DATA SCIENCE II: Machine Learning MTH 9899 Baruch College

Lecture 3: Decision Trees and Gradient Descent

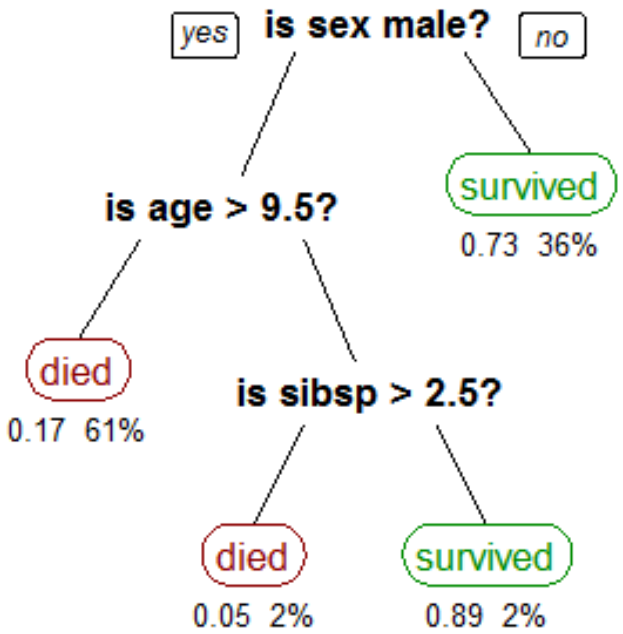
Adrian Sisser

April 25, 2018

Outline

Decision Trees

Decision Trees are a form of supervised ML that seek to build a simple set of decision rules to make predictions. They consist of a series of decisions based on the data attributes, and end in a leaf.

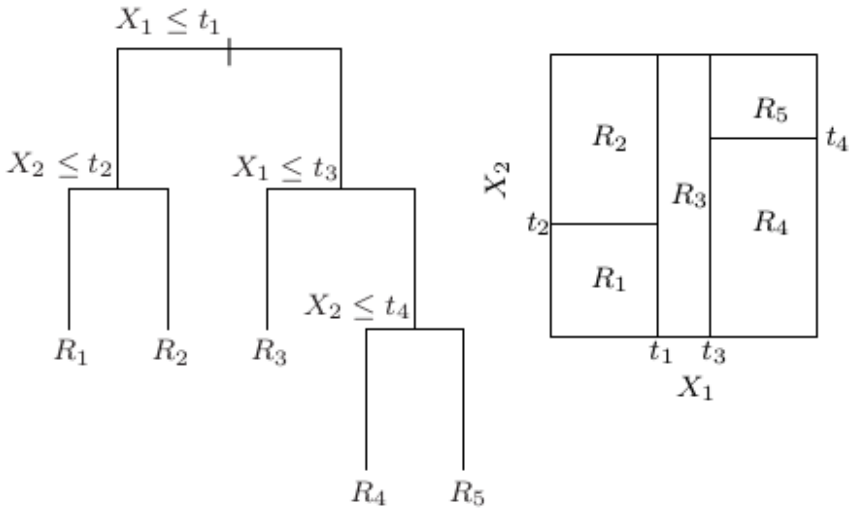


Types of Trees

- Classification Trees are used to predict what class a piece of data belongs to. The output can be a predicted class or a probability mass function across the different classes.
- Regression Trees estimate a continuous variable for the output. Traditionally, Regression Trees give a point estimate, which is the average of all of the points in a given leaf. This effectively gives us a piecewise constant fit.
- Model Trees are a form of regression tree, but instead of having a constant prediction, each leaf consists of a fitted model. This could be any of the other models we have talked about, including: LSQ, LASSO, Ridge, Logistic, etc.
- One important point about model trees is that they you can have much shallower trees since internal structure can be captured by the model.

What can Decision Trees Do?

Decision trees partition the data up into complex regions, allowing us to find features in the data that we might not be able to otherwise.



ID3 Algorithm

ID3 is one of the original decision tree algorithms. It is limited to taking discrete inputs and classifying discrete outputs. It's very primitive compared to modern algos, but still insightful to understand.

