Predictive Analytics: practical 2 solutions

```
library("caret")
data(FuelEconomy, package = "AppliedPredictiveModeling")
set.seed(25)
```

Cross validation and the bootstrap

- Fit a linear regression model to the `cars2010` data set with `FE` as the response, using `EngDispl`, `NumCyl` and `NumGears` as predictors.

The data set can be loaded `data("FuelEconomy", package = "AppliedPredictiveModeling")`.

```
mLM = train(FE ~ EngDispl + NumCyl + NumGears, method = "lm", data = cars2010)
```

- What is the training error rate (RMSE) for this model?

Hint: The training error can be found by taking the square root of the average square residuals. The `sqrt` and `resid` functions may be useful.

```
res = resid(mLM)
(trainRMSE = sqrt(mean(res * res)))

## [1] 4.59
```

- Re–train your model using the validation set approach to estimate a test RMSE, make your validation set equivalent to half of the entire data set.

```
## pick an index for samples floor just rounds down so we only try to sam-
ple
## a whole number
index = sample(nrow(cars2010), floor(nrow(cars2010)/2))
## set up a train control object
tcVS = trainControl(method = "cv", index = list(Fold1 = (1:nrow(cars2010))[-index]))
## train the model
mLMVS = train(FE ~ EngDispl + NumCyl + NumGears, method = "lm", data = cars2010,
    trControl = tcVS)
```

- How does this compare to the training error that we estimated above?

```
# it's larger, often training error under estimates test error
getTrainPerf(mLMVS)

##   TrainRMSE TrainRsquared method
## 1     4.426        0.6454     lm

trainRMSE

## [1] 4.59
```

- Go through the same process using the different methods for estimating test error. That is leave one out and k–fold crossvalidation as well as bootstrapping.

10–fold cross validation can be shown to be a good choice for almost any situation.

```
# set up train control objects
tcLOOCV = trainControl(method = "LOOCV")
tcKFOLD = trainControl(method = "cv", number = 10)
tcBOOT = trainControl(method = "boot")
```

```
# train the model
mLMLOOCV = train(FE ~ EngDispl + NumCyl + NumGears, method = "lm", data = cars2010,
    trControl = tcLOOCV)
mLMKFOLD = train(FE ~ EngDispl + NumCyl + NumGears, method = "lm", data = cars2010,
    trControl = tcKFOLD)
mLMBOOT = train(FE ~ EngDispl + NumCyl + NumGears, method = "lm", data = cars2010,
    trControl = tcBOOT)
```

- How do these estimates compare with the validation set approach?

```
getTrainPerf(mLMVS)

##   TrainRMSE TrainRsquared method
## 1     4.426        0.6454     lm

getTrainPerf(mLMLOOCV)

##   TrainRMSE TrainRsquared method
## 1     4.612        0.6214     lm

getTrainPerf(mLMKFOLD)

##   TrainRMSE TrainRsquared method
## 1     4.593        0.6275     lm

getTrainPerf(mLMBOOT)

##   TrainRMSE TrainRsquared method
## 1     4.644        0.6204     lm

# all lower than validation set, we mentioned it tended to over es-
timate
# test error
```

- The object returned by `train` also contains timing information that can be accessed via the `times` component of the list. Which of the methods is fastest?

The `$` notation can be used pick a single list component.

```
mLMVS$times$everything

##    user  system elapsed
##   0.432   0.000   0.437

mLMLOOCV$times$everything

##    user  system elapsed
##   6.116   0.012   6.153

mLMKFOLD$times$everything

##    user  system elapsed
##   0.648   0.000   0.651

mLMBOOT$times$everything

##    user  system elapsed
##   0.552   0.004   0.563
```

