Topic and thesaurus extraction from a document collection

Template *Mathematica* code using NPR transcripts

Anton Antonov
Mathematica for Prediction blog
Mathematica for Prediction project at GitHub
October 2013

Introduction

In this paper we present a template for descriptive statistics analysis and topic and thesaurus extraction for a collection of documents. Both the analysis and topic and thesaurus extraction belong to the field of Natural Language Processing (NLP). The collection of documents used is comprised of National Public Radio (NPR) podcast transcripts, which are available at http://www.npr.org -- see for example http://www.npr.org/templates/transcript/transcript.php?storyId=230950294. (We use nearly 5000 transcripts in this paper.)

The template has the following steps.

1. Ingestion of documents.
2. Removal of stop words and word stemming.
3. Linear vector space representation.
4. Computation of descriptive statistics.
5. Application of different weight functions to the linear vector space representation.
6. Topic extraction with a matrix factorization method.
7. Statistical thesaurus finding using the factorization in step 6.

We describe these steps in detail and give some theoretical clarifications.

For the conversion of documents into points of a linear vector space we use the *Mathematica* package DocumentTermMatrixConstruction.m provided by the project MathematicaForPrediction at GitHub, see [1].

For the topic extraction we use the *Mathematica* package NonNegativeMatrixFactorization.m also provided by the project MathematicaForPrediction at GitHub, see [2].

In general, in this paper we are speak about documents, but we use the word “transcript” when we want to hint the origin of the document.

1. Reading and ingestion of documents

Obviously, the gathering and ingestion of the documents can be done in many ways depending on the sources and storage schemes. With *Mathematica* we can easily ingest from web pages or databases. In any case in this paper we assume that the collection of documents is a list of strings.

Here is a table of the first 100 characters of six randomly selected documents from the collection (which is assigned to the symbol `documents`).

```
Grid[List /@ Map[StringTake[#, {1, 100}] &,
   documents[[RandomInteger[{1, 400}, 6]]]],
 Alignment → Left, Dividers → All]
```

ROBERT SIEGEL, host:This is ALL THINGS CONSIDERED from NPR News. I'm Robert Siegel.MICHELE NORRIS, h
MELISSA BLOCK, host:Oh, heartbreak.(soundbite of Elvis singing)Elvis Presley's first number one pop
ALEX CHADWICK, host:This DAY TO DAY from NPR News. Readiness for a a possible avian flu epidemic was
RENEE MONTAGNE, host:It's become a holiday tradition on MORNING EDITION to invite commentator M
(Soundbite of music)JENNIFER LUDDEN, host:If I told you this music from a new CD called "H1Bees"--th
ROBERT SIEGEL, host:This is ALL THINGS CONSIDERED from NPR News. I'm Robert Siegel.The artist known

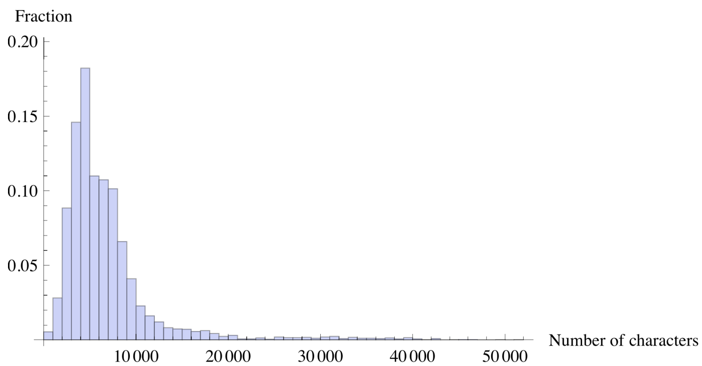
We have $\approx$ 5000 documents:

```
documents // Length
```

5123

Here is a histogram of their string lengths:

```
Histogram[StringLength /@ documents, Automatic, "Probability",
 AxesLabel → {"Number of characters", "Fraction"}]
```



2. Removal of stop words and word stemming

Stop words

In information retrieval “stop words” are removed from texts prior to natural language processing. Loosely speaking stop words have little semantic meaning. See [3].

Here is the list of 319 stop words in English we use (assigned to the symbol `stopWords`):

```
Magnify[stopWords, 0.7]
```

```
{a, about, above, across, after, afterwards, again, against, all, almost, alone, along, already,
 also, although, always, am, among, amongst, amoungst, amount, an, and, another, any, anyhow,
 anyone, anything, anyway, anywhere, are, around, as, at, back, be, became, because, become,
 becomes, becoming, been, before, beforehand, behind, being, below, beside, besides, between,
 beyond, bill, both, bottom, but, by, call, can, cannot, cant, co, computer, con, could,
 couldnt, cry, de, describe, detail, do, done, down, due, during, each, eg, eight, either,
 eleven, else, elsewhere, empty, enough, etc, even, ever, every, everyone, everything,
 everywhere, except, few, fifteen, fify, fill, find, fire, first, five, for, former, formerly,
 forty, found, four, from, front, full, further, get, give, go, had, has, hasnt, have, he,
 hence, her, here, hereafter, hereby, herein, hereupon, hers, herself, him, himself, his,
 how, however, hundred, i, ie, if, in, inc, indeed, interest, into, is, it, its, itself,
 keep, last, latter, latterly, least, less, ltd, made, many, may, me, meanwhile, might,
 mill, mine, more, moreover, most, mostly, move, much, must, my, myself, name, namely,
 neither, never, nevertheless, next, nine, no, nobody, none, noone, nor, not, nothing, now,
 nowhere, of, off, often, on, once, one, only, onto, or, other, others, otherwise, our,
 ours, ourselves, out, over, own, part, per, perhaps, please, put, rather, re, same, see,
 seem, seemed, seeming, seems, serious, several, she, should, show, side, since, sincere,
 six, sixty, so, some, somehow, someone, something, sometime, sometimes, somewhere, still,
 such, system, take, ten, than, that, the, their, them, themselves, then, thence, there,
 thereafter, thereby, therefore, therein, thereupon, these, they, thick, thin, third, this,
 those, though, three, through, throughout, thru, thus, to, together, too, top, toward,
 towards, twelve, twenty, two, un, under, until, up, upon, us, very, via, was, we, well, were,
 what, whatever, when, whence, whenever, where, whereafter, whereas, whereby, wherein,
 whereupon, wherever, whether, which, while, whither, who, whoever, whole, whom, whose,
 why, will, with, within, without, would, yet, you, your, yours, yourself, yourselves}
```

Here is a list of additional stop words -- these are words that appear in more than 60% of the NPR transcripts.

term	%	term	%	term	%
copyright	1.	npr	1.	provided	1.
transcript	1.	host	0.991802	like	0.87156
just	0.865508	soundbite	0.844622	know	0.800703
new	0.776498	time	0.755222	people	0.733555
music	0.726332	news	0.724966	think	0.695881
don	0.686902	really	0.68007	going	0.6748
way	0.670115	years	0.669334	ve	0.654109
called	0.643568	say	0.632247	things	0.623072

The list of additional stop words can be derived with the following commands.

```
wordsTally = Tally[
   Flatten[Map[Complement[Union[Select[StringSplit[ToLowerCase[#],
          {{Whitespace, "\n", " ", ".", ",", "!", "?", ";",
            ":", "-", "\"", "'", "(", ")", "“", "`"}}],
         StringLength[#] >= 2 &]], stopWords] &, documents]]];
```

```
wordsTally // Length
```

```
67 092
```

```
wordsTally[[1 ;; 45, 1]]
```

```
{act, ahead, alex, alley, american, apartment, argument, art,
 ask, assert, attracted, audience, audio, backstage, band,
 beat, beats, beginning, beginnings, betty, bikini, boring,
 brings, buns, butter, called, came, carried, cause, chadwick,
 chance, cinna, cinnamon, computers, copyright, couldn, course,
 culture, david, day, didn, different, doesn, don, drag}
```

```
newStopWords =
  SortBy[Select[wordsTally, #[[2]] > 0.6 Length[documents] &], -#[[2]] &];
```

```
newStopWords[[All, 2]] = N[newStopWords[[All, 2]] / Length[documents]];
newStopWords
```

```
{{copyright, 1.}, {npr, 1.}, {provided, 1.},
 {transcript, 1.}, {host, 0.991802}, {like, 0.87156},
 {just, 0.865508}, {soundbite, 0.844622},
 {know, 0.800703}, {new, 0.776498}, {time, 0.755222},
 {people, 0.733555}, {music, 0.726332}, {news, 0.724966},
 {think, 0.695881}, {don, 0.686902}, {really, 0.68007},
 {going, 0.6748}, {way, 0.670115}, {years, 0.669334},
 {ve, 0.654109}, {called, 0.643568}, {say, 0.632247},
 {things, 0.623072}, {right, 0.609994}, {got, 0.607261}}
```

Stemming

Stemming is a process of reducing inflected or derived words to their root, base, or stem;

see [4].

In this paper we are going to use ther word “terms” to mean “stemmed words”.

Here is table with popular terms within the document collection and words that are stemmed to them.

term	words					
abl	able	ables				
creat	create	created	creates	creating		
critic	critic	critical	critically	criticism	criticisms	criticize
die	die	died	dies	dying		
earli	early					
far	far					
month	month	monthly	months			
school	school	schooled	schooling	schools		
second	second	secondly	seconds			
understand	understand	understandable	understandably	understanders	understanding	understandings
walk	walk	walked	walking	walks		

For stemming we can use *Mathematica*’s function `WordData`:

```
WordData[#, "PorterStem"] & /@ {"able", "schooling", "critical"}
```

```
{abl, school, critic}
```

Using an external stemmer

We can also use and external stemmer as the stemmer called snowball see http://snowball.-tartarus.org. In this case we do the following steps.

1. Find all individual words used in the document collection.
2. Export all words into a text file.
3. Using the function Run invoke the stemmer with appropriate command arguments.
4. Read the output of the stemmer.
5. Make a list of rules for replacing words with their stems.

Example code using an external stemmer

```
allWords = wordsTally⟦All, 1⟧;
```

```
wordsToStem = Complement[Select[allWords,
    StringMatchQ[#, LetterCharacter ..] &], stopWords];
wordsToStem // Length
```

```
63 241
```

```
Export["~/MathFiles/text_words.txt", wordsToStem]
```

```
~/MathFiles/text_words.txt
```

```
Run["~/snowball/libstemmer_c/stemwords
   -l english -i ~/MathFiles/text_words.txt
   -o ~/MathFiles/text_words_stemmed.txt"]
```

```
0
```

```
stemmedWords =
  StringSplit[Import["~/MathFiles/text_words_stemmed.txt"]];
stemmedWords // Length
```

```
63241
```

```
stemmingRules = Dispatch[Thread[wordsToStem → stemmedWords]];
```

3. Linear vector space representation

Given a document its words can be taken without regard of their order in the document. We say we turn the document into a “bag of words”. If we use stemming then we turn the document into a bag of terms (stemmed words).

Let us assume that the number of documents in the collection is m and the total number of words used in all documents is n. With the bag-of-words transformation each document can seen as a point in a $\mathbb{R}^n$ linear vector space, each axis of which corresponds to a word. Then the whole document collection can be seen as a sparse matrix in $\mathbb{R}^{m \times n}$.

Assume that we have ordered in some way all the words (terms) in the document collection and in the space of words (terms) $\mathbb{R}^n$ the axis e_w corresponds to the word (term) w. We represent the document D as a point in $\mathbb{R}^n$ in the following way:
1. turn D into a bag of words;
2. stem the words of D;
3. for each term w:
3.1. if w does not appear in D then the coordinate of e_w is 0,
3.2. if w appears f_w times in D then the coordinate of e_w is f_w.

In this representation we can derive the document × term frequency matrix $F \in \mathbb{R}^{m \times n}$ that corresponds to the document collection. The frequency matrix F is further transformed to reflect better the significance of the words in the document collection using different weight functions. (See the section “Weight functions”.)

We can compute the representation of the document collection into a linear vector space with the functions provided in the package DocumentTermMatrixConstruction.m, [1].

```
Get["~/MathFiles/MathematicaForPrediction/
   DocumentTermMatrixConstruction.m"]
```

The function `DocumentTermMatrix` takes a list of strings and returns a sparse matrix and a list of terms. The returned sparse matrix is the representation of the document collection into a linear vector space with axes corresponding to the returned terms.

```
AbsoluteTiming[
 {F, terms} = DocumentTermMatrix[ToLowerCase /@ documents,
    {stemmingRules, Join[stopWords, newStopWords]}];
]
{68.068904, Null}
```

```
F
SparseArray[<1 403 565>, {5123, 45 627}]
```

```
terms // Length
45 627
```

Depending on the documents source it can happen that a number of terms are not words or stems of words. For example, in the list of terms found with the previous command using `DocumentTermMatrix` we find more than 3500 terms that are not comprised of letter characters.

```
nonWords = Select[terms, ! StringMatchQ[#, LetterCharacter ..] &];
nonWords // Length
RandomSample[nonWords, 12]
3674
{country…mr, …vaduz, $443, sand…thompson, me…mr, 1925s,
 it…tyler, �and�seabrook, know…gross, 1400s, ‘60s, �abandoned}
```

If we just want to convert a string into a bag of words we can use the function `ToBagOfWords` (which is used by `DocumentTermMatrix`).

```
wordBag = ToBagOfWords[
   ToLowerCase@documents⟦1⟧, {stemmingRules, stopWords}];
SortBy[Tally[wordBag], -#⟦2⟧ &]⟦1 ;; 12⟧
{{like, 19}, {sing, 16}, {hanna, 15},
 {soundbit, 15}, {song, 13}, {got, 12}, {bikini, 11},
 {kill, 11}, {want, 9}, {band, 8}, {npr, 7}, {tigr, 7}}
```

4. Computation of descriptive statistics

Here are some of the basic descriptive statistics we can do over the collection of documents.

1. Total number of documents.

2. Total number of words and total number of stemmed words (terms).

3. Number of terms per document.

4. Number of documents per term.
5. Average number of words in each document.
6. Other statistics, like number of characters, title frequency, etc.

Documents per term

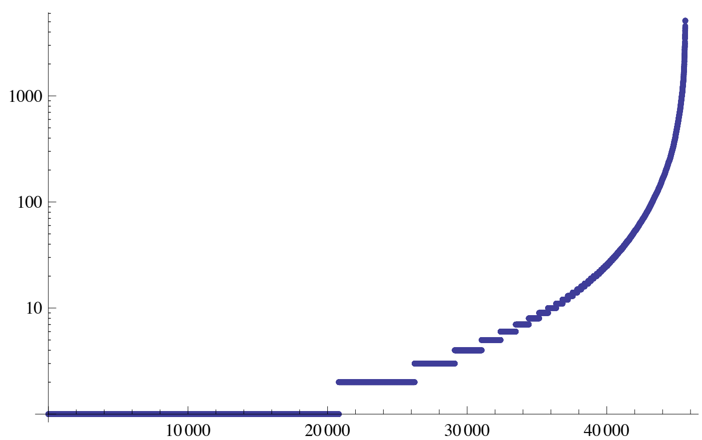
Let us compute descriptive statistics for the number of documents per term.

```
documentsPerTerm = Total /@ Transpose[Clip[F, {0, 1}]];
TableForm[{{Min, Max, Mean, Median, StandardDeviation},
  Through[{Min, Max, N[Mean[#]] &, Median,
     N[StandardDeviation[#]] &}[documentsPerTerm]]}]
```

Min	Max	Mean	Median	StandardDeviation
1	5123	30.7617	2	172.999

For this kind of data using `ListLogPlot` is more informative than `Histogram`:

```
ListLogPlot[Sort[documentsPerTerm], PlotRange → All]
```



Terms per document

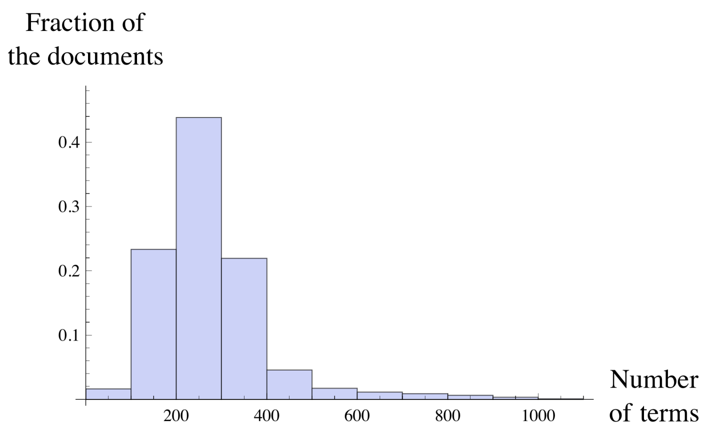
Let us compute descriptive statistics for the number of terms per document.

```
termsPerDocument = Total /@ Clip[F, {0, 1}];
TableForm[{{Min, Max, Mean, Median, StandardDeviation},
  Through[{Min, Max, N[Mean[#]] &, Median,
     N[StandardDeviation[#]] &}[termsPerDocument]]}]
```

Min	Max	Mean	Median	StandardDeviation
6	1117	273.973	251	125.379

We can get an idea of the terms distribution with a histogram.

```
Histogram[termsPerDocument, {0, 1100, 100},
  "Probability", AxesLabel → (Style[#, FontSize → 14] & /@
    {"Number\nof terms", "Fraction of\nthe documents"})]
```



5. Weight functions

We can take the approach used in search engines for calculating weights for document-term matrices. (See [5].)

Frequency matrix

We use the following definitions of the frequency matrix F.

Each entry $f_{i\,j}$ of the matrix F is the number of occurrences of the term j in the list of terms of the document i.

Weights

The matrix F is transformed into the matrix M. Each entry of the matrix F is transformed with the formula

$m_{i\,j} = l_{i\,j}\, g_j\, d_i$

where

$l_{i\,j}$ -- local term weight;

g_j -- global term weight;

d_i -- normalization weight.

Various formulas exist for these weights and one of the challenges is to find the right combination for each collection of documents we work with.

weight type	name	formula
local	Binary	$\chi(f_{ij})$
local	Logarithmic	$\log(f_{ij}+1)$
local	Term frequency (TF)	f_{ij}
global	None	1
global	Inverse document frequency (IDF)	$\log\left(\frac{7472}{\sum_j \chi(f_{ij})}\right)$
global	Global frequency inverse document frequency (GFIDF)	$\frac{\sum_j f_{ij}}{\sum_j \chi(f_{ij})}$
global	Normal	$\frac{1}{\sqrt{\sum_i f_{ij}^2}}$
normalization	None	1
normalization	Cosine	$\frac{1}{\sqrt{\sum_j g_j\, l_{ij}}}$

After applying the chosen weight functions to the elements of *F* we get the matrix *M*. This re-weighting of *F* can be done using the function `WeightTerms` from the package DocumentTermMatrixConstruction.m, [1].

```
AbsoluteTiming[
 M = WeightTerms[F, GlobalTermWeight["GFIDF", #1, #2] &, # &, # &]
]
```

```
{1.403356, SparseArray[<1 403 565>, {5123, 45 627}]}
```

6. Topic extraction

Using a matrix factorization method we can extract topics from *M*.

Topic extraction is very similar to dimension reduction and traditionally for dimension reduction the thin Singular Value Decomposition (SVD) is applied to *M*. Because SVD generally produces vectors with mixed positive and negative coordinates we would have difficulties to interpret them into topics.

We use Non-Negative Matrix Factorization (NNMF) for topic extraction from *M*, see [6,7]. The vectors produced by NNMF have positive coordinates and can be easily interpreted. NNMF is not unique (SVD is). NNMF has convergence issues and because of them the initialization of NNMF is important, see [6] for more details.

Describing the algorithms for SVD and NNMF is beyond the scope of this document. Sparse matrix linear algebra libraries usually have SVD implemented. (*Mathematica*'s SVD function is named `SingularValueDecomposition`.)

Topics

Assume we have ten thousand documents, and hence ten thousand bags of words. Topic extraction can be seen as finding a certain number of bags, say 200, for which the following statement is true:

Given a document, 80% of its characterizing words are contained in a small number of the topic bags of words.

We can say that a document is characterized by the topics it consists of. Or in other words the documents are decomposed into topics.

The topics are the rows of the right factor in a SVD or NNMF for the document × term matrix M.

We need to decide which terms comprise a topic. This is best done by some outlier detection procedure. Alternatively, we can simply do the following: given a topic vector t take a certain number of terms that have the largest (and non-zero) coordinates in t.

Theoretical interpretations

Consider the NNMF factorization of $M \in \mathbb{R}^{m \times n}$

$$M \approx W H,\ W \in \mathbb{R}^{m \times k},\ H \in \mathbb{R}^{k \times n},\ W \geq 0,\ H \geq 0. \tag{1}$$

The factorization is derived by solving the (non-linear) optimization problem

$$\begin{aligned} &\min \|M - W H\|_F^2, \\ &W \geq 0, \\ &H \geq 0. \end{aligned} \tag{2}$$

Let us interpret the factors W and H. Each row of the document×term matrix M represents a document in the space of terms. In (1) the integer k is chosen the be much smaller than n, $k <<$ n. The rows of the factor H group the terms into k vectors and those k vectors are used to express each document: each row of H is a basis vector. Assume that (1) is done in such a way that the norms of the rows of H are 1. The i-th row of W, that corresponds to the i-th document in the collection, has coordinates for the basis given by the rows of H. This interpretation follows from the equation

$$M_i \approx \sum_{j=1}^{k} w_{i,j} H_j, \tag{3}$$

in which we denoted with M_i the i-th row of M, with H_j the j-th row of H, and with $w_{i,j}$ the entry of W at row i and column j. We say that each row of H is a topic and with W we have mapped each document into the space of topics. The number of topics is k. In other words with W we reduced the dimension of the document collection matrix representation M.

Using W we can cluster the documents or find nearest neighbors using the Euclidean distance -- if two documents use the same set of topics to a similar degree then these documents are similar.

Note that each column i of W corresponds to a i-th topic (row) in H. Let us denote the i-th column of W with $W(:, i)$. We can reason about the i-th topic properties looking at $W(:, i)$.

If a small fraction of the coordinates of $W(:, i)$ are non-zero and large then that topic is somewhat specialized and does not mesh much with the others. If almost all coordinates of $W(:, i)$ are non-zero then the topic is presented in almost every document and it is probably made of words with little semantic meaning (within the document collection).

Let us take an alternative point of view. We can say that each column of M represents a term in the space of documents in which each document is a basis vector. Assume that we change (1) in such a way that the norms of the columns of W are 1. Then we can cluster the columns of H using the Euclidean distance in oreder to derive a statistical thesaurus based on the document collection.

Note that the basis given by the rows of H is not orthogonal, (2) ensures the positivity of the coordinates of the basis vectors but not their orthogonality.

Computation

In order to extract topics from the document collection we are going to use the NNMF implementation provided by the MathematicaForPrediction project at GitHub, see [2]:

```
Get["~/MathFiles/MathematicaForPrediction/
   NonNegativeMatrixFactorization.m"]
```

First let us select only those terms that are present in at least, say, 25 documents. We can say that the rest of the terms are not significant. We do this mostly to speed up the computations, but also, in effect, we are filtering out terms that do not come from natural language words.

```
pos = Flatten[Position[documentsPerTerm, s_?NumberQ /; s ≥ 25]];
pos // Length
```

```
5739
```

```
M1 = M[[All, pos]]
```

```
SparseArray[<1 261 785>, {5123, 5739}]
```

Next we initialize the NNMF factors W and H. The initialization is not necessary since the package function `GDCLS` for computing NNMF does the “standard” initialization of W and H -- the entries of W are random numbers in $[0, 1]$ and all entries of H are 0. The initialization we present here, though, speeds up the convergence and it can be used as a base for more complicated initialization procedures like the ones described in [6]. In order to initialize the i-th column of W we randomly select `p` columns of M and their sum becomes a i-th column of W. (We do this `k` times.) This procedure is done faster if we transpose the matrices M and W.

```
{k, p} = {60, 12};
{m, n} = Dimensions[M1];
M1 = Transpose[M1];
M1 = Map[# &, M1];
H = ConstantArray[0, {k, n}];
W = Table[Total[RandomSample[M1, p]], {k}];
Do[
 W[[i]] = W[[i]] / Norm[W[[i]]];
 , {i, 1, Length[W]}]
W = Transpose[W];
M1 = SparseArray[M1];
M1 = Transpose[M1];
```

The package [2] provides two functions for NNMF: `GDCLS` and `GDCLSGlobal`. The later is used to continue the NNMF factorization iterations for given three symbols associated with the matrices in (1) and hence we can use `GDCLSGlobal` with the initialized factors.

```
W = SparseArray[W];
H = SparseArray[H];
{W, H} = GDCLSGlobal[M1, W, H, "MaxSteps" → 6,
    "PrintProfilingInfo" → True]; // AbsoluteTiming
```

```
1 {160.705660, Null}
2 {162.414051, Null}
3 {170.033691, Null}
4 {169.355056, Null}
5 {167.001808, Null}
6 {168.205880, Null}
{708.228433, Null}
```

The extracted topics

In order to interpret the rows of H as topics we need to change the product $W\,H$ in such a way that the norms of the rows of H are 1. This can be done with the function `RightNormalizeMatrixProduct` of [2]:

```
{W, H} = RightNormalizeMatrixProduct[W, H];
```

In order to print out the interpretations of the rows of H as topics we need to convert H from a sparse array to a list of lists structure. (We do this for W too.)

```
{W, H} = Normal /@ {W, H};
```

The function `BasisVectorInterpretation` of [2] can be used to get the larges coordinates of a vector and find the terms corresponding to them.

```
BasisVectorInterpretation[H〚2〛, 12, terms〚pos〛]
```

```
{{0.539105, kid}, {0.349029, know}, {0.261493, parent},
 {0.225796, hansen}, {0.175637, school}, {0.164315, conan},
 {0.157248, say}, {0.154121, children}, {0.144062, stori},
 {0.1239, thing}, {0.120232, like}, {0.118959, think}}
```

Now we can construct a table of topics. Note that because of the convergence issues of NNMF it is a good idea to run the computations several times with different initializations. As rule the more prominent topics would appear in all experiments.

```
topicsTbl =
  Table[
   (
    t = BasisVectorInterpretation[H〚ind〛, 12, terms〚pos〛];
    TableForm[{NumberForm[#〚1〛 / t〚1, 1〛, {4, 3}], #〚2〛} & /@ t]
   ), {ind, 1, k}];
```

```
Magnify[#, 0.68] &@Grid[Partition[
   ColumnForm /@ Transpose[{Style[#, Red] & /@ Range[k], topicsTbl}],
   5], Dividers → All, Alignment → Left]
```

1		2		3		4		5	
1.000	go	1.000	kid	1.000	soundbit	1.000	right	1.000	think
0.927	war	0.647	know	0.953	npr	0.778	yeah	0.969	raz
0.898	militari	0.485	parent	0.502	music	0.731	laughter	0.738	actual
0.842	soldier	0.419	hansen	0.449	year	0.451	know	0.729	hansen
0.792	command	0.326	school	0.421	say	0.381	go	0.643	realli
0.676	afghanistan	0.305	conan	0.383	host	0.276	soundbit	0.501	kind
0.644	iraq	0.292	say	0.309	new	0.251	applaus	0.398	sort
0.640	tim	0.286	children	0.303	news	0.245	like	0.391	thing
0.624	forc	0.267	stori	0.288	provid	0.240	don	0.364	mean
0.569	armi	0.230	thing	0.251	copyright	0.220	let	0.335	play
0.512	combat	0.223	like	0.248	transcript	0.210	thank	0.327	like
0.429	troop	0.221	think	0.218	band	0.208	conan	0.289	peopl

6		7		8		9		10	
1.000	say	1.000	gun	1.000	know	1.000	simon	1.000	record
0.933	like	0.725	block	0.965	islam	0.142	song	0.374	label
0.535	npr	0.241	shot	0.418	peac	0.123	yeah	0.278	band
0.391	jay	0.199	shoot	0.287	kind	0.113	soundbit	0.274	music
0.380	case	0.194	peopl	0.251	conan	0.110	scott	0.198	album
0.373	year	0.183	like	0.238	world	0.106	thank	0.180	hansen
0.355	report	0.179	think	0.218	today	0.101	hansen	0.154	make
0.315	state	0.176	foster	0.211	peopl	0.068	npr	0.145	releas
0.297	law	0.149	got	0.198	call	0.063	just	0.137	song
0.284	judg	0.146	riddl	0.194	come	0.061	edit	0.122	musician
0.266	court	0.139	year	0.183	got	0.055	work	0.118	year
0.251	militari	0.134	kill	0.182	think	0.049	year	0.117	big

11		12		13		14		15	
1.000	know	1.000	new	1.000	obama	1.000	conan	1.000	plant
0.660	like	0.607	orlean	0.826	think	0.533	know	0.930	nuclear
0.626	just	0.425	citi	0.811	campaign	0.214	warren	0.755	edg
0.185	realli	0.365	music	0.763	mccain	0.177	yeah	0.667	davi
0.165	kind	0.344	conan	0.604	senat	0.169	dog	0.637	worker
0.127	mean	0.242	york	0.546	elect	0.123	thank	0.464	water
0.124	yeah	0.229	play	0.524	conan	0.113	yes	0.463	radiat
0.109	think	0.228	jazz	0.442	presid	0.111	hallelujah	0.408	go
0.101	don	0.227	musician	0.409	republican	0.097	young	0.406	just
0.093	band	0.214	band	0.402	polit	0.093	nation	0.396	fuel
0.091	play	0.150	just	0.395	democrat	0.093	jone	0.384	grass
0.088	want	0.139	like	0.380	state	0.090	neal	0.382	flatow

16		17		18		19		20	
1.000	lyden	1.000	conan	1.000	like	1.000	great	1.000	languag
0.949	song	0.911	black	0.384	kind	0.869	song	0.876	foreign
0.536	love	0.895	patient	0.309	realli	0.712	know	0.708	music
0.529	jacki	0.604	peopl	0.249	stewart	0.662	gross	0.622	spoken
0.504	record	0.570	race	0.228	think	0.470	record	0.443	soundbit
0.450	time	0.561	reed	0.225	film	0.433	wainwright	0.296	say
0.426	did	0.506	white	0.212	album	0.369	kind	0.272	eyr
0.423	bobbi	0.479	doctor	0.210	smith	0.363	music	0.203	african
0.423	soundbit	0.447	gordon	0.203	black	0.346	realli	0.202	unidentifi
0.363	just	0.389	american	0.195	soundbit	0.315	jazz	0.202	npr
0.303	play	0.338	care	0.189	gross	0.276	yeah	0.196	sing
0.298	said	0.334	talk	0.163	sort	0.268	conan	0.146	mexico

21		22		23		24		25	
1.000	gross	1.000	sister	1.000	peopl	1.000	conan	1.000	song
0.935	know	0.807	simon	0.536	aid	0.721	raitt	0.630	block
0.677	mcdonald	0.799	think	0.491	know	0.669	soundbit	0.366	raz
0.579	bess	0.685	kate	0.424	say	0.588	play	0.321	soundbit
0.565	hand	0.507	tell	0.409	talk	0.406	thank	0.251	know
0.459	monk	0.498	know	0.395	john	0.328	just	0.230	yeah
0.387	time	0.436	want	0.355	norri	0.315	time	0.197	like
0.290	sing	0.419	realli	0.335	think	0.313	laughter	0.175	thing
0.243	sort	0.387	mother	0.316	want	0.277	go	0.168	kind
0.226	like	0.348	did	0.294	conan	0.260	don	0.142	realli
0.225	work	0.310	new	0.261	hear	0.251	guitar	0.136	write
0.221	thing	0.310	time	0.244	michel	0.246	hand	0.136	pretti

26		27		28		29		30	
1.000	blue	1.000	song	1.000	food	1.000	know	1.000	like
0.887	note	0.715	jone	0.713	conan	0.432	flatow	0.427	ben
0.527	record	0.564	stewart	0.327	lot	0.270	think	0.394	band
0.513	conan	0.539	album	0.323	eat	0.235	fact	0.342	harper
0.491	jazz	0.527	record	0.297	just	0.235	univers	0.247	song
0.259	said	0.423	soundbit	0.267	famili	0.206	mean	0.217	new
0.206	time	0.385	think	0.257	hunger	0.155	low	0.208	metal
0.205	artist	0.320	pop	0.238	make	0.147	say	0.208	stewart
0.196	year	0.303	pesca	0.238	thing	0.143	thing	0.205	fold
0.177	bruce	0.299	sort	0.237	children	0.142	go	0.201	movi
0.176	think	0.266	new	0.235	need	0.132	make	0.188	record
0.174	michael	0.265	green	0.235	dog	0.131	look	0.183	make

31		32		33		34		35	
1.000	wait	1.000	sing	1.000	gross	1.000	ari	1.000	say
0.904	gross	0.639	song	0.777	know	0.373	conan	0.995	simon
0.843	like	0.300	soundbit	0.591	happen	0.264	thank	0.967	watson
0.710	know	0.203	love	0.565	jay	0.228	like	0.735	play
0.284	yeah	0.153	rock	0.513	peopl	0.210	soundbit	0.703	like
0.262	pop	0.132	music	0.365	said	0.203	talk	0.584	good
0.218	new	0.131	honey	0.288	life	0.193	laughter	0.571	guitar
0.195	stew	0.131	sweet	0.276	stori	0.189	india	0.548	sing
0.181	spanish	0.118	singer	0.274	think	0.170	love	0.526	said
0.161	russel	0.106	album	0.267	day	0.161	chorus	0.502	don
0.157	bad	0.101	come	0.261	book	0.153	just	0.454	just
0.151	tom	0.095	voic	0.246	did	0.145	hope	0.450	time

36		37		38		39		40	
1.000	woodi	1.000	say	1.000	david	1.000	martin	1.000	conan
0.819	guthri	0.702	peopl	0.980	promis	0.150	think	0.512	vega
0.549	know	0.618	govern	0.630	love	0.121	know	0.254	korea
0.370	place	0.557	like	0.491	song	0.085	just	0.245	thank
0.354	land	0.468	think	0.461	fall	0.084	album	0.230	north
0.291	honey	0.435	npr	0.313	hal	0.079	don	0.216	don
0.288	conan	0.372	want	0.292	don	0.070	want	0.213	women
0.283	children	0.367	make	0.285	write	0.068	say	0.209	woman
0.281	peopl	0.366	work	0.249	way	0.066	thing	0.206	peopl
0.271	jeff	0.337	way	0.247	think	0.066	peopl	0.192	korean
0.266	sweet	0.325	presid	0.244	wrote	0.066	realli	0.191	war
0.252	rock	0.286	go	0.227	time	0.064	time	0.190	think

41		42		43		44		45	
1.000	song	1.000	william	1.000	lynn	1.000	gross	1.000	flatow
0.652	old	0.162	hank	0.387	gross	0.310	play	0.867	song
0.454	gross	0.109	soundbit	0.365	think	0.289	song	0.616	express
0.417	know	0.092	mother	0.306	boston	0.235	did	0.569	music
0.415	wainwright	0.091	song	0.214	say	0.235	record	0.531	wainwright
0.243	got	0.091	year	0.167	daughter	0.177	start	0.458	contrera
0.240	low	0.090	know	0.155	miner	0.162	like	0.348	peopl
0.236	year	0.080	white	0.153	coal	0.158	laughter	0.293	soundbit
0.235	soundbit	0.078	best	0.151	don	0.144	yeah	0.289	scienc
0.210	day	0.068	say	0.135	peopl	0.118	terri	0.247	kind
0.209	record	0.058	day	0.131	tribut	0.110	didn	0.236	yeah
0.182	pool	0.056	time	0.123	smith	0.110	piano	0.231	face

46		47		48		49		50	
1.000	banjo	1.000	book	1.000	music	1.000	conan	1.000	cox
0.778	play	0.651	read	0.230	soundbit	0.563	jazz	0.324	know
0.513	band	0.299	hard	0.198	compos	0.535	musician	0.291	music
0.474	bluegrass	0.266	like	0.143	know	0.443	think	0.267	say
0.346	gross	0.230	think	0.142	play	0.410	make	0.253	go
0.320	tune	0.219	pearl	0.120	jazz	0.400	work	0.225	talk
0.290	soundbit	0.208	neari	0.100	classic	0.392	generat	0.216	toni
0.285	earl	0.205	inskeep	0.087	piec	0.346	talk	0.195	green
0.282	yeah	0.201	simon	0.079	listen	0.338	care	0.193	want
0.261	music	0.175	polit	0.078	think	0.337	busi	0.190	got
0.249	monro	0.155	applaus	0.078	like	0.331	thank	0.189	thing
0.243	del	0.148	realli	0.077	hear	0.313	year	0.180	time

51		52		53		54		55	
1.000	glass	1.000	dream	1.000	play	1.000	johnson	1.000	cornish
0.894	music	0.606	song	0.843	music	0.231	kennedi	0.342	npr
0.840	soundbit	0.381	dog	0.600	soundbit	0.205	davi	0.254	say
0.500	day	0.370	parton	0.552	guitar	0.162	power	0.248	audi
0.482	record	0.354	just	0.445	sound	0.158	presid	0.161	year
0.326	known	0.349	hard	0.439	string	0.157	robert	0.138	news
0.271	song	0.326	work	0.370	instrument	0.150	say	0.136	thing
0.258	laughter	0.258	want	0.269	hansen	0.132	know	0.133	host
0.243	cash	0.242	diamond	0.252	hear	0.105	time	0.119	block
0.234	gross	0.239	make	0.233	note	0.103	right	0.111	feel
0.205	life	0.201	don	0.223	siegel	0.096	year	0.104	just
0.193	way	0.195	thank	0.213	just	0.086	civil	0.103	scott
56		57		58		59		60	
1.000	ray	1.000	countri	1.000	coffe	1.000	jone	1.000	music
0.437	yeah	0.913	doe	0.275	like	0.545	jame	0.937	billi
0.410	conan	0.738	song	0.217	charl	0.399	gross	0.899	ellington
0.308	sing	0.695	cash	0.113	say	0.311	play	0.672	conan
0.304	just	0.591	gross	0.112	tree	0.303	thing	0.528	duke
0.250	play	0.350	john	0.099	world	0.298	band	0.462	work
0.190	ami	0.259	list	0.097	farm	0.257	lincoln	0.440	life
0.175	girl	0.243	did	0.089	big	0.236	record	0.392	talk
0.153	music	0.227	know	0.088	npr	0.232	dont	0.365	jazz
0.148	thank	0.211	good	0.086	trade	0.222	time	0.317	lush
0.144	emili	0.203	like	0.082	head	0.199	look	0.236	did
0.137	like	0.188	year	0.082	soundbit	0.192	sing	0.218	time

7. Statistical thesaurus

We can also find a statistical thesaurus that fits the body of the documents. For example, the words “pollution”, “fossil”, ”greenhouse”, “gasoline” are found together in the NPR transcripts.

The statistical thesaurus for the *i*-th term can be found by taking, say, 20 nearest neighbors of the *i*-th column from the right matrix factor in a SVD or NNMF (using the Euclidean distance).

Computation

In order to find a statistical thesaurus for the collection of documents represented with M we normalize the product $W\,H$ in such a way that the norms of the columns of W are 1. (The alternative normalization making the norms of the rows of H to be 1 uses a different point of view of what is a statistical thesaurus.)

```
{W, H} = NormalizeMatrixProduct[W, H];
```

Instead using clustering we are going to demonstrate the thesaurus finding using nearest neighbors. So, we pre-compute the following nearest neighbors function:

```
HNF = Nearest[Range[Dimensions[H][[2]]],
  DistanceFunction → (Norm[H[[All, #1]] - H[[All, #2]]] &)]
```

```
NearestFunction[{5739, 1}, <>]
```

Next we define a function that would find the thesaurus entry for a given word:

```
Clear[StatThesaurus];
StatThesaurus[word_String, n_Integer: 20] :=
  Block[{sword, tpos, inds},
   sword = word /. stemmingRules;
   tpos = Position[terms[[pos]], sword];
   If[Length[tpos] == 0, {},
    inds = HNF[tpos[[1, 1]], n];
    terms[[pos]][[inds]]
   ]
  ];
```

Here is a table of invoking `StatThesaurus` over a set of words:

```
Magnify[#, 0.7] &@
 Grid[Prepend[Map[{#, StatThesaurus[#, 15]} &, {"senate", "obama",
     "war", "food", "fbi", "singer", "jazz", "school", "homeland",
     "marathon"}], Style[#, Blue, FontFamily → "Times"] & /@
    {"word", "statistical thesaurus"}], Dividers → All,
  Alignment → Left, Spacings → {Automatic, 0.75}]
```

word	statistical thesaurus
senate	{senat, elect, republican, mccain, democrat, campaign, vote, barack, polit, presid, state, clinton, parti, ken, candid}
obama	{obama, campaign, mccain, senat, elect, republican, democrat, presid, polit, vote, state, barack, race, clinton, parti}
war	{war, command, soldier, forc, afghanistan, iraq, armi, militari, tim, combat, general, troop, ground, chief, colonel}
food	{food, eat, hunger, stamp, pam, program, meal, famili, shore, struggl, buy, million, hungri, get, need}
fbi	{fbi, suspici, bradi, tsarnaev, terror, templ, arrest, ir, 9/11, strike, file, surveil, chicken, incid, suspect}
singer	{singer, voic, gonna, babi, group, sweet, rock, spirit, honey, soul, unintellig, stranger, danc, heard, god}
jazz	{jazz, musician, artist, busi, michael, listen, art, today, classic, york, pianist, piano, bruce, cours, alfr}
school	{school, parent, teacher, boy, mom, colleg, studi, famili, bulli, program, adult, student, help, get, joe}
homeland	{homeland, tribal, cathedr, infant, personnel, samba, rapid, stanford, salsa, temporari, techno, meantim, evolut, bomber, isra}
marathon	{marathon, fist, pill, tonk, honki, suspect, surveil, stripe, tsarnaev, memoir, bomb, dress, mrs, bradi, flag}

8. Topic initialization with thesaurus entries

From the explanations about the NNMF initialization and thesaurus computation we note that we can use the thesaurus entries to initialize the columns of *W* in (1).

First we initialize the *W* as above (using smaller number of topics `k`).

In[1118]:=
```
{k, p} = {20, 12};
{m, n} = Dimensions[M1];
M1 = Transpose[M1];
M1 = Map[# &, M1];
H = ConstantArray[0, {k, n}];
W = Table[Total[RandomSample[M1, p]], {k}];
Do[
 W⟦i⟧ = W⟦i⟧ / Norm[W⟦i⟧];
 , {i, 1, Length[W]}]
W = Transpose[W];
M1 = SparseArray[M1];
M1 = Transpose[M1];
```

We use the thesaurus query function `StatThesaurus` to derive candidate topics.

```
candidateTopics =
  Map[StatThesaurus[#, 15] &, {"senate", "obama", "war", "food",
    "fbi", "singer", "jazz", "school", "homeland", "marathon"}];
```

Next we convert the terms in the topics into indices in the list of selected terms. (See above how `pos` was computed.)

```
candidateTopicsInds =
  Map[Position[terms⟦pos⟧, #]⟦1, 1⟧ &, candidateTopics, {-1}];
```

Similar to the initialization above for each topic candidate *t* we sum the columns of *M* corresponding to the terms in *t* and assign that sum to a column of *W*.

In[1128]:=
```
M1 = Transpose[M1];
W = Transpose[W];
Wcols = Map[Total[M1⟦#⟧, 1] &, candidateTopicsInds];
Do[W⟦i⟧ = Wcols⟦i⟧, {i, Length[Wcols]}]
Do[
 W⟦i⟧ = W⟦i⟧ / Norm[W⟦i⟧];
 , {i, 1, Length[W]}]
W = Transpose[W];
M1 = Transpose[M1];
```

Perform six NNMF iterations.

```
In[1135]:= W = SparseArray[W];
H = SparseArray[H];
{W, H} = GDCLSGlobal[M1, W, H, "MaxSteps" → 6,
    "PrintProfilingInfo" → True]; // AbsoluteTiming
```

```
1 {159.675066, Null}
2 {155.599868, Null}
3 {160.776146, Null}
4 {162.018519, Null}
5 {160.181757, Null}
6 {163.227890, Null}
Out[1137]= {666.392510, Null}
```

Normalize (the norms of the rows of *H* are 1).

```
In[1138]:= {W, H} = RightNormalizeMatrixProduct[W, H];
{W, H} = Normal /@ {W, H};
```

And here is the new table of topics.

```
In[1140]:= topicsTbl =
  Table[
   (
    t = BasisVectorInterpretation[H⟦ind⟧, 16, terms⟦pos⟧];
    TableForm[{NumberForm[#⟦1⟧ / t⟦1, 1⟧, {4, 3}], #⟦2⟧} & /@ t]
   ), {ind, 1, k}];
```

```
In[1141]:= Magnify[#, 0.68] &@Grid[Partition[
   ColumnForm /@ Transpose[{Style[#, Red] & /@ Range[k], topicsTbl}],
   5], Dividers → All, Alignment → Left]
```

1		2		3		4		5	
1.000	conan	1.000	peopl	1.000	conan	1.000	conan	1.000	say
0.321	senat	0.954	obama	0.954	war	0.380	know	0.823	npr
0.316	state	0.929	think	0.595	militari	0.334	peopl	0.660	report
0.309	go	0.641	presid	0.581	go	0.295	thank	0.606	mall
0.305	vote	0.614	know	0.549	command	0.283	talk	0.509	peopl
0.300	republican	0.581	black	0.526	iraq	0.239	just	0.491	polic
0.285	democrat	0.495	american	0.456	forc	0.222	food	0.369	year
0.279	elect	0.486	white	0.377	afghanistan	0.221	don	0.322	news
0.242	time	0.457	race	0.345	soldier	0.189	go	0.317	case
0.217	mccain	0.451	polit	0.327	general	0.185	make	0.313	said
0.215	parti	0.440	talk	0.305	tim	0.179	need	0.301	secur
0.212	right	0.427	campaign	0.299	year	0.176	lot	0.301	boston
0.211	think	0.404	say	0.297	think	0.165	yeah	0.296	fbi
0.199	question	0.300	mccain	0.291	troop	0.161	work	0.288	host
0.178	ken	0.293	barack	0.276	armi	0.142	thing	0.283	america
0.177	thank	0.289	just	0.275	talk	0.137	laughter	0.280	block

6		7		8		9		10	
1.000	sing	1.000	music	1.000	martin	1.000	music	1.000	lynn
0.605	song	0.476	soundbit	0.555	kid	0.752	flatow	0.715	gross
0.398	soundbit	0.423	play	0.505	think	0.438	languag	0.244	know
0.226	love	0.264	jazz	0.467	parent	0.406	contrera	0.156	miner

Out[1141]=

0.210	rock	0.236	conan	0.450	school	0.377	peopl	0.155	did
0.165	music	0.214	record	0.411	say	0.373	soundbit	0.155	coal
0.159	sweet	0.191	new	0.404	just	0.329	foreign	0.155	daughter
0.157	honey	0.182	musician	0.340	famili	0.318	martin	0.151	yeah
0.145	singer	0.160	npr	0.323	year	0.315	npr	0.130	think
0.139	npr	0.129	band	0.315	want	0.287	latin	0.127	tribut
0.116	just	0.129	hear	0.305	children	0.232	say	0.124	don
0.116	album	0.127	sound	0.290	thing	0.229	scienc	0.123	home
0.115	say	0.118	listen	0.285	npr	0.203	express	0.121	didn
0.115	block	0.117	just	0.271	time	0.190	felix	0.108	white
0.114	voic	0.110	year	0.263	stori	0.179	host	0.107	got
0.112	gonna	0.109	compos	0.249	said	0.148	like	0.102	like

11		12		13		14		15	
1.000	gross	1.000	like	1.000	song	1.000	think	1.000	gross
0.456	day	0.932	doe	0.341	soundbit	0.925	martin	0.563	song
0.420	sing	0.301	just	0.298	call	0.703	idea	0.538	know
0.241	burnett	0.290	soundbit	0.245	raz	0.594	peopl	0.395	record
0.227	movi	0.288	song	0.165	album	0.476	thing	0.269	did
0.221	did	0.286	yeah	0.157	record	0.437	kind	0.221	time
0.203	stephen	0.257	countri	0.153	new	0.418	right	0.199	just
0.182	music	0.239	john	0.136	gross	0.402	go	0.188	woodi
0.181	film	0.207	laughter	0.133	yeah	0.378	don	0.180	end
0.162	mcdonald	0.205	gross	0.129	sing	0.367	write	0.177	said
0.158	want	0.182	littl	0.122	write	0.363	sort	0.151	like
0.157	write	0.181	good	0.116	wainwright	0.352	mean	0.147	write
0.142	lyric	0.158	band	0.115	band	0.351	make	0.146	play
0.139	work	0.147	sort	0.113	love	0.331	just	0.143	love
0.135	bess	0.138	don	0.106	laughter	0.324	way	0.141	got
0.131	play	0.137	think	0.105	got	0.324	david	0.141	guthri

16		17		18		19		20	
1.000	know	1.000	know	1.000	women	1.000	simon	1.000	music
0.521	like	0.726	record	0.653	martin	0.481	know	0.835	lunden
0.308	realli	0.623	conan	0.326	think	0.423	right	0.830	soundbit
0.307	just	0.477	blue	0.308	say	0.390	say	0.734	say
0.258	gross	0.438	note	0.289	know	0.347	like	0.666	npr
0.256	yeah	0.409	say	0.253	woman	0.332	npr	0.467	martin
0.214	think	0.338	song	0.252	just	0.276	play	0.335	fight
0.188	kind	0.323	year	0.224	want	0.241	johnson	0.333	year
0.164	mean	0.258	warren	0.217	peopl	0.241	just	0.332	new
0.127	play	0.247	make	0.192	like	0.237	come	0.327	realli
0.124	thing	0.242	mean	0.166	don	0.229	year	0.270	right
0.106	laughter	0.234	peopl	0.160	honey	0.218	peopl	0.268	news
0.105	don	0.232	said	0.158	sweet	0.205	band	0.247	davi
0.100	sort	0.228	think	0.147	men	0.202	new	0.244	just
0.096	want	0.224	work	0.138	rock	0.200	bring	0.243	peopl
0.095	martin	0.222	artist	0.135	children	0.195	read	0.243	sing

References

[1] Anton Antonov, Implementation of document-term matrix construction and re-weighting functions in *Mathematica*, source code at GitHub, https://github.com/antononcube/MathematicaForPrediction, package DocumentTermMatrixConstruction.m, (2013).

[2] Anton Antonov, Implementation of non-negative matrix factorization in *Mathematica*, source code at GutHub, https://github.com/antononcube/MathematicaForPrediction, pack-

age NonNegativeMatrixFactorization.m, (2013).

[3] Stop words, Wikipedia entry, http://en.wikipedia.org/wiki/Stop_words .

[4] Stemming, Wikipedia entry, http://en.wikipedia.org/wiki/Stemming .

[5] Michael Berry, Murray Browne, “Understanding Search Engines: Mathematical Modeling and Text Retrieval”. SIAM, 2005.
http://books.google.com/books/about/Understanding_Search_Engines.html?id=J21ooXWVdzkC
http://www.amazon.com/Understanding-Search-Engines-Mathematical-Environments/dp/0898715814

[6] Russell Albright, et al., Algorithms, Initializations, and Convergence for the Nonnegative Matrix Factorization, http://meyer.math.ncsu.edu/meyer/ps_files/nmfinitalgconv.pdf

[7] Michael Berry, et al., Algorithms and Applications for Approximate Nonnegative Matrix Factorization, preprint Elsevier Preprint (2006), http://users.wfu.edu/plemmons/papers/BBLPP-rev.pdf.