Practical Machine Learning Notes

Coursera Course by John Hopkins University

INSTRUCTORS: Dr. Jeff Leek, Dr. Brian Caffo, Dr. Roger D. Peng

Contents

Intro

- This course covers the basic ideas behind machine learning/prediction
  - Study Design - training vs. test sets
  - Conceptual issues - out of sample error, overfitting, ROC curves
  - Practical Implementation - the caret package
- What this course depends on:
  - The Data Scientist's Toolbox
  - R Programming
- What would be useful
  - Exploratory Analysis
  - Reproducible Research
  - Regression Models
  - (Notes on these 5 courses are all in my GitHub repoes)

GitHub Link for Lectures

Practical Machine Learning lectures on GitHub

Course Book

The book for this course is available on this site

Instructor's Note

"*Welcome to Practical Machine Learning! This course will focus on developing the tools and techniques for understanding, building, and testing prediction functions.*

These tools are at the center of the Data Science revolution. Many researchers, companies, and governmental organizations would like to use the cheap and abundant data they are collecting to predict what customers will like, what services to offer, or how to improve people's lives.

Jeff Leek and the Data Science Track Team"

Prediction, Errors, and Cross Validation

Prediction

Prediction Motivation

- Who predicts things?
  - Local governments -> pension payments
  - Google -> whether you will click on an ad
  - Amazon -> what movies you will watch
  - Insurance companies -> what your risk of death is
  - Johns Hopkins -> who will succeed in their programs
- Why predict things
  - Glory (Nerd cred for accomplishing certain feats)
    - A lot of competitions are hosted on **Kaggle**
  - Riches (Completing some competition that offers a reward)
  - Save lives
    - **On cotype DX** reveals the underlying biology that changes treatment decisions 37% of the time.

More Resources

- **A course on more advanced material about ML**
- **List of machine learning resources on Quora**
- **List of machine learning resources from Science**

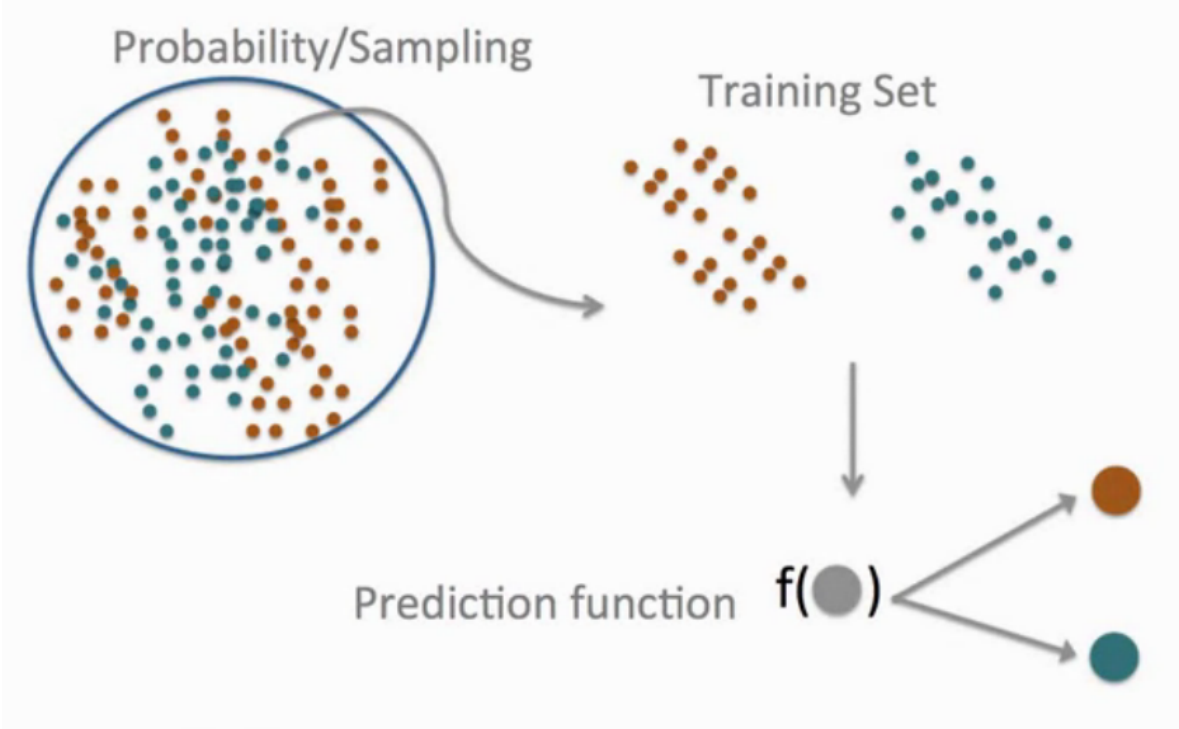
- **Advanced notes from MIT open courseware**
- **Advanced notes from CMU**
- **Kaggle - machine learning competitions**

What is Prediction?

Main Idea

- One focus of ML is on what algorithms are the best for extracting information and using it to predict.

- Although the method used for producing a training set is also quite important



- One starts off with a dataset

1) One uses Probability/Sampling to select a Training Set
2) One measures characteristics of this training set to create a Predicition Function
3) One then uses the Prediction Function to take an uncolored dot and predict if it's red or blue
4) One would then go on to test how well their Prediction Function works

What Can Go Wrong

- An example is **Google Flu trends** (**A free overview of the issue witht he accuracy**)
  - Google tried to predict rate of flu using what people would search
  - Originally the algorithm was able to represent how many cases would appear in a region within a certain time
  - Although they didn't account for the fact that the terms would change over time
  - The way the terms were being used wasn't well understood so when terms changed they weren't able to accurately account for the change.
  - It also overestimated as it the search terms it looked at were often cofactors with other illnesses

Componets of a Predictor

1) Question

- Any problem in data science starts with a question, "What are you trying to predict and what are you trying to predict it with?"

2) Input Data

- Collect best input data you can to use to predict

3) Features

- From that data one builds features that they will use to predict

4) Algorithm

- One uses ML algorithms to develop a function

5) Parameters

- Estimate parameters of the algorithm

6) Evaluation

- Apply algorithm to a data set to evaluate how well the algorithm works

Example

- Start with a general question, "Can I automatically detect emails that are SPAM and those that are not?"

- Make the question more concrete, "Can I use quantitative characteristics of the emails to classify them as SPAM/HAM?"

- Find input data
  - In this instance there is data avaliable in R via the `kernlab` package

  - Note that this data set won't necessarily be the perfect data as it doesn't contain all the emails ever sent, or the emails sent to you personally

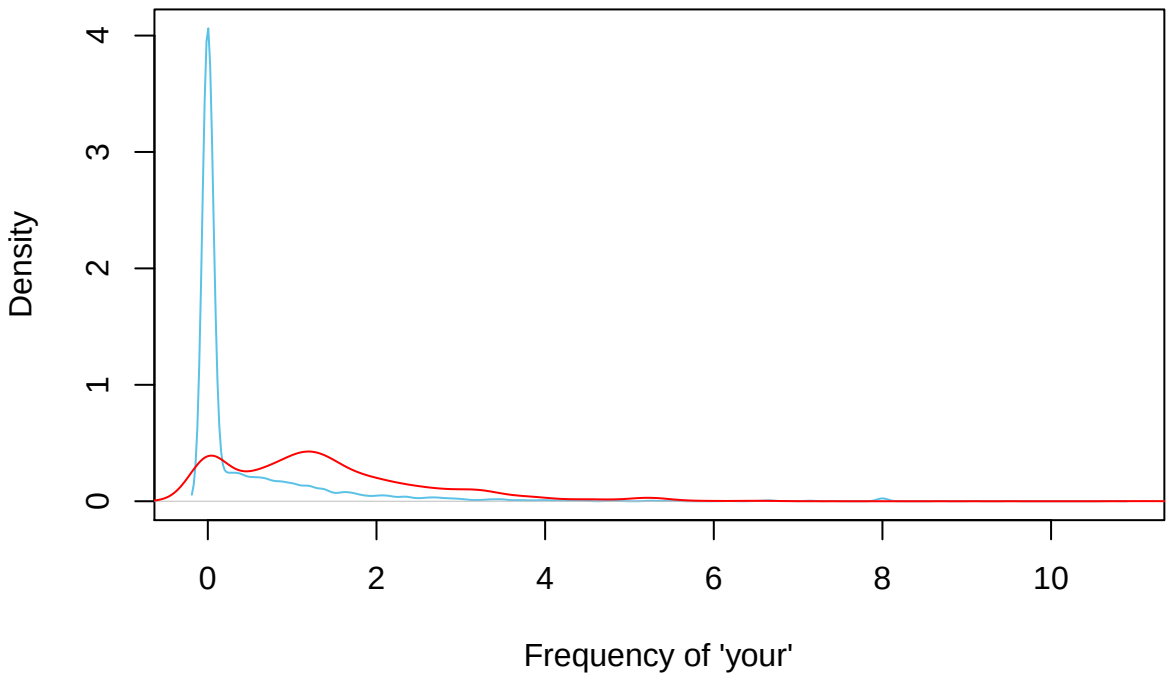
- Quantify features, such as the frequency of certain words or typeface. The `spam` dataset from `kernlab` contains these types of frequency.

```
library(kernlab)
data(spam)
str(spam)
```

```
## 'data.frame':    4601 obs. of  58 variables:
##  $ make             : num  0 0.21 0.06 0 0 0 0 0 0.15 0.06 ...
##  $ address          : num  0.64 0.28 0 0 0 0 0 0 0 0.12 ...
##  $ all              : num  0.64 0.5 0.71 0 0 0 0 0 0.46 0.77 ...
##  $ num3d            : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ our              : num  0.32 0.14 1.23 0.63 0.63 1.85 1.92 1.88 0.61 0.19 ...
##  $ over             : num  0 0.28 0.19 0 0 0 0 0 0 0.32 ...
##  $ remove           : num  0 0.21 0.19 0.31 0.31 0 0 0 0.3 0.38 ...
##  $ internet         : num  0 0.07 0.12 0.63 0.63 1.85 0 1.88 0 0 ...
##  $ order            : num  0 0 0.64 0.31 0.31 0 0 0 0.92 0.06 ...
##  $ mail             : num  0 0.94 0.25 0.63 0.63 0 0.64 0 0.76 0 ...
##  $ receive          : num  0 0.21 0.38 0.31 0.31 0 0.96 0 0.76 0 ...
##  $ will             : num  0.64 0.79 0.45 0.31 0.31 0 1.28 0 0.92 0.64 ...
##  $ people           : num  0 0.65 0.12 0.31 0.31 0 0 0 0 0.25 ...
##  $ report           : num  0 0.21 0 0 0 0 0 0 0 0 ...
##  $ addresses        : num  0 0.14 1.75 0 0 0 0 0 0 0.12 ...
##  $ free             : num  0.32 0.14 0.06 0.31 0.31 0 0.96 0 0 0 ...
##  $ business         : num  0 0.07 0.06 0 0 0 0 0 0 0 ...
##  $ email            : num  1.29 0.28 1.03 0 0 0 0.32 0 0.15 0.12 ...
##  $ you              : num  1.93 3.47 1.36 3.18 3.18 0 3.85 0 1.23 1.67 ...
##  $ credit           : num  0 0 0.32 0 0 0 0 0 3.53 0.06 ...
##  $ your             : num  0.96 1.59 0.51 0.31 0.31 0 0.64 0 2 0.71 ...
##  $ font             : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ num000           : num  0 0.43 1.16 0 0 0 0 0 0 0.19 ...
##  $ money            : num  0 0.43 0.06 0 0 0 0 0 0.15 0 ...
##  $ hp               : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ hpl              : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ george           : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ num650           : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ lab              : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ labs             : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ telnet           : num  0 0 0 0 0 0 0 0 0 0 ...
```

