Chapter 3

January 12, 2021

1 Housekeeping

[2]:
```
sessionInfo()
options(repr.plot.width=14,repr.plot.antialias='subpixel',repr.plot.res=218)
update.packages()
```

```
R version 4.0.3 Patched (2020-10-12 r79333)
Platform: x86_64-apple-darwin17.0 (64-bit)
Running under: macOS Big Sur 10.16

Matrix products: default
BLAS:   /Library/Frameworks/R.framework/Versions/4.0/Resources/lib/libRblas.dylib
LAPACK: /Library/Frameworks/R.framework/Versions/4.0/Resources/lib/libRlapack.dylib

locale:
[1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8

attached base packages:
[1] stats     graphics  grDevices utils     datasets  methods   base

loaded via a namespace (and not attached):
 [1] compiler_4.0.3  ellipsis_0.3.1  IRdisplay_0.7.0 pbdZMQ_0.3-3
 [5] tools_4.0.3     htmltools_0.5.0 pillar_1.4.6    base64enc_0.1-3
 [9] crayon_1.3.4    uuid_0.1-4      IRkernel_1.1.1  jsonlite_1.7.1
[13] digest_0.6.25   lifecycle_0.2.0 repr_1.1.0      rlang_0.4.8
[17] evaluate_0.14
```

2 Semiconductors

Read in data

[3]:
```
SC <- read.csv("semiconductor.csv")
```

Full model

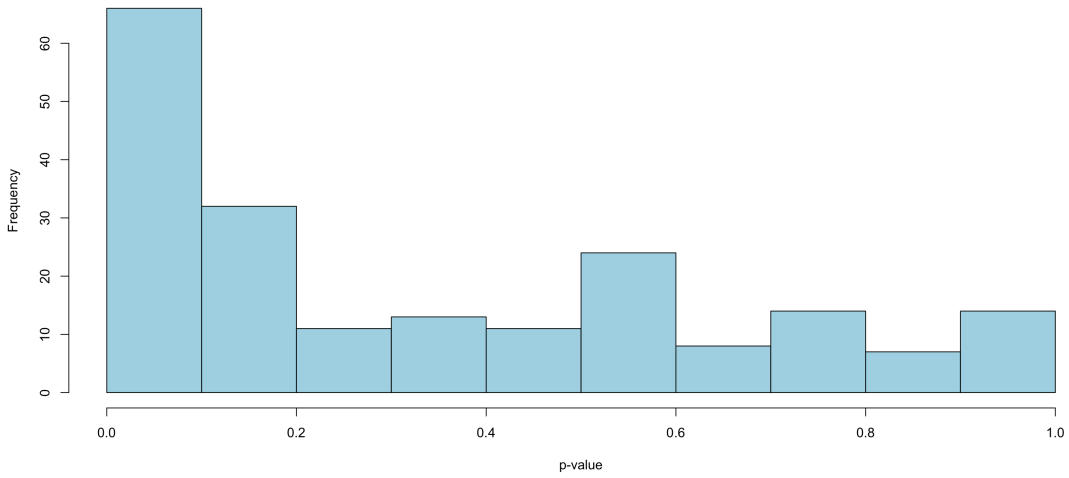
[4]:
```
full <- glm(FAIL ~ ., data=SC, family=binomial)
1 - full$deviance/full$null.deviance
```

```
Warning message:
“glm.fit: fitted probabilities numerically 0 or 1 occurred”
```

0.562143192474375

Grab and plot p-values

[5]:
```
pvals <- summary(full)$coef[-1,4] #-1 to drop the intercept
hist(pvals, xlab="p-value", main="", col="lightblue")
```



Do a Benjamini-Hochberg correction with 10% FDR

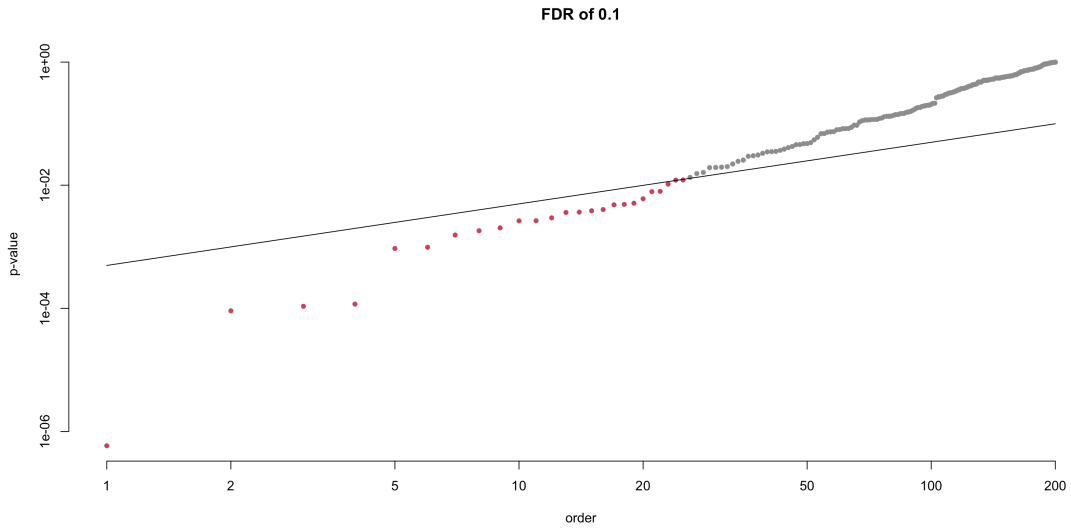
[6]:
```
fdr_cut <- function(pvals, q=0.1){
  pvals <- sort(pvals[!is.na(pvals)])
  N <- length(pvals)
  k <- rank(pvals, ties.method="min")
  alpha <- max(pvals[ pvals<= (q*k/(N+1)) ])

  plot(pvals, log="xy", xlab="order", main=sprintf("FDR of %g",q),
   ylab="p-value", bty="n", col=c(8,2)[(pvals<=alpha) + 1], pch=20)
  lines(1:N, q*(1:N)/(N+1))

  return(alpha)
}

fdr_cut(pvals)
```

0.0121704339598333



Using this cutoff, estimate a more parsimonious model

[7]:
```
( signif <- which(pvals <= 0.0121704339598325) )
cutvar <- c("FAIL", names(signif))
cut <- glm(FAIL ~ ., data=SC[,c("FAIL", names(signif))], family="binomial")
1 - cut$deviance/cut$null.deviance # new in-sample R2
```

SIG2 2 **SIG17** 17 **SIG19** 19 **SIG20** 20 **SIG22** 22 **SIG24** 24 **SIG46** 46 **SIG47** 47 **SIG49** 49 **SIG50** 50 **SIG52** 52 **SIG59** 59 **SIG61** 61 **SIG69** 69 **SIG102** 102 **SIG111** 111 **SIG136** 136 **SIG142** 142 **SIG154** 154 **SIG166** 166 **SIG178** 178 **SIG186** 186 **SIG189** 189 **SIG200** 200

0.179499721344181

Define deviance so it is not necessarily in-sample

[8]:
```
deviance <- function(y, pred, family=c("gaussian","binomial")){
        family <- match.arg(family)
        if(family=="gaussian"){
                return( sum( (y-pred)^2 ) )
        }else{
                if(is.factor(y)) y <- as.numeric(y)>1
                return( -2*sum( y*log(pred) + (1-y)*log(1-pred) ) )
        }
}
```

For R2, also do null deviance (not necessarily in-sample)

[9]:
```
R2 <- function(y, pred, family=c("gaussian","binomial")){
        fam <- match.arg(family)
        if(fam=="binomial"){
```

```
                if(is.factor(y)){ y <- as.numeric(y)>1 }
        }
        dev <- deviance(y, pred, family=fam)
        dev0 <- deviance(y, mean(y), family=fam)
        return(1-dev/dev0)
}
```

