Effective R: LGBM

Ryuta Yoshimatsu

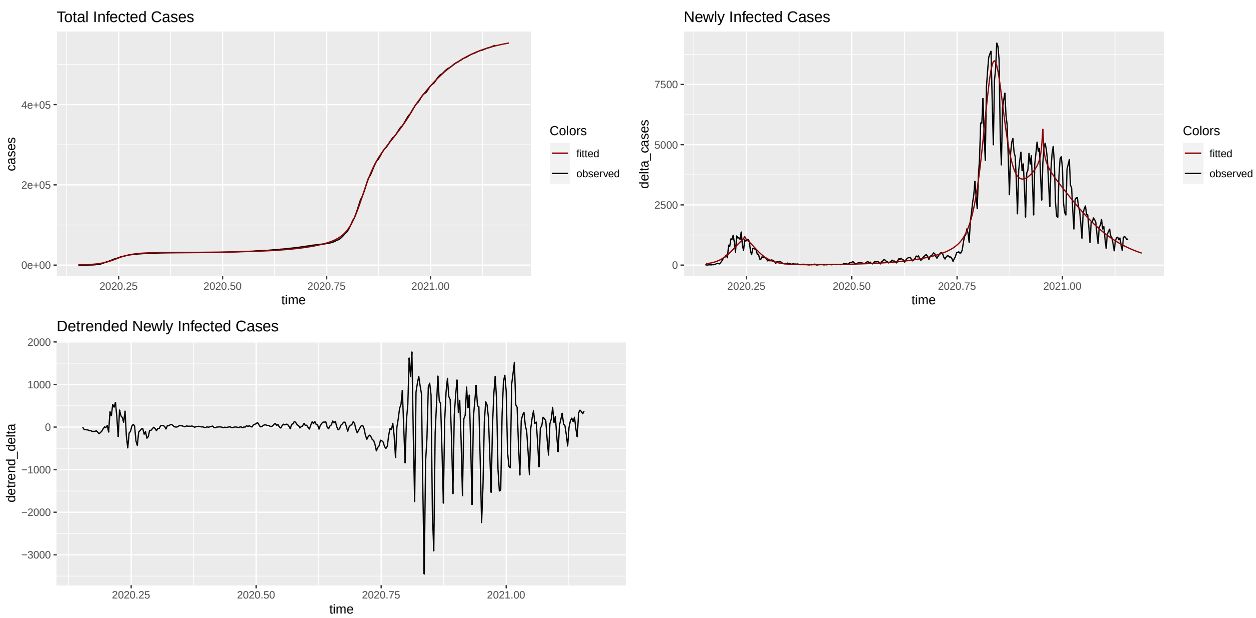
Contents

```r
library(data.table)
library(Matrix)
library(dplyr)
library(MLmetrics)
library(lightgbm)
library(ggplot2)
library(gridExtra)
library(grid)
library(graphics)
library(TTR)
library(forecast)
library(lubridate)
library(mltools)
library(data.table)
library(ggplotify)
library(gridBase)
library(tsibble)
library(fable)
```

Read the data.

```r
df <- fread("fitted.csv")
df <- df[, delta_cases := c(0, diff(cases))][]
df <- df[, trend := c(0, diff(fitted))][]
df <- df %>% mutate(detrend_delta = delta_cases - trend)

p1 <- ggplot(df, aes(x=time)) +
  geom_line(aes(y = cases, color="observed")) +
  geom_line(aes(y = fitted, color="fitted")) +
  ggtitle("Total Infected Cases") +
  scale_color_manual(name = "Colors", values = c("observed" = "black", "fitted" = "darkred"))

p2 <- ggplot(df, aes(x=time)) +
  geom_line(aes(y = delta_cases, color="observed")) +
  geom_line(aes(y = trend, color="fitted")) +
  ggtitle("Newly Infected Cases") +
  scale_color_manual(name = "Colors", values = c("observed" = "black", "fitted" = "darkred"))

p3 <- ggplot(df, aes(x=time)) +
  geom_line(aes(y = detrend_delta), color = "black") +
  ggtitle("Detrended Newly Infected Cases")
```

