Chapter 10

Chapter 10. Case Study: Compressive Strength of Concrete Mixtures

10.1 Model building strategy

10.2 Model performance

10.3 Optimizing compressive strength

10.4 Computing

```r
library(AppliedPredictiveModeling)
data(concrete)
str(concrete)
```

```
## 'data.frame':    1030 obs. of  9 variables:
##  $ Cement             : num  540 540 332 332 199 ...
##  $ BlastFurnaceSlag   : num  0 0 142 142 132 ...
##  $ FlyAsh             : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ Water              : num  162 162 228 228 192 228 228 228 228 228 ...
##  $ Superplasticizer   : num  2.5 2.5 0 0 0 0 0 0 0 0 ...
##  $ CoarseAggregate    : num  1040 1055 932 932 978 ...
##  $ FineAggregate      : num  676 676 594 594 826 ...
##  $ Age                : int  28 28 270 365 360 90 365 28 28 28 ...
##  $ CompressiveStrength: num  80 61.9 40.3 41 44.3 ...
```

```r
str(mixtures)
```

```
## 'data.frame':    1030 obs. of  9 variables:
##  $ Cement             : num  0.2231 0.2217 0.1492 0.1492 0.0853 ...
##  $ BlastFurnaceSlag   : num  0 0 0.0639 0.0639 0.0569 ...
##  $ FlyAsh             : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ Water              : num  0.0669 0.0665 0.1023 0.1023 0.0825 ...
##  $ Superplasticizer   : num  0.00103 0.00103 0 0 0 ...
##  $ CoarseAggregate    : num  0.43 0.433 0.418 0.418 0.42 ...
##  $ FineAggregate      : num  0.279 0.278 0.266 0.266 0.355 ...
##  $ Age                : int  28 28 270 365 360 90 365 28 28 28 ...
##  $ CompressiveStrength: num  80 61.9 40.3 41 44.3 ...
```

```r
library(Hmisc)
```

```
## Loading required package: lattice
```

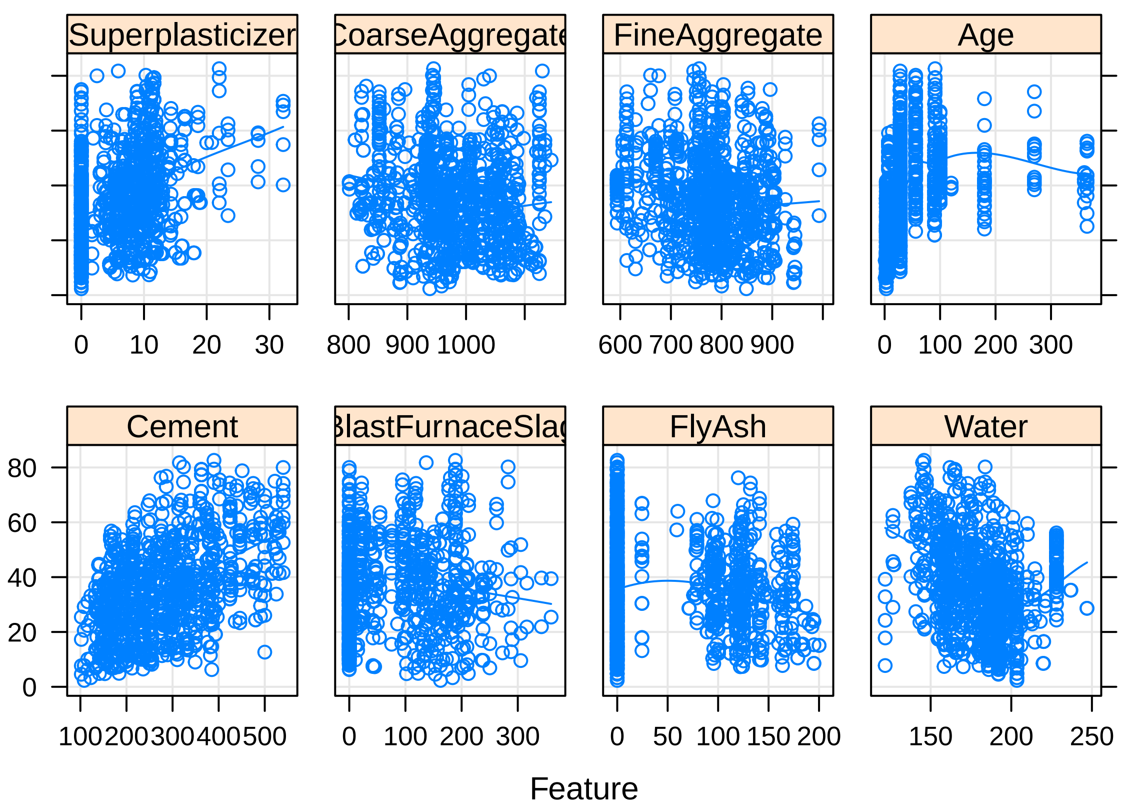
```
## Loading required package: survival
```

```
## Loading required package: Formula

## Loading required package: ggplot2

##
## Attaching package: 'Hmisc'

## The following objects are masked from 'package:base':
##
##     format.pval, units
```

```
library(caret)
```

```
##
## Attaching package: 'caret'

## The following object is masked from 'package:survival':
##
##     cluster
```

```
featurePlot(x = concrete[,-9], y = concrete$CompressiveStrength,
            # add some space between the panels
            between = list(x=1, y=1),
            # add a background grid ('g') and a smoother ('smooth')
            type = c("g","p","smooth"))
```



```r
library(plyr)
```

```
##
## Attaching package: 'plyr'

## The following objects are masked from 'package:Hmisc':
##
##     is.discrete, summarize
```

```r
averaged <- ddply(mixtures,
                  .(Cement, BlastFurnaceSlag, FlyAsh, Water, Superplasticizer, CoarseAggregate, FineAggr
                  function(x) c(CompressiveStrength = mean(x$CompressiveStrength)))
set.seed(975)
forTraining <- createDataPartition(averaged$CompressiveStrength, p=3/4)[[1]]
trainingSet <- averaged[forTraining,]
testSet <- averaged[-forTraining]
```

```r
modFormula <- paste("CompressiveStrength ~ (.)^2 + I(Cement^2) + I(BlastFurnaceSlag^2) + I(FlyAsh^2) +
                    "I(Water^2) + I(Superplasticizer^2) + I(CoarseAggregate^2) + I(FineAggregate^2) + I
modFormula <- as.formula(modFormula)
```

```r
controlObject <- trainControl(method = "repeatedcv", repeats = 5, number = 10)
```

```r
set.seed(669)
linearReg <- train(modFormula, data = trainingSet, method = "lm", trControl = controlObject)
linearReg
```

