CAPS Notes - Lecture 3
Decision Trees, Random Forest and Boosted Trees

Leo Klenner, Henry Fung, Cory Combs

Last updated: 11/11/2019

Overview

In session 3, we will first focus on a type of statistical learning approach called "tree-based" models for regression and classification problems. A main feature of tree-based models is the segmentation of the predictor space into a number of smaller regions from which the prediction for a test observation is made. The sequence of splitting rules can be represented in a tree diagram; hence, these methods are somewhat more intuitive and explainable than the linear regression model.

Second, we will focus on two techniques that arguments the basic decision tree model to improve its predictive accuracy: bagging and boosting. These are powerful methods that can make low-bias and low-variance predictions with small test error.

Third, we will discuss a basic principle in machine learning: **no one model works best for every problem**. There are different tradeoffs that we must make depending on the nature of the data problem and the resources that we have at our disposal. In this context, we will compare the merits and limitations of the linear regression, decision tree, random forest, and boosted tree models.

The session 3 notes cover:

- Making predictions with decision tree models
- Bootstrap Aggregation (Bagging)
- Bagged Trees
- Random Forests
- Gradient Boosting
- Model selection and comparison

1. Some basic facts about decision tree models

- The goal is to split the predictor space into regions $R_1 ... R_j$ such that that the resulting residual sum of square (RSS) is minimized.
- We cannot try out every conceivable way of splitting the predictor space! Therefore, we have to use a **top-down greedy approach**: at each point in the tree-building process, the best split is made. By "best", we mean the split that would result in the minimal RSS. It is "greedy" because only the best split at a particular step is made. This is in contrast to a forward-looking process in which a split is chosen so that it will result in a better split that further reduces the RSS in some future step.
- We need to decide which predictor P to split and at which cut-off value s to split. After making the first split, for all subsequent splits, we look for the best predictor and cut-off value to split a region defined in the previous step rather than the entire predictor space. We repeat this process until a stopping criterion is reached (ex: a minimum number of train observations in each region).
- Decision tree suffers from **overfitting**: if I have a large tree (split the predictor space into many small regions), the model would make very good predictions for the train observations (with low bias); but it will not make accurate predictions for test observations that were not used to train the model. There is a **bias-variance trade-off** that we have to make.
- We can add **pruning** to the tree-building process. First, we "grow" the tree as large as possible (until a stopping criterion is met). Second, we prune the tree back to obtain a "parsimonious" subtree with the lowest RSS through a process called **cost complexity pruning**. Essentially, the pruning process gives the "simplest" subtree with the smallest number of terminal nodes (leaves) and the minimal RSS.
- Although pruning reduces overfitting, but it also faces some serious problem of its own--more on this in the next section.

How do we make predictions using a trained decision tree model?

Suppose we have a set of training observations of 10,000 students with the following variables:

- Dependent Variable (also called Response Variable): *Test score (Y)*
- Independent Variables (also called Predictors):
  - *Family income (X_1)*
  - *Distance between home and school (X_2)*

Using the process described in the previous section, we train a decision tree model. We want to use the trained decision tree model to predict the test score for a test observation (a student that does not belong in the train dataset), given his/her family income and distance between home and school.

As shown in Figure 1, the predictor space $\{Distance, Income\}$ is segmented into 3 non-overlapping regions (the splitting rule comes from the trained decision tree model). For a test observation (X_1^*,X_2^*), represented by the black dot in Figure 1, we predict his/her test score ($\hat{Y}$) with the mean test score of R1 $\bar{Y}_1$ because (X_1^*,X_2^*) belongs to R1.

