**By Gabriel Vasconcelos**

When dealing with forecasting models there is an issue that generates a lot of confusion, which is the difference between direct and recursive forecasts. I believe most people are more used to recursive forecasts because they are the first we learn when studying ARIMA models.

Suppose you want to forecast the variable y, hsteps ahead using only past information of yand consider the two equations below:

\displaystyle y_{t+1} = \alpha_0 + \phi_1 y_{t} + \varepsilon_{t+1}  
\displaystyle y_{t+h} = \alpha_0 + \phi_1 y_{t} + \varepsilon_{t+h}

The first equation is an AR(1) and the second equation is a more general version where we estimate a model directly to the forecasting horizon we want. Now let’s see how the one step ahead forecast would be for each model. In the first equation we would have \hat{y}_{t+1} = \hat{\alpha}_0 + \hat{\phi_1} y_{t}and in the second equation we must make h=1to obtain the same result.

For two steps ahead things start to change. In the first equation we have:

\displaystyle \hat{y}_{t+2}=\hat{\alpha}_0 + \hat{\phi_1} \hat{y}_{t+1} = \hat{\alpha}_0 (1+ \hat{\phi_1}) + \hat{\phi_1}^2 y_{t}

and in the second:

\displaystyle \hat{y}_{t+2}=\hat{\alpha}_{0,2} + \hat{\phi}_{1,2} y_{t}

Note that there is an extra number 2 subscript in the coefficients of the equation above. It indicates that both \alpha_0and \phi_1will depend on the choice of h. Now we can generalize both cases to any hin order to have:

\displaystyle \hat{y}_{t+h}=\hat{\alpha}_0\sum_{i=1}^h \hat{\phi}_1^{i-1}+\hat{\phi}_1^hy_t

\displaystyle \hat{y}_{t+h}=\hat{\alpha}_{0,h} + \hat{\phi}_{1,h} y_{t}

The first case is called recursive forecast and the second case is called direct forecast. In the recursive forecast we only need to estimate one model and use its coefficients to iterate on the forecasting horizon until we have the horizon we want. In the direct forecast we need to estimate one different model for each forecasting horizon but we do not need to iterate the forecast. The first out-of-sample prediction of the direct forecast will be already on the desired horizon.

**Multivariate Problems**

Now suppose we want to forecast yusing past information of yand x. The recursive model would be:

\displaystyle y_{t+1}=\alpha_0 + \phi_1 y_t + \gamma_1 x_t +\varepsilon_{t+1}

The one step ahead forecast would be:

\displaystyle \hat{y}_{t+1}=\hat{\alpha}_0 + \hat{\phi}_1 y_t + \hat{\gamma}_1 x_t

The two step ahead forecast would be:

\displaystyle \hat{y}_{t+2}=\hat{\alpha}_0 + \hat{\phi}_1 \hat{y}_{t+1} + \hat{\gamma}_1 x_{t+1}

Now we have a problem. In the two steps ahead forecast we can just replace \hat{y}_{t+1}by the one step ahead equation just like we did in the univariate case. However, we do not have a way to obtain x_{t+1}. In fact, if we use other variables in recursive forecasts we must also forecast these variables in a Vector Autorregressive (VAR) framework for example. In the direct case nothing changes: we could just estimate an equation for y_{t+h}on y_tand x_t.

**Example**

In this example we are going to forecast the Brazilian inflation using past information of the inflation, the industrial production and the unemployment rate. The recursive model will be a VAR(3) and the direct model will be a simple regression with three lags of each variable.

#library devtools

#install\_github(gabrielrvsc/HDeconometrics)

library(HDeconometrics)

library(reshape2)

library(ggplot2)

# = Load data = #

data("BRinf")

data = BRinf[ , c(1, 12, 14)]

colnames(data) = c("INF", "IP", "U")

**Recursive Forecast**

First we are going to estimate the recursive forecasts for 1 to 24 steps (months) ahead. The VAR will forecast all variables but we are only interested in the inflation. The plot below shows that the forecast converges very fast to the yellow line, which is the unconditional mean of the inflation in the training set. Recursive forecasts using AR or VAR on stationary and stable data will always converge to the unconditional mean unless we include more features in the model such as exogenous variables.

