Test4

Mao Soldevilla

4/10/2020

```
library(caret)
library(ggplot2)
library(AppliedPredictiveModeling)
library(rattle)
library(dplyr)
library(forecast)
library(gbm)
```

```
FALSE Warning: package 'gbm' was built under R version 4.0.2
```

```
FALSE Loaded gbm 2.1.8
```

1. For this quiz we will be using several R packages. R package versions change over time, the right answers have been checked using the following versions of the packages.

AppliedPredictiveModeling: v1.1.6 caret: v6.0.47 ElemStatLearn: v2012.04-0 pgmm: v1.1 rpart: v4.1.8 gbm: v2.1 lubridate: v1.3.3 forecast: v5.6 e1071: v1.6.4 If you aren't using these versions of the packages, your answers may not exactly match the right answer, but hopefully should be close.

Load the vowel.train and vowel.test data sets:

```
#library(ElemStatLearn)
#data(vowel.train)
#data(vowel.test)
#due to ElemStatLearn was retired from CRAN, we download from other sources
vowel.train <- read.csv("https://web.stanford.edu/~hastie/ElemStatLearn/datasets/vowel.train")
vowel.test <- read.csv("https://web.stanford.edu/~hastie/ElemStatLearn/datasets/vowel.test")
```

Cleaning data

```
str(vowel.train)
```

```
## 'data.frame':    528 obs. of  12 variables:
##  $ row.names: int  1 2 3 4 5 6 7 8 9 10 ...
##  $ y        : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ x.1      : num  -3.64 -3.33 -2.12 -2.29 -2.6 ...
##  $ x.2      : num  0.418 0.496 0.894 1.809 1.938 ...
##  $ x.3      : num  -0.67 -0.694 -1.576 -1.498 -0.846 ...
```

```
##  $ x.4      : num  1.779 1.365 0.147 1.012 1.062 ...
##  $ x.5      : num  -0.168 -0.265 -0.707 -1.053 -1.633 ...
##  $ x.6      : num  1.627 1.933 1.559 1.06 0.764 ...
##  $ x.7      : num  -0.388 -0.363 -0.579 -0.567 0.394 0.217 0.322 -0.435 -0.512 -0.466 ...
##  $ x.8      : num  0.529 0.51 0.676 0.235 -0.15 -0.246 0.45 0.992 0.928 0.702 ...
##  $ x.9      : num  -0.874 -0.621 -0.809 -0.091 0.277 0.238 0.377 0.575 -0.167 0.06 ...
##  $ x.10     : num  -0.814 -0.488 -0.049 -0.795 -0.396 -0.365 -0.366 -0.301 -0.434 -0.836 ...
```

```
names(vowel.test)
```

```
##  [1] "row.names" "y"         "x.1"       "x.2"       "x.3"       "x.4"
##  [7] "x.5"       "x.6"       "x.7"       "x.8"       "x.9"       "x.10"
```

```
vowel.test <- vowel.test[, -1]
vowel.train <- vowel.train[, -1]
vowel.test$y <- as.factor(vowel.test$y)
vowel.train$y <- as.factor(vowel.train$y)
head(vowel.train)
```

```
##   y    x.1   x.2    x.3   x.4    x.5   x.6    x.7    x.8    x.9   x.10
## 1 1 -3.639 0.418 -0.670 1.779 -0.168 1.627 -0.388  0.529 -0.874 -0.814
## 2 2 -3.327 0.496 -0.694 1.365 -0.265 1.933 -0.363  0.510 -0.621 -0.488
## 3 3 -2.120 0.894 -1.576 0.147 -0.707 1.559 -0.579  0.676 -0.809 -0.049
## 4 4 -2.287 1.809 -1.498 1.012 -1.053 1.060 -0.567  0.235 -0.091 -0.795
## 5 5 -2.598 1.938 -0.846 1.062 -1.633 0.764  0.394 -0.150  0.277 -0.396
## 6 6 -2.852 1.914 -0.755 0.825 -1.588 0.855  0.217 -0.246  0.238 -0.365
```

```
head(vowel.test)
```

```
##   y    x.1    x.2    x.3    x.4    x.5   x.6    x.7   x.8    x.9   x.10
## 1 1 -1.149 -0.904 -1.988  0.739 -0.060 1.206  0.864 1.196 -0.300 -0.467
## 2 2 -2.613 -0.092 -0.540  0.484  0.389 1.741  0.198 0.257 -0.375 -0.604
## 3 3 -2.505  0.632 -0.593  0.304  0.496 0.824 -0.162 0.181 -0.363 -0.764
## 4 4 -1.768  1.769 -1.142 -0.739 -0.086 0.120 -0.230 0.217 -0.009 -0.279
## 5 5 -2.671  3.155 -0.514  0.133 -0.964 0.234 -0.071 1.192  0.254 -0.471
## 6 6 -2.509  1.326  0.354  0.663 -0.724 0.418 -0.496 0.713  0.638 -0.204
```

