The development of the TextSEM Package

The package TextSEM includes various functions to analyze text data in the SEM framework.

List of functions

The following functions are available in the packages. The exported functions are:

- sem.sentiment for dictionary based sentiment analysis
- sem.encode and sem.emb for sentence embedding based analysis
- sem.topic for analysis based on topic modeling

sem.sentiment

```
#' Structural Equation Modeling with Sentiment Analysis
#'
#' This function integrates sentiment analysis into a structural equation model (SEM) by calculating sentiment scores for specified text variables and incorporating them as additional variables in the SEM.
#'
#' @param model The structural equation model specified as a character string.
#' @param df A data frame containing the input data.
#' @param text_vars A character vector of text variable names in the data frame for which sentiment analysis should be performed.
#' @param text_stats A character vector of text sentiment statistics to be added to the SEM. Currently supports only 'OverallSenti' (overall sentiment).
#' @param polarity_dt A data table for polarity lexicon to be used for sentiment analysis. Defaults to `lexicon::hash_sentiment_jockers_rinker`.
#' @param valence_shifters_dt A data table for valence shifters to be used for sentiment analysis. Defaults to `lexicon::hash_valence_shifters`.
#' @param missing The method for handling missing data in the SEM. Defaults to 'ML' (maximum likelihood).
#' @param fixed.x Logical. If `TRUE`, the exogenous variables are treated as fixed. Defaults to `FALSE`.
#' @param ... Additional arguments passed to the `sentiment_by` function from the `sentimentr` package and to the `sem` function from the `lavaan` package.
#'
```

```
#' @return A list containing three items:
#' \item{model}{A character string representing the modified SEM with added sentiment variables.}
#' \item{data}{A data frame with added text sentiment statistics.}
#' \item{estimates}{The fitted SEM model object.}
#' @importFrom lavaan lavParseModelString lavaanify sem
#' @importFrom sentimentr sentiment_by
#' @importFrom data.table rbindlist
#' @importFrom sentiment.ai sentiment_score
#' @export
#'
sem.sentiment <- function(model,
                          df,
                          text_vars,
                          method="sentimentr",
                          text_stats=c('sentiment'),
                          polarity_dt = lexicon::hash_sentiment_jockers_rinker,
                          valence_shifters_dt = lexicon::hash_valence_shifters,
                          missing = 'ML',
                          fixed.x = FALSE,
                          ...){

  ## parse the model
  model_info <- lavParseModelString(model)
  model_var <- unique(c(model_info$lhs, model_info$rhs))

  ## get the list of text variables in the model
  text_vars <- text_vars[text_vars %in% model_var]
  # print("text_vars")
  # print(text_vars)

  N <- length(text_vars) # Number of text variables
  if (N > 0){
    ## now get the sentiment score of the text
    text_scores <- list()

    batch_sentiment <- function(text, batch_size = 200, ...) {
      if(method == "sentimentr"){
        text_batches <- split(text, ceiling(seq_along(text) / batch_size))
        scores <- data.table::rbindlist(lapply(text_batches, sentiment_by))$ave_sentiment
      }else if(method == "sentiment.ai"){
```

```
    scores <- unname(sentiment_score(text))
  }
  return(scores)
}

for(i in 1:N){
  sentiment_result <- batch_sentiment(df[, text_vars[i]]) # Compute sentiment scores
  text_scores[[i]] <- sentiment_result
}
names(text_scores) <- text_vars
# print("text_score")
# print(as.data.frame(text_score))

print("456")
data_new <- cbind(df, as.data.frame(text_scores))
names(data_new) <- c(names(df), paste0(rep(text_vars, each = length(text_stats)), '.', text_stats))
print("data_new")
print(names(data_new))

model_lavaanify <- lavaanify(model)
model_user <- model_lavaanify[model_lavaanify$user==1, ]
# print("model_user")
# print(model_user)

model_new <- c()
for(i in 1:nrow(model_user)){
