[1]:

```
## Importing packages

# This R environment comes with all of CRAN and many other helpful packag
# You can see which packages are installed by checking out the kaggle/rst
# https://github.com/kaggle/docker-rstats

library(tidyverse) # metapackage with lots of helpful functions

## Running code

# In a notebook, you can run a single code cell by clicking in the cell a
# the blue arrow to the left, or by clicking in the cell and pressing Shi
# you can run code by highlighting the code you want to run and then clic
# at the bottom of this window.

## Reading in files

# You can access files from datasets you've added to this kernel in the '
# You can see the files added to this kernel by running the code below.

list.files(path = "../input")

## Saving data

# If you save any files or images, these will be put in the "output" dire
# can see the output directory by committing and running your kernel (usi
# Commit & Run button) and then checking out the compiled version of your
```

```
── Attaching packages ─────────────────────────────────────── tidyver
se 1.2.1 ──
✔ ggplot2 3.1.0.9000     ✔ purrr   0.3.0
✔ tibble  2.0.1          ✔ dplyr   0.7.8
✔ tidyr   0.8.2          ✔ stringr 1.3.1
✔ readr   1.3.1          ✔ forcats 0.3.0
── Conflicts ──────────────────────────────────────── tidyverse_con
flicts() ──
✖ dplyr::filter() masks stats::filter()
✖ dplyr::lag()    masks stats::lag()
```

[2]:
```
#
# Logistic Regression
# https://youtu.be/TxvEVc8YNlU
#
```

[3]:
```
# Here we use the command require, which is similar to library.
# I tend to use require. It's sort of more evocative
require(ISLR)
#?require
```

```
Loading required package: ISLR
```

[4]:
```
# names is useful for seeing what's on the data frame
names(Smarket)
```

'Year' 'Lag1' 'Lag2' 'Lag3' 'Lag4' 'Lag5' 'Volume' 'Today' 'Direction'

[5]:

```
# Summary gives you a simple summary of each of the variables on the Smar
summary(Smarket)
```

