Generalized Boosted Models

Amanda Kube

Washington University in St. Louis

December 1, 2018

Overview

What is Boosting?

First some vocabulary

What is Boosting?

First some vocabulary

- Weak learner - a learning model that does (slightly) better than chance

What is Boosting?

First some vocabulary

- Weak learner - a learning model that does (slightly) better than chance
- Strong learner - a learning model that is closely correlated with the real outcomes (ie. does *much* better than chance)

What is Boosting?

First some vocabulary

- Weak learner - a learning model that does (slightly) better than chance
- Strong learner - a learning model that is closely correlated with the real outcomes (ie. does *much* better than chance)

Idea: Use a set of weak learners to create a strong learner

Decision Trees

Almost always, decision stumps are used as the weak learner

So first, we need to learn a bit about decision trees...

Decision Trees

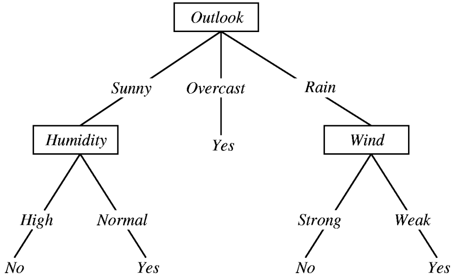
Almost always, decision stumps are used as the weak learner

So first, we need to learn a bit about decision trees...

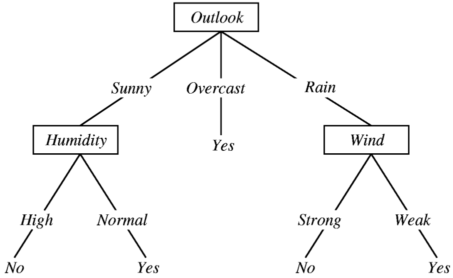
Imagine you are trying to decide whether or not to play tennis.

There are several factors involved in your decision. Let's focus on weather, humidity, and wind

Decision Trees Example



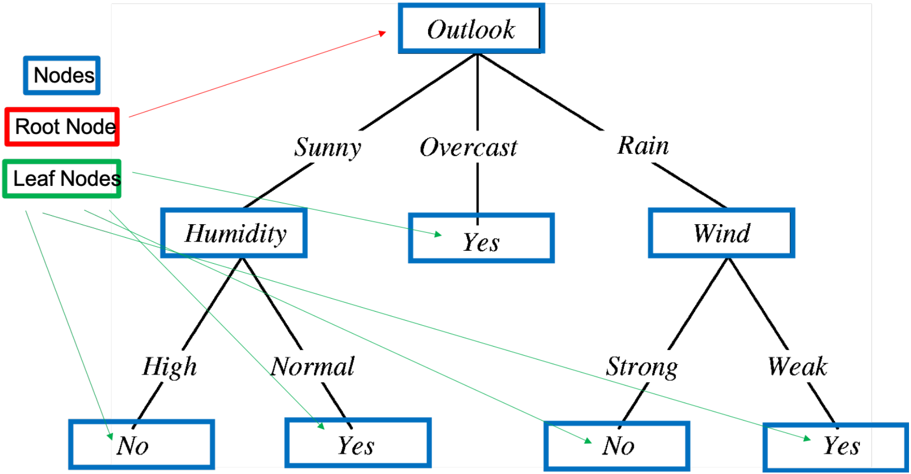
Decision Trees Example



What decision will we make if we have the following new data:

Obs	Outlook	Humidity	Wind
1	Sunny	Normal	Weak
2	Rain	High	Strong

Decision Trees Vocabulary



More on Decision Trees

Decision trees can be used for classification (like this example) or for regression, where a number will be at each leaf node - usually the average of the outcomes for all data points that make it to that node.

More on Decision Trees

Decision trees can be used for classification (like this example) or for regression, where a number will be at each leaf node - usually the average of the outcomes for all data points that make it to that node.

They can be fit using the ID3 Algorithm which uses information gain and entropy to pick the splits. (I will let Jacob talk more about this later.)

More on Decision Trees

Decision trees can be used for classification (like this example) or for regression, where a number will be at each leaf node - usually the average of the outcomes for all data points that make it to that node.

They can be fit using the ID3 Algorithm which uses information gain and entropy to pick the splits. (I will let Jacob talk more about this later.)