Set up folds

[10]:
```
n <- nrow(SC) # the number of observations
K <- 10 # the number of `folds'
# create a vector of fold memberships (random order)
foldid <- rep(1:K,each=ceiling(n/K))[sample(1:n)]
# create an empty dataframe of results
OOS <- data.frame(full=rep(NA,K), cut=rep(NA,K))
```

The OOS experiment in a loop over folds

[11]:
```
for(k in 1:K){
        train <- which(foldid!=k) # train on all but fold `k'

        ## fit the two regressions
        rfull <- glm(FAIL~., data=SC, subset=train, family=binomial)
        rcut <- glm(FAIL~., data=SC[,cutvar], subset=train, family=binomial)

        ## get predictions: type=response so we have probabilities
        predfull <- predict(rfull, newdata=SC[-train,], type="response")
        predcut <- predict(rcut, newdata=SC[-train,], type="response")

        ## calculate and log R2
        OOS$full[k] <- R2(y=SC$FAIL[-train], pred=predfull, family="binomial")
        OOS$cut[k] <- R2(y=SC$FAIL[-train], pred=predcut, family="binomial")

        ## print progress
        cat(k, " ")
}
```

```
Warning message:
‘‘glm.fit: fitted probabilities numerically 0 or 1 occurred’’
```

```
1
```

```
Warning message:
‘‘glm.fit: fitted probabilities numerically 0 or 1 occurred’’
```

```
2
```

```
Warning message:
‘‘glm.fit: fitted probabilities numerically 0 or 1 occurred’’
```

```
3
```

```
Warning message:
‘‘glm.fit: fitted probabilities numerically 0 or 1 occurred’’
```

```
4
```

```
Warning message:
‘‘glm.fit: algorithm did not converge’’
Warning message:
‘‘glm.fit: fitted probabilities numerically 0 or 1 occurred’’
```

```
5
```

```
Warning message:
‘‘glm.fit: fitted probabilities numerically 0 or 1 occurred’’
```

```
6
```

```
Warning message:
‘‘glm.fit: fitted probabilities numerically 0 or 1 occurred’’
```

```
7
```

```
Warning message:
‘‘glm.fit: fitted probabilities numerically 0 or 1 occurred’’
```

```
8
```

```
Warning message:
‘‘glm.fit: fitted probabilities numerically 0 or 1 occurred’’
```

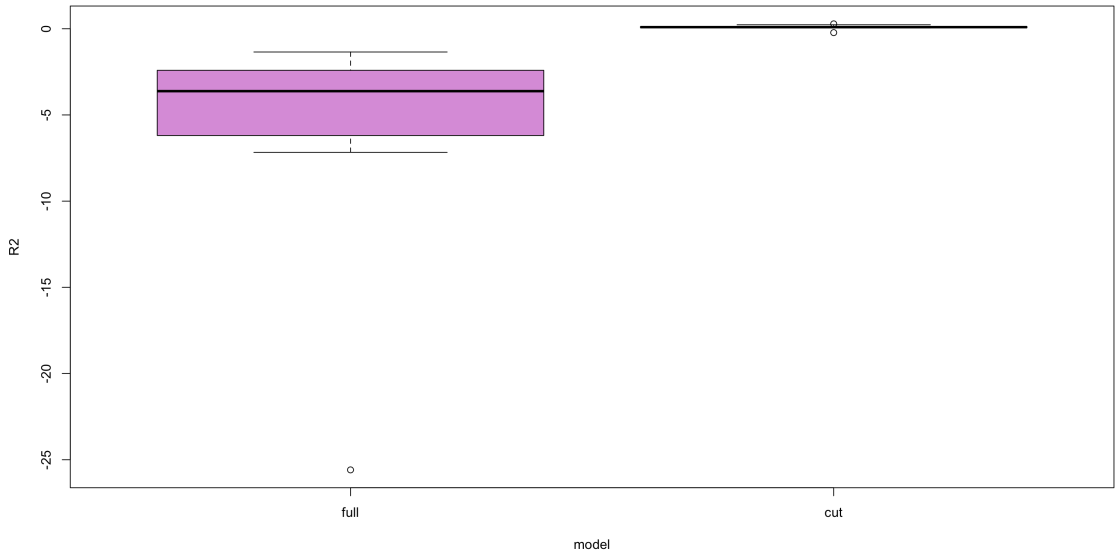
```
9
```

```
Warning message:
‘‘glm.fit: fitted probabilities numerically 0 or 1 occurred’’
```

```
10
```

Plotting the distribution of R2 over the folds for the two models

[12]:
```
par(mai=c(.9,.9,.1,.1))
boxplot(OOS, col="plum", ylab="R2", xlab="model", bty="n")
```



Of just check average R2:

[13]:
```
colMeans(OOS)
```

full -6.03034906613343 **cut** 0.0893384626517727

2.1 Forward stepwise

Null model:

[14]:
```
null <- glm(FAIL~1, data=SC)
```

Stepwise, also with time elapsed logged:

[15]:
```
system.time(fwd <- step(null, scope=formula(full), dir="forward"))
length(coef(fwd)) # chooses around 70 coef
```

```
Start:  AIC=115.03
FAIL ~ 1

         Df Deviance     AIC
+ SIG2    1   91.699  92.588
+ SIG189  1   92.132  99.534
+ SIG13   1   92.327 102.664
+ SIG166  1   92.406 103.923
+ SIG61   1   92.463 104.832
+ SIG173  1   92.588 106.832
+ SIG50   1   92.619 107.323
+ SIG167  1   92.743 109.302
+ SIG24   1   92.793 110.103
```

```
+ SIG19   1   74.555 -77.122
+ SIG121  1   74.555 -77.122
+ SIG87   1   74.555 -77.122
+ SIG70   1   74.555 -77.121
+ SIG158  1   74.555 -77.121
+ SIG123  1   74.555 -77.121
```

```
   user  system elapsed
 75.984   1.486 156.811
```

69

Use Taddy's library for LASSO

[16]:
```
install.packages("gamlr")
library(gamlr)
```

```
The downloaded binary packages are in
/var/folders/dk/2_0472cd7h35shgpvb9y6g6xp8980h/T//RtmprBsVEf/downloaded_packages
```

```
Loading required package: Matrix
```

Note we'll need to define our own design matrices here. This is extra work, but worth it because sparse matrices are way more efficient for many cases, and specifically, the data often comes in triplets exactly like here:

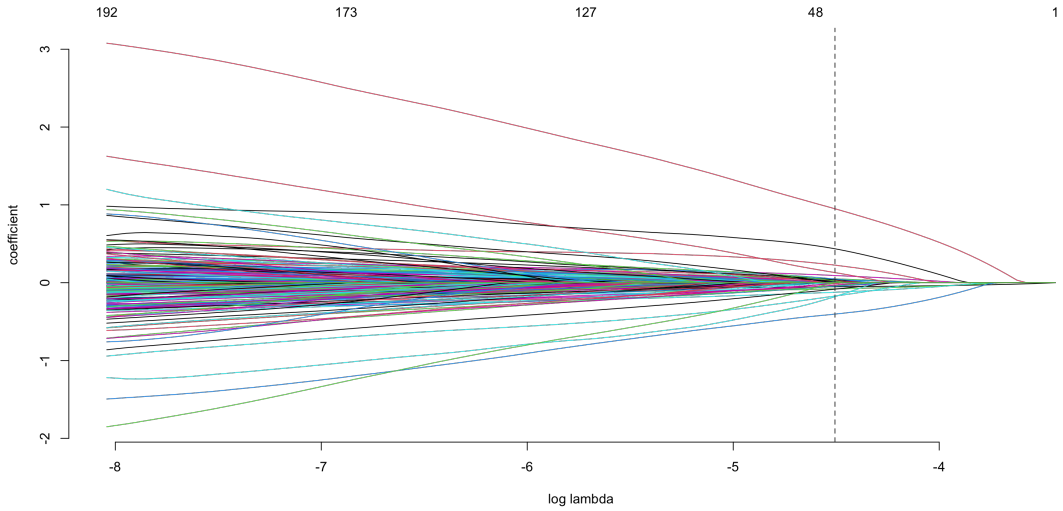
[17]:
```
scx <- sparse.model.matrix(FAIL ~ ., data=SC)[,-1]
```

After our X matrix, it is often also convenient to have the outcome y as a vector of its own:

[18]:
```
scy <- SC$FAIL
```

Fitting a LASSO, and having a look at the path:

[19]:
```
sclasso <- gamlr(scx, scy, family="binomial")
plot(sclasso) # the ubiquitous path plot
```



Note that this plot is not in the book, by then we switched to browser data! Let's finish up with the semiconductor example first before we follow the book along.

