2.3 Rolling Validation for Multiple Horizons

Due to the dependencies of instances near each other in time, k-fold cross-validation will not work for time-series data. A simple form of rolling validation divides a dataset into an initial training set and test set. For example, we can use the first 60% of the samples as the training set and rest as the test set. For horizon $h = 1$ forecasting, the first value in the test set is forecasted based on the model produced by training on the training set. The error is the difference between the actual value in the test set and the forecasted value which is used to calculate the symmetric mean absolute percentage error (sMAPE). sMAPE is used as the primary performance metric, which is one of the standard performance metrics in time-series forecasting. Although, other performance metrics such as MAPE can also be used.

$$\text{sMAPE} = 100 * mean(2|y_t - f_t|/|y_t| + |f_t|)$$

where y_t is the true valued time-series vector and f_t is the forecasted time-series vector.

For simple, efficient models, the process of rolling forward to forecast the next value in the test set often involves retraining the model by including the first value from the test set into the training set (at the end) and discarding the first value from the training set. To maintain the same size for the training set, we remove the first value from the training set. We adjust the frequency of retraining for the statistical models such that we forecast kt samples ahead in the test set before including them in the training set and retraining our model.

2.3.1 Rolling Validation in R

The rolling validation function in R is used to generate multi-horizon out-of-sample forecasts for a given time-series. The *forecast* package developed by Dr. Rob J. Hyndman is used to get the ARIMA model in R. The entire code is given at the end and we begin with the stepwise explanation for the code below:

We begin by first installing and using the *forecast* package in our program. Package Installation is required only for first time setup and can be commented out for later runs.

```r
install.packages("forecast")
library(forecast)
```

Listing 2.11: Step 1

We then begin with our initial variable setup for our program. This includes training/test size, horizons, all the hyperparameters of our ARIMA/SARIMA model and the rolling validation function. The variable `zero_reference` is used as first day starts on 1st Jan 1970 in R language and until 25th Feb 2020, which is the start of our model data, the number of days is equal to 18317.

```r
zero_reference = 18317 #Difference between 1970-01-01 to 2020-02-25

ts_size = 376  # Number of Sample/Timesteps in entire time series
tr_size = 222  # Number of Samples/Timesteps in Training set
```

```r
te_size = 154   # Number of Samples/Timesteps in Training set
horizons = 14   # Number of days to forecast ahead in the future
kt = 1          # Retraining Frequency (No. of Samples)
rt = 0          # Total Number of Retrainings Occured
p = 1           # Non-seasonal Auto-Regressive Order
d = 0           # Non-seasonal Differencing
q = 0           # Non-seasonal Moving Average Order
PP = 3          # Seasonal Auto-Regressive Order
DD = 1          # Seasonal Differencing
QQ = 1          # Seasonal Moving Average Order
s = 7           # Seasonal Period
```

Listing 2.12: Step 2

Once we have our initial setup ready, we then pull our dataset from remote URL, in this case GitHub repo, and then get the response variable, in this case deathIncrease, and finally convert it into a zoo object. A zoo object is used to store and easily manipulate regular and irregular time-series vectors.

```r
# Pull in the dataset from given URL
covid_data <- read.csv(file = "https://raw.githubusercontent.com/scalation
    /data/master/COVID/CLEANED_35_Updated.csv")
#Convert reponse column into "zoo" object
response_variable = covid_data$deathIncrease
#@see: stackoverflow.com/questions/33714660/what-is-the-difference-the-zoo
    -object-and-ts-object-in-r
covid_ts <- zoo(response_variable, seq(as.Date("2020-01-13"), as.Date("
    2021-03-07"), by = "days"), frequency = 1)
```

Listing 2.13: Step 3

Once we have our dataset ready, we then build our forecast matrix for out-of-sample forecast. Denoting forecast matrix as f_m, we have $f_m \in R^{h \times te\_size}$, where h is the number of horizons we are forecasting for and te_size is the size of the test set.

```r
#Build forecasting matrix to hold all rolling forecasts. Matrix Dimension:
     horizons by size of test set
forecast_matrix <- matrix(data = c(-1.2), nrow = horizons, ncol = te_size)
```

