|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| *start(df)*  *end(df)*  *frequency(df)* | when first record was made  when last record was made  num of records per unit time | | | | *deltat(df)*  *time(df)*  *cycle(df)* | time increment btw records  calculate vector of time indices for observations  position in cycle of each observation | |
| *diff(..., lag = period)* | | seasonal diff transformation: remove periodic trends | | | | | |
| *ts.plot(cbind(df1, df2))* | | 2 plots on same graph | | *diff(log(...))* | | | generate log returns |
| Autoregressive  Mean-centred version | | Yt =  Large value of = greater autocorrelation. Negative values of = oscillatory time series  If = 0 and = 1, then , which is a random walk. Then {Yt} is not stationary | | | | | |
| *arima.sim(model = list(ar = phi), n = 50)*  Persistence = high correlation btw obs and its lag  *AR <- arima(df, order = c(1,0,0))* | | Simulate AR model  Anti-persistence = large variation btw obs and its lag  Fit data to AR model (on R, ar1 = , intercept = , sigma^2 = ) | | | | | |
| *arima.sim(model = list(ma = theta), n = 50)*  *MA <- arima(df, order = c(0,0,1))*  *fitted <- df - residuals(MA)* | | Simulate MA model  Fit data to MA model (on R, ma1 = , intercept = , sigma^2 = )  . Residuals = = | | | | | |
| MA(1) (e.g. )  AR(2) (e.g. Yt = ) | | *arima.sim(model = list(order = c(0,0,1), ma = ), n = 100)*  *arima.sim(model = list(order = c(2,0,0), ar = c(0, )), n = 100)* | | | | | |
| *acf2(df)*  *sarima(df, p = .., d = .., q = ..)* | | Calculate ACF and PACF pairs  Fit data to model | | | | | |
| *sarima.for(df, n.ahead = .., p, d, q)* | | Forecasting | | | | | |
| *acf2(df, max.lag = 60)*  *sarima(df, 0, 0, 0, P = .., D = .., Q = .., S = ..)* | |  | | | | | |
| Mixed seasonal model  *lx <- log(x). dlx <- diff(lx). ddlx <- diff(dlx, 12)*  *sarima.for(df, n.ahead = .., p, d, q, P, D, Q, S)* | | SARIMA(p, d, q) (P, D, Q)S model  Log to standardise var. diff to remove trend (but still have seasonal behavior). diff again to get stationary. All 3 = {d = 1, D = 1}  Forecasting | | | | | |
| Simple exponential smoothing | | Use all obs for forecast with more recent obs having higher weights  *fc <- ses(df, h = 5)*  *summary(fc)*Choose and by minimizing SSE = | | | | | |
| Holt's linear trend  *df %>% holt(h = 5) %>% autoplot* | | Small = slope hardly change, so linear trend. High = slope change rapidly nonlinear trend  Choose , , , by minimizing SSE | | | | | |
| Damped trend method  - allows trend to dampen over time, s.t. it levels off to a constant value  *holt(df, damped = TRUE, h = 5)* | | . .  . Damping param: 0 < < 1  Larger = less damping short run forecasts are trended, long run forecasts are constant | | | | | |
| Holt-Winters additive mtd  (deal w trend & seasonality)  *hw(df, seasonal = "additive", h = 3)* | | . .  .  = seasonal component from final year of data. 0 ≤ , ≤ 1. 0 ≤ ≤ 1 -  m = period of seasonality (e.g. m = 4 for quarterly data). Seasonal component averages 0 | | | | | |
| Holt-Winters multiplicative mtd  (deal w trend & seasonality)  *hw(df, seasonal = "multiplicative", h = 3)* | | . .  .  = seasonal component from final year of data. 0 ≤ , ≤ 1. 0 ≤ ≤ 1 -  Seasonal component averages 1. Trend is still linear, but seasonality is multiplicative | | | | | |
| *BoxCox.lambda(df). ets(df, lambda)* | | Find estimate of lambda to stabalise var using BoxCox transformation | | | | | |
| ARIMA(p, d, q, include.constant = TRUE)  *auto.arima(df)*  - auto.arima based on Hyndman-Khandakar algo  ARIMA(p, d, q, P, D, Q, M)  *auto.arima(df, lambda, stepwise, stationary)* | | I: Integrated (opp of differencing). d = num of times ts needs to be differenced to make it stationary  Selects p and q by minimizing AICc value. Select d via unit root tests. Estimate params using MLE  - AICc can only be compared btw model of same class (ARIMA/ETS only), & same amt of differencing  p/q = num of ordinary AR/MA lags. d = num of lag-1 diff. P/Q = num of seasonal AR/MA lags  D = num of seasonal diff. m = num of obs per year.  lambda for Box-Cox transformation. stepwise = FALSE (to search for more models). stationary = TRUE | | | | | |
| Dynamic regression (regression w ARIMA errors)  *fit <- auto.arima(df[, y], xreg = df[, x])*  *forecast(fit, xreg = c(xt, xt+1, ...))* | | , where = some external info and is an ARIMA process  Compared to ordinary regression where is WN  xreg = matrix of predictor variables to include in model  Need input future values of x | | | | | |
| Dynamic harmonic regression  *auto.arima(df, xreg = fourier(df, K = 1), seasonal = FALSE, lambda = 0)*  *forecast(fit, xreg = fourier(df, K = 1, h = 24))* | | Every periodic fn can be approx by sums of sin and cos terms (Fourier terms) for large enough K  . m = seasonal period  Since sk(t) and ck(t) are already seasonal terms, can be modeled as a non-seasonal ARIMA process  Fourier terms assumes no change in seasonal pattern, SARIMA allows seasonal pattern to evolve over time. Higher K allows more complicated seasonal pattern.  Use harmonic regression when m is too large (e.g. 52)  Try diff values of K and select model w lowest AICc value. K ≤ m/2  Can also add other predictor variables to xreg | | | | | |
| Harmonic regression for multiple seasonality  *fit <- tslm(taylor ~ fourier(df, K = c(10, 10))*  *forecast(fit, newdata = data.frame(fourier(df, K = c(10, 10), h = 52))* | | | auto.arima takes long time to fit long time series. Use tslm with Fourier terms instead (time series linear model). For multiple seasonality, specify K for each seasonal period | | | | |
| TBATS  *fit <- tbats(df)*  TBATS(Box-Cox, ARMA(p, q), damping, Fourier terms(<Seasonal period, K>}, <Period2, K2>) | | Trigo terms for seasonality (similar to Fourier terms) + Box-Cox transformations for heterogeneity + ARMA errors for short-term dynamics + Trend (can be damped) + Seasonal (including multiple & non-integer periods)  TBATS is entirely automated, but might be slow and not the best model | | | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *ls1 <- list(A = seq(1, 5, by = 2), B = seq(1, 5, length = 4)) ... ls1$A[2]*  *x <- c(1,2,3), y <- c("1", "2", "3"). df <- data.frame(x, y) ... df$x. df[, "x"]. df[c(3,2), ]*  *read.csv(). head(). tail(). summary(). ls(). length(). seq(). mean(). median(). sd(). var().*  *library(tidyverse); tbl <- as\_tibble(df)*  *filter(tbl, condition1, condition2, ...). rename(new = old, new2 = old2...)*  *select(tbl, col2: col4). select(tbl, !c(col2: col4)). select(tbl, last\_col(offset = 1):last\_col())*  *?function. or help(function)*  *tbl %>% filter(cond1) %>% select(col2:col4)*  *ggplot(tbl) + geom\_point(mapping=aes(x = col1, y = col2)* | | creates list  creates dataframe  common functions  tibble  filter, rename  select, mutate, arrange  Get R documentation  piping | ```{r}  # Write R code  ``` | R markdown  Code chunks |
| |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | X, Y indep: E(XY) = E(X)E(Y) | Var(X) = E(X - µ)2. E(X) = µ | If a is a constant, Cov(a, X) = 0 | | Cov(X, X) = Var(X) | Cov(aX, bY) = abCov(X, Y) | | Corr(X, Y) = | If X, Y indep, then Cov(X, Y) = 0 | | Let U = , V = , then Cov(U, V) = | | | | Cov(X, Y) = Cov(Y, X) = E[(X - µX)(Y - µY)] = E(XY) - µXµY = | | | Var() = | | | | | | | |
| f(x) =  Taylor series expansion  ex  ln(1 + x) | Gaussian dist pdf. X ~ N(µ, ). If both X, Y are jointly Gaussian, then they are indep iff Cov(X, Y) = 0  f(x) =  1 + x +  x - | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *ts <- tsibble(*  *year = 2015:2019, y = c(123,39,23,32,11)*  *index = year)*  *q <- seq(as.Date("2016-01-01"), as.Date(2016-12-31"), by = "1 day")*  By Year*: Use integers in R*  Quarter*: yearquarter(q)*  *filter(ts, col1 %in% c("123", "456")) %>%*  *autoplot(.vars=y) + geom\_point() +*  *scale\_color\_discrete(labels=c("key1", "key2 ")) +*  *labs(title = "", y = "", x = "")* | | | Create tibble w time component  Types of col in Tsibble: 1. measurement/obs/record, 2. Index, 3. Key  Creates datetime seq. by = day/week/month/quarter/year  For Index column. Also have Monthly: *yearmonth()*,  Weekly: *yearweek()*. Daily: *as\_date()*, *ymd()*. Sub-daily *as\_datetime()*, *ymd\_hms()*  Must only have 1 index col in tsibble. Can have multiple measurement/key cols | |
