

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

David Ardia
University of Neuchâtel
Laval University

Keven Bluteau
University of Neuchâtel
Vrije Universiteit Brussel

Samuel Borms
University of Neuchâtel
Vrije Universiteit Brussel

Kris Boudt
Vrije Universiteit Brussel
Vrije Universiteit Amsterdam

Abstract

We provide a hands-on introduction to optimized textual sentiment indexation using the **sentometrics** package. Driven by data availability, sentiment analysis of texts is increasingly used to capture the information value of qualitative data in relation to quantitative variables. The **sentometrics** package allows to compute sentiment scores of multiple texts at once, to aggregate the textual sentiment scores into multiple time series, and use these time series to predict any other variable. The workflow of the package is illustrated using a built-in corpus of texts from two major U.S. journals to predict an index of economic policy uncertainty.

Keywords: penalized regression, R, **sentometrics**, textual sentiment, time series.

1. Introduction

Individuals, firms, governments and other organizations regularly take in written information from various sources to inform themselves for their future decisions. The sentiment transmitted through texts is therefore a prime candidate driver of many variables. Texts, qualitative by nature, need to be quantified first before they can be related to other quantitative data. A popular way to do so is by assigning a sentiment score that represents the polarity of a text. This is called *textual sentiment*. Over the past decade, the number of academic articles that transform texts into quantitative variables using this approach has steadily grown. The creation of lexicon-based news measures to forecast stock market volatility (Caporin and Poli 2017), the usage of sentiment in texts to improve inflation predictions (Beckers *et al.* 2017), and the application of sentiment analysis to track political preferences (Ceron *et al.* 2014) are recent examples respectively in the fields of finance, economics and politics.

The **sentometrics** software package positions itself as both supplementary and integrative with respect to the current text mining R universe. It combines the strengths of text mining and econometric analysis, bringing forward a well-defined modelling workflow, specifically applied to studying the evolution of textual sentiment and its impact on other quantities. The adopted workflow was developed by Ardia *et al.* (2017), who show that the incorporation of textual sentiment indices allows to better predict Germany's industrial production growth compared to using traditional survey-based sentiment indices only. The package extends most typical text mining workflows by (i) stretching out fine-grained textual sentiment into the construction of various sentiment time series (going by the synonyms of measures or indices), and (ii) using these measures in a flexible prediction setup. The **sentometrics** package greatly benefits from three major R packages: **quanteda** (Benoit *et al.* 2017) for corpus construction and corpus manipulation for sentiment analysis, **data.table** (Dowle and Srinivasan 2017) for the aggregation of textual sentiment into time series, and **glmnet** (Friedman *et al.* 2010) and **caret** (Kuhn 2017) for sparse model estimation.

The remainder of the paper is structured as follows: Section 2 introduces the methodology behind the **sentometrics** package, Section 3 explains the main control functions and illustrates the typical package workflow, and Section 4 concludes.

2. The framework behind sentometrics

There are five main steps in the workflow, from corpus construction to model estimation. The steps are explained below, together with the appropriate functions from the **sentometrics** package that relate to each step. The next section demonstrates how to use the functions. We minimize the mathematical details to clarify the exposition and stay close to the actual implementation. We refer to Ardia *et al.* (2017) for a more elaborate overview of the methodology.

Step 1: Acquire and pre-process a selection of texts and generate relevant features

The first step consists in the acquisition of a corpus of texts, of any size and from any source available. Texts could be crawled from the web or retrieved from news databases. A minimal requirement is that every text has a timestamp associated. These texts are ideally cleaned

such that graphical and web-related elements (*e.g.*, HTML tags) are removed. This results in a set of documents $d_{n,t}$ for $n = 1, \dots, N_t$ and time points $t = 1, \dots, T$, where N_t is the total number of documents at time t .

Secondly, *features* have to be defined and mapped to the documents. Features can come in many forms: news sources, entities (individuals, companies or countries discussed in the texts), or text topics. The mapping implicitly permits to subdivide the corpus into many smaller groups with a common interest. Many data providers tend to enrich their textual data with information that can be used as features. If this is not the case, topic modelling or entity recognition techniques are valid alternatives.¹ Human classification or keyword(s) occurrence searches are simpler options. The extraction and inclusion of features is an important part of the analysis and should be related to the variable that is meant to be predicted. The texts and features have to be structured in a rectangular fashion. Every row is a document that is mapped to the features through numerical values $w_{n,t}^k$ where the features are denoted by $k = 1, \dots, K$. The values are indicative of the relevance of a feature to a document; they are often kept binary as a way to select which documents belong to which features.

When the rectangular data structure is set up, it needs to be passed to the `sentocorpus()` function to create a more formal corpus, resulting in a `sentocorpus` object. The reason for this intermediate step is twofold: it controls whether all corpus requirements for further analysis are met, and it allows to perform more efficient operations on the corpus both for corpus manipulation and sentiment calculation. The `add_features()` function can be used to add or generate new features, as shown in the illustration. When the corpus is constructed, it is up to the user to decide which texts have to be kept for the actual sentiment analysis.

Step 2: Choose lexicons and compute textual sentiment

For every document, a sentiment score must be computed. There are many ways to do so, from straightforward to very complex. We take a *lexicon*-based approach, augmented with *valence shifters*. Lexicon-based analysis looks for words (*unigrams*) that are included in a pre-defined word list of polarized (positive and negative) words. Textual sentiment is then a weighted summation of all scores associated to these detected words, per document. Incorporating valence shifters goes one step beyond, taking a *bigram* perspective. The impact of the word appearing before the detected word is evaluated as well.² A common example is ‘not good’, which under the default approach would get a score of 1 (‘good’), but now ends up having a score of -1 due to the negator ‘not’. This procedure is by far perfect from a natural language processing standpoint, but has proven efficient and powerful in many applications.

The **sentometrics** package benefits from built-in word lists in English, French and Dutch, with the latter two in most cases as a translation. However, all lexicons are applied to the corpus, without discrimination based on language. At least one lexicon and possibly one list of valence shifters need to be provided, which is done through the `setup_lexicons()` function.