Set the variable y to be a factor variable in both the training and test set. Then set the seed to 33833. Fit (1) a random forest predictor relating the factor variable y to the remaining variables and (2) a boosted predictor using the "gbm" method. Fit these both with the train() command in the caret package.

```
modFitrf <- train(y ~ ., data = vowel.train, method = "rf", prox = TRUE)
#modFitrf <- randomForest::randomForest(y ~ ., data = vowel.train)
tc = trainControl(method = "cv", number=10)
modFitgbm <- train(y ~ ., data = vowel.train, method = "gbm", trControl = tc, verbose = FALSE)
set.seed(33833)
predrf <- predict(modFitrf, newdata = vowel.test)
predgbm <- predict(modFitgbm, newdata = vowel.test)
```

```
combDF <- data.frame(predrf, predgbm, y = vowel.test$y)
str(combDF)
```

```
## 'data.frame':    462 obs. of  3 variables:
##  $ predrf : Factor w/ 11 levels "1","2","3","4",..: 1 2 3 4 5 6 7 8 9 1 ...
##  $ predgbm: Factor w/ 11 levels "1","2","3","4",..: 1 2 6 4 5 6 7 8 9 10 ...
##  $ y      : Factor w/ 11 levels "1","2","3","4",..: 1 2 3 4 5 6 7 8 9 10 ...
```

```
modFitcomb <- train(y ~ ., data = combDF)
predcomb <- predict(modFitcomb, newdata = vowel.test)
```

What are the accuracies for the two approaches on the test data set? What is the accuracy among the test set samples where the two methods agree?