One (fast) way to pick a lambda: the AICc parameter

[20]:
```
scbeta <- coef(sclasso)
log(sclasso$lambda[which.min(AICc(sclasso))])
sum(scbeta!=0) # chooses 30 (+intercept) @ log(lambda) = -4.5
```

seg24: -4.50608019637309

31

See how conservative the BIC can get:

[21]:
```
BICseg <- which.min(BIC(sclasso))
scb.bic <- coef(sclasso, s=BICseg)
sum(scb.bic!=0)
```

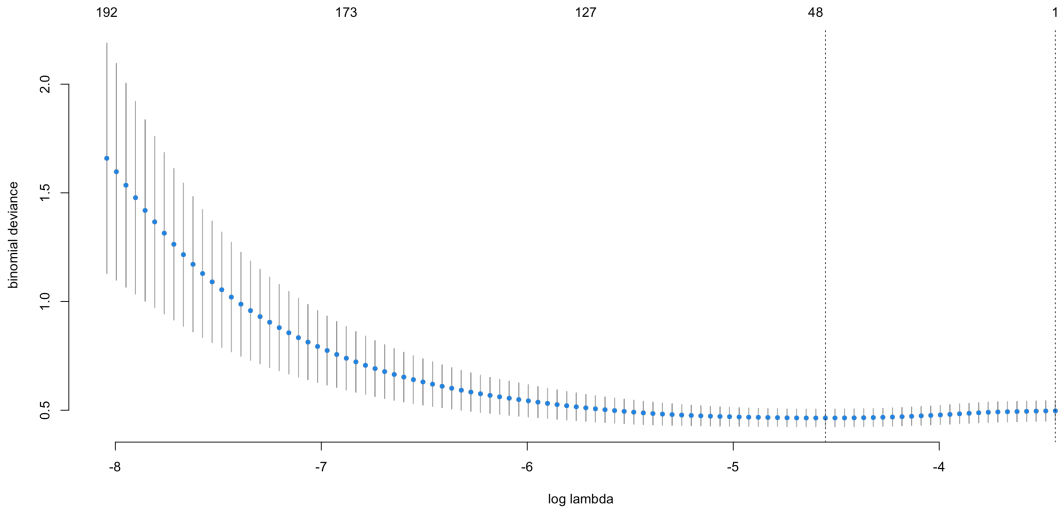
1

We are just left with the intercept!

Of course, we just learned about the wonderful principle of cross-validation, so let's try that too:

[22]:
```
sccvl <- cv.gamlr(scx, scy, family="binomial", verb=TRUE)
plot(sccvl, bty="n")
```

```
fold 1,2,3,4,5,done.
```



Let's select the lambda that minimizes deviance here:

[23]:
```
scb.min <- coef(sccvl, select="min")
log(sccvl$lambda.min)
sum(scb.min!=0)
```

-4.55259706693863

40

Note that I got a different answer than Taddy here. There is Monte Carlo uncertainty/noise in the cross-validation, in the sense that the exact cuts are random, i.e. which observation goes into which fold. As the deviance was pretty flat around the minimal lambda, even some small noise could mean that we got a log lambda of 5, not 4.8. That also means more coefficients.

If we go with the default of the gamlr package, which wanted to respect the default in the glmnet package, you get the biggest lambda (smallest in absolute value) with average out-of-sample deviance no more than one standard deviation away from the minimum.

[24]:
```
scb.1se <- coef(sccvl)
log(sccvl$lambda.1se)
sum(scb.1se!=0)
```

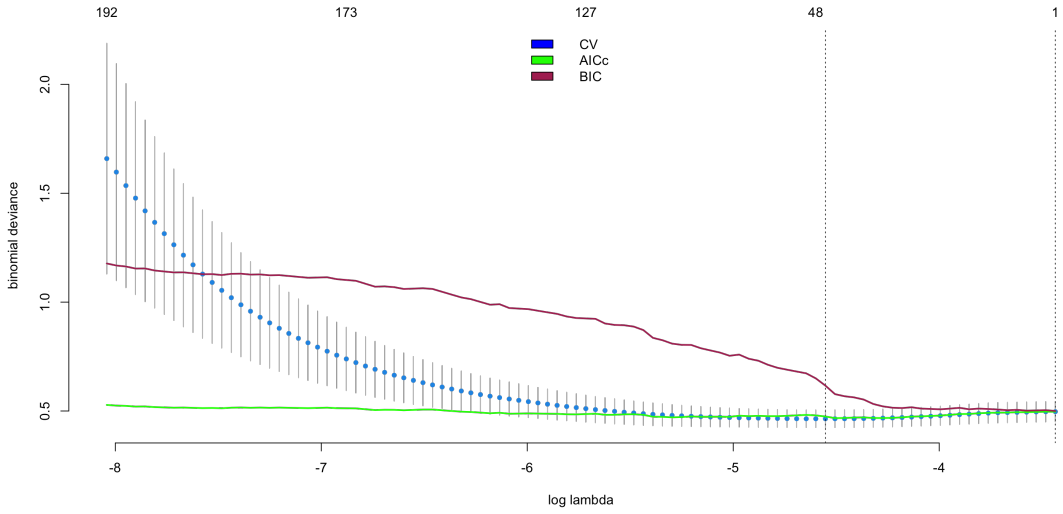
-3.43619217336576

1

Again, this lambda is conservative enough, that you're left with only the intercept in the model.

Luckily, we also collected the information criteria for all the models we ran for all these possible lambdas, and we can check Taddy's promise that AICc is close to the cross-validated deviance, at least where it matters (around the minimum):

[25]:
```
plot(sccvl, bty="n")
lines(log(sclasso$lambda),AICc(sclasso)/n, col="green", lwd=2)
lines(log(sclasso$lambda),BIC(sclasso)/n, col="maroon", lwd=2)
legend("top", fill=c("blue","green","maroon"),
       legend=c("CV","AICc","BIC"), bty="n")
```



We can just check quickly the information criteria for the two models we considered originally, motivated by a Benjamini-Hochberg FDR cut:

[26]:
```
BIC(full)
BIC(cut)

AIC(full)
AIC(cut)

AICc(full)
AICc(cut)
```

1787.18350165154

782.714783678427

722.332076862736

650.270576615143

786.021488627442

651.166510453874

Note that the cut model has 25 variables. For the lasso, some of these criteria suggested to keep only the intercept, or have 30 variables. But none of them would prefer the full model to the cut

model.

2.2 Lasso in the browser data

First let's setup the browser and spending datasets. The browsing history table has three colums: [machine] id, site [id], [# of] visits

[27]:
```
web <- read.csv("browser-domains.csv")
```

Read in the actual website names and relabel site factor and also factor machine id.