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

¹Pure text mining capabilities in the R programming language are provided by the packages **cleanNLP** (Arnold 2017), **quanteda** (Benoit *et al.* 2017), **tidytext** (Silge and Robinson 2016), **qdap** (Rinker 2013), and **tm** (Feinerer *et al.* 2008).

²Valence shifters can also appear in positions other than right before a certain word. The **sentimentr** package (Rinker 2017) does a good job accounting for this. We decided not to implement this layer of complexity, as it would take too long to compute sentiment for tens of thousands of documents at once. It is kept for a future version of the package.

$\sum_{i=1}^{Q_d} \omega_i s_{i,n,t}^{\{l\}}$, with Q_d the total number of unique unigrams in the text. The score $s_{i,n,t}^{\{l\}}$ is the sentiment value attached to unigram i from document $d_{n,t}$, based on lexicon l . The weights ω_i define the importance of each unigram, for example, its frequency. These scores are subsequently multiplied by the feature weights to obtain lexicon- and feature-specific sentiment scores, as $s_{n,t}^{\{l,k\}} \equiv s_{n,t}^{\{l\}} \times w_{n,t}^k$, with k the index denoting the feature. If the document does not correspond to the feature, the value of $s_{n,t}^{\{l,k\}}$ is zero.

The sentiment calculation is performed with `compute_sentiment()`. This function outputs a `data.table` with all values for $s_{n,t}^{\{l,k\}}$, which serves as the basis for aggregation into time series in the next step.

Step 3: Aggregate into textual sentiment time series

In this step, the purpose is to aggregate the individual sentiment scores and obtain various representative time series. Two main aggregations are performed. The first, across-document, collapses all sentiment scores across documents within the same frequency (*e.g.*, day or month, as defined by t) into one score. The weighted sum that does so is $s_t^{\{l,k\}} \equiv \sum_{n=1}^{N_t} \theta_n s_{n,t}^{\{l,k\}}$, where the weights θ_n define the importance of each document n at time t (for instance, based on the length of the text). The second, across-time, smooths the newly aggregated sentiment scores over time, 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 $b = 1, \dots, B$ go with different values for b_t to smooth the time series in various ways, with a time lag of τ . The first $\tau - 1$ observations are dropped from the original time indices, such that $u = \tau, \dots, T$ becomes the time index for the ultimate time series. This leaves $N = T - \tau + 1$ time series observations.

The number of obtained time series is equal to $P \equiv L \times K \times B$. Every time series covers one aspect of each dimension, best described as ‘the textual sentiment as computed by lexicon l for feature k aggregated across time using scheme b ’. A single time series incorporating such combination is designated by $s_u^p \equiv s_u^{\{l,k,b\}}$ across all values of u , for $p = 1, \dots, P$. The ensemble of time series captures both different information and the same information in different ways.

The entire aggregation setup is specified by means of the `ctr_agg()` function, including the within-document aggregation needed for the sentiment analysis. The `sentomeasures()` function performs both the sentiment calculation (via `compute_sentiment()`) and time series aggregation (via `perform_agg()`), outputting a *sentomeasures* object.

Step 4: Calibrate sparse regression model and perform (out-of-sample) predictions

The sentiment measures are now regular time series variables that can be applied in regressions. In case of a linear regression, the reference equation is:

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

The target variable y_{u+h} is often a variable to forecast, that is, $h > 0$. Other (non-sentiment) variables are captured by \mathbf{x}_u and γ is the associated parameter vector. All other variables are the textual sentiment variables as constructed before. Let $\beta = (\beta_1, \dots, \beta_P)^\top$ and $\mathbf{s}_u = (s_u^1, \dots, s_u^P)^\top$. Logistic regression (binomial and multinomial) is available in the package based on the same underlying linear structure.

The typical large dimensionality of P relative to the number of observations and the potential multicollinearity pose a problem to regular least squares regression. Instead, estimation and

variable selection through a penalized regression relying on the elastic net regularization of [Zou and Hastie \(2005\)](#) is more appropriate. Regularization, in short, shrinks the coefficients of the least informative variables towards zero. It consists of optimizing the least squares or likelihood function including a penalty component. The elastic net optimization problem for a linear regression is expressed as:

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

The tilde denotes standardized variables, and $\|\cdot\|_p$ is the ℓ_p -norm. The standardization is required for the regularization, but the coefficients are rescaled back once obtained. The rescaled estimates of the model coefficients for the textual sentiment indices are in $\hat{\beta}$, usually a sparse vector, depending on the severity of the shrinkage. The parameter $0 \leq \alpha \leq 1$ defines the trade-off between the Ridge ([Hoerl and Kennard 1970](#)), ℓ_2 , and the LASSO ([Tibshirani 1996](#)), ℓ_1 , regularization, respectively when it is equal to 0 and 1. 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 two parameters are determined in a data-driven way; they are calibrated such that they are optimal to the regression equation at hand. The **sentometrics** package allows four ways to do the calibration, either through cross-validation, or based on three information criteria adapted to elastic net ([Tibshirani and Taylor 2012](#)).

An analysis of typical interest is the sequential estimation of a regression model and out-of-sample prediction. For a given sample size $m < N$, a regression model is estimated for the first m observations and used to predict the target variable one-step ahead, at $m + 1$. This procedure is repeated, leading to a series of estimates. These are compared with the realized values to assess the average out-of-sample prediction performance.

The type of model, the calibration approach and other modelling decisions are defined via the `ctr_model()` function. The (iterative) model estimation and calibration is performed by calling `sento_model()`. The output is a *sentomodel* (one model) or a *sentomodeliter* (a collection of iteratively estimated *sentomodel* objects and out-of-sample predictions) object.

Step 5: Evaluate prediction performance and retrieve sentiment attributions

The *sentomodeliter* object carries an overview of performance measures relevant to the type of model estimated. Additionally, it can easily be plotted to visually inspect the prediction precision. A good way to compare different models, sentiment-based or not, is to construct a model confidence set ([Hansen et al. 2011](#)). This set isolates the models that are statistically the best in terms of predictive ability, within a confidence level. The wrapper `perform_MCS()` makes available the function from the **MCS** package ([Catania and Bernardi 2017](#)) that carries out this comparative analysis.