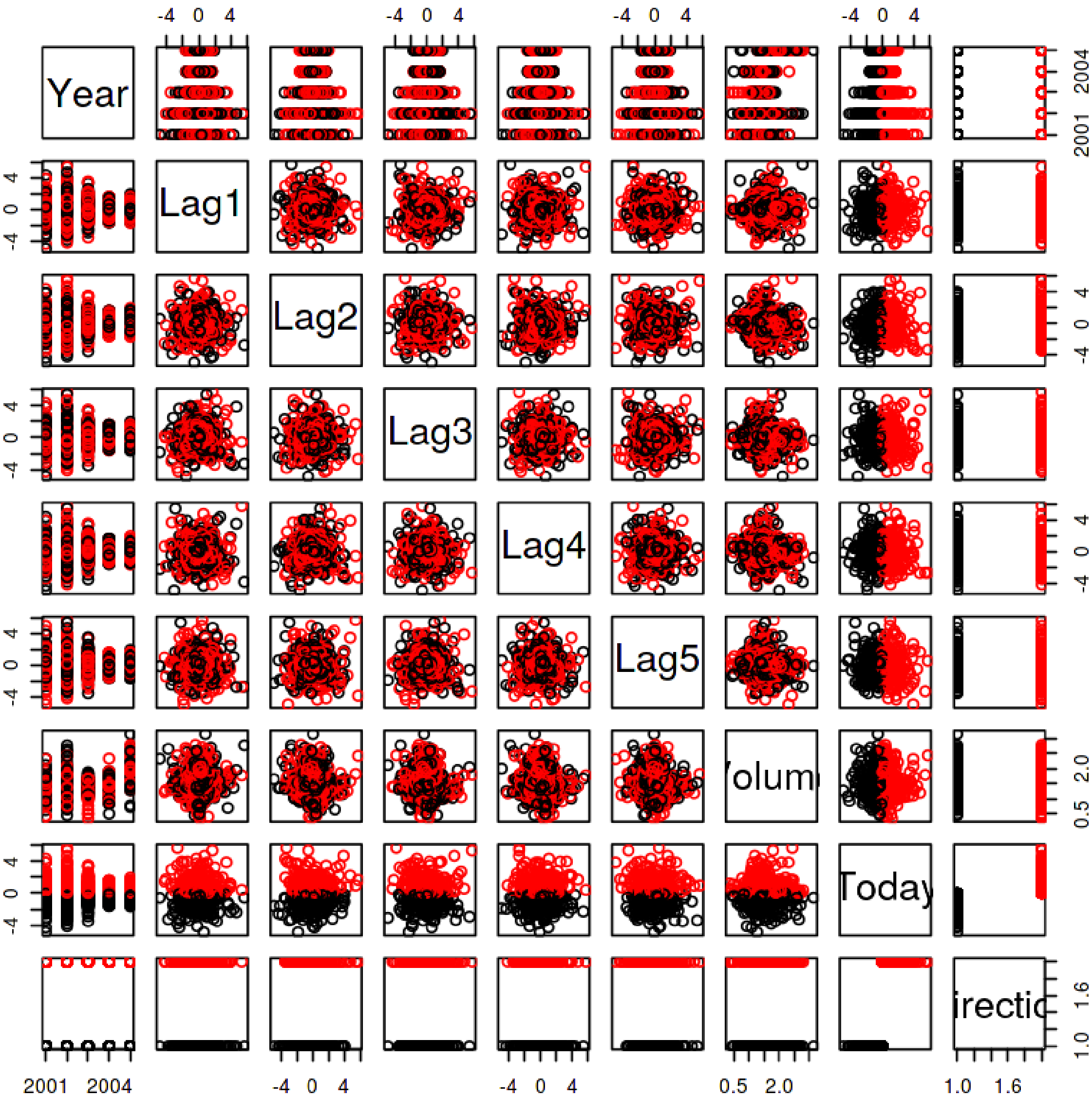
```
     Year           Lag1                Lag2                Lag3          
 Min.   :2001   Min.   :-4.922000   Min.   :-4.922000   Min.   :-4.92
2000
 1st Qu.:2002   1st Qu.:-0.639500   1st Qu.:-0.639500   1st Qu.:-0.64
0000
 Median :2003   Median : 0.039000   Median : 0.039000   Median : 0.03
8500
 Mean   :2003   Mean   : 0.003834   Mean   : 0.003919   Mean   : 0.00
1716
 3rd Qu.:2004   3rd Qu.: 0.596750   3rd Qu.: 0.596750   3rd Qu.: 0.59
6750
 Max.   :2005   Max.   : 5.733000   Max.   : 5.733000   Max.   : 5.73
3000
     Lag4                Lag5              Volume           Today          
 Min.   :-4.922000   Min.   :-4.92200   Min.   :0.3561   Min.   :-4.9
22000
 1st Qu.:-0.640000   1st Qu.:-0.64000   1st Qu.:1.2574   1st Qu.:-0.6
39500
 Median : 0.038500   Median : 0.03850   Median :1.4229   Median : 0.0
38500
 Mean   : 0.001636   Mean   : 0.00561   Mean   :1.4783   Mean   : 0.0
03138
 3rd Qu.: 0.596750   3rd Qu.: 0.59700   3rd Qu.:1.6417   3rd Qu.: 0.5
96750
 Max.   : 5.733000   Max.   : 5.73300   Max.   :3.1525   Max.   : 5.7
33000
 Direction 
 Down:602  
 Up  :648  
```

[6]:

```
# And we can also do help on these data objects and get some details of e
#?Smarket

# So we're going to use the direction as a response and
# see if we can predict it as a binary response using logistic regression
```

[7]:

```
# And we told it to use as the color indicator, actually our binary respo
# And that's a useful way, when you've got a two-class variable for seein
# which are members of each class.
pairs(Smarket,col=Smarket$Direction)
#?pairs
```



