Forecasting Notes

December 17, 2020

1 DataCamp: forecasting in R course

1.1 Exploring time series

- We can explore time series using the *forecast* package.
- We can use the *stats::window()* function to select a segment of a timeseries based on *c(year, period)* (quarter) vs indices
- 3 key patters in timeseries
  - Trend - long term increase or decrease
  - Seasonality - regular pattern over fixed period e.g. annual
  - Cyclical - regular pattern but with no fixed period
- *forecast::ggseasonplot()* creates plots by year, rather than continuous
- *forecast::ggsubseriesplot()* creates plots by quarter over all years
- Lag plots can be used to plot one observation against another (state-space) for autocorrelation
  - *forecast::gglagplot()* creates the lag plot
  - *forecast::ggAcf()* calculates the autocorrelation and creates a plot for each lag
  - Trends induce positive correlations in the early lags
  - Seasonality will induce peaks at the seasonal lags
  - Cyclicity induces peaks at the average cycle length
- "White noise" a purely random time series and is basis of forecasting models
  - Can use sampling data of ACF to estimate bounds of significance
  - Use Ljung-Box test to test a group of autocorrelations together, rather than each separately
    - Apply to *diff()* of timeseries
    - $p < 0.05$ would constitute a "fail" as it means there is information in the residuals that hasn't been captured by the model

1.2 Benchmark methods and forecast accuracy

- Run lots of simulations based on statistical model and the mean/median is the "point forecast"
  - Should provide prediction intervals
- **Naive** forecast model is very simple, and provides a baseline for more complicated models, that sometimes don't perform better
  - Uses most recent obs as next obs

    - *forecast::naive()* fits Naive forecast
    - *forecast::snaive()* fits seasonal Naive forecast

- See how good forecast is by testing on data already seen
    - Fitted values are forecasts based on all prior values (one-step forecasts)
        - When parameters estimated, not really forecasts as all data was used to estimate parameters
    - Use residuals to evaluate model (**always check residuals before moving forward with model**)
        - Residuals should look like (gaussian) white noise if good model
        - Make 4 assumptions (first two critical, last two convenient)
            1. Residuals should be uncorrelated
                - Otherwise there is information in residuals that should have been captured by forecasting methods
            2. Residuals have mean zero
            3. Residuals have constant variance
            4. Residuals are normally distributed (required for gaussian white noise vs white noise)
        - *forecast::checkresiduals()* plots residuals, autocorrelation, histogram, and performs Ljung-Box test
- Forecast errors $\neq$ residuals
    - Forecast errors are
        - Errors on **test** set
        - Based on multi-step forecasts
    - Residuals are
        - Errors on **training** set
        - Based on one-step forecasts
- Best to use **Mean Absolute Scaled Error** over **MAE** or **MSE** when comparing errors in forecasts on different time series as may have different scales
    - $MASE = \frac{MAE}{Q}$, where Q = scaling factor
- *forecast::accuracy()* computes common metrics used for evaluating both residuals and forecast errors
- Cross-validation can be performed in a number of ways
    - Multiple one-step forecasts can be made progressively moving forward by one observation each time (rolling origin) and averaging metrics
        - Can be applied to multi-step forecasting e.g. two-steps ahead, and three-steps ahead etc
    - *forecast::tsCV()* can perform cross-validation
        - Need to compute own error measures

```
library(forecast)
library(fpp2)
library(tidyverse)
sq <- function(u){u^2}
for(h in 1:10){
    oil %>% tsCV(forecastfunction = naive, h = h) %>%
    sq() %>% mean(na.rm = TRUE) %>% print()
}
## [1] 2355.753
## [1] 4027.511
## [1] 5924.514
## [1] 7950.841
## [1] 9980.589
## [1] 12072.91
## [1] 14054.23
## [1] 15978.92
## [1] 17687.33
## [1] 19058.95
```

        - Can see how the RMSE is increasing with increasing forecast horizon (h)

