

A monad for Latent Semantic Analysis workflows

Version 0.9

Anton Antonov
MathematicaForPrediction at WordPress
MathematicaForPrediction at GitHub
MathematicaVsR at GitHub
September 2019

Introduction

In this document we describe the design and implementation of a (software programming) monad, [Wk1], for Latent Semantic Analysis workflows specification and execution. The design and implementation are done with Mathematica / Wolfram Language (WL).

What is Latent Semantic Analysis (LSA)? : A statistical method (or a technique) for finding relationships in natural language texts that is based on the so called Distributional hypothesis, [Wk2, Wk3]. (The Distributional hypothesis can be simply stated as “linguistic items with similar distributions have similar meanings”; for insightful philosophical and scientific discussion see [MS1].) LSA can be seen as the application of Dimensionality reduction techniques over matrices derived with the Vector space model.

The goal of the monad design is to make the specification of LSA workflows (relatively) easy, straightforward, by following a certain main scenario and specifying variations over that scenario.

The monad is named LSAMon and it is based on the State monad package “StateMonadCodeGenerator.m”, [AAp1, AA1], the document-term matrix making package "DocumentTermMatrixConstruction.m", [AAp4, AA2], the Non-Negative Matrix Factorization (NNMF) package "NonNegativeMatrixFactorization.m", [AAp5, AA2], and the package “SSparseMatrix.m”, [AAp2, AA5], that provides matrix objects with named rows and columns.

The data for this document is obtained from WL’s repository and it is manipulated into a certain ready-to-utilize form (and uploaded to GitHub.)

The monadic programming design is used as a Software Design Pattern. The LSAMon monad can be also seen as a Domain Specific Language (DSL) for the specification and programming of machine learning classification workflows.

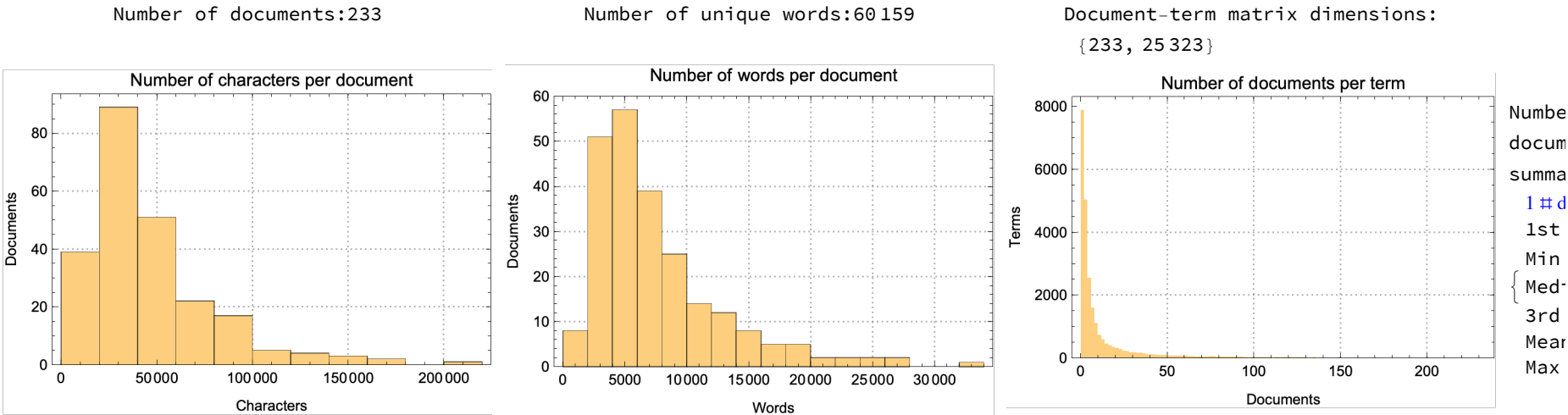
Here is an example of using the LSAMon monad over a collection of documents that consists of 233 US state of union speeches.

Out[]:=

```
LSAMonUnit[aStateOfUnionSpeeches] =>
  LSAMonMakeDocumentTermMatrix =>
  LSAMonEchoDocumentsStatistics =>
  LSAMonApplyTermWeightFunctions["IDF", "None", "Cosine"] =>
  LSAMonExtractTopics[24, "MaxSteps" -> 20, Method -> "NNMF"] =>
  LSAMonEchoTopicsTable["NumberOfTableColumns" -> 6] =>
  LSAMonExtractStatisticalThesaurus[{"vote", "bank", "health", "war"}, 12] =>
  LSAMonEchoStatisticalThesaurus
```

```
Uplift text data into the monad.
Make the document-term contingency matrix.
Echo the documents collection statistics.
Apply LSI weight functions.
Extract topics using NNMF.
Echo topics table.
Extract statistical thesaurus.
Echo thesaurus entries.
```

» Context value "documents":



» topics table:

1	1.000 isthmus 0.842 panama 0.481 granada 0.450 colombia 0.392 canal 0.307 kansas 0.246 transit 0.237 route 0.235 treaty 0.216 revolution 0.208 territory 0.182 1856	5	1.000 jobs 0.916 tonight 0.630 americans 0.624 don 0.502 budget 0.476 let 0.475 businesses 0.466 help 0.457 spending 0.436 college 0.418 cuts 0.402 deficit	9	1.000 interstate 0.741 processes 0.731 railroads 0.695 railways 0.652 tasks 0.625 unrest 0.581 storage 0.569 industrial 0.460 matter 0.458 presently 0.448 floating 0.444 thoughtful	13	1.000 program 0.742 communist 0.582 economic 0.570 atomic 0.530 programs 0.384 farm 0.369 federal 0.352 housing 0.335 insurance 0.333 collective 0.331 billion 0.324 budget	17	1.000 fighting 0.986 japanese 0.928 planes 0.728 soviet 0.619 production 0.528 nurses 0.521 victory 0.475 allies 0.460 air 0.440 conquest 0.435 korea 0.425 weapons	21	1.000 terrorists 0.983 iraq 0.757 iraqi 0.675 terror 0.557 terrorist 0.468 afghanistan 0.446 qaeda 0.361 tonight 0.281 empower 0.269 fight 0.265 help 0.265 americans
2	1.000 spain 0.558 colonies 0.490 likewise 0.398 presumed 0.373 respecting 0.371 treaty 0.368 duties 0.354 negotiation 0.354 object 0.354 tribes 0.352 fortifications 0.349 minister	6	1.000 tonight 0.969 iraq 0.633 jobs 0.575 terrorists 0.500 americans 0.491 drug 0.485 businesses 0.449 seniors 0.419 infrastructure 0.417 marriage 0.412 iraqi 0.380 regime	10	1.000 kansas 0.901 constitution 0.811 mexico 0.710 1858 0.682 slavery 0.619 whilst 0.580 duties 0.563 california 0.527 territory 0.512 subject 0.488 slaves 0.472 minister	14	1.000 gentlemen 0.564 militia 0.425 objects 0.414 indians 0.412 burthens 0.371 requisite 0.363 pleasing 0.350 legislature 0.343 tribes 0.318 persuasion 0.298 impressions 0.298 papers	18	1.000 enemy 0.699 savages 0.608 british 0.485 militia 0.440 lake 0.355 whilst 0.330 captain 0.321 regulars 0.291 prisoners 0.285 command 0.282 proofs 0.259 orders	22	1.000 interstate 0.866 corporations 0.785 forest 0.584 islands 0.579 man 0.506 standpoint 0.451 industrial 0.450 philippine 0.430 conditions 0.423 type 0.422 cable 0.421 philippines
3	1.000 soviet 0.543 1980 0.411 salt 0.389 oil 0.372 nuclear 0.354 afghanistan 0.331 inflation 0.257 israel 0.247 tonight 0.242 commitment 0.223 strategic 0.172 iran	7	1.000 banks 0.928 recovery 0.832 000 0.704 banking 0.687 economic 0.606 depression 0.525 veterans 0.520 agencies 0.519 unemployment 0.511 corporation 0.469 relief 0.441 reserve	11	1.000 silver 0.802 gold 0.540 notes 0.510 coinage 0.488 cent 0.447 circulation 0.420 currency 0.417 1890 0.410 000 0.408 1893 0.388 bonds 0.333 coin	15	1.000 programs 0.954 oil 0.801 billion 0.689 percent 0.689 energy 0.515 strategic 0.506 nuclear 0.497 inflation 0.496 decrees 0.494 major 0.471 spending 0.464 1980	19	1.000 consular 0.803 award 0.795 convention 0.755 diplomatic 0.728 chile 0.701 treaty 0.690 boundary 0.677 cuba 0.665 arbitration 0.653 1883 0.638 majesty 0.625 international	23	1.000 soviet 0.798 communist 0.727 rulers 0.718 tonight 0.664 challenge 0.596 atomic 0.574 21st 0.478 parents 0.433 child 0.421 children 0.397 communists 0.376 drugs
4	1.000 texas 0.907 mexico 0.468 annexation 0.424 paper 0.416 specie 0.408 mexican 0.399 bank 0.387 banks 0.362 notes 0.343 1837 0.341 currency 0.300 oregon	8	1.000 democratic 0.793 democracy 0.748 economic 0.736 today 0.638 program 0.603 housing 0.523 farm 0.492 bargaining 0.481 inflation 0.440 income 0.431 recovery 0.430 ideals	12	1.000 gentlemen 0.673 philadelphia 0.637 commissioners 0.543 article 0.505 captures 0.422 damages 0.412 6th 0.403 majesty 0.392 french 0.383 engagements 0.352 amity 0.320 7th	16	1.000 vietnam 0.517 tonight 0.289 billion 0.192 percent 0.169 try 0.143 commitments 0.135 communist 0.133 americans 0.124 crime 0.121 programs 0.119 help 0.107 propose	20	1.000 000 0.528 flood 0.521 veterans 0.477 cent 0.465 1929 0.459 marketing 0.448 farmer 0.402 agriculture 0.371 aviation 0.357 court 0.354 league 0.353 board	24	1.000 vessels 0.938 harbors 0.739 augmentation 0.717 dispositions 0.680 orleans 0.674 port 0.665 louisiana 0.621 lessen 0.568 ohio 0.555 ceded 0.554 mississippi 0.546 mediterranean

» statistical thesaurus:

word	statistical thesaurus
bank	{bank, 1837, annexation, moneys, paper, regarded, 1842, calculated, treasury, payment, specie, exchange}
health	{health, working, care, economy, education, communities, costs, investment, move, got, workers, start}
vote	{vote, compete, choice, code, republican, person, climate, asked, earned, leader, protect, says}
war	{war, separates, nations, hasten, proclaim, loose, overrun, lighten, weigh, means, splendid, surrendered}

The table above is produced with the package “MonadicTracing.m”, [Aap2, AA1], and some of the explanations below also utilize that package.

As it was mentioned above the monad LSAMon can be seen as a DSL. Because of this the monad pipelines made with LSAMon are sometimes called “specifications”.

Remark: With “term” we mean “a word, a word stem, or other type of token”.

Remark: LSA and Latent Semantic Indexing (LSI) are considered more or less to be synonyms. I think that “latent semantic analysis” sounds more universal and that “latent semantic indexing” as a name refers to a specific Information Retrieval technique. Below we refer to “LSI functions” like “IDF” and “TF-IDF” that are applied within the generic LSA workflow.

Contents description

The document has the following structure.

- The sections "Package load" and "Data load" obtain the needed code and data.
 - (Needed and put upfront from the “Reproducible research” point of view.)

- The sections "Design consideration" and "Monad design" provide motivation and design decisions rationale.
- The sections "LSAMon overview" and "Monad elements" provide technical descriptions needed to utilize the LSAMon monad .
 - (Using a fair amount of examples.)
- The section "Unit tests" describes the tests used in the development of the LSAMon monad.
 - (The random pipelines unit tests are especially interesting.)
- The section "Future plans" outlines future directions of development.
 - (The most interesting and important one is the “conversational agent” direction.)
- The section "Implementation notes" just says that LSAMon’s development process and this document follow the ones of the classifications workflows monad ClCon, [AA6].

Remark: One can read only the sections "Introduction", "Design consideration", "Monad design", and "LSAMon overview". That set of sections provide a fairly good, programming language agnostic exposition of the substance and novel ideas of this document.

Package load

The following commands load the packages [AAp1--AAp7]:

```
In[154]:= Import["https://raw.githubusercontent.com/antononcube/MathematicaForPrediction/master/MonadicProgramming/MonadicLatentSemanticAnalysis.m"]
Import[
  "https://raw.githubusercontent.com/antononcube/MathematicaForPrediction/master/MonadicProgramming/MonadicTracing.m"]
```

Data load

In this section we load data that is used in the rest of the document. The text data was obtained through WL’s repository, transformed in a certain more convenient form, and uploaded to GitHub.

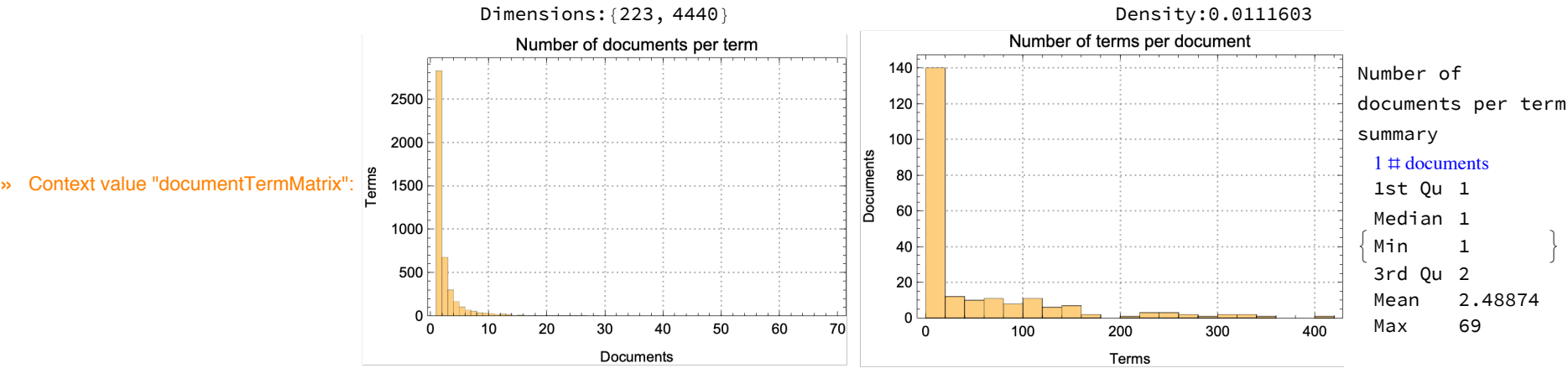
The text summarization and plots are done through LSAMon, which in turn uses the function RecordsSummary from the package “MathematicaForPredictionUtilities.m”, [AAp7].

Hamlet

```
In[376]:= textHamlet = ToString /@ Flatten[
  Import["https://raw.githubusercontent.com/antononcube/MathematicaVsR/master/Data/MathematicaVsR-Data-Hamlet.csv"]];

In[377]:= TakeLargestBy[Tally[DeleteStopwords[ToLowerCase[Flatten[TextWords /@ textHamlet]]]], #[[2]] &, 20]
Out[377]:= {{ham, 358}, {lord, 225}, {king, 196}, {o, 124}, {queen, 120}, {shall, 114}, {good, 109}, {hor, 109}, {come, 107}, {hamlet, 107},
  {thou, 105}, {let, 96}, {thy, 86}, {pol, 86}, {like, 81}, {sir, 75}, {'t, 75}, {know, 74}, {enter, 73}, {th, 72}}

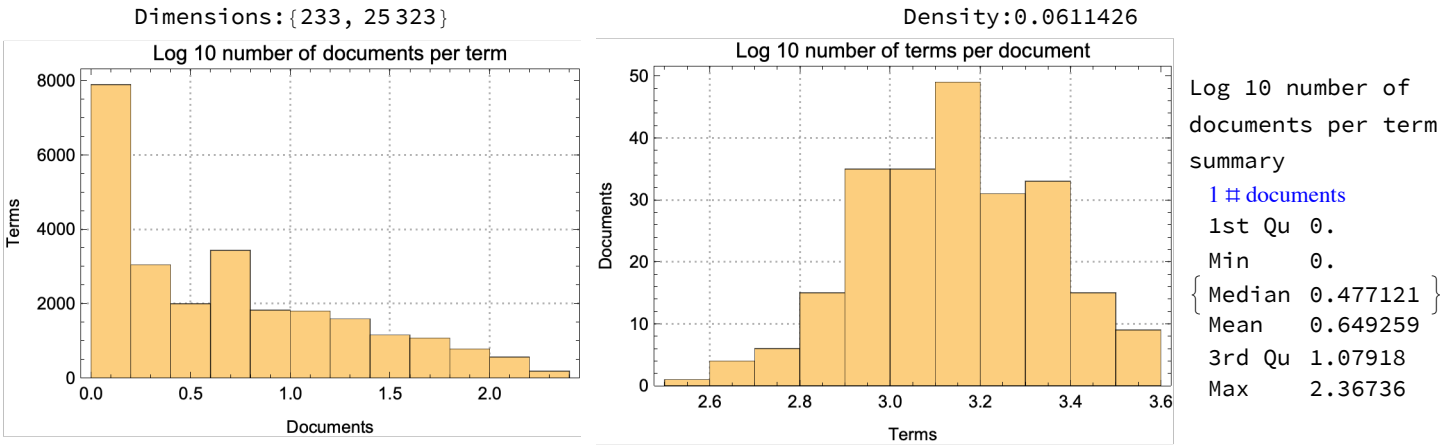
In[378]:= LSAMonUnit[textHamlet] ==> LSAMonMakeDocumentTermMatrix ==> LSAMonEchoDocumentTermMatrixStatistics;
```



USA state of union speeches

```
In[389]:= url =
  "https://github.com/antononcube/MathematicaVsR/blob/master/Data/MathematicaVsR-Data-StateOfUnionSpeeches.JSON.zip?raw=true";
str = Import[url, "String"];
filename = First@Import[StringToStream[str], "ZIP"];
aStateOfUnionSpeeches = Association@ImportString[Import[StringToStream[str], {"ZIP", filename, "String"}], "JSON"];
```

```
In[383]:= lsaObj =  
  LSAMonUnit[aStateOfUnionSpeeches] ⇒ LSAMonMakeDocumentTermMatrix ⇒ LSAMonEchoDocumentTermMatrixStatistics["LogBase" → 10];  
» Context value "documentTermMatrix":
```



```
In[384]:= TakeLargest[ColumnSumsAssociation[lsaObj ⇒ LSAMonTakeDocumentTermMatrix], 12]  
Out[384]= <|government → 7106, states → 6502, congress → 5023, united → 4847, people → 4103,  
  year → 4022, country → 3469, great → 3276, public → 3094, new → 3022, 000 → 2960, time → 2922|>
```

Stop words

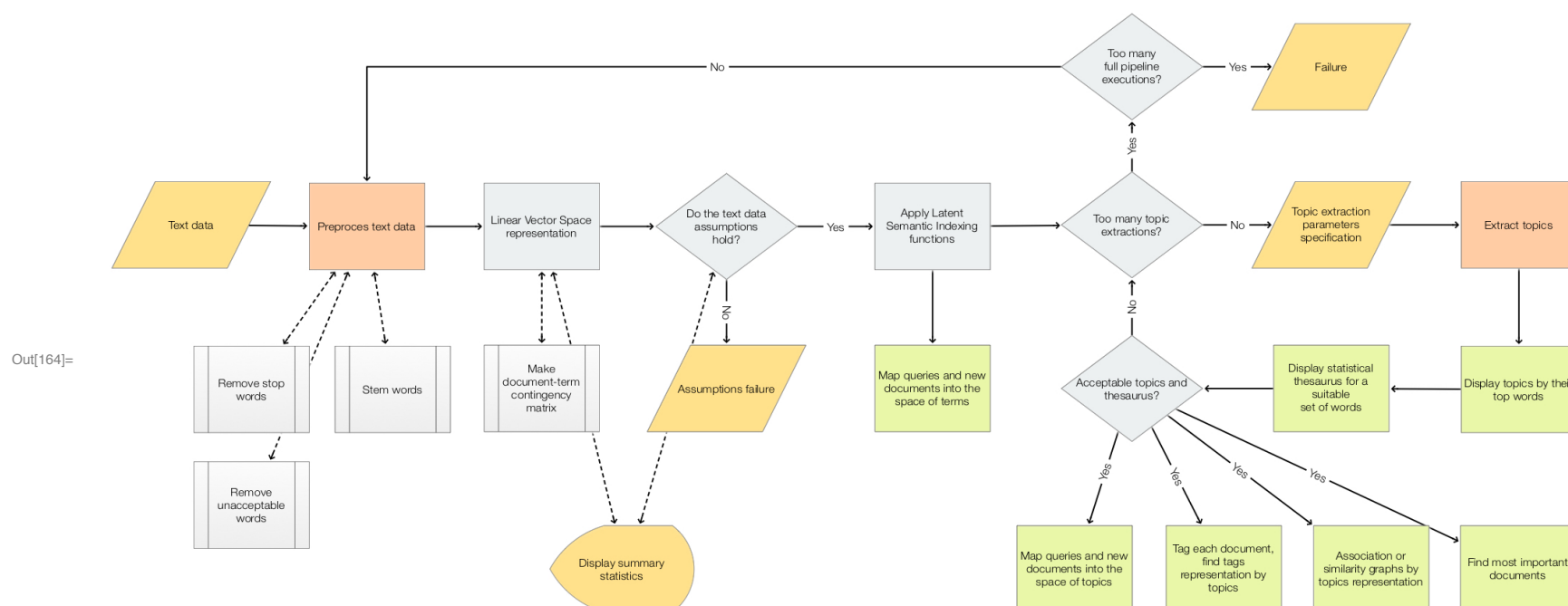
In some of the examples below we want to explicitly specify the stop words. Here are stop words derived using the built-in functions DictionaryLookup and DeleteStopwords.

```
In[162]:= stopWords = Complement[DictionaryLookup["*"], DeleteStopwords[DictionaryLookup["*"]]];  
  
In[163]:= Short[stopWords]  
Out[163]//Short= {a, about, above, across, add-on, after, again, against, all, almost, alone,  
  <<283>>, yet, you, you'd, you'll, your, you're, yours, yourself, yourselves, you've}
```

Design considerations

The steps of the main LSA workflow addressed in this document follow.

1. Get a collection of documents with associated ID's.
 2. Create a document-term matrix.
 - 2.1. Here we apply the Bag-or-words model and Vector space model.
 - 2.1.1. The sequential order of the words is ignored and each document is represented as a point in a multi-dimensional vector space.
 - 2.1.2. That vector space axes correspond to the unique words found in the whole document collection.
 - 2.2. Consider the application of stemming rules.
 - 2.3. Consider the removal of stop words.
 3. Apply matrix-entries weighting functions.
 - 3.1. Those functions come from LSI.
 - 3.2. Functions like "IDF", "TF-IDF", "GFIDF".
 4. Extract topics.
 - 4.1. One possible statistical way of doing this is with Dimensionality reduction.
 - 4.2. We consider using Singular Value Decomposition (SVD) and Non-Negative Matrix Factorization (NNMF).
 5. Make and display the topics table.
 6. Extract and display a statistical thesaurus of selected words.
 7. Map search queries or unseen documents over the extracted topics.
 8. Find the most important documents in the document collection. (Optional.)
- The following flow-chart corresponds to the list of steps above.



In order to address:

- the introduction of new elements in LSA workflows,
- workflows elements variability, and
- workflows iterative changes and refining,

it is beneficial to have a DSL for LSA workflows. We choose to make such a DSL through a functional programming monad, [Wk1, AA1].

Here is a quote from [Wk1] that fairly well describes why we choose to make a classification workflow monad and hints on the desired properties of such a monad.

[...] The monad represents computations with a sequential structure: a monad defines what it means to chain operations together. This enables the programmer to build pipelines that process data in a series of steps (i.e. a series of actions applied to the data), in which each action is decorated with the additional processing rules provided by the monad. [...]

Monads allow a programming style where programs are written by putting together highly composable parts, combining in flexible ways the possible actions that can work on a particular type of data. [...]

Remark: Note that quote from [Wk1] refers to chained monadic operations as “pipelines”. We use the terms “monad pipeline” and “pipeline” below.

Monad design

The monad we consider is designed to speed-up the programming of LSA workflows outlined in the previous section. The monad is named LSAMon for “**L**atent **S**emantic **A**nalysis **M**onad”.

We want to be able to construct monad pipelines of the general form:

$$\text{LSAMon}[_] \xrightarrow{\text{LSAMonBind}[\text{LSAMon}[_], f_-]} f_1 \xrightarrow{\text{LSAMonBind}[\text{LSAMon}[_], f_-]} f_2 \xrightarrow{\text{LSAMonBind}[\text{LSAMon}[_], f_-]} \dots \xrightarrow{\text{LSAMonBind}[\text{LSAMon}[_], f_-]} f_k \quad (1)$$

LSAMon is based on the State monad, [Wk1, AA1], so the monad pipeline form (1) has the following more specific form:

$$\text{LSAMon}[pval_-, context_-] \xrightarrow{\text{QRMonBind}[m_-, f_-]} \dots \left(\begin{array}{ll} f_i[\$LSAMonFailure] & m \equiv \$LSAMonFailure \\ f_i[x_-, c_Association] & m \text{ is } LSAMon[x_-, c_Association] \\ \$LSAMonFailure & \text{otherwise} \end{array} \right) \xrightarrow{\text{LSAMonBind}[m_-, f_-]} \dots \quad (2)$$

This means that some monad operations will not just change the pipeline value but they will also change the pipeline context.

In the monad pipelines of LSAMon we store different objects in the contexts for at least one of the following two reasons.

1. The object will be needed later on in the pipeline, or
2. The object is (relatively) hard to compute.

Such objects are document-term matrix, Dimensionality reduction factors and the related topics.

Let us list the desired properties of the monad.

- Rapid specification of non-trivial LSA workflows.
- The monad works with associations with string values, list of strings.
- The monad use the Linear vector spaces model .
- The document-term frequency matrix is can be created after removing stop words and/or word stemming.
- It is easy to specify and apply different LSI weight functions. (Like “IDF” or “GFIDF”).
- The monad can do dimension reduction with SVD and NMF and corresponding matrix factors are retrievable with monad functions.
- Documents (or query strings) external to the monad a easily mapped into monad’s Linear vector space of terms and the Linear vector space of topics.
- The monad allows of cursory examination and summarization of the data.
- The pipeline values can be of different types. Most monad functions modify the pipeline value; some modify the context; some just echo results.

- It is easy to obtain the pipeline value, context, and different context objects for manipulation outside of the monad.
- It is easy to tabulate extracted topics and related statistical thesauri.
- It is easy to specify and apply re-weighting functions for the entries of the document-term contingency matrices.

The LSAMon components and their interactions are fairly simple.

The main LSAMon operations implicitly put in the context or utilize from the context the following objects:

- document-term matrix,
- the factors obtained by matrix factorization algorithms,
- extracted topics.

Note the that the monadic set of types of LSAMon pipeline values is fairly heterogenous and certain awareness of “the current pipeline value” is assumed when composing LSAMon pipelines.

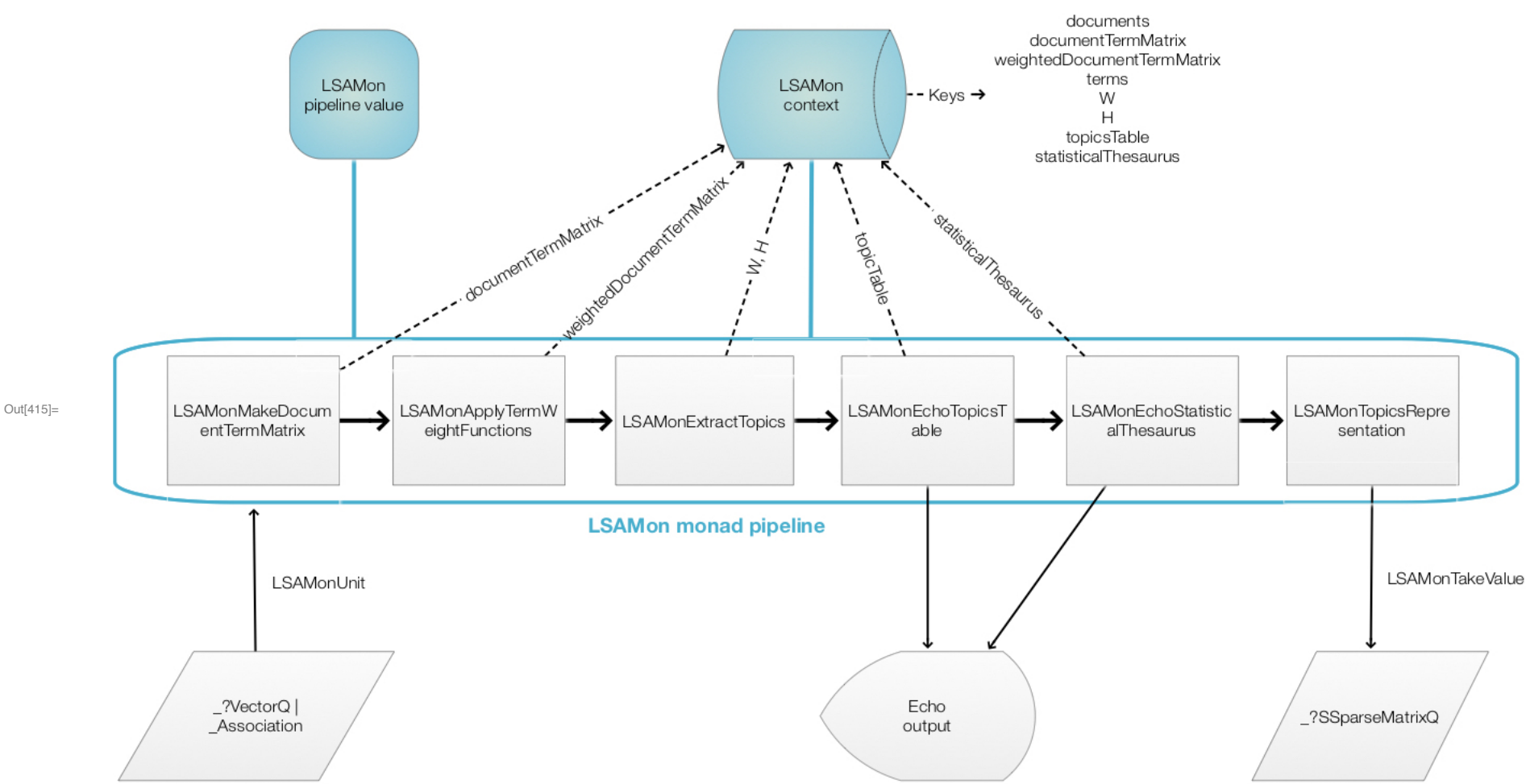
Obviously, we can put in the context any object through the generic operations of the State monad of the package “StateMonadGenerator.m”, [AAp1].

LSAMon overview

When using a monad we lift certain data into the “monad space”, using monad’s operations we navigate computations in that space, and at some point we take results from it.

With the approach taken in this document the “lifting” into the LSAMon monad is done with the function LSAMonUnit. Results from the monad can be obtained with the functions LSAMonTakeValue, LSAMonContext, or with the other LSAMon functions with the prefix “LSAMonTake” (see below.)

Here is a corresponding diagram of a generic computation with the LSAMon monad:



Remark: It is a good idea to compare the diagram with formulas (1) and (2).

Let us examine a concrete LSAMon pipeline that corresponds to the diagram above. In the following table each pipeline operation is combined together with a short explanation and the context keys after its execution.

Here is the output of the pipeline:

The LSAMon functions are separated into four groups:

- operations,
- setters and droppers,
- takers,
- State Monad generic functions.

Monad functions interaction with the pipeline value and context

An overview of the those functions is given in the tables in next two sub-sections. The next section, “Monad elements”, gives details and examples for the usage of the LSAMon operations.

#	name	echoes result	puts in context	uses from context	uses pipeline value
1	operations				

2	LSAMonApplyTermWeightFunctions	no	{weightedDocumentTermMatrix, globalWeights, localWeightFunction, normalizerFunction}	{documentTermMatrix}	through LSAMonGetDocuments
3	LSAMonDocumentCollectionQuery	no	none	none	no
4	LSAMonEchoDocumentsStatistics	yes	none	{documents, documentTermMatrix}	no
5	LSAMonEchoDocumentTermMatrixStatistics	yes	none	{documentTermMatrix}	no
6	LSAMonEchoStatisticalThesaurus	yes	none	{statisticalThesaurus}	no
7	LSAMonEchoTopicsTable	yes	none	{topicsTable}	no
8	LSAMonExtractStatisticalThesaurus	no	{statisticalThesaurus}	{terms, topicColumnPositions}	no
9	LSAMonExtractTopics	no	{W, H, topicColumnPositions, automaticTopicNames, method}	{weightedDocumentTermMatrix}	no
10	LSAMonFindMostImportantDocuments	no	none	{weightedDocumentTermMatrix}	tries first
11	LSAMonGetDocuments	no	none	{documents}	if the context does not have "documents".
12	LSAMonInterpretBasisVector	no	none	{W, H}	no
13	LSAMonMakeDocumentTermMatrix	no	{documentTermMatrix, stemmingRules, stopWords}	{documents}	no
14	LSAMonMakeGraph	no	none	{documentTermMatrix, weightedDocumentTermMatrix}	tries first
15	LSAMonMakeTopicsTable	no	{topicsTable}	{W, H}	no
16	LSAMonRepresentByTerms	no	none	{documentTermMatrix, globalWeights, localWeightFunction, normalizerFunction, stemmingRules, stopWords}	no
17	LSAMonRepresentByTopics	no	none	{documentTermMatrix, globalWeights, localWeightFunction, normalizerFunction, stemmingRules, stopWords, H}	no
18	LSAMonRepresentDocumentTagsByTopics	no	{docTopicIndices}	{documentTermMatrix, W, H}	no
19	setters				
20	LSAMonSetAutomaticTopicNames	no	automaticTopicNames	none	no
21	LSAMonSetContext	no	context	none	no
22	LSAMonSetDocuments	no	documents	none	no
23	LSAMonSetDocumentTermMatrix	no	documentTermMatrix	none	no
24	LSAMonSetGlobalWeightFunction	no	globalWeightFunction	none	no
25	LSAMonSetGlobalWeights	no	globalWeights	none	no
26	LSAMonSetH	no	H	none	no
27	LSAMonSetLocalWeightFunction	no	localWeightFunction	none	no
28	LSAMonSetMethod	no	method	none	no
29	LSAMonSetNormalizerFunction	no	normalizerFunction	none	no
30	LSAMonSetStatisticalThesaurus	no	statisticalThesaurus	none	no
31	LSAMonSetStemmingRules	no	stemmingRules	none	no
32	LSAMonSetStopWords	no	stopWords	none	no
33	LSAMonSetTerms	no	terms	none	no
34	LSAMonSetTopicColumnPositions	no	topicColumnPositions	none	no
35	LSAMonSetTopicsTable	no	topicsTable	none	no
36	LSAMonSetValue	no	value	none	no
37	LSAMonSetW	no	W	none	no

Out[]=

38	LSAMonSetWeightedDocumentTermMatrix	no	weightedDocumentTermMatrix	none	no
39	droppers				
40	LSAMonDropAutomaticTopicNames	no	no	automaticTopicNames	no
41	LSAMonDropDocuments	no	no	documents	no
42	LSAMonDropDocumentTermMatrix	no	no	documentTermMatrix	no
43	LSAMonDropFromContext	no	no	fromContext	no
44	LSAMonDropGlobalWeightFunction	no	no	globalWeightFunction	no
45	LSAMonDropGlobalWeights	no	no	globalWeights	no
46	LSAMonDropH	no	no	H	no
47	LSAMonDropLocalWeightFunction	no	no	localWeightFunction	no
48	LSAMonDropMethod	no	no	method	no
49	LSAMonDropNormalizerFunction	no	no	normalizerFunction	no
50	LSAMonDropStatisticalThesaurus	no	no	statisticalThesaurus	no
51	LSAMonDropStemmingRules	no	no	stemmingRules	no
52	LSAMonDropStopWords	no	no	stopWords	no
53	LSAMonDropTerms	no	no	terms	no
54	LSAMonDropTopicColumnPositions	no	no	topicColumnPositions	no
55	LSAMonDropTopicsTable	no	no	topicsTable	no
56	LSAMonDropW	no	no	W	no
57	LSAMonDropWeightedDocumentTermMatrix	no	no	weightedDocumentTermMatrix	no
58	takers				
59	LSAMonTakeAutomaticTopicNames	no	no	automaticTopicNames	no
60	LSAMonTakeContext	no	no	context	no
61	LSAMonTakeDocuments	no	no	documents	no
62	LSAMonTakeDocumentTermMatrix	no	no	documentTermMatrix	no
63	LSAMonTakeGlobalWeightFunction	no	no	globalWeightFunction	no
64	LSAMonTakeGlobalWeights	no	no	globalWeights	no
65	LSAMonTakeH	no	no	H	no
66	LSAMonTakeLocalWeightFunction	no	no	localWeightFunction	no
67	LSAMonTakeMatrix	no	no	matrix	no
68	LSAMonTakeMethod	no	no	method	no
69	LSAMonTakeNormalizerFunction	no	no	normalizerFunction	no
70	LSAMonTakeStatisticalThesaurus	no	no	statisticalThesaurus	no
71	LSAMonTakeStemmingRules	no	no	stemmingRules	no
72	LSAMonTakeStopWords	no	no	stopWords	no
73	LSAMonTakeTerms	no	no	terms	no
74	LSAMonTakeTexts	no	no	texts	no
75	LSAMonTakeTopicColumnPositions	no	no	topicColumnPositions	no
76	LSAMonTakeTopicsTable	no	no	topicsTable	no
77	LSAMonTakeValue	no	no	value	no
78	LSAMonTakeW	no	no	W	no
79	LSAMonTakeWeightedDocumentTermMatrix	no	no	weightedDocumentTermMatrix	no
80	LSAMonTakeWeightedMatrix	no	no	weightedMatrix	no

State monad functions

Here are the LSAMon State Monad functions (generated using the prefix “LSAMon”, [AAp1, AA1].)

#	name	description
1	LSAMonAddToContext	LSAMonAddToContext[varName_String] adds to the monad context the monad value under key varName. LSAMonAddToContext[arg_Association] joins the monad context with arg. LSAMonAddToContext[] joins the monad context with the monad value.
2	LSAMonBind	Monad binding function.
3	LSAMonDropFromContext	Drop from the monad context elements with the specified keys.
4	LSAMonEcho	Echoes the argument. If no argument is given the short print of the monad object is echoed.
5	LSAMonEchoContext	Echoes the monad context.
6	LSAMonEchoFunctionContext	Echoes function application over the monad context.
7	LSAMonEchoFunctionValue	Echoes function application over the monad value.
8	LSAMonEchoValue	Echoes the monad value.
9	LSAMonFail	Failure.
10	LSAMonIf	LSAMonIf[f_, fYes_, fNo_] executes fYes[LSAMonUnit[xs,context]] if f[LSAMonUnit[xs,context]] is True; fNo[LSAMonUnit[xs,context]] otherwise.
11	LSAMonIfElse	LSAMonIfElse[testFunc_, fYes_, fNo_] executes fYes[xs, context] if TrueQ[testFunc[xs, context]]; otherwise fNo[xs, context].
12	LSAMonModifyContext	LSAMonModifyContext[f] replaces the monad context f[context].
13	LSAMonOption	If the application of the argument to the monad produces monad failure the monad is unchanged.
14	LSAMonPutContext	Replaces the monad context with the argument.
15	LSAMonPutValue	Replaces the monad value with the argument.
16	LSAMonRetrieveFromContext	LSAMonRetrieveFromContext[varName_String] retrieves from the monad context the value of the key varName.
17	LSAMonSetContext	Replaces the monad context with the argument.
18	LSAMonSetValue	Replaces the monad value with the argument.
19	LSAMonSucceed	Success.
20	LSAMonTakeContext	Takes the monad context.
21	LSAMonTakeValue	Takes the monad value.
22	LSAMonUnit	LSAMon monad unit constructor.
23	LSAMonUnitQ	LSAMon monad unit test.
24	LSAMonWhen	Shorter version of LSAMonIfElse.

Main monad functions

Here are the usage descriptions of the main (not monad-supportive) LSAMon functions, which are explained in detail in the next section.

#	name	description
1	LSAMonApplyTermWeightFunctions	Apply term weight functions to entries of the document-term matrix.
2	LSAMonDocumentCollectionQ	Gives True if the argument is a text collection.
3	LSAMonEchoDocumentsStatistics	Echo statistics for the text collection.
4	LSAMonEchoDocumentTermMatrixStatistics	Echo document-term matrix statistics.
5	LSAMonEchoStatisticalThesaurus	Echo the statistical thesaurus entries for a specified list of words.
6	LSAMonEchoTopicsTable	Echo the a table with the extracted topics.
7	LSAMonExtractStatisticalThesaurus	Extract the statistical thesaurus for specified list of words.
8	LSAMonExtractTopics	Extract topics.
9	LSAMonFindMostImportantDocuments	Find the most important texts in the text collection.
10	LSAMonGetDocuments	Get monad's document collection.
11	LSAMonInterpretBasisVector	Interpret the a specified basis vector.
12	LSAMonMakeDocumentTermMatrix	Make the document-term matrix.
13	LSAMonMakeGraph	Make a graph of the document-term, document-document, or term-term relationships.
14	LSAMonMakeTopicsTable	Make a table of topics.
15	LSAMonRepresentByTerms	Find the terms representation of a matrix or a document.
16	LSAMonRepresentByTopics	Find the topics representation of a matrix or a document.
17	LSAMonRepresentDocumentTagsByTopics	Find the topic representation corresponding to a list of tags. Each monad document is expected to have a tag. One tag might correspond to multiple documents.

Monad elements

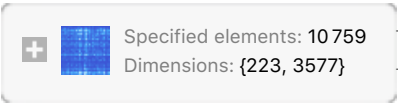
In this section we show that LSAMon has all of the properties listed in the previous section.

The monad head

The monad head is LSAMon. Anything wrapped in LSAMon can serve as monad's pipeline value. It is better though to use the constructor LSAMonUnit. (Which adheres to the definition in [Wk1].)

```
In[167]:= LSAMon[textHamlet, <| |>] ==> LSAMonMakeDocumentTermMatrix[Automatic, Automatic] ==> LSAMonEchoFunctionContext[Short];

» <| documents -> <| id.0001 -> 1604, id.0002 -> THE TRAGEDY OF HAMLET, PRINCE OF DENMARK, <<220>>, id.0223 -> THE END |>,

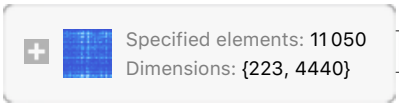
documentTermMatrix -> SparseArray[], <<1>>, ... -> <<1>>, stemmingRules -> Automatic |>
```

Lifting data to the monad

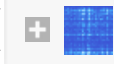
The function lifting the data into the monad QRMon is QRMonUnit.

The lifting to the monad marks the beginning of the monadic pipeline. It can be done with data or without data. Examples follow.

```
In[168]:= LSAMonUnit[textHamlet] ==> LSAMonMakeDocumentTermMatrix ==> LSAMonTakeDocumentTermMatrix

Out[168]:= SparseArray[
```

```
In[169]:= LSAMonUnit[] => LSAMonSetDocuments[textHamlet] => LSAMonMakeDocumentTermMatrix => LSAMonTakeDocumentTermMatrix

Out[169]:= SparseArray[ Specified elements: 11050
Dimensions: {223, 4440}]
```

(See the sub-section “Setters, droppers, and takers” for more details of setting and taking values in LSAMon contexts.)

Currently the monad can deal with data in the following forms:

- vectors of strings,
- associations with string values.

Generally, WL makes it easy to extract columns datasets order to obtain vectors or matrices, so datasets are not currently supported in LSAMon.

Making of the document-term matrix

As it was mentioned above with “term” we mean “a word or a stemmed word”. Here is are examples of stemmed words.

```
In[170]:= WordData[#, "PorterStem"] & /@ {"consequential", "constitution", "forcing", ""}

Out[170]:= {consequenti, constitut, forc, }
```

The fundamental model of LSAMon is the so called Vector space model (or the closely related Bag-of-words model.)
The document-term matrix is a linear vector space representation of the documents collection. That representation is further used in LSAMon to find topics and statistical thesauri.

Here is an example of ad hoc construction of a document-term matrix using a couple of paragraphs from “Hamlet”.

```
In[171]:= inds = {10, 19};
aTempText = AssociationThread[inds, textHamlet[[inds]]]

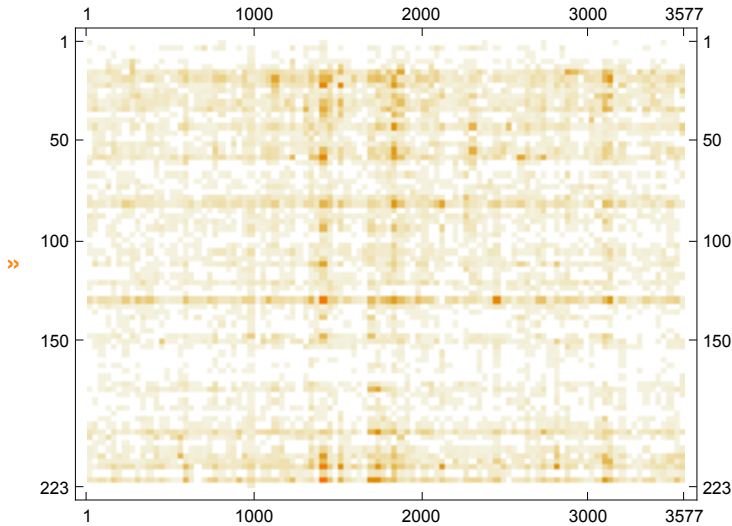
Out[172]:= <| 10 -> ACT I. Scene I. Elsinore. A platform before the Castle., 19 -> Scene II. Elsinore. A room of state in the Castle. |>

In[173]:= MatrixForm@CrossTabulate[Flatten[KeyValueMap[Thread[{#1, #2}] &, TextWords /@ ToLowerCase[aTempText]], 1]]

Out[173]//MatrixForm=
(
| a act before castle elsinore i. ii in of platform room scene state the |
10 | 1 1 1 1 1 2 0 0 0 1 0 1 0 1 |
19 | 1 0 0 1 1 0 1 1 1 0 1 1 1 1 |
)
```

When we construct the document-term matrix we (often) want to stem the words and (almost always) want to remove stop words. LSAMon’s function LSAMonMakeDocumentTermMatrix makes the document-term matrix and takes specifications for stemming and stop words.

```
In[174]:= lsaObj =
LSAMonUnit[textHamlet] =>
LSAMonMakeDocumentTermMatrix["StemmingRules" -> Automatic, "StopWords" -> Automatic] =>
LSAMonEchoFunctionContext[MatrixPlot[#documentTermMatrix] &] =>
LSAMonEchoFunctionContext[TakeLargest[ColumnSumsAssociation[#documentTermMatrix], 12] &];
```



```
>> <| ham -> 359, lord -> 232, king -> 204, come -> 138, queen -> 121,
shall -> 114, hamlet -> 112, good -> 111, let -> 109, hor -> 109, thou -> 107, like -> 89 |>
```

We can retrieve the stop words used in a monad with the function LSAMonTakeStopWords.

```
In[175]:= Short[lsaObj => LSAMonTakeStopWords]

Out[175]//Short= {a, about, above, across, add-on, after, again, against, all, almost, alone,
<<283>>, yet, you, you'd, you'll, your, you're, yours, yourself, yourselves, you've}
```

We can retrieve the stemming rules used in a monad with the function LSAMonTakeStemmingRules.

```
In[176]:= Short[lsaObj => LSAMonTakeStemmingRules]

Out[176]//Short= Automatic
```

The specification Automatic for stemming rules uses WordData[#, "PorterStem"]&.

Instead of the options style signature we can use positional signature.

```
Options style: LSAMonMakeDocumentTermMatrix[StemmingRules -> {}, StopWords -> Automatic]
Positional style: LSAMonMakeDocumentTermMatrix[{}, Automatic]
```

LSI weight functions

After making the document-term matrix we will most likely apply LSI weight functions, [Wk2], like “GFIDF” and “TF-IDF”. (This follows the “standard” approach used in search engines for calculating weights for document-term matrices; see [MB1].)

Frequency matrix

We use the following definition of the frequency document-term matrix F .

Each entry f_{ij} of the matrix F is the number of occurrences of the term j in the document i .

Weights

Each entry of the weighted document-term matrix M derived from the frequency document-term matrix F is expressed with the formula

$$m_{ij} = g_j l_{ij} d_i,$$

where

g_j -- global term weight;

l_{ij} -- local term weight;

d_i -- normalization weight.

Various formulas exist for these weights and one of the challenges is to find the right combination of them when using different document collections.

Here is table of weight functions formulas.

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

Computation specifications

LSAMon function `LSAMonApplyTermWeightFunctions` delegates the LSI weight functions application to the package "DocumentTermMatrixConstruction.m", [Aap4].

Here is an example.

```
In[182]:= lsaHamlet = LSAMonUnit[textHamlet] ==> LSAMonMakeDocumentTermMatrix;

In[183]:= wmat =
  lsaHamlet ==>
    LSAMonApplyTermWeightFunctions["IDF", "TermFrequency", "Cosine"] ==>
    LSAMonTakeWeightedDocumentTermMatrix;

In[184]:= TakeLargest[ColumnSumsAssociation[wmat], 6]
Out[184]:= <| enter -> 21.7403, ham -> 9.8253, hamlet -> 9.16829, polonius -> 8.79678, scene -> 8.56075, king -> 8.1436 |>
```

Instead of using the positional signature of `LSAMonApplyTermWeightFunctions` we can specify the LSI functions using options.

```
In[393]:= wmat2 =
  lsaHamlet ==>
    LSAMonApplyTermWeightFunctions["GlobalWeightFunction" -> "IDF",
    "LocalWeightFunction" -> "TermFrequency", "NormalizerFunction" -> "Cosine"] ==>
    LSAMonTakeWeightedDocumentTermMatrix;

In[394]:= TakeLargest[ColumnSumsAssociation[wmat2], 6]
Out[394]:= <| hamlet -> 10.4925, poloniu -> 9.74654, king -> 9.28721, elsinor -> 9.07375, queen -> 7.86772, lord -> 7.74499 |>
```

Here we are summaries of the non-zero values of the weighted document-term matrix derived with different combinations of global, local, and normalization weight functions.

<div><div>1 IDF, Binary, Cosine</div><div>Min 0.0150041 1st Qu 0.0573778 Median 0.0760402 Mean 0.102338 3rd Qu 0.107593 Max 1.</div></div>	<div><div>1 IDF, None, RowStochastic</div><div>Min 0.000811524 1st Qu 0.00332414 Median 0.00613544 3rd Qu 0.0114367 Mean 0.020181 Max 1.</div></div>	<div><div>1 GFIDF, None, None</div><div>1st Qu 1. Median 1. Min 1. 3rd Qu 1.5 Mean 2.24491 Max 237.5</div></div>	<div><div>1 Binary, None, Cosine</div><div>Min 0.0181339 1st Qu 0.0342393 Median 0.057735 3rd Qu 0.090167 Mean 0.091407 Max 1.</div></div>	<div><div>1 None, Log, RowStochastic</div><div>Min 0.00204538 1st Qu 0.00362849 Median 0.00655579 3rd Qu 0.0113535 Mean 0.020181 Max 1.</div></div>	<div><div>1 ColumnStochastic, Log, None</div><div>Min 0.00193617 1st Qu 0.0462098 Median 0.173287 Mean 0.271695 3rd Qu 0.549306 Max 0.693147</div></div>
<div><div>1 IDF, Binary, None</div><div>Min 1.15868 1st Qu 2.84222 Mean 3.56758 Median 3.79773 3rd Qu 4.71402 Max 4.71402</div></div>	<div><div>1 GFIDF, Binary, Cosine</div><div>Min 0.0336221 1st Qu 0.048543 Median 0.0642267 3rd Qu 0.093449 Mean 0.0965935 Max 1.</div></div>	<div><div>1 GFIDF, None, RowStochastic</div><div>Min 0.00082284 1st Qu 0.00205098 Median 0.00444174 3rd Qu 0.00933166 Mean 0.020181 Max 1.</div></div>	<div><div>1 Binary, None, None</div><div>1st Qu 1 3rd Qu 1 Median 1 Min 1 Mean 1.32217 Max 27</div></div>	<div><div>1 None, None, Cosine</div><div>Min 0.0181339 1st Qu 0.0342393 Median 0.057735 3rd Qu 0.090167 Mean 0.091407 Max 1.</div></div>	<div><div>1 ColumnStochastic, Log, RowStochastic</div><div>Min 0.0000298676 1st Qu 0.00128507 Median 0.00425429 3rd Qu 0.0112668 Mean 0.020181 Max 1.</div></div>
<div><div>1 IDF, Binary, RowStochastic</div><div>Min 0.000762208 1st Qu 0.00373532 Median 0.0061038 3rd Qu 0.0115251 Mean 0.020181 Max 1.</div></div>	<div><div>1 GFIDF, Binary, None</div><div>1st Qu 1. Median 1. Min 1. Mean 1.32217 3rd Qu 1.33333 Max 13.</div></div>	<div><div>1 Binary, Binary, Cosine</div><div>Min 0.0494468 1st Qu 0.0614295 Median 0.0811107 3rd Qu 0.105409 Mean 0.106594 Max 1.</div></div>	<div><div>1 Binary, None, RowStochastic</div><div>Min 0.00168634 1st Qu 0.002849 Median 0.00595238 3rd Qu 0.0106383 Mean 0.020181 Max 1.</div></div>	<div><div>1 None, None, None</div><div>1st Qu 1 3rd Qu 1 Median 1 Min 1 Mean 1.32217 Max 27</div></div>	<div><div>1 ColumnStochastic, None, Cosine</div><div>Min 0.000411024 1st Qu 0.0124279 Median 0.0396686 Mean 0.0780288 3rd Qu 0.101774 Max 1.</div></div>
<div><div>1 IDF, Log, Cosine</div><div>Min 0.0164353 1st Qu 0.0560995 Median 0.0738618 Mean 0.101692 3rd Qu 0.107561 Max 1.</div></div>	<div><div>1 GFIDF, Binary, RowStochastic</div><div>Min 0.00198353 1st Qu 0.00338414 Median 0.00610167 3rd Qu 0.0105852 Mean 0.020181 Max 1.</div></div>	<div><div>1 Binary, Binary, None</div><div>1st Qu 1 3rd Qu 1 Max 1 Mean 1 Median 1 Min 1</div></div>	<div><div>1 None, Binary, Cosine</div><div>Min 0.0494468 1st Qu 0.0614295 Median 0.0811107 3rd Qu 0.105409 Mean 0.106594 Max 1.</div></div>	<div><div>1 None, None, RowStochastic</div><div>Min 0.00168634 1st Qu 0.002849 Median 0.00595238 3rd Qu 0.0106383 Mean 0.020181 Max 1.</div></div>	<div><div>1 ColumnStochastic, None, None</div><div>Min 0.0027933 1st Qu 0.0769231 Median 0.25 Mean 0.40181 3rd Qu 1. Max 1.</div></div>
Out[186]= <div><div>1 IDF, Log, None</div><div>Min 0.803133 1st Qu 2.15194 Mean 2.723 Median 2.78706 3rd Qu 3.26751 Max 12.9073</div></div>	<div><div>1 GFIDF, Log, Cosine</div><div>Min 0.0132742 1st Qu 0.0262098 Median 0.0436552 3rd Qu 0.0748374 Mean 0.0810793 Max 1.</div></div>	<div><div>1 Binary, Binary, RowStochastic</div><div>Min 0.00244499 1st Qu 0.00377358 Median 0.00657895 3rd Qu 0.0111111 Mean 0.020181 Max 1.</div></div>	<div><div>1 None, Binary, None</div><div>1st Qu 1 3rd Qu 1 Max 1 Mean 1 Median 1 Min 1</div></div>	<div><div>1 ColumnStochastic, Binary, Cosine</div><div>Min 0.000246694 1st Qu 0.0104195 Median 0.036084 Mean 0.0766439 3rd Qu 0.101229 Max 1.</div></div>	<div><div>1 ColumnStochastic, None, RowStochastic</div><div>Min 0.0000294977 1st Qu 0.00135327 Median 0.0042479 3rd Qu 0.0111001 Mean 0.020181 Max 1.</div></div>
<div><div>1 IDF, Log, RowStochastic</div><div>Min 0.000952831 1st Qu 0.00360635 Median 0.00633097 3rd Qu 0.0111849 Mean 0.020181 Max 1.</div></div>	<div><div>1 GFIDF, Log, None</div><div>1st Qu 0.693147 Median 0.693147 Min 0.693147 3rd Qu 1.03972 Mean 1.15397 Max 37.4467</div></div>	<div><div>1 Binary, Log, Cosine</div><div>Min 0.0386884 1st Qu 0.0502109 Median 0.0752375 Mean 0.102772 3rd Qu 0.102919 Max 1.</div></div>	<div><div>1 None, Binary, RowStochastic</div><div>Min 0.00244499 1st Qu 0.00377358 Median 0.00657895 3rd Qu 0.0111111 Mean 0.020181 Max 1.</div></div>	<div><div>1 ColumnStochastic, Binary, None</div><div>Min 0.0027933 1st Qu 0.0625 Median 0.2 Mean 0.381212 3rd Qu 0.5 Max 1.</div></div>	
<div><div>1 IDF, None, Cosine</div><div>Min 0.0115357 1st Qu 0.0440683 Median 0.065596 Mean 0.0958 3rd Qu 0.0993633 Max 1.</div></div>	<div><div>1 GFIDF, Log, RowStochastic</div><div>Min 0.00144179 1st Qu 0.00270264 Median 0.00517701 3rd Qu 0.0099619 Mean 0.020181 Max 1.</div></div>	<div><div>1 Binary, Log, None</div><div>Log[2] 9355 Log[3] 1031 Log[4] 323 Log[5] 127 Log[6] 76 Log[7] 40 (Other) 98</div></div>	<div><div>1 None, Log, Cosine</div><div>Min 0.0386884 1st Qu 0.0502109 Median 0.0752375 Mean 0.102772 3rd Qu 0.102919 Max 1.</div></div>	<div><div>1 ColumnStochastic, Binary, RowStochastic</div><div>Min 0.0000163371 1st Qu 0.00115157 Median 0.00419756 3rd Qu 0.0112038 Mean 0.020181 Max 1.</div></div>	
<div><div>1 IDF, None, None</div><div>Min 1.15868 1st Qu 3.20995 Median 4.02088 Mean 4.36558 3rd Qu 4.71402 Max 81.8626</div></div>	<div><div>1 GFIDF, None, Cosine</div><div>Min 0.00335833 1st Qu 0.0113981 Median 0.026682 3rd Qu 0.0548158 Mean 0.06707 Max 1.</div></div>	<div><div>1 Binary, Log, RowStochastic</div><div>Min 0.00204538 1st Qu 0.00362849 Median 0.00655579 3rd Qu 0.0113535 Mean 0.020181 Max 1.</div></div>	<div><div>1 None, Log, None</div><div>Log[2] 9355 Log[3] 1031 Log[4] 323 Log[5] 127 Log[6] 76 Log[7] 40 (Other) 98</div></div>	<div><div>1 ColumnStochastic, Log, Cosine</div><div>Min 0.000413022 1st Qu 0.0115956 Median 0.0380513 Mean 0.0775075 3rd Qu 0.102593 Max 1.</div></div>	

Extracting topics

Streamlining topic extraction is one of the main reasons LSAMon was implemented. The topic extraction correspond to the so called “syntagmatic” relationships between the terms, [MS1].

Theoretical outline

The original weighed document-term matrix M is decomposed into the matrix factors W and H .

$$M \approx W \cdot H, \quad W \in \mathbb{R}^{k \times m}, \quad H \in \mathbb{R}^{k \times n}.$$

The i -th row of M is expressed with the i -th row of W multiplied by H .

The rows of H are the topics. SVD produces orthogonal topics; NMF does not.

The i -the document of the collection corresponds to the i -th row W . Finding the Nearest Neighbors (NN’s) of the i -th document using the rows similarity of the matrix W gives document NN’s through topic similarity.

The terms correspond to the columns of H . Finding NN’s based on similarities of H ’s columns produces statistical thesaurus entries.

The term groups provided by H ’s rows correspond to “syntagmatic” relationships. Using similarities of H ’s columns we can produce term clusters that correspond to “paradigmatic” relationships.

Computation specifications

Here is an example using the play “Hamlet” in which we specify additional stop words.

```
In[188]:= stopWords2 = {"enter", "exit", "[exit", "ham", "hor", "laer", "pol", "oph", "thy", "thee", "act", "scene"};

In[400]:= SeedRandom[2381]
lsaHamlet =
LSAMonUnit[textHamlet] ==>
LSAMonMakeDocumentTermMatrix["StemmingRules" -> Automatic, "StopWords" -> Join[stopWords, stopWords2]] ==>
LSAMonApplyTermWeightFunctions["GlobalWeightFunction" -> "IDF",
  "LocalWeightFunction" -> "None", "NormalizerFunction" -> "Cosine"] ==>
LSAMonExtractTopics["NumberOfTopics" -> 12, "MinNumberOfDocumentsPerTerm" -> 10, Method -> "NNMF", "MaxSteps" -> 20] ==>
LSAMonEchoTopicsTable["NumberOfTableColumns" -> 6, "NumberOfTerms" -> 10];
```

» topics table:

1	1.000 ham	3	1.000 room	5	1.000 daughter	7	1.000 oph	9	1.000 rosenkrantz	11	1.000 player
	0.860 lord		0.579 castl		0.763 pass		0.718 mark		0.840 guildenstern		0.098 enter
	0.728 ro		0.375 scene		0.642 love		0.571 nai		0.322 enter		0.072 hamlet
	0.492 pol		0.279 elsinor		0.615 fair		0.531 ophelia		0.147 king		0.049 plai
	0.299 king		0.115 act		0.429 pol		0.501 prai		0.104 queen		0.044 welcom
	0.268 sir		0.105 state		0.176 follow		0.427 queen		0.048 ophelia		0.036 tell
	0.234 oph		0.012 poloniu		0.115 old		0.225 sweet		0.016 lord		0.035 present
	0.191 bring		0.002 hamlet		0.096 right		0.204 ladi		0.013 thank		0.034 hear
	0.176 mother		0.002 hor		0.094 time		0.144 denmark		0.009 command		0.030 work
	0.161 good		0.002 life		0.082 doubt		0.130 sing		0.008 fit		0.029 ham
2	1.000 laer	4	1.000 elsinor	6	1.000 hor	8	1.000 laert	10	1.000 poloniu	12	1.000 enter
	0.506 king		0.917 scene		0.658 ham		0.371 enter		0.400 enter		0.576 hamlet
	0.472 thy		0.632 castl		0.265 thee		0.229 ophelia		0.096 queen		0.369 ghost
	0.425 father		0.194 act		0.248 lord		0.201 king		0.067 king		0.348 horatio
	0.375 shall		0.169 near		0.237 thou		0.076 attend		0.033 ophelia		0.124 ophelia
	0.365 let		0.029 room		0.237 speak		0.061 lord		0.022 denmark		0.117 king
	0.346 thou		0.010 welcom		0.201 horatio		0.038 queen		0.019 daughter		0.074 queen
	0.324 queen		0.006 speech		0.197 let		0.031 laer		0.016 mother		0.048 attend
	0.316 come		0.005 thee		0.197 night		0.026 gertrud		0.015 act		0.039 gentleman
	0.301 good		0.005 plai		0.190 sir		0.021 denmark		0.012 room		0.038 lord

Here is an example using the USA presidents “state of union” speeches.


```
In[411]:= SeedRandom[7681]
lsaSpeeches =
LSAMonUnit[aStateOfUnionSpeeches] ==>
LSAMonMakeDocumentTermMatrix["StemmingRules" -> Automatic, "StopWords" -> Automatic] ==>
LSAMonApplyTermWeightFunctions[
  "GlobalWeightFunction" -> "IDF", "LocalWeightFunction" -> "None", "NormalizerFunction" -> "Cosine"] ==>
LSAMonExtractTopics["NumberOfTopics" -> 36, "MinNumberOfDocumentsPerTerm" -> 40, Method -> "NNMF", "MaxSteps" -> 12] ==>
LSAMonEchoTopicsTable["NumberOfTableColumns" -> 6, "NumberOfTerms" -> 10];
```

» topics table:

1 1.000 tonight 0.816 tranquil 0.748 indian 0.685 insurrect 0.639 murder 0.488 drug 0.450 review 0.398 subsist 0.360 emperor 0.353 matur	7 1.000 spain 0.395 savag 0.378 florida 0.361 presum 0.360 likewis 0.334 articl 0.319 provinc 0.290 ratifi 0.233 minist 0.221 majesti	13 1.000 arbitr 0.878 cuba 0.851 award 0.663 majesti 0.635 island 0.611 exposit 0.571 convent 0.553 canal 0.505 consular 0.497 spain	19 1.000 coloni 0.767 slave 0.729 emigr 0.713 lake 0.597 tribe 0.583 commenc 0.572 indian 0.552 slaveri 0.548 pension 0.543 000	25 1.000 tonight 0.469 budget 0.379 let 0.343 america 0.332 billion 0.320 dream 0.315 todai 0.305 goal 0.298 program 0.274 help	31 1.000 soviet 0.190 oil 0.186 help 0.172 korea 0.164 aggress 0.155 challeng 0.155 threat 0.150 nuclear 0.145 weapon 0.126 world
2 1.000 gold 0.742 silver 0.486 circul 0.451 note 0.406 currenc 0.307 coin 0.295 bond 0.269 000 0.257 metal 0.242 speci	8 1.000 todai 0.796 group 0.790 dictat 0.783 slave 0.765 uniti 0.675 democrat 0.605 schedul 0.599 slaveri 0.546 program 0.538 ahead	14 1.000 terror 0.536 cruiser 0.400 hundr 0.390 america 0.382 weapon 0.342 fifti 0.317 tonight 0.293 fight 0.286 ideal 0.283 group	20 1.000 bank 0.529 slaveri 0.478 speci 0.428 currenc 0.351 slave 0.290 california 0.281 disburs 0.265 note 0.261 treasuri 0.237 territori	26 1.000 enemi 0.866 savag 0.720 british 0.705 militia 0.483 command 0.432 lake 0.413 victori 0.326 prison 0.295 confin 0.263 captur	32 1.000 democraci 0.308 recoveri 0.243 problem 0.193 modern 0.176 democrat 0.162 religion 0.156 unemploy 0.140 interpret 0.139 billion 0.136 specul
3 1.000 000 0.416 cent 0.412 method 0.408 veteran 0.401 agricultur 0.376 consolid 0.317 board 0.307 court 0.289 flood 0.281 bureau	9 1.000 tonight 0.823 drug 0.709 job 0.679 parent 0.668 help 0.650 child 0.619 school 0.580 children 0.561 challeng 0.541 colleg	15 1.000 mexico 0.897 texa 0.522 mexican 0.253 california 0.228 territori 0.211 steamer 0.166 articl 0.153 levi 0.150 postag 0.147 minist	21 1.000 persuad 0.976 militia 0.884 expedi 0.806 pleas 0.748 effectu 0.745 requisit 0.739 legislatur 0.738 uniform 0.682 paper 0.664 bless	27 1.000 tribe 0.706 indian 0.605 provis 0.592 hitherto 0.569 presum 0.551 tend 0.549 lessen 0.541 calcul 0.522 besid 0.512 ensu	33 1.000 job 0.508 colleg 0.411 tonight 0.395 student 0.376 clean 0.369 panama 0.355 compani 0.344 oil 0.315 recoveri 0.268 crisi
4 1.000 minist 0.990 french 0.726 franc 0.564 pari 0.544 chamber 0.503 british 0.502 council 0.440 instruct 0.413 belliger 0.402 king	10 1.000 militia 0.885 harbor 0.844 port 0.802 vessel 0.718 amiti 0.688 instruct 0.634 legislatur 0.625 boat 0.616 shall 0.608 offend	16 1.000 goal 0.711 inflat 0.539 louisiana 0.489 acquisit 0.466 achiev 0.442 todai 0.412 level 0.391 ohio 0.351 sensibl 0.334 help	22 1.000 german 0.898 fight 0.666 air 0.661 germani 0.658 win 0.596 enemi 0.557 victori 0.511 alli 0.406 job 0.399 partnership	28 1.000 program 0.534 billion 0.496 percent 0.478 oil 0.457 job 0.450 spend 0.427 nuclear 0.409 inflat 0.376 technolog 0.367 help	34 1.000 recoveri 0.911 program 0.684 farm 0.554 econom 0.511 group 0.466 unemploy 0.466 relief 0.431 bank 0.422 democrat 0.418 veteran
5 1.000 likewis 0.682 unsettl 0.615 augment 0.586 coloni 0.486 suppress 0.437 vessel 0.433 spain 0.417 sentiment 0.412 vest 0.406 presum	11 1.000 territori 0.720 jurisdict 0.703 admiss 0.686 constitut 0.655 represent 0.652 book 0.651 agit 0.618 nicaragua 0.616 britain 0.598 assert	17 1.000 forest 0.868 corpor 0.616 man 0.567 island 0.452 supervis 0.417 type 0.417 railroad 0.355 mere 0.340 moreov 0.334 alaska	23 1.000 herewith 0.904 territori 0.876 respectfulli 0.837 california 0.816 panama 0.655 grade 0.648 certif 0.609 depart 0.603 subject 0.591 consular	29 1.000 bank 0.667 deposit 0.427 surplu 0.427 currenc 0.406 distribut 0.377 specul 0.370 paper 0.356 circul 0.331 note 0.312 ration	35 1.000 job 0.734 cut 0.477 deficit 0.358 spend 0.280 percent 0.274 tonight 0.267 lot 0.258 tell 0.256 program 0.245 invest
6 1.000 railwai 0.480 concert 0.472 railroad 0.360 commiss 0.358 concili 0.341 arbitr 0.324 postpon 0.316 thorough 0.304 hesit 0.303 eight	12 1.000 domain 0.884 debat 0.842 thought 0.780 readi 0.778 ship 0.777 spent 0.764 task 0.735 counsel 0.733 alter 0.706 speak	18 1.000 program 0.279 feder 0.270 budget 0.249 problem 0.233 assist 0.232 studi 0.226 mobil 0.220 highwai 0.208 korea 0.202 employe	24 1.000 tariff 0.577 manufactur 0.487 bond 0.448 taxat 0.423 scheme 0.415 articl 0.392 cent 0.357 price 0.350 treasuri 0.321 consum	30 1.000 commission 0.562 captur 0.551 majesti 0.498 articl 0.458 award 0.395 treati 0.361 appoint 0.354 sick 0.342 damag 0.313 adjourn	36 1.000 silver 0.381 coin 0.373 indian 0.349 pension 0.290 fiscal 0.287 cent 0.265 june 0.247 citizenship 0.243 method 0.234 statut

Note that in both examples:

- 1. stemming is used when creating the document-term matrix,
 - 2. the default LSI re-weighting functions are used: “IDF”, “None”, “Cosine”,
 - 3. the dimension reduction algorithm NNMF is used.
- Things to keep in mind.
- 0. The interpretability provided by NNMF comes at a price.
 - 0. NNMF is prone to get stuck into local minima, so several topic extractions (and corresponding evaluations) have to be done.
 - 0. We would get different results with different NNMF runs with the same parameters. (NNMF uses random numbers initialization.)
 - 0. The NNMF topic vectors are not orthogonal.
 - 0. SVD is much faster than NNMF, but it topic vectors are hard to interpret.
 - 0. Generally, the topics derived with SVD are stable, do not change with different runs with the same parameters.
 - 0. The SVD topics vectors are orthogonal, which provides for quick to find representations of documents not in the monad’s document collection.

Extracting statistical thesauri

The statistical thesaurus extraction corresponds to the “paradigmatic” relationships between the terms, [MS1].

```
In[410]:= lsaSpeeches=>
  LSAMonExtractStatisticalThesaurus[
    "Words" -> Map[WordData[#, "PorterStem"] &, {"bank", "war", "economy", "school", "port", "health", "enemy", "nuclear"}],
    "NumberOfNearestNeighbors" -> 12] =>
  LSAMonEchoStatisticalThesaurus;
```

» statistical thesaurus:

word	statistical thesaurus
bank	{bank, deposit, french, currenc, chamber, specul, charter, instal, paper, king, exchang, credit}
economi	{economi, respons, share, higher, earn, creat, launch, home, build, walk, speed, strengthen}
enemi	{enemi, victori, british, command, lake, savag, prison, superior, volunt, warfar, confin, captur}
health	{health, coverag, invest, economi, care, lot, stai, expand, reward, educ, work, sure}
nuclear	{nuclear, strateg, ensur, energi, decad, technolog, basic, proud, inflat, middl, tax, prioriti}
port	{port, vessel, accommod, extent, foot, fortif, commenc, revenu, intercour, coast, happi, sum}
school	{school, teacher, child, children, famili, centuri, thank, expand, parent, technolog, poverti, educ}
war	{war, forc, peac, right, great, power, good, mean, provid, success, continu, hope}

Mapping queries and documents to terms

One of the most natural operations is to find the representation of an arbitrary document (or sentence or a list of words) in monad’s Linear vector space of terms. This is done with the function LSAMonRepresentByTerms.

Here is an example in which a sentence is represented as a one-row matrix (in that space.)

```
In[192]:= obj =
  lsaHamlet=>
  LSAMonRepresentByTerms["Hamlet, Prince of Denmark killed the king."] =>
  LSAMonEchoValue;
```

» value: SparseArray[]

Here we display only the non-zero columns of that matrix.

```
In[193]:= obj=>LSAMonEchoFunctionValue[Function[...]];
» ( 1 | denmark hamlet kill king princ
    | 0.437157 0.251741 0.569795 0.20823 0.614405 )
```

Transformation steps

Assume that LSAMonRepresentByTerms is given a list of sentences. Then that function performs the following steps.

- 1. The sentence is split into a list of words.
- 2. If monad’s document-term matrix was made by removing stop words the same stop words are removed from the list of words.
- 3. If monad’s document-term matrix was made by stemming the same stemming rules are applied to the list of words.
- 4. The LSI global weights and the LSI local weight and normalizer functions are applied to sentence’s contingency matrix.

Equivalent representation

Let us look convince ourselves that documents used in the monad to built the weighted document-term matrix have the same representation as the corresponding rows of that matrix.

Here is an association of documents from monad’s document collection.

```
In[194]:= inds = {6, 10};
queries = Part[lsaHamlet=>LSAMonTakeDocuments, inds];
queries
```

```
Out[196]= < | id.0006 -> Getrude, Queen of Denmark, mother to Hamlet. Ophelia, daughter to Polonius.,
            id.0010 -> ACT I. Scene I. Elsinore. A platform before the Castle. | >
```

```
In[197]:= lsaHamlet=>
  LSAMonRepresentByTerms[queries] =>
  LSAMonEchoFunctionValue[Function[...]];
» ( | castl daughter denmark elsinor getrud hamlet mother ophelia platform poloniu queen
    | id.0006 | 0. 0.404674 0.331177 0. 0.633921 0.190712 0.294272 0.294272 0. 0.283923 0.185821
    | id.0010 | 0.497202 0. 0. 0.420682 0. 0. 0. 0. 0.758826 0. 0. )
```

```
In[198]:= lsaHamlet =>
  LSAMonEchoFunctionContext [Function[...], +];
```

	castl	daughter	denmark	elsinor	getrud	hamlet	mother	ophelia	platform	poloniu	queen
id.0006	0.	0.404674	0.331177	0.	0.633921	0.190712	0.294272	0.294272	0.	0.283923	0.185821
id.0010	0.497202	0.	0.	0.420682	0.	0.	0.	0.	0.758826	0.	0.

Mapping queries and documents to topics

Another natural operation is to find the representation of an arbitrary document (or a list of words) in monad's Linear vector space of topics. This is done with the function `LSAMonRepresentByTopics`.

Here is an example.

```
In[395]:= inds = {6, 10};
queries = Part[lsaHamlet ==> LSAMonTakeDocuments, inds];
Short /@ queries
```

```
Out[397]:= <| id.0006 -> Getrude, Queen of Denmark, mother to Hamlet. Ophelia, daughter to Polonius.,
id.0010 -> ACT I. Scene I. Elsinore. A platform before the Castle. |>
```

```
In[398]:= lsaHamlet =>
  LSAMonRepresentByTopics[queries] ==>
  LSAMonEchoFunctionValue [Function[...], +];
```

	castl-elsinor-room	thou-heaven-come	ro-lord-sir	laert-ophelia-attend	room-poloniu-elsinor	mark-nai-queen	ghost-father-ham
id.0006	-0.0271462	0.00316746	0.00451831	0.0413948	0.0683847	0.062368	0.0380085
id.0010	0.485099	-0.00245866	-0.0000512204	-0.000950562	-0.0566728	0.00040852	-0.000297085

```
In[399]:= lsaHamlet =>
  LSAMonEchoFunctionContext [Function[...], +];
```

	castl-elsinor-room
id.0006	0.
id.0010	0.27764

Theory

In order to clarify what the function `LSAMonRepresentByTopics` is doing let us go through the formulas it is based on.

The original weighed document-term matrix M is decomposed into the matrix factors W and H .

$$M \approx W \cdot H, \quad W \in \mathbb{R}^{k \times m}, \quad H \in \mathbb{R}^{k \times n}.$$

The i -th row of M is expressed with the i -th row of W multiplied by H .

$$m_i \approx w_i \cdot H$$

For a query vector $q_0 \in \mathbb{R}^m$ we want to find its topics representation vector $x \in \mathbb{R}^k$:

$$q_0 \approx x \cdot H$$

Denote with $H^{(-1)}$ the inverse or pseudo-inverse matrix of H . We have:

$$q_0 \cdot H^{(-1)} \approx (x \cdot H) \cdot H^{(-1)} = x \cdot (H \cdot H^{(-1)}) = x \cdot I,$$

$$x \in \mathbb{R}^k,$$

$$H^{(-1)} \in \mathbb{R}^{k \times n},$$

$$I \in \mathbb{R}^{k \times k}.$$

In `LSAMon` for SVD $H^{(-1)} = H^T$; for NMF is $H^{(-1)}$ is the pseudo-inverse of H .

The vector x obtained with `LSAMonRepresentByTopics`.

Finding the most important documents

There are several algorithms we can apply for finding the most important documents in the collection. `LSAMon` utilizes two types algorithms: (1) graph centrality measures based, and (2) matrix factorization based. With certain graph centrality measures the two algorithms are equivalent. In this subsection we demonstrate the matrix factorization algorithm (that uses SVD.)

Definition: The most important sentences have the most important words and the most important words are in the most important sentences.

That definition can be used to derive an iterations-based model that can be expressed with SVD or eigenvector finding algorithms, [LE1].

Here we pick an important part of the play "Hamlet".

```
In[204]:= focusText = First@Pick[textHamlet, StringMatchQ[textHamlet, ___ ~~ "to be" ~~ __ ~~ "or not to be" ~~ ___, IgnoreCase -> True]];
Short[focusText]
```

```
Out[205]//Short= Ham. To be, or not to be- that is the question: Whether 'tis nobler in the mind to suff
... youth Blasted with ecstasy. O, woe is me T' have seen what I have seen, see what I see!
```

Here we find the top 3 most important sentences from that part.

```
In[414]:= LSAMonUnit[StringSplit[ToLowerCase[focusText], {"", ".", ";", "!", "?"}]] =>
  LSAMonMakeDocumentTermMatrix["StemmingRules" -> {}, "StopWords" -> Automatic] =>
  LSAMonApplyTermWeightFunctions =>
  LSAMonFindMostImportantDocuments[3] =>
  LSAMonEchoFunctionValue[GridTableForm];
```

#	1	2	3	4
1	1.	30	id.0030	and enterprises of great pith and moment with this regard their currents turn awry and lose the name of action
» 2	3.35433×10^{-8}	3	id.0003	or not to be- that is the question: whether 'tis nobler in the mind to suffer the slings and arrows of outrageous fortune or to take arms against a sea of troubles
3	4.95717×10^{-9}	81	id.0081	for the power of beauty will sooner transform honesty from what it is to a bawd than the force of honesty can translate beauty into his likeness


Setters, droppers, and takers

The values from the monad context can be set, obtained, or dropped with the corresponding “setter”, “dropper”, and “taker” functions as summarized in a previous section.

For example:

```
In[207]:= p =
  LSAMonUnit[textHamlet] => LSAMonMakeDocumentTermMatrix[Automatic, Automatic];
```

```
In[208]:= p => LSAMonTakeMatrix
```

```
Out[208]:= SparseArray[ Specified elements: 10759  
Dimensions: {223, 3577}]
```

If other values are put in the context they can be obtained through the (generic) function `LSAMonTakeContext`, [AAp1]:

```
In[209]:= Short@ (p => QRMonTakeContext) ["documents"]
```

```
Out[209]//Short= <| id.0001 -> 1604, id.0002 -> THE TRAGEDY OF HAMLET, PRINCE OF DENMARK, <<220>>, id.0223 -> THE END |>
```

Another generic function from [AAp1] is `LSAMonTakeValue` (used many times above.)

Here is an example of the “data dropper” `LSAMonDropDocuments`:

```
In[210]:= Keys[p => LSAMonDropDocuments => QRMonTakeContext]
```

```
Out[210]= {documentTermMatrix, terms, stopWords, stemmingRules}
```

(The “droppers” simply use the state monad function `LSAMonDropFromContext`, [AAp1]. For example, `LSAMonDropDocuments` is equivalent to `LSAMonDropFromContext[“documents”]`.)

Unit tests


The development of `LSAMon` was done with two types of unit tests: (i) directly specified tests, [AAp7], and (ii) tests based on randomly generated pipelines, [AA8].

The unit test package should be further extended in order to provide better coverage of the functionalities and illustrate -- and postulate -- pipeline behavior.

Directly specified tests

Here we run the unit tests file “MonadicLatentSemanticAnalysis-Unit-Tests.wlt”, [AAp8].

```
AbsoluteTiming[
  testObject = TestReport["~/MathematicaForPrediction/UnitTests/MonadicLatentSemanticAnalysis-Unit-Tests.wlt"]
]
```

```
Out[220]= {8.01681, TestReportObject[ Title: Test Report: MonadicLatentSemanticAnalysis-Unit-Tests.wlt  
Success rate: 100% Tests run: 25}]
```

The natural language derived test ID’s should give a fairly good idea of the functionalities covered in [AAp3].

```
In[221]:= Values[Map[#["TestID"] &, testObject["TestResults"]]]
```

```
Out[221]= {LoadPackage, USASpeechesData, HamletData, StopWords, Make-document-term-matrix-1, Make-document-term-matrix-2,
  Apply-term-weights-1, Apply-term-weights-2, Topic-extraction-1, Topic-extraction-2, Topic-extraction-3,
  Topic-extraction-4, Statistical-thesaurus-1, Topics-representation-1, Take-document-term-matrix-1,
  Take-weighted-document-term-matrix-1, Take-document-term-matrix-2, Take-weighted-document-term-matrix-2,
  Take-terms-1, Take-Factors-1, Take-Factors-2, Take-Factors-3, Take-Factors-4, Take-StopWords-1, Take-StemmingRules-1}
```

Random pipelines tests

Since the monad `LSAMon` is a DSL it is natural to test it with a large number of randomly generated “sentences” of that DSL. For the `LSAMon` DSL the sentences are `LSAMon` pipelines. The package “MonadicLatentSemanticAnalysisRandomPipelinesUnitTests.m”, [AAp9], has functions for generation of `QRMon` random pipelines and running them as verification tests. A short example follows.

Generate pipelines:

```
SeedRandom[234]
pipelines = MakeLSAMonRandomPipelines[100];
Length[pipelines]
```

Out[]:= 100

Here is a sample of the generated pipelines:

```
(*Block[{DoubleLongRightArrow,pipelines=RandomSample[pipelines,6]},Clear[DoubleLongRightArrow];
pipelines=pipelines/.{_TemporalData->"tsData",_?MatrixQ->"distData"};
GridTableForm[
  Map[List@ToString[DoubleLongRightArrow@##,FormatType->StandardForm]&,pipelines],TableHeadings->{"pipeline"}]
AutoCollapse[]*)
```

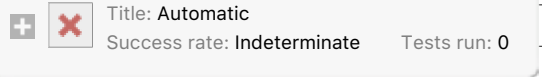
Here we run the pipelines as unit tests:

```
In[213]:= AbsoluteTiming[
  res = TestRunLSAMonPipelines[pipelines, "Echo" -> False];
]
```

Out[213]:= {3. × 10⁻⁶, Null}

From the test report results we see that a dozen tests failed with messages, all of the rest passed.

```
In[214]:= rpTRObj = TestReport[res]
```

Out[214]:= TestReportObject []

(The message failures, of course, have to be examined -- some bugs were found in that way. Currently the actual test messages are expected.)

Implementation notes

The implementation methodology of the LSAMon monad packages [AAp3, AAp9] followed the methodology created for the ClCon monad package [AAp10, AA6]. Similarly, this document closely follows the structure and exposition of the ClCon monad document “A monad for classification workflows”, [AA6]. A lot of the functionalities and signatures of LSAMon were designed and programed through considerations of natural language commands specifications given to a specialized conversational agent. (As discussed in the previous section.)

References

Packages

[AAp1] Anton Antonov, State monad code generator Mathematica package, (2017), MathematicaForPrediction at GitHub.
URL: <https://github.com/antononcube/MathematicaForPrediction/blob/master/MonadicProgramming/StateMonadCodeGenerator.m> .

[AAp2] Anton Antonov, Monadic tracing Mathematica package, (2017), MathematicaForPrediction at GitHub.
URL: <https://github.com/antononcube/MathematicaForPrediction/blob/master/MonadicProgramming/MonadicTracing.m> .

[AAp3] Anton Antonov, Monadic Latent Semantic Analysis Mathematica package, (2017), MathematicaForPrediction at GitHub.
URL: <https://github.com/antononcube/MathematicaForPrediction/blob/master/MonadicProgramming/MonadicLatentSemanticAnalysis.m> .

[AAp4] Anton Antonov, Implementation of document-term matrix construction and re-weighting functions in Mathematica, (2013), MathematicaForPrediction at GitHub.
URL: <https://github.com/antononcube/MathematicaForPrediction/blob/master/DocumentTermMatrixConstruction.m> .

[AAp5] Anton Antonov, Non-Negative Matrix Factorization algorithm implementation in Mathematica, (2013), MathematicaForPrediction at GitHub.
URL: <https://github.com/antononcube/MathematicaForPrediction/blob/master/NonNegativeMatrixFactorization.m> .

[AAp6] Anton Antonov, SSparseMatrix Mathematica package, (2018), MathematicaForPrediction at GitHub.
URL: <https://github.com/antononcube/MathematicaForPrediction/blob/master/SSparseMatrix.m> .

[AAp7] Anton Antonov, MathematicaForPrediction utilities, (2014), MathematicaForPrediction at GitHub.
URL: <https://github.com/antononcube/MathematicaForPrediction/blob/master/MathematicaForPredictionUtilities.m> .

[AAp8] Anton Antonov, Monadic Latent Semantic Analysis unit tests, (2019), MathematicaVsR at GitHub.
URL: <https://github.com/antononcube/MathematicaForPrediction/blob/master/UnitTests/MonadicLatentSemanticAnalysis-Unit-Tests.wlt> .

[AAp9] Anton Antonov, Monadic Latent Semantic Analysis random pipelines Mathematica unit tests, (2019), MathematicaForPrediction at GitHub.
URL: <https://github.com/antononcube/MathematicaForPrediction/blob/master/UnitTests/MonadicLatentSemanticAnalysisRandomPipelinesUnitTests.m> .

[AAp10] Anton Antonov, Monadic contextual classification Mathematica package, (2017), MathematicaForPrediction at GitHub.
URL: <https://github.com/antononcube/MathematicaForPrediction/blob/master/MonadicProgramming/MonadicContextualClassification.m> .

ConverationalAgents Packages

[AAp10] Anton Antonov, Time series workflows grammar in EBNF, (2018), ConversationalAgents at GitHub, <https://github.com/antononcube/Conversation->

alAgents.

[AAp11] Anton Antonov, QRMon translator Mathematica package,(2018), ConversationalAgents at GitHub, <https://github.com/antononcube/ConversationalAgents>.

MathematicaForPrediction articles

[AA1] Anton Antonov, "Monad code generation and extension", (2017), MathematicaForPrediction at GitHub, <https://github.com/antononcube/MathematicaForPrediction>.

[AA2] Anton Antonov, "Topic and thesaurus extraction from a document collection", (2013), MathematicaForPrediction at GitHub.

[AA3] Anton Antonov, "The Great conversation in USA presidential speeches", (2017), MathematicaForPrediction at WordPress.
URL: <https://mathematicaforprediction.wordpress.com/2017/12/24/the-great-conversation-in-usa-presidential-speeches> .

[AA4] Anton Antonov, "Contingency tables creation examples", (2016), MathematicaForPrediction at WordPress.

[AA5] Anton Antonov, "RSparseMatrix for sparse matrices with named rows and columns", (2015), MathematicaForPrediction at WordPress.

[AA6] Anton Antonov, "A monad for classification workflows", (2018), MathematicaForPrediction at WordPress.
URL: <https://mathematicaforprediction.wordpress.com/2018/05/15/a-monad-for-classification-workflows/> .

Other

[Wk1] Wikipedia entry, Monad,
URL: [https://en.wikipedia.org/wiki/Monad_\(functional_programming\)](https://en.wikipedia.org/wiki/Monad_(functional_programming)) .

[Wk2] Wikipedia entry, Latent semantic analysis,
URL: https://en.wikipedia.org/wiki/Latent_semantic_analysis .

[Wk3] Wikipedia entry, Distributional semantics,
URL: https://en.wikipedia.org/wiki/Distributional_semantics .

[Wk4] Wikipedia entry, Non-negative matrix factorization,
URL: https://en.wikipedia.org/wiki/Non-negative_matrix_factorization .

[LE1] Lars Elden, Matrix Methods in Data Mining and Pattern Recognition, 2007, SIAM. ISBN-13: 978-0898716269.

[MB1] Michael W. Berry & Murray Browne, Understanding Search Engines: Mathematical Modeling and Text Retrieval, 2nd. ed., 2005, SIAM. ISBN-13: 978-0898715811.

[MS1] Magnus Sahlgren, "The Distributional Hypothesis", (2008), Rivista di Linguistica. 20 (1): 33–53.

[PS1] Patrick Scheibe, Mathematica (Wolfram Language) support for IntelliJ IDEA, (2013-2018), Mathematica-IntelliJ-Plugin at GitHub.
URL: <https://github.com/halirutan/Mathematica-IntelliJ-Plugin> .