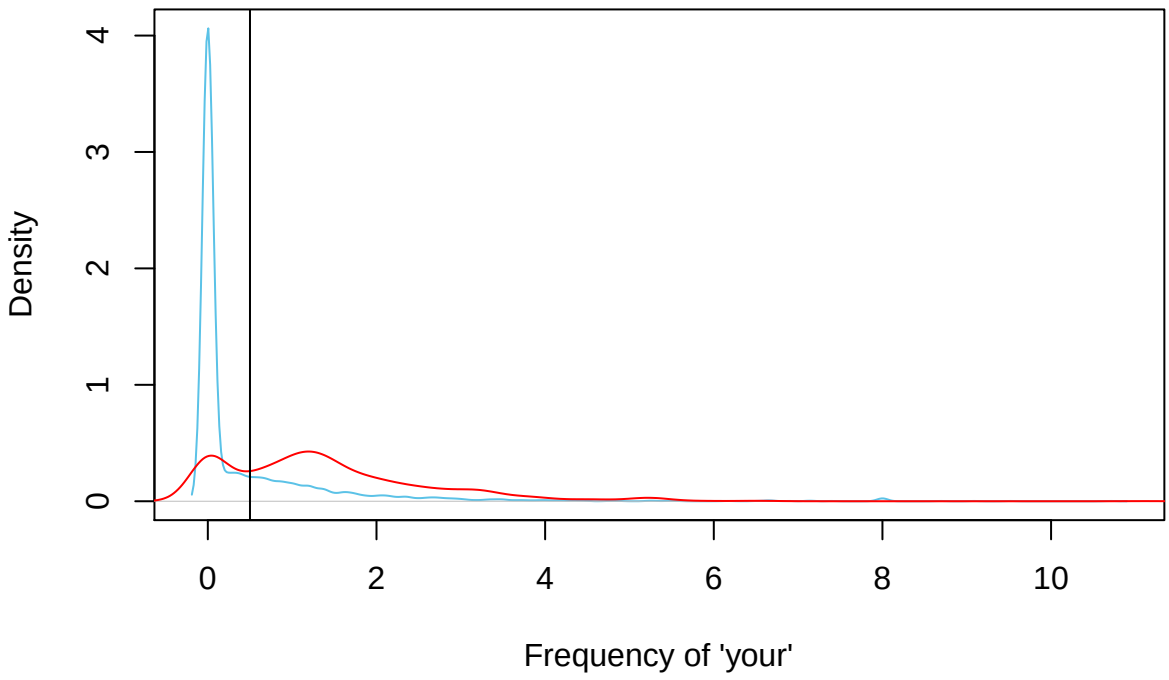
```
##  $ num857           : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ data             : num  0 0 0 0 0 0 0 0 0.15 0 ...
##  $ num415           : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ num85            : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ technology       : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ num1999          : num  0 0.07 0 0 0 0 0 0 0 0 ...
##  $ parts            : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ pm               : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ direct           : num  0 0 0.06 0 0 0 0 0 0 0 ...
##  $ cs               : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ meeting          : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ original         : num  0 0 0.12 0 0 0 0 0 0.3 0 ...
##  $ project          : num  0 0 0 0 0 0 0 0 0 0.06 ...
##  $ re               : num  0 0 0.06 0 0 0 0 0 0 0 ...
##  $ edu              : num  0 0 0.06 0 0 0 0 0 0 0 ...
##  $ table            : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ conference       : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ charSemicolon    : num  0 0 0.01 0 0 0 0 0 0 0.04 ...
##  $ charRoundbracket : num  0 0.132 0.143 0.137 0.135 0.223 0.054 0.206 0.271 0.03 ...
##  $ charSquarebracket: num  0 0 0 0 0 0 0 0 0 0 ...
##  $ charExclamation  : num  0.778 0.372 0.276 0.137 0.135 0 0.164 0 0.181 0.244 ...
##  $ charDollar       : num  0 0.18 0.184 0 0 0 0.054 0 0.203 0.081 ...
##  $ charHash         : num  0 0.048 0.01 0 0 0 0 0 0.022 0 ...
##  $ capitalAve       : num  3.76 5.11 9.82 3.54 3.54 ...
##  $ capitalLong      : num  61 101 485 40 40 15 4 11 445 43 ...
##  $ capitalTotal     : num  278 1028 2259 191 191 ...
##  $ type             : Factor w/ 2 levels "nonspam","spam": 2 2 2 2 2 2 2 2 2 2 ...
```

```
plot(density(spam$your[spam$type=="nonspam"]),
     col = "#5BC2E7", main = "", xlab = "Frequency of 'your'")
lines(density(spam$your[spam$type=="spam"]), col = "#FF0000")
```



- It can be seen here "your" appears more often in SPAM emails than it does in HAM
- One could use this idea to create a cut-off point for predicting if a message is SPAM
- The proposed algorithm
  - Find a value of C
  - If the frequency of $'your' > C$ predict the message is SPAM

```
plot(density(spam$your[spam$type=="nonspam"]),
     col = "#5BC2E7", main = "", xlab = "Frequency of 'your'")
lines(density(spam$your[spam$type=="spam"]), col = "#FF0000")
abline(v = 0.5, col = "#000000")
```



- Choosing 0.5 would contain most spam messages and avoid the second spike of HAM emails

- We then evaluate this predictor

```
prediction <- ifelse(spam$your > 0.5, "spam", "nonspam")
res <- table(prediction, spam$type)/length(spam$type)
res
```

```
##
## prediction   nonspam      spam
##    nonspam 0.4590306 0.1017170
##    spam    0.1469246 0.2923278
```

- In this case our accuracy is $0.459 + 0.2923 = 0.7514$, or an accuracy of approximately 75.14%, although this is an opptamistic measure of the overall error, which will be discussed further later.

Relative Importance of Steps

question > data > features/variables > algorithms
"*The commbinaiton of some data and an aching desire for an answer does not ensure that a reasonable answer can be extracted from a given body of data.*" -John Tukey

- In other words, an important component of prediction is knowing when to give up, that is

that the data is not sufficient

Input Data: Garbage in = Garbage out

1. May be easy (movie ratings -> new movie ratings)
2. May be harder (gene expression data -> disease)
3. Depends on what is a "good prediction".
4. Often more **data > better models**
5. The most important step is collecting the right data

Features: They matter!

- Properties of good features
  - Lead to data compression
  - Retain relevant information
  - Are created based on expert application knowledge
- Common mistakes
  - Trying to automate feature selection (Although they may be automated with care)
  - Not paying attention to data-specific quirks
  - Throwing away information unnecessarily

Algorithm: They Matter Less Than You'd Think

- The above table shows that the Linear Discrimenate Analysis (Lindisc) error often was not that far off from the best method
- Using the best approach doesn't always largely improve the error

Issues to Consider

- The "Best" machine learning method would be:
  - Interpretable
    - If predictor is to be presented to an uninformed audience you'd want to to be understandable by them
  - Simple
    - Helps with interpretablity

TABLE 1
Performance of linear discriminant analysis and the best result we found on ten randomly selected data sets

Data set	Best method e.r.	Lindisc e.r.	Default rule	Prop linear
Segmentation	0.0140	0.083	0.760	0.907
Pima	0.1979	0.221	0.350	0.848
House-votes16	0.0270	0.046	0.386	0.948
Vehicle	0.1450	0.216	0.750	0.883
Satimage	0.0850	0.160	0.758	0.889
Heart Cleveland	0.1410	0.141	0.560	1.000
Splice	0.0330	0.057	0.475	0.945
Waveform21	0.0035	0.004	0.667	0.999
Led7	0.2650	0.265	0.900	1.000
Breast Wisconsin	0.0260	0.038	0.345	0.963

Figure 1: Linear vs Model

  - Accurate
    - Getting a model to be interpretable can sometimes hurt the accuracy
  - Fast
    - Quick build the model, train, and test
  - Scalable
    - Easy to apply to a large dataset (either fast or parallelizable)

Prediction is About Accuracy Tradeoffs

- Tradeoffs are made for interpretability, speed, simplicity, or scalability.
- Interpretability matters, decision tree-like results are more interpretable
  - "**if** total cholesterol ≥ 160 **and** they smoke **then** *10 year CHD risk* $\geq$ *5%* **else if** they smoke **and** systolic blood pressure ≥ 140 **then** *10 year CHD risk* $\geq 5\%$ **else** *10 year CHD risk* $< 5\%$"
- Scalability matter
  - in "The Netflix $1 Million Challenge" Netflix never implemented the solution itself because the algorithm wasn't scalable and took way too long on the big data sets that Neflix was working with, so they went with something that was less accurate but more scalable.

