Time Series Analysis & Forecasting

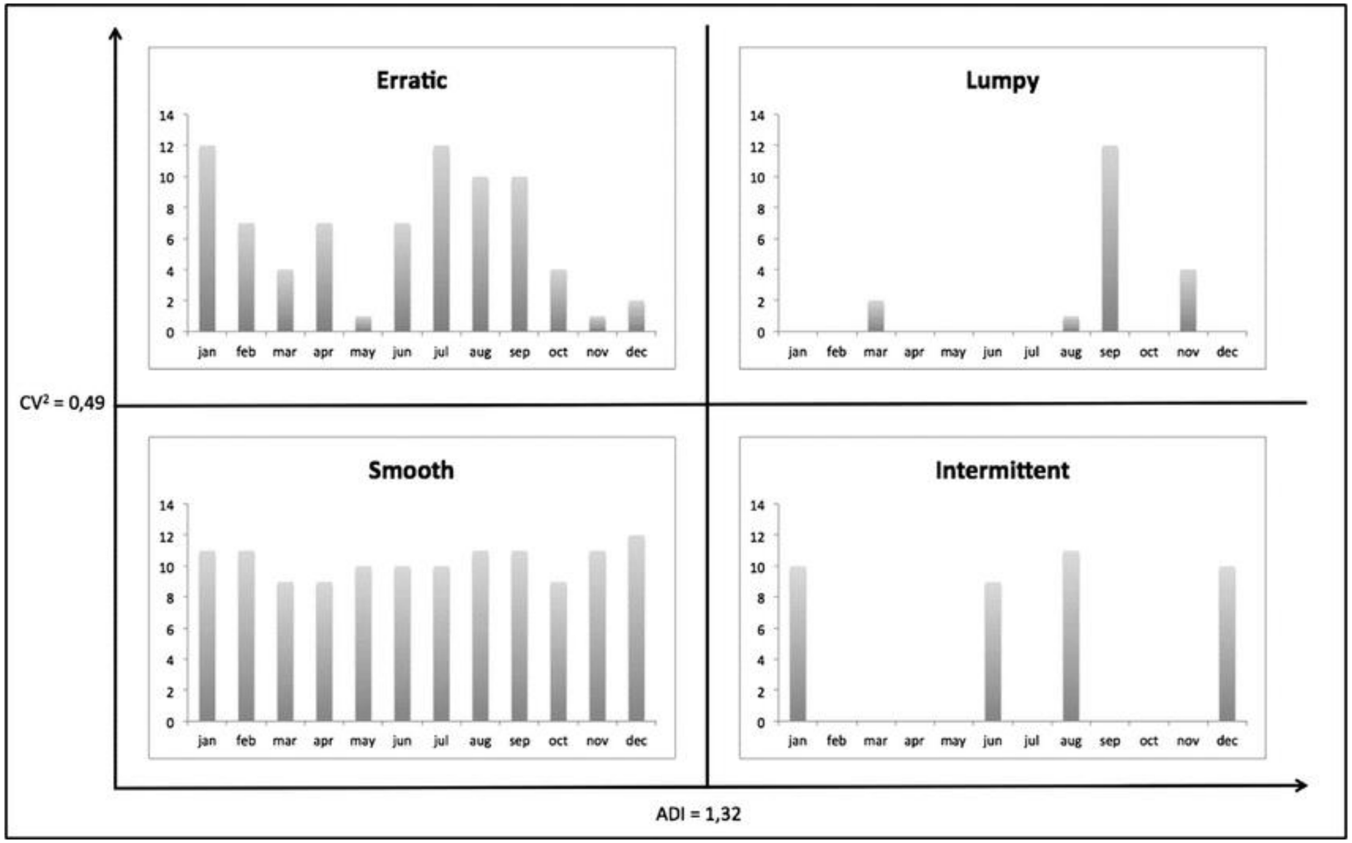
Class 6

Arnab Bose, Ph.D.

MSc Analytics

University of Chicago

Time Series Demand Forecastability



Average Demand Interval (ADI) – measures the demand regularity in time by computing the average interval between two demands.