Individual decision trees are often not useful - eg. they can change drastically from small changes in input - BUT ensembles of trees can be very useful!

Two common ways to use multiple trees:

- Bagging (Random Forests)
- Boosting (GBMs)

More on Decision Trees

Decision trees can be used for classification (like this example) or for regression, where a number will be at each leaf node - usually the average of the outcomes for all data points that make it to that node.

They can be fit using the ID3 Algorithm which uses information gain and entropy to pick the splits. (I will let Jacob talk more about this later.)

Individual decision trees are often not useful - eg. they can change drastically from small changes in input - BUT ensembles of trees can be very useful!

Two common ways to use multiple trees:

- Bagging (Random Forests)
- Boosting (GBMs)

We are interested in boosting using short decision trees (one or two splits). These are sometimes referred to as decision "stumps".

What is Boosting?

There are many different boosting algorithms. For example

- AdaBoost (Adaptive Boosting)
- XGBoost
- LPBoost
- LogitBoost

What is Boosting?

There are many different boosting algorithms. For example

- AdaBoost (Adaptive Boosting)
- XGBoost
- LPBoost
- LogitBoost

All boosting algorithms have a similar format:

Basic Algorithm

for n iterations do
    learn weak classifier
    add to strong classifier
    re-weight data

Generalized Boosting

Boosting has many forms.

We can vary several elements of the boosting algorithm

- Loss function (ie squared-error, hinge, absolute)
- Type of weak learner (ie decision stump, weak regression models)
- Optimization (ie gradient decent, Newton-Raphson)

Why Boost?

We've seen examples of strong learners:

- OLS
- Logistic Regression
- Decision Tree

If we already know how to make a strong learner, why boost?

Why Boost?

We've seen examples of strong learners:

- OLS
- Logistic Regression
- Decision Tree

If we already know how to make a strong learner, why boost?
Boosting is more work, we have to learn multiple models and iteratively re-weight data...

Why Boost?

We've seen examples of strong learners:

- OLS
- Logistic Regression
- Decision Tree

If we already know how to make a strong learner, why boost?
Boosting is more work, we have to learn multiple models and iteratively re-weight data...

Boosting reduces bias *and* variance

Why Boost?

We've seen examples of strong learners:

- OLS
- Logistic Regression
- Decision Tree

If we already know how to make a strong learner, why boost?
Boosting is more work, we have to learn multiple models and iteratively re-weight data...

Boosting reduces bias *and* variance

Remember the bias-variance trade-off?

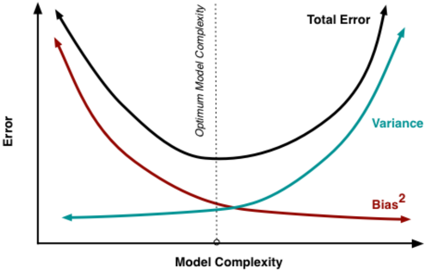
Bias and Variance

Bias Variance Trade-off

Expected squared error can be decomposed as follows

$$Error(x) = E[(f(x) - \hat{f}(x))^2] = (E[\hat{f}(x)] - f(x))^2 + E[(\hat{f}(x) - E[\hat{f}(x)])^2] + \epsilon$$

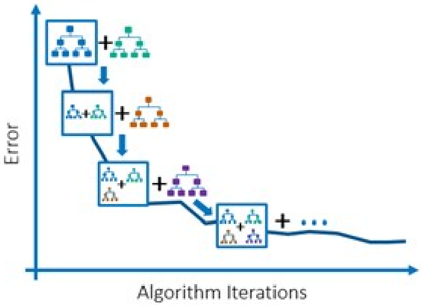
$$Error(x) = Bias^2 + Variance + IrreducibleError$$



The Magic of Boosting

Boosting can decrease *both*!

- Bias: Each time we re-weight the training data, we are telling the model to focus on observations we are poor at classifying
- Variance: We average our weak learners which decreases variance compared to any single weak learner.



