Changepoint Analysis

Table of contents

Preparation . . . 1
Loading Required Packages and Data . . . 1
Analysis . . . 2
Fitting the Model . . . 2
Visualisation . . . 5
Bootstrapping Confidence Intervals . . . 5
Predicting Average and Individual Trajectories . . . 7
Selecting a Random Sample for Plotting . . . 7
Creating the Plot . . . 7
Model Performance . . . 9
Evaluating the Model . . . 9
Cross-Validation . . . 10

To illustrate changepoint analysis, we fit a linear-linear changepoint model.

To reproduce the results, it is necessary to prepare the data set, plot base, and training and test data sets, as outlined in the "Data Preparation" section.

Preparation

Loading Required Packages and Data

Load the necessary packages, data sets, and other supporting files. Each element serves a specific purpose:

- `tidyverse`: For data manipulation and visualisation.
- `segmented`: To fit the changepoint model.
- `lme4`: Fitting a changepoint model in `segmented` requires a lme-object created using `lme4`.

- `caret`: To compute model performance indices.
- `plot_base`: A pre-configured ggplot object for visualisation.
- **Training and Test Data sets**: Required for cross-validation.

```
# Load necessary packages
library(tidyverse)
library(segmented)
library(lme4)
library(caret)

# Load the data set
load("data/wido.rdata")

# Load the pre-configured plot base
plot_base <- readRDS("objects/plot_base.rds")

# Load training and test datasets for cross-validation
training_datasets <- readRDS("objects/training_datasets.rds")
test_datasets <- readRDS("objects/test_datasets.rds")
```

Analysis

Fitting the Model

Fitting a changepoint model in `segmented` requires a lme-object created using `lme4`. Create a lme-object of the model, without the changepoint. Use this lme-object to fit the changepoint model. The comments in the code below indicate what should be filled in. Instead of "pdDiag" (= correlations between random effects are constrained to be 0), "pdSymm" (= random effects and their correlations are unconstrained), and "pdBlocked" (= specify which random effects can be correlated, and which correlations are constrained to be 0) are also possible. A starting value for the changepoint needs to be specified, but using bootstrap resampling mitigates sensitivity to starting values.

```
# Create a linear mixed-effects model object
lme_object <- lme(lifesatisfaction ~ mnths,
                  random = ~ mnths | id,
                  data = wido)
```

```r
# Fit the changepoint model
cp <- segmented.lme(
  lme_object,  # The linear mixed-effects model object
  ~ mnths,     # A one-sided formula indicating the variable with a
   ↪  changepoint
  random =     # A list of the random effects
    list(id = pdDiag( ~ 1 + mnths + U + G0)), # U = the difference-in-slopes
     ↪  parameter; G0 = the changepoint
  # Note that instead of "pdDiag" above, "pdBlocked" and "pdSymm" are also
   ↪  possible
  psi = 0,     # Provide a starting value for the changepoint
  control = seg.control( # Use bootstrap to mitigate potential sensitivity to
   ↪  starting values
    display = F,
    n.boot = 100,
    seed = 123
  )
)

# Display the summary of the model
summary(cp)
```

```
Segmented mixed-effects model fit by REML
       AIC      BIC    logLik
  5409.571 5461.311 -2695.785
 Bootstrap restarting on 100 samples;  5 different solution(s)

Random effects:
 Formula: ~1 + mnths + U + G0 | id
 Structure: Diagonal
        (Intercept)       mnths            U       G0  Residual
StdDev:   0.8028599 0.005873525 0.0003828891 1.592956 0.6456952

Fixed effects:
                 Value Std.Error   DF t-value p-value
(Intercept)  4.680470 0.0620050 2111   75.48       0
-- leftS:
mnths       -0.007652 0.0006958 2111  -11.00       0
-- diffS:
U            0.015487 0.0011710 2111   13.23
-- break:
```

```
G0           5.368898 4.6777896 2111
 psi.link = identity

Standardized Within-Group Residuals:
        Min          Q1         Med          Q3         Max
-5.39537983 -0.49421689  0.06929748  0.57182047  3.49614881

Number of Observations: 2322
Number of Groups: 208
```

```
# Display the slope estimates
slope(cp)
```

```
                  Est.       St.Err    t value     0.95.low      0.95.up
leftSlope  -0.007651572 0.0006958491 -10.996022 -0.009015412 -0.006287733
rightSlope  0.007835602 0.0010132926   7.732813  0.005849585  0.009821619
```

```
# Compute confidence intervals for the model parameters
intervals(cp$lme.fit, which = "all")
```

```
Approximate 95% confidence intervals

 Fixed effects:
                  lower         est.        upper
(Intercept)  4.568734113  4.690331382  4.811928651
mnths       -0.009651489 -0.008286868 -0.006922246
U            0.013879689  0.016176031  0.018472374
G0          -3.804660960  5.368897948 14.542456856

 Random Effects:
  Level: id
                       lower         est.        upper
sd((Intercept)) 7.229097e-01 0.8028599098  0.891652219
sd(mnths)       4.958933e-03 0.0058735248  0.006956798
sd(U)           1.988277e-07 0.0003828891  0.737342242
sd(G0)          5.594838e-02 1.5929562719 45.354478502

 Within-group standard error:
    lower      est.     upper
0.6253740 0.6456952 0.6666767
```