The aggregation into textual sentiment time series is entirely linear. This allows, based on the estimated coefficients $\hat{\beta}$, to compute every corresponding underlying dimension's sentiment *attribution* to a given prediction. For example, the attribution of a certain feature k in the forecast of the target variable, at a particular date, is the weighted sum of the model coefficients and the values of the sentiment measures in which k plays a role. In a similar vein, the attribution can be computed for all features, lexicons, time weighting schemes and individual documents. Through attribution, a prediction is broken down in its respective

components. The functions `retrieve_attributions()` and `plot_attributions()` provide insights into the evolution of the prediction attribution along the different dimensions.

3. The R package *sentometrics*

The *sentometrics* package takes care of *Steps 2–5* in full. The texts collection in *Step 1* is, obviously, left to the user. The other parts (cleaning, features generation and selection) are also mostly left to the user, but support for these tasks is present within the package. In what follows, an example shows how to put the steps into practice using the *sentometrics* package. We first explain the two key control functions, `ctr_agg()` and `ctr_model()`. Then we illustrate the workflow using built-in data. Finally, we briefly touch upon other handy functions that are present in the package.

3.1. Deep-dive into the two main control functions

Aggregation control

The required arguments of the `ctr_agg()` function are: `howWithin`, `howDocs`, `howTime`, `do.ignoreZeros`, `by`, `lag` and `fill`. The optional arguments are `alphasExp`, `ordersAlm`, `do.inverseAlm`, `do.normalizeAlm`, `weights` and `dfm`. We focus the explanation on the first three mandatory arguments, central to the aggregation:

- **howWithin:** This argument defines how sentiment is aggregated within the same document, that is, it sets the weights ω_i . Do you simply want to take the difference between the number of positive and negative words? Do you prefer to do the same but normalize the sentiment score based on the total number of words within the document? Or do you think that words appearing very frequently across all documents should be down-weighted? You can respectively select the "counts", "proportional" or "tf-idf" option. The last is an abbreviation for term frequency-inverse document frequency, a well-known concept within computational linguistics (Spärck 1972).
- **howDocs:** This argument defines how sentiment is aggregated across all documents at the same date (or frequency), that is, it sets the weights θ_n . There are two options: either "equal_weight", which gives the same weight to every document, or "proportional", which gives higher weights to documents with more words, relative to the document population at a given date. The `do.ignoreZeros` argument can be used to ignore documents with zero sentiment in the computation of the across-document weights. By default these documents are overlooked. This avoids the incorporation of documents not relevant to a particular feature (as in those cases $s_{n,t}^{\{l,k\}}$ is exactly zero, because $w_{n,t}^k = 0$), which could lead to a bias of sentiment towards zero.³
- **howTime:** This argument defines how sentiment is aggregated across dates, to smoothen the time series and acknowledge that sentiment at a given point is at least partly based

³It also ignores the documents which are relevant to a feature, but exhibit zero sentiment. This can occur if there none of the words have a polarity, or the weighted number of positive and negative words offset each other.

on sentiment and information from the past. The `lag` argument has the role of τ ; it dictates how far to go back. The `"equal_weight"` option is similar to a simple weighted moving average, `"linear"` and `"exponential"` are two options which give weights to the observations according to a linear or an exponential curve, and `"almon"` does the same but based on Almon polynomials. The last two curves have respective arguments to define their shape(s), being `alphasExp`, and `ordersAlm`, `do.inverseAlm` and `do.normalizeAlm`. If desired, user-constructed weights can be supplied via `weights` as a named `data.frame`. All the weighting schemes define the different b_t values. The `fill` argument is of sizeable importance here. It is used to add in dates for which not a single document was available. These added, originally missing, dates are given a value of 0 (`"zero"`) or the most recent value (`"latest"`). The option `"none"` accords to not filling up the date sequence at all. Adding in dates (or not) impacts the time aggregation by respectively combining the latest *consecutive* dates, or the latest *available* dates.

Via the `by` argument, the time interval at which the time series have to be aggregated is chosen: daily (`"day"`), weekly (`"week"`), monthly (`"month"`) or yearly (`"year"`). The `dfm` argument, short for document-feature matrix, is there to provide additional flexibility in terms of which words are part of the universe the lexicons will look through to find polarized words. It can be safely ignored; more information is found in the documentation manual, see `help("ctr_agg")`.

Modelling control

The `ctr_model()` function requires at least the following arguments: `model`, `type`, `intercept`, `h`, `do.iter`, `alphas` and `do.progress`. The `model` argument can be `"gaussian"` (linear), `"binomial"` or `"multinomial"`. The `type` specifies the calibration procedure to find the most appropriate α and λ . It is either `"cv"` (cross-validation) or one of three information criteria (`"BIC"`, `"AIC"` or `"Cp"`). The argument `alphas` can be altered to change the possible values for `alpha`. The possible values for λ are generated internally (by the `glmnet::glmnet()` function call). To enact an iterative model estimation and a one-step ahead out-of-sample analysis, set `do.iter=TRUE`. The argument `h`, positive or negative, shifts the response variable according to y_{u+h} .⁴ The arguments `intercept` and `do.progress` respectively add an intercept to the model and print progress statements, or not.

The arguments `trainWindow`, `testWindow` and `oos` are needed when model calibration is performed through cross-validation, that is, when `type="cv"`. The cross-validation implemented is based on the “rolling forecasting origin” principle. As an example, assume there are 120 observations in total, `trainWindow=80`, `testWindow=10` and `oos=5`. In the first round of cross-validation, a model is estimated for a certain α and λ combination with the first 80 observations, then 5 observations are skipped, and predictions are generated for observations 86 to 95. The next round does the same but with all observations moved one step forward. This is done until the end of the total sample is reached and repeated for all possible parameter combinations, relying on the `caret::train()` function. The optimal α and λ couple is the one that induced the lowest average prediction error.

⁴Be careful, if the input response variable is not aligned date-wise with the sentiment measures and the other explanatory variables, `h` cannot be interpreted as the exact prediction horizon. In other words, `h` only shifts the input variables as they are provided, no more.