```
## Linear Regression
##
## 745 samples
##   8 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 5 times)
## Summary of sample sizes: 669, 671, 670, 671, 670, 670, ...
## Resampling results:
##
##   RMSE      Rsquared   MAE
##   7.793733  0.7729528  5.918212
##
## Tuning parameter 'intercept' was held constant at a value of TRUE
```

```r
set.seed(669)
plsModel <- train(modFormula, data = trainingSet, method = "pls", preProc = c("center", "scale"),
                  tuneLength = 15, trControl = controlObject)
plsModel
```

```
## Partial Least Squares
##
## 745 samples
```

```
##   8 predictor
##
## Pre-processing: centered (44), scaled (44)
## Resampling: Cross-Validated (10 fold, repeated 5 times)
## Summary of sample sizes: 669, 671, 670, 671, 670, 670, ...
## Resampling results across tuning parameters:
##
##   ncomp  RMSE       Rsquared   MAE
##    1     10.686223  0.5771316  8.528067
##    2      9.825898  0.6412962  7.639411
##    3      9.203467  0.6828567  7.231871
##    4      8.786256  0.7099623  6.823773
##    5      8.629392  0.7213823  6.692590
##    6      8.538292  0.7280432  6.681010
##    7      8.361594  0.7392440  6.491124
##    8      8.240697  0.7476095  6.386984
##    9      8.027148  0.7587206  6.131429
##   10      7.879233  0.7676063  6.038992
##   11      7.804616  0.7721922  5.943035
##   12      7.786906  0.7733306  5.891280
##   13      7.764694  0.7746082  5.866912
##   14      7.771397  0.7743318  5.912051
##   15      7.761502  0.7744411  5.910179
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was ncomp = 15.
```

```r
enetGrid <- expand.grid(.lambda = c(0, 0.001, 0.01, 0.1), .fraction = seq(0.05, 1, length = 20))
set.seed(669)
enetModel <- train(modFormula, data = trainingSet, method = "enet", preProc = c("center", "scale"),
                   tuneGrid = enetGrid, trControl = controlObject)
enetModel
```

```
## Elasticnet
##
## 745 samples
##   8 predictor
##
## Pre-processing: centered (44), scaled (44)
## Resampling: Cross-Validated (10 fold, repeated 5 times)
## Summary of sample sizes: 669, 671, 670, 671, 670, 670, ...
## Resampling results across tuning parameters:
##
##   lambda  fraction  RMSE       Rsquared   MAE
##   0.000   0.05       8.777012  0.7358542   6.798465
##   0.000   0.10       8.248646  0.7533304   6.324277
##   0.000   0.15       8.024659  0.7609786   6.116170
##   0.000   0.20       7.925163  0.7664959   6.029859
##   0.000   0.25       7.862966  0.7696993   5.980759
##   0.000   0.30       7.826529  0.7714277   5.947684
##   0.000   0.35       7.807004  0.7723666   5.928068
##   0.000   0.40       7.802434  0.7725936   5.922536
##   0.000   0.45       7.793479  0.7730582   5.915434
##   0.000   0.50       7.790497  0.7732144   5.913092
```

```
##   0.000   0.55    7.789149  0.7732684   5.912908
##   0.000   0.60    7.787889  0.7733203   5.912581
##   0.000   0.65    7.787272  0.7733347   5.912544
##   0.000   0.70    7.787400  0.7733138   5.913269
##   0.000   0.75    7.789100  0.7732161   5.914956
##   0.000   0.80    7.790896  0.7731133   5.916643
##   0.000   0.85    7.791748  0.7730607   5.917267
##   0.000   0.90    7.792713  0.7730075   5.917893
##   0.000   0.95    7.793444  0.7729683   5.918231
##   0.000   1.00    7.793751  0.7729514   5.918212
##   0.001   0.05   13.142709  0.5750190  10.633818
##   0.001   0.10   10.891908  0.6499043   8.741199
##   0.001   0.15    9.593049  0.6760233   7.626615
##   0.001   0.20    9.025760  0.7036191   7.098592
##   0.001   0.25    8.568958  0.7314534   6.711028
##   0.001   0.30    8.222272  0.7507305   6.394074
##   0.001   0.35    8.024949  0.7610572   6.194023
##   0.001   0.40    7.872511  0.7688826   6.034336
##   0.001   0.45    7.796956  0.7727254   5.948120
##   0.001   0.50    7.759411  0.7747532   5.903182
##   0.001   0.55    7.738986  0.7758719   5.873889
##   0.001   0.60    7.737689  0.7759426   5.869894
##   0.001   0.65    7.734799  0.7761008   5.866643
##   0.001   0.70    7.729080  0.7764370   5.863658
##   0.001   0.75    7.726199  0.7766122   5.863542
##   0.001   0.80    7.727076  0.7765742   5.865776
##   0.001   0.85    7.729168  0.7764624   5.868577
##   0.001   0.90    7.731895  0.7763136   5.872968
##   0.001   0.95    7.735363  0.7761354   5.878010
##   0.001   1.00    7.738564  0.7759703   5.881658
##   0.010   0.05   14.146214  0.5609364  11.463882
##   0.010   0.10   12.384686  0.6007118   9.985400
##   0.010   0.15   10.981399  0.6481591   8.816088
##   0.010   0.20    9.974103  0.6678370   7.985084
##   0.010   0.25    9.422237  0.6825885   7.457748
##   0.010   0.30    9.075469  0.6999963   7.140302
##   0.010   0.35    8.760429  0.7195500   6.871052
##   0.010   0.40    8.482733  0.7358725   6.633893
##   0.010   0.45    8.261971  0.7479854   6.431623
##   0.010   0.50    8.118409  0.7555217   6.287704
##   0.010   0.55    8.004568  0.7615365   6.173620
##   0.010   0.60    7.914113  0.7663576   6.081125
##   0.010   0.65    7.844873  0.7700306   6.007294
##   0.010   0.70    7.804707  0.7721590   5.959749
##   0.010   0.75    7.788006  0.7730959   5.937320
##   0.010   0.80    7.778180  0.7736755   5.923836
##   0.010   0.85    7.766426  0.7743512   5.912421
##   0.010   0.90    7.756297  0.7749317   5.902650
##   0.010   0.95    7.747961  0.7754104   5.894736
##   0.010   1.00    7.741237  0.7757971   5.888871
##   0.100   0.05   14.894640  0.5339395  12.077796
##   0.100   0.10   13.651374  0.5713014  11.058967
##   0.100   0.15   12.568825  0.5907603  10.150393
##   0.100   0.20   11.625171  0.6274678   9.342165
```

```
##   0.100   0.25      10.813480  0.6520992   8.680603
##   0.100   0.30      10.171387  0.6638884   8.156503
##   0.100   0.35       9.709773  0.6722540   7.746155
##   0.100   0.40       9.426224  0.6788217   7.471123
##   0.100   0.45       9.228571  0.6868858   7.259851
##   0.100   0.50       9.047891  0.6970140   7.078173
##   0.100   0.55       8.877877  0.7075570   6.927110
##   0.100   0.60       8.728624  0.7167362   6.795128
##   0.100   0.65       8.600260  0.7246006   6.679025
##   0.100   0.70       8.502921  0.7306001   6.592592
##   0.100   0.75       8.417488  0.7359141   6.517971
##   0.100   0.80       8.349558  0.7402414   6.458044
##   0.100   0.85       8.293547  0.7438918   6.408645
##   0.100   0.90       8.243270  0.7472534   6.363470
##   0.100   0.95       8.199857  0.7502549   6.322449
##   0.100   1.00       8.173282  0.7522897   6.295281
##
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were fraction = 0.75 and lambda