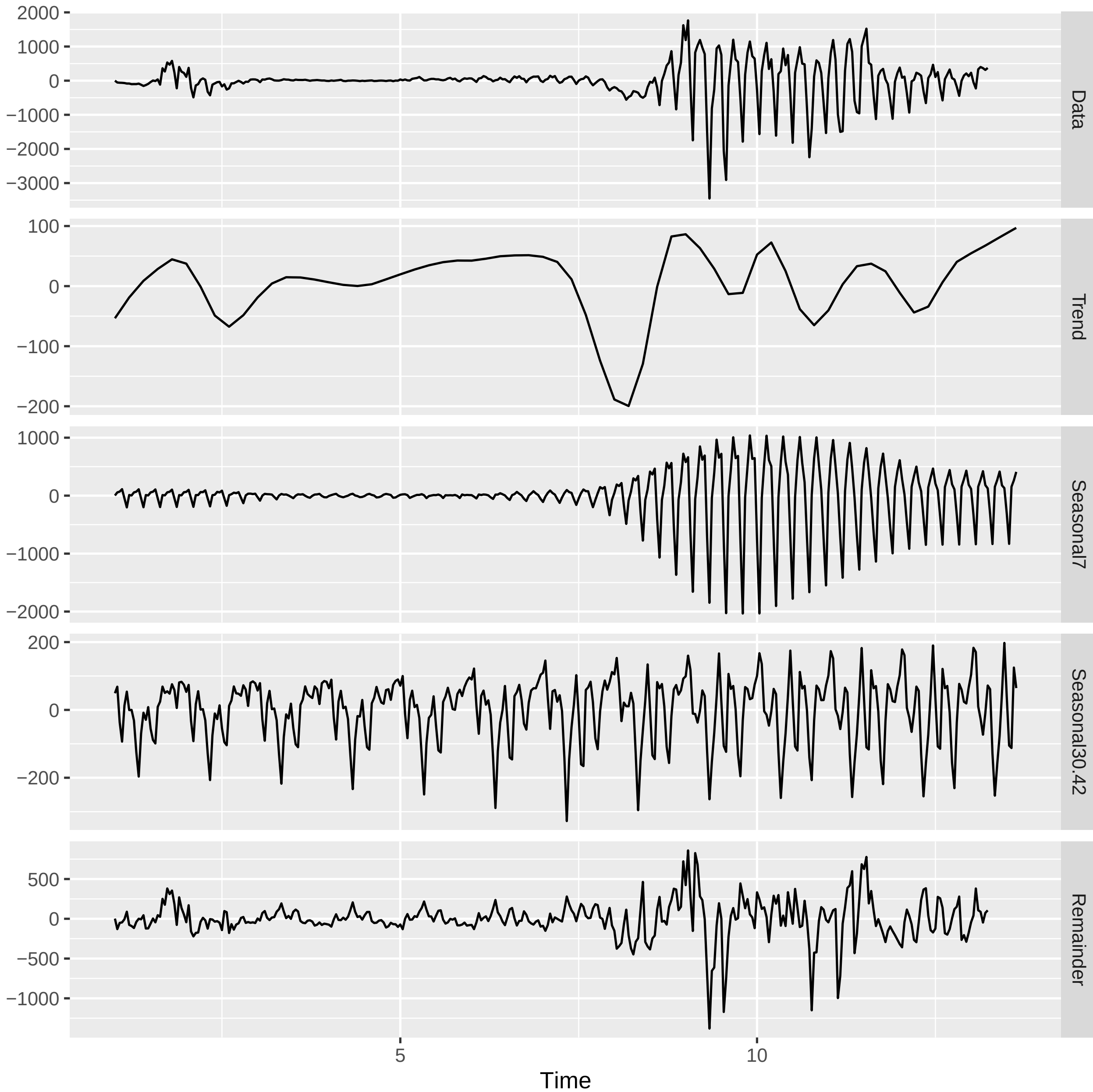
```r
# Create a grid of plots
grid.arrange(
  p1, p2, p3,
  nrow = 2,
  bottom = textGrob(
    "",
    gp = gpar(fontface = 3, fontsize = 9),
    hjust = 1,
    x = 1
  )
)
```



1. Linear combination of three logistic curves does very well at fitting to the total infected cases and estimating the overall trend.
2. But it does not to capture the seasonality.

Decompose the detrended series into trend, seasonal and residual components.

```r
detrend_delta.ts <- msts(df$detrend_delta, seasonal.periods=c(7, 30.4167))
detrend_delta.ts %>% mstl() %>% autoplot()
```



1. Detrended cases of newly infected incidents has two prominent seasonality components: weekly and monthly.

We perform regression with ARIMA errors including Fourier terms with base periodicity at 7 and 30.4167 as additional regressors. The smoothness of the seasonal pattern are controlled by K (the number of Fourier sin and cos pairs – the seasonal pattern is smoother for smaller values of K). We use AIC to find the optimal K.

```
bestfit <- list(aicc=Inf)
for(i in 1:3)
  for(j in 1:5)
  {
    fit <- auto.arima(detrend_delta.ts, xreg=fourier(detrend_delta.ts, K=c(i,j)), stationary=FALSE, sea
    if(fit$aicc < bestfit$aicc)
      bestfit <- fit
    else break;
```