The arguments `nSample`, `start` and `do.parallel` are used for iterative modelling, when `do.iter=TRUE`. The first argument is the size of the sample to re-estimate the model each time, that is, m . The second argument can be used to only run a later subset of the iterations; by default it is set to 1 which runs all iterations. The total number of iterations is equal to `length(y) - nSample - abs(h) - oos`, with `y` the response variable as a vector. If `do.parallel=TRUE` and a parallel backend is registered, the `%dopar%` construct from the `foreach` (Weston 2015b) package is utilized to speed up the estimation procedure.

3.2. Illustration of the main workflow

The very first step is to load the *sentometrics* package, assuming it has been properly installed.

```
R> library("sentometrics")
```

As highlighted, the input texts have to be structured rectangularly, with every row a document. The `data.frame` structure fits this need. We demonstrate the workflow using the `esu` built-in dataset, a collection of news articles from The Wall Street Journal and The Washington Post between 1995 and 2014.⁵ The data is loaded below:

```
R> data("usnews")
R> class(usnews)
```

```
[1] "data.frame"
```

The `"id"`, `"date"` and `"text"` columns are mandatory. All other columns are reserved for features, of type `numeric`. For this particular corpus, there are four original features. The first two indicate the news source, the latter two the relevance of every document to the U.S. economy. The feature values $w_{n,t}^k$ are binary.

```
R> colnames(usnews)
```

```
[1] "id" "date" "text" "wsj" "wapo" "economy" "noneconomy"
```

To access the texts, simply do `usnews[["text"]]`. To put the texts and features into a corpus structure, call the `sentocorpus()` function. If you have no features available, the corpus can still be created without any feature columns in the input `data.frame`. In that situation, a dummy feature called `"dummy"` with a score of 1 for all texts is added to the *sentocorpus* output object.

```
R> corpusAll <- sentocorpus(usnews)
R> class(corpusAll)
```

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

⁵The data originates from <https://www.crowdflower.com/data-for-everyone> (under “Economic News Article Tone and Relevance”).

The `sentocorpus()` function creates a *sentocorpus* object on top of the **quanteda**'s package *corpus* object. This means that many functions from **quanteda** to manipulate corpora can be applied to a *sentocorpus* object as well. This is helpful as can be seen below, where we limit the corpus to all articles between January 1995 and September 2014.

```
R> corpus <- quanteda::corpus_subset(corpusAll, date < "2014-10-1")
R> corpus
```

Corpus consisting of 4,097 documents and 5 docvars.

The docvars are the "date" column and all feature columns. To round off *Step 1*, we add two features related to uncertainty using `add_features()`. The features `el` and `war` give a score of 1 to documents in which respectively the words "election" and "war" appear, which turns out to be the case for 139 and 112 documents. Overall, this gives $K = 6$ features. The `add_features()` function is even more useful when the sole feature present is the automatically created "dummy" feature.⁶

```
R> corpus <- add_features(corpus,
+ keywords=list(el="election", war="war"))
R> c(sum(corpus$documents$el), sum(corpus$documents$war))

[1] 256 1486
```

In *Step 2*, we begin by selecting the lexicons to include. Herunder, the built-in lexicons and valence word lists are first loaded. The well-known Loughran and McDonald ([Loughran and McDonald 2011](#)) and Henry ([Henry 2008](#)) word lists together with an English valence word list are supplied to `setup_lexicons()`.⁷ We also add a self-made lexicon of the word 'uncertainty' and synonyms, and attach a higher than usual negative sentiment score to these words. We proceed with $L = 3$.

```
R> data("lexicons")
R> data("valence")
R> mine <- data.frame(w=c("uncertainty", "anxiety", "concern",
+ "distrust", "worries"), s=c(-2, -2, -2, -2, -2))
R> lexiconsIn <- c(list(myLexicon=mine), lexicons[c("LM_eng", "HENRY_eng")])
R> lex <- setup_lexicons(lexiconsIn=lexiconsIn,
+ valenceIn=valence[["valence_eng"]])
```

The lexicons look like below. This format of the word lists are `data.tables`, similar to what is used in the **sentimentr** package. The second half of the lexicon represents the original words copied but preceded by a 'NOT_'. When computing sentiment, all negators in the valence word list (*i.e.*, in `lex[["valence"]]`) are searched for in the corpus texts and also replaced by 'NOT_', glued together to the next word. This way, the lexicon can be applied to the modified corpus still considering all words as unigrams. The underlying assumption is that all negators have the same impact, multiplying the lexicon score by -1 . The same logic holds for other valence shifters, such as amplifiers (*e.g.*, very).

⁶To delete a feature, one can do `quanteda::docvars(corpus, field="featureName") <- NULL`.

⁷As for now, the built-in valence word lists only include negators.

```
R> lex[["myLexicon"]]
```

```

          x  y
1:      anxiety -2
2:      concern -2
3:    distrust -2
4:  uncertainty -2
5:      worries -2
6:   NOT_anxiety  2
7:   NOT_concern  2
8:   NOT_distrust  2
9: NOT_uncertainty  2
10:  NOT_worries  2

```

The sentiment calculation in *Step 2* and the aggregation in *Step 3* are performed in conjunction with each other. We aggregate sentiment at a monthly frequency, weight the words for within-document aggregation based on the term frequency-inverse document frequency statistic, weight the documents for across-document aggregation proportionally w.r.t. to the number of words in the document, and have an equally-weighted, a linear and several Almon polynomial time aggregation schemes for a lag of 12 months. The time weighting schemes accord to $B = 8$. We ignore documents with zero sentiment for across-document aggregation, and fill missing dates with a value of zero for across-time aggregation, as per default.

```
R> ctrAgg <- ctr_agg(howWithin="tf-idf",
+   howDocs="proportional",
+   howTime=c("equal_weight", "linear", "almon"),
+   do.ignoreZeros=TRUE,
+   by="month",
+   fill="zero",
+   lag=12,
+   ordersAlm=1:3,
+   do.inverseAlm=TRUE,
+   do.normalizeAlm=TRUE)
```

All is set to construct the sentiment measures, done by calling the `sento_measures()` function. The `summary()` generic displays a brief overview of the composition of the sentiment time series. The output *sentomeasures* object is a classed `list` with as most important elements "measures" (the textual sentiment time series), "sentiment" (the original sentiment scores per document) and "stats" (a selection of summary statistics). Alternatively, the same output can be obtained with the functions `compute_sentiment()` and `perform_agg()`.

```
R> sentMeas <- sento_measures(corpus, lexicons=lex, ctr=ctrAgg)
R> summary(sentMeas)
```

This *sentomeasures* object contains 144 textual sentiment time series with 226 observations each, at a monthly frequency.

