Statistical Learning 1: Preprocessing

Ross Jacobucci

Resources

Kaggle – Data mining competitions www.kaggle.com

Andrew Ng's Machine Learning Coursera course https://www.coursera.org/course/ml

Statistical Learning Online course https://class.stanford.edu/courses/HumanitiesandScience/StatLearning/Winter2015/about

Based on An Introduction to Statistical Learning book – Free http://www-bcf.usc.edu/~gareth/ISL/

Applied Predictive Modeling Free book through library – by author of caret package http://appliedpredictivemodeling.com/

caret package tutorial: http://topepo.github.io/caret/

JSS tutorial: http://www.jstatsoft.org/v28/i05/paper

Let's get started

Relevant packages:

```
library(caret)
library(e1071) # for svm()
library(psych) # for describe()
library(pROC) # for roc
```

If we want an example with a continuous outcome, we can used this dataset:

```
library(AppliedPredictiveModeling)
data(solubility)

# Outcome variables
summary(solTestY)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -10.410  -3.953  -2.480  -2.797  -1.372   1.070
```

```
summary(solTrainY)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -11.620  -3.955  -2.510  -2.719  -1.360   1.580
```

```
# Predictors
# solTestX;solTestXtrans
# solTrainX;solTrainXtrans
```

Before we get started, we are going to handle missing values. A lot of statistical learning packages have problems with missing values. To take care of this, I am going to impute all missing values for the entire dataset. This is not the best strategy, but since the focus of this talk isn't on missingness, we are going to proceed.

For single imputation, I like to use k-nearest neighbors in the preProcess() from caret (method= "knnImpute")

Just for the predictors.

Also, worth centering (mean=0) and scaling(sd=1) each variable. This will speed up computation, and is necessary for many algorithms such as PCA which shows bias for differently scaled variables

But in viewing the dataset, it looks like rows where there are NA's, the rows are completely missing. Therefore, we will just remove rows of NA's

Almost all datasets that come with packages don't have missingness and are pretty clean. As an example of pre processing, we will use the HolzingerSwineford1939 dataset

```
# first find number of missing
# can get from summary()
# also from describe() in psych package
library(psych)
library(lavaan)
HS <- HolzingerSwineford1939
# describe(HS)
# summary(HS)
# another way
apply(apply(HS,2,is.na),2,sum)
```

```
##     id    sex  ageyr  agemo school  grade     x1     x2     x3     x4
##      0      0      0      0      0      1      0      0      0      0
##     x5     x6     x7     x8     x9
##      0      0      0      0      0
```

```
# only one missing value
```

To center and scale, as well as impute, we can use preProcess() from caret. As an example, we will just use the x variables from HS

```
pred.pre <- preProcess(HS[,7:15],method=c("center","scale","knnImpute"))
XX.pre <- predict(pred.pre,HS[,7:15])
summary(XX.pre)
```

```
##        x1                 x2                 x3
##  Min.   :-3.65683   Min.   :-3.25962   Min.   :-1.7687
##  1st Qu.:-0.65880   1st Qu.:-0.71174   1st Qu.:-0.7740
##  Median : 0.05502   Median :-0.07477   Median :-0.1109
##  Mean   : 0.00000   Mean   : 0.00000   Mean   : 0.0000
##  3rd Qu.: 0.62607   3rd Qu.: 0.56220   3rd Qu.: 0.7733
##  Max.   : 3.05305   Max.   : 2.68543   Max.   : 1.9891
##        x4                 x5                x6                x7
##  Min.   :-2.62938   Min.   :-2.5886   Min.   :-1.8645   Min.   :-2.64476
##  1st Qu.:-0.62500   1st Qu.:-0.6513   1st Qu.:-0.6909   1st Qu.:-0.64949
##  Median :-0.05232   Median : 0.1236   Median :-0.1694   Median :-0.09081
##  Mean   : 0.00000   Mean   : 0.0000   Mean   : 0.0000   Mean   : 0.00000
```

```
##  3rd Qu.: 0.52036   3rd Qu.: 0.7048   3rd Qu.: 0.4826   3rd Qu.: 0.66739
##  Max.   : 2.81108   Max.   : 2.0608   Max.   : 3.6120   Max.   : 2.98190
##        x8                  x9
##  Min.   :-2.44622   Min.   :-2.57280
##  1st Qu.:-0.66864   1st Qu.:-0.61846
##  Median :-0.02674   Median : 0.04216
##  Mean   : 0.00000   Mean   : 0.00000
##  3rd Qu.: 0.56579   3rd Qu.: 0.70278
##  Max.   : 4.41720   Max.   : 3.84073
```

```r
# or, can just use scale() from base R
# ?scale
```