[28]:
```
sitenames <- scan("browser-sites.txt", what="character")
web$site <- factor(web$site, levels=1:length(sitenames), labels=sitenames)
web$id <- factor(web$id, levels=1:length(unique(web$id)))
```

Calculate total visits per machine and percent of time on each site. The handy function tapply(a,b,c) does c(a) for every level of factor b.

[29]:
```
machinetotals <- as.vector(tapply(web$visits,web$id,sum))
visitpercent <- 100*web$visits/machinetotals[web$id]
```

This goes into a sparse matrix. (Try to pick this up, this is very useful.)

[30]:
```
xweb <- sparseMatrix(
        i=as.numeric(web$id), j=as.numeric(web$site), x=visitpercent,
        dims=c(nlevels(web$id),nlevels(web$site)),
        dimnames=list(id=levels(web$id), site=levels(web$site)))
```

As a sanity check, see what sites household #1 visited

[31]:
```
head(xweb[1, xweb[1,]!=0])
```

atdmt.com 4.0520260130065 **yahoo.com** 11.855927963982 **msn.com** 0.250125062531266 **google.com** 6.52826413206603 **aol.com** 0.150075037518759 **questionmarket.com** 1.35067533766883

Read in the spending data, with the first column as the (here: trivial) names for the rows. Also transform it immediately to matrix.

[32]:
```
yspend <- read.csv("browser-totalspend.csv", row.names=1)
yspend <- as.matrix(yspend)
```

Run my lasso models with different penalty parameters (lambda), and plot this path.

[ ]:
```
spender <- gamlr(xweb, log(yspend), verb=TRUE)
plot(spender)
```

```
*** n=10000 observations and p=1000 covariates ***
segment 1: lambda = 0.2325, dev = 2.783e+04, npass = 0
segment 2: lambda = 0.2219, dev = 2.778e+04, npass = 3
```

```
segment 99: lambda = 0.002435, dev = 2.089e+04, npass = 5
segment 100: lambda = 0.002325, dev = 2.088e+04, npass = 5
```

We can also run many regressions stepwise and plot coefficients analogously.

[34]:
```
stepspend <- gamlr(xweb, log(yspend), verb=TRUE, gamma=Inf, lmr=.1)
par(mai=c(.9,.9,.1,.1))
plot(stepspend, df=FALSE, select=FALSE)
```

```
*** n=10000 observations and p=1000 covariates ***
segment 1: lambda = 0.2325, dev = 2.783e+04, npass = 0
segment 2: lambda = 0.2271, dev = 2.78e+04, npass = 3
segment 3: lambda = 0.2219, dev = 2.729e+04, npass = 3
segment 4: lambda = 0.2168, dev = 2.729e+04, npass = 1
segment 5: lambda = 0.2118, dev = 2.729e+04, npass = 1
segment 6: lambda = 0.207, dev = 2.729e+04, npass = 1
segment 7: lambda = 0.2022, dev = 2.729e+04, npass = 1
segment 8: lambda = 0.1975, dev = 2.729e+04, npass = 1
segment 9: lambda = 0.193, dev = 2.729e+04, npass = 1
segment 10: lambda = 0.1886, dev = 2.729e+04, npass = 1
segment 11: lambda = 0.1842, dev = 2.729e+04, npass = 1
segment 12: lambda = 0.18, dev = 2.728e+04, npass = 3
segment 13: lambda = 0.1759, dev = 2.696e+04, npass = 4
segment 14: lambda = 0.1718, dev = 2.696e+04, npass = 1
segment 15: lambda = 0.1679, dev = 2.666e+04, npass = 4
segment 16: lambda = 0.164, dev = 2.666e+04, npass = 1
segment 17: lambda = 0.1602, dev = 2.666e+04, npass = 1
segment 18: lambda = 0.1566, dev = 2.666e+04, npass = 1
segment 19: lambda = 0.153, dev = 2.666e+04, npass = 1
segment 20: lambda = 0.1494, dev = 2.666e+04, npass = 1
segment 21: lambda = 0.146, dev = 2.666e+04, npass = 1
segment 22: lambda = 0.1426, dev = 2.666e+04, npass = 1
segment 23: lambda = 0.1394, dev = 2.666e+04, npass = 1
segment 24: lambda = 0.1362, dev = 2.646e+04, npass = 4
segment 25: lambda = 0.133, dev = 2.646e+04, npass = 1
segment 26: lambda = 0.13, dev = 2.646e+04, npass = 1
segment 27: lambda = 0.127, dev = 2.646e+04, npass = 1
segment 28: lambda = 0.1241, dev = 2.646e+04, npass = 3
segment 29: lambda = 0.1212, dev = 2.63e+04, npass = 5
segment 30: lambda = 0.1184, dev = 2.615e+04, npass = 4
segment 31: lambda = 0.1157, dev = 2.615e+04, npass = 1
segment 32: lambda = 0.113, dev = 2.615e+04, npass = 3
segment 33: lambda = 0.1104, dev = 2.589e+04, npass = 4
segment 34: lambda = 0.1079, dev = 2.589e+04, npass = 1
segment 35: lambda = 0.1054, dev = 2.589e+04, npass = 1
segment 36: lambda = 0.103, dev = 2.589e+04, npass = 3
segment 37: lambda = 0.1006, dev = 2.578e+04, npass = 5
segment 38: lambda = 0.09832, dev = 2.556e+04, npass = 6
```

```
segment 87: lambda = 0.03146, dev = 2.282e+04, npass = 7
segment 88: lambda = 0.03073, dev = 2.277e+04, npass = 5
segment 89: lambda = 0.03003, dev = 2.274e+04, npass = 5
segment 90: lambda = 0.02934, dev = 2.268e+04, npass = 9
segment 91: lambda = 0.02866, dev = 2.257e+04, npass = 10
segment 92: lambda = 0.028, dev = 2.25e+04, npass = 6
segment 93: lambda = 0.02736, dev = 2.249e+04, npass = 4
segment 94: lambda = 0.02673, dev = 2.245e+04, npass = 7
segment 95: lambda = 0.02611, dev = 2.238e+04, npass = 7
segment 96: lambda = 0.02551, dev = 2.232e+04, npass = 6
segment 97: lambda = 0.02493, dev = 2.228e+04, npass = 5
segment 98: lambda = 0.02435, dev = 2.225e+04, npass = 6
segment 99: lambda = 0.02379, dev = 2.222e+04, npass = 7
segment 100: lambda = 0.02325, dev = 2.218e+04, npass = 5
```

Notice that the stepwise coefficients jump around (as you go from right to left and more and more variables are added to your model). This is why this method is very fragile. The lasso plot above is much smoother.

So go back to that, and check the coefficients it would select (by default, it will give you the model picked by the AICc criterion). You can drop the intercept and check which websites predict low and high spending:

[35]:
```
B <- coef(spender)
B <- B[-1,]
B[which.min(B)]
B[which.max(B)]
```

cursormania.com: -0.998142951092763

shopyourbargain.com: 1.29424564074213

We can repeat the same exercise with Bayes's Information Criterion.

