Анализ и прогнозирование временных рядов в BSTS и Prophet

Шрамова Виктория

2017-06-30

Введение

Стандартным методом анализа временных рядов является построение *ARIMA* -моделей (autoregressive integrated moving average)[1], которые являются обобщением *ARIMA*-моделей для работы с нестационарными временными рядами. Формально модель *ARIMA(p, d, q)* можно определить следующим образом:

[1] Box, George; Jenkins, Gwilym (1970). *Time Series Analysis: Forecasting and Control*. San Francisco: Holden-Day.

$$\Delta^d y_t = \gamma + \sum_{i=1}^{p} \alpha_i \Delta^d y_{t-i} + \sum_{j=1}^{q} \beta_j \varepsilon_{t-j} + \varepsilon_t,$$

где y_t — нестационарный временной ряд, ε_t — случайная ошибка (белый шум), $\gamma, \alpha_i, \beta_j$ — параметры модели (неизвестные константы, которые нужно оценить), Δ^d — оператор разности порядка d, воздействие которого на y_t делает ряд стационарным (при $d = 0$ получаем *ARMA*-модель).

Несмотря на универсальность и хорошее качество подгонки *ARIMA*- моделей, данный метод зачастую вызывает сложности интерпретации результатов из-за необходимости дифференцирования временного ряда. Также невозможно выделить и анализировать компоненты, лежащие в основе структуры исходного временного ряда (модель *ARIMA* создаёт стационарный временной ряд, удаляя тренд и сезонность изначального ряда).

В данной работе предлагается рассмотреть модели байесовских структурных временных рядов как альтернативный метод моделирования и прогнозирования временных рядов. Реализация данных моделей осуществлена в пакетах BSTS и Prophet языка программирования R, более подробно о них будет рассказано ниже. Однако стоит отметить, что оба пакета основаны на моделях байесовских структурных временных рядов и различаются лишь в том, на какие компоненты раскладывается временной ряд, и способе моделирования данных компонент. Качество прогнозов, получаемых данными методами, будет сравниваться с помощью метрики *RMSE*:

$$RMSE = \left(\frac{1}{n} \sum_{t=1}^{n} (A_t - F_t)^2 \right)^{\frac{1}{2}},$$

где A_t — истинное значение временного ряда в момент времени t; F_t — прогнозное значение временного ряда в момент времени t.

Пакет BSTS и модели структурных временных рядов

Пакет BSTS[2] предназначен для работы с моделями байесовских структур-

[2] Автор: Steven L. Scott, Google

ных временных рядов. Данные модели представляют собой модели регрессии, в которых объясняющие переменные являются функциями от времени, с возможностью указания априорного распределения переменных и их параметров. Иными словами, в байесовских структурных моделях такие ненаблюдаемые элементы, как тренд, сезонность, случайные ошибки и другие релевантные компоненты явно моделируются и анализируются. Данный метод противопоставляется философии *ARIMA*-моделей, где тренд и сезонность ряда явно не моделируются.

Базовая структурная модель может быть представлена в виде: [3]

$$y_t = \mu_t + \gamma_t + \varepsilon_t, \quad t = 1, \ldots, T \tag{1.1}$$

где μ_t — тренд, γ_t — сезонная компонента, ε_t — случайная ошибка ($\varepsilon_t \sim WN(0, \sigma_\varepsilon^2)$).

В зависимости от спецификации модели из уравнения (1.1) можно исключать отдельные компоненты или добавлять новые (например, авторегрессионную составляющую). Далее будут представлены примеры возможных спецификаций модели.

[3] A. C. Harvey and N. Shephard. *Structural time series models*. In G. Maddala, C. Rao, and H. Vinod, editors, Handbook of Statistics, vol. 11, ch. 10, p. 261–302. Elsevier, 1993.

Здесь и далее $WN(0, \sigma^2)$ будет обозначать процесс белого шума.

Локальный уровень

Модель локального уровня имеет вид:

$$y_t = \mu_t + \varepsilon_t, \quad \varepsilon_t \sim WN(0, \sigma_\varepsilon^2) \tag{1.2}$$

$$\mu_t = \mu_{t-1} + \eta_t, \quad \eta_t \sim WN(0, \sigma_\eta^2) \tag{1.3}$$

Уравнение (1.3) представляет собой случайное блуждание и задаёт локальный уровень (следующее значение уровня повторяет предыдущее с некоторой ошибкой), уравнение (1.2) задаёт временной ряд как локальный уровень с некоторой случайной ошибкой ε_t.

Априорное распределение в пакете BSTS:

Добавление локального уровня производится с помощью функции *AddLocalLevel()*, при этом предполагается, что

1) $\sigma_\eta^2 \sim IG(\alpha, \beta)$ — имеет обратное Гамма-распределение и $\eta_t | \sigma_\eta^2 \sim N(0, \sigma_\eta^2)$. Из-за того, что $\eta_t | \sigma_\eta^2 \sim N(0, \sigma_\eta^2)$, априорное распределение дисперсии ошибки σ_η^2 можно выразить через масштабированное обратное Хи-квадрат распределение (Scaled inverse chi-squared distribution): $\sigma_\eta^2 \sim \chi^2_{ScI}(\nu, \tau^2)$, причём через его параметры можно выразить параметры обратного Гамма-распределения: $\alpha = \frac{\nu\tau^2}{2}$, $\beta = \frac{\nu}{2}$. Таким образом, $\sigma_\eta^2 \sim \chi^2_{ScI}(\nu, \tau^2) \Leftrightarrow \sigma_\eta^2 \sim IG(\frac{\nu\tau^2}{2}, \frac{\nu}{2})$.

2) Начальное значение уровня имеет нормальное распределение:

$$\mu_0 \sim N(\mu, \sigma^2).$$

Параметры τ^2 и ν (соответственно, α и β) можно задать внутри функции *AddLocalLevel()* в аргументе *sigma.prior* (NULL по умолчанию), который является объектом класса *SdPrior*. Для этого при инициализации *sigma.prior* определяется числовой аргумент *sigma.guess* (для τ, принимает только положительные значения), в котором указывается априорное мнение о величине стандартного отклонения σ, и числовой аргумент *sample.size* (для ν, принимает положительные, но не обязательно целые значения), в котором указывается степень нашей уверенности в априорной величине σ (как если бы оценка априорной величины σ рассчитывалась по n =*sample.size* наблюдениям: чем больше n, тем точнее оценка и больше наша уверенность в априорной величине σ). Например, для $\tau^2 = 4$ и $\nu = 3$ (соответственно, для $\alpha = 6$, $\beta = 1.5$): sigma.prior = SdPrior(*sigma.guess* = 2, *sample.size* = 3).

Параметры μ и σ можно задать внутри функции *AddLocalLevel()* в аргументе *initial.state.prior* (NULL по умолчанию), который является объектом класса *NormalPrior*. Для этого при инициализации *initial.state.prior* определяется числовой аргумент *mu*, в котором указывается априорное мнение о величине математического ожидания μ, и числовой аргумент *sigma*, в котором указывается априорное мнение о величине стандартного отклонения σ. Например, для $\mu = 2$ и $\sigma = 1$: initial.state.prior = NormalPrior(mu = 2, sigma = 1).

Дефолтные значения аргументов функции AddLocalLevel():

```
AddLocalLevel <- function(
          state.specification = NULL,
          y,
          sigma.prior = NULL,
          initial.state.prior = NULL,
          sdy,
          initial.y)
```

```
# sdy is standard deviation of the time series
# if sdy, initial.y are ignored
sdy <- sd(y, na.rm = TRUE)
initial.y <- y[1]
# if sigma.prior = NULL
# default sample.size = .01
sigma.prior <- SdPrior(sigma.guess = .01 * sdy,
                       upper.limit = sdy)
# if initial.state.prior = NULL
initial.state.prior <- NormalPrior(mu = initial.y,
                                   sigma = sdy)
```

```
# generate random walk
rand$rand <- arima.sim(list(order = c(0,1,0)),
       n = length(rand$Year)-1)
training_cut_date <- as.Date(as.character("
       1920-01-01"),format = "%Y-%m-%d")
data_train <- rand[rand$Year <= training_cut_
       date,]
yts <- xts(data_train$rand, order.by=data_train
       $Year)
# fit BSTS with local level
ss <- AddLocalLevel(list(), yts)
model <- bsts(yts, state.specification = ss,
       niter = 500)
```

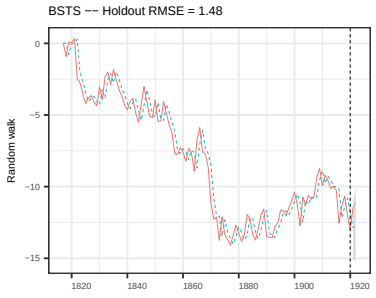


Рис. 1: Example of Local level (1.2), (1.3)

Локальный линейный тренд

Модель локального линейного тренда имеет вид:

$$y_t = \mu_t + \varepsilon_t, \quad \varepsilon_t \sim WN(0, \sigma_\varepsilon^2)$$

$$\mu_t = \mu_{t-1} + \beta_{t-1} + \eta_t, \quad \eta_t \sim WN(0, \sigma_\eta^2) \tag{1.4}$$

$$\beta_t = \beta_{t-1} + \zeta_t, \quad \zeta_t \sim WN(0, \sigma_\zeta^2) \tag{1.5}$$

Уравнения (1.4) и (1.5) задают стохастический тренд: случайные составляющие η_t и ζ_t позволяют меняться во времени уровню и наклону тренда соответственно. Уравнение ряда представляет собой стохастический тренд с некоторым случайным отклонением ε_t. Заметим, что чем больше значения дисперсий σ_η^2 и σ_ζ^2, тем сильнее случайные отклонения от тренда. Если $\sigma_\eta^2 = \sigma_\zeta^2 = 0$, то получается частный случай детерминированного линейного тренда, представимого как $\mu_t = \alpha + \beta t$ или $\mu_t = \mu_{t-1} + \beta$

Априорное распределение в пакете BSTS:

Добавление локального линейного тренда производится с помощью функции *AddLocalLinearTrend()*, при этом предполагается, что

1) $\sigma_\eta^2 \sim IG(\alpha_\eta, \beta_\eta)$, $\sigma_\zeta^2 \sim IG(\alpha_\zeta, \beta_\zeta)$ и $\eta_t|\sigma_\eta^2 \sim N(0, \sigma_\eta^2)$, $\zeta_t|\sigma_\zeta^2 \sim N(0, \sigma_\zeta^2)$.