Errors

In and Out of Sample Errors

- **In Sample Error** - Sometimes called *resubstitution error*. The error rate you get on the same data set you used to build your predictor.
- **Out of Sample Error** - Sometimes called *generalization error*. The error rate you get on a new data set.
- Key Ideas

1) Out of sample error is what you care about
2) In sample error $<$ out of sample error

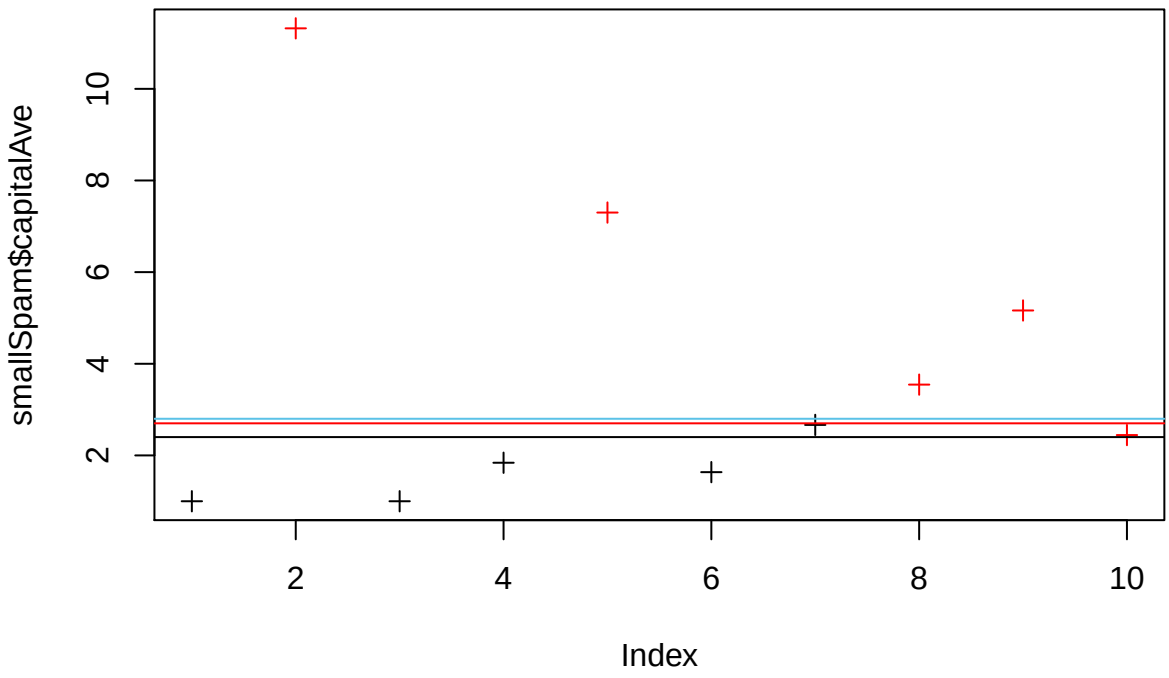
- Sometimes you want to give up some accuracy on the data you have to have greater accuracy on unkown data.

3) The reason is overfitting

- Matching your algorithm to the noise of the data you have

Spam Example

```
library(kernlab)
data(spam)
RNGkind(sample.kind = "Rounding")
set.seed(333)
smallSpam <- spam[sample(dim(spam)[1], size = 10),]
spamLabel <- (smallSpam$type == "spam")*1 + 1
plot(smallSpam$capitalAve, col = spamLabel, pch = 3)
abline(h = 2.7, col = "#FF0000")
abline(h = 2.40, col = "#000000")
abline(h = 2.80, col = "#5BC2E7")
```



Prediction 1

- capitalAve > 2.7 = "spam"

- capitalAve < 2.40 = "nonspam"
- We can add 2 params to make the prediction perfect for the training set
  - capitalAve between 2.40 and 2.45 = "spam"

  - capitalAve between 2.45 and 2.7 = "nonspam"

```
rule1 <- function(x){
  prediction <- rep(NA, length(x))
  prediction[x > 2.7] <- "spam"
  prediction[x < 2.40] <- "nonspam"
  prediction[(x >= 2.40 & x <= 2.45)] <- "spam"
  prediction[(x > 2.45 & x <= 2.70)] <- "nonspam"
  return(prediction)
}
table(rule1(smallSpam$capitalAve),smallSpam$type)
```

```
##
##             nonspam spam
##   nonspam       5    0
```

```
##    spam          0    5
```

Prediction 2

- (Note: The blue line in the plot is for 2.8)
- capitalAve > 2.80 = "spam"
- capitalAve ≤ 2.80 = "nonspam"
- This algorithm won't be perfect on the training data

```
rule2 <- function(x){
  prediction <- rep(NA, length(x))
  prediction[x > 2.8] <- "spam"
  prediction[x <= 2.8] <- "nonspam"
  return(prediction)
}
table(rule2(smallSpam$capitalAve), smallSpam$type)
```

```
##
##           nonspam spam
##   nonspam       5    1
##   spam          0    4
```

Apply 2 rulesets to all spam data

```
table(rule1(spam$capitalAve),spam$type)
```

```
##
##           nonspam spam
##   nonspam    2141  588
##   spam        647 1225
```

```
table(rule2(spam$capitalAve),spam$type)
```

```
##
##           nonspam spam
##   nonspam    2224  642
##   spam        564 1171
```

```
paste0("Rule 1 accuracy: ",
       mean(rule1(spam$capitalAve) == spam$type))
```

```
## [1] "Rule 1 accuracy: 0.731580091284503"
```

```
paste0("Rule 2 accuracy: ",
       mean(rule2(spam$capitalAve) == spam$type))
```

```
## [1] "Rule 2 accuracy: 0.737883068898066"
```

```
paste0("Rule 1 total correct: ",
       sum(rule1(spam$capitalAve) == spam$type))
```

```
## [1] "Rule 1 total correct: 3366"
```

```
paste0("Rule 2 total correct: ",
       sum(rule2(spam$capitalAve) == spam$type))
```

```
## [1] "Rule 2 total correct: 3395"
```

Overfitting

- Why is the ruleset with a *better* out of sample error (`rule2`) the one with a *worse* in sample error?
  - It's because we overfitted `rule1`
  - **Wikipedia on Overfitting**
- All data have two parts
  - Signal - Part we are trying to use to predict
  - Noise - Random variation in data set
- The goal of a predictor is to find the signal and ignore the noise
- You can always design a perfect in-sample predictor
  - Doing this will capture botht he signal and the noise
  - As such a predictor won't perform as well on new samples

Prediction Study Design

1) Define your error rate
2) Split data into:

- Training set to build model
- Testing set to validate model
- Validation set (optional) to also validate the model

3) On the training set pick features (using cross-validation to pick which features are most important in your model)
4) On the training set pick prediction function (also using cross-validation)
5a) If no validation set:

- Apply the best model to the test set exactly 1 time

  - If we apply multiple models to the test set and pick the best one we're kind of using the test set to train the model, giving an optimistic error rate
    5b) If there is a validation set:
- Apply the model the the test set and refine the model

- Then apply best model to validation set once

Benchmarks

- One should know what the prediction benchmarks are for their algorithm to help troubleshoot when something is going wrong. Often a benchmark is something like "all zeros" which tells the error rate if all values were set to 0, pretty much just ignore all the features of the dataset and taking a general average.

Study Design of Netflix Contest

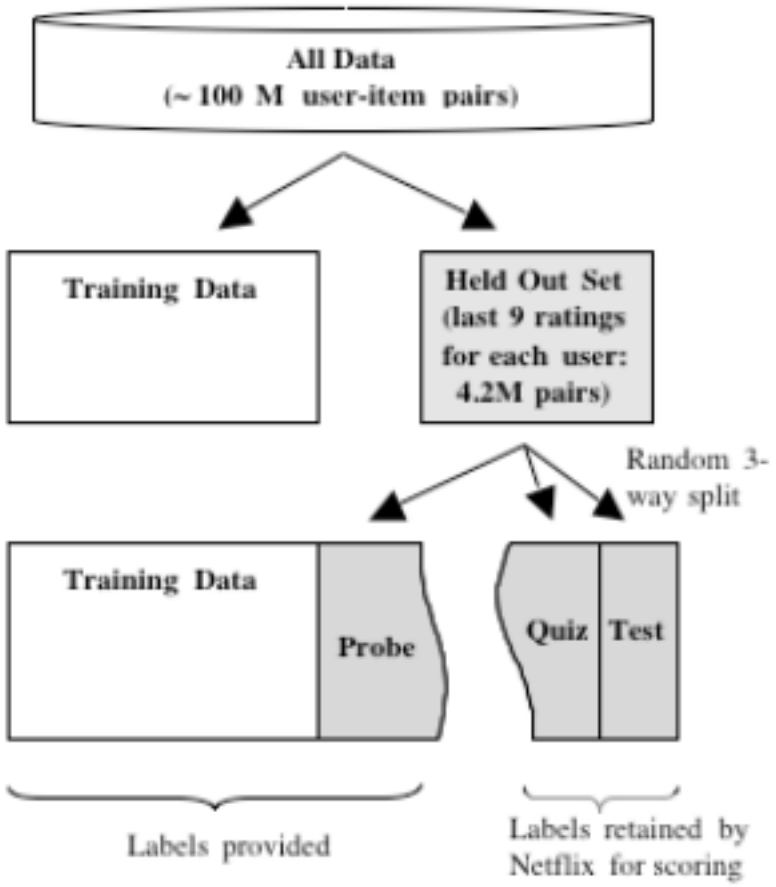


Figure 2: Netflix Design

- Of all the data they split it into a training set (*Training Data*) that they shared with competators and a test & validation set (*Held Out Set*)that they did not share.

- They shared a test set (*Probe*) so competators could test their out of sample error.

- They would then take your model and test it on the *Quiz* set, although one could submit multiple models and get a scoring from the *Quiz* set. So they held out the last bit of data as a validation set (*Test*) that would be used at the end of the competition on each model only once.

Avoid Small Sample Sizes

- Suppose you are predicting a binary outcome
  - Diseased/healthy
  - (Not) Clicking on an ad
- One classifier is flipping a coin
- Probability of perfect classification is approximately $(\frac{1}{2})^{samplesize}$
  - $n = 1$ flipping coin 50% change of 100% accuracy
  - $n = 2$ flipping coin 25% change of 100% accuracy
  - $n = 10$ flipping coin 0.10% change of 100% accuracy
- So lower sample sizes make it harder to know if your high accuracy is from chance or true.

