Announcements

- Make sure to sign in on the google form
- Pset 8 due November 11 at 5 pm
- Midterm 2 November 11 through November 18
- No section next week or the week after
- Last section on November 29.

Decision Tree Algorithms

Consider a prediction problem where we want to predict Y_i from a single X_i. Recall the heuristic we used to describe how a decision tree decides where to split a node: consider many possible split points x from $\min(\{X_i : i \in 1, ...n\})$ to $\max(\{X_i : i \in 1, ...n\})$ and calculate the SSE for each

$$\sum_{i:X_i<x}(Y_i - \bar{Y}_{\text{left}})^2 + \sum_{i:X_i \geq x}(Y_i - \bar{Y}_{\text{right}})^2$$

where $\bar{Y}_{\text{left}}$ is the mean of the Y_i for i such that $X_i < x$ (and analogously for $\bar{Y}_{\text{right}}$). Then, choose the x that minimizes this SSE. This is a nice story, but the implementation is a bit more clever.

1. Consider an algorithm that did what was described above, and suppose that it divided the interval from $\min(\{X_i : i \in 1, ...n\})$ to $\max(X_i : i \in 1, ...n\})$ into k equal segments. Give a reason this would perform suboptimally when $n < k$ and a different reason it would perform suboptimally when $n > k$. How many additions, subtractions, multiplications, and divisions (where each operation only involves two numbers) would be required to choose the best split? (Note that becuase we're only concerned with the algorithm here, we'll treat both the X_i and Y_i as fixed.)

When $n < k$, we are considering too many splits: there will be multiple points evaluated that give the same SSE, so we are wasting computation. When $n > k$, some intervals will contain multiple X_i, so we are missing splits that could potentially be better than any we've considered. With this method, to calculate the SSE for a single split point, we would need $n-2$ additions and 2 divisions to calculate the group means, and we would need n subtractions, n multiplications, and $n-1$ additions. Total, we would need $k(2n-3)$ additions, kn subtractions, kn multiplications, and $2k$ divisions to evaluate all the split points.

2. A better idea would be to only consider our observed X_i (or the average between consecutive X_i) as potential split points. How many additions, subtractions, multiplications, and divisions would be required to choose the best split now? (Assume that at least one data point is in the "left" group and at least one is in the "right" group.)

There are now $n-1$ split points we have to consider, so we can apply the results from above to show that we would need $(n-1)(2n-3)$ additions, $(n-1)n$ subtractions, $(n-1)n$ multiplications, and $2(n-1)$ divisions to evaluate all the split points. Note that if $k < n-1$ in the last problem, it would have taken fewer operations, but it might have missed the best split.

3. We can still do better! Let L_s (left sum) be the sum of the Y_i for the smallest s X_i, and let R_s (right sum) be the sum of the Y_i for the rest of the data points. Let L'_s and R'_s be the corresponding means. Show that the SSE can be written in terms of a constant (a piece that doesn't change with s) and a piece that only depends on L'_s, R'_s, and s (and the constants n and $\bar{Y}$). Next, find a simple equation for L_{s+1}, R_{s+1}, L'_{s+1}, and R'_{s+1} in terms of L_s and R_s. Finally, determine how many additions, subtractions, multiplications, and divisions would be required to choose the best split. Hint for the first part: Use (twice) the fact that

$$\sum_{i=1}^{n}(Y_i - \bar{Y})^2 = \sum_{i=1}^{n}(Y_i - c)^2 - n(c - \bar{Y})^2$$

First part:

$$\begin{aligned}\text{SSE} &= \sum_{i:X_i<X_{(s+1)}} (Y_i - L_s')^2 + \sum_{i:X_i\geq X_{(s+1)}} (Y_i - R_s')^2 \\ &= \sum_{i:X_i<X_{(s+1)}} (Y_i - \bar{Y})^2 - s(\bar{Y} - L_s')^2 + \sum_{i:X_i\geq X_{(s+1)}} (Y_i - \bar{Y})^2 - (n-s)(\bar{Y} - R_s')^2 \\ &= \sum_{i=1}^{n} (Y_i - \bar{Y})^2 - s(\bar{Y} - L_s')^2 - (n-s)(\bar{Y} - R_s')^2\end{aligned}$$

Therefore, we simply need to iterate until we find the largest $s(\bar{Y} - L_s')^2 + (n-s)(\bar{Y} - R_s')^2$.