Testing erros

```
rf <- confusionMatrix(predrf, vowel.test$y)
rf
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  1  2  3  4  5  6  7  8  9 10 11
##         1  29  0  0  0  0  0  0  0  0  2  0
##         2  12 26  3  0  0  0  0  0  0 19  1
##         3   1 12 32  3  0  0  0  0  0  0  0
##         4   0  0  3 29  1  1  0  0  0  0  2
##         5   0  0  0  0 16  6  6  0  0  0  0
##         6   0  0  4  9 21 24  5  0  0  0  6
##         7   0  0  0  0  3  0 28  7  5  0  3
##         8   0  0  0  0  0  0  0 30  5  0  0
##         9   0  4  0  0  0  0  3  5 24  4 12
##         10  0  0  0  0  0  0  0  0  2 17  0
##         11  0  0  0  1  1 11  0  0  6  0 18
##
## Overall Statistics
##
##                Accuracy : 0.5909
##                  95% CI : (0.5445, 0.6361)
##     No Information Rate : 0.0909
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.55
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 1 Class: 2 Class: 3 Class: 4 Class: 5 Class: 6
## Sensitivity           0.69048  0.61905  0.76190  0.69048  0.38095  0.57143
```

```
## Specificity           0.99524  0.91667  0.96190  0.98333  0.97143  0.89286
## Pos Pred Value        0.93548  0.42623  0.66667  0.80556  0.57143  0.34783
## Neg Pred Value        0.96984  0.96010  0.97585  0.96948  0.94009  0.95420
## Prevalence            0.09091  0.09091  0.09091  0.09091  0.09091  0.09091
## Detection Rate        0.06277  0.05628  0.06926  0.06277  0.03463  0.05195
## Detection Prevalence  0.06710  0.13203  0.10390  0.07792  0.06061  0.14935
## Balanced Accuracy     0.84286  0.76786  0.86190  0.83690  0.67619  0.73214
##                      Class: 7 Class: 8 Class: 9 Class: 10 Class: 11
## Sensitivity           0.66667  0.71429  0.57143   0.40476   0.42857
## Specificity           0.95714  0.98810  0.93333   0.99524   0.95476
## Pos Pred Value        0.60870  0.85714  0.46154   0.89474   0.48649
## Neg Pred Value        0.96635  0.97190  0.95610   0.94357   0.94353
## Prevalence            0.09091  0.09091  0.09091   0.09091   0.09091
## Detection Rate        0.06061  0.06494  0.05195   0.03680   0.03896
## Detection Prevalence  0.09957  0.07576  0.11255   0.04113   0.08009
## Balanced Accuracy     0.81190  0.85119  0.75238   0.70000   0.69167
```

```
gbm <- confusionMatrix(predgbm, vowel.test$y)
gbm
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  1  2  3  4  5  6  7  8  9 10 11
##         1  30  0  0  0  0  0  0  0  0  2  0
##         2   9 20  1  0  0  0  1  0  0 12  0
##         3   1 11  8  3  0  0  0  0  0  0  0
##         4   0  1  8 22  3  0  1  0  0  0  3
##         5   0  0  0  4 19  7  1  0  0  0  0
##         6   0  1 16 12 10 29  0  0  1  0  9
##         7   0  0  1  0  6  0 37  6  5  0  9
##         8   0  0  0  0  0  0  2 31  7  1  0
##         9   0  7  0  0  0  0  0  5 29  8 17
##         10  2  0  0  0  0  0  0  0  0 19  0
##         11  0  2  8  1  4  6  0  0  0  0  4
##
## Overall Statistics
##
##                Accuracy : 0.5368
##                  95% CI : (0.4901, 0.583)
##     No Information Rate : 0.0909
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.4905
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 1 Class: 2 Class: 3 Class: 4 Class: 5 Class: 6
## Sensitivity           0.71429  0.47619  0.19048  0.52381  0.45238  0.69048
## Specificity           0.99524  0.94524  0.96429  0.96190  0.97143  0.88333
## Pos Pred Value        0.93750  0.46512  0.34783  0.57895  0.61290  0.37179
## Neg Pred Value        0.97209  0.94749  0.92255  0.95283  0.94664  0.96615
```

```
## Prevalence             0.09091  0.09091  0.09091  0.09091  0.09091  0.09091
## Detection Rate         0.06494  0.04329  0.01732  0.04762  0.04113  0.06277
## Detection Prevalence   0.06926  0.09307  0.04978  0.08225  0.06710  0.16883
## Balanced Accuracy      0.85476  0.71071  0.57738  0.74286  0.71190  0.78690
##                       Class: 7 Class: 8 Class: 9 Class: 10 Class: 11
## Sensitivity            0.88095  0.73810  0.69048   0.45238  0.095238
## Specificity            0.93571  0.97619  0.91190   0.99524  0.950000
## Pos Pred Value         0.57813  0.75610  0.43939   0.90476  0.160000
## Neg Pred Value         0.98744  0.97387  0.96717   0.94785  0.913043
## Prevalence             0.09091  0.09091  0.09091   0.09091  0.090909
## Detection Rate         0.08009  0.06710  0.06277   0.04113  0.008658
## Detection Prevalence   0.13853  0.08874  0.14286   0.04545  0.054113
## Balanced Accuracy      0.90833  0.85714  0.80119   0.72381  0.522619
```

```
comb <- confusionMatrix(predcomb, vowel.test$y)
comb
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  1  2  3  4  5  6  7  8  9 10 11
##         1  30  0  0  0  0  0  0  0  0  2  0
##         2  10 36 10  3  0  0  0  0  0 15  1
##         3   0  2 26  0  0  0  0  0  0  0  0
##         4   0  0  2 29  1  1  0  0  0  0  1
##         5   0  0  2  3 24  5  1  0  0  0  1
##         6   0  0  2  7 11 32  0  0  0  0  7
##         7   0  0  0  0  6  0 41  6  5  0  4
##         8   0  0  0  0  0  0  0 32  7  0  0
##         9   0  4  0  0  0  0  0  4 29  5 16
##         10  2  0  0  0  0  0  0  0  0 20  0
##         11  0  0  0  0  0  4  0  0  1  0 12
##
## Overall Statistics
##
##                Accuracy : 0.6732
##                  95% CI : (0.6283, 0.7158)
##     No Information Rate : 0.0909
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.6405
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 1 Class: 2 Class: 3 Class: 4 Class: 5 Class: 6
## Sensitivity           0.71429  0.85714  0.61905  0.69048  0.57143  0.76190
## Specificity           0.99524  0.90714  0.99524  0.98810  0.97143  0.93571
## Pos Pred Value        0.93750  0.48000  0.92857  0.85294  0.66667  0.54237
## Neg Pred Value        0.97209  0.98450  0.96313  0.96963  0.95775  0.97519
## Prevalence            0.09091  0.09091  0.09091  0.09091  0.09091  0.09091
## Detection Rate        0.06494  0.07792  0.05628  0.06277  0.05195  0.06926
## Detection Prevalence  0.06926  0.16234  0.06061  0.07359  0.07792  0.12771
```

```
## Balanced Accuracy     0.85476  0.88214  0.80714  0.83929  0.77143  0.84881
##                      Class: 7 Class: 8 Class: 9 Class: 10 Class: 11
## Sensitivity           0.97619  0.76190  0.69048   0.47619   0.28571
## Specificity           0.95000  0.98333  0.93095   0.99524   0.98810
## Pos Pred Value        0.66129  0.82051  0.50000   0.90909   0.70588
## Neg Pred Value        0.99750  0.97636  0.96782   0.95000   0.93258
## Prevalence            0.09091  0.09091  0.09091   0.09091   0.09091
## Detection Rate        0.08874  0.06926  0.06277   0.04329   0.02597
## Detection Prevalence  0.13420  0.08442  0.12554   0.04762   0.03680
## Balanced Accuracy     0.96310  0.87262  0.81071   0.73571   0.63690
```