The corpus has following features: wsj wapo economy noneconomy el war

Following lexicons were used to calculate sentiment: myLexicon LM_eng HENRY_eng

Following scheme was applied for aggregation within documents: tf-idf

Following scheme was applied for aggregation across documents: proportional

Following schemes were applied for aggregation across time: equal_weight

linear almon1 almon1_inv almon2 almon2_inv almon3 almon3_inv

Aggregate statistics:

mean	sd	max	min	meanCorr
-0.00793	0.00522	0.00327	-0.02029	0.19508

A *sentomeasures* object can be easily plotted across each of its dimensions. For example, Figure 1 shows a time series of average sentiment for every feature involved.

```
R> p <- plot(sentMeas, group="features") +
+ guides(colour=guide_legend(nrow=1))
+ p
```

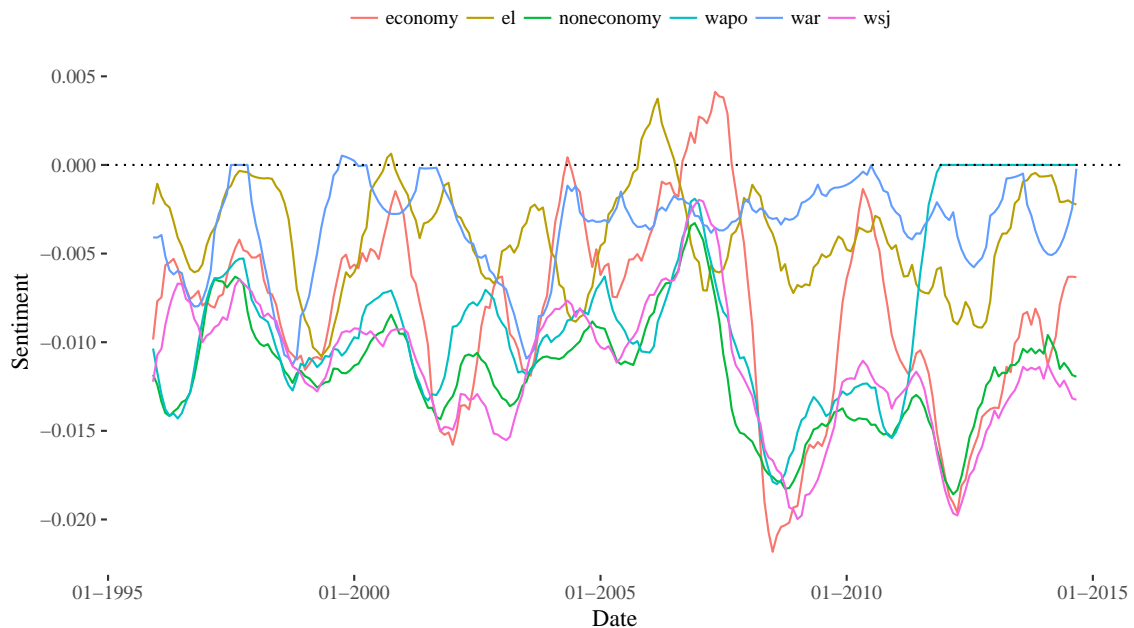


Figure 1: Average textual sentiment time series across features

Step 4 consists of the modelling. The package's built-in target variable is an index of economic policy uncertainty, the 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. Between 1995 and 2014, the index fluctuated between 85 and 350. We will analyse how well the sentiment in the texts of our corpus relates to this index. The length of the dependent variable should be equal to the number of observations in the sentiment

measures. We omit other explanatory variables but they can easily be added as a named `data.frame` via the `x` argument of `sent_model()`.

```
R> data("epu")
R> y <- epu[epu[["date"]] >= sentMeas[["measures"]$date[1], ]$index
R> length(y) == nrow(sentMeas[["measures"]])

[1] TRUE
```

Let us now apply the iterative analysis to illustrate the simplicity in use compared with the rich output. The modelling control is set up below. We made sure the months of `epu` coincide with the months in `sentMeas`, such that `h = 1` means the model aims at forecasting next month's EPU index value. The model is linear and the calibration is based on a Bayesian-like information criterion. We set a sample size of 36 months for a total sample of 226 observations. The potential `alpha` values are close to 0.5, corresponding to a good mix between the Ridge and the LASSO regularization.

```
R> ctrIter <- ctr_model(model="gaussian",
+   type="BIC",
+   h=1,
+   alphas=c(0.3, 0.5, 0.7),
+   do.iter=TRUE,
+   nSample=36,
+   do.parallel=TRUE)
```

The `doParallel` package ([Weston 2015a](#)) is used to register the backend for parallel computation, given `do.parallel=TRUE`. The output of the `sent_model()` call is a *sentomodeliter* object, since `do.iter=TRUE`.

```
R> require("doParallel")
R> cl <- makeCluster(4)
R> registerDoParallel(cl)
R> out <- sent_model(sentMeas, y, ctr=ctrIter)
R> stopCluster(cl)
R> summary(out)
```

Model specifications

```
Model type: gaussian
Calibration: via BIC information criterion
Sample size: 36
Total number of iterations/predictions: 189
Optimal average elastic net alpha parameter: 0.48
Optimal average elastic net lambda parameter: 12.18
```

Out-of-sample performance

Mean directional accuracy: 39.36 %

Root mean squared prediction error: 41.46

Mean absolute deviation: 29.13

In-sample performance

Average fraction of deviance explained: 54.76 %

A more detailed view of the different performance measures, in this case directional accuracy, root mean squared and absolute errors, is obtained via `out[["performance"]]`. A list of the individual *sentomodel* objects is found under `out[["models"]]`. A simple plot to visualize the out-of-sample fit of any *sentomodeliter* object can be produced using `plot()`. We see in Figure 2 that the combination of different textual sentiment times series derived from two U.S. journals alone do a decent job in forecasting the EPU index.

