STA130 W9

Wednesday, November 13, 2019 5:59 PM

1. **Testing:** A hypothesis test evaluating evidence against a particular value for parameter
Statistical method(s):one sample tests(for a proportion p)
Randomization tests to compare the value of a parameter access 2 groups
2. **Estimation**: Confidence interval estimating a parameter (gives range of plausible values for a parameter)
Statistical method(s):Bootstrap method for CIs
3. **Prediction**: Predict value of a variable for an observation using a statistical model based on other variables
Statistical method(s):1. classification trees 2. linear regression

Using data to make predictions

Types of variables

The **x variables** are often called **predictors** , covariates, independent variables, explanatory variables, inputs, or features
The **y variable** is often called response, output, outcome , or dependent variable

Types of responses for prediction

There are many types of models to choose from to make predictions.
Classification trees: useful when the response y is categorical (today)
Linear regression: useful when the response y is numerical (next week)

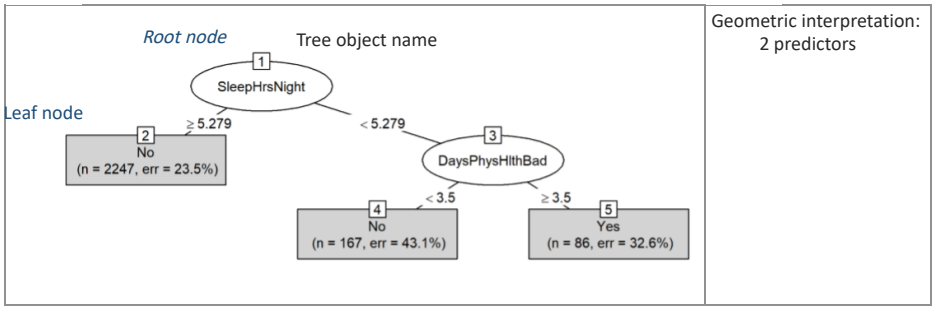
Example: Predicting which people have trouble sleeping
Goal: Build a classification tree to predict which individuals have trouble sleeping based on a sample of 2500 individuals surveyed in the US between 2009 and 2012

Classification (Decision) Trees

An upside down tree that "grows" from the **root node** towards the **terminal (leaf) nodes**
Terminal nodes are indicated by rectangular boxes, and non-terminal nodes by ovals.
Each non-terminal node splits to create two **child** nodes.
Separates response for predictors

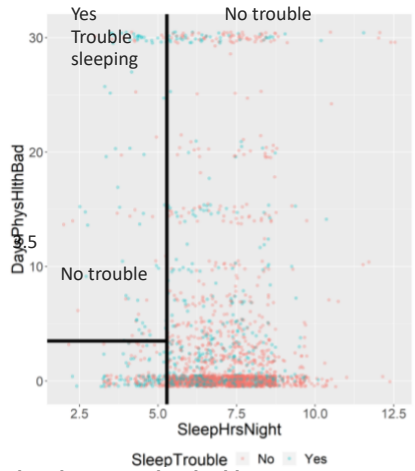
```
library(rpart);  to build
library(partykit); to plot
                    y          All of the potential predictors separate by "+" signs
tree <- rpart(SleepTrouble~ SleepHrsNight + DaysPhysHlthBad, data=data)
plot(as.party(tree), gp=gpar(cex=0.8), type="simple")              Data to build the potential predictor
                                Text size
```

Splitting upside-down trees



Geometric interpretation:
2 predictors

Root node:	1	Parent nodes	1,3
Terminal(leaf)nodes:	2,4,5	Child nodes	2,3,4,5



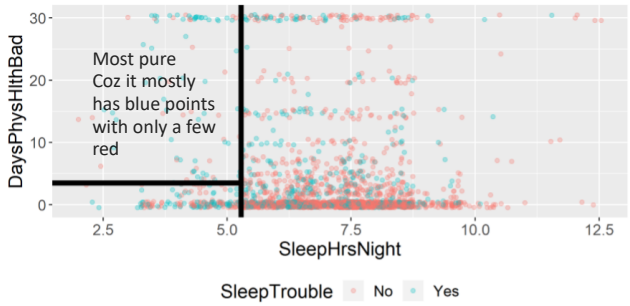
What do we need to build a tree

A response variable (y , what we want to predict)

A set of candidate predictors (x_i , i=1,…,M)
A set of binary questions (e.g. is $x_1 \geq 25$) -> split node
A method to evaluate if a split is "good"
A rule to use to decide when to stop splitting
A way to make a prediction (yes/no) for each terminal node