| Time Series Patterns  When looking at graph, consider:  1) Is there a trend? Linear? Change over time?  2) Seasonal effect? Period?  3) Sudden dips/spikes? When?  4) Non-constant variance? | Trend: trend exists when there is a long-term incr/decr in data. Does not have to be linear  Level: height of series on the ordinate axis  Seasonal: seasonal pattern exists when a series is influenced by factors like quarters/month/day/time. (For ST4253, seasonality always fixed and known period: monthly have period of 12, quarterly 4)  Cycle: rise and fall not of fixed period | | | |
| Seasonal plot  *filter(ts, col1=="xxx",*  *Quarter <= yearquarter("1995 Q4")) %>%*  *gg\_season(y=yyy, labels="left")* | Similar to time plot, except time plot is chopped up into individuals periods, aligned and plotted on an axes for a single period  Compare same period across years  See if similar pattern occur across years  E.g. Arrivals in Q2 generally lower than other quarters  Lines are close to horizontal = no seasonal effect | | | |
| Subseries plots (study seasonal effects)  *gg\_subseries(ts, y = yyy)* | Find patterns within each season  Q1 plot | Q2 plot | Q3 plot | Q4 plot | | | |
| Scatterplots  *library(GGally)*  *pivot\_wider(ts, names\_from = "Origin", values\_from="Arrivals") %>%*  *ggpairs(columns=2:5)*  Diagonal = density plots  Scatterplot w diagonal pattern suggests 2 time series are similar (e.g. UK-US)  Can also see strongest correlation is for US-UK pair  Can use UK arrivals to predict US arrivals | | | |  |
| Lag plots  Scatter plot by plotting yt-h on the abscissa (horizontal axis) and yt on the ordinate(vertical axis)  *gg\_lag(ts, geom = "point", alpha = 0.3, y = yyy)*  For lag 1,2,3,4 there is a strong linear r/s, i.e. to predict yt+1, can use yt, yt-1, yt-2, yt-3 |  | | | |
| White Noise (corr = cov = 0)  On graph, adjacent points don't tell us anything about neighbouring points  Sample ACF  *ACF(ts) %>% autoplot()*  If most vertical lines are close to 0 and within dashed lines (95% CI) = true autocorrelations are all 0, i.e. series is WN | | Consider a ts that consists of *uncorrelated* r.v. {et}, s.t. for all t ≥ 1, E(et) = 0, Var(et) =  This is known as White Noise(WN), et ~ WN(0, )  Special WN: et ~ N(0, ) for all t. This is iid Gaussian, so it is aka Gaussian WN.  rh = estimate of correlation btw yt and yt-h = , h = 0, 1, 2, ...  where T is length of series, is sample mean of T observations  rh is to estimate correlation btw yt and yt-h, , the correlation at lag h | | |
| Theoretical ACF for GWN  Let be autocovariance fn for GWN, i.e. = Cov(et, et-h) | When h = 0, = Cov(et, et) = Var(et) = . When h ≥ 1, = Cov(et, et-h) = 0  So if we let = Corr(et, et-h), then | | | |
| Random walk: yt = yt-1 + et, where et ~ WN(0, ), and y0 = 0  Var increase over time  Covariance also not constant  r10 is not a good estimate of Cov(y10, y20) | y1 = y0 + e1 = e1. ... yt = . = Corr(ys, yt) = Cov(ys, yt) /  E(yt) = E() = = 0. Var(yt) = var() = = t (since et uncorrelated)  For s ≥ t ≥ 1, cov(ys, yt) = Cov(, ) = t = min{s,t}  Corr(ys, yt) = t / =  Let t = s-1, then as s ∞, = 1  1) r = estimate of correlation assume cov btw y10 and y20 is same as y20 and y30 and ...  2) var(y20) ≠ var(y10) | | | |
| Sample ACF is used for:  ACF is +ve when obs are both above/below mean. ACF is -ve when obs are on opp side of mean | 1) Check if residuals are WN (close to 0 and btw blue lines)  2) Indicate if there is a strong trend remaining in the data  3) Indicate seasonality, if any  4) If dies down relatively quickly, suggest ARIMA models are appropriate | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Population adjustments  *covid\_data <- readRDS(".....rds") %>%*  *filter(between(date, ymd("2021-01-01"), ymd("2021-12-31")))* | | | | Track data per person or per 1000 people...  *autoplot(covid\_data, .vars = patients\_per\_million) + labs(x = , y = ) + scale\_color\_discrete(labels = c("key1", "key2"))* |
| Calendar adjustments  *gg\_season(ts, y=obs, labels = "left")*  *mutate(ts, adj\_y = obs/days\_in\_month(index))*  *%>% gg\_season(y = adj\_obs, labels = "left")*  Trading day variation  Holiday variation | | | Remove effect of num of days in month, weekend effects, ... to make data simpler  See that obs are lower in months w < 31 days. This causes jagged pattern at troughs on graph  To rectify diff in obs due to num of days in month  Due to changing num of times each day of the week occurs in a mth  Due to presence/absence of a holiday in a mth | |
| To adjust for trading day variation:  Let {yt} be a monthly TS that is a total of some variable for each mth.  Let y't = (yt / num of days in mth t) \* 30.4375 (from 365.25/12) | | | We assume y't = Tt + St + Ct + et (trend effect + seasonal effect + trading day effect + noise)  where Ct = , where = 0 and dit = fraction of mth t that is day i \* 30.4375  To estimate : 1) Decompose y't into y't = Tt + St + e't. Obtain wt = y't - -  2) Regress wt on d1t, d2t, ..., d7t. Obtain estimates  3) Calendar adjusted time series is y''t = y't - | |
| Inflation adjustments  In SG, we have a monthly price index, Consumer Price Index (CPI) | | | Price index for year t1 relative to year t2 =  Adj price at time t1 relative to time t2 = | |
| Transformations  wt = , wt = , wt = log(yt)  Box-Cox Transformations  Suppose yt > 0 t  Using L'Hopital's rule Let g() = ()/  *l1 <- guerrero(ts$obs)*  *mutate(ts, transformed = box\_cox(y, l1)) %>% autoplot(.vars = transformed)* | | | Use if TS has diff variation/variance at diff levels of the TS  Log transformation are more interpretable: if log10 is used, an increase of 1 on log scale = multiplication of 10 on original scale  , = 1, 1/2, 1/3, 0, -1 (linear transformation, square root + linear, cube root + linear, natural log, inverse transformation)  = log yt  Compute optimal value of to use  Intuitive understanding: w = h(y), Var(w) = Var(h(y)) ≈ [h'(E(y))]2 \* Var(Y) (which we want to find a for Var(w) to be constant) | |
| Back-transforming Forecasts    To reverse transformations to obtain forecasts on original scale  f(w) is pdf of w  = E(w)  ∫ f(w) dw = 1  ∫ w \* f(w) dw = E(w) =  ∫ (w - )2 f(w) dw = E[(w - )2] = Var(w) | Want to estimate E(yt) and no transformation used. We use to estimate E(yt) and estimator is unbiased, i.e. E() = E(yt)  Problem w using transformations is that the back-transformed forecast ≠ the mean, but the median of the forecast dist. | | | |
| Suppose we use transformation wt = log(yt). So we expect yt = exp wt. Then using estimate, , and get = exp .  Using Taylor's expansion, h(w) = h(a) + h'(a)(w-a) + h''(a) + ...,  However, E(yt) = E(ew) = ∫ewf(w) dw ≈ = = = =  So E(yt) ≠ E(exp ), i.e the back-transformed estimate is biased. | | | |
| So if estimates , then formula below estimates E(yt) for . (proof above only shows for )  To adjust for this bias, use back transformation:  where is the var of wt. | | | |
| Features  Summary statistics: *features(ts, obs, quantile)*  Tiled statistics: *features(ts, obs, features = feature\_set(tags="tile"))*  Roll statistics: *features(ts, obs, features = feature\_set(tags="roll"))* | | Quick comparison btw multiple TS  Quantiles for indication of spread and symmetry of nums  TS is divided into non-overlapping windows. Mean and var are computed for each window. Var of window means = how "stable" series is. Var of window var = how "lumpy" series is  Roll statistics computed using overlapping windows. Used to identify where TS had sharp changes in level, variability and distribution.  Shift\_level\_index = index in TS w largest shift in level. Shift\_var\_index = index in TS w largest shift in var | | |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Benchmark Forecasting Mtds  h = forecasting horizon  *library(fpp3)*  *report(mtd)*  *glance(mtd)*  *tidy(mtd)*  Use these mtds as benchmarks to compare w better models  # show\_gap connects last obs w forecast | | | | 1. Average Mtd: Forecast of all future values = mean of historical data, i.e.  *mean\_mtd <- model(ts, avg = MEAN(y)); report(mtd)* # look at model summaries (also have others on LHS)  *forecast(mean\_mtd, h = 10) %>% autoplot(data = ts, level=NULL)* # level for prediction intervals | | | | | |
| 2. Naive Mtd: Forecast = most recent observation, i.e. , where et ~ WN(0, ) and y0 = 0  *mean\_and\_naive <- model(ts, avg = MEAN(y), naive = NAIVE(y))* # This creates a tibble w 2 models, MEAN & NAIVE  # NAIVE can be replace by *RW()*: random walk w/o drift, since yt = yt-1 + et, or yT+h = yT + . So E(yT+h|y1,…,yT) = yT  *forecast(mean\_and\_naive, h=10) %>% autoplot(data = ts, level=NULL)* # will product 2 forecasts | | | | | |