# = 24 out-of-sample (test) observations = #

train = data[1:132, ]

test = data[-c(1:132), ]

# = Estimate model and compute forecasts = #

VAR = HDvar(train, p = 3)

recursive = predict(VAR, 24)

df = data.frame(date = as.Date(rownames(data)),

INF = data[,"INF"],

fitted=c(rep(NA, 3), fitted(VAR)[ ,1], rep(NA, 24)),

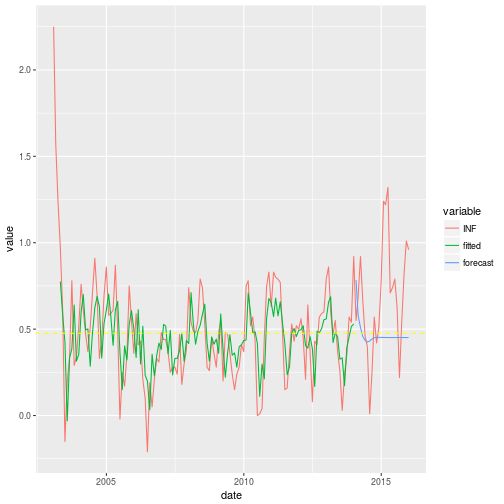
forecast = c(rep(NA,132), recursive[, 1]))

# = Plot = #

dfm = melt(df,id.vars = "date")

ggplot(data = dfm) + geom\_line(aes(x = date, y = value, color = variable))+

geom\_hline(yintercept = mean(train[ ,1]), linetype = 2,color = "yellow")



**Direct Forecast**

In direct forecasts we will need to estimate 24 models for the inflation to obtain the 24 forecasts. We must arrange the data in the right way for the model to estimate the regression on the right lags. This is where most people get confused. Let’s look at an univariate example using the function embed to arrange the data. I created a variable ythe is just the sequence from 1 to 10. The embed function was used to generate a matrix with y_tand its first three lags (we lost three observations because of the lags). The model for h=1is a regression of the column y_ton the remaining columns.

Formal Ways to Compare Forecast

When working with time-series forecasting we often have to choose between a few potential models and the best way is to test each model in pseudo-out-of-sample estimations. In other words, we simulate a forecasting situation where we drop some data from the estimation sample to see how each model perform.

Naturally, if you do only one (or just a few) forecasting test you results will have no robustness and in the next forecast the results may change drastically. Another possibility is to estimate the model in, let’s say, half of the sample, and use the estimated model to forecast the other half. This is better than a single forecast but it does not account for possible changes in the structure of the data over the time because you have only one estimation of the model. The most accurate way to compare models is using rolling windows. Suppose you have, for example, 200 observations of a time-series. First you estimate the model with the first 100 observations to forecast the observation 101. Then you include the observation 101 in the estimation sample and estimate the model again to forecast the observation 102. The process is repeated until you have a forecast for all 100 out-of-sample observations. This procedure is also called expanding window. If you drop the first observation in each iteration to keep the window size always the same then you have a fixed rolling window estimation. In the end you will have 100 forecasts for each model and you can calculate RMSE, MAE and formal tests such as Diebold & Mariano.

In general, the fixed rolling window is better than the expanding window because of the following example. Suppose we have two models:

\displaystyle y_t = b_1x_{1,t-1}+\varepsilon_t  
y_t = b_1x_{1,t-1}+b_2x_{2,t-1}+\varepsilon_t

Let’s assume that the true value of b_2 is zero. If we use expanding windows the asymptotic theory tells us that \hat{b}_2 will go to zero and both models will be the same. If that is the case, we may be unable to distinguish which model is more accurate to forecast y_t. However, the first model is better than the second model in small samples and it is just as good in large samples. We should be able to identify this feature. Fixed rolling windows keep the sample size fixed and they are free from this problem conditional on the sample size. In this case, the Diebold & Mariano test becomes the Giacomini & White test.

Application

In this example we are going to use some inflation data from the AER package. First let’s have a look at the function embed. This function is very useful in this rolling window framework because we often include lags of variables in the models and the function embed creates all lags for us in a single line of code. Here is an example:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | library(AER)  library(xts)  library(foreach)  library(reshape2)  library(ggplot2)  library(forecast)    ## = embed = ##  x1 = c(1, 2, 3, 4, 5)  x2 = c(11, 12, 13, 14, 15)  x = cbind(x1, x2)  (x\_embed = embed(x, 2)) |
| 1  2  3  4  5 | ##      [,1] [,2] [,3] [,4]  ## [1,]    2   12    1   11  ## [2,]    3   13    2   12  ## [3,]    4   14    3   13  ## [4,]    5   15    4   14 |

As you can see. The first two columns show the variables x1 and x2 at lag 0 and the second column shows the same variables with one lag. We lost one observation because of the lag operation.

To the real example!!! We are going to estimate a model to forecast the US inflation using four autorregressive variables (four lags of the inflation), four lags of the industrial production and dummy variables for months. The second model will be a simple random walk. I took the first log-difference on both variables (CPI and industrial production index). The code below loads and prepare the data with the embed function.