##  = 0.001.
```

```r
set.seed(669)
earthModel <- train(CompressiveStrength ~., data = trainingSet, method = "earth",
                    tuneGrid = expand.grid(.degree = 1, .nprune = 2:25),
                    trControl = controlObject)
```

```
## Loading required package: earth
```

```
## Loading required package: plotmo
```

```
## Loading required package: plotrix
```

```
## Loading required package: TeachingDemos
```

```
##
## Attaching package: 'TeachingDemos'
```

```
## The following objects are masked from 'package:Hmisc':
##
##     cnvrt.coords, subplot
```

```r
earthModel
```

```
## Multivariate Adaptive Regression Spline
##
## 745 samples
##   8 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 5 times)
## Summary of sample sizes: 669, 671, 670, 671, 670, 670, ...
```

```
## Resampling results across tuning parameters:
##
##   nprune  RMSE       Rsquared   MAE
##    2      13.148975  0.3496710  10.507195
##    3      10.912980  0.5526332   8.688464
##    4       9.567298  0.6557580   7.682023
##    5       8.371548  0.7380841   6.614260
##    6       7.849630  0.7696691   6.166590
##    7       7.598993  0.7838576   5.924741
##    8       7.242498  0.8037416   5.686294
##    9       6.992479  0.8173911   5.458558
##   10       6.805225  0.8272864   5.286304
##   11       6.690375  0.8331375   5.170200
##   12       6.547506  0.8400401   5.072728
##   13       6.527175  0.8410921   5.058401
##   14       6.482777  0.8429867   5.024928
##   15       6.429592  0.8456354   4.988556
##   16       6.422618  0.8460278   4.988980
##   17       6.418216  0.8462514   4.984509
##   18       6.418216  0.8462514   4.984509
##   19       6.418216  0.8462514   4.984509
##   20       6.418216  0.8462514   4.984509
##   21       6.418216  0.8462514   4.984509
##   22       6.418216  0.8462514   4.984509
##   23       6.418216  0.8462514   4.984509
##   24       6.418216  0.8462514   4.984509
##   25       6.418216  0.8462514   4.984509
##
## Tuning parameter 'degree' was held constant at a value of 1
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were nprune = 17 and degree = 1.
```

```r
set.seed(669)
svmRModel <- train(CompressiveStrength ~., data = trainingSet, method = "svmRadial", tuneLength = 15,
                   preProc = c("center","scale"), trControl = controlObject)
svmRModel
```

```
## Support Vector Machines with Radial Basis Function Kernel
##
## 745 samples
##   8 predictor
##
## Pre-processing: centered (8), scaled (8)
## Resampling: Cross-Validated (10 fold, repeated 5 times)
## Summary of sample sizes: 669, 671, 670, 671, 670, 670, ...
## Resampling results across tuning parameters:
##
##   C         RMSE      Rsquared   MAE
##      0.25   7.870614  0.7790384  5.953356
##      0.50   7.209753  0.8099314  5.335899
##      1.00   6.710677  0.8326131  4.903147
##      2.00   6.375877  0.8481605  4.608550
##      4.00   6.179668  0.8568585  4.424947
##      8.00   6.125571  0.8589827  4.355597
```

```
##     16.00  6.148312  0.8574860  4.310860
##     32.00  6.154154  0.8572965  4.255487
##     64.00  6.233592  0.8543944  4.245437
##    128.00  6.365059  0.8496627  4.243643
##    256.00  6.739452  0.8357280  4.342215
##    512.00  7.379936  0.8120553  4.509602
##   1024.00  8.118553  0.7873896  4.680840
##   2048.00  9.126569  0.7574043  4.914763
##   4096.00  9.583443  0.7433138  5.086622
##
## Tuning parameter 'sigma' was held constant at a value of 0.1181394
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were sigma = 0.1181394 and C = 8.
```

```r
library(caret)
nnetGrid <- expand.grid(.decay = c(0.001, 0.01, 0.1), .size = seq(1,27,by=2), .bag = FALSE)
set.seed(669)
nnetModel <- train(CompressiveStrength ~., data = trainingSet, method = "avNNet",
                   tuneGrid = nnetGrid, preProc = c("center","scale"), linout = TRUE, trace = FALSE,
                   maxit = 1000, trControl = controlObject)
```

```
## Warning: executing %dopar% sequentially: no parallel backend registered
```

```r
nnetModel
```

```
## Model Averaged Neural Network
##
## 745 samples
##   8 predictor
##
## Pre-processing: centered (8), scaled (8)
## Resampling: Cross-Validated (10 fold, repeated 5 times)
## Summary of sample sizes: 669, 671, 670, 671, 670, 670, ...
## Resampling results across tuning parameters:
##
##   decay  size  RMSE      Rsquared   MAE
##   0.001   1    9.070254  0.6952077  6.808618
##   0.001   3    6.435429  0.8446947  4.923199
##   0.001   5    5.862001  0.8712621  4.445956
##   0.001   7    5.501846  0.8867430  4.055439
##   0.001   9    5.360940  0.8924604  3.922281
##   0.001  11    5.308555  0.8937872  3.814884
##   0.001  13    5.131138  0.9008374  3.711853
##   0.001  15    4.997122  0.9071646  3.644431
##   0.001  17    5.031218  0.9052018  3.640009
##   0.001  19    4.981265  0.9068542  3.625473
##   0.001  21    4.909116  0.9094943  3.508300
##   0.001  23    4.837231  0.9124794  3.426997
##   0.001  25    4.728064  0.9157376  3.398020
##   0.001  27    4.747206  0.9146598  3.336234
##   0.010   1    9.056154  0.6934993  6.782774
##   0.010   3    6.449917  0.8437576  4.939089
##   0.010   5    5.861583  0.8713064  4.437875
```

```
##   0.010   7    5.437815  0.8889819  4.023065
##   0.010   9    5.395636  0.8912909  3.933531
##   0.010  11    5.244811  0.8965617  3.781682
##   0.010  13    5.139412  0.9011020  3.712262
##   0.010  15    5.080820  0.9030142  3.675943
##   0.010  17    4.987965  0.9060166  3.639735
##   0.010  19    4.911595  0.9096202  3.524085
##   0.010  21    4.886657  0.9103715  3.484043
##   0.010  23    4.829494  0.9117777  3.440912
##   0.010  25    4.754707  0.9149276  3.382214
##   0.010  27    4.660324  0.9185007  3.263351
##   0.100   1    9.059092  0.6932968  6.787898
##   0.100   3    6.450409  0.8439978  4.943384
##   0.100   5    5.870099  0.8707027  4.446506
##   0.100   7    5.540672  0.8851626  4.125624
##   0.100   9    5.312418  0.8936771  3.870423
##   0.100  11    5.231820  0.8967252  3.776462
##   0.100  13    5.047369  0.9030714  3.637641
##   0.100  15    4.980330  0.9062083  3.606705
##   0.100  17    4.912710  0.9092401  3.547962
##   0.100  19    4.840151  0.9115538  3.491534
##   0.100  21    4.866478  0.9110126  3.407998
##   0.100  23    4.749349  0.9149668  3.307439
##   0.100  25    4.734429  0.9161555  3.252730
##   0.100  27    4.661788  0.9176969  3.221960
##
## Tuning parameter 'bag' was held constant at a value of FALSE
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were size = 27, decay = 0.01 and bag
