSE) for a regression, or the classification error for classification.

This is valid because we only use independent samples to estimate the test error, similarly to cross validation.

57

Bagging: Bootstrap Aggregation

Bagging often significantly improves the prediction accuracy of trees. But the resulting mode is not easy to interpret. A "bagged tree" is not as easy to interpret—how to parcellate the feature space for individual decisions?

So bagging increases prediction power at the expense of decreased interpretability.

58

Bagging: Importance of Variables

While more difficult to interpret than single trees, an overall summary of the importance of each predictor in bagging can be calculated using RSS (regression trees) or Gini index (classification trees).

For bagging regression-trees, record the total RSS decrease due to splits over a given predictor, averaged over all B bagged trees. Larger values indicates more important predictors. Similarly, for bagging classification trees, add up the total amount that the Gini index is decreased by splits over a given predictor, averaged over all B trees.

59

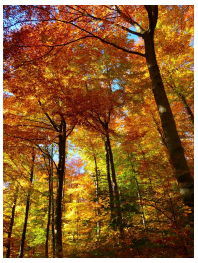
Random forest

AGGREGATION/ENSEMBLE TECHNIQUES

60

Random Forests

Random forests is a similar technique to bagging, though an improvement. We again build B decision trees based on bootstrapped examples. The key difference is that, with random forrests, a tree can only be split on a random subset of the predictors, m (typically $m \approx \sqrt{p}$ of the overall p predictors).



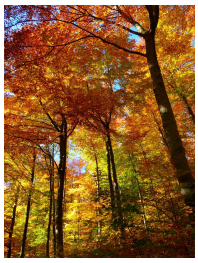
61

Random Forests

Random forests is a similar technique to bagging, though an improvement. We again build B decision trees based on bootstrapped examples. The key difference is that, with random forrests, a tree can only be split on a random subset of the predictors, m (typically $m \approx \sqrt{p}$ of the overall p predictors).

Rationale: create more decorrelated trees that could better search potential splits in feature space.

For instance, a first split that appears very strong might be used by virtually all trees, leaving a better split further down the road forever undiscovered.



62

Random Forests: Toy Example

Suppose we want to model someone is likely to default on a loan based on some 4 features.

$y \leftarrow$ default $\{\text{yes}, \text{no}\}$

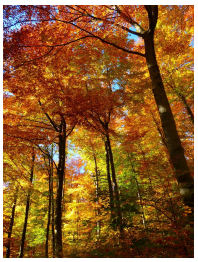
based on

$x_1 \leftarrow$ Credit history (CH)

$x_2 \leftarrow$ Available Credit (AC)

$x_3 \leftarrow$ Salary (S)

$x_4 \leftarrow$ Age (A)



63

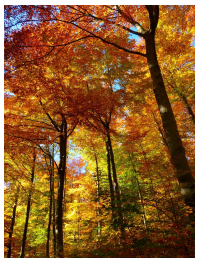
Random Forests: Toy Example

Feature set:

{CH, AC, S, A}

With bagging, all 4 features are considered at every tree split. So, if S appears (greedily & myopically) best, it might (almost) always be selected, obscuring a better, say, CH split down the road.

With random forests, only (say) 2 features are randomly considered at every tree split. So, the CH split is more likely to be discovered.



64

Random Forests

Here is an example where a single tree had 45.7% error. The number of splits allowed in every tree is designated by m. The error for bagging ($m = p$) somewhat higher than for random forest ($m = \sqrt{p}$).

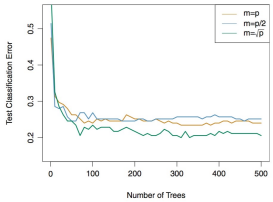
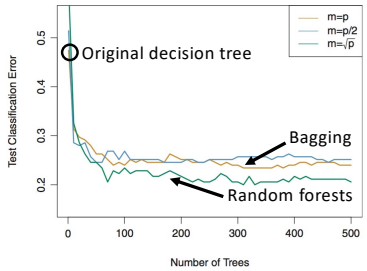


Image Source

65

Random Forests

How much do bagging and random forests help simple decision trees?