`R> plot(out)`

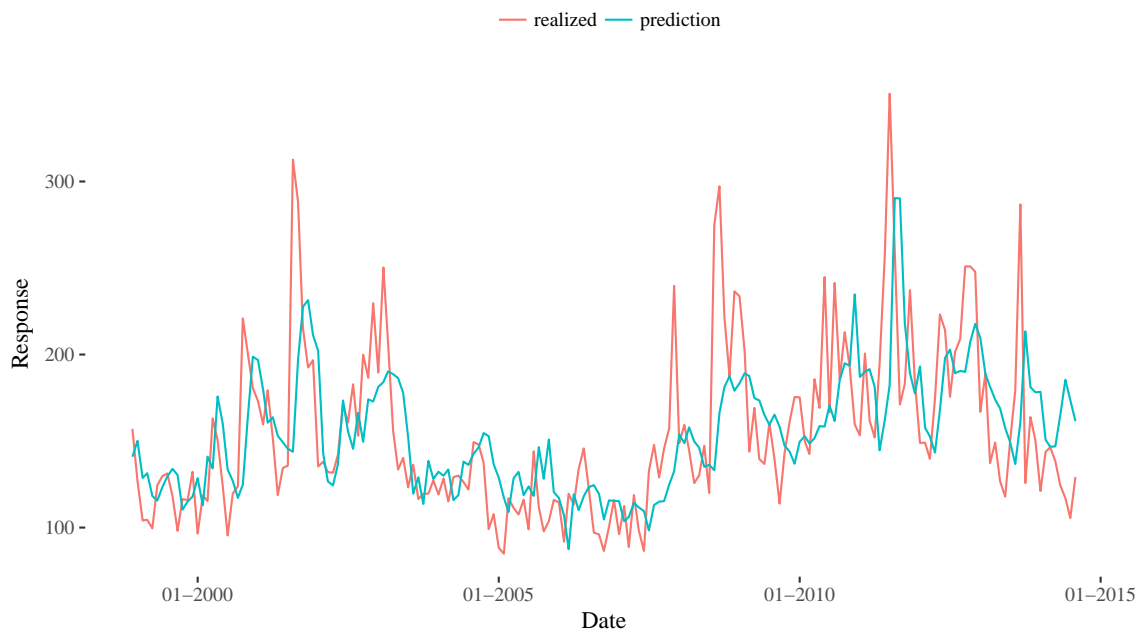


Figure 2: Realized EPU index values versus out-of-sample predictions

The final step, *Step 5*, mainly focuses on the attribution analysis. It can be invoked by the `retrieve_attributions()` function with the modelling object and the related sentiment measures object as inputs. It generates attributions for all out-of-sample dates at the level

of individual documents, lexicons, features and time. The function can be applied for *sentometrics* objects as well, for any specific dates using the `refDates` argument. The `do.normalize` argument, if set to `TRUE`, normalizes all attributions between -1 and 1 .

```
R> attributions <- retrieve_attributions(out, sentMeas, do.normalize=FALSE)
R> names(attributions)
```

```
[1] "documents" "lexicons" "features" "time"
```

Plotting out the attributions is effortless with the `plot_attributions()` function, according to any of the dimensions, except for individual documents. The attributions are displayed stacked on top of each other, per date. The y-axis represents the attribution to the prediction of the level of the EPU index. Inspecting Figure 3, one sees for example a clearly evolving influence of the Wall Street Journal feature on the predictions.

```
R> a <- plot_attributions(attributions, group="features") +
+   guides(fill=guide_legend(nrow=1))
R> a
```

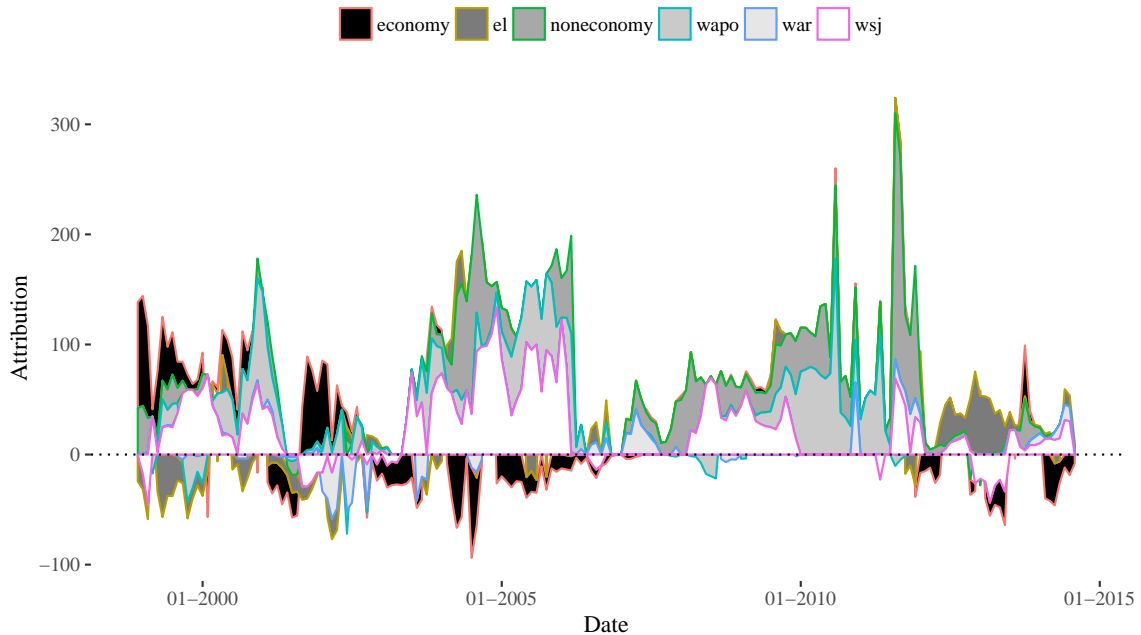


Figure 3: Prediction attribution to each of the six features

3.3. Merging functionalities

Apart from several convenience functions such as `predict()`, `scale()`, `fill_measures()` and `select_measures()`, the biggest non-discussed functionality is the further merging of the sentiment measures. For instance, all sentiment measures composed of three particular