[36]:
```
coef(spender, select=which.min(BIC(spender)))
```

```
1001 x 1 sparse Matrix of class "dgCMatrix"
                                      seg31
intercept                       5.929791309
atdmt.com                       .
yahoo.com                       .
whenu.com                       .
weatherbug.com                  .
msn.com                         .
google.com                      .
aol.com                         .
questionmarket.com              0.013756425
googlesyndication.com-o02       .
casalemedia.com                 .
mywebsearch.com                 .
```

```
webkinz.com                    .
dealerconnection.com           .
streamaudio.com                .
grantmedia.com                 .
home123info.com                .
exittracking.com               .
worldsex.com                   .
yfdmedia.com                   .
automotive.com                 .
cursormania.com                .
tradedoubler.com               .
bedbathandbeyond.com           0.093914776
equifax.com                    .
hotornot.com                   .
falkag.de                      .
chicagotribune.com             .
airtran.com                    .
thebreastcancersite.com        .
charmingshoppes.com            .
ugo.com                        .
cox.com                        .
spicymint.com                  .
real.com-o01                   .
targetnet.com                  .
effectivebrand.com             .
dallascowboys.com              .
leadgenetwork.com              .
in.us                          .
vistaprint.com                 .
```

Or eyeball the AIC criteria for the various models:

[37]:
```
AIC(spender)
```

seg1 10236.6778380035 **seg2** 10221.4099122476 **seg3** 10205.6500168242 **seg4** 10191.2685292531
seg5 10178.1466207609 **seg6** 10159.7571235249 **seg7** 10136.4973218469 **seg8** 10107.9092724091
seg9 10072.5138639634 **seg10** 10028.4113861962 **seg11** 9986.66692821009 **seg12** 9945.58526973275
seg13 9908.7850752273 **seg14** 9873.4714661598 **seg15** 9829.16897111623 **seg16** 9782.72838361129
seg17 9740.22556220716 **seg18** 9703.93966640551 **seg19** 9661.84150030097 **seg20** 9624.06091166597
seg21 9584.78998750367 **seg22** 9551.02023144592 **seg23** 9517.20791694037 **seg24** 9477.32111108323
seg25 9447.79117026423 **seg26** 9412.07746693157 **seg27** 9375.1609337076 **seg28** 9349.05799243475
seg29 9315.9221803998 **seg30** 9274.98245978843 **seg31** 9242.221101539 **seg32** 9211.88820024744
seg33 9179.43313622812 **seg34** 9159.19479178462 **seg35** 9131.73869268208 **seg36** 9097.18738444891
seg37 9068.1823936803 **seg38** 9036.08434406966 **seg39** 9005.60247808104 **seg40** 8976.58111343056
seg41 8949.77014866577 **seg42** 8933.62028778718 **seg43** 8905.67771668071 **seg44** 8875.5077538076
seg45 8855.90270972236 **seg46** 8838.81217541375 **seg47** 8821.15540511281 **seg48** 8803.88575769943
seg49 8790.8930037326 **seg50** 8790.03173102287 **seg51** 8776.22664674342 **seg52** 8761.74471902535
seg53 8768.03135845668 **seg54** 8764.38957513671 **seg55** 8763.19172589733 **seg56** 8760.20861396519

seg57 8759.60385383997 **seg58** 8759.90475374427 **seg59** 8754.8048402195 **seg60** 8762.72744644017
seg61 8761.38977974488 **seg62** 8751.36277807511 **seg63** 8760.78900987018 **seg64** 8759.34098170299
seg65 8770.65362829039 **seg66** 8774.57809905817 **seg67** 8777.31219522373 **seg68** 8779.04408907606
seg69 8771.70877962425 **seg70** 8784.51394622799 **seg71** 8789.98078333818 **seg72** 8796.80439405428
seg73 8800.7614965544 **seg74** 8811.89219608988 **seg75** 8827.85603721927 **seg76** 8840.34130336324
seg77 8863.46556138292 **seg78** 8863.41887017837 **seg79** 8879.6811805437 **seg80** 8890.35034798351
seg81 8903.33853473553 **seg82** 8917.42234780525 **seg83** 8926.26311652674 **seg84** 8927.85557976219
seg85 8934.39553032519 **seg86** 8939.74819468554 **seg87** 8947.8369748867 **seg88** 8964.44141095861
seg89 8981.55993355088 **seg90** 8999.05383809619 **seg91** 9018.8482724595 **seg92** 9029.07962930815
seg93 9043.9010595961 **seg94** 9050.89724995585 **seg95** 9050.33837228859 **seg96** 9066.14131072219
seg97 9076.30142572639 **seg98** 9083.09093891837 **seg99** 9102.08245687615 **seg100** 9109.33038790145

You can again do cross-validation to select a model along the lasso path. By default, the package would use the 1se rule again, but you can again just go for the minimum.

Let's compare how to website's coefficients change across the models favored by different criteria. (Not surprisingly, 1se is more conservative with fewer variables included.)

[38]:
```
cv.spender <- cv.gamlr(xweb, log(yspend), verb=TRUE)
beta1se <- coef(cv.spender)
betamin <- coef(cv.spender, select="min")
cbind(beta1se,betamin)[c("tvguide.com","americanexpress.com"),]
```

```
fold 1,2,3,4,5,done.
```

```
2 x 2 sparse Matrix of class "dgCMatrix"
                          seg29      seg48
tvguide.com         .           .
americanexpress.com 0.009602647 0.04897335
```

You collect the deviance and the coefficients across these models (over a range of lambdas) next to each other.

[ ]:
```
par(mfrow=c(1,2))
plot(cv.spender)
plot(cv.spender$gamlr)
```

We can check which lambdas do the various criteria prefer:

As the chapter "promised," 1SE is conservative, but not as much as BIC, and AICc is close to deviance-minimizing value in cross-validation.

[40]:
```
log(spender$lambda[which.min(AICc(spender))])
log(spender$lambda[which.min(AIC(spender))])
log(spender$lambda[which.min(BIC(spender))])
log(cv.spender$lambda.min)
log(cv.spender$lambda.1se)
```

seg52: -3.83132828797401

seg62: -4.29649699362937

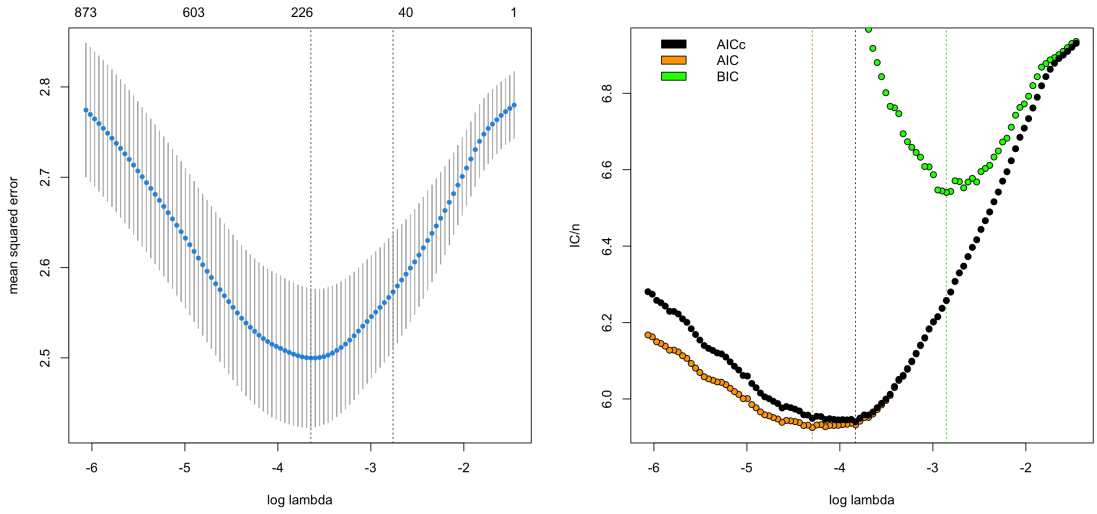
seg31: -2.85447400609775

-3.64526080571186

-2.76144026496667

We can also plot the deviance and the values of various information criteria over these models (which all differ by their lambda). To put both on the same scale, we should divide the criteria by the number of observations. Try reversing engineering which lines added vertical lines to the plots, and which added further data points.

