Decision Trees: An Information Theoretic Perspective

Jack Hart

In this document I am attempting to show a new(ish) interpretation of decision trees for classification from an information theory framework. First I will describe a basic decision tree performing binary classification from this perspective. Next I will extend this to a problem and compare it's results to using a KL-Divergence.

The Traditional Process

Assume we are looking at a simple decision tree splitting on one variable, X, for binary classification. The goal of a decision tree for algorithms such as C4.5 or ID3 is to split the data into two nodes that create the "pursest" split, achieved through minimizing entropy. This is ususally done by itterating over the ordered values of X and calculating the the weighted average of the entropy of each split.

I'll demonstrate here that the entropy of a decision tree for each node is equal to the entropy of a Bernoulli distribution. This is due to the fact that the definition of entropy extends simply to a Bernoulli random variable. But furthermore, the weighted averages across all splits in the data can be interpreted as part of the caluclation for conditional entropy of two bernoulli random variables.

X is sampled from two class distributions of usually unknown shape and generating proccess. Let's assume we take an iid sample of X of size n. Even if X is continuous, an approximation of the sample space can be the number of k unique elements sampled of X: $\mathcal{X} = \{x_1, .., x_l\}^k$. The probabilities of each of these elements can be approximated by their frequency of the data: $\mathcal{P} = \{p_1, .., p_l\}^k$.

When training the decision tree, you will split the data k times across an ordered $\mathcal{X}$ sample space. Each of the k possible splits can be viewed as a bernoulli processes with each split s having a $p_s = \sum_{i=1}^{s} p_i$. For any given split, there is a resulting p_{s1} for another bernoulli process, representing the probability of class 1. Therefore, for a given split, we can see that the entropy is equivalent the conditional entropy of these two processes:

$$H(Y|split) = \sum_{s \in split} p_s H(Y|split = s), split = \{p_0, p_1\}$$

$$H(Y|split) = -\sum_{s \in split} p_s * p_{s0} log(p_{s0}) - p_{s1} log(p_{s1})), split = \{p_0, p_1\}$$

The entropy of each of the k splits are calculated and then the minimum is chosen. This proccess is equivalent to how current decision trees are trained. However, I've never seen it explained as conditional entropy **perhaps this is interesting?**

The following function models a basic decision tree for classification.