66

Advantages of Random Forests

Generally, the decorrelation in random forests at least slightly improves the prediction rate over bagging (here again with $m = \sqrt{p}$).

For random forests $m = \sqrt{p}$ is typically a good value.

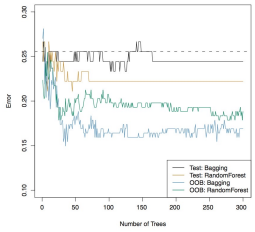


Image Source

67

Random Forests—Common Technique

- Random forests, typically with $m = \sqrt{p}$, is one of the most used techniques in machine-learning
  - Arguably most common technique among classical (non-neural-network) machine-learning ones
  - Possibly second only to neural networks in terms of overall use in AI

68

Why do the algorithms work?

- Bagging & random forests—the logic
  - If we had n independent training sets from the population, built a model from each training set and take the average prediction, the variance of that multi-training-set model would decrease, and accuracy rise
  - But that is impractical. We do not have n independent training sets, just one. Instead, we take our training set and sample from it with replacement.
  - Why with replacement? Sampling without replacement is permuting and would not do much. Sampling with replacement results in different training sets.

69

Why sample with replacement?

Example:

Sampling with replacement	Sampling without replacement
$(p_1, p_2, p_3, p_4, p_5)$	$(p_1, p_2, p_3, p_4, p_5)$
Create 4 new data sets of same size:	Create 4 new data sets of same size:
$(p_2, p_1, p_1, p_3, p_5)$	$(p_2, p_1, p_4, p_3, p_5)$
$(p_3, p_4, p_4, p_5, p_4)$	$(p_3, p_4, p_1, p_2, p_5)$
$(p_3, p_1, p_2, p_4, p_1)$	$(p_3, p_1, p_5, p_2, p_4)$
$(p_5, p_5, p_3, p_5, p_3)$	$(p_5, p_2, p_3, p_1, p_4)$
Each sampled dataset is different	All sampled datasets are just permutations of the original dataset

70

Why do the algorithms work?

- Bagging & random forests—the logic (cont.)
  - After constructing n bootstrapped (i.e., sampled with replacement) training sets, we build a decision tree on each of them
    - Typically, each tree is not deep, and no pruning is used
  - This aggregate method, termed bagging, increases the accuracy a lot over a single decision tree
  - But this comes at the cost of the method being harder to interpret

71

Why do the algorithms work?

- Bagging & random forests—the logic (cont. II)
  - However, remember that decision-trees is a greedy algorithm. Hence, it makes the best decision for every split without regard of how it will affect future splits.
  - Imagine that some feature, i, seems great to split on whenever it is present. But splitting on i ends up taking the tree in the wrong direction. And it makes the tree overall worse than not splitting on i.
    - An algorithm that looks ahead would have discovered this and not split on i.
    - But decision trees, being greedy, do not look ahead

72

Why do the algorithms work?

- Bagging & random forests—the logic (cont. III)
  - The solution is not to let most trees split on i. This lies at the heart of random forests.
  - In random forests, the trees cannot just split on whichever p feature appears greedily best. Only a small subset of features, say m features, can be used on every split. And those m features are selected randomly on every split. Typically, $m = \sqrt{p}$ is selected.
  - So, if there are, say, 100 features, and only $\sqrt{100} = 10$ are selected on every split, feature i will be a candidate for splitting only rarely—1 in 10 splits in this case. And that will lead to better trees.

73

Why do the algorithms work?