Answer

Random Forest accuracy: 0.5909091

Boosted predictor accuracy: 0.5367965

Combined predictor accuracy: 0.6731602

2. Load the Alzheimer's data using the following commands

```
#library(caret)
#library(gbm)
set.seed(3433)
#library(AppliedPredictiveModeling)
data(AlzheimerDisease)
adData = data.frame(diagnosis,predictors)
inTrain = createDataPartition(adData$diagnosis, p = 3/4)[[1]]
training = adData[ inTrain,]
testing = adData[-inTrain,]
dim(training)
```

```
## [1] 251 131
```

```
dim(testing)
```

```
## [1]  82 131
```

```
unique(training$diagnosis)
```

```
## [1] Control  Impaired
## Levels: Impaired Control
```

Set the seed to 62433 and predict diagnosis with all the other variables using a random forest ("rf"), boosted trees ("gbm") and linear discriminant analysis ("lda") model. Stack the predictions together using random forests ("rf"). What is the resulting accuracy on the test set? Is it better or worse than each of the individual predictions?

Training

```r
set.seed(62433)
modFitrf <- train(diagnosis ~ ., data = training, method = "rf", prox = TRUE)
modFitrf
```

```
## Random Forest
##
## 251 samples
## 130 predictors
##   2 classes: 'Impaired', 'Control'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 251, 251, 251, 251, 251, 251, ...
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##     2   0.7618341  0.1807322
##    68   0.8056679  0.4703519
##   134   0.7936095  0.4540126
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 68.
```

```r
tc = trainControl(method = "cv", number = 10)
modFitgbm <- train(diagnosis ~ ., data = training, method = "gbm", trControl = tc, verbose = FALSE)
modFitgbm
```

```
## Stochastic Gradient Boosting
##
## 251 samples
## 130 predictors
##   2 classes: 'Impaired', 'Control'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 226, 225, 226, 225, 226, 226, ...
## Resampling results across tuning parameters:
##
##   interaction.depth  n.trees  Accuracy   Kappa
##   1                   50      0.7970256  0.4160158
##   1                  100      0.7966923  0.4376535
##   1                  150      0.8008590  0.4485518
##   2                   50      0.8010128  0.4419677
##   2                  100      0.8050128  0.4651078
##   2                  150      0.8091795  0.4671649
##   3                   50      0.8126923  0.4830656
##   3                  100      0.7928462  0.4276544
##   3                  150      0.8248590  0.5311792
##
## Tuning parameter 'shrinkage' was held constant at a value of 0.1
##
## Tuning parameter 'n.minobsinnode' was held constant at a value of 10
```

```
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were n.trees = 150, interaction.depth =
##  3, shrinkage = 0.1 and n.minobsinnode = 10.
```

```
tc0 <- trainControl(method = "cv")
modFitlda <- train(diagnosis ~ ., data = training, method = "lda", trcontrol = tc0)
```

```
## Warning in lda.default(x, grouping, ...): variables are collinear
```

```
## Warning in lda.default(x, grouping, ...): variables are collinear
```

```
## Warning: model fit failed for Resample21: parameter=none Error in lda.default(x, grouping, ...) :
##   variable 131 appears to be constant within groups
```

