Decision Trees

Introduction

Decision trees are a tree-structured model used for classification in machine learning problems. Each leaf node corresponds to a prediction, and each internal node partitions the data in multiple sets based on the value of an input variable.

A decision tree partitions the feature space to make each partition have as little class uncertainty as possible.

Example

Consider the following example. We have a feature space $\mathbb{R}^2$, where each feature is of the form $x = \{x_1, x_2\}$. The root node asks if $x_1 < t_1$. If the condition is true, we go left, otherwise we go right. The left child of the root asks if $x_2 < t_2$. The right child of the root asks if $x_2 < t_3$. In this way, we partition a 2-dimensional space into rectangles. More generally, we are perfoming axis-parallel splits in the $\mathbb{R}^d$ input space.

For classification problems, we store the counts of classes at each leaf node, and draw from this distribution to get a prediction. For regression problems, we store the mean response in each region and return this as the prediction.

Building the tree

```
function build_tree(current_samples):
    If current_samples are all of the same class, return

    For each feature, j:
        Find the information gain from splitting on j (or the Gini index)

    Create a decision node that splits on feature j* with greatest information gain
    Add sublists obtained by splitting on j* as children of new node

    build_tree(left_sublist)
    build_tree(right_sublist)
```

Evaluating splits

How do we measure information gain? Given a list of training samples, we want to find on which feature we should split the space. At a given leaf, we can fit a multinoulli model to the data at the leaf, $\mathcal{D}$, using the counts of the classes:

$$\hat{\pi}_c = \frac{1}{|\mathcal{D}|} \sum_{i \in \mathcal{D}} \mathbb{I}(y_i = c)$$

Here, $\hat{\pi}_c$ is the probably of class c at the given node.

Let's calculate the entropy of the data. This is a measure of the disorder of the data. If all the data were of the same class, we would have the logarithm of 1 which is 0. The higher the entropy, the more disorderly the data. Again, if we can C classes, the entropy of the data is:

$$\mathbb{H}(\hat{\pi}) = -\sum_{c=1}^{C} \hat{\pi}_c \log \hat{\pi}_c$$

Information gain is related to entropy by the following formula:

$$IG(X, Y) = \mathbb{H}(Y) - \mathbb{H}(Y|X)$$

This is the information gain from X on Y.

We can also compute something called the Gini Index, which is the expected error rate.

$$G = \sum_{c=1}^{C} \hat{\pi}_c(1 - \hat{\pi}_c) = 1 - \sum_{c} \hat{\pi}_c^2$$

So, we want to decision a split on the data that minimizes the Gini score. We evaluate splits based on their ability to minimize the amount of mix in the subsequent 2 groups that are created. This weighted average is given by:

$$G_{\text{weighted ave}} = \frac{i}{m} G_{left} + \frac{m-i}{m} G_{right}$$

where i is the number of samples that are partitioned to the left of the node, of a total of m samples at the parent node.

CART Algorithm

To build the decision tree, we need an algorithm: CART (classification and regression trees). CART chooses splits at nodes to minimize the *average of the Gini index of the children weighted by their size.*

Now, this is a large space to search: we must iterate through every feature, and every possible split on that feature. Since features can be continuous, this might seem like an infinite space. However, all we need to do is consider (for a given feature) every midpoint between adjacent values. The naive solution does the following:

```
for every feature, j
    sort the values for this feature
    for every midpoint between adjacent values
        compute the Gini index for the samples on the left
        compute the Gini index for the samples on the right
        compute the weighted average of the above values, weighted by the number of samples in each spli

    if this is the best split so far, update a champion variable
```