time aggregation schemes can be merged together as an average. The merging functionality is foremost a way to either diminish or expand the dimensionality of the sentiment measures. It is carried out in two steps: first by specifying another, yet simple, control function, and then by the actual `merge_measures()` call which returns a modified *sentomeasures* object. Here, both built-in lexicons, both news sources and two of the time weighting schemes are respectively merged, or else said collapsed, into one. The `do.keep=FALSE` option means the original measures are not kept after merging, such that the number of sentiment time series effectively goes down.

```
R> ctrMerge <- ctr_merge(sentMeas,
+   time=list(W=c("equal_weight", "linear")),
+   lexicons=list(LEX=c("LM_eng", "HENRY_eng")),
+   features=list(JOUR=c("wsj", "wapo")),
+   do.keep=FALSE)
R> sentMeasMerged <- merge_measures(ctrMerge)
R> sentMeasMerged[c("features", "lexicons", "time")]

$features
[1] "economy" "noneconomy" "el" "war" "JOUR"

$lexicons
[1] "myLexicon" "LEX"

$time
[1] "almon1" "almon1_inv" "almon2" "almon2_inv" "almon3" "almon3_inv" "W"
```

The sentiment measures can also be merged into one measure we refer to as global sentiment. Each of the different components of the dimensions has to receive a weight that stipulates the importance in the global sentiment index. In the example below, the highest weight is put on the initial features as well as on the McDonald and Loughran lexicon. The output is a one-column `data.frame`, and can thus be easily used as a variable in any other regression analysis. The global sentiment measure is plotted in Figure 4. The drop in sentiment during the worldwide financial crisis is most apparent.

```
R> glob <- to_global(sentMeas,
+   lexicons=c(0.10, 0.70, 0.20),
+   features=c(0.20, 0.20, 0.20, 0.20, 0.10, 0.10),
+   time=1)
R> g <- ggplot(data=glob, aes(x=as.Date(row.names(glob)), y=global)) +
+   geom_line() +
+   scale_x_date(name="Date") +
+   scale_y_continuous(name="Sentiment") +
+   ggthemes::theme_tufte(base_size=12) +
+   theme(legend.title=element_blank())
R> g
```

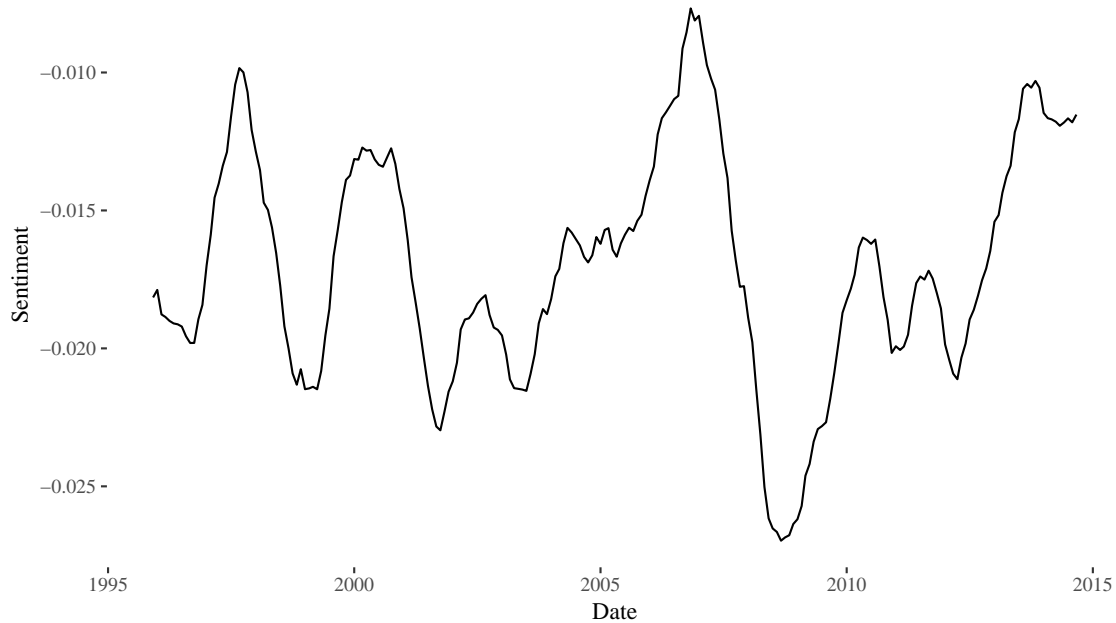


Figure 4: Globally merged sentiment measure

4. Conclusion and future directions

The R package **sentometrics** provides a means to calculate sentiment for a large number of texts, aggregate the textual sentiment into a vast quantity of time series at a desired frequency, and use this information in a flexible prediction model. It can thus be deployed to relate a corpus of texts, qualitative data, to a quantitative target variable and retrieve information on which type of sentiment is most informative. This package is a first step towards a consistent econometric analysis in R of the added value of information in texts. In the future, the links with current text mining packages will be enforced to improve the transition from pure textual analysis, corpus construction and features generation to the econometric analysis. We will also expand and improve the textual sentiment calculation engine(s). On the last note, we aim to expand the number of available models, as well as estimation and calibration approaches. If you use R or **sentometrics**, please cite the software in publications. In case of the latter, use `citation("sentometrics")`.

Computational details

The example was produced using R 3.4.3 (R Core Team 2017) and **sentometrics** version 0.2. Computations were performed on a Windows 10 Pro machine, x86 64-w64-mingw32/x64 (64-bit) with Intel(R) Core(TM) i7-3632QM CPU 2x 2.20 GHz. The parallelized `sento_model()` call takes roughly 30 seconds. Code for the illustration is available in the R script `run_vignette.R` located in the **examples** folder on the dedicated GitHub repository.

R itself and all packages used are available from CRAN at <http://CRAN.R-project.org/>. The version under development is available on the **sentometrics** website at <http://sborms.github.io/sentometrics/>, which provides additional examples of how to use the package.

