Algoritma Academy: Classification 2

Samuel Chan

10 January, 2024

Before you go ahead and run the codes in this coursebook, it's often a good idea to go through some initial setup. Under the *Libraries and Setup* tab you'll see some code to initialize our workspace, and the libraries we'll be using for the projects. You may want to make sure that the libraries are installed beforehand by referring back to the packages listed here. Under the *Training Focus* tab we'll outline the syllabus, identify the key objectives and set up expectations for each module.

Background

Algoritma

The following coursebook is produced by the team at Algoritma for its Data Science Academy workshops. The coursebook is intended for a restricted audience only, i.e. the individuals and organizations having received this coursebook directly from the training organization. It may not be reproduced, distributed, translated or adapted in any form outside these individuals and organizations without permission.

Algoritma is a data science education center with bootcamp programs offered in:

- Bahasa Indonesia (Jakarta campus)
- English (Singapore campus)

Lifelong Learning Benefits

If you're an active student or an alumni member, you also qualify for all our future workshops, 100% free of charge as part of your **lifelong learning benefits**. It is a new initiative to help you gain mastery and advance your knowledge in the field of data visualization, machine learning, computer vision, natural language processing (NLP) and other sub-fields of data science. All workshops conducted by us (from 1-day to 5-day series) are available to you free-of-charge, and the benefits **never expire**.

Second Edition

This coursebook is initially written in 2017.

This is the second edition, written in late August 2020. Some of the code has been refactored to work with the latest major version of R, version 4.0. I would like to thank the incredible instructor team at Algoritma for their thorough input and assistance in the authoring and reviewing process.

Libraries and Setup

We'll set-up caching for this notebook given how computationally expensive some of the code we will write can get.

```
knitr::opts_chunk$set(cache=TRUE)
options(scipen = 9999)
```

You will need to use `install.packages()` to install any packages that are not already downloaded onto your machine. You then load the package into your workspace using the `library()` function:

```
library(dplyr)
library(grid)
library(gtools)
library(e1071)
library(tm)
library(SnowballC)
library(ROCR)
library(partykit)
library(caret)
library(class)
library(gmodels)
```

Training Objectives

In this workshop we'll learn to apply probabilities, boosting, bootstrap aggregation, k-fold cross validation, ensembling methods, and a variety of other techniques as we build some of the most widely used machine learning algorithms today. We will also discover the tradeoff between model interpretability and performance, and will learn to exploit the advantages of ensemble-based methods: some of the most competitive machine learning algorithms by the end of day 3.

- **Naive Bayes**
- Law of Probabilities
- Bayes Theorem
- Laplace Smoothing
- Relation to Logistic Regression
- **Decision Tree**
- Application of Decision Tree
- Pre-pruning and Post-pruning
- Splitting Criteria
- Interpretation
- **Ensemble-based Methods**

- Competitive algorithms
- Adaptive Boosting
- **Random Forest**
- Boostrap Aggregation
- Feature Selection
- k-Fold Cross Validation

Naive Bayes

Theory: Law of Probability

In probability theory, supposed we want to know the probability of two events occuring: sunny weather and our local football team winning their next game – if we believe both of these events are unrelated, we would call them **independent events**. Knowing the outcome of one event (say, 40% probability it would be sunny) gives us no additional information about the outcome of the other.

On the flip side are **dependent events**, where the occurence of one event has predictive attributes on the outcome of the other event. For a cheap and overly used example: the word "Lottery" in an email body is predictive of a spam-email. So knowing the occurence of "Lottery" would change our estimated probability of an email being spam.

Do you think the following are dependent events in the case of a loan applicant?

- Event 1: Company downsizing by more than 50% on public records (2% chance)
- Event 2: Declaration of a bankruptcy (3% chance)
- Event 1: Company hiring a new sales manager (30% chance)
- Event 2: Company hiring an additional accountants (40% chance)

When calculating the probability of independent events happening at the same time, we would simply use the equation of: $P(A \cap B) = P(A) * P(B)$

In other words, if we were to estimate the probability of a company **both** hiring a new sales manager and replacing an accountant in a given year, then we simply multiply the two probabilities (0.3 * 0.4 = 0.12). That tells us that there's a 12% chance of both events happening in a given year.

In the case of our spam classifier, say 5 in 100 emails contain the word "lottery" (0.05) and 20 in 100 emails are spam (0.2), then if we mistake the two events as being independent we would obtain a wrong estimate of the joint probability. In other words, if you are asked to estimate the proportion of emails in your inbox where an email is both spam and contain the word lottery, your estimate of 0.01 is likely being too optimistic and far from the actual proportion.

Because P(lottery) and P(spam) are dependent, calculating the joint probability require a formula that correctly describes this relationship, a formula famously known as **Bayes' theorem**.

Conditional probability with Bayes' theorem

Consider the following formula for conditional probability, which is read as *the probability of event A, given that event B occured:*

$P(A|B) = \frac{P(A \cap B)}{P(B)}$

Observe how our estimate of $P(A|B)$ really is $P(A \cap B)$, a measure of how often A and B are observed to occure together, over $P(B)$, a measure of how often B is observed to occur in general. In other words, Bayes' theorem makes the case that our best estimate of $P(A|B)$ is the proportion of trials in which A occured with B out of all the trials in which B occured. It adjusts $P(A \cap B)$ for the probability of B occuring.

If we move the denominator to the left side of the equation and rearrange the formula, we would have:

$P(B \cap A) = P(B|A) * P(A)$

Understanding that $P(A \cap B) = P(B \cap A)$, we can then pluck this formula into the original formula:

$P(A|B) = \frac{P(A \cap B)}{P(B)} = \frac{P(B|A)P(A)}{P(B)}$

In fact, the rewritten formula is a more common representation of the Bayes Theorem in practice, for reasons that will become apparent in the next section.

Revisiting our Lottery and spam example, supposed without knowledge of the email's content (not knowing whether it contains the word "lottery" or not), how would we estimate the probability of this email being "spam"? Naturally, we would say it's 20%, because this is the probability that any prior message is spam. This estimate is known as the **prior probability**.

The probability that the word "lottery" was used in previous spam messages, or $P(lottery|spam)$, is known as the **likelihood**. If I told you that of all 100 spam emails, 4 of them contain the word "lottery", what I gave you is the likelihood of seeing "lottery" from a spam email.

The probability that the word "lottery" appeared in any messages at all, or $P(lottery)$ is known as the **marginal likelihood**.

Applying Bayes' theorem we can estimate a **posterior probability**, the measure of how likely an email is to be spam given the evidence. Posterior probability is the probability an event will happen after all evidence or background information has been taken into account. If the posterior probability is greater than 50%, our model should classify the email as spam and flag it as such:

$P(spam|lottery) = \frac{P(lottery|spam)P(spam)}{P(lottery)}$
$P(spam|lottery) = \frac{0.04*0.2}{0.05}$
$P(spam|lottery) = 0.16$

The first line above is just a representation of the following:
Posterior Probability = (Likelihood * Prior probability) / Marginal Likelihood

And the last line tells us that the probability of an email being spam given the presence of the word "lottery" is 16%.

To get an intuition of how this work, consider the numerator: it's computing the probability of a spam and the word "lottery" happening at the same time. We then discount this probability for the occurence of the word "lottery" in general, including the instances where it appears in a genuine email.

Discussion:

If you were to change the Marginal Likelihood from 0.05 to 0.01, what is the posterior probability?

P(spam|lottery) = (0.04 * 0.2) / (0.01) = 0.8

Characteristics of Naive Bayes

By now, an important question we may have is how Naive Bayes treat datasets with a large number of features with predictive attributes. Particularly, does it operate on the assumption that all features are similarly weighted in their importance to our target feature? The answer is *yes*. In fact, Naive Bayes is named so because of the "naive" assumption it makes:

- That all features of the dataset are equally important and independent
  In real world applications, this assumption is rarely true. It is almost a certainty that certain keywords such as "lottery" are more indicative of a spam message than other keywords (eg. "Free" for example). Additionally, text found in email bodies are not independent from each other: an email that contains the word "Viagra" is more likely to also contain the word "Enhancement" or "Drugs".

Despite the "naive" assumptions it makes above, the Naive Bayes algorithm continues to be a very commonly used and vital tool in classification tasks. In cases where the assumptions of a naive bayes classifer are violated the result are still often times impressive (paper by Domingos and Pazzani[1] or paper by Zhang[2]) despite sub-optimality.

Potential limitations of a naive bayes classifier:
- Skewness due to data scarcity
In some cases our data may have a distribution where scarce observations lead to probabilities approximating close to 0 or 1, which introduces a heavy bias into our model that could lead to poor performance on unseen data. If we only have **1 email** with the word "prince" (as in Nigerian prince) that happen to be a scam, the estimated probability is no longer a fair estimation of the posterior probability for p(scam|prince). In other words, a naive bayes classifier built with this data will end up predicting all emails with the word "prince" as a scam.

- A common treatment of continuous features is "binning"
  By transforming continuous variables to discrete variables through "binning", we may lose considerable variability or important structures in the data.

In Day 2 of the workshop, we'll learn about smoothing our probabilities as well as other techniques to mitigate both of the limitations I mentioned above. I will also highlight a few techniques in improving your model accuracy and performance evaluation. We'll do this by working through some real-life application of the naive bayes classifier, and combine that with text mining techniques to produce a text classifier.

Bayes Theorem Application: Customer Churn

As a recap, recall that we use the Bayes Theorem to calculate conditional probability: $P(A|B) = \frac{P(A \cap B)}{P(B)} = \frac{P(B|A)P(A)}{P(B)}$

For a motivational example, let's consider a simple case of classification: we'd like to predict if a customer would renew their existing subscription with our business given their last email correspondence. We did some initial work in R and discovered that the word "terrible" is a great predictor for whether a customer stayed with us or if he/she churn.

From existing data, we have:
- 400 customers
- 40 of 400 churned; 360 of them did not

[1] Domingos, P. and Pazzani, M., 1997. On the optimality of the simple Bayesian classifier under zero-one loss. Machine learning, 29(2-3), pp.103-130.

[2] Zhang, H., 2005. Exploring conditions for the optimality of naive Bayes.International Journal of Pattern Recognition and Artificial Intelligence, 19(02), pp.183-198.

- 6 of the 40 churned customers use the word “terrible” in their last correspondence
- 2 of the 360 staying customers use the word “terrible” in their last correspondence

P(Terrible | Churn) = 6/40, or 0.15 P(Churn) - 50/400 = 40/400, or 0.1 P(Terrible) = 8/400 = 0.02

From the model we hypothesized :
P(Churn | Terrible) = (P(Terrible | Churn) * P(Churn))/P(Terrible) P(Churn | Terrible) = (0.15*0.1)/0.02 = 0.75

The possibility of a customer churning (or “not renewing”) their services with our company is 75% given the evidence of the word “terrible” in their email correspondence with us.

Let’s take a look at an extended example of the above illustration. Supposed the dataset you’re working with gave you the following:
- 1000 customers
- 250 of 1000 churned; 750 of them did not
- 45 of the 250 churned customers use the word “terrible” in their last correspondence; While only 11 of the 750 stayed customer used it
- 5 of the 250 churned customers use the word “recommended” in their last correspondence; While 155 of the 750 stayed customer used it
- 85 of the 250 churned customers sent the email during the 2359 to 0600 period; While 156 of the 750 stayed customer sent it inside that period

Now let’s use the conditional probability we’ve learned to obtain the probability of a customer, named Sally, churning the evidence that her last email with us contained the word “terrible”, does not include the word “recommended” and sent at 1640 time.

We can then rewrite the question to take on a more compact form:
P(Churn | Terrible and !Recommend and !Midnight) or more formally:

$$P(Churn|T \cap \neg R \cap \neg M) = \frac{P(T \cap \neg R \cap \neg M|Churn)P(Churn)}{P(T \cap \neg R \cap \neg M)}$$

This formula is computationally expensive to solve because of all the possible intersecting events, and in real-world applications we would have an even more complex dataset with hundreds of predictive features. To cope with this, Naive Bayes assumes independence among each events (Arsene Wenger being the manager neither increase nor decrease the likelihood of Arsenal playing more youngsters). This assumption makes it possible to simply the formula using the probability rule for independent events, which we recall is:

$P(A \cap B) = P(A) * P(B)$ – when A and B are independent events.

So assuming independence of the variables, we can compute the probability of a churn given the evidence of that 3 events (or non-events) as:
$\frac{P(T)*P(\neg R)*P(\neg M)*P(Churn)}{P(T \cap \neg R \cap \neg M)}$

$\frac{P(\frac{45}{250}*\frac{250-5}{250}*\frac{250-85}{250})*\frac{250}{1000}}{P(T*\neg R*\neg M)}$

P(T) = (45+11)/1000 = 0.056

P(!R) = (1000-5-155)/1000 = 0.84

P(!M) = (1000-85-156)/1000 = 0.759

$P(T * \neg R * \neg M)$
$= P(T * \neg R * \neg M|churn) * P(churn) + P(T * \neg R * \neg M|stay) * P(stay)$
= 0.03601752

```
(45/250) * (245/250) * (165/250) * (250/1000) +
(11/750) * (595/750) * (594/750) * (750/1000)
```

```
## [1] 0.03601752
```

$\frac{(.18*.98*.66)*.25}{0.03601752}$
$= 0.8081067$

```
(.18 * .98 * .66 *.25)/(0.03601752)
```

```
## [1] 0.8081067
```

So assuming independence of the variables, we can compute the probability of no-churn given the evidence of that 3 events (or non-events) as:
$\frac{P(T)*P(\neg R)*P(\neg M)*P(Stay)}{P(T \cap \neg R \cap \neg M)}$

$\frac{P(\frac{11}{750}*\frac{750-155}{750}*\frac{750-156}{750})*\frac{750}{1000}}{P(T*\neg R*\neg M)}$
$= \frac{.0147*.793*.792*0.75}{0.03601752} = 0.1922486$

```
(.0147 * .793 * .792 * 0.75 )/0.03601752
```

```
## [1] 0.1922486
```

Notice that the odds of Sally churning given the evidence in her last email correspondence with us are going to be 0.808/0.192 is 4.2-to-1. Our naive bayes classifier is going to predict a "churn".