| 3. Random Walk w Drift: Constant diff btw successive observations apart from the random noise, i.e. yt = + yt-1 + et for t ≥ 1, where et ~ WN(0, ) and y0 = 0. OR . Since E(yT+h|y1, …, yT) =  Naive mtd to estimate is to use average of diffs btw lag 1 obs, i.e.  Then forecast,  *rwf <- model(ts, rwf = RW(y ~ drift())); forecast(rwf, h=10) %>% autoplot(data = ts, level=NULL)* | | | | | |
| 4. Seasonal Naive Mtd: Forecast = last observed value from the same season of the previous year  Suppose period of time series is m, e.g. m = 12 for monthly data. Then forecast, , where k =  *sn <- model(ts, sn = SNAIVE(y)); forecast(sn, h=10) %>% autoplot(data = ts, level=NULL, show\_gap=FALSE)* | | | | | |
| Residuals | | et = yt - = yt - . Residuals are based on 1-step ahead forecasts.   |  |  |  | | --- | --- | --- | | Suppose we use wt | Innovation residuals = residuals on the transformed scale | Residuals aka training errors aka Response residuals | | wt = log yt | wt - | yt - , where is back-transform of w bias adjustment factor | | | | | | | | |
| Extracting Residuals and Fitted Values | | | | | *fitted\_and\_resids <- augment(mean\_mtd) # returns a tsibble* | | | | |
| Essential Properties of Residuals | | | | 1. Residuals are uncorrelated. If residuals are correlated: ARIMA model might be appropriate.  2. Residuals have mean 0. If mean ≠ 0: mean should be added to forecast to correct for the bias | | | | | |
| Desired Properties of Residuals | | | | 1. Residuals have constant variance. 2. Residuals are normally distributed  These 2 properties make it easier to compute prediction intervals (using Normal Approx).  But sometimes, it is impossible to fix them. 1 soln is to transform the data | | | | | |
| E.g. | *gg\_tsresiduals(mean\_mtd) # plot TS of residuals + ACF + histogram*  If WN: Residuals time plot no autocorrelation. ACF all btw blue line. Histogram looks like Gaussian curve. Ljung Box test have p-value > 0.05. | | | | | | | | |
| *ts <- readRDS(""); l1 <- guererro(ts$y); snaive <- model(ts, SNAIVE(box\_cox(value, l1)))*  *fcast <- forecast(snaive, h = 12, point\_forecast = list(.median=median, .mean=mean)) #OR .median=median(y)*  *autoplot(fcast, data=filter(ts, index >= yearmonth("1990 Jan")), level=NULL, point\_forecast=list(mean=mean), show\_gap = FALSE) +*  *labs("Bias-adjusted forecast of mean (in red)") +*  *autolayer(fcast, level=NULL, point\_forecast = list(median=median), color="red")* | | | | | | | | |
| Tests of Autocorrelation | | | Portmanteau tests: test whether the first h autocorrelations (taken tgt) are significantly diff from what is expected from a WN process. 1) Box-Pierce test. 2) Ljung-Box test | | | | | | |
| 1) Box-Pierce Test: Q = , where rk = autocorrelation, h is maximum lag being considered, T is num of obs  If each rk is close to 0, then Q will be small. If some rk are large, then Q is large, then conclude that residuals are autocorrelated  Rule-of thumb: Use h = 10 for non-seasonal data. Use h = 2m for seasonal data, where m = period of seasonality.  However, if the h > T/5, then use T/5 instead | | | | | | |
| 2) Ljung-Box test: Q\* = . Large value of Q\* suggests residuals are not from a WN series. | | | | | | |
| For both Box-Pierce and Ljung-Box: H0: , i.e. residuals are uncorrelated  Under H0: both Q and Q\* ~ dist w (h - K) degrees of freedom, where K = num of parameters in the model  If test is computed based on raw data, then set K = 0  *augment(mean\_and\_naive) %>% features(.innov, features=feature\_set(tags="portmanteau"), lag=10) #Apply both tests to model* | | | | | | |
| Evaluating Forecast Accuracy | | Training data = to estimate the parameters of a model. Test data = evaluate its forecast accuracy.  Test set usually 20% of total sample. Test set should ideally be as large as maximum forecast horizon | | | | | | | |
| Forecast error = diff btw observed and its forecast =  Training data = {y1, …, yT}. Test data = {yT+1, yT+2, …}. Note residuals based on training data, forecast error based on test data | | | | | | | |
| 1) Scale-Dependent Errors: computed values, eT+k are on the same scale as the data  - Any accuracy measure that is based only on eT+k (instead of a standardisation of it).  - Cannot be used to make comparisons btw series of diff scale | | | | | MAE (Mean abs error) = .  RMSE (root mean squared error) = | | |
| 2) Scale-Independent error: To compare forecast errors across diff series w diff scale | | | | | | | |
| 2a) Percentage error =  However, if yT+k is close to 0, then will have extreme values, ∞  also tend to penalise -ve errors more than positive.  If {yT} is a non-negative series, then for any obs, there is a max positive pT but negative pT is unbounded | | | | | | Mean abs percentage error, MAPE =  symmetric MAPE,  sMAPE = | |
| 2b) Scaled error: Errors are scaled using the training MAE from the naive mtd. (training MAE is acting as baseline)  Scaled error < 1 if it is better than naive mtd forecast. Scaled error > 1 if worse than naive mtd forecast - scaled error = qj = . For seasonal data, qj =  Both numerator and denominator are on the same scale, hence qj is scale-indep | | | | | | | Mean abs scaled error,  MASE = |
| *train <- filter\_index(ts, . ~ "Dec 2004"); test <- filter\_index(ts, "Jan 2005" ~ .)*  *models <- model(train, avg=MEAN(y), naive=NAIVE(y))*  *fc <- forecast(models, h=4); accuracy(fc, ts)* | | | | | | | |
| A line with blue dots  Description automatically generatedCross validation | | | | A close-up of a grid  Description automatically generatedMulti-step cross validation | | | |
| *stretched <- stretch\_tsibble(ts, .init = 20, .step=1) # generate multiple version of ts, each truncated at the next obs*  *stretched %>% model(mean = MEAN(Quotes), naive=NAIVE(Quotes)) %>% forecast(h=1) %>%*  *filter(Month <= yearmonth("2005 Apr")) %>% accuracy(insurance)* | | | | | | | |
| Prediction Intervals (PI) | | For 95% prediction interval of the next obs = , where is an estimate of the SD of the forecast  When forecasting one step ahead, the SD of the forecast dist ≈ SD of the residuals. , i.e.  When there are no parameters estimated, the two SD are identical.  When parameters are estimated, then the SD of the forecast distribution is slightly larger than the residual SD.  As the forecast horizon increases, the prediction intervals generally increase in width. (further ahead, more uncertainty)   |  |  |  |  | | --- | --- | --- | --- | | Mean | Naive | Seasonal naive | Random Walk w Drift | |  |  |  |  | | | | | | | | |
| 1) Mean model assumes E(yt) = (which is a parameter), et ~ WN(0,). According to model, actual point in future should be  , where . is our estimate of  But since we estimate , our prediction has var = var() = var() + var() (indep) =  2) Naive model assumes . Note is fixed constant here, not a parameter  Variance of prediction = var() = var() = var() = h | | | | | | | |
| If a transformation has been used, then the prediction interval should be computed on the transformed scale.  End-points of PI should be back-transformed to give a PI on the original scale.  The new intervals will have the same coverage, but they will no longer be symmetric. | | | | | | | |
| *autoplot(fcast, data=filter(ts, index >= yearmonth("1990 Jan")), level=c(80, 95), #point\_forecast=list(mean=mean), show\_gap = FALSE)*  *# level for 80 & 95% PI # mean=bias adjusted, median = w/o bias adjustment* | | | | | | | |
| ## Simulating from the residuals. (if residuals not normal) In contrast to calculating , the estimate of SD of forecast  *fc2 <- eggs\_mdl %>% forecast(h = 50, bootstrap = TRUE) %>% mutate(.median = median(eggs))* | | | | | | | |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Decomposition | | Time series can have diff patterns: trends, cycles and seasonal effects  Can decompose TS into 3 components: trend-cycle component, seasonal component, remainder component (unable to explain) | | | | | | |
| Additive Decomposition | | yt = St + Tt + Rt, where St = seasonal component, Tt = trend-cycle component, Rt = remainder component  Model is appropriate if seasonal variation does not change w level of TS | | | | | | |
| Multiplicative Decomposition | | yt = St \* Tt \* Rt  Model is appropriate if variation around the trend-cycle appears to be proportional to level of TS  Can also log yt = log St + log Tt + log Rt to transform multiplicative model to additive model | | | | | | |
