Practical ML - Week 4

AF

03/04/2021

Week 4: Regularised regression and combining predictors

Week 4.1: Regularised regression

Basic idea

1. Fit a **regression model**
2. **Penalize** (or shrink) **large coefficients** corresponding to some of the predictive variables

Pros:

- Can help with the **bias/variance tradeoff** (e.g. if certain predictors are highly correlated with each other, you might not want to include them both in the regression model as they will have very high variance, although this might slightly increase the bias)
- Can help with **model selection** (e.g. using the **LASSO** technique)

Cons:

- May be **computationally demanding** on large data sets
- Does not perform as well as random forests and boosting

A motivating example

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \epsilon$$

where X_1 and X_2 are nearly perfectly correlated (co-linear, i.e. almost exactly the same variable). You can approximate this model by:

$$Y = \beta_0 + (\beta_1 + \beta_2) X_1 + \epsilon$$

The result is:

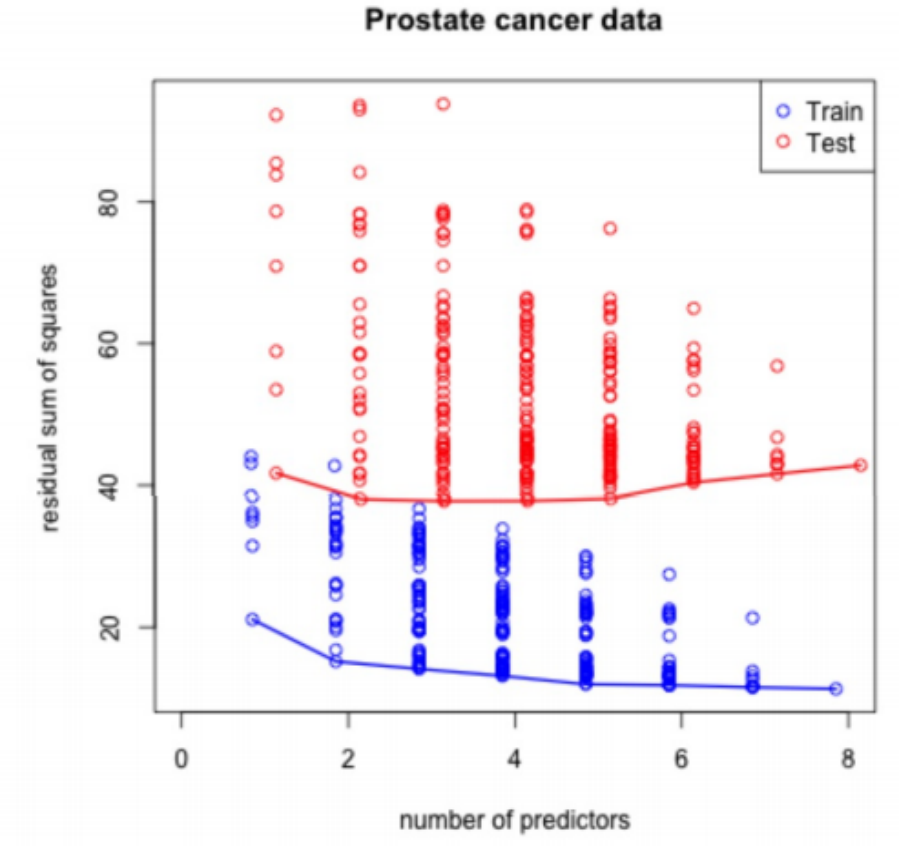
- You will get a good estimate of Y
- The estimate (of Y) will be **biased** (because we chose to leave one of the predictors out)
- We may **reduce variance** in the estimate

```
prostate<-read.table("http://www-stat.stanford.edu/~tibs/ElemStatLearn/datasets/prostate.data", sep="\t",h
ead=T,row.names=1)
str(prostate)
```

```
## 'data.frame':    97 obs. of  10 variables:
##  $ lcavol : num  -0.58 -0.994 -0.511 -1.204 0.751 ...
##  $ lweight: num  2.77 3.32 2.69 3.28 3.43 ...
##  $ age    : int  50 58 74 58 62 50 64 58 47 63 ...
##  $ lbph   : num  -1.39 -1.39 -1.39 -1.39 -1.39 ...
##  $ svi    : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ lcp    : num  -1.39 -1.39 -1.39 -1.39 -1.39 ...
##  $ gleason: int  6 6 7 6 6 6 6 6 6 6 ...
##  $ pgg45  : int  0 0 20 0 0 0 0 0 0 0 ...
##  $ lpsa   : num  -0.431 -0.163 -0.163 -0.163 0.372 ...
##  $ train  : logi  TRUE TRUE TRUE TRUE TRUE TRUE ...
```

- We might want to be able to make a prediction about prostate cancer, based on a large number of predictors in the dataset