Параметры α_η и β_η можно задать в аргументе *level.sigma.prior* (NULL по умолчанию), α_ζ и β_ζ можно задать в аргументе *slope.sigma.prior* (NULL по умолчанию).

2) Начальные значения уровня и наклона тренда имеют нормальное распределение:

$\mu_0 \sim N(\mu, \sigma_\mu^2)$, $\beta_0 \sim N(\beta, \sigma_\beta^2)$.

Параметры μ и σ_μ можно задать в аргументе *initial.level.prior* (NULL по умолчанию), β и σ_β можно задать в аргументе *initial.slope.prior* (NULL по умолчанию).

Дефолтные значения аргументов функции AddLocalLinearTrend():

```
# if sdy, initial.y are ignored
sdy <- sd(y, na.rm = TRUE)
initial.y <- y[1]
# if level.sigma.prior = NULL
# default sample.size = .01
level.sigma.prior <- SdPrior(.01 * sdy, upper.limit = sdy)
# if slope.sigma.prior = NULL
slope.sigma.prior <- SdPrior(.01 * sdy, upper.limit = sdy)
#if initial.level.prior = NULL
initial.level.prior <- NormalPrior(initial.y, sdy)
#if initial.slope.prior = NULL
initial.slope.prior <- NormalPrior(0, sdy)
```

Полулокальный линейный тренд

Данная модель похожа на модель локального линейного тренда, однако более эффективна для долгосрочного прогнозирования. В ней предполагается, что уровень тренда изменяется согласно процессу случайного блуждания (как и для локального линейного тренда), но наклон тренда изменяется согласно *AR(1)*-процессу, центрированному относительно некоторого ненулевого значения D (долгосрочный наклон тренда). таким образом, модель имеет вид:

$$y_t = \mu_t + \varepsilon_t, \;\; \varepsilon_t \sim WN(0, \sigma_\varepsilon^2)$$

$$\mu_t = \mu_{t-1} + \delta_{t-1} + \eta_t, \;\; \eta_t \sim WN(0, \sigma_\eta^2) \tag{1.6}$$

$$\delta_t = D + \phi(\delta_{t-1} - D) + \zeta_t, \;\; \zeta_t \sim WN(0, \sigma_\zeta^2), \tag{1.7}$$

где ϕ — коэффициент при *AR(1)*-процессе.

Априорное распределение в пакете BSTS:

Добавление полулокального линейного тренда производится с помощью функции *AddSemilocalLinearTrend()*, при этом предполагается, что

1) $\sigma_\eta^2 \sim IG(\alpha_\eta, \beta_\eta)$, $\sigma_\zeta^2 \sim IG(\alpha_\zeta, \beta_\zeta)$ и $\eta_t|\sigma_\eta^2 \sim N(0, \sigma_\eta^2)$, $\zeta_t|\sigma_\zeta^2 \sim N(0, \sigma_\zeta^2)$.

2) Начальные значения уровня и наклона тренда имеют нормальное распределение:

   $\mu_0 \sim N(\mu, \sigma_\mu^2)$, $\delta_0 \sim N(\delta, \sigma_\delta^2)$.

3) Долгосрочный наклон тренда D имеет нормальное распределение: $D \sim N(d, \sigma_d^2)$.

```
AddLocalLinearTrend <- function (
          state.specification = NULL,
          y,
          level.sigma.prior = NULL,
          slope.sigma.prior = NULL,
          initial.level.prior = NULL,
          initial.slope.prior = NULL,
          sdy,
          initial.y)
```

```
# aust - export in Australia
# make train data
training_cut_date <- as.Date(as.character("
      1993-10-01"),format = "%Y-%m-%d")
data_train <- aust[aust$Quarter <= training_cut
      _date,]
yts <- xts(data_train$exports, order.by=data_
      train$Quarter)
# fit BSTS with trend only
ss <- AddLocalLinearTrend(list(), yts)
model <- bsts(yts, state.specification = ss,
      niter = 500)
```

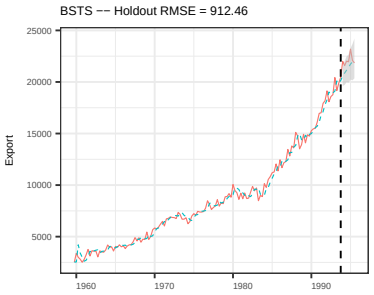


Рис. 2: Example of Local linear trend (1.4), (1.5)

(источник данных)

Параметры α_η и β_η можно задать в аргументе *level.sigma.prior* (NULL по умолчанию), α_ζ и β_ζ можно задать в аргументе *slope.sigma.prior* (NULL по умолчанию).

Параметры μ и σ_μ можно задать в аргументе *initial.level.prior* (NULL по умолчанию), δ и σ_δ можно задать в аргументе *initial.slope.prior* (NULL по умолчанию).

Параметры d и σ_d можно задать в аргументе *slope.mean.prior* (NULL по умолчанию).

4) Коэффициент *AR(1)*- процесса ϕ имеет усечённое нормальное распределение, то есть $\phi \sim N(\phi_0, \sigma_\phi^2)$ на интервале (a, b). Если априорно ограничить значения ϕ интервалом $(-1, 1)$, то наклон тренда будет демонстрировать краткосрочные стационарные отклонения от долгосрочного наклона D.

Дефолтные значения аргументов функции AddSemilocalLinearTrend():

Значения аргументов, совпадающих с аргументами функции *AddlocalLinearTrend()*, по умолчанию присваиваются такими же, для остальных аргументов:

```
# if sdy, initial.y are ignored
sdy <- sd(y, na.rm = TRUE)
initial.y <- y[1]
# sdy is standard deviation of the time series
# if slope.mean.prior = NULL
slope.mean.prior <- NormalPrior(0, sdy)
# if slope.ar1.prior = NULL
slope.ar1.prior <- Ar1CoefficientPrior(mu = 0, sigma = 1,
force.stationary = TRUE, force.positive = FALSE,
                                       initial.value = mu)
```

Указанные параметры можно задать внутри функции *AddSemilocalLinearTrend()* в аргументе *slope.ar1.prior* (NULL по умолчанию), который является объектом класса *Ar1CoefficientPrior*: при инициализации *slope.ar1.prior* ϕ_0 задаётся через аргумент *mu* (*mu* = 0 по умолчанию), σ_ϕ задаётся через аргумент *sigma* (*sigma* = 1 по умолчанию), ограничение ϕ интервалом $(-1, 1)$ задаётся с помощью *force.stationary* = TRUE (TRUE по умолчанию), а чтобы ϕ принимал только неотрицательные значения, нужно присвоить аргументу *force.positive* значение TRUE (FALSE по умолчанию). Например, для $\phi \in (0, 1)$, $\phi_0 = 0.5$, $\sigma_\phi^2 = 0.01$: slope.ar1.prior = Ar1CoefficientPrior(mu = 0.5, sigma = 0.1, force.positive = TRUE).

```
AddSemilocalLinearTrend <- function (
          state.specification = list(),
          y = NULL,
          level.sigma.prior = NULL,
          slope.mean.prior = NULL,
          slope.ar1.prior = NULL,
          slope.sigma.prior = NULL,
          initial.level.prior = NULL,
          initial.slope.prior = NULL,
          sdy = NULL,
          initial.y = NULL)
```

Сезонность

Модель детерминированной сезонности предполагает, что сезонные эффекты вместе нивелируют друг друга, то есть в сумме равны нулю. Однако можно позволить сезонным эффектам изменяться во времени, сделав их сумму равной случайному отклонению ω_t с нулевым математическим ожиданием и дисперсией σ_ω^2. Тогда, если число сезонов равно s,

$$\sum_{j=0}^{s-1} \gamma_{t-j} = \omega_t \Leftrightarrow \gamma_t = -\sum_{j=1}^{s-1} \gamma_{t-j} + \omega_t \tag{1.8}$$

структурная модель временного ряда представима в виде:

$$y_t = \gamma_t + \varepsilon_t, \quad \varepsilon_t \sim WN(0, \sigma_\varepsilon^2) \tag{1.9}$$

$$\gamma_t = -\sum_{j=1}^{s-1} \gamma_{t-j} + \omega_t, \quad \omega_t \sim WN(0, \sigma_\omega^2)$$

где уравнение (1.9) моделирует временной ряд как сезонную компоненту с некоторой случайной ошибкой ε_t.

Априорное распределение в пакете BSTS:

Добавление сезонной компоненты производится с помощью функции *AddSeasonal()*, при этом предполагается, что

1) $\sigma_\omega^2 \sim IG(\alpha_\omega, \beta_\omega)$ и $\omega_t | \sigma_\omega^2 \sim N(0, \sigma_\omega^2)$.

```
# aust - export in Australia
# make train data
training_cut_date <- as.Date(as.character("
      1993-10-01"),format = "%Y-%m-%d")
data_train <- aust[aust$Quarter <= training_cut
      _date,]
yts <- xts(data_train$exports, order.by=data_
      train$Quarter)
# fit BSTS with trend only
ss <- AddSemilocalLinearTrend(list(), yts)
model <- bsts(yts, state.specification = ss,
      niter = 500)
```

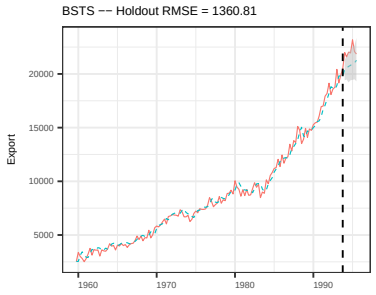


Рис. 3: Example of Semilocal linear trend (1.6), (1.7)

(источник данных)

Моделируемое количество сезонов можно задать в аргументе *nseasons*.

2) Начальные значения вектора сезонной компоненты (первые s значений вектора γ_t) имеют нормальное распределение:

$\gamma_0, \dots, \gamma_{s-1} \sim N(\gamma, \sigma_\gamma^2)$.