Rules of Thumb for Prediction Study Design

- If you have a large sample size
  - 60% training
  - 20% test
  - 20% validation
- If you have a medium smaple size
  - 60% training
  - 40% testing
- If you have a small sample size
  - Do cross validation
  - Reprot caveat of small sample size

Some Principles to Remember

- Set the test/validation set aside and *don't look at it*
- In general *randomly* sample training and test sets
- Your data sets must reflect structure of the problem
  - If predictions evolve with time split train/test in time chunks (called backtesting in finance)
- All subsets should reflect as much diversity as possible
  - Random assignment does this
  - You can also try to balance by features - but this is tricky

Types of Errors

Basic Terms

In general, **Positive** = identified and **Negative** = rejected. Whereas **True** and **False** indicate correctness. Therefore:
* **True positive** = correctly identified
* **False positive** = incorrectly identified
* **True negative** = correctly rejected
* **False negative** = incorrectly rejected

Medical testing example:
* **True positive** = Sick people correctly diagnosed as sick
* **False positive** = Healthy people incorrectly identified as sick
* **True negative** = Healthy people correctly identified as healthy
* **False negative** = Sick people incorrectly identified as healthy

- Sensitivity and Specificity
  - **Sensitivity** - **True Positve Rate**, *Pr(positive test|disease)*, proportion of actual positives that are correctly identified
  - **Specificity** - **True Negative Rate**, *Pr(negative test|no disease)* proportion of actual negatives that are correctly identified
  - High amount of either ussually entails a high amount of the False ... Rate of the respective type, as ensuring you get all the positive/negative usually means you have to include some of the respective false samples.
  - Sensitivity is likely to diagnois a positive (**Sen**tence the innocent)
  - Specificity is going to be sure the positives are positive, even if they miss some (**Sp**are the innocent)
- Other key quantities
  - **Positive Predictive Value** - *Pr(disease|positive test)*

- **Negative Predictive Value** - *Pr(no disease | negative test)*
- **Accuracy** - *Pr(correct outcome) = Pr(positive test/disease) + Pr(negative test/no disease)*

Key Quantities as Fractions

```
##                 Actually_Positive Actually_Negative
## Tested_Postive                 TP                FP
## Tested_Negative                FN                TN
```

- Sensitivity = $\frac{TP}{(TP+FN)}$
- Specificity = $\frac{TN}{(FP+TN)}$
- Positive Predictive Value = $\frac{TP}{(TP+FP)}$
- Negative Predictive Value = $\frac{TN}{(FN+TN)}$
- Accuracy = $\frac{(TP+TN)}{(TP+FP+FN+TN)}$

Screening Tests Example

Assume that some disease has a 0.1% prevalence in the population. Assume we have a test kit for that disease that works with 99% sensitivity and 99% specificity. What is the probability of a person having the disease given the test result is positive, if we randomly select a subject from: (We'll look at the expected values if we sampled 100000 people) * The general population?

```
##                 Actually_Positive Actually_Negative
## Tested_Postive                 99               999
## Tested_Negative                 1             98901
```

- Sensitivity = $\frac{99}{(99+1)} = 99\%$
- Specificity = $\frac{98901}{(999+98901)} = 99\%$
- Positive Predictive Value = $\frac{99}{(99+999)} = 9.016\%$
- Negative Predictive Value = $\frac{98901}{(1+98901)} = 99.999\%$
- Accuracy = $\frac{(99+98901)}{100000} = 99\%$
- A high risk sub-population with 10% disease prevalence

```
##                 Actually_Positive Actually_Negative
## Tested_Postive                9900               900
## Tested_Negative                100             89100
```

- Sensitivity = $\frac{9900}{(9900+100)} = 99\%$
- Specificity = $\frac{89100}{(900+89100)} = 99\%$
- Positive Predictive Value = $\frac{9900}{(9900+900)} = 91.667\%$
- Negative Predictive Value = $\frac{89100}{(100+89100)} = 99.888\%$
- Accuracy = $\frac{(9900+89100)}{100000} = 99\%$
- This low Postitive Predictive Value shows the issues with predicting a very rare event from a population versus something that's more prevalent.

For Continous Data

We evaluate error by *mean squared error* and it's root
* **Mean squared error (MSE)** - $\frac{1}{n}\sum_{i=1}^{n}(Prediction_i - Truth_i)^2$
* **Root mean squared error (RMSE)** - $\sqrt{\frac{1}{n}\sum_{i=1}^{n}(Prediction_i - Truth_i)^2}$

Common Error Measures

1. Mean squared error (or root mean squared error)
   - Continous data, sensitive to outliers
2. Median absolute deviation
   - Median of distance between predicted and observed and take absolute value, rather than squared distance (this requries all values to be positive).
   - Continous data, often more robust
3. Sensitivity (recall)
   - If you want few missed positives
4. Specificity
   - If you want few negatives called positives
5. Accuracy

- Weights false positives/negatives equally

6. Concordance

- An example is **kappa**

7. Predictive value of a positive (precision)

- When you are screening and prevalence is low

Receiver Operating Characteristics (ROC Curves)

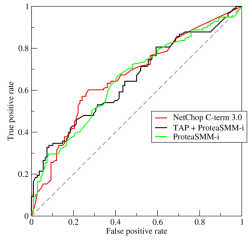
- Used to meaure the quality of a prediction algorithm

- **Wikipedia**
- Predictions are often quantitative
  - Proabbility fo being alive
  - Prediction on a scale from 1 to 10

- The *cutoff* you choose gives different results

- The curve informs you of the tradeoff of giving up some specificity for sensitivity (or vice versa)

- The curves plot the $P(FP)$ (x-axis) versus $P(TP)$ (y-axis)



Area Under the Curve

- The area under the curve (AKA the integral) describes the effectiveness of a given algorithm

- AUC = 0.5: random guessing

- AUC = 1: perfect classifier (given a certain value of the perdiction algorithm)
  - As such the closer to the top left of the plot a curve is the better it is.

- In general (depending on field & probability) an AUC above 0.8 is considered "good"

Cross Validation

Cross Validation

- A widely used tool for detecting relevant features and building models.

Key Ideas

1. Accuracy on the training set (resubstitution accuracy) is optimistic
2. A better estimate comes from an independent set (test set accuracy)
3. But we can't use the test set when building the model or it becomems part of the training set
4. So we estimate the test set accuracy with the training set

Cross-Validation Approach:
1. Use the training set
2. Split it into training/test sets (seperate from actual test set)
3. Build a model on the training set
4. Evaluate on the test set
5. Repeat with new training/test sets and average the estimated errors

What Cross-Validation is used for:
1. Picking variables to include in a model
2. Picking th etype of prediction function to use
3. Picking the parameters in the prediction function
4. Comparing different predictors

Ways to Pick Subsets

- Breaks data into K equal sized data sets.
- Leave out 1 sample then train on all the others, repeat for all samples

Considerations

- For time series data, data must be used in "chunks"
- For k-fold cross validation
  - Larger k = less bias, more variance
  - Smaller k = more bias, less variance
- Random sampling must be done *without replacement*
- Random sampling with replacement is the *bootstrap*
  - Underestimates of the error