Laplace Estimator

Let's explore another important feature of Naive Bayes. Supposed we add another predictor, and this being whether the word "expensive" is used in the last correspondence: - 4 of the 250 churned customers use the word "expensive" in their last correspondence; While only 0 of the 750 stayed customer used it

It looks like our solution is generally accepted as inexpensive, even among the churned customers (they did not renew their service with us for reasons unrelated to cost). Sally's email, we learn does contain the word "expensive".

The updated probability of a churn given the evidence of that 4 events (or non-events) as:
$\frac{P(T)*P(\neg R)*P(\neg M)*P(E)*P(Churn)}{P(T \cap \neg R \cap \neg M \cap \neg E)}$

The updated probability of no-churn given the evidence of that 4 events (or non-events) as:
$\frac{P(T)*P(\neg R)*P(\neg M)*P(E)*P(Stay)}{P(T \cap \neg R \cap \neg M \cap \neg E)}$

Notice that in the second equation (probability of no-churn), we are going to get a 0 because $P(E|no-churn)$ is essentially 0. Our classifier, trained with the data we have fed into the algorithm, is going to predict a "churn" for every customer whom last correspondence with us include the word "expensive".

More generally, this problem will occur whenever an event we use as predictors never occur for one or more levels of the class (In our example, the event "expensive" is not represented in at least one level: the "stay" level). This led to a 0 percent value that are then multiplied in a chain, resulting in this one factor nullifying all other evidence.

A solution would be the Laplace estimator, which propose to add a small (usually 1) number to each of the counts in the frequency table. This subsequently ensures that each class-feature combination has a non-zero probability of occuring.

Adding a Laplace value of 1, we can recalculate the probability of a customer leaving our service. Because we add 1 to each counts in the frequency table, we now have the following:

Instead of 250 churned customers, our updated count of churned customers would have 254, so we can allocate the 4 to each of the cases below:

- 45+1 of the 254 churned customers use the word "terrible" in their last correspondence; While only 11+1 of the 754 stayed customer used it
- 5+1 of the 254 churned customers use the word "recommended" in their last correspondence; While 155+1 of the 754 stayed customer used it
- 85+1 of the 254 churned customers sent the email during the 2359 to 0600 period; While 156+1 of the 754 stayed customer sent it inside that period
- 4+1 of the 254 churned customers use the word "expensive" in their last correspondence; While only 0+1 of the 754 stayed customer used it

The updated probability of a churn given the evidence of that 4 events (or non-events) as:
$\frac{P(T)*P(\neg R)*P(\neg M)*P(E)*P(Churn)}{P(T\cap\neg R\cap\neg M\cap E)}$
$= \frac{(\frac{45+1}{254}*\frac{254-6}{254}*\frac{254-86}{254}*\frac{4+1}{254})*\frac{250}{1000}}{P(T*\neg R*\neg M)}$

The numerator of the equation would be:

```
(46/254)*((254-6)/254)*((254-86)/254)*(5/254)*(250/1000)
```

```
## [1] 0.0005755644
```

The updated probability of no-churn given the evidence of that 4 events (or non-events) as:
$\frac{P(T)*P(\neg R)*P(\neg M)*P(E)*P(Stay)}{P(T\cap\neg R\cap\neg M\cap E)} = \frac{(\frac{11+1}{754}*\frac{754-156}{754}*\frac{754-157}{754}*\frac{0+1}{754})*\frac{750}{1000}}{P(T*\neg R*\neg M)}$

The numerator of the equation would be:

```
(12/754)*((754-156)/754)*((754-157)/754)*(1/754)*(750/1000)
```

```
## [1] 0.000009941059
```

The Probability of Sally churning given the new evidence is 0.98, and we've also made the naive bayes classifier more robust with this "fail-safe".

```
0.0005755644/(0.0005755644+0.000009941059)
```

```
## [1] 0.9830214
```

```
0.000009941059/(0.0005755644+0.000009941059)
```

```
## [1] 0.01697859
```

Practical Examples of Smoothing (more intuition)

Consider another case of a spam classification algorithm trained with all the emails in our mailing system over the last 3 months. Supposed the word "measurable" have never appeared in any of the emails in the last 3 months and hence the model we've developed was never exposed to this word. According to the model, P(Measurable|Spam) and P(Measurable|NotSpam) are both going to be 0, and the numerator consequently will also be 0. The probability of it being either of the class is going to be 0.

Furthermore, consider another email our classification algorithm is supposed to predict, "CLAREANCE SALE: 99% OFF Discount, Gucci, LV, and Everything Else Must GO!! CALL 230-331-1000-4 GOOD QUALITY FAST DELIVERY "

The email contains predictor features that strongly indicate a spam, yet because of the maybe-deliberate spelling mistake our classifer has to deal with the word "CLAREANCE", which does not exist in our training set. The consequence of that is a numerator that is 0, leading to the posterior probability of 0. The evidence presented by other predictors are nullified.

Well, could we have considered ignoring any features in the test dataset that are not represented in the training set? Say our spam classifier ignores the word "measurable" because it have not appeared in the past 3 months of data - is that a sound strategy? Allow me to give you an illustration of the idea above.

Consider our training data consist of spam and legit emails, and our classifier is built with 4 predictors (labelled A to D respectively). 40% of them were spam emails and 60% legit. Below is the likelihood ratio of the 4 features, classified by Spam and Legit emails respectively:

Spam: A=3/4, B=2/4, C=0/4, D=1/4

Legit: A=0/6, B=4/6, C=1/6, D=0/6

Supposed the test email we'd like to classify has the feature of (A,C,D,E – E being a word not within our training data)

Strategy 1: Ignore any features that are not represented at least once in **either** class
This strategy will see us removing features A, C and D, this leaves us with only B as the predictor.

The test email we'd like to classify has features (A,C,D,E): This strategy clearly fail because we're unable to classify this test email (Remember: A, C, and D were excluded from the classifier).

Strategy 2: Ignore any features that are not represented at least once in **any** classes
This strategy will see us retain all features, because they are represented at least once in any class.

The test email we'd like to classify has features (A,C,D,E): This strategy fail because the probability of this email being Spam and Legit are both 0 due to the inclusion of a 0 in our numerator

Strategy 3: Instead of ignoring / remove any predictors, we'll use Laplace Smoothing instead
The Laplace (Add-One) Smoothing allows us to classify our test email correctly. It also has the added benefit of not having to drop potentially significant "evidence" by excluding variables. It adjust the likelihood of a feature so even unknown words or words that were previously not represented in a class will have a non-zero probability.

As an interesting aside, the popular natural language toolkit, NLTK has a naive bayes classifier and it used to ignore any features that had zero counts in any of the classes. In later versions, it used an expected likelihood estimation[3] for the probability of a feature given a label which see the algorithm adding 0.5 to each count. Another popular machine learning library, scikit-learn allows user to choose from Laplace smoothing (add-one) and Lidstone smoothing (when the additive value is lesser than 1) through an `alpha` parameter[4].

For features unseen in the training data, commonly a strategy is to ignore that feature and use other features[5] - even though from implementation to implementation (and versions of versions) this may vary, so it's always a good idea to check the source code if you're not writing your own algorithms.

Some other notes:
- Despite it's naive assumptions, naive Bayes classifiers are been a pretty consistent tool in many real-world situations, and are popular choice for document classification and spam filtering.

- Training a Naive Bayes classifer is very fast. Classifying with the classifier on large data is usually also very fast compared to many other methods (particularly in the field of text / document classification). The theoretical reasons why that is has been illustrated in the class, but in simple words: the class-conditional probability can be "decoupled" and hence independently estimated as a one-dimensional distribution. This "in turn helps alleviate problems stemming from the curse of dimensionality". Read The Optimality of Naive Bayes for an academic reference[6].

[3]Natural Language Processing with Python
[4]Naive Bayes, Scikit Learn Documentation
[5]Source code for nltk.classify.naivebayes
[6]Zhang, H., The Optimality of Naive Bayes

Building a spam classifier with Naive Bayes

The following example uses dataset from Tiago A. Almeida and Jose Maria Gomez Hidalgo, a collection of about 10,000 legitimate messages collected for research at the National University of Singapore (NUS). The messages largely originate from Singaporeans and mostly from students of the University.

The label of interest has two classes: "spam" for spam and "ham" for non-spam messages.

```
sms <- read.csv("data_input/spam.csv", encoding = "UTF-8")
str(sms)
```

```
## 'data.frame':    5572 obs. of  5 variables:
##  $ v1 : chr  "ham" "ham" "spam" "ham" ...
##  $ v2 : chr  "Go until jurong point, crazy.. Available only in bugis n great world la e buffet... Cir
##  $ X  : chr  "" "" "" "" ...
##  $ X.1: chr  "" "" "" "" ...
##  $ X.2: chr  "" "" "" "" ...
```

Dive Deeper
There are some pre-processing steps required of this dataset. Are you able to prepare the data for this exercise by:

1. Removing the last 3 variables as they have no valuable information

2. Renaming the first 2 variables to "label" and "text", in that order
   Assign the resulting dataframe after the first two steps name of `sms`, overriding the original `sms` dataframe. You can use the base R function or any other methods / techniques you've learned for task (1) and (2) above.

3. What is the proportion of "spam" messages in your `sms` dataframe?
   Your Answer:

4. Use `sms[sample(nrow(sms), 5),"text"]` to randomly sample 5 of the messages from your dataset. Everything you evaluate this line of code, you'll get a different result. How do you ensure reproducibility?
   Your Answer:

5. Is your `label` variable a factor? If it isn't, convert it to a factor of 2 levels

I'm going to use the `dplyr` library for (1) and (2), and I'll randomly inspect some of the sms messages:

```
RNGkind(sample.kind = "Rounding")
```

```
## Warning in RNGkind(sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
```

```
set.seed(100)
sms <- read.csv("data_input/spam.csv", encoding = "UTF-8") %>%
  rename(label=v1, text=v2) %>%
  mutate(label=as.factor(label))

sms[sample(nrow(sms), 10),"text"]
```

```
##  [1] "Yeah I don't see why not"
##  [2] "Dad went out oredi... "
##  [3] "There is no sense in my foot and penis."
##  [4] "Hi the way I was with u 2day, is the normal way&this is the real me. UR unique&I hope I know u
##  [5] "Hello madam how are you ?"
##  [6] "All these nice new shirts and the only thing I can wear them to is nudist themed ;_; you in mu
##  [7] "DO U WANT 2 MEET UP 2MORRO"
##  [8] "I did. One slice and one breadstick. Lol"
##  [9] "Slaaaaave ! Where are you ? Must I summon you to me all the time now ? Don't you wish to come
## [10] "Hey i booked the kb on sat already... what other lessons are we going for ah? Keep your sat nig
```

As an extra exercise, can you print out a few of the "spam" messages to see if there's any "word" that could be strong predictors of spam?

```
### Your Answer Below:
```

Text Mining in R

We also load the `tm` package, which adds text processing functionality commonly associated with text mining (extra materials: read my beginner's introduction to text mining). We will use `tm`'s `VCorpus` function to create our corpus from the sms dataset we just loaded in.

A corpus is the technical term for a structured set of text electronically stored for statistical analysis, natural language processing (nlp) tasks and other computational linguistics work. Once we have a corpus, we can use `$content` to access the content stored within each document.

```
library(tm)
# VCorpus requires a source object, which can be created using VectorSource
sms.corpus <- VCorpus(VectorSource(sms$text))
# Inspecting the first 2 items in our corpus:
sms.corpus[[1]]$content
```

```
## [1] "Go until jurong point, crazy.. Available only in bugis n great world la e buffet... Cine there g
```

```
sms.corpus[[2]]$content
```

```
## [1] "Ok lar... Joking wif u oni..."
```

```
# alternatively
# lapply(sms.corpus[1:2]$content, as.character)
```

First thing in almost any text mining project is to perform the necessary pre-processing steps on our corpus. We can use the various `content_transformer()` functions to clean up our corpus data:

```
# convert all text to lower so YES == yes == Yes
sms.corpus <- tm_map(sms.corpus, content_transformer(tolower))
# Create a custom transformer to substitute punctuations with a space " "
transformer <- content_transformer(function(x, pattern) {
    gsub(pattern, " ", x)
})
```

```
# Remove numbers, stopwords ("am", "and", "or", "if")
sms.corpus <- tm_map(sms.corpus, removeNumbers)
sms.corpus <- tm_map(sms.corpus, removeWords, stopwords("english"))

# Substitute ".", "/", "@" and common punctuations with a white space
sms.corpus<- tm_map(sms.corpus, transformer, "/")
sms.corpus <- tm_map(sms.corpus, transformer, "@")
sms.corpus <- tm_map(sms.corpus, transformer, "-")
sms.corpus <- tm_map(sms.corpus, transformer, "\\.")

# For all other punctuations, simply strip them using the built-in function
sms.corpus.new <- tm_map(sms.corpus, removePunctuation)
```

`tm` has a `stemDocument()` function, which we will use to reduce each word into their root form so "winning" and "win" will be transformed into "win", allowing them to be treated as the same concept instead of different ideas.

```
# see how stemming works
library(SnowballC)
wordStem(c("do", "doing", "kiss", "kisses", "kissing"))
```

```
## [1] "do"   "do"   "kiss" "kiss" "kiss"
```

Applying it to our corpus, and also apply the `stripWhitespace` treatment:

```
sms.corpus.new <- tm_map(sms.corpus.new, stemDocument)
sms.corpus.new <- tm_map(sms.corpus.new, stripWhitespace)
```

Another important aspect of text mining is a process called tokenization. It's the practice of splitting each corpus element into individual phrases. Conveniently, the `tm` package helps us with this tokenization process through its `DocumentTermMatrix`, essentially allowing us to create a DTM / Sparse matrix:

```
sms.dtm <- DocumentTermMatrix(sms.corpus.new)
#Examine our dtm
inspect(sms.dtm)
```

```
## <<DocumentTermMatrix (documents: 5572, terms: 6183)>>
## Non-/sparse entries: 42895/34408781
## Sparsity           : 100%
## Maximal term length: 38
## Weighting          : term frequency (tf)
## Sample             :
##       Terms
## Docs   call can come day free get just ltgt now will
##   1085    0   0    1   2    0   1    0    0   0   12
##   1579    0   0    0   0    0   0    0   18   0    0
##   1863    0   0    0   0    0   0    0    0   0    0
##   2158    0   0    0   0    0   0    0    0   0    0
##   2370    0   0    0   0    1   0    0    0   0    0
##   2380    0   1    0   1    0   0    0    1   0    0
##   2434    0   3    0   0    1   1    0    6   0    0
```

```
##   2848    0   0    0   0    0   0    0    0   0    0
##   3016    0   0    0   6    0   0    0    2   0    0
##   5105    0   0    0   0    1   0    0    0   0    0
```

A close equivalent of the DocumentTermMatrix (DTM) is the TermDocumentMatrix (TDM), and they both refer to a Matrix-like format that displays the frequence of terms that occur in a collection of documents. The former term is used when the rows correspond to documents in the collection and columns correspond to terms, while the latter (TDM) is when the roles are inversed (rows corresponding to terms while columns correspond to documents).