Now let's check to make sure there isn't any significant skewness or kurtosis

```r
describe(HS)
```

```
##         vars   n   mean     sd median trimmed    mad   min    max  range
## id         1 301 176.55 105.94 163.00  176.78 140.85  1.00 351.00 350.00
## sex        2 301   1.51   0.50   2.00    1.52   0.00  1.00   2.00   1.00
## ageyr      3 301  13.00   1.05  13.00   12.89   1.48 11.00  16.00   5.00
## agemo      4 301   5.38   3.45   5.00    5.32   4.45  0.00  11.00  11.00
## school*    5 301   1.52   0.50   2.00    1.52   0.00  1.00   2.00   1.00
## grade      6 300   7.48   0.50   7.00    7.47   0.00  7.00   8.00   1.00
## x1         7 301   4.94   1.17   5.00    4.96   1.24  0.67   8.50   7.83
## x2         8 301   6.09   1.18   6.00    6.02   1.11  2.25   9.25   7.00
## x3         9 301   2.25   1.13   2.12    2.20   1.30  0.25   4.50   4.25
## x4        10 301   3.06   1.16   3.00    3.02   0.99  0.00   6.33   6.33
## x5        11 301   4.34   1.29   4.50    4.40   1.48  1.00   7.00   6.00
## x6        12 301   2.19   1.10   2.00    2.09   1.06  0.14   6.14   6.00
## x7        13 301   4.19   1.09   4.09    4.16   1.10  1.30   7.43   6.13
## x8        14 301   5.53   1.01   5.50    5.49   0.96  3.05  10.00   6.95
## x9        15 301   5.37   1.01   5.42    5.37   0.99  2.78   9.25   6.47
##          skew kurtosis   se
## id      -0.01    -1.36 6.11
## sex     -0.06    -2.00 0.03
## ageyr    0.69     0.20 0.06
## agemo    0.09    -1.22 0.20
## school* -0.07    -2.00 0.03
## grade    0.09    -2.00 0.03
## x1      -0.25     0.31 0.07
## x2       0.47     0.33 0.07
## x3       0.38    -0.91 0.07
## x4       0.27     0.08 0.07
## x5      -0.35    -0.55 0.07
## x6       0.86     0.82 0.06
## x7       0.25    -0.31 0.06
## x8       0.53     1.17 0.06
## x9       0.20     0.29 0.06
```

```r
# nothing horrible
```

If there was significant skewness or kurtosis:

use BoxCox among others available in preProcess()

```
preProc <- preProcess(XX.pre,method=c("BoxCox"))
# to see all options ?preProcess
```

Overfitting

Demonstration what happens with so many predictors

```
mat <- data.frame(matrix(0, nrow = 2000, ncol = 100))
for(i in 1:ncol(mat)) mat[,i] <- rnorm(2000)
mat$y.bin <- rbinom(1000,1,.5)
mat$y.reg <- rnorm(2000)

out3 = lm(y.reg ~ ., data= mat)
summary(out3)$r.squared
```

```
## [1] 0.04966195
```

```
# r2 = 0.09
# summary(out3)


# use Support Vector Machines

svm.out1 <- svm(y.reg ~ ., data= mat) # from e1071
# summary(svm.out1)
f.predict <- predict(svm.out1,mat)
(pred.rsq <- cor(f.predict, mat$y.reg)**2)
```

```
## [1] 0.776725
```

So, we can get an R-squared of .78 just a result of having enough random predictors?