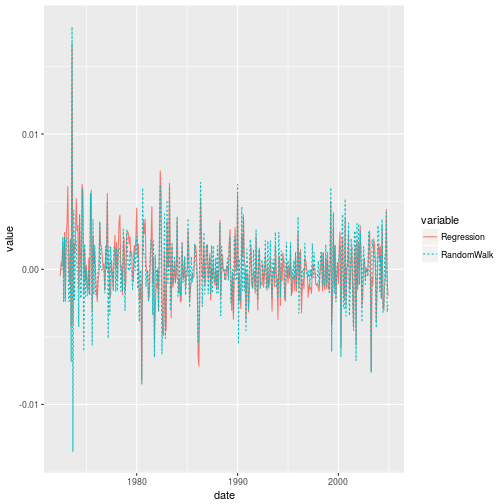
|  |  |  |
| --- | --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | ## = Load Data = ##  data("USMacroSWM")  data = as.xts(USMacroSWM)[ , c("cpi", "production"), ]  data = cbind(diff(log(data[ ,"cpi"])), diff(log(data[ ,"production"])))[-1, ]    ## = Prep data with embed = ##  lag = 4  X = embed(data, lag + 1)  X = as.data.frame(X)  colnames(X) = paste(rep(c("inf", "prod"), lag + 1),                      sort(rep(paste("l", 0:lag, sep = ""),2)), sep = "" )  X$month = months(tail(index(data), nrow(X)))  head(X) | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | ##         infl0       prodl0        infl1       prodl1        infl2  ## 1 0.005905082  0.000000000 -0.002275314  0.004085211  0.000000000  ## 2 0.006770507 -0.005841138  0.005905082  0.000000000 -0.002275314  ## 3 0.007618231  0.005841138  0.006770507 -0.005841138  0.005905082  ## 4 0.019452426  0.007542827  0.007618231  0.005841138  0.006770507  ## 5 0.003060112  0.009778623  0.019452426  0.007542827  0.007618231  ## 6 0.006526018  0.013644328  0.003060112  0.009778623  0.019452426  ##         prodl2        infl3       prodl3        infl4       prodl4  ## 1 -0.008153802  0.017423641  0.005817352  0.006496543  0.005851392  ## 2  0.004085211  0.000000000 -0.008153802  0.017423641  0.005817352  ## 3  0.000000000 -0.002275314  0.004085211  0.000000000 -0.008153802  ## 4 -0.005841138  0.005905082  0.000000000 -0.002275314  0.004085211  ## 5  0.005841138  0.006770507 -0.005841138  0.005905082  0.000000000  ## 6  0.007542827  0.007618231  0.005841138  0.006770507 -0.005841138  ##       month  ## 1      June  ## 2      July  ## 3    August  ## 4 September  ## 5   October  ## 6  November |

The following code estimates 391 fixed rolling windows with a sample size of 300 in each window:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | # = Number of windows and window size  w\_size = 300  n\_windows = nrow(X) - 300    # = Rolling Window Loop = #  forecasts = foreach(i=1:n\_windows, .combine = rbind) %do%{      # = Select data for the window (in and out-of-sample) = #    X\_in = X[i:(w\_size + i - 1), ] # = change to X[1:(w\_size + i - 1), ] for expanding window    X\_out = X[w\_size + i, ]      # = Regression Model = #    m1 = lm(infl0 ~ . - prodl0, data = X\_in)    f1 = predict(m1, X\_out)      # = Random Walk = #    f2 = tail(X\_in$infl0, 1)      return(c(f1, f2))  } |

Finally, the remaining code calculates the forecasting errors, forecasting RMSE across the rolling windows and the Giacomini & White test. As you can see, the test rejected the null hypothesis of both models being equally accurate and the RMSE was smaller for the model with the lags, production and dummies.

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | # = Calculate and plot errors = #  e1 = tail(X[ ,"infl0"], nrow(forecasts)) - forecasts[ ,1]  e2 = tail(X[ ,"infl0"], nrow(forecasts)) - forecasts[ ,2]  df = data.frame("date"=tail(as.Date(index(data)), n\_windows), "Regression" = e1, "RandomWalk" = e2)  mdf = melt(df,id.vars = "date")  ggplot(data = mdf) + geom\_line(aes(x = date, y = value, linetype = variable, color = variable)) |



|  |  |
| --- | --- |
| 1  2 | # = RMSE = #  (rmse1 = 1000 \* sqrt(mean(e1 ^ 2))) |
| 1 | ## [1] 2.400037 |