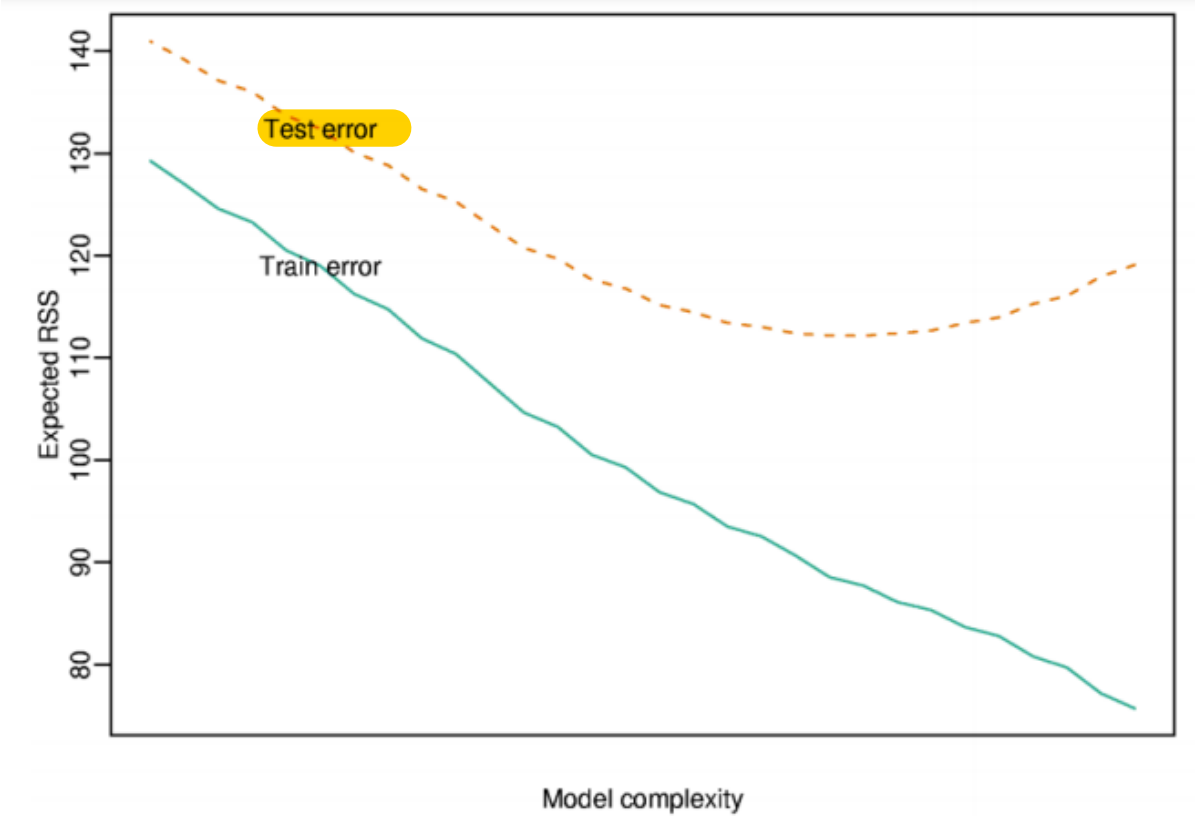
Subset selection



Code here (http://users.umiacs.umd.edu/~hcorrada/PracticalML/src/selection.R)

- Suppose we predict with all possible combinations of predictor variables, so we build one regression model for every possible combinations of predictors
- As the number of predictors increases, the training set error goes down (this will always happen)
- The testing error, instead, goes down up to a certain point as the number of predictors in the model increases. Then it reaches a plateau and start instead to increase! This increase happens because we are **overfitting** the data in the training set and we may not want to include so many predictors in the model.

Most common pattern



- This is an incredibly common pattern! (RSS = Residual Sum of Squares of the predicted error)
- In the **training set**, the higher the complexity of the model, the lower the error (almost a monotone relationship), always.
- In the **testing set**, increasing the model complexity, the error decreases, then reaches a plateau and then starts to increase (this means the model has become too complex and overfits the data)

Model selection approach: split samples

- No method better when data/computation time permits it
- Approach
    1. Divide data into training/test/validation
    2. Treat validation as test data, train all competing models on the train data and pick the best one on validation.
    3. To appropriately assess performance (error rate) on new data apply to test set
    4. You may re-split and reperform steps 1-3
- Two common problems
    - Limited data
    - Computational complexity

http://www.biostat.jhsph.edu/~ririzarr/Teaching/649/ (http://www.biostat.jhsph.edu/~ririzarr/Teaching/649/)

Decomposing expected prediction error

Assume $Y_i = f(X_i) + \epsilon_i$ (ϵ_i is the error term)

$EPE(\lambda) = E\left[\{Y - \hat{f}_\lambda(X)\}^2\right]$ (EPE = Expected Prediction Error)

Suppose $\hat{f}_\lambda$ is the estimate from the training data (using a particular set of tuning parameter λ) and look at a new data point $X = x^*$

$$E\left[\{Y - \hat{f}_\lambda(x^*)\}^2\right] = \sigma^2 + \{E[\hat{f}_\lambda(x^*)] - f(x^*)\}^2 + var[\hat{f}_\lambda(x_0)]$$

Expected error between our outcome and its prediction = **Irreducible error** + **Bias**^{2} + **Variance**

- You can trade-off bias and variance –> this is the idea behind what regularised regression does

Another issue for high-dimensional data

```
small<-prostate[1:5,]
dim(small) # 5 samples, 9 features
```

```
## [1]  5 10
```

```
lm(lpsa ~ ., data=small)
```

```
##
## Call:
## lm(formula = lpsa ~ ., data = small)
##
## Coefficients:
## (Intercept)       lcavol      lweight          age         lbph          svi
##     9.60615      0.13901     -0.79142      0.09516           NA           NA
##         lcp      gleason        pgg45    trainTRUE
##          NA     -2.08710           NA           NA
```

- Some of the predictor variables would become useless because you have more predictors that you have samples (you have designed matrices that cannot be inverted)

Hard thresholding

- This is one approach to deal with the issue above
- Model $Y = f(X) + \epsilon$
- Set $\hat{f}_\lambda(x) = x'\beta$
- Constrain only λ coefficients to be nonzero.
- Selection problem is after choosing λ figure out which $p - \lambda$ coefficients to make nonzero

Regularization for regression

If the β_j's are **unconstrained**: * They can explode (if very high correlated variables are used for predictions) * And hence are susceptible to very high variance

To control variance, we might regularize/shrink the coefficients.

$$PRSS(\beta) = \sum_{j=1}^{n}(Y_j - \sum_{i=1}^{m}\beta_{1i}X_{ij})^2 + P(\lambda;\beta)$$

where $PRSS$ is a penalized form of the sum of squares ($P(\lambda;\beta)$ means that if the βs are too big, they will be shrinked down). Things that are commonly looked for

- Penalty reduces **complexity**
- Penalty reduces ***variance**
- Penalty **respects structure** of the problem

Some approaches for regularised regression will be discussed in the following sections.

Ridge regression

Solve:

$$\sum_{i=1}^{N}\left(y_i-\beta_0+\sum_{j=1}^{p}x_{ij}\beta_j\right)^2+\lambda\sum_{j=1}^{p}\beta_j^2$$