Example:

Bagging	Random forests
Overall features on which to split: $(p_1, p_2, \cdots, p_{100})$	Overall features on which to split: $(p_1, p_2, \cdots, p_{100})$
Features on which to split for first 3 splits: $(p_1, p_2, \cdots, p_{100})$ $(p_1, p_2, \cdots, p_{100})$ $(p_1, p_2, \cdots, p_{100})$	Features on which to split for first 3 splits: $(p_{13}, p_3, p_7, p_{32}, p_{75}, p_{19}, p_{62}, p_{81}, p_{59}, p_{22})$ $(p_{27}, p_{11}, p_{72}, p_{49}, p_5, p_{99}, p_{63}, p_{12}, p_{91}, p_{86})$ $(p_{63}, p_{91}, p_{26}, p_{93}, p_{72}, p_{65}, p_{54}, p_{43}, p_{15}, p_{28})$
All features are OK on each split	Only 10 of the 100 features are OK on each split

74

Boosting

AGGREGATION/ENSEMBLE TECHNIQUES

75

Boosting

Boosting can be (superficially) viewed as improved bagging. For bagging, the trees were grown independently from each other, in parallel. In boosting the growth is sequential (or serial), each tree is grown using information from previously grown trees—each tree is fit on a modified version of the original data set. Hence boosting—unlike bagging—is not easy to parallelize.

Boosting is a powerful, general-purpose technique (especially *AdaBoost*).

Image Source

76

Boosting

The idea is to apply several weak-learners one after the other. Weak learners predict only slightly better than chance. Each weak learner "chops away" at the residual errors of the previous learners, slowly explaining more of the variance that has remained unexplained by the previous learners.

Boosting is thus a slow learning method that gradually course-corrects towards a better model fit. The weighted aggregation of many such slow learners ("voting by committee") often works well.

AdaBoost is the first practical and most common boosting algorithm in use.

Image Source

77

AdaBoost—General Scheme

Here $\alpha_1, \ldots, \alpha_M$ are computed by the boosting algorithm to give higher influence to the more accurate classifiers in the sequence.

In each step, observations that were misclassified by $G_{m-1}(x)$ have their weights increased and those correctly classified have their weights decreased. Hence, hard to classify observations receive increasing influence on successive classifiers, forcing those to concentrate on previously misclassified samples.

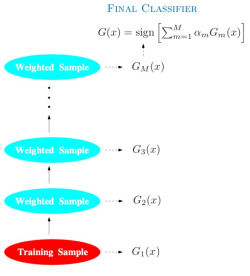


Image Source

78

AdaBoost—Algorithm

For binary classification, $Y \in \{-1,1\}$ (M interations, N samples):

1. Initialize observation weights uniformly: $w_i = \frac{1}{N}$, for $i = 1, \ldots, N$.
2. For $m = 1$ to M
   a) Fit classifier $G_m(x)$ to the training data using weights $\{w_i\}_{i=1}^N$.
   b) Compute
   $$\text{err}_m = \frac{\sum_{i=1}^N w_i \, I(y_i \neq G_m(x_i))}{\sum_{i=1}^N w_i}.$$
   c) Compute $\alpha_m = \log((1 - \text{err}_m)/\text{err}_m)$.
   d) $w_i \leftarrow w_i \cdot \exp[\alpha_m \cdot I(y_i \neq G_m(x_i))]$, $i = 1, \ldots, N$.
3. Output $G(x) = \text{sign}[\sum_{m=1}^M \alpha_m \, G_m(x)]$.

79

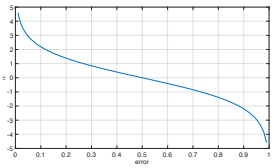
AdaBoost—Algorithm

How are the different week learners integrated into an overall classifier?

