Decision Trees

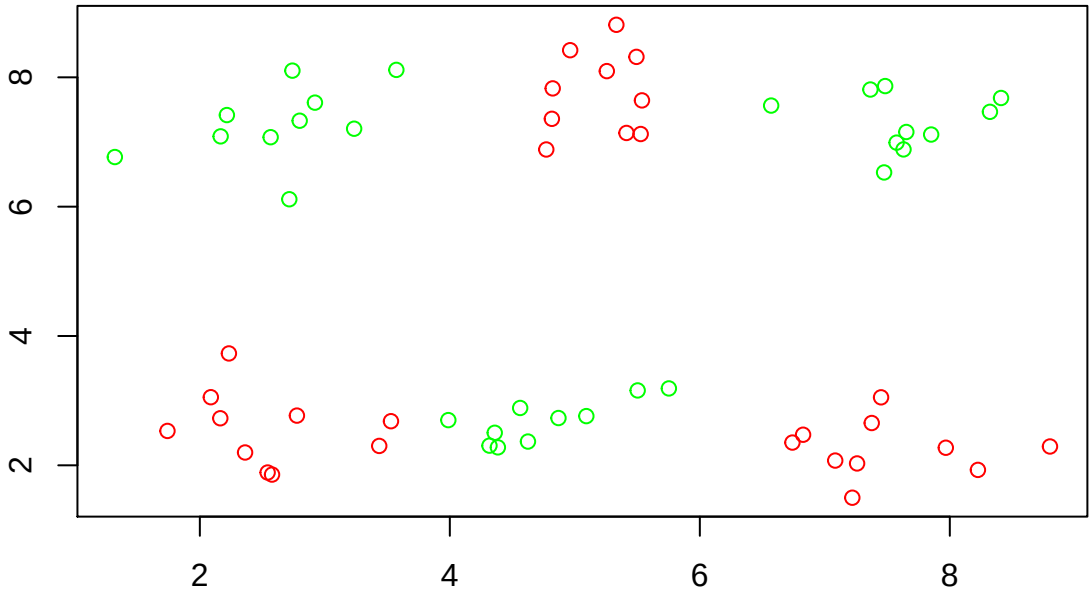
D2K Course Staff

3/27/2019

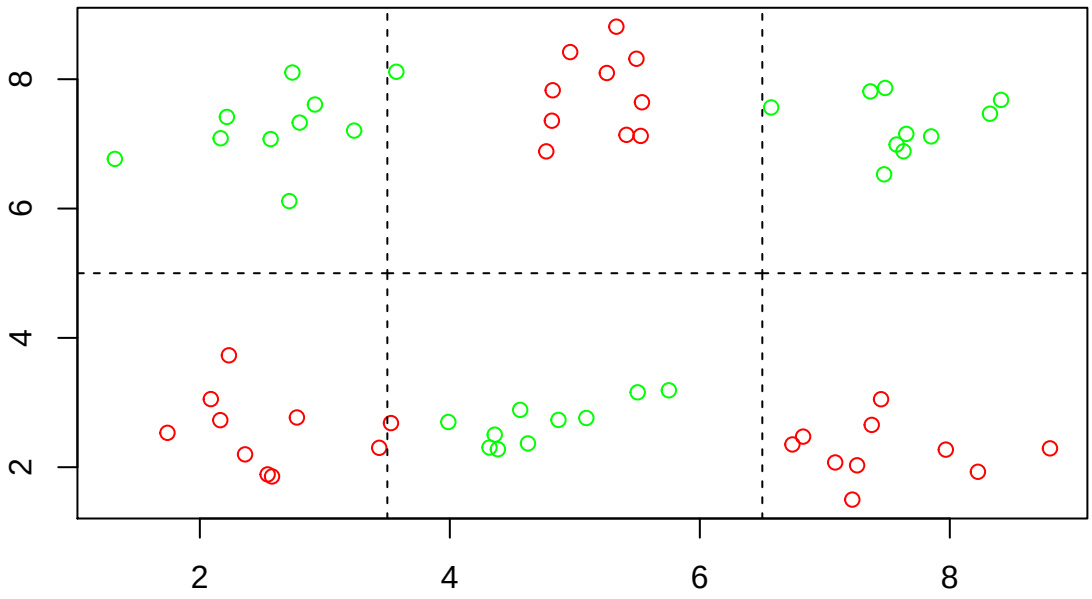
Introduction

Up until this point, the methods we have covered have been *linear* methods. We have been able to achieve some non-linear decision boundaries through kernelization, or a transformation of the linear function. However, what if we decide we don't want to limit ourselves to linear formulations in order to perform prediction tasks? Is there a chance we can get better predictions if we choose to not restrict ourselves to these archaic constraints?