| E.g. To interpret additive decomposition | | *dcmp <- TS %>% model(stl = STL(y))*  *components(dcmp)*  *components(dcmp) %>% as\_tsibble() %>% autoplot(y, colour="gray") + geom\_line(aes(y=trend), colour="red")* | | | In table, value = yt, trend = , season\_year = , remainder = , season\_adjust = yt =  See season\_adjusted line overlaid on TS | | | |
| A graph of different types of graphs  Description automatically generated with medium confidence*components(dcmp) %>% autoplot()*  From top to bottom: yt, , ,  Grey bar on left are of same value across all plots.  i.e. Trend and seasonality explains a larger proportion of the series, and remainder explaining less.  Note remainder is largest at around spikes/dips in the trend  If seasonal variaion is not of primary interest, we should focus on the seasonally adjusted series, i.e.  For additive models: yt . For multiplicative model: yt /  Since seasonally adjusted data still contain remainder component, it will not be smooth  *components(dcmp) %>% as\_tsibble() %>% autoplot(y, colour="gray") + geom\_line(aes(y=season\_adjust), colour="blue")* | | | | | | |
| Decomposition Features | | Strength of trend = Ft = . Small value of Ft (close to 0) indicates Var(Rt) ≈ Var(Tt + Rt), i.e. trend is not the "driving force" compared to residual noise. If Ft close to 1, then trend is very impt compared to residual noise  Strength of seasonality = FS = | | | | | | |
| *features(TS, y, features=feature\_set(tags="seasonal") # trend\_strength* = Ft, *seasonal\_strength\_year* = FS. Also have other values below | | | | | | |
| seasonal\_peak\_year  seasonal\_trough\_year  spikiness  linearity  curvature | | timing of peaks within a season  timing of throughs within a season  prevalence of spikes in Rt of the STL decomposition = var of leave-one-out variances of Rt  linearity of Tt of the STL decomposition. Based on coefficient of a Linear Regression applied to Tt  curvature of Tt of the STL decomp. Based on coeff from an orthogonal quadratic regression applied to Tt | | | | |
| Moving Average Filters | 1st step in TS decomposition is to estimate trend-cycle. Use Moving Average to smooth the input process  MA of order m is a special case of a linear filter: , where m = 2k+1 is an odd num. This operator aka m-MA  m-MA is a simple mtd to estimate trend-cycle component. . It averages nearby values to return smoothed version of TS  m = known seasonal period, so as to remove seasonal variation. This happens if sum of seasonal components for each period = 0  m-MA aka finite MA filter / finite order linear filter / time invariant linear filter / low-pass filter | | | | | | | |
| E.g. computing MA | | *ts2 <- mutate(ts, `5-MA` = slide\_dbl(y, mean, .before=2, .after=2, .complete=TRUE))*  Smoother series capture the main movement of the TS w/o all the minor fluctuations. Larger m = smoother curve | | | | | | |
| Computations when m is Even | | | If m is even, convention is to take one obs more from the future than the past. E.g. m = 2k. Then filter =  *ts <- mutate(ts, `4-MA` = slide\_dbl(y, mean, .before=1, .after=2, .complete=TRUE))*  When m = 2k+1 is odd, m-MA operation is symmetric: k earlier & later obs & middle obs. Each obs multiplied by 1/, (symmetric in weights assigned to obs before and after middle one)  When m = 2k is even, not symmetric. Repeated application of the MA filter can yield a filter w the symmetry properties we had. Applying a 2 m - MA means 1) apply m-MA to raw data w extra obs from future. 2) Apply 2-MA to the new col w an extra obs from the past, i.e. 4-MA: . 2 4-MA:  Now, num of obs on both sides of t are same. Weights assigned to points same dist from t but on opp sides are the same  *ts <- mutate(ts, `2x4-MA` = slide\_dbl(`4-MA`, mean, .before=1, .after=0, .complete=TRUE))*  2 m - MA aka centred moving average of order m. Note, odd order MA don't have to be centered – already symmetric  MA thus help to estimate trend cycle from seasonal data, by removing the seasonal variation | | | | | |
| Removing Seasonal Variation using a Filter | | Intuitively, seasonal variation is that periodic component which is centred around the trend-cycle component.  Hence, reasonable to assume seasonal components sum to 0  E.g. Quarterly data implies St = St+4 for all t, and for all t. The appropriate filter to use is the 2 4-MA | | | | | | |
| Guidelines | | Use 2 12-MA for monthly date and 7-MA for daily data | | | | | | |
| Weighted Moving Averages | | Weighted m-MA: , where k = (m-1)/2  Simple m-MA is the special case where aj = 1/m for all j. 2 4-MA is another special case, w weights 1/8, 1/4, 1/4, 1/4, 1/8  Weighted MA yield smoother estimates than simple MA since weights are slowly and , instead of abruptly including/excluding them  By choosing appropriate weights, can design filter to remove higher-order terms that are likely noise  To design filter: start w window size m, fit a polynomial trend within that window by minimising least squares error. Replace obs in the middle of that window w point on fitted polynomial  E.g. want to find weights for a-2, a-1, a0, a1, a2 (i.e. m = 5) by fitting quadratic trend. WLOG, can consider points at t = -2, -1, 0, 1, 2  Fn to minimize = h(b0, b1, b2) = . Only need to estimat b0  Taking partial derivatives (w.r.t b0, b1, b2) one at a time and setting to 0, we get ∑ yt = 5b0 + 10b2. ∑ tyt = 10b1. ∑ t2yt = 10b0 + 34b2  Solving for b0: b0 = . Weighted 5-MA are  Weighted 15-MA: [a0, a1, …, a7] = . Allow cubic trend to pass through | | | | | | |
| E.g. Weighted MA | | Creating TS w quadratic trend. *x <- 1:10; y <- ts(x^2) %>% as\_tsibble()*  To apply weighted MA: *stats::filter(y$value, c(-6/70, 24/70, 17/35, 24/70, -6/70)*  Series is unmodified by filter, i.e. filter allows a signal to pass through undistorted | | | | | | |
| Decomposition algos | | Classical Additive Decomp. Assume seasonal effect is same for all t  1) If m is even, use a 2 m-MA (if m is odd, use simple m-MA) to compute  2) Calculate the de-trended series,  3) Estimate the seasonal component: a) Average the de-trended values for each month. E.g., average all de-trended March values to obtain the estimate of the effect of the March season.  b) Adjust the seasonal component so that they sum to 0 to get  c) This last step ensures that there is no confounding of the seasonal effects with the level of the time series, and allows us to view the seasonal effects as deviations from the trend-cycle.  4) Calculate the remainder component using  Suppose additive model is appropriate for our series,  Whatever we estimate for the trend-cycle and seasonal components, we could always add/subtract an arbitrary value to each of them, i.e.  To avoid this non-identifiability, constraint our seasonal effects to sum to 0. Also allows us to interpret the average seasonal effect as 0.  Suppose we have our initial estimates of the seasonal effects,  Then can adjust these by setting for t = 1, …, m | | | | | | |
| Classical Multiplicative Decomp. Assume seasonal effect is same for all t  1) If m is even, use a 2 m-MA (if m is odd, use simple m-MA) to compute  2) Calculate the de-trended series,  3) Estimate the seasonal component: a) Average the de-trended values for each month.  b) Adjust the seasonal component so that they sum to m to get  c) This ensures that the average of the seasonal effects = 1; each is then a multiplicative deviation from the trend-cycle  4) Calculate the remainder component using  To constraint our seasonal effects to sum to m. Suppose we have our initial estimates of the seasonal effects,  Then can adjust these by setting for t = 1, …, m | | | | | | |
| E.g. *ts\_dc <- model(ts, class\_add = classical\_decomposition(y, "additive")) # OR multiplicative*  *components(ts\_dc) %>% autoplot()* | | | | | | |
| Cons of classical: Since we are using MA filters, we are unable to estimate the trend-cycle for the beginning and end of the series. The classical approach assumes that the seasonal variation is the same over time. It is also not robust to outliers. | | | | | | |
| X11 Decomp: based on classical decomp, but has some improvements:  - By using one-sided linear filters, it obtains trend-cycle estimates for all time points.  - Allows seasonal effect to vary over time.  - Includes use of a regression model for the remainder component  - Annual holidays are included in the seasonal components.  - The process iterates the algo to achieve smoother estimates.  X11-ARIMA, X12-ARIMA and X13-ARIMA are all improvements on X11 *OR y ~ regression(variables='td', aictest=null) #td = trading day variation* | | | | *ts\_dc <- model(ts,*  *class\_add = classical\_decomposition(y, "additive"),*  *class\_mult = classical\_decomposition(y, "multiplicative"),*  *x11 = X\_13ARIMA\_SEATS(y ~ x11()) )*  *select(ts\_dc, class\_mult) %>%*  *components() %>% autoplot()*  *select(ts\_dc, x11) %>%*  *components() %>% autoplot()* | | |