Дефолтные значения аргументов функции AddSeasonal():

```
# if number of time periods each season is to last is ignored
season.duration = 1
# if sdy is ignored
sdy <- sd(y, na.rm = TRUE)
# sdy is standard deviation of the time series
# if sigma.prior = NULL
sigma.prior <- SdPrior(.01 * sdy, upper.limit = sdy)
# if initial.state.prior = NULL
initial.state.prior <- NormalPrior(0, sdy)
```

Сезонность также может быть задана через набор тригонометрических функций (а именно комбинаций sin и cos) с сезонной частотой (измеряется в радианах) $\lambda_j = \frac{2\pi j}{s}, j = 1, \dots, [s/2]$, где $[s/2] = s/2$, если s чётное и $[s/2] = (s-1)/2$, если s нечётное. Тогда сезонный эффект в момент времени t равен:

$$\gamma_t = \sum_{j=1}^{[s/2]} (\gamma_j \cos \lambda_j t + \gamma_j^* \sin \lambda_j t) \tag{1.10}$$

Так же как и в случае линейного тренда, уравнение сезонности (1.10) может быть задано рекурсивно, тогда структурная модель будет выглядеть следующим образом:

$$y_t = \gamma_t + \varepsilon_t, \quad \varepsilon_t \sim WN(0, \sigma_\varepsilon^2)$$

$$\gamma_t = \sum_{j=1}^{[s/2]} \gamma_{j,t}, \tag{1.11}$$

$$\begin{pmatrix} \gamma_{j,t} \\ \gamma_{j,t}^* \end{pmatrix} = \begin{pmatrix} \cos \lambda_j & \sin \lambda_j \\ -\sin \lambda_j & \cos \lambda_j \end{pmatrix} \begin{pmatrix} \gamma_{j,t-1} \\ \gamma_{j,t-1}^* \end{pmatrix} + \begin{pmatrix} \omega_{j,t} \\ \omega_{j,t}^* \end{pmatrix} \tag{1.12}$$

где $\omega_{j,t}$ и $\omega_{j,t}^*, j = 1, \dots, [s/2]$ — некоррелированные процессы белого шума с нулевым математическим ожиданием и общей дисперсией σ_ω^2. Компонента $\gamma_{j,t}^*$ необходима для построения стохастической модели и не имеет важной интерпретации.

Априорное распределение в пакете BSTS:

Добавление тригонометрической сезонной компоненты производится с помощью функции *AddTrig()*, при этом предполагается, что

Параметры α_ω и β_ω можно задать в аргументе *sigma.prior* (NULL по умолчанию).

Параметры γ и σ_γ можно задать в аргументе *initial.state.prior* (NULL по умолчанию).

```
AddSeasonal <- function(
          state.specification,
          y,
          nseasons,
          season.duration = 1,
          sigma.prior = NULL,
          initial.state.prior = NULL,
          sdy)
```

```
# lynx - Number of lynx trapped
# make train data
training_cut_date <- as.Date(as.character("
      1924-01-01"),format = "%Y-%m-%d")
data_train <- lynx[lynx$Year <= training_cut_
      date,]
yts <- xts(data_train$lynx, order.by=data_train
      $Year)
# fit BSTS with local level and seasonal
      component
ss <- AddLocalLevel(list(), yts)
ss <- AddSeasonal(ss, yts,nseasons=10)
model <- bsts(yts, state.specification = ss,
      niter = 500)
```

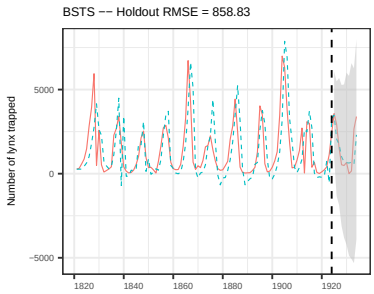


Рис. 4: Example of Seasonal component (1.8), (1.9) with Local level

(источник данных)

Период колебаний, т.е. число периодов времени, необходимое для повторения наибольшего цикла, задаётся в переменной *period* (например, для годового цикла *period* = 12). Частоты колебаний λ_j, т.е. число повторений каждой циклической компоненты в течение периода, задаётся в параметре *frequencies* в виде вектора положительных чисел.

```
AddTrig <- function(
          state.specification = NULL,
          y,
          period,
          frequencies,
          sigma.prior = NULL,
          initial.state.prior = NULL,
          sdy)
```

Параметры α_{γ_j} и β_{γ_j} можно задать в аргументе *sigma.prior* (NULL по умолчанию).

1) $\sigma^2_{\gamma_j} \sim IG(\alpha_{\gamma_j}, \beta_{\gamma_j})$ и $\gamma^*_j|\sigma^2_{\gamma_j} \sim N(0, \sigma^2_{\gamma_j})$.

2) Начальные значения вектора коэффициентов γ^*_j (первые $[s/2]$ значений вектора γ^*_j) имеют многомерное нормальное распределение:

$\gamma^*_0, \ldots, \gamma^*_{[s/2]} \sim MN(\gamma, \Sigma_\gamma)$.

Дефолтные значения аргументов функции AddTrig():

```
# if sdy is ignored
sdy <- sd(y, na.rm = TRUE)
# if sigma.prior = NULL
sigma.prior <- SdPrior(.01 * sdy, upper.limit = sdy)
# if initial.state.prior = NULL
dimension <- 2 * length(frequencies)
initial.state.prior <- MvnPrior(
        mean = rep(0, dimension),
        variance = diag(rep(sdy, dimension)^2))
```

Авторегрессионная компонента

Добавление авторегрессионной составляющей аналогично рассмотрению *AR(p)*-процесса:

$$y_t = \sum_{i=1}^{p} \alpha_i y_{t-i} + \varepsilon_t, \quad \varepsilon_t \sim WN(0, \sigma^2_\varepsilon)$$

в котором текущее значение временного ряда линейно зависит от p предыдущих значений (лагов) этого ряда.

Априорное распределение в пакете BSTS:

Добавление авторегрессинной компоненты производится с помощью функции *AddAr()*, при этом предполагается, что

1) $\sigma^2_\varepsilon \sim IG(\alpha_\varepsilon, \beta_\varepsilon)$ и $\varepsilon_t|\sigma^2_\varepsilon \sim N(0, \sigma^2_\varepsilon)$.

2) Вектор начальных значений временного ряда (первые p значений вектора y_t) имеет многомерное нормальное распределение: $(y_0, \ldots, y_{p-1})^T \sim MN(\mu, \Sigma)$.

Дефолтные значения аргументов функции AddAr():

```
# if number of lags is ignored
lags = 1
# if sdy is ignored
sdy <- sd(y, na.rm = TRUE)
# sdy is standard deviation of the time series
# if sigma.prior = NULL
```

Параметры γ и Σ_γ можно задать в аргументе *initial.state.prior* (NULL по умолчанию).

```
# mean temperature, Fisher River
data_train <- temp[temp$Date <= training_cut_
      date,]
yts <- xts(data_train$temp, order.by=data_train
      $Date)
# fit BSTS with trig only
ss <- AddTrig(list(), yts,period=730,
      frequencies = 1)
model <- bsts(yts, state.specification = ss,
      niter = 500)
# BSTS predictions
pred <- predict(model, horizon = 365, burn =
      burn, quantiles = c(.025, .975))
# actual vs fitted data, abs values
d2 <- data.frame(abs(c(...))...)
```

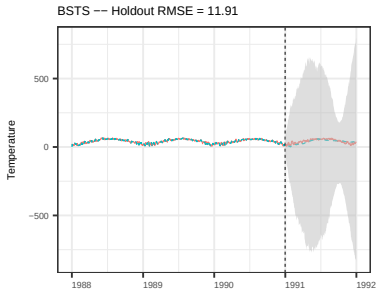


Рис. 5: Example of Trigonometric Seasonal component (1.11), (1.12)

(источник данных)

На графиках 4 и 5 видны существенные отклонения прогнозов от истинных значений, широкие предиктивные интервалы. Из-за особенностей построения тригонометрического тренда (как суммы sin и cos) для примера на графике 5 нужно увеличить период колебаний в 2 раза и использовать прогнозы, взятые по модулю. В данном примере использовать BSTS сложнее, чем Prophet (будет показано далее), необходимо дополнительно учитывать особенность формулы для моделирования сезонности временного ряда.
Моделируемое количество лагов p можно задать в аргументе *lags*.

Параметры α_ε и β_ε можно задать в аргументе *sigma.prior* (NULL по умолчанию).

Параметры μ (математическое ожидание) и Σ (ковариационная матрица) можно задать внутри функции *AddAr()* в аргументе *initial.state.prior* (NULL по умолчанию), который является объектом класса *MvnPrior*: при инициализации *initial.state.prior* μ задаётся через аргумент *mean*, Σ задаётся через аргумент *variance*.

Например, для *AR(2)*-процесса и

$$\mu = \begin{pmatrix} 1 \\ 2 \end{pmatrix}, \ \Sigma = \begin{pmatrix} 1 & 1 \\ 1 & 9 \end{pmatrix}$$

initial.state.prior = MvnPrior(mean=c(1,2), variance=matrix(c(1, 1, 1, 9), nrow=2, ncol=2)).

```
sigma.prior <- SdPrior(.01 * sdy)
# if initial.state.prior = NULL
# the stationary distribution of the AR(p) process
# will be used as the initial state distribution.
```

```
AddAr <- function(
          state.specification,
          y,
          lags = 1,
          sigma.prior = NULL,
          initial.state.prior = NULL,
          sdy)
```

```
residual.sd <- .001
# Actual values of the AR coefficients
true.phi <- c(-.7, .3, .15)
ar <- arima.sim(model = list(ar = true.phi),
                n = 100, sd = 3)
# Layer some noise on top of the AR process.
y <- ar + rnorm(n, 0, residual.sd)
ss <- AddAr(list(), lags = 3, sigma.prior =
       SdPrior(3.0, 1.0))
# Fit the model with knowledge with residual.sd
# essentially fixed at the true value.
model <- bsts(y, state.specification=ss, niter
       = 500,
              prior = SdPrior(residual.sd,
                      100000))
```

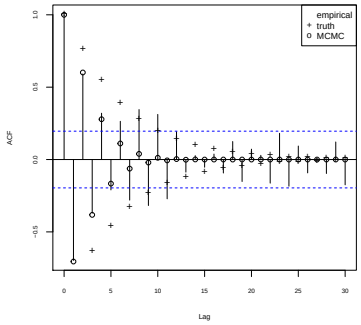


Рис. 6: Example of AR-component (L.Scott)

Праздники

Праздники также оказывают значительное влияние на динамику временных рядов (например, продажи, потребление, цены). Праздничные дни зачастую не являются строго периодичными, что создаёт проблему для их моделирования с помощью сезонной компоненты (например, День благодарения в США отмечается в четвертый четверг ноября, поэтому дата этого праздника не будет одинаковой для каждого года). Поскольку влияние конкретного праздника на динамику временных рядов часто бывает одинаковым из года в год, включение этой компоненты является важным для построения качественных прогнозов.