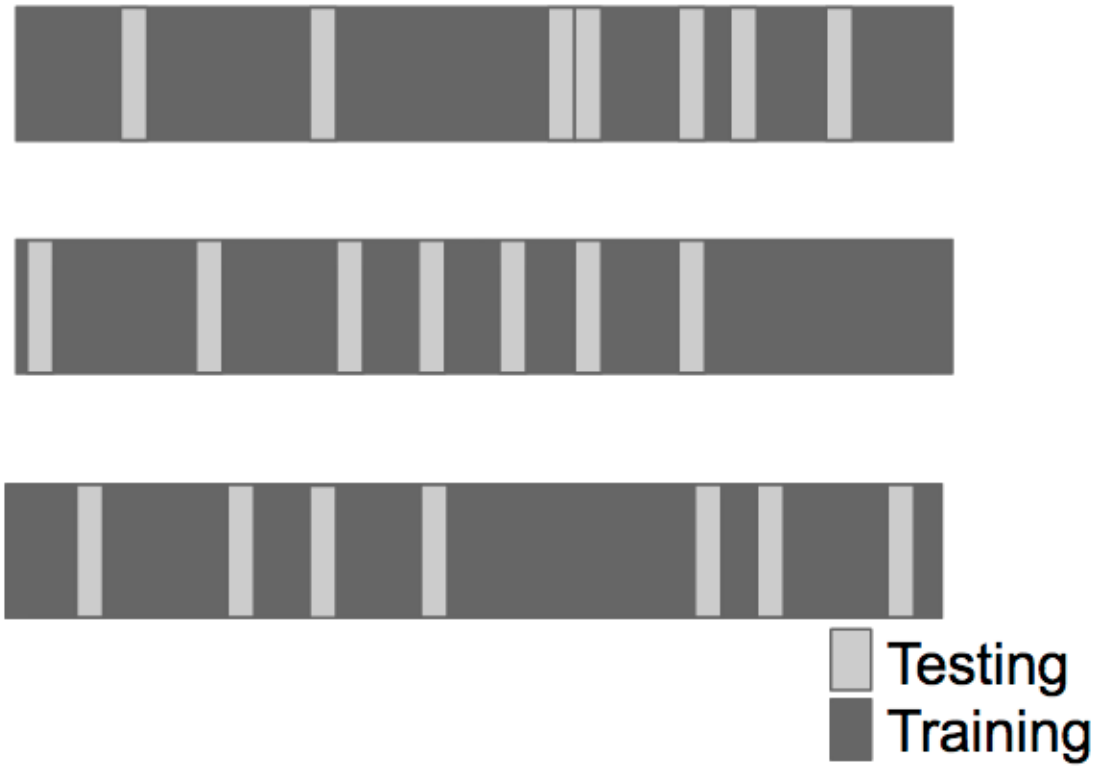


Figure 3: Random Subsampling

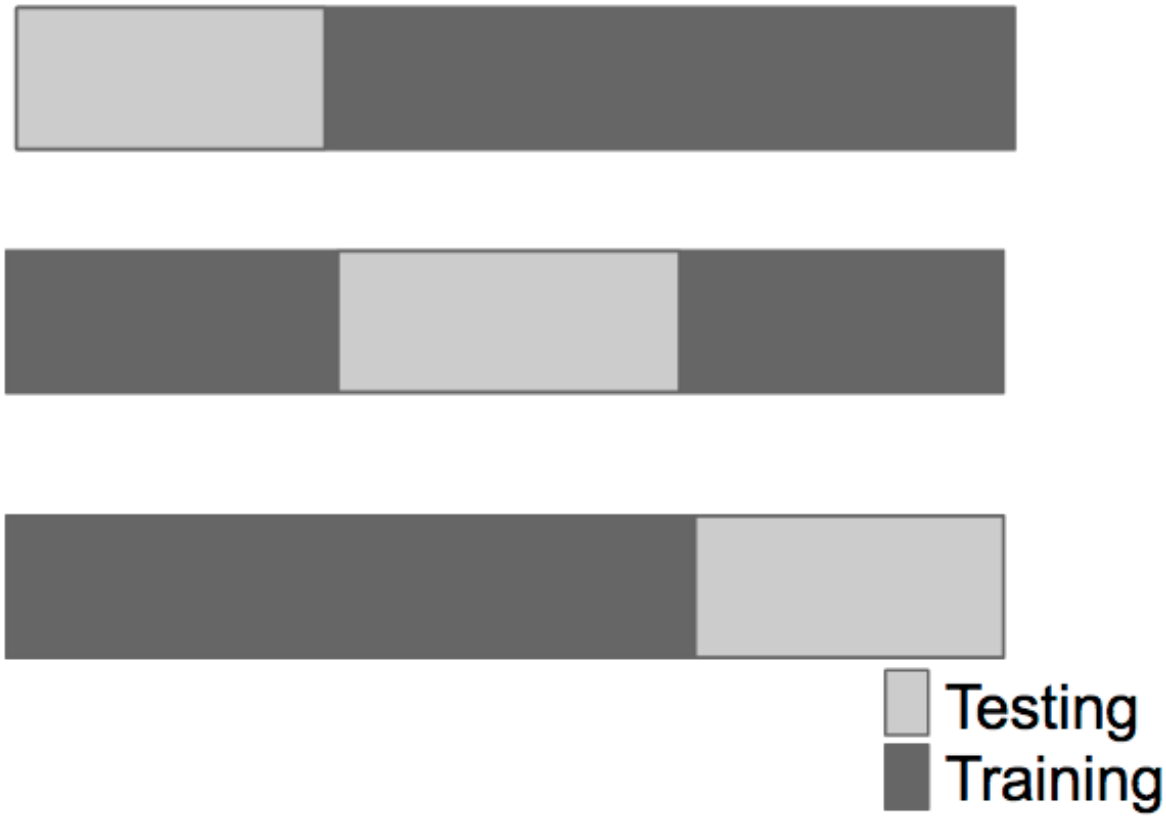


Figure 4: K-folds

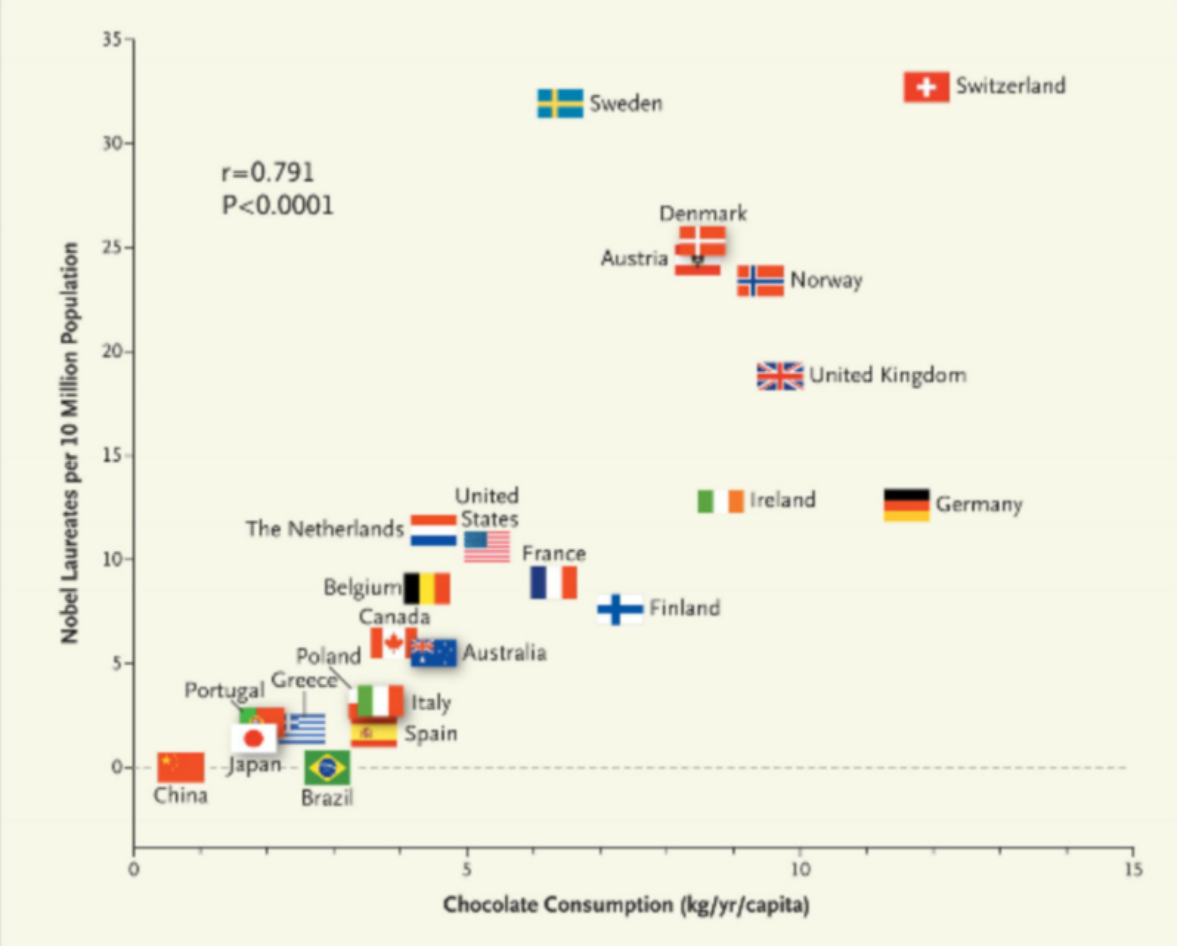
Figure 5: Leave One Out

  - Can be corrected, but it is complicated (**0.632 Bootstrap**)

- If you cross-validate to pick predictors estimate you must estiamte errors on independent data.

What Data Should You Use?

- If you want to predict something about X use data related to X (Use like to predict like)
  - To predict player performance use data about player performance
  - To predict movie preferences use data about movie preferences
  - To predict hospitalizations sue data about hospitalizations
- The closer the data is to the process you want to predict the better the predictions will be
- The looser connection the harder the prediction may be
  - *On Cotype DX* uses gene expression to predict one's longevity and effectiveness of treatments for those with breast cancer.
- Unrelated data is the most common mistake, **for example:**



– There are many alternate variables one could look at that are more realisticly correlated

Quiz 1

(Note 1-4 were multiple choice of material covered in the notes)

5. Suppose that we have created a machine learning algorithm that predicts whether a link will be clicked with 99% sensitivity and 99% specificity. The rate the link is clicked is 1/1000 of visits to a website. If we predict the link will be clicked on a specific visit, what is the probability it will actually be clicked?

```
sens <- 0.99  #TP/(TP+FN)
spec <- 0.99  #TN/(FP+TN)
rate <- 1/1000 #sum(TP+FN), 1-r is sum(FP+TN)
#ppv = TP/(TP + FP)

#rate = TP+FN,
#rate - TP = FN
#sens * (TP+FN) = TP,
#sens * (TP + rate - TP) = TP,
#TP = sens * rate
```

```r
TP <- sens * rate

#1 - rate = FP + TN,
#1 - rate - FP = TN

#spec * (FP+TN) = TN,
#spec * (FP+1 - rate - FP) = 1 - rate - FP
#spec * (1 - rate) = 1 - rate - FP
# FP = 1 - rate - (spec* (1-rate))
# FP = (1-rate)*(1-spec)
FP <- (1-rate)*(1-spec)
ppv <- TP/(TP + FP)
ppv
```

```
## [1] 0.09016393
```

The Caret Package

Caret Package

Caret Package

- Can be installed with `install.packages("caret")`, details about the package **can be found on cran**

Functionality

- Some preprocessing (cleaning)
  - `preProcess`
- Data splitting
  - `createDataPartition`
  - `createResample`
  - `createTimeSlices`
- Training/testing functions
  - `train`
  - `predict`
- Model comparison
  - `confusionMatrix`

Machine Learning Algorithms in Base R

- Linear discriminant analysis
- Regression
- Naive Bayes
- Support vector machines
- Classification and regression trees
- Random forests
- Boosting
- etc.
- The interface for these algorithms is slightly different

obj Class	Package	predict Function Syntax
lda	MASS	predict(obj) (no options needed)
glm	stats	predict(obj, type = "response")
gbm	gbm	predict(obj, type = "response", n.trees)
mda	mda	predict(obj, type = "posterior")
rpart	rpart	predict(obj, type = "prob")
Weka	RWeka	predict(obj, type = "probability")
LogitBoost	caTools	predict(obj, type = "raw", nIter)

- The `caret` package unifies these differences

SPAM Example: Data Splitting

```
library(caret)
```

```
## Loading required package: lattice
## Loading required package: ggplot2
##
## Attaching package: 'ggplot2'
## The following object is masked from 'package:kernlab':
##
##     alpha
```

```r
library(kernlab) #For data
data(spam)
set.seed(32343)
inTrain <- createDataPartition(y = spam$type,
                               p = 0.75, #Proportion to subset
                               list = FALSE) # =F returns indecies
training <- spam[inTrain, ]
testing <- spam[-inTrain, ]
dim(spam)
```

```
## [1] 4601   58
```

```r
dim(training)
```

```
## [1] 3451   58
```

```r
dim(spam)[1]*0.75 #Showing it took 75%
```

```
## [1] 3450.75
```

```r
set.seed(32343)
modelFit <- train(type ~ ., data = training, method = "glm")
modelFit
```

```
## Generalized Linear Model
##
## 3451 samples
##   57 predictor
##    2 classes: 'nonspam', 'spam'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 3451, 3451, 3451, 3451, 3451, 3451, ...
## Resampling results:
##
##   Accuracy   Kappa
##   0.9159559  0.8227903
```

- `Resampling: Bootstrapped (25 reps)` indicates that it used the Bootstrap method, with 25 replicates. It corrects for the error that can occur when using the bootstrap method

```r
modelFit$finalModel
```

```
##
## Call:  NULL
##
## Coefficients:
##       (Intercept)               make            address                all
##         -1.473e+00         -2.692e-01         -1.386e-01          8.483e-02
##             num3d                our               over             remove
##          3.634e+00          5.485e-01          6.465e-01          2.539e+00
```

```
##          internet             order              mail           receive
##         7.626e-01         4.876e-01         1.035e-01        -4.715e-01
##              will            people            report         addresses
##        -1.121e-01        -1.117e-01         6.115e-02         2.411e+00
##              free          business             email               you
##         9.280e-01         7.069e-01         8.624e-02         9.791e-02
##            credit              your              font             num000
##         9.980e-01         2.176e-01         1.327e-01         2.126e+00
##             money                hp               hpl            george
##         7.839e-01        -2.013e+00        -8.222e-01        -8.944e+00
##            num650               lab              labs            telnet
##         9.072e-02        -2.061e+00         8.513e-03         5.930e-01
##            num857              data            num415             num85
##         1.061e+00        -8.148e-01        -1.355e+01        -2.857e+00
##        technology           num1999             parts                pm
##         1.236e+00         6.600e-02        -5.405e-01        -1.218e+00
##            direct                cs           meeting          original
##        -3.825e-01        -4.398e+01        -3.209e+00        -1.245e+00
##           project                re               edu             table
##        -1.349e+00        -9.048e-01        -1.822e+00        -2.303e+00
##        conference      charSemicolon   charRoundbracket  charSquarebracket
##        -2.908e+00        -1.316e+00        -7.632e-02        -6.030e-01
##   charExclamation         charDollar          charHash         capitalAve
##         3.538e-01          5.103e+00         2.956e+00        -9.560e-03
##       capitalLong       capitalTotal
##         9.809e-03          7.312e-04
##
## Degrees of Freedom: 3450 Total (i.e. Null);  3393 Residual
## Null Deviance:       4628
## Residual Deviance: 1382  AIC: 1498
```

