Forecast Evaluation Example

Juniper L. Simonis, Ethan P. White, S. K. Morgan Ernest

April 02, 2021

This supplement walks through how to evaluate a probabilistic forecast using scoring rules as detailed in the main text.

Load Dependencies

This work is done in R, version 4.0.3 (R Core Team 2020)

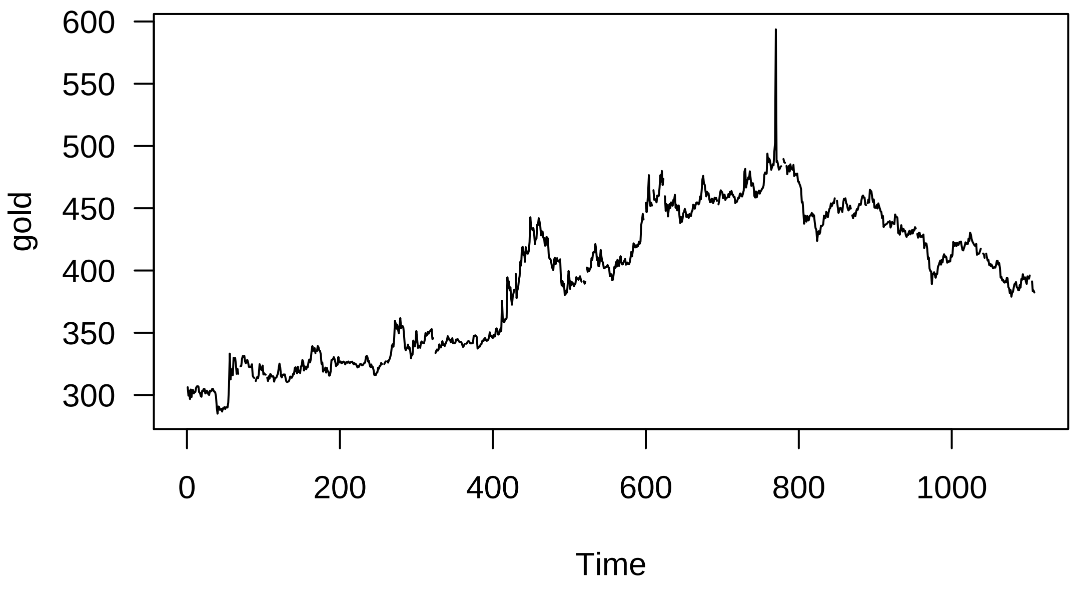
For the present work, we'll be using the **`scoringRules`** (v1.0.1) (Jordan, Krüger, and Lerch 2019) and **`forecast`** (v8.14) (Hyndman et al. 2021) packages.

```r
install.packages(c("scoringRules", "forecast"))
library(forecast)
library(scoringRules)
```

Prep and Define Data Set

We'll use the gold prices data from `forecast`:

```r
data(gold)
plot(gold, las = 1)
```



We'll use the last 11 data points (of 1108; 1%) for testing.

```
gold_train <- gold[1:1097]
gold_test <- gold[1098:1108]
```

Fit the Models

We'll fit the data with a flexible Auto-ARIMA (AA) model using **forecast**'s `auto.arima` function:

```
train_aa_mod <- auto.arima(gold_train)
train_aa_mod
```

```
## Series: gold_train
## ARIMA(2,1,1) with drift
##
## Coefficients:
##          ar1     ar2      ma1   drift
##       0.3367  0.1195  -0.6748  0.0804
## s.e.  0.1184  0.0547   0.1124  0.1045
##
## sigma^2 estimated as 33.54:  log likelihood=-3382.59
## AIC=6775.17   AICc=6775.23   BIC=6800.17
```

and a fixed first-order autoregressive (AR 1) model using **stats**'s `arima` function:

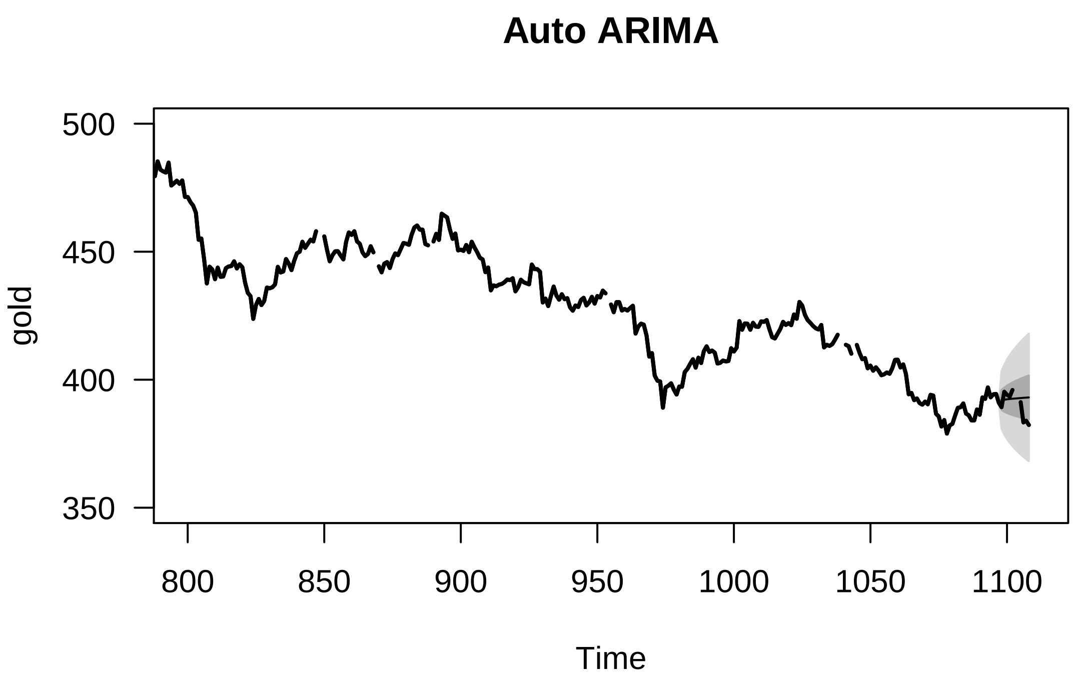
```
train_ar1_mod <- arima(gold_train, order = c(1, 0, 0))
train_ar1_mod
```

```
##
## Call:
## arima(x = gold_train, order = c(1, 0, 0))
##
## Coefficients:
##          ar1  intercept
##       0.9943   387.2623
## s.e.  0.0030    28.2033
##
## sigma^2 estimated as 36.95:  log likelihood = -3445.42,  aic = 6896.84
```

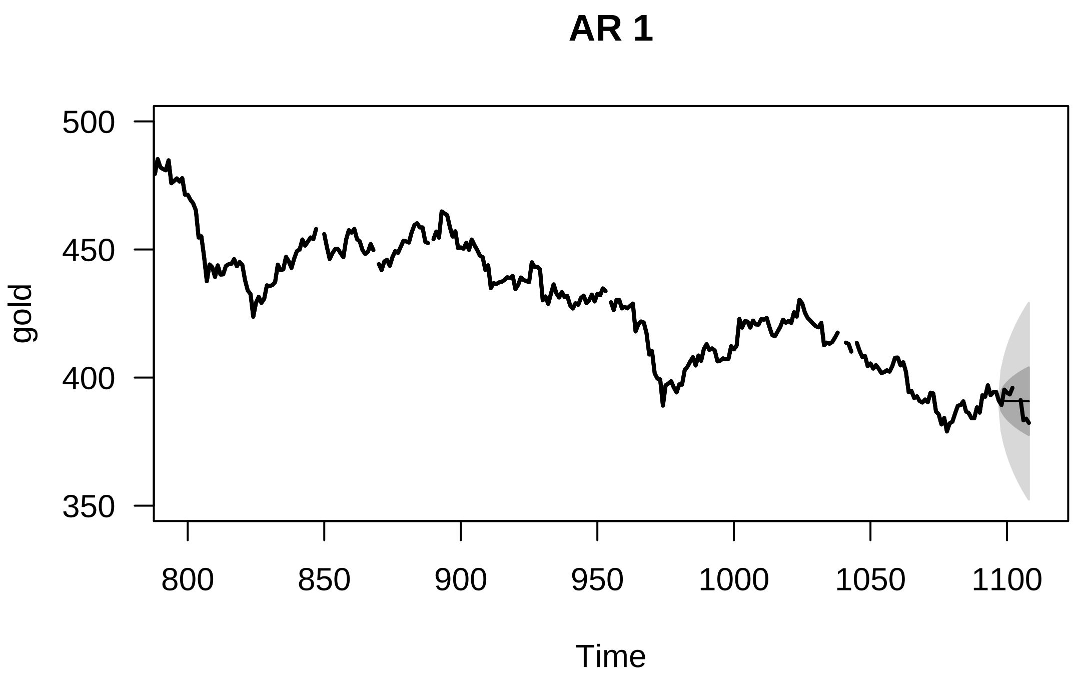
Forecast

We'll use the `forecast` function to provide forecasted values with confidence intervals for plotting:

```
plot_aa_forecast <- data.frame(forecast(train_aa_mod, 11, level = c(50, 95)))
plot(gold, xlim = c(800, 1110), ylim = c(350, 500), las = 1, lwd = 2,
     main = "Auto ARIMA")
polygon(c(1097:1108, 1108:1097),
        c(gold_train[1097], plot_aa_forecast[,4], plot_aa_forecast[11:1,5],
          gold_train[1097]),
        lwd = 1, col = grey(0.7, 0.5), border = grey(0.7, 0.5))
polygon(c(1097:1108, 1108:1097),
        c(gold_train[1097], plot_aa_forecast[,2], plot_aa_forecast[11:1,3],
          gold_train[1097]),
        lwd = 1, col = grey(0.5, 0.5), border = grey(0.5, 0.5))
points(1098:1108, plot_aa_forecast[,1], type = "l", lwd = 1)
points(gold, lwd = 2, type = "l")
```



```
plot_ar1_forecast <- data.frame(forecast(train_ar1_mod, 11,
                                         level = c(50, 95)))
plot(gold, xlim = c(800, 1110), ylim = c(350, 500), las = 1, lwd = 2,
     main = "AR 1")
polygon(c(1097:1108, 1108:1097),
        c(gold_train[1097], plot_ar1_forecast[,4], plot_ar1_forecast[11:1,5],
        gold_train[1097]),
        lwd = 1, col = grey(0.7, 0.5), border = grey(0.7, 0.5))
polygon(c(1097:1108, 1108:1097),
        c(gold_train[1097], plot_ar1_forecast[,2], plot_ar1_forecast[11:1,3],
        gold_train[1097]),
        lwd = 1, col = grey(0.5, 0.5), border = grey(0.5, 0.5))
points(1098:1108, plot_ar1_forecast[,1], type = "l", lwd = 1)
points(gold, lwd = 2, type = "l")
```



For evaluation, it is more helpful to have the parameters of the predicted distributions (mean and standard error) or to use them to draw representative samples. We can get the parameters from the `predict` function but first need to put the time in the model fit objects as part of the calls, otherwise it is empty:

```
train_aa_mod$call$xreg <- forecast:::getxreg(train_aa_mod)
train_ar1_mod$call$xreg <- forecast:::getxreg(train_ar1_mod)
```

Then we create our predicted time as a similar object:

```
test_xreg <- `colnames<-`(as.matrix(1098:1108), "drift")
```

And then we can forecast objects including both the point and error terms for each time point and generate a set of 1000 sample paths for each

```
test_aa_forecast <- predict(object = train_aa_mod, n.ahead = 11,
                            newxreg = test_xreg)
test_aa_forecast <- data.frame(time = 1098:1108,
                               pred = test_aa_forecast$pred,
                               se = test_aa_forecast$se)

aa_forecast_paths <- matrix(NA, nrow = 11, ncol = 1000)
for(i in 1:1000){
  aa_forecast_paths[,i] <- simulate(train_aa_mod, xreg = test_xreg,
                                    bootstrap = TRUE, future = TRUE)
}
```

```
test_ar1_forecast <- predict(object = train_ar1_mod, n.ahead = 11)
test_ar1_forecast <- data.frame(time = 1098:1108,
                                pred = test_ar1_forecast$pred,
                                se = test_ar1_forecast$se)
```

```r
ar1_forecast_paths <- matrix(NA, nrow = 11, ncol = 1000)
for(i in 1:1000){
  ar1_forecast_paths[,i] <- simulate(train_ar1_mod, nsim = 11,
                                     bootstrap = TRUE, future = TRUE)
}
```

Evaluate the Forecasts

For evaluations, we'll first need to remove the `NA` values in the data stream (missing observations), as the functions we'll use don't allow for them:

```r
gold_test_NA <- is.na(gold_test)
gold_test_no_NAs <- gold_test[!gold_test_NA]
aa_forecast_paths_no_NAs <- aa_forecast_paths[!gold_test_NA, ]
ar1_forecast_paths_no_NAs <- ar1_forecast_paths[!gold_test_NA, ]
```

We'll score the forecasts use the Continuous Ranked Probability Score (CRPS) and log score (Matheson and Winkler 1976), both of which **`ScoringRules`** has functions to calculate. However, their functions produce scores where lower is better, so we simply multiple them by `-1`:

```r
aa_crps <- -crps_sample(gold_test_no_NAs, aa_forecast_paths_no_NAs)
aa_logs <- -logs_sample(gold_test_no_NAs, aa_forecast_paths_no_NAs)

ar1_crps <- -crps_sample(gold_test_no_NAs, ar1_forecast_paths_no_NAs)
ar1_logs <- -logs_sample(gold_test_no_NAs, ar1_forecast_paths_no_NAs)
```

Taking the means for the scores across the points, we see that the Auto Arima performs better (higher score; less negative) by both rules:

```r
mean(aa_crps)
```

```
## [1] -3.181804
```

```r
mean(aa_logs)
```

```
## [1] -3.157294
```

```r
mean(ar1_crps)
```

```
## [1] -3.134613
```

```r
mean(ar1_logs)
```

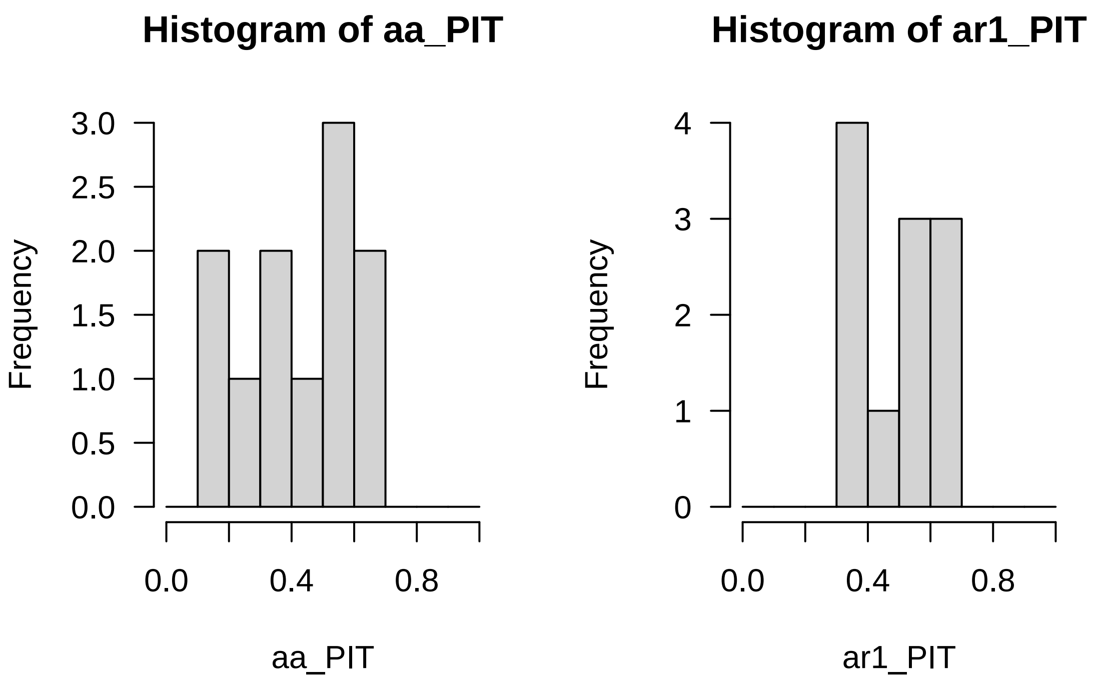
```
## [1] -3.242627
```

We can also use the standard `pnorm` function to calculate the Probability Integral Transform (PIT) (Dawid 1984) for each point for the two models:

```r
aa_PIT <- pnorm(gold_test_no_NAs, mean = test_aa_forecast$pred,
                sd = test_aa_forecast$se)
ar1_PIT <- pnorm(gold_test_no_NAs, mean = test_ar1_forecast$pred,
                 sd = test_ar1_forecast$se)
```

Both PITs show non-uniform distributions, with overrepresentation in the medians, indicating imprecision in the forecasts, although moreso for the AR 1:

```r
par(mfrow = c(1, 2))
hist(aa_PIT, breaks = seq(0, 1, 0.1), las = 1)
hist(ar1_PIT, breaks = seq(0, 1, 0.1), las = 1)
```



References

Dawid, A. P. 1984. "Present Position and Potential Developments: Some Personal Views: Statistical Theory: The Prequential Approach." *Journal of the Royal Statistical Society. Series A (General)* 147 (2): 278–92. http://www.jstor.org/stable/2981683.

Hyndman, Rob, George Athanasopoulos, Christoph Bergmeir, Gabriel Caceres, Leanne Chhay, Mitchell O'Hara-Wild, Fotios Petropoulos, Slava Razbash, Earo Wang, and Farah Yasmeen. 2021. *forecast: Forecasting Functions for Time Series and Linear Models*. https://pkg.robjhyndman.com/forecast/.

Jordan, Alexander, Fabian Krüger, and Sebastian Lerch. 2019. "Evaluating Probabilistic Forecasts with scoringRules." *Journal of Statistical Software* 90 (12): 1–37. https://doi.org/10.18637/jss.v090.i12.

Matheson, James E., and Robert L. Winkler. 1976. "Scoring Rules for Continuous Probability Distributions." *Management Science* 22 (10): 1087–96. http://www.jstor.org/stable/2629907.

R Core Team. 2020. *R: A Language and Environment for Statistical Computing*. Vienna, Austria: R Foundation for Statistical Computing. https://www.R-project.org/.