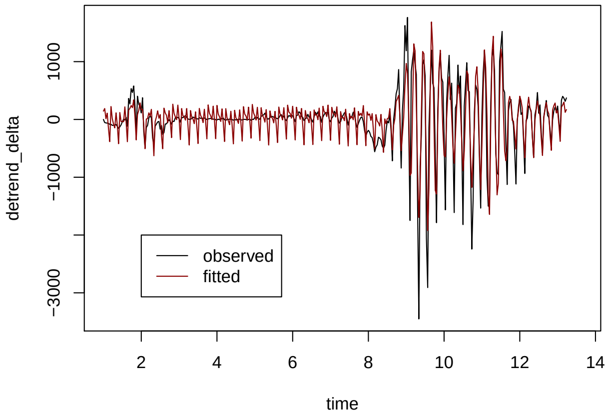
```r
  }
summary(bestfit)
```

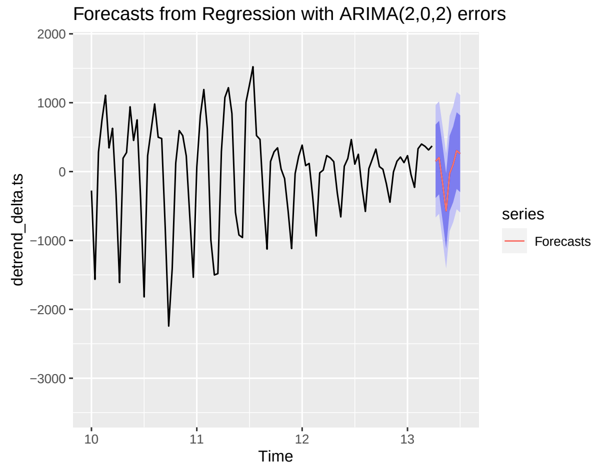
```
## Series: detrend_delta.ts
## Regression with ARIMA(2,0,2) errors
##
## Coefficients:
##          ar1      ar2      ma1     ma2      S1-7       C1-7      S2-7     C2-7
##       1.1251  -0.9340  -0.7847  0.8289  333.7887  -164.3726  134.7933  73.1698
## s.e.  0.0232   0.0199   0.0301  0.0399   93.0041    92.8901   18.4457  18.4869
##           S3-7      C3-7     S1-30    C1-30
##       -20.2640  106.4635  -41.3396  44.4247
## s.e.   20.2289   20.3071   31.3695  31.2162
##
## sigma^2 estimated as 108456:  log likelihood=-2650.7
## AIC=5327.39   AICc=5328.42   BIC=5378.2
##
## Training set error measures:
##                    ME     RMSE      MAE  MPE MAPE      MASE       ACF1
## Training set 2.841227 323.9123 214.1709 -Inf  Inf 0.4116287 0.07435475
```

```r
# Number of days to forecast
h <- sum(is.na(df$cases))
print(h)
```

```
## [1] 12
```

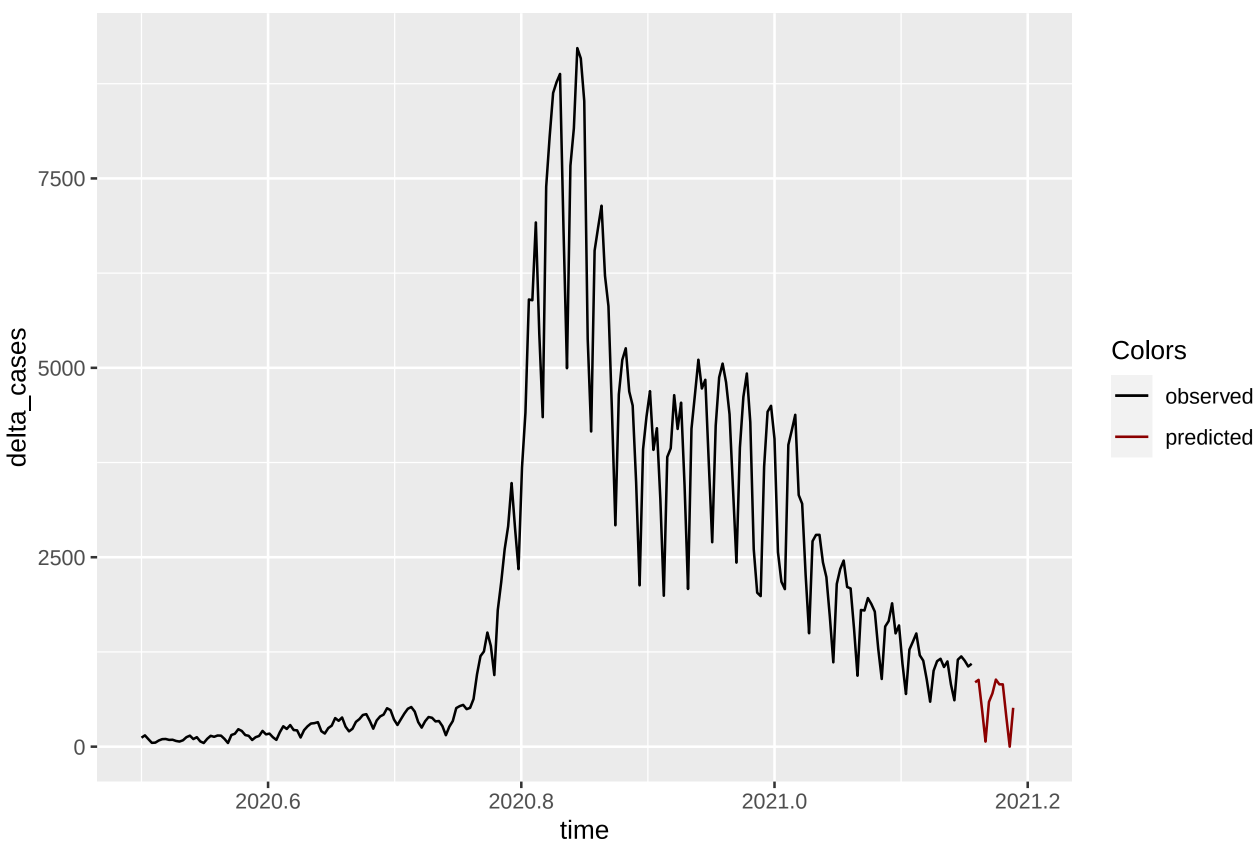
```r
detrend_delta.forecasts <- forecast(bestfit, xreg=fourier(detrend_delta.ts, K=c(3,1), h=h))

par(mfrow=c(1, 2))

plot(bestfit$x, col="black", xlab="time", ylab="detrend_delta")
lines(fitted(bestfit), col="darkred")
legend(2, -2000, legend=c("observed", "fitted"), col=c("black", "darkred"), lty=1:1, cex=1)
plot.new()

vps <- baseViewports()
pushViewport(vps$figure)        ## I am in the space of the base plot
vp1 <- plotViewport(c(1,1,1,1))  ## Create new vp with margins
f <- autoplot(detrend_delta.forecasts) + autolayer(detrend_delta.forecasts$mean, series="Forecasts") + 
print(f, vp=vp1)
```






```
seasonal <- append(data.frame(fitted(bestfit))[1:(length(fitted(bestfit))-h),], data.frame(detrend_delt
df <- cbind(df, seasonal)
df <- df %>% mutate(predicted = trend + seasonal)
df$predicted[df$predicted < 0] <- 0
```

```
ggplot(df, aes(x=time)) +
  xlim(2020.5, 2021.2) +
  geom_line(aes(y=delta_cases, color="observed")) +
  geom_line(data=tail(df, h), aes(x=time, y=predicted, color="predicted")) +
  scale_color_manual(name = "Colors", values = c("observed"="black", "predicted"="darkred"))
```



```r
tail(df, 12)
```