Second part: At each step, use the following update rules:

Left sum and mean updates:

$$L_{s+1} \leftarrow L_s + Y_{s+1}, L_{s+1}' \leftarrow \frac{L_{s+1}}{s+1}$$

Right sum and mean updates:

$$R_{s+1} \leftarrow R_s - Y_{s+1}, R_{s+1}' \leftarrow \frac{R_{s+1}}{s-1}$$

Third part:

For each s, we need to calculate and store

$$s(\bar{Y} - L_s')^2 + (n-s)(\bar{Y} - R_s')^2$$

Our first left and right sums take $n-2$ additions together, and the left and right means take 2 divisions together. Then, each of the $n-2$ update-calculate-store steps takes 3 additions, 5 subtractions, 2 divisions, and 4 multiplications. Thus, total, we would need $n-2+3(n-2)$ additions, $5(n-2)$ subtractions, $4(n-2)$ multiplications, and $2+2(n-2)$ divisions to evaluate all the split points. Notice that the number of computations has gone from a quadratic in n to linear in n!

You can see the actual R implementation at this link!

Categorical decision trees

In this class, we mainly look at decision trees as a non-parametric way of making a prediction about a continuous variable of interest. However, another (probably more common) use of decision trees is for predicting categorical variables. While looking at the sum of squares error is a reasonable way to build decision trees for continuous predictions, other methods can be more useful for categorical predictions. This question will look at two such options. Throughout this problem, imagine that we have a data set $\mathbf{X}$ $(n \times p)$ and a set of true categories $\vec{Y}$ $(n \times 1)$ with k possible values (k categories).

1. A first method involves calculating the entropy of the parent and child nodes. Entropy is defined as:

$$E = -\sum_{i=1}^{k} p_i \log_2 p_i$$

where p_i is the proportion of the items in the node from class i. (If $p_i = 0$, we treat its $p_i \log_2 p_i$ term as 0 since $p_i \log_2 p_i \to 0$ as $p_i \to 0$.) When splitting using this metric, the parent node's entropy E_{Parent} is calculated, and the weighted average of the children's entropies are calculated ($\frac{1}{n_{\text{Left}}} E_{\text{Left}} + \frac{1}{n_{\text{Right}}} E_{\text{Right}}$). The split that yields the lowest entropy is chosen. Show that the entropy of a node is minimized when the node is "pure" (there are only items of one class in the node), and show (for $k=2$) that the entropy of the node is maximized when there are an equal proportion of items from each class in the node. (This second statement can be extended to show that the entropy is maximized when the distribution of classes in a node is uniform.)

Minimzation: Because $0 \leq p_i \leq 1$, $\log_2 p_i$ is non-positive and 0 only when $p_i = 1$. Therefore, the entropy will be non-negative with $E = 0$ only when $p_i = 1$ for some i and $p_j = 0$ for the rest.

Maximization: For $k = 2$,

$$E = -[p_1 \log_2 p_1 + (1 - p_1) \log_2 (1 - p_1)]$$

so

$$E' = -\left[\frac{p_1}{p_1 \ln(2)} + \log_2 p_1 - \frac{1 - p_1}{(1 - p_1)\ln(2)} - \log_2(1 - p_1)\right] = \log_2(1 - p_1) - \log_2(p_1)$$

Setting this equal to 0 shows that $p_1 = 0.5$.

2. Because calculating logs can be computationally expensive, a more common method is to maximize the Gini value:

$$G = \sum_{i=1}^{k} p_i^2$$

where p_i is the proportion of the items in the node from class i. Explain why the Gini value can be interpreted as the probability that a randomly chosen item in the node would be assigned its correct class when assigning classes randomly according to the proportions in the node. Also, find when the Gini value is maximized and find when it is minimized for $k = 2$.