- Using k–fold cross validation to estimate test error investigate how the number of folds effects the resultant estimates and computation time.

```
# a number of trainControl objects
tc2 = trainControl(method = "cv", number = 2)
tc5 = trainControl(method = "cv", number = 5)
tc10 = trainControl(method = "cv", number = 10)
tc15 = trainControl(method = "cv", number = 15)
tc20 = trainControl(method = "cv", number = 20)
# train the model using each
mLM2 = train(FE ~ EngDispl + NumCyl + NumGears, method = "lm", data = cars2010,
    trControl = tc2)
mLM5 = train(FE ~ EngDispl + NumCyl + NumGears, method = "lm", data = cars2010,
    trControl = tc5)
mLM10 = train(FE ~ EngDispl + NumCyl + NumGears, method = "lm", data = cars2010,
    trControl = tc10)
mLM15 = train(FE ~ EngDispl + NumCyl + NumGears, method = "lm", data = cars2010,
    trControl = tc15)
mLM20 = train(FE ~ EngDispl + NumCyl + NumGears, method = "lm", data = cars2010,
    trControl = tc20)
# use a data frame to store all of the relevant information
(info = data.frame(Folds = c(2, 5, 10, 15, 20), Time = c(mLM2$times$everything[1],
    mLM5$times$everything[1], mLM10$times$everything[1], mLM15$times$everything[1],
    mLM20$times$everything[1]), Estimate = c(mLM2$results$RMSE, mLM5$results$RMSE,
    mLM10$results$RMSE, mLM15$results$RMSE, mLM20$results$RMSE)))

##   Folds  Time Estimate
## 1     2 0.504    4.594
## 2     5 0.620    4.615
## 3    10 0.656    4.593
## 4    15 0.696    4.579
## 5    20 0.728    4.543

# as there are more folds it takes longer to compute, not an is-
sue with such
# a small model but something to consider on more complicated mod-
els.
# Estimates are going down as the number of folds increases.  This is
# because for each held out fold we are using a greater propor-
tion of the
# data in training so expect to get a better model.
```

- Experiment with adding terms to the model, transformations of the predictors and interactions say and use cross validation to estimate test error for each. What is the best model you can find? You can still use the `validate` and `mark` functions to look at how your models fair on the unseen data.

Penalised regression

The `diabetes` data set in the `lars` package contains measurements of a number of predictors to model a response y, a measure of disease progression. There are other columns in the data set which contain interactions so we will extract just the predictors and the response. The data has already been normalized.

```r
## load the data in
data(diabetes, package = "lars")
diabetesdata = cbind(diabetes$x, y = diabetes$y)
```

- Try fitting a lasso, ridge and elastic net model using all of the main effects, pairwise interactions and square terms from each of the predictors.[1]

```r
modelformula = as.formula(paste("y~(.)^2 + ", paste0("I(", colnames(diabetesdata),
    "^2)", collapse = "+")))
mLASSO = train(modelformula, data = diabetesdata, method = "lasso")
mRIDGE = train(modelformula, data = diabetesdata, method = "ridge")
mENET = train(modelformula, data = diabetesdata, method = "enet")
```

[1] Hint: see notes for shortcut on creating model formula. Also be aware that if the predictor is a factor a polynomial term doesn't make sense

- Try to narrow in on the region of lowest RMSE for each model, don't forget about the `tuneGrid` argument to the train function.

`fraction = 0` is the same as the null model.