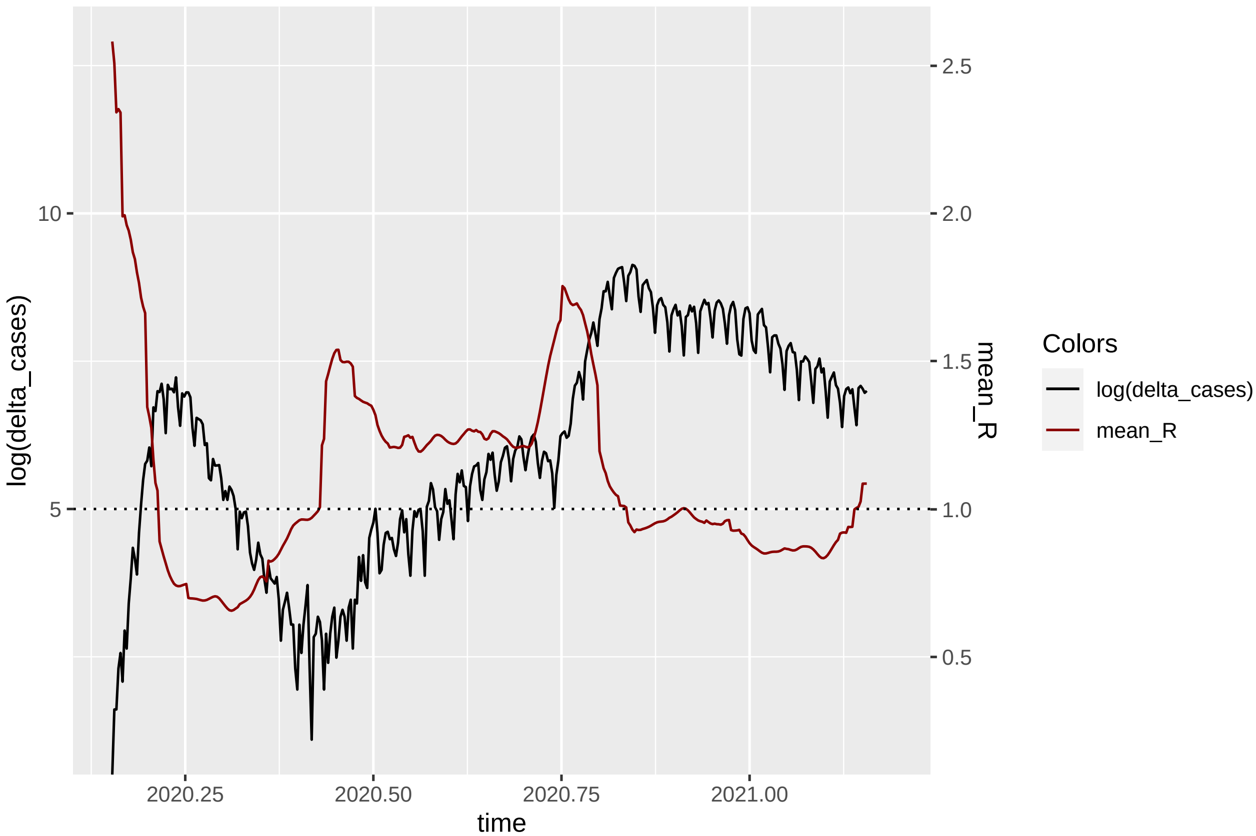
```
##      V1     time cases   fitted delta_cases    trend detrend_delta   seasonal
##  1: 368 2021.158    NA 546994.6          NA 699.6363            NA  149.49383
##  2: 369 2021.161    NA 547673.0          NA 678.3324            NA  203.47211
##  3: 370 2021.164    NA 548330.6          NA 657.6335            NA -167.98972
##  4: 371 2021.167    NA 548968.1          NA 637.5259            NA -570.90562
##  5: 372 2021.169    NA 549586.1          NA 617.9957            NA  -27.52722
##  6: 373 2021.172    NA 550185.2          NA 599.0292            NA  103.81550
##  7: 374 2021.175    NA 550765.8          NA 580.6130            NA  304.37104
##  8: 375 2021.178    NA 551328.5          NA 562.7335            NA  259.40520
##  9: 376 2021.180    NA 551873.9          NA 545.3774            NA  276.78872
## 10: 377 2021.183    NA 552402.4          NA 528.5317            NA -129.05690
## 11: 378 2021.186    NA 552914.6          NA 512.1833            NA -541.34402
## 12: 379 2021.189    NA 553410.9          NA 496.3194            NA   16.43415
##     predicted
##  1: 849.13018
##  2: 881.80451
##  3: 489.64380
##  4:  66.62024
##  5: 590.46844
##  6: 702.84472
##  7: 884.98402
##  8: 822.13868
##  9: 822.16615
## 10: 399.47478
## 11:   0.00000
## 12: 512.75354
```

