The R Package **sentometrics** to Compute, Aggregate and Predict with Textual Sentiment

David Ardia
Keven Bluteau
Samuel Borms
Kris Boudt

Université de Neuchâtel/Vrije Universiteit Brussel

International Conference on Data Science in Finance with R, Vienna

September 14, 2018

Outline

- Positioning sentiment analysis
  - What?
  - Relevance in finance
  - In R
- Introduction to the R package **sentometrics**
- Conclusion

Sentiment analysis

Natural Language Processing

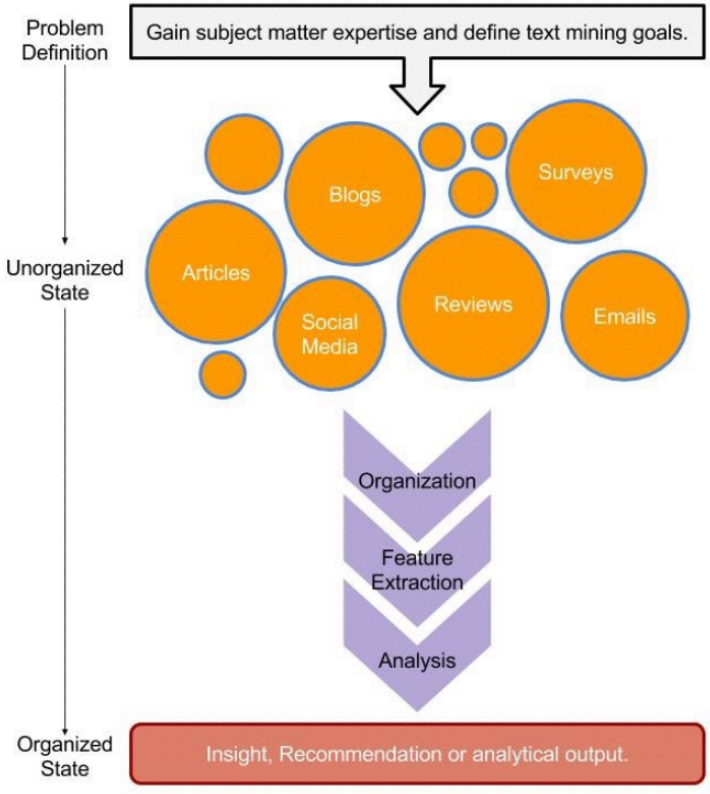
Natural language processing (NLP) is the broad field of processing, analyzing and understanding speech and texts.

It investigates:

- Speech recognition
- Text summarization
- Part of speech (POS) tagging
- Automatic question answering
- Translation
- ...
- **Text mining**

Text mining workflow

Text mining is the process of distilling actionable insights from text.



1 - Problem definition & specific goals

2 - Identify text to be collected

3 - Text organization

4 - Feature extraction

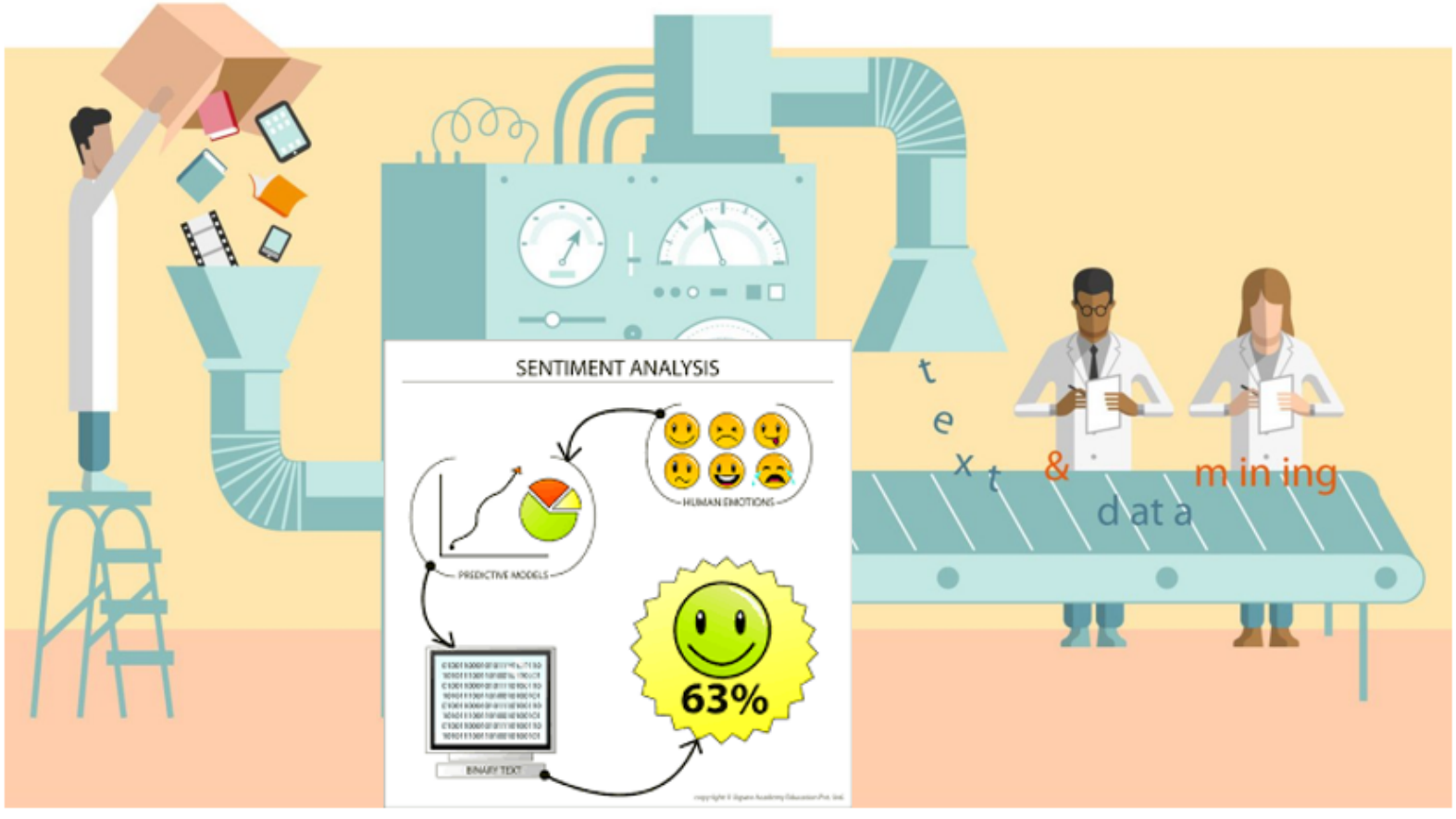
5 - Analysis

6 - Reach an insight, recommendation or output

Source: https://www.youtube.com/watch?v=mmz95b1k0J0

Sentiment analysis (i)

We focus on **textual sentiment analysis**.



Sentiment = polarity = tone = opinion = emotion = (semantic) orientation.

Sentiment analysis (ii)

Underlying purpose: **attaching scores to (parts of) texts on a scale from negative to positive**.

Textual sentiment scores comprise two elements:

- Polarity (positive or negative)
- Strength (how positive or how negative)

Usually, the largest emphasis is put on the first element.

Subjectivity analysis of texts is different (it doesn't say whether a text is negative, neutral or positive).

Sentiment analysis is complex

A number of examples:

1. You look pretty.

Positive, right?

2. You look beautiful.

Positive. Stronger? Less strong?

3. You look pretty, but you looked better last week. Today, it is raining. My cat is sick. She is feeling better than yesterday, though.

Neutral? Negative? Positive? Which parts?

Imagine texts of many sentences, and it is not so hard to realize that getting the sentiment of texts right is difficult.

Two main approaches to sentiment analysis

There are two main approaches to computing textual sentiment.

APPROACH 1: Machine learning (classification).

One way to cope with this complexity: training a large corpus annotated by humans = *supervised learning*.

Neural networks, support vector machines and other machinery. Can also be unsupervised.

APPROACH 2: Lexicon–based sentiment analysis.

Another way to deal with sentiment analysis: using sentiment lexicons.

Two main approaches to sentiment analysis

There are two main approaches to computing textual sentiment.

APPROACH 1: Machine learning (classification).

One way to cope with this complexity: training a large corpus annotated by humans = *supervised learning*.

Neural networks, support vector machines and other machinery. Can also be unsupervised.

APPROACH 2: Lexicon–based sentiment analysis.

Another way to deal with sentiment analysis: using sentiment lexicons.

Lexicons

A **lexicon** is a dictionary of words that have been carefully selected and scored in terms of polarity information.

A lexicon on its turn may be the result of a supervised learning process... They can be made application–specific.

Lexicon–based sentiment analysis starts from an (ordered) **bag–of–words**.