- This shows all how all the variables are weighted

SPAM Example: Prediction

```
predictions <- predict(modelFit, newdata = testing)
predictions[1:30]
```

```
##  [1] spam    spam    spam    spam    spam    nonspam spam    spam    spam
## [10] spam    spam    spam    spam    spam    spam    spam    spam    spam
## [19] spam    spam    spam    nonspam spam    nonspam spam    spam    spam
## [28] spam    spam    spam
## Levels: nonspam spam
```

SPAM Example: Confusion Matrix

```
confusionMatrix(predictions, testing$type)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction nonspam spam
##    nonspam     672   52
##    spam         25  401
##
##                Accuracy : 0.933
##                  95% CI : (0.917, 0.9468)
##     No Information Rate : 0.6061
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.8583
##
##  Mcnemar's Test P-Value : 0.003047
##
##             Sensitivity : 0.9641
##             Specificity : 0.8852
##          Pos Pred Value : 0.9282
##          Neg Pred Value : 0.9413
##              Prevalence : 0.6061
##          Detection Rate : 0.5843
##    Detection Prevalence : 0.6296
##       Balanced Accuracy : 0.9247
##
##        'Positive' Class : nonspam
##
```

- First gives a table of the predicted vs. true value
- Gives summary statistics
  - Accuracy & 95% CI for the accuracy
  - No Information Rate is the average loss, L, of f over all combinations of y_i and x_j, expressed with the formula: $\frac{1}{n^2}\sum_{i=1}^{n}\sum_{j=1}^{n} L(y_i, f(x_j))$
  - The **Mcnemar's Test P-Value** has a null hypothesis that the error rates are equivelant

Further Information

- Caret tutorials:
  - **PDF caret tutorial**
  - **cran vignette PDF**
  - **A paper introducing the caret package**

Data Slicing

SPAM Example: Data Splitting

```
library(caret)
library(kernlab); data(spam)
inTrain <- createDataPartition(y = spam$type,
                               p = 0.75, list = FALSE)
training <- spam[inTrain, ]
testing <- spam[-inTrain, ]
```

SPAM Example: K-fold

```
set.seed(32323)
folds <- createFolds(y=spam$type, k = 10,
                     list = TRUE, returnTrain = TRUE) #Returns sample positions
sapply(folds, length)
```

```
## Fold01 Fold02 Fold03 Fold04 Fold05 Fold06 Fold07 Fold08 Fold09 Fold10
##   4141   4140   4141   4142   4140   4142   4141   4141   4140   4141
```

```
folds[[1]][1:10]
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

```
#You can also have it return the test set
set.seed(32323)
folds <- createFolds(y=spam$type, k = 10,
                     list = TRUE, returnTrain = FALSE) #Returns sample positions
sapply(folds, length)
```

```
## Fold01 Fold02 Fold03 Fold04 Fold05 Fold06 Fold07 Fold08 Fold09 Fold10
##    460    461    460    459    461    459    460    460    461    460
```

```
folds[[1]][1:10]
```

```
##  [1] 24 27 32 40 41 43 55 58 63 68
```

SPAM Example: Resampling

```
set.seed(32323)
folds <- createResample(y = spam$type, times = 10,
                        list = TRUE)
sapply(folds, length)
```

```
## Resample01 Resample02 Resample03 Resample04 Resample05 Resample06 Resample07
##       4601       4601       4601       4601       4601       4601       4601
## Resample08 Resample09 Resample10
##       4601       4601       4601
```

```
folds[[1]][1:10] #Contains some resampled values
```

```
##  [1]  1  2  3  3  3  5  5  7  8 12
```

SPAM Example: Time Slices

```
set.seed(32323)
tme <- 1:1000
folds <- createTimeSlices(y = tme,
                          initialWindow = 20, #consecutive ... training set
                          horizon = 10)#Consecutive values in each test set
names(folds)
```

```
## [1] "train" "test"
```

```
folds$train[[1]]
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
```

```
folds$test[[1]]
```

```
##  [1] 21 22 23 24 25 26 27 28 29 30
```

Training Options

```
## Still using SPAM set
library(caret)
library(kernlab); data(spam)
inTrain <- createDataPartition(y = spam$type, p = 0.75, list = FALSE)
training <- spam[inTrain, ]
testing <- spam[-inTrain, ]
modelFit <- train(type ~., data = training, method = "glm")
```

Train Options

```
args(caret:::train.default)
```

```
## function (x, y, method = "rf", preProcess = NULL, ..., weights = NULL,
##     metric = ifelse(is.factor(y), "Accuracy", "RMSE"), maximize = ifelse(metric %in%
##         c("RMSE", "logLoss", "MAE"), FALSE, TRUE), trControl = trainControl(),
##     tuneGrid = NULL, tuneLength = ifelse(trControl$method ==
##         "none", 1, 3))
## NULL
```

- One can change ...
  - `preProcess` to change preprocessing options (covered later)
  - `weights` to assign weights to the variables (Useful for unbalanced training set)

- `metric` to change what is measured, default is that Accuracy is measured for catagorigal variables and RMSE otherwise. Below are some of the other options:
  - RMSE = Root mean squared error
  - RSquared = R^2 from regression models
  - Accuracy = Fraction correct
  - Kappa = A measure of **concordance**
- `trControl` = Calls to `trainControl` function which has more of it's own settings:

```
args(trainControl)
```

```
## function (method = "boot", number = ifelse(grepl("cv", method),
##     10, 25), repeats = ifelse(grepl("[d_]cv$", method), 1, NA),
##     p = 0.75, search = "grid", initialWindow = NULL, horizon = 1,
##     fixedWindow = TRUE, skip = 0, verboseIter = FALSE, returnData = TRUE,
##     returnResamp = "final", savePredictions = FALSE, classProbs = FALSE,
##     summaryFunction = defaultSummary, selectionFunction = "best",
##     preProcOptions = list(thresh = 0.95, ICAcomp = 3, k = 5,
##         freqCut = 95/5, uniqueCut = 10, cutoff = 0.9), sampling = NULL,
##     index = NULL, indexOut = NULL, indexFinal = NULL, timingSamps = 0,
##     predictionBounds = rep(FALSE, 2), seeds = NA, adaptive = list(min = 5,
##         alpha = 0.05, method = "gls", complete = TRUE), trim = FALSE,
##     allowParallel = TRUE)
## NULL
```

```
- ‘method‘ will determine how it samples data, along with the ‘number‘ of times and how many t
- ‘initialWindow‘ and ‘horizon‘ is for time based data
- ‘savePredictions‘ if true will return all the predictors of each model
- ‘summaryFunction‘ will determine the kind of summary returned
- ‘preProcOptions‘ (Preprocessing options)
- ‘seeds‘ is available to set seeds for all the diffrent resampling layers, helpful when runni
```

trainControl Resampling

- `method`
  - *boot* = bootstrapping
  - *boot632* = bootstrapping with adjustment
  - *cv* = cross validation
  - *repeatedcv* = repeated cross validation
  - *LOOCV* = leave one out cross validation
- `number`

  - For boot/cross validation
  - Number of subsamples to take

- `repeats`
  - Number of times to repeat subsampling
  - If value is big this can slow things down
- **More info on model training and tuning**

Plotting Predictors

```
#The example in this lesson will be using wages data from the ISLR package
library(ISLR); data(Wage)
library(tidyverse)
library(caret)
summary(Wage)
```

```
##       year           age                     maritl           race
##  Min.   :2003   Min.   :18.00   1. Never Married: 648   1. White:2480
##  1st Qu.:2004   1st Qu.:33.75   2. Married      :2074   2. Black: 293
##  Median :2006   Median :42.00   3. Widowed      :  19   3. Asian: 190
##  Mean   :2006   Mean   :42.41   4. Divorced     : 204   4. Other:  37
##  3rd Qu.:2008   3rd Qu.:51.00   5. Separated    :  55
##  Max.   :2009   Max.   :80.00
##
##                education                     region               jobclass
##  1. < HS Grad      :268   2. Middle Atlantic   :3000   1. Industrial :1544
##  2. HS Grad        :971   1. New England       :   0   2. Information:1456
##  3. Some College   :650   3. East North Central:   0
##  4. College Grad   :685   4. West North Central:   0
##  5. Advanced Degree:426   5. South Atlantic    :   0
##                           6. East South Central:   0
##                           (Other)              :   0
##             health      health_ins      logwage           wage
##  1. <=Good     : 858   1. Yes:2083   Min.   :3.000   Min.   : 20.09
##  2. >=Very Good:2142   2. No : 917   1st Qu.:4.447   1st Qu.: 85.38
##                                      Median :4.653   Median :104.92
##                                      Mean   :4.654   Mean   :111.70
##                                      3rd Qu.:4.857   3rd Qu.:128.68
##                                      Max.   :5.763   Max.   :318.34
##
```

- From this we can see the data is all from Males in the Middle Atlantic region

```
## Creating training/test sets
set.seed(1618033)
inTrain <- createDataPartition(y = Wage$wage,
```

```
                                  p = 0.7, list = FALSE)
training <- Wage[inTrain, ]
testing <- Wage[-inTrain, ]
dim(training); dim(testing)
```

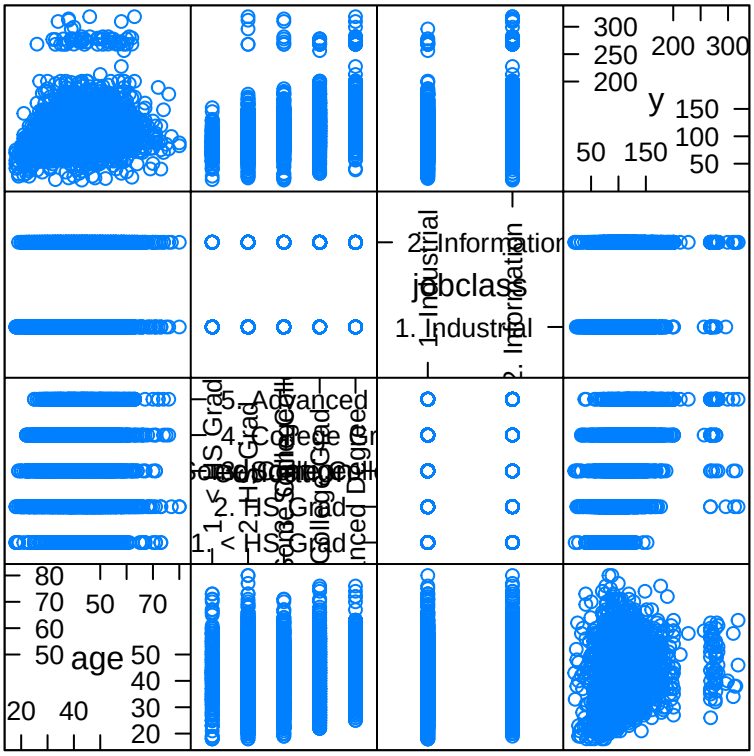
```
## [1] 2102   11
```