Square of the Coefficient of Variation (CV^2) – measures the variation in quantities

https://www.researchgate.net/figure/Demand-patterns-smooth-intermittent-erratic-and-lumpy_fig1_320089625

Time Series Forecastability Zones

Smooth Demand – regular in time intervals and quantity → easy to forecast.

Intermittent Demand – high variation in time intervals but regular in quantity → high forecast errors.

Erratic Demand – regular in time intervals but high variation in quantity → high forecast errors.

Lumpy Demand – high variation in time intervals and quantity → difficult to forecast.

Count Time Series

Period	1	2	3	4	5	6	7	8	9	10	11	12
Demand	6	5					14			21		17

$$ADI = \frac{total \ \# \ of \ periods}{demand \ buckets} = \frac{12}{5} = 2.4$$

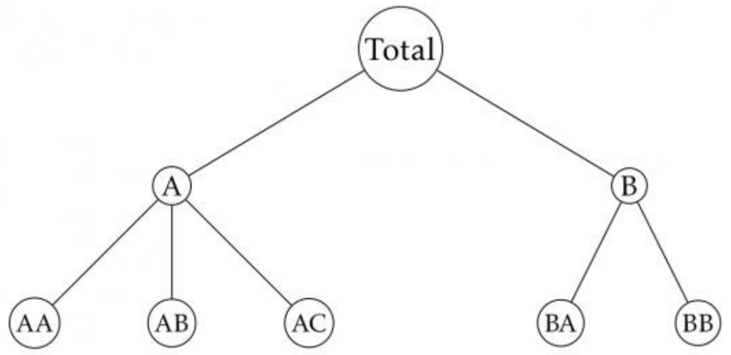
$$CV = \frac{standard \ deviation}{average} = \frac{6.95}{12.6} = 0.552$$

$$CV^2 = 0.304$$

Therefore, this is an intermittent demand pattern.

One way to handle such demand forecasting is using Count TS Forecasting - https://otexts.com/fpp2/counts.html

Hierarchical Time Series

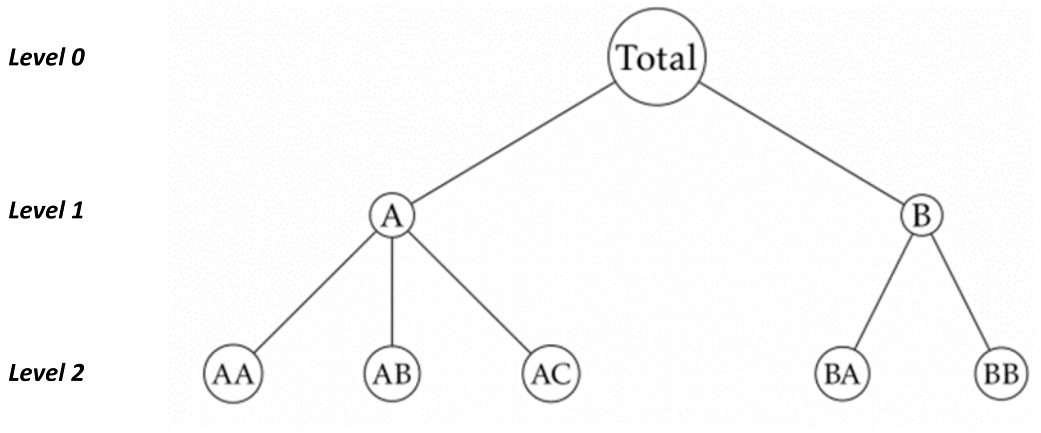


A two level hierarchical tree diagram*

- Bottom-up approach – generate forecasts for the bottom level TS and keep aggregating up the hierarchy. Pro: no loss of info. Con: noisy data that is challenge to model and forecast
- Top-down approach – generate forecasts for the "Total" series at the top and then disaggregate downward using a set of proportions to distribute to each series at the lower levels. Pro: simple to model. Con: unable to capture individual TS
- Middle-out approach – chose a "middle level" and forecast for all TS at this level. For TS above, aggregate like bottom-up. For series below, disaggregate like top-down.