|  |  |
| --- | --- |
| 1 | (rmse2 = 1000 \* sqrt(mean(e2 ^ 2))) |
| 1 | ## [1] 2.62445 |

|  |  |
| --- | --- |
| 1  2 | # = DM test = #  (dm = dm.test(e1, e2)) |
| 1  2  3  4  5  6  7 | ##  ##  Diebold-Mariano Test  ##  ## data:  e1e2  ## DM = -1.977, Forecast horizon = 1, Loss function power = 2,  ## p-value = 0.04874  ## alternative hypothesis: two.sided |

y=1:10

lags=embed(y,4)

colnames(lags)=c("yt","yt-1","yt-2","yt-3")

lags

## yt yt-[1 yt-2 yt](https://www.google.com/maps/search/1+yt-2+yt?entry=gmail&source=g)-3

## [1,] 4 3 2 1

## [2,] 5 4 3 2

## [3,] 6 5 4 3

## [4,] 7 6 5 4

## [5,] 8 7 6 5

## [6,] 9 8 7 6

## [7,] 10 9 8 7

If we want to run a model for two steps ahead we must remove the fist observation in the y_tcolumn and the last observation in the lagged columns:

lags = cbind(lags[-1, 1], lags[-nrow(lags), -1])

colnames(lags) = c("yt", "yt-2", "yt-3", "yt-4")

lags

## yt yt-2 yt-3 yt-4

## [1,] 5 3 2 1

## [2,] 6 4 3 2

## [3,] 7 5 4 3

## [4,] 8 6 5 4

## [5,] 9 7 6 5

## [6,] 10 8 7 6

Now we just run a regression of the y_tcolumns on the other columns. For h=3we must do the same procedure again and for a general hwe must remove the first h-1observations of the first column and the last h-1observations of the remaining columns.

The embed function also works for matrices with multiple columns. I will use it on the data to make it ready for the model. The code below runs the direct forecast for the forecasting horizons 1 to 24. The plot does not have a fitted line because there is one fitted model for each horizon. You can see that the direct forecasting is considerably different from the recursive case. It does not converge to the unconditional mean.

# = Create matrix with lags = #

X = embed(data, 4)

train = X[1:129, ]

test = X[-c(1:129), ]

ytrain = train[ ,1]

# = The Xtest is the same for all horizons = #

# = The last three observations (lags) of the train for each variable = #

Xtest = train[nrow(train), 1:9]

# = Remove the first three columns of the train = #

# = They are the three variables in t = #

Xtrain = train[ ,-c(1:3)]

ytest = test[, 1]

# = Run 24 models and forecasts = #

direct = c()

for(i in 1:24){

model = lm(ytrain ~ Xtrain) # = Run regression = #

direct[i] = c(1, Xtest) %\*% coef(model) # = Calculate Forecast = #

ytrain = ytrain[-1] # = Remove first observation of yt = #

Xtrain = Xtrain[-nrow(Xtrain), ] # = Remove last observation of the other variables = #

}

# = plot = #

df = data.frame(data=as.Date(rownames(data)),

INF = data[ ,"INF"],

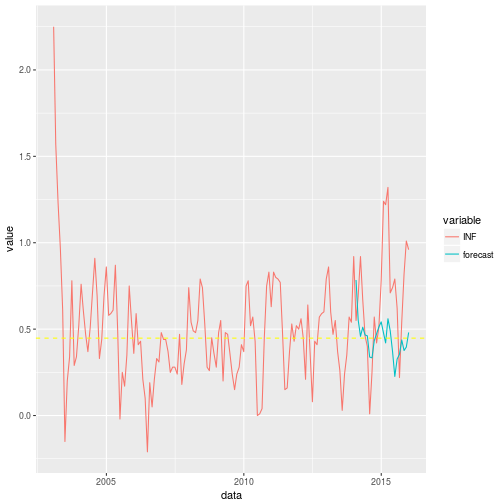
forecast = c(rep(NA, 132), direct))

dfm = melt(df,id.vars = "data")

ggplot(data = dfm) +

geom\_line(aes(x = data, y = value, color = variable)) +

geom\_hline(yintercept = mean(train[ ,1]), linetype=2, color="yellow")



As mentioned before, the one step ahead forecast is the same in both cases:

print(direct[1])

## [1] 0.7843985

print(recursive[1])

## [1] 0.7843985

**Conclusion**

* Recursive forecast:
  + Single model for all horizons,
  + must iterate the forecast using the coefficients for horizons different from one,
  + Forecasts converge to the unconditional mean for long horizons.
* Direct forecast:
  + One model for each horizon,
  + No need for iteration,
  + Forecasts does not converge to the unconditional mean,
  + Must be careful to arrange the data.