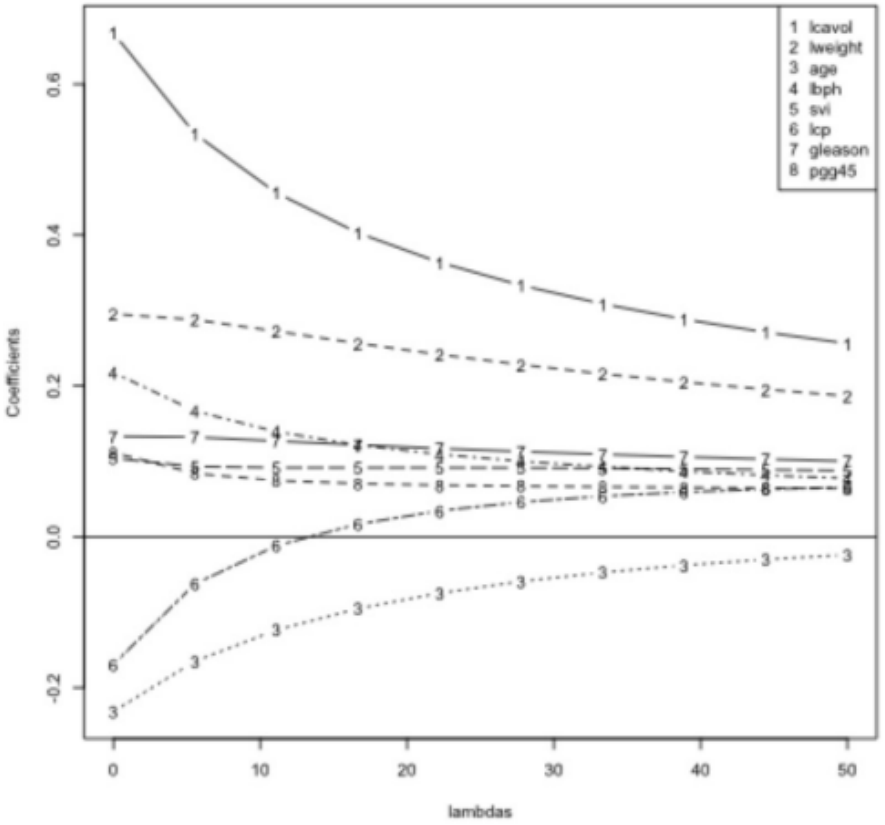
$ {i=1}^N (y_i - 0 + {j=1}^p x{ij}_j)^2 $ represents the squared distance between the outcome value and its prediction, while $\lambda\sum_{j=1}^{p}\beta_j^2$ would penalise the model whether the βs terms get too big.

equivalent to solving

$\sum_{i=1}^{N}\left(y_i-\beta_0+\sum_{j=1}^{p}x_{ij}\beta_j\right)^2$ subject to $\sum_{j=1}^{p}\beta_j^2\leq s$ where s is inversely proportional to λ

Inclusion of λ makes the problem non-singular even if X^TX is not invertible (i.e.when we have more predictors than sample observations a ridge regression model can still be fit).

Ridge coefficient paths



- As lambda increases, we penalise the larger βs (coefficients) more and more, so they all tend to get closer and closer to 0

Tuning parameter λ

- λ controls the size of the coefficients
- λ controls the amount of regularization
- As $\lambda\rightarrow 0$ we obtain the least square solution (as you obtain from a standard linear model)
- As $\lambda\rightarrow\infty$ we have $\hat{\beta}_{\lambda=\infty}^{ridge}=0$ (i.e. all coefficients tend to go towards 0)
- Choosing the optimal tuning λ parameter (for the bias/variance trade-off) can be done with cross-validation or other techniques

Lasso

It is a similar approach to ridge regression for regularisation, and it uses a slightly different form of penalty.

$\sum_{i=1}^{N} \left(y_i - \beta_0 + \sum_{j=1}^{p} x_{ij}\beta_j \right)^2$ subject to $\sum_{j=1}^{p} |\beta_j| \leq s$

also has a **lagrangian form**

$$\sum_{i=1}^{N} \left(y_i - \beta_0 + \sum_{j=1}^{p} x_{ij}\beta_j \right)^2 + \lambda \sum_{j=1}^{p} |\beta_j|$$

For **orthonormal design matrices** (not the norm!) this has a closed form solution

$$\hat{\beta}_j = sign(\hat{\beta}_j^0)(|\hat{\beta}_j^0| - \gamma)^+$$

but not in general.

- The lasso approach shrinks all the coeffiecients and set some of them exactly to 0, performing model selection
- In `caret` methods to fit different kinds of penalised regression models are:
    - `ridge`
    - `lasso`
    - `relaxo`

Week 4.2: Combining predictors (or ensembling methods)

Key ideas

- You can **combine classifiers** by **averaging/voting** (including very different classifiers)
- Combining classifiers **improves accuracy**
- Combining classifiers **reduces interpretability**
- **Boosting**, **bagging**, and **random forests** are variants on this theme (although they are all examples of the same classifier types combined together)

Basic intuition - majority vote

Suppose we have 5 completely independent classifiers

If accuracy is 70% for each: * $10 \times (0.7)^3(0.3)^2 + 5 \times (0.7)^4(0.3)^2 + (0.7)^5$ (majority vote: 3 out of 5 + 4 out of 5 + 5 out of 5 classifiers) * 83.7% majority vote accuracy

With 101 independent classifiers * 99.9% majority vote accuracy

Approaches for combining classifiers

1. **Bagging**, **boosting**, **random forests**
    - Usually **combine similar classifiers**
2. **Combining different classifiers**
    - **Model stacking**
    - **Model ensembling**

Example with Wage data

Create training, test and validation sets

```
library(ISLR); data(Wage); library(ggplot2); library(caret);
Wage <- subset(Wage,select=-c(logwage))

# Create a building data set and validation set
inBuild<-createDataPartition(y=Wage$wage, p=0.7, list = F)
validation<-Wage[-inBuild,]; buildData<-Wage[inBuild,]

inTrain<-createDataPartition(y=buildData$wage, p=0.7, list = F)
testing<-buildData[-inTrain,]; training<-buildData[inTrain,]
dim(validation);dim(testing);dim(training)
```

```
## [1] 898  10
```

```
## [1] 628  10
```

```
## [1] 1474   10
```

Build two different models

```
mod1<-train(data = training, wage ~ ., method = 'glm') ## linear model
mod2<-train(data = training, wage ~ ., method = 'rf', trControl = trainControl(method='cv'), number = 3) #
# random forest
mod1
```