Let us, for example, look at a scatter plot of two classes "red" and "green".



Clearly, there is distinct separation between the two classes. However, we cannot achieve this separation using any one linear function. But, what if we split the data into different regions like this?



We can see that we can certainly separate our data into distinct classes if we allow ourselves to partition the data into multiple regions. We can express this principle mathematically by dividing the input space X into disjoint subspaces or regions where we define

$$X = \bigcup_{i=0}^{n} R_i$$
$$s.t.\ R_i \cap R_j = \emptyset\ for\ i \neq j$$

where n is a positive integer.

Splitting the Data

When building decision trees, we use a *greedy* algorithm, meaning that we optimize for a local solution first and foremost. Secondly, the algorithm is *recursive*, meaning that we continue to partition the data on top of the previous partitions. The main idea behind building decision trees is to partition the data by determing a threshold for a variable, and then splitting the data on that threshold. Each split of the data on a particular variable then produces two child regions from the parent region

$$R_1 = \{X | X_j < t, X \in R_p\}$$
$$R_2 = \{X | X_j \geq t, X \in R_p\}$$

where R_1 and R_2 are the two child regions of the parent region X, which for the first split is the input space. We then perform all subsequent splits of the data on child regions of the data, this is the part of the process that is **recursive**. The number of splits we make on the data is referred to as the **depth** of the tree.

Loss Functions for Decision Trees

As with all predictive algorithms, we seek to minimize error in our predictions. In order to do so, we must decide what thresholds we will be splitting our data into regions by determining which values *maximizes* our *decrease* in loss. Mathematically, we represent it as

$$L(R_p) - \frac{|R_1|L(R_1) + |R_2|L(R_2)}{|R_1| + |R_2|}$$

We use decrease in loss for our metric as the decision tree algorithms are **recursive** and the loss in a child region is a product of the parent region.

Classification

Decision trees can be used for both classification and regression tasks, however, we will first cover common loss functions used for classification tasks. Let us first note that for any given region, R, we can define $\hat{p}_c$ to be the proportion of data points that are of class c.

Misclassification Loss

We define misclassification loss to be

$$L_{misclass}(R) = 1 - \max_c(\hat{p}_c)$$

where we define the loss to be the proportion of data points that would be misclassified if we were to guess the majority class for all data points in that region. Even though this is a good loss metric to report the final predictive accuracy of a model, it is not sensitive to changes in class probabilities during the training process and is a *bad* choice for growing a decision tree. Often, we use other loss metrics during the training process in order to achieve better final predictive accuracy.

Cross-Entropy Loss

A common loss metric for decision tress is cross-entropy loss. This is a popular choice for training decision trees as it is more sensitive to changes in class probabilities. We can define cross-entropy loss as

$$L_{cross}(R) = -\Sigma_c \hat{p}_c log_2 \hat{p}_c$$

where the logarithm function is of base 2. The advantage of using cross-entropy loss as compared to misclassification loss is that the former is a convex function. For purposes of optimization, this makes it far easier to use.

Gini Loss

Another loss metric that is commonly used for decision trees is Gini Loss, or Gini Impurity. It is closely related to cross-entropy loss and is also a convex function. Mathematically, it is expressed as the following:

$$L_{Gini} = \Sigma_c \hat{p}_c(1 - \hat{p}_c) = 1 - \Sigma_c \hat{p_c}^2$$

We can think about this metric as how often a data point would be misclassified if it was randomly classified given the probability distributions of the data point in the region.

Regression

As we mentioned previously, the decision tree algorithm can be extended to regression tasks as well as classification tasks. Although it is not used in this context as commonly, it is still a perfectly valid method to experiment for regression tasks.

Squared Loss

When growing decision trees for regression, we still perform feature splitting to generate child regions from parent regions. *But*, our prediction for a particular region is the average of the values $\hat{y} = \frac{\Sigma_{i \in R} y_i}{|R|}$. We define squared loss to be the function,

$$L_{squared}(R) = \frac{\Sigma_{i \in R}(y_i - \hat{y})^2}{|R|}$$

which closely resembles the MSE loss function commonly used in regression.