(Compare this prediction with currently available number of cases and uplift if smaller.)

```r
estimated_R <- read.csv(url("https://raw.githubusercontent.com/covid-19-Re/dailyRe-Data/master/CHE-esti
estimated_R <- estimated_R %>% group_by(date) %>% summarise(mean_R = mean(median_R_mean))
estimated_R$date <- as.Date(estimated_R$date, "%Y-%m-%d")
estimated_R <- estimated_R[which(estimated_R$date >= as.Date('2020-02-25', "%Y-%m-%d")), ]
mean_R <- estimated_R$mean_R
for (i in (length(mean_R)+1):(length(df$time)))
  mean_R[i] <- NA
```

```r
df <- cbind(df, mean_R)
```

```r
coefficient <- 5
ggplot(df, aes(x=time)) +
  geom_line(aes(y=log(delta_cases), colour='log(delta_cases)')) +
  geom_line(aes(y=mean_R*coefficient, colour='mean_R')) +
  scale_y_continuous(name="log(delta_cases)", sec.axis = sec_axis(~.*1./coefficient, name="mean_R")) +
  scale_color_manual(name = "Colors", values = c("log(delta_cases)" = "black", "mean_R" = "darkred")) +
  geom_hline(yintercept=coefficient, linetype = 'dotted', color='black')
```

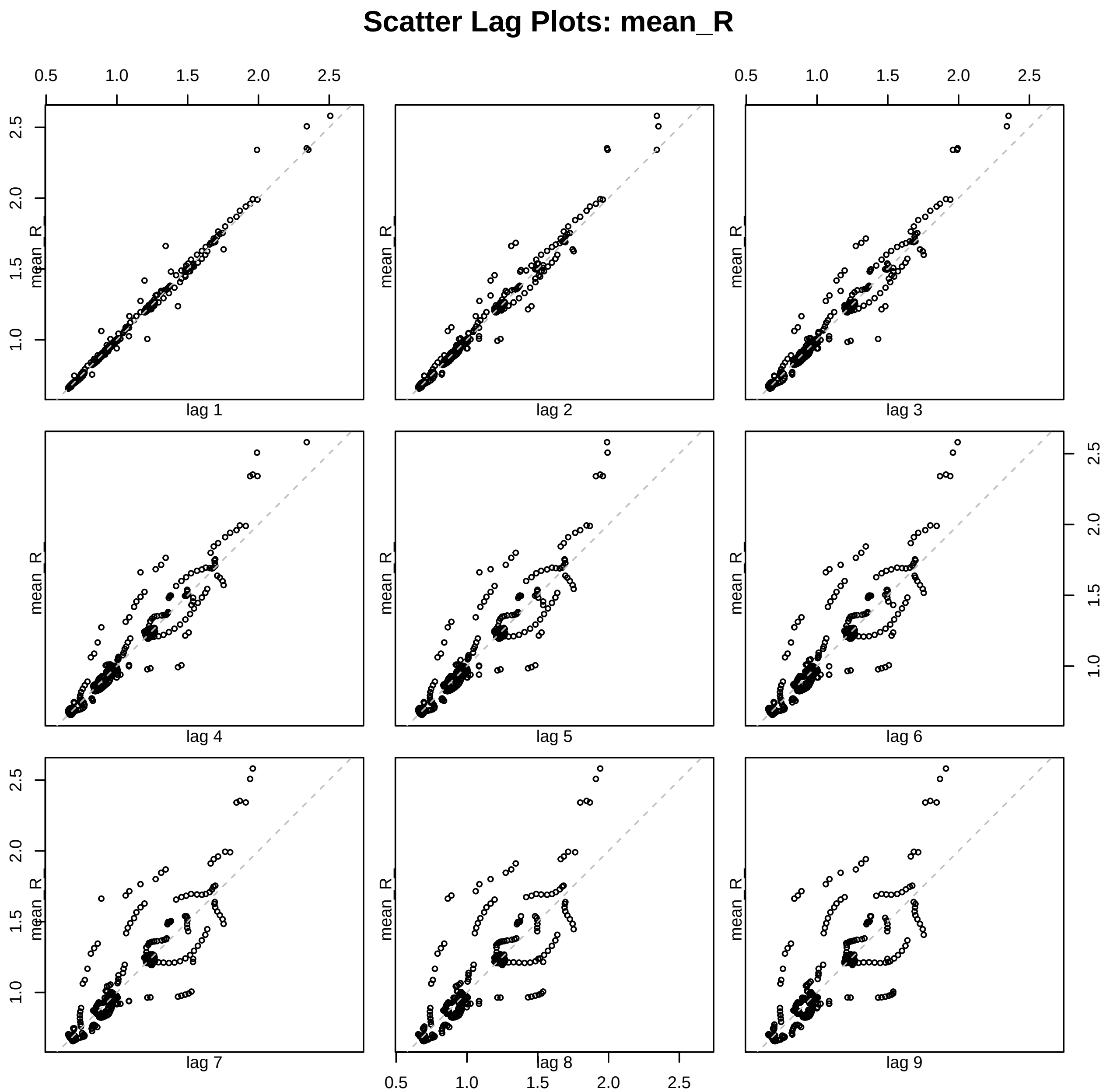



```
correlation_df <- df %>% select(delta_cases, mean_R)
correlation_df$delta_cases <- log(correlation_df$delta_cases)
correlation_df <- correlation_df[-1,]
correlation_df <- correlation_df[complete.cases(correlation_df), ]
cat('correlation between number of cases and the log of effective R: ', cor(correlation_df$delta_cases,
```

```
## correlation between number of cases and the log of effective R:  -0.3683602
```