[8]:

```
# first 5 elements
Smarket$Direction[1:5]
```

Up Up Down Up Up

▶ **Levels**:

[9]:

```
# Logistic regression
```

[10]:

```
# And so that tells GLM to put fit a logistic regression
# model instead of one of the many other models that can be
# fit to the GLM.
glm.fit=glm(Direction~Lag1+Lag2+Lag3+Lag4+Lag5+Volume,
            data=Smarket,family=binomial)

glm.fit
```

```
Call:  glm(formula = Direction ~ Lag1 + Lag2 + Lag3 + Lag4 + Lag5 + 
    Volume, family = binomial, data = Smarket)

Coefficients:
(Intercept)         Lag1         Lag2         Lag3         Lag4
Lag5
  -0.126000    -0.073074    -0.042301     0.011085     0.009359
0.010313
     Volume
   0.135441

Degrees of Freedom: 1249 Total (i.e. Null);  1243 Residual
Null Deviance:      1731
Residual Deviance: 1728         AIC: 1742
```

[11]:
```
summary(glm.fit)
```

```
Call:
glm(formula = Direction ~ Lag1 + Lag2 + Lag3 + Lag4 + Lag5 +
    Volume, family = binomial, data = Smarket)

Deviance Residuals:
   Min      1Q  Median      3Q     Max
-1.446  -1.203   1.065   1.145   1.326

Coefficients:
             Estimate Std. Error z value Pr(>|z|)
(Intercept) -0.126000   0.240736  -0.523    0.601
Lag1        -0.073074   0.050167  -1.457    0.145
Lag2        -0.042301   0.050086  -0.845    0.398
Lag3         0.011085   0.049939   0.222    0.824
Lag4         0.009359   0.049974   0.187    0.851
Lag5         0.010313   0.049511   0.208    0.835
Volume       0.135441   0.158360   0.855    0.392

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 1731.2  on 1249  degrees of freedom
Residual deviance: 1727.6  on 1243  degrees of freedom
AIC: 1741.6

Number of Fisher Scoring iterations: 3
```

[12]:

```
# And it seems like none of the coefficients are significant here.
# Again, not a big surprise for these kinds of data.
# It doesn't necessarily mean it won't be able to make any kind of reason
# It just means that possibly these variables are very correlated.
# Actually, the plot doesn't suggest that.
# Anyway, none are significant.

# And it gives the null deviance, which is the deviance just for the mean
# So that's the log likelihood if you just use the mean model,
# and then the deviance for the model with all the predictors in,
# that's the residual deviance.
# And there was a very modest change in deviance.
# It looks like four units on six degrees of freedom

# => by involving 6 degrees of freedom (log1 log2 log3 log4 log5 volume)
# , the deviance only reduced by 4 units
```

[13]:

```
# https://stats.stackexchange.com/questions/108995/interpreting-residual-
# If your Null Deviance is really small, it means that the Null Model exp
# the data pretty well. Likewise with your Residual Deviance.
# you should see that the degrees of freedom reported on the Null are alw
# higher than the degrees of freedom reported on the Residual.
# That is because :
# Null Deviance df = Saturated df - Null df = n-1
# Residual Deviance df = Saturated df - Proposed df = n-(p+1)

# https://www.theanalysisfactor.com/r-glm-model-fit/
# deviance: it’s a measure of "badness" of fit—higher numbers indicate wo
# R reports two forms of deviance – the null deviance and the residual de
# The null deviance shows how well the response variable is predicted by
# that includes only the intercept (grand mean).
```

[14]:

```
# we can make predictions from the fitted model.
# And so we assign to glm.probs the predict of glm.fit, and we
# tell it type equals response.
glm.probs=predict(glm.fit,type="response")
glm.probs[1:5]
# And it gives you a vector of fitted probabilities.
# We can look at the first five, and we see that they're very
# close to 50%, which is, again, not too surprising.
# We don't expect to get strong predictions in this case.
# So this is a prediction of whether the market's going to
# be up or down based on the lags and the other predictors.
#?predict
```