[41]:
```
ll <- log(spender$lambda) ## the sequence of lambdas
par(mfrow=c(1,2))
plot(cv.spender)
plot(ll, AIC(spender)/n,
        xlab="log lambda", ylab="IC/n", pch=21, bg="orange")
abline(v=ll[which.min(AIC(spender))], col="orange", lty=3)
abline(v=ll[which.min(BIC(spender))], col="green", lty=3)
abline(v=ll[which.min(AICc(spender))], col="black", lty=3)
points(ll, BIC(spender)/n, pch=21, bg="green")
points(ll, AICc(spender)/n, pch=21, bg="black")
legend("topleft", bty="n",
        fill=c("black","orange","green"),legend=c("AICc","AIC","BIC"))
```



We can also mark in the path plot which model (value of the penalty parameter lambda) the various methods (arguments) would prefer most.

[ ]:
```
plot(spender, col="grey")
abline(v=ll[which.min(AICc(spender))], col="black", lty=2)
```

```
abline(v=ll[which.min(AIC(spender))], col="orange", lty=2)
abline(v=ll[which.min(BIC(spender))], col="green", lty=2)
abline(v=log(cv.spender$lambda.min), col="blue", lty=2)
abline(v=log(cv.spender$lambda.1se), col="purple", lty=2)
legend("topright", bty="n", lwd=1,
       col=c("black","orange","green","blue","purple"),
       legend=c("AICc","AIC","BIC","CV.min","CV.1se"))
```

2.3 On to the orange juice

As always, we need to read in the data first.

[43]:
```
oj <- read.csv("oj.csv")
```

This is a cool thing to know about lasso models, easy to miss if you saw conventional econometrics before: It is better to include dummy variables for all categories of a factor, and not leave out any. The following functions will make the missing value, NA, be the reference value in R, so all "real" values will be included.

[44]:
```
xnaref <- function(x){
        if(is.factor(x))
                if(!is.na(levels(x)[1]))
                        x <- factor(x,levels=c(NA,levels(x)),exclude=NULL)
        return(x) }

naref <- function(DF){
        if(is.null(dim(DF))) return(xnaref(DF))
        if(!is.data.frame(DF))
                stop("You need to give me a data.frame or a factor")
        DF <- lapply(DF, xnaref)
        return(as.data.frame(DF))
}
```

Now we can build the design matrix for all variables. We can make sure we like what we see by checking three rows for example:

[45]:
```
oj$brand <- naref(oj$brand)
xbrand <- sparse.model.matrix( ~ brand, data=oj)
xbrand[c(100,200,300),]
oj$brand[c(100,200,300)]
```

```
3 x 3 sparse Matrix of class "dgCMatrix"
    (Intercept) brandminute.maid brandtropicana
100           1                .              1
200           1                1              .
300           1                .              .
```

1. 'tropicana' 2. 'minute.maid' 3. 'dominicks'

2.4 Hockey

This is the (very) rare case where you don't start by loading the dataset from a separate data file of your own, as we will use the example data set that came with the software library.

[46]:
```
data(hockey)
```

Let's familiarize ourselves with the data we see. `Goal` is a key dataframe, and there are also separate sparse matrices `player` where players playing together at the time of a goal get the same indicator (1s for the home team, -1 for away).

[47]:
```
head(goal)
player[1:3,2:7]
```

A data.frame: 6 × 7

	homegoal <dbl>	season <chr>	team.away <chr>	team.home <chr>	period <int>	differential <int>	playoffs <dbl>
1	0	20022003	DAL	EDM	1	0	0
2	0	20022003	DAL	EDM	1	-1	0
3	1	20022003	DAL	EDM	2	-2	0
4	0	20022003	DAL	EDM	2	-1	0
5	1	20022003	DAL	EDM	3	-2	0
6	1	20022003	DAL	EDM	3	-1	0

```
3 x 6 sparse Matrix of class "dgCMatrix"
     ERIC_BREWER ANSON_CARTER JASON_CHIMERA MIKE_COMRIE ULF_DAHLEN ROB_DIMAIO
[1,]           1            .             1           .          .         -1
[2,]           .            1             .           1         -1          .
[3,]           .            1             .           1          .         -1
```

We will be build the design matrix from three separate [sparse] matrices.

[48]:
```
x <- cbind(config,team,player)
```

We also pick a binary outcome: whether the goal was a goal for the home team.

[49]:
```
y <- goal$homegoal
```

For a lasso model without cross-validation (so information criteria only), we can use the basic command, only paying attention here not to shrink [penalize] the dummies for team configurations and team-seasons.

[50]:
```
nhlreg <- gamlr(x, y, verb=TRUE,
        free=1:(ncol(config)+ncol(team)),
        family="binomial", standardize=FALSE)
```

```
*** n=69449 observations and p=2776 covariates ***
segment 1: lambda = 0.001289, dev = 8.096e+04, npass = 292
segment 2: lambda = 0.001231, dev = 8.095e+04, npass = 3
segment 3: lambda = 0.001175, dev = 8.094e+04, npass = 3
segment 4: lambda = 0.001121, dev = 8.094e+04, npass = 3
segment 5: lambda = 0.00107, dev = 8.093e+04, npass = 4
```

Let's work with the coefficients for players in the model (with a specific lambda) the default AICc selection would pick.

[51]:
```
Baicc <- coef(nhlreg)[colnames(player),]
```

What do we learn from the intercept? This is the effect on odds that a goal is home rather than away, regardless of any information about what teams are playing or who is on ice. This is the home ice advantage. We find that home-ice increases odds you've scored by 8%.

[52]:
```
exp(coef(nhlreg)[1])
```

1.08235434370196

Next we can look at the player effects, first of all how many are significantly different from zero. Be careful, here "significant" does not mean "statistically significant" in a hypothesis testing sense. Only that the regularization method of lasso left us with coefficient different from zero for this player. The AICc-preferred regression finds 646 significant player effects.

[53]:
```
sum(Baicc!=0)
```

646

Here is a smart way to look at the 10 largest coefficients, which here correspond to the top 10 players.

[54]:
```
Baicc[order(Baicc, decreasing=TRUE)[1:10]]
```

PETER_FORSBERG	0.754825396377924	**TYLER_TOFFOLI**	0.629257724204375
ONDREJ_PALAT	0.628404016979746	**ZIGMUND_PALFFY**	0.442699707228187
SIDNEY_CROSBY	0.413117430590147	**JOE_THORNTON**	0.383763161152591
PAVEL_DATSYUK	0.376198105861326	**LOGAN_COUTURE**	0.368210275928359
ERIC_FEHR	0.367728287603729	**MARTIN_GELINAS**	0.357761285251082

And the worst:

[55]:
```
Baicc[order(Baicc)[1:10]]
```

TIM_TAYLOR	-0.864321442159657	**JOHN_MCCARTHY**	-0.565188583104681 **P.**
J._AXELSSON	-0.428381072743795	**NICLAS_HAVELID**	-0.385458319628831
THOMAS_POCK	-0.384412819094146	**MATHIEU_BIRON**	-0.351210053668418
CHRIS_DINGMAN	-0.334224250311527	**DARROLL_POWE**	-0.333990643063014
RAITIS_IVANANS	-0.312948053218133	**RYAN_HOLLWEG**	-0.298876863973549

As in the book, we can practice our intuition with an example. For instance, the model says that whenever a goal is scored, Pittsburgh's odds of having scored (rather than scored on) increase by 51% if Sidney Crosby is on the ice.

[56]:
```
exp(Baicc["SIDNEY_CROSBY"])
```

SIDNEY_CROSBY: 1.51152251449339

And the Blue Jackets (or Kings, pre 2011-12) odds of having scored drop by around 22% if Jack Johnson is on the ice.