The correlation coefficient between R and new cases is -0.25, which suggests that there is no strong 'linear' correlation between the two variables. But, since R is the rate of exponential growth in the ODE of any transmission models, I would expect some degree of relationship between the log of new cases and R.

```
mean_R_ <- na.omit(df$mean_R)
lag.plot(mean_R_, lags=9, main="Scatter Lag Plots: mean_R")
```



Prepare the training data table for modeling.

```
# Prepare training data table
X <- df %>% select(time, delta_cases, mean_R)
X <- X  %>% rename(cases = delta_cases, R = mean_R)
```

Function to create lagged and rolling window features.

```
create_features <- function(dt) {

  # Add lag vectors: table must be sorted by date!
  R_lags <- c(1, 2, 3)
  R_lag_cols <- paste0("R_lag_", R_lags)
  dt[, (R_lag_cols) := shift(.SD, R_lags), .SDcols="R"]

  cases_lag <- c(7, 28)
```

```r
  cases_lag_cols <- paste0("cases_lag_", cases_lag)
  dt[, (cases_lag_cols) := shift(.SD, cases_lag), .SDcols="cases"]

  # Add rolling window vectors: table must be sorted by date!
  R_windows <- c(7)
  R_roll_cols <- paste0("R_rmean_", t(outer(R_lags, R_windows, paste, sep="_")))
  dt[, (R_roll_cols) := frollmean(.SD, R_windows, na.rm=TRUE), .SDcols=R_lag_cols] # Rolling features o

  cases_windows <- c(7, 28)
  cases_roll_cols <- paste0("cases_rmean_", t(outer(cases_lag, cases_windows, paste, sep="_")))
  dt[, (cases_roll_cols) := frollmean(.SD, cases_windows, na.rm=TRUE), .SDcols=cases_lag_cols] # Rollin

  return(dt)
}
```

```r
X <- create_features(X)
X <- na.omit(X)
head(X)
```

```
##        time cases         R   R_lag_1   R_lag_2   R_lag_3 cases_lag_7
## 1: 2020.230  1124 0.7738825 0.7925571 0.8164745 0.8397172         777
## 2: 2020.232  1136 0.7592228 0.7738825 0.7925571 0.8164745        1089
## 3: 2020.235  1068 0.7473265 0.7592228 0.7738825 0.7925571        1072
## 4: 2020.238  1375 0.7413192 0.7473265 0.7592228 0.7738825        1235
## 5: 2020.240   829 0.7389072 0.7413192 0.7473265 0.7592228         950
## 6: 2020.243   606 0.7393087 0.7389072 0.7413192 0.7473265         535
##    cases_lag_28 R_rmean_1_7 R_rmean_2_7 R_rmean_3_7 cases_rmean_7_7
## 1:            0   0.9077559   0.9612601   1.0266522        461.0000
## 2:            5   0.8628588   0.9077559   0.9612601        582.0000
## 3:            5   0.8195824   0.8628588   0.9077559        689.7143
## 4:           10   0.7991491   0.8195824   0.8628588        818.1429
## 5:           13   0.7815000   0.7991491   0.8195824        893.8571
## 6:            8   0.7670986   0.7815000   0.7991491        926.5714
##    cases_rmean_7_28 cases_rmean_28_7 cases_rmean_28_28
## 1:         173.9091         0.000000          0.000000
## 2:         213.6957         2.500000          2.500000
## 3:         249.4583         3.333333          3.333333
## 4:         288.8800         5.000000          5.000000
## 5:         314.3077         6.600000          6.600000
## 6:         322.4815         6.833333          6.833333
```

```r
# Split the training data set into train and eval:
#     train consists of data from "2020-02-25" to 14 days prior to the last record available in X
#     val  consists of data from the last 14 days of X

# Indexes for the training set
idx <- c(1:(length(X$time)-14))

# Labels for the training set
y <- X$R

# Drop columns "time" and "R"
X[, c("time", "R") := NULL]
```

Convert a data frame to a numeric matrix: return the matrix obtained by converting all the variables in a data frame to numeric mode and then binding them together as the columns of a matrix.

```
X <- data.matrix(X)
```

Construct lgb dataset.

```
xtrain <- lgb.Dataset(X[idx, ], label=y[idx])
xval <- lgb.Dataset(X[-idx, ], label=y[-idx])
```

