Changepoint Analysis

Table of contents

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.

## 1 Preparation

### 1.1 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")

## 2 Analysis

### 2.1 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)  
  
# 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

## 3 Visualisation

### 3.1 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, ])

### 3.2 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)

### 3.3 Selecting a Random Sample for Plotting

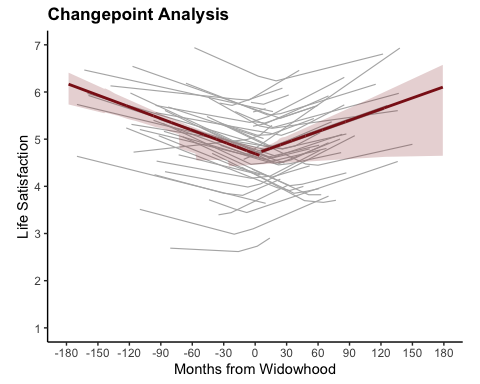
For better visualisation, select a random sample of individuals to display their individual trajectories.

# 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)

### 3.4 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.

# 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"))



## 4 Model Performance

### 4.1 Evaluating the Model

Assess the model’s performance using the Bayesian Information Criterion (BIC), R-squared (R²), 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

### 4.2 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 R², 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