Week 3: Predicting with trees, Random Forests, & Model Based Predictions

Lectures[*]**:** 1. Predicting with trees, 2. Bagging, 3. Random Forests, 4. Boosting, 5. Model Based Prediction

Juan David Leongómez

21 March, 2021

Contents

[*]All lectures by Jeffrey Leek (John Hopkins Bloomberg Scool of Public Health).

1 Lecture 1: Predicting with trees

This lecture is about predicting with trees. The basic idea is that if you have a bunch of variables that you want to use to predict an outcome, you can take each of those variables, and use it to split the outcome into different groups. And, so as you split the outcomes into different groups, then you can evaluate the homogeneity of the outcome within each group. And continue to split again if necessary, until you get outcomes that are separated into groups that are homogeneous enough, or that they are small enough that you need to stop.

This approach is easy to interpret right, and you tend to get better performance in non-linear settings than with the linear regression models we talked about in the previous lectures.

1.1 Key ideas

- Iteratively split variables into groups
- Evaluate "homogeneity" within each group
- Split again if necessary

Pros:

- Easy to interpret
- Better performance in nonlinear settings

Cons:

- Without pruning/cross-validation can lead to overfitting
- Harder to estimate uncertainty

- Results may be variable

1.2 Example tree

```
knitr::include_graphics("treexample.jpg")
```

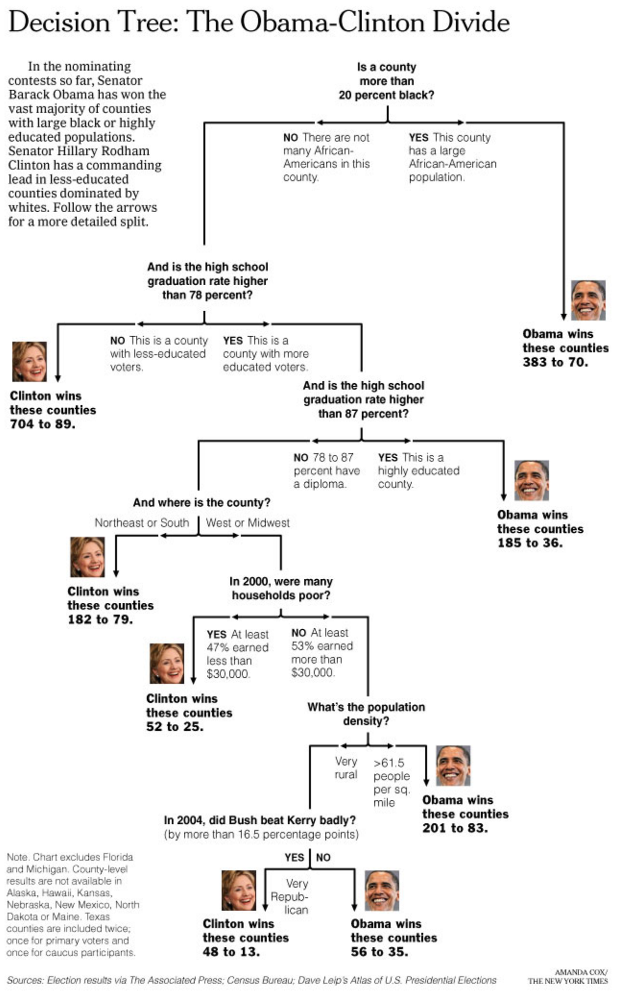

Decision Tree: The Obama-Clinton Divide
In the nominating contests so far, Senator Barack Obama has won the vast majority of counties with large black or highly educated populations. Senator Hillary Rodham Clinton has a commanding lead in less-educated counties dominated by whites. Follow the arrows for a more detailed split.
Is a county more than 20 percent black?
NO There are not many African-Americans in this county.
YES This county has a large African-American population.
Obama wins these counties 383 to 70.
And is the high school graduation rate higher than 78 percent?
NO This is a county with less-educated voters.
YES This is a county with more educated voters.
Clinton wins these counties 704 to 89.
And is the high school graduation rate higher than 87 percent?
NO 78 to 87 percent have a diploma.
YES This is a highly educated county.
Obama wins these counties 185 to 36.
And where is the county?
Northeast or South
West or Midwest
Clinton wins these counties 182 to 79.
In 2000, were many households poor?
YES At least 47% earned less than $30,000.
NO At least 53% earned more than $30,000.
Clinton wins these counties 52 to 25.
What's the population density?
Very rural
>61.5 people per sq. mile
Obama wins these counties 201 to 83.
In 2004, did Bush beat Kerry badly? (by more than 16.5 percentage points)
YES
NO
Very Repub-lican
Clinton wins these counties 48 to 13.
Obama wins these counties 56 to 35.
Note: Chart excludes Florida and Michigan. County-level results are not available in Alaska, Hawaii, Kansas, Nebraska, New Mexico, North Dakota or Maine. Texas counties are included twice; once for primary voters and once for caucus participants.
Sources: Election results via The Associated Press; Census Bureau; Dave Leip's Atlas of U.S. Presidential Elections
AMANDA COX/THE NEW YORK TIMES

Figure 1: Decision tree from the New York Times during the 2008 elections, when Barack Obama was running against Hillary Clinton for the Democratic nomination for President. And they were trying to decide what would be a prediction rule for whether a county would vote for Obama or for Hillary Clinton.

1.3 Basic algorithm

1. Start with all variables in one group
2. Find the variable/split that best separates the outcomes
3. Divide the data into two groups ("leaves") on that split ("node")
4. Within each split, find the best variable/split that separates the outcomes
5. Continue until the groups are too small or sufficiently "pure"

1.4 Measures of impurity

http://en.wikipedia.org/wiki/Decision_tree_learning

$$\hat{p}_{mk} = \frac{1}{N_m} \sum_{x_i \ in \ Leaf \ m} \not{\Vdash}(y_i = k)$$

1.4.1 Misclassification Error

$$1 - \hat{p}_{mk(m)}; k(m) = \text{most; common; k}$$

- 0 = perfect purity (**no misclassification**)
- 0.5 = no purity

1.4.2 Gini index

Not to be confused with the GINI coefficient.

$$\sum_{k \neq k'} \hat{p}_{mk} \times \hat{p}_{mk'} = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk}) = 1 - \sum_{k=1}^{K} p_{mk}^2$$

- 0 = perfect purity
- 0.5 = no purity

1.4.3 Deviance/information gain

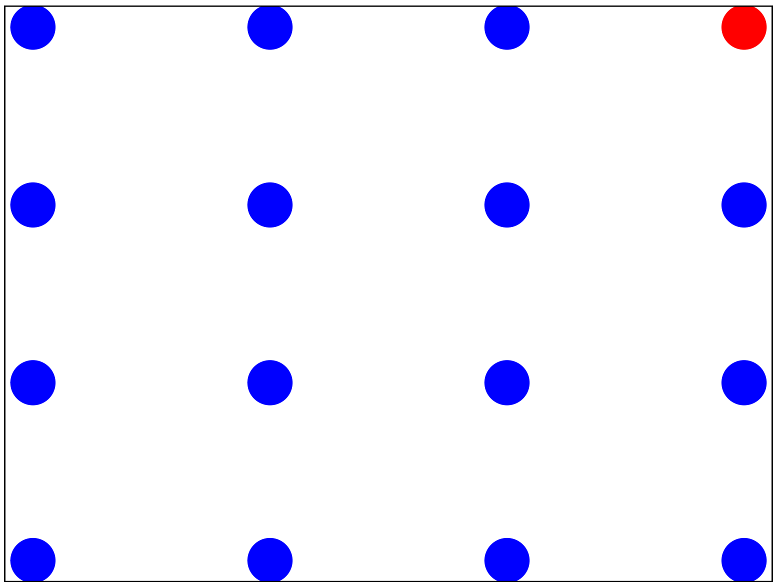
$$-\sum_{k=1}^{K} \hat{p}_{mk} \log_2 \hat{p}_{mk}$$

Deviance gain when you use log_e, and information gain when you use base 2 (i.e. log_2).

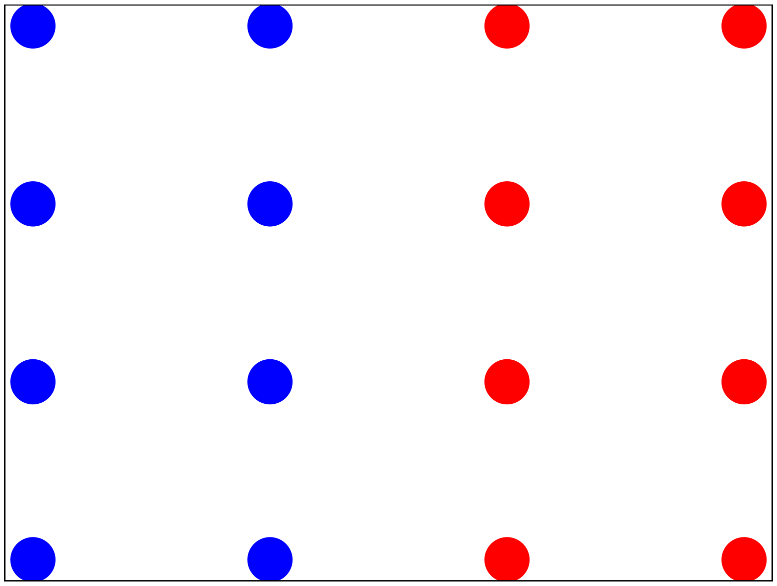
- 0 = perfect purity
- 1 = no purity

1.4.4 Example of measures of impurity

Suppose we had a variable that was trying to split the dots into the blue and the red dots. The first example is good, as classification is high (impurity is low), whereas the first examples is bad (basically, a coin flip for classification).



- **Misclassification:** $1/16 = 0.06$
- **Gini:** $1 - [(1/16)^2 + (15/16)^2] = 0.12$
- **Information:**$-[1/16 \times log2(1/16) + 15/16 \times log2(15/16)] = 0.34$



- **Misclassification:** $8/16 = 0.5$
- **Gini:** $1 - [(8/16)^2 + (8/16)^2] = 0.5$
- **Information:**$-[1/16 \times log2(1/16) + 15/16 \times log2(15/16)] = 1$

1.5 Example: iris data

We will try to predict the species, based on petal/sepal width/length:

```
library(ggplot2)

data(iris)

table(iris$Species)
```

```
##
##     setosa versicolor  virginica
##         50         50         50
```

1.5.1 Crate training and test sets

```
library(caret)
```

```
## Loading required package: lattice
```

```
inTrain <- createDataPartition(y=iris$Species,
                               p=0.7, list=FALSE)
training <- iris[inTrain,]
testing <- iris[-inTrain,]

dim(training)
```

```
## [1] 105   5
```

```
dim(testing)
```

```
## [1] 45  5
```

1.5.2 Iris petal widths/sepal width

This is a relatively easy classification problem. It might be a little bit challenging for a linear model, but not necessarily challenging for this more advanced model with classification trace.

```
library(ggpubr)

ggplot(training, aes(x = Petal.Width, y = Sepal.Width, color = Species)) +
  geom_point(alpha = 0.7) +
  theme_pubclean()
```