Demonstration

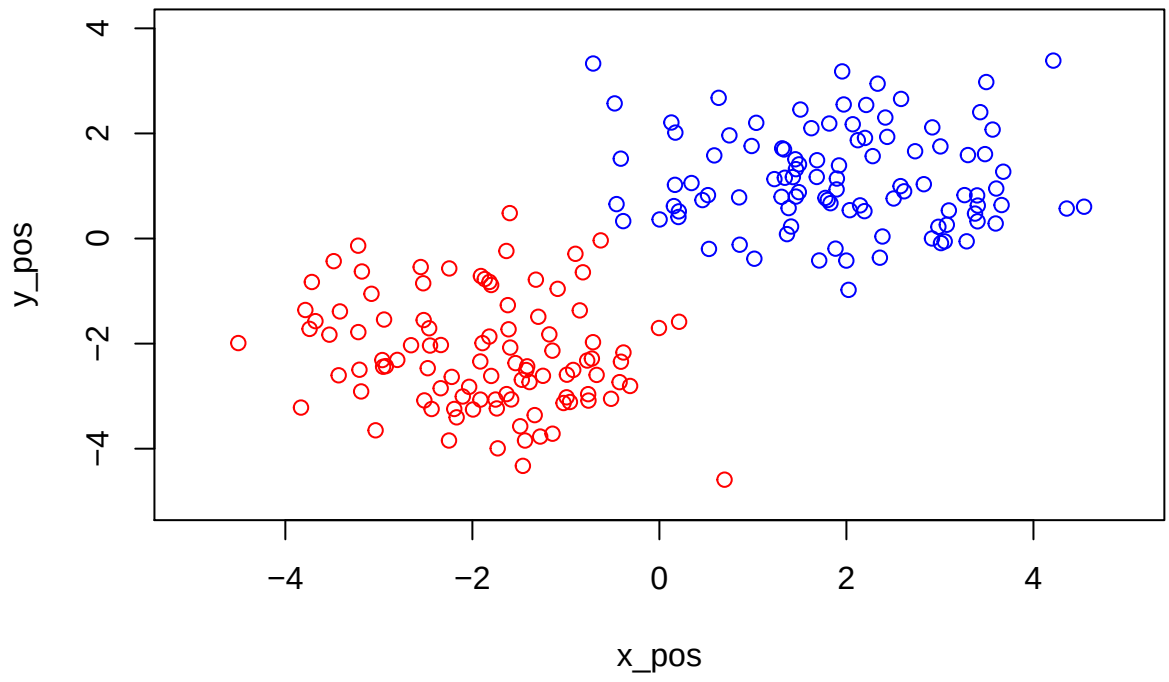
Let's generate some 2-dimensional data and try to create one entropy-minimizing split.

```
NUM_SAMPLES = 200

# Generate points of two classes.
x_pos = rnorm(NUM_SAMPLES/2, mean=2, sd=1)
y_pos = rnorm(NUM_SAMPLES/2, mean=1, sd=1)
x_neg = rnorm(NUM_SAMPLES/2, mean=-2, sd=1)
y_neg = rnorm(NUM_SAMPLES/2, mean=-2, sd=1)

xs = c(x_pos, x_neg)
ys = c(y_pos, y_neg)
data = cbind(xs, ys)
labels = c(rep(1, times=NUM_SAMPLES/2), rep(0, times=NUM_SAMPLES/2))

# Plot points.
plot(x_pos, y_pos, col="blue", xlim=c(-5,5), ylim=c(-5,4))
points(x_neg, y_neg, col="red")
```




```
champ_feature = -1
champ_split = 1e10
champ_gini = 1e10

for (j in 1:2) {

  # Sort feature values.
  sorted = sort(data[,j])

  # For every midpoint, compute the Gini index for both halves of the split.
  midpoints = rollmean(sorted, 2)
  for (m in midpoints) {

    # Tally the class counts on the left and right.
    lpos = 0
    lneg = 0
    rpos = 0
    rneg = 0

    for (i in 1:NUM_SAMPLES) {
      value = data[i,j]
      if (value < m) {
        if (labels[i] == 1) {
          lpos = lpos + 1
        } else {
          lneg = lneg + 1
```

```
        }
      } else {
        if (labels[i] == 1) {
          rpos = rpos + 1
        } else {
          rneg = rneg + 1
        }
      }
    }

    left_total = lpos + lneg
    right_total = rpos + rneg
    gini_left = 1 - (lneg / left_total)^2 * (lpos / left_total)^2
    gini_right = 1 - (rneg / right_total)^2 * (rpos / right_total)^2
    gini = gini_left * (left_total / NUM_SAMPLES) + gini_right * (right_total / NUM_SAMPLES)

    # See if this split performs better than anything seen so far.
    if (gini < champ_gini) {
      champ_gini = gini
      champ_feature = j
      champ_split = m
    }
  }
}
```

What was the entropy-minimizing split?

```
print(champ_feature)
```

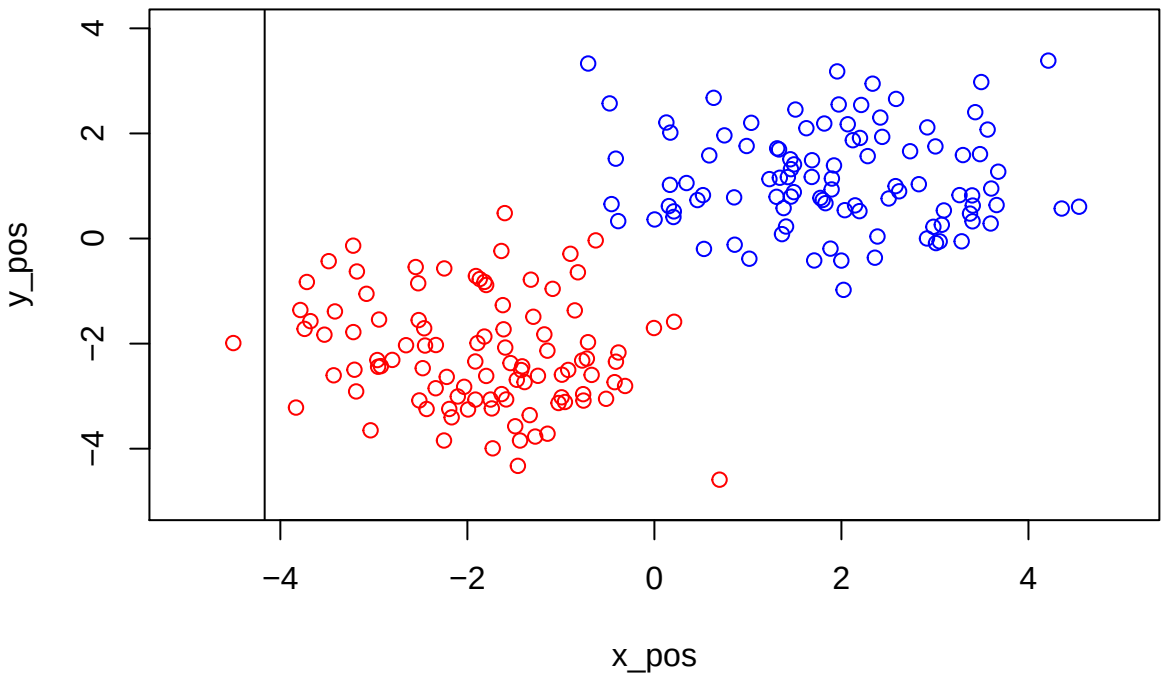
```
## [1] 1
```

```
print(champ_split)
```

```
## [1] -4.167888
```

Let's see what this split actually looks like.

```
plot(x_pos, y_pos, col="blue", xlim=c(-5,5), ylim=c(-5,4))
points(x_neg, y_neg, col="red")
abline(v=champ_split)
```


y_pos
4
2
0
−2
−4
−4
−2
0
2
4
x_pos