What kind of questions can we use for splits
For the trees we will build, we need a **categorical response** and **binary splits** (TRUE / FALSE) based on predictors.
For example, we could use: True for observations in this set and false for the rest
Numerical predictors such as SleepHrsNight (self-reported average # of hours of sleep on weekdays):
e.g. SleepHrsNight > 6
Categorical predictors such as MaritalStatus:
e.g. MaritalStatus %in% c("Married", "LivePartner")

"Good" split
A "good" split is one that makes its **child** nodes as pure as possible (i.e. homogeneous with respect to the response)
A node is **pure** if it contains only observations from one class
A node is **impure** if it contains an equal mix of all the classes (ex: 50% "no sleep trouble" and 50% "sleep trouble")



Which area looks the most pure? The least pure?

What is the "best" split?
For each candidate split, we can calculate the decrease in impurity ΔI
ΔI measures how much purer the (two) children nodes would be, compared to the parent node

When we want to split a node, we look at:
(i) each potential predictor variable and
(ii) each possible split for each variable and calculate ΔI for each one
The "best" split is the one with **the biggest decrease** in impurity ΔI
*R does this automatically

When to stop splitting?
Two competing goals when building a tree:
1. We want each terminal (leaf) node to be as pure as possible
2. We don't want a tree that is too complex

A simple "stop-splitting" rule is to set **a threshold β >0 .**
***for this course default is ok**
If none of the possible splits for a node makes the tree at least β-units more pure, then we don't split any further.

Evaluating the accuracy of a tree(how good our tree is)
Negative: no trouble sleeping
Positive: is trouble sleeping

	Actually Negative(no trouble)	Actually positive(do have trouble)	Total
Predict negative	# TN	# FN	# predict negative
Predict positive	# FP	# TP	# predict positive
Total	# actually negative	# actually positive	N (total)

True positive rate (sensitivity): # TP / # Actually positive
True negative rate (specificity): # TN / # Actually negative
False negative rate: # FN / # Actually positive = 1 - TPR
False positive rate: # FP / # Actually negative = 1 - TNR
Accuracy: (# TN + # TP) / N
*sensitivity: proportion of actual positives correctly predicted by our model
*specificity: proportion of actual negatives correctly predicted by our model
*all value between 0-1
Outcome of interest: Goal is to predict people who have trouble sleeping so "has trouble sleeping" is a **positive** (yes) outcome. If the tree correctly classifies a person as having trouble sleeping based on the predictors, it is a true positive.

Validating classification trees
Suppose we built a tree and want to evaluate how **valid** it is for prediction.
Strategy:
1. Randomly divided data into "training" and "testing" datasets
   Ex: 80% for training and 20% for testing
2. Fit the tree using the training data
3. Run "test" data through the fitted tree and check how many observations are correctly classified

Build tree for training data, test the tree using **testing data**(or we have nothing to test it(data it hasn't seen before))

```
n <- nrow(data); n
## [1] 2500(original sample size)

#Randomly select 80% of observations to put in the training dataset
set.seed(1111)
training_indices <- sample(1:n, size=round(0.8*n)) random rows of the original sample
                                  Giving you a random list of 2000 numbers between 1 and 2500 without replacement
train <- data[training_indices,]
        a vector of row numbers
       Data[]->picks out the rows with indices in this vector
nrow(train)
## [1] 2000

Test dataset includes all observations NOT in the training data
test <- data[-training_indices,] *include all row number not in this vector
nrow(test)
## [1] 500
```

Fitting a tree using the training data

```
tree <- rpart(SleepTrouble ~ SleepHrsNight + DaysPhysHlthBad,
           data=train)
plot(as.party(tree), type="simple", gp=gpar(cex=0.8))
```

Constructing 2x2 "confusion matrix" for this tree

```
Vector of pred.                                    Categorical prediction(yes/no)
predictions <- predict(object = tree, newdata = test, type = "class")
                              The Tree we built    What data we want to make predictions for

head(predictions)
Table(predictions) for 500 obs in test data
```

Calculating prediction accuracy

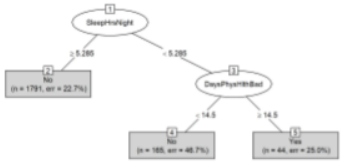
```
table(predictions, test$SleepTrouble)
```

## predictions No Yes (true values from test data) ## No 356 136 categorical pred. for each obs ## Yes 3 5	gives us the predicted probability of Yes trouble sleeping No trouble sleeping For each obs. In "newdata"

Accuracy = (356+ 5)/500 =
False positive rate = #FP / # n. no trouble = 3/359
False negative rate = #FN / # n. with trouble = 136/141 -> to high, not good
Good tree - low FPR and low FNR