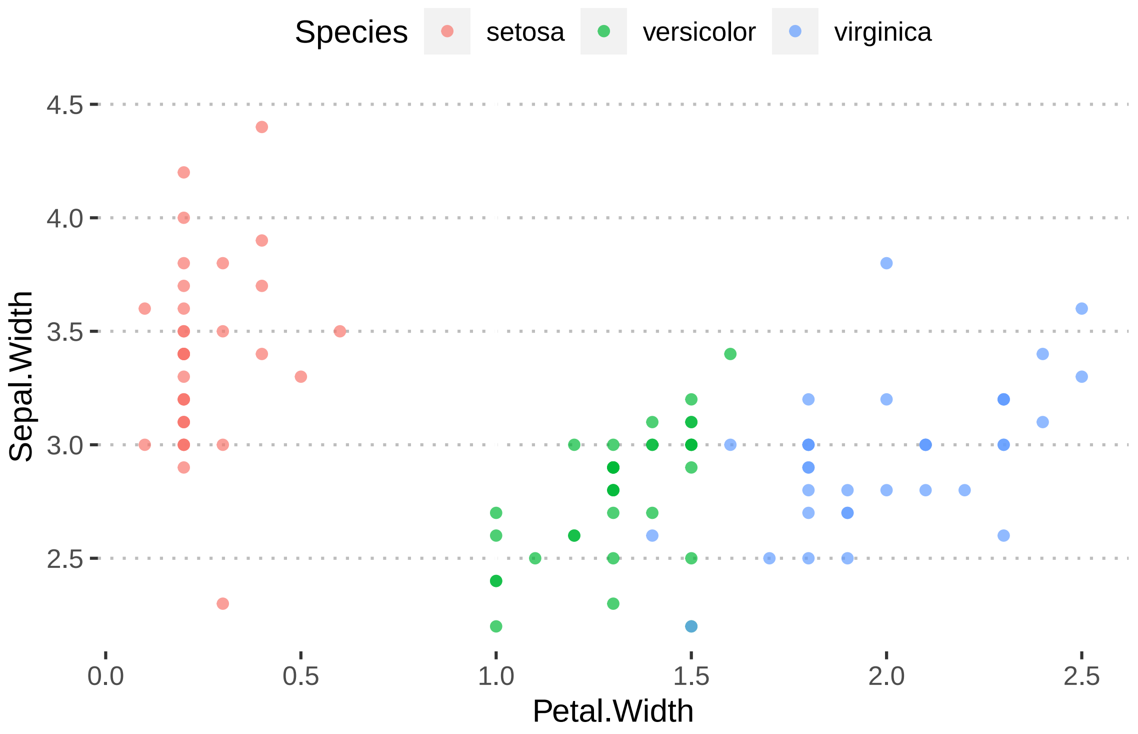


Figure 2: Classification of species in the iris data set by `Petal.Width` and `Sepal.Width`. There are distinct clusters for each species.

Train the model, using `method = "rpart"`.

```
modFit <- train(Species ~ .,method="rpart",data=training)
modFit$finalModel
```

```
## n= 105
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
## 1) root 105 70 setosa (0.33333333 0.33333333 0.33333333)
##   2) Petal.Length< 2.35 35  0 setosa (1.00000000 0.00000000 0.00000000) *
##   3) Petal.Length>=2.35 70 35 versicolor (0.00000000 0.50000000 0.50000000)
##     6) Petal.Width< 1.65 38  3 versicolor (0.00000000 0.92105263 0.07894737) *
##     7) Petal.Width>=1.65 32  0 virginica (0.00000000 0.00000000 1.00000000) *
```

Here, for example, there is a split that says: `Petal.Length` < 2.45, the species is `setosa`.

This classification tree can be plotted:

```
par(mar = c(3, 2, 3, 2))
plot(modFit$finalModel,
     uniform = TRUE,
     main = "Classification Tree")
text(modFit$finalModel, use.n=TRUE, all=TRUE, cex=.8)
```

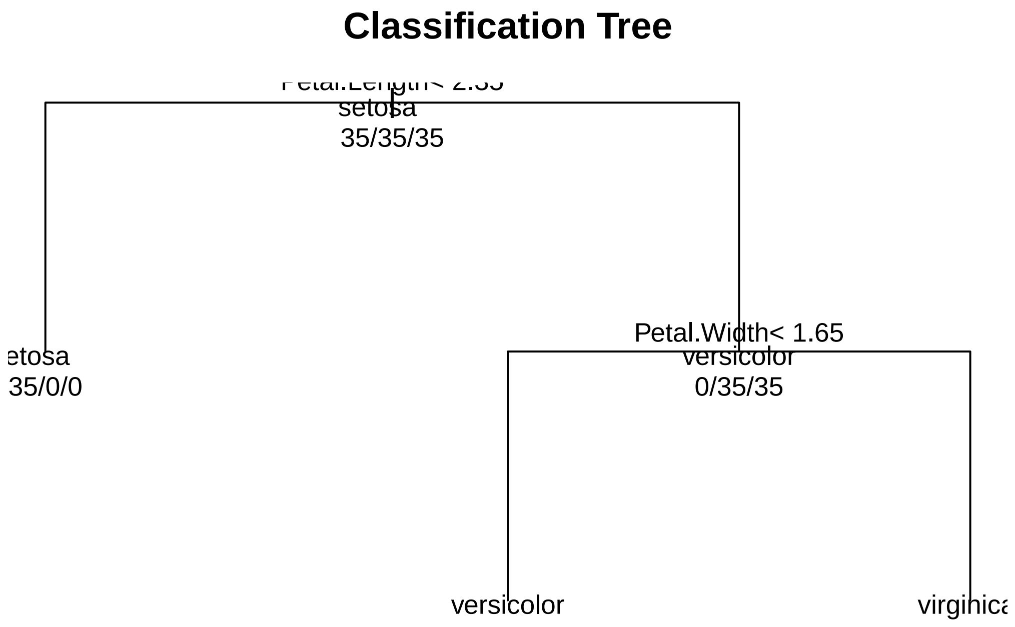


Figure 3: Classification Tree of species in the iris data set by `Petal.Width` and `Sepal.Width`.

Or, for a nicer plot:

```
library(rattle)

fancyRpartPlot(modFit$finalModel)
```

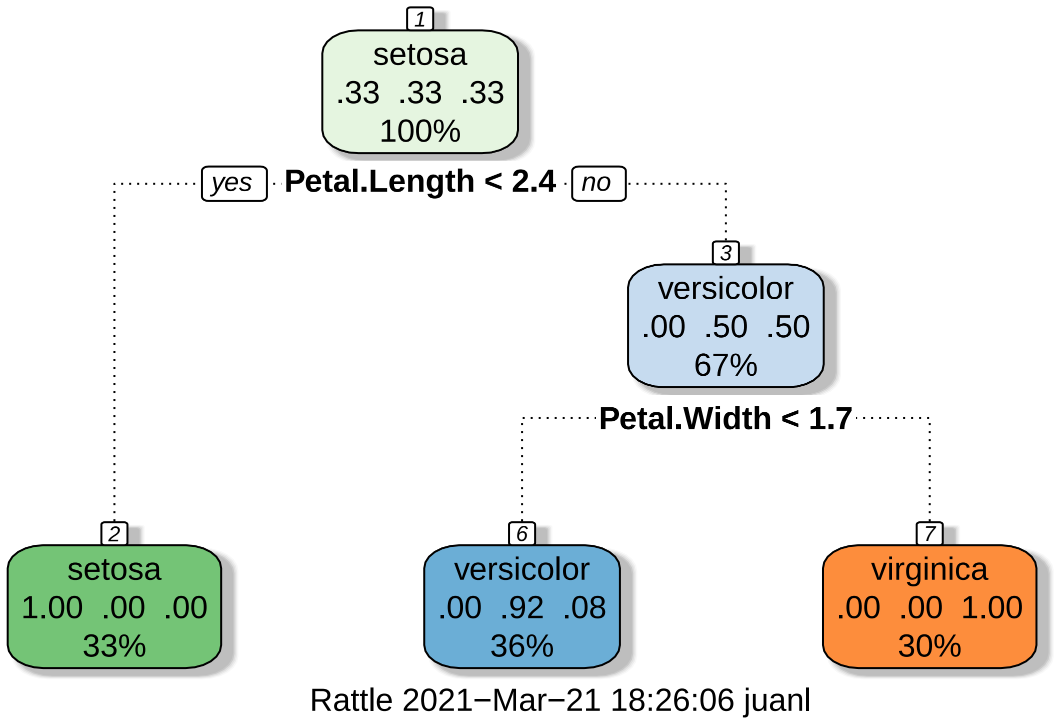


Figure 4: Classification Tree, using the `rattle` package, of species in the iris data set by `Petal.Width` and `Sepal.Width`.

1.5.3 Predicting new values

```
predict(modFit, newdata = testing)
```

```
##  [1] setosa     setosa     setosa     setosa     setosa     setosa
##  [7] setosa     setosa     setosa     setosa     setosa     setosa
## [13] setosa     setosa     setosa     versicolor versicolor versicolor
## [19] versicolor versicolor versicolor versicolor virginica  versicolor
## [25] versicolor virginica  versicolor versicolor versicolor versicolor
## [31] virginica  virginica  virginica  virginica  virginica  virginica
## [37] versicolor virginica  virginica  virginica  virginica  virginica
## [43] virginica  virginica  virginica
## Levels: setosa versicolor virginica
```

(**I discovered you can also use `confusionMatrix` to get accuracy**).

```
caret::confusionMatrix(testing$Species,
                predict(modFit, testing))
```

```
## Confusion Matrix and Statistics
##
##             Reference
## Prediction   setosa versicolor virginica
##   setosa         15          0         0
##   versicolor      0         13         2
##   virginica       0          1        14
##
## Overall Statistics
```

```
##
##                Accuracy : 0.9333
##                  95% CI : (0.8173, 0.986)
##     No Information Rate : 0.3556
##     P-Value [Acc > NIR] : 5.426e-16
##
##                   Kappa : 0.9
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: setosa Class: versicolor Class: virginica
## Sensitivity                 1.0000            0.9286           0.8750
## Specificity                 1.0000            0.9355           0.9655
## Pos Pred Value              1.0000            0.8667           0.9333
## Neg Pred Value              1.0000            0.9667           0.9333
## Prevalence                  0.3333            0.3111           0.3556
## Detection Rate              0.3333            0.2889           0.3111
## Detection Prevalence        0.3333            0.3333           0.3333
## Balanced Accuracy           1.0000            0.9320           0.9203
```

1.6 Example (mine) with regression

To predict `Petal.Length`.

```
modFit <- train(Petal.Length ~ .,
                method="rpart",
                data=training)
```

```
## Warning in nominalTrainWorkflow(x = x, y = y, wts = weights, info = trainInfo, :
## There were missing values in resampled performance measures.
```

```
modFit$finalModel
```

```
## n= 105
##
## node), split, n, deviance, yval
##       * denotes terminal node
##
## 1) root 105 324.522300 3.748571
##   2) Petal.Width< 0.8 35   0.664000 1.460000 *
##   3) Petal.Width>=0.8 70  48.886430 4.892857
##     6) Speciesvirginica< 0.5 35   5.982857 4.214286 *
##     7) Speciesvirginica>=0.5 35  10.671430 5.571429 *
```

Here, for example, there is a split that says: `Petal.Width` < 0.8, `Petal.Length` is 1.5 (??).