$y \sim (.) \wedge 2$ is short hand for a model that includes pairwise interactions for each predictor, so if we use this we should only need to add the square terms

```r
# examine previous output then train over a finer grid near the bet-
ter end
mLASSOfine = train(modelformula, data = diabetesdata, method = "lasso", tuneGrid = data.frame(fraction = seq(
    0.5, by = 0.05)))
mLASSOfine$results

##   fraction  RMSE Rsquared RMSESD RsquaredSD
## 1     0.10 17.05   0.9519 0.9567   0.005946
## 2     0.15 17.24   0.9504 0.9892   0.006123
## 3     0.20 17.37   0.9497 0.9561   0.006068
## 4     0.25 17.44   0.9493 0.9231   0.005907
## 5     0.30 17.50   0.9490 0.9225   0.005951
## 6     0.35 17.55   0.9487 0.9379   0.006099
## 7     0.40 17.60   0.9484 0.9717   0.006347
## 8     0.45 17.64   0.9482 1.0222   0.006686
## 9     0.50 17.69   0.9479 1.1017   0.007208

# best still right down at the 0.1 end
mLASSOfiner = train(modelformula, data = diabetesdata, method = "lasso", tuneGrid = data.frame(fraction = seq
    0.15, by = 0.01)))
mLASSOfiner$results

##    fraction  RMSE Rsquared RMSESD RsquaredSD
## 1      0.01 47.19   0.9547 16.897   0.002963
## 2      0.02 31.61   0.9548 17.013   0.003805
## 3      0.03 24.98   0.9548 13.622   0.003924
## 4      0.04 21.64   0.9551 11.061   0.004742
## 5      0.05 20.05   0.9548  8.822   0.005219
## 6      0.06 19.12   0.9544  7.002   0.005496
## 7      0.07 18.51   0.9539  5.373   0.005399
## 8      0.08 18.02   0.9536  3.985   0.005360
## 9      0.09 17.70   0.9532  2.748   0.005244
## 10     0.10 17.44   0.9528  1.771   0.005035
## 11     0.11 17.27   0.9525  1.211   0.004916
## 12     0.12 17.20   0.9522  1.055   0.004895
## 13     0.13 17.21   0.9520  1.063   0.004865
## 14     0.14 17.25   0.9517  1.085   0.004846
## 15     0.15 17.31   0.9513  1.092   0.004796

# best is
mLASSOfiner$bestTune
```

```
##    fraction
## 12     0.12

mRIDGEfine = train(modelformula, data = diabetesdata, method = "ridge", tuneGrid = data.frame(lambda = seq(0,
    0.1, by = 0.01)))
mRIDGEfine$results

##    lambda  RMSE Rsquared RMSESD RsquaredSD
## 1    0.00 18.22   0.9466 1.1003   0.006809
## 2    0.01 17.07   0.9527 0.9093   0.005535
## 3    0.02 16.99   0.9530 0.9285   0.005702
## 4    0.03 17.02   0.9527 0.9545   0.005924
## 5    0.04 17.12   0.9521 0.9823   0.006169
## 6    0.05 17.28   0.9512 1.0104   0.006425
## 7    0.06 17.47   0.9501 1.0385   0.006687
## 8    0.07 17.68   0.9489 1.0666   0.006954
## 9    0.08 17.92   0.9475 1.0949   0.007224
## 10   0.09 18.18   0.9461 1.1233   0.007496
## 11   0.10 18.45   0.9445 1.1520   0.007771

mRIDGEfiner = train(modelformula, data = diabetesdata, method = "ridge", tuneGrid = data.frame(lambda = seq(0
    0.03, by = 0.001)))
mRIDGEfiner$results

##    lambda  RMSE Rsquared RMSESD RsquaredSD
## 1   0.005 17.08   0.9512 0.7848   0.005952
## 2   0.006 17.05   0.9514 0.7894   0.005986
## 3   0.007 17.02   0.9515 0.7940   0.006019
## 4   0.008 17.00   0.9517 0.7986   0.006052
## 5   0.009 16.98   0.9518 0.8032   0.006085
## 6   0.010 16.96   0.9518 0.8078   0.006119
## 7   0.011 16.95   0.9519 0.8124   0.006153
## 8   0.012 16.94   0.9520 0.8170   0.006189
## 9   0.013 16.93   0.9520 0.8216   0.006225
## 10  0.014 16.92   0.9520 0.8263   0.006261
## 11  0.015 16.91   0.9521 0.8310   0.006299
## 12  0.016 16.91   0.9521 0.8357   0.006336
## 13  0.017 16.91   0.9521 0.8404   0.006375
## 14  0.018 16.91   0.9521 0.8451   0.006414
## 15  0.019 16.90   0.9521 0.8499   0.006454
## 16  0.020 16.90   0.9521 0.8546   0.006494
## 17  0.021 16.91   0.9521 0.8594   0.006534
## 18  0.022 16.91   0.9521 0.8641   0.006575
## 19  0.023 16.91   0.9520 0.8689   0.006617
## 20  0.024 16.91   0.9520 0.8736   0.006658
## 21  0.025 16.92   0.9520 0.8784   0.006701
## 22  0.026 16.92   0.9520 0.8831   0.006743
## 23  0.027 16.93   0.9519 0.8878   0.006786
## 24  0.028 16.93   0.9519 0.8925   0.006829
## 25  0.029 16.94   0.9518 0.8972   0.006872
## 26  0.030 16.95   0.9518 0.9019   0.006916

# the best one
mRIDGEfiner$bestTune

##    lambda
## 15  0.019
```

```
mENETfine = train(modelformula, data = diabetesdata, method = "enet", tuneGrid = expand.grid(lambda = c(0.001
    0.01, 0.1), fraction = c(0.4, 0.5, 0.6)))
mENETfine$results

##   lambda fraction  RMSE Rsquared RMSESD RsquaredSD
## 1  0.001      0.4 16.14   0.9568 0.7568   0.003251
## 4  0.010      0.4 16.08   0.9598 1.0548   0.003111
## 7  0.100      0.4 23.09   0.9561 1.6711   0.003129
## 2  0.001      0.5 16.53   0.9546 0.8246   0.004372
## 5  0.010      0.5 15.65   0.9598 0.6997   0.002566
## 8  0.100      0.5 17.01   0.9572 0.8009   0.003267
## 3  0.001      0.6 16.74   0.9535 0.8508   0.004708
## 6  0.010      0.6 16.09   0.9572 0.7347   0.003236
## 9  0.100      0.6 16.57   0.9548 0.7593   0.004618

mENETfiner = train(modelformula, data = diabetesdata, method = "enet", tuneGrid = expand.grid(lambda = seq(0.
    0.1, length.out = 10), fraction = 0.5))
mENETfiner$results

##    lambda fraction  RMSE Rsquared RMSESD RsquaredSD
## 1   0.001      0.5 16.59   0.9554 0.8761   0.003816
## 2   0.012      0.5 15.84   0.9594 0.6638   0.002687
## 3   0.023      0.5 15.94   0.9591 0.5673   0.002533
## 4   0.034      0.5 16.15   0.9584 0.5295   0.002625
## 5   0.045      0.5 16.38   0.9577 0.5441   0.002696
## 6   0.056      0.5 16.60   0.9572 0.6060   0.002649
## 7   0.067      0.5 16.79   0.9569 0.6843   0.002622
## 8   0.078      0.5 16.93   0.9567 0.7439   0.002630
## 9   0.089      0.5 17.05   0.9565 0.7899   0.002654
## 10  0.100      0.5 17.15   0.9562 0.8235   0.002699

mENETfiner$bestTune

##   fraction lambda
## 2      0.5  0.012
```