Acknowledgments

We acknowledge Google Summer of Code 2017 for their financial support. Thanks to Andres Algaba and Nabil Bouamara for helpful comments.

References

- Ardia D, Bluteau K, Boudt K (2017). “Questioning the News about Economic Growth: Sparse Forecasting using Thousands of News-Based Sentiment Values.” *Working paper*. URL <https://ssrn.com/abstract=2976084>.
- Arnold T (2017). “**cleanNLP**: A Tidy Data Model for Natural Language Processing using **cleanNLP**.” R package version 1.10.0, URL <https://CRAN.R-project.org/package=cleanNLP>.
- Baker S, Bloom N, Davis S (2016). “Measuring Economic Policy Uncertainty.” *NBER Working paper*. URL http://www.policyuncertainty.com/media/EPU_BBD_Mar2016.pdf.
- Beckers B, Kholodilin K, Ulbricht D (2017). “Reading between the Lines: Using Media to Improve German Inflation Forecasts.” *DIW Berlin Discussion Paper No. 1665*. doi:10.2139/ssrn.2970466.
- Benoit K, Watanabe K, Nulty P, Obeng A, Wang H, Lauderdale B, Lowe W (2017). *quanteda: Quantitative Analysis of Textual Data*. R package version 0.99.12, URL <http://quanteda.io>.
- Caporin M, Poli F (2017). “Building News Measures from Textual Data and an Application to Volatility Forecasting.” *Econometrics*, **5**(3), 35. doi:10.3390/econometrics5030035.
- Catania L, Bernardi M (2017). *MCS: Model Confidence Set Procedure*. R package version 0.1.3, URL <https://CRAN.R-project.org/package=MCS>.
- Ceron E, Curini L, Iacus S, Porro G (2014). “Every Tweet Counts? How Sentiment Analysis of Social Media Can Improve Our Knowledge of Citizens’ Political Preferences with an Application to Italy and France.” *New Media & Society*, **16**(2), 340–358. doi:10.1177/1461444813480466.
- Dowle M, Srinivasan A (2017). *data.table: Extension of ‘data.frame’*. R package version 1.10.4-2, URL <https://CRAN.R-project.org/package=data.table>.
- Feinerer I, Hornik K, Meyer D (2008). “Text Mining Infrastructure in R.” *Journal of Statistical Software*, **22**(5), 1–54. doi:10.18637/jss.v025.i05.

- Friedman J, Hastie T, Tibshirani R (2010). “Regularization Paths for Generalized Linear Models via Coordinate Descent.” *Journal of Statistical Software*, **33**(1), 1–22. doi:10.18637/jss.v033.i01.
- Hansen P, Lunde A, Nason J (2011). “The Model Confidence Set.” *Econometrica*, **79**, 453–497. doi:10.3982/ECTA5771.
- Henry E (2008). “Are Investors Influenced by How Earnings Press Releases are Written?” *Journal of Business Communication*, **45**(4), 363–407. doi:10.1177/0021943608319388.
- Hoerl A, Kennard R (1970). “Ridge Regression: Biased Estimation for Nonorthogonal Problems.” *Technometrics*, **12**, 55–67. doi:10.1080/00401706.1970.10488634.
- Kuhn M (2017). *caret: Classification and Regression Training*. R package version 6.0-77, URL <https://CRAN.R-project.org/package=caret>.
- Loughran T, McDonald B (2011). “When is a Liability not a Liability? Textual Analysis, Dictionaries, and 10-Ks.” *Journal of Finance*, **66**(1), 35–65. doi:10.1111/j.1540-6261.2010.01625.x.
- R Core Team (2017). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. R version 3.4.3, URL <https://www.R-project.org/>.
- Rinker TW (2013). *qdap: Quantitative Discourse Analysis Package*. R package version 2.2.5, URL <http://github.com/trinker/qdap>.
- Rinker TW (2017). *sentimentr: Calculate Text Polarity Sentiment*. R package version 1.0.0, URL <http://github.com/trinker/sentimentr>.
- Silge J, Robinson D (2016). “tidytext: Text Mining and Analysis Using Tidy Data Principles in R.” **1**(3). doi:10.21105/joss.00037.
- Spärck J (1972). “A Statistical Interpretation of Term Specificity and Its Application in Retrieval.” *Journal of Documentation*, **28**, 11–21. doi:doi:10.1108/eb026526.
- Tibshirani R (1996). “Regression Shrinkage and Selection via the LASSO.” *Journal of the Royal Statistical Society: Series B*, **58**, 267–288. doi:10.1111/j.1467-9868.2011.00771.x.
- Tibshirani R, Taylor J (2012). “Degrees of Freedom in LASSO Problems.” *Annals of Statistics*, **4**, 1198–1232. doi:10.1214/12-AOS1003.
- Weston S (2015a). *doParallel: Foreach Parallel Adaptor for the ‘parallel’ Package*. R package version 1.0.10, URL <https://CRAN.R-project.org/package=doParallel>.
- Weston S (2015b). *foreach: Provides Foreach Looping Construct for R*. R package version 1.4.3, URL <https://CRAN.R-project.org/package=foreach>.
- Zou H, Hastie T (2005). “Regularization and Variable Selection via the Elastic Net.” *Journal of the Royal Statistical Society: Series B*, **67**, 301–320. doi:10.1111/j.1467-9868.2005.00503.x.

Affiliation:

David Ardia
Institute of Financial Analysis
University of Neuchâtel, Switzerland
& Department of Finance, Insurance and Real Estate
Laval University, Canada
E-mail: david.ardia@unine.ch

Keven Bluteau
Institute of Financial Analysis
University of Neuchâtel, Switzerland
& Vrije Universiteit Brussel, Belgium
E-mail: keven.bluteau@unine.ch

Samuel Borms
Institute of Financial Analysis
University of Neuchâtel, Switzerland
& Vrije Universiteit Brussel, Belgium
E-mail: samuel.borms@unine.ch

Kris Boudt
Solvay Business School
Vrije Universiteit Brussel, Belgium
& Vrije Universiteit Amsterdam, The Netherlands
E-mail: kris.boudt@vub.be