This classification plot:

```
library(rattle)

fancyRpartPlot(modFit$finalModel)
```

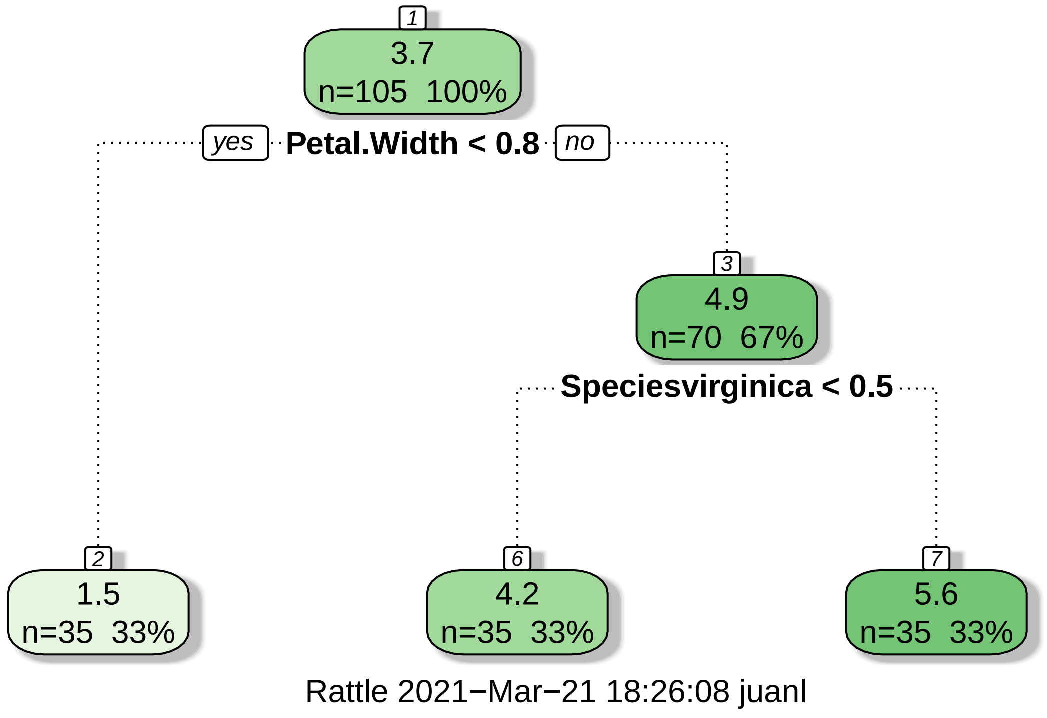


Figure 5: Classification Tree, using the `rattle` package, of species in the iris data set by `Petal.Width` and `Sepal.Width`.

1.6.1 Predicting new values

```
predict(modFit, newdata = testing)
```

```
##        1        7       10       11       13       17       18       19
## 1.460000 1.460000 1.460000 1.460000 1.460000 1.460000 1.460000 1.460000
##       23       25       27       33       36       43       45       51
## 1.460000 1.460000 1.460000 1.460000 1.460000 1.460000 1.460000 4.214286
##       54       55       57       61       64       70       71       74
## 4.214286 4.214286 4.214286 4.214286 4.214286 4.214286 4.214286 4.214286
##       77       78       81       83       84       94      105      114
## 4.214286 4.214286 4.214286 4.214286 4.214286 4.214286 5.571429 5.571429
##      115      118      123      125      134      138      140      142
## 5.571429 5.571429 5.571429 5.571429 5.571429 5.571429 5.571429 5.571429
##      143      145      148      149      150
## 5.571429 5.571429 5.571429 5.571429 5.571429
```

RMSE:

```
sqrt(sum((predict(modFit, newdata = testing) - testing$Petal.Length)^2))
```

```
## [1] 3.166682
```

Regression between actual and predicted values:

```
ggplot(testing, aes(x = Petal.Length, y = predict(modFit, newdata = testing))) +
  geom_smooth(method = "lm",
              color = "red") +
  geom_point(alpha = 0.3,
```

```r
             color = "red") +
  labs(x = "Actual petal length",
       y = "Predicted petal length") +
  stat_cor(aes(label = paste(..rr.label..,
                             cut(..p..,
                                 breaks = c(-Inf,
                                            0.0001,
                                            0.001,
                                            0.01,
                                            0.05,
                                            Inf),
                                 labels = c("'****'",
                                            "'***'",
                                            "'**'",
                                            "'*'",
                                            "''")),
                             sep = "~")),
           color = "black") +
  theme_pubclean()
```

```
## `geom_smooth()` using formula 'y ~ x'
```

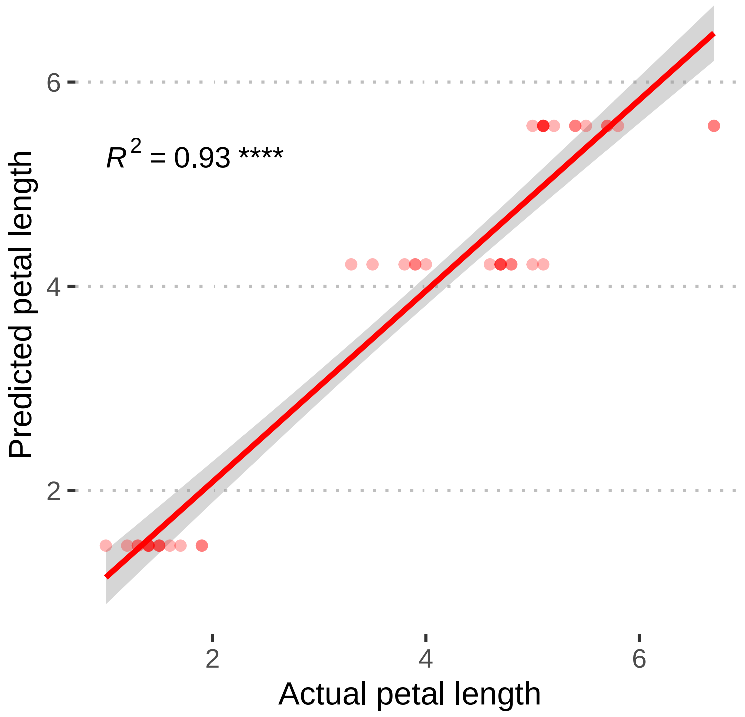


Figure 6: Linear regression between actual and predicted `Petal.Length` from the model applied to testing dataset, with 95% prediction intervals.

1.7 Notes and further resources

- Classification trees are non-linear models
  - They use interactions between variables

  - Data transformations may be less important (monotone transformations)
  - Trees can also be used for regression problems (continuous outcome)
    - Then you can use RMSE as a measure of impurity

- Note that there are multiple tree building options in R both in the caret package - `party`, `rpart` and out of the caret package - `tree`
- Introduction to statistical learning (James et al., 2013)
- Elements of Statistical Learning (Hastie et al., 2009)
- Classification and regression trees (Breiman et al., 1984; Krzywinski & Altman, 2017)

---

2 Lecture 2: Bagging (*bootstrap aggregating*)

This lecture is about bagging, which is short for bootstrap aggregating. The basic idea is that when you fit complicated models, sometimes if you average those models together, you get a smoother model fit, that gives you a better balance between potential bias in your fit and variance in your fit.

2.1 Bootstrap aggregating (bagging)

Basic idea:

1. Resample cases and recalculate predictions
2. Average or majority vote

Notes:

- Similar bias
- Reduced variance
- More useful for non-linear functions

2.2 Ozone data

```
library(ElemStatLearn)

data(ozone,package="ElemStatLearn")

ozone <- ozone[order(ozone$ozone),]
head(ozone)
```

```
##     ozone radiation temperature wind
## 17      1         8          59  9.7
## 19      4        25          61  9.7
## 14      6        78          57 18.4
## 45      7        48          80 14.3
## 106     7        49          69 10.3
## 7       8        19          61 20.1
```

http://en.wikipedia.org/wiki/Bootstrap_aggregating

2.2.1 Bagged loess

Here, we resample the data set ten different times, using a loop over ten different samples of the data set. Each time I'm going to sample **with replacement** from the entire data set.

Then, create a new data set, `ozone0`, which is the resample data set for that particular element of the loop. And that is the subset of the data set corresponding to our random sample.

Then, reorder the data set every time by the ozone variable (why will be clear later).

Then, fit a loess curve each time (a smooth curve that you can fit through the data, very similar to the `spline` model fits that we saw in a previous example with modelling with linear regression).

The basic idea is fitting a smooth curve relating temperature, to the ozone variables. Temperature is the outcome, and ozone is the predictor, and each time the resample data set is used as the data set to build that predictor on.

We use a common `span` for each time, the span being a measure of how smooth that fit will be.

Then, predict for every single loess curve the outcome for a new data set for the exact same values. Here, always predict for `ozone` values 1 to 155. So the i^{th} row of this `ll` object is now the prediction from the loess curve, from the i^{th} resample of the date ozone.

```r
ll <- matrix(NA, nrow = 10, ncol = 155)

for(i in 1:10){
  ss <- sample(1:dim(ozone)[1],
               replace = T)
  ozone0 <- ozone[ss,]
  ozone0 <- ozone0[order(ozone0$ozone),]
  loess0 <- loess(temperature ~ ozone,
                  data=ozone0,
                  span=0.2)
  ll[i,] <- predict(loess0,
                    newdata = data.frame(ozone = 1:155))
}
```

In short: resampled the data set ten different times, fit a smooth curve through it those ten different times, and then what we are going to average those values.

```r
plot(ozone$ozone, ozone$temperature, pch = 19, cex = 0.5)
for (i in 1:10){lines(1:155,
                    ll[i,],
                    col = "grey",
                    lwd = 2)}
lines(1:155, apply(ll, 2, mean),
      col = "red",
      lwd = 2)
```

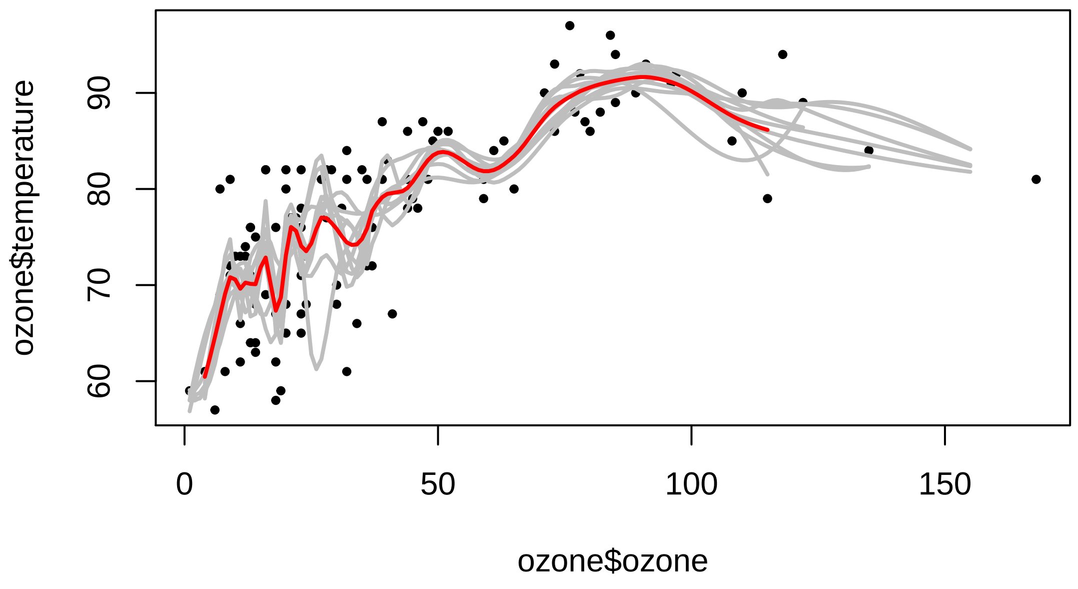


Figure 7: Bagged loess plot in base R. The gery lines represent each model fitted to each resample, and the red line represents the average of those models.