Пакет BSTS также позволяет добавлять модель случайного блуждания для праздничных дней к структурной модели. Получившаяся модель предполагает, что каждый рабочий день между праздниками смещается в календаре относительно своей даты в прошлом году в соответствии со случайным блужданием.

Априорное распределение в пакете BSTS:

Добавление эффекта праздников может производиться разными функциями в зависимости от данных, например, с помощью функции *AddNamedHolidays()*, при этом предполагается, что

1) Стандартное отклонение приращений процесса случайного блуждания имеет обратное Гамма-распределение: $\sigma^2 \sim IG(\alpha, \beta)$.
2) Начальные значения компоненты праздников (количество которых равно указанному количеству праздников в году) имеют нормальное распределение.

Параметры α и β можно задать в аргументе *sigma.prior* (NULL по умолчанию).

Параметры которого можно задать в аргументе *initial.state.prior* (NULL по умолчанию).

Дефолтные значения аргументов функции AddNamedHolidays():

```
# NamedHolidays() returns a vector with full names of holidays
# if list of named holidays is ignored
named.holidays <- NamedHolidays()
# if sdy is ignored
sdy <- sd(y, na.rm = TRUE)
# if sigma.prior = NULL
sigma.prior <- SdPrior(.01 * sdy, upper.limit = sdy)
# if initial.state.prior = NULL
NormalPrior(0, sdy)
# time0 can be ignored if y is a zoo or xts object
```

```
AddNamedHolidays <- function(
          state.specification = NULL,
          named.holidays = NamedHolidays(),
          y, sigma.prior = NULL,
          initial.state.prior = NULL,
          sdy = sd(as.numeric(y),
          na.rm = TRUE),time0 = NULL,
          days.before = 1,
          days.after = 1)
```

```
# then time0 is time of the first observation
time0 <- .SetTimeZero(time0, y)
# days.before is the number of days the influence of the named
# holidays extends prior to the actual holiday. If ignored
days.before <- 1
#  days.after is the number of days the influence of the named
# holidays extends after the actual holiday. If ignored
days.after <- 1
```

Комбинации

В зависимости от задачи в модель (1.1) можно включать любые комбинации описанных выше компонент, задавая спецификацию временного ряда. Например, объединив (1.3) и (1.8) (или (1.3) и (1.11)), получим модель *локального уровня и сезонности*:

$$y_t = \mu_t + \gamma_t + \varepsilon_t, \quad \varepsilon_t \sim WN(0, \sigma_\varepsilon^2), \tag{1.13}$$

с μ_t из (1.3) и γ_t из (1.8) или (1.11)

Из этой модели легко получить *модель локального линейного тренда и сезонности*, взяв μ_t из (1.4):

$$y_t = \mu_t + \gamma_t + \varepsilon_t, \quad \varepsilon_t \sim WN(0, \sigma_\varepsilon^2), \tag{1.14}$$

с μ_t из (1.4) и γ_t из (1.8) или (1.11)

Если же взять μ_t из (1.6), можно получить *модель полулокального линейного тренда и сезонности*:

$$y_t = \mu_t + \gamma_t + \varepsilon_t, \quad \varepsilon_t \sim WN(0, \sigma_\varepsilon^2), \tag{1.15}$$

с μ_t из (1.6) и γ_t из (1.8) или (1.11)

Аналогично в (1.1) можно добавить авторегрессионную компоненту или эффект праздников в зависимости от предполагаемой структуры временного ряда (и получив, например, *модель локального линейного тренда, сезонности и праздников*).

Для этого в спецификацию модели нужно последовательно добавить локальный уровень и сезонность:

```
data(AirPassengers)
y <- log(AirPassengers)
ss <- AddLocalLevel(list(), y)
ss <- AddSeasonal(ss, y, nseasons = 12)
# Fitting the model
model <- bsts(y, state.specification = ss,
       niter = 500)
```

Для этого в спецификацию модели нужно последовательно добавить локальный линейный тренд и сезонность:

```
data(AirPassengers)
y <- log(AirPassengers)
ss <- AddLocalLinearTrend(list(),y)
ss <- AddSeasonal(ss,y,nseasons = 12)
# Fitting the model
model <- bsts(y, state.specification = ss,
       niter = 500)
```

Для этого в спецификацию модели нужно последовательно добавить полулокальный линейный тренд и сезонность:

```
data(AirPassengers)
y <- log(AirPassengers)
ss <- AddSemilocalLinearTrend(list(),y)
ss <- AddSeasonal(ss, y, nseasons = 12)
# Fitting the model
model <- bsts(y, state.specification = ss,
       niter = 500)
```

Пакет Prophet и модели структурных временных рядов

Пакет Prophet[4] создан для работы с моделями байесовских структурных временных рядов.[5]. В своей модели авторы предполагают, что временной ряд представим в виде суммы трёх компонент: тренда, сезонности и праздников, динамика которых может меняться со временем. Данная спецификация временного ряда идейно очень схожа с пакетом BSTS в том плане, что: 1) исходный ряд представляется в виде суммы ненаблюдаемых объясняющих компонент и 2) можно задавать априорное распределение для используемых компонент и параметров. Однако существенной особенностью модели Prophet

[4] Авторы: Sean Taylor, Ben Letham, Facebook

[5] A. Harvey and S. Peters. *Estimation procedures for structural time series models.* Journal of Forecasting, 9:89–108, 1990.

является довольно сложная компонента, отвечающая за тренд, которая (в отличие от локального и полулокального тренда в BSTS) позволяет задавать количество и моменты изменений тренда, а также допускает возможность скачков.

Базовая модель структурных временных рядов имеет вид: [6]

$$y(t) = g(t) + s(t) + h(t) + \varepsilon_t, \tag{2.1}$$

где $g(t)$ является функцией тренда, которая моделирует непериодические изменения в значениях временного ряда, $s(t)$ отражает сезонные изменения, $h(t)$ моделирует эффекты праздников (которые могут происходить нерегулярно и длиться более одного дня), ε_t является случайной ошибкой и имеет нормальное распределение.

Модель (2.1) схожа с обобщёнными аддитивными моделями (GAM)[7] в том смысле, что представляет собой регрессию, в которой к объясняющим переменным могут быть применены нелинейные сглаживающие функции. В модели (2.1) используется время в качестве единственного регрессора, к которому может быть применено сразу несколько нелинейных функций. В итоге, по аналогии с (1.1), временной ряд раскладывается на сумму нескольких компонент, которые могут быть явно извлечены и проанализированы, однако задача прогнозирования сводится к нахождению наиболее похожей на $y(t)$ кривой, зависящей от t.

Стоит отметить, что теоретическая модель для пакета Prophet изначально была разработана для прогнозирования различных показателей социальной сети Facebook (например, числа пользователей), что отражается в специфичном моделировании отдельных компонент (2.1). Однако получившаяся модель показывает хорошие результаты в работе с другими показателями делового сектора и может быть использована для решения более общих задач прогнозирования.

Тренд

Для построения тренда (в первоначальной задаче - роста числа пользователей Facebook) используется модель, схожая моделью роста населения, где происходит нелинейный рост, ограниченный сверху естественными условиями (например, число пользователей Facebook ограничено числом людей, имеющим доступ к Интернету). Тренд в (2.1) задаётся формулой:

$$g(t) = \frac{C(t)}{1 + e^{-k(t-b)}}, \tag{2.2}$$

где $C(t)$ — ограничение сверху, k — темп роста, b — параметр смещения.

$C(t)$ в формуле (2.2) — естественное ограничение среды, меняющееся со временем, которое также является максимально возможным значением функции $g(t)$. Наиболее простыми видами $C(t)$ являются постоянное ограничение ($C(t) = K$) и линейное ограничение ($C(t) = Mt + K$), хотя спецификация зависит от конкретной задачи и знаний области.

[6] Sean J. Taylor and Benjamin Letham. *Forecasting at Scale*. Facebook research blog, 2017.

[7] Hastie, T. J.; Tibshirani, R. J. (1990). Generalized Additive Models. Chapman & Hall/CRC.

Необходимо также учесть, что темп роста k может меняться со временем — для этого в модель явным образом введём S моментов времени $s_j, \quad j = 1, \ldots, S$, в которых k позволено меняться (заметим, что при введении дискретного числа точек, в которых может изменяться параметр k, мы получим кусочно-заданную функцию тренда). Определим вектор $\delta \in R^S$, где δ_j — это изменение темпа роста, произошедшее в момент времени s_j. Тогда темп роста в любой момент времени t можно определить как сумму $k + \sum_{j:t \geq s_j} \delta_j$ или $k + a(t)^T \delta$, где $a(t) \in \{0, 1\}^S$ и

$$a_j(t) = \begin{cases} 1, & t \geq s_j \\ 0, & t < s_j \end{cases}$$

При изменении темпа роста k необходимо также пересчитывать значение b для предотвращения разрывов в функции тренда. Величина корректировки в момент времени s_j рассчитывается по формуле:

$$\gamma_j = \left(s_j - b - \sum_{l<j} \gamma_l \right) \left(1 - \frac{k + \sum_{l<j} \delta_l}{k + \sum_{l \leq j} \delta_l} \right)$$

Таким образом, нелинейная кусочно-заданная функция тренда имеет вид:

$$g(t) = \frac{C(t)}{1 + e^{-(k + a(t)^T \delta)(t - (b + a(t)^T \gamma))}} \tag{2.3}$$

На графиках 7 и 8 видно, что в зависимости от спецификации $C(t)$ и распределения δ меняются поведение тренда (2.3) и ширина предиктивных интервалов.