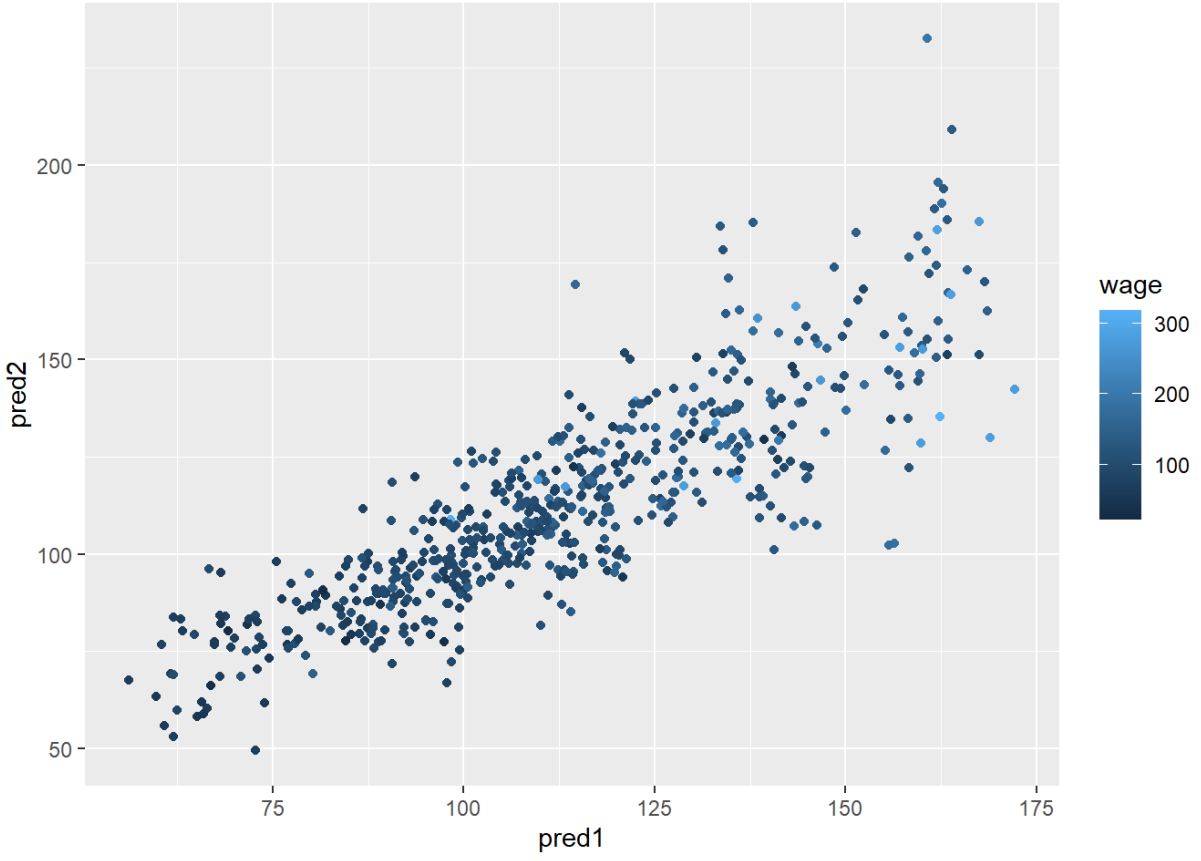
```
## Generalized Linear Model
##
## 1474 samples
##    9 predictor
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 1474, 1474, 1474, 1474, 1474, 1474, ...
## Resampling results:
##
##   RMSE      Rsquared   MAE
##   35.23594  0.3269348  23.98763
```

```
mod2
```

```
## Random Forest
##
## 1474 samples
##    9 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 1327, 1326, 1329, 1326, 1325, 1327, ...
## Resampling results across tuning parameters:
##
##   mtry  RMSE      Rsquared   MAE
##    2    37.43421  0.3589230  25.58041
##   13    35.60048  0.3241509  24.89927
##   24    36.88835  0.2968863  26.03239
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was mtry = 13.
```

Predict on the testing set

```
pred1<-predict(mod1,testing)
pred2<-predict(mod2,testing)
qplot(pred1,pred2, colour = wage, data=testing)
```



Fit a model that combines predictors

```
preDF<-data.frame(pred1,pred2,wage=testing$wage)
combModFit<- train(data=preDF, wage ~ ., method = 'gam') ## generalised additive model
combPred<- predict(combModFit,preDF)
combModFit
```

```
## Generalized Additive Model using Splines
##
## 628 samples
##   2 predictor
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 628, 628, 628, 628, 628, 628, ...
## Resampling results across tuning parameters:
##
##   select  RMSE      Rsquared   MAE
##   FALSE   35.97038  0.3002851  23.98030
##    TRUE   35.90717  0.3016378  23.96019
##
## Tuning parameter 'method' was held constant at a value of GCV.Cp
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were select = TRUE and method = GCV.Cp.
```

Testing errors

```
sqrt(sum((pred1-testing$wage)^2))
```

```
## [1] 882.3763
```

```
sqrt(sum((pred2-testing$wage)^2))
```

```
## [1] 933.9148
```

```
sqrt(sum((combPred-testing$wage)^2))
```

```
## [1] 875.7089
```

Predict on validation data set and evaluate the models

Since the test set was used to try and blend the two models together, it is not a good representation of the out of sample error –> I use the validation set to create predictions from the two models and then use them to predict the validation set outcomes using the model already created using the testing set (`combModFit`):

```
pred1V <- predict(mod1,validation); pred2V <- predict(mod2,validation)
predVDF <- data.frame(pred1=pred1V,pred2=pred2V)
combPredV <- predict(combModFit,predVDF)
sqrt(sum((pred1V-validation$wage)^2))
```

```
## [1] 972.1637
```

```
sqrt(sum((pred2V-validation$wage)^2))
```

```
## [1] 1014.249
```

```
sqrt(sum((combPredV-validation$wage)^2))
```

```
## [1] 971.7451
```

Notes and further resources

- Even simple blending can be useful
- Typical model for binary/multiclass data
  - Build an odd number of models
  - Predict with each model
  - Predict the class by majority vote
- This can get dramatically more complicated
  - Simple blending in caret: caretEnsemble (https://github.com/zachmayer/caretEnsemble) (use at your own risk!)
  - Wikipedia ensemble learning (http://en.wikipedia.org/wiki/Ensemble_learning)
- When doing model ensembling it is is important to keep in mind that it can lead to **increase in computational complexity** (tradeoff accuracy vs scalability)

Week 4.3: Forecasting

Forecasting is a very specific kind of prediction problem, typically applied to **time-series data**. This data has some very specific kind of dependent structure and additional challenges that must be taken into account when performing prediction.

What is different?