Or, in `ggplot2`, using the function `autoplot` from the package `zoo`:

```
#ll <- matrix(NA, nrow = 10, ncol = 155)
#
#for(i in 1:10){
#  ss <- sample(1:dim(ozone)[1],
#               replace = T)
#  ozone0 <- ozone[ss,]
#  ozone0 <- ozone0[order(ozone0$ozone),]
#  loess0 <- loess(temperature ~ ozone,
#                  data=ozone0,
#                  span=0.2)
#  ll[i,] <- predict(loess0,
#                    newdata = data.frame(ozone = 1:155))
#}

library(zoo)
x <- 1:155
z <- zoo(t(ll), x)  # use t so that series are columns

autoplot(z, facet = NULL) +
  geom_line(color = "grey", size = 1, akpha = 0.6) +
  geom_point(data = ozone, aes(x= ozone, y = temperature),
             colour = "black",
             alpha = 0.5) +
  geom_line(data = data.frame(x = 1:155, y = colMeans(ll)),
            aes(x = x, y = y),
            color = "red",
            size = 1) +
  scale_y_continuous(sec.axis = sec_axis(~ (.-32)*(5/9),
                     name = "Daily maximum temperature (°C)")) +
  labs(x = "Ozone concentration (ppb)", y = "Daily maximum temperature (°F)") +
  theme_pubclean() +
  theme(legend.position = "none")
```

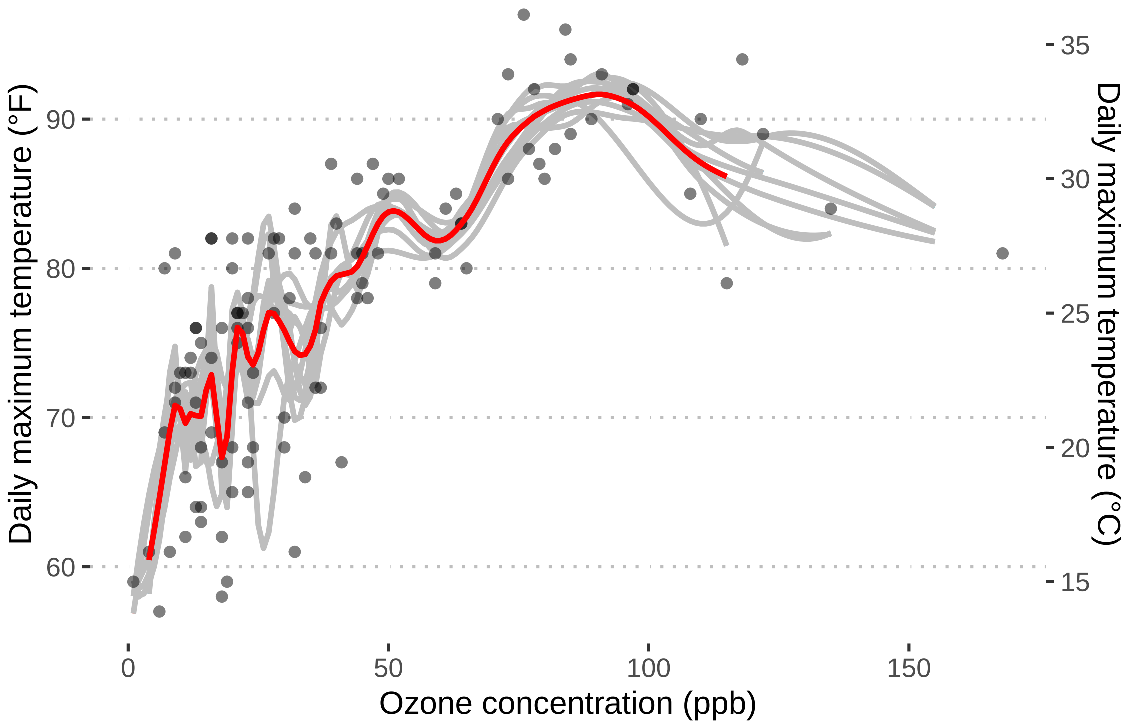


Figure 8: Bagged loess plot in `ggplot2`, using the `autoplot` function from the `zoo` package. Again, the gery lines represent each model fitted to each resample, and the red line represents the average of those models. Y axis is in both °F and °C (something I wanted to try). ppb = parts per billon (equivalent to µg/kg).

2.3 Bagging in `caret`

- Some models perform bagging for you, in `train` function consider `method` options
  - `bagEarth`
  - `treebag`
  - `bagFDA`
- Alternatively you can bag any model you choose using the `bag` function

2.3.1 More bagging in `caret`

```
library(caret) #necessary to avoid an error
predictors <- data.frame(ozone = ozone$ozone) #create data.frame with predictors
temperature <- ozone$temperature #object with DV
treebag <- caret::bag(predictors,
               temperature, #outcome
               B = 10, # number of subsamples
               bagControl = bagControl(fit = ctreeBag$fit,
                                       predict = ctreeBag$pred,
                                       aggregate = ctreeBag$aggregate))
```

A default plot

```r
plot(ozone$ozone, temperature, col = "lightgrey",
     pch=19)
points(ozone$ozone, predict(treebag$fits[[1]]$fit, predictors),
       pch=19,
       col="red")
points(ozone$ozone, predict(treebag, predictors),
       pch=19,
       col="blue")
```

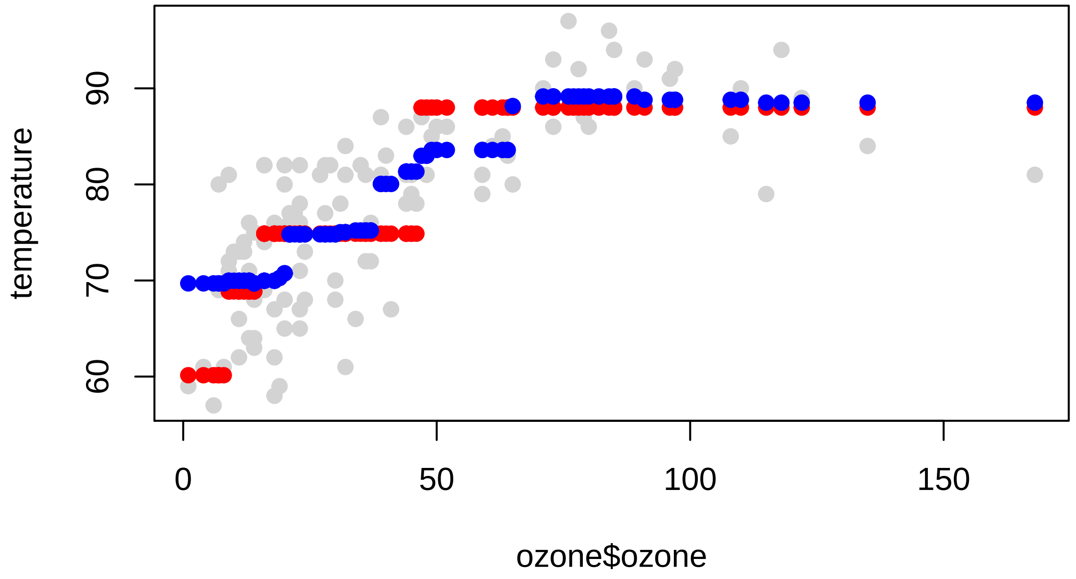


Figure 9: The red dots represent the fit from a single conditional regression tree. It does not capture the trend very well (first dots are just flat, even though there appears to be a trend upward in the data points). The red dots represent the average over ten different bagged model fits with these conditional regression trees. There is an increase here in the values in the blue fit, which is the fit from the bagged regression.

Or, in `ggplot2`:

```r
ggplot(ozone, aes(x = ozone, y = temperature)) +
  geom_point(color = "lightgrey") +
  geom_point(aes(x = ozone, y = predict(treebag$fits[[1]]$fit, predictors)),
             color = "red",
             alpha = 0.6) +
  geom_point(aes(x = ozone, y = predict(treebag, predictors)),
             color = "blue",
             alpha = 0.6) +
  scale_y_continuous(sec.axis = sec_axis(~ (.-32)*(5/9),
                                         name = "Daily maximum temperature (°C)")) +
  labs(x = "Ozone concentration (ppb)", y = "Daily maximum temperature (°F)") +
  theme_pubclean() +
  theme(legend.position = "none")
```

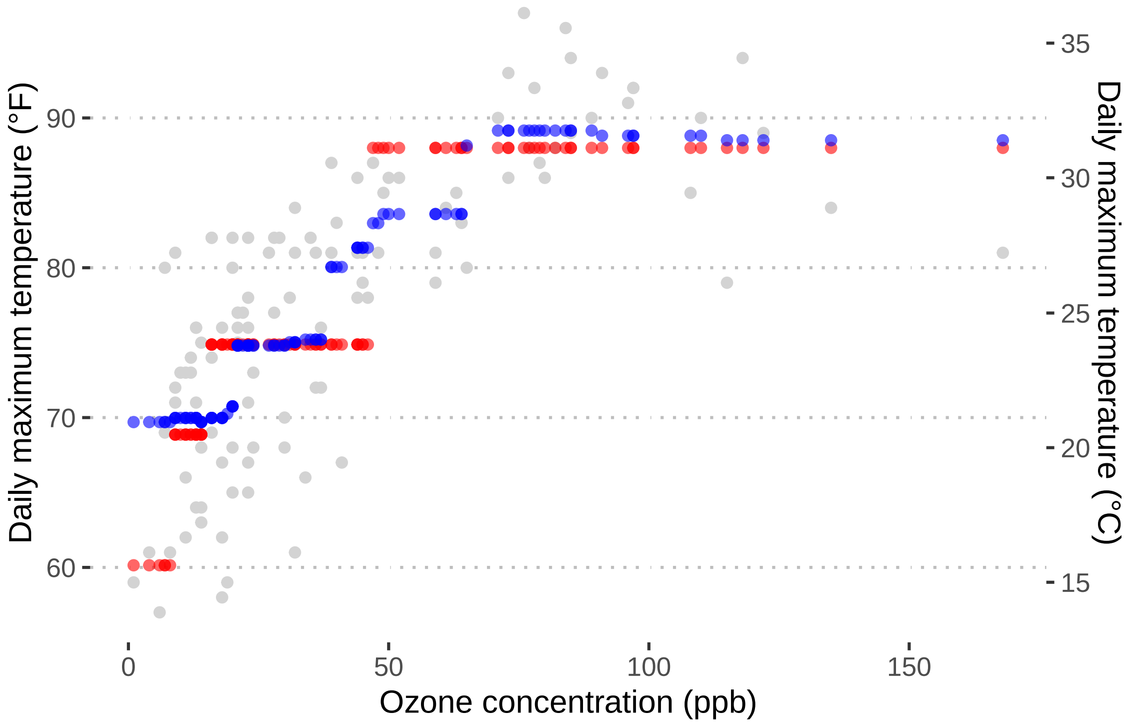


Figure 10: The red dots represent the fit from a single conditional regression tree. It does not capture the trend very well (first dots are just flat, even though there appears to be a trend upward in the data points). The red dots represent the average over ten different bagged model fits with these conditional regression trees. There is an increase here in the values in the blue fit, which is the fit from the bagged regression.