```
## [1] 898  11
```

Looking at the Data

featurePlot

```
featurePlot(x = training[, c("age", "education", "jobclass")],
            y = training$wage,
            plot = "pairs")
```



Scatter Plot Matrix

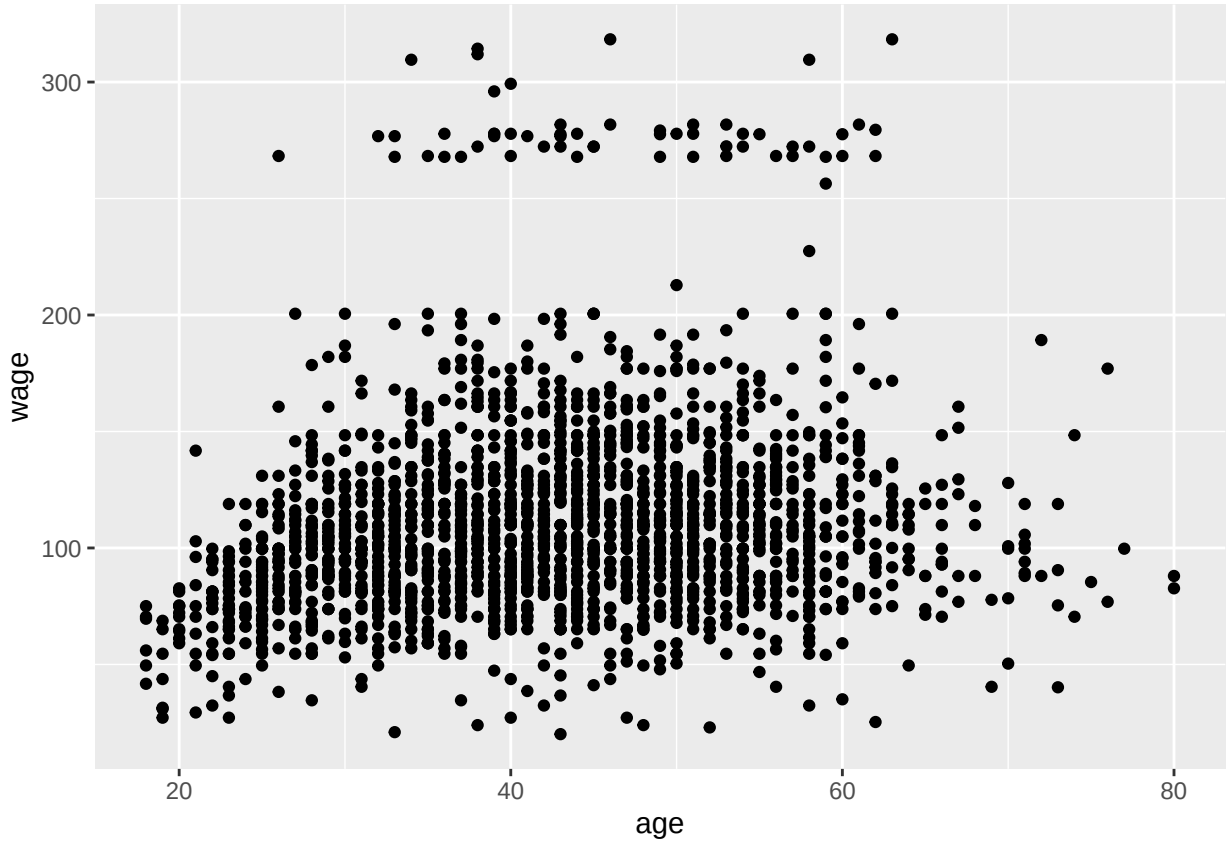
- When looking at this plot one is looking for trends in the data, for example the second column of the first row shows a semi-positive correlation to the X, when deciphering the mess of words one can see this is our y, `wage`, against `education`; indicating higher education might correlate to a higher wage.

- Two catagorical variables against eachother are hard to decipher meaning from as they largely overlap.

qplot

- Quick plots in style of `ggplot`

```
qplot(age, wage, data = training)
```

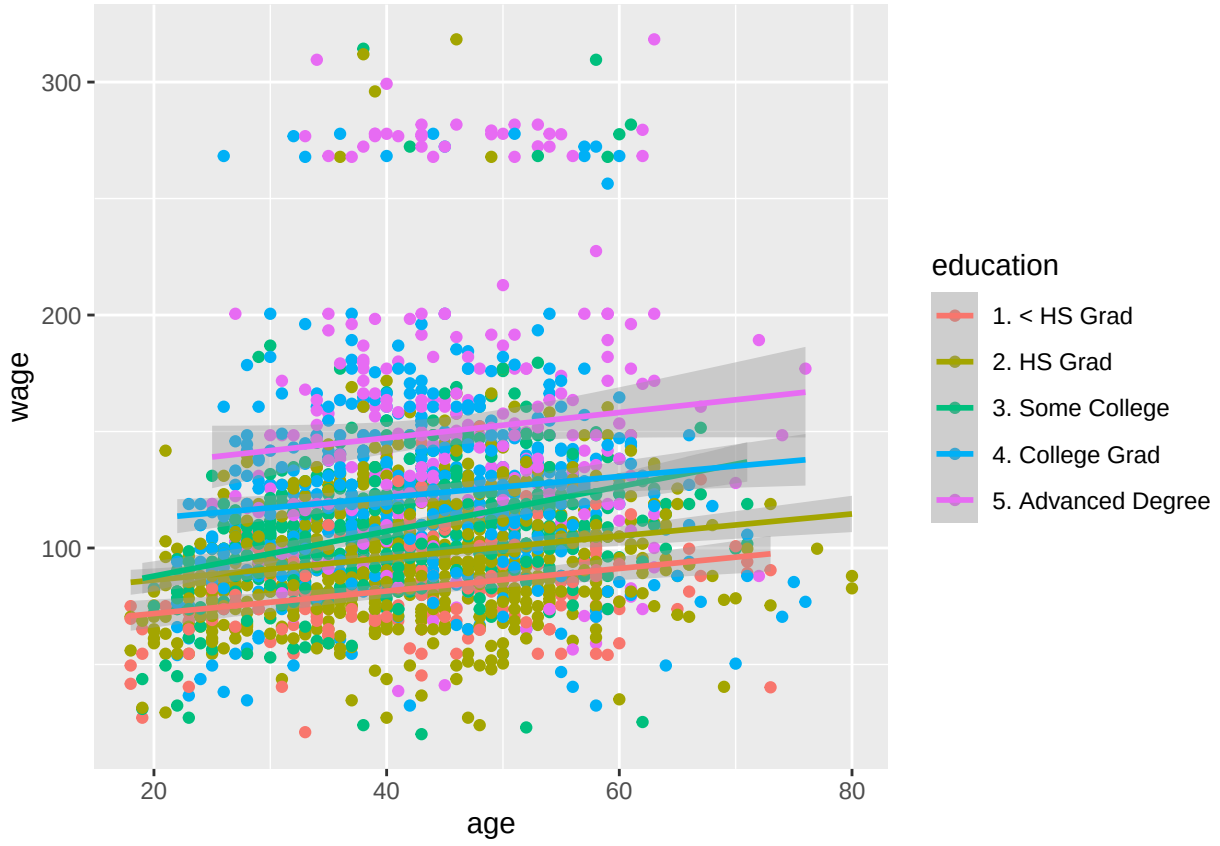


- The odd subset of wages that are away from the others may be cause for some concern, as such we'd want to investigate this before making the model

```
qplot(age, wage, colour = jobclass, data = training)
```

- You can also add regression smoothers to investigate differences more

```
plot <- qplot(age, wage, colour = education, data = training)
plot + geom_smooth(method = 'lm', formula = y ~ x)
```



Using cut2 to make factors

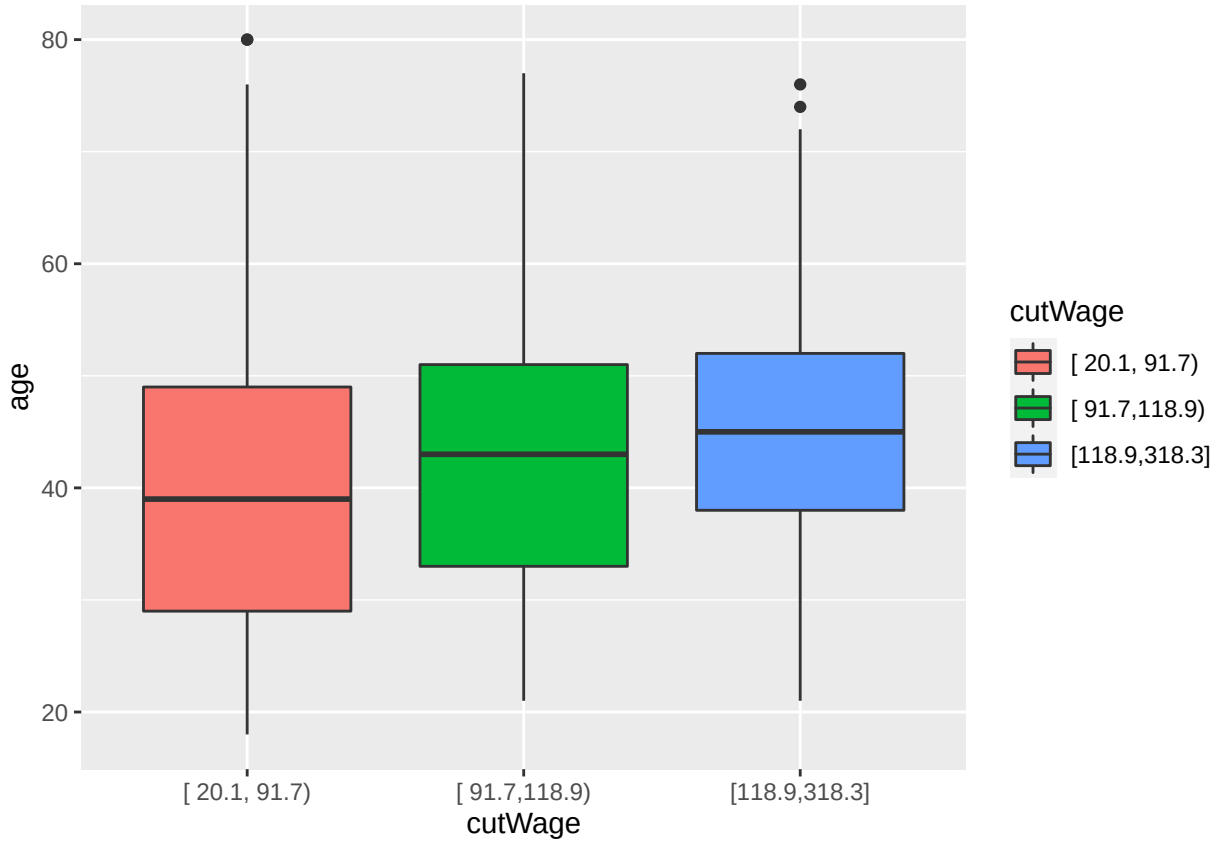
```
library(Hmisc)
```