When we start to use more powerful algorithms, the propensity for over-fitting – fitting random noise and not "real" variation – increases drastically

We need a way to prevent over-fitting, but also want to use something powerful enough to capture meaningful effects. This is the concept of bias-variance tradeoff

Bias = how accurately are capturing the effects? Variance = how variable are our estimated effects across partitions of the dataset?

p.36 ISLR

These concepts are best demonstrated with cross-validation

Cross-Validation – p.71 APM

```
# ?sample
set.seed(3425)
idx <- sample(2000, 1000,replace=FALSE)
train <- mat[idx,]; dim(train)
```

```
## [1] 1000  102
```

```
test <- mat[-idx,]; dim(test)
```

```
## [1] 1000  102
```

```
lm.out2 <- lm(y.reg ~., train)
# summary(lm.out2)
# r2 = 0.12
pred.testLM <- predict(lm.out2,test)
(pred.rsq <- cor(pred.testLM, test$y.reg)**2)
```

```
## [1] 0.0002286291
```

```
# same thing with svm
svm.out2 <- svm(y.reg ~., train)
# train
pred.trainSVM <- predict(svm.out2,train)
cor(pred.trainSVM, train$y.reg)**2
```

```
## [1] 0.7941319
```

```
# test
pred.testSVM <- predict(svm.out2,test)
cor(pred.testSVM, test$y.reg)**2
```

```
## [1] 0.002017737
```

How about if we cross-validate our real regression example?

```
set.seed(3425)
sol.dat <- data.frame(solTrainX,solTrainY)
rowN <- nrow(sol.dat)
idxx <- sample(rowN, rowN/2,replace=FALSE)
trainX <- sol.dat[idxx,];dim(trainX)
```

```
## [1] 475 229
```

```
testX <- sol.dat[-idxx,];dim(testX)
```

```
## [1] 476 229
```

```
lm.out4 <- lm(solTrainY ~., trainX)
summary(lm.out4)$r.squared
```

```
## [1] 0.9628939
```

```r
pred.testX <- predict(lm.out4,testX)
```

```
## Warning in predict.lm(lm.out4, testX): prediction from a rank-deficient
## fit may be misleading
```

```r
cor(pred.testX, testX$solTrainY,use="complete.obs")**2
```

```
## [1] 0.8318436
```

When sample size is the problem, in comparison to having ample predictors, bootstrapping might be the best solution.

Sampling w/ replacement – predict on whatever samples were not sampled in first part – generally not as good when sample sizes are small

```r
bootSplits <- createResample(sol.dat$solTrainY,times=2,2)
str(bootSplits)
```

```
## List of 2
##  $ Resample1: int [1:951] 1 2 6 9 11 12 13 13 13 15 ...
##  $ Resample2: int [1:951] 2 3 7 10 10 11 12 12 15 15 ...
```

```r
lm.out44 <- lm(solTrainY ~., sol.dat[bootSplits[[1]],])
summary(lm.out44)$r.squared
```

```
## [1] 0.9594291
```

```r
pred.testX2 <- predict(lm.out44,sol.dat[bootSplits[[2]],])
```

```
## Warning in predict.lm(lm.out44, sol.dat[bootSplits[[2]], ]): prediction
## from a rank-deficient fit may be misleading
```

```r
cor(pred.testX2, sol.dat[bootSplits[[2]],]$solTrainY,use="complete.obs")**2
```

```
## [1] 0.9082652
```

Use of the train() function

General framework for running data mining algorithms from the caret package.

The train() function from the caret package is a general wrapper where you can using CV, bootstrapping, do pre-processing and run almost all of the statistical learning algorithms from all of the popular packages.

Can use very advanced methods such as support vector machines.

Using our simulated random noise from before

```r
# ?train
# ?trainControl

my.control <- trainControl(method="cv")

out.tr <- train(y.reg ~., data=train,method="svmRadial",trControl=my.control)
```

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

```r
out.tr
```

```
## Support Vector Machines with Radial Basis Function Kernel
##
## 1000 samples
##  101 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
##
## Summary of sample sizes: 900, 900, 900, 900, 900, 900, ...
##
## Resampling results across tuning parameters:
##
##   C     RMSE      Rsquared     RMSE SD     Rsquared SD
##   0.25  1.010839  0.003798040  0.03341260  0.005534844
##   0.50  1.020622  0.004129571  0.03777746  0.005357757
##   1.00  1.033425  0.003984690  0.04265120  0.004440483
##
## Tuning parameter 'sigma' was held constant at a value of 0.005103895
## RMSE was used to select the optimal model using  the smallest value.
## The final values used for the model were sigma = 0.005103895 and C = 0.25.
```

So that brings us back down to reality. Two important things to keep in mind. 1. with some methods, it can be really easy to overfit your data, meaning that it won't generalize. 2. This is especially pertinent when the number of variables is large in relation to number of respondents.

Let's go into easier ways to either use cross-validation or bootstrapping to prevent over-fitting

Using the caret package primarily

Whether to use Cross-Validation or Bootstrapping?