##  = FALSE.
```

```r
set.seed(669)
rpartModel <- train(CompressiveStrength ~., data = trainingSet, method = "rpart",
                    tuneLength = 30, trControl = controlObject)
```

```
## Warning in nominalTrainWorkflow(x = x, y = y, wts = weights, info =
## trainInfo, : There were missing values in resampled performance measures.
```

```r
rpartModel
```

```
## CART
##
## 745 samples
##   8 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 5 times)
## Summary of sample sizes: 669, 671, 670, 671, 670, 670, ...
## Resampling results across tuning parameters:
##
##   cp           RMSE      Rsquared   MAE
##   0.003414359   7.872686  0.7705089   6.118249
##   0.003484928   7.887426  0.7695502   6.131358
```

```
##   0.003849507   8.031232  0.7610090   6.257793
##   0.004047591   8.108501  0.7561988   6.324919
##   0.004137208   8.129317  0.7550278   6.340190
##   0.004489645   8.239545  0.7480107   6.463527
##   0.004769860   8.296016  0.7440922   6.538068
##   0.005030873   8.329187  0.7416987   6.571282
##   0.005710238   8.459175  0.7336122   6.684161
##   0.005924990   8.528279  0.7293367   6.744097
##   0.006148500   8.575033  0.7262811   6.782601
##   0.006219119   8.590068  0.7253691   6.802488
##   0.006941738   8.679955  0.7186922   6.908074
##   0.007014985   8.695146  0.7175055   6.914936
##   0.008015936   8.743165  0.7139315   6.962339
##   0.008383300   8.778860  0.7114655   6.990469
##   0.009140156   8.837722  0.7074148   7.045350
##   0.009943621   8.877831  0.7044571   7.079607
##   0.015558045   9.478248  0.6613944   7.577637
##   0.016562920   9.586246  0.6540507   7.661178
##   0.019852002  10.055631  0.6195152   8.027382
##   0.019919193  10.063548  0.6188917   8.034147
##   0.020138914  10.077859  0.6179432   8.044790
##   0.023903544  10.284039  0.6027430   8.242514
##   0.034924404  10.679404  0.5710176   8.547191
##   0.045288048  11.120520  0.5369476   8.881875
##   0.061930086  11.641197  0.4910965   9.270130
##   0.076473221  12.431002  0.4181979  10.036235
##   0.149546411  13.554794  0.3124399  10.887600
##   0.252785984  15.701867  0.1790439  12.636016
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was cp = 0.003414359.
```

```r
set.seed(669)
ctreeModel <- train(CompressiveStrength ~., data = trainingSet, method = "ctree",
                    tuneLength = 10, trControl = controlObject)
ctreeModel
```

```
## Conditional Inference Tree
##
## 745 samples
##   8 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 5 times)
## Summary of sample sizes: 669, 671, 670, 671, 670, 670, ...
## Resampling results across tuning parameters:
##
##   mincriterion  RMSE      Rsquared   MAE
##   0.0100000     7.877881  0.7691471  5.891529
##   0.1188889     7.896399  0.7681181  5.907782
##   0.2277778     7.910486  0.7673999  5.920438
##   0.3366667     7.911665  0.7672684  5.920528
##   0.4455556     7.976115  0.7631754  5.963488
##   0.5544444     7.982651  0.7627044  5.974668
```

```
##   0.6633333     8.016273  0.7612542  6.016877
##   0.7722222     8.102449  0.7555385  6.089743
##   0.8811111     8.292412  0.7430016  6.220249
##   0.9900000     9.271014  0.6765780  7.000579
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was mincriterion = 0.01.
```

```
library(rJava)
library(RWeka)
set.seed(669)
mtModel <- train(CompressiveStrength ~., data =trainingSet, method = "M5",
                 trControl = controlObject)
mtModel
```

```
## Model Tree
##
## 745 samples
##   8 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 5 times)
## Summary of sample sizes: 669, 671, 670, 671, 670, 670, ...
## Resampling results across tuning parameters:
##
##   pruned  smoothed  rules  RMSE         Rsquared    MAE
##   Yes     Yes       Yes    2679.399819  0.06207210  1316.953001
##   Yes     Yes       No     3415.308059  0.19798488  1937.190064
##   Yes     No        Yes       6.669281  0.83289468     4.809588
##   Yes     No        No        6.367282  0.84954382     4.502208
##   No      Yes       Yes    3365.560568  0.07614309  1610.584523
##   No      Yes       No     3415.322426  0.19798879  1937.185180
##   No      No        Yes       8.365232  0.74527816     5.825794
##   No      No        No        6.849163  0.82743188     4.830694
##
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were pruned = Yes, smoothed = No
##  and rules = No.
```

```
set.seed(669)
treebagModel <- train(CompressiveStrength ~., data = trainingSet, method = "treebag",
                      trControl = controlObject)
treebagModel
```

```
## Bagged CART
##
## 745 samples
##   8 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 5 times)
## Summary of sample sizes: 669, 671, 670, 671, 670, 670, ...
## Resampling results:
```

```
##
##   RMSE      Rsquared   MAE
##   7.523585  0.7966815  6.002134
```

```r
set.seed(669)
rfModel <- train(CompressiveStrength ~., data = trainingSet, method = "rf", tuneLength = 10,
                 ntrees = 1000, importance = TRUE, trControl = controlObject)
```

```
## note: only 7 unique complexity parameters in default grid. Truncating the grid to 7 .
```

```r
rfModel
```

```
## Random Forest
##
## 745 samples
##   8 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 5 times)
## Summary of sample sizes: 669, 671, 670, 671, 670, 670, ...
## Resampling results across tuning parameters:
##
##   mtry  RMSE      Rsquared   MAE
##   2     5.725210  0.8967061  4.180267
##   3     5.283863  0.9057512  3.804914
##   4     5.174175  0.9061300  3.704644
##   5     5.156886  0.9050625  3.682797
##   6     5.155618  0.9040789  3.679515
##   7     5.169286  0.9029358  3.687142
##   8     5.202152  0.9013393  3.705115
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was mtry = 6.