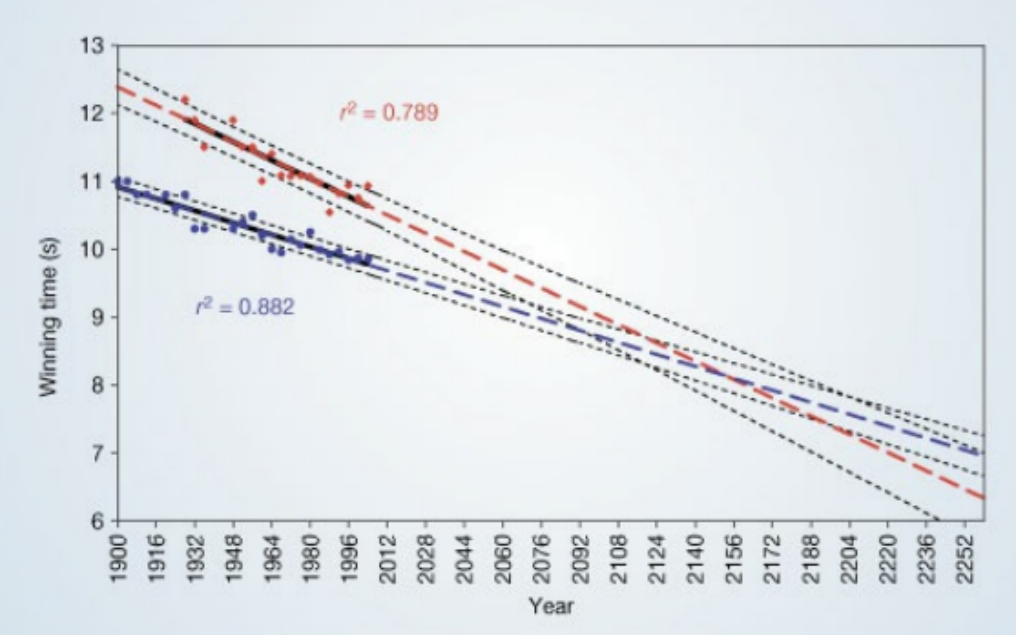
- Data are **dependent over time** (prediction is more challenging in comparison to independent samples)
- Specific **pattern types**
  - **Trends** - long term increase or decrease
  - **Seasonal patterns** - patterns related to time of week, month, year, etc.
  - **Cycles** - patterns that rise and fall periodically
- **Subsampling** into training/test is **more complicated** (you cannot randomly assign samples into training and testing sets)
- Similar issues arise in **spatial data**
  - Dependency between nearby observations
  - Location specific effects (they need to be modelled when doing predictions)
- Typically goal is to predict one or more observations into the future.
- All standard predictions can be used (with caution!)

Beware spurious correlations!

Time series for different events can often be correlated one with the other for reasons that do not make them good for predicting one from the other (the same happens also with geospatial data and spuriously correlated heatmaps).

Beware extrapolation!

Example of winning times in Olympic race games: paper authors extrapolated current trends of race times and predicted much shorter times in the future (up to negative times in the more distant future)



Forecasting example using Google data

```
#install.packages("quantmod")
library(quantmod)
from.dat <- as.Date("01/01/08", format="%m/%d/%y")
to.dat <- as.Date("12/31/13", format="%m/%d/%y")
##'getSymbols.google' is defunct.
## Google Finance stopped providing data in March, 2018.
## Using src="yahoo" as an alternative
getSymbols("GOOG", src="yahoo", from = from.dat, to = to.dat)
```

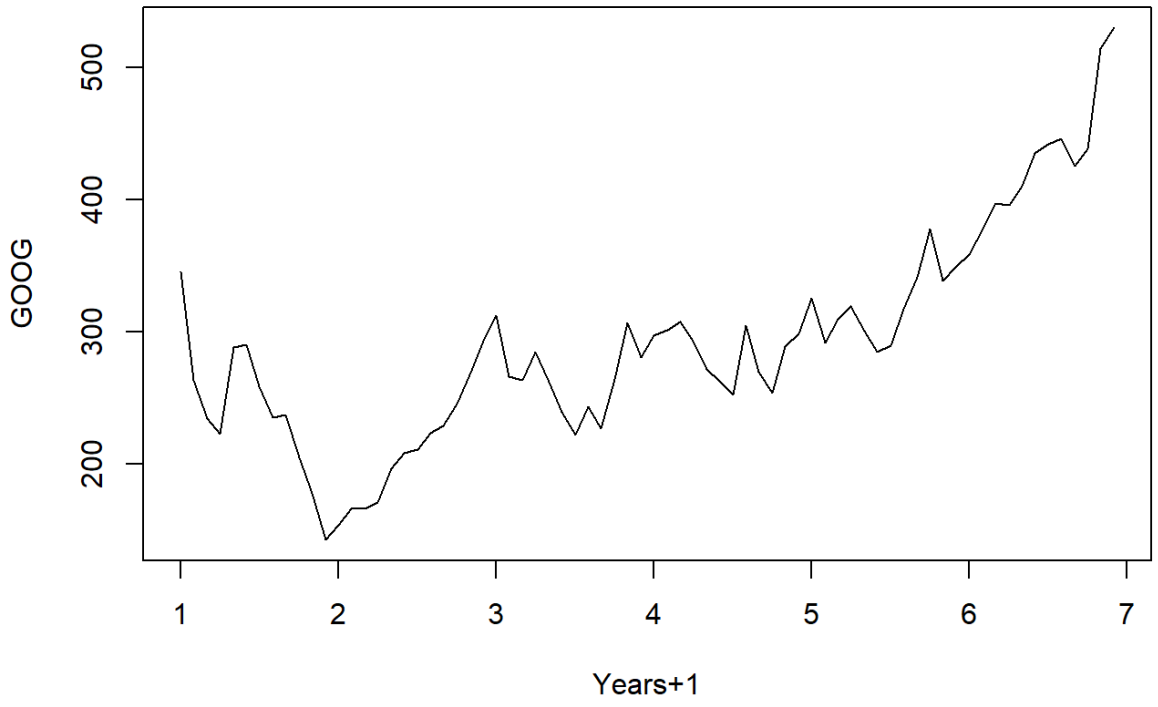
```
## [1] "GOOG"
```

```
head(GOOG)
```

```
##            GOOG.Open GOOG.High GOOG.Low GOOG.Close GOOG.Volume GOOG.Adjusted
## 2008-01-02  345.1413  347.3829 337.5996   341.3157     8646087      341.3157
## 2008-01-03  341.3505  342.1426 336.9969   341.3854     6529382      341.3854
## 2008-01-04  338.5759  339.2086 326.2770   327.2733    10759780      327.2733
## 2008-01-07  325.7490  329.9034 317.4850   323.4128    12854803      323.4128
## 2008-01-08  325.2808  328.7478 314.3218   314.6606    10718225      314.6606
## 2008-01-09  313.8436  325.4501 310.0927   325.3804    13529924      325.3804
```

Summarize monthly and store as time series

```
mGoog <- to.monthly(GOOG)
googOpen <- Op(mGoog)
ts1 <- ts(googOpen,frequency=12) ## create a time-series object
plot(ts1,xlab="Years+1", ylab="GOOG")
```