  row <- model_user[i,]
  # print(row)
  if((row['lhs'] %in% text_vars) && (row['rhs'] %in% text_vars)){
    model_new <- c(model_new, paste0(rep(paste0(row['lhs'], '.', text_stats), each = length(text_stats)),
                                     ' ', row['op'], ' ', rep(paste0(row['rhs'], '.', text_stats), length(text_stats))))
  } else if(row['lhs'] %in% text_vars){
    model_new <- c(model_new, paste0(row['lhs'], '.', text_stats, ' ', row['op'], ' ', row['rhs']))
  } else if(row['rhs'] %in% text_vars){
    model_new <- c(model_new, paste0(row['lhs'], ' ', row['op'], ' ', row['rhs'], '.', text_stats))
  } else{
    model_new <- c(model_new, paste0(row['lhs'],  ' ', row['op'], ' ', row['rhs']))
  }
}
# print(model_new)
```

```
    # model_new <- paste0(model_new, collapse = '\n')
  }
  model_res <- sem(model=model_new, data=data_new,
                     missing = missing, fixed.x = fixed.x)

  return(list(model=model_new, data=data_new, estimates=model_res))
}
```

sem.encode

```
#' Generate Sentence Embeddings using Sentence Transformers
#'
#' This function generates sentence embeddings for a given vector of text using a specified pre-trained model from the `sentence_transformers` Python package.
#'
#' @param text_vector A character vector containing the text data to be embedded.
#' @param encoder A character string specifying the name of the pre-trained model to be used for generating embeddings.
#' @param reduce_method Dimension reduction method for embeddings. Can be either "SVD" or "PCA".
#' @param reduce_dim An integer denoting the size of embedding after reduction.
#'
#' @return A matrix of sentence embeddings with each row corresponding to a sentence from `text_vector` and each column representing a dimension of the embedding space.
#' @import reticulate
#' @export
#'
#' @examples
#' \dontrun{
#' # Example usage
#' text_vector <- c("This is a sentence.", "This is another sentence.")
#' model_name <- "paraphrase-MiniLM-L6-v2"
#' embeddings <- sem.emb(text_vector, model_name)
#' print(embeddings)
#' }
sem.encode <- function(text_vector, encoder = "all-mpnet-base-v2", reduce_method = "SVD", reduce_dim = 5){
  models.sbert = c("all-mpnet-base-v2", "paraphrase-MiniLM-L6-v2")
  models.gpt = c("text-embedding-3-small", "text-embedding-3-large", "text-embedding-ada-002")

  normalize_l2 <- function(vector) {
    norm <- norm(vector, type = "2")
  if (norm == 0) norm <- 1  # Avoid division by zero
  return(vector / norm)
}

generate_emb <- function(text){
  response = openai$embeddings$create(input=text, model=encoder, encoding_format="float")
  emb = response$data[[1]]$embedding
  normalized_emb = normalize_l2(emb)
  return(normalized_emb)
}

if(encoder %in% models.sbert){
  sbert <- import("sentence_transformers")
  model <- sbert$SentenceTransformer(encoder)
  embeddings <- model$encode(text_vector)
} else if(encoder %in% models.gpt){
  openai <- import("openai")
  openai$api_key <- Sys.getenv("OPENAI_API_KEY")
  embs <- lapply(text_vector, generate_emb)
  embeddings <- do.call(rbind, embs)
} else {
  stop("Encoder not supported.")
}

if(reduce_method == "SVD"){
  svd_result <- svd(embeddings)
  U <- svd_result$u
  D <- diag(svd_result$d[1:reduce_dim])
  reduced_emb <- U[, 1:reduce_dim] %*% D
  print(dim(reduced_emb))  # Should be n x reduce_dim
  print(reduced_emb)
} else if(reduce_method == 'PCA'){
  pca_result <- prcomp(embeddings, scale. = TRUE, center = TRUE)
  reduced_emb <- pca_result$x[, 1:reduce_dim]
  # print(dim(reduced_emb))  # Should be n x reduce_dim
  # print(reduced_emb)
}

# Rename the columns
colnames(reduced_emb) <- paste0('v', 1:ncol(reduced_emb))
```

```
  rownames(reduced_emb) <- 1:nrow(reduced_emb)
  return(as.matrix(reduced_emb))
}

#' Structural Equation Modeling with Embeddings
#'
#' This function performs Structural Equation Modeling (SEM) using text embeddings. It checks if the specified
`.rda` file with embeddings exists. If the file exists, it loads the embeddings; otherwise, it generates the
embeddings using the `sem.encode` function. The embeddings are then incorporated into the SEM model.