[57]:
```
exp(Baicc["JACK_JOHNSON"])
```

JACK_JOHNSON: 0.781348817454316

Please pay attention to the discussion why we did not standardize the variables in this special case. We did not do that because all variables were 0-or-1 here.

Without `standardize=FALSE`, we'd be multiplying the penalty for each coefficient (player effect) by that player's standard deviation in onice. For a binary variable, the variance is just the product of the probability of occurence and its complement, so p(1-p). The standard deviation is the square root of this.

The players with big SD in onice are guys who play a lot. Players with small SD are those who play little (almost all zeros).

So weighting penalty by SD in this case is exactly what you don't want: a bigger penalty for people with many minutes on ice, a smaller penalty for those who seldom play. Indeed, running the regression without `standardize=FALSE` leads to a bunch of farm teamers coming up tops. Hockey fans might check this by looking at these names.

[58]:
```
nhlreg.std <-  gamlr(x, y,
        free=1:(ncol(config)+ncol(team)), family="binomial")
Bstd <- coef(nhlreg.std)[colnames(player),]
Bstd[order(Bstd, decreasing=TRUE)[1:10]]
```

JEFF_TOMS 1.73807057093868 **RYAN_KRAFT** 1.48264190785706 **COLE_JARRETT** 1.21193180029381 **TOMAS_POPPERLE** 1.1107806009022 **DAVID_LIFFITON** 1.09748720705051 **ALEXEY_MARCHENKO** 1.0297324086158 **ERIC_SELLECK** 1.00600145560387 **MIKE_MURPHY** 0.960093892487883 **DAVID_GOVE** 0.926489469569342 **TOMAS_KANA** 0.879280162065019

Let's do the proper cross-validation here (with default settings but `standardize`)

[59]:
```
cv.nhlreg <- cv.gamlr(x, y,
        free=1:(ncol(config)+ncol(team)),
        family="binomial", verb=TRUE, standardize=FALSE)
```

```
fold 1,2,3,4,5,done.
```

Again, let's plot the deviance and the lasso path over lambda values together (these are saved into the `cv.nhlreg` thing already):

[ ]:
```
par(mfrow=c(1,2))
plot(cv.nhlreg)
plot(cv.nhlreg$gamlr)
```

We can check the log lambdas selected under various criteria. AIC and AICc give exactly the same answer here (because the number of observation is much, much larger than the number of estimated parameters) and both are close to the cross-validated minimizing answer. This is not

surprising, as AIC and AICc both try to approximate out-of-sample deviance (mean-squared error here).

[61]:
```
log(nhlreg$lambda[which.min(AICc(nhlreg))])
log(nhlreg$lambda[which.min(AIC(nhlreg))])
log(nhlreg$lambda[which.min(BIC(nhlreg))])
log(cv.nhlreg$lambda.min)
log(cv.nhlreg$lambda.1se)
```

seg55: -9.1655549833554

seg55: -9.1655549833554

seg1: -6.65364397281645

-9.07252124222433

-8.04915008978253

Not surprisingly, we also see for coefficients that the cross-validated minimizing model gives very similar answers to the AICc or AIC-minimizing ones above. So around 600 players having scores different from zero, Forsberg being the best one.

[62]:
```
Bcvmin <- coef(cv.nhlreg, select="min")[colnames(player),]
sum(Bcvmin!=0)
sort(Bcvmin,decreasing=TRUE)[1:10]
```

601

PETER_FORSBERG	0.747145004183073	**ONDREJ_PALAT**	0.584697858854544
TYLER_TOFFOLI	0.579515236167802	**ZIGMUND_PALFFY**	0.409819472567313
SIDNEY_CROSBY	0.406134445524612	**JOE_THORNTON**	0.378755399377023
PAVEL_DATSYUK	0.365972174822703	**LOGAN_COUTURE**	0.357190687550205
ERIC_FEHR	0.354998563878224	**MATT_MOULSON**	0.346883251811761

The 1SE rule accounts for uncertainty about out-of-sample error (the noise between the folds, also reflecting Monte Carlo noise), and thus chooses a simpler model. See that a not much larger log lambda led to half as many coefficients different from zero. Surprisingly or not, even the top ten changes a little.

[63]:
```
Bcv1se <- coef(cv.nhlreg)[colnames(player),]
sum(Bcv1se!=0)
sort(Bcv1se,decreasing=TRUE)[1:10]
```

176

PETER_FORSBERG	0.57841233658444	**SIDNEY_CROSBY**	0.313019862086442
JOE_THORNTON	0.279390960335059	**PAVEL_DATSYUK**	0.256320208148928
LUBOMIR_VISNOVSKY	0.252972839673257	**ALEX_OVECHKIN**	0.245034229988879
ALEXANDER_SEMIN	0.238214680372767	**RYAN_GETZLAF**	0.218667690302885
HENRIK_SEDIN	0.209378193394701	**MARIAN_GABORIK**	0.209259344879027

BIC is way, way more conservative than all these options. It actually goes with saying nobody is different from average.

[64]:
```
Bbic <- coef(nhlreg,select=which.min(BIC(nhlreg)))[colnames(player),]
sum(Bbic!=0)
```

0

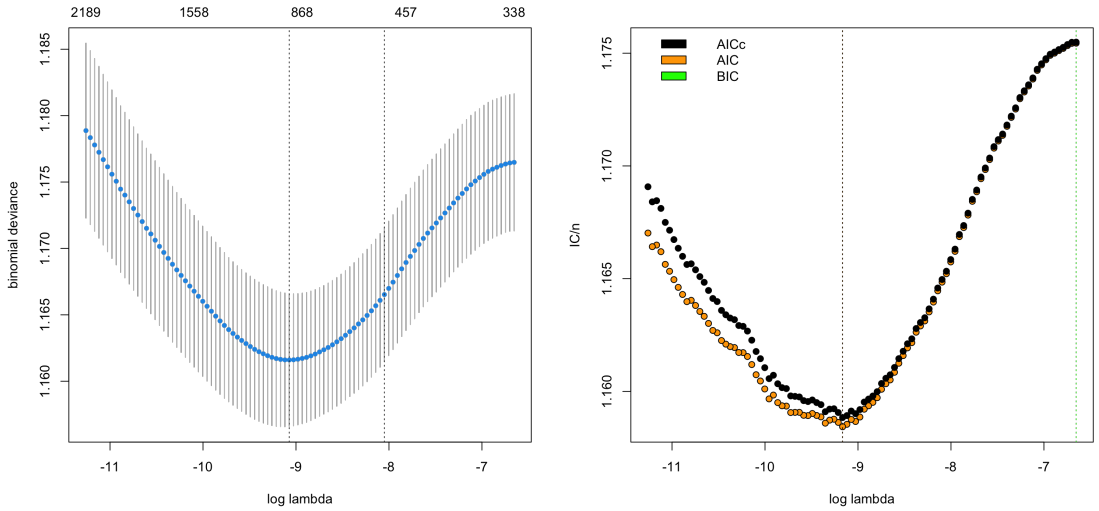
The BIC is trying to find lambda with the highest probability of having the minimum out-of-sample error, which is subtly different than finding the lambda corresponding to lowest expected OOS error. For example, if there is more uncertainty about OOS error at the lambda with minimal expectation, then it could be that another value with higher expected error but lower uncertainty around this value will have a higher probability of being best. In this case, the BIC says there is much uncertainty at everything other than the null model, so that the null model ends up highest probability of being best.

As an aside: note that the null model here is not just an intercept, but rather includes onice configuration information along with information about the team and season. So the BIC is not saying that no players matter, but rather that it cannot confidently tell them apart from their team's average level of play in a given season.

Finally, check out some plots to compare model selections.