We can view the coefficients via

```
coef = predict(mLASSO$finalModel,
  mode = "fraction",
  s = mLASSO$bestTune$fraction,# which ever fraction was chosen as best
  type = "coefficients"
)
```

- How many features have been chosen by the `lasso` and `enet` models?

```
# use predict to find the coefficients
coefLASSO = predict(mLASSOfiner$finalModel, mode = "fraction", type = "coefficient",
    s = mLASSO$bestTune$fraction, )
sum(coefLASSO$coefficients != 0)

## [1] 65

coefENET = predict(mENETfiner$finalModel, mode = "fraction", type = "coefficient",
    s = mENET$bestTune$fraction)
sum(coefENET$coefficients != 0)

## [1] 24
```

- How do these models compare to principal components and partial least squares regression?

```
mPCR = train(modelformula, data = diabetesdata, method = "pcr", tuneGrid = data.frame(ncomp = 1:7))
mPLS = train(modelformula, data = diabetesdata, method = "pls", tuneGrid = data.frame(ncomp = 1:7))
mPLS2 = train(modelformula, data = diabetesdata, method = "pls", tuneGrid = data.frame(ncomp = 5:15))
getTrainPerf(mLASSOfiner)

##   TrainRMSE TrainRsquared method
## 1      17.2        0.9522  lasso

getTrainPerf(mRIDGEfiner)

##   TrainRMSE TrainRsquared method
## 1      16.9        0.9521  ridge

getTrainPerf(mENETfiner)

##   TrainRMSE TrainRsquared method
## 1     15.84        0.9594   enet

getTrainPerf(mPCR)

##   TrainRMSE TrainRsquared method
## 1     16.36        0.9561    pcr

getTrainPerf(mPLS2)

##   TrainRMSE TrainRsquared method
## 1     15.65        0.9592    pls

# The elastic net model has the lowest estimated test error, all are fairly
# similar. The elastic net model suggests only 21 non--zero co-
efficients out
# of all of those included in the model.
```