Textual sentiment analysis then boils down to matching words in a text with a lexicon, and adding up the associated polarity scores.

Valence shifters

However, not only polarized words itself make up sentiment, but also the surrounding words.

How does words in proximity of the word 'good' change the meaning of it?

- not good
- very good
- relatively good
- good, but
- not doing good
- ...

The words 'not', 'very' and others are called **valence shifters**. They affect (inverse, strengthen or weaken) neighbouring words with a given polarity.

In fact, the entire text structure plays their role in determining sentiment. The question of how complex to make sentiment analysis is one of a trade–off: **accuracy versus (computational) efficiency**. Application–specific lexicons augmented with valence shifters is our answer to this.

Financial applications of sentiment analysis

Strategic word positioning of CEO letters (Boudt and Thewissen, 2018).

Study on **return predictability** of news sentiment (Heston and Sinha, 2017).

Use of lexicon–based news measures (Caporin and Poli, 2017) and sentiment classification from internet message boards (Antweiler and Murray, 2004) helps to forecast stock market **volatility**.

Regression approach of market reaction to words (Jegadeesh and Wu, 2013).

Development of finance–specific word lists (Loughran and McDonald, 2011).

Prediction of earnings and stock returns mainly comes from negative words (Tetlock et al., 2008). Pessimism in media predicts market decline, extreme sentiment increases volume (Tetlock, 2007).

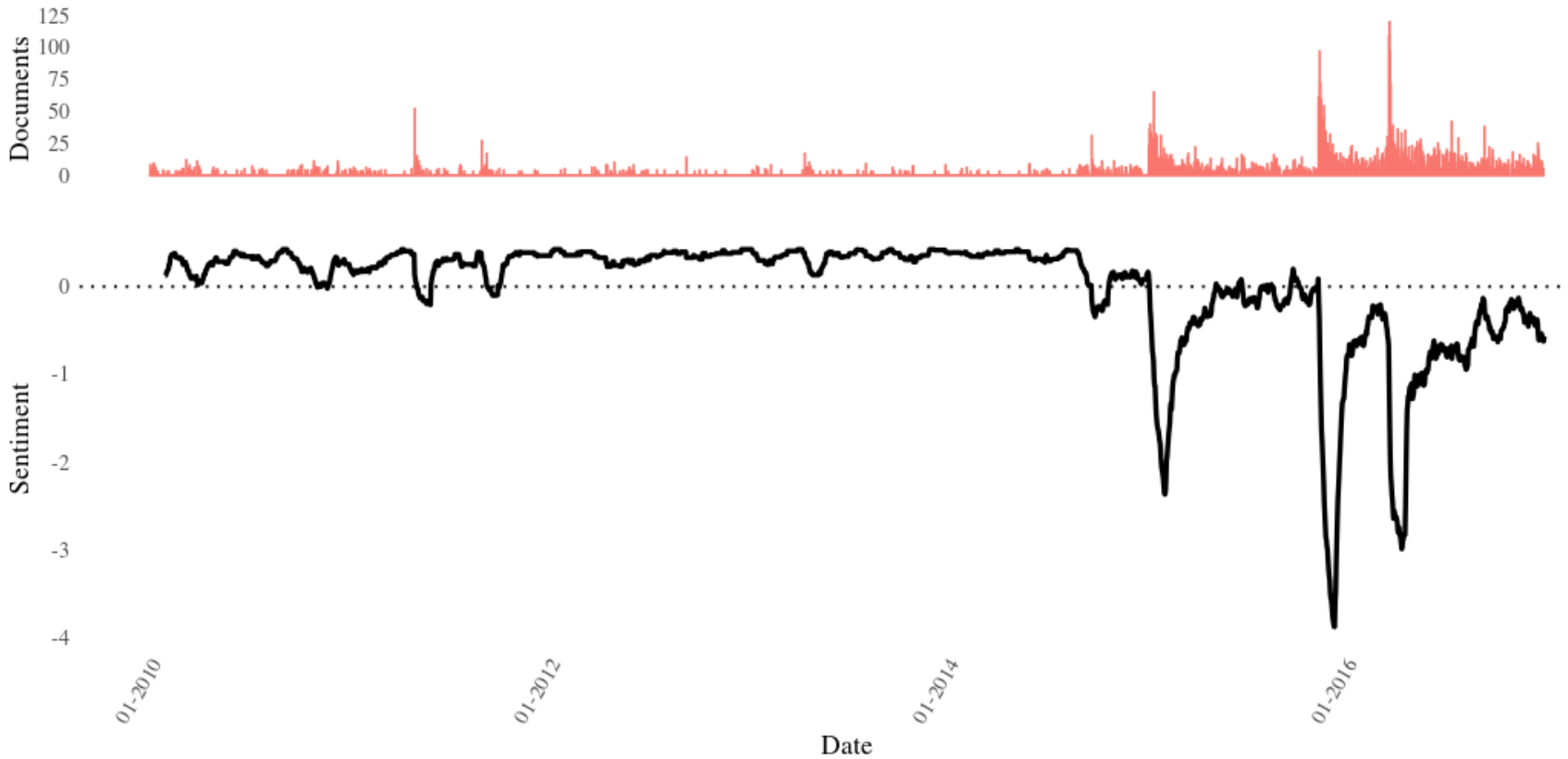
Impact often of a short–term nature. . .

. . . but research has shown that news is definitely not an ex–post sideshow!

Other applications

Predicting macroeconomic variables (e.g., Ardia et al., 2018, Beckers et al. 2017), monitoring company reputation (Amigó et al., 2014), tracking political preferences (Ceron et al. 2014).

Event detection.



Shiny app: https://sentometrics.shinyapps.io/textsent/.

Sentiment analysis with R

There are a number of packages in R which you can use to do sentiment analysis.

General–purpose NLP/text mining packages:

- **tm**, **quanteda**, **qdap**, **tidytext**, **openNLP**, **cleanNLP**.

Specific sentiment analysis packages:

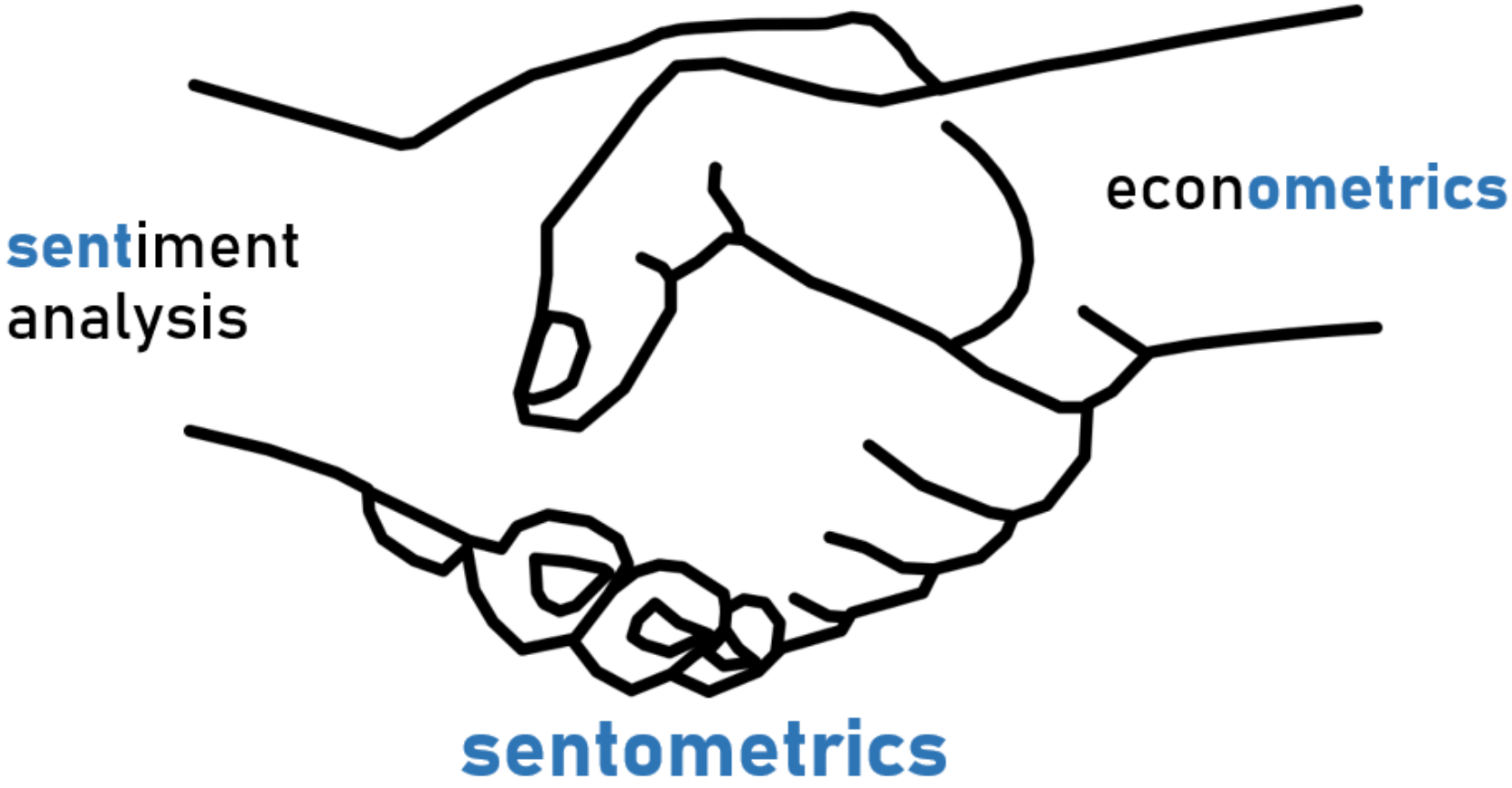
- **sentimentr**, **syuzhet**, **SentimentAnalysis**, **meanr**, **RSentiment**.