Visualisation

Bootstrapping Confidence Intervals

Use bootstrapping to estimate the confidence intervals for the predicted values of the model. This provides a robust measure of uncertainty. Create custom functions to perform the bootstrap resampling.

```
# For reproducibility
set.seed(123)

# Create a custom function to fit the model and generate predictions based on
 ↪  the estimated fixed effects
predict_fun <- function(data, mnths_vals) {

  # Create a linear mixed-effects model object
  lme_object <- lme(fixed = lifesatisfaction ~ mnths, random = ~mnths | id,
 ↪  data = data)

  # Apply the segmented mixed-effects model
  cp_model <- segmented.lme(
    obj = lme_object,
    seg.Z = ~mnths,
    random = list(id = pdDiag(~1 + mnths + U + G0)),
    psi = 0,
    control = seg.control(display = F, n.boot = 0),
    data = data
  )

  # Create an empty vector to store predictions for the given mnths values
  predictions <- numeric(length(mnths_vals))

  # Predict the fixed effects for each level of mnths
  for (i in 1:length(mnths_vals)) {
    mnth <- mnths_vals[i]

    # Use the breakpoint to compute predictions
    predictions[i] <- if_else(mnth <
 ↪  cp_model$lme.fit$coefficients$fixed[[4]],
                         (cp_model$lme.fit$coefficients$fixed[[1]] +
                            (cp_model$lme.fit$coefficients$fixed[[2]] *
 ↪  mnth)),
```

```
                        (cp_model$lme.fit$coefficients$fixed[[1]] +
                           ((cp_model$lme.fit$coefficients$fixed[[2]] +
 ↪  cp_model$lme.fit$coefficients$fixed[[3]]) * mnth)))
  }

  # Return the predicted fixed effects
  return(predictions)
}

# Manual Bootstrap Process
n_iter <- 100  # Number of bootstrap iterations

# Create an empty matrix to store the predictions
bootstrap_predictions <- matrix(NA, nrow = n_iter, ncol =
 ↪  length(seq(min(wido$mnths), max(wido$mnths), by = 1)))

# Define a sequence of mnths values (the levels for which predictions are to
 ↪  be made)
mnths_seq <- seq(min(wido$mnths), max(wido$mnths), by = 1)

# Loop over bootstrap iterations
for (i in 1:n_iter) {
  # Resample the data with replacement
  bootstrap_sample <- wido[sample(nrow(wido), replace = TRUE), ]

  # Predict fixed effects for the resampled data based on the defined mnths
   ↪  sequence
  bootstrap_predictions[i, ] <- predict_fun(data = bootstrap_sample,
 ↪  mnths_vals = mnths_seq)
}

# The bootstrap_predictions matrix now contains the predictions for each
 ↪  iteration and mnths value

# Calculate 95% confidence intervals from bootstrapped predictions for each
 ↪  mnths level
ci95 <- apply(bootstrap_predictions, 2, quantile, probs = c(0.025, 0.975),
 ↪  na.rm = TRUE)

# Store the lower and upper bounds in a new data frame that matches the mnths
 ↪  sequence
bootci_results <- data.frame(mnths = mnths_seq, lower_bound = ci95[1, ],
 ↪  upper_bound = ci95[2, ])
```