* https://www.otexts.org/fpp/9/4 , Accessed May 2018

R code – Hierarchical Time Series



```
library("hts")
nodes <- list(2, c(3, 2))
abc <- ts(5 + matrix(sort(rnorm(500)), ncol = 5, nrow = 100))
x <- hts(abc, nodes)
summary(x)
smatrix(x)
plot(x)
fcst <- forecast(x, method="mo", fmethod ="arima", level = 1, h=10)
plot(fcst)
```

R hts package – arima model params ?

- hts forecasting does not output the arima (P, D, Q)
- Because a separate ARIMA model is estimated for every series in the hierarchy. For large hierarchies, that can involve thousands or even millions of models. There is no point in storing all the resulting information when you only want forecasts.
- If you want the individual models, then fit them explicitly to all series. You can get the matrix of every series (included aggregated series) using *aggts()*:

```
y <- aggts(fcst)

models <- list()

for(i in 1:ncol(y))

     models[[i]] <- auto.arima(y[,i])
```

State Space Models (1/2)

Common mathematical formulation that includes regression and ARIMA models

- Why is this formulation important?
    - Real world motivation – factor in measurement error and inject prior knowledge
    - Most software (including R) uses state space representation of model implementations
    - Incorporates Bayesian approach to forecasting

- From R help on arima {stats}
    - The exact likelihood is computed via a state-space representation of the ARIMA process, and the innovations and their variance found by a Kalman filter.

- From R help on Arima {forecast}
    - Largely a wrapper for the arima function in the stats package. The main difference is that this function allows a drift term.

State Space Models (2/2)

- Based on Markov property that process future is dependent on current process state and independent of process past
- Assume system is represented by state vector X_t at time t
- Has 2 equations:
  - <u>Observation or measurement equation</u> – describes how TS observations are done from the state vector

$$Y_t = h_t' X_t + \varepsilon_t$$

h_t' - known vector of constants

ε_t - observation error

  - <u>State or system equation</u> – describes how the state vector evolves through time

$$X_t = AX_{t-1} + Ga_t$$

$A \ and \ G$ – known matrices

a_t - process noise

Bayesian Structural Time Series

Use preexisting structural state space components to build more complex models fitted with

1. structural model including specification of priors
2. Kalman filter to update state estimates
3. Regression (if any) with spike-and-slab method
4. Model averaging using Bayesian approach for forecasting

The R package bsts runs multiple Markov Chain Monte Carlo (MCMC) computations of the posterior.

BSTS with Regression and Seasonality

State space representation with trend μ_t (latent variable), seasonality τ_t and regression $\beta^t\, x_t$

$$y_t = \mu_t + \tau_t + \beta^t\, x_t + \varepsilon_t$$

$$\mu_{t+1} = \mu_t + \delta_t + \eta_{0t}$$

$$\delta_{t+1} = \delta_t + \eta_{1t}$$

$$\tau_{t+1} = -\sum_{s=1}^{S-1} \tau_t + \eta_{1t}$$

μ_t - trend

τ_t - seasonal pattern

$\beta^t\, x_t$ - regression component

δ_t - slope of the local linear trend

$\eta_{0t}, \eta_{1t}, \eta_{2t}$ - measurement/process noise

https://www.unofficialgoogledatascience.com/2017/07/fitting-bayesian-structural-time-series.html

https://multithreaded.stitchfix.com/blog/2016/04/21/forget-arima/

R code - bsts

```
library(datasets)
library(bsts)
ts.plot(AirPassengers)
Y <- window(AirPassengers, start=c(1949, 1), end=c(1959,12))
y <- log10(Y)
ss <- AddLocalLinearTrend(list(), y)
ss <- AddSeasonal(ss, y, nseasons = 12)
model1 <- bsts(y, state.specification = ss, niter = 100)
plot(model1)
plot(model1, “components”)
plot(model1, "seasonal", nseasons = 12)
pred <- predict.bsts(model1, horizon = 12, quantiles = c(.025, .975))
plot(pred)
```

Google Forecasting Model

"Large-Scale Parallel Statistical Forecasting Computations in R" in Week 6 Reading

Highlights

1. Automates data cleansing and smoothing – account for missing data, outlier, level changes (due to product launch/promotions) and yearly/weekly seasonality.