All these packages have something to offer, but none of it has it all:
user–friendliness, speed, flexibility, complexity, aggregation.

We have tried to fill this gap with **sentometrics**.

The R package sentometrics

[s|S]entometrics



sentometrics is the R package.

Sentometrics is the research effort:

- Vrije Universiteit Brussel + Université de Neuchâtel + industry.

What does the **sentometrics** package do?

Altogether, **sentometrics** integrates the *qualification* of sentiment from texts, the *aggregation* into different sentiment measures and the optimized *prediction* based on these measures. See `?sentometrics`.

The package accounts for the intrinsic challenge that, for a given text, sentiment can be computed in many ways, as well as the large number of possibilities to pool sentiment across texts and time.

Two related working papers (see SSRN):

- Ardia, Bluteau, Borms and Boudt (2018), *The R Package sentometrics to Compute, Aggregate and Predict with Textual Sentiment*.
- Ardia, Bluteau and Boudt (2018), *Questioning the news about economic growth: Sparse forecasting using thousands of news–based sentiment values*.

```
library("sentometrics") # version 0.5 soon on CRAN!
```

Typical package workflow(s)

The package has two typical usages.

USAGE #1

Step 1.1: Acquire and pre–process a selection of texts and generate features.

Step 1.2: Choose lexicons and compute textual sentiment.

Step 1.3: Aggregate into textual sentiment time series.

Step 1.4: Analyse (plot, manipulate, pinpoint important documents and dates).

USAGE #2

Step 2.1: Create the relevant sentiment measures (see Steps 1.1–1.3).

Step 2.2: Regress sparse model (in–sample, out–of–sample).

Step 2.3: Evaluate prediction performance and sentiment attributions.

USAGE #1: Step 1.1

Step 1.1: Acquire and pre–process a corpus, and generate features.

Step 1.2: Choose lexicons and compute textual sentiment.

Step 1.3: Aggregate into textual sentiment time series.

Step 1.4: Analyse (plot, manipulate, pinpoint important documents and dates).

Corpus construction

The texts and features have to be structured in a rectangular fashion. Every row is a document n from a given date t that is mapped to features through numerical values $w_{n,t}^k \in [0, 1]$, indicative of the relevance of a feature to a document (often binary), where the features are denoted by $k = 1, ..., K$.

Examples of features:

- news source
- topic
- entity (organisation, person)
- event/time frame
- geography
- ...

How a corpus looks in the package (i)

Load our built–in corpus, a `data.frame`. This is a collection of texts annotated in terms of relevance to the U.S. economy or not, from The Wall Street Journal and The Washington Post, and between 1995 and 2014.

```
data("usnews", package = "sentometrics")
```

How many texts are there?

```
dim(usnews)
```

```
## [1] 4145    7
```

What are the columns?

```
colnames(usnews)
```

[1] “id” “date” “texts” “wsj” “wapo”
[6] “economy” “noneconomy”

How a corpus looks in the package (ii)

For this corpus, the features are structured in a binary way:

```
usnews[99:108, -c(1:3)]
```

```
##      wsj wapo economy noneconomy
## 99     1    0       0          1
## 100    1    0       1          0
## 101    0    1       0          1
## 102    1    0       1          0
## 103    0    1       0          1
## 104    0    1       1          0
## 105    0    1       0          1
## 106    1    0       0          1
## 107    1    0       0          1
## 108    0    1       0          1
```

A value of 1 means the text is relevant to that particular feature, and 0 if not.

How a corpus looks in the package (iii)

Let's pick out one text:

```
usnews[["texts"]][927]
```

[1] "A jittery Wall Street pummeled stock prices for most of the trading session yesterday before bargain hunters helped pull prices back up toward the close. The Dow Jones industrials lost 27.86 points, or 0.27 percent, to close at 10,275.53, but that was far above the intra-day low of about 10,080. Fear is the issue here, an innate insecurity that too much of a good thing has gone on for long, said Michael Farr, president and chief investment officer of Farr, Miller & Washington, a D.C.-based financial-consulting firm. Bellwethers Microsoft and Intel took big hits. The software giant touched an intra-day low of 89-116 but finished at 91-716, a little lower than Monday's close of 92 18. Intel at one point dipped to 75 14 before recovering to finish at 77 12. On Monday it had closed at 78-316. Investors, concerned about another round of interest-rate increases, have been driving stock prices down for the past few weeks. The Federal Reserve's rate-policy-setting committee will meet next week to evaluate the economy, though most Fed watchers believe there is little chance the Fed will raise rates next week."

One last step in preparing your corpus

We wrap this corpus `data.frame` into a more formal corpus structure, relying on the **quanteda** package. It goes like this:

```
corpus <- sento_corpus(usnews)
```

Your corpus is now a *sentocorpus* object, on top of **quanteda**'s *corpus* class.

```
class(corpus)
```

```
## [1] "sentocorpus" "corpus"      "list"
```

You need to pass your corpus to this function to make it **sentometrics**–ready.

Feature generation

You will have to think about creating additional features that could be informative for your analysis.

Machine learning is useful here, for example topic modelling or entity recognition.

We make available a ad–hoc approach for feature generation, called **keywords occurrence search** (you can specify regex patterns if you like):

```
keywords <- list(politics = c("democrat", "republican",
                              "Obama", "election", "president"),
                 war = c("war", "terrorist attack"))
corpus <- add_features(corpus, keywords = keywords)
```

To see how many documents have been given the indication of 1:

```
sum(corpus$documents[, "politics"])
```

```
## [1] 624
```

USAGE #1: Step 1.2

Step 1.1: Acquire and pre–process a corpus, and generate features.

Step 1.2: Choose lexicons and compute textual sentiment.

Step 1.3: Aggregate into textual sentiment time series.

Step 1.4: Analyse (plot, manipulate, pinpoint outliers).

Built–in lexicons

The package provides a number of built–in lexicons and valence word lists.

```
data("list_lexicons", package = "sentometrics")
data("list_valence_shifters", package = "sentometrics")
```

Below how the well–known Loughran and McDonald (2011) word list looks like:

```
list_lexicons[["LM_en"]][1:7, ]
```

```
##                x     y
##           <char> <num>
## 1:       abandon    -1
## 2:     abandoned    -1
## 3:    abandoning    -1
## 4:   abandonment    -1
## 5: abandonments    -1
## 6:      abandons    -1
## 7:     abdicated    -1
```

Self–made lexicons

It is also relatively easy to define your own lexicons. Make a two–column `data.frame` with the words and the scores, as shown here:

```
myLexicon <- data.frame(w = c("uncertainty", "anxiety", "crisis",
                              "concern", "distrust", "worries"),
                        s = c(-1, -2, -3, -2, -0.5, -2))
```

This allows you to make lexicons that need to capture specific words and have specific scores, which could be more relevant than the generic lexicons.

Preparing the lexicons

All the lexicons (and up to one valence word list) to use for the sentiment analysis are cross-checked and made ready by the `setup_lexicons()` function.

We consider two built–in lexicons, our own, and a bunch of lexicons from the R package **lexicon**, and include valence shifters:

```
lexIn <- c(
  list(myLexicon = myLexicon),
  list_lexicons[c("GI_en", "LM_en")],
  list(
    nrc = lexicon::hash_sentiment_nrc,
    huliu = lexicon::hash_sentiment_huliu,
    sentiword = lexicon::hash_sentiment_sentiword,
    jockers = lexicon::hash_sentiment_jockers,
    senticnet = lexicon::hash_sentiment_senticnet
  )
)
valIn <- list_valence_shifters[["valence_en"]]
lex <- setup_lexicons(lexIn, valIn)
```

The output is a classed `list` called *sentolexicons* .