1	0.507084133395402
2	0.481467878454591
3	0.481138835214201
4	0.515222355813022
5	0.510781162691538

[15]:

```
# We can turn those probabilities into classifications by
# thresholding at 0.5. And so we do that by using the if/else command.
# So if/else takes effect, in this case, glm.probs, a vector of logicals.
# So glm.probs bigger than 0.5.
# So that'll be a vector of trues and falses.
# And then if/else says that, element by element, if it's
# true, you're going to call it up.
# Otherwise, you're going to call it down.
glm.pred=ifelse(glm.probs>0.5,"Up","Down")
```

[16]:

```
attach(Smarket)
```

[17]:
```
# And now we're going to look at our performance.
# And now we can make a table of glm.pred, which is our ups and downs
# from our prediction, against the true direction.
table(glm.pred,Direction)

# And we get a table, and we see there's lots of elements on the off diag
# On the diagonals is where we do correct classification, and
# on the off diagonals is where we make mistakes.
```

```
        Direction
glm.pred Down  Up
    Down  145 141
    Up    457 507
```

[18]:
```
# And we can actually get our mean classification performance.
# So that's cases where glm.pred is equal to the direction.
# And we just take the mean of those, so it'll give you a
# proportion, in this case, 0.52.
mean(glm.pred==Direction)
```

0.5216

[19]:
```
# Well, we may have overfit on the training data.
# So what we're going to do now is divide our data up into a
# training and test set.
```

[20]:

```
# So what we'll do is we'll make a vector of logicals.
# And what it is is train is equal to year less than 2005.
# For all those observations for which year is less than 2005,
# we'll get a true. Otherwise, we'll get a false.
train = Year<2005

# look at the first 10 elements
train[1:10]
```

TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE

[21]:

```
# And now we refit our glm.fit, except we say subset equals train.
# And so it will use any those observations for which train is true.
# So now that means we fit just to the data in years less than 2005.
glm.fit=glm(Direction~Lag1+Lag2+Lag3+Lag4+Lag5+Volume,
            data=Smarket,family=binomial, subset=train)
```

[22]:

```
# And now, when we come to predict, we're going to predict on the remaini
# which is years 2005 or greater.
# And so we use the predict function again.
# And for the new data, we give it Smarket, but index by not trained.
glm.probs=predict(glm.fit,newdata=Smarket[!train,],type="response")
```

[23]:

```
glm.pred=ifelse(glm.probs >0.5,"Up","Down")
```

[24]:

```
# And let's make a subset, a new variable, direction.2005,
# for the test data, which is the response
# variable, direction, which is just for our test data.
# In other words, not trained.
Direction.2005=Smarket$Direction[!train]
```

[25]:

```
# So now this is on test data.
table(glm.pred,Direction.2005)
mean(glm.pred==Direction.2005)

# So now we actually get slightly less than 50%.
# So we're doing worse than the null rate, which is 50%.
# Well, we might be overfitting. And that's why we're doing worse on the
```

```
        Direction.2005
glm.pred Down Up
    Down   77 97
    Up     34 44
```

0.48015873015873

[26]:

```
# So now we're going to fit a smaller model.
# So we're going to just use lag1 and lag2 and leave out
# all the other variables. The rest of it calls the same.
```

[27]:

```
#Fit smaller model
glm.fit=glm(Direction~Lag1+Lag2,
            data=Smarket,family=binomial, subset=train)
glm.probs=predict(glm.fit,newdata=Smarket[!train,],type="response")
glm.pred=ifelse(glm.probs >0.5,"Up","Down")
table(glm.pred,Direction.2005)
mean(glm.pred==Direction.2005)
```

```
        Direction.2005
glm.pred Down  Up
    Down   35  35
    Up     76 106
```

0.55952380952381

[28]:
```
# we get a correct classification of just
# about 56%???, which is not too bad at all.
106/(76+106)
```