Predicting Average and Individual Trajectories

Predict both the population-level (fixed effects) and individual-level (random effects) trajectories of life satisfaction.

```
# Predict population-level trajectories based on fixed effects
wido <- wido %>%
  mutate(lifesatisfaction_cp_f = if_else(mnths <
   ↪  cp$lme.fit$coefficients$fixed[[4]],
                              (cp$lme.fit$coefficients$fixed[[1]] +
 ↪  (cp$lme.fit$coefficients$fixed[[2]] * mnths)),
                              (cp$lme.fit$coefficients$fixed[[1]] +
 ↪  ((cp$lme.fit$coefficients$fixed[[2]] +
 ↪  cp$lme.fit$coefficients$fixed[[3]]) * mnths))))

# Obtain the individual-level predictions from the cp model object
wido$lifesatisfaction_cp_r <- fitted(cp)
```

Selecting a Random Sample for Plotting

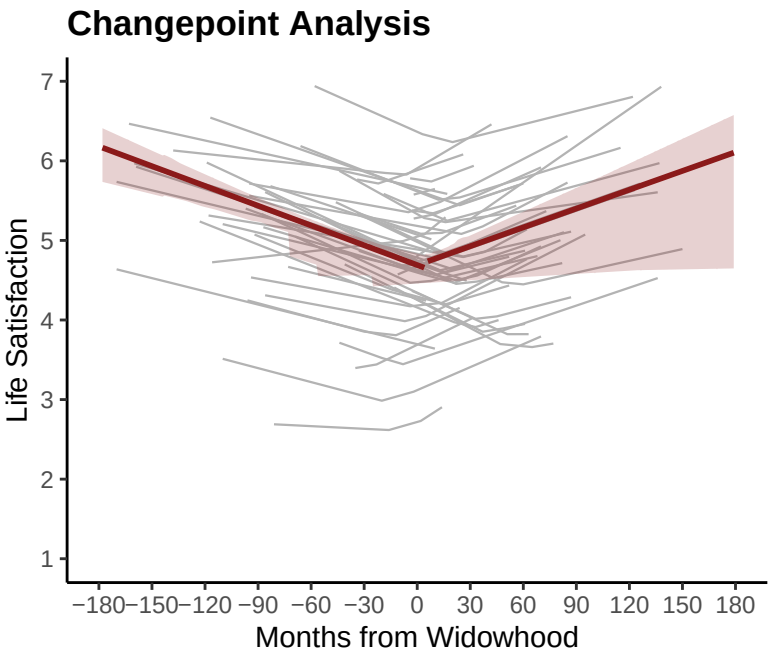
For better visualisation, select a random sample of individuals to display their individual trajectories.

```
# For reproducibility
set.seed(123)

# Randomly sample 50 participants
rsample_ids <- sample(unique(wido$id), 50)

# Filter the data to include only the randomly selected participants
wido_rsample <- wido %>%
  filter(id %in% rsample_ids)
```

Creating the Plot

Combine all elements to create the plot, which includes individual trajectories, the population trajectory, and the confidence interval of the population trajectory.

```r
# Create the plot using the pre-configured plot base
plot_base +
  geom_line(
    data = wido_rsample,
    aes(mnths, lifesatisfaction_cp_r, group = id),
    color = "grey70",
    linewidth = 0.4
  ) +
  geom_line(
    data = wido,
     aes(
      x = mnths,
      y = ifelse(mnths == round(cp$lme.fit$coefficients$fixed[[4]], 0), NA,
       ↪  lifesatisfaction_cp_f)
      ),
    color = "firebrick4",
    linewidth = 1
  ) +
  geom_ribbon(
    data = bootci_results,
    aes(ymin = lower_bound, ymax = upper_bound, x = mnths),
    alpha = 0.2,
    fill = "firebrick4"
  ) +
  ggtitle("Changepoint Analysis") +
  theme(plot.title = element_text(size = 13, face = "bold"))
```