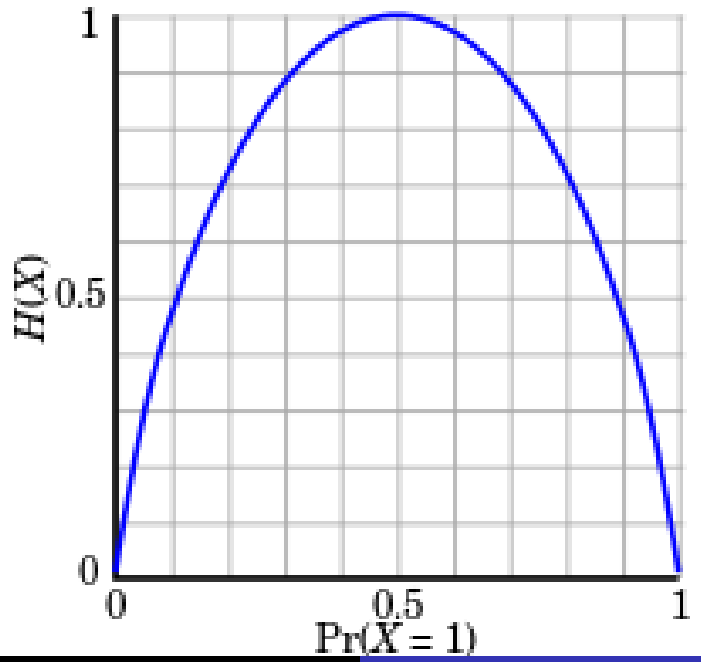
ID3 Algorithm

function BUILDTREE(S, A)
    $a \leftarrow$ FINDBESTATTRIBUTE(S, A)
    if a is Null **then**
        return LEAF(S) ▷ The leaf summarizes the class(es) in S
    end if
    $\{S_i\} \leftarrow$ SPLIT(S, a) ▷ Create subsets of S based on each value of a
    $A \leftarrow A - a$ ▷ We won't split on a again
    $N \leftarrow$ SPLITNODE(a)
    for $a_i \in a$ **do** ▷ a_i are the distinct values in a
        $N[a_i] \leftarrow$ BUILDTREE(S_i, A)
    end for
    return N
end function

ID3 - Finding the Best Split

ID3 chooses it's splits by looking at Entropy and Information Gain. Entropy is a measure of the 'disorder' of a variable and is defined as:

$$H(X) = \sum_i -p(x_i)\log_2 p(x_i)$$



ID3 - Finding the Best Split

Our goal is to find a split that best reduces the Entropy in the subsets vs the original set. With ID3 we can end up with more than 2 splits. We define Information Gain as the decrease in Entropy when we split the input S into subsets, which we will try to maximize:

$$IG(S,a) = H(S) - \sum_i \frac{|S_i|}{|S|} H(S_i)$$

ID3 - Limitations

- Limited to Classification
- We can only handle attributes with a relatively small set of discrete values. What happens when the number of possible values gets too high?
- Any ordering information in categorical variables is lost, ie if you bin up a continuous variable, you lose most if it's value.
- We are using a **greedy** search to find split points, which doesn't lead to an optimal tree. **This is common to almost all tree algorithms**

C4.5

C4.5 is a successor to ID3. Major changes in C4.5 are:

- Continuous attributes can be handled by looking at all possible values and we compute the information gain by splitting at this value. What are the drawbacks of this?
- We deal with the issues mentioned on the last slide, namely that too many possible values in a discrete variable create a bias. We do this by using Gain Ratio instead of Information Gain:

$$GainRatio(S,a) = \frac{IG(S,a)}{H(a)}$$

C4.5

Another major change from ID3 to C4.5 is *pruning*, or removing nodes from the tree that add complexity, but lead to overfitting. For C4.5, we look at the observed error rate, $\hat{r}$ in a given branch of the tree. We then calculate a confidence interval around what the true rate, r should be, at a given significance, z:

$$r_{upper} = \hat{r} + z\sqrt{\frac{\hat{r}(1-\hat{r})}{N}}$$

We calculate r_{upper} for replacing a given branch with a leaf, and compare it to average weighted r_{upper} of the current branch leafs. If the r_{upper} for the leaf itself is less than the average, we prune.

CART

CART (Classification And Regression Trees) is alternative to ID3/C4.5 which extends to include regression. CART uses a new metric for Classification, GINI:

$$G(S) = \sum_{i=1}^{K} \frac{|S_i|}{|S|}(1 - \frac{|S_i|}{|S|})$$

We can compute $G(S)$ for each attribute a, and choose the attribute that minimizes $G(S)$.
For regression, we also need a metric. CART chooses Variance Reduction:

$$VR(S) = \operatorname{var} S - \sum_{i=0}^{K} \frac{|S_i|}{|S|} \operatorname{var} S_i$$

CART - Pruning

CART also uses a different method for pruning. First, let's define a cost of complexity measure (for regression):

$$T \quad \text{The set of leaf nodes in a given tree}$$

$$C_\alpha = \sum_{i=1}^{|T|} \sum_{j=1}^{|T_i|} (y_j - \mu_i)^2 + \alpha |T|$$

We can now go through and build a series of trees, starting from the full tree down to one node, using a process called *weakest-link pruning*. When we do that, we will get a series of optimal trees for different values of α. Now, we know a set of optimal trees for each α, how do we pick what α to use?