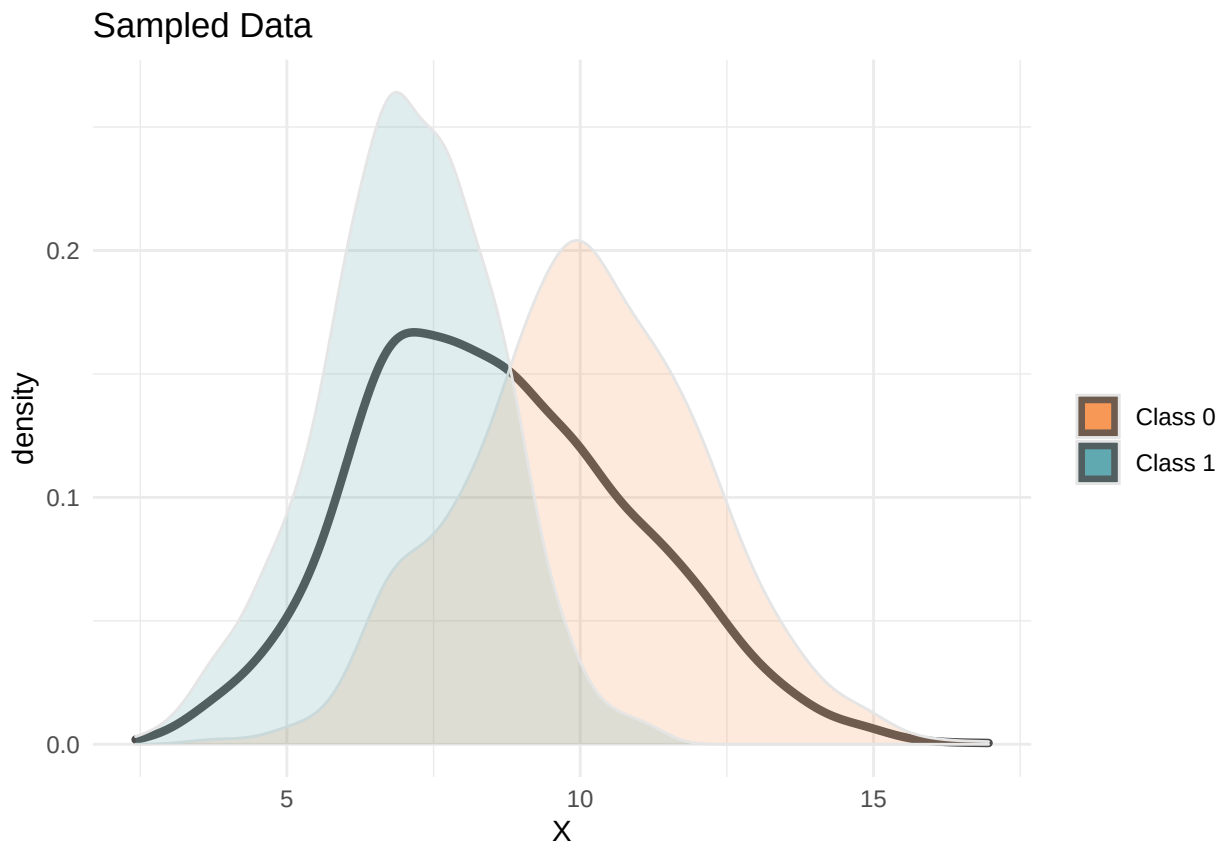
```
find_minimum_entropy <- function(df){

  # default values
  threshold <- NA
  best_entropy_left <- 1000
  best_entropy_right <- 1000
  best_average_entropy <- 1000

  # order data by X
  df_ordered <- df %>% arrange(X)
```

```
  for(i in 1:(nrow(df_ordered)-1)){
    # subset data
    left_y <- df_ordered$Y[1:i]
    right_y <- df_ordered$Y[(i+1):nrow(df_ordered)]

    # find class probabilites
    probabilities_left <-prop.table(table(left_y))
    probabilities_right <-prop.table(table(right_y))

    prob_x_left <- i / nrow(df_ordered)
    prob_x_right <- (nrow(df_ordered) - i) / nrow(df_ordered)

    # calculate conditional entropy
    entropy_left <- prob_x_left * (-1*sum(probabilities_left*log2(probabilities_left)))
    entropy_right <- prob_x_right * (-1*sum(probabilities_right*log2(probabilities_right)))

    entropy_split = entropy_left + entropy_right
    # weighted average of entropy
    #  entropy_split = ( (length(left_y) * entropy_left) + (length(right_y) * entropy_right) ) / (lengt

    left_ent_lst <<- c(left_ent_lst, entropy_left)
    right_ent_lst <<- c(right_ent_lst, entropy_right)
    total_ent_lst <<- c(total_ent_lst, entropy_split)

    if(entropy_split < best_average_entropy){
      threshold <- df_ordered$X[i]
      best_entropy_left <- entropy_left
      best_entropy_right <- entropy_right
      best_average_entropy <- entropy_split
    }
  }
  return(c(threshold, best_entropy_left, best_entropy_right,best_average_entropy))
}
```

Well Behaved Data Generating Process

For simplicity, this examples will have an equal representation of classes in the sample of X. Here is a plot of the dataset we will be classifying and the two underlying class distributions.