Example in R

Below, we will show how to grow and train a decision tree in R using the `tree` package on the `iris` data set. First, we load the data and the required packages for our analysis.

```
# Load libraries and data
library(tree)
data(iris)
summary(iris)
```

```
  Sepal.Length    Sepal.Width     Petal.Length    Petal.Width
 Min.   :4.300   Min.   :2.000   Min.   :1.000   Min.   :0.100
 1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.600   1st Qu.:0.300
 Median :5.800   Median :3.000   Median :4.350   Median :1.300
 Mean   :5.843   Mean   :3.057   Mean   :3.758   Mean   :1.199
 3rd Qu.:6.400   3rd Qu.:3.300   3rd Qu.:5.100   3rd Qu.:1.800
 Max.   :7.900   Max.   :4.400   Max.   :6.900   Max.   :2.500
       Species
 setosa    :50
 versicolor:50
 virginica :50
```

Before training our model, we of course need to split the data into training and a test set. For this module, we will use the query set to compare the unpruned and pruned decision trees.

```
# Set seed for reproducibility
set.seed(123)

# Create training/test index
tr_idx <- sample(1:nrow(iris), nrow(iris) * 0.6, replace = FALSE)

# Create training/validation set
iris_train <- iris[tr_idx, ]
```

```r
iris_test <- iris[-tr_idx, ]

# Create query set from test set
qr_idx <- sample(1:nrow(iris_test), nrow(iris_test) * 0.5, replace = FALSE)

iris_query <- iris_test[qr_idx, ]

iris_test <- iris_test[-qr_idx, ]
```

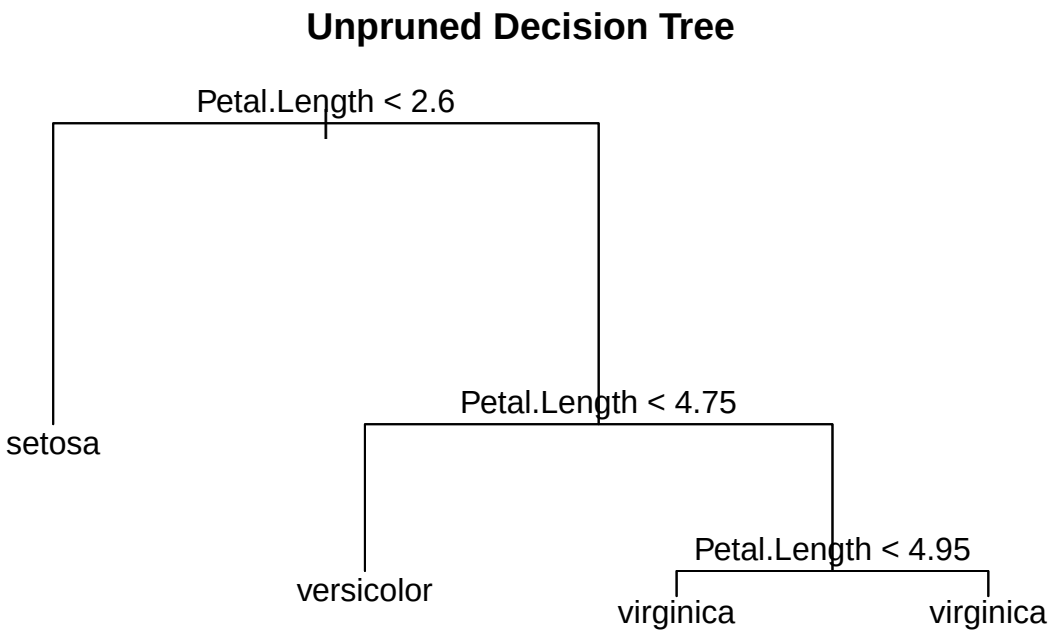
Now that we have chunked our data, we need to grow our decision tree on the training data before evaluating it on the unseen test data. One thing to note about decision trees is that when *they often have high variance*. This is partially a function of how deep the tree is grown, but it is incredibly easy to overfit to the training data with this method. Thus, it is very important that we use cross validation to tune the tree to its optimal depth before predicting on our test data.