| Seasonal and Trend decomposition using LOESS  (STL) | | “Loess” refers to a locally weighted regression model. This is used in place of a moving average filter for estimating the trend-cycle.  In comparison to the ordinary linear regression model, loess is able to estimate nonlinear relationships. | | | | | | |
| Advantages of STL  - Unlike X11, STL can handle any type of seasonality (fixed pattern)  - Like X11, it allows seasonal component to change over time (same pattern, but value of pattern changes)  - Smoothness of the trend cycle can be controlled by the user.  - Can be made robust to outliers, so that occasional unusual observations will not affect estimates of the trend cycle. | | | Disadvantages of STL  - Can only handle additive models.  - This shortcoming can be somewhat overcome by transforming model first.  - There are several parameters for this approach.  - There are defaults for several of them except one. | | | |
| Loess Fitting. Suppose and are measurements of an indep and dependent variables.  Loess regression curve, is a smoothing of given that can be computed for any values of . To compute :  1) Choose a value , that will serve as the span.  2) The values of that are closest to will be given a weight, based on how far they are from , typically through a kernel function.  3) values that are closer to will receive a larger weight.  4) Perform a weighted least squares regression using the above weights.  *geom\_smooth(span=%ofpoints, method="loess", method.args=list(degree=0))* | | | | | | |
| STL algo consists of two loops:  - Outer loop for robustness to outliers in the time series. (If sure no outliers, no need outer loop; usually 5-10 iterations)  - Inner loop to estimate trend and seasonal components. Recall that seasonal component can vary over time. (usually 1-2 iterations)  - The loess algo is repeatedly used as the smoother, except in one portion of the procedure.  In the outer loop: {  1) The remainder component is estimated.  2) They are assigned a robustness weight (points w larger "residuals" given lower weight), which is used in the inner loop. }  In inner loop, w a given set of robustness weights and an initial trend-cycle estimate: {  1) Series is detrended using  2) Individual subseries are smoothed using loess (*s.window*) to get . E.g. if data has monthly freq, 12 smoothings are carried out  3) Combine smoothed subseries into 1, apply MA filter twice. Estimate loess (*l.window*) smoothing for this combined series, .  (to identify and extracts any residual trend)  4) , to yield an estimate of the seasonal component.  5) New series, is smoothed by a loess smoother (*t.window*) to obtain a new trend-cycle estimate, . } | | | | | | |
| *model(ts, stl1 = STL(Y ~ trend(window=, degree=1) + season(window=) + lowpass())*  *dc\_stl <- model(ts,*  *stl1 = STL(value ~ trend(window=13), robust=TRUE),*  *stl2 = STL(value ~ trend(window=10) + season(window="periodic"), robust=TRUE))*  *select(dcmp\_stl, stl2) %>%*  *components() %>% autoplot()* | | | | | | *t.window, s.window, l.window*  Usually, just use default for trend and lowpass  *t.window*: trend cycle window  *s.window*: seasonal window  *s.window* = “periodic” : assume seasonal component don't change.  *robust* : model not influenced by outliers much |
| *select(dcmp\_stl, stl2) %>% components() %>% as\_tsibble() %>% mutate(raw\_mth =*  *Month(index, label=TRUE) + ggplot(aes(x=index)) + geom\_line(aes(y=season\_year))*  *+ geom\_point(aes(y=Y-trend)) + facet\_wrap(~raw\_mth, nrow=4)* | | | | | | For all subseries, check if *s.window* needs to be change or not. If line underfit, decr window. Overfit = incr window |
| Forecasting w decomp | | *fit\_dcmp <- model(ts, dcmp\_fc = decomposition\_model(*  *STL(value ~ trend(window=13) + season(window="periodic"), robust=TRUE),*  *NAIVE(season\_adjust), SNAIVE(season\_year)))*  *forecast(fit\_dcmp, h=12) %>% autoplot(elecequip)* | | | | | In *decomposition\_model*, specify:  - decomposition method,  - forecasting mtd for the seasonally adjusted series,  - forecasting mtd for the season effect. | |
| Tut | | Suppose TS is , where f is a smooth and cts fn of t, and et ~ WN(0, ).  Can use m-MA to estimate f(t), w m = 2k + 1: , t = k+1, …, n-k. i.e. Larger m = smaller var, higher bias  Taylor expansion of f about t:  Then, | | | | | | |
|  | | Symmetric MA {aj}, j = -q, …, 0, …, q passes an arbitrary polynomial of deg k w/o distortion,  i.e. mt = kth deg polynomial mt = c0 + c1t+ … + cktk iff and for r = 1, …, k  Proof: mt = , and mt+j = . Note since MA is symmetric, aj = a-j  RHS = . Now need to show  So just need to show . Using binomial expansion,    (using 2 conditions above) | | | | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Methods vs Models | | | | Forecasting mtd = algo that provides a point pred in future. Statistical model = process that generates data - probability dist for future  - Point forecast can then be obtained by taking mean/median of that probability dist  Note = forecast of yt+h given y1, …, yt (in-sample). = forecast of yT+h, given y1, …, yT (out-of-sample)  Mtd e.g. for all h (Mean mtd used to generate forecasts w/o any further assumptions)  Model e.g. , where et ~ GWN(0, ). Model implies ~ N(, ) for all h. Once we estimate and , we can use the mean of the estimated dist to forecast yt+h: . Can also forecast that w Prob 0.95,  Model allows us to compute prediction intervals (PI). But requires making distributional assumptions  A model fully specifies the data generating process, whereas a forecast mtd does not. |
| State space models | | | | Let xt be a "state vector" containing unmeasured components that describe the level, trend and seasonality of the series  Then a linear innovations state space can be written as – (measurement eqn). And – (transition/state eqn), where et is a WN process, g and w are vectors and F is a matrix  Coefficient matrices and vectors could contain parameters that need to be estimated, but don't involve state xt-1  This is known as the Innovations formula = assume identical errors in both eqns  Alternative: assume there is 1 error for the measurement eqn, et and an independent error, zt for the transition eqn  Instead of F, g and w being matrices, can also generalised s.t. and , where functions w and r take in vector and return scalar, while fns f and g returns a vector |
| Exponential Smoothing (ETS) | | | | In exponential smoothing, view trend as a combination of a local level, l and a local growth term b.  Let T­h = forecast of trend h time periods ahead, and = damping parameter (0 < < 1)   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | Trend type | None | Additive | Additive damped | Multiplicative | Multiplicative damped | | Formula |  |  |  |  |  |   After choosing trend (diag in lect 6 notes), need include seasonal component and error term, each either additively or multiplicatively.  For exponential smoothing mtds, type of error don't matter – point forecasts will be the same. Errors only affect PI of ETS models  Ignoring the error components, there are 15 basic exponential smoothing mtds/ ETS model (Error, Trend, Seasonal) to consider   |  |  |  |  | | --- | --- | --- | --- | | Trend | Seasonal None | Seasonal Add | Seasonal Mult | | N (None) | N,N (SES) | N,A | N,M | | A (Additive) | A,N (Holt's linear mtd) | A,A (Additive Holt-Winters mtd) | A,M (Mult Holt-Winters mtd) | | Ad (Additive damped) | Ad,N (Additive damped trend mtd) | Ad,A | Ad,M | | M (Mult) | M,N (Exp trend mtd) | M,A | M,M | | Md (Mult damped) | Md,N (Mult Holt-Winters mtd) | Md,A | Md,M |   For each model, there are 2 possible state space model. If same params are used, both models give same point forecasts, but diff PI |
| SES (N,N) – Simple exponential smoothing | | | | Definition 1) = adjusting forecast for next period using forecast error from previous period, where  Defn 2) = weighted avg of most recent obs and most recent forecast  . Thus the forecast is a weighted avg of all past obs  For this defn, we need to know values of and . Size of determines impact of on forecast  Deciding on value of to use = initialisation problem  Defn 3) . . Via a) evolution of components (only consider level of series, ) and b) forecast fn.  a) can be considered as the transition eqn and b) as the measurement eqn  If we want to forecast for a longer forecast horizon, SES model returns a fixed value.  Thus forecasts eqn should have been . Only suitable for series w no trend or seasonal component |
