Lasso regression modelling in R

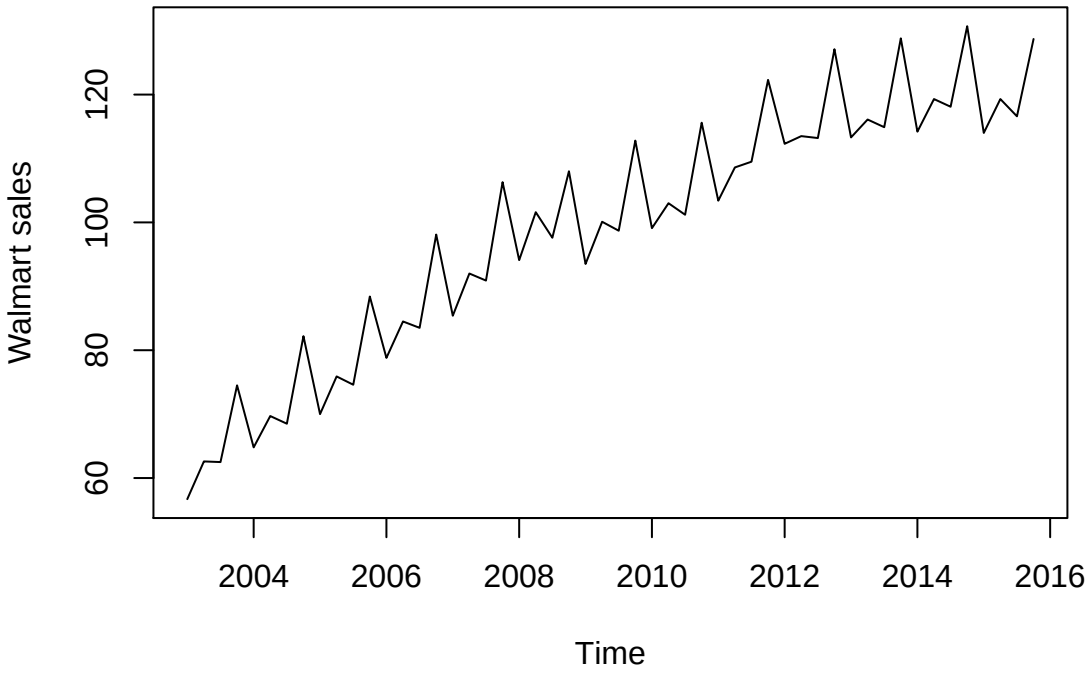
Nikolaos Kourentzes (nikolaos@kourentes.com)

Contents

1 Load dataset

First, we will load some data. We will use the quarterly Walmart sales.

```
x <- ts(read.csv("./Lab3data3.csv"),frequency=4,start=c(2003,1))
plot(x[,1],ylab="Walmart sales")
```



2 Build a benchmark regression

This section is a summary of the advanced regression part in the previous lab. We go through this to have a benchmark for the lasso regression, and also to construct lags as needed.

We will retain the last 4 years as a test set and use the rest to build the models.

```
y.trn <- window(x[,1],end=c(2013,4))
y.tst <- window(x[,1],start=c(2014,1))
```

We construct lags, as in the previous workshop. First, we find how many observations we have.

```
n <- length(y.trn)
X <- array(NA,c(n,6))
# Loop to create lags
for (i in 1:6){
  X[i:n,i] <- y.trn[1:(n-i+1)]
}
# Name the columns
colnames(X) <- c("y",paste0("lag",1:5))
X <- as.data.frame(X)
head(X)
```

```
##      y lag1 lag2 lag3 lag4 lag5
## 1 56.7   NA   NA   NA   NA   NA
## 2 62.6 56.7   NA   NA   NA   NA
## 3 62.5 62.6 56.7   NA   NA   NA
## 4 74.5 62.5 62.6 56.7   NA   NA
## 5 64.8 74.5 62.5 62.6 56.7   NA
## 6 69.7 64.8 74.5 62.5 62.6 56.7
```