#'
#' @param sem_model A character string specifying the SEM model.
#' @param data A data frame containing the input data.
#' @param text_var A character string specifying the name of the text variable in the data frame.
#' @param pca_dim A integer specifying reduced dimension through PCA.
#' @param encoder A character string specifying the encoder model to be used for generating embeddings.
Defaults to "all-mpnet-base-v2".
#' @param reduce_dim An integer denoting the size of embedding after reduction.
#' @param emb_filepath A character string specifying the path to the `.rda` file containing the embeddings. If
`NULL`, embeddings are generated using `sem.encode`.
#'
#' @return The result of the `lavaan::sem` function, which is an object of class `lavaan`.
#' @importFrom lavaan sem
#' @importFrom stats prcomp
#' @export
#'
#' @examples
#' \dontrun{
#' sem_model <- 'rating ~ book + difficulty + comments'
#' res <- sem.emb(sem_model = sem_model, data = prof.nest, text_var = "comments",
#'                  pca_dim = 10, emb_model = "all-mpnet-base-v2")
#' summary(res, fit=TRUE)
#' }
#'
sem.emb <- function(sem_model, data, text_var, encoder = "all-mpnet-base-v2", emb_filepath = NULL,
reduce_method = "SVD", reduce_dim = 5){

  df <- data

  # Check if the file path ends with .rda
  load_flag = FALSE
  if (!is.null(emb_filepath)) {
    if (grepl("\\.rda$", emb_filepath)) {
      if (file.exists(emb_filepath)) {
        # Load the file
        print("Loading embeddings from file...")
        embeddings <- get(load(emb_filepath))
        if (is.matrix(embeddings)) {
          if (nrow(embeddings) == nrow(df)) {
            print("Success.")
            load_flag = TRUE
          } else {
            print("Incorrect dimension.")
          }
        } else {
          print("Loaded object is not a matrix.")
        }
      } else {
        print("File doesn't exist.")
      }
    } else {
      stop("The specified file is not an `.rda` file.")
    }
  }

  if (!load_flag){
    print("Generating embeddings, this might take a while...")
    embeddings <- TextSEM::sem.encode(df[[text_var]], reduce_method = reduce_method, reduce_dim = reduce_dim)
    print("Success.")
  }
  replace_vars_in_model <- function(model, var_old, var_new) {
    replacement <- paste(var_new, collapse = " + ")
    updated_model <- gsub(paste0("\\b", var_old, "\\b"), replacement, model)
    return(updated_model)
  }

  colnames(embeddings) <- paste0(text_var, '.', colnames(embeddings))
  model_new <- replace_vars_in_model(sem_model, text_var, colnames(embeddings))

  df <- cbind(df, embeddings)
  df <- as.data.frame(df)
  estimates <- lavaan::sem(model = model_new, data = df)

  list(model = model_new, data = df, estimates = estimates)
}
```

sem.topic

```
#' Perform Latent Dirichlet Allocation on a Data Frame
#'
#' This function takes a data frame and performs text preprocessing followed by Latent Dirichlet Allocation (LDA) for topic modeling.
#'
#' @param data A data frame containing the data.
#' @param text_var A variable in the data frame containing the text data to be analyzed.
#' @param n_topic Number of topics to be extracted.
#' @param method The method to be used for LDA fitting; currently method = "VEM" or method= "Gibbs" are supported.
#' @param sparse A numeric for the maximal allowed sparsity in the range from bigger zero to smaller one.
#' @param seed Random seed for LDA estimation
#'
#' @return A topic model object of class "LDA" from the `topicmodels` package.