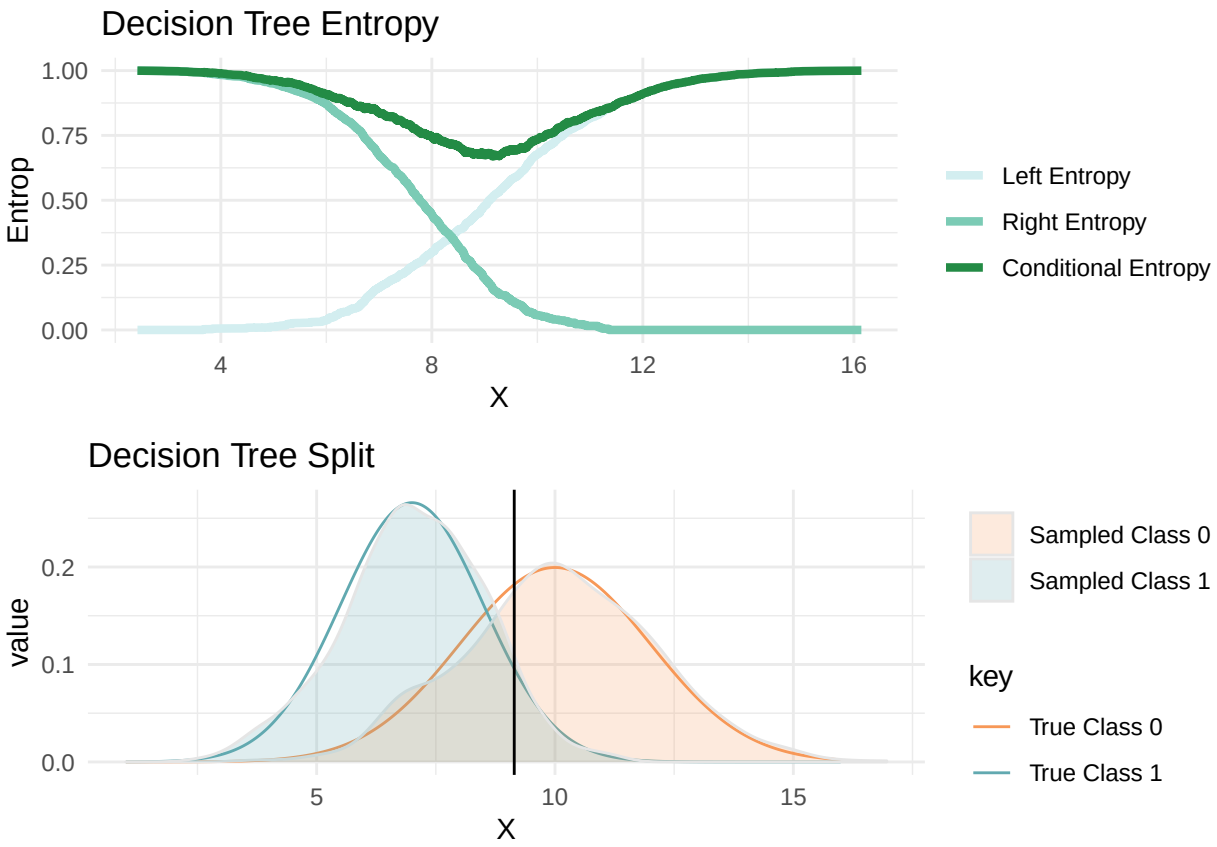
```
# sample data
set.seed(83)
x1 <- rnorm(1000, mean = 10, sd = 2)
x2 <- rnorm(1000, mean = 7, sd = 1.5)
y <- c(replicate(1000, 0), replicate(1000,1) )

# what the data looks like
coul <- colorRampPalette(brewer.pal(4, "Spectral") )(8)[c(3,7)]
df <- data.frame("X" = c(x1, x2), "Y" = y)
ggplot(data = df) + geom_density(aes(x=X), color="grey30", size=1.5) +
  geom_density(aes(x=X, group = Y, fill = as.factor(Y)), color = "grey90", alpha = .2)  +
  theme_minimal() + scale_fill_manual(label=c("Class 0", "Class 1"), values = coul) +
  labs(title="Sampled Data", fill="")
```