2.4 Parts of the `bag` function

```
ctreeBag$fit
```

```
## function (x, y, ...)
## {
##     loadNamespace("party")
##     data <- as.data.frame(x, stringsAsFactors = TRUE)
##     data$y <- y
##     party::ctree(y ~ ., data = data)
## }
## <bytecode: 0x00000000273c0b08>
## <environment: namespace:caret>
```

```
ctreeBag$pred
```

```
## function (object, x)
## {
##     if (!is.data.frame(x))
##         x <- as.data.frame(x, stringsAsFactors = TRUE)
##     obsLevels <- levels(object@data@get("response")[, 1])
##     if (!is.null(obsLevels)) {
##         rawProbs <- party::treeresponse(object, x)
##         probMatrix <- matrix(unlist(rawProbs), ncol = length(obsLevels),
##             byrow = TRUE)
##         out <- data.frame(probMatrix)
##         colnames(out) <- obsLevels
##         rownames(out) <- NULL
##     }
##     else out <- unlist(party::treeresponse(object, x))
##     out
## }
## <bytecode: 0x00000000273c1550>
## <environment: namespace:caret>
```

```
ctreeBag$aggregate
```

```
## function (x, type = "class")
## {
##     if (is.matrix(x[[1]]) | is.data.frame(x[[1]])) {
##         pooled <- x[[1]] & NA
##         classes <- colnames(pooled)
##         for (i in 1:ncol(pooled)) {
##             tmp <- lapply(x, function(y, col) y[, col], col = i)
##             tmp <- do.call("rbind", tmp)
##             pooled[, i] <- apply(tmp, 2, median)
##         }
##         if (type == "class") {
##             out <- factor(classes[apply(pooled, 1, which.max)],
##                 levels = classes)
##         }
##         else out <- as.data.frame(pooled, stringsAsFactors = TRUE)
##     }
##     else {
##         x <- matrix(unlist(x), ncol = length(x))
##         out <- apply(x, 1, median)
##     }
##     out
## }
## <bytecode: 0x00000000273bf748>
## <environment: namespace:caret>
```

2.5 Notes and further resources

Notes:

- Bagging is most useful for nonlinear models
- Often used with trees - an extension is random forests
- Several models use bagging in caret's `train` function

The basic idea is to resample your data, refit your nonlinear model, then average those model fits together over resamples to get a smoother model fit, than you would've got from any individual fit on its own.

Further resources:

- Bagging
- Bagging and boosting
- Elements of Statistical Learning (Hastie et al., 2009)

3 Lecture 3: Random Forests

This lecture is about random forests, which you can think of as an extension to bagging for classification and regression trees.

The basic idea is very similar to bagging in the sense that we bootstrap samples, so we take a resample of our observed data, and our training data set. And then we rebuild classification or regression trees on each of those bootstrap samples.

3.1 Random forests

1. Bootstrap samples
2. At each split, bootstrap variables
3. Grow multiple trees and vote or average those trees

Pros:

1. Accuracy

Cons:

1. Speed
2. Interpretability
3. Overfitting
   - an be complicated by the fact that it's very hard to understand which trees are leading to that overfitting, and so it is very important to use cross validation when building random forests

```
knitr::include_graphics("ranforests.png")
```

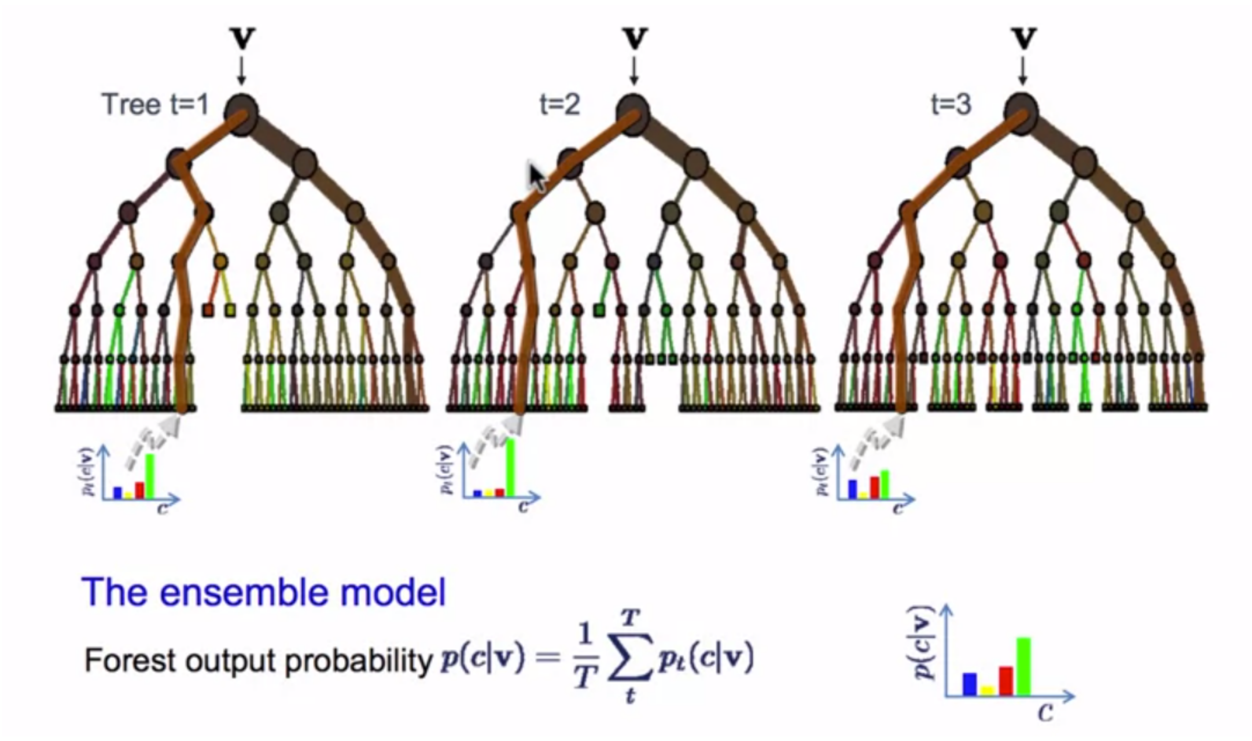

v
Tree t=1
t=2
t=3
$p_t(c|\mathbf{v})$
c
The ensemble model
Forest output probability $p(c|\mathbf{v}) = \frac{1}{T}\sum_t^T p_t(c|\mathbf{v})$
$p(c|\mathbf{v})$

Figure 11: Example of random forests: the ensemble model.

The idea is that you build a large number of trees where each tree is based on a bootstrap sample.

So, for example, the first tree is built on a random subsample of the data. And then at each node we allow a different subset of the variables to potentially contribute to the splits.

- Then if we get a new observation, say V, we run that observation through tree one, and it ends up at a leaf down here at the bottom of that tree, and so it gets a particular prediction here.
- Then, we take that same observation V, we run it through the next tree, and it goes down a slightly different leaf, and it gets a slightly different set of predictions.
- And finally we go down the third tree, and we get an even different set of predictions.

Then what we do is we basically average those predictions together in order to get the predictive probabilities of each class across all the different trees.

$$p(c|v) = \frac{1}{T}\sum_{t}^{T} p_t(c|v)$$

3.2 Example: `iris` data

```
library(ggplot2)

data(iris)

inTrain <- createDataPartition(y = iris$Species,
                               p = 0.7,
                               list = FALSE)
training <- iris[inTrain,]
testing <- iris[-inTrain,]
```

In the caret package, use the `train` function just like for the other model building, send the training data set `method = "rf"`, which is the random forest method.

```
modFit <- train(Species~ .,
                data = training,
                method = "rf",
                prox = TRUE)
modFit
```

```
## Random Forest
##
## 105 samples
##   4 predictor
##   3 classes: 'setosa', 'versicolor', 'virginica'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 105, 105, 105, 105, 105, 105, ...
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##   2     0.9465620  0.9185302
##   3     0.9465364  0.9184600
##   4     0.9423798  0.9122692
##
```

```
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 2.
```

The tuning parameter in particular is the number of trees, or number of repeated trees (`mtry`) that it's going to build.

3.2.1 Getting a single tree

With the code below, for example, I will get the information of the second (`k = 2`) tree.

```
library(randomForest)
```

```
## randomForest 4.6-14

## Type rfNews() to see new features/changes/bug fixes.

##
## Attaching package: 'randomForest'

## The following object is masked from 'package:rattle':
##
##     importance

## The following object is masked from 'package:ggplot2':
##
##     margin
```

```
getTree(modFit$finalModel,
        k = 2)
```

```
##   left daughter right daughter split var split point status prediction
## 1             2              3         4        0.80      1          0
## 2             0              0         0        0.00     -1          1
## 3             4              5         4        1.65      1          0
## 4             6              7         3        4.95      1          0
## 5             0              0         0        0.00     -1          3
## 6             0              0         0        0.00     -1          2
## 7             8              9         3        5.05      1          0
## 8             0              0         0        0.00     -1          3
## 9             0              0         0        0.00     -1          2
```

- Each of these rows corresponds to a particular split
- Then:
  - Left daughter of the tree
  - Right daughter of the tree
- Then, which variable we're splitting on (`split var`)
- what's the value where that variable is split (`split point`)
- what the prediction is going to be out of that particular split

3.2.2 Class "center" (`classCenter`)

```
irisP <- classCenter(training[,c(3,4)],
                     training$Species,
                     modFit$finalModel$prox)
irisP <- as.data.frame(irisP)
irisP$Species <- rownames(irisP)
```

```r
ggplot(training, aes(x = Petal.Width, y = Petal.Length, colour = Species)) +
  geom_point(alpha = 0.4) +
  geom_point(size = 7,
             shape = 4,
             data = irisP) +
  theme_pubclean()
```

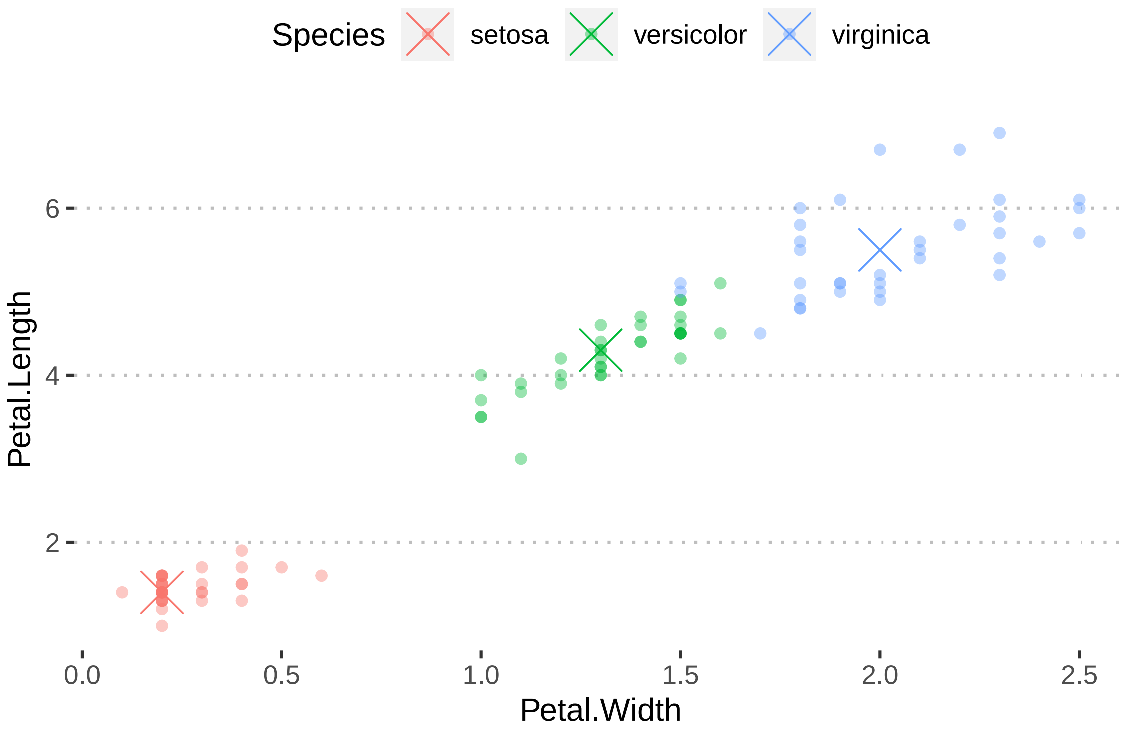


Figure 12: Classification of species in the iris data set by `Petal.Width` and `Petal.Length`. Colored X represent centres for the predicted values. There are distinct clusters for each species.