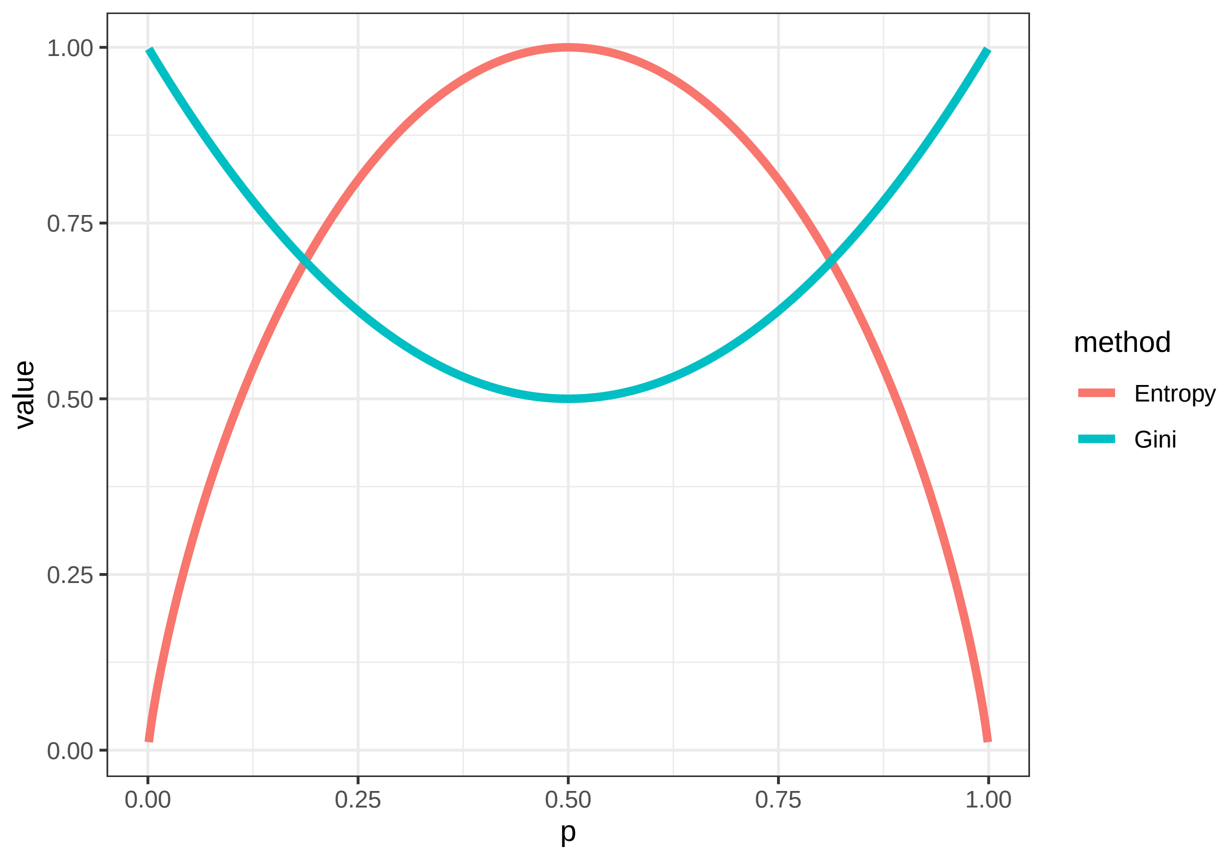
We can interpret this as LOTP where we want $P(\text{Correct classification})$, so we use

$$\sum_{i=1}^{k} P(\text{Correct classification}|\text{Class } i) P(\text{Class } i)$$

The probability of choosing an item of class i from the node is p_i, and the probability of it being randomly assigned the (correct) class i is also p_i, so this comes out to G. Since the Gini value is always between 0 and 1, the Gini value is maximized when the node is pure (i.e. when $p_i = 1$ for one of the classes and 0 for the rest). Taking the derivative for the $k = 2$ case gives $G' = 2p_1 - 2(1 - p_1)$, so setting this to 0 gives $4p_1 = 2 \implies p_1 = p_2 = 0.5$.