When we use `inspect()` on our dtm, it displays a sample; we would need to use `as.matrix()` to yield the full matrix - however be cautious of large dtm as they can be very memory-hungry!

Taking a quick glance at the sample show us common words found in common sms conversations. By default, these are terms that appear most frequently in our document. Many of them have appeared in at least 300 sms within out dataset:

```
findFreqTerms(sms.dtm, 300)
```

```
## [1] "call" "can"  "come" "get"  "just" "now"  "will"
```

With our dtm ready, we can now split our data into train and test sets.

```
set.seed(100)
split_75 <- sample(nrow(sms.dtm), nrow(sms.dtm)*0.75)
sms_train <- sms.dtm[split_75, ]
sms_test <- sms.dtm[-split_75, ]
```

Now that we have the two train and test sets (in DTM format), we will store the ground truth labels as well - they are necessary for the training of the classifer as well as simply model evaluation later:

```
train_labels <- sms[split_75, 1]
test_labels <- sms[-split_75, 1]
```

> To make predictions on new data, you can create a DTM for the new data that has been adjusted to the terms contained in the train data

Let's verify that the ham and spam messages are roughly evenly distributed between the train and test sets:

```
prop.table(table(train_labels))
```

```
## train_labels
##       ham      spam
## 0.8631251 0.1368749
```

```
prop.table(table(test_labels))
```

```
## test_labels
##       ham      spam
## 0.8743719 0.1256281
```

To reduce noise, we want to train our naive bayes classifier using only words that appear in at least 20 messages (~0.32%) from the data train:

```r
set.seed(100)
sms_freq <- findFreqTerms(sms_train, 20)
# take a look at some of the words that will be used as predictors in our classifier:
sms_freq[sample(length(sms_freq), 10)]
```

```
##  [1] "god"   "face"  "msg"   "away"  "live"  "lot"   "start" "help"  "mobil"
## [10] "cos"
```

We'll then subset our DTM to get all the rows (corresponding to documents) but only include columns (terms) where it has appeared in at least 20 messages:

```r
sms_train <- sms_train[,sms_freq]
sms_test <- sms_test[,sms_freq]
```

Classical implementation of naive bayes in text classification usually see the classifier trained on data with categorical features but the value in our DTM are currently "counts". Before we train our classifer, let's convert the numeric values to a simple categorical variable. Specifically, we'll replace the counts with a binary of 1 (when the term is present in the document) and 0 (when the term is not).

Practice! Can you complete the following function? The function should convert x **to a factor of 1 or 0** depending if its value is greater than or equal to one.

```r
bernoulli_conv <- function(x){
  x <- __________
}
```

Try and evaluate the following code to see if your function worked!

```r
counts <- c(3,0,0,1,4,0)

# Uncomment the following code:
#sapply(counts, bernoulli_conv)
```

Once you have the converter function, we can now apply it to our train and test DTM.

```r
# Takes an input, "x" and assign x to a 1 or 0
bernoulli_conv <- function(x){
        x <- as.factor(as.numeric(x > 0))
}

train_bn <- apply(sms_train, 2, bernoulli_conv)
test_bn <- apply(sms_test, 2, bernoulli_conv)
```

If you inspect the `train_bn` and `test_bn` respectively, you will find that most of the values in our matrices are 0; A matrix with such characteristic is also referred to as a "sparse matrix".

```r
set.seed(101)
train_bn[1:6,sample(ncol(train_bn), 10)]
```

```
##       Terms
##        home around repli poli even get nice guess pain miss
```

```
##    [1,] "0"  "0"    "0"   "0"  "0"  "0" "0"  "0"   "0"  "0"
##    [2,] "0"  "0"    "0"   "0"  "0"  "0" "0"  "0"   "0"  "0"
##    [3,] "0"  "0"    "0"   "0"  "0"  "0" "0"  "0"   "0"  "0"
##    [4,] "0"  "0"    "0"   "0"  "0"  "0" "0"  "0"   "0"  "0"
##    [5,] "0"  "0"    "0"   "0"  "0"  "0" "0"  "0"   "0"  "0"
##    [6,] "0"  "0"    "0"   "0"  "0"  "0" "1"  "0"   "0"  "0"
```

Our matrices should now only contain 1 or 0, but we'll be careful to ensure that our values are of factor type and not numerical as that may lead to our naive bayes using a gaussian function instead. Remember: 1 and 0 are binary-valued and thus have to be represented as binary-valued and factors are the appropriate type for this.

```
head(train_bn[train_bn[,4] == "1" | train_bn[,5] == "1", 1:7])
```

```
##       Terms
##        abt account actual address afternoon aight alreadi
##    [1,] "0" "0"     "0"    "1"     "0"       "0"   "0"
##    [2,] "0" "0"     "0"    "0"     "1"       "0"   "0"
##    [3,] "0" "0"     "0"    "0"     "1"       "0"   "0"
##    [4,] "0" "0"     "0"    "1"     "0"       "0"   "0"
##    [5,] "0" "1"     "0"    "1"     "0"       "0"   "0"
##    [6,] "0" "0"     "0"    "1"     "0"       "0"   "0"
```

Precautionary Measures As a precautionary measure, I want to make sure that all the 353 "terms" in our train and test datasets are represented at least once. Verifying that both the train and test dataset have essentially the same number of columns (although this is not required for Naive Bayes to work).

```
ncol(sms_train)
```

```
## [1] 353
```

```
ncol(sms_test)
```

```
## [1] 353
```

```
sum(dimnames(sms_train)$Terms == dimnames(sms_test)$Terms)
```

```
## [1] 353
```

Recall about the non-zero probability we learned about in the previous section? If one of the terms have appeared 0 times in our dtm, that corresponding class-feature combination has a 0/4179 probability of occuring which necessitates the use of laplace smoothing later in our model construction.

```
num_train <- colSums(apply(train_bn, 2, as.numeric))
num_test <- colSums(apply(test_bn, 2, as.numeric))

num_train[num_train < 3]
```

```
## named numeric(0)
```

```
num_test[num_test < 3]
```

```
##    address   deliveri friendship        haf       hand      price        sad
##          2          2          1          1          1          2          1
##       wife
##          2
```

All the terms are represented at least once in both the train and test sets, a non-smoothed model would have no problem in the classification phase of the model. Why do you think our real-world dataset here did not encounter an issue of non-representation in either of the test and test sets?

Answer:

There are many great implementation of the naive bayes algorithm in R, but I'll start with demonstrating the one from `e1071`. Make sure this package is installed on your system so you're able to load it into your workspace.

The Naive Bayes Classifer is constructed using the `naiveBayes()` function, which computes the conditional a-posterior probabilities of a categorical class variable given independent predictor variables using the Bayes rule we learned earlier. By default, it applies a 0 for `laplace` (no Laplace smoothing). We could specify `laplace=1` to enable an add-one smoothing - even though in this case that is of limited use:

```
library(e1071)
spam_model <- naiveBayes(train_bn, train_labels, laplace = 1)
```

Let's use the generic `predict()` function to predict with the model we've constructed, on the test set data to get a sense of it's performance on unseen data:

```
spam_prediction <- predict(spam_model, test_bn)
```

Let's create a confusion matrix as well as it's estimated accuracy on unseen data:

```
table(prediction = spam_prediction, actual=test_labels)
```

```
##           actual
## prediction  ham spam
##       ham  1205   23
##       spam   13  152
```

```
sum(spam_prediction == test_labels)/length(test_labels)*100
```

```
## [1] 97.41565
```

Dive Deeper: Recall about the ROC Curve we learned in the last lesson. The ROC curve is a graphical representation of the Sensitivity-Specificity pair corresponding to each particular decision threshold. From the confusion matrix above, try and answer the following questions.

1. What is the Sensitivity (Recall) of your Spam Classifier?
   Your Answer:
2. What is the Specificity of your Spam Classifier?
   Your Answer:

3. Between minimizing False Positives and False Negatives, which is the more important one from a customer experience perspective? Argue your case.

Because I am compelled to think that "legit sms being flagged as spam" is a more severe offence than the other way round, I'm going to hypothesized that more users would choose to let a small amount of spam messages slip through than an overly sensitive filter where legitimate messages are blocked.

We'll take a closer look at the **False Positives** from our prediction. There should be 5 of them (as per our confusion matrix):

```
inspect(sms_test[test_labels == "ham" & spam_prediction != test_labels,])
```

```
## <<DocumentTermMatrix (documents: 13, terms: 353)>>
## Non-/sparse entries: 74/4515
## Sparsity           : 98%
## Maximal term length: 10
## Weighting          : term frequency (tf)
## Sample             :
##       Terms
## Docs   call charg com free get later may mobil send sms
##   1250    1     0   0    0   0     1   0     0    0   1
##   1497    0     0   0    0   2     0   1     1    2   0
##   2057    0     1   0    0   1     0   0     1    0   0
##   2379    0     0   2    1   0     0   0     1    1   1
##   2652    1     0   0    0   1     0   0     0    0   0
##   4252    0     0   0    0   0     0   0     0    1   0
##   4418    1     0   0    1   1     0   0     0    0   0
##   4801    2     0   0    0   1     0   2     0    0   0
##   5009    0     0   0    0   0     0   0     1    0   1
##   86      1     0   0    0   0     1   0     0    0   1
```

What did you learn from the above sample DTM? Terms like "free", "send", "get", "com" (as in: visit somefreeprize.com) appearing in a sentence seem to be classified as spam when they are actually legit. We can use the index of these documents to print the original sms from our data:

```
sms[c(1497, 1506, 2379, 4801, 5168),2]
```

```
## [1] "I'm always on yahoo messenger now. Just send the message to me and i.ll get it you may have to s
## [2] "Total video converter free download type this in google search:)"
## [3] "Hi, Mobile no.  &lt;#&gt;  has added you in their contact list on www.fullonsms.com It s a great
## [4] "The guy at the car shop who was flirting with me got my phone number from the paperwork and call
## [5] "Oh did you charge camera"
```

We can see that in fact document #2379 should have been a spam by most conventional definitions of what a "spam" is. On unseen data, it correctly classified 97.77% of our messages - and considering the speed and ease of implementation this is not a bad performance at all!

ROC Curve

To plot our ROC Curve, we can have our naive bayes model return the a-posterior probabilities for each class instead of the "class" with maximal probability.

```
spam_prediction_raw <- predict(spam_model, test_bn, type = "raw")
head(spam_prediction_raw)
```

```
##                      ham          spam
## [1,] 0.9130927593810212 0.08690724062
## [2,] 0.0000000008046685 0.99999999920
## [3,] 0.9996792880101368 0.00032071199
## [4,] 0.9968973009734643 0.00310269903
## [5,] 0.9526071799948397 0.04739282001
## [6,] 0.9999325164376512 0.00006748356
```

Practice! Can you create a data frame with 2 variables, namely `prediction` and `trueclass`, where `prediction` should be the probability of the text document being spam and the `trueclass` is the actual "ground truth" labels?

```
## Write your answer here:
```

Once you have the estimate probability of a "positive" class in one vector and the true class in another vector (whether they belong to a dataframe doesn't matter), you can then create the ROC Curve just as you've done in my previous class.