Our sentiment calculation: some notation

A set of documents $d_{n,t}$ for $n = 1, ..., N_t$ and time points $t = 1, ..., T$, where N_t is the total number of documents at time t. The time points can be at a daily, weekly, monthly or yearly frequency.

Every text is mapped to the features through values $w_{n,t}^k$.

For every lexicon $l = 1, ..., L$, each document $d_{n,t}$ gets assigned a sentiment score $s_{n,t}^{\{l\}}$.

These scores are subsequently multiplied by the feature weights to obtain lexicon– and feature–specific sentiment scores. We have: $s_{n,t}^{\{l,k\}} \equiv s_{n,t}^{\{l\}} \times w_{n,t}^k \equiv (\sum_{i=1}^{Q_d} \omega_i v_i s_{i,n,t}^{\{l\}}) \times w_{n,t}^k$. The weights ω_i define the importance of each word i in document n at time t, v_i is the valence shifting impact, Q_d is the total number of words in the document.

We support three levels of sentiment complexity

1. Simple unigram matching [*unigrams* approach]
   - $v_i = 1$

2. Integration of valence shifters before a polarized word [*bigrams* approach]
   - deals with 'not good', 'hardly successful' and 'very bad'
   - valence shifters have a score column `"y"`

3. Integration of valence shifters if in cluster [*clusters* approach]
   - deals with 'not a good economy' (i.e., looks for valence shifters in a fixed cluster around a polarized word)
   - valence shifters have a type column `"t"` (negators, amplifiers, deamplifiers)

Compute sentiment only

To simply compute sentiment:

```
s <- compute_sentiment(quanteda::texts(corpus),
                       lex, # complexity 2
                       how = "counts")
s[1:6, 1:7]
```

```
##    word_count myLexicon GI_en LM_en   nrc huliu sentiword
##         <num>     <num> <num> <num> <num> <num>     <num>
## 1:        213         0    -2    -2    -5    -5    0.7681
## 2:        110         0     6     2     0     2    0.4583
## 3:        202         0     1    -3    -2    -1   -1.5750
## 4:        153         0     1     2     3     3    3.2988
## 5:        245         0     2    -7     6    -2   -5.3576
## 6:        212         0     3    -3     7    -1   -1.5417
```

This goes fast! Calculation for 100,000 documents runs c. 20 seconds, irrespective of size and amount of lexicons.

USAGE #1: Step 1.3

Step 1.1: Acquire and pre–process a corpus, and generate features.

Step 1.2: Choose lexicons and compute textual sentiment.

Step 1.3: Aggregate into textual sentiment time series.

Step 1.4: Analyse (plot, manipulate, pinpoint outliers).

Aggregation formally

Three main aggregations are performed:

1. *Within–document*.

See sentiment calculation.

2. *Across–document*.

Aggregate document sentiment scores per time point, as $s_t^{\{l,k\}} \equiv \sum_{n=1}^{N_t} \theta_n s_{n,t}^{\{l,k\}}$. The weights θ_n define the importance of each document n at time t.

3. *Across–time*.

Smooth the time series, as $s_u^{\{l,k,b\}} \equiv \sum_{t=t_\tau}^{u} b_t s_t^{\{l,k\}}$, where $t_\tau \equiv u - \tau + 1$. The time weighting schemes $b = 1, ..., B$ go with different values for b_t, with a time lag of τ. The first $\tau - 1$ observations are dropped, such that $u = \tau, ..., T$ becomes the time index for the ultimate time series.

The total number of obtained time series is equal to $P \equiv L \times K \times B$. Define $s_u^p \equiv s_u^{\{l,k,b\}}$ as a single sentiment index, $p = 1, \ldots, P$.

Aggregation in the package

The entire aggregation setup is specified by means of the `ctr_agg()` function, including the within–document aggregation needed for the sentiment analysis.

Your first resort is always `?ctr_agg`.

Aggregation within documents

The `howWithin` argument defines how sentiment is aggregated within the same document, that is, it sets the weights ω_i. Options:

- `"counts"` (difference between the number of positive and negative words)
- `"proportional"` (... divided by the total number of words)
- `"proportionalPol"` (... divided by the total number of polarized words)

Aggregation across documents

The `howDocs` argument defines how sentiment is aggregated across all documents at the same date (or frequency), that is, it sets the weights θ_n.

Options:

- `"equal_weight"` (gives same weight to every document)
- `"proportional"` (gives higher weights to documents with more words)

The `do.ignoreZeros` argument can be used to ignore documents with zero sentiment in the computation of the across–document weights. This primarily avoids the incorporation of documents not relevant to a particular feature, which could lead to a bias of sentiment towards zero.

Aggregation across time (i)

The `howTime` argument defines how to smoothen the time series, acknowledging that sentiment at a given point is at least partly based on sentiment and information from the past. The weighting schemes define the different b_t values.

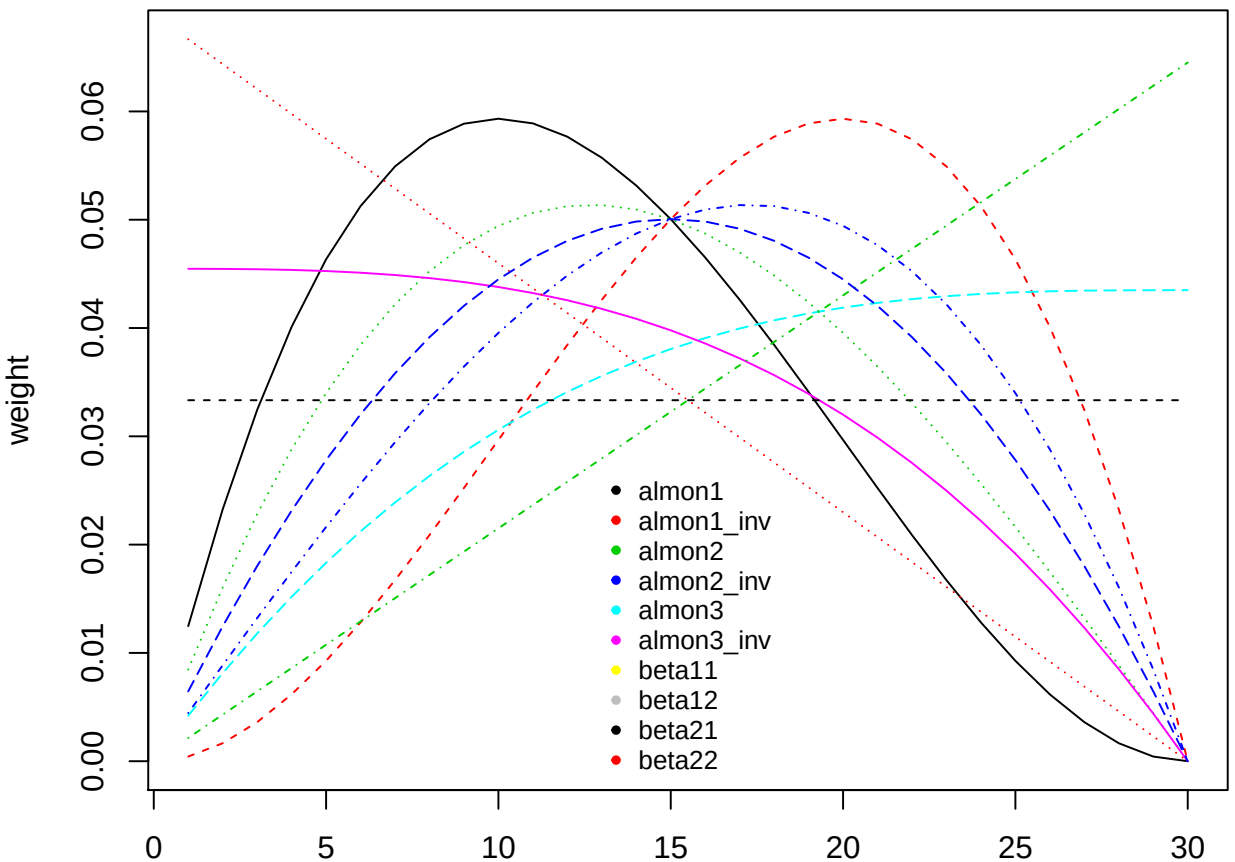
Options:

- `"equal_weight"` (simple moving average)
- `"linear"` (linear moving average)
- `"exponential"` (exponential moving average)
- `"almon"` (Almon polynomial weighting)
- `"beta"` (Beta weighting)
- `"weights"` (user–defined weights as a named `data.frame`)

The `lag` argument defines τ. The `fill` argument specifies to add in dates (or *not*) for which no documents were available, such that the time aggregation weighs over the latest consecutive (or *available*) dates.