#' @import dplyr
#' @importFrom tidytext unnest_tokens cast_dtm
#' @importFrom tm removeSparseTerms
#' @importFrom topicmodels LDA
#' @importFrom SnowballC wordStem
#' @importFrom utils data
#' @export
#'
#' @examples
#' \dontrun{
#' data(prof.nest)
#' lda.model <- sem.lda(df, text_var = c("comments"), n_topic = c(6))
#' lda.model
#' }
sem.lda <- function(df, text_var, n_topic, method = "VEM", sparse = .995, seed = 42){

  df["row_index"] <- 1:nrow(df)

  # Split text into terms (words)
  df.tm <- unnest_tokens(df, word, {{text_var}})

  ## Remove stopwords
  data(stopwords, envir = environment())
  df.tm <- df.tm %>% anti_join(filter(stopwords, lexicon == "evaluation"), by = join_by(word))

  ## Stem words
  df.tm$word <- SnowballC::wordStem(df.tm$word)
  df.tm <- df.tm %>%
    filter(!grepl("[[:digit:]]", word))

  ## Build Document-term matrix: https://en.wikipedia.org/wiki/Document-term_matrix
  df.dtm <- df.tm %>%
    count(.data[["row_index"]], word) %>%    ## word frequency
    tidytext::cast_dtm(.data[["row_index"]], word, n)  ## convert to dtm matrix
  df.dtm <- tm::removeSparseTerms(df.dtm, sparse)

  ## Latent Dirichlet Allocation (LDA): https://en.wikipedia.org/wiki/Latent_Dirichlet_allocation
  topicmodels::LDA(df.dtm, k = n_topic, control=list(seed = seed))
}



#' Perform Structural Equation Modeling with Latent Dirichlet Allocation
#'
#' This function performs structural equation modeling (SEM) combined with Latent Dirichlet Allocation (LDA) to analyze text data.
#'
#' @param model A description of the user-specified model. Typically, the model is described using the lavaan model syntax. See model.syntax for more information. Alternatively, a parameter table (eg. the output of the lavaanify() function) is also accepted.
#' @param data A data frame containing the data.
#' @param text_vars A character vector of text variable names in the data frame containing the text data to be analyzed.
#' @param n_topics A numeric vector containing number of topics to be extracted for each text variable.
#' @param method The method to be used for LDA fitting; currently method = "VEM" or method= "Gibbs" are supported.
#' @param sparse A numeric for the maximal allowed sparsity in the range from bigger zero to smaller one.
#' @param seed Random seed for LDA estimation
#'
```

```
#' @return A list containing four elements:
#' \item{model}{A character string representing the modified SEM with added topic variables.}
#' \item{data}{A data frame with added topic statistics.}
#' \item{estimates}{The fitted SEM model object.}
#' \item{lda}{A vector of LDA model objects.}
#' @import dplyr
#' @importFrom tidytext tidy
#' @importFrom lavaan sem
#' @importFrom tidyr spread
#' @importFrom stats setNames
#' @export
#'
#' @examples
#' \dontrun{
#' data(prof.nest)
#' model <- 'rating ~ book + difficulty + comments + tags'
#' res <- sem.topic(model = model,
#'                  data = prof.nest,
#'                  text_vars = c('comments', 'tags'),
#'                  n_topics = c(6, 3))
#' summary(res$model, fit=TRUE)
#' }
sem.topic <- function(model, data, text_vars, n_topics, method = "VEM", sparse = .995, seed = 42){

  df <- data
  df["row_index"] <- 1:nrow(df)

  lda_objects = c()
  for(i in 1:length(text_vars)){
    # print(i)

    # Get LDA matrix
    df.lda <- sem.lda(df, text_vars[i], n_topics[i], method = method)
    lda_objects <- c(lda_objects, df.lda)

    ## Gamma (per-document-per-topic probability): the proportion of the document that is made up of words from the assigned topic
    document.prob <- tidytext::tidy(df.lda, matrix = "gamma")
    document.prob <- document.prob %>%
      tidyr::spread(key=topic, value=gamma, sep='')
    ## Combine the data with gamma
    # Rename the columns: topic_i -> text_var.topic_i
    names(document.prob)[2:(n_topics[i] + 1)] <- paste(rep(text_vars[i], n_topics[i]),
names(document.prob)[2:(n_topics[i] + 1)], sep = ".")