3. For a 2-class classification problem, plot the Entropy and Gini values on the same graph.

```
library(ggplot2)
p <- seq(0.001, 0.999, 0.001)
entropy = -(p * log(p, 2) + (1-p) * log(1-p, 2))
gini = p^2 + (1-p)^2

df = data.frame(p=rep(p, 2),
                value=c(entropy, gini),
                method=rep(c("Entropy", "Gini"), each=length(p)))

ggplot(df, aes(x=p, y=value, color=method)) +
  geom_line(size=1.5) +
  theme_bw()
```



The Entire Arboretum

The packages `rpart` and `randomForest` will be the main workhorses of our decision trees and random forests. The most important functions for us will be:

- `rpart(formula, data, control)` where `control` is something like `list(minsplit=1,cp=0,maxdepth=20)`, where `minsplit` says we want the node to be split if it has at least 1 observation; `cp` (complexity parameter) says we will accept any improvement in fit; and `maxdepth` is that maximum node depth.
- `prune(tree, cp)` where `cp` is the complexity parameter.
- `randomForest(formula, data, maxnodes, mtry, ntree)` where `mtry` is the number of variables randomly sampled as candidates at each split, and `ntree` is the number of trees to grow.

This question will deal with a data set of country-level statistics from this source with an explanation of the data encoding found here.

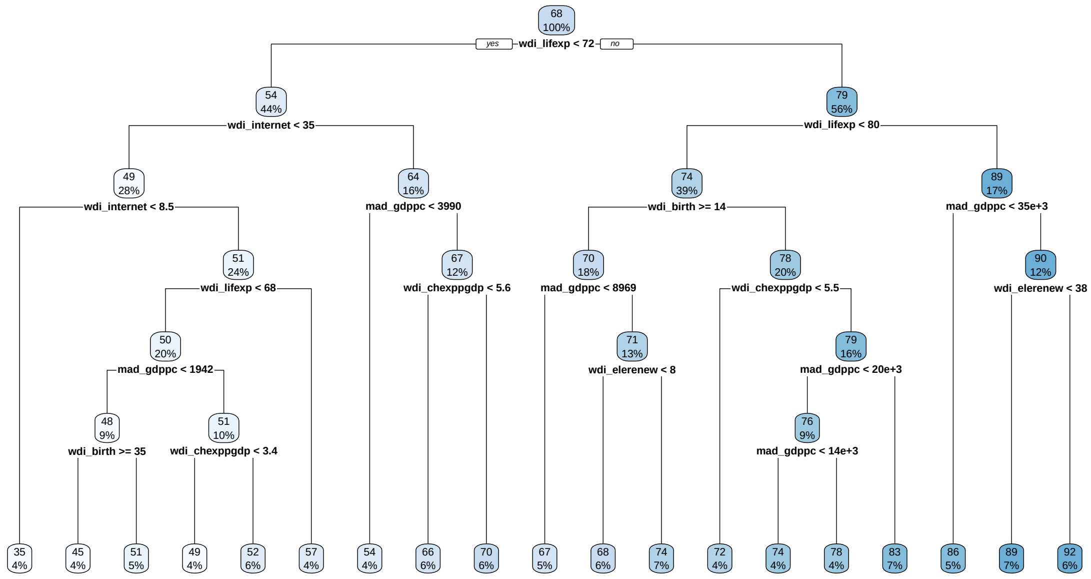
A few useful columns:

- `spi_ospi`: Overall social progress index on 0-100 scale
- `mad_gdppc`: GDP per capita
- `wdi_internet`: Percent of population using the internet
- `wdi_birth`: Birth rate per 1000 people
- `wdi_chexppgdp`: Current health expenditures as percent of GDP
- `wdi_elerenew`: Percent of total electricity output that's renewable
- `wdi_lifexp`: Life expectancy at birth

- `wdi_wip`: Proportion of seats held by women in national parliaments
- `wdi_popurb`: Percentage of total population that is urban
- `wdi_imig`: Proportion of people born outside the country in which they live