```

```r
gbmGrid <- expand.grid(.interaction.depth = seq(1,7,by=2), .n.trees = seq(100,1000,by=50), .shrinkage =
set.seed(669)
gbmModel <- train(CompressiveStrength ~., data = trainingSet, method = "gbm", tuneGrid = gbmGrid,
                  verbose = FALSE, trControl = controlObject)
gbmModel
```

```
## Stochastic Gradient Boosting
##
## 745 samples
##   8 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 5 times)
## Summary of sample sizes: 669, 671, 670, 671, 670, 670, ...
## Resampling results across tuning parameters:
##
##   shrinkage  interaction.depth  n.trees  RMSE       Rsquared   MAE
##   0.01       1                   100     13.408232  0.6551488  10.802714
```

```
##    0.01         1                     150     12.388832  0.6812070    9.967102
##    0.01         1                     200     11.586277  0.6970829    9.323092
##    0.01         1                     250     10.924824  0.7145335    8.784200
##    0.01         1                     300     10.371396  0.7343720    8.324943
##    0.01         1                     350      9.887559  0.7506648    7.920545
##    0.01         1                     400      9.459813  0.7652262    7.556920
##    0.01         1                     450      9.085214  0.7769527    7.244847
##    0.01         1                     500      8.757975  0.7863264    6.968943
##    0.01         1                     550      8.471299  0.7944107    6.729095
##    0.01         1                     600      8.220498  0.8010423    6.514983
##    0.01         1                     650      8.006153  0.8064770    6.329914
##    0.01         1                     700      7.810619  0.8116722    6.156072
##    0.01         1                     750      7.640753  0.8158181    6.001346
##    0.01         1                     800      7.488284  0.8199939    5.861187
##    0.01         1                     850      7.350006  0.8238927    5.735941
##    0.01         1                     900      7.226585  0.8272740    5.624120
##    0.01         1                     950      7.115569  0.8302925    5.528313
##    0.01         1                    1000      7.014670  0.8330040    5.440224
##    0.01         3                     100     11.298723  0.7428430    9.078254
##    0.01         3                     150      9.929900  0.7757588    7.948132
##    0.01         3                     200      8.932354  0.7996407    7.138627
##    0.01         3                     250      8.185685  0.8176386    6.529502
##    0.01         3                     300      7.618020  0.8305676    6.052286
##    0.01         3                     350      7.181156  0.8409151    5.671332
##    0.01         3                     400      6.842858  0.8489927    5.365940
##    0.01         3                     450      6.575762  0.8557407    5.121214
##    0.01         3                     500      6.360178  0.8616245    4.918873
##    0.01         3                     550      6.192746  0.8664423    4.759564
##    0.01         3                     600      6.049515  0.8707761    4.619507
##    0.01         3                     650      5.925225  0.8748194    4.498531
##    0.01         3                     700      5.818941  0.8784049    4.397485
##    0.01         3                     750      5.725804  0.8816491    4.309829
##    0.01         3                     800      5.643151  0.8844786    4.231134
##    0.01         3                     850      5.570776  0.8869623    4.163104
##    0.01         3                     900      5.508126  0.8891151    4.104117
##    0.01         3                     950      5.452754  0.8910455    4.050294
##    0.01         3                    1000      5.402795  0.8927974    4.001278
##    0.01         5                     100     10.432017  0.7810479    8.372100
##    0.01         5                     150      8.972824  0.8111337    7.174050
##    0.01         5                     200      7.969023  0.8319335    6.353242
##    0.01         5                     250      7.247654  0.8471989    5.741123
##    0.01         5                     300      6.740330  0.8580292    5.293329
##    0.01         5                     350      6.364194  0.8670204    4.948964
##    0.01         5                     400      6.086594  0.8740260    4.690157
##    0.01         5                     450      5.865748  0.8802347    4.476362
##    0.01         5                     500      5.691967  0.8853448    4.303288
##    0.01         5                     550      5.554800  0.8894653    4.164922
##    0.01         5                     600      5.438684  0.8931443    4.048801
##    0.01         5                     650      5.340219  0.8962985    3.954099
##    0.01         5                     700      5.257575  0.8989510    3.873151
##    0.01         5                     750      5.188228  0.9012040    3.805527
##    0.01         5                     800      5.128197  0.9032209    3.745285
##    0.01         5                     850      5.073455  0.9050600    3.691094
##    0.01         5                     900      5.022743  0.9068160    3.639243
```

```