1.3 Exponential smoothing

1.3.1 Simplet exponential smoothing models

- Balance naive and mean forecast models by including all information, but more heavily weighting more recent observations
- $\hat{y}_{t+h|t} = \alpha y_t + \alpha(1-\alpha)y_{t-1} + \alpha(1-\alpha)^2 y_{t-2} + \ldots$, where $0 \leq \alpha \leq 1$
  - Describing a function where weights (α terms) decrease exponentially as you go back in time
- Equation can be re-written as:
  - $\hat{y}_{t+h|t} = \ell_t$, where $\ell_t = \alpha y_t + (1-\alpha)\ell_{t-1}$
  - ℓ_t is known as the "level" and is the "smoothing function"
    - It is the smoothed value, so updates over time
    - Need to estimate ℓ_0, the initial value, and then just update
- We choose α and ℓ_0 to minimize SSE (least squares)
  - $SSE = \sum_{t=1}^{T} \left(y_t - \hat{y}_{t|t-1}\right)^2$
    - Have to use this non-linear optimization routine to minimize
- *forecast::ses()* function performs simple exponential smoothing
- *ggplot2::autolayer(fitted(*ses model*))* useful way of overlaying fitted values as a layer to an *autoplot()* rather than creating a new plot
- Only works well when no trend or seasonality

1.3.2 Holt's linear trend model

- Adjusts SES by adding linear trend
- Forecast $\hat{y}_{t+h|t} = \ell_t + hb_t$ where:
  - $\ell_t = \alpha y_t + (1-\alpha)(\ell_{t-1} + b_{t-1})$ ("level")
  - $b_t = \beta^*(\ell_t - \ell_{t-1}) + (1-\beta^*)b_{t-1}$ ("trend")
    - β^* controls how quickly the slope can change
    - Because slope can change, often referred to as local linear trend
  - $0 \leq \alpha, \beta^* \leq 1$
  - We choose smoothing parameters α and β^*, and state parameters ℓ_0 and b_0 to minimize SSE (least squares)
- A modification can be made to allow the model to "dampen" and taper off to a value
  - $\hat{y}_{t+h|t} = \ell_t + (\phi + \phi^2 + \ldots + \phi^h)b_t$ ("forecast")
  - $\ell_t = \alpha y_t + (1-\alpha)(\ell_{t-1} + \phi b_{t-1})$ ("level")
  - $b_t = \beta^*(\ell_t - \ell_{t-1}) + (1-\beta^*)\phi b_{t-1}$ ("trend")
  - $0 \leq \phi \leq 1$
    - When $\phi = 1$, produces Holt linear trend

1.3.3 Holt-Winter's model

- Adapted to deal with seasonality
- Two versions:
  1. Additive
     - $\hat{y}_{t+h|t} = \ell_t + hb_t + s_{t-m+h_m^+}$ ("forecast")
     - $\ell_t = \alpha(y_t - s_{t-m}) + (1-\alpha)(\ell_{t-1} + b_{t-1})$ ("level")
     - $b_t = \beta^*(\ell_t - \ell_{t-1}) + (1-\beta^*)b_{t-1}$ ("trend")
     - $s_t = \gamma(y_t - \ell_{t-1} - b_{t-1}) + (1-\gamma)s_{t-m}$
     - $s_{t-m+h_m^+}$ is a seasonal component
       * m is the period of seasonality e.g. quarter
       * seasonal component averages **zero**
     - $0 \leq \alpha \leq 1, 0 \leq \beta^* \leq 1, 0 \leq \gamma \leq 1-\alpha$
  2. Multiplicative
     - $\hat{y}_{t+h|t} = \ell_t + hb_t + s_{t-m+h_m^+}$ ("forecast")
     - $\ell_t = \alpha\frac{y_t}{s_{t-m}} + (1-\alpha)(\ell_{t-1} + b_{t-1})$ ("level")
     - $b_t = \beta^*(\ell_t - \ell_{t-1}) + (1-\beta^*)b_{t-1}$ ("trend")
       * Trend is stil linear
     - $s_t = \gamma\frac{y_t}{\ell_{t-1} - b_{t-1}} + (1-\gamma)s_{t-m}$
       * Seasonality is now multiplicative
     - $s_{t-m+h_m^+}$ is a seasonal component
       * m is the period of seasonality e.g. quarter
       * seasonal component averages **one**
     - $0 \leq \alpha \leq 1, 0 \leq \beta^* \leq 1, 0 \leq \gamma \leq 1-\alpha$
     - Use multiplicative when seasonal variation increases with the level of the series (as time goes on the smoothed value increases)
- Can add damping to trend of HW models, as with Holt's linear trend models
  - Damping can be either additive or multiplicative, however, multiplicative trend damping generally doesn't work well
- *forecast::hw()* is used for the Holt-Winter's (and therefore Holt's linear trend) models
  - Set *seasonality = "additive/multiplicative"*