Example

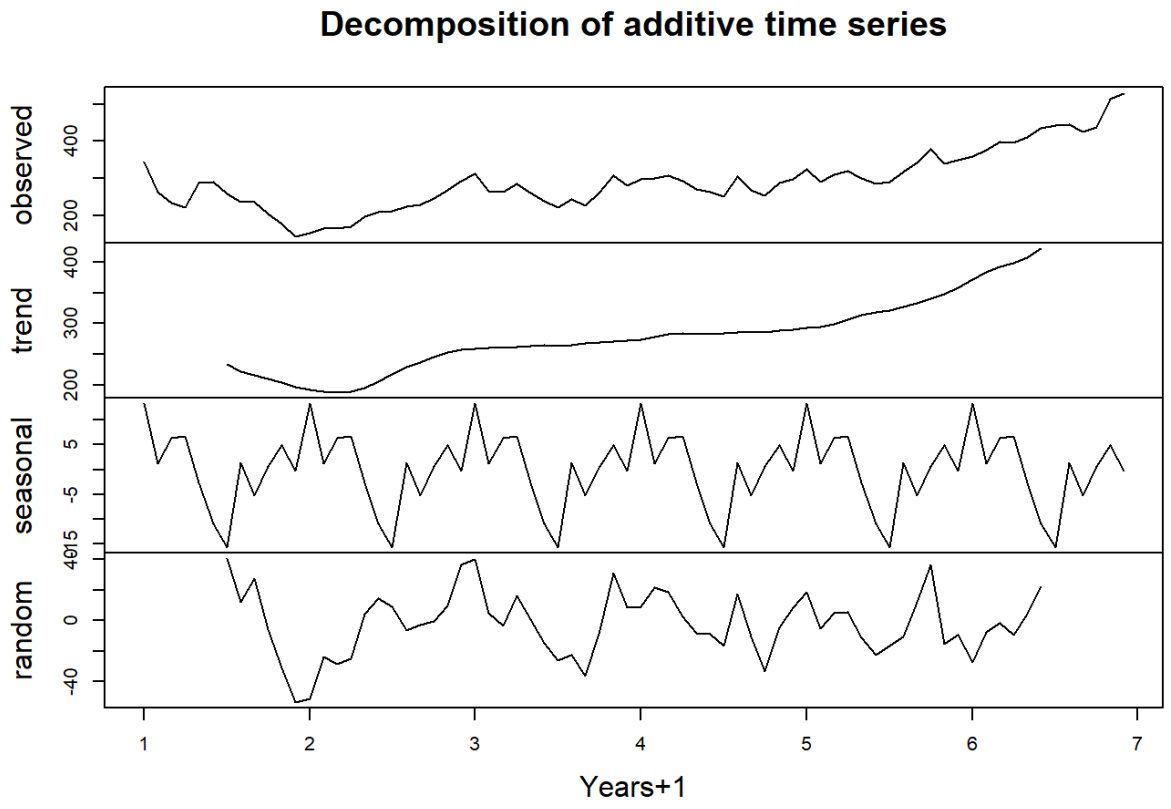
time series decomposition

- **Trend** - Consistently increasing pattern over time
- **Seasonal** - When there is a pattern over a fixed period of time that recurs.
- **Cyclic** - When data rises and falls over non fixed periods

https://www.otexts.org/fpp/6/1 (https://www.otexts.org/fpp/6/1)

Decompose a time series into parts

```
plot(decompose(ts1), xlab="Years+1")
```



Training and test sets

```
## training and test sets get built using consecutive time points
ts1Train <- window(ts1,start=1,end=5)
ts1Test <- window(ts1,start=5,end=(7-0.01))
ts1Train
```

```
##        Jan      Feb      Mar      Apr      May      Jun      Jul      Aug
## 1 345.1413 263.3479 234.8746 223.0340 288.0752 290.1624 258.8199 235.3728
## 2 153.7238 166.5208 166.0426 171.2481 196.7774 208.5832 211.3080 223.5322
## 3 312.3044 266.3018 263.6119 284.6082 262.2670 239.3180 221.8136 243.5820
## 4 297.1263 301.1163 307.7365 293.2807 271.8311 263.0341 252.4239 304.4688
## 5 325.2509
##        Sep      Oct      Nov      Dec
## 1 237.4948 204.8073 178.1224 142.8047
## 2 228.9817 245.5795 267.5372 292.9669
## 3 226.6405 264.0104 306.7154 280.4488
## 4 269.3654 253.9731 288.9669 298.8797
## 5
```

```
ts1Test
```

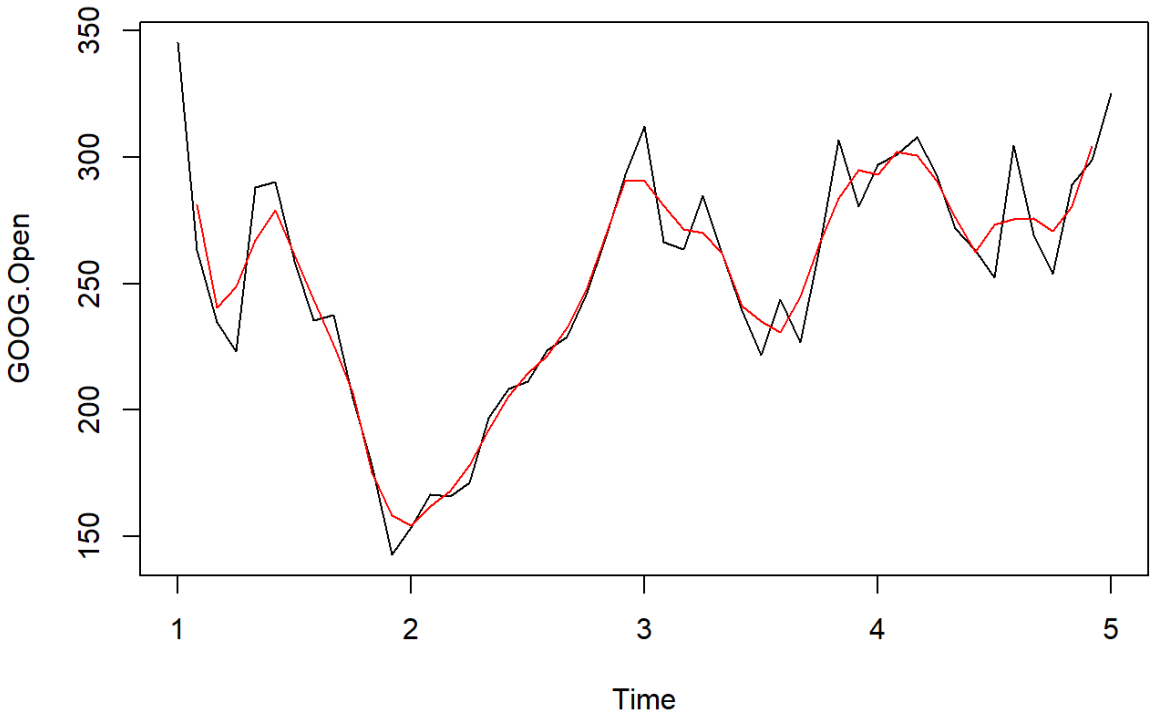
```
##        Jan      Feb      Mar      Apr      May      Jun      Jul      Aug
## 5 325.2509 291.3778 309.9682 319.1886 300.7676 284.8274 289.8237 317.4601
## 6 358.3668 377.6844 397.4104 396.0206 410.0929 434.8700 441.5699 445.8289
##        Sep      Oct      Nov      Dec
## 5 340.9969 378.1078 338.4813 349.8088
## 6 425.5848 438.4815 513.9685 529.7693
```