Model Trees

So far, the models we've looked at have all had very simple point estimates in each leaf. It seems like we're leaving something on the table. A *Model Tree* is a regression tree, where each leaf contains some form of model rather than a simple point estimate.

Model Trees

By now, we know that the 2 main things we'll have to do are:

- Splitting - Come up with an efficient way to figure out which attribute (and where within that attribute) to split.
- Pruning (or Stopping) - How can we keep trees sizes reasonable sizes to avoid overfitting.

Model Trees - Splitting

Before, we looked at Variance Reduction as a useful metric for regression cases. We'll do the same, but using the variance of the residuals from the regression:

$$VR(S) = \text{var}\, \epsilon_S - \sum_{i=0}^{K} \frac{|S_i|}{|S|} \text{var}\, \epsilon_{S_i}$$

For a continuous attribute, we have to worry about ways to quickly calculate this, we can choose to do a simple line search. If we do this along with some smart online regression techniques, we can make it fast enough to work. There are other techniques, such as the fluctuation test, but they're quite involved and we won't go into more detail.
For pruning, we can look at analogous cost-complexity measures.

Model Trees

Some caveats and thoughts on Model Trees:

- Can we stop early? Look at the adjusted R^2 and see if we have a real noticeable improvement.
- Bayesian Shrinkage of 'small leaves'
- How to pick out 'fringe' values

Ensembles are collections of predictors, where we hope to gain value by having multiple predictors with minimal correlation, then combining them in a smart way.

- Single-Model ensembles are where we build multiple of the same type of model, ie a Regression Forest.
- Multi-Model ensembles are where we are combining different types of learners together.
- Yet another approach is Boosting, where we build an ensemble serially, focusing on errors from previous models.

Predicting from an Ensemble

How do we combine the members of an ensemble?

- The simplest way for regression is to average them.
- For classification, we can use the votes as a PDF.
- It can often be helpful to build a 'predictor' on top of the predictions from an ensemble - this is called Stacking and is very popular.

Introduction to Random Forests

A Random Forest is an ensemble of Trees.

- If we hope to add value by adding more trees, we need to reduce the correlation between the trees.
- By combining lots of uncorrelated trees, we can reduce the variance of the predictor.

Our goal is to build trees with as little correlation as possible. How?

Bagging

One way to create uncorrelated predictors is **Bagging** - **B**ootstrap **Agg**regation.

Bootstrapping This is a common resampling technique in statistics where we generate random samples of our input with replacement. By generating new samples with replacement, we generate datasets with as little overlap as possible. It can be shown that if you choose N points from a dataset of size N with replacement, you end up with $1 - \frac{1}{e} = 0.632$ of your input points in the output - the rest are duplicates.

Aggregation This refers to the fact that we are going to fit models to different Boostrapped samples, and aggregate them together to form a better model.

Bagging

How many trees do we need? Where B is the size of the ensemble,

$$\begin{aligned} \bar{\hat{y}} &= \frac{1}{B} \sum_{i=1 \ldots N} \hat{y}_i \\ \sigma^2(\bar{\hat{y}}) &= \rho \sigma^2_{\hat{y}_i} + \frac{1-\rho}{B} \sigma^2_{\hat{y}_i} \end{aligned}$$

As an aside, this applies to much more than random forests. It can apply to heterogeneous ensembles as well.

Reducing Correlation

What else can we do to make our trees less correlated?

- Variable Selection - We only test a random subset of variables at each split point. This is VERY effective. How many should we test? Some popular choices are $\sqrt{f}$ of $f/3$ for classification/regression.
- Probabilistically - We can pick which variable to split on randomly, in proportion to the improvement they offer.

Extremely Randomized Trees

Extra Trees (or Extremely Randomized Trees) take things a step further. For Extra Trees, we choose the split point for each variable randomly.
This takes things even further, and decreases the correlation between trees in the forest.

Extra Trees are often considered the best "off the shelf" learner for many problems.

Expectations for R^2

Should out-of-sample R^2 be positive? Not if we're talking about a poor predictor.

$$R^2 = 1 - \frac{\text{Var}[y - \hat{y}]}{\text{Var}[y]}$$

If we have random predictors and the out-of-sample x and y are independent, what is the variance of the residual?

$$\text{Var}[y - \hat{y}] = \text{Var}[y] + \text{Var}[\hat{y}] - 2\text{Cov}[y, \hat{y}]$$

Since they are independent, the Cov will be very small and the R^2 will be negative.

A Quick Review

Last class, we introduced Backpropagation:

$$w'_{l,ij} = w_{l,ij} - \eta \frac{\partial E}{\partial w_{l,ij}}$$

Now that we have a reasonable way to update weights, we have to figure out the most efficient way to do that.