1.3.4 Innovations state space models

- The exponential smoothing models discussed are known as **state space** models
  - Each model consists of an equation that describes the observed data, and some state equations that describe how the unobserved components or states (level, trend, seasonal) change over time
  - To demonstrate, let's look at an SES model
    1. Recall $\hat{y}_{t+h|t} = \ell_t$ and $\ell_t = \alpha y_t + (1-\alpha)\ell_{t-1}$
    2. Rewrite the level function in the "error correction" form
       * $\ell_t = \alpha y_t + (1-\alpha)\ell_{t-1}$
       * $\ell_t = \ell_{t-1} + \alpha(y_t - \ell_{t-1})$
       * $\ell_t = \ell_{t-1} + \alpha e_t$ where:
         · $e_t = y_t - \ell_{t-1}$
         · $e_t = y_t - \hat{y}_{t|t-1}$
         · e_t is therefore the residual at time t
       * Assuming residuals are normally and independently distributed with mean 0 and variance σ^2 ($e_t = \varepsilon_t \sim NID(0, \sigma^2)$)

3. Rewrite *measurement* and *state* equations
    * $y_t = \ell_{t-1} - \varepsilon_t$
    * $\ell_t = \ell_{t-1} + \alpha\varepsilon_t$

- Additive and multiplicative models will produce the same "point forecast", however, they will differ in their prediction intervals as they will exhibit different errors
- We can label state space models as **ETS** models (Error, Trend, Seasonal) with the following possible labels:
    - Error = $\{A, M\}$, where A, M = Additive, Multiplicative
    - Trend = $\{N, A, A_d\}$, where N, A_d = None, Additive damped
    - Seasonal = $\{N, A, M\}$
- Multiplicative errors means noise increases with the level of the series (prediction intervals get much wider than additive)
- ETS is useful as it allows us to:
    - Use MLE to optimize parameters
    - Generate prediction intervals for all models
    - Automatically select the best exponential smoothing model for timeseries
        * Minimize bias-corrected version of AIC ($\mathrm{AIC_c}$)
            · Similar to cross-validation, but much faster
            · Equivalent to minimizing SSE in models with additive errors
- *forecasts::ets()* automatically selects ETS model using $\mathrm{AIC_c}$
    - Need to pass to *forecast::forecast()* function for predictions
- ETS models are not necessarily better than simpler ones e.g. seasonal naive

1.4 Forecasting with ARIMA models

- ETS use multiplicative errors and seasonality to handle variance that increases with levels
    - Can use transformations to adjust instead
- Box-Cox transformations used to create common variance

$$w_t = \begin{cases} \log(y_t) & \lambda = 0 \\ \left(y_t^{\lambda} - 1\right)/\lambda & \lambda \neq 0 \end{cases}$$

    - *forecast::BoxCox.lambda()* will select best value of λ
    - Essential to use transformation of some kind with ARIMA models
- Non-seasonal autoregressive integrated moving average models
    - Autoregressive models are regression of time series against lagged **values** of series
    - Moving average models are regression against lagged **errors**
    - Adding two you get an ARMA model
        * Last p observations, and last q errors are used as predictors in the equation
        * Can only work with stationary data so need to difference data first
            · Stationary timeseries have properties that don't depend on the time i.e. no trends or seasonal components, but can have cyclical patterns
    - **Integrated** is the opposite of differencing (e.g. integration vs differentiation)
        * If timeseries needs to be differenced d times, known as ARIMA (p, d, q) model
            · d = number of lag-1 differences

        - p = number of ordinary AR lags: $y_{t-1}, y_{t-2}, \ldots, y_{t-p}$
        - q = number of ordinary MA lags: $\varepsilon_{t-1}, \varepsilon_{t-2}, \ldots, \varepsilon_{t-q}$
  - *forecast::auto.arima()* automatically selects best values of p, d, q by minimizing AIC$_c$
    - Parameters estimated using MLE
    - Can only compare AIC$_c$ values between models of same class (e.g. ARIMA vs ARIMA, not ARIMA vs ETS) and same differencing
    - Stepwise search to test parameters means that may find local minimum
    - *drift* term refers to c, which is the average change between consecutive observations
  - Use cross-validation to compare models of different classes
    - Find MSE of each class

```
library(tidyverse)
library(forecast)
library(fpp2)

# ETS function for CV
fets <- function(x, h) {
    forecast(ets(x), h = h)
}
# ARIMA function for CV
farima <- function(x, h) {
    forecast(auto.arima(x), h=h)
}

# Compute CV errors for ETS on austa as e1
e1 <- tsCV(austa, fets, h = 1)

# Compute CV errors for ARIMA on austa as e2
e2 <- tsCV(austa, farima, h = 1)

# Find MSE of each model class
mean(e1^2, na.rm = TRUE)
## [1] 0.05623684
mean(e2^2, na.rm = TRUE)
## [1] 0.04336277
```