3.2.3 Predict new values

```r
pred <- predict(modFit, testing)

#to see if prediction is equal to the testing data set species
testing$predRight <- pred == testing$Species
table(pred, testing$Species)
```

```
##
## pred         setosa versicolor virginica
##   setosa         15          0         0
##   versicolor      0         13         1
##   virginica       0          2        14
```

There are a couple of values that were missed (or, misclassified). To look at them:

```r
ggplot(testing, aes(x = Petal.Width, y = Petal.Length, colour = predRight)) +
  geom_point(alpha = 0.6) +
  theme_pubclean()
```

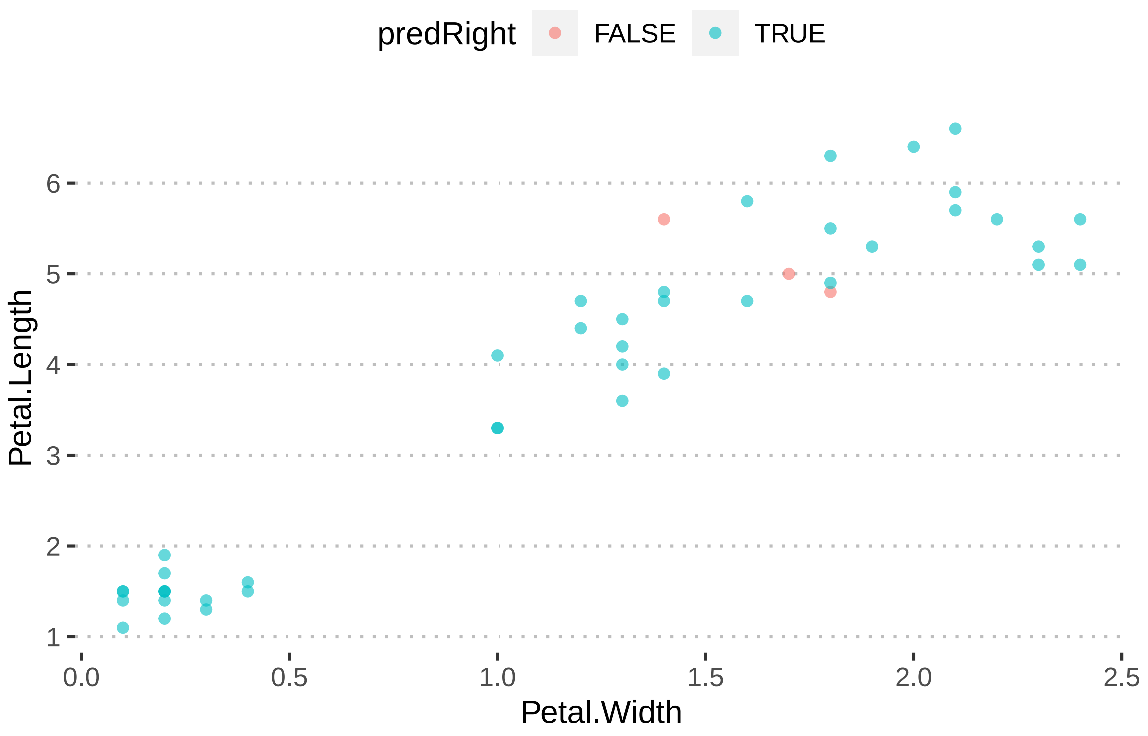


Figure 13: Newdata predictions. Red points represent misclassified values.

3.3 Notes and further resources

Notes:

- Random forests are usually one of the two top performing algorithms along with boosting in prediction contests.
- Random forests are difficult to interpret but often very accurate.
- Care should be taken to avoid overfitting (see `rfcv` function)
  - check out the `rfcv` function to make sure that cross validation is being performed, but the `train` function in `caret` also handles that for you

Further resources:

- Random forests
- Random forest Wikipedia
- Elements of Statistical Learning (Hastie et al., 2009)

4 Lecture 4: Boosting

This lecture is about Boosting, which along with random forest, is one of the most accurate out of the box classifiers that you can use.

The basic idea here is:

- take a large number of possibly weak predictors,
- take those possibly weak predictors

- weight them (in a way that takes advantage of their strengths), and add them up.
- Get a stronger predictor

4.1 Basic idea behind boosting

1. Start with a set of classifiers $h_1, \ldots, h_k$
   - Examples: All possible trees, all possible regression models, all possible cutoffs.

- Create a classifier that combines classification functions: $f(x) = \text{sgn}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$.
  - Goal is to minimize error (on training set)
  - Iterative, select one h at each step
  - Calculate weights based on errors
  - Upweight missed classifications and select next h

The most famous boosting algorithm is Adaboost (Adaboost on Wikipedia)

http://www.boosting.org/papers/MeiRae03.pdf

4.1.1 Simple example

Suppose we're trying to separate the blue plus signs, from the red minus signs, and we have two variables to predict with.

Here,variable one is plotted on the x-axis, and variable two on the y-axis (Fig. 14).

```
knitr::include_graphics("ada1.png")
```

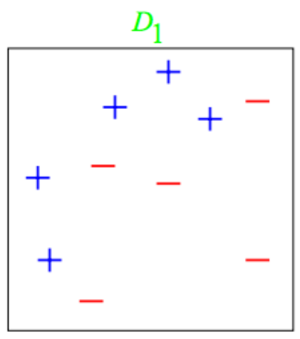


Figure 14: Example of boosting.

4.1.1.1 Round 1: adaboost We could start off with a really simple classifier (Fig. 15).

We could say just draw a line a vertical line that separates these points well. Here is a classifier that says anything to the left of this vertical line is a blue plus, and anything to the right is a red minus.

However, we have misclassified these three points in the top right.

So the thing that we would do is build that classifier, calculate the error rate, in this case we're missing about 30% of the points.

And then we would **upweight** those points that we missed. Here I've shown that **upweighting**, by drawing them in a larger scale. So those pluses are now **upweighted**, for building the next classifier.

```
knitr::include_graphics("adar1.png")
```

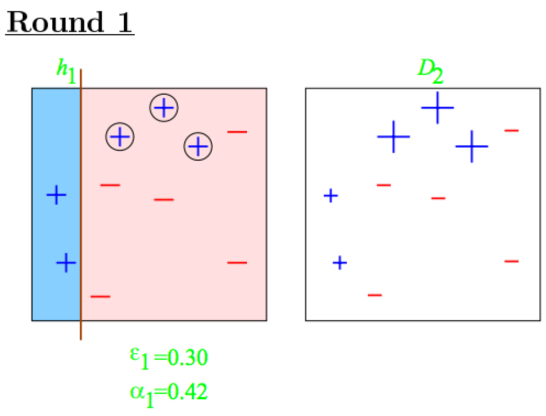


Figure 15: Example of boosting: round 1.

4.1.1.2 Rounds 2 & 3: adaboost We would then build the next classifier.

Our second classifier would be one that drew a vertical line on the right, so that this vertical line would classify everything to the right of that line as a red minus, and everything to the left, as a blue plus.

And so here we again, misclassified three points, and those three points are now **upweighted**, and they are also drawn larger for the next iteration.

So we can again calculate the error rate, and use that to calculate the weights for the next step.

```
knitr::include_graphics("ada2.png")
```

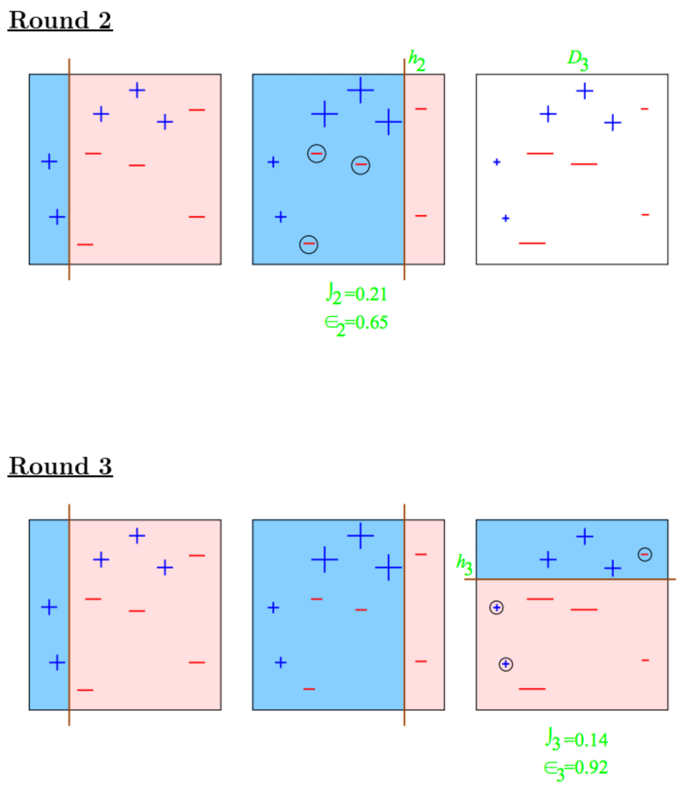


Figure 16: Example of boosting: rounds 2 and 3.

Then the third classifier, will intentionally try to classify those points that we misclassified in the last couple of rounds.

So, for example, the pluses and minuses previously misclassified need to be correctly classified.

To do that we now draw a horizontal line, and we say anything below that horizontal line is a red minus, anything above is a blue plus, and now we misclassify a minus on the top, and two points towards the bottom.