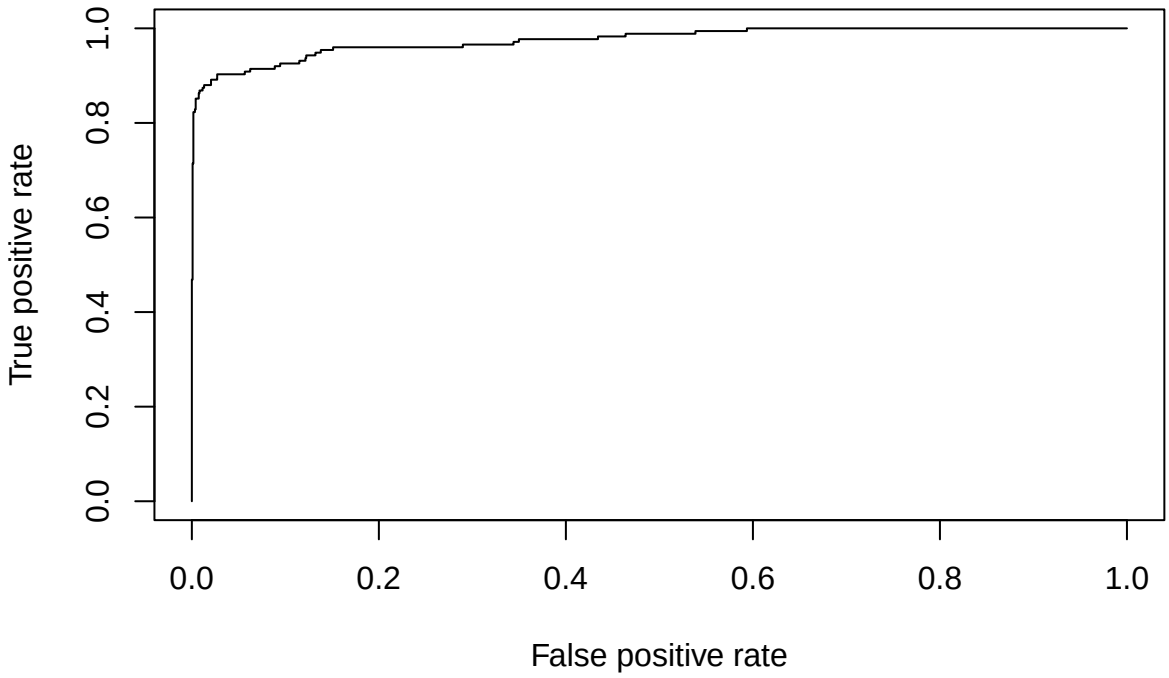
Reference answer for `spam_df`:

```
spam_df <- data.frame("prediction"=spam_prediction_raw[,2], "trueclass"=as.numeric(test_labels=="spam"))
head(spam_df)
```

```
##      prediction trueclass
## 1 0.08690724062         0
## 2 0.99999999920         1
## 3 0.00032071199         0
## 4 0.00310269903         0
## 5 0.04739282001         0
## 6 0.00006748356         0
```

Creating our ROC Curve:

```
library(ROCR)
spam_roc <- ROCR::prediction(spam_df$prediction, spam_df$trueclass)
plot(performance(spam_roc, "tpr", "fpr"))
```



Predicting party affiliation from key votes on social issues

For a second example of a Naive Bayes classifier in action, we'll use the 1984 United States Congressional Voting records, made available[7] by Congressional Quarterly Almanac with credits to Jeff Schlimmer.

A copy of the dataset is in the `data_input` directory: we'll go ahead and read it as well as taking a quick glance at our data:

```
votes <- read.csv("data_input/votes.txt", stringsAsFactors = TRUE)
head(votes)
```

```
##   republican n y n.1 y.1 y.2 y.3 n.2 n.3 n.4 y.4 X. y.5 y.6 y.7 n.5 y.8
## 1 republican n y   n   y   y   y   n   n   n   n  n   y   y   y   n   ?
## 2   democrat ? y   y   ?   y   y   n   n   n   n  y   n   y   y   n   n
## 3   democrat n y   y   n   ?   y   n   n   n   n  y   n   y   n   n   y
## 4   democrat y y   y   n   y   y   n   n   n   n  y   ?   y   y   y   y
## 5   democrat n y   y   n   y   y   n   n   n   n  n   n   y   y   y   y
## 6   democrat n y   n   y   y   y   n   n   n   n  n   n   ?   y   y   y
```

The dataset has 16 predictor variables and a target variable, which is currently named `republican`. Let's rename the dataset and I'll explain the variable as much as I can.

[7] Schlimmer, J., Congressional Voting Records Data Set, UCI Machine Learning Repository

```r
names(votes) <- c("party",
                  "hcapped_infants",
                  "watercost_sharing",
                  "adoption_budget_reso",
                  "physfee_freeze",
                  "elsalvador_aid",
                  "religious_grps",
                  "antisatellite_ban",
                  "nicaraguan_contras",
                  "mxmissile",
                  "immigration",
                  "synfuels_cutback",
                  "education_funding",
                  "superfundright_sue",
                  "crime",
                  "dutyfree_exps",
                  "expadmin_southafr"
                  )
```

It measures the 16 key votes on different subjects (handicapped infants, religion, immigration, military, education, economic etc). For each of the votes, three answers are possible: "y" for "yes", "n" for "nay" and unknown.

I've done a bit of research and explain some of the predictor variables above to the best of my knowledge:

- `party`: Party affiliation of the individual
- `hcapped_infants`: Handicapped Infants Protection Act (prohibits medical professionals from withholding nutrition or medical treatment from a handicapped infant)
- `watercost_sharing`: Water Project Cost Sharing: intended partly to stopping unnecessary projects (if beneficiaries know they must pay part of the cost)
- `adoption_budget_reso`: Requires the concurrent resolution on the budget to be adopted before legislation providing new budget authority
- `physfee_freeze`: Imposing a one-year freeze on Medicare payments for physicians, prevent doctors from charging mostly-elderly-or-disabled beneficiaries more
- `elsalvador_aid`: Military (arms) aid increase for El Salvador
- `religious_grps`: Legislation to guarantee equal access to school facilities by student religious groups
- `antisatellite_ban`: Preventing funds appropriated for catchall "any other act" could be used to test anti-satellite weaponry for a year
- `nicaraguan_contras`: US aid to the contrast in Nicaraguan countries
- `mxmissile`: Approval of the LGM-118 Peacekeeper, aka MX Missile Program
- `immigration`: Immigration Reform and Control Act
- `synfuels_cutback`: Funding cutback to The Synthetic Fuels Corporation (SFC)

- `education_funding`: As part of the Budget Reconciliation Act, by revising the education budget
- `superfundright_sue`: An amendment aimed at deleting a provision giving citizens the right to sue the EPA in certain cases to force action on dumps
- `crime`: The Comprehensive Crime Control Act of 1984
- `dutyfree_exps`: Granting duty-free treatment to particular items and articles (eg. water chestnuts and bamboo shoots)
- `expadmin_southafr`: Export Administration Act Amendments, which introduces some form of export (or loans) controls to South Africa

Possible values:
- "y" in our attribute value indicates "yea", which includes "voted for", "paired for", and "announced for"
- "n" in our attribute value indicates "nay", which includes "voted against", "paired against" and "announced against"
- It is important to also recognize the "?" in this dataset does not mean that the value of the attribute is unknown. It simply means that the value is not "yea" or "nay"

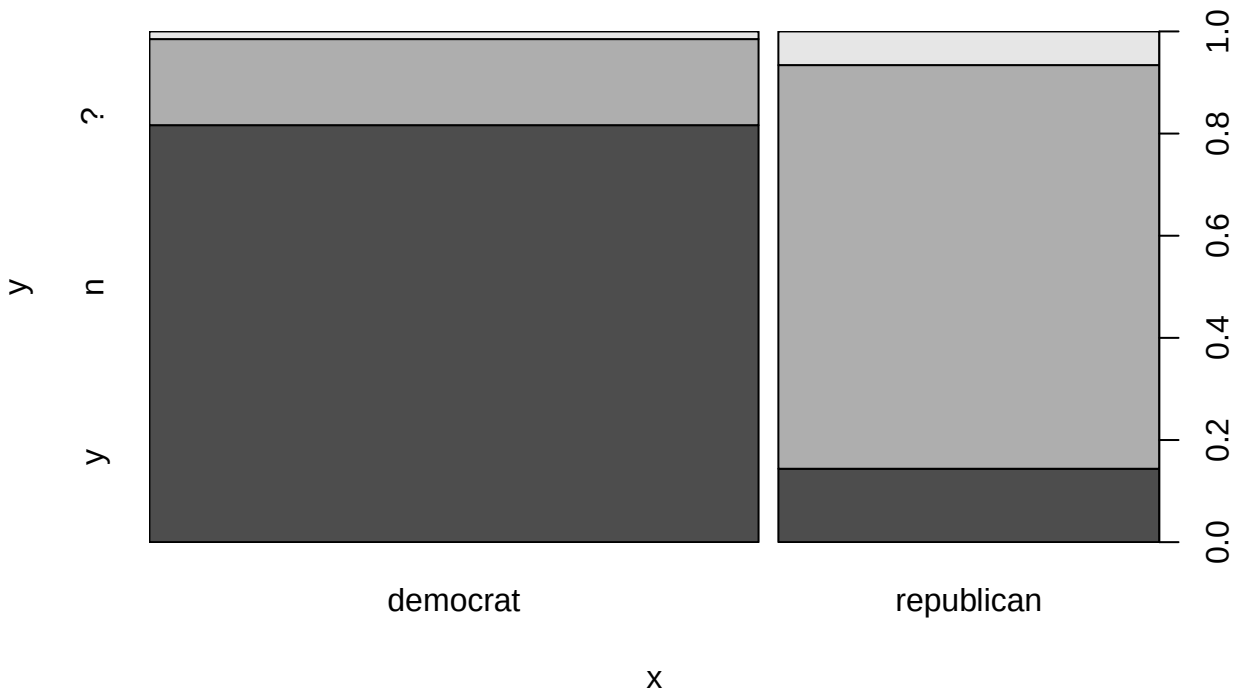
We can quickly get a sense of how some of these features could be good predictors in our naive bayes model:

```
table(votes$party, votes$hcapped_infants)
```

```
##
##                ?   n   y
##   democrat     9 102 156
##   republican   3 133  31
```

We can also use `plot` to visualize the relationship between party affiliations and their position on a particular subject / topic:

```
plot(votes$party, votes$nicaraguan_contras)
```



When you're ready, let's again partition our data into the `train` and `test` sets:

```
set.seed(100)
split_votes <- sample(nrow(votes), nrow(votes)*0.75)
votes.train <- votes[split_votes, ]
votes.test <- votes[-split_votes, ]
```

We'll create our naive bayes classifier and name it `votes_model`, which we'll immediately use for our prediction:

```
library(e1071)
votes_model <- naiveBayes(party ~ ., votes.train)
votes_prediction <- predict(votes_model, votes.test)
```

Printing out the confusion matrix of our `votes_model` and obtaining an estimate of our model's accuracy on unseen data:

```
table("prediction"=votes_prediction, "actual"=votes.test$party)
```

```
##             actual
## prediction   democrat republican
##   democrat         60          3
##   republican       13         33
```

```r
sum(votes_prediction == votes.test$party)/length(votes.test$party)
```

```
## [1] 0.853211
```

If you print the model (`print(votes_model)`) itself, you will find the a-priori probabilities as well as the conditional probabilities of each class ("?", "n" and "y") given a party affiliation.

The a-priori probability is just the probability of being in each class, which in simpler words, refer to our assessment of the probability before (hence "prior-i") we have considered the "information": what policies they voted in favor / or against of.

```r
prop.table(table(votes.train$party))
```

```
##
##   democrat republican
##  0.5969231  0.4030769
```

Let's consider the conditional probabilities. I'm going to

```r
# Loop over all attribute variables in the dataset (v1, v2 ... vn) and create cross-classifying table a
# 1st loop: table(hcapped_infants, class)
# 2nd loop: table(watercost_sharing, class)
# ... 17th loop: table(expadmin_southafr, class)

freqtable.votes <- lapply(votes.train[,c(2:17)], table, votes.train[,1])

# freqtable.votes is now a List of 9 tables.
# each table has variable v as row, and one column for each of the class variable (possible outcome)

freqtable.bc <- lapply(freqtable.votes, t)
freqtable.bc$hcapped_infants
```

```
##
##                ?   n   y
##   democrat     5  72 117
##   republican   2 106  23
```

For the table to be consistent with the naiveBayes object's format, we transpose it and then calculate the likelihood of democrats to vote in favor of the Handicapped Infants Protection Act:

```r
paste("A-Priori Probabiltiies:")
```

```
## [1] "A-Priori Probabiltiies:"
```

```r
prop.table(table(votes.train$party))
```

```
##
##   democrat republican
##  0.5969231  0.4030769
```

```
paste("Democrats likelihood to vote in favor of the Handicapped Infants Protection Act:", round(5/(5+72
```

```
## [1] "Democrats likelihood to vote in favor of the Handicapped Infants Protection Act: 0.0257732 0.37
```

From the above exercises, we obtain by hand the a-priori probabilities and the conditional probabilities that a person would vote "Y" on `hcapped_infants` given the evidence of him / her being Democrat. We can verify our manual calculation by comparing it to the Naive Bayes classifier from the `e1071` package (observe the A-priori probabilities section):

```
votes_model
```

```
##
## Naive Bayes Classifier for Discrete Predictors
##
## Call:
## naiveBayes.default(x = X, y = Y, laplace = laplace)
##
## A-priori probabilities:
## Y
##   democrat republican
##  0.5969231  0.4030769
##
## Conditional probabilities:
##             hcapped_infants
## Y                    ?          n          y
##   democrat   0.02577320 0.37113402 0.60309278
##   republican 0.01526718 0.80916031 0.17557252
##
##             watercost_sharing
## Y                    ?          n          y
##   democrat   0.08247423 0.44845361 0.46907216
##   republican 0.12977099 0.41984733 0.45038168
##
##             adoption_budget_reso
## Y                    ?          n          y
##   democrat   0.01546392 0.08762887 0.89690722
##   republican 0.02290076 0.83969466 0.13740458
##
##             physfee_freeze
## Y                    ?           n           y
##   democrat   0.020618557 0.943298969 0.036082474
##   republican 0.015267176 0.007633588 0.977099237
##
##             elsalvador_aid
## Y                    ?          n          y
##   democrat   0.03608247 0.76804124 0.19587629
##   republican 0.02290076 0.03816794 0.93893130
##
##             religious_grps
## Y                    ?          n          y
##   democrat   0.04123711 0.51030928 0.44845361
##   republican 0.01526718 0.10687023 0.87786260
```

```
##
##             antisatellite_ban
## Y                           ?          n          y
##   democrat   0.02577320 0.22164948 0.75257732
##   republican 0.03816794 0.73282443 0.22900763
##
##             nicaraguan_contras
## Y                           ?          n          y
##   democrat   0.02061856 0.14948454 0.82989691
##   republican 0.06106870 0.80916031 0.12977099
##
##             mxmissile
## Y                           ?          n          y
##   democrat   0.07216495 0.21134021 0.71649485
##   republican 0.01526718 0.87022901 0.11450382
##
##             immigration
## Y                           ?          n          y
##   democrat   0.02061856 0.50000000 0.47938144
##   republican 0.01526718 0.38931298 0.59541985
##
##             synfuels_cutback
## Y                           ?          n          y
##   democrat   0.03092784 0.46391753 0.50515464
##   republican 0.06106870 0.80916031 0.12977099
##
##             education_funding
## Y                           ?          n          y
##   democrat   0.07216495 0.80412371 0.12371134
##   republican 0.08396947 0.12977099 0.78625954
##
##             superfundright_sue
## Y                           ?          n          y
##   democrat   0.05670103 0.70618557 0.23711340
##   republican 0.06870229 0.14503817 0.78625954
##
##             crime
## Y                           ?          n          y
##   democrat   0.04123711 0.64432990 0.31443299
##   republican 0.05343511 0.01526718 0.93129771
##
##             dutyfree_exps
## Y                           ?          n          y
##   democrat   0.06185567 0.34536082 0.59278351
##   republican 0.06106870 0.85496183 0.08396947
##
##             expadmin_southafr
## Y                           ?          n          y
##   democrat   0.31443299 0.03092784 0.65463918
##   republican 0.14503817 0.29007634 0.56488550
```

Decision Tree

Decision trees and tree-based models are powerful, incredibly versatile and represent one of the most popular choice for machine learning tasks. Their output are also a powerful form of representing rules, which has the benefit of being interpretable and adaptable to almost any scenario.