The Math Behind the Magic

Let's derive the Gradient Boosting Machine (Friedman, 2001)

The Math Behind the Magic

Let's derive the Gradient Boosting Machine (Friedman, 2001)

Let $\Psi(y, f)$ be our loss function.
We want to find $\hat{f}(x)$ that minimizes our expected loss

$$\hat{f}(x) = argmin_{f(x)} E_{y,x}[\Psi(y, f)] = argmin_{f(x)} E_y[E_{y|x}\Psi(y, f)|x]$$

So we can focus on finding $\hat{f}(x)$ such that

$$\hat{f}(x) = argmin_{f(x)} E_{y|x}[\Psi(y, f)|x]$$

The Math Behind the Magic

If we are using a parametric regression model, we wish to find $\hat{\beta}$ such that:

$$\hat{\beta} = argmin_{f(x)} \sum_{i=1}^{n} \Psi(y_i, f(x_i; \beta))$$

The Math Behind the Magic

If we are using a parametric regression model, we wish to find $\hat{\beta}$ such that:

$$\hat{\beta} = argmin_{f(x)} \sum_{i=1}^{n} \Psi(y_i, f(x_i; \beta))$$

But we are using a nonparametric model so we wish to decrease this function:

$$J(f) = \sum_{i=1}^{n} \Psi(y_i, f(x_i))$$

The Math Behind the Magic

If we are using a parametric regression model, we wish to find $\hat{\beta}$ such that:

$$\hat{\beta} = argmin_{f(x)} \sum_{i=1}^{n} \Psi(y_i, f(x_i; \beta))$$

But we are using a nonparametric model so we wish to decrease this function:

$$J(f) = \sum_{i=1}^{n} \Psi(y_i, f(x_i))$$

We can do this using gradient decent to move in the direction of greatest decrease. Let ρ be our stepsize.

$$\hat{f} \leftarrow \hat{f} - \rho \nabla J(f)$$

The Math Behind the Magic

One such update will not be enough, but this is the reasoning behind the Gradient Boosting Machine. We can use different loss functions, different step sizes, etc.

The Math Behind the Magic

One such update will not be enough, but this is the reasoning behind the Gradient Boosting Machine. We can use different loss functions, different step sizes, etc.
A very general form of boosting:

$$\hat{f}(x) \leftarrow \hat{f}(x) + \lambda E[z(y, \hat{f}(x))|x]$$

where λ is our stepsize and $E[z(y, \hat{f}(x))|x]$ is our regression.

We can also use stochasitc gradient decent to improve the algorithm and runtime by subsampling during each iteration.

The Math Behind the Magic

One such update will not be enough, but this is the reasoning behind the Gradient Boosting Machine. We can use different loss functions, different step sizes, etc.
A very general form of boosting:

$$\hat{f}(x) \leftarrow \hat{f}(x) + \lambda E[z(y, \hat{f}(x))|x]$$

where λ is our stepsize and $E[z(y, \hat{f}(x))|x]$ is our regression.

We can also use stochasitc gradient decent to improve the algorithm and runtime by subsampling during each iteration.

Takeaway: Boosting is flexible and there are several parameters you can change to fit your particular problem.

The Algorithm

Initialize $\hat{f}(\mathbf{x})$ to be a constant, $\hat{f}(\mathbf{x}) = \arg\min_\rho \sum_{i=1}^N \Psi(y_i, \rho)$.
For t in $1, \ldots, T$ do

1. Compute the negative gradient as the working response

$$z_i = -\frac{\partial}{\partial f(\mathbf{x}_i)} \Psi(y_i, f(\mathbf{x}_i)) \bigg|_{f(\mathbf{x}_i)=\hat{f}(\mathbf{x}_i)} \tag{1}$$

2. Fit a regression model, $g(\mathbf{x})$, predicting z_i from the covariates $\mathbf{x}_i$.
3. Choose a gradient descent step size as

$$\rho = \arg\min_\rho \sum_{i=1}^N \Psi(y_i, \hat{f}(\mathbf{x}_i) + \rho g(\mathbf{x}_i)) \tag{2}$$

4. Update the estimate of $f(\mathbf{x})$ as

$$\hat{f}(\mathbf{x}) \leftarrow \hat{f}(\mathbf{x}) + \rho g(\mathbf{x}) \tag{3}$$

Figure 1: Friedman's Gradient Boost algorithm

Fitting a gbm in R

Use the gbm package

gbm(formula = formula(data), distribution = "bernoulli", data = list(), weights, var.monotone = NULL, n.trees = 100, interaction.depth = 1, n.minobsinnode = 10, shrinkage = 0.1, bag.fraction = 0.5, train.fraction = 1, cv.folds = 0, keep.data = TRUE, verbose = FALSE, class.stratify.cv = NULL, n.cores = NULL)

Most arguments are similar to other modeling arguments in R, so we focus on the most unafmiliar (and very important) modeling choice for gbm.