1. Fit a complex regression tree to predict `spi_ospi` from the above predictors, plot the decision tree, and report the RMSE and R^2. Use `minsplit=1, cp=0, maxdepth=20` in `rpart`.

```
# Fit tree
tree1 <- rpart(spi_ospi ~ mad_gdppc + wdi_internet + wdi_birth + wdi_chexppgdp +
                          wdi_elerenew + wdi_lifexp + wdi_wip + wdi_popurb + wdi_imig,
               countries, control = list(minsplit=1,cp=0,maxdepth=20))

# Plot tree
rpart.plot(tree1)
```



```
# RMSE
RMSE(countries$spi_ospi[!is.na(countries$spi_ospi)],
     predict(tree1, new=countries)[!is.na(countries$spi_ospi)])
```

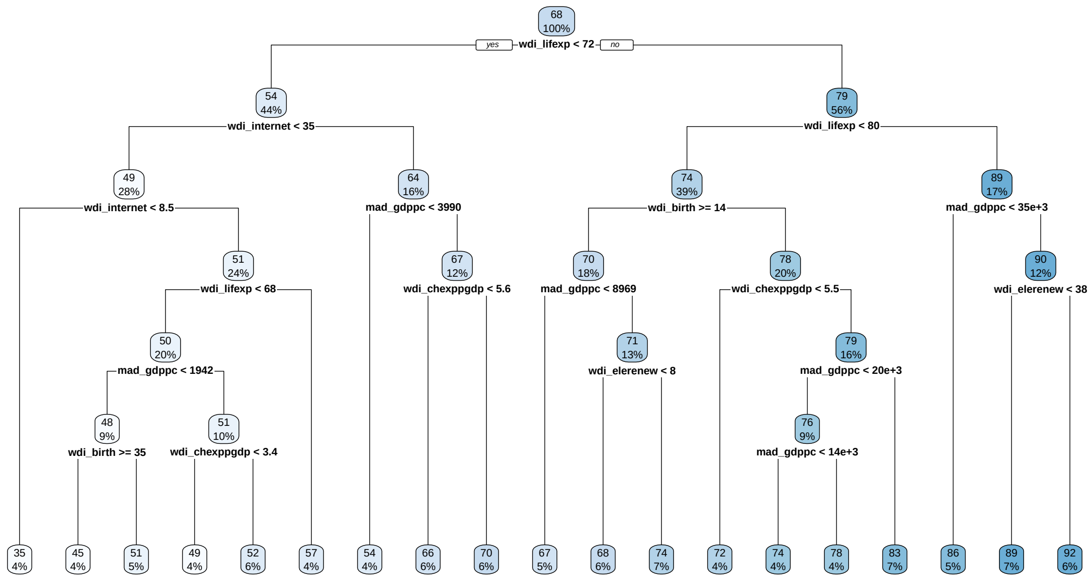
```
## [1] 2.919353
```

```
# Rsq
cor(countries$spi_ospi[!is.na(countries$spi_ospi)],
    predict(tree1, new=countries)[!is.na(countries$spi_ospi)])^2
```

```
## [1] 0.9645421
```

2. Fit a well-pruned regression tree to predict `spi_ospi` from the above predictors, plot the tree, and report the RMSE and R^2. Start with `minsplit=1, cp=0, maxdepth=20)` in `rpart` and use the default 10-fold cross-validation to prune based on `cp`. Note: `rpart` automatically performs 10-fold CV, and the best `cp` can be determined from a tree with the command `tree1$cptable[,"CP"][which.min(tree1$cptable[,"xerror"])]`

```r
tree2 = prune(tree1,cp = tree1$cptable[,"CP"][which.min(tree1$cptable[,"xerror"])])

rpart.plot(tree2)
```



```r
# RMSE
RMSE(countries$spi_ospi[!is.na(countries$spi_ospi)],
     predict(tree2, new=countries)[!is.na(countries$spi_ospi)])
```

