Tree-based ML

DATA624, Spring 2020

Calvin Wong, Murali Kunissery, Omar Pineda

Agenda:

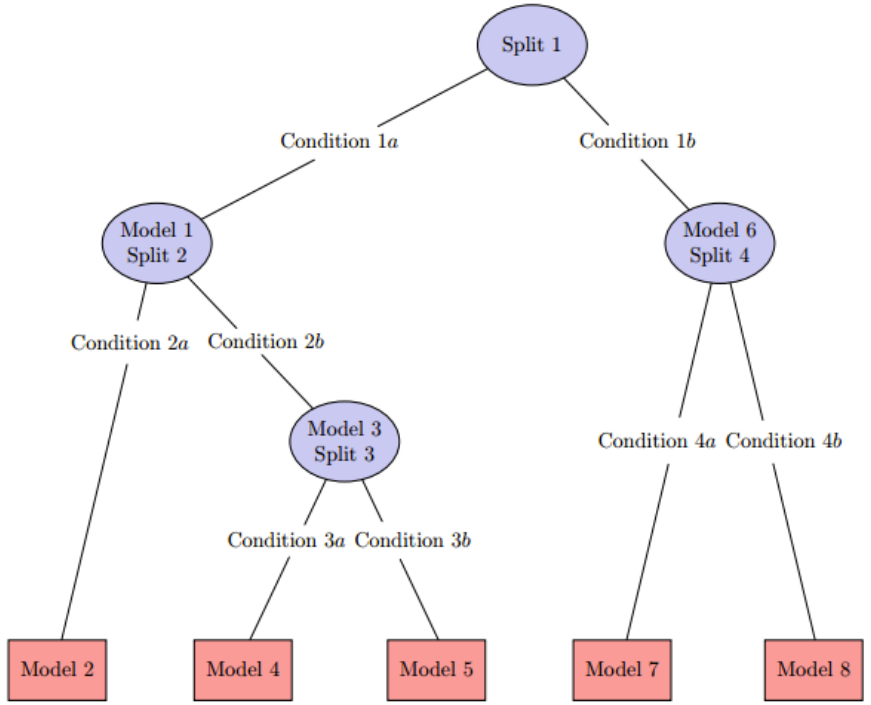
- Basic Regression Trees
- Regression Model Trees
- Bagged Trees
- Random Forests
- Boosting

Basic Regression Trees

- Tree-based models consist of one or more nested if-then statements for the predictors that partition the data. A model is used to predict the outcome within these partitions.
- The final regions are called leaves and the points where the splits occur are nodes.
- For basic regression trees, the models in the nodes are simple averages (constants)
- Benefits: highly interpretable, easy to implement, can handle many predictors without a a need to pre-process
- Challenges: instability, less than optimal predictive performance, selection bias, these models may not do a good job predicting samples whose true outcomes are extremely high or low
- Notes: We can over-fit if we have too many splits in the tree. If two predictors are highly correlated, the choice of which to use is somewhat random


```
library(caret)
library(rpart)
rpartTree <- train(y ~ ., data = trainData, method = 'rpart')
```

Regression Model Trees



- The terminal nodes predict the outcome using a linear model (as opposed to the simple average).
- When a sample is predicted, it is often a combination of the predictions from different models along the same path through the tree.
- The split that is associated with the largest reduction in error is chosen and a linear model is created within the partitions using the split variable in the model.
- Once the complete set of linear models have been created, each undergoes a simplification procedure to potentially drop some of the terms.
- Also incorporates a type of smoothing to decrease the potential for over-fitting.
- Once the tree is fully grown, it is pruned back by finding inadequate subtrees and removing them.

```
Library(caret)
library(RWeka)
m5tree <- train(y ~ ., data = trainData, method = 'M5')
```

Bagged Trees
Introduction

“Bagging predictors is a method of for generating multiple versions of a predictor and using these to get an aggregated predictor”

Leo Breiman, University of California
Bagging Predictors – September 1994

Bootstrap Aggregation aka “Bagging“

for $i = 1$ *to* m **do**
  Generate a bootstrap sample of the original data
  Train an unpruned tree model on this sample
end

Algorithm 8.1: Bagging

Why use bagging

- Bagging effectively reduces the variance of a prediction through its aggregation process
  - For models that produce an unstable prediction, bagging reduces the variance in the prediction
  - Breiman states that we can see a reduction in test set misclassification rates from 20% to 47%

Table 1 Missclassification Rates (Percent)

Data Set	$\bar{e}_S$	$\bar{e}_B$	**Decrease**
waveform	29.0	19.4	33%
heart	10.0	5.3	47%
breast cancer	6.0	4.2	30%
ionosphere	11.2	8.6	23%
diabetes	23.4	18.8	20%
glass	32.0	24.9	22%
soybean	14.5	10.6	27%