0.582417582417582

[29]:
```
# And so using the smaller model, it appears to have done better here.
# And if we do a summary of that guy, let's see if anything
# became significant by using the smaller model, given that
# it gave us better predictions.
summary(glm.fit)
```

```
Call:
glm(formula = Direction ~ Lag1 + Lag2, family = binomial, data = Smar
ket,
    subset = train)

Deviance Residuals:
   Min      1Q  Median      3Q     Max
-1.345  -1.188   1.074   1.164   1.326

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)  0.03222    0.06338   0.508    0.611
Lag1        -0.05562    0.05171  -1.076    0.282
Lag2        -0.04449    0.05166  -0.861    0.389

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 1383.3  on 997  degrees of freedom
Residual deviance: 1381.4  on 995  degrees of freedom
AIC: 1387.4

Number of Fisher Scoring iterations: 3
```

[30]:
```
#
# Linear Discriminant Analysis
# https://youtu.be/2cl7JiPzkBY
#
```

[31]:
```
require(ISLR)
require(MASS)
```

```
Loading required package: MASS

Attaching package: ‘MASS’

The following object is masked from ‘package:dplyr’:

    select
```

[32]:
```
## Linear Discriminant Analysis
```

[33]:
```
?lda
```

[34]:
```
# And we're going to use the subset, which is years less than 2005,
# because later on, we're going to make predictions for year 2005.
# So you can put that directly in the formula, subset equals year less th
lda.fit=lda(Direction~Lag1+Lag2,data=Smarket, subset=Year<2005)
```

[35]:
```
# And it fits it so very quickly, and we print it by just typing its name
lda.fit
```

```
Call:
lda(Direction ~ Lag1 + Lag2, data = Smarket, subset = Year <
    2005)

Prior probabilities of groups:
    Down       Up
0.491984 0.508016

Group means:
            Lag1        Lag2
Down  0.04279022  0.03389409
Up   -0.03954635 -0.03132544

Coefficients of linear discriminants:
            LD1
Lag1 -0.6420190
Lag2 -0.5135293
```

[36]:
```
# So the prior probabilities are just the proportions of ups and downs in
# It's roughly 50%, which says something about the market, It's kind of a

# It summarizes the group means for the two groups, for the downs and the
# It looks like there may be a slight difference in these two groups.

# And then it gives the LDA coefficients.
# So if you remember the LDA function fits a linear function for separati
# And so, it's got two coefficients.
```

[37]:

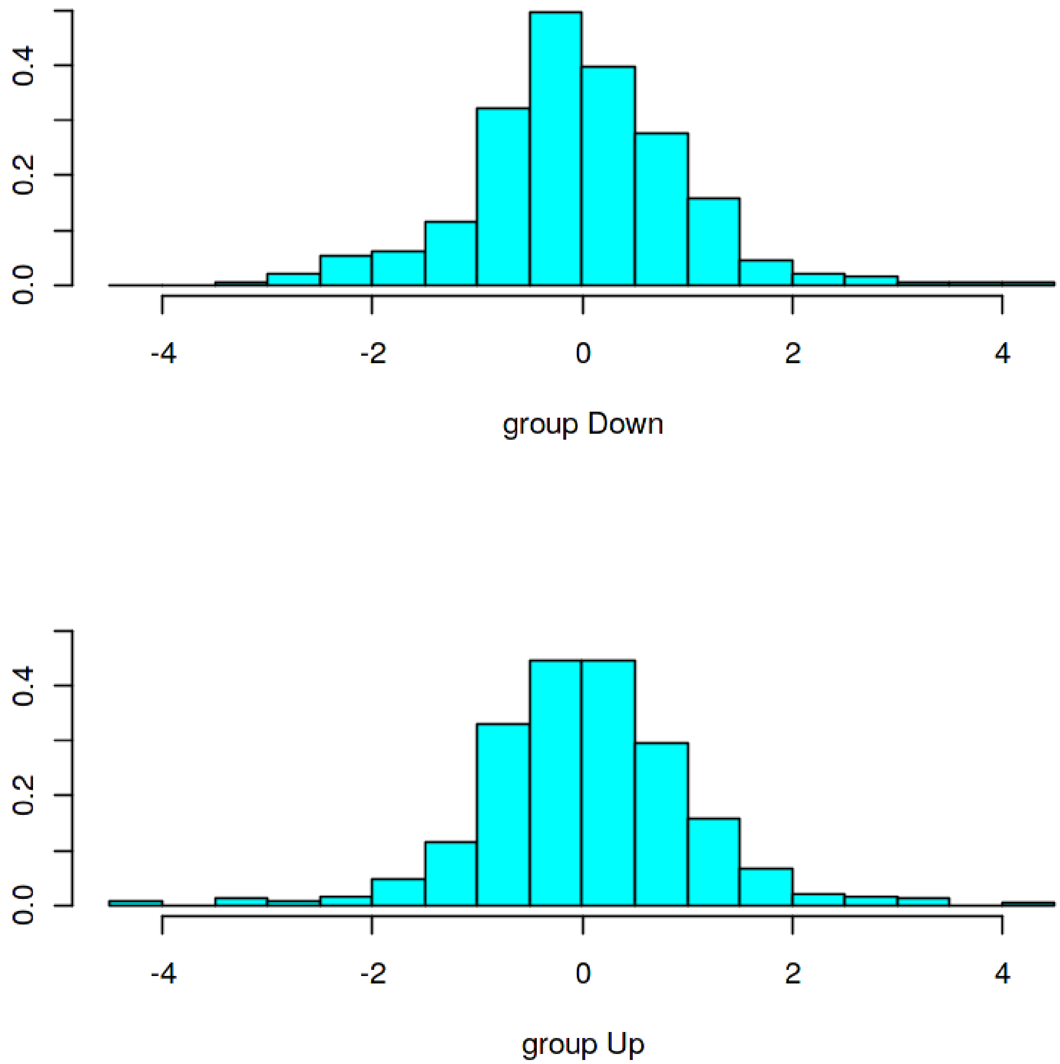
```
# https://stats.stackexchange.com/questions/87479/what-are-coefficients-o
# LDA has 2 distinct stages: extraction and classification.
# At extraction, latent variables called discriminants are formed,
# as linear combinations of the input variables.
# The coefficients in that linear combinations are called discriminant co
# On the 2nd stage, data points are assigned to classes by those discrimi
# not by original variables.
```

[38]:
```
lda.fit[1:10]
```

```
$prior
    Down       Up 
0.491984 0.508016 

$counts
Down   Up 
 491  507 

$means
            Lag1        Lag2
Down  0.04279022  0.03389409
Up   -0.03954635 -0.03132544

$scaling
            LD1
Lag1 -0.6420190
Lag2 -0.5135293

$lev
[1] "Down" "Up"  

$svd
[1] 1.363832

$N
[1] 998

$call
lda(formula = Direction ~ Lag1 + Lag2, data = Smarket, subset = Year 
< 
    2005)

$terms
Direction ~ Lag1 + Lag2
attr(,"variables")
list(Direction, Lag1, Lag2)
attr(,"factors")
          Lag1 Lag2
Direction    0    0
```

```
Lag1         1    0
Lag2         0    1
attr(,"term.labels")
[1] "Lag1" "Lag2"
attr(,"order")
[1] 1 1
attr(,"intercept")
[1] 1
attr(,"response")
[1] 1
attr(,".Environment")
<environment: R_GlobalEnv>
attr(,"predvars")
list(Direction, Lag1, Lag2)
attr(,"dataClasses")
Direction      Lag1      Lag2
 "factor" "numeric" "numeric"

$xlevels
named list()
```

[39]:
```
# It plots a linear discriminant function separately,
# the values of the linear discriminant function,
# separately for the up group and the down group.
# And when we look at this, it looks to the eye like there's really not m
plot(lda.fit)
```