Listing 2.14: Step 4

We are now ready to begin with the rolling validation, which is done by looping over the test set as given below:

```r
# Start rolling validation: run through entire test set 1 sample at a time
for (i in 1:te_size) {

    # Get training data: start=1+18317 from 1970-01-01 which is 2020-02-26
    & end=221+18317+1 which is 2020-10-04(inclusive)
    # This also shifts the training window ahead by 1 sample at a time
   into the test set on every every iteration
```

```
    # For eg. For the next iteration: start=2+18317 from 1970-01-01 which is 2020-02-27 & end=221+18317+2 which is 2020-10-05(inclusive)
    # So we dropped the sample for 2020-02-26 from the training in 2nd iteration and included 1 from the test set. That's how we roll forward.
    model_data <- window(covid_ts, start = as.Date(i + zero_reference), end = as.Date(221 + zero_reference + i), frequency = 1)

    # Retraining after kt samples
    if (i == 1) {
        arima_model <- Arima(y = model_data, order = c(p, d, q), seasonal = list(order = c(PP, DD, QQ), period = s), method = "ML")
    } else {
        if ((i %% kt) == 0) { #if iteration mod retrain frequency is 0, then it is time to retrain the model with new training data
            rt = rt + 1
            arima_model <- Arima(y = model_data, order = c(p, d, q), seasonal = list(order = c(PP, DD, QQ), period = s), method = "ML")
        } else {
            # if not, then
            # 1) use the new shifted data as the model data and
            # 2) do not restimate the model parameters again: done by specifying the old model reference variable in "model" attribute of the function call
            arima_model <- Arima(y = model_data, model = arima_model)
        }
    }
    # Once we have the model set: either with new data or with new data and retrained get next "horizon" number of forecasts.
    future <- forecast(arima_model, h = horizons, level = 0)
    # Get the single point forecast vector as forecast() function also returns high/max and low/min values
    forecasts <- future$mean
    # Start filling the forecast matrix 1 by 1 for "horizon" number of forecasts
    for (k in 1:horizons) {
        if (k + i - 1 <= te_size) { # Condition to prevent overflow at the end of the test set;
            forecast_matrix[k, k+i-1] = forecasts[k]
        }
    }
}
```

Listing 2.15: Step 5

After the rolling validation is complete, we have our out-of-sample forecasts collected in the forecasting matrix but the initial h number of days for the corresponding h^{th} horizon does not have any forecast because we cannot generate $h = 2$ forecast for day 1 or $h = 3$ forecast for day 2 and so on. In order to tackle this issue, we use the previous horizon forecast by copying the latest generated value into all the remaining rows for that single day. Once that is done we store the transpose of the matrix where we get each horizon forecast as column vectors.

```r
# For first 14 days copy the values of the previous day as we cannot have
   h=2 forecast for first day of test set, h=3 forecast for the 2nd day of
    test set and so on;
for(i in 2:horizons){
  for(j in 1:i){
    forecast_matrix[i,j] = forecast_matrix[i-1,j]
  }
}
# Transpose the forecast matrix to be of dimension: te_size by horizons
forecast_matrix <- t(forecast_matrix)
```

Listing 2.16: Step 6

We now get our true values for the test set

```r
# Get true values of the test data
test_data <- window(covid_ts, start = as.Date((tr_size+1) + zero_reference
   ), end = as.Date(ts_size + zero_reference), frequency = 1)
```

Listing 2.17: Step 7

Next, we build the matrix to store SMAPE scores for all horizons.

```r
# Generate matrix to store smape scores for each horizon
smape_scores = matrix(data = NA, nrow = horizons, ncol = 1)
```

Listing 2.18: Step 8

Finally, we loop through each column in our forecasting matrix and compare it with true values of the test set to calculate smape score and store it in our smape score matrix.

```r
# Calculate the smape scores and store in the smape matrix, horizon by
   horizon.
for (i in 1:horizons) {
  smape_scores[i, 1] <- mean(abs((test_data - forecast_matrix[, i]) / (
   test_data + forecast_matrix[, i]))) * 200 #smape formula
  print(paste0("h=",i,",smape=",smape_scores[i,1]))
}
```

Listing 2.19: Step 9

Below is the the full code for all above steps combined. The code is also available on github at: https://github.com/scalation/data/blob/master/COVID/covid_forecasting.r

```r
#install.packages("forecast")  #Unccomment and run this for the first time
    R is setup on your machine
library(forecast)

zero_reference = 18317 #Difference between 1970-01-01 to 2020-02-25

```

```
ts_size = 376  # Number of Sample/Timesteps in entire time series
tr_size = 222  # Number of Samples/Timesteps in Training set
te_size = 154  # Number of Samples/Timesteps in Training set
horizons = 14  # Number of days to forecast ahead in the future
kt = 1         # Retraining Frequency (No. of Samples)
rt = 0         # Total Number of Retrainings Occured
p = 1          # Non-seasonal Auto-Regressive Order
d = 0          # Non-seasonal Differencing
q = 0          # Non-seasonal Moving Average Order
PP = 3         # Seasonal Auto-Regressive Order
DD = 1         # Seasonal Differencing
QQ = 1         # Seasonal Moving Average Order
s = 7          # Seasonal Period

# Pull in the dataset from given URL
covid_data <- read.csv(file = "https://raw.githubusercontent.com/scalation
   /data/master/COVID/CLEANED_35_Updated.csv")

#Convert reponse column into "zoo" object
response_variable = covid_data$deathIncrease
#@see: stackoverflow.com/questions/33714660/what-is-the-difference-the-zoo
   -object-and-ts-object-in-r
covid_ts <- zoo(response_variable, seq(as.Date("2020-01-13"), as.Date("
   2021-03-07"), by = "days"), frequency = 1)

#Build forecasting matrix to hold all rolling forecasts. Matrix Dimension:
    horizons by size of test set
forecast_matrix <- matrix(data = c(-1.2), nrow = horizons, ncol = te_size)

# Start rolling validation: run through entire test set 1 sample at a time
for (i in 1:te_size) {

    # Get training data: start=1+18317 from 1970-01-01 which is 2020-02-26
    & end=221+18317+1 which is 2020-10-04(inclusive)
    # This also shifts the training window ahead by 1 sample at a time
   into the test set on every every iteration
    # For eg. For the next iteration: start=2+18317 from 1970-01-01 which
   is 2020-02-27 & end=221+18317+2 which is 2020-10-05(inclusive)
    # So we dropped the sample for 2020-02-26 from the training in 2nd
   iteration and included 1 from the test set. That's how we roll forward.
    model_data <- window(covid_ts, start = as.Date(i + zero_reference),
   end = as.Date(221 + zero_reference + i), frequency = 1)

    # Retraining after kt samples
    if (i == 1) {
        arima_model <- Arima(y = model_data, order = c(p, d, q), seasonal
   = list(order = c(PP, DD, QQ), period = s), method = "ML")
```

```r
  } else {
    if ((i %% kt) == 0) { #if iteration mod retrain frequency is 0, then it is time to retrain the model with new training data
      rt = rt + 1
      arima_model <- Arima(y = model_data, order = c(p, d, q), seasonal = list(order = c(PP, DD, QQ), period = s), method = "ML")
    } else {
      # if not, then
      # 1) use the new shifted data as the model data and
      # 2) do not restimate the model parameters again: done by specifying the old model reference variable in "model" attribute of the function call
      arima_model <- Arima(y = model_data, model = arima_model)
    }
  }
  # Once we have the model set: either with new data or with new data and retrained get next "horizon" number of forecasts.
  future <- forecast(arima_model, h = horizons, level = 0)
  # Get the single point forecast vector as forecast() function also returns high/max and low/min values
  forecasts <- future$mean
  # Start filling the forecast matrix 1 by 1 for "horizon" number of forecasts
  for (k in 1:horizons) {
    if (k + i - 1 <= te_size) { # Condition to prevent overflow at the end of the test set;
      forecast_matrix[k, k+i-1] = forecasts[k]
    }
  }
}
# For first 14 days copy the values of the previous day as we cannot have h=2 forecast for first day of test set, h=3 forecast for the 2nd day of test set and so on;
for(i in 2:horizons){
 for(j in 1:i){
  forecast_matrix[i,j] = forecast_matrix[i-1,j]
 }
}
# Transpose the forecast matrix to be of dimension: te_size by horizons
forecast_matrix <- t(forecast_matrix)

# Get true values of the test data
test_data <- window(covid_ts, start = as.Date((tr_size+1) + zero_reference), end = as.Date(ts_size + zero_reference), frequency = 1)

# Generate matrix to store smape scores for each horizon
smape_scores = matrix(data = NA, nrow = horizons, ncol = 1)
```

```

# Calculate the smape scores and store in the smape matrix, horizon by
  horizon.
for (i in 1:horizons) {
  smape_scores[i, 1] <- mean(abs((test_data - forecast_matrix[, i]) / (
    test_data + forecast_matrix[, i]))) * 200 #smape formula
  print(paste0("h=",i,",smape=",smape_scores[i,1]))
}
```

Listing 2.20: Full Code