Prediction accuracy: validation vs using same data to build/test

Build the tree using the training dataset
Make predictions for the testing dataset



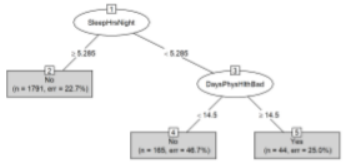
```
m.val
```

```
##
## pred   No Yes
##   No  356 136
##   Yes   3   5
```

```
sum(diag(m.val))/sum(m.val)
```

```
## [1] 0.722
```

Build the tree using the training dataset
Make predictions for the training dataset



```
m.trOnly
```

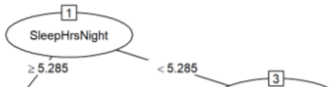
```
##
## pred    No  Yes
##   No  1473  483
##   Yes   11   33
```

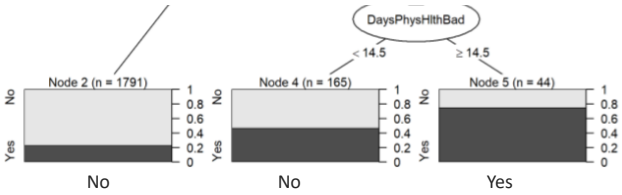
```
sum(diag(m.trOnly))/sum(m.trOnly)
```

```
## [1] 0.753
```

Extension: Making predictions in the terminal nodes?

```
tree <- rpart(SleepTrouble ~ SleepHrsNight + DaysPhysHlthBad, data=train)
plot(as.party(tree), type="extended", gp=gpar(cex=0.8))
                              Alternative to "simple"
                              Shows proportion of yes/no observation from the data used to build the tree
```





No No Yes

We just used **"majority rules"** to choose predicted label for each terminal node, but we could use a different cutpoint.

Extension: Making predictions in the terminal nodes?

In our data, only 26% of all individuals have trouble sleeping

```
table(data$SleepTrouble)/nrow(data)
## ##     No    Yes
## 0.7372 0.2628
```

checking which observations have more than 26% predicted prob of trouble sleeping

So we could say that when we build our tree, **terminal node with a t least 26% of individuals with trouble sleeping** are linked with a prediction of "Trouble Sleeping" (i.e. terminal nodes with a higher proportion than all observations in the dataset)

Previously, with "majority rules", we were using the cutpoint value 50%

The **predict** function can be used to calculate the **probability** of each possible label for each new observation in the **test** dataset.

Previously, we used **predict** to get predicted labels directly

```
predictions <- predict(object = tree, newdata = test, type = "class")
is.factor(predictions)
## [1] TRUE
```

But we can also get **the predicted probability of each label**

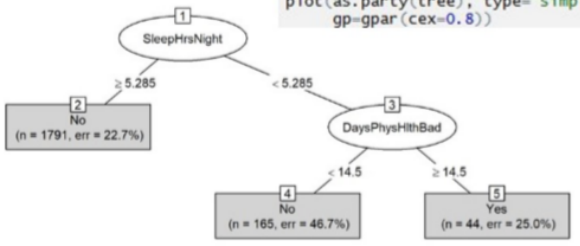
```
pred_probs <- predict(object = tree, newdata = test, type = "prob")
head(pred_probs)
```

```
##        No      Yes
## 1 0.773311 0.226689
## 2 0.773311 0.226689
## 3 0.773311 0.226689
## 4 0.773311 0.226689
## 5 0.773311 0.226689
## 6 0.773311 0.226689
```

specify the second column of the 2x2matrix
->is the vector of predicted probabilities for "yes trouble sleeping"

Tree built using the TRAINING data; n and err are related to `train` dataset

```
tree <- rpart(SleepTrouble ~ SleepHrsNight +
                DaysPhysHlthBad, data=train)
plot(as.party(tree), type="simple",
     gp=gpar(cex=0.8))
```



Observations 1, 10, 27 and 28 from the test dataset

```
test %>% mutate(i = 1:nrow(test)) %>%
  filter(i %in% c(1,10,27,28)) %>%
  select(i, SleepHrsNight, DaysPhysHlthBad, SleepTrouble)
A tibble: 4 x 4
     i SleepHrsNight DaysPhysHlthBad SleepTrouble
<int>         <dbl>           <int> <fct>
    1          6.34               0 Yes
   10          4.47               0 No
   27          3.70              15 No
   28          9.50               3 Yes
```