4.1.1.3 Completed classifier

We can add those classifiers, by adding their errors ($\in_i$) times the classification given by each line (Fig. 17).

```
knitr::include_graphics("ada3.png")
```

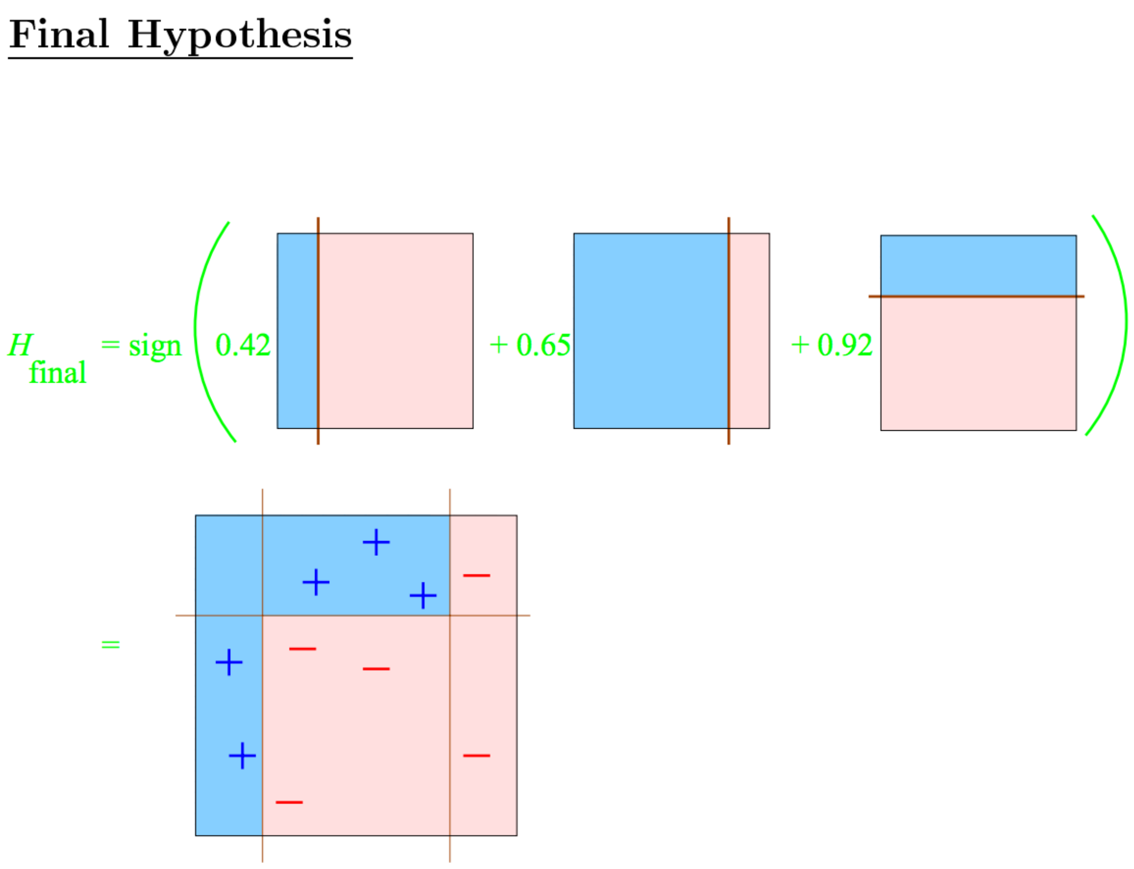

Final Hypothesis
$H_{\text{final}} = \text{sign}\left(0.42 \ldots + 0.65 \ldots + 0.92 \ldots\right)$
=
+
−

Figure 17: Example of boosting: complete classifier.

Once you add these classification rules up together, you can see that our classifier works much better now.

We get a much better classification when adding them up, that correctly classifies all of the blue pluses, and all of the red minuses together.

4.2 Boosting in R

- Boosting can be used with any subset of classifiers
- One large subclass is gradient boosting
- R has multiple boosting libraries. Differences include the choice of basic classification functions and combination rules.
  - gbm - boosting with trees.
  - mboost - model based boosting
  - ada - statistical boosting based on additive logistic regression
  - gamBoost for boosting generalized additive models
- Most of these are available in the caret package

4.3 Wage example

```r
library(ISLR)
data(Wage)
library(ggplot2)
library(caret)

Wage <- subset(Wage,select = -c(logwage)) #to remove the predictor that we care
inTrain <- createDataPartition(y = Wage$wage,
                               p = 0.7,
                               list = FALSE)
training <- Wage[inTrain,]
testing <- Wage[-inTrain,]
```

4.3.1 Fit the model

`verbose = FALSE` helps to avoid a lot of output when using `method = "gbm"`.

```r
modFit <- train(wage ~.,
                method = "gbm",
                data = training,
                verbose = FALSE)
modFit
```

```
## Stochastic Gradient Boosting
##
## 2102 samples
##    9 predictor
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 2102, 2102, 2102, 2102, 2102, 2102, ...
## Resampling results across tuning parameters:
##
##   interaction.depth  n.trees  RMSE      Rsquared   MAE
##   1                   50      34.43044  0.3197279  23.24530
##   1                  100      33.89264  0.3274786  22.84695
##   1                  150      33.82006  0.3287792  22.83067
##   2                   50      33.95516  0.3260213  22.85102
##   2                  100      33.87183  0.3269658  22.83140
##   2                  150      33.95933  0.3242069  22.90833
##   3                   50      33.93006  0.3249306  22.81764
##   3                  100      34.04167  0.3211942  22.97824
##   3                  150      34.18985  0.3166787  23.13596
##
## Tuning parameter 'shrinkage' was held constant at a value of 0.1
##
## Tuning parameter 'n.minobsinnode' was held constant at a value of 10
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were n.trees = 150, interaction.depth =
##  1, shrinkage = 0.1 and n.minobsinnode = 10.
```

4.3.2 Plot the results

```r
ggplot(testing, aes(x = predict(modFit, testing), y = wage)) +
  geom_point(alpha = 0.6) +
  stat_cor(aes(label = paste(..rr.label..,
                             cut(..p..,
                                 breaks = c(-Inf,
                                            0.0001,
                                            0.001,
                                            0.01,
                                            0.05,
                                            Inf),
                                 labels = c("'****'",
                                            "'***'",
                                            "'**'",
                                            "'*'",
                                            "''")),
                             sep = "~")),
           color = "black") +
  theme_pubclean()
```

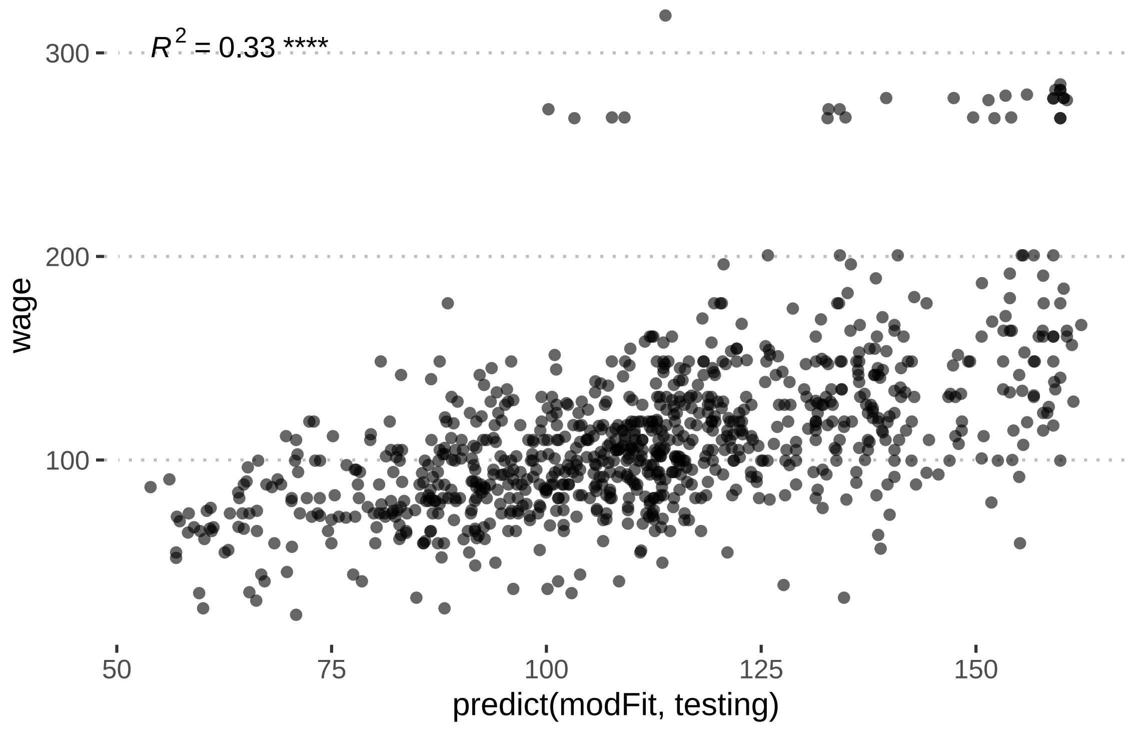


Figure 18: Newdata predictions. Red points represent misclassified values.

4.4 Notes and further reading

- A couple of nice tutorials for boosting
  - Freund and Shapire - http://www.cc.gatech.edu/~thad/6601-gradAI-fall2013/boosting.pdf
  - Meir and Rätsch - <http://www.boosting.org/papers/MeiRae03.pdf>
- Boosting, random forests, and model ensembling are the most common tools that win Kaggle and other

prediction contests.

- http://www.netflixprize.com/assets/GrandPrize2009_BPC_BigChaos.pdf
- https://kaggle2.blob.core.windows.net/wiki-files/327/09ccf652-8c1c-4a3d-b979-ce2369c985e4/Willem%20Mestrom%20-%20Milestone%201%20Description%20V2%202.pdf

---

5 Lecture 5: Model Based Prediction

This lecture's about model based prediction. The basic idea here is that we're going to assume the data follow a specific probabilistic model. Then we're going to use Bayes' theorem to identify optimal classifiers based on that probabilistic model.

The advantage is that this approach can take advantage of some structure that might appear in the data. For example the fall of distribution. And that may lead to some computational conveniences. There may also be reasonable accurate on real problems, particularly the real problems that appear to follow the data distribution that underlies our whole holistic model.

5.1 Basic idea

1. Assume the data follow a probabilistic model
2. Use Bayes' theorem to identify optimal classifiers

Pros:

- Can take advantage of structure of the data
- May be computationally convenient
- Are reasonably accurate on real problems

Cons:

- Make additional assumptions about the data
  - These assumptions don't have to be exactly satisfied in order for the prediction algorithms to work very well. But if they're very far off, the algorithms may fail
- When the model is incorrect you may get reduced accuracy

5.2 Model based approach

1. Our goal is to build parametric model for conditional distribution $P(Y = k|X = x)$
   - For the conditional distribution the probability that Y our outcome equals some specific class k given a particular set of predictor variables so that our X variables equal the little value of x.
2. A typical approach is to apply Bayes theorem:

$$Pr(Y = k|X = x) = \frac{Pr(X = x|Y = k)Pr(Y = k)}{\sum_{\ell=1}^{K} Pr(X = x|Y = \ell)Pr(Y = \ell)}$$

   - In other words we want to know something about the probability Y equals k (that Y comes from class k) given the variables that we've observed. We write that down using Bayes' theorem as the probability $X = x$ given $Y = k$, times the probability $Y = k$, divided by the law of total probability here below.

$$Pr(Y = k|X = x) = \frac{f_k(x)\pi_k}{\sum_{\ell=1}^{K} f_\ell(x)\pi_\ell}$$

- We then assume some parametric model for the distribution of the features given the class ($f_k(x)$), and we assume a prior that each particular element comes from a specific class (π_k).
- Then we can basically model the distribution, the probability that $Y = k$ given a particular set of predictor variables the fraction here where we have a model for the x variables and a model for the prior probability.

3. Typically prior probabilities π_k are set in advance.
4. A common choice for $f_k(x) = \frac{1}{\sigma_k\sqrt{2\pi}}e^{-\frac{(x-\mu_k)^2}{\sigma_k^2}}$, a Gaussian distribution
   - It may be a multivariate Gaussian distribution if there are multiple x variables
5. Estimate the parameters (μ_k,σ_k^2) from the data.
6. Classify to the class with the highest value of $P(Y = k|X = x)$

5.3 Classifying using the model

A range of models use this approach

- Linear discriminant analysis assumes $f_k(x)$ is multivariate Gaussian with same covariances
- Quadratic discriminant analysis assumes $f_k(x)$ is multivariate Gaussian with different covariances
- Model based prediction assumes more complicated versions for the covariance matrix
- Naive Bayes assumes independence between features for model building

Elements of Statistical Learning (Hastie et al., 2009)

5.4 Why linear discriminant analysis?

$$log\frac{Pr(Y = k|X = x)}{Pr(Y = j|X = x)}$$

$$= log\frac{f_k(x)}{f_j(x)} + log\frac{\pi_k}{\pi_j}$$

$$= log\frac{\pi_k}{\pi_j} - \frac{1}{2}(\mu_k + \mu_j)^T\Sigma^{-1}(\mu_k + \mu_j)$$

$$+x^T\Sigma^{-1}(\mu_k - \mu_j)$$

Basically, a variable will have a higher probability of one class if it's on one side of the line and a higher probability of being in another class if it's on the other side of the line. For more detailed info, check Elements of Statistical Learning (Hastie et al., 2009).