Simple moving average

One way of doing forecasting. The simple moving average averages up all the values around a prticular time point

$$Y_t = \frac{1}{2 * k + 1} \sum_{j=-k}^{k} y_{t+j}$$

```
library(forecast)
plot(ts1Train)
lines(ma(ts1Train,order=3),col="red")
```



Exponential

smoothing

An alternative way of performing forecasting is using exponential smoothing. this technique weighs near-by time points more heavily than time points that are farther away.

Example - simple exponential smoothing

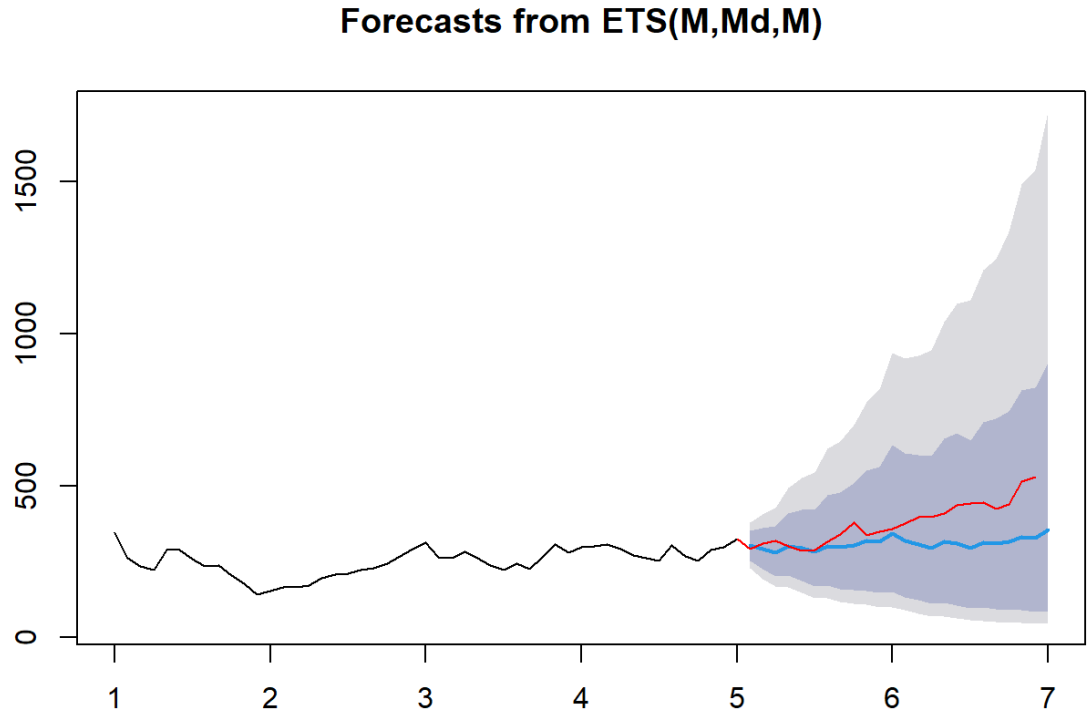
$$\hat{y}_{t+1} = \alpha y_t + (1 - \alpha)\hat{y}_{t-1}$$

There is a large number of different classes of smoothing models that you can choose:

	Seasonal Component		
Trend Component	N (None)	A (Additive)	M (Multiplicative)
N (None)	(N,N)	(N,A)	(N,M)
A (Additive)	(A,N)	(A,A)	(A,M)
A_d (Additive damped)	(A_d,N)	(A_d,A)	(A_d,M)
M (Multiplicative)	(M,N)	(M,A)	(M,M)
M_d (Multiplicative damped)	(M_d,N)	(M_d,A)	(M_d,M)

https://www.otexts.org/fpp/7/6 (https://www.otexts.org/fpp/7/6)

```
ets1 <- ets(ts1Train,model="MMM") ## exponential trend smoothing
fcast <- forecast(ets1)
plot(fcast); lines(ts1Test,col="red")
```



- The coloured lines represented the forecasted predictions and the grey areas identifies the prediction boundaries for the model

Get the accuracy

The accuracy of the forecasting model can be estimated using the testing set

```
accuracy(fcast,ts1Test)
```

```
##                      ME     RMSE      MAE        MPE      MAPE      MASE
## Training set  0.2000302 26.09189 20.65928 -0.3572633  8.302526 0.3933832
## Test set     69.8911746 93.08104 72.04076 16.2879469 17.032408 1.3717625
##                       ACF1 Theil's U
## Training set -0.0002846501        NA
## Test set      0.7575349604  3.375609
```

Notes and further resources

- Forecasting and timeseries prediction (http://en.wikipedia.org/wiki/Forecasting) is an entire field
- Rob Hyndman's Forecasting: principles and practice (https://www.otexts.org/fpp/) is a good place to start
- Cautions
  - Be wary of **spurious correlations**
  - Be careful how far you predict (**extrapolation**)
  - Be wary of **dependencies over time**
- See quantmod (http://cran.r-project.org/web/packages/quantmod/quantmod.pdf) or quandl (http://www.quandl.com/help/packages/r) packages for finance-related problems.