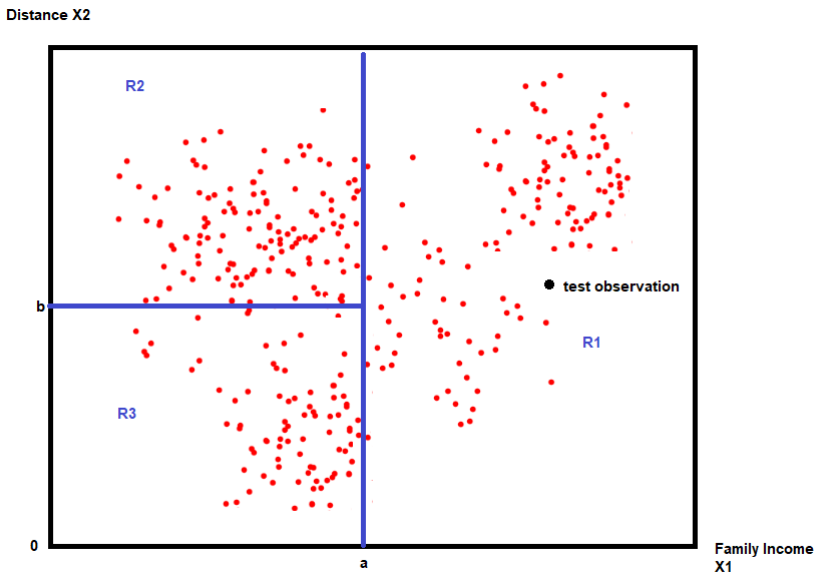

Figure 1: The predictor space is segmented into regions R1, R2, and R3. Since the test observation (black dot) belongs to region R1, its predicted ($\hat{Y}$) is the mean response value of all train observations (red dots) of region R1.

The regression tree model can be represented by a tree diagram in Figure 2. The tree consists of a series of splitting decisions (nodes and branches). We first split the predictor space into two branches. All train observations with $X_1 < a$ is assigned the left branch, and the remainder is assigned the right branch. Second, we further split the observations with $X_1 < a$ according to their X_2 values. There are three "leaves" or terminal nodes in this model. The number assigned to each terminal node is the mean test score of the training observations in regions R1 to R3.

Given the family income and distance of a test observation, we determine which region in the predictor space that it belongs to (R1, R2, or R3). If it belongs to R1, then the predicted response value (test score) for this individual would be 85.

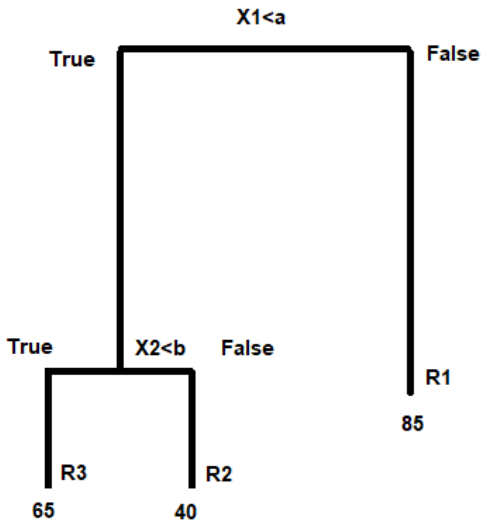

Figure 2: Tree diagram of the regression tree. If a test observation falls into $R1$, then his/her predicted test score is 85.

Fact sheet of the Decision Tree model

Use cases	Regression and classification problems
Decision Tree vs. regression models	- If the underlying relationship between X and Y is linear, then linear regression outperforms trees. Otherwise, tree-based models might outperform regression. - Tree model can be easily visualized and explained to the non-technical members of your team.
Pros and Cons of tree-based models	Pros: Easy to explain, can handle discrete predictors without dummy variables./> Cons: - Does not have the same level of predictive accuracy as linear regression (need bagging and boosting). - Suffers from high variance-- a small change in data leads to large changes in the tree model. - Need to tune hyperparameters (ex: depth of tree and other "settings" of the model are decided by the user (rather than estimated from the data by the algorithm).

2. Bootstrap Aggregation (Bagging)

- Bagging is a technique that can improve the predictive accuracy of a number of statistical learning methods. Here, we will apply bagging to decision tree models.
- Decision trees suffer from high variance. This means if the train data changes, the predictions made by the model will be very different. For example, we randomly split a dataset (10,000 observations) into two halves: train set and test set. We train a decision tree with train set and test with the test set. The predictions made by the model will be very different if the dataset is split in a different way.
- In contrast, a linear regression model has low variance. This is because if a different train dataset is used to train the model, the model would make similar predictions (since only the slope of the linear regression line will change a bit).
- However, there is a **bias-variance tradeoff**: a complex nonlinear model that fits the train observations very well have low bias but high variance (because the model will change a lot if different train observations are used). A simple linear regression model might not fit the train observations as well (high bias), but compared to a nonlinear model, it wouldn't change a lot if we use different train datasets (the line will only steepen or flatten a bit). So you can either have a model with low bias but high variance, or a model with high bias but low variance.

Can we get the best of both worlds?

- **Answer:** Yes, we can use a process called bagging (bootstrap aggregation). It is designed to reduce the variance of large tree models that have low bias.
- **Fact:** Averaging a set of independent observations reduces variance.
- To see why, suppose we have n independent random variables ($z_1 \ldots z_n$), each with variance σ^2 . We defined $\bar{z}$ as the arithmetic mean of $z_1 \ldots z_n$. The variance of $\bar{z}$ is σ^2/n . So, the variance of the average of a large set of independent variables is low.