Why use bagging

- Bagging models can provide their own internal estimate of predictive performance that correlates well with either cross-validation estimates or test set estimates
  - Out-of-bag can be used to assess the predictive performance of that specific model
  - Out-of-bag estimate

Conditions

- Bagging can push a good but unstable procedure a significant step towards optimality, however, can degrade the performance of stable procedures
- Computational costs and memory requirements increase as the number of bootstrap samples increases
- Bagged model is much less interpretable than a model that is not bagged

Random Forest
Introduction

Reducing correlation among trees, known as de-correlating trees, is then the next logical step to improving the performance of bagging

Through multiple attempts of generalization the original bagging algorithm, in 2001, Breiman constructed a unified algorithm called **random forests**

```
Select the number of models to build, m
for i = 1 to m do
    Generate a bootstrap sample of the original data
    Train a tree model on this sample
    for each split do
        Randomly select k (< P) of the original predictors
        Select the best predictor among the k predictors and
        partition the data
    end
    Use typical tree model stopping criteria to determine when a
    tree is complete (but do not prune)
end
```

Algorithm 8.2: Basic Random Forests

Why use Random Forest

- With bagging, the trees are not completely independent of each other since all of the original predictors are considered at every split of every tree
- Reducing correlation among trees, known as de-correlating trees, is then the next logical step to improving the performance of bagging.
- Compared to bagging, random forests is more computationally efficient on a tree-by-tree basis since the tree building process only needs to evaluate a fraction of the original predictors at each split

Conditions

- The larger the forest, the better the results
  - a starting point, the authors suggest using at least 1,000 trees. If the cross-validation performance profiles are still improving at 1,000 trees, then incorporate more trees until performance levels off

- The ensemble nature of random forests makes it impossible to gain an understanding of the relationship between the predictors and the response. However, because trees are the typical base learner for this method, it is possible to quantify the impact of predictors in the ensemble.

Boosting: - A quick history of its evolution.

- Boosting models were original developed for classification problems
- AdaBoost provides a practical implementation of boosting a weak learner to a strong learner.
- AdaBoost was a powerful prediction tool and was widely adopted in applications with gene expressions
- Further research enabled boosting to be interpreted as a forward stagewise additive model that minimizes exponential loss
- The new research enabled the method to be extended to regression problems
- Evolved from AdaBoost Algorithm to Friedman's stochastic gradient boosting machine which encompasses both classification and regression.

Boosting: - Basic Principle

- Given a loss function (e.g..squared error for a regression) and a weak learner (e.g.. regression trees) , the algorithm seeks to find an additive model that minimizes the loss function.
- The algorithm is typically initialized with the best guess of the response (e.g., the mean of the response in regression). The gradient (e.g., residual) is calculated, and a model is then fit to the residuals to minimize the loss function
- The current model is added to the previous model and the procedure continues for a specified number of iterations.

Boosting: - Algorithm & Diff with Random forest

1 Select tree depth, D, and number of iterations, K

2 Compute the average response, $\bar{y}$, and use this as the initial predicted value for each sample

3 for $k = 1$ *to* K do

4 Compute the residual, the difference between the observed value and the *current* predicted value, for each sample

5 Fit a regression tree of depth, D, using the residuals as the response

6 Predict each sample using the regression tree fit in the previous step

7 Update the predicted value of each sample by adding the previous iteration's predicted value to the predicted value generated in the previous step

8 end

- Random Forest –
  - ✓ All trees created independently
  - ✓ Each tree has max depth
  - ✓ Each tree contributes equally to final model
  - ✓ Can be parallelly processed for better computation time
- Gradient Boosting –
  - ✓ Dependent on past trees
  - ✓ Have minimum tree depth
  - ✓ Contributes unequally to final model
  - ✓ Cannot be parallelly processed

Boosting : - Overfitting & Regularization

- The gradient booting machine could be susceptible to over fitting .
- Despite using weak learners, boosting still employs the greedy strategy of choosing the optimal weak learner at each stage.
- A remedy for that is to constrain the learning process by employing regularization, or shrinkage
  - In the previous algorithm shown, Instead of adding the predicted value for a sample to previous iteration's predicted value, only a fraction of the current predicted value is added to the previous iteration's predicted value
  - This fraction is commonly referred to as the learning rate and is parameterized by the symbol, λ.
  - This parameter can take values between 0 and 1 and becomes another tuning parameter for the model
  - that the value of the parameter is inversely proportional to the computation time required to find an optimal model

π

Questions?