Each learner, G_m, is given weight, α_m, as a function of its error. Then the sign of this weighted sum is the output:

$$G(x) = \text{sign}\left[\sum_{m=1}^M \alpha_m \, G_m(x)\right].$$

$$\alpha = \log((1 - \text{error})/\text{error})$$



80

AdaBoost—Algorithm

1. We initialize all weights uniformly.
2. For each iteration, m:
   a) We run classifier G_m on the training set with weights $\{w_i\}$.
   b) We compute the error, err_m, for that classifier (based on the weights).
   c) Based on err_m, we then compute α_m (the weight of G_m in the final classifier G—line 3).
   d) The weight for each sample, w_i, increases exponentially by α_m if G_m misclassifies x_i. So, observations misclassified by G_m increase their weight by e^{α_m}, increasing their influence on the next classifier, G_{m+1}.
3. The final classifier, G, then gives higher influence to more accurate classifiers in the sequence.

For binary classification, $Y \in \{-1,1\}$ (M classifiers, N samples):
1. Initialize observation weights $w_i = 1/N$ for $i = 1, \ldots, N$.
2. For $m = 1$ to M
   a) Fit classifier $G_m(x)$ to the training data using weights $\{w_i\}_{i=1}^N$.
   b) Compute $\text{err}_m = \frac{\sum_{i=1}^N w_i \, I(y_i \neq G_m(x_i))}{\sum_{i=1}^N w_i}$.
   c) Compute $\alpha_m = \log((1 - \text{err}_m)/\text{err}_m)$.
   d) $w_i \leftarrow w_i \cdot \exp[\alpha_m \cdot I(y_i \neq G_m(x_i))]$, $i = 1, \ldots, N$.
3. Output $G(x) = \text{sign}[\sum_{m=1}^M \alpha_m \, G_m(x)]$.

81

AdaBoost—General Scheme

This is the general scheme again, after scrutinizing the algorithm:

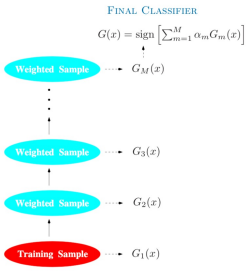


Image Source

82

Boosting

Boosting tends to improve accuracy of weak learners considerably.

Example: For a 10-dimensional (simulated) dataset of Gaussians, a weak, binary learner (tree stump) has a test-error rate of 45.8% (top dotted line)—close to chance level at 50%. After 400 iterations, its error rate dramatically decreases to 5.8%. Its accuracy on the test set is thus much better than a much larger tree (lower dotted line, 24.7%).

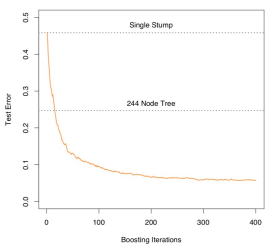


Image Source

83

Boosting

- Boosting serially combines many weak models to create a stronger model.
- Can perform better than random forests and can be computatationally faster.
- If iterations number gets too large, boosting may (slowly) overfit, unlike bagging or random forest

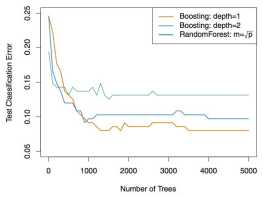


Image Source

86

Summary

- Decision trees (for regression and classification) are simple models, easy to understand and interpret. But they tend to overfit and predict badly.
  - Regression trees use an RSS cost function
  - Classification trees use the Gini Index & Cross-Entropy
- Pruning helps mitigate overfiting in trees
- Other techniques to improve decision-tree performance much more:
  - Bagging: create many trees by bootstrapping training set samples
  - Random Forests: bagging with randomly limited variables to split on
  - Boosting: incremental, slow learning using small trees, each working on the residuals of the previous trees.

87

Take 🏠 message for rest of course: Tree-based methods

- Aggregating over weak learning rule—e.g., decisions trees, in independent manner improves it considerably. Can lead to strong learning rules.
- This can be done using bootstrapping over the training set (bagging), preferably in a decorrelated manner (random forest).
- It can also be done more systematically, where each weak learner works on the residuals of the previous learners, slowly removing any variance left unexplained (boosting).
- Random forests often works best in practice

88 / 49

88