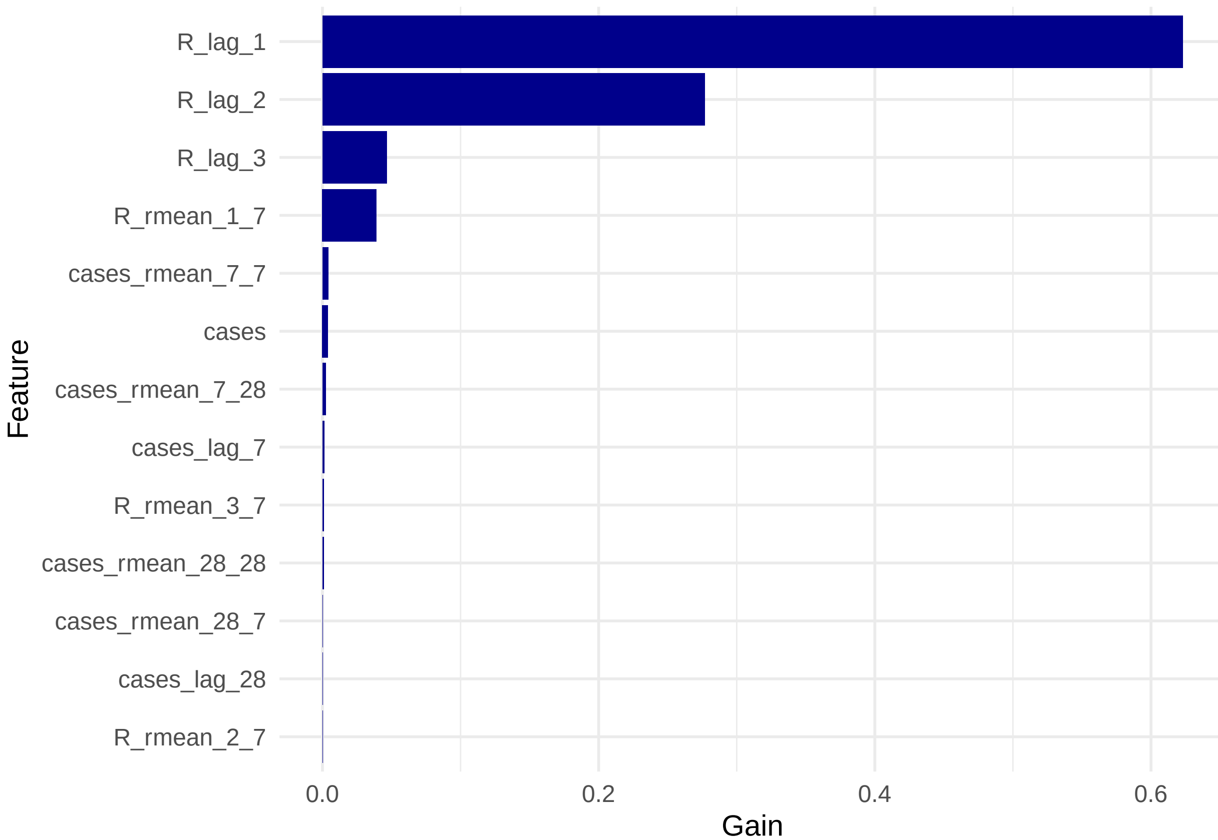
We use Poisson regression (from generalize linear model family), which is suitable for counts. The model assumes the errors are Poission distributed and thus could capture a skew, discrete distribution, and the restriction to response variables to be non-negative is applied.

```
# Configure lgb hyper parameters
p <- list(objective = "poisson",  # Training parameter
          metric ="rmse",         # Training parameter
          force_row_wise = TRUE,  # Training parameter: force row-wise histogram building
          learning_rate = 0.075,  # Training parameter
          num_leaves = 34,        # Regularization parameter
          min_data = 10,          # Regularization parameter
          sub_feature = 0.8,      # Regularization parameter
          sub_row = 0.75,         # Regularization parameter
          bagging_freq = 1,       # Regularization parameter
          lambda_l2 = 0.1,        # Regularization parameter
          nthread = 2)            # Training parameter

model.lgb <- lgb.train(params = p,
                   data = xtrain,
                   nrounds = 500,              # Training parameter (max number of trees)
                   valids = list(val = xval),
                   early_stopping_rounds = 100, # Training parameter (min number of trees to stop)
                   eval_freq = 50,             # Training parameter
                   verbose = -1)
```

```
cat("Best rmse on the validation set:", model.lgb$best_score, "at", model.lgb$best_iter, "iteration")
```

```
## Best rmse on the validation set: 0.01906252 at 71 iteration
```

```
imp <- lgb.importance(model.lgb)
imp[order(-Gain)
    ][1:length(imp$Feature), ggplot(.SD, aes(reorder(Feature, Gain), Gain)) +
        geom_col(fill = "darkblue") +
        xlab("Feature") +
        coord_flip() +
        theme_minimal()]
```