    document.prob$document <- as.numeric(document.prob$document)
    df <- left_join(df, document.prob, by=join_by(row_index==document))
  }

  lda_objects <- setNames(lda_objects, text_vars)

  ## Rewrite the lavaan model by replacing text_var with text_var.topic_i
  model_lavaanify <- lavaanify(model)
  model_user <- model_lavaanify[model_lavaanify$user==1, ]
  model_new <- c()

  # Remove the last topic component
  df_topic <- setNames(n_topics - 1, text_vars)

  for(i in 1:nrow(model_user)){
    row <- model_user[i,]
    # print(row)
    if((row['lhs'] %in% text_vars) && (row['rhs'] %in% text_vars)){
      left <- paste0(rep(paste0(row['lhs'], '.topic'), df_topic[as.character(row$lhs)]),
1:df_topic[as.character(row$lhs)])
      right <- paste0(rep(paste0(row['rhs'], '.topic'), df_topic[as.character(row$rhs)]),
1:df_topic[as.character(row$rhs)])
    } else if(row['lhs'] %in% text_vars){
      left <- paste0(rep(paste0(row['lhs'], '.topic'), df_topic[as.character(row$lhs)]),
1:df_topic[as.character(row$lhs)])
      right <- as.character(row$rhs)
      model_new <- c(model_new, paste0(row['lhs'], '.topic', text_stats, ' ', row['op'], ' ', row['rhs']))
    } else if(row['rhs'] %in% text_vars){
      left <- as.character(row$lhs)
      right <- paste0(rep(paste0(row['rhs'], '.topic'), df_topic[as.character(row$rhs)]),
1:df_topic[as.character(row$rhs)])
    } else{
      left <- as.character(row$lhs)
      right <- as.character(row$rhs)
    }
    combinations <- expand.grid(left, right)
    model_new <- c(model_new, paste(combinations$Var1, row['op'], combinations$Var2))
  }

  # print(model_new)
  # model_new <- paste0(model_new, collapse = '\n')

  estimates <- lavaan::sem(model = model_new, data = df)

  list(model = model_new, data = df, estimates = estimates, lda = lda_objects)
}

#' Plot Top Terms in LDA Topics
#'
#' This function plots the top terms in each topic from a Latent Dirichlet Allocation (LDA) model.
#'
#' @param df.lda A fitted LDA model object.
#'
#' @return A ggplot object showing the top terms in each topic.
#' @import dplyr
#' @import ggplot2
#' @importFrom tidytext tidy scale_x_reordered
#' @importFrom stats reorder
#' @export
#'
#' @examples
#' \dontrun{
#' # Assuming 'lda_model' is a fitted LDA model object
#' sem.topic.plot(lda_model)
#' }
sem.topic.plot <- function(df.lda){

  df.topics <- tidy(df.lda, matrix = "beta")

  ## terms & topics
  df.terms <- df.topics %>%
    group_by(topic) %>%
    top_n(10, beta) %>%
    ungroup() %>%
    arrange(topic, -beta)
  # df.terms %>% print(n=60)


  reorder_within <- function (x, by, within, fun = mean, sep = "___", ...)
  {
    new_x <- paste(x, within, sep = sep)
    stats::reorder(new_x, by, FUN = fun)
  }


  ## plot the topics and terms
  df.terms %>%
    mutate(topic=as.factor(topic), term = reorder_within(term, beta, topic, sep="")) %>%
    ggplot(aes(term, beta, fill = topic)) +
    geom_col(show.legend = FALSE) + facet_wrap(~topic, scales = "free", labeller = "label_both") +
    xlab("Terms") + ylab("Topics") + coord_flip() + tidytext::scale_x_reordered() + scale_fill_grey()+
    theme(axis.text=element_text(size=10),
          axis.title=element_text(size=12))
}
```

Revision #4
Created 30 December 2024 19:42:56 by Admin
Updated 1 January 2025 19:35:50 by Admin