Predicted probabilities for observations 1, 10, 27, and 28 from the test dataset

```
pred_probs[c(1,10,27,28),]
```

```
##           No       Yes
## 1  0.7733110 0.2266890
## 10 0.5333333 0.4666667
## 27 0.2500000 0.7500000
## 28 0.7733110 0.2266890
```

where do we indicate yes/no

So if we use a 26% cutoff:

```
m26 <- table(pred_probs[,2] > 0.26,
             test$SleepTrouble)
row.names(m26) <- c("Pred No Trouble",
                    "Pred Trouble")
m26
```

```
##
##                    No Yes
##   Pred No Trouble 333 120
##   Pred Trouble     26  21
```

```
(m26[1,1] + m26[2,2]) / sum(m26)
```

```
## [1] 0.708
```

Compared to a 50% cutoff:

```
m50 <- table(pred_probs[,2] > 0.50,
             test$SleepTrouble)
row.names(m50) <- c("Pred No Trouble",
                    "Pred Trouble")
m50
```

```
##
##                    No Yes
##   Pred No Trouble 356 136
##   Pred Trouble      3   5
```

```
(m50[1,1] + m50[2,2]) / sum(m50)
```

```
## [1] 0.722
```

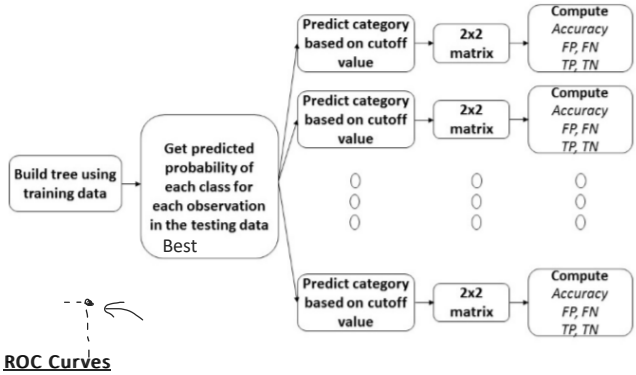
Or if we didn't bother fitting a tree and just predicted "No Trouble" for all:

```
##
## Pred No Trouble 359 141
## Pred Trouble      0   0
```

Accuracy = 0.708	Accuracy = 0.722	Accuracy = 0.718
False positive rate = 26 / (333+26) = 0.07	False positive rate = 3 / (356+3) = 0.008	False positive rate = 0
False negative rate = 120 / (120+21) = 0.85	False negative rate = 136 / (136+5) = 0.96	False negative rate = 1

2x2 matrix **"confusion matrix"**

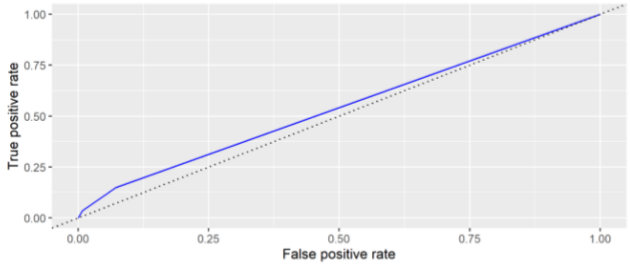
Choice of cutpoint affects accuracy, sensitivity, and specificity



ROC Curves

Used to compare various classifiers (i.e. various cutpoints) from a single tree

The ROC curve is a plot of the **true positive rate vs the false positive rate** for various cutpoints.



ROC Curves

Used to compare various classifiers (i.e. various cutpoints) from a single tree

The ROC curve is a plot of the true positive rate vs the false positive rate for various cutpoints.

- **ne ROC curve corresponds to one tree**
- **Different points along the ROC curve correspond to different choices of cutpoint(which each lead to a particular TPR and FPR)**

Dotted diagonal line : Represents a useless predictor, where actual positives and actual negatives are predicted as positives at the same rate (i.e. just flipping a coin to make the prediction)

- **Don't use** predictor below the doted line

A good classifier is as close to **the top-left corner** as possible (low FPR and high TPR)

- A perfect predictor: 0 FPR 1 TPR

- Different predictor will have different tree and different ROC

How to choose which tree to use (and which cutpoint)?

We want **high accuracy** (trade off between False Negatives and False Positives)

We want a tree that is **not too complex** - this makes it more interpretable (i.e. the tree "makes sense")

Limitations of training/testing set approach

The 'accuracy' we calculate for a tree **depends on** which observations are in the training/testing sets (random)

Only a subset of observations are used to build the tree

- **Statistical methods perform better** when **more data** is used to fit them
- This approach may make the error rate look worse than it would be if we had used all the data to fit the model (instead of reserving some observations for testing)

*data need to be **representative** for the population