Decision trees, random forests and other tree-based models are so widely used it's behind many of the technology we use today. It's behind the face recognition feature in iOS devices, and is what Microsoft and Xbox engineers chose to implement when it needs its Kinect to "quickly and accurately predict 3D positions of body joints from a single depth image using no temporal information."[8]

The whitepaper authors and researchers (linked above) observed that by using decision trees, they were able to perform their tasks in order of magnitude faster; As they compare it to a nearest neighbor + chamfer matching approach they still achieve more accurate results while being 100x faster.

For an illustration, imagine we want to build a simple model that classify different specifies of iris flower based on the flower's sepal and petal measurements:

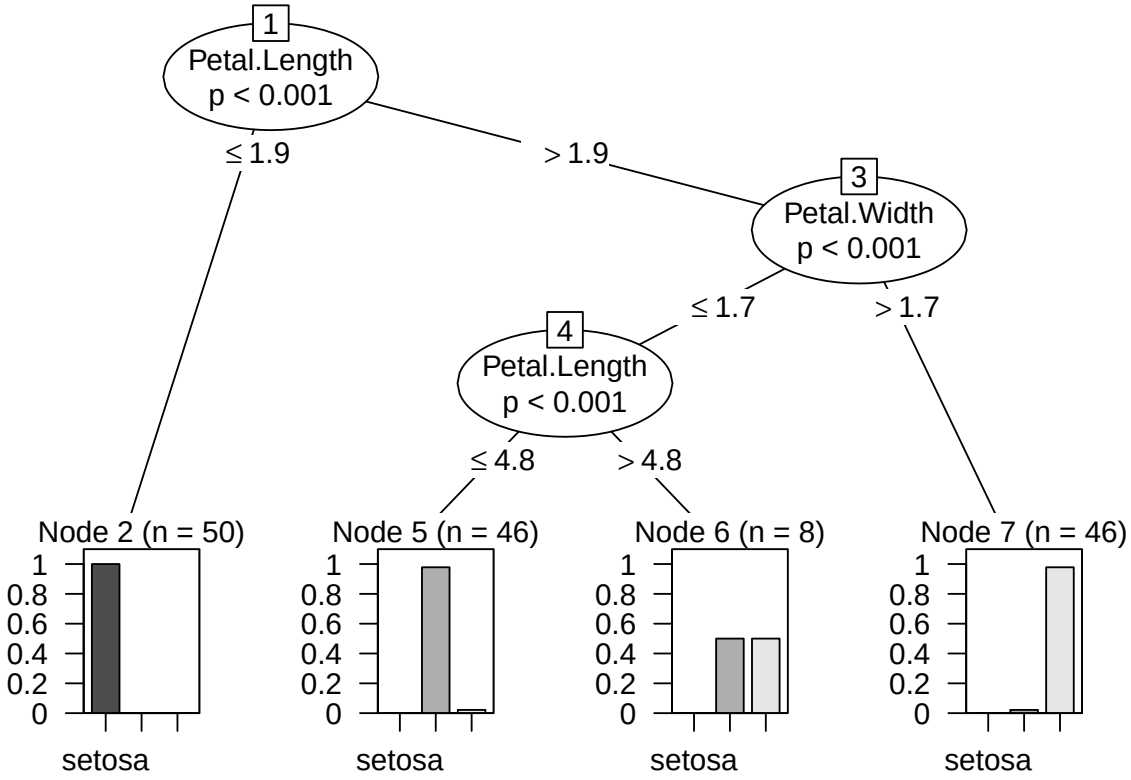
```
head(iris)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa
```

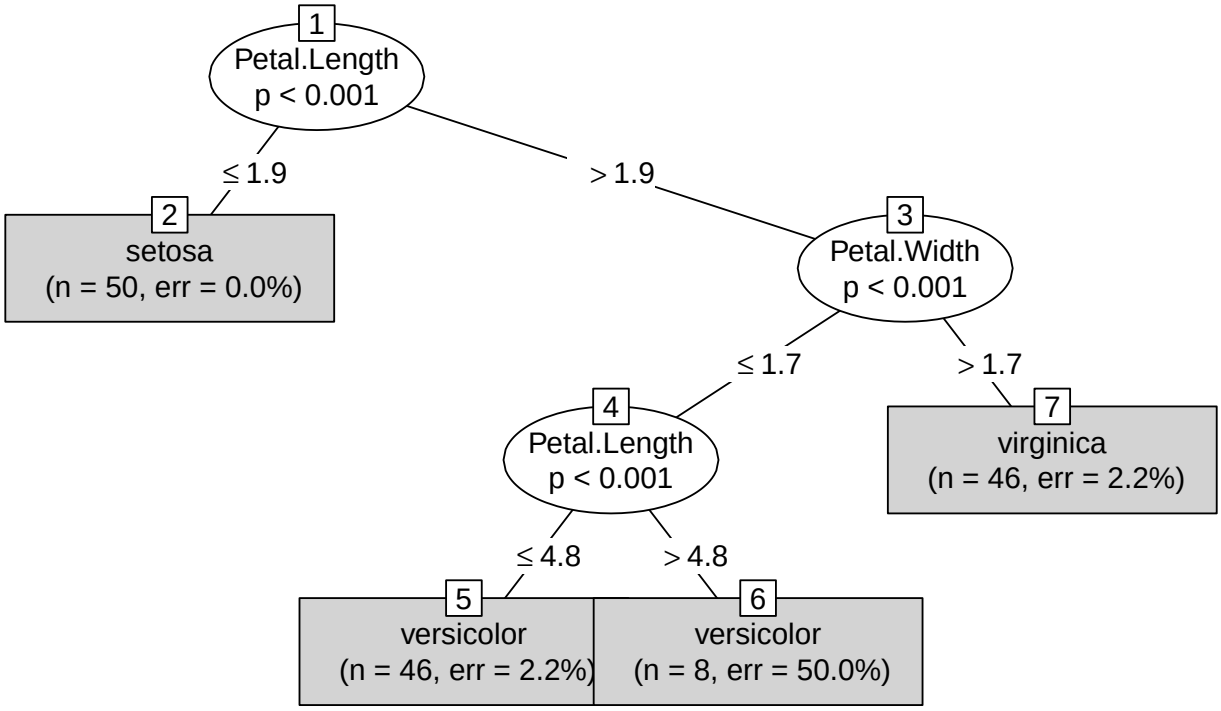
We can use the `ctree` package to create a decision tree model, we'll name it `species` and proceed to plot the model:

```
library(partykit)
species <- ctree(Species ~ ., iris)
plot(species)
```

[8] Real-Time Human Pose Recognition in Parts from Single Depth Images




```
plot(species, type="simple")
```



Notice how our decision tree is useful because it produces rules that are are easy to interpret even for non-technical professionals. A reasonable decision tree model can be useful for field workers tasked to categorize type of flowers because they need only to remember one or two simple rules.

How does a decision tree chooses which feature among the datasets to split on? In the earlier example, how does it pick Petal Length over Petal Width as its first splitting criterion?

Simplistically, we can imagine that the tree will choose to split the data in such a way that the resulting nodes will contain datapoints that contains as high proportion of a single class as possible. The degree to which this is possible is measured as "purity" and various decision tree implementations use a similar standard to achieve splits that ultimately increase homogeneity within the groups.

[Optional] Information Gain and Splitting

One measurement of purity is entropy, which measures homogeneity within the group. In a binary classification problem, the entropy for a completely pure set is 0 and for a set with equal occurrences for both classes is 1. A decision node (split) that lead to high entropy means we get set of values that are quite diverse and random and hence not ideal for classification. The mathematical representation of entropy is:

Entropy(S) $= \sum_{i=1}^{c} -p_i log_2(p_i)$

- Where S is a given segment of data
- c is the number of class levels
- p_i is the proportion of values in the class i

Suppose we have a partition of data with imported (70%) and local (30%), the entropy is: p being 0.7 / 0.3

```
-0.7*log2(0.7)-0.3*log2(0.3)
```

```
## [1] 0.8812909
```

p being 0.5 / 0.5

```
-0.5*log2(0.5)-0.5*log2(0.5)
```

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

p being 0.99 / 0.01

```
-0.99*log2(0.99)-0.01*log2(0.01)
```

```
## [1] 0.08079314
```

In a binary classification case, if we have the proportion of examples in one class (p), we will know the proportion in the other class (1-p). From this we can plot the entropy for all possible values of p:

```
curve(-x*log2(x) - (1-x)*log2(1-x), xlab="proportion of one class", ylab="Entropy", lwd=3)
```

So when our decision tree calculates the change in homogeneity resulting from a given split on each possible feature (known as **information gain**), it is calculating the difference between the entropy in the segment before(S1) and after the split(S2):
Information Gain (F) = Entropy(S1) - Entropy(S2)

Post-split, our data is divided into more than one partition so the calculation or S2 has to consider the weighted sum of entropy across all of the partitions. In other words, the total entropy for S2 is the sum of the entropy of each of the partitions weighted by their proportion of examples in that partition.

When dealing with numeric features, most decision tree classifications will test various splits by setting a numeric threshold and hence converting our numbers to a two-level categorical feature and allowing our information gain formula to work as usual.

Pruning and tree-size

After solving the split problem, the decision tree has one other problem to solve: it needs to know when to stop growing. If the tree grow indefinitely (and it can) it will end up splitting all data points until they are perfectly classified and yield an overly specific model (overfitting; or the case of high variance and low bias). One approach is to set a pre-determined number of levels upon which we command our tree to stop growing. This is an approach known as **pre-pruning**, but it has the obvious downside of us having to make an informed guess around the optimal depth / size of the tree. The alternative, **post-pruning** relies on a strategy that grow the tree to too large a size and then pruning it later on once all important structures and classification patterns were discovered. Getting the best fit decision tree model means we manage to strike a good balance with our bias-variance and precision-recall tradeoffs.