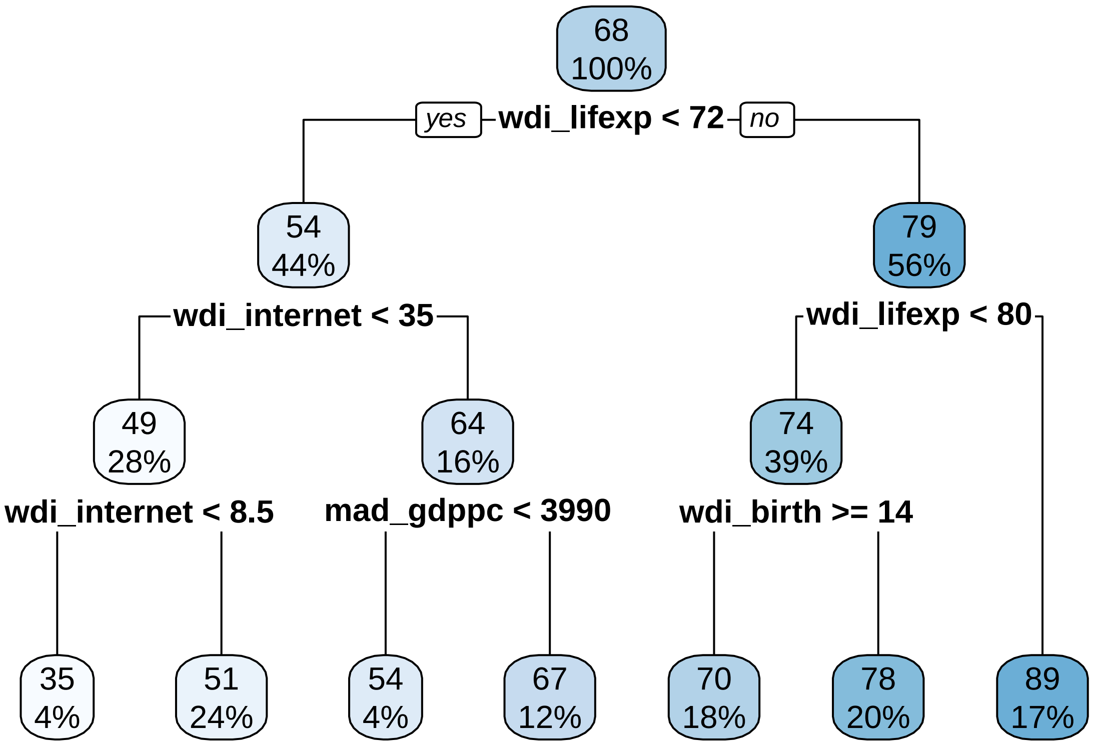
```
## [1] 2.919353
```

```r
# Rsq
cor(countries$spi_ospi[!is.na(countries$spi_ospi)],
    predict(tree2, new=countries)[!is.na(countries$spi_ospi)])^2
```

```
## [1] 0.9645421
```

Here, the tree doesn't actually change, but if we used something like `cp = 0.01`, we would get the following:

```r
treeextra = prune(tree1,cp = 0.01)

rpart.plot(treeextra)
```



```r
# RMSE
RMSE(countries$spi_ospi[!is.na(countries$spi_ospi)],
     predict(treeextra, new=countries)[!is.na(countries$spi_ospi)])
```

```
## [1] 4.314438
```

```r
# Rsq
cor(countries$spi_ospi[!is.na(countries$spi_ospi)],
    predict(treeextra, new=countries)[!is.na(countries$spi_ospi)])^2
```

```
## [1] 0.9225559
```

3. Fit a well-tuned random forest model to predict `spi_ospi` from the above predictors considering `mtry = c(2,3,4)` and `maxnodes = c(3,5,10)` with `ntree=200`. Report RMSE and R^2 as well as the best `mtry` and best `maxnodes`.

```r
param_grid = expand.grid(mtries = c(2, 3, 4), maxnodes = c(3, 5, 10))

curRMSE <- Inf

bestmtry <- 0
bestnode <- 0

# Random forest doesn't handle missing data well so keep only observations with no NAs
eval_subset = countries[rowSums(is.na(countries[,c("spi_ospi", "mad_gdppc",
                                                    "wdi_internet", "wdi_birth",
                                                    "wdi_chexppgdp", "wdi_elerenew",
                                                    "wdi_lifexp", "wdi_wip",
                                                    "wdi_popurb", "wdi_imig")]))==0,]

for (i in nrow(param_grid)) {
  mtry = param_grid[i, 1]
  maxnode = param_grid[i, 2]

  # Fit the random forest
  rf <- randomForest(spi_ospi ~ mad_gdppc + wdi_internet + wdi_birth + wdi_chexppgdp +
                       wdi_elerenew + wdi_lifexp + wdi_wip + wdi_popurb + wdi_imig,
                     countries, maxnodes=maxnode, mtry=mtry, ntree=200, na.action = na.omit)

  # If it has better RMSE, keep it as the best
  if (RMSE(eval_subset$spi_ospi, predict(rf, new=eval_subset)) < curRMSE) {
    bestnode <- maxnode
    bestmtry <- mtry
    rfsave <- rf
  }
}

# RMSE
RMSE(eval_subset$spi_ospi, predict(rfsave, new=eval_subset))
```

```
## [1] 3.458011
```

```r
# Rsq
cor(eval_subset$spi_ospi, predict(rfsave, new=eval_subset))^2
```

```
## [1] 0.9511681
```

```r
bestmtry
```