```
## Loading required package: survival
##
## Attaching package: 'survival'
## The following object is masked from 'package:caret':
##
##     cluster
## Loading required package: Formula
##
## Attaching package: 'Hmisc'
## The following objects are masked from 'package:dplyr':
##
##     src, summarize
## The following objects are masked from 'package:base':
##
##     format.pval, units
```

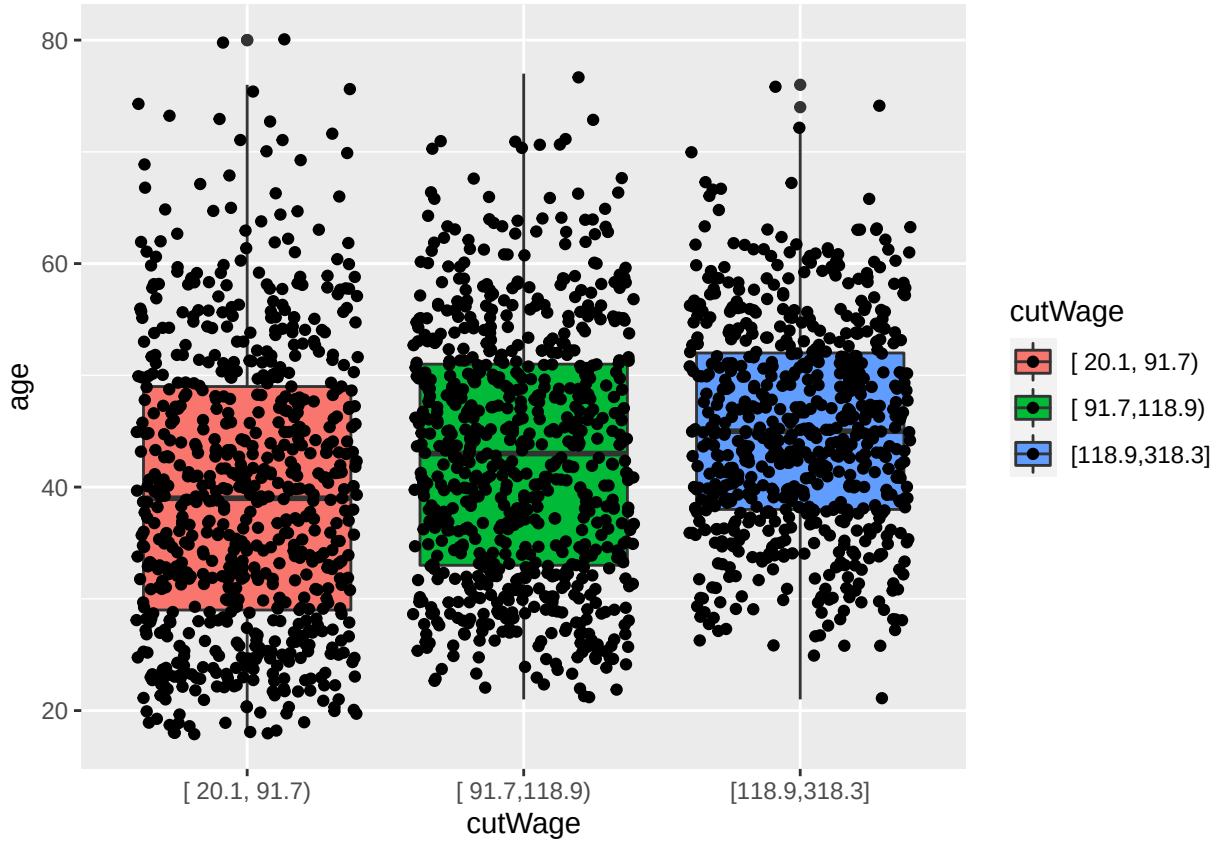
```
cutWage <- cut2(training$wage, g = 3)
table(cutWage)
```

```
## cutWage
## [ 20.1, 91.7) [ 91.7,118.9) [118.9,318.3]
##           704           727           671
```

```
bplot1 <- qplot(cutWage, age, data = training, fill = cutWage,
                geom = c("boxplot"))
bplot1
```




```
#Add points overlayed on boxplot
bplot2 <- qplot(cutWage, age, data = training, fill = cutWage,
                geom = c("boxplot", "jitter"))
bplot2
```



- Many points for each plot indicates the boxplots are well representing the data, if there were only a few then it would suggest the boxplots are not as representative

Tables

```
t1 <- table(cutWage, training$jobclass)
t1
```

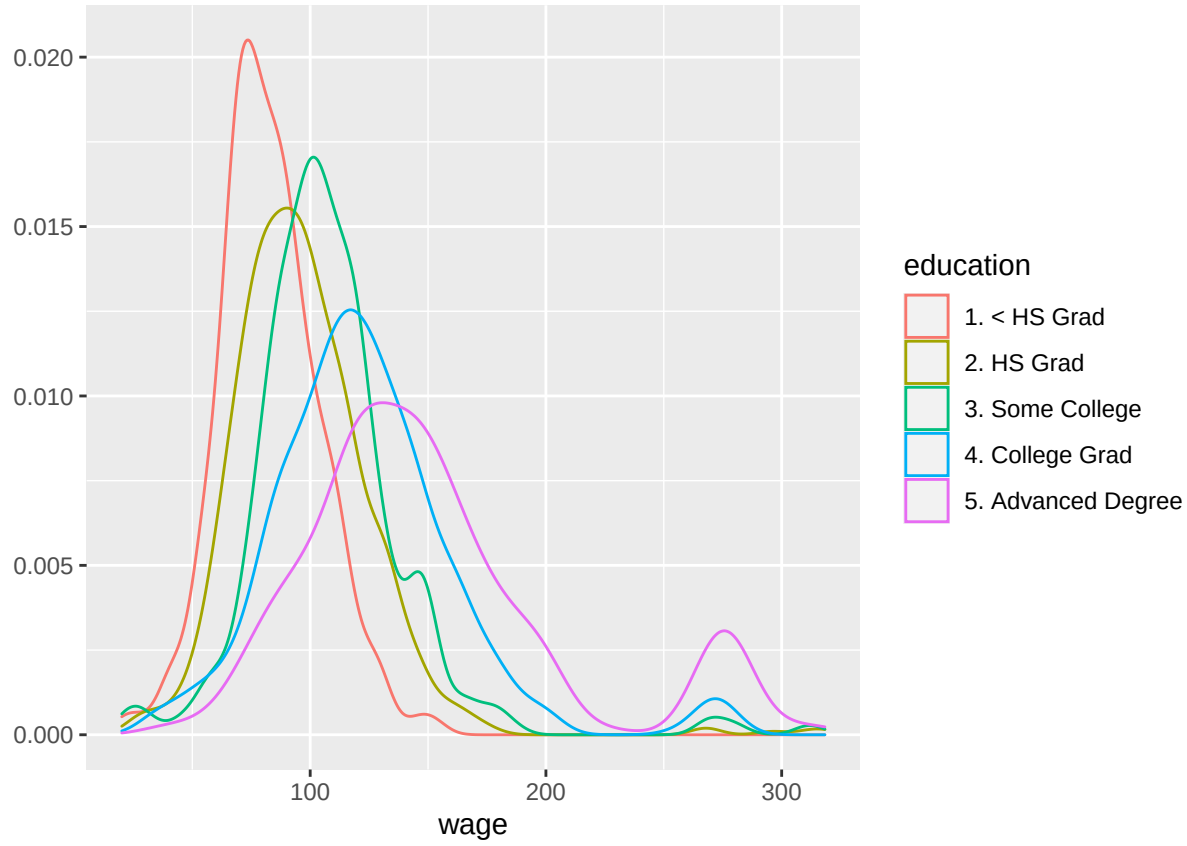
```
##
## cutWage          1. Industrial 2. Information
##   [ 20.1, 91.7)            439            265
##   [ 91.7,118.9)            380            347
##   [118.9,318.3]            266            405
```

```
prop.table(t1, 1) #Shows proportions, 1 for by row
```

```
##
## cutWage          1. Industrial 2. Information
##   [ 20.1, 91.7)     0.6235795      0.3764205
##   [ 91.7,118.9)     0.5226960      0.4773040
##   [118.9,318.3]     0.3964232      0.6035768
```

Density Plots

```
qplot(wage, colour = education, data = training, geom = "density")
```



Notes and Further Reading

- Make your plots only in the training set
  - Don't use the test set for exploration!
- Things one should be looking for
  - Imbalance in outcomes/predictors
  - Outliers
  - Groups of points not explained by a predictor
  - Skewed variables
- **ggplot2 tutorial**
- **caret visualizations**

Reminder to Commit (05), Delete this line *AFTER* Committing

Preprocessing

Basic Preprocessing

Covariate Creation

Preprocessing with Principal Components Analysis (PCA)

Reminder to Commit (06), Delete this line *AFTER* Committing

Predicting

Predicting with Regression

Predicting with Regression Multiple Covariates

Reminder to Commit (07), Delete this line *AFTER* Committing

Quiz 2

Reminder to Commit (Q2), Delete this line *AFTER* Committing

Predicting with Trees, Random Forests, & Model Based Predictions

Trees

Predicting with Trees

Bagging

Reminder to Commit (08), Delete this line *AFTER* Committing

Random Forests

Random Forests

Boosting

Reminder to Commit (09), Delete this line *AFTER* Committing

Model Baded Predictions

Model Based Predictions

Reminder to Commit (10), Delete this line *AFTER* Committing

Quiz 3

Reminder to Commit (Q3), Delete this line *AFTER* Committing

Regularized Regression and Combining Predictors

Regularized Regression

Combining Predictors

Reminder to Commit (11), Delete this line *AFTER* Committing

Forecasting

Unsupervised Prediction

Reminder to Commit (12), Delete this line *AFTER* Committing

Quiz 4

Reminder to Commit (Q4), Delete this line *AFTER* Committing

Course Project

Reminder to Commit (P1), Delete this line *BEFORE* Committing