Decision Tree Example: Predicting Diabetes from Diagonistic Measurement

The following dataset is originally donated to the UCI Machine Learning Repository and organized by Friedrich Leisch. It contains 376 observations of variables such as `age`, `pregant`, `glucose` etc and a binary target variable (`pos` or `neg`):

```
diabetes <- read.csv("data_input/diabetes-clean.csv", stringsAsFactors = TRUE)
head(diabetes)
```

```
##   pregnant glucose pressure triceps insulin mass pedigree age diabetes
## 1        1      89       66      23      94 28.1    0.167  21      neg
## 2        0     137       40      35     168 43.1    2.288  33      pos
## 3        3      78       50      32      88 31.0    0.248  26      pos
## 4        2     197       70      45     543 30.5    0.158  53      pos
## 5        1     189       60      23     846 30.1    0.398  59      pos
## 6        5     166       72      19     175 25.8    0.587  51      pos
```

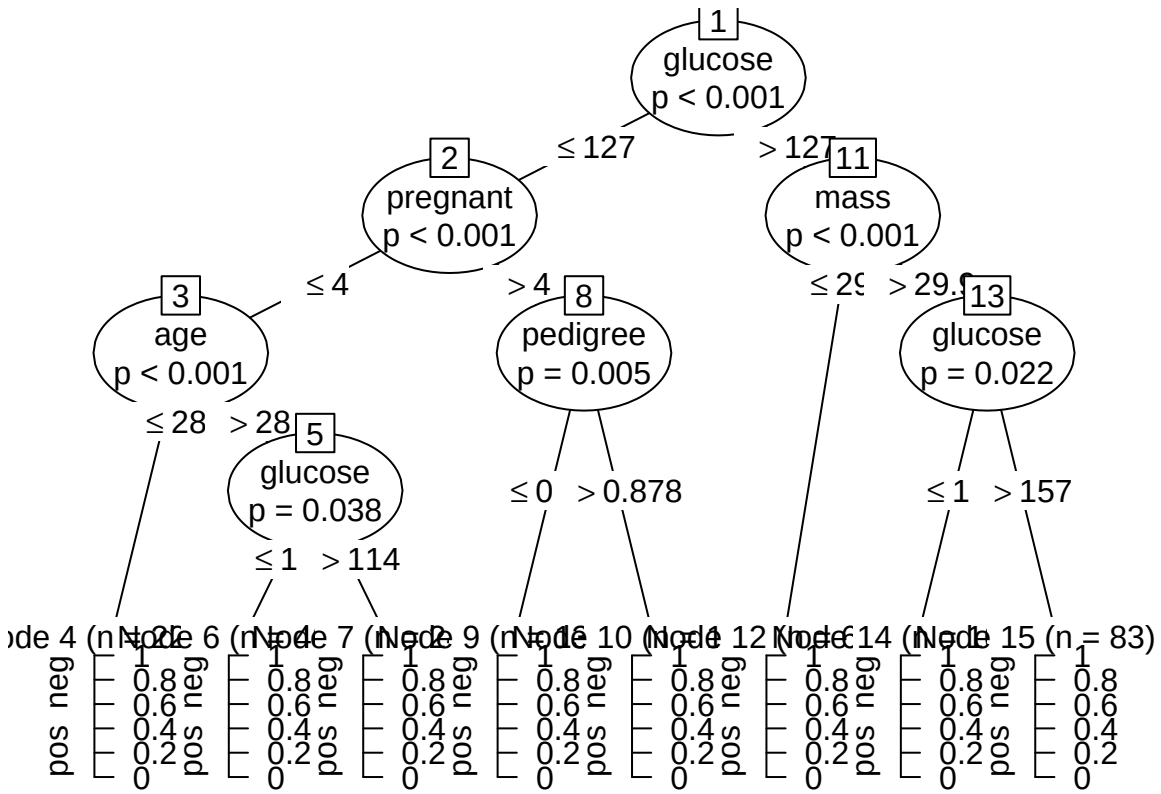
- `pregnant`: Number of times pregnant
- `glucose`: Plasma glucose concentration (glucose tolerance test)
  -`pressure`: Diastolic blood pressure (mm Hg)
  -`triceps`: Triceps skin fold thickness (mm)
- `insulin`: 2-Hour serum insulin (mu U/ml)
- `mass`: Body mass index (weight in kg/(height in m)^2)
- `pedigree`: Diabetes pedigree function
- `age`: Age (years)
- `diabetes`: Test for Diabetes

We'll create our train and test set by random sampling:

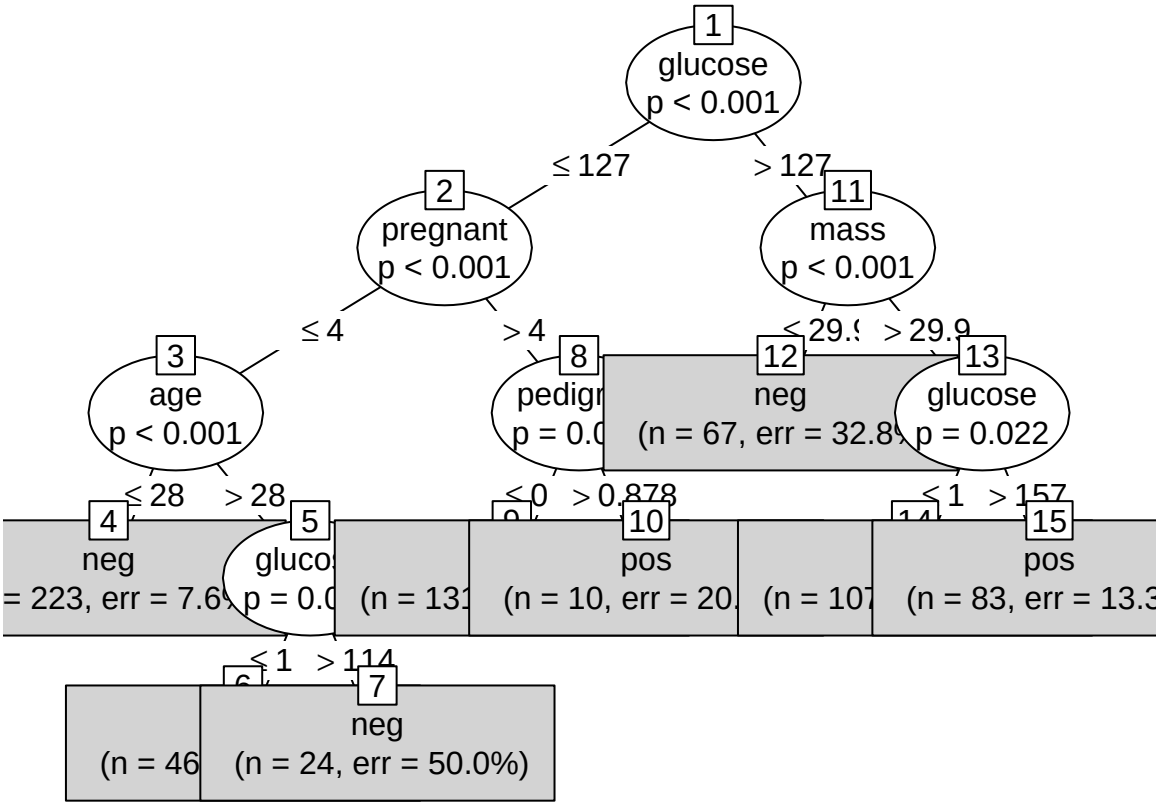
```
set.seed(100)
in_diabetes_train <- sample(nrow(diabetes), nrow(diabetes)*0.9)
diab_train <- diabetes[in_diabetes_train, ]
diab_test <- diabetes[-in_diabetes_train, ]
```

Creating our decision tree

```
diab_model <- ctree(diabetes ~ ., diab_train)
plot(diab_model)
```




```
plot(diab_model, type="simple")
```



Discussion Using the model we've just built (`diab_model`), can you estimate the model's performance (using `predict()`) on our hold-out test set?

What is the accuracy of your model?

Let's look at the underlying structure of our decision tree. We expect to see: - The rules that made up our decision tree - The number of terminal nodes () - The number of inner nodes

We can also use the `width()` and `depth()` functions

```
diab_model
```

```
##
## Model formula:
## diabetes ~ pregnant + glucose + pressure + triceps + insulin +
##     mass + pedigree + age
##
## Fitted party:
## [1] root
## |   [2] glucose <= 127
## |   |   [3] pregnant <= 4
## |   |   |   [4] age <= 28: neg (n = 223, err = 7.6%)
## |   |   |   [5] age > 28
## |   |   |   |   [6] glucose <= 114: neg (n = 46, err = 17.4%)
## |   |   |   |   [7] glucose > 114: neg (n = 24, err = 50.0%)
## |   |   [8] pregnant > 4
## |   |   |   [9] pedigree <= 0.878: neg (n = 131, err = 29.8%)
```

```
## |   |   |   [10] pedigree > 0.878: pos (n = 10, err = 20.0%)
## |   [11] glucose > 127
## |   |   [12] mass <= 29.9: neg (n = 67, err = 32.8%)
## |   |   [13] mass > 29.9
## |   |   |   [14] glucose <= 157: pos (n = 107, err = 37.4%)
## |   |   |   [15] glucose > 157: pos (n = 83, err = 13.3%)
##
## Number of inner nodes:    7
## Number of terminal nodes: 8
```

```
width(diab_model)
```

```
## [1] 8
```

```
depth(diab_model)
```

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

If you've wanted to plot an AUC curve for your model, you can use `type="prob"` in your predict function and the decision tree would return a probability:

```
predict(diab_model, head(diab_test[,-9]), type="prob")
```

```
##          neg        pos
## 3  0.6716418 0.32835821
## 4  0.9237668 0.07623318
## 10 0.7022901 0.29770992
## 18 0.7022901 0.29770992
## 30 0.7022901 0.29770992
## 33 0.9237668 0.07623318
```

Considerations on Decision Tree

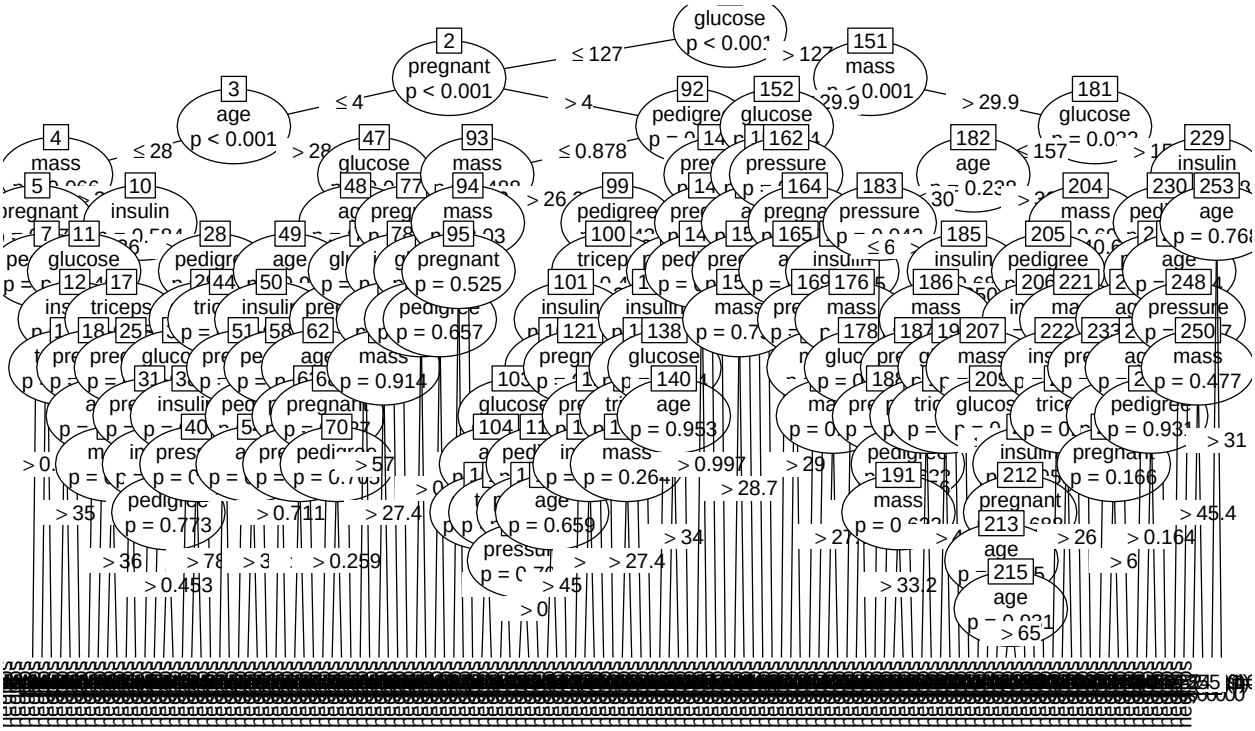
The decision tree models we've created offered a great deal of advantage in terms of its ease of interpretation. However, recall from your stepwise regressions class the concept of "greedy algorithms" - a decision tree is also a greedy algorithm in the way it picks an attribute at every step of the model construction. Because the tree is built in a top-down fashion, it picks an attribute that result in the highest information gain, a strategy that while converge to local optima does not guarantee global optimality. Some effort to reducing the greedy effect of local-optimality has been introduced (k-steps look-ahead trees, by considering more than a single attribute at some of the construction stages; dual information distance (DID) as splitting criterion etc [9])

Decision Trees can be non-robust as their model is very sensitive to the training data it is trained on: changes in training data can result in rather drastic changes to the tree. Decision trees are also famous for being prone to overfitting, even though most decision tree implementation you'll use in R have sensible defaults that negate such as effect (avoid growing an overly complex tree). To make this more concrete, I'm going to implement the same decision tree algorithm but overwriting its default:
- `mincriterion`: Act as a "regulator" for the depth of the tree, smaller values result in larger trees; When mincriterion is 0.8, p-value must be smaller than 0.2 in order for a node to split
- `minsplit` and `minbucket`: Set to 0 so the minimum criterion is **always** met and thus splitting never stop

The resulting model and tree:

[9] Efficient Construction of Decision Trees by the Dual Information Distance Method

```
diab_model <- ctree(diabetes ~ ., diab_train, control = ctree_control(mincriterion=0.005, minsplit=0, m
plot(diab_model)
```



And we can obtain a ~99% in-sample accuracy, which is meaningless and potentially misleading:

```
sum(predict(diab_model, diab_train[,-9])==diab_train$diabetes)/nrow(diab_train)
```

```
## [1] 0.9898698
```

One other weakness in a decision tree is that all predictor terms are by nature assumed to interact. Intuitively, to predict an outcome of cancer risk: the "age" variable have to combine with the "exposure to radiation" one node above it, which in turn have to combine with the "genetic" variable even further up the tree. This rule-based system can be too rigid for some cases of machine learning tasks.

On its own however, decision trees is quite robust to the the problem of multicollinearity. Imagine variable A (body fat percentage) and variable B (BIA, bioelectrical impedance analysis which measures your body's resistance to light electrical current, which is also a way to measure body fat) both explain the effect of body fat on a person's risk to a health condition. A decision tree, being a greedy algorithm, will choose the one that has the highest information gain in one split, whereas a method such as logistic regression would have used both.

Finally, another important trait of decision tree is that it is robust and insensitive to outliers. Intuitively, imagine a measurement of height with values: 167, 170, 161, 176, 158, and 213 in cm. A decision tree's splitting criterion is going to be at a point that greedily maximizes the homogeneity within the resulting groups (the purer the better) and it is not hard to reason that, mathematically, a split at say height=168 would lead to that objective much more consistently that a split at say 200cm (thus separating only that one observation in its daughter nodes). Similarly, if there is a data entry error and you have one observation with height of 400cm, the tree's "purity objective" is still going to offset the outlier's influence. Contrast this approach with the linear regression models we've learned in previous workshop, and I'm sure you can see why it is more robust to outliers in its design.