First, we will formulate the model and plot what it looks before we prune it with cross-validation.

```r
# Set seed for reproducibility
set.seed(123)

# Build decision tree without cross-validation
tree_model <- tree(Species ~ ., data = iris_train)

# Plot decision tree model
plot(tree_model)
text(tree_model, pretty = 0)
title(main = "Unpruned Decision Tree")
```



Next, we can compute the prediction error of the unpruned decision tree on the query set to evaluate the model's performance.

```r
# Predict on query set
tree_pred <- predict(tree_model, iris_query, type = "class")

# Compute misclassification error
```

```
unpruned_error <- 1 - mean(tree_pred == iris_query$Species)
unpruned_error
```

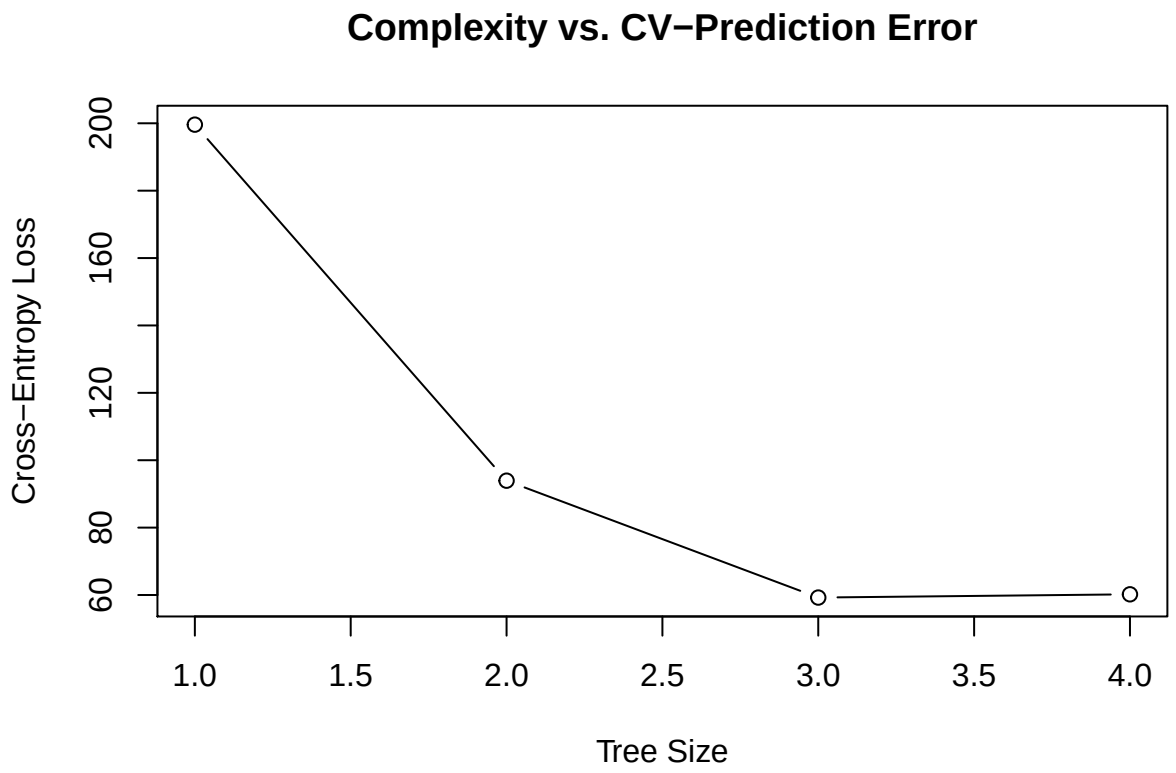
```
[1] 0.06666667
```

We can see that even without pruning our decision tree, our model performs quite well on new data. This is not particularly surprising since we know that the `iris` data set is easily separable. Now that we have built a basic decision tree model, we can go back and "prune" the tree using cross-validation to find the optimal depth and features to split the data on.

```
# Set seed for reproducibility
set.seed(123)

# Tune model using cross-validation
cv_tree = cv.tree(tree_model, rand = c(1:10))

# Plot complexity vs. prediction error
plot(cv_tree$size, cv_tree$dev, type = "b", xlab = "Tree Size",
    ylab = "Cross-Entropy Loss")
title(main = "Complexity vs. CV-Prediction Error")
```