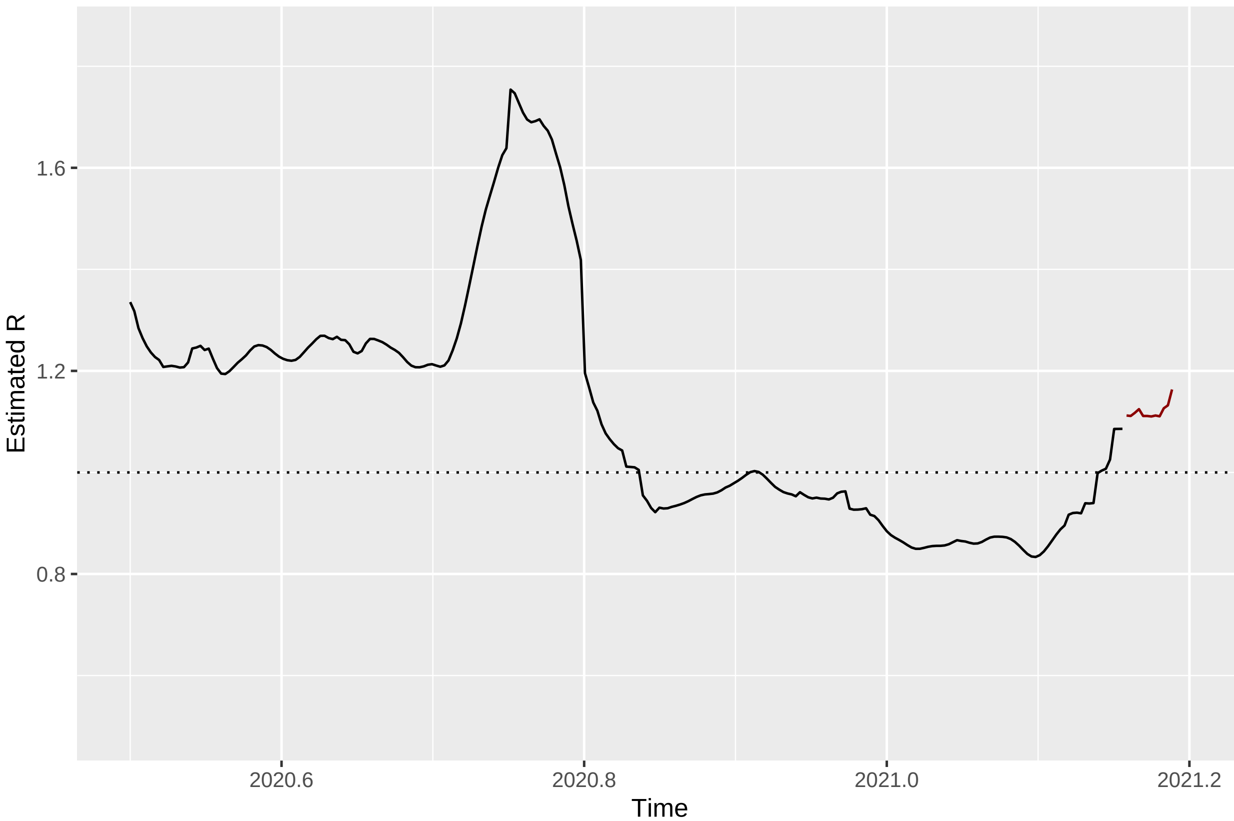
As we are using lag features we have to forecast day by day in order to use the latest predictions for the current day.

```
# Loop from (today - h) to today
tday <- Sys.Date()
fday <- tday - h + 1
count <- 1

for (day in as.list(seq(fday, tday, by="day")))
{
  # Take the subset of the data set only necessary for calculating lagged and rolling mean features for
  X.subset <- df
  if (count != h){
    X.subset <- head(X.subset, -(h-count))
  }else{}
  X.subset$delta_cases[is.na(X.subset$delta_cases)] <- X.subset$predicted[is.na(X.subset$delta_cases)]
  X.subset <- X.subset %>% select(time, delta_cases, mean_R)
  X.subset <- X.subset  %>% rename(cases=delta_cases, R=mean_R)
  insert_row <- length(X.subset$cases)

  # Create features
  X.subset <- create_features(X.subset)

  # Construct a matrix only with the 'day'
  X.subset <- tail(X.subset, n=1)
  X.subset[, c("time", "R") := NULL]
  X.subset <- data.matrix(X.subset)
```

```r
  # Update mean_R column of df
  R_prediction <- predict(model.lgb, X.subset)

  df$mean_R[insert_row] <- R_prediction

  cat(as.character(day),'\t', 'Predicted R ', R_prediction,'\n')
  count <- count + 1
}
```

```
## 2021-02-27    Predicted R  1.112205
## 2021-02-28    Predicted R  1.111234
## 2021-03-01    Predicted R  1.117422
## 2021-03-02    Predicted R  1.124724
## 2021-03-03    Predicted R  1.111214
## 2021-03-04    Predicted R  1.111214
## 2021-03-05    Predicted R  1.110243
## 2021-03-06    Predicted R  1.112205
## 2021-03-07    Predicted R  1.110548
## 2021-03-08    Predicted R  1.126424
## 2021-03-09    Predicted R  1.132182
## 2021-03-10    Predicted R  1.163416
```

```r
ggplot(df, aes(x=time)) +
  geom_line(data = head(df, n=-h), aes(x=time, y=mean_R, color='historical'), color='black') +
  geom_line(data = tail(df, n=h), aes(x=time, y=mean_R, color='predicted'), color='darkred') +
  geom_hline(yintercept=1.0, linetype = 'dotted', color='black') +
  xlim(2020.5, 2021.2) +
  ylim(0.5, 1.85) +
  xlab("Time") +
  ylab("Estimated R")
```



1.6
1.2
0.8
Estimated R
2020.6
2020.8
2021.0
2021.2
Time