Week 4.4: Unsupervised prediction

Key ideas

- Sometimes you don't know the labels for prediction
- To build a predictor
  - Create clusters (although this is not a perfectly noiseless process)
  - Name clusters (although interpreting the clusters is a very challenging problem)
  - Build predictor for clusters
- In a new data set
  - Predict clusters

Iris example ignoring species labels

```
data(iris); library(ggplot2); library(caret)
inTrain <- createDataPartition(y=iris$Species,
                               p=0.7, list=FALSE)
training <- iris[inTrain,]
testing <- iris[-inTrain,]
dim(training); dim(testing)
```

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

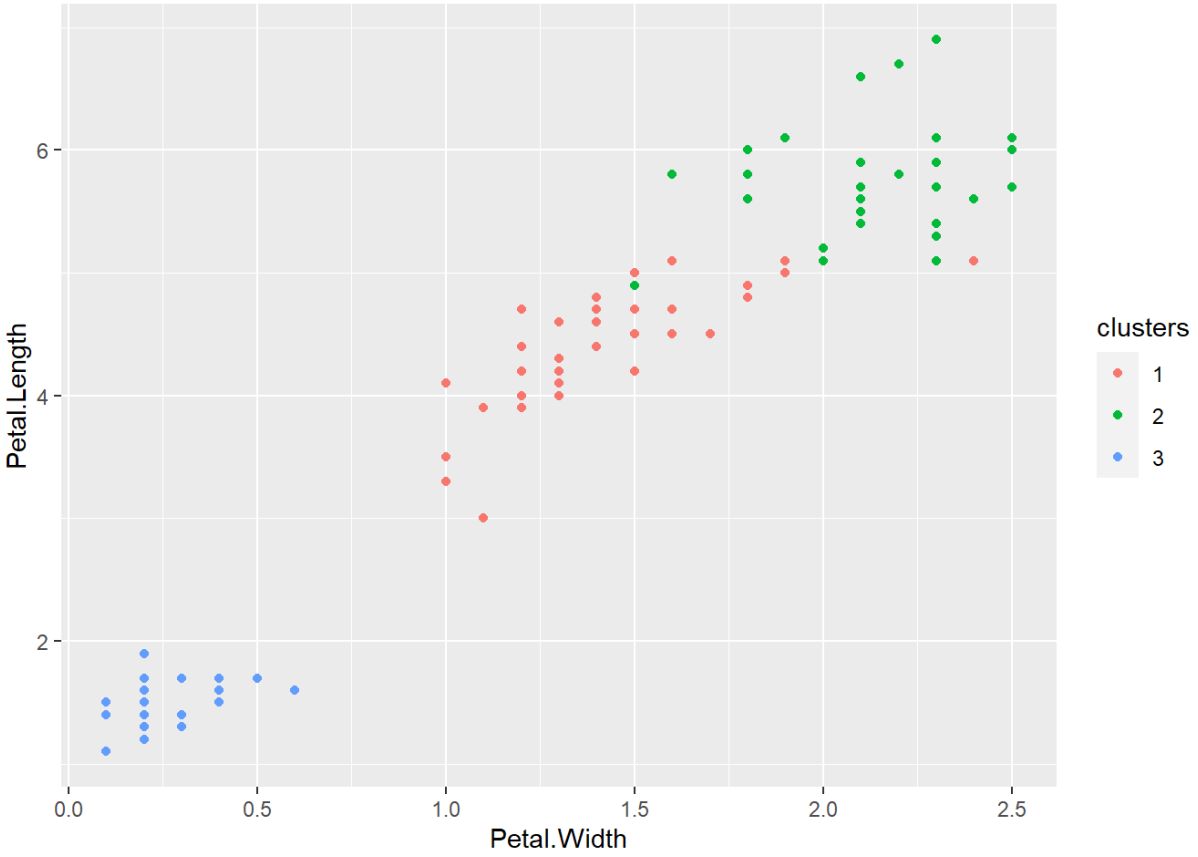
```
## [1] 45  5
```

Cluster with k-means

```
kMeans1<-kmeans(subset(training, select=-c(Species)), centers = 3)
kMeans1
```

```
## K-means clustering with 3 clusters of sizes 43, 27, 35
##
## Cluster means:
##   Sepal.Length Sepal.Width Petal.Length Petal.Width
## 1     5.934884    2.806977     4.383721   1.4279070
## 2     6.885185    3.100000     5.759259   2.1259259
## 3     5.040000    3.400000     1.471429   0.2514286
##
## Clustering vector:
##   1   3   6   7   8   9  10  11  12  14  15  16  18  19  21  24  25  26  27  28
##   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3
##  29  30  31  32  34  36  37  38  40  41  42  44  46  48  50  51  52  53  57  58
##   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   1   1   2   1   1
##  59  61  62  64  66  67  68  70  71  72  74  75  76  77  79  80  83  84  86  87
##   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
##  89  91  92  93  94  95  96  97  99 100 101 102 103 104 105 106 107 109 110 111
##   1   1   1   1   1   1   1   1   1   1   2   1   2   2   2   2   1   2   2   2
## 113 115 116 118 119 120 121 124 125 126 127 128 129 130 131 136 139 140 141 142
##   2   1   2   2   2   1   2   1   2   2   1   1   2   2   2   2   1   2   2   2
## 144 145 147 148 149
##   2   2   1   2   2
##
## Within cluster sum of squares by cluster:
## [1] 29.07070 14.97111 10.86286
##  (between_SS / total_SS =  88.5 %)
##
## Available components:
##
## [1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss"
## [6] "betweenss"    "size"         "iter"         "ifault"
```

```
training$clusters<-as.factor(kMeans1$cluster)
qplot(data=training,Petal.Width,Petal.Length,colour=clusters)
```



Compare to real labels

```
table(kMeans1$cluster,training$Species)
```

```
##
##     setosa versicolor virginica
##   1      0         34         9
##   2      0          1        26
##   3     35          0         0
```

Build predictor

```
modFit <- train(clusters ~.,data=subset(training,select=-c(Species)),method="rpart")
table(predict(modFit,training),training$Species)
```

```
##
##     setosa versicolor virginica
##   1      0         35        11
##   2      0          0        24
##   3     35          0         0
```

- The clustering categories are used as outcome variable.

Apply on test

```
testClusterPred <- predict(modFit,testing)
table(testClusterPred ,testing$Species)
```

```
##
## testClusterPred setosa versicolor virginica
##               1      0         15         5
##               2      0          0        10
##               3     15          0         0
```

Notes and further reading

- The `cl_predict` function in the **clue package** provides similar functionality
- Beware over-interpretation of clusters!
- This is one basic approach to recommendation engines (http://en.wikipedia.org/wiki/Recommender_system)
- Elements of statistical learning (http://www-stat.stanford.edu/~tibs/ElemStatLearn/)
- Introduction to statistical learning (http://www-bcf.usc.edu/~gareth/ISL/)