Часто в целях создания более экономной модели можно использовать кусочно-постоянную функцию для изменения темпа роста k. В этом случае тренд будет иметь вид:

$$g(t) = (k + a(t)^T \delta)t + (b + a(t)^T \gamma), \tag{2.4}$$

где как и раньше k означает темп роста, δ — изменение темпа роста, b — параметр смещения, γ необходимо для непрерывности функции тренда.

Сезонность

Для аппроксимации сезонной компоненты как набора периодических функций от времени t авторы предлагают использовать ряды Фурье: пусть P — это ожидаемый период временного ряда (например, $P = 7$ для еженедельных данных, $P = 30$ для ежемесячных данных), $2N$ — количество используемых членов ряда Фурье (выбор большего N позволит приблизить более сложные периодические функции, однако может привести к переобучению). Тогда аппроксимация сезонности будет выглядеть следующим образом:

$$s(t) = \sum_{n=-N}^{N} c_n e^{i \frac{2\pi n t}{P}} \tag{2.5}$$

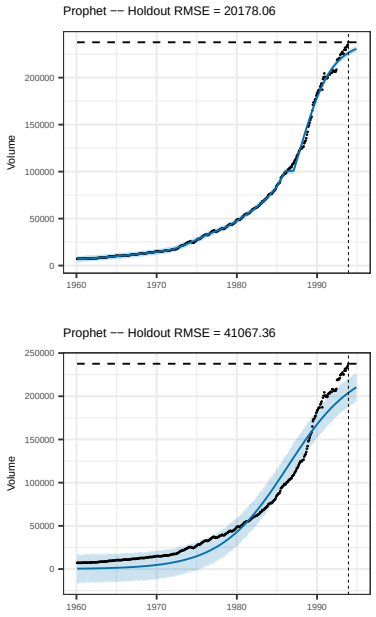


Рис. 7: Trend (2.3) with $C(t) = K, \quad \delta \sim DExp(0, 5)$ and $\delta \sim DExp(0, 0.02)$.

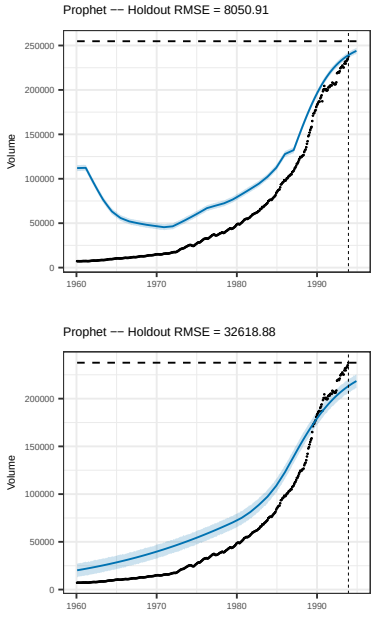


Рис. 8: Trend (2.3) with $C(t) = Mt + K, \quad \delta \sim DExp(0, 5)$ and $\delta \sim DExp(0, 0.02)$.

(источник данных)

На основе (2.1) можно построить вектор сезонов для каждого значения t, например, для недельной сезонности ($P = 7$) и $N = 3$ получим

$$X(t) = \left(e^{i\frac{2\pi(-3)t}{7}}, \dots, e^{i\frac{2\pi(3)t}{7}}\right)$$

Для добавления сезонной компоненты в модель (2.1) также потребуется оценить параметры $c_n,\ n = -N, \dots, N$, которые можно объединить в один вектор β. Тогда выражение (2.5) можно переписать в виде произведения вектора сезонов на вектор неизвестных параметров:

$$s(t) = X(t)\beta, \tag{2.6}$$

задав при этом априорное распределение вектора β (например, $\beta \sim N(0, \sigma^2)$).

На графике 9 изображены результаты применения (2.6) на данных с выраженной годовой сезонностью ($P = 365.25$ из-за годовой периодичности данных, $2N = 20$ — количество используемых членов ряда Фурье для аппроксимации, $\beta \sim N(0, 10^2)$ — априорное распределение параметров c_n).

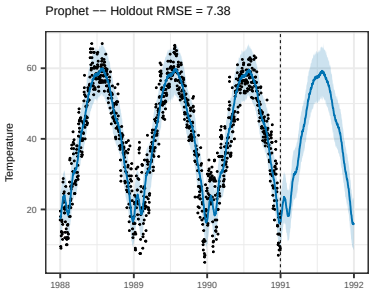


Рис. 9: Seasonality (2.6) with $N = 10$ and $P = 365.25$, $\beta \sim N(0, 10^2)$

(источник данных)

Праздники

Пакет Prophet также даёт возможность добавлять эффект праздников во временной ряд, позволяя исследователю включать конкретные прошлые или будущие события в анализ. Важно учитывать не только международные праздники (например, Рождество), но и характерные для рассматриваемой страны события (например, День благодарения в США).

Эффекты праздников предполагаются независимыми, что позволяет легко включить их рассмотрение в модель (2.1). Для каждого праздника $i,\ i = 1, \dots, L$ определим D_i как множество прошлых и будущих календарных дат этого праздника. Тогда можно задать функцию индикатора $I(t \in D_i)$, показывающую, попал ли момент времени t на праздник i, и присвоить каждому празднику i оказываемый эффект на прогноз k_i. Получим суммарный эффект праздников на момент t:

$$h(t) = \sum_{i=1}^{L} k_i I(t \in D_i) \tag{2.7}$$

Выражение (2.7) можно обобщить, добавив в него эффект от нескольких дней вблизи конкретного праздника (это связано с тем, что люди ведут себя по-разному до наступления праздника и после его окончания). Для того чтобы учесть этот факт, в уравнение (2.7) могут быть добавлены дополнительные эффекты (подобно k_i) для дней, окружающих праздник (например, три дня до D_i и два дня после D_i для праздника i).

Подобно сезонности, выражение (2.7) можно переписать в матричном виде:

$$h(t) = Z(t)k, \tag{2.8}$$

где $Z(t)$ — вектор индикаторов праздника для каждого значения t:

$$Z(t) = (I(t \in D_1), \ldots, I(t \in D_L)),$$

$k = (k_1, \ldots, k_L)^T$ — вектор оказываемых эффектов, для которого можно задать априорное распределение(например, $k \sim N(0, \nu^2)$).

Реализация в пакете Prophet

Пакет Prophet реализован на двух языках программирования: R и Python. Для обучения моделей и построения прогнозов Prophet использует библиотеку Stan языка программирования C++, которая автоматически вызывается при наборе команд из пакета Prophet. Как и в BSTS, пользователю достаточно работать с готовыми функциями пакета, подавая в качестве аргументов данные и необходимые параметры теоретических моделей. Отличием от BSTS является то, что для задания всех компонент временного ряда используется одна функция *prophet()*, в аргументах которой можно указать вид компоненты и необходимые для её спецификации параметры.

Так, аргументы функции *prophet()* для моделирования тренда имеют вид:

1) *df* — исследуемый временной ряд, который обязан содержать наблюдения *y* и время наблюдения *ds*. Если функция тренда нелинейная, то *df* должен содержать ограничения сверху *cap* ($C(t)$ в модели) в каждый момент времени *ds*. В Stan из аргумента *df* подаётся количество наблюдений *int T*, время наблюдений *vector[T] t*, ограничения сверху *vector[T] cap*, временной ряд *vector[T] y*.

2) *growth* — функция тренда, может принимать два значения: *"linear"* (по умолчанию) задаёт линейную функцию тренда вида (2.4), *"logistic"* задаёт нелинейную (логистическую) функцию тренда вида (2.3). Указание значения аргумента *growth* отобразится на выборе одной из двух моделей тренда в Stan, в которую затем будут подаваться параметры для тренда, сезонности и праздников.

3) *changepoints* — вектор из моментов времени s_j, $j = 1, \ldots, S$, в которых может изменяться темп роста тренда. По умолчанию *changepoints* = NULL и вектор подбирается автоматически. В Stan данная переменная подаётся как *real t_change[S]*.

4) *n.changepoints* — количество моментов времени S, в которых может изменяться темп роста тренда. Если указан аргумент *changepoints*, то *n.changepoints* автоматически определяется как длина указанного вектора. Если аргумент *changepoints* не указан, то *n.changepoints* подбирается автоматически из первых 80% наблюдений *df*. По умолчанию *n.changepoints* = 25. В Stan данная переменная подаётся как *int S*.

Функция *prophet()* со значением аргументов по умолчанию:

```
prophet(df = df, growth = "linear",
changepoints = NULL, n.changepoints = 25,
yearly.seasonality = "auto",
weekly.seasonality = "auto",
holidays = NULL,
seasonality.prior.scale = 10,
holidays.prior.scale = 10,
changepoint.prior.scale = 0.05,
mcmc.samples = 0, interval.width = 0.8,
uncertainty.samples = 1000, fit = TRUE, ...)
```

Входные данные модели линейного тренда в Stan:

```
data {
  int T;                // Sample size
  int<lower=1> K;       // Number of seasonal
                        // vectors
  vector[T] t;          // Day
  vector[T] y;          // Time-series
  int S;                // Number of changepoints
  matrix[T, S] A;       // Split indicators
  real t_change[S];     // Index of changepoints
  matrix[T,K] X;        // season vectors
  real<lower=0> sigma; // scale on seasonality
                        // prior
  real<lower=0> tau;    // scale on changepoints
}                       // prior
```

Входные данные модели нелинейного тренда в Stan:

```
data {
  int T;                // Sample size
  int<lower=1> K;       // Number of seasonal
                        // vectors
  vector[T] t;          // Day
  vector[T] cap;        // Capacities
  vector[T] y;          // Time-series
  int S;                // Number of changepoints
  matrix[T, S] A;       // Split indicators
  real t_change[S];     // Index of changepoints
  matrix[T,K] X;        // season vectors
  real<lower=0> sigma; // scale on seasonality
                        // prior
  real<lower=0> tau;    // scale on changepoints
}                       // prior
```

Априорные распределения параметров в Stan:

```
model {
  # priors
  k ~ normal(0, 5);
  m ~ normal(0, 5);
  delta ~ double_exponential(0, tau);
  sigma_obs ~ normal(0, 0.5);
  beta ~ normal(0, sigma);

  # Non-linear Likelihood
  y ~ normal(cap ./ (1 + exp(-(k + A * delta) .
        * (t - (m + A * gamma)))) + X * beta,
        sigma_obs)

  # Linear Likelihood
  y ~ normal((k + A * delta) .* t + (m + A *
        gamma) + X * beta, sigma_obs)
}
```

5) *changepoint.prior.scale* — числовой параметр (τ), контролирующий автоматический подбор *changepoints*: чем больше значение параметра, тем больше гибкости в подборе роста тренда в *changepoints*. В Stan данная переменная подаётся как *real<lower=0> tau*, от которой зависит распределение вектора δ (вектор изменений темпа роста тренда): $\delta \sim DExp(0, tau)$, где *DExp(0, tau)* — двойное экспоненциальное распределение (распределение Лапласа). По умолчанию *changepoint.prior.scale* = 0.05.

