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 an 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 and 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.)

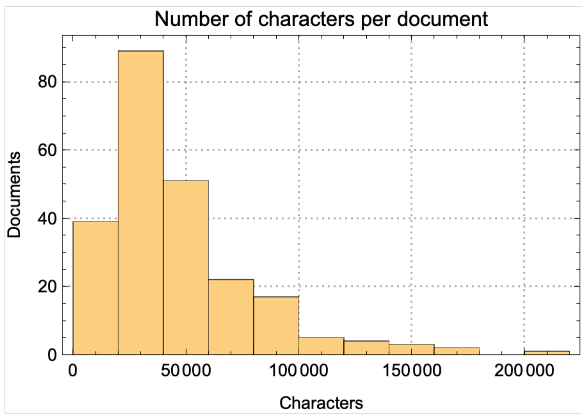
The monadic programming design is used as a Software Design Pattern. The `LSAMon` monad can be also seen as a Domain Specific Language (DSL) for the specification and programming of machine learning classification workflows.

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

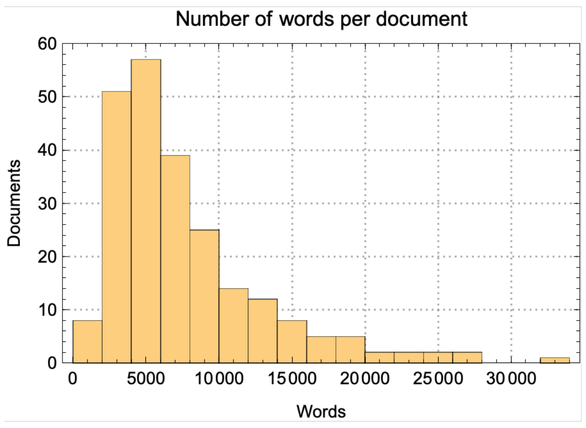
```
Out[*]=
LSAMonUnit[aStateOfUnionSpeeches] ⟹                                                   Uplift text data into the monad.
  LSAMonMakeDocumentTermMatrix ⟹                                                      Make the document-term contingency matrix.
  LSAMonEchoDocumentsStatistics ⟹                                                     Echo the documents colllection statistics.
  LSAMonApplyTermWeightFunctions["IDF", "None", "Cosine"] ⟹                           Apply LSI weight functions.
  LSAMonExtractTopics[24, "MaxSteps" → 20, Method → "NNMF"] ⟹                         Extract topics using NNMF.
  LSAMonEchoTopicsTable["NumberOfTableColumns" → 6] ⟹                                 Echo topics table.
  LSAMonExtractStatisticalThesaurus[{"vote", "bank", "health", "war"}, 12] ⟹          Extract statistical theasurus.
  LSAMonEchoStatisticalThesaurus                                                        Echo thesaurus entries.
```

» Context value "documents":

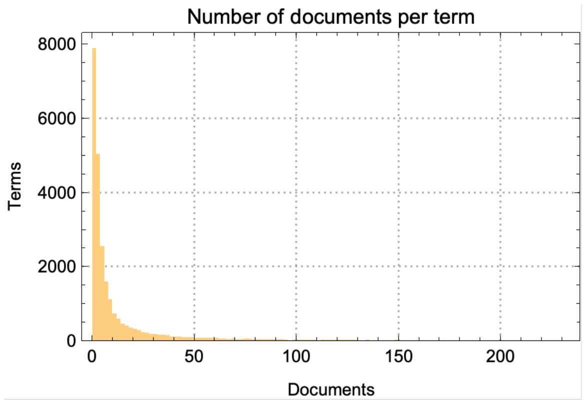
Number of documents:233



Number of unique words:60159



Document-term matrix dimensions:
{233, 25323}



Numbe
docum
summa
1 # d
1st
Min
Med
3rd
Mear
Max

» topics table:

1		5		9		13		17		21	
1.000	isthmus	1.000	jobs	1.000	interstate	1.000	program	1.000	fighting	1.000	terrorists
0.842	panama	0.916	tonight	0.741	processes	0.742	communist	0.986	japanese	0.983	iraq
0.481	granada	0.630	americans	0.731	railroads	0.582	economic	0.928	planes	0.757	iraqi
0.450	colombia	0.624	don	0.695	railways	0.570	atomic	0.728	soviet	0.675	terror
0.392	canal	0.502	budget	0.652	tasks	0.530	programs	0.619	production	0.557	terrorist
0.307	kansas	0.476	let	0.625	unrest	0.384	farm	0.528	nurses	0.468	afghanistan
0.246	transit	0.475	businesses	0.581	storage	0.369	federal	0.521	victory	0.446	qaeda
0.237	route	0.466	help	0.569	industrial	0.352	housing	0.475	allies	0.361	tonight
0.235	treaty	0.457	spending	0.460	matter	0.335	insurance	0.460	air	0.281	empower
0.216	revolution	0.436	college	0.458	presently	0.333	collective	0.440	conquest	0.269	fight
0.208	territory	0.418	cuts	0.448	floating	0.331	billion	0.435	korea	0.265	help
0.182	1856	0.402	deficit	0.444	thoughtful	0.324	budget	0.425	weapons	0.265	americans
2		6		10		14		18		22	
1.000	spain	1.000	tonight	1.000	kansas	1.000	gentlemen	1.000	enemy	1.000	interstate
0.558	colonies	0.969	iraq	0.901	constitution	0.564	militia	0.699	savages	0.866	corporations
0.490	likewise	0.633	jobs	0.811	mexico	0.425	objects	0.608	british	0.785	forest
0.398	presumed	0.575	terrorists	0.710	1858	0.414	indians	0.485	militia	0.584	islands
0.373	respecting	0.500	americans	0.682	slavery	0.412	burthens	0.440	lake	0.579	man
0.371	treaty	0.491	drug	0.619	whilst	0.371	requisite	0.355	whilst	0.506	standpoint
0.368	duties	0.485	businesses	0.580	duties	0.363	pleasing	0.330	captain	0.451	industrial
0.354	negotiation	0.449	seniors	0.563	california	0.350	legislature	0.321	regulars	0.450	philippine
0.354	object	0.419	infrastructure	0.527	territory	0.343	tribes	0.291	prisoners	0.430	conditions
0.354	tribes	0.417	marriage	0.512	subject	0.318	persuasion	0.285	command	0.423	type
0.352	fortifications	0.412	iraqi	0.488	slaves	0.298	impressions	0.282	proofs	0.422	cable
0.349	minister	0.380	regime	0.472	minister	0.298	papers	0.259	orders	0.421	philippines
3		7		11		15		19		23	
1.000	soviet	1.000	banks	1.000	silver	1.000	programs	1.000	consular	1.000	soviet
0.543	1980	0.928	recovery	0.802	gold	0.954	oil	0.803	award	0.798	communist
0.411	salt	0.832	000	0.540	notes	0.801	billion	0.795	convention	0.727	rulers
0.389	oil	0.704	banking	0.510	coinage	0.689	percent	0.755	diplomatic	0.718	tonight
0.372	nuclear	0.687	economic	0.488	cent	0.689	energy	0.728	chile	0.664	challenge
0.354	afghanistan	0.606	depression	0.447	circulation	0.515	strategic	0.701	treaty	0.596	atomic
0.331	inflation	0.525	veterans	0.420	currency	0.506	nuclear	0.690	boundary	0.574	21st
0.257	israel	0.520	agencies	0.417	1890	0.497	inflation	0.677	cuba	0.478	parents
0.247	tonight	0.519	unemployment	0.410	000	0.496	decrees	0.665	arbitration	0.433	child
0.242	commitment	0.511	corporation	0.408	1893	0.494	major	0.653	1883	0.421	children
0.223	strategic	0.469	relief	0.388	bonds	0.471	spending	0.638	majesty	0.397	communists
0.172	iran	0.441	reserve	0.333	coin	0.464	1980	0.625	international	0.376	drugs
4		8		12		16		20		24	
1.000	texas	1.000	democratic	1.000	gentlemen	1.000	vietnam	1.000	000	1.000	vessels
0.907	mexico	0.793	democracy	0.673	philadelphia	0.517	tonight	0.528	flood	0.938	harbors
0.468	annexation	0.748	economic	0.637	commissioners	0.289	billion	0.521	veterans	0.739	augmentation
0.424	paper	0.736	today	0.543	article	0.192	percent	0.477	cent	0.717	dispositions
0.416	specie	0.638	program	0.505	captures	0.169	try	0.465	1929	0.680	orleans
0.408	mexican	0.603	housing	0.422	damages	0.143	commitments	0.459	marketing	0.674	port
0.399	bank	0.523	farm	0.412	6th	0.135	communist	0.448	farmer	0.665	louisiana
0.387	banks	0.492	bargaining	0.403	majesty	0.133	americans	0.402	agriculture	0.621	lessen
0.362	notes	0.481	inflation	0.392	french	0.124	crime	0.371	aviation	0.568	ohio
0.343	1837	0.440	income	0.383	engagements	0.121	programs	0.357	court	0.555	ceded
0.341	currency	0.431	recovery	0.352	amity	0.119	help	0.354	league	0.554	mississippi
0.300	oregon	0.430	ideals	0.320	7th	0.107	propose	0.353	board	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", "Monad elements", and “The utilization of SSparseMatrix objects” 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, AAp11]:

In[∘]:=
```
Import["https://raw.githubusercontent.com/antononcube/MathematicaForPrediction/master/MonadicProgramming/
   MonadicLatentSemanticAnalysis.m"]
Import[
 "https://raw.githubusercontent.com/antononcube/MathematicaForPrediction/master/MonadicProgramming/MonadicTracing.m"]
Import["https://raw.githubusercontent.com/antononcube/MathematicaForPrediction/master/Misc/HeatmapPlot.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[∘]:=
```
textHamlet = ToString /@ Flatten[
    Import["https://raw.githubusercontent.com/antononcube/MathematicaVsR/master/Data/MathematicaVsR-Data-Hamlet.csv"]];
```

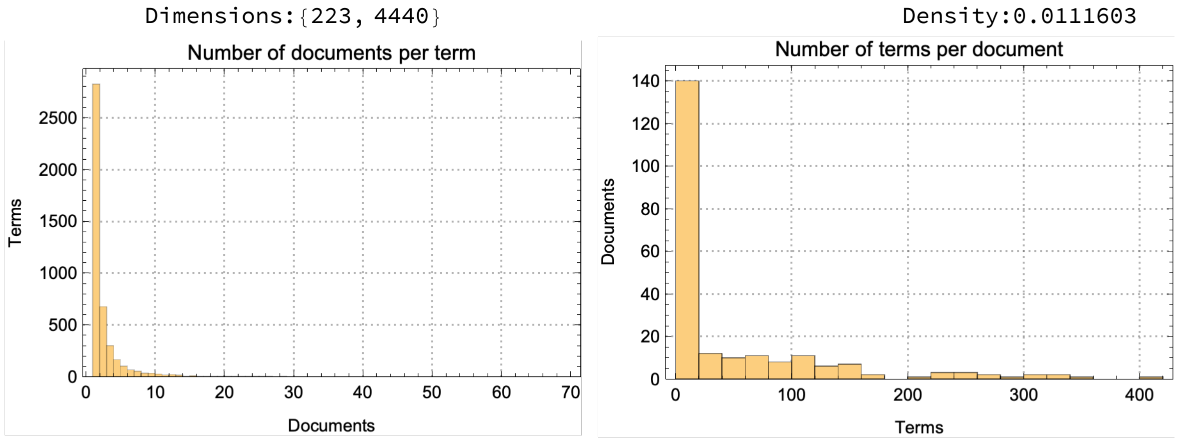
In[∘]:=
```
TakeLargestBy[Tally[DeleteStopwords[ToLowerCase[Flatten[TextWords /@ textHamlet]]]], #[[2]] &, 20]
```

Out[∘]=
```
{{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[∘]:=
```
LSAMonUnit[textHamlet] ⟹ LSAMonMakeDocumentTermMatrix ⟹ LSAMonEchoDocumentTermMatrixStatistics;
```

```
Dimensions:{223, 4440}          Density:0.0111603
```

» Context value "documentTermMatrix":




```
Number of
documents per term
summary
  1 # documents
  1st Qu  1
  Median  1
  Min     1
  3rd Qu  2
  Mean    2.48874
  Max     69
```

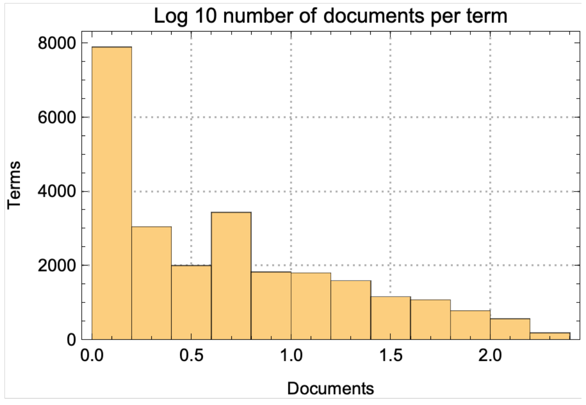
USA state of union speeches