Aggregation across time (ii)

```
w <- cbind(almons(30, 1:3), betas(30, 1:2, 1:2))
matplot(w, type="l", ylab="weight")
legend("bottom", names(w), col=1:10, pch=20, bty="n", cex=0.8)
```



An example aggregation control

Can you make sense of the options we selected here?

```
ctrAgg <- ctr_agg(howWithin = "counts",
                  howDocs = "proportional",
                  howTime = c("equal_weight", "linear", "almon"),
                  do.ignoreZeros = TRUE,
                  by = "month",
                  fill = "latest",
                  lag = 6,
                  ordersAlm = 1:3,
                  do.inverseAlm = TRUE)
```

Chaining it all together

You now have:

- A corpus
- One or more lexicon(s)
- An extensive aggregation specification

The `sento_measures()` function takes in these three elements. It performs both the sentiment calculation and aggregation into time series, outputting a *sentomeasures* object, structured as a `list` of various elements:

```
sentMeas <- sento_measures(corpus, lex, ctrAgg)
get_measures(sentMeas)[1:4, 1:2]
```

```
##          date myLexicon--wsj--equal_weight
##        <Date>                         <num>
## 1: 1995-06-01                        -1.812
## 2: 1995-07-01                        -1.333
## 3: 1995-08-01                        -1.000
## 4: 1995-09-01                        -1.000
```

USAGE #1: Step 1.4

Step 1.1: Acquire and pre–process a corpus, and generate features.

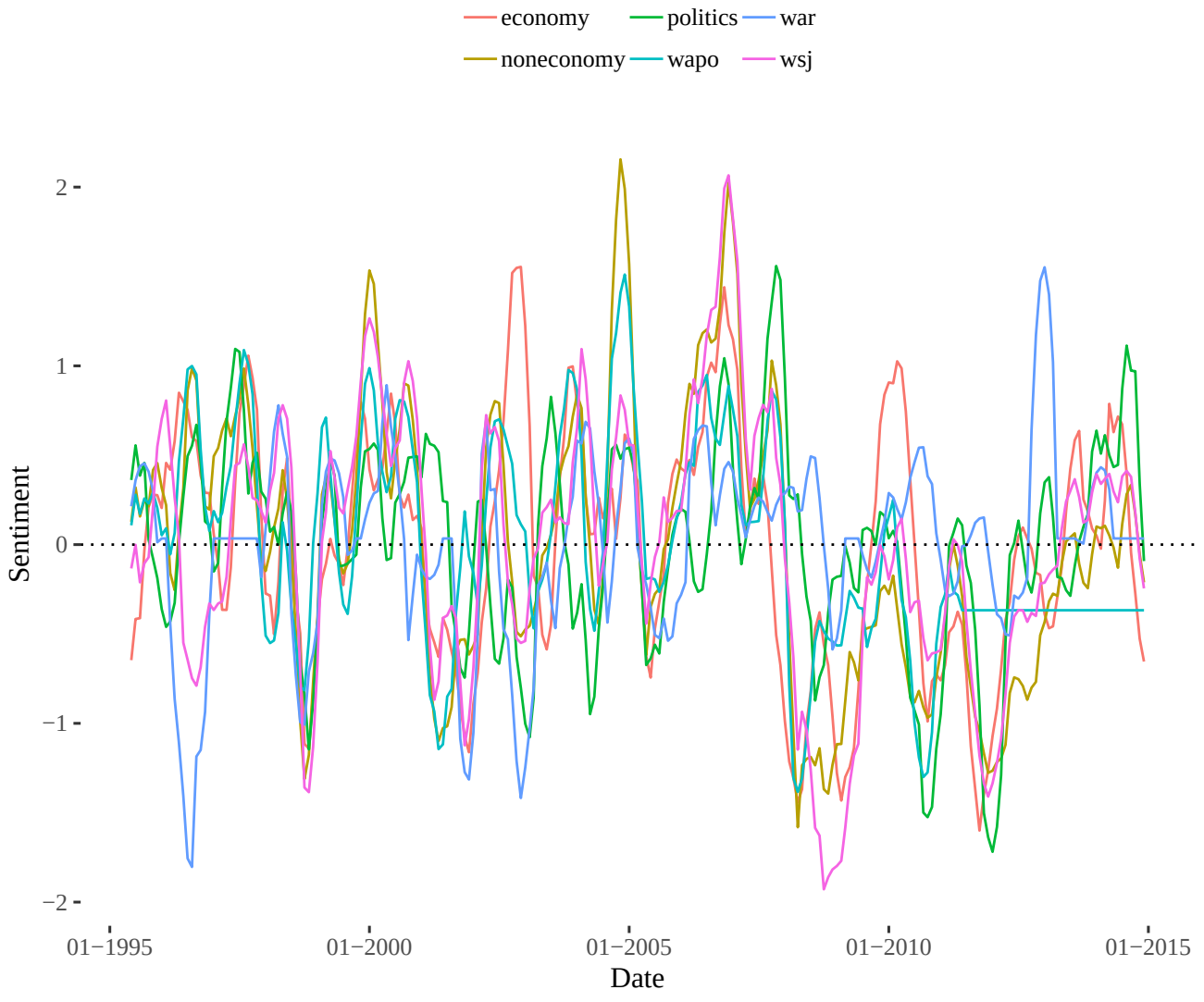
Step 1.2: Choose lexicons and compute textual sentiment.

Step 1.3: Aggregate into textual sentiment time series.

Step 1.4: Analyse (plot, manipulate, pinpoint outliers).

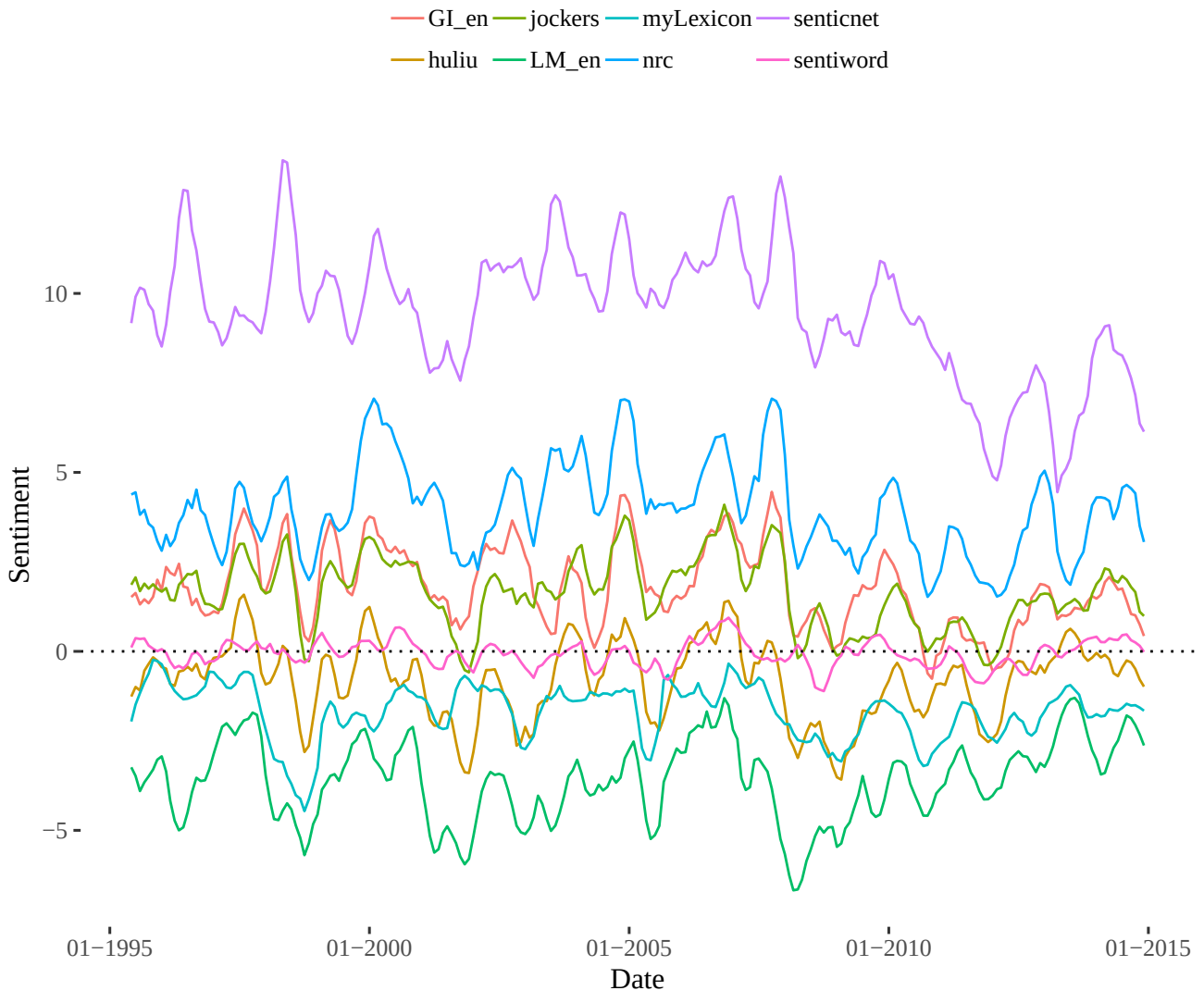
Visual inspection of the sentiment time series (i)

```
plot(scale(sentMeas), group = "features")
```