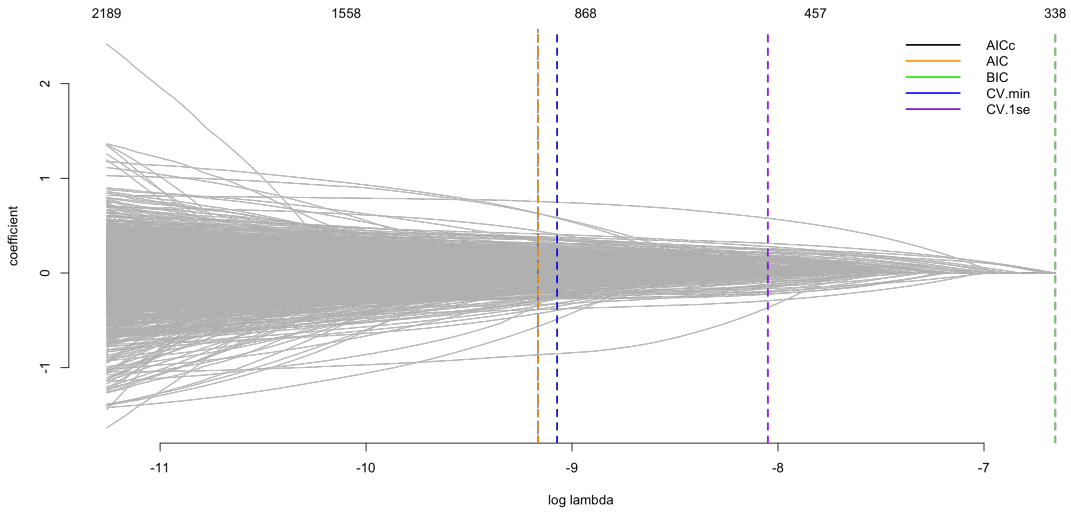
First the sequence of lambdas:

[65]:
```
ll <- log(nhlreg$lambda)
n <- nrow(goal)
par(mfrow=c(1,2))
plot(cv.nhlreg)
plot(ll, AIC(nhlreg)/n,
        xlab="log lambda", ylab="IC/n", pch=21, bg="orange")
abline(v=ll[which.min(AIC(nhlreg))], col="orange", lty=3)
abline(v=ll[which.min(BIC(nhlreg))], col="green", lty=3)
abline(v=ll[which.min(AICc(nhlreg))], col="black", lty=3)
points(ll, BIC(nhlreg)/n, pch=21, bg="green")
points(ll, AICc(nhlreg)/n, pch=21, bg="black")
legend("topleft", bty="n",
        fill=c("black","orange","green"),legend=c("AICc","AIC","BIC"))
```



Then all these criteria's preferred models/lambdas along the path:

[66]:
```
par(mfrow=c(1,1))
plot(nhlreg, col="grey")
abline(v=ll[which.min(AICc(nhlreg))], col="black", lty=2, lwd=2)
abline(v=ll[which.min(AIC(nhlreg))], col="orange", lty=2, lwd=2)
abline(v=ll[which.min(BIC(nhlreg))], col="green", lty=2, lwd=2)
abline(v=log(cv.nhlreg$lambda.min), col="blue", lty=2, lwd=2)
abline(v=log(cv.nhlreg$lambda.1se), col="purple", lty=2, lwd=2)
legend("topright", bty="n", lwd=2,
       col=c("black","orange","green","blue","purple"),
       legend=c("AICc","AIC","BIC","CV.min","CV.1se"))
```



2.5 Advanced Topic: Uncertainty Quantification for the Lasso

We'll need the coefficients from the AICc-preferred lasso model from above.

[67]:
```
Bhat <- coef(nhlreg)
```

2.5.1 A parametric bootstrap

Start with the fitted values of the model we picked here (which came 61st on the lasso path).

[68]:
```
Qlowpen <- drop(predict(nhlreg, x, select=61, type="response"))
```

Then set up a small bootstrap (each iteration will take time to rerun the lasso and pick the AICc-preferred model).

[69]:
```
Bparboot <- sparseMatrix(dims=c(nrow(Bhat),0),i={},j={})
B <- 100
for(b in 1:B){
  yb <- rbinom(nrow(x), Qlowpen, size=1)
  fitb <- gamlr(x, yb,
        free=1:(ncol(config)+ncol(team)),
        family="binomial", standardize=FALSE)
  Bparboot <- cbind(Bparboot, coef(fitb))
  print(b)
}
```

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

```
[1] 68
[1] 69
[1] 70
[1] 71
[1] 72
[1] 73
[1] 74
[1] 75
[1] 76
[1] 77
[1] 78
[1] 79
[1] 80
[1] 81
[1] 82
[1] 83
[1] 84
[1] 85
[1] 86
[1] 87
[1] 88
[1] 89
[1] 90
[1] 91
[1] 92
[1] 93
[1] 94
[1] 95
[1] 96
[1] 97
[1] 98
[1] 99
[1] 100
```

We can try any player, and invert its extreme t-statistics around the point estimate (the top error will give the lower end of the confidence interval):

[70]:
```
WHO <- "SIDNEY_CROSBY"
fB <- exp(Bhat[WHO,])
tval <- quantile(exp(Bparboot[WHO,]), c(.95,.05))
2*fB - tval
```

95\% 1.40542406369885 **5\%** 1.75575777069571

Or we can repeat the same for the intercept, which we just agreed to be interpreted as the home ice advantage here.

[71]:
```
fB <- exp(Bhat[1,])
tval <- quantile(exp(Bparboot[1,]), c(.95,.05))
2*fB - tval
```

95\% 1.06831035841102 **5\%** 1.09970626043512

2.5.2 Subsampling

Smaller samples than the original can go a surprisingly long way.

Calculate the subsample sizes (which will not be the size for all subsamples, as `n` is not divisible by 4 here).

[72]:
```
n <- nrow(x)
B <- 100
( m <- round(n/4) )
```

17362

Then collect the odds ratios for hundred subsamples.

[ ]:
```
Esubs <- sparseMatrix(dims=c(nrow(Bhat),0),i={},j={})
for(b in 1:B){
        subs <- sample.int(n, m)
        fitb <- gamlr(x[subs,], y[subs],
                free=1:(ncol(config)+ncol(team)),
                family="binomial", standardize=FALSE)
        print(log(fitb$lambda[which.min(AICc(fitb))]))
        plot(fitb)
        eb <- (exp(coef(fitb)) - exp(Bhat))
        Esubs <- cbind(Esubs,  eb)
        print(b)
}
```

```
    seg27
-7.608565
[1] 1
    seg33
-7.769084
[1] 4
    seg30
-7.896912
[1] 35
    seg28
-7.635004
[1] 66
    seg19
-7.207652
[1] 80
```

```
    seg29
-7.625389
```

We can calculate a confidence interval for the intercept (home ice advantage), we just need to remember to scale the t-statistics with the learning rates (and hope that it applies).

[74]:
```
thetahat <- exp(Bhat[1,])
tval <- quantile(Esubs[1,], c(.95,.05))
thetahat - tval*sqrt(m)/sqrt(n)
```

95\% 1.0653859447063 **5\%** 1.09393256569162

Or for a player:

[75]:
```
WHO <- "SIDNEY_CROSBY"
thetahat <- exp(Bhat[WHO,])
tval <- quantile(Esubs[WHO,], c(.95,.05))
thetahat - tval*sqrt(m)/sqrt(n)
```

95\% 1.42228885129811 **5\%** 1.71528096321619

We can also plot the subsampled errors for him:

[76]:
```
par(mai=c(.9,.9,.1,.1))
hist(Esubs[WHO,],col=8, main="",  xlab="e_b: subsampled errors for Crosby␣
 ↪effect")
```