- **The ideal case**: We go back to our previous example where we try to predict student test scores with their family income and their home-school distances. Let's say I go around and collected the test scores, family income, and distance from school of 10,000 students from $B = 400$ schools. In other words, I have 400 separate train datasets, each with 10,000 observations. Using the 400 datasets, I train 400 decision tree models. If I have a test observation (information about the family income and distance of a student called "Cory" who does not belong any of the 400 schools), I can get 400 predicted test scores of Cory using the 400 trained models ($\hat{Y^{(1)}}$,$\hat{Y^{(2)}}$, ..., $\hat{Y^{(400)}}$) . And if I take the average of the 400 predicted test scores, I get a **final low variance predicted test score** of Cory: $\hat{Y}$ = mean ($\hat{Y^{(1)}}$,$\hat{Y^{(2)}}$, ..., $\hat{Y^{(400)}}$)
- **Bootstrap:** What if I don't have 400 separate train datasets? What if I can only collect data from a single school (due to time and cost)? In this case, we can artificially generate 399 other datasets (each with 10,000 students) by **sampling with replacement**. That is, with my single dataset (imagine it as a bag of 10,000 balls), I randomly pick an observation (a ball), record its test score, family income, and home-school distance, and then put the ball back into the bag. I repeat this process until I have 10,000 "students" in my bootstrapped dataset. Note that this dataset might not have unique students (since a student can be picked multiple times). The main idea is that this bootstrapped dataset can *approximate* the actual dataset that I would have had if I have a separate and independent set of student data from another school. By training 400 models with my 400 datasets (399 of which are bootstrapped), I can aggregate the 400 predicted test scores (by averaging) and generate a low variance predicted test score for a test observation such as Cory. This overall process is called bootstrap aggregation (bagging).

Applying Bagging to Decision Trees

- We construct **B** number of decision trees using **B** bootstrapped train datasets (we called these "bagged trees"). To minimize bias, we **don't** prune the bagged trees; instead, we allow the trees to grow so that each tree has many terminal nodes (ex: their predictor space is segmented into many small regions). Hence, each tree has low bias but high variance. To take care of the high variance problem, we apply bagging: we use B decision trees to get B predicted test scores for a test observation . Then we aggregate the predictions by averaging and we end up with a single low-bias, low-variance predicted test score.
- For classification problems, obviously we cannot aggregate the B predicted response values by taking their average. Instead, we form our final classification by taking the *majority vote* of the B predictions. For example, I want to predict whether a restaurant is good or bad (2 classes) given some information about the restaurant (chef's experience, price of dishes, etc.). I have B predictions (good or bad) from my bootstrapped models. My final classification would be the most frequently occurring class amongst the B predictions.
- Bagging improves prediction accuracy at the cost of interpretability since we can no longer identify which variables are the most important in predicting the response.
- However, we can compute "variable importance" of a bagged tree by recording the total reduction of RSS due to the splits over a given predictor for each bagged tree, then we take the average over the B trees. A large value of "variable importance" indicates an important predictor.

3. Random Forests

- Recall our earlier example: if we have n **independent** random variables ($z_1 \dots z_n$), each with identical variance σ^2, then Var($\bar{z}$) = σ^2/n

- However, if ($z_1 ... z_n$) are not correlated, then the variance of $\bar{z}$ will be higher. In the n=2 case, we have:

  $Var(\bar{z})$ = $\sigma^2/2$ + cov(z_1,z_2). Thus, variance of $\bar{z}$ will be higher because of the extra covariance term.

- Bagged trees are not independent from each other because the trees were built on bootstrapped training sample. For this reason, the variance of its predicted response can be further reduced if we can somehow "decorrelate" the bagged trees. That is exactly what the Random Forest Model does

- Like before, the Random Forest model consists of a series of bagged trees. However, when building a bagged tree using bootstrapped data, each time a split in the tree is made, only m out of the total n predictors are considered. For example, I want to predict the test scores with n=5 predictors: family income, parent's education, home-school distance, IQ score, and student-teacher ratio. In my bagged tree, whenever the algorithm decides on a split, it will only look at a random subset of m=3 predictors (ex: IQ score, parent's education, family income). It might pick a different set of predictors in a subsequent split.
  - **Why is this useful in decorrelating the bagged trees?** Suppose family income is a very important predictor for test score. In other words, when a splitting decision is made, almost every bagged tree will select family income since it will result in a large reduction in RSS. For this reason, "family income" would appear in every bagged tree (it will likely be selected as the first predictor for splitting). In this case, every bagged tree will be similar and therefore will be highly correlated.
  - By randomly selecting a subset of m predictors to consider at every split, $(p-m)/p$ predictors will be excluded from consideration (and this might include "family income"). This ensure other predictors that has moderate association with test score will be considered and selected in the splitting decision. In sum, this process ensures that a "richer" variety of bagged trees will be generated and therefore less correlation between the trees in a Random Forest Model. As a rule of thumb, we set $m = \sqrt{p}$
  - Since the bagged trees are decorrelated, we will end up having a smaller variance in the predicated response variable from the Random Forest Model. For this reason, the Random Forest have a lower test error (higher predictive accuracy of test observations).