The gbm() function

The distribution argument can be used to specify response distribute and thus the type of loss we want to use.
Current options include:

- gaussian (squared error)
- laplace (absolute loss)
- tdist (t-distribution loss)
- bernoulli (logistic regression for 0-1 outcomes)
- huberized (huberized hinge loss for 0-1 outcomes)
- adaboost (the AdaBoost exponential loss for 0-1 outcomes)
- poisson (count outcomes)
- coxph (right censored observations)
- quantile
- pairwise (ranking measure using the LambdaMart algorithm)

Boston Housing Data

Boston Housing Data from the MASS package in R

Response Variable

- medv - The median value of the homes (in 1000s)

Predictor Variables

- crim - per capita crime rate
- zn - proportion zoned for lots over 25000 sq. ft.
- indus - proportion of non-retain business acres
- chas - tract bounds Charles River
- nox - nitrogen oxides concentration
- rm - average number of rooms
- age - proportion of units built before 1940
- dis - weighted mean of distances to 5 Boston employment centers
- rad - index of accessibility to radial highways
- tax - full-value property-tax rate (per $10000)
- ptratio - pupil-teacher ratio by town
- lstat - lower status of the population (percent)

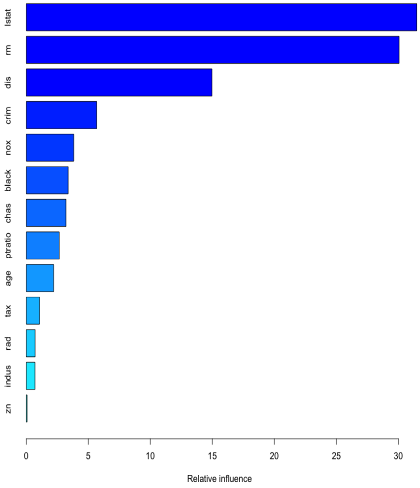
R Example

The following example is adapted from
https://datascienceplus.com/gradient-boosting-in-r/

```
library(gbm)
library(MASS)
train=sample(1:506,size=374)
Boston.boost=gbm(medv ~ . ,data = Boston[train,],
    distribution = "gaussian", n.trees = 1000)
Boston.boost
summary(Boston.boost)
```

R Example

Variable Importance

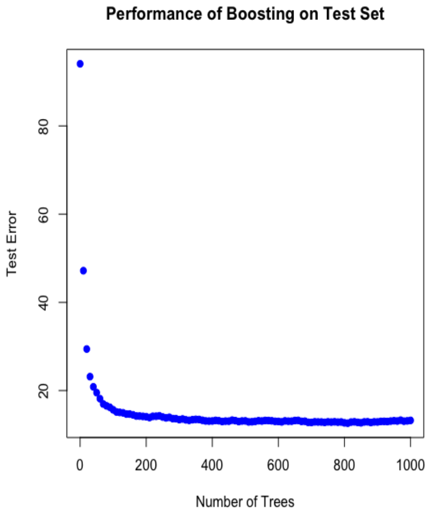



```
> summary(Boston.boost)
            var     rel.inf
lstat     lstat 31.48285216
rm           rm 30.05253703
dis         dis 14.96511194
crim       crim  5.67760338
nox         nox  3.82751897
black     black  3.37871925
chas       chas  3.20211316
ptratio ptratio  2.65902201
age         age  2.20829317
tax         tax  1.07056569
rad         rad  0.71023104
indus     indus  0.69698261
zn           zn  0.06844958
```