## 0.01 5 950 4.978475 0.9083151 3.594233
## 0.01 5 1000 4.937258 0.9096870 3.553904
## 0.01 7 100 9.953795 0.8072899 7.987481
## 0.01 7 150 8.445990 0.8323543 6.754113
## 0.01 7 200 7.445253 0.8505726 5.909038
## 0.01 7 250 6.761729 0.8636591 5.313501
## 0.01 7 300 6.288165 0.8734810 4.882814
## 0.01 7 350 5.945504 0.8813680 4.562857
## 0.01 7 400 5.693214 0.8877733 4.318650
## 0.01 7 450 5.493770 0.8932819 4.119809
## 0.01 7 500 5.337397 0.8978597 3.964451
## 0.01 7 550 5.216499 0.9014102 3.843997
## 0.01 7 600 5.116835 0.9044654 3.743097
## 0.01 7 650 5.031375 0.9071049 3.659292
## 0.01 7 700 4.960049 0.9093697 3.587134
## 0.01 7 750 4.896744 0.9113802 3.525169
## 0.01 7 800 4.845110 0.9130034 3.473193
## 0.01 7 850 4.802250 0.9143478 3.428754
## 0.01 7 900 4.757414 0.9157555 3.383053
## 0.01 7 950 4.717420 0.9170536 3.341280
## 0.01 7 1000 4.687209 0.9180021 3.307265
## 0.10 1 100 6.993033 0.8321831 5.419430
## 0.10 1 150 6.389181 0.8518877 4.918367
## 0.10 1 200 6.064834 0.8643196 4.667164
## 0.10 1 250 5.892307 0.8709802 4.521822
## 0.10 1 300 5.783373 0.8752711 4.434966
## 0.10 1 350 5.702220 0.8786239 4.358798
## 0.10 1 400 5.634057 0.8814566 4.297897
## 0.10 1 450 5.594558 0.8832018 4.265072
## 0.10 1 500 5.533839 0.8856826 4.210436
## 0.10 1 550 5.506477 0.8867174 4.188093
## 0.10 1 600 5.480430 0.8877689 4.162470
## 0.10 1 650 5.449938 0.8890555 4.137292
## 0.10 1 700 5.423441 0.8900809 4.111791
## 0.10 1 750 5.403334 0.8908272 4.089650
## 0.10 1 800 5.389339 0.8914356 4.076502
## 0.10 1 850 5.368391 0.8923396 4.059633
## 0.10 1 900 5.356522 0.8927975 4.050661
## 0.10 1 950 5.337204 0.8934907 4.033327
## 0.10 1 1000 5.329122 0.8938370 4.025818
## 0.10 3 100 5.511954 0.8874502 4.090608
## 0.10 3 150 5.192063 0.8992936 3.789500
## 0.10 3 200 5.020746 0.9055759 3.607616
## 0.10 3 250 4.899122 0.9098583 3.482497
## 0.10 3 300 4.814669 0.9126354 3.390953
## 0.10 3 350 4.753089 0.9148448 3.322548
## 0.10 3 400 4.710882 0.9163324 3.275263
## 0.10 3 450 4.669713 0.9177009 3.230385
## 0.10 3 500 4.638258 0.9186212 3.177885
## 0.10 3 550 4.623920 0.9190984 3.147633
## 0.10 3 600 4.606915 0.9196233 3.124190
## 0.10 3 650 4.575781 0.9206467 3.084115
## 0.10 3 700 4.570009 0.9207898 3.065095
## 0.10 3 750 4.550714 0.9214319 3.044797
```

```
##    0.10       3                    800      4.545676  0.9214937   3.029587
##    0.10       3                    850      4.528767  0.9219912   3.012004
##    0.10       3                    900      4.518706  0.9223490   2.996599
##    0.10       3                    950      4.510020  0.9225747   2.984907
##    0.10       3                   1000      4.501867  0.9228142   2.974770
##    0.10       5                    100      5.069881  0.9041655   3.683303
##    0.10       5                    150      4.842329  0.9119213   3.428780
##    0.10       5                    200      4.684920  0.9171027   3.251278
##    0.10       5                    250      4.612459  0.9194361   3.167145
##    0.10       5                    300      4.548568  0.9213030   3.092963
##    0.10       5                    350      4.512878  0.9223791   3.026909
##    0.10       5                    400      4.488575  0.9230514   2.978984
##    0.10       5                    450      4.478741  0.9233253   2.951562
##    0.10       5                    500      4.463117  0.9238548   2.930247
##    0.10       5                    550      4.459924  0.9238816   2.922005
##    0.10       5                    600      4.449217  0.9241181   2.903601
##    0.10       5                    650      4.441497  0.9242909   2.884301
##    0.10       5                    700      4.439447  0.9243698   2.877370
##    0.10       5                    750      4.434740  0.9245242   2.867685
##    0.10       5                    800      4.430784  0.9246731   2.858256
##    0.10       5                    850      4.430148  0.9247291   2.849789
##    0.10       5                    900      4.429475  0.9247547   2.846831
##    0.10       5                    950      4.431020  0.9247439   2.843908
##    0.10       5                   1000      4.431541  0.9247625   2.842844
##    0.10       7                    100      4.758041  0.9146464   3.399127
##    0.10       7                    150      4.570878  0.9206183   3.186360
##    0.10       7                    200      4.472065  0.9237591   3.069588
##    0.10       7                    250      4.423471  0.9252304   2.984222
##    0.10       7                    300      4.391501  0.9261840   2.925138
##    0.10       7                    350      4.368156  0.9269318   2.891745
##    0.10       7                    400      4.363086  0.9270675   2.867464
##    0.10       7                    450      4.358199  0.9272364   2.852076
##    0.10       7                    500      4.357564  0.9272552   2.842390
##    0.10       7                    550      4.356455  0.9272748   2.835096
##    0.10       7                    600      4.359504  0.9271056   2.829517
##    0.10       7                    650      4.361090  0.9270594   2.826787
##    0.10       7                    700      4.361784  0.9270635   2.822007
##    0.10       7                    750      4.359089  0.9272070   2.816839
##    0.10       7                    800      4.359723  0.9271281   2.810090
##    0.10       7                    850      4.361653  0.9270598   2.807575
##    0.10       7                    900      4.362211  0.9269941   2.801533
##    0.10       7                    950      4.365182  0.9268894   2.804304
##    0.10       7                   1000      4.371133  0.9266107   2.806374
##
## Tuning parameter 'n.minobsinnode' was held constant at a value of 1
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were n.trees = 550,
##  interaction.depth = 7, shrinkage = 0.1 and n.minobsinnode = 1.
```

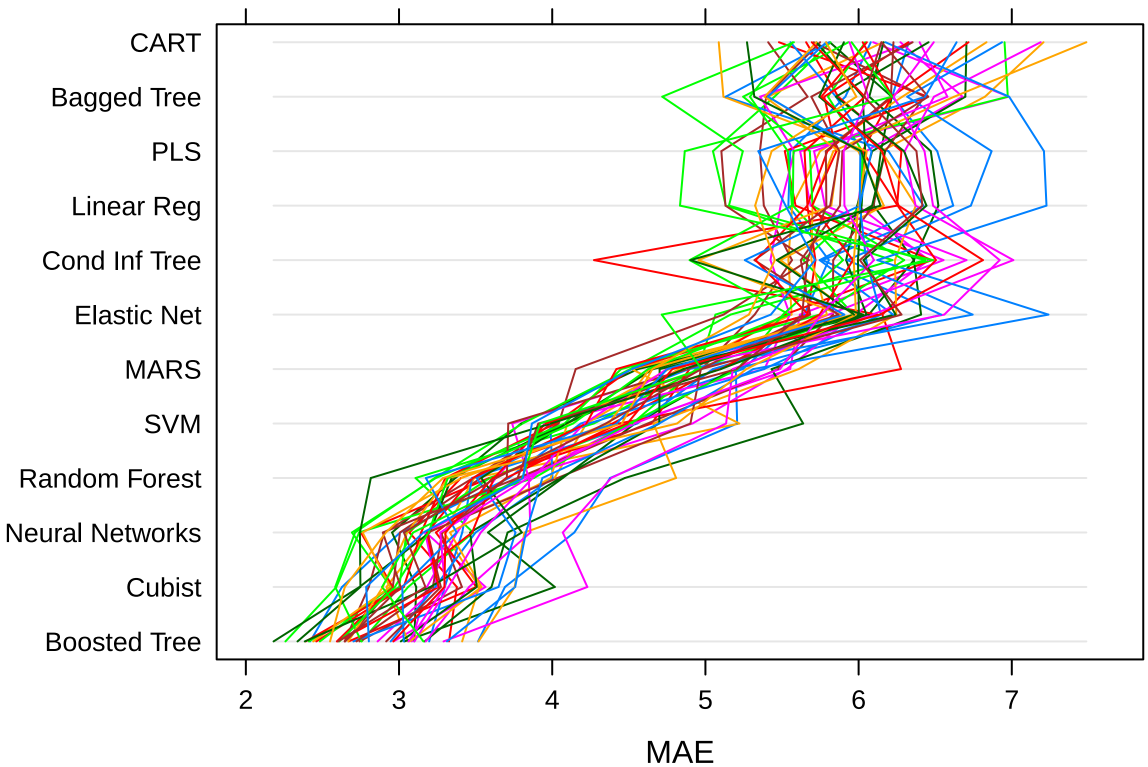
```
cubistGrid <- expand.grid(.committees = c(1,5,10,50,75,100), .neighbors=c(0,1,3,5,7,9))
set.seed(669)
cbModel <- train(CompressiveStrength ~., data = trainingSet, method = "cubist",
                 tuneGrid = cubistGrid, trControl = controlObject)
cbModel
```