Next, we'll implement the tree and plot the resulting conditional entropy. Conditional entropy was minimized with an X vlaue of about 9.144. Plotting this on the sample data we see it doesn't perfectly get the optimal point, but it's close.

```
# global variables because I'm lazy
left_ent_lst <<- c()
right_ent_lst <<- c()
total_ent_lst <<- c()

# find best split by itterating over data -- returns some basic metrics
tree_values <- find_minimum_entropy(df)
tree_values
```

```
## [1] 9.1448525 0.5100138 0.1594013 0.6694151
```



Unbalanced Data Generating Process / Classes

Are Decision trees biased by unbalanced classes?

[Haven't looked into this yet. My hunch is yes.]

Decision Trees – An alternative Proccess

Now, what if instead of the normal process, we used the properties of KL-Divergence. This process will instead use KL-divergence to find the split distribution of classes that maximizes the differences in the class distributions.

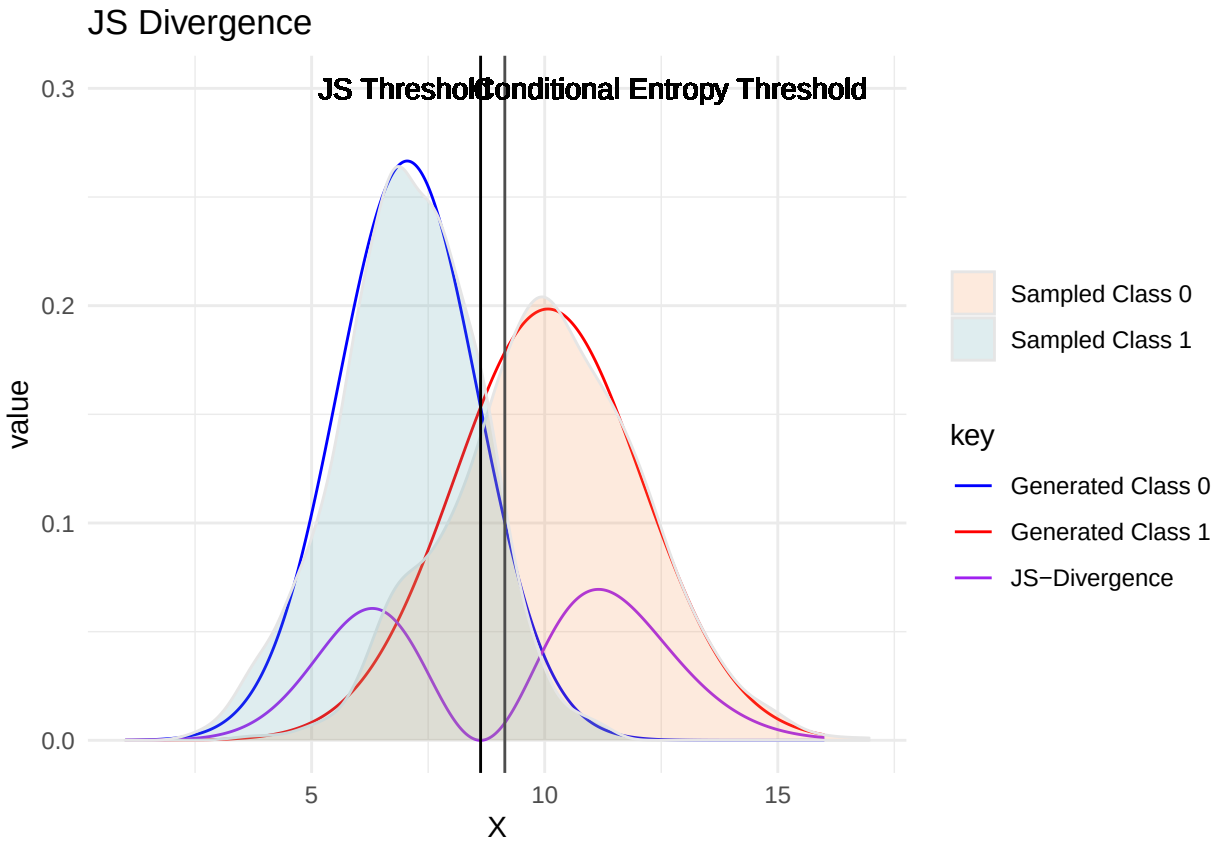
Well Behaved Data Generating Process

We will use the same simple dataset as before. If we know the underlying distribution, this process is very straightforward. We can find the optimal split by calculating the sufficient statistics for the class distributions and then the subsiquent divercences.