Keeping in mind those characteristics, as we move past the Optional sub-chapter and into the final parts of this workshop.

[Optional] Relations to Other Machine Learning Algorithms

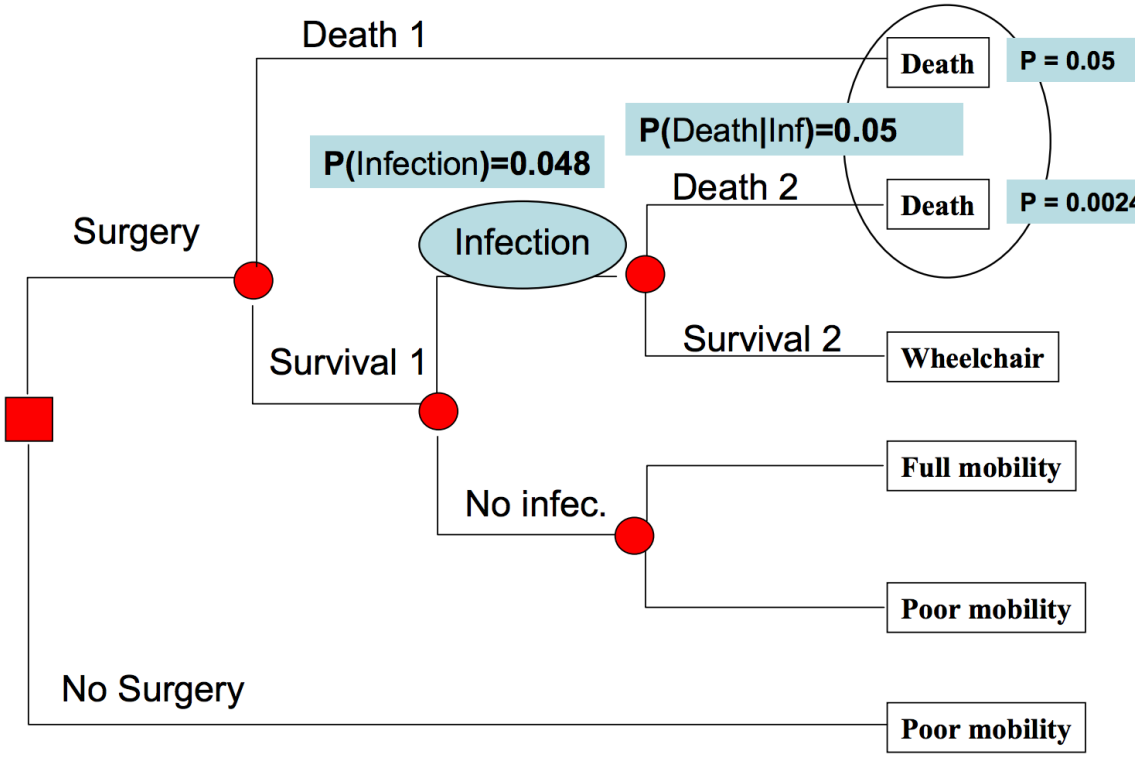
In this intermediate class, we've covered the logistic regression and learn how to use the obtained model to interpret relative risk or success ratio, and relate these odd ratios when comparing groups with different sets of characteristics.

Conceptually, we can also observe the link between a logistic regression function and the naive Bayes (in a binary case) function by rewriting the decision function for naive Bayes as "predict class C_1 if the odds of $p(C_1|x)$ exceed those of $p(C_2|x)$". Expressing this in log-space gives us:

$log\frac{p(C_1|x)}{p(C_2|x)} = \log\ p(C_1|x)$ - $\log\ p(C_2|x) > 0$[10]

The left-hand side of this equation is the log-odds, or the *logit* that we've learned about in the first day and that is the quantity predicted by the model that underlies logistic regression. If you have followed through on both the logistic regression and naive bayes classes, you'll observe that from the summary of the model that a decision tree in fact observes the Bayes rule when estimating an outcome:

P(Infection|Death) = P(Death|Infection)* P(Infection)/P(Death) =
= 0.05*0.048/0.0524 = 0.0024/0.0524 = 0.045



In machine learning, Bayes rule and bayesian methods are used frequently, from building language models (ngrams models), to speech recognition models, to market-basket analysis (discovery of association rules) and are even present in the probabilities output of tree-based models.

[10]Domingos, P. and Pazzani, M., 1997. On the optimality of the simple Bayesian classifier under zero-one loss. Machine learning, 29(2-3), pp.103-130.

Cross-Validation K-fold

One interesting and highly useful technique to evaluate predictive models is called the Cross-Validation K-fold. What it does is partition our original dataset into k equal-sized samples (say we call them 'bins'). Of these smaller bins, a single one of them is used as the test set and the remaining *k-1* bins used as training data. This process is repeated for k times (the folds) so each bin is used exactly once as the test set. The obvious benefit of this technique is that all observations are used as both training and test sets.

```
# library(animation)
ani.options(interval = 1, nmax = 15)
cv.ani(main = "Demonstration of the k-fold Cross Validation",
  bty = "l")
```

Random Forest

Random forest is an ensemble-based state-of-the-art algorithm built on the decision tree method we learned about above and is also known for its versatility and performance. Among the family of ensemble-based classifier include a technique called boosting and it works by combining the performance of weak learners to gain an overall boosted performance.

The idea of ensembling is largely in principle and doesn't necessarily reference any particular algorithm. They describe any meta-algorithms that combine several machine learning techniques into one predictive model in order to decrease variance, reduce bias, or improve predictions.

When we apply the ensemble-based approach on a decision tree model, the trees we built are usually trained using resampled data. In the prediction phase, these trees then vote for a final prediction. Another way to apply ensemble methods on our tree model is known as bagging (bootstrap aggregation). Bagging proposes the idea of creating many subsets of training sample through random sampling (with replacement). Then each of these sets of training sample are used to train one unit of decision tree. This leads us to an "ensemble" of trees, and we'll use the average of all the predictions from these different trees in the prediction phase.

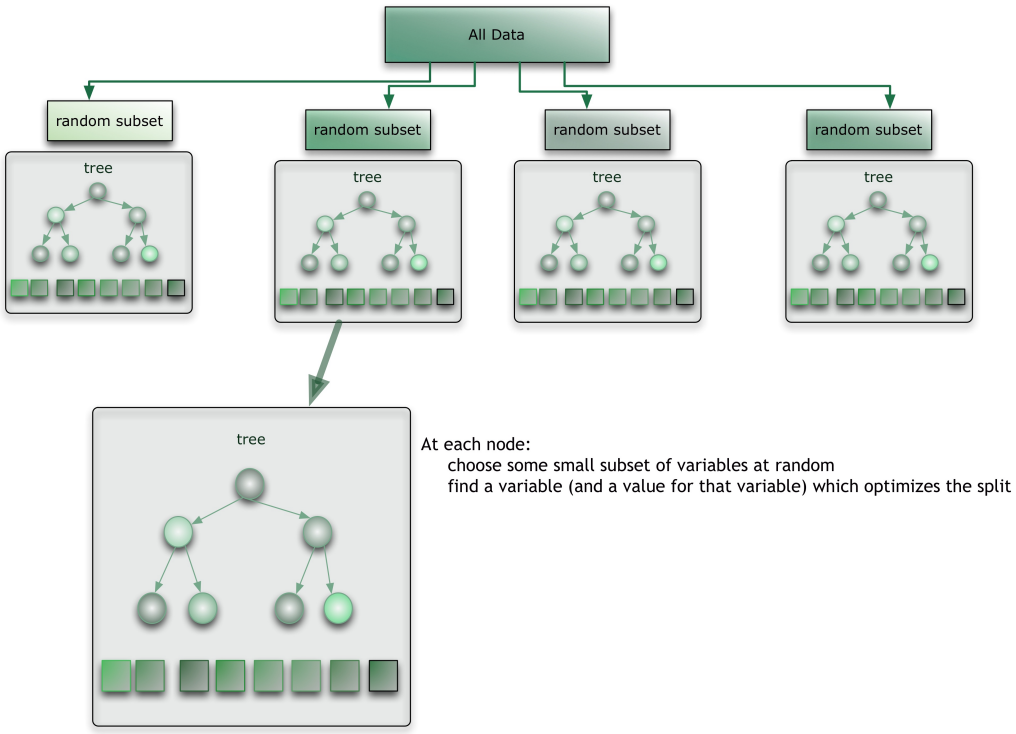
Random Forest extends the idea of bagging by taking one more measure: in addition to creating subsets from the training set, each of the tree is also trained using a random selection of features (rather than using all features). Because each tree is built with a random set of predictors and training samples, the collective of it is called a Random Forest, which is a lot more robust as a model compared to a single tree.

Among many of its advantages, random forest can be used to solve for both regression and classification tasks, handles extremely large datasets well (since the ensemble approach means it only use a small sampled subset from the full dataset), would solve for the dimensionality problems through implicit feature selection while treating noisy data (missing values and outlier values) out of the box.

Ensemble-based Methods

Being an ensemble method, it also inherit the principles of ensemble-based methods: - Combines multiple learners to arrive at a better one
- Voting mechanism
- Weighted combinations

What that means is that instead of growing a single decision tree we grow multiple trees, with each tree voting for a possible class for each new example we reveal to the forest. In the case of classification, our forest then classify the new example through that voting mechanism while in the case of regression it simply takes the average of output.



Credit: Dr. Arshavir Blackwell, CitizenNet

Case Example: Predicting the Quality of Exercise

With the advent of IoT (Internet of Things), wearable technology and fitness trackers have seen a rapid adoption. This in turn contributes to massive advances in the technology, one that pushes beyond simple analytics and drawing expertise from predictive analytics and machine learning.

In the following example, we'll put ourselves in the shoes of a data scientist employed in one of these firms, and we'll learn how we can apply what we've learned in this workshop to build a prediction model that automatically "remind" a user if the user is performing an exercise in a sub-optimal form.

The data for this project (credit to Velloso et. al [11]) is of 6 young participants who were asked to perform one set of dumbbell biceps curl in five different fashion:
Class A: Exactly according to specification
Class B: Throwing elbows to the front
Class C: Lifting the dumbbell only halfway
Class D: Lowering the dumbbell only halfway
Class E: Throwing the hips to the front

Class B and E are common mistakes in the exercise while Class A correspond to an execution of the exercise according to specification. The exercises were performed by six male participants aged between 20 - 28.

```
fb <- read.csv("data_input/fitbit.csv")
fb[,c("user_name", "new_window", "classe")] <- lapply(fb[,c("user_name", "new_window", "classe")], as.fa

summary(fb$classe)
```

[11] Velloso, E.; Bulling, A.; Gellersen, H.; Ugulino, W.; Fuks, H. "Qualitative Activity Recognition of Weight Lifting Exercises. Proceedings of 4th International Conference in Cooperation with SIGCHI (Augmented Human '13)". Stuttgart, Germany: ACM SIGCHI, 2013.

```
##    1    2    3    4    5
## 5580 3797 3422 3216 3607
```

Let's split our data into train and test sets

```
set.seed(100)

fb_intrain <- sample(nrow(fb), nrow(fb)*0.8)

fb_train <- fb[fb_intrain, ]
fb_test <- fb[-fb_intrain, ]
```

I'm also going to use the `nearZeroVar()` function to eliminate features that has almost no variance as they can contribute nearly nothing to the model construction:

```
library(caret)

n0_var <- nearZeroVar(fb_train)
fb_train <- fb_train[,-n0_var]
```

And to make sure that the classes are equally represented in proportion in both the train and test sets:

```
prop.table(table(fb_train$classe))
```

```
##
##         1         2         3         4         5
## 0.2852137 0.1918201 0.1746831 0.1628337 0.1854494
```

```
prop.table(table(fb_test$classe))
```

```
##
##         1         2         3         4         5
## 0.2810191 0.2002548 0.1732484 0.1681529 0.1773248
```

I'm going to create our Random Forest now, using a 5-Fold Cross Validation, with 3 repeats. A 10-Fold with 5 repeats will look like this and is not the same as a 50-Fold:

The following chunk is set with the parameter `eval=F` because it may be time-consuming. You can try and repeat this exercise at home, but feel free to skip ahead to the next chunk where I've read my pre-built model into the workspace (and thus saving precious time):

```
set.seed(417)
# ctrl <- trainControl(method="repeatedcv", number=5, repeats=3)
# fb_forest <- train(classe ~ ., data=fb_train, method="rf", trControl = ctrl)
```

Read `fb_forest.RDS`, which is the `fb_forest` I created above:

```
fb_forest <- readRDS("model/fb_forest.RDS")
fb_forest
```