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

```
modFitlda
```

```
## Linear Discriminant Analysis
##
## 251 samples
## 130 predictors
##   2 classes: 'Impaired', 'Control'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 251, 251, 251, 251, 251, 251, ...
## Resampling results:
##
##   Accuracy   Kappa
##   0.6560956  0.2455014
```

Testing

```
predrf <- predict(modFitrf, testing)
predgbm <- predict(modFitgbm, testing)
predlda <- predict(modFitlda, testing)
predDF <- data.frame(predrf, predgbm, predlda, diagnosis = testing$diagnosis)
modFitcomb <- train(diagnosis ~ ., data = predDF, method = "rf")
```

```
## note: only 2 unique complexity parameters in default grid. Truncating the grid to 2 .
```

```
modFitcomb
```

```
## Random Forest
##
## 82 samples
##  3 predictor
```

```
##  2 classes: 'Impaired', 'Control'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 82, 82, 82, 82, 82, 82, ...
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##   2     0.8957025  0.7228913
##   3     0.8902355  0.7063168
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 2.
```

```
predcomb <- predict(modFitcomb, testing)
```

Accuracy

```
table(predrf, testing$diagnosis)
```

```
##
## predrf     Impaired Control
##   Impaired       14       0
##   Control         8      60
```

```
confrf <- confusionMatrix(predrf, testing$diagnosis)
table(predgbm, testing$diagnosis)
```

```
##
## predgbm    Impaired Control
##   Impaired       15       0
##   Control         7      60
```

```
confgbm <- confusionMatrix(predgbm, testing$diagnosis)
table(predlda, testing$diagnosis)
```

```
##
## predlda    Impaired Control
##   Impaired       19       4
##   Control         3      56
```

```
conflda<- confusionMatrix(predlda, testing$diagnosis)
table(predcomb, testing$diagnosis)
```

```
##
## predcomb   Impaired Control
##   Impaired       16       0
##   Control         6      60
```

```
confcomb <- confusionMatrix(predcomb, testing$diagnosis)
```

Answer

The results are: Random forest Accuracy = 0.902439

boosted trees Accuracy = 0.9146341

linear discriminant analysis accuracy = 0.9146341

combined accuracy = 0.9268293

for then the combined accuracy is better than others.

3. Load the concrete data with the commands:

```
set.seed(3523)
library(AppliedPredictiveModeling)
data(concrete)
inTrain = createDataPartition(concrete$CompressiveStrength, p = 3/4)[[1]]
training = concrete[ inTrain,]
testing = concrete[-inTrain,]
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 ...
```

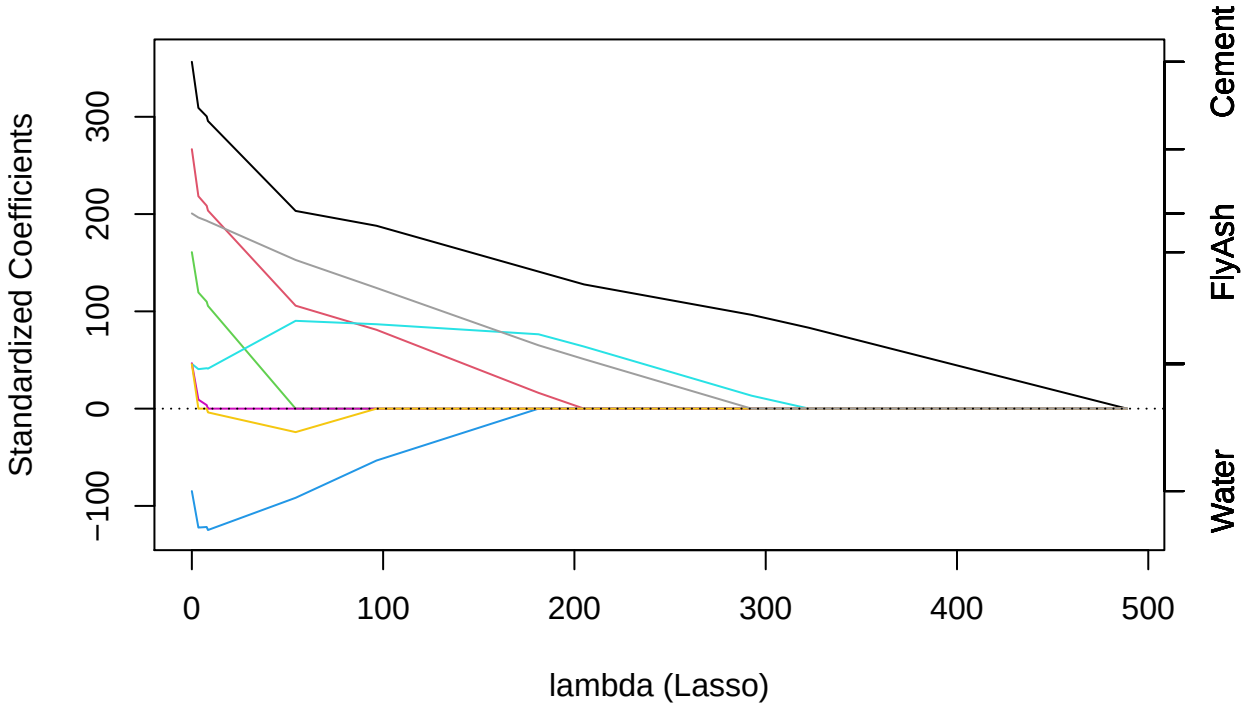
Set the seed to 233 and fit a lasso model to predict Compressive Strength. Which variable is the last coefficient to be set to zero as the penalty increases? (Hint: it may be useful to look up ?plot.enet).

```
library(elasticnet)
```

```
## Loading required package: lars
```

```
## Loaded lars 1.2
```

```
set.seed(233)
fitMod <- train(CompressiveStrength ~ ., data =  training, method = "lasso")
plot.enet(fitMod$finalModel, xvar = "penalty", use.color = TRUE)
```



According the plot, the last coefficient to be set to zero is Cement.

4. Load the data on the number of visitors to the instructors blog from here:

https://d396qusza40orc.cloudfront.net/predmachlearn/gaData.csv

Using the commands:

```
library(lubridate) # For year() function below
```

```
##
## Attaching package: 'lubridate'
```

```
## The following objects are masked from 'package:base':
##
##     date, intersect, setdiff, union
```

```
url <- "https://d396qusza40orc.cloudfront.net/predmachlearn/gaData.csv"
dat = read.csv(url)
training = dat[year(dat$date) < 2012,]
testing = dat[(year(dat$date)) > 2011,]
tstrain = ts(training$visitsTumblr)
```

Fit a model using the bats() function in the forecast package to the training time series. Then forecast this model for the remaining time points. For how many of the testing points is the true value within the 95% prediction interval bounds?

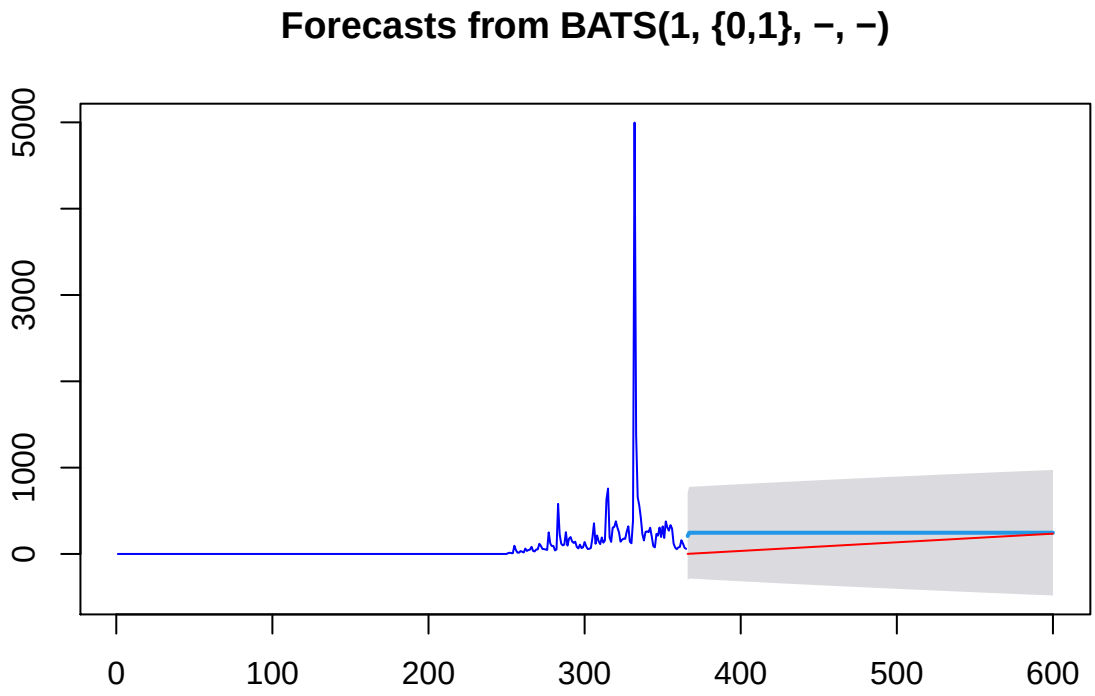
Data Cleaning

```
str(training)
```

```
## 'data.frame':    365 obs. of  3 variables:
##  $ X           : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ date        : chr  "2011-01-01" "2011-01-02" "2011-01-03" "2011-01-04" ...
##  $ visitsTumblr: int  0 0 0 0 0 0 0 0 0 0 ...
```

Fitting

```
modFit <- bats(tstrain)
fcast <- forecast(modFit, level = 95, h = nrow(testing))
plot(fcast, col = "blue")
lines(testing, col = "red")
```




```
accuracy(fcast, testing$visitsTumblr)
```

```
##                     ME     RMSE       MAE      MPE   MAPE      MASE        ACF1
## Training set  10.49059 258.7813  40.17169     -Inf    Inf 0.8425522 0.01240395
## Test set     174.66089 294.9901 199.92648 22.73707 40.045 4.1932146         NA
```

```
count <- 0
for (i in 1:nrow(testing)) {
        if (testing$visitsTumblr[i] > fcast$lower[i]) {
                if(testing$visitsTumblr[i] < fcast$upper[i]) {
                count <- count + 1}
        }
}
acc <- count/nrow(testing)
```

The accuracy is 0.9617021

5. Load the concrete data with the commands:

```
set.seed(3523)
library(AppliedPredictiveModeling)
data(concrete)
inTrain = createDataPartition(concrete$CompressiveStrength, p = 3/4)[[1]]
training = concrete[ inTrain,]
testing = concrete[-inTrain,]
```

Set the seed to 325 and fit a support vector machine using the e1071 package to predict Compressive Strength using the default settings. Predict on the testing set. What is the RMSE?

Fitting

```
set.seed(325)
library(e1071)
```

```
## Warning: package 'e1071' was built under R version 4.0.2
```

```
#fitMod <- train(CompressiveStrength ~ ., data =  training, method = "e1071")
fitMod <- svm(CompressiveStrength ~ ., data =  training)
```

Predicting

```
pred <- predict(fitMod, testing)
summary(pred)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   11.05   24.08   34.77   35.43   46.80   74.27
```

```
sqrt(sum((pred - testing$CompressiveStrength)^2) / nrow(testing))
```

```
## [1] 7.962075
```