Fact sheet of the Random Forest Model

Use cases	**Regression and Classification problems**
Difference between Random Forest and Decision Trees	- Lower test error than decision trees due to lower bias and variance in the model.
Pros and Cons	- Higher predictive accuracy than both linear regression and decision trees. - Low interpretability, cannot be visualized using a decision tree diagram. Harder to explain to non-technical team members. - Computationally expensive, especially if cross-validation is used to build each individual decision tree in the Random Forest model. - Have to tune additional hyperparameters such as number of trees in the forest.

4. Gradient Boosting

- Like bagging, boosting is a general approach that can be applied to many statistical learning approaches to improve their performances. Its main feature is that the model can "learn from its past errors" to improve its performance. In gradient boosting, a series of models are built where each model takes into account the error from the previous model. Thus, error is reduced sequentially.
- We start by developing "model #1" to predict the response variable from the independent variables. Whatever prediction error that is left from "model # 1" will be subsequently modelled by a second "model # 2" , and so on. We repeat this process k times.
- In the end, then models 1, 2, ... k will be aggregated and we get a final model. This is an example of **Ensemble Modelling** in which many "weak learners" are combined to form a "strong learner".
- All this seems a bit abstract, let's be more specific and look deeper into how gradient boosting can be applied to decision trees.

Details of gradient boost

- Given a dataset (ex: the characteristics of 10,000 students), I fit a decision tree model (we call this model #1) to the data and get:
  - The predicted response values (ex: test score) for each observation $\hat{Y_i^{m1}}$ (the superscript m1 denotes the model number). Recall that in a decision tree, if student i falls into region X, then the predicted response value of student i is simply the average test score of all students in region X.
  - The residual of each observation r_i^{m1} = $(Y_i - \hat{Y_i^{m1}})$
- Now we build another decision tree model. For the second decision tree model (we call it model #2), instead of using Y (test score) as our response variable, we will now use the residual of model #1 r^{m1} as the response variable. In other words, we want to develop a second decision tree model that generates predictions that are as close to the residuals of model #1 as possible.
- **Why do we do this?** Imagine hypothetically that we developed a model #2 such that it *exactly* predicts the residuals r^{m1}. In this case, r^{m1} = $\hat{Y^{m2}}$ -- we model the residuals of model #1 exactly. If we then combine model #1 and model #2 by summing the predicted response values of the two models: $\hat{Y}$ = $\hat{Y^{m1}}$ + $\hat{Y^{m2}}$, the resulting $\hat{Y}$ ill exactly equal to the actual Y (with zero residuals).
- Of course, in real life this rarely happens. Instead, there will likely be differences between r^{m1} and $\hat{Y^{m2}}$. We will do two things:
  - Update the predicted response value: $\hat{Y}$ = $\hat{Y^{m1}}$ + $\hat{Y^{m2}}$
  - Update the residuals: r^{m2} = r^{m1} - $\hat{Y^{m2}}$
- Now we will build another decision model (model #3) for the residuals r_i^{m2} . We continue this process k number of times (k is usually specified by the user). At each iteration, the residuals of the model r_i will be reduced incrementally.
- In the end we get a final predicted response value that has a small residual:
  - $\hat{Y}$ = $\hat{Y^{m1}}$ + $\hat{Y^{m2}}$ + ... + $\hat{Y^{mk}}$
- There are two popular machine learning methods that are widely used in the data community are:

- XGBoost: basically, same as gradient boost but with a few additional tweaks to speed up implementation.
- AdaBoost: Instead of setting residuals as the response variable for models 2,...,k, we modify the observations that are used to train decision trees 2...k based on the residuals of the previous tree. For more details, take a look at these links:
  - https://www.youtube.com/watch?v=ErDgauqnTHk
  - https://www.youtube.com/watch?v=LsK-xG1cLYA

Fact sheet of Boosted Trees

Use cases	**Regression and Classification problems**
Difference between Random Forest and Decision Trees	- Lower test error than Random Forest.
Pros and Cons	- High predictive accuracy, faster training time than random forest. - No interpretability since we primarily focus on modelling the error. - Have to tune additional hyperparameters such as number of trees to build, the learning rate, and number of splits for each tree.