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 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`).

In[1]:=
```
Get["~/MathFiles/MathematicaForPrediction
    Documentation/NPRTranscripts-documents.m"];
```

In[2]:=
```
Grid[List /@ Map[StringTake[#, {1, 100}] &,
   documents[[RandomInteger[{1, 400}, 6]]]],
 Alignment → Left, Dividers → All]
```

Out[2]=

ED GORDON, host:Pianist and composer Ramsey Lewis has a long history in swing music and   popular
(Soundbite of music)JACKIE LYDEN, host:The angelic tune you're hearing might remind you of notes fro
MADELEINE BRAND, host:This is DAY TO DAY. I'm Madeleine Brand.The hard rock group System of a Down
(Soundbite of guitar music)SCOTT SIMON, host:If you flew in a dirigible over Austin in the dark of n
ROBERT SIEGEL, host:In the early 1970s, singer-songwriter Judee Sill seemed headed for big   thin
(Soundbite of applause; music)Ms. CAROLE KING (Singer-Songwriter): (Singing) Welcome to my living

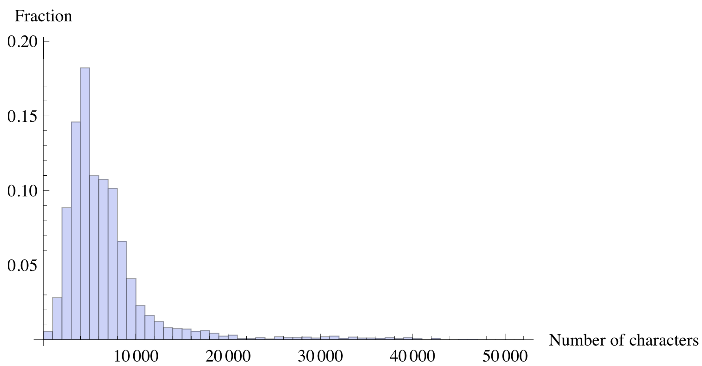
We have $\approx$ 5000 documents:

In[3]:=
```
documents // Length
```

Out[3]= 5123

Here is a histogram of their string lengths:

In[4]:=
```
Histogram[StringLength /@ documents, Automatic, "Probability",
 AxesLabel → {"Number of characters", "Fraction"}]
```

Out[4]=


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`):

In[127]:=
```
stopWords = ReadList["~/MathFiles/DataMining/stop_words", Word];
Magnify[stopWords, 0.7]
```

Out[128]=
```
{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.

```
In[136]:= tblData = newStopWords; nCols = 3;
Magnify[#, 0.7] &@Grid[Prepend[Flatten /@ Partition[tblData, nCols],
    Style[#, Blue, FontFamily → "Times"] & /@
     Flatten[Table[{"term", "%"}, {nCols}]]],
   Dividers → {Flatten@Append[Table[{True, False}, {nCols}], True],
     {True, True, False}}, Alignment → Left]
```

Out[136]=

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.

```
In[130]:= wordsTally = Tally[
    Flatten[Map[Complement[Union[Select[StringSplit[ToLowerCase[#],
           {{Whitespace, "\n", "  ", ".", ",", "!", "?", ";",
             ":", "-", "\"", "'", "(", ")", "“", "`"}}],
          StringLength[#] >= 2 &]], stopWords] &, documents]]];
```

```
In[131]:= wordsTally // Length
```

Out[131]= 67 092

```
In[132]:= wordsTally[[1 ;; 45, 1]]
```

Out[132]= {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}

```
In[133]:= newStopWords =
   SortBy[Select[wordsTally, #[[2]] > 0.6 Length[documents] &], -#[[2]] &];
```

In[134]:=
```
newStopWords⟦All, 2⟧ = N[newStopWords⟦All, 2⟧ / Length[documents]];
newStopWords
```

Out[135]=
```
{{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 the word “terms” to mean “stemmed words”.

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

In[14]:=
```
(*inds=Flatten[Position[transcriptsPerTerm,
    t_/;(0.24≤(t/Dimensions[tranMat]⟦1⟧)<=0.25)]];
tblData=Map[{#,Cases[List@@@stemmingRules⟦1⟧,{x_,#}:>x,∞]}&,
  tranTerms⟦inds⟧];tblData=Flatten/@tblData;
tblData=Prepend[tblData,
  Style[#,Blue,FontFamily→"Times"]&/@{"term","words"}];
Magnify[#,0.5]&@Grid[If[Length[#]>5,Take[#,7],#]&/@tblData,
  Alignment→Left,Dividers→{{False,True},{True,True}}]*)
```

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

In[15]:=
```
WordData[#, "PorterStem"] & /@ {"able", "schooling", "critical"}
```

Out[15]=
```
{abl, school, critic}
```

Using an external stemmer

We can also use an external stemmer, such as the stemmer called snowball (see http://s-nowball.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

In[137]:=
```
allWords = wordsTally[[All, 1]];
```

In[138]:=
```
wordsToStem = Complement[
   Select[allWords, StringMatchQ[#, LetterCharacter ..] &],
   Join[stopWords, newStopWords]];
wordsToStem // Length
```

Out[139]= 63 241

In[140]:=
```
Export["~/MathFiles/text_words.txt", wordsToStem]
```

Out[140]= ~/MathFiles/text_words.txt

In[141]:=
```
Run["~/snowball/libstemmer_c/stemwords
   -l english -i ~/MathFiles/text_words.txt
   -o ~/MathFiles/text_words_stemmed.txt"]
```

Out[141]= 0

In[142]:=
```
stemmedWords =
  StringSplit[Import["~/MathFiles/text_words_stemmed.txt"]];
stemmedWords // Length
```

Out[143]= 63 241

In[144]:=
```
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 be 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].

In[145]:=
```
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.

In[146]:=
```
AbsoluteTiming[
 {F, terms} = DocumentTermMatrix[ToLowerCase /@ documents,
    {stemmingRules, Join[stopWords, newStopWords]}];
]
```

Out[146]=
```
{67.144092, Null}
```

In[147]:=
```
F
```

Out[147]=
```
SparseArray[<1 403 565>, {5123, 45 627}]
```

In[148]:=
```
terms // Length
```

Out[148]=
```
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.

In[149]:=
```
nonWords = Select[terms, ! StringMatchQ[#, LetterCharacter ..] &];
nonWords // Length
RandomSample[nonWords, 12]
```

Out[150]=
```
3674
```

Out[151]=
```
{baby…ydstie, 12months, �sentimental, running…hansen, spoke…ms,
 bar/sperm, really…, cracks…ms, sin…ms, 15s, band…, …turning}
```

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

```
In[152]:= wordBag = ToBagOfWords[
    ToLowerCase@documents⟦1⟧, {stemmingRules, stopWords}];
SortBy[Tally[wordBag], -#⟦2⟧ &]⟦1 ;; 12⟧

Out[153]= {{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.

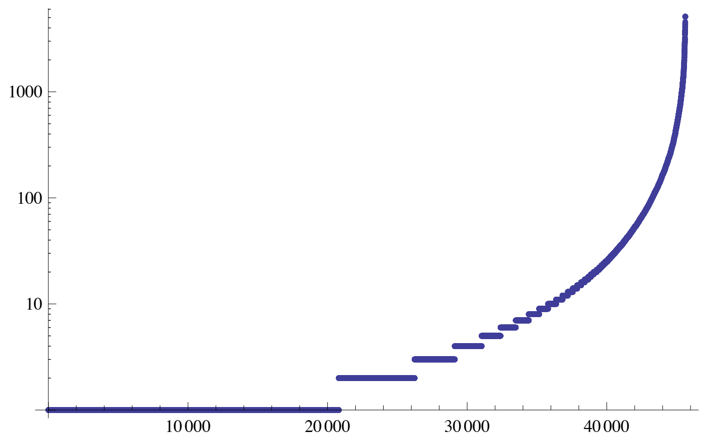
```
In[154]:= documentsPerTerm = Total /@ Transpose[Clip[F, {0, 1}]];
TableForm[{{Min, Max, Mean, Median, StandardDeviation},
  Through[{Min, Max, N[Mean[#]] &, Median,
     N[StandardDeviation[#]] &}[documentsPerTerm]]}]

Out[155]//TableForm=
```

Min	Max	Mean	Median	StandardDeviation
1	5123	30.7617	2	172.999

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

In[156]:=
```
ListLogPlot[Sort[documentsPerTerm], PlotRange → All]
```

Out[156]=


Terms per document

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

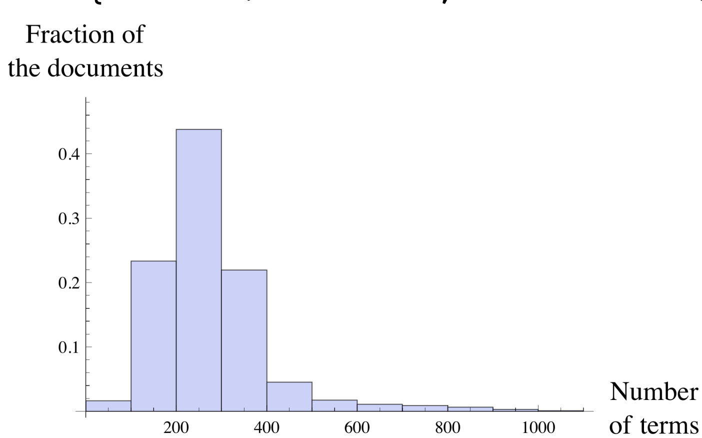
In[157]:=
```
termsPerDocument = Total /@ Clip[F, {0, 1}];
TableForm[{{Min, Max, Mean, Median, StandardDeviation},
  Through[{Min, Max, N[Mean[#]] &, Median,
     N[StandardDeviation[#]] &}[termsPerDocument]]}]
```

Out[158]//TableForm=

Min	Max	Mean	Median	StandardDeviation
6	1117	273.973	251	125.379

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

In[159]:=
```
Histogram[termsPerDocument, {0, 1100, 100},
 "Probability", AxesLabel → (Style[#, FontSize → 14] & /@
    {"Number\nof terms", "Fraction of\nthe documents"})]
```

Out[159]=


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.

Out[40]=

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

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].

In[160]:=
```
AbsoluteTiming[
 M = WeightTerms[F, GlobalTermWeight["GFIDF", #1, #2] &, # &, # &]
]
```

Out[160]=
```
{1.193199, 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 interpreting 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 $M \in \mathbb{R}^{m \times n}$

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

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 order 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]:

In[268]:=
```
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.

In[269]:=
```
pos = Flatten[Position[documentsPerTerm, s_?NumberQ /; s ≥ 25]];
pos // Length
```

Out[270]=
```
5739
```

In[271]:=
```
M1 = M[[All, pos]]
```

Out[271]=
```
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 an *i*-th column of *W*. (We do this `k` times.) This procedure is done faster if we transpose the matrices *M* and *W*.

In[272]:=
```
{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 latter 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.

In[282]:=
```
W = SparseArray[W];
H = SparseArray[H];
{W, H} = GDCLSGlobal[M1, W, H, "MaxSteps" → 6,
    "PrintProfilingInfo" → True]; // AbsoluteTiming
```

```
1 {153.249095, Null}
2 {158.384875, Null}
3 {162.077230, Null}
4 {157.462106, Null}
5 {154.771453, Null}
6 {162.025170, Null}
Out[284]= {657.219713, 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]:

```
In[285]:= {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.)

```
In[286]:= {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.

```
In[287]:= BasisVectorInterpretation[H[[2]], 12, terms[[pos]]]
Out[287]= {{0.514672, music}, {0.389925, soundbit}, {0.219064, peopl},
  {0.214781, wainwright}, {0.21392, npr}, {0.14723, wind},
  {0.13333, say}, {0.120903, year}, {0.118399, man},
  {0.10836, recent}, {0.0948179, sing}, {0.0934409, pool}}
```

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.

```
In[288]:= topicsTbl =
  Table[
   (
    t = BasisVectorInterpretation[H[[ind]], 12, terms[[pos]]];
    TableForm[{NumberForm[#[[1]] / t[[1, 1]], {4, 3}], #[[2]]} & /@ t]
   ), {ind, 1, k}];
```

```
In[289]:= Magnify[#, 0.68] &@Grid[Partition[
    ColumnForm /@ Transpose[{Style[#, Red] & /@ Range[k], topicsTbl}],
    5], Dividers → All, Alignment → Left]
```

1		2		3		4		5	
1.000	say	1.000	music	1.000	know	1.000	say	1.000	low
0.866	npr	0.758	soundbit	0.905	like	0.829	presid	0.610	know
0.488	year	0.426	peopl	0.719	music	0.711	johnson	0.609	song
0.438	news	0.417	wainwright	0.610	just	0.669	secur	0.432	sort
0.424	unidentifi	0.416	npr	0.443	kind	0.662	mall	0.346	realli
0.415	go	0.286	wind	0.394	think	0.636	report	0.320	record
0.396	time	0.259	say	0.358	speak	0.507	go	0.293	gross
0.371	day	0.235	year	0.347	peopl	0.468	peopl	0.242	album
0.368	state	0.230	man	0.266	realli	0.465	right	0.225	old
0.342	conan	0.211	recent	0.256	mean	0.453	npr	0.215	yes
0.324	report	0.184	sing	0.225	lot	0.394	davi	0.208	new
0.310	like	0.182	pool	0.224	languag	0.389	polic	0.202	kind
6		7		8		9		10	
1.000	school	1.000	know	1.000	sing	1.000	gun	1.000	song
0.359	like	0.906	conan	0.750	record	0.683	block	0.917	sing
0.263	student	0.400	yeah	0.518	raz	0.420	say	0.672	soundbit
0.232	just	0.337	aid	0.491	time	0.309	state	0.339	music
0.226	high	0.273	sure	0.483	just	0.298	peopl	0.326	love
0.203	year	0.242	talk	0.420	love	0.252	right	0.253	like
0.167	say	0.238	peopl	0.387	say	0.239	npr	0.224	head
0.164	npr	0.229	john	0.293	moment	0.206	ban	0.193	npr
0.138	young	0.196	thank	0.274	sit	0.197	year	0.165	singer
0.132	colleg	0.150	go	0.268	mcdonald	0.193	weapon	0.130	album
0.130	teacher	0.143	blue	0.262	hansen	0.183	law	0.123	got
0.127	educ	0.142	note	0.242	blue	0.181	riddl	0.121	sound
11		12		13		14		15	
1.000	peopl	1.000	soundbit	1.000	blue	1.000	simon	1.000	music
0.769	say	0.978	alan	0.898	note	0.122	soundbit	0.683	billi
0.760	year	0.790	lyric	0.885	adam	0.111	song	0.631	soundbit
0.745	npr	0.763	song	0.666	record	0.105	thank	0.626	ellington
0.527	think	0.720	yeah	0.428	jazz	0.103	yeah	0.541	sing
0.522	book	0.701	sing	0.356	just	0.091	scott	0.454	berri
0.395	children	0.669	like	0.355	like	0.089	sing	0.439	jame
0.394	work	0.652	laughter	0.285	album	0.087	just	0.418	roll
0.391	food	0.590	write	0.277	music	0.077	music	0.417	littl
0.383	famili	0.429	yes	0.256	bruce	0.066	read	0.391	opera
0.362	age	0.409	norri	0.219	soundbit	0.062	yes	0.375	work
0.360	make	0.375	work	0.218	year	0.058	think	0.362	duke
16		17		18		19		20	
1.000	like	1.000	monk	1.000	band	1.000	parton	1.000	kid
0.834	know	0.778	peopl	0.679	like	0.653	conan	0.841	know
0.507	think	0.610	say	0.374	know	0.544	know	0.452	like
0.482	gross	0.390	rais	0.368	play	0.362	just	0.430	parent
0.474	mean	0.373	make	0.299	music	0.317	want	0.367	hansen
0.461	realli	0.360	money	0.286	soundbit	0.299	dolli	0.360	just
0.327	tim	0.338	book	0.280	yeah	0.229	year	0.297	thing
0.323	book	0.311	npr	0.243	just	0.228	got	0.256	kind
0.301	record	0.302	work	0.170	kind	0.226	dream	0.255	children
0.297	interview	0.298	state	0.156	album	0.221	love	0.241	think
0.288	just	0.283	way	0.153	rock	0.212	work	0.241	realli
0.235	did	0.279	year	0.136	call	0.202	lot	0.209	yeah

Out[289]=

21		22		23		24		25	
1.000	sing	1.000	conan	1.000	play	1.000	countri	1.000	guitar
0.696	honey	0.375	music	0.677	banjo	0.878	song	0.866	play
0.679	sweet	0.330	record	0.669	dave	0.768	doe	0.376	know
0.656	rock	0.296	cours	0.606	yeah	0.698	cash	0.372	soundbit
0.350	children	0.236	thank	0.583	ray	0.357	music	0.311	like
0.329	gonna	0.225	laughter	0.553	gross	0.329	soundbit	0.236	just
0.247	spirit	0.224	soundbit	0.507	earl	0.291	john	0.232	music
0.214	say	0.212	great	0.370	sing	0.281	list	0.217	watson
0.205	stranger	0.185	note	0.368	record	0.274	gross	0.213	sound
0.162	think	0.178	yes	0.353	just	0.196	year	0.212	record
0.134	robinson	0.178	talk	0.336	promis	0.179	great	0.211	blue
0.122	group	0.173	musician	0.327	hear	0.176	sing	0.166	good

26		27		28		29		30	
1.000	music	1.000	martin	1.000	think	1.000	dream	1.000	patient
0.256	compos	0.128	just	0.844	conan	0.839	song	0.725	npr
0.235	soundbit	0.119	think	0.609	peopl	0.741	like	0.697	care
0.220	play	0.102	peopl	0.462	know	0.384	sing	0.693	say
0.177	piec	0.093	know	0.451	talk	0.359	just	0.674	health
0.132	classic	0.090	don	0.433	thing	0.314	doe	0.602	use
0.110	work	0.086	laughter	0.425	lot	0.297	hard	0.601	block
0.102	realli	0.083	right	0.419	kind	0.293	wainwright	0.533	horn
0.099	write	0.081	want	0.417	don	0.262	new	0.532	just
0.098	hear	0.081	go	0.394	thank	0.231	don	0.522	doctor
0.097	siegel	0.076	like	0.340	want	0.224	realli	0.447	year
0.095	piano	0.074	say	0.285	have	0.209	know	0.411	provid

31		32		33		34		35	
1.000	block	1.000	nuclear	1.000	lewi	1.000	song	1.000	flatow
0.829	deal	0.919	plant	0.277	laughter	0.399	day	0.747	song
0.699	pesca	0.780	edg	0.173	yeah	0.366	sing	0.539	yeah
0.491	kim	0.772	davi	0.157	right	0.247	know	0.471	scienc
0.384	yeah	0.653	worker	0.145	go	0.238	way	0.381	soundbit
0.313	song	0.488	radiat	0.100	like	0.237	soundbit	0.316	laughter
0.300	soundbit	0.395	fuel	0.092	mean	0.154	record	0.254	sing
0.208	realli	0.385	power	0.087	news	0.153	write	0.253	sun
0.206	like	0.353	just	0.087	lee	0.151	love	0.248	come
0.204	think	0.349	water	0.081	jerri	0.131	just	0.205	right
0.196	did	0.335	happen	0.076	man	0.130	kind	0.204	element
0.177	come	0.326	japan	0.075	soundbit	0.124	think	0.200	like

36		37		38		39		40	
1.000	know	1.000	sing	1.000	like	1.000	conan	1.000	new
0.214	tell	0.774	gospel	0.482	just	0.589	hard	0.706	orlean
0.193	think	0.618	music	0.442	kind	0.457	sing	0.491	music
0.167	gross	0.571	jone	0.406	soundbit	0.356	like	0.299	npr
0.147	just	0.566	know	0.337	wertheim	0.307	thank	0.261	soundbit
0.141	stori	0.344	harri	0.244	film	0.292	ari	0.234	siegel
0.134	father	0.302	franklin	0.242	music	0.292	applaus	0.233	citi
0.123	like	0.293	song	0.231	moment	0.218	just	0.189	musician
0.116	mother	0.284	singer	0.226	think	0.183	laughter	0.183	say
0.112	realli	0.267	love	0.215	sound	0.168	don	0.157	york
0.111	did	0.217	god	0.212	nail	0.164	read	0.154	jazz
0.086	don	0.200	record	0.206	npr	0.160	yes	0.147	play

41		42		43		44		45	
1.000	gross	1.000	music	1.000	life	1.000	raz	1.000	raitt
0.138	did	0.623	soundbit	0.990	right	0.557	hansen	0.362	conan
0.123	just	0.421	glass	0.919	abort	0.514	music	0.295	soundbit
0.120	laughter	0.295	play	0.845	music	0.306	soundbit	0.203	bonni
0.113	lynn	0.202	coleman	0.515	davi	0.246	npr	0.189	music
0.110	like	0.194	sound	0.490	peopl	0.228	weekend	0.181	know
0.106	air	0.130	way	0.462	johnson	0.215	call	0.172	don
0.100	terri	0.127	listen	0.437	soundbit	0.200	song	0.146	hmm
0.094	fresh	0.117	record	0.429	think	0.196	mean	0.142	thank
0.082	kind	0.112	piano	0.379	state	0.185	new	0.124	blue
0.074	day	0.108	year	0.373	time	0.161	thing	0.120	song
0.069	love	0.105	hear	0.370	realli	0.160	year	0.120	go
46		47		48		49		50	
1.000	campaign	1.000	said	1.000	music	1.000	like	1.000	music
0.866	obama	0.472	song	0.587	jazz	0.864	song	0.581	rap
0.823	mccain	0.285	did	0.549	play	0.620	just	0.526	say
0.626	think	0.233	say	0.313	soundbit	0.518	soundbit	0.447	soundbit
0.386	senat	0.233	gross	0.273	musician	0.509	know	0.399	hip
0.336	sort	0.227	record	0.211	new	0.307	raz	0.385	studi
0.267	polit	0.212	love	0.148	record	0.300	yeah	0.385	hop
0.224	conan	0.210	know	0.128	monk	0.286	laughter	0.348	peopl
0.205	elect	0.204	got	0.124	listen	0.206	jay	0.337	npr
0.194	report	0.167	guy	0.123	trumpet	0.202	album	0.291	rapper
0.193	clinton	0.153	didn	0.121	compos	0.198	have	0.286	like
0.191	go	0.147	mccartney	0.117	jone	0.191	got	0.265	mcdonald
51		52		53		54		55	
1.000	song	1.000	conan	1.000	black	1.000	song	1.000	know
0.918	sing	0.540	laughter	0.733	race	0.943	sing	0.643	brain
0.802	tucker	0.532	soundbit	0.729	white	0.515	soundbit	0.586	rain
0.632	sister	0.493	like	0.719	obama	0.411	love	0.435	think
0.626	album	0.425	yeah	0.671	think	0.388	lynn	0.375	say
0.464	kate	0.397	hallelujah	0.640	like	0.363	like	0.346	mean
0.384	new	0.378	think	0.638	peopl	0.359	caus	0.314	kind
0.357	soundbit	0.360	idea	0.493	american	0.236	got	0.314	jazz
0.345	ken	0.345	realli	0.481	barack	0.231	yeah	0.294	realli
0.325	music	0.337	know	0.439	elect	0.204	album	0.273	peopl
0.313	love	0.262	dog	0.428	presid	0.202	man	0.230	don
0.254	npr	0.236	warren	0.420	franc	0.197	hansen	0.215	murder
56		57		58		59		60	
1.000	hansen	1.000	montagn	1.000	wait	1.000	know	1.000	republican
0.580	song	0.325	hoffman	0.406	know	0.373	flatow	0.833	senat
0.548	soundbit	0.316	rene	0.361	like	0.234	just	0.724	democrat
0.541	studio	0.242	inskeep	0.290	gross	0.234	go	0.524	parti
0.445	sing	0.232	npr	0.213	yeah	0.223	thing	0.519	vote
0.424	music	0.225	soundbit	0.156	don	0.205	univers	0.496	presid
0.386	record	0.202	music	0.154	new	0.204	right	0.469	think
0.359	play	0.153	morn	0.152	tom	0.201	yeah	0.405	elect
0.300	album	0.122	mile	0.128	bad	0.189	mean	0.347	hous
0.279	yeah	0.116	host	0.124	wife	0.128	make	0.344	know
0.244	love	0.112	steve	0.115	thing	0.124	littl	0.324	state
0.237	sound	0.109	just	0.109	want	0.122	don	0.300	polit

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, with which we make the norms of the rows of H to be 1, uses a different point of view of what a statistical thesaurus is.)

```
In[290]:= {W, H} = NormalizeMatrixProduct[W, H];
```

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

```
In[291]:= HNF = Nearest[Range[Dimensions[H]〚2〛],
            DistanceFunction → (Norm[H〚All, #1〛 - H〚All, #2〛] &)]

Out[291]= NearestFunction[{5739, 1}, <>]
```

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

```
In[292]:= 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:

In[347]:=
```
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}]
```

Out[347]=

word	statistical thesaurus
senate	{senat, democrat, republican, parti, vote, polit, elect, governor, voter, clinton, dean, conserv, ydstie, state, presid}
obama	{obama, mccain, campaign, polit, elect, barack, candid, senat, clinton, race, presid, palin, presidenti, report, white}
war	{war, afghanistan, kill, soldier, took, iraq, men, combat, job, journalist, later, danger, forc, camera, take}
food	{food, struggl, job, age, eat, million, money, help, administr, program, 000, hunger, buy, retir, bylin}
fbi	{fbi, suspici, guard, agent, terror, walter, van, terrorist, agenc, enforc, attack, reform, troop, homeland, chief}
singer	{singer, hit, heart, voic, beauti, babi, songwrit, soul, promis, produc, long, god, written, fall, norri}
jazz	{jazz, musician, trumpet, classic, piano, orchestra, listen, art, pianist, artist, hour, player, york, bass, whitehead}
school	{school, student, high, colleg, teacher, educ, young, boy, class, program, chicago, food, girl, communiti, close}
homeland	{homeland, alter, file, pentagon, minneapoli, analysi, 9/11, surveil, vulner, warn, maureen, agenda, incid, lobbi, assad}
marathon	{marathon, assassin, fighter, 9/11, staffer, airport, taliban, deploy, dalla, incid, alleg, staff, regim, citizen, assad}

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

```
In[318]:= candidateTopics =
           Map[StatThesaurus[#, 15] &, {"senate", "obama", "war", "food",
             "fbi", "singer", "jazz", "school", "homeland", "marathon"}];
          candidateTopics
Out[319]= {{senat, democrat, republican, parti, vote, polit, elect, governor,
            voter, clinton, dean, conserv, ydstie, state, presid},
           {obama, mccain, campaign, polit, elect, barack, candid, senat,
            clinton, race, presid, palin, presidenti, report, white},
           {war, afghanistan, kill, soldier, took, iraq, men, combat,
            job, journalist, later, danger, forc, camera, take},
           {food, struggl, job, age, eat, million, money, help,
            administr, program, 000, hunger, buy, retir, bylin},
           {fbi, suspici, guard, agent, terror, walter, van, terrorist,
            agenc, enforc, attack, reform, troop, homeland, chief},
           {singer, hit, heart, voic, beauti, babi, songwrit, soul,
            promis, produc, long, god, written, fall, norri},
           {jazz, musician, trumpet, classic, piano, orchestra, listen,
            art, pianist, artist, hour, player, york, bass, whitehead},
           {school, student, high, colleg, teacher, educ, young, boy,
            class, program, chicago, food, girl, communiti, close},
           {homeland, alter, file, pentagon, minneapoli, analysi, 9/11,
            surveil, vulner, warn, maureen, agenda, incid, lobbi, assad},
           {marathon, assassin, fighter, 9/11, staffer, airport, taliban,
            deploy, dalla, incid, alleg, staff, regim, citizen, assad}}
```

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

```
In[320]:= {k, p} = {30, 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];
```

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

```
In[330]:= 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[331]:= 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[338]:= W = SparseArray[W];
H = SparseArray[H];
{W, H} = GDCLSGlobal[M1, W, H, "MaxSteps" → 6,
    "PrintProfilingInfo" → True]; // AbsoluteTiming

1 {159.094831, Null}
2 {157.256328, Null}
3 {156.308874, Null}
4 {157.914375, Null}
5 {156.410203, Null}
6 {158.792878, Null}

Out[340]= {648.550510, Null}
```

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

```
In[341]:= {W, H} = RightNormalizeMatrixProduct[W, H];
{W, H} = Normal /@ {W, H};
```

And here is the new table of topics.

```
In[343]:= topicsTbl =
  Table[
   (
    t = BasisVectorInterpretation[H[[ind]], 16, terms[[pos]]];
    TableForm[{NumberForm[#[[1]] / t[[1, 1]], {4, 3}], #[[2]]} & /@ t]
   ), {ind, 1, k}];

In[344]:= Magnify[#, 0.68] &@Grid[Partition[
    ColumnForm /@ Transpose[{Style[#, Red] & /@ Range[k], topicsTbl}],
    5], Dividers → All, Alignment → Left]
```

1		2		3		4		5	
1.000	vote	1.000	obama	1.000	know	1.000	peopl	1.000	dog
0.891	republican	0.855	think	0.370	like	0.841	say	0.685	mall
0.769	elect	0.747	campaign	0.329	war	0.778	npr	0.474	report
0.730	democrat	0.661	mccain	0.251	just	0.701	make	0.394	polic
0.678	senat	0.396	peopl	0.226	tim	0.609	year	0.326	train
0.646	parti	0.363	presid	0.224	go	0.536	work	0.316	america
0.624	go	0.357	polit	0.217	think	0.469	food	0.307	walter
0.589	presid	0.293	senat	0.214	mean	0.451	money	0.264	say
0.579	state	0.274	barack	0.192	realli	0.410	go	0.244	suspici
0.502	npr	0.265	talk	0.164	talk	0.410	thing	0.235	secur
0.461	voter	0.256	race	0.155	book	0.400	just	0.233	law
0.390	poll	0.244	sort	0.140	kill	0.374	want	0.228	npr
0.361	polit	0.221	white	0.137	peopl	0.366	lot	0.227	van
0.356	hous	0.214	go	0.134	kind	0.363	don	0.226	case
0.317	block	0.202	want	0.130	got	0.359	time	0.211	unit
0.299	right	0.196	elect	0.124	work	0.336	good	0.211	court
6		7		8		9		10	
1.000	song	1.000	music	1.000	know	1.000	rule	1.000	johnson
0.657	sing	0.324	soundbit	0.866	like	0.873	frank	0.380	kennedi
0.346	soundbit	0.321	jazz	0.675	martin	0.754	say	0.332	say
0.277	love	0.192	musician	0.658	school	0.677	deriv	0.276	presid
0.148	singer	0.144	record	0.626	just	0.610	regul	0.268	davi
0.146	just	0.144	play	0.495	young	0.567	financi	0.261	power
0.129	time	0.130	npr	0.472	kid	0.559	davi	0.224	right
0.119	music	0.110	new	0.447	think	0.528	trade	0.218	robert
0.116	know	0.100	listen	0.393	kind	0.527	know	0.161	time
0.112	write	0.086	compos	0.372	gross	0.505	gross	0.157	civil
0.108	record	0.086	classic	0.358	girl	0.497	consum	0.150	use
0.102	block	0.084	sound	0.312	thing	0.480	reform	0.138	man
0.097	album	0.081	conan	0.309	boy	0.458	mean	0.132	doe
0.091	don	0.080	blue	0.276	want	0.449	just	0.123	know
0.087	did	0.078	artist	0.229	yeah	0.411	bank	0.118	washington
0.081	gross	0.076	hear	0.222	don	0.385	right	0.116	leader
11		12		13		14		15	
1.000	know	1.000	conan	1.000	laughter	1.000	know	1.000	like
0.471	think	0.206	thank	0.846	think	0.707	martin	0.657	band
0.444	just	0.152	talk	0.794	peopl	0.448	think	0.491	soundbit
0.279	mean	0.123	yeah	0.748	martin	0.290	just	0.382	music
0.258	realli	0.119	laughter	0.746	right	0.232	don	0.350	just
0.228	thing	0.114	know	0.617	simon	0.222	peopl	0.346	sing
0.205	flatow	0.104	neal	0.569	say	0.213	mean	0.331	member
0.192	sort	0.102	caller	0.506	book	0.187	realli	0.244	yeah
0.180	go	0.098	nation	0.489	question	0.163	like	0.178	simon
0.178	peopl	0.090	yes	0.470	answer	0.162	want	0.165	npr
0.168	don	0.080	ari	0.460	thing	0.153	thing	0.156	thank
0.159	yeah	0.079	let	0.448	said	0.152	term	0.138	metal
0.157	actual	0.077	800	0.434	time	0.149	talk	0.137	right
0.155	lot	0.076	989	0.413	yeah	0.139	feel	0.133	heavi
0.148	right	0.069	think	0.412	don	0.139	tell	0.132	laughter
0.145	kind	0.067	don	0.403	yes	0.129	go	0.132	come

Out[344]=

16		17		18		19		20	
1.000	sing	1.000	martin	1.000	play	1.000	song	1.000	record
0.637	honey	0.687	khan	0.560	know	0.798	raz	0.981	blue
0.590	rock	0.440	know	0.397	like	0.746	record	0.872	year
0.588	sweet	0.222	yeah	0.373	just	0.735	like	0.854	note
0.354	martin	0.220	sing	0.330	yeah	0.506	soundbit	0.646	said
0.317	children	0.202	soundbit	0.261	band	0.432	album	0.542	just
0.298	gonna	0.153	let	0.239	gross	0.376	track	0.460	martin
0.225	say	0.118	got	0.237	soundbit	0.307	kind	0.421	time
0.223	spirit	0.117	littl	0.218	laughter	0.302	call	0.415	ago
0.217	think	0.117	album	0.213	guitar	0.302	stewart	0.321	day
0.187	stranger	0.114	just	0.183	realli	0.294	sound	0.302	want
0.132	thing	0.098	life	0.177	did	0.288	band	0.293	thank
0.122	robinson	0.095	tell	0.134	time	0.241	sort	0.284	raz
0.116	music	0.088	fight	0.130	kind	0.239	yeah	0.277	love
0.114	group	0.087	laughter	0.128	jone	0.223	just	0.275	album
0.111	peopl	0.080	go	0.127	record	0.221	sing	0.269	jazz

21		22		23		24		25	
1.000	gross	1.000	new	1.000	know	1.000	black	1.000	music
0.709	like	0.829	know	0.494	like	0.393	like	0.709	soundbit
0.589	know	0.488	song	0.371	yeah	0.377	peopl	0.527	piec
0.284	realli	0.326	like	0.340	simon	0.349	white	0.410	simon
0.276	think	0.284	orlean	0.297	laughter	0.270	just	0.357	raz
0.255	just	0.277	yeah	0.293	soundbit	0.251	american	0.339	compos
0.240	kind	0.274	album	0.276	wait	0.224	african	0.300	huizenga
0.171	felt	0.240	conan	0.264	don	0.218	race	0.224	think
0.144	film	0.173	york	0.204	just	0.188	gordon	0.222	symphoni
0.138	did	0.165	realli	0.178	ari	0.158	know	0.218	realli
0.130	mean	0.163	hansen	0.172	won	0.155	music	0.217	npr
0.130	low	0.158	call	0.155	sing	0.150	soundbit	0.193	hear
0.124	sort	0.147	sort	0.153	got	0.143	say	0.176	siegel
0.118	movi	0.145	kind	0.142	block	0.140	reed	0.157	orchestra
0.118	feel	0.135	go	0.139	thank	0.136	mean	0.156	tom
0.116	stori	0.130	just	0.139	flatow	0.133	man	0.150	sound

26		27		28		29		30	
1.000	brown	1.000	soundbit	1.000	gross	1.000	know	1.000	stew
0.173	jame	0.924	say	0.339	song	0.789	state	0.841	know
0.157	soundbit	0.917	npr	0.228	record	0.607	gross	0.779	read
0.136	say	0.552	sing	0.178	did	0.512	right	0.715	sing
0.131	just	0.474	report	0.167	lynn	0.485	peopl	0.671	like
0.113	peopl	0.400	news	0.160	die	0.458	abort	0.649	simon
0.113	npr	0.396	like	0.134	day	0.401	life	0.647	pass
0.092	right	0.331	host	0.123	said	0.369	think	0.447	make
0.089	talk	0.326	music	0.122	got	0.368	say	0.446	hard
0.086	go	0.322	day	0.121	terri	0.356	davi	0.417	music
0.085	got	0.315	year	0.112	band	0.263	law	0.374	unidentifi
0.084	thing	0.270	stand	0.111	wainwright	0.245	did	0.361	strang
0.082	smith	0.263	man	0.110	pop	0.231	nuclear	0.357	man
0.080	music	0.253	new	0.110	didn	0.226	way	0.335	bear
0.076	man	0.249	unidentifi	0.110	call	0.224	time	0.324	right
0.075	vulner	0.209	provid	0.106	air	0.224	reason	0.302	thing

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, package 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.