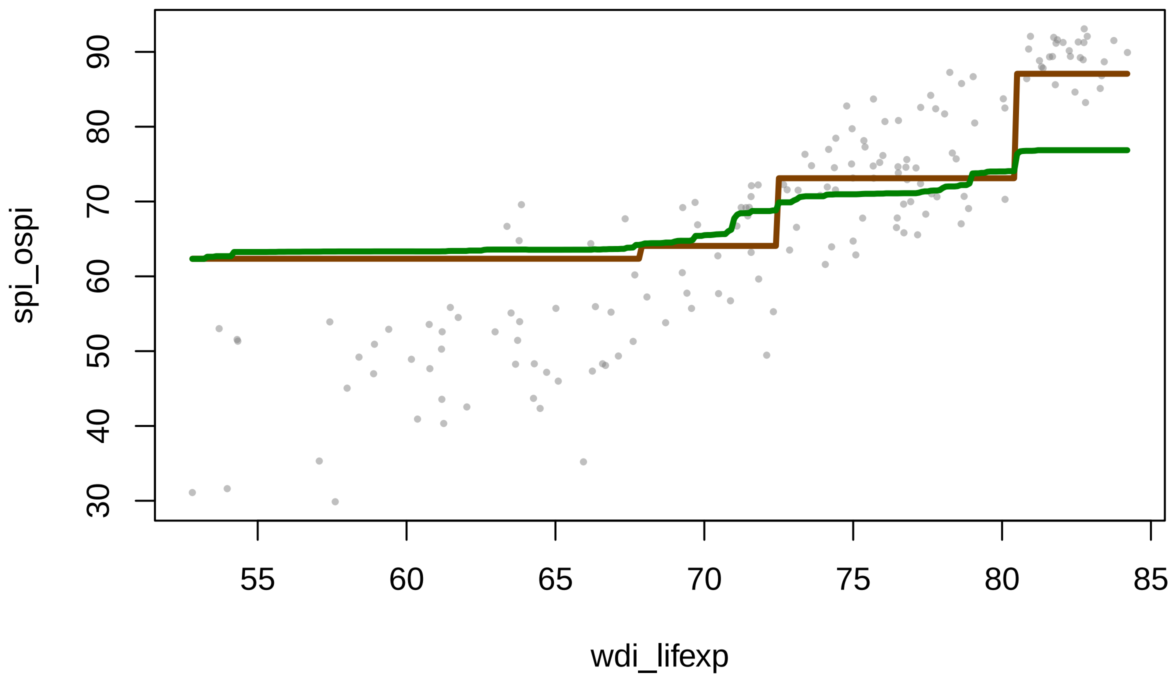
```
## [1] 4
```

```r
bestnode
```

```
## [1] 10
```

4. Interpret the relationship of `spi_ospi` with `wdi_lifexp` in the first tree and the best random forest model through predictions on a single plot. Describe what you see in a few sentences.