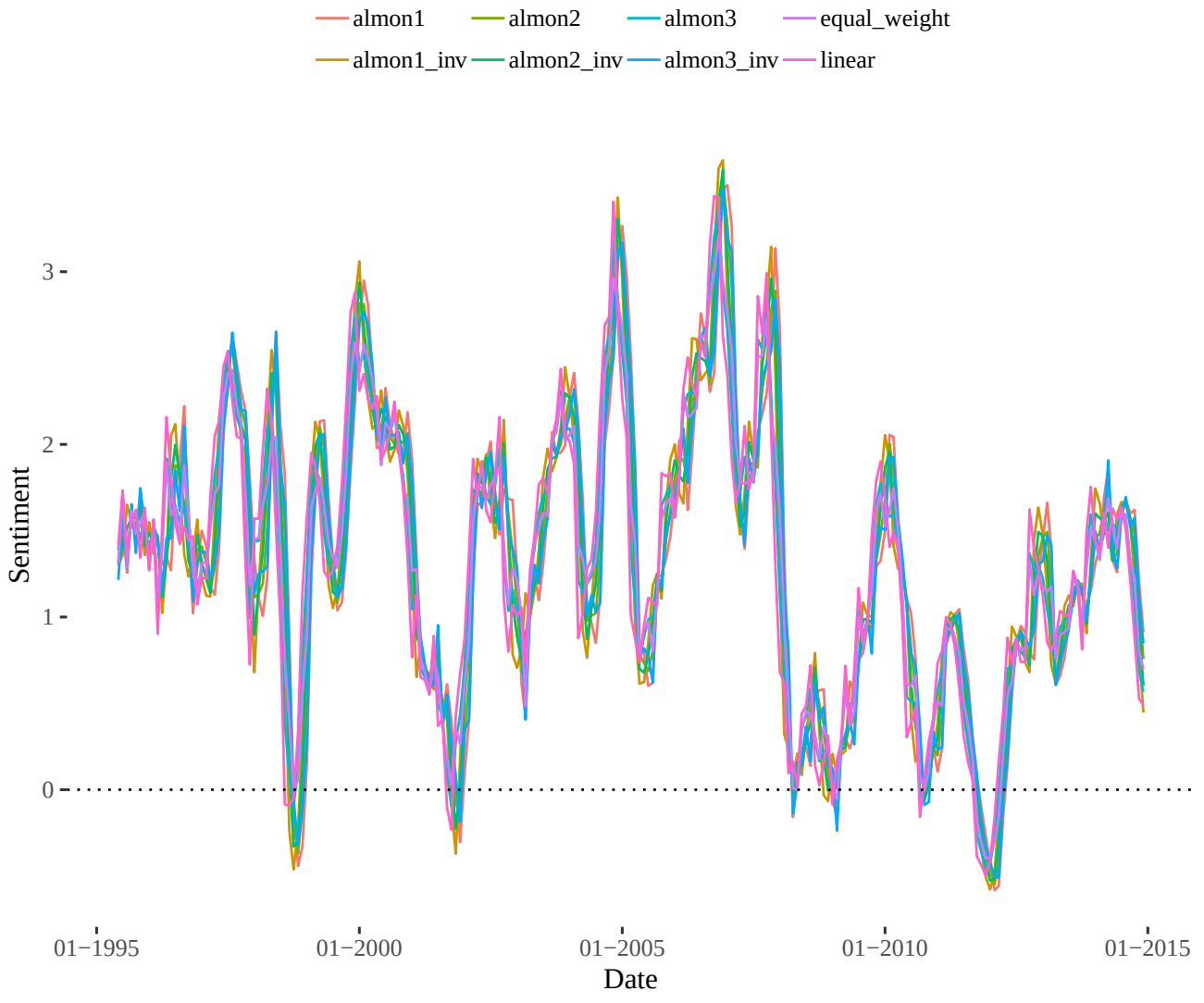
Visual inspection of the sentiment time series (ii)

```
plot(sentMeas, group = "lexicons")
```



Visual inspection of the sentiment time series (iii)

```
plot(sentMeas, group = "time")
```



Further manipulation

Functions to manipulate a *sentomeasures* object: `scale()`, `diff()`, `measures_delete()`, `measures_select()`, `measures_fill()`, `measures_subset()`, `measures_merge()`, and `measures_global()`.

The `peakdocs()` function gets the dates when sentiment was most extreme:

```
peaks <- peakdocs(sentMeas, corpus, n = 5, type = "negative")
peaks[["dates"]]
```

[1] “1996-06-01” “1996-05-01” “1996-07-01” “2007-12-01” “2007-10-01”

USAGE #2: Step 2.2

Step 2.1: Create the relevant sentiment measures *(see Steps 1.1–1.3)*.

Step 2.2: Regress sparse model (in–sample, out–of–sample).

Step 2.3: Evaluate prediction performance and sentiment attributions.

Sparse regression (i)

Remember, $P \equiv L \times K \times B$. This could be large.

When $P >> N$, for N the sample size, we face a **high–dimensional** problem which regular least squares cannot solve reliably. . .

Sparse regression (ii)

One way to deal with sparse regressions is to use the **elastic net** specification. It regularizes the coefficient's estimation and does variable selection.

Let $\boldsymbol{x}_u$ be a vector of non–sentiment explanatory variables. The linear regression equation incorporating all sentiment measures:

$$y_{u+h} = \delta + \boldsymbol{\gamma}^\top \boldsymbol{x}_u + \beta_1 s_u^1 + ... + \beta_p s_u^p + ... + \beta_P s_u^P + \epsilon_{u+h} = \delta + \boldsymbol{\gamma}^\top \boldsymbol{x}_u + \boldsymbol{\beta}^\top \boldsymbol{s}_u + \epsilon_{u+h}.$$

The elastic net objective function (on standardized variables):

$$\min_{\tilde{\delta}, \widetilde{\boldsymbol{\gamma}}, \widetilde{\boldsymbol{\beta}}} \left\{ \frac{1}{N} \sum_{u=\tau}^{T} (y_{u+h} - \tilde{\delta} - \widetilde{\boldsymbol{\gamma}}^\top \widetilde{\boldsymbol{x}}_u - \widetilde{\boldsymbol{\beta}}^\top \widetilde{\boldsymbol{s}}_u)^2 + \lambda \left[\alpha \left\| \widetilde{\boldsymbol{\beta}} \right\|_1 + (1-\alpha) \left\| \widetilde{\boldsymbol{\beta}} \right\|_2^2 \right] \right\}.$$

The $\lambda \geq 0$ parameter defines the level of regularization. When λ is equal to zero, the problem reduces to simple least squares estimation.

The parameter $0 \leq \alpha \leq 1$ defines the trade–off between the Ridge, ℓ_2, and the LASSO, ℓ_1, regularizaton, respectively when it is equal to 0 and 1.

USAGE #2: Step 2.2

Step 2.1: Create the relevant sentiment measures (see Steps 1.1–1.3).

Step 2.2: Regress sparse model (in–sample, out–of–sample).

Step 2.3: Evaluate prediction performance and sentiment attributions.

Get some data

The package's built–in target variable is the Economic Policy Uncertainty (EPU) index (Baker et al., 2016). This indicator is a normalized index of the number of news articles, from ten large U.S. newspapers, discussing economic policy uncertainty.

```
data("epu", package = "sentometrics")
y <- epu[epu$date %in% get_dates(sentMeas), "index"]
```

The example is foremost illustrative.

Define the model structure

Set the type of model, type of calibration and out–of–sample parameters.

```
ctrIter <- ctr_model(model = "gaussian",
                     type = "BIC",
                     h = 1,
                     do.difference = FALSE,
                     alphas = c(0, 0.5, 1),
                     do.iter = TRUE,
                     nSample = 60,
                     do.progress = FALSE)
```

You can run a linear (`gaussian`) or a logistic (`binomial`, `multinomial`) model.

The calibration of the hyperparameters λ and α is done based on an information criterion or cross–validation.