Basic of Optimization

We are trying to find the parameter set, Θ (including weights, biases, any other 'meta params' you might define), that minimizes the total error. Our error surface is FAR from convex. This means we are usually going to finding **local** optima - we have to be wary of optimization techniques that don't introduce enough variance to help find better local optima.

Basic Gradient Descent

The most basic form of Gradient Descent involves calculating the gradient of the error with respect to the entire dataset, but this has a few drawbacks:

- If your data size is too big, it might not work at all, ie won't fit in memory.
- It can be incredibly inefficient. Your using all of your points to calculate a gradient, but you might have been able to figure out the step direction much faster.
- By using all the data to calculate each step, we have very little variance in our step directions. This makes finding a global optimum harder.

Stochastic Gradient Descent

Stochastic Gradient Descent (SGD) attempts to avoid the problems from before by calculating the gradient and updating the params on a row by row. We will choose the order randomly, thus the term stochastic.

- By using one row at a time, an outlier can cause a step in a wildly different direction than you want to be going in.
- Convergence take a long time
- The high variance of the step directions makes it easier to get out of local optima
- It's inefficient - computers are very good at performing bulk computations (especially GPUs) - operating on 1 row at a time is inherently slow.

Minibatches

Let's do the the obvious choice: Mix the 2.
Minibatches refers to training on randomly batches of the randomly shuffled data.

- By choosing a good batch size we can manage a trade-off between variance (ie the ability to get out of local minima) and the speed of training
- Choosing batch size isn't easy, usually we use a few hundred examples at a time.

Note that in practice, when people say SGD they usually mean SGD with Minibatches.

Overview

A wide variety of trick and techniques have been developed to help speed up convergence in a few ways:

- Managing the learning rate
- Introducing a 'Memory' of recent steps
- Heterogeneous steps across different features.

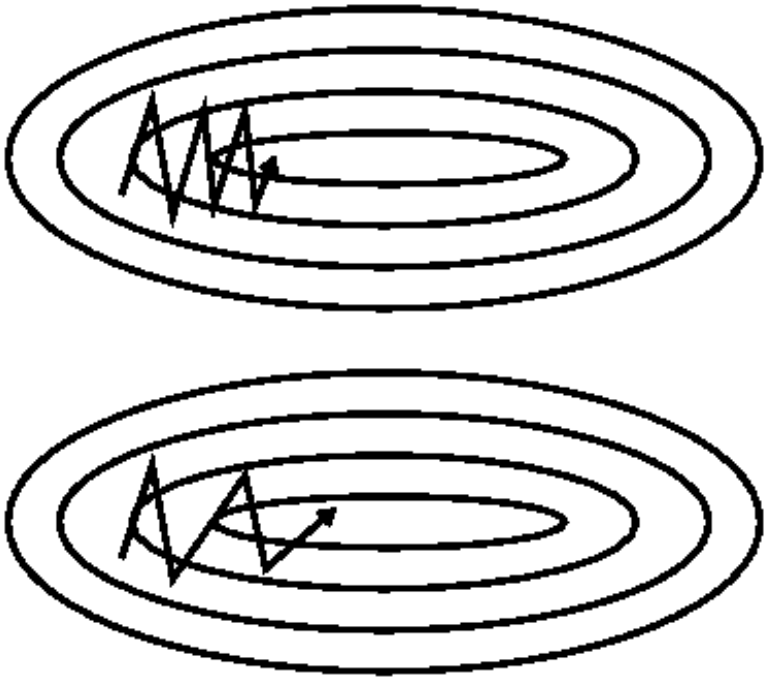
Our original update rule was:

$$\theta_{i+1} = \theta_i - \eta \frac{\partial E}{\partial \theta_i}$$

With Momentum, we blend the update for this step with the update from a previous step:

$$\theta_{i+1} = \theta_i - \eta \frac{\partial E}{\partial \theta_i} + \gamma(\theta_i - \theta_{i-1})$$

By doing this, we avoid the unnecessary variance induced by asymmetric error surfaces.



[1] Source: http://sebastianruder.com/optimizing-gradient-descent/index.html

Nesterov Momentum

Nesterov Momentum is based off a simple observation about the basic momentum algorithm. From basic momentum, we know:

$$\theta_{i+1} = \theta_i - \eta \frac{\partial E}{\partial \theta_i} + \gamma(\theta_i - \theta_{i-1})$$

This means that we already know a large part of the update for the next step, so we can calculate the gradient starting from there, instead of the current weights:

$$\theta_{i+1} = \theta_i - \eta \frac{\partial E(\theta_i + \gamma(\theta_i - \theta_{i-1}))}{\partial \theta_i} + \gamma(\theta_i - \theta_{i-1})$$