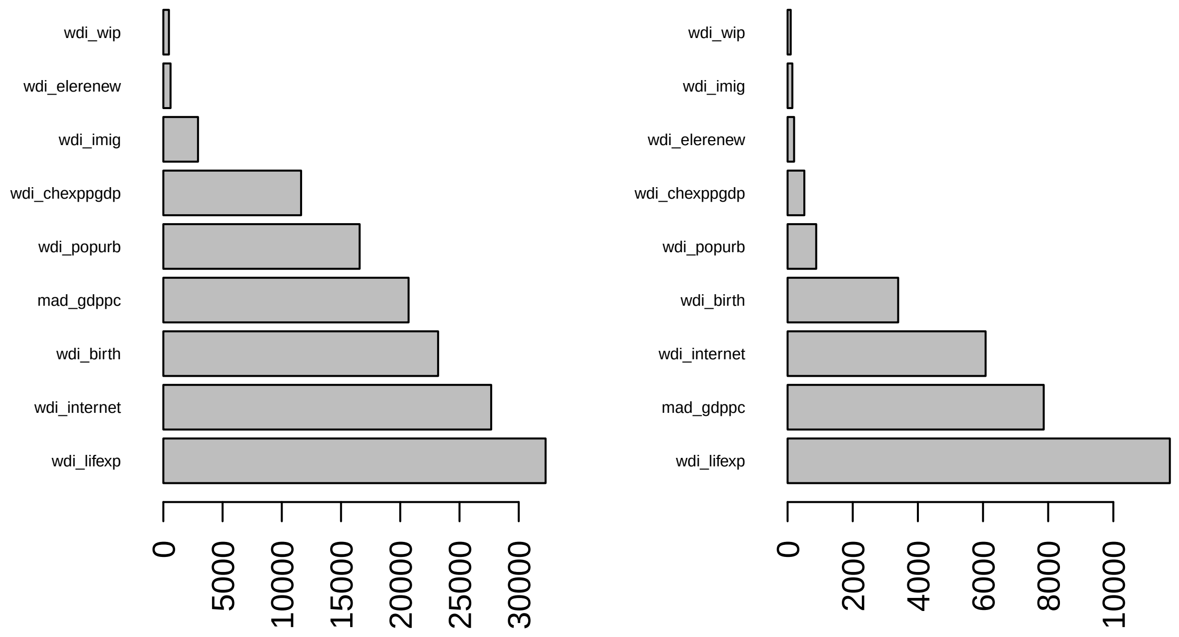
```r
# Copy original data
dummy_df = eval_subset

# Sequence of possible life expectancies
dummylifexp = seq(min(eval_subset$wdi_lifexp), max(eval_subset$wdi_lifexp), 0.1)

# Show original points
plot(spi_ospi ~ wdi_lifexp, data=countries, col=rgb(0.5,0.5,0.5,0.5), cex=0.5, pch=16)

# Get predictions on the dummy data
yhats=matrix(NA,nrow=nrow(dummy_df),ncol=length(dummylifexp))
yhats2=matrix(NA,nrow=nrow(dummy_df),ncol=length(dummylifexp))
for(i in 1:nrow(dummy_df)){
  # For each original data points, test all dummy life expectancies
  rows=dummy_df[rep(i,length(dummylifexp)),]
  rows$wdi_lifexp=dummylifexp
  yhat = predict(tree1,new=rows)
  yhats[i,]=yhat
  yhat2 = predict(rfsave, new=rows)
  yhats2[i,]=yhat2
}

# Take the mean over the predicted OSPI at each dummy life expectancy
mean_yhat = apply(yhats,2,mean)
mean_yhat2 = apply(yhats2, 2, mean)

# Tree in brown
lines(mean_yhat~dummylifexp,col=rgb(0.5,0.25,0,1),lwd=3)

# Forest in green
lines(mean_yhat2~dummylifexp,col=rgb(0,0.5,0,1),lwd=3)
```



The tree is in brown and the forest is in green. As expected, the forest is more smooth than the single tree. Neither seems to be capturing the very low and very high values particularly well, which suggests that other predictors are doing more of the work there.

5. Which predictors are most important in the first tree and the best random forest? How do they compare in relative importance?

```r
par(mfrow=c(1, 2))
barplot(tree1$variable.importance, horiz=T, las=2, cex.names=0.5)
barplot(sort(rfsave$importance[,1], decreasing = T), horiz=T, las=2, cex.names=0.5)
```



In both models, a country's life expectancy is the largest predictor of the OSPI (though it is more important in the random forest than in the single tree). The birth rate, access to the internet, and the GDP per capita are also important in both models though they differ in relative importance.