Run it

Run the rolling out–of–sample analysis. The package **glmnet** is used as backend.

```
datesIn <- tail(get_dates(sentMeas), -1)
sentMeasIn <- measures_subset(sentMeas, date %in% datesIn)
out <- sento_model(sentMeasIn,
                   x = data.frame(lag = head(y, -1)),
                   y = tail(y, -1),
                   ctr = ctrIter)
```

The output out is an object of class *sentomodeliter*.

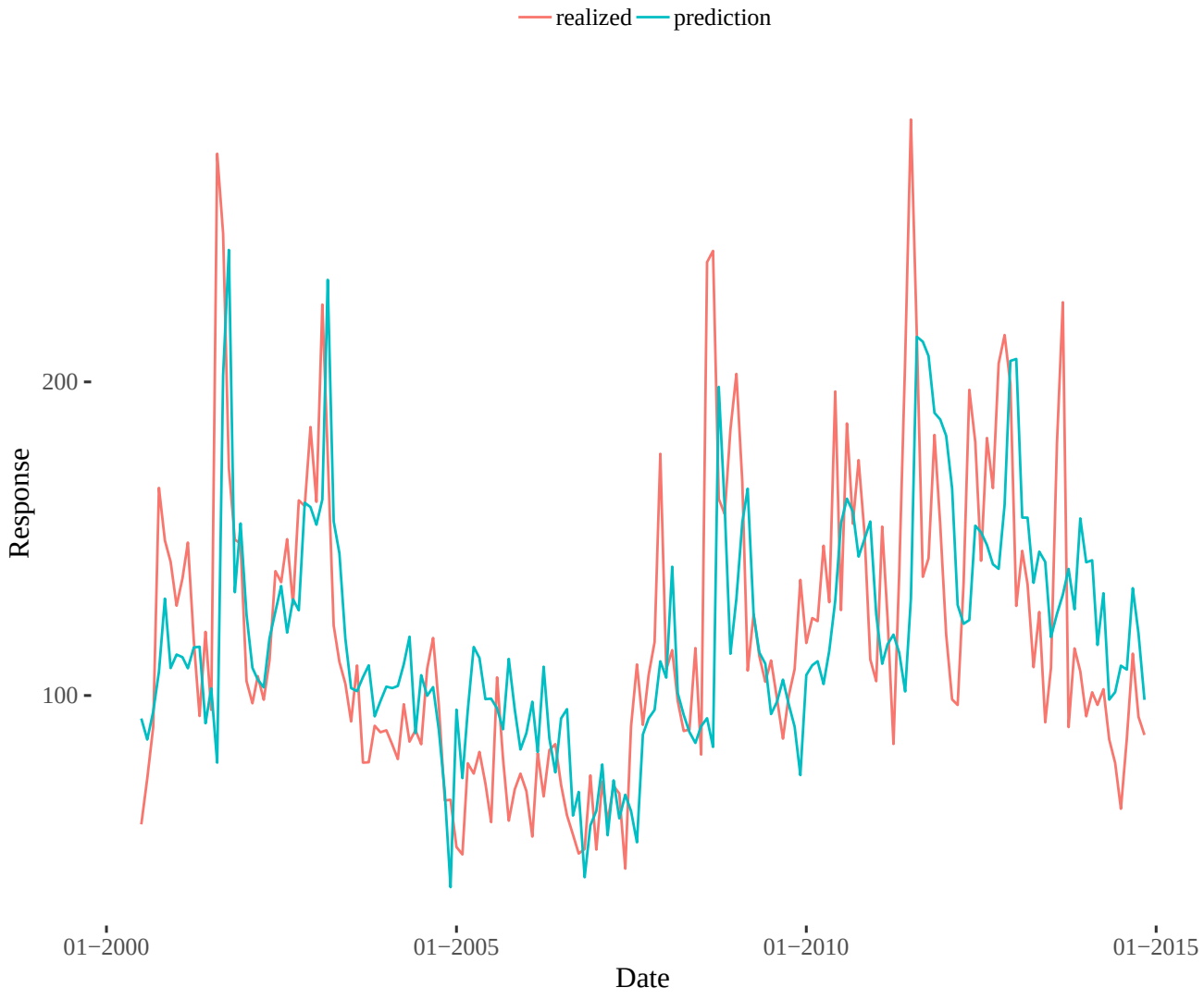
Model summary

```
summary(out)
```

```
## Model specification
## - - - - - - - - - - - - - - - - - - - - -
##
## Model type: gaussian
## Calibration: via BIC information criterion
## Sample size: 60
## Total number of iterations/predictions: 173
## Optimal average elastic net alpha parameter: 0.7
## Optimal average elastic net lambda parameter: 840.1
##
## Out-of-sample performance
## - - - - - - - - - - - - - - - - - - - - -
##
## Mean directional accuracy: 42.44 %
## Root mean squared prediction error: 41.13
## Mean absolute deviation: 28.56
```

Plot it

```
plot(out)
```



Attribution

Sentiment attribution is a qualitative model "diagnostic" to help capture the (time–varying) meaning of your model.

Based on the estimated coefficients $\hat{\beta}$, we can compute every corresponding underlying dimension's sentiment attribution to a given prediction. For example:

Prediction $=$ attribution feat. $1 + \ldots +$ attribution feat. $K +$ other.

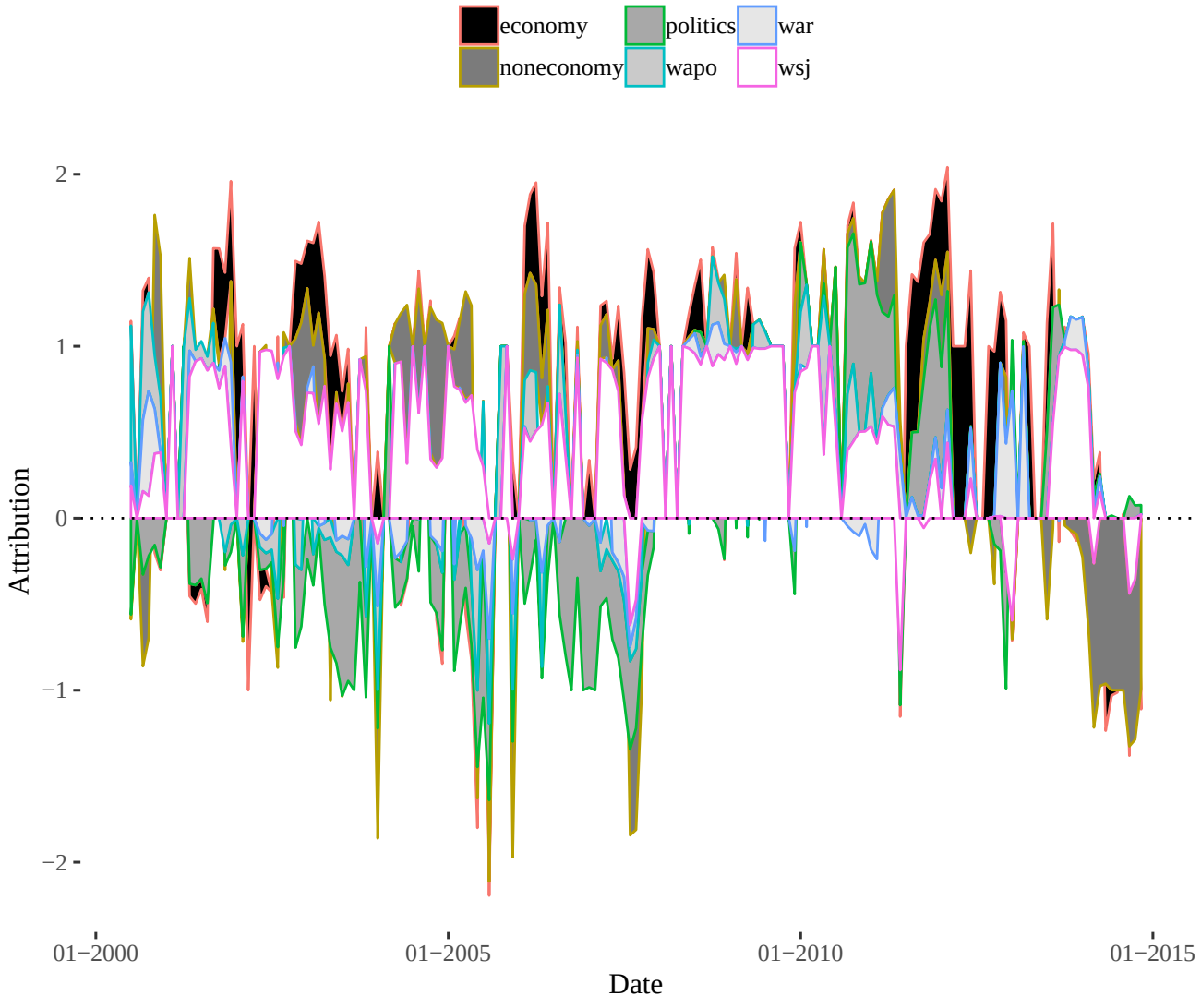
Attribution can be broken down for all features, lexicons, time weighting schemes, time lags and individual documents.