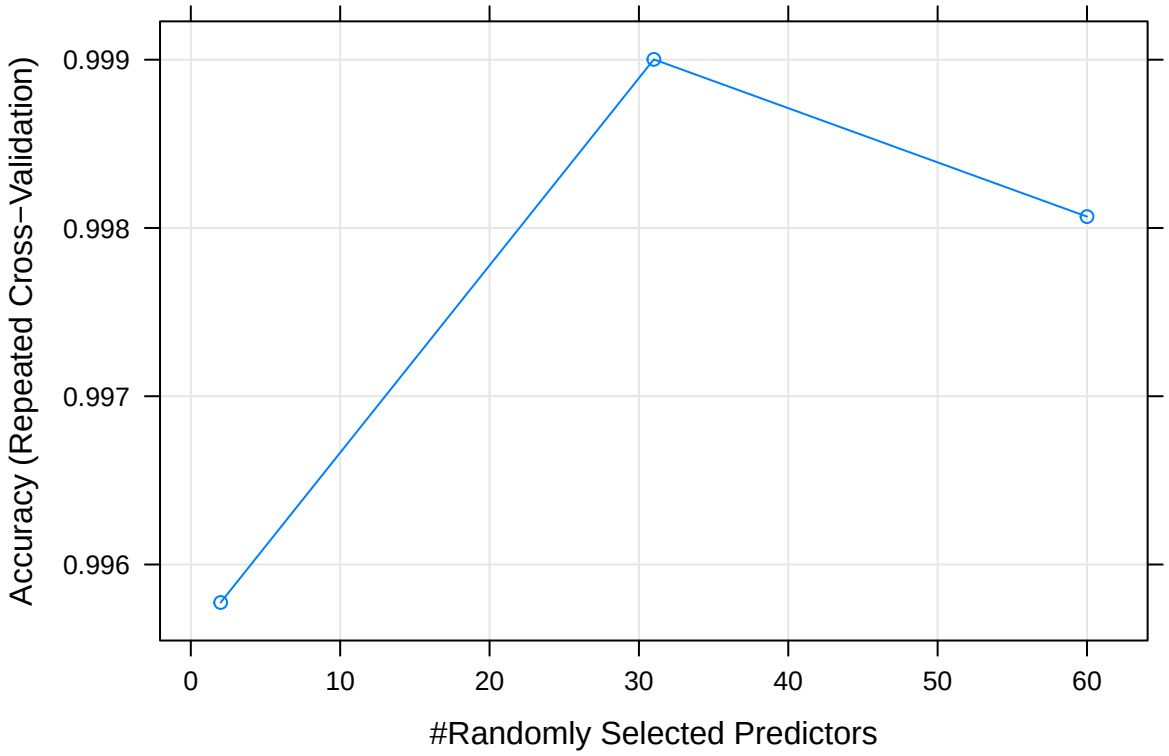
```
## Random Forest
##
## 15697 samples
##    56 predictor
##     5 classes: '1', '2', '3', '4', '5'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold, repeated 3 times)
## Summary of sample sizes: 12557, 12557, 12559, 12557, 12558, 12557, ...
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##    2    0.9957743  0.9946543
##   31    0.9990020  0.9987376
##   60    0.9980676  0.9975556
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 31.
```

From the above summary, we learn that the random forest have tried different values of `mtry`. This refers to the number of variables available for splitting at each tree node. Where in the case of a standard decision tree, all variables are considered at each point of splitting - this is not the case in random forest. Under a random forest model, each split in a particular tree would consider `mtry` amount of variables from the set of predictors available. At each split, a different random set of variables are selected as candidate before the tree assess the "purity" (or information gain) it can obtain from the candidate set and thus split it with that variable.

By default the `randomForest()` uses 500 trees, so we can imagine that all variables are used (just a question of how much they are used) at some point when searching for split points whilst growing the trees.

It also say Accuracy was used to select the optimal model, and that give us a model with `mtry=31`. We can get a visual representation of this selection process and confirm that the model constructed with `mtry=31` gives us the highest cross-validation accuracy.

```
plot(fb_forest)
```



Let's see how many times does our `fb_forest` prediction agree with the test set's label:

```
table(predict(fb_forest, fb_test), fb_test$classe)
```

```
##
##        1    2    3    4    5
##   1 1103    0    0    0    0
##   2    0  786    0    0    0
##   3    0    0  680    0    0
##   4    0    0    0  660    0
##   5    0    0    0    0  696
```

We observe that there are 0 mis-classification out of 3925 predictions.

```
sum(predict(fb_forest, fb_test)==fb_test$classe)
```

```
## [1] 3925
```

```
nrow(fb_test)
```

```
## [1] 3925
```

A 100% accuracy on unseen data! An impressive result for a seemingly complex structure.

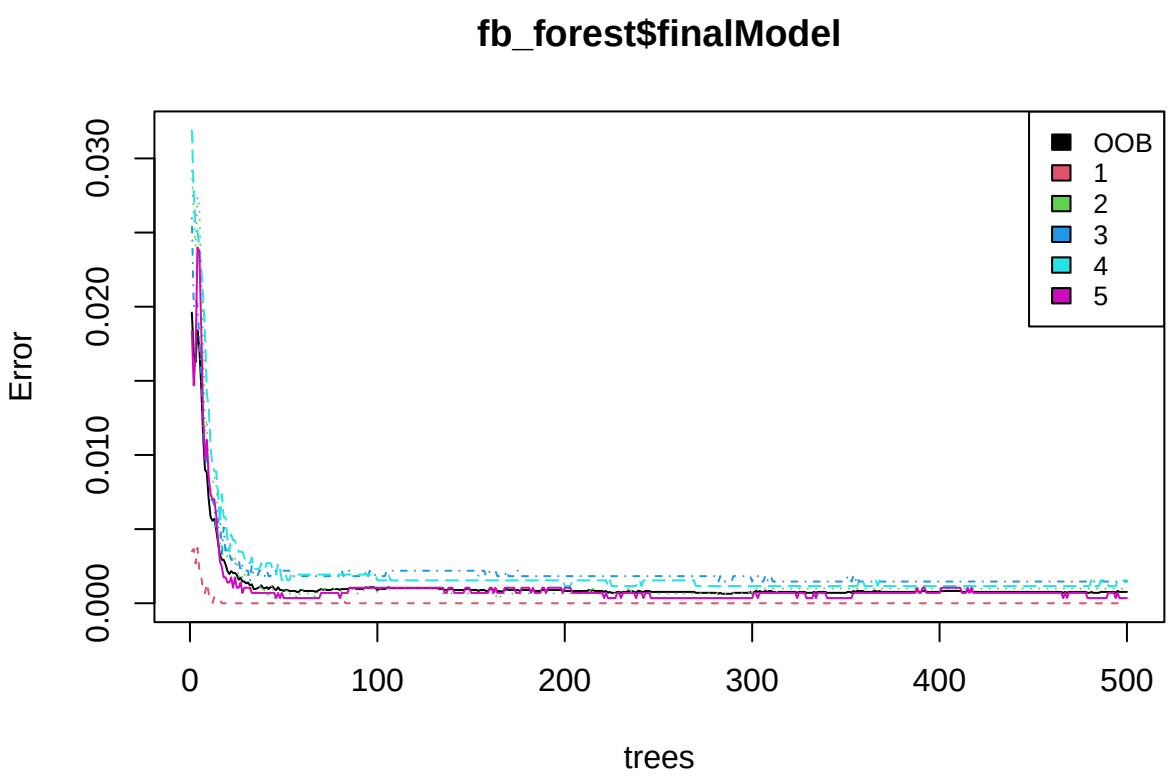
Other tips on our random forest model:
- We could also use `varImp(x)`, `importance(x)` or `varImpPlot(x)` to get a list of the most important variables used in our random forest. This give us a total decrease in node impurities (measured by the Gini index for classification and RSS for regression) from splitting on the variable averaged across the 500 trees

- To inspect the tree size (number of nodes), we can use `hist(treesize(x))`
- We can inspect each of the tree in the forest, we can use `getTree(x, k)` where `k` is the k-th tree we'd like to extract
- If you'd like to see how our model's accuracy changes as the number of trees in our forest grow, you can use `plot` in conjunction with `legend()`
- `x` has to be a `randomForest` object, so a useful tip is to save the `$finalModel` from `fb_forest` as a new variable `x`
- The default syntax for making predictions is `predict(model, test, type="response")` but `type` could be either "response", "prob" or "votes"

Most of the above tips are helpful in helping you diagnose and "learn" from the model. While many would argue that random forest, being a black box model, can offer no true information beyond its job in accuracy; my own experience is that paying special attention to attributes like variable importance for example often do help gain valuable information about our data.

To emphasize on the model diagnostics point above, let's plot our random forest. Observe from the plot above that the random forest has managed to classify all observations into class 1 (`classe=1`) relatively early on in the process, while the forest has ~30-40 trees; However, it seems to take much longer to correctly classify observations from class 4. In your own experiment, it is often helpful to use visualizations like these to "diagnose" your model and boost performance. A visualization such as the following may even hint at the necessity for a random forest to go beyond 500 trees, or if the complexity of data warrant such a case.

```
plot(fb_forest$finalModel)
legend("topright", colnames(fb_forest$finalModel$err.rate),col=1:6,cex=0.8,fill=1:6)
```



Finally, I would also add that using a random forest - we are not required to split our dataset into train, cross-validation and test sets. We did so merely for academic purposes. In practice, the random forest already have out-of-bag estimates (OOB) that can be used as a reliable estimate of its true accuracy on unseen examples.

Specifically, for each tree that our random forest constructs, any example not selected to train the decision model is used to test the tree's performance on unseen data. Then, for each observation in our dataset, our random forest count the votes to determine the final predicted class. This total error rate is the out-of-bag error rate.

Have we not partition a test set, we could have used the OOB estimate that is given to us by our random forest - as they represent an unbiased estimate of its accuracy on unseen data:

```
fb_forest$finalModel
```

```
##
## Call:
##  randomForest(x = x, y = y, mtry = param$mtry)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 31
##
##         OOB estimate of  error rate: 0.08%
## Confusion matrix:
##      1    2    3    4    5  class.error
## 1 4470    0    0    0    0 0.0000000000
## 2    2 3032    1    0    0 0.0009884679
```

```
## 3    0    4 2731    0    0 0.0014625229
## 4    0    0    3 2585    1 0.0015449981
## 5    0    0    0    1 2867 0.0003486750
```

A random forest is a high-performance algorithm and is a regular feature in many machine learning competitions. Empirically, many researchers have studied the performance of random forest compared to other classification algorithms and came to the following conclusion [12]:

> We evaluate 179 classifiers arising from 17 families (Bayesian, neural networks, support vector machines, decision trees, rule-based classifiers, boosting, bagging, random forests and other ensembles, generalized linear models, nearest-neighbors, partial least squares and principal component regression, logistic and multinomial regression, and other methods), implemented in Weka, R (with and without the caret package), C and Matlab, including all the relevant classifiers available today. We use 121 data sets, which represent the whole UCI data base (excluding the large-scale problems) and other own real problems, in order to achieve significant conclusions about the classifier behavior, not dependent on the data set collection. The classifiers most likely to be the bests are the random forest (RF) versions, the best of which (implemented in R and accessed via caret) achieves 94.1% of the maximum accuracy overcoming 90% in the 84.3% of the data sets … The random forest is clearly the best family of classifiers (3 out of 5 bests classifiers are RF), followed by SVM (4 classifiers in the top-10), neural networks and boosting ensembles (5 and 3 members in the top-20, respectively).

Graded Quiz

This section of the workshop needs to be completed in the classroom in order to obtain a score that count towards your final grade.

Learn-By-Building

Use any of the 3 classification algorithms you've learned to predict the risk status of a bank loan. The variable `default` in the dataset indicates whether the applicant did default on the loan issued by the bank. Start by reading the `loan.csv` dataset in, a dataset that is originally from Professor Dr. Hans Hofmann:

```
loans <- read.csv("data_input/loan.csv")
str(loans)
```

```
## 'data.frame':    1000 obs. of  17 variables:
##  $ checking_balance    : chr  "< 0 DM" "1 - 200 DM" "unknown" "< 0 DM" ...
##  $ months_loan_duration: int  6 48 12 42 24 36 24 36 12 30 ...
##  $ credit_history      : chr  "critical" "good" "critical" "good" ...
##  $ purpose             : chr  "furniture/appliances" "furniture/appliances" "education" "furniture/ap
##  $ amount              : int  1169 5951 2096 7882 4870 9055 2835 6948 3059 5234 ...
##  $ savings_balance     : chr  "unknown" "< 100 DM" "< 100 DM" "< 100 DM" ...
##  $ employment_duration : chr  "> 7 years" "1 - 4 years" "4 - 7 years" "4 - 7 years" ...
##  $ percent_of_income   : int  4 2 2 2 3 2 3 2 2 4 ...
##  $ years_at_residence  : int  4 2 3 4 4 4 4 2 4 2 ...
##  $ age                 : int  67 22 49 45 53 35 53 35 61 28 ...
##  $ other_credit        : chr  "none" "none" "none" "none" ...
```

[12] Fernández-Delgado, M. et. al, Do we Need Hundreds of Classifiers to Solve Real World Classification Problems?

```
##  $ housing             : chr  "own" "own" "own" "other" ...
##  $ existing_loans_count: int  2 1 1 1 2 1 1 1 1 2 ...
##  $ job                 : chr  "skilled" "skilled" "unskilled" "skilled" ...
##  $ dependents          : int  1 1 2 2 2 2 1 1 1 1 ...
##  $ phone               : chr  "yes" "no" "no" "no" ...
##  $ default             : chr  "no" "yes" "no" "no" ...
```

Let's go over some of these features:
`checking_balance` and `savings_balance`: Status of existing checking / savings account
`credit_history`: Between critical, good, perfect, poor and very good
`purpose`: Between business, car(new), car(used), education, furniture and renovations
`employment_duration`: Present employment since
`percent_of_income`: Installment rate in percentage of disposable income
`years_at_residence`: Present residence since
`other_credit`: Other installment plans (bank / store)
`housing`: Between rent, own, or for free
`job`: Between management, skilled, unskilled and unemployed
`dependents`: Number of people being liable to provide maintenance for
`phone`: Between none and yes (registered under customer name)

Use an R Markdown document to lay out your process, and explain the methodology in 1 or 2 brief paragraph. The student should be awarded the full (3) points when:
- The preprocessing steps are done, and the student show an understanding of holding out a test / cross validation set for an estimate of the model's performance on unseen data
- The model's performance is sufficiently explained (accuracy may **not** be the most helpful metric here! Recall about what you've learned regarding specificity and sensitivity)
- The student demonstrated extra effort in evaluating his/her model, and proposes ways to improve the accuracy obtained from the initial model

Annotations