No right answer, and in most cases give very similar results. When sample size is an issue, bootstrapping may do better, as the resulting sample sizes for each bootstrap sample = starting N.

See pages 69-73 in Applied Predictive Modeling

Just easier to leave dataset and specify within the actual prediction script

Data Mining Labs/boot.png

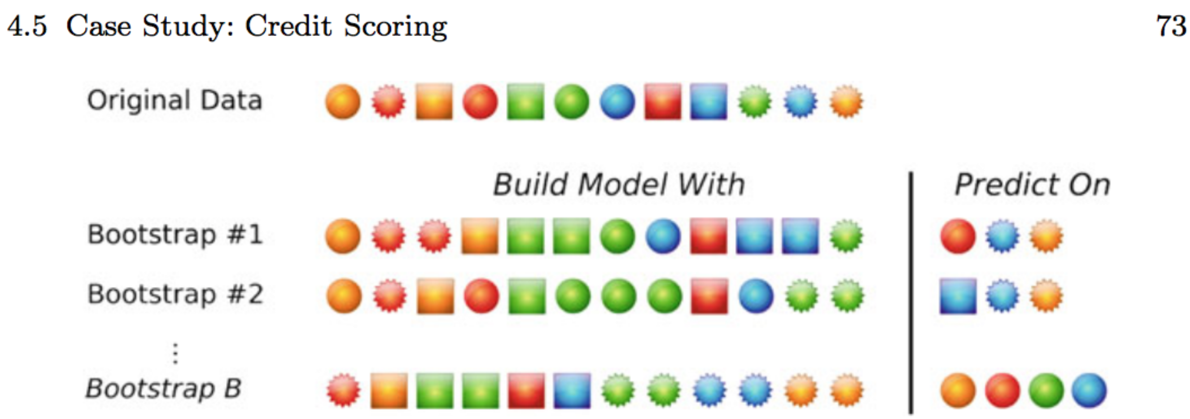

Figure 1: bootstrapping

Data Mining Labs/cv.png

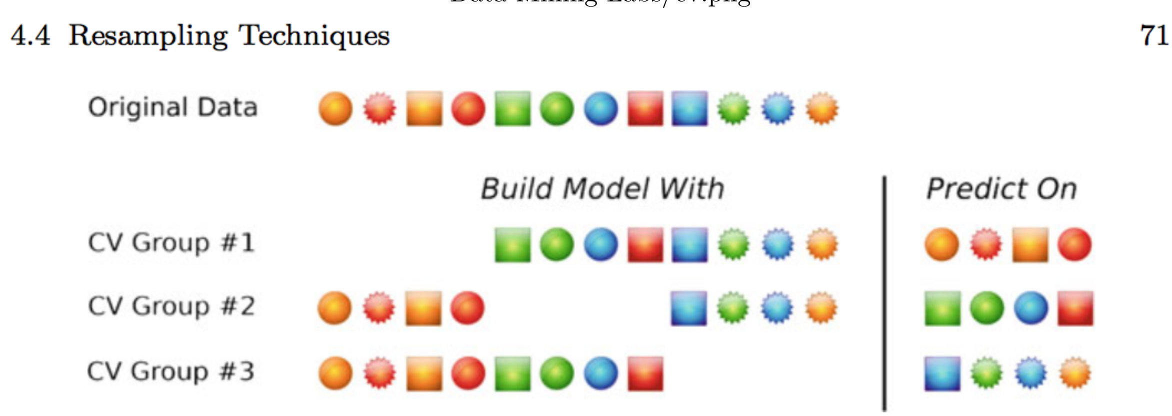

Figure 2: K-fold Cross-Validation

How do set up parallel with caret: http://topepo.github.io/caret/parallel.html

```
 library(doSNOW)
 getDoParWorkers()
 cl <- makeCluster(4, type="SOCK")
 registerDoSNOW(cl)


 my.control <- trainControl(method="cv",allowParallel=T)



system.time(penal.out1 <- train(y.reg ~., train,method="penalized",
                                trControl = my.control,
                               tuneLength=5))
penal.out1
# quartz()
 plot(penal.out1)
```

See that maybe the tuning parameters that were used, defaults, don't capture the potential "best" model – see the the lowest RMSE is at the edge of the graph

I would either specify own tune grid, which I will demonstrate in part 3, or would just specify a wider tuneLength (15 or so)