Compute sentiment attributions as follows:

```
attrib <- retrieve_attributions(out,
                                sentMeasIn,
                                do.normalize = TRUE)
```

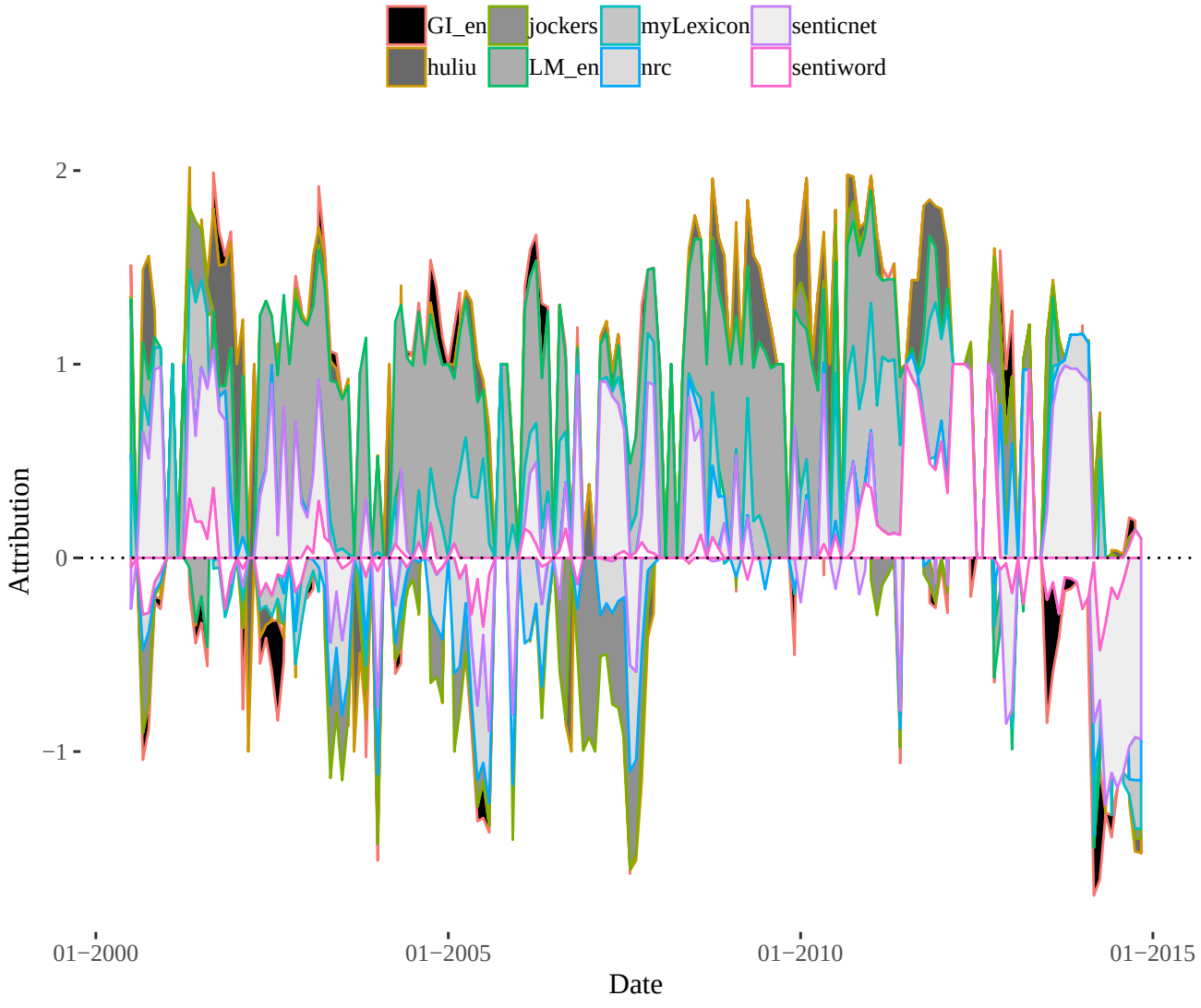
Plot the attributions (i)

```
plot_attributions(attrib, group = "features")
```



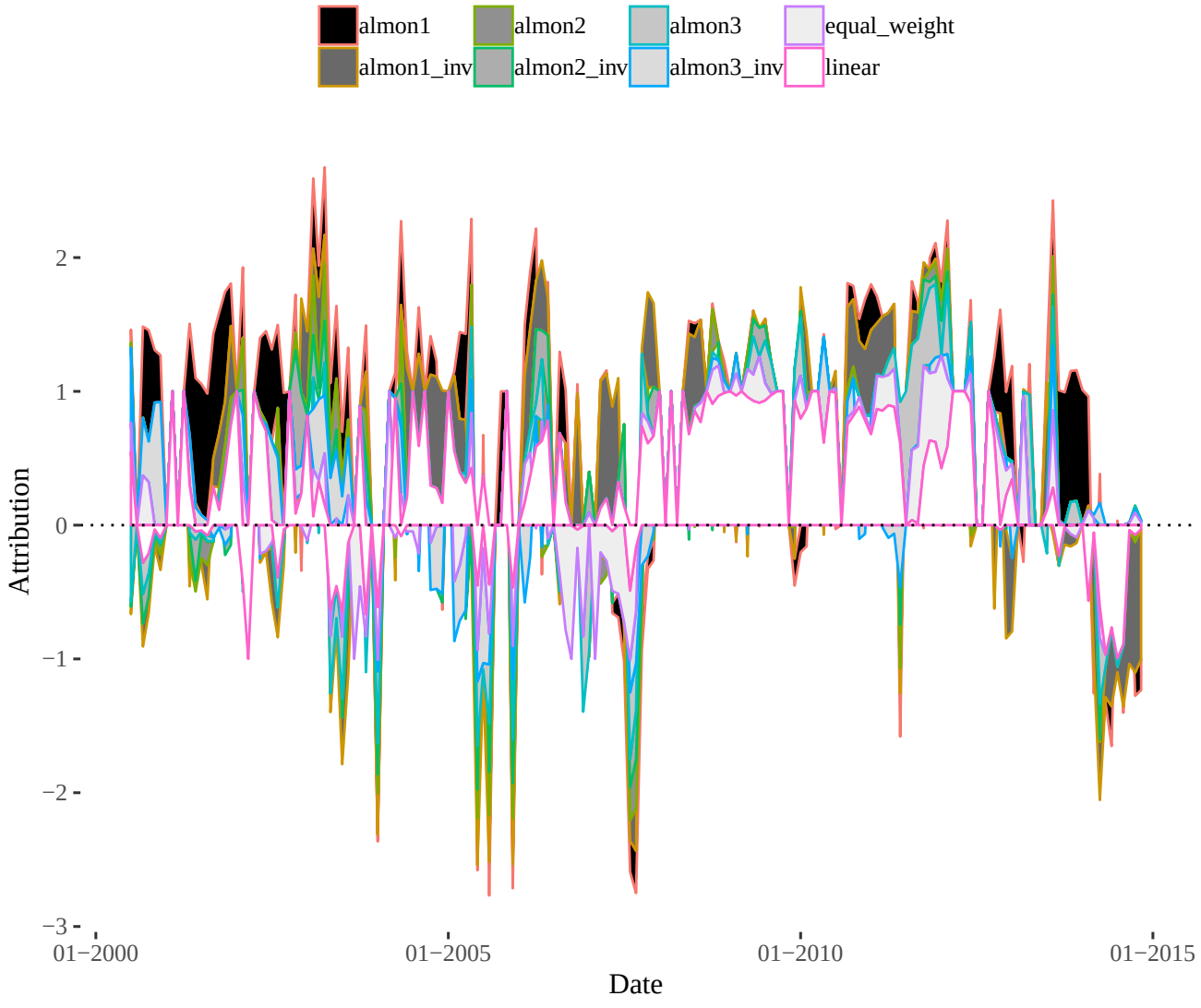
Plot the attributions (ii)

```
plot_attributions(attrib, group = "lexicons")
```



Plot the attributions (iii)

```
plot_attributions(attrib, group = "time")
```



Conclusion

This is a toolbox

Now it is up to you. Play with the degrees of freedom:

- Input corpus (news articles, Twitter data, blog posts and more)
- Corpus features
- Frequency (daily, weekly, monthly, yearly)
- Lexicons (general, domain–specific, binary, weighted)
- Aggregation into time series
- Classification or regression
- Enrich existing models with sentiment variables

Many applications to think of!

Thanks for your time. Any questions?