```
## Cubist
##
## 745 samples
##   8 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 5 times)
## Summary of sample sizes: 669, 671, 670, 671, 670, 670, ...
## Resampling results across tuning parameters:
##
##   committees  neighbors  RMSE      Rsquared   MAE
##     1         0          6.420174  0.8463069  4.540286
##     1         1          5.897824  0.8715286  3.880708
##     1         3          5.746809  0.8768567  3.918941
##     1         5          5.798512  0.8747026  3.993180
##     1         7          5.898296  0.8703782  4.077439
##     1         9          5.986160  0.8664909  4.147584
##     5         0          5.468360  0.8881736  3.950258
##     5         1          5.069094  0.9034530  3.362232
##     5         3          4.945719  0.9075667  3.426271
##     5         5          4.965829  0.9069703  3.471306
##     5         7          5.030238  0.9044330  3.522793
##     5         9          5.099628  0.9018099  3.577718
##    10         0          5.336288  0.8939129  3.855355
##    10         1          4.973978  0.9068334  3.308117
##    10         3          4.835247  0.9116377  3.335196
##    10         5          4.853425  0.9111533  3.375375
##    10         7          4.920402  0.9086174  3.429716
##    10         9          4.990672  0.9060399  3.487260
##    50         0          5.211924  0.8992408  3.749507
##    50         1          4.785585  0.9135036  3.170088
##    50         3          4.675813  0.9172392  3.213638
##    50         5          4.703035  0.9165743  3.260615
##    50         7          4.773109  0.9140691  3.314614
##    50         9          4.847811  0.9114594  3.375176
##    75         0          5.192628  0.9001347  3.731868
##    75         1          4.761354  0.9143807  3.149852
##    75         3          4.660553  0.9178322  3.200134
##    75         5          4.688934  0.9170935  3.247248
##    75         7          4.758595  0.9146053  3.304175
##    75         9          4.832880  0.9120164  3.365018
##   100         0          5.178420  0.9006795  3.721140
##   100         1          4.754882  0.9147072  3.146039
##   100         3          4.652632  0.9181648  3.197148
##   100         5          4.680640  0.9174254  3.242286
##   100         7          4.750429  0.9149319  3.298256
##   100         9          4.824251  0.9123607  3.357949
##
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were committees = 100 and neighbors
##  = 3.
```

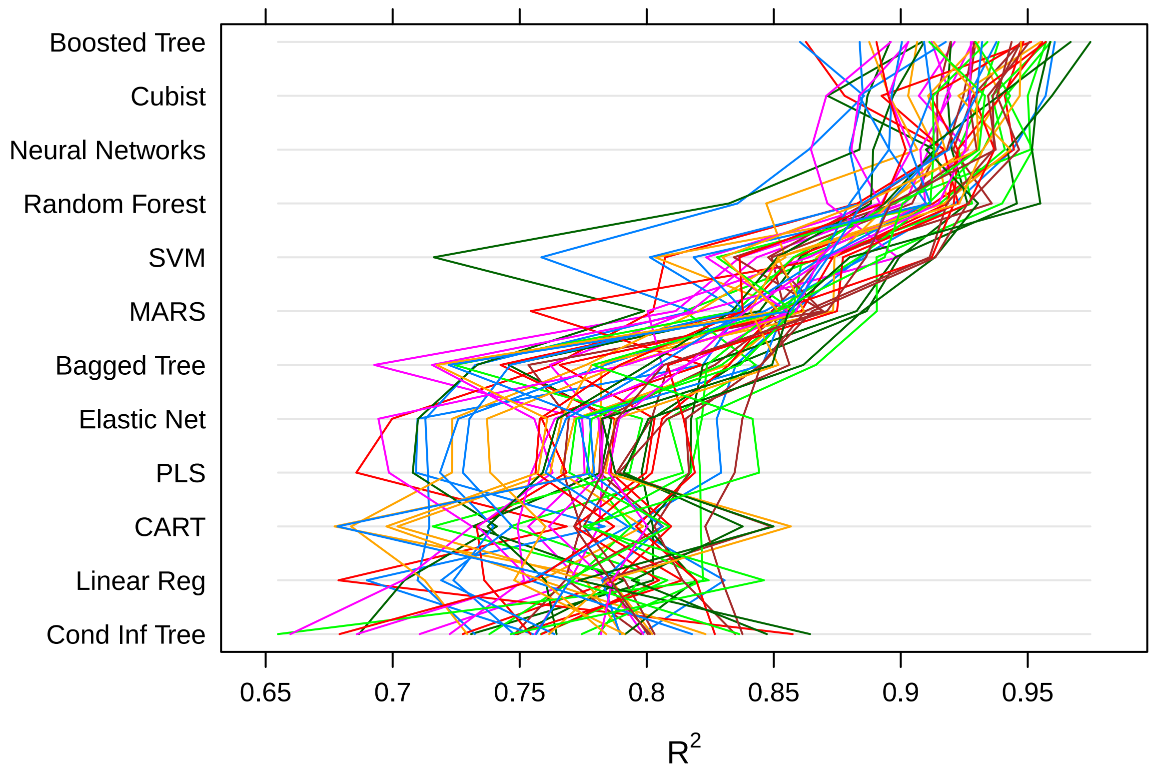
```r
allResamples <- resamples(list("Linear Reg" = linearReg,
                               "PLS" = plsModel,
```

```r
                                  "Elastic Net" = enetModel,
                                  MARS = earthModel,
                                  SVM = svmRModel,
                                  "Neural Networks" = nnetModel,
                                  CART = rpartModel,
                                  "Cond Inf Tree" = ctreeModel,
                                  "Bagged Tree" = treebagModel,
                                  "Boosted Tree" = gbmModel,
                                  "Random Forest" = rfModel,
                                  Cubist = cbModel))
```

```r
# Plot the RMSE values
library(MASS)
library(caret)
parallelplot(allResamples)
```




```r
# Using R-Squared
parallelplot(allResamples, metric = "Rsquared")
```




```
nnetPredictions <- predict(nnetModel, testData)
```

```
## Error in predict.train(nnetModel, testData): object 'testData' not found
```

```
gbmPredictions <- predict(gbmModel, testData)
```

```
## Error in predict.train(gbmModel, testData): object 'testData' not found
```

```
cbPredictions <- predict(cbModel, testData)
```

```
## Error in predict.train(cbModel, testData): object 'testData' not found
```

```
age28Data <- subset(trainingData, Age == 28)
```

```
## Error in subset(trainingData, Age == 28): object 'trainingData' not found
```

```
# Remove the age and compressive strength columns and then center and scale the predictor columns
pp1 <- preProcess(age28Data[,-(8:9)],c("center","scale"))
```

```
## Error in preProcess(age28Data[, -(8:9)], c("center", "scale")): object 'age28Data' not found
```

```
scaledTrain <- predict(pp1, age28Data[,1:7])
```

```
## Error in predict(pp1, age28Data[, 1:7]): object 'pp1' not found
```

```r
set.seed(91)
startMixture <- sample(1:nrow(age28Data),1)
```

```
## Error in nrow(age28Data): object 'age28Data' not found
```

```r
starters <- scaledTrain[startMixture, 1:7]
```

```
## Error in eval(expr, envir, enclos): object 'scaledTrain' not found
```

```r
pool <- scaledTrain
```

```
## Error in eval(expr, envir, enclos): object 'scaledTrain' not found
```

```r
index <- maxDissim(starters, pool, 14)
```

```
## Error in loadNamespace("proxy"): there is no package called 'proxy'
```

```r
startPoints <- c(startMixture, index)
```

```
## Error in eval(expr, envir, enclos): object 'startMixture' not found
```

```r
starters <- age28Data[startPoints,1:7]
```

```
## Error in eval(expr, envir, enclos): object 'age28Data' not found
```

```r
startingValues <- starters[,-4]
```

```
## Error in eval(expr, envir, enclos): object 'starters' not found
```

```r
# The inputs to the function are a vector of six mixture proportions (in argument 'x') and the model us


modelPrediction <- function(x, mod) {