| For defn 2: can show that sum of weights = 1.  So . Note |
| Holt's Linear Mtd (A,N) | | | | Assume that at each point, there is a linear trend bt and level lt from which that trend starts. Need to estimate  2 smoothing eqns for each component. and . Forecast eqn: (linear fn of h)  is an estimate of level at time t = weighted avg of yt and the one-step-ahead forecast ()  is an estimate of trend at time t = weighted avg of and the current tred, estimated by  = smoothing param for the level, while = smoothing param for the trend. Both and  Holt's linear mtd assume data follow a constant trend indefinitely into future. Empirical evidence shot mtd tend to over-forecast |
| Additive Damped Trend (Ad,N) | | | | Damped trend mtd introduces a param that weakens/softens the trend to a flat line some time in the future  Effect of trend is damped each time it enters the forecast and level fns. And  Since , . This means in short term, forecasts have a trend but in long run, they are constant  When = 1, model = Holt's linear mtd. rarely set to be < 0.8, since it has a strong effect. Usually |
| Holt-Winters Seasonal Mtds | | | | Holt-Winters seasonal models contains 3 components: trend bt, level lt, and the seasonal component st.  Corresponding smoothing params = . Freq of seasonality = m (for quartely data, m = 4)  If seasonal variations roughly constant in series: Use Additive mtd. In this model, seasonal component add up to ≈ 0 within each year  If seasonal variations prop to level of series: Use Mult mtd. In this model, seasonal components add up to ≈ m within each year |
| Additive seasonality (A,A). , where  Level eqn = weighted avg btw seasonally adjusted obs and the non-seasonal forecast  Seasonal eqn = weighted avg btw current seasonal index and seasonal index of the same season in the previous year  Equivalent formulation for seasonal component is , where  "Proof": Using level eqn, |
| Mult Seasonality (A,M). , where |
| Summary | | Note h-1 = km + , where k . So (i.e. h - km = ). Let   |  |  |  |  | | --- | --- | --- | --- | | Season  Trend | N | A | M | | N |  |  |  | | A |  |  |  | | Ad |  |  |  | | | |
| State Space Models | For each mtd above, there are 2 state space models - 1 w additive errors, 1 w multiplicative errors  For state space model, have to identify 1) state vector and 2) source of error that appears in both state and measurement eqn | | | |
|  | | | |
| ETS(A,A,N). Want it in the form of and . And  We know . And for (A,A,N), (for 1-step ahead forecast)    , where  So . . Since  Suppose , then = forecast from Holt's linear mtd  For most models, . But will not hold for models w multiplicative trend or multiplicative seasonality for h ≥ 2 | | | |
| ETS(M,A,N). and and .  However, now relative error = . So And  . So  , where  So .  Suppose , then = forecast from Holt's linear mtd | | | |
| In general, and . Full model list in lect 6  For additive models, , . For multiplicative models,  The models with multiplicative error/trend/seasonality could involve division by 0, and so are numerically unstable.  The multiplicative error models are unstable when the data values contain zeros or negative values.  If the data are not strictly positive, use only the 6 fully additive models. | | | |
| Computations | | | | With the state space models, and given x0, y1. We can compute |
| Residuals & Forecast errors | | | | For *forecast* package, 1-step forecasts defined as . The residuals are the estimates of the innovation (forecast) errors.  For the state space models with additive errors, residuals = one-step forecast (innovation) errors.  For models with multiplicative errors, residuals = . Forecast/innovations = |
| Linear Innovations State Space models | | | | – (measurement eqn), describes effect of past on yt. And – (transition eqn)  yt denotes observed values, xt is state vector containing info on level, growth and seasonal patterns.  Error term, et ~ GWN(0, ) and is the only source of noise in model; also known as the innovation in the model  F is transition matrix. describes effect of past on current state xt. Vector g determines extent of effect of innovation on state  Vectors w, g and matrix F is fixed over time (in this course). And are parameters we need to estimate  Given the initial state vector, the pdf for **y** =  Since assume et are Gaussian, |
| ETS(A,N,N). . . Where  When , local levels dont change at all:  When , model is a random walk model:  Conditional expectation for 1-step forecast is  Conditional var  If 0 < < 1, then forecast can be interpreted as a weighted avg of previous values, w older values being assigned less weight.  For this model, stability condition is satisfied if 0 < < 2  Note that |
| ETS(A,A,N). . . Where  When , and , then model is stable  However, in practice, restrictions 0 < < 1 and 0 < < are usually applied. Corresponding exponential smoothing mtd is (A,N). |
| ETS(A,A,A). . . .    Seasonal components are normalised to prevent confounding w the level.  The usual parameter space are 0 < < 1 and 0 < < and 0 < < 1 - |
| Nonlinear innovations State Space Models | | | | and , where functions w and r take in vector and return scalar, while fns f and g returns a vector, et is a WN process.  Joint dist of variables:  Assume et follow Gaussian dist, |
| ETS(M,N,N). . . So  Where state vector . .  (same as ETS(A,N,N))  But conditional var (diff from ETS(A,N,N))  So forecast var will depend on level of process (leading to diff PI)  When , state does not change; i.e. identical to additive model except for a parametrisation  When , model is |
| ETS(M,A,N). . .  Where  Special cases: global trend. random walk w drift. fixed level and trend |
| Estimation in State Space models | | | | Initial state x­0, and params are unknown and have to be estimated from data  - Smoothing params, e.g. and for ETS(A,A,N) model. Refer to these as a vector  - Initial state . - Innovations var |
| MLE: Likelihood fn for generatl state space model is  Assuming Gaussian innovations, likelihood can be written as  Log-likelihood,  To maximise log-likelihood: to get  Then log-likelihood becomes  Equivalent to min aka augmented sum of squares criterion (power to 2/n L(), ignore constant?)  Instead of using the likelihood function, we could target to find the parameters  by minimising the one-step MSE, MAE or some other error metric. Yet another method is to minimise the residual variance. |
| Num of parameters. Suppose we have weekly data, i.e. m = 52. And we wish to fit an ETS(A,A,A) model, we would then have to estimate 52 + 2 = 54 seed states () and 3 smoothing parameters ()  Huge num = hard to compute. Solution: use heuristic methods of estimation OR assume certain weeks have same effects |
| Initial values of x0. 1) For initial seasonal component, perform a classical decomposition of the process  2) For the initial level component, perform a linear regression of the first yt values on 1,2,…,10 and use intercept term as  3) For initial growh component, use slop estimated from 2) as if it is additive trend. If multiplicative trend, use , where b is slope from 2) and a is intercept |
| Prediction Intervals | | | When forecasting TS, sources of uncertainty are 1) model choice, 2) future innovations et+1, et+2, …, 3) Parameters estimates  In practice, we only consider 2) the uncertainty in future innovations  Prediction dist = dist of future values, given the model, its estimated parameters and xt. So  Forecast mean is . Forecast variance is . | |
| Analytical expressions for the forecast var are only available for some of the models  Class 1: easy to derive  Class 2 and 3: can derive, but involve making a few further assumptions  Class 4 and 5: cannot derive, use simulation to obtain prediction intervals | |
| ETS(A,N,N). . . Where  One-step conditional mean is  Prediction error, or var of this forecast =  When h = 2,  Prediction error, or var of this forecast = | |
| In general for class 1 models, .  E.g. for ETS(A,N,A),  Then  Forecast var = . Which can be simplified to ()   |  |  |  | | --- | --- | --- | | Model | Forecast var |  | | (A,N,N) |  |  | | (A,A,N) |  | . | | (A,Ad,N) |  | | | (A,N,A) |  | | | (A,A,A) |  | | | (A,A­d,A) |  | | | |