Since the data that an even distribution of classes, we can use the **Jensen-Shannon Divergence**. From what I can tell, this is a symetrized KL-divergence. And, at least for this example, its minimum is the maximum divergence between class distributions.

The following code caluclates the JS-Divergence when we know the underlying distribution. In this case, given our approximations are correct, the minimum JS-Divergence does a much better job at classifying the data than the original decision tree with conditional entropy did.

```r
# calculate sample statistics
mean_1 <- sum(df$X * df$Y) / sum(df$Y)
mean_2 <- sum(df$X * (df$Y ==0) ) / sum(df$Y == 0)

sd_1 <- sqrt(sum( (df$X-mean_1)^2 * df$Y) / sum(df$Y) )
sd_2 <- sqrt(sum( (df$X-mean_2)^2 * (df$Y ==0) ) / sum((df$Y ==0) ))

D_KL <- dnorm(seq(1,16, .005),  mean=mean_1, sd=sd_1) * log2( dnorm(seq(1,16, .005),  mean=mean_1, sd=sd
D_KL2 <- dnorm(seq(1,16, .005),  mean=mean_2, sd=sd_2)* log2( dnorm(seq(1,16, .005),  mean=mean_2, sd=sd

#JS Div
JS_div <- 0.5 * D_KL + 0.5 * D_KL2
```