Аргументы функции *prophet()* для моделирования сезонности имеют вид:

6) *yearly.seasonality* — аргумент, с помощью которого задаётся наличие годовой сезонности временного ряда. Принимает три возможных значения: TRUE — сезонность есть, FALSE — сезонности нет или 'auto' — функция сама определяет наличие сезонности (при количестве наблюдений не меньше 730 функция будет строить сезонность). Параметры $2N$ (количество членов ряда Фурье) и P (период) задаются авторами библиотеки константами: $N = 10,\ P = 365.25$ соответственно. По умолчанию *yearly.seasonality* = 'auto'.

Константы N и P определены в функции *make_all_seasonality_features()*.

7) *weekly.seasonality* — аргумент, с помощью которого задаётся наличие недельной сезонности временного ряда. Принимает три возможных значения: TRUE — сезонность есть, FALSE — сезонности нет или 'auto' — функция сама определяет наличие сезонности (при количестве наблюдений не меньше 14 функция будет строить сезонность). Параметры $2N$ (количество членов ряда Фурье) и P (период) задаются авторами библиотеки константами: $N = 3,\ P = 7$ соответственно. По умолчанию *weekly.seasonality* = 'auto'.

Константы N и P определены в функции *make_all_seasonality_features()*.

8) *seasonality.prior.scale* — аргумент, с помощью которого задаётся априорное значение стандартного отклонения σ вектора β ($\beta \sim N(0, \sigma^2)$). В Stan данная переменная подаётся как *real<lower=0> sigma* и используется в априорном распределении *beta ~ normal(0, sigma)*. По умолчанию *seasonality.prior.scale* = 10.

Аргументы функции *prophet()* для моделирования праздников имеют вид:

9) *holidays* — аргумент, с помощью которого задаётся праздничная компонента временного ряда. Представляет собой таблицу, столбцы которой содержат названия праздников и их даты, а также дополнительно могут содержать количество дней до (*lower_window*) и после (*upper_window*) указанной даты, которые должны быть включены в праздник (например, *lower_window* = -2 включит 2 дня в праздник до указанной даты праздника). По умолчанию *holidays* = NULL, функция *make_all_seasonality_features* сама подбирает праздники под указанные даты временного ряда.

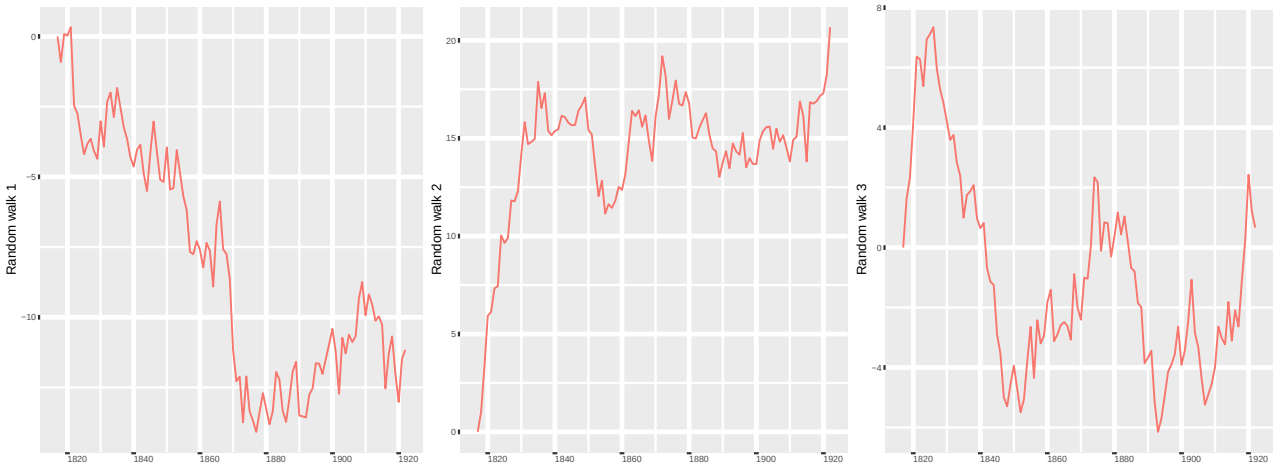
10) *holidays.prior.scale* — аргумент, с помощью которого задаётся априорное значение стандартного отклонения ν вектора эффектов праздников k ($k \sim N(0, \nu^2)$). В модель Stan данный аргумент напрямую не подаётся, а используется в функции *make_holiday_features()* при определении соотношения между стандартными отклонениями сезонной и праздничной компонент *scale.ratio*: чем меньше это соотношение, тем ближе к нулю будут эффекты праздников в сравнении с сезонными эффектами. По умолчанию *holidays.prior.scale* = 10.

$$scale.ratio = \frac{holidays.prior.scale}{seasonality.prior.scale}$$

Результаты

Локальный уровень

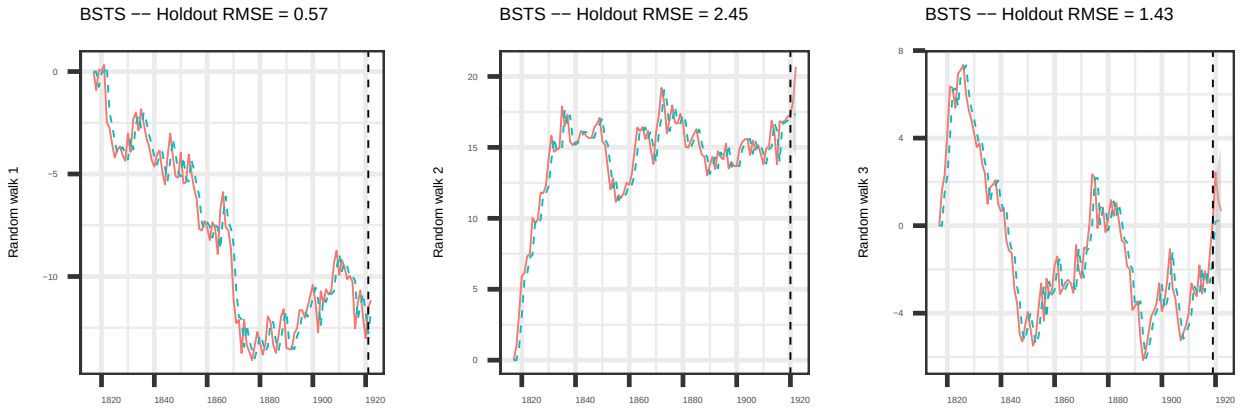
Рассмотрим пример работы BSTS на примерах сгенерированных данных случайного блуждания. Изобразим временные ряды:



Обучим модель BSTS и сделаем прогнозы на 1, 2 и 3 шага вперёд. Для этого зададим спецификацию модели как локальный уровень:

```
# fit BSTS with local level
ss <- AddLocalLevel(list(), yts)
```

Изобразим прогнозные значения: из графиков и значений *RMSE* видно, что с увеличением горизонта прогнозирования ошибка стремительно растёт. Это связано с непредсказуемостью поведения случайного блуждания, что отражается в широких предиктивных интервалах, изображённых на графиках.



Локальный линейный тренд

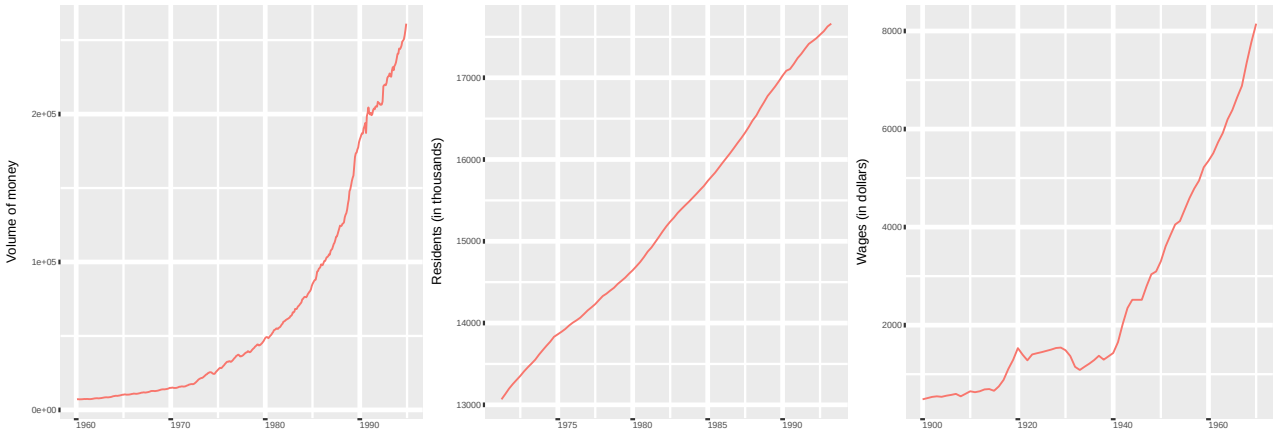
Рассмотрим пример работы BSTS и Prophet на месячных данных объема денежной массы в Австралии с февраля 1960 по декабрь 1994 года[8], на годовых данных средней зарплаты в США с 1900 по 1970 год[9] и на квартальных данных о численности жителей Австралии с марта 1971 по март 1993 года[10]. Представленные ряды обладают линейным трендом, который мы явно укажем при спецификации обеих моделей.