| Intervals via Simulation. Suppose required forecast horizon is h w model conditional on most recent state xT. Then for i = 1,…,M,  1) Generate obs , starting w xT, from the fitted model  2a) Each eT+k valus is obtained from a random num generator assuming a Gaussian or other appropriate dist OR  2b) Bootstrap to resample from historical values of et if unsure about innovations dist (bootstrap also appropriate when et are Gaussian but yt are not, due to it being a non-linear model)  Usually take M = 5000. Then take mean of simulated values at each h as the point forecast. E.g. h = 1, take mean of {}  Can use quantiles to obtain PI. E.g. for 95% PI for h = 3, take 0.025 and .975 quantiles of {}  *mtd4 <- model(TS, add = ETS(Y ~ error("A") + trend(method="A") + season("A")))*  *mtd4\_fc <- forecast(mtd4, h=16); autoplot(mtd4\_fc, data=TS, level=95) + labs(title = "Prediction intervals")* | |
| Model Selection | | | | 1. Split the time series into a training and test set 2. Fit each model using the training set (via MLE). 3. Assess the forecast accuracy of each model using the test set. 4. Choose the model with the lowest forecast accuracy. 5. Refit the chosen model to the full time series and use these new parameters for forecasting future observations. 6. An alternative was to use cross-validation to select the best model. |
| Sometimes, test set is too small to draw reliable conclusions or diff to decide which error metric to use  Instead, can use penalized likelihood method. This fits model to entire data, and compute likelihood for that data  Model w highest likelihood is chosen. However, likelihood is penalised for num of params used.  , where q = num of params in + num of free states in OR |
| *mtd5 <- model(TS, ets1 = ETS(Y))* # will fit best ETS model. *report(mtd5); gg\_tsresiduals(mtd5)* |
| Tut 7 | | | | Theta Decomposition |
|  | | | | *stl\_seas\_adj <- model(ts, stl\_robust2 = STL(Y ~ season(window=5), robust=TRUE)) %>% components() %>% as\_tsibble()*  *ses1 <- select(stl\_seas\_adj, season\_adjust) %>% model(ses\_model1 = ETS(season\_adjust ~ error("A") + trend("N") + season("N")))*  *report(ses1)* # For SES model, can see and w optimal (RMSE)  *ses2 <- select(stl\_seas\_adj, season\_adjust) %>% model(opt\_alpha = ETS(season\_adjust ~ error("A") + trend("N") + season("N")),*  *fixed\_alpha = ETS(season\_adjust ~ error("A") + trend("N", alpha=0.1) + season("N")))*  *autoplot(augment(ses2), .vars= season\_adjust, col="gray") + geom\_line(aes(y=.fitted, col=.model))*  *accuracy(ses2)* |

|  |  |  |
| --- | --- | --- |
| Stationarity | A strictly stationary TS is one for which the joint dist of {} is identical to the dist of {} for all k, all time points t1, t2, …, tk and all h = 0, ±1, ±2, …  Strict stationarity implies E(ys) = E(yt) = for all s,t. And ACVF = = cov(ys, yt) = for all s,t and h | |
| A weakly stationary TS is a finite variance process s.t. is a constant (don't depend on t) And depends on s and t only through s - t  For this course, stationary = weakly stationary. Let denote mean and h = s - t and ACVF = | |
| E.g. 3-MA filter on WN process et ~ WN(0, ).  ACVF = . Since E(yt) = 0 = constant, and ACVF depends on s-t, yt is stationary  E.g. 2: yt = + et. E(yt) = which is not indep of t, so yt is not stationary  E.g. 3: yt = yt-1 + et. y0 = 0. Cov(y1, y2) = cov(y0 + e1, y1 + e2) = cov(y0 + e1, y0 + e1 + e2) =  But cov(y2, y3) = cov(y0 + e1 + e2, y0 + e1 + e2 + e3) = 2. So even though s - t is the same, ACVF is diff, so yt is not stationary | |
| Properties of stationary processes: 1) = var(yt). 2) (can be proved using Cauchy-Schwarz inequality). 3)  If {yt} is a stationary TS, then for all s, the joint dist of (yt, …, yt+s) don't depend on t  A stationary series is - roughly horizontal, - has constant var, - has no predictable patterns in long-term,  - a series w trend and/or seasonality is not stationary due to its changing mean | |
| For a stationary series: - ACF drops to 0 relatively quickly  For non stationary data: - ACF decreases slowly, value of r is often large and positive for many lags  To have stationary TS, can use transformations to stabilize var OR to difference the data | |
| Differencing | A single differencing stabilizes the mean of a TS by removing changes in the level of a TS.  The differenced series will have only T-1 values.  *mutate(df, diff1 = difference(Y)) %>% gg\_tsdisplay(diff1, plot\_type = 'histogram')*  *mutate(df, diff1 = difference(Y)) %>% features(diff1, feature\_set(tags='portmanteau')* | |
| Occasionally, differenced data will not appear stationary, and may be necessary to difference data a second time.  . In practice, shouldn't be neccesary to go beyond 2nd order difference | |
| Seasonal differencing. , where m = num of seasons. For monthly data, m = 12  The new series = "lag-m differences". If seasonal differenced data appears to be WN, then an appropriate model would be yt = yt-m + et = seasonal naive | |
| Twice differenced series. E.g do both seasonal and a first difference    When both seasonal and first diff are applied, doesn't matter which is done first  However, if there is a strong seasonal pattern, seasonal differencing should be done first. This is because, sometimes, the seasonally differenced series alone is close enough to stationarity - there might be no need to do the first differencing as well. If we performed the first differencing first, the strong seasonality would compel us to perform the second (seasonal) differencing too.  First differences are the change from one obs and the next. Seasonal differences are the change between one year and the next.  Higher order differencing should be avoided as they are difficult to intrepret. | |
| Unit Root Tests. The Kwaitkowski-Phillips-Schmidt-Shin (KPSS) test can be used to test if a series is stationary  H0: data is stationary and non-seasonal. H1: data is not stationary  KPSS test can be repeatedly applied to successive differencing to determine num of differencings that should be carried out.  *features(TS, Y, list(unitroot\_ndiffs)).* # Use *unitroot\_nsdiffs* to determine optimal num of seasonal differencing | |
| Backshift Notation. Byt = yt-1. i.e. shift data back one period. B(Byt) = B2yt = yt-2. Note Bc = c (where c is a constant)  For monthly data, to denote same month last year, B12yt = yt-12  First difference: . Second order diff:  In general, a dth order diff can be written as (1 - B)dyt  Seasonal diff followed by a first diff: (1 - B)(1 - Bm)yt = (1 - B - Bm + Bm+1)yt = yt - yt-1 - yt-m + yt-m-1 | |
| Auto-regressive Models (AR) | AR(p) = , where et is GWN. are constants, w  When E(yt) = 0,  When E(yt) = ≠ 0, then  So  This is a multiple linear regression w lagged values of yt as predictors  Diff values of parameters results in diff TS patterns. Var of error term et will only change scale of series, not the patterns  The defn don't guarantee process is stationary. Need to impose conditions on so that process is staionary | |
| Consider AR(1) model,  If , yt = WN. If and c = 0, yt = random walk. If and c ≠ 0, yt = random walk w drift.  If , yt tend to oscillate btw +ve and -ve values | |
| Autoregressive Operator. .  . So , where is a polynomial in B = autoregressive operator  AR(p) is stationary if the (complex) roots of the polynomial falls outside the unit circle | |
| Thrm: A linear process yt is defined to be a linear WN et and is given by , where  For a linear process, the autocovariance fn | |
| For AR(1), , . So ,  From thrm,  . So comparing coeff of B, , ,  In general, . So  So  (using result if |x| < 1, then and replace x with )  OR  If || < 1, then Taylor expansion of RHS yields:  (using result Cov(X,Y) = E(XY) - E(X)E(Y), but both mean = 0 for this case)  So = and  Also | |
| Also,  So . | |
| Stationarity Conditions: For AR(1) model, require .  Need = to be outside unit circle. So to get . For |z| > 1,  For AR(2) model, AND AND . Use quadratic formula to find |roots| > 1 | |
| Causal AR process. Still possible to have a stationary AR(1) process where > 1  . So comparing to linear form, and  So since tend to 0. And is a linear process, hence stationary. But not casual as it is in terms of future et | |
| Moving Average Models (MA) | MA(q) = , where is WN. Each value of yt as weighted MA of past few forecast errors  Past forecast errors are used as predictors, although past errors et are not observed  Same as AR(p) models, changing values of parameters changes behaviour of TS. Var of et only changes scale of series  MA model here used for forecasting future values, while MA filter is to estimate trend-cycle of past values | |