5.5 Decision boundaries

This is what the decision boundaries tend to look like, for these prediction models.

```
knitr::include_graphics("ldaboundary.png")
```

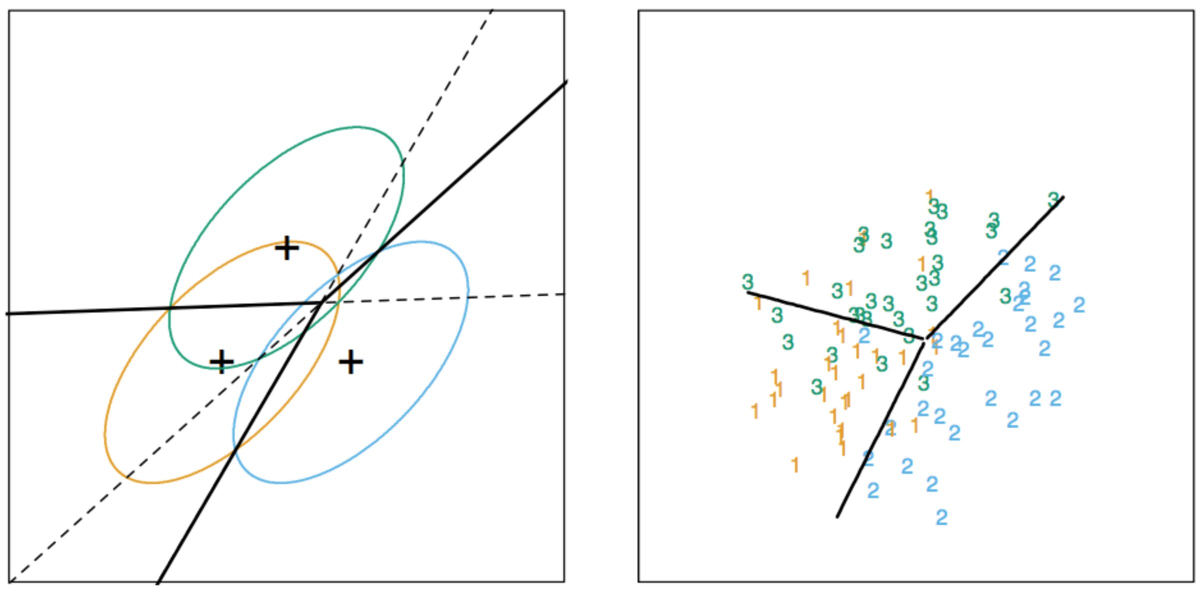


Figure 19: Example of decision boundaries.

Imagine we have three different groups of points. We are trying to classify into class one, class two or class three, and we have two variables that we're using to classify and that's the x and the y axis.

What would end up, what we would end up doing is fitting one Gaussian distribution (each *circle* on the left panel of Fig 19).

We would basically draw lines where the probability switches over from being higher in this class to that class.

Basically: you fit Gaussian distributions to the data and then use those Gaussian distributions to draw lines that assign the prop points to the highest posterior probabilities.

5.6 Discriminant function

$$\delta_k(x) = x^T\Sigma^{-1}\mu_k - \frac{1}{2}\mu_k\Sigma^{-1}\mu_k + log(\mu_k)$$

- Decide on class based on $\hat{Y}(x) = argmax_k\delta_k(x)$
- We usually estimate parameters with maximum likelihood

5.7 Naive Bayes

Suppose we have many predictors, we would want to model: $P(Y = k|X_1, \dots, X_m)$

We could use Bayes Theorem to get:

$$\begin{aligned} P(Y = k|X_1, \dots, X_m) &= \frac{\pi_k P(X_1, \dots, X_m|Y = k)}{\sum_{\ell=1}^{K} P(X_1, \dots, X_m|Y = k)\pi_\ell} \\ &\propto \pi_k P(X_1, \dots, X_m|Y = k) \end{aligned}$$

This can be written:

$$P(X_1, \dots, X_m, Y = k) = \pi_k P(X_1|Y = k)P(X_2, \dots, X_m|X_1, Y = k)$$

$$= \pi_k P(X_1|Y=k)P(X_2|X_1,Y=k)P(X_3,\dots,X_m|X_1,X_2,Y=k)$$
$$= \pi_k P(X_1|Y=k)P(X_2|X_1,Y=k)\dots P(X_m|X_1\dots,X_{m-1},Y=k)$$

We could make an assumption to write this:

$$\approx \pi_k P(X_1|Y=k)P(X_2|Y=k)\dots P(X_m|,Y=k)$$

Naive Bayes: One assumption you could make to make this quite a bit easier would be to just assume that all of the predictor variables are independent of each other.

In which case they drop out of this conditioning argument, and you end up with the prior probability times the probability of each feature by itself conditional on being in each class.

Now this is kind of a naive assumption because we're assuming that all the features are independent even though we know they're probably not. And that's why this method has the title **Naive Bayes**.

It still works reasonably well in a large number of applications. And it's particularly useful when you have a very large number of features that are, binary or are categorical variables. This very frequently comes up in text classification and classification of other kind of document classification.

5.8 Example: iris

```
data(iris)
library(ggplot2)

table(iris$Species)
```

```
##
##     setosa versicolor  virginica
##         50         50         50
```

5.8.1 Create training and test sets

```
inTrain <- createDataPartition(y = iris$Species,
                               p = 0.7,
                               list = FALSE)
training <- iris[inTrain,]
testing <- iris[-inTrain,]
dim(training)
```

```
## [1] 105   5
```

```
dim(testing)
```

```
## [1] 45  5
```

5.8.2 Build predictions

`method = "lda"` for **Linear discriminant analysis (LDA)**, or `method = "nb"` for **Naive Bayes**.

```
modlda <- train(Species ~ .,
               data = training,
               method="lda")

modnb <- train(Species ~ .,
              data = training,
               method = "nb")

plda <- predict(modlda,testing)
pnb <- predict(modnb,testing)
table(plda,pnb)
```

```
##              pnb
## plda          setosa versicolor virginica
##   setosa          15          0         0
##   versicolor       0         15         0
##   virginica        0          1        14
```

Here the two models agree in all but one value.

So even though we know that the features or the predictors are dependent here, using the naive Bayes classification yields very similar prediction rules to the linear discriminate analysis classification. The two outputs but overall they perform very similarly.

5.8.3 Comparison of results

```
equalPredictions <- (plda == pnb)

ggplot(testing, aes(x = Petal.Width, y = Sepal.Width, colour = equalPredictions)) +
  geom_point(alpha = 0.6) +
  theme_pubclean()
```

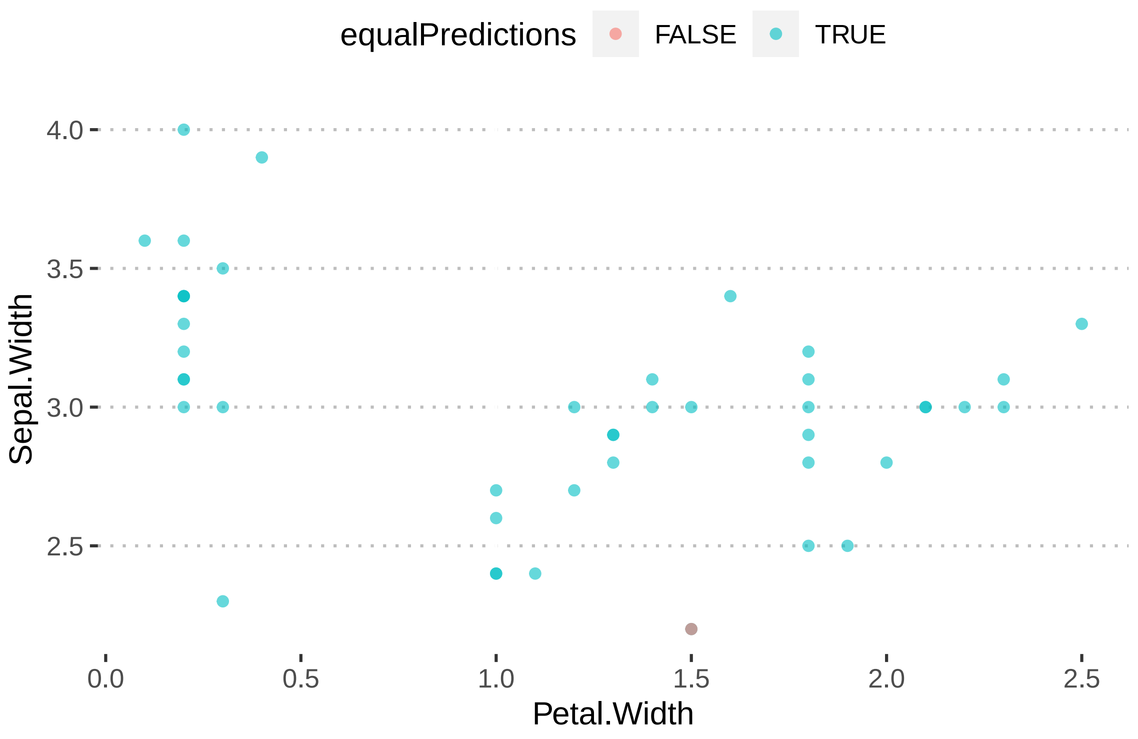


Figure 20: Comparison between predictions made using LDA and Naive Bayes. Red points represent values without the same classification.

5.9 Notes and further reading

- Introduction to statistical learning (James et al., 2013)
- Elements of Statistical Learning (Hastie et al., 2009)
- Model based clustering (Fraley & Raftery, 2002)
- Linear Discriminant Analysis
- Quadratic Discriminant Analysis

References

Breiman, L., Friedman, J., Stone, C. J., & Olshen, R. A. (1984). *Classification and Regression Trees.* Chapman and Hall/CRC. https://www.routledge.com/Classification-and-Regression-Trees/Breiman-Friedman-Stone-Olshen/p/book/9780412048418

Fraley, C., & Raftery, A. E. (2002). Model-Based Clustering, Discriminant Analysis, and Density Estimation. *Journal of the American Statistical Association*, *97*(458), 611–631. https://doi.org/10.1198/016214502760047131

Hastie, T., Tibshirani, R., & Friedman, J. H. (2009). *The elements of statistical learning: Data mining, inference, and prediction* (2nd ed). Springer. https://web.stanford.edu/~hastie/ElemStatLearn//printings/ESLII_print12_toc.pdf

James, G., Witten, D., Hastie, T., & Tibshirani, R. (2013). *An introduction to statistical learning: With applications in R.* Springer. https://www.statlearning.com/s/ISLR-Seventh-Printing-xwa7.pdf

Krzywinski, M., & Altman, N. (2017). Classification and regression trees. *Nature Methods*, *14*(8), 757–758. https://doi.org/10.1038/nmeth.4370