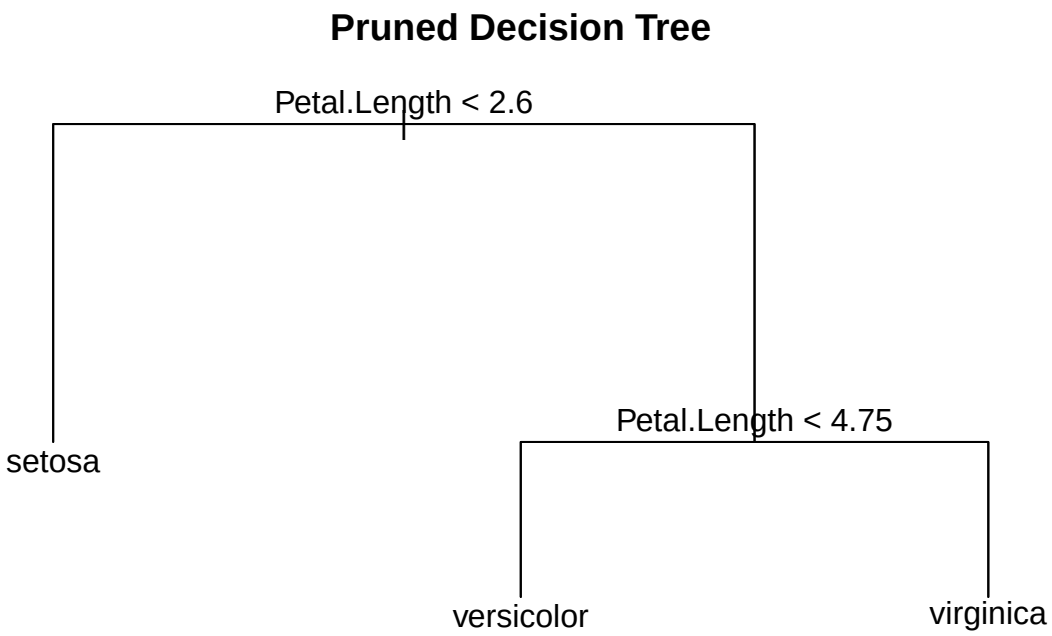
Now we can prune the tree to the optimal depth of 3 splits and plot our pruned decision tree.

```
# Set seed for reproducibility
set.seed(123)

# Prune decision tree model
pruned_tree <- prune.tree(tree_model, best = 3, method = "deviance")

# Plot decision tree model
plot(pruned_tree)
```

```r
text(pruned_tree, pretty = 0)
title(main = "Pruned Decision Tree")
```



We can now report the prediction error on the query set of our pruned tree and see how it compares to the unpruned tree.

```r
# Predict on query set
pruned_tree_pred <- predict(pruned_tree, iris_query, type = "class")

# Compute misclassification error
pruned_error <- 1 - mean(pruned_tree_pred == iris_query$Species)
pruned_error
```

```
[1] 0.06666667
```

Well, our pruned tree predicts with the exact same accuracy as our unpruned tree. Generally, this will not be the case. However, this data set is simple enough that it is logical that pruning would not have a significant effect on the quality of the model.

Finally, we can report the accuracy of our pruned tree model on the test set.

```r
# Predict on test set
test_pred = predict(pruned_tree, iris_test, type = "class")

# Compute misclassification error
test_error <- 1 - mean(test_pred == iris_test$Species)
test_error
```

```
[1] 0.06666667
```

Conclusion and Final Thoughts

Decision trees have lots of clear benefits as compared to linear models and other predictive models, but also some major drawbacks. It is important to look at the pros and cons of any model when handling prediction tasks.

Pros

- Highly interpretable
- Works well with non-linear data
- Can be used to determine decision rules in applied settings

Cons

- Lots of variance
- Works poorly with linear data
- Often don't predict very well by themselves

Even though there are clear and strong downsides to decision trees for predictive modeling, lots of these cons can be mitigated through techniques like ensembling which we will touch on in other modules.

References

Le, James. "Decision Trees in R." DataCamp Community, 19 June 2018, www.datacamp.com/community/tutorials/decision-trees-R.

Townshend, Raphael John Lamarre. "Decision Trees." CS229 Lecture Notes, Stanford University, cs229.stanford.edu/notes/cs229-notes-dt.pdf.