| Moving Average Operator. Using backshift operator: , where = MA operator  Consider MA(1) model w mean 0: . .  Var() = . So .  Just as we inverted the AR(p) model earlier, we can bring the MA operator to the other side to yield (when || < 1):  = aka infinite AR representation | |
| Non uniqueness of MA(q) Models: Consider yt = 5et-1 + et and yt = 0.2et-1 + et. Both have same ACF  To prevent this, use MA(q) models w an infinite AR representation = invertible models  General condition for invertibility is that the complex roots of lie outside the unit circle on the complex plane  For MA(1) model:  For MA(2) model, AND AND | |
| Note MA model always stationary as its form is already the same as form for linear process | |
| ARIMA Models | ARMA models combine AR(p) and MA(q) models:  Suppose . Then . So  Predictors include both lagged values of yt and lagged errors. Have to impose conditions on the coeff to ensure stationarity & invertibility  Using backshift operator:  Model is stationary if roots of are outside unit circle. Model is invertible if roots of are outside unit circle | |
| ARIMA = AutoRegressive Integrated Moving Average. Combine ARMA models w differencing  , where is the differenced series  In backshift notation, if follows an ARMA model, then yt is an ARIMA process.  Specify ARIMA model by ARIMA(p,d,q). p = order of AR part. q = order of MA part. d = degree of first differencing involved  ARIMA(0,0,0) = WN. ARIMA(0,1,0) w c = 0 = RW. ARIMA(0,1,0) w c ≠ 0 = RW w drift. ARIMA(p,0,0) = AR(p). ARIMA(0,0,q) = MA(q).  Alternative form:  E.g. for ARIMA(1,1,1): . = AR(1) part. (1 - B) = first diff part. = MA(1) part | |
| ACF shows autocorrelations which measures r/s btw yt and yt-k for diff values of k  However, if yt and yt-1 are correlated, then yt-1 and yt-2 are also correlated. So yt and yt-2 could be correlated simply because both are correlated to yt-1. So how to measure what new info there is in yt-2, that could be used in forecasting yt  Consider AR(1) process: . From earlier: . In fact, covariances at any positive lag h will be positive (although smaller as h incr). ACF = . And  . And . And . And for AR(1)  For , find that minimises .  Taking derivative and setting to 0, can get . So .  Similarly for , find that minimises to get .  So, PACF for lag 2 =  OR  We have 'partialed out' the dependence on yt-1. Correlation btw yt and yt-2, after accounting for effect of yt-1 is 0 | |
| Let = regression of on . Let = regression of on  Find s.t. is minimized  PACF of is denoted for h = 1,2,…. where . And | |
| Partial Autocorrelations measures r/s btw yt and yt-k, when effects of other time lags {1,2,3,…,k-1} are removed  Partial autocorrelation at lag k = . Partial autocorrelation fn (PACF) = plot for all k  is computed as the estimate of in the AR model,  is effect of the , given that all other terms are already in the model  Note that . For confidence bands when plotting PACF, use same critical values of (same as ACF) | |
| MA(q) ACF. MA(q) process w finite q is always stationary, and has an ACF = , i.e. cuts off after lag q | |
| ACF and PACF behaviour  ARIMA(p,d,0) if differenced data show ACF exponentially decaying or sinusoidal AND PACF has sig spike at lag p but not beyond lag p  ARIMA(0,d,q) if differenced data show PACF exponentially decaying or sinusoidal AND ACF has sig spike at lag q but not beyond lag q   |  |  |  |  | | --- | --- | --- | --- | |  | AR(p) | MA(q) | ARMA(p,q) | | ACF | Tails off | Cuts off at lag q | Tails off | | PACF | Cuts off at lag p | Tails off | Tails off | | |
| E.g. *gg\_tsdisplay(TS, Y, 'partial')* # Plot time plot, ACF, PACF. If line above blue dashed line = lag is significant  If plots suggest ARIMA(3,0,0): *arima\_mods <- model(TS, arima300 = ARIMA(Y ~ pdq(3,0,0) + PDQ(0,0,0))); report(arima\_mods)*  Should also consider "nearby" candidate models. *arima\_mods <- model(TS, arima300 = ..., arima201 = ...);*  *fc1 <- forecast(arima\_mods, h = 8);*  *fcast\_table <- pivot\_wider(fc1, 'Quarter', names\_from = '.model', values\_from = '.mean') %>% mutate(across(c(2,3,), ~round(.x, digits=4)))*  *datatable(fcast\_table)* | |
| Estimation and Order Selection | General process for forecasting using an ARIMA model.For given values of p,d,q, R will find estimates for the parameters, by maximising log-likelihood  ARIMA() in R uses a unit root tests, AIC minimisation and MLE to obtain an ARIMA model.  Default is to search through models in a step-wise manner, i.e. some models might be skipped. Can override by setting *stepwise=FALSE*  1) Determine 0 ≤ d ≤ 2 using repeated KPSS tests  2) Include constant unless d = 2.  3) Select and by minimising AICc. Instead of searching through all possible models  1. Fit 4 initial model: ARIMA(2,d,2), ARIMA(0,d,0), ARIMA(1,d,0), ARIMA(0,d,1). If d ≤ 1, fit extra model: ARIMA(0,d,0) w/o  2. Best model (smallest AICc value) fitted in 1. is set to be the current model  3. Consider variations on the current model: Vary p and/or q by ±1 OR include/exclude from the current model  4) Repeat step 3.3 until no lower AICc can be found  ARIMA Modeling Procedure.  If not using automated algo, can use the steps on the right | |
| Point and Interval Forecasts | Point Forecasts. 1) Expand ARIMA eqn s.t. yt is on the LHS and all other terms on RHS. 2) Rewrite eqn by replacing t w T+h  3) On RHS, replace future obs by their forecasts, future errors by 0, and past errors by the corresponding residuals  Start w h = 1, and repeat for h = 2, 3, … until all required forecasts have been computed | |
| E.g. Forecasts for ARIMA(3,1,1):  Then expand backshift operator on LHS to get  Then,  For h = 1, we replace t by T + 1:  Then replace eT+1 by 0 and eT by :  For h = 2, forecast would be:  Note for point forecast: | |
| Computing Residuals. Consider MA(1) process: . Suppose y1 = 5.2, y2 = 6.1, y3 = 6  Set all innovations before t = 1 to be 0. Then y1 = 6 + e1 + 0.23e0. Then = E(6 + e1 + 0.23e0) = 6 + 0 + 0.23E(0) = 6. So e1 = y1 - = - 0.9  y2 = 6 + e2 + 0.23e1. = 6 + 0 + 0.23(-0.9) = 5.793. e2 = y2 - = 0.307 | |
| Forecast Intervals. Consider MA(q) model:  When we condition on y1, …, yT, all innovations up to and including time T are know. They don't contribute to the variance of yT+h|T  Hence, , where we take for i > q  A 95% forecast interval for ARIMA forecasts is: , where = estimated forecast var | |
| Seasonal ARIMA models | A seasonal ARIMA model is formed by including additional seasonal terms in ARIMA models: ARIMA(p,d,q)(P,D,Q)m  - (p,d,q) = non-seasonal part of model. (P,D,Q)m = seasonal part of model. m = num of obs or periods in a season  SARIMA: | |
| |  |  |  |  | | --- | --- | --- | --- | |  | SAR(P)m | SMA(Q)m | SARMA(P, Q)m | | ACF\* | Tails off | Cuts off lag Qm | Tails off | | PACF\* | Cuts off lag Pm | Tails off | Tails off |   The seasonal part of an AR or MA model can be seen in the seasonal lags of the PACF and ACF.  \* Values at nonseasonal lag are 0 | |
| E.g.    : SARIMA(0,1,1)(0,1,1)12 | E.g. SARIMA(1,1,1)(1,1,1)12 : |
| ETS vs ARIMA | |  |  |  | | --- | --- | --- | | ETS | ARIMA | Parameters | | ETS(A,N,N) | ARIMA(0,1,1) |  | | ETS(A,A,N) | ARIMA(0,2,2) | . | | ETS(A,Ad,N) | ARIMA(1,1,2) | . . | | ETS(A,N,A) | ARIMA(0,1,m)(0,1,0)m |  | | ETS(A,A,A) | ARIMA(0,1,m+1)(0,1,0)m |  | | ETS(A,Ad,A) | ARIMA(1,0,m+1)(0,1,0)m |  |   The ETS and ARIMA model classes overlap with the additive ETS models having equivalent ARIMA forms.ETS(A,N,N) and ARIMA(0,1,1)  .    = ARIMA(0,1,1)  For invertibility,  ETS(A,A,N) and ARIMA(0,2,2)  .      So | |
| Tut | *features(train\_set, Y, features = feature\_set(tags=c("decomposition", "boxcox")))* #Suppose boxcox lambda = 0.110  *ets\_models <- model(train\_set, ets\_auto = ETS(Y)); forecast(ets\_models, h=18) %>% accuracy(TS)*  *select(ets\_models, ets\_auto) %>% report(); states <- ets\_models$ets\_auto[[1]]$fit$states; tail(states)*  *ets\_after\_transform <- model(train\_set, ets\_boxcox = ETS(box\_cox(Y, 0.110))); forecast(ets\_after\_transform, h=18) %>% accuracy(TS)*  # Decomposition model: ETS on season and RW w drift on seasonally adjusted series  *model(train\_set, stl1 = STL(box\_cox(Y, 0.110))) %>% components %>% autoplot()*  *stl\_models <- model(train\_set, stl1 = decomposition\_model( STL(box\_cox(Y, 0.110)), ETS(season\_year), RW(season\_adjust ~ drift())))*  *forecast(stl\_models, h=18) %>% accuracy(TS)* | |
|  |  | |
|  |  | |
|  |  | |
|  |  | |
|  |  | |
|  |  | |
|  |  | |