[8] источник данных

[9] источник данных

[10] встроенный набор данных в R

Изобразим временные ряды:

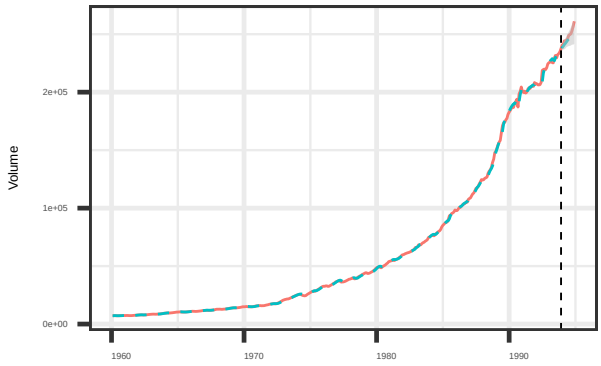


Обучим модели BSTS и Prophet и сделаем прогнозы на несколько месяцев (лет) вперёд. Для этого зададим линейный тренд:

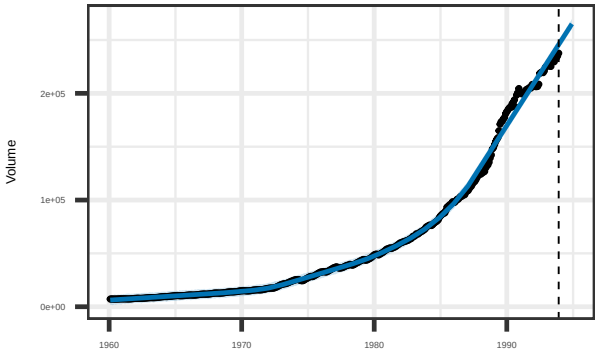
```
# fit BSTS with trend only
ss <- AddLocalLinearTrend(list(), yts)

# fit Prophet with trend only
m <- prophet(yts_df, growth='linear',yearly.seasonality=FALSE, weekly.seasonality=FALSE)
```

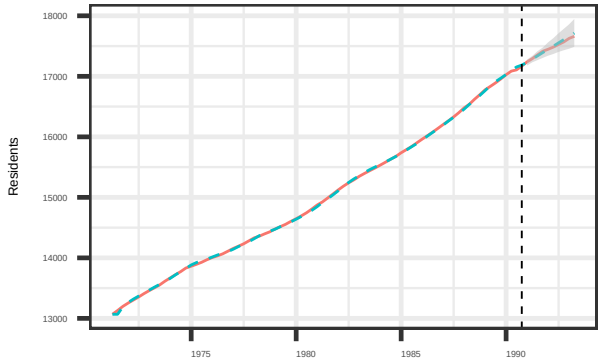
BSTS −− Holdout RMSE = 3617.43



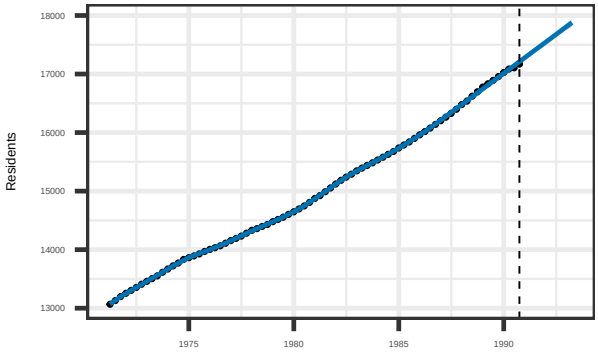
Prophet −− Holdout MAPE = 7938.25



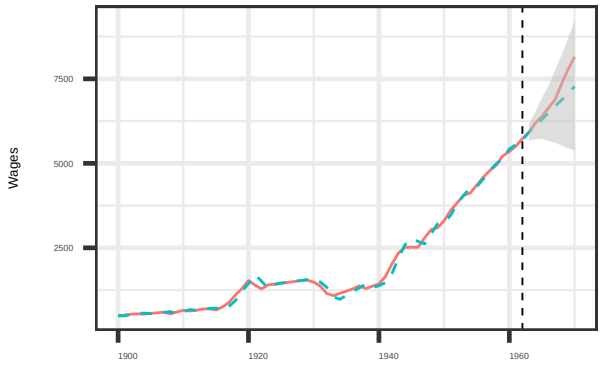
BSTS −− Holdout RMSE = 26.94



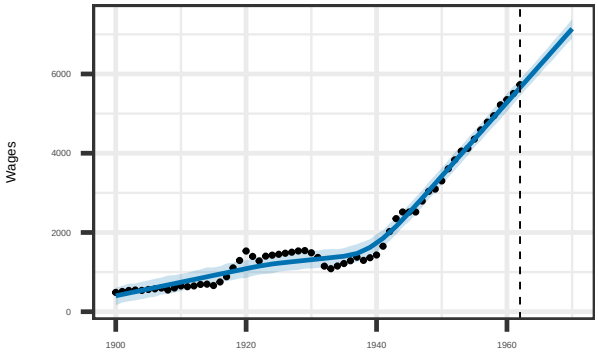
Prophet −− Holdout RMSE = 131.33



BSTS −− Holdout RMSE = 441.99



Prophet −− Holdout RMSE = 528.82



Из полученных прогнозов и значений *RMSE* видно, что оба алгоритма показали хорошее качество работы с трендом, при этом модель локального линейного тренда BSTS обладает меньшим значением ошибки прогнозов, чем модель линейного тренда Prophet, лучше настраиваясь на данные.

Локальный линейный тренд и сезонность

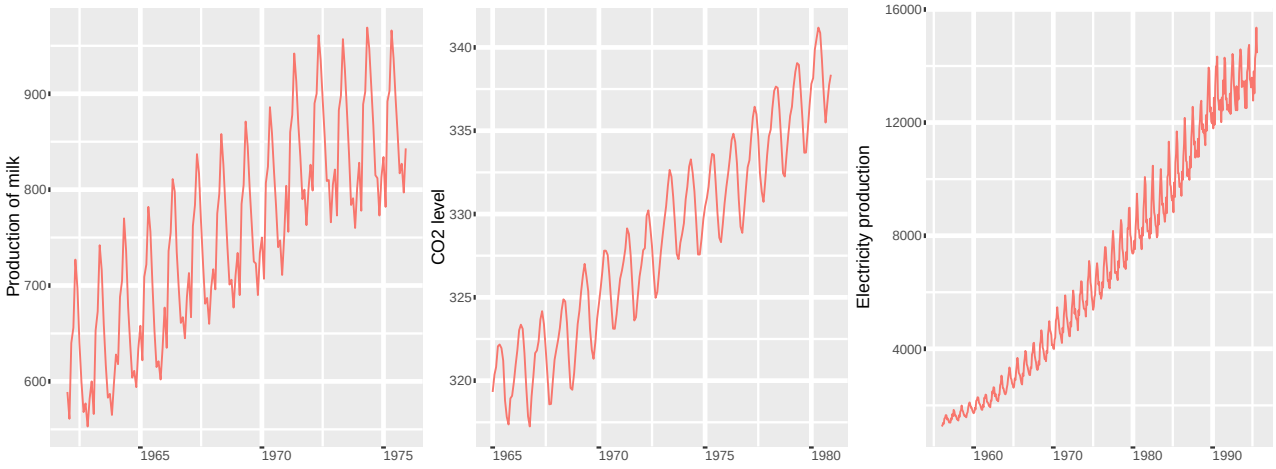
Рассмотрим пример работы BSTS и Prophet на месячных данных объёма производства молока (в фунтах на корову) с января 1962 по декабрь 1975 года[11], на месячных данных уровня углекислого газа у вулкана Мауна-Лоа с января 1965 по декабрь 1980 года[12], на месячных данных производства электричества в Австралии с января 1956 по август 1995 года[13]. Представленные ряды обладают сезонностью и линейным трендом, который мы явно укажем при спецификации обеих моделей.

[11] источник данных

[12] источник данных

[13] источник данных

Изобразим временные ряды:



Обучим модели BSTS и Prophet и сделаем прогнозы на несколько месяцев (лет) вперёд. Для этого зададим тренд и сезонность:

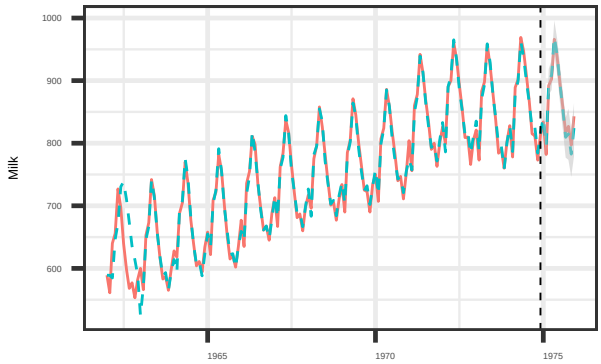
```
# fit BSTS with trend and seasonal component
ss <- AddLocalLinearTrend(list(), yts)
ss <- AddSeasonal(ss, yts, nseasons = 12)

# fit Prophet with trend and seasonal
m <- prophet(yts_df, growth='linear',yearly.seasonality="auto", weekly.seasonality=FALSE)
```

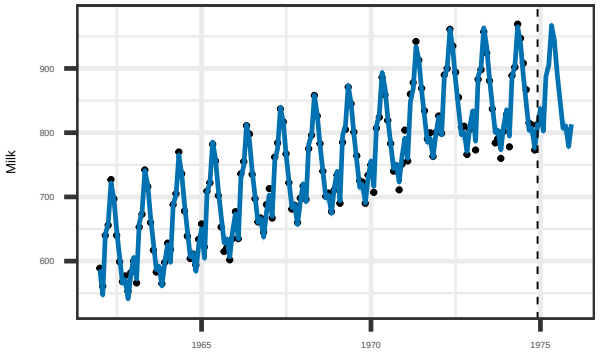
Изобразим прогнозы, а также компоненты[14] (тренд и сезонность), которые мы явно указывали при моделировании временных рядов. Извлечение составляющих временного ряда является одним из преимуществ моделей BSTS и Prophet перед ARIMA-моделью, в которой данные компоненты удаляются с помощью дифференцирования и не анализируются.