Model Performance

Evaluating the Model

Assess the model's performance using the Bayesian Information Criterion (BIC), R-squared (R^2), Mean Absolute Error (MAE), and Root Mean Squared Error (RMSE).

```
# Compute BIC for the fitted model
round(BIC(cp), 2)
```

```
[1] 5461.31
```

```
# Calculate R², MAE, and RMSE for the fixed effects predictions
data.frame(
  R2_FE = round(R2(wido$lifesatisfaction_cp_f, wido$m_lifesat_per_mnth), 2),
  MAE_FE = round(MAE(wido$lifesatisfaction_cp_f, wido$m_lifesat_per_mnth),
  ↪  2),
  RMSE_FE = round(RMSE(wido$lifesatisfaction_cp_f, wido$m_lifesat_per_mnth),
  ↪  2)
)
```

```
  R2_FE MAE_FE RMSE_FE
1  0.17   0.35    0.48
```

```
# Calculate R², MAE, and RMSE for the random effects predictions
data.frame(
  R2_RE = round(R2(wido$lifesatisfaction_cp_r, wido$lifesatisfaction), 2),
  MAE_RE = round(MAE(wido$lifesatisfaction_cp_r, wido$lifesatisfaction), 2),
  RSME_RE = round(RMSE(wido$lifesatisfaction_cp_r, wido$lifesatisfaction), 2)
)
```

```
  R2_RE MAE_RE RSME_RE
1   0.7   0.46    0.61
```

Cross-Validation

To assess the replicability of the model, perform cross-validation using the training and test datasets. For each training dataset, fit the model and compute performance metrics for the associated test dataset R^2, MAE, and RMSE.

```
# Initialise vectors to store performance metrics
R2_values <- c()
MAE_values <- c()
RMSE_values <- c()

# Loop over the datasets
for (i in 1:length(training_datasets)) {
  # Get the current training and test dataset
  training_data <- training_datasets[[i]]
  test_data <- test_datasets[[i]]

  # Fit the initial linear mixed model
  fit_lme <- lme(lifesatisfaction ~ mnths, random = ~mnths | id, data =
 ↪  training_data)

  # Apply the segmented mixed-effects model
  cp <- segmented.lme(
    fit_lme,
    ~mnths,
    random = list(id = pdDiag(~1 + mnths + U + G0)),  # Adjust as needed
     ↪  based on your actual random effects
    psi = 0,  # Initial breakpoint value for segmentation
```

```
    control = seg.control(display = F, n.boot = 100, seed = 123)
  )

  # Predict fixed effects from the segmented model
  test_data <- test_data %>%
    mutate(pred_cp_f = if_else(mnths < cp$lme.fit$coefficients$fixed[[4]],
                               (cp$lme.fit$coefficients$fixed[[1]] +
 ↪  (cp$lme.fit$coefficients$fixed[[2]]*mnths)),
                               (cp$lme.fit$coefficients$fixed[[1]] +
 ↪  ((cp$lme.fit$coefficients$fixed[[2]] +
 ↪  cp$lme.fit$coefficients$fixed[[3]])*mnths))))

  # Compute average test trajectory
  test_data <- test_data %>%
    group_by(mnths) %>%
    mutate(mean_ls = mean(lifesatisfaction, na.rm = TRUE))

  # Compute performance metrics
  R2_value <- R2(test_data$pred_cp_f, test_data$mean_ls)
  RMSE_value <- RMSE(test_data$pred_cp_f, test_data$mean_ls)
  MAE_value <- MAE(test_data$pred_cp_f, test_data$mean_ls)

  # Store the metrics
  R2_values <- c(R2_values, R2_value)
  RMSE_values <- c(RMSE_values, RMSE_value)
  MAE_values <- c(MAE_values, MAE_value)
}

# Compute average performance metrics (mean)
  average_R2 <- mean(R2_values)
  average_MAE <- mean(MAE_values)
  average_RMSE <- mean(RMSE_values)

# Compute average performance metrics (SD)
  sd_R2 <- sd(R2_values)
  sd_MAE <- sd(MAE_values)
  sd_RMSE <- sd(RMSE_values)

# Combine the mean and standard deviation into one data.frame
combined_metrics <- data.frame(
  Metric = c("R²", "MAE", "RMSE"),
  Mean = round(c(average_R2, average_MAE, average_RMSE), 2),
```

```
  SD = round(c(sd_R2, sd_MAE, sd_RMSE), 2)
)

# Print the combined metrics
print(combined_metrics)
```

```
  Metric Mean   SD
1     R² 0.07 0.07
2    MAE 0.60 0.11
3   RMSE 0.81 0.17
```