We build a stepwise regression. To do that we start from the complete model first (`fit1`) and then find the stepwise solution (`fit2`)

```
# The complete model
fit1 <- lm(y~.,data=X)
# The stepwise model
fit2 <- step(fit1)
```

```
## Start:  AIC=58.18
## y ~ lag1 + lag2 + lag3 + lag4 + lag5
##
##        Df Sum of Sq     RSS     AIC
## - lag2  1      0.35  127.79  56.286
## - lag3  1      1.28  128.71  56.567
## <none>               127.44  58.178
## - lag5  1    105.04  232.48  79.624
## - lag1  1    110.17  237.61  80.475
## - lag4  1   1590.97 1718.40 157.638
##
## Step:  AIC=56.29
## y ~ lag1 + lag3 + lag4 + lag5
##
##        Df Sum of Sq     RSS     AIC
## - lag3  1      1.51  129.29  54.743
## <none>               127.79  56.286
## - lag5  1    104.99  232.78  77.674
## - lag1  1    111.14  238.92  78.691
## - lag4  1   2717.34 2845.12 175.302
##
```

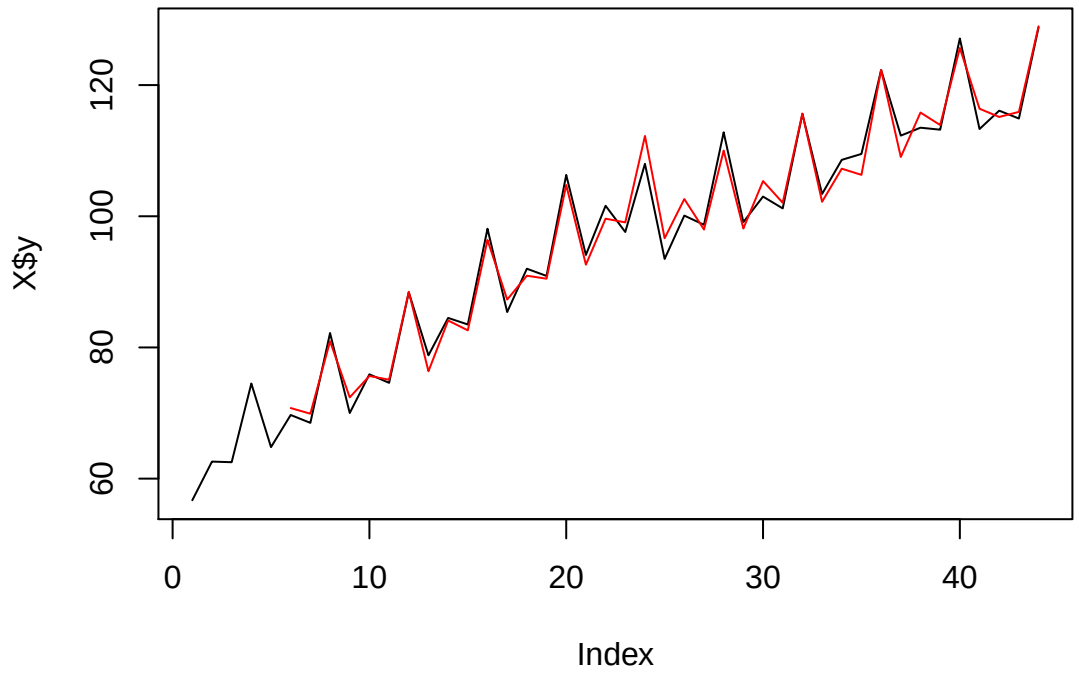
```
## Step:  AIC=54.74
## y ~ lag1 + lag4 + lag5
##
##        Df Sum of Sq     RSS     AIC
## <none>               129.29  54.743
## - lag1  1    110.09  239.39  76.766
## - lag5  1    116.20  245.50  77.749
## - lag4  1   2910.88 3040.17 175.888
```

```
summary(fit2)
```

```
##
## Call:
## lm(formula = y ~ lag1 + lag4 + lag5, data = X)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -4.2420 -1.2261  0.2523  1.3036  3.2640
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   4.5873     2.4497   1.873   0.0695 .
## lag1          0.6783     0.1242   5.459 3.99e-06 ***
## lag4          0.9824     0.0350  28.071  < 2e-16 ***
## lag5         -0.6927     0.1235  -5.609 2.53e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.922 on 35 degrees of freedom
##   (5 observations deleted due to missingness)
## Multiple R-squared:  0.9868, Adjusted R-squared:  0.9856
## F-statistic: 870.6 on 3 and 35 DF,  p-value: < 2.2e-16
```

And here is the model fit.

```
# In-sample fit:
plot(X$y,type="l")
frc <- predict(fit2,X)
lines(frc,col="red")
```



Let us produce some forecasts.