- Seasonal ARIMA
  - Uses same parameters, but uppercase to signify seasonal
    - D = number of seasonal differences
    - P = number of seasonal AR lags: $y_{t-m}, y_{t-2m}, \ldots, y_{t-Pm}$
    - Q = number of seasonal MA lags: $\varepsilon_{t-m}, \varepsilon_{t-m}, \ldots, \varepsilon_{t-Qm}$
    - m = seasonal period (number of observations per year)
  - Non-linear as seasonal components multiply with non-seasonal components
  - Allow seasonality to **change over time**
    - Mainly affected by seasonality at end of series, and not as much by the start
  - When using **at least two differences**, forecast will have trend without needing to include drift term (e.g. ordinary and seasonal differencing)

1.5 Advanced methods

- Often want to include information from other variables to improve our forecast (rather than just relying on changes in outcome variable)
- Dynamic regression

  - $y_t = \beta_0 + \beta_1 x_{1,t} + \ldots + \beta_r x_{r,t} + e_t$
    * Looks very similar to standard linear regression equation, but error term is ARIMA process
  - *forecast::auto.arima(df[, "outcome"], xreg = df[, "var"])* used to fit dynamic regression model
  - *forecast::forecast(fit, xreg = rep(value, period))* used to forecast different scenarios of the input variable (value and period)

- Dynamic harmonic regression
  - Uses Fourier terms to account for seasonality
  - $y_t = \beta_0 + \beta_1 x_{1,t} + \ldots + \beta_{t,r} x_{t,r} + \sum_{k=1}^{K} [\alpha_k s_k(t) + \gamma_k c_k(t)] + e_t$
    * $s_k(t) = \sin\left(\frac{2kt}{m}\right)$
    * $c_k(t) = \cos\left(\frac{2kt}{m}\right)$
    * $m =$ seasonal period
    * Usually don't use ARIMA errors as seasonality already accounted for
    * Assume seasonal pattern doesn't change over time
  - *forecast::auto.arima(df, xreg = fourier(df, K = k), seasonal = FALSE, lambda = w)* fits dynamic harmonic regression
    * Select K to minimize AIC_c (cross-validation?)
    * $K < \frac{m}{2}$
    * *seasonal = FALSE* indicates don't want to produce ARIMA errors
  - *fit %>% forecast::forecast(xreg = fourier(df, K = k, h = h))* forecasts using the dynamic harmonic regression model
    * Selecting h indicates want to forecast forward
  - Fourier terms vs ARIMA is that handles large values of m well
    * E.g. seasonal period of weekly data approx 52
    * When annual seasonality with daily data ($m = 365$)