[40]:

```
# So now we're going to see how well our rule predicts on the year 2005
# I'm doing this in a slightly different way

# And we use a command in R-- a useful command-- called subset.
# And so the first argument is the data frame that you're going to subset
# which is s market. And then following that are some logical expressions
# can use variables in that data frame to define the subset.
# And that will create a data frame with just the 2005 observations.
Smarket.2005=subset(Smarket,Year==2005)
```

[41]:

```
# And so now we can use that as the test data, or the place where we want
lda.pred=predict(lda.fit,Smarket.2005)
```

[42]:

```
# print the first 5 of these
lda.pred[1:5,]
# So I was assuming it was in a matrix format, and it's not.
# So what format is it?
```

```
Error in lda.pred[1:5, ]: incorrect number of dimensions
Traceback:
```

[43]:

```
class(lda.pred)
```

'list'

[44]:

```
# And when you have a list of variables, and each of the
# variables have the same number of observations, a convenient
# way of looking at such a list is through data frame.
data.frame(lda.pred)[1:5,]
```

	class	posterior.Down	posterior.Up	LD1
999	Up	0.4901792	0.5098208	0.08293096
1000	Up	0.4792185	0.5207815	0.59114102
1001	Up	0.4668185	0.5331815	1.16723063
1002	Up	0.4740011	0.5259989	0.83335022
1003	Up	0.4927877	0.5072123	-0.03792892

[45]:

```
# The thing we're really interested in here is the classification,
# We'll do a table of that, and we get the little confusion matrix
table(lda.pred$class,Smarket.2005$Direction)

# ones, we can just take the mean of that, and it'll give
# us our current classification rate, which in this case is about 0.56.
mean(lda.pred$class==Smarket.2005$Direction)
```

```
      Down  Up
Down    35  35
Up      76 106
```

0.55952380952381

[46]:

```
#
# Nearest Neighbor Classification
# https://youtu.be/9TVVF7CS3F4
#
```

[47]:
```
## K-Nearest Neighbors
```

[48]:
```
# This time, we're going to look at k-nearest neighbor classification,
# which it's one of those tools that's a verysimple classification rule,
# but it's effective a lot of the time.
# Some experts have written that k-nearest neighbors do the best about or
# And it's so simple that, in the game of doing classification,
# you always want to have k-nearest neighbors in your toolbox.
```

[49]:
```
library(class)
```

[50]:
```
#?knn
```

[51]:
```
attach(Smarket)
```

```
The following objects are masked from Smarket (pos = 5):

    Direction, Lag1, Lag2, Lag3, Lag4, Lag5, Today, Volume, Year
```

[52]:
```
# So what we'll do is we'll make a matrix of lag1 and lag2.
Xlag=cbind(Lag1,Lag2)
```

[53]:

```
# let's look at the first five rows of that matrix.
Xlag[1:5,]
```

Lag1	Lag2
0.381	-0.192
0.959	0.381
1.032	0.959
-0.623	1.032
0.614	-0.623

[54]:

```
# And then we'll make a indicator variable Train which is year less than
train=Year<2005
```

[55]:

```
# we're ready to call k and n, so we give our matrix x lag,
# and right in line there we index it by Train which is just using the tr
# And then, for the test observations, we give it x lag not Train.
# So those not train will be, therefore, those that are equal to 2005
# k=1:
# that means what the algorithm does is,
# it says to classify a new observation,
# you go into the training set in the x space, the feature space,
# and you look for the training observation that's
# closest to your test point in Euclidean distance and you classify to it
knn.pred=knn(Xlag[train,],Xlag[!train,],Direction[train],k=1)
```

[56]:
```
table(knn.pred,Direction[!train])
# so its useless ...
```

```
knn.pred Down Up
    Down   43 58
    Up     68 83
```

[57]:
```
mean(knn.pred==Direction[!train])
```

0.5

[ ]:

[ ]: