

WEEK 4: TEXT MINING 2

SECU0057

BENNETT KLEINBERG

6 FEB 2020

Applied Data Science

MIND

How a Computer Program Helped Show J.K. Rowling write A Cuckoo's Calling

Author of the *Harry Potter* books has a distinct linguistic signature

By Patrick Juola on August 20, 2013

[URL to the article](#)

WEEK 4: TEXT MINING 2

TODAY

- ngrams
- sentiment analysis
- trajectory analysis

WHAT IS TFIDF?

PROBLEM?

```
## [1] "It is a great time to be alive"
```

SO FAR...

- we used tokens as unit of analysis
- but: sometimes multiple tokens might reveal more

N-GRAMS

= sequences of n tokens

- unigrams: $n = 1$
- bigrams: $n = 2$
- trigrams: $n = 3$

N-GRAMS WITH QUANTEDA

```
dfm(x = text
    , ngrams = 1
    , verbose = F
    , remove_punct = T
    , stem = F
    )
```

UNIGRAMS

```
## Document-feature matrix of: 1 document, 8 features (0% sparse).  
## 1 x 8 sparse Matrix of class "dfm"  
##           features  
## docs      it is a great time to be alive  
##  text1    1  1  1      1      1  1  1      1
```

BIGRAMS

```
ngrams = 2
```

```
## Document-feature matrix of: 1 document, 7 features (0% sparse).  
## 1 x 7 sparse Matrix of class "dfm"  
##           features  
## docs      it_is is_a a_great great_time time_to to_be be_alive  
## text1      1    1      1          1        1      1      1
```

TRIGRAMS

```
ngrams = 3
```

```
## Document-feature matrix of: 1 document, 6 features (0% sparse).  
## 1 x 6 sparse Matrix of class "dfm"  
##           features  
## docs    it_is_a is_a_great a_great_time great_time_to time_to_be  
## text1      1           1           1           1           1  
##           features  
## docs    to_be_alive  
## text1      1
```

WHAT HAPPENS WHEN WE INCREASE N ?

N-GRAMS ON A CORPUS

```
unigrams = dfm(x = data_corpus_inaugural[1:3]  
               , ngrams = 1  
               )
```

```
## Warning: 'as.data.frame.dfm' is deprecated.  
## Use 'convert(x, to "data.frame")' instead.  
## See help("Deprecated")
```

document	fellow-citizens	of	the	senate	and
1789-Washington	1	71	116	1	48
1793-Washington	0	11	13	0	2
1797-Adams	3	140	163	1	130

WITH PREPROCESSING

```
unigrams = dfm(x = data_corpus_inaugural[1:3]  
               , ngrams = 1  
               , stem = T  
               , remove = stopwords()  
               , remove_punct = T  
               )
```

```
## Warning: 'as.data.frame.dfm' is deprecated.  
## Use 'convert(x, to "data.frame")' instead.  
## See help("Deprecated")
```

document	fellow-citizen	senat	hous	repres	among
1789-Washington	1	1	2	2	1
1793-Washington	0	0	0	0	0
1797-Adams	3	1	3	3	4

BIGRAMS

```
bigrams = dfm(x = data_corpus_inaugural[1:3]  
              , ngrams = 2  
              , stem = T  
              , remove = stopwords()  
              , remove_punct = T  
              )
```

```
## Warning: 'as.data.frame.dfm' is deprecated.  
## Use 'convert(x, to "data.frame")' instead.  
## See help("Deprecated")
```

document	fellow- citizen_of	of_the	the_senat	senat_and	and_of
1789- Washington	1	20	1	1	2
1793- Washington	0	4	0	0	1
1797-Adams	0	29	0	0	2

What happens when we increase n ?

PROBLEM?

- essentially we're back at where we started
- same old problem with ~~word~~ ngram importance

```
dfm_tfidf(bigrams
          , scheme_tf = 'prop'
          , scheme_df = 'inverse'
          , k = 1)
```



```
## Warning: 'as.data.frame.dfm' is deprecated.  
## Use 'convert(x, to "data.frame")' instead.  
## See help("Deprecated")
```

document	fellow- citizen_of	of_the	the_senat	senat_and	and_of
1789- Washington	0.000123	-0.001749	0.000123	0.000123	-0.000175
1793- Washington	0.000000	-0.003730	0.000000	0.000000	-0.000932
1797-Adams	0.000000	-0.001564	0.000000	0.000000	-0.000108

MULTIPLE N N-GRAMS

```
dfm(x = text
    , ngrams = 1:3
    , verbose = F
    , remove_punct = T
    , stem = F
    )
```

***N*-GRAMS**

- generalisation of “single” tokens
- often used in **bag-of-word models**
- common in predictive modelling (Week 7: machine learning 1)

SENTIMENT ANALYSIS

SENTIMENT ANALYSIS: AIM

- measure positive/negative tone
- “emotionality” of a text
- builds on the “cognition -> language” nexus

BASIC SENTIMENT ANALYSIS

1. tokenise text
2. construct a lexicon of sentiment words
3. judge the sentiment words
4. match tokens with sentiment lexicon

Example: YouTube vlog transcripts

1. TOKENISE TEXT

```
## tokens from 1 document.
## caseyneistat_1 :
##      [1] "I"           "think"       "a"
##      [4] "cop"         "is"          "giving"
##      [7] "this"        "guy"         "on"
##     [10] "a"           "bike"        "a"
##     [13] "ticket"      "red"         "light"
##     [16] "apparently"  "like"        "the"
##     [19] "biker"       "I"           "just"
##     [22] "turned"      "red"         "bike"
##     [25] "and"         "well"        "they're"
##     [28] "just"        "posted"      "up"
##     [31] "poor"        "guy"         "those"
##     [34] "good"        "boys"        "this"
##     [37] "close"       "your"        "friends"
##     [40] "came"        "cool"        "I"
##     [43] "didn't"     "look"        "good"
##     [46] "at"         "all"         "oh"
```

2. LEXICON OF SENTIMENT WORDS

- do all words have a potential sentiment?
- maybe focus on adjectives/adverbs, maybe verbs?

Lucky us: many sentiment lexicons exists

2. LEXICON OF SENTIMENT WORDS

The `lexicon` package

```
##           x  y
##  1:    abandon -1
##  2:  abandoned -1
##  3: abandonment -1
##  4:         abba  1
##  5:   abduction -1
##  6:   aberrant  -1
##  7:  aberration -1
##  8:     abhor  -1
##  9:  abhorrent -1
## 10:    ability  1
```

LEXICON::HASH_SENTIMENT_NRC

```
##                                x
##  1:                            a a
##  2:      a bad amount of money
##  3:              a bad mother
##  4:      a bad taste in my mouth
##  5:              a bag o' beagles
##  6: a bag of old and moldy ass
##  7:                            a barlow
##  8:                            a barney
##  9:                            a batman
## 10:                            a bitch
```

LEXICON::HASH_SENTIMENT_SOCAL_GOOGLE

```
head(lexicon::hash_sentiment_socal_google[,1], 10)
```

```
##           x
## 1:      a pillar
## 2:      ab liva
## 3:      able
## 4:  above average
## 5: above mentioned
## 6:      abrasive
## 7:      absent
## 8:      absolute
## 9:      abstract
## 10:     absurd
```

3. JUDGE THE SENTIMENT WORDS

Normal strategy

- crowdsourced annotation
- decide on a judgment scale
- multiple judgments per word
- assess inter-rater reliability

3. JUDGE THE SENTIMENT WORDS

Again: mostly already done for you

```
head(lexicon::hash_sentiment_nrc, 10)
```

```
##           x  y
## 1:   abandon -1
## 2:  abandoned -1
## 3: abandonment -1
## 4:      abba  1
## 5:  abduction -1
## 6:  aberrant  -1
## 7:  aberration -1
## 8:      abhor -1
## 9:  abhorrent -1
## 10:  ability  1
```

Binary judgment: -1 or 1

SCALE JUDGMENTS

```
head(lexicon::hash_sentiment_slangs, 10)
```

```
##           x      y
## 1:         a a -0.5
## 2:   a bad amount of money -0.5
## 3:         a bad mother -0.5
## 4:   a bad taste in my mouth -0.5
## 5:         a bag o' beagles -0.5
## 6: a bag of old and moldy ass -0.5
## 7:         a barlow -0.5
## 8:         a barney -0.5
## 9:         a batman  0.5
## 10:        a bitch -0.5
```

Finer judgment: -1.00, -0.50, 0.50, 1.00

CONTINUOUS JUDGMENT

```
head(lexicon::hash_sentiment_socal_google, 10)
```

```
##           x           y
## 1:      a pillar  2.9045431
## 2:      ab liva -0.9578700
## 3:      able    2.6393740
## 4:  above average  3.2150018
## 5: above mentioned  2.5815803
## 6:      abrasive  1.6751913
## 7:      absent   0.4850649
## 8:      absolute  0.4737880
## 9:      abstract  2.0068557
## 10:     absurd  -3.7778744
```

4. MATCH TOKENS WITH SENTIMENT LEXICON

- Classic approach: one sentiment score (`syuzhet` package)

[...] was devastated when she found out that she had colon cancer. She entered the third phase of her cancer. It was especially terrifying because her husband passed away from lung cancer after receiving chemotherapy. [...]

4. MATCH TOKENS WITH SENTIMENT LEXICON

Classic approach: one sentiment score

```
syuzhet::get_sentiment(example_text)
```

```
## [1] -1.55
```

Strategy: match words from sentiment lexicon, then calculate a document average.

PROBLEM?

4. MATCH TOKENS WITH SENTIMENT LEXICON

- Newer approach: sentiment for each sentence

The `sentimentr` (Rinker) package

```
sentimentr::sentiment(example_text)
```

##	element_id	sentence_id	word_count	sentiment
## 1:	1	1	16	-0.1625000
## 2:	1	2	8	-0.2651650
## 3:	1	3	15	-0.5034878
## 4:	1	4	12	-0.1443376
## 5:	1	5	16	-0.4218750

4. MATCH TOKENS WITH SENTIMENT LEXICON

- Newer approach: sentiment for each sentence
 - needs punctuated data
 - requires good sentence disambiguation
 - without punctuation: whole string = 1 sentence
- What about valence shifters?

This is not ideal.

SENTIMENT TRAJECTORY ANALYSIS

Papers: [vlogs](#), [news channels](#), [drill lyrics](#)

Idea: - sentiment is dynamic within texts - static approaches
mask sentiment dynamics - worst case: sentiment completely
off

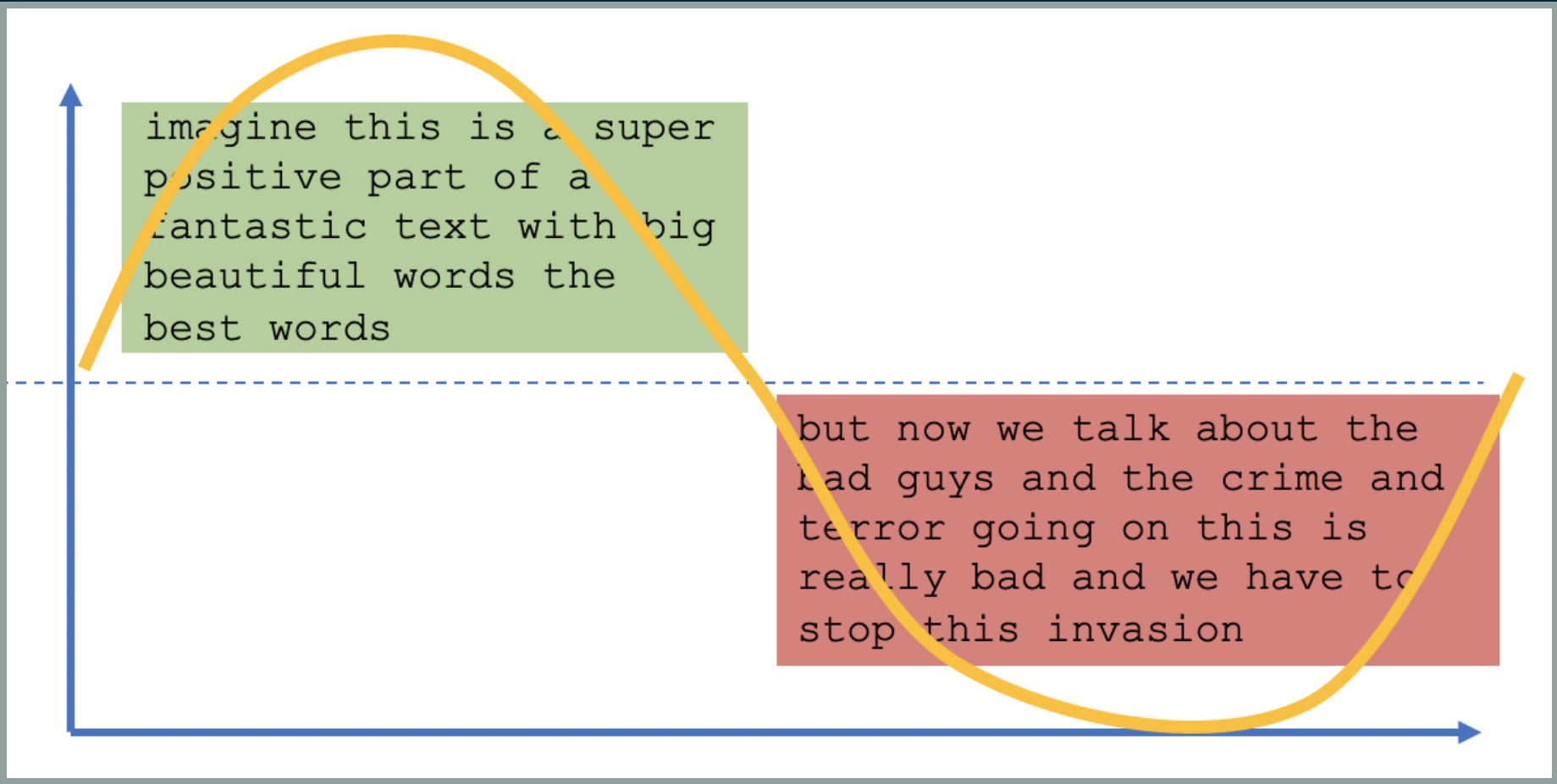
Inspired by Matthew Jockers' [work](#)

imagine this is a super positive part of a
fantastic text with big beautiful words
the best words

+

but now we talk about the bad guys and the
crime and terror going on this is really
bad and we have to stop this invasion

-



imagine this is a super
positive part of a
fantastic text with big
beautiful words the
best words

but now we talk about the
bad guys and the crime and
terror going on this is
really bad and we have to
stop this invasion

SENTIMENT TRAJECTORIES

1. Parse text input into words
2. Match sentiment lexicon to each word
 - Match valence shifters to each context
 - Apply valence shifter weights
 - Build a naïve context around the sentiment
 - Return modified sentiment

SENTIMENT TRAJECTORY ANALYSIS

Package from:

https://github.com/ben-aaron188/naive_context_s

```
## [1] "For external lexicon-databases, please set the current wd to the
```

text	index
it	25
was	26
especially	27
terrifying	28
because	29
her	30
husband	31
passed	32
away	33
from	34
lung	35

MATCH SENTIMENT

text	index	y
it	25	NA
was	26	NA
especially	27	NA
terrifying	28	-1
because	29	NA
her	30	NA
husband	31	NA
passed	32	NA
away	33	NA
from	34	NA
lung	35	NA

MATCH VALENCE SHIFTERS

text	index	y.x	y.y
it	25	NA	NA
was	26	NA	NA
especially	27	NA	2
terrifying	28	-1	NA
because	29	NA	NA
her	30	NA	NA
husband	31	NA	NA
passed	32	NA	NA
away	33	NA	NA
from	34	NA	NA
lung	35	NA	NA

VALENCE SHIFTERS

- 1 = negator (not, never, ...): -1.00
- 2 = amplifier (very, totally, ...): 1.50
- 3 = deamplifier (hardly, barely, ...): 0.50
- 4 = adversative conjunction (but, however, ...): 0.25

APPLY VALENCE SHIFTER WEIGHTS

text	index	sentiment	valence	weights
it	25	NA	NA	1.0
was	26	NA	NA	1.0
especially	27	NA	2	1.5
terrifying	28	-1	NA	1.0
because	29	NA	NA	1.0
her	30	NA	NA	1.0
husband	31	NA	NA	1.0
passed	32	NA	NA	1.0
away	33	NA	NA	1.0
from	34	NA	NA	1.0
lung	35	NA	NA	1.0

SENTIMENT TRAJECTORIES

Build 'naive' context around sentiment

- 2 words around sentiment word

text	index	sentiment	valence	weights
was	26	NA	NA	1.0
especially	27	NA	2	1.5
terrifying	28	-1	NA	1.0
because	29	NA	NA	1.0
her	30	NA	NA	1.0

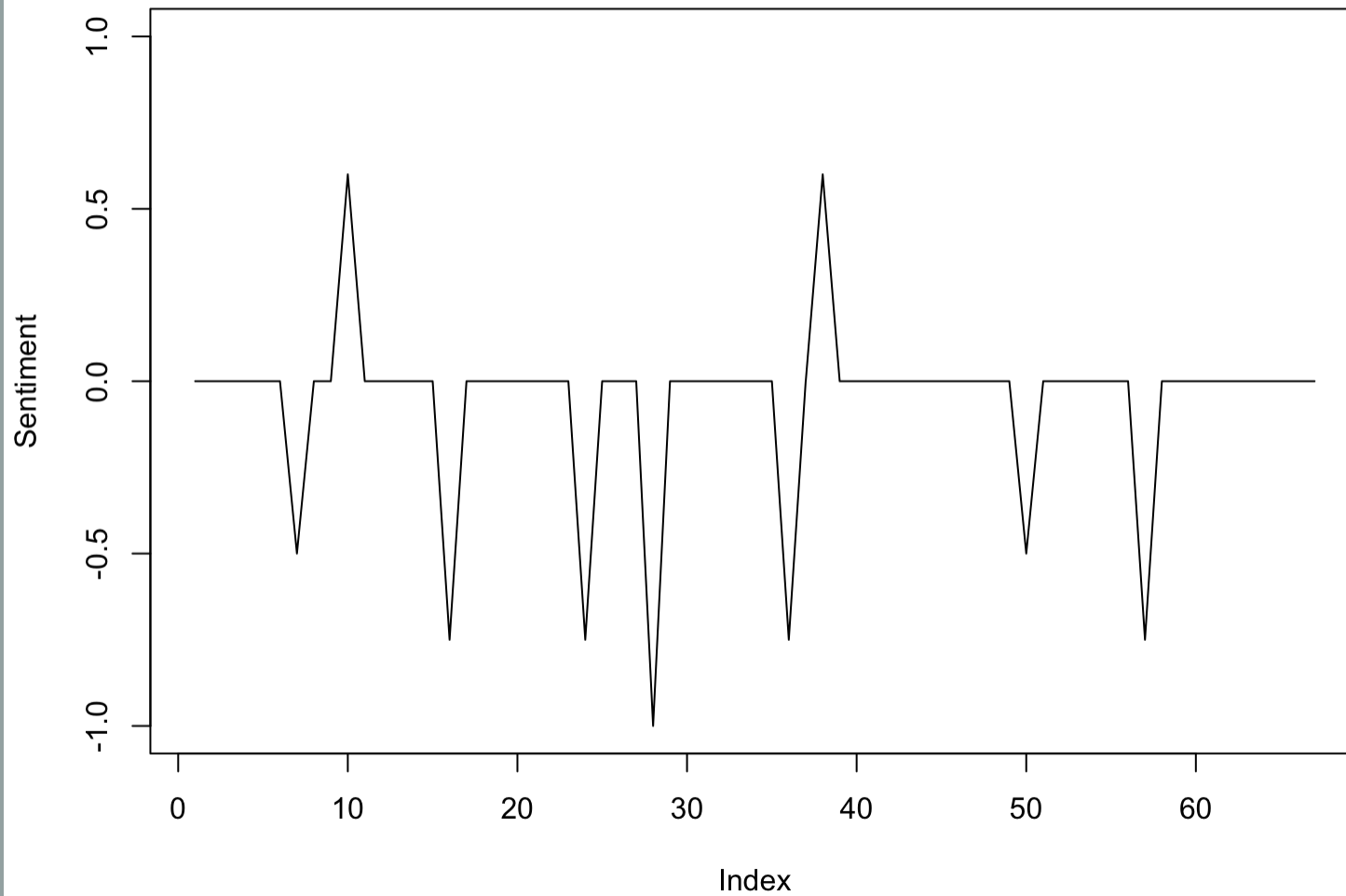
CALCULATE MODIFIED SENTIMENT

```
## [1] "sentiment change for \"devastated\": -0.5 --> -0.5"
## [1] "sentiment change for \"found\": 0.6 --> 0.6"
## [1] "sentiment change for \"cancer\": -0.75 --> -0.75"
## [1] "sentiment change for \"cancer\": -0.75 --> -0.75"
## [1] "sentiment change for \"terrifying\": -1 --> -1.5"
## [1] "sentiment change for \"cancer\": -0.75 --> -0.75"
## [1] "sentiment change for \"receiving\": 0.6 --> 0.6"
## [1] "sentiment change for \"exposed\": -0.5 --> -0.125"
## [1] "sentiment change for \"cancer\": -0.75 --> -0.75"
```

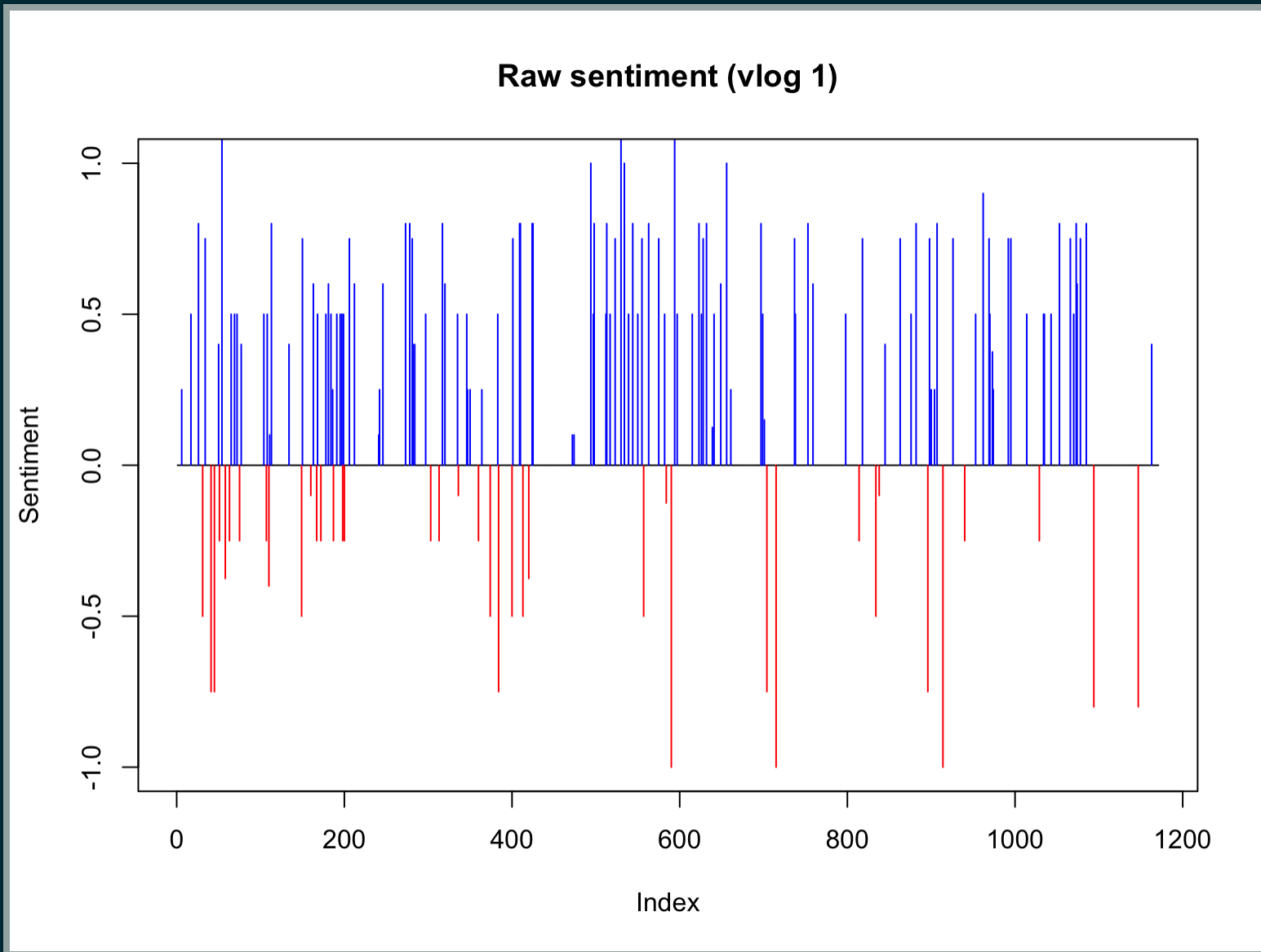
text	sentiment	valence	weights	sentiment_score_mod
was	NA	NA	1.0	0.0
especially	NA	2	1.5	0.0
terrifying	-1	NA	1.0	-1.5
because	NA	NA	1.0	0.0
her	NA	NA	1.0	0.0

TRAJECTORIES

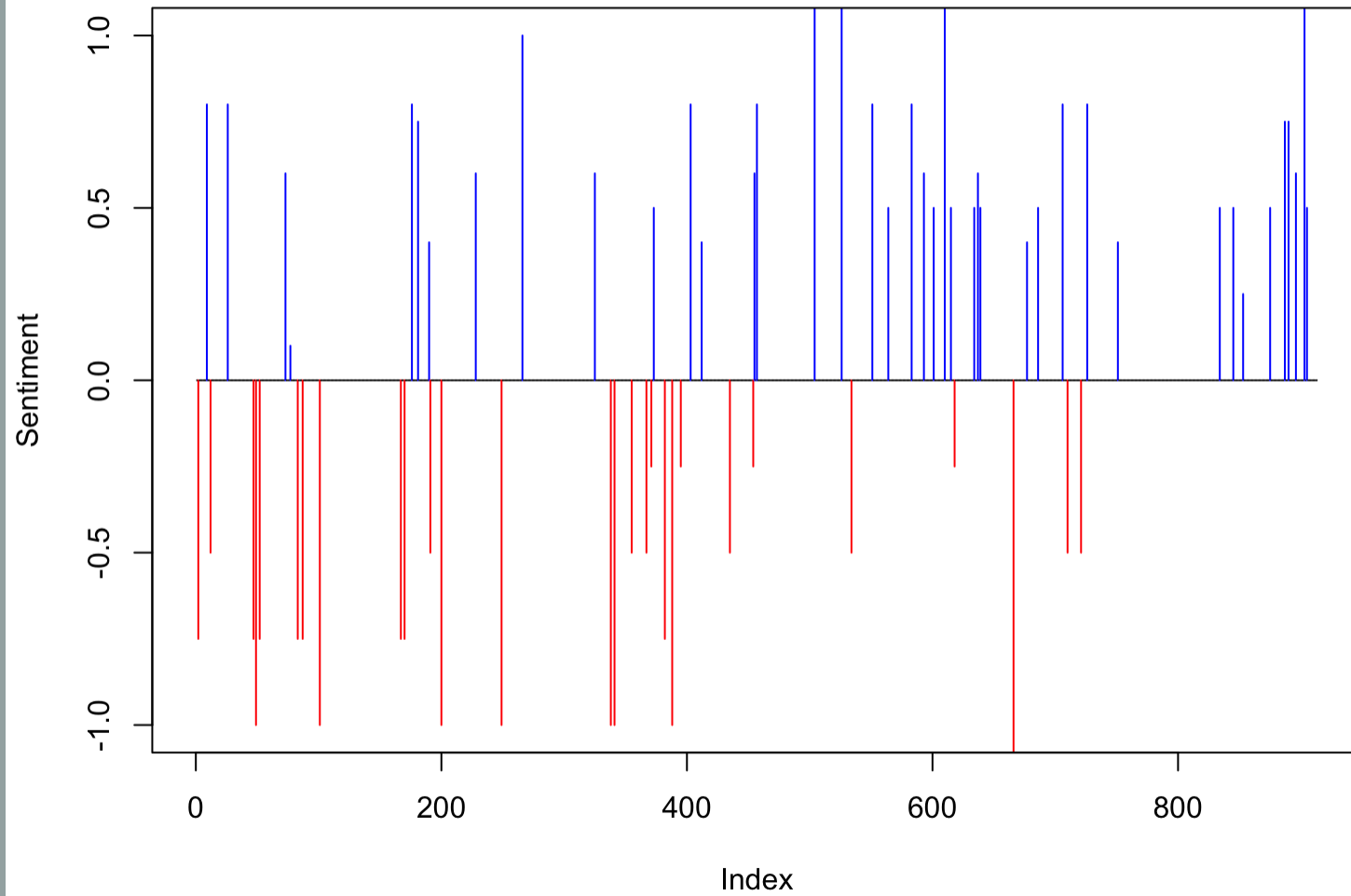
Raw sentiment (example string)



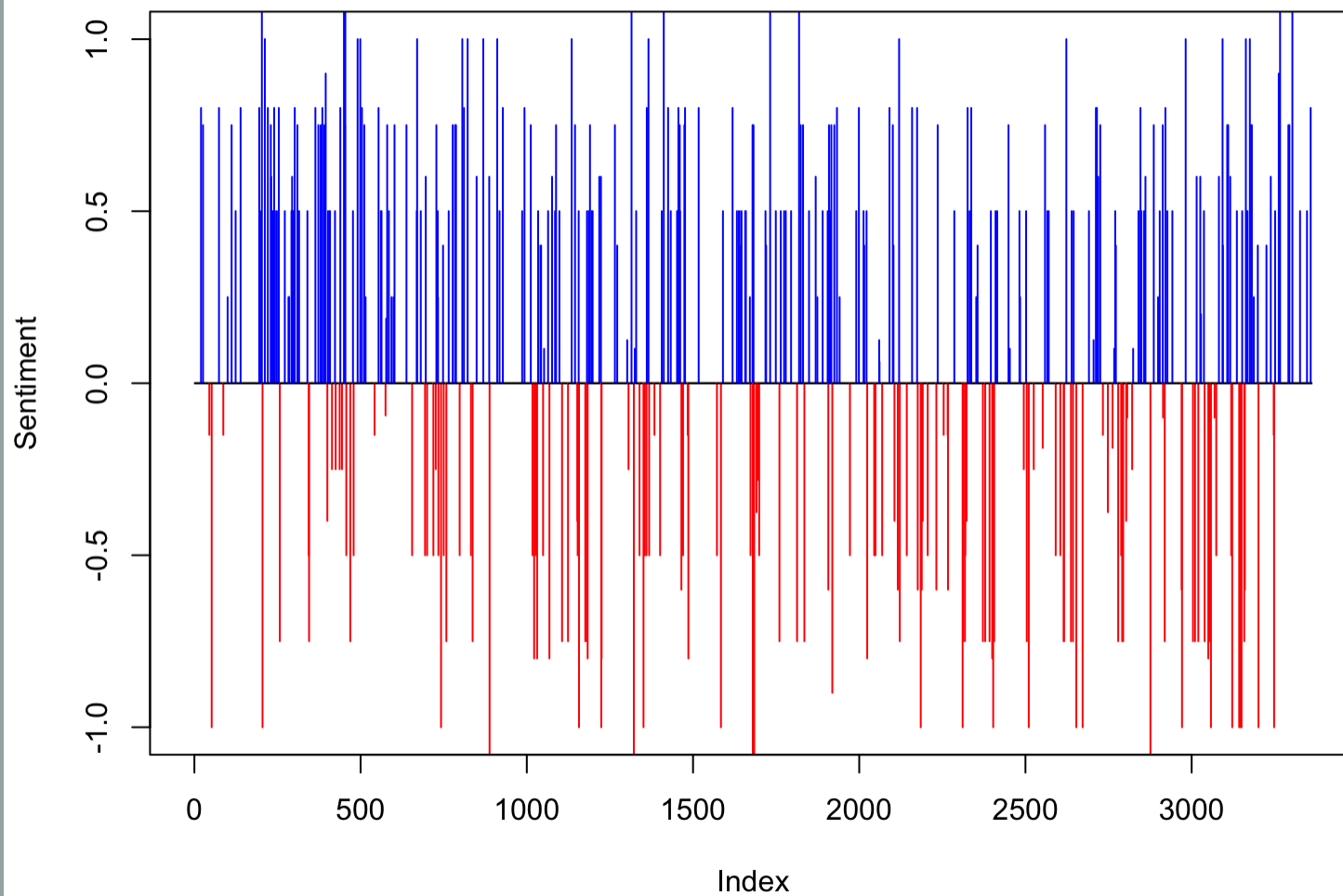
VLOG DATA



Raw sentiment (vlog 5)



Raw sentiment (vlog 99)

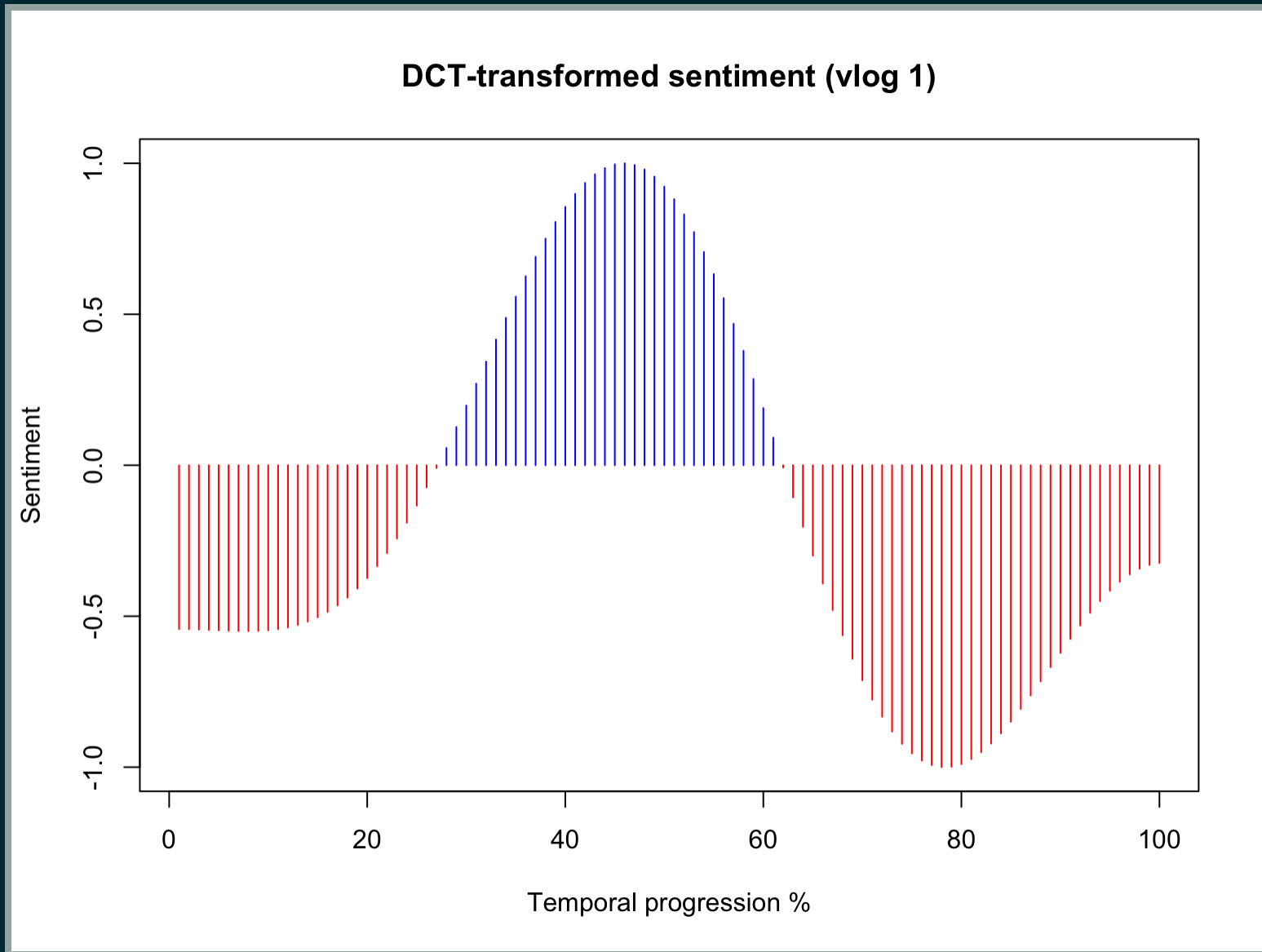


PROBLEM?

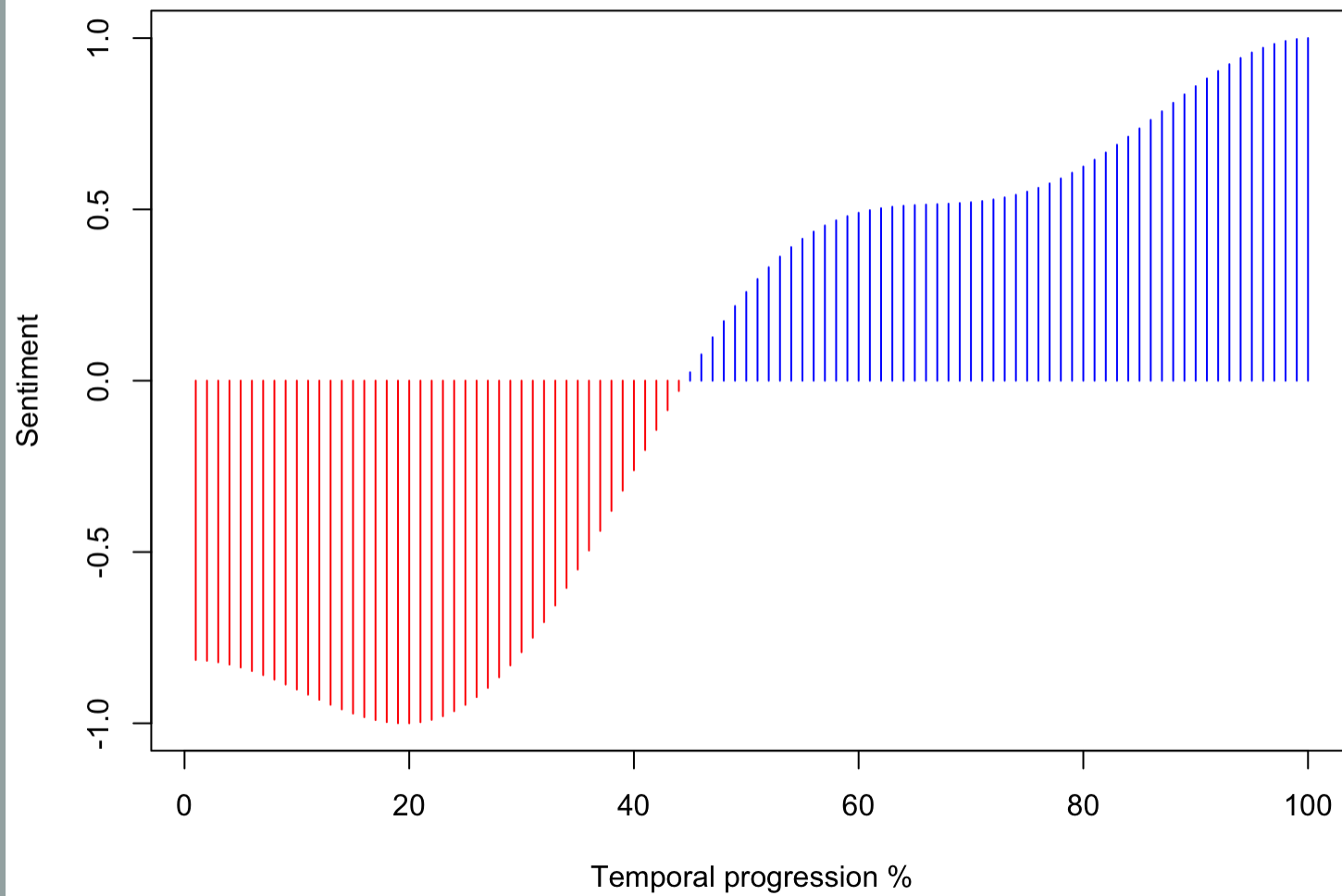
SOLUTION: LENGTH-STANDARDISATION

- aim: transform all sentiment values to a vector
- standard vector length for comparisons
- here: 100 values with Discrete Cosine Transformation (explainer on Fourier Transformation)

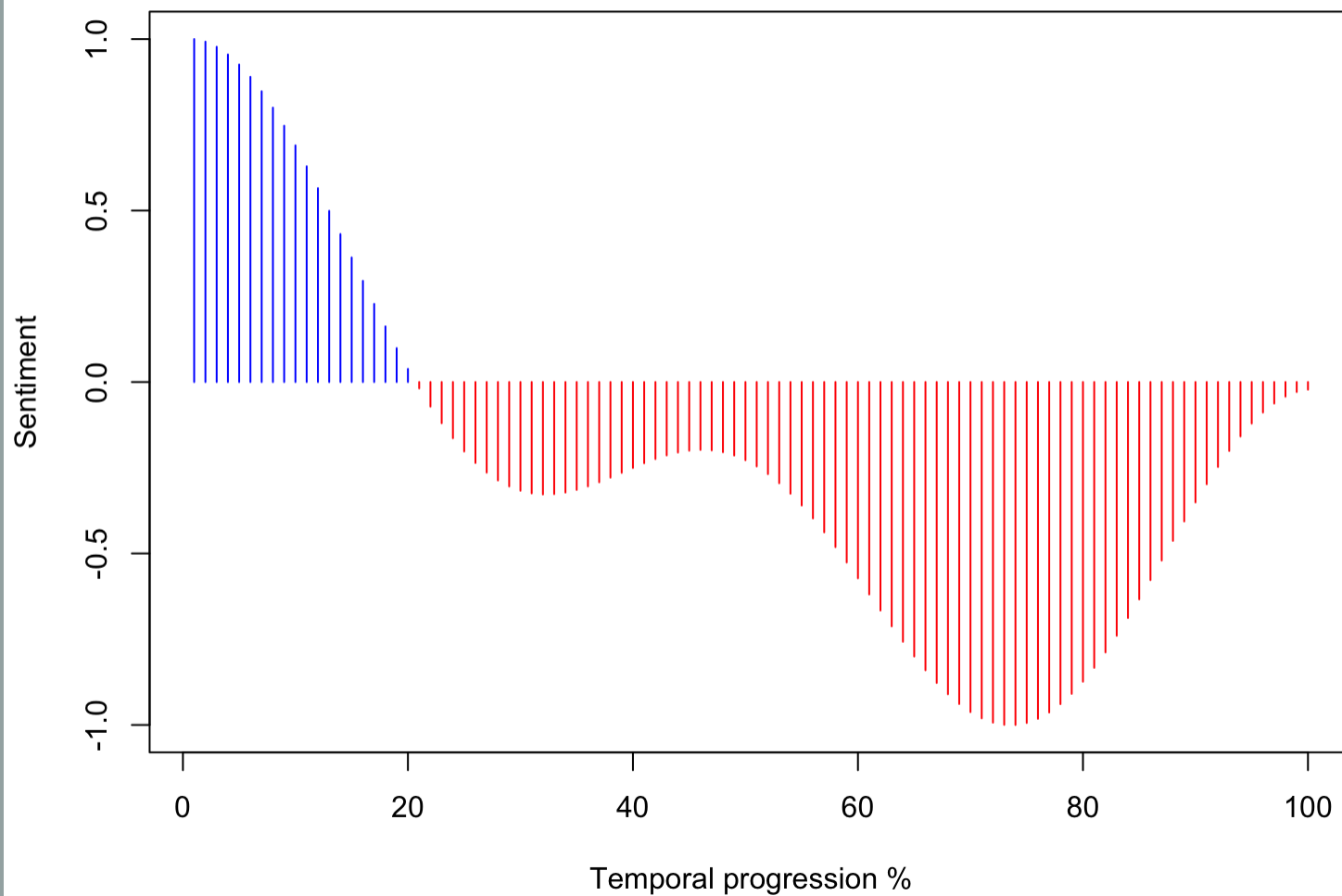
AFTER DISCRETE COSINE TRANSFORMATION



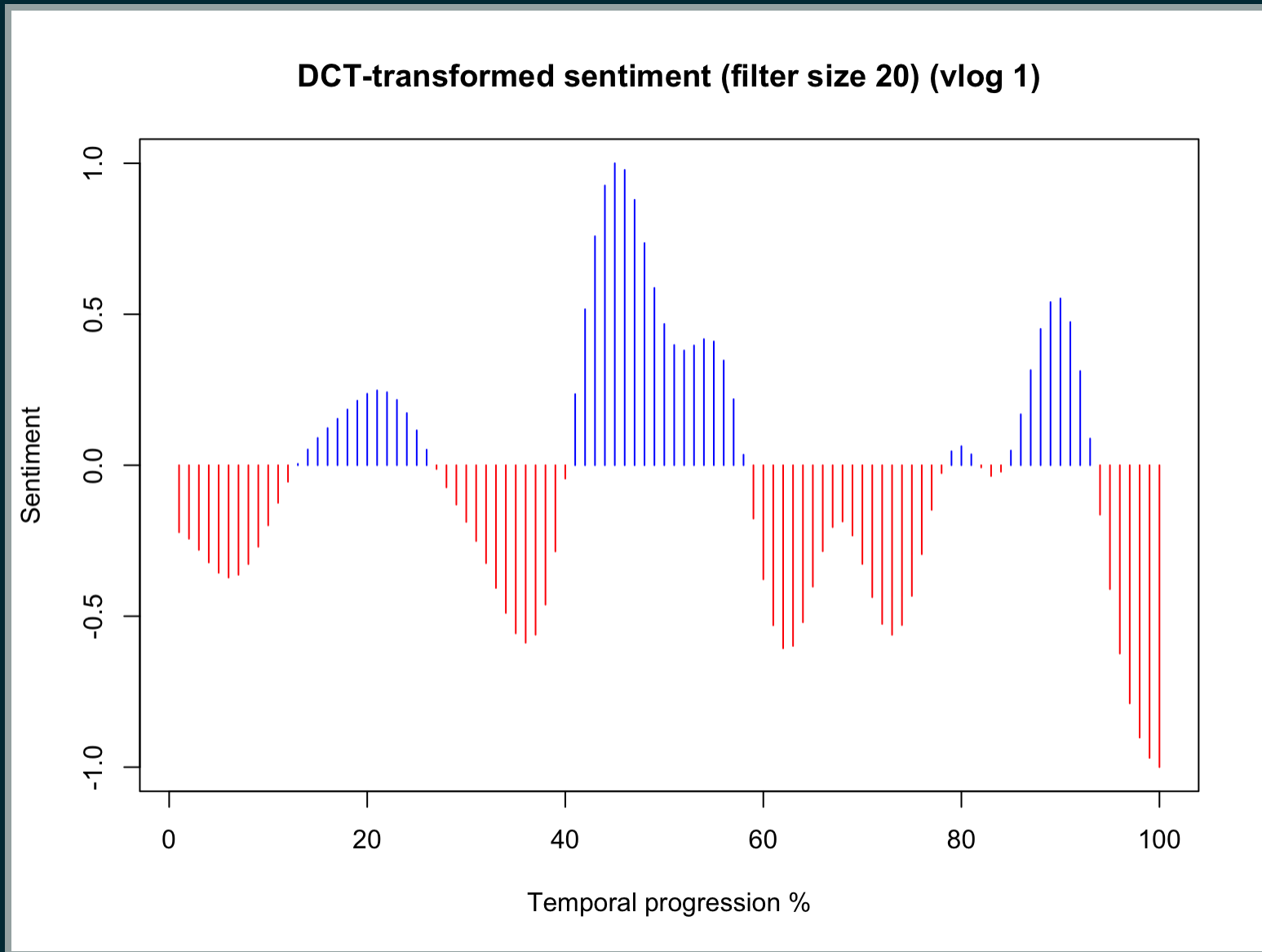
DCT-transformed sentiment (vlog 5)



DCT-transformed sentiment (vlog 99)



BEWARE OF THE “FILTER” SIZE



WHAT'S NEXT?

- Today's tutorial: ngrams, POS tagging, sentiment analysis, trajectory analysis
- Homework: KWIC analysis, trajectory parameters

Next week: Text Mining 3