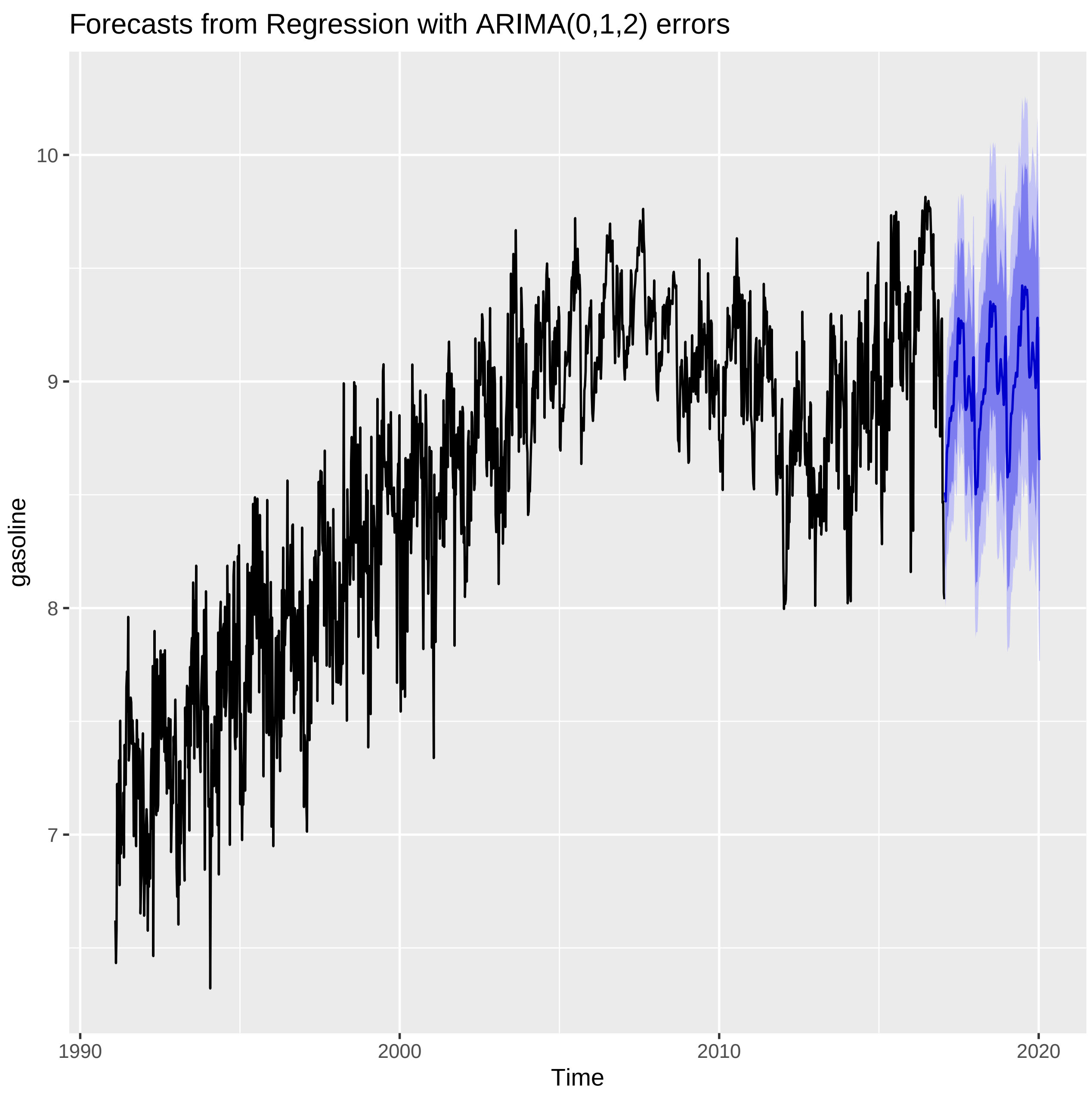
```r
library(tidyverse)
library(forecast)
library(fpp2)

# Set up harmonic regressors of order 13
harmonics <- fourier(gasoline, K = 13)

# Fit regression model with ARIMA errors
fit <- auto.arima(gasoline, xreg = harmonics, seasonal = FALSE)

# Forecasts next 3 years
newharmonics <- fourier(gasoline, K = 13, h = 52*3)
fc <- forecast(fit, xreg = newharmonics)

# Plot forecasts fc
autoplot(fc)
```



- Harmonic regressions are useful when there is multiple seasonality in a timeseries e.g. daily and weekly seasonality
    - Need to specify both values of K in *fourier()* e.g. *fourier(df, K = c(10, 10))*
    - *forecast(fit, newdata = data.frame(fourier(df, K = c(10, 10), h = 20*48)))*
- Sometimes *forecast::auto.arima()* is too slow when fitting very long timeseries (particularly with multiple seasonality), so use Fourier terms in *forecast::tslm()* function
    - *tslm(df ∼ fourier(df, K = c(10, 10)))*
    - Works like regular *lm()*

- TBATS model
    - **T**rigonometric terms for seasonality
        - Allows seasonality to change over time, unlike Fourier terms
    - **B**ox-Cox transformation for homogeneity
    - **A**RMA errors for short-term dynamics
    - **T**rend and level terms, including damping, like ETS model
    - **S**easonal (including multiple and non-integer periods)

- Very general and handles large range of timeseries
- Very useful for timeseries with large seasonal periods and multiple seasonal periods
- Have to be careful with its automation
  * Point forecast often good
  * Prediction intervals can be very wide
  * Slow with large timeseries
- *forecast::tbats(df)* used to fit model