2. Does time aggregation (team found weekly data worked best for most of their forecasts) and conceptual disaggregation (team also found that sometimes disaggregation by geography/product category for forecasts and then aggregating model outputs worked well).

3. Ensembles forecasts from different model results – ensemble "wisdom of crowds" also helps to quantify uncertainty/prediction intervals.

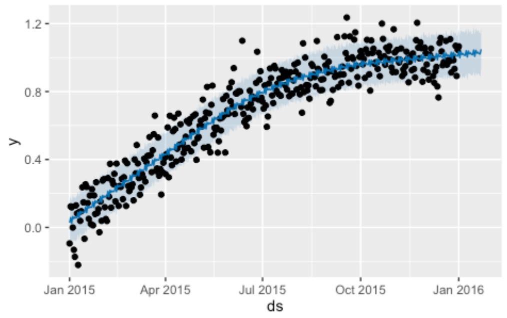
Prophet from Facebook

1. Curve-fitting approach similar to Generalized Additive Models (GAM) – class of regression models
2. Incorporates analyst-in-the-loop modeling to influence model parameters and apriori user-specified data
3. TS represented as $y_t = g(t) + s(t) + h(t) + \varepsilon_t$, where
   1. $g(t)$ is trend function that models non-periodic changes using a logistic growth curve with step intervention for trend changes with changepoints either user-specified or chosen automatically
   2. $s(t)$ is seasonal component that models periodic changes using Fourier in frequency domain
   3. $h(t)$ is user-specified list of holidays
   4. ε_t is Gaussian error
4. No regular interval TS assumption
5. Model fitting using Stan* for non-linear additive models

*Probabilistic programming language in C++ https://mc-stan.org

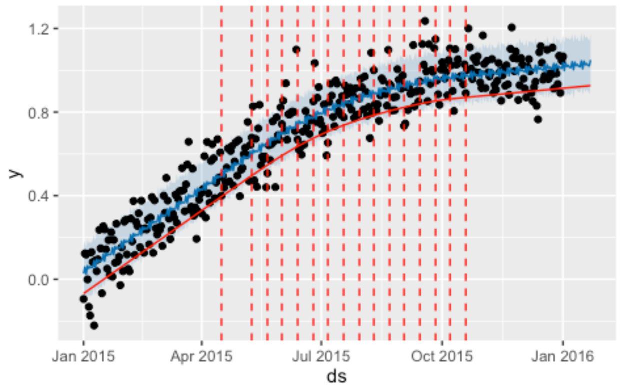
R code – Prophet (1/3)

```
library(prophet)

## Loading required package: Rcpp

## Loading required package: rlang

history <- data.frame(ds = seq(as.Date('2015-01-01'),
as.Date('2016-01-01'), by = 'd'), y = sin(1:366/200) +
rnorm(366)/10)
plot(history)
m <- prophet(history, daily.seasonality = T)

## Disabling yearly seasonality. Run prophet with
yearly.seasonality=TRUE to override this.


future = make_future_dataframe(m, periods = 21)
forecast = predict(m, future)
plot(m, forecast)
```



R code – Prophet (2/3)

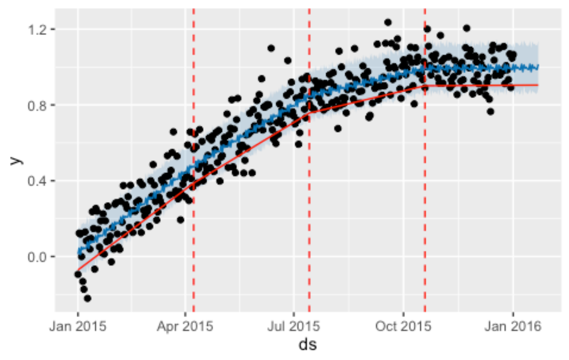
```
# the trend change points are automatically detected
# and then uses L1 regularization to use the min possible
plot(m, forecast) + add_changepoints_to_plot(m)
```



https://facebook.github.io/prophet/docs/trend_changepoints.html

R code – Prophet (3/3)

```
# with number of changepoints specified, higher the
value, more flexibility in trend (move towards overfit)

# this is equivalent to L1 reg
m3 <- prophet(history, daily.seasonality = T,
n.changepoints = 3, changepoint.prior.scale = 0.1)

## Disabling yearly seasonality. Run prophet with
yearly.seasonality=TRUE to override this.

future = make_future_dataframe(m3, periods = 21)
forecast = predict(m3, future)
plot(m3, forecast) + add_changepoints_to_plot(m3)
```