  if(x[1]<0|x[1]>1) return(10^38)
  if(x[2]<0|x[2]>1) return(10^38)
  if(x[3]<0|x[3]>1) return(10^38)
  if(x[4]<0|x[4]>1) return(10^38)
  if(x[5]<0|x[5]>1) return(10^38)
  if(x[6]<0|x[6]>1) return(10^38)

  # Determine the water proportion
  x <- c(x, 1-sum(x))
  # Check the water range
  if(x[7]<0.05) return(10^38)
  # Convert the vector to a data frame, assign names and fix age at 28 days
  tmp <- as.data.frame(t(x))
  names(tmp) <- c('Cement','BlastFurnaceSlag','FlyAsh','Superplasticizer','CoarseAggregate','FineAggrega
  tmp$Age <- 28
  # Get the model prediction, square them to get back to the original units, then return the negative o
  -predict(mod, tmp)
}
```

```r
cbResults <- startingValues
```

```
## Error in eval(expr, envir, enclos): object 'startingValues' not found
```

```r
cbResults$Water <- NA
```

```
## Error in cbResults$Water <- NA: object 'cbResults' not found
```

```r
cbResults$Prediction <- NA
```

```
## Error in cbResults$Prediction <- NA: object 'cbResults' not found
```

```r
## Loop over each starting point and conduct the search
for(i in 1:nrow(cbResults))
 {
 results<-optim(unlist(cbResults[i,1:6]),
 modelPrediction,
 method = "Nelder-Mead",
 ## Use method = 'SANN' for simulated annealing
 control=list(maxit=5000),
 ## The next option is passed to the
 ## modelPrediction() function
 mod = cbModel)
 ##Savethepredictedcompressivestrength
 cbResults$Prediction[i]<--results$value
 ##Alsosavethefinalmixturevalues
 cbResults[i,1:6]<-results$par
 }
```

```
## Error in nrow(cbResults): object 'cbResults' not found
```

```r
 ## Calculate the water proportion
 cbResults$Water <- 1 - apply(cbResults[,1:6], 1, sum)
```

```
## Error in apply(cbResults[, 1:6], 1, sum): object 'cbResults' not found
```

```r
 ## Keep the top three mixtures
 cbResults <- cbResults[order(-cbResults$Prediction),][1:3,]
```

```
## Error in eval(expr, envir, enclos): object 'cbResults' not found
```

```r
 cbResults$Model <- "Cubist"
```

```
## Error in cbResults$Model <- "Cubist": object 'cbResults' not found
```

```r
nnetResults <- startingValues
```

```
## Error in eval(expr, envir, enclos): object 'startingValues' not found
```

```r
nnetResults$Water <- NA
```

```
## Error in nnetResults$Water <- NA: object 'nnetResults' not found
```

```r
nnetResults$Prediction <- NA
```

```
## Error in nnetResults$Prediction <- NA: object 'nnetResults' not found
```

```r
for(i in 1:nrow(nnetResults))
 {
 results<-optim(unlist(nnetResults[i,1:6,]),
 modelPrediction,
 method = "Nelder-Mead",
 control=list(maxit=5000),
 mod = nnetModel)
 nnetResults$Prediction[i]<--results$value
 nnetResults[i,1:6]<-results$par
 }
```

```
## Error in nrow(nnetResults): object 'nnetResults' not found
```

```r
nnetResults$Water <- 1 - apply(nnetResults[,1:6], 1, sum)
```

```
## Error in apply(nnetResults[, 1:6], 1, sum): object 'nnetResults' not found
```

```r
nnetResults <- nnetResults[order(-nnetResults$Prediction),][1:3,]
```

```
## Error in eval(expr, envir, enclos): object 'nnetResults' not found
```

```r
nnetResults$Model <- "NNet"
```

```
## Error in nnetResults$Model <- "NNet": object 'nnetResults' not found
```

```r
 ## Run PCA on the data at 28\,days
 pp2 <- preProcess(age28Data[, 1:7], "pca")
```

```
## Error in preProcess(age28Data[, 1:7], "pca"): object 'age28Data' not found
```

```r
 ## Get the components for these mixtures
 pca1 <- predict(pp2, age28Data[, 1:7])
```

```
## Error in predict(pp2, age28Data[, 1:7]): object 'pp2' not found
```

```r
 pca1$Data <- "Training Set"
```

```
## Error in pca1$Data <- "Training Set": object 'pca1' not found
```

```r
## Label which data points were used to start the searches
pca1$Data[startPoints] <- "Starting Values"
```

```
## Error in pca1$Data[startPoints] <- "Starting Values": object 'pca1' not found
```

```r
## Project the new mixtures in the same way (making sure to
## re-order the columns to match the order of the age28Data object).
pca3 <- predict(pp2, cbResults[, names(age28Data[, 1:7])])
```

```
## Error in predict(pp2, cbResults[, names(age28Data[, 1:7])]): object 'pp2' not found
```

```r
pca3$Data <- "Cubist"
```

```
## Error in pca3$Data <- "Cubist": object 'pca3' not found
```

```r
pca4 <- predict(pp2, nnetResults[, names(age28Data[, 1:7])])
```

```
## Error in predict(pp2, nnetResults[, names(age28Data[, 1:7])]): object 'pp2' not found
```

```r
pca4$Data <- "Neural Network"
```

```
## Error in pca4$Data <- "Neural Network": object 'pca4' not found
```

```r
## Combine the data, determine the axis ranges and plot
pcaData <- rbind(pca1, pca3, pca4)
```

```
## Error in rbind(pca1, pca3, pca4): object 'pca1' not found
```

```r
pcaData$Data <- factor(pcaData$Data,
levels = c("Training Set","Starting Values",
"Cubist","Neural Network"))
```

```
## Error in factor(pcaData$Data, levels = c("Training Set", "Starting Values", : object 'pcaData' not fo
```

```r
lim <- extendrange(pcaData[, 1:2])
```

```
## Error in extendrange(pcaData[, 1:2]): object 'pcaData' not found
```

```r
xyplot(PC2 ~ PC1, data = pcaData, groups = Data,
auto.key = list(columns = 2),
xlim = lim, ylim = lim,
type = c("g", "p"))
```

```
## Error in eval(substitute(groups), data, environment(x)): object 'pcaData' not found
```