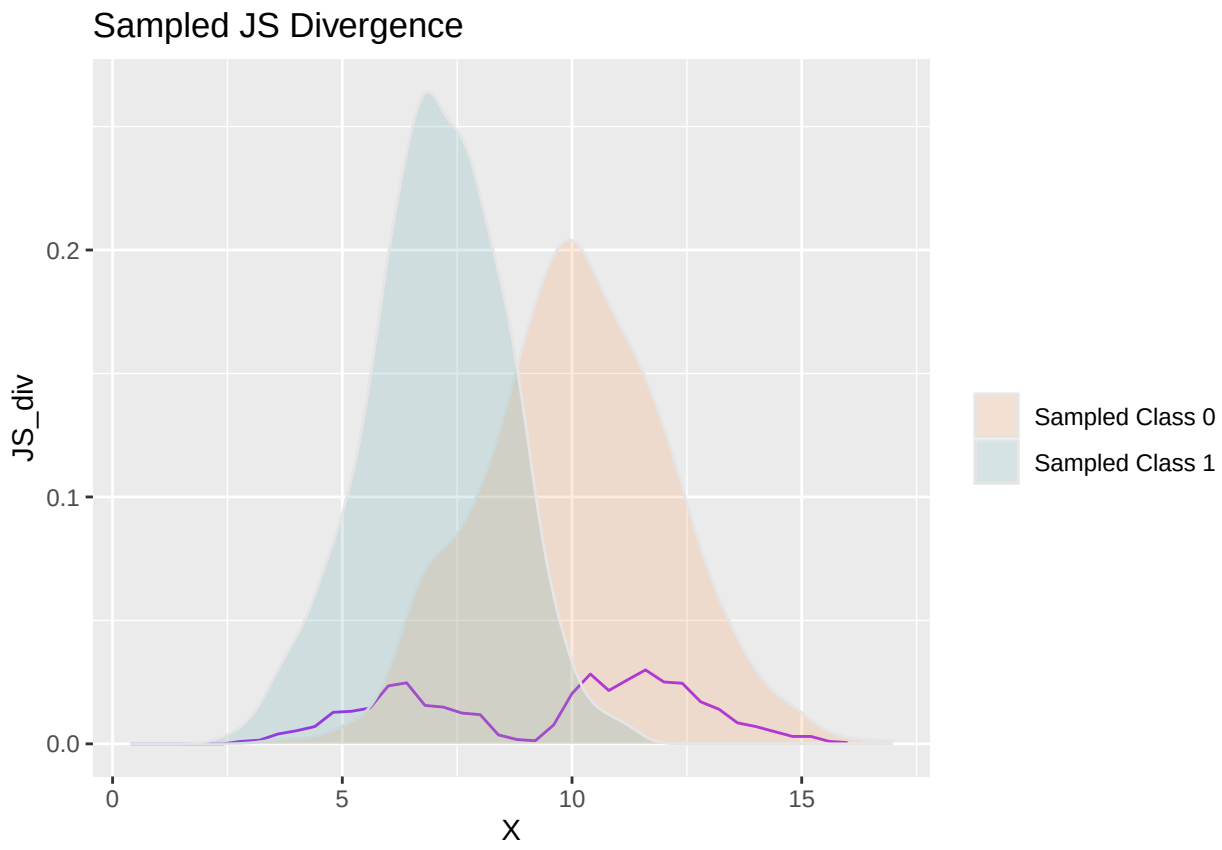
Proposition

We could drop our assumption of the shape of the data and calculate an approximation of the JS-Divergence with the data. This will have to be done by binning the sampled X values and then using the subsiquent class probabilites of each bin to approximate the JS-Divergences. The optimal bin was found to be the one between 8.8 and 9.2. Which is higher (and farther off) than the original JS-Divergence. It also contains the threshold from the traditional process.

```r
bins = seq(0,16,by=.4)
discrete_1_x <- cut(df_ordered$X[df_ordered$Y == 1], breaks=bins, right = FALSE)
dicrete_0_x <- cut(df_ordered$X[df_ordered$Y == 0], breaks=bins, right = FALSE)
```

```r
probs_1 <- unname(prop.table(table(discrete_1_x)))
probs_0 <- unname(prop.table(table(dicrete_0_x)))

# calculate the divergence
x_vals <- data.frame("X" = bins[-1],
           'prob_1' = probs_1,
           'prob_0' = probs_0) %>% dplyr::select(X, prob_1.Freq, prob_0.Freq)

D_KL <- x_vals$prob_1.Freq * log2( x_vals$prob_1.Freq / ( (x_vals$prob_1.Freq + x_vals$prob_0.Freq) / 2)
D_KL[is.na(D_KL)] <- 0.0 # there are some divide by zero errors
D_KL2 <- x_vals$prob_0.Freq * log2( x_vals$prob_0.Freq / ( (x_vals$prob_1.Freq + x_vals$prob_0.Freq) / 2
D_KL2[is.na(D_KL2)] <- 0.0

JS_div <- 0.5 * D_KL + 0.5 * D_KL2

# optimal bin selected
names(prop.table(table(discrete_1_x)))[23]
```

```
## [1] "[8.8,9.2)"
```



Unbalanced Data Generating Process / Classes

Are Decision trees biased by unbalanced classes?

[Haven't looked into this yet. My hunch is no.]

My questions

1. Can I find a theoretical connection between traditional decision trees using conditional entropy and using KL divergence?

- This I think is the most important question? The equations for conditional entropy and Divergence look really similar. I'm wondering if there's an underlying relationship, or maybe this is my wishfull thinking.

2. Is there a structured way I can prove/explain that using the traditional multi-noulli + Bernoulli conditional entropy calculations is more flexible? Is it? Is it better at generalizing when the data generating function is unknown? i.e. makes less assumptions.
3. How can I add the complexity of this process beyond binary classification, one continuous varibale, ect.

- I think have more than one continuous varibale would mean I would calculate the KL-Divergence of multivariate distributions.