R code – Specifying Holidays in Prophet

```
# special events

superbowls <- data_frame(
  holiday = 'superbowl',
  ds = as.Date(c('2010-02-07', '2014-02-02', '2016-02-07')),
  lower_window = 0,
  upper_window = 1
)

s <- prophet(holidays = superbowls)

# country holidays
s <- add_country_holidays(s, country_name = 'US')
s <- fit.prophet(s, history)
```

So how does Prophet compare ? Jury is not looking good ..

" … concern is not that it is not ranking first, but that at best it is almost 16% worse than exponential smoothing (and at worst almost 44%!) .."

https://kourentzes.com/forecasting/2017/07/29/benchmarking-facebooks-prophet/

Performance is concerning - https://www.microprediction.com/blog/prophet

" … compared the results against the one from Prophet with no parameter setting, and found that the ARIMA with fine tuned parameters produced the best result for this data .."

https://blog.exploratory.io/is-prophet-better-than-arima-for-forecasting-time-series-fa9ae08a5851

Orbit – Uber Forecasting Package

Object-O**R**iented **B**ayes**I**an **T**ime Series (ORBIT) – a general interface for Bayesian time series modeling.

Uses a programming language (like Stan) for posterior approximation.

Available only in Python.

Has 2 major types of bsts models

1. Seasonal Local/Global Trend model (LGT)

$$y_t = \mu_t + s_t + \epsilon_t$$
$$\mu_t = l_{t-1} + \xi_1 b_{t-1} + \xi_2 l_{t-1}^{\lambda}$$

2. Damped Local Trend model (DLT)

$$y_t = \mu_t + s_t + r_t + \epsilon_t$$
$$\mu_t = D(t) + l_{t-1} + \theta b_{t-1}$$

https://eng.uber.com/orbit/

So how does Orbit compare to Prophet?

“ … shows that our models consistently deliver better accuracy than other candidate time series models in terms of SMAPE. Orbit is also computationally efficient. For example, the average compute time per series with full MCMC sampling and prediction from a subset of M4 weekly data is about 2.5 minutes and 16 ms. The run time for the same series in Prophet is about 10 minutes for sampling and 2.4 s for prediction. That’s a 4x speed up in training, and orders of magnitude difference in prediction. “

E. Ng, Z. Wang, H. Chen, S. Yang, S. Smyl, *Orbit: Probabilistic Forecast with Exponential Smoothing*, https://arxiv.org/pdf/2004.08492.pdf

Silverkite from LinkedIn

A time series forecasting library from LinkedIn.

https://engineering.linkedin.com/blog/2021/greykite--a-flexible--intuitive--and-fast-forecasting-library

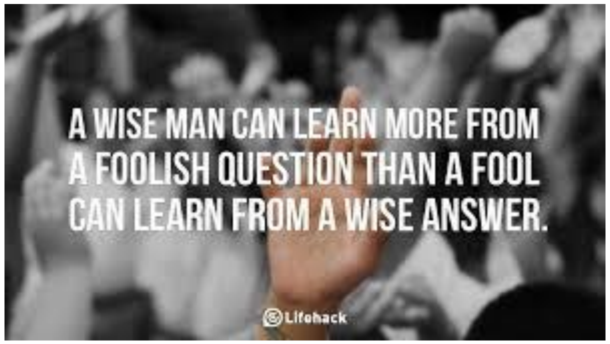
Textbook Chapters

Materials covered available in book chapters:

FPP: 11, 12

PTS: 7, 11, 16

Thank You


A WISE MAN CAN LEARN MORE FROM A FOOLISH QUESTION THAN A FOOL CAN LEARN FROM A WISE ANSWER.
Lifehack