R Example

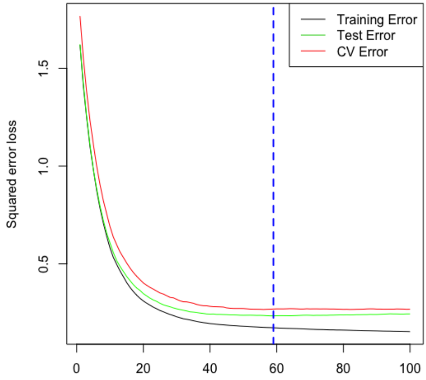
```
n.trees = seq(from=0 ,to=1000, by=10)
predmatrix<-predict(Boston.boost,Boston[-train,],
    n.trees = n.trees)
test.error<-with(Boston[-train,],
    apply((predmatrix-medv)^2,2,mean))
plot(n.trees , test.error , pch=19,col="blue",
    xlab="Number of Trees",ylab="Test Error",
    main = "Performance of Boosting on Test Set")
```

R Example


Performance of Boosting on Test Set
Test Error
20
40
60
80
0
200
400
600
800
1000
Number of Trees

R Example

```
best.iter <- gbm.perf(gbm1, method = "cv")
legend("topright",legend=c("Training Error",
    "Test Error","CV Error"), col = c(1,3,2), lty=1)
```



R Example

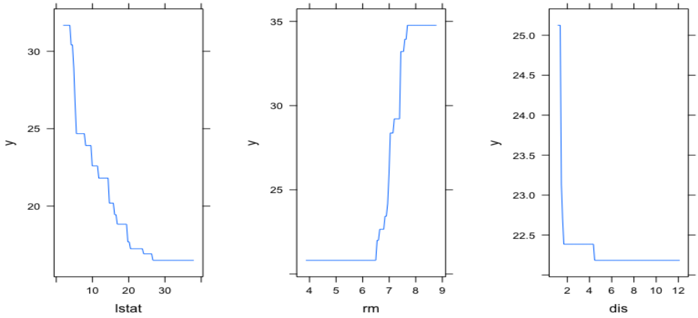
SplitVar: -1 indicates no split (leaf node), otherwise refers to column of training set (starting at 0)
SplitCode: Value that goes to left node
ErrorReduction: How much does this split help our overall fit
Prediction: For leaves, the prediction given to values ending at this node

```
> print(pretty.gbm.tree(Boston.boost, i.tree = 1))
  SplitVar SplitCodePred LeftNode RightNode MissingNode ErrorReduction Weight Prediction
0       12     5.1950000        1         2           3       8271.966    187  0.0315508
1       -1     1.6164553       -1        -1          -1          0.000     28  1.6164553
2       -1    -0.2475519       -1        -1          -1          0.000    159 -0.2475519
3       -1     0.0315508       -1        -1          -1          0.000    187  0.0315508
```

R Example

Univariate partial dependence plots

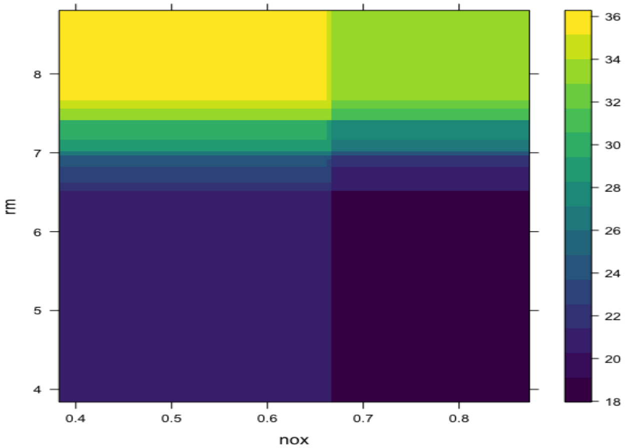
```
p1 <- plot(Boston.boost, i.var = "lstat", n.trees = best.i
p2 <- plot(Boston.boost, i.var = "rm", n.trees = best.iter
p3 <- plot(Boston.boost, i.var = "dis", n.trees = best.ite
grid.arrange(p1, p2, p3, ncol = 3)
```



R Example

Bivariate partial dependence plot

```
plot(Boston.boost, i.var = 5:6, n.trees = best.iter)
```



Summary

Benefits

Summary

Benefits

- Flexible
- Low bias and variance

Summary

Benefits

- Flexible
- Low bias and variance

Drawbacks

Summary

Benefits

- Flexible
- Low bias and variance

Drawbacks

- Time consuming/Complex to model
- Hard to interpret