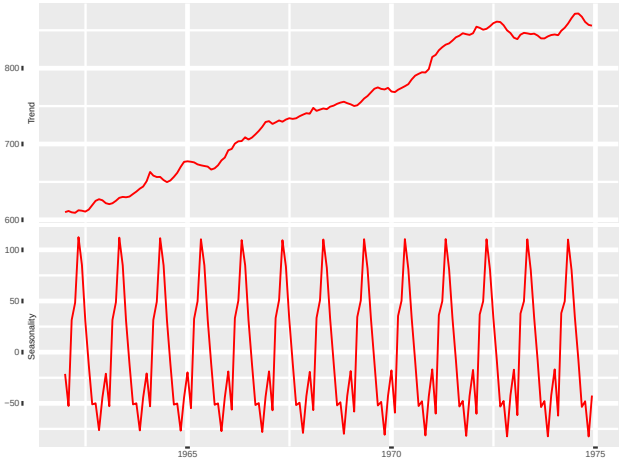
[14] Стоит отметить особенность функции *prophet_plot_components()* для построения компонент ряда: данная функция возвращает список из графиков для каждой компоненты, поэтому изображение для отдельной компоненты *i* можно получить как *prophet_plot_components()[[i]]*

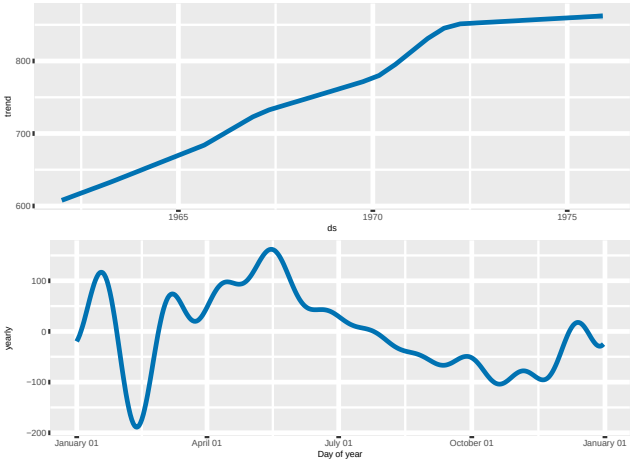
BSTS −− Holdout RMSE = 9.49



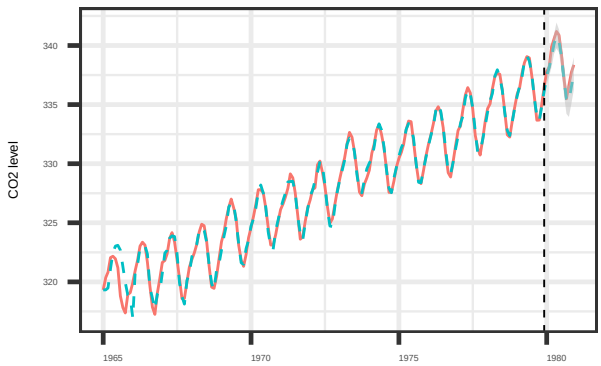
Prophet −− Holdout RMSE = 13.58



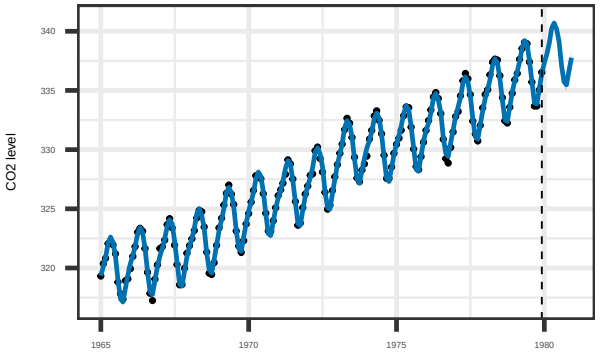


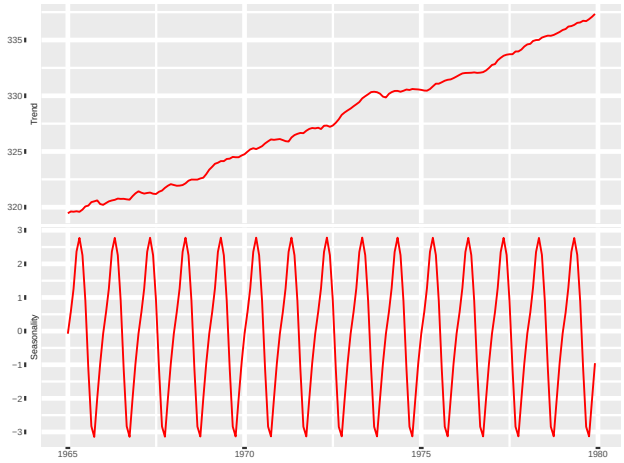


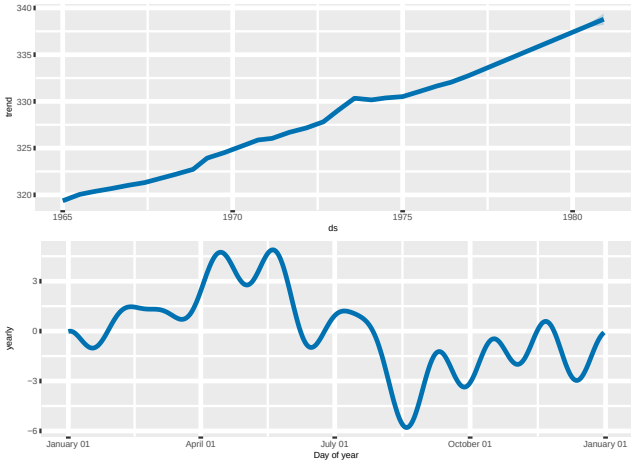
BSTS −− Holdout RMSE = 0.66



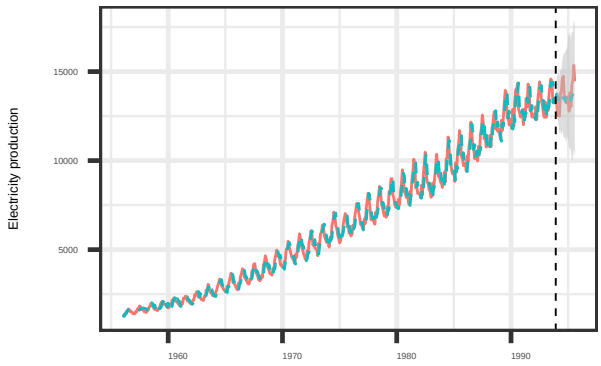
Prophet −− Holdout RMSE = 0.63



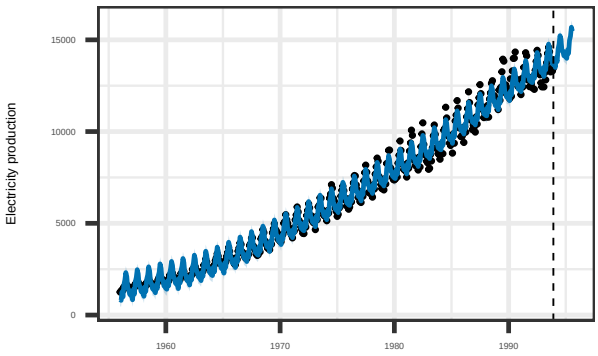


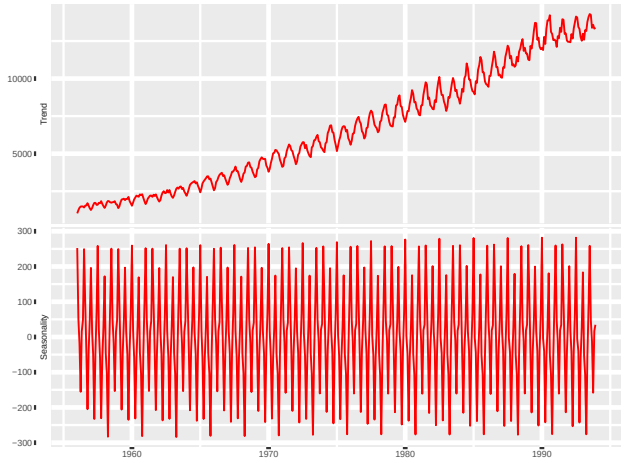


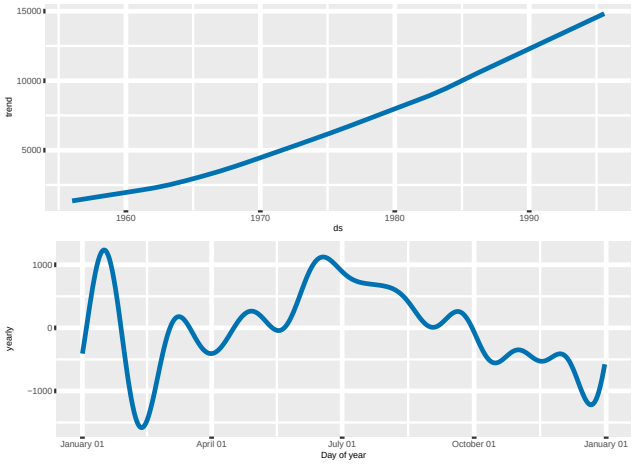
BSTS −− Holdout RMSE = 671.83



Prophet −− Holdout RMSE = 821.11







Из полученных прогнозов и значений *RMSE* видно, что оба алгоритма показали хорошее качество подгонки тренда и сезонности, при этом модель локального линейного тренда и сезонности BSTS обладает меньшим значением ошибки прогнозов, чем модель линейного тренда и сезонности Prophet.

Стоит, однако, отметить преимущество Prophet по сравнению с BSTS в работе с пропущенными значениями временного ряда: Prophet устойчив к пропускам в данных; BSTS пытается заполнить пропуски, что приводит к несовпадению размерностей прогнозов и истинных значений (поэтому перед применением BSTS нужно убедиться, что пропуски в данных заполнены каким-то значением).

(Ссылка на репозиторий с используемыми данными и кодом)

Список литературы

N. Shephard. A. C. Harvey. Structural time series models. *Handbook of Statistics*, 11:261–302, 1993.

S. Peters A. Harvey. Estimation procedures for structural time series models. *Journal of Forecasting*, 9:89–108, 1990.

G. Jenkins. G. Box. Time series analysis: Forecasting and control. *San Francisco: Holden-Day*, 1970.

R. J. Hastie, T. J.; Tibshirani. Generalized additive models. *Chapman & Hall/CRC*, 1990.

K. Larsen. Sorry arima, but i'm going bayesian. apr 2016.

B. Letham S. J. Taylor. Forecasting at scale. 2017a.

B. Letham S. J. Taylor. *Package prophet*. Facebook, apr 2017b.

Steven L. Scott. *Package bsts*. Google, may 2017.