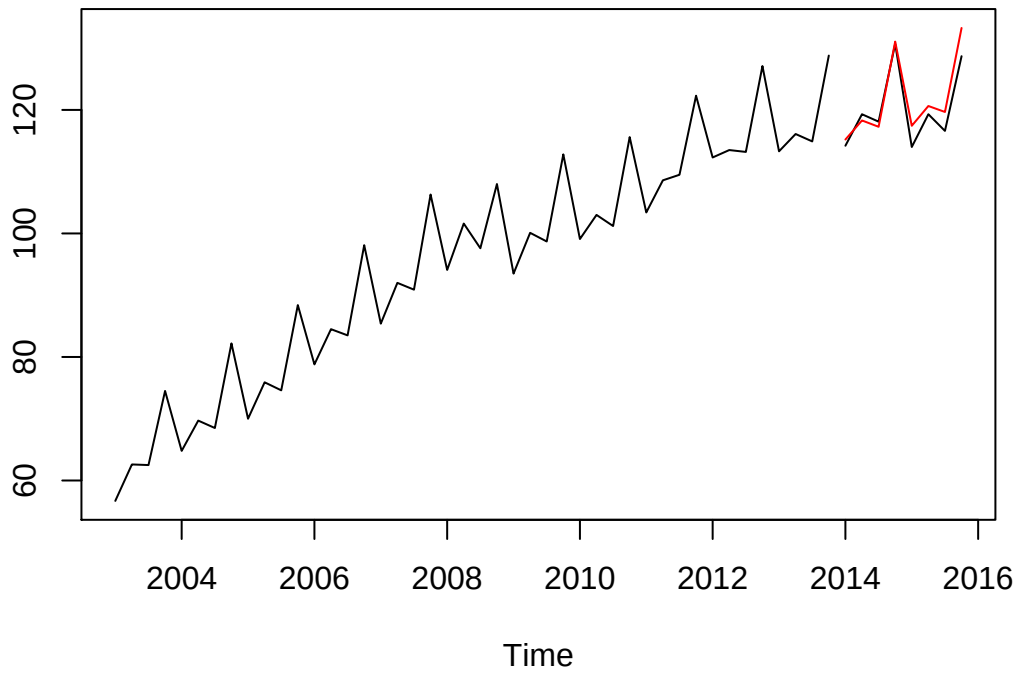
```
# Initialise an array to save the forecasts
frc1 <- array(NA,c(8,1))

for (i in 1:8){
  # For the Xnew we use the last five observations as before
  Xnew <- tail(y.trn,5)
  # Add to that the forecasted values
  Xnew <- c(Xnew,frc1)
  # Take the relevant 5 values. The index i helps us to get the right ones
  Xnew <- Xnew[i:(4+i)]
  # If i = 1 then this becomes Xnew[1:5].
  # If i = 2 then this becomes Xnew[2:6] - just as the example above.
  # Reverse the order
  Xnew <- Xnew[5:1]
  # Make Xnew an array and name the inputs
  Xnew <- array(Xnew, c(1,5)) # c(1,5) are the dimensions of the array
  colnames(Xnew) <- paste0("lag",1:5) # I have already reversed the order
  # Convert to data.frame
  Xnew <- as.data.frame(Xnew)
  # Forecast
  frc1[i] <- predict(fit2,Xnew)
}
frc1
```

```
##          [,1]
## [1,] 115.2038
## [2,] 118.2922
## [3,] 117.2685
```

```
## [4,] 131.0602
## [5,] 117.4294
## [6,] 120.6364
## [7,] 119.6665
## [8,] 133.2663
```

And we plot the result.

```
frc1 <- ts(frc1,frequency=frequency(y.tst),start=start(y.tst))
ts.plot(y.trn,y.tst,frc1,col=c("black","black","red"))
```



3 Lasso regression

To develop lasso models we need to use the package **glmnet**. The following command will check if it is installed, install it if needed, and load it.

```
if (!require("glmnet")){install.packages("glmnet")}; library(glmnet)
```

```
## Loading required package: glmnet
```

```
## Loading required package: Matrix
```

```
## Loaded glmnet 3.0-2
```

The command we are interested in is `cv.glmnet()`. This fits a lasso and cross-validates an appropriate value for lambda. We will repeat the OLS regression model (`fit2`) with lasso.

For lasso the inputs must be as an array without NA values

```
# I remove the first 5 rows by -(1:5) that contain NAs
# For the explanatories I remove the first column
xx <- as.matrix(X[-(1:5),-1])
```

```
# For the target I retain only the first column
yy <- as.matrix(X[-(1:5),1])
```

Estimate the model

```
lasso <- cv.glmnet(x=xx,y=yy)
```

To see the coefficients and the variables that stayed in the model we can use the function `coef()`

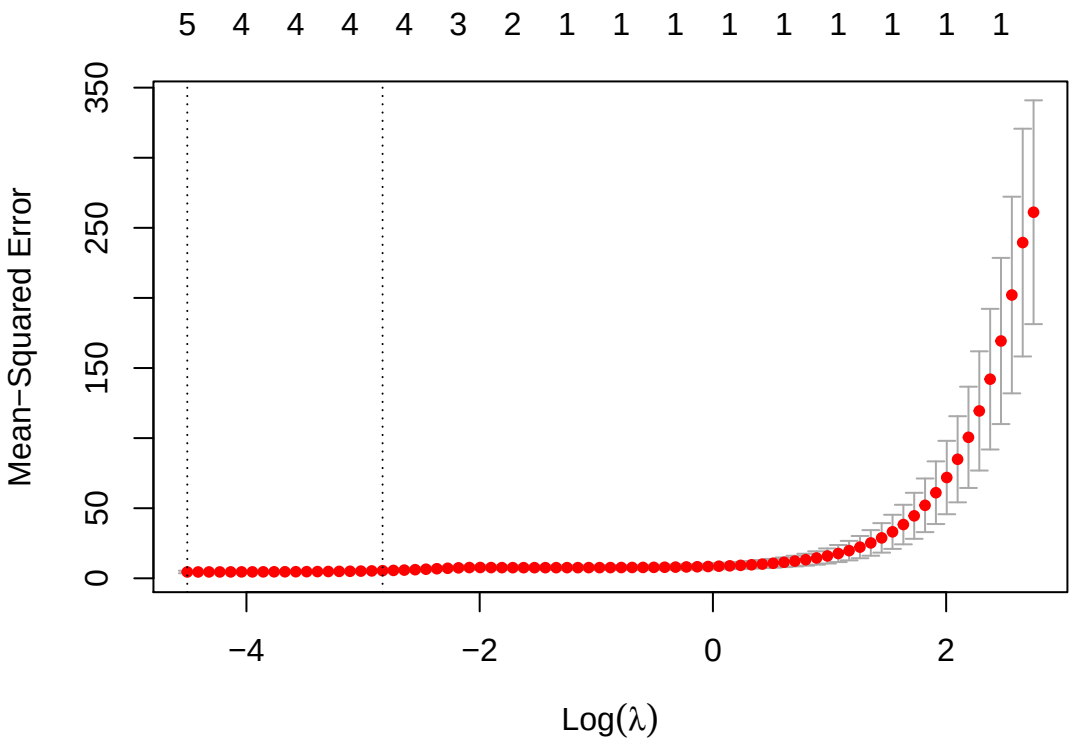
```
coef(lasso)
```

```
## 6 x 1 sparse Matrix of class "dgCMatrix"
##                       1
## (Intercept)  9.28233363
## lag1         0.35160381
## lag2         .
## lag3        -0.01437959
## lag4         0.94730429
## lag5        -0.34699074
```

Lasso suggests that lag2 is excluded, while the others are retained.

We can also look at the cross-validated error, which helps identify the appropriate lambda.

```
plot(lasso)
```



The vertical axis shows the cross-validated error. The horizontal axis has different values of lambda. Note that its scale is in logarithms. The numbers at the top show how many variables are included for any value of lambda. As the cross-validated error is calculate using k-folds (the value of k can be controlled in `cv.glmnet()` using the argument `nfold` that by default is equal to 10), for each value of lambda, we have a distribution of MSE. The red dots in the plot are the means, while the grey whiskers show the spread. In choosing lambda we have two options, either choose the one with the minimum error, or the one that is one-standard deviation of the cross-validated error away from the

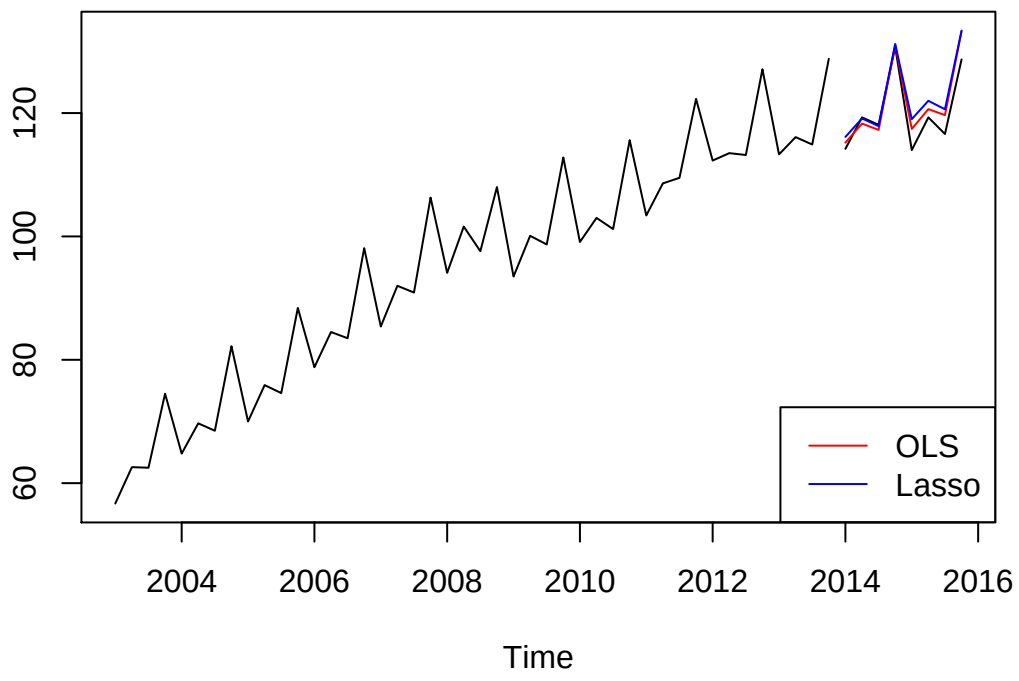
minimum errors. These two are denoted by the vertical dotted lines. Empirical evidence suggests that the second option works best and is what is selected by default.

From this point on, we build forecasts in the same way as with normal regression. We can use `predict()` to get forecasts as usual. The only difference is that we do not need to make the inputs as a data.frame.

```
frc2 <- array(NA,c(8,1))
for (i in 1:8){
  # Create inputs - note for lasso we do not transform these into data.frame
  Xnew <- c(tail(y.trn,5),frc2)
  Xnew <- (Xnew[i:(4+i)])[5:1]
  Xnew <- array(Xnew, c(1,5))
  colnames(Xnew) <- paste0("lag",1:5)
  # Forecast
  frc2[i] <- predict(lasso,Xnew)
}
```

Plot the result

```
# Transform to time series
frc2 <- ts(frc2,frequency=frequency(y.tst),start=start(y.tst))
# Plot together with fit2
ts.plot(y.trn,y.tst,frc1,frc2,col=c("black","black","red","blue"))
legend("bottomright",c("OLS","Lasso"),col=c("red","blue"),lty=1)
```



Finally, we can use the same function to build a *ridge* regression, or an *elastic net*. To do this we need to use the argument `alpha` in the `cv.glmnet()` function. With `alpha=1` we get lasso. With `alpha=0` we get ridge. Any other value for `alpha` between zero and one defines a mixing parameter for the two lambdas of elastic nets.

```
ridge <- cv.glmnet(x=xx,y=yy,alpha=0)
coef(ridge)
```

```
## 6 x 1 sparse Matrix of class "dgCMatrix"
##                       1
## (Intercept) 13.90165913
## lag1         0.08621839
## lag2         0.19391826
## lag3         0.03936103
## lag4         0.55924642
## lag5         0.02074574
```

We can compare these with the result from lasso:

```
cc <- as.matrix(cbind(coef(lasso),coef(ridge)))
colnames(cc) <- c("lasso","ridge")
round(cc,3)
```

```
##              lasso  ridge
## (Intercept)  9.282 13.902
## lag1         0.352  0.086
## lag2         0.000  0.194
## lag3        -0.014  0.039
## lag4         0.947  0.559
## lag5        -0.347  0.021
```

4 Exercises on lasso regression

1. Evaluate the performance of the ols, lasso, and ridge forecasts. Which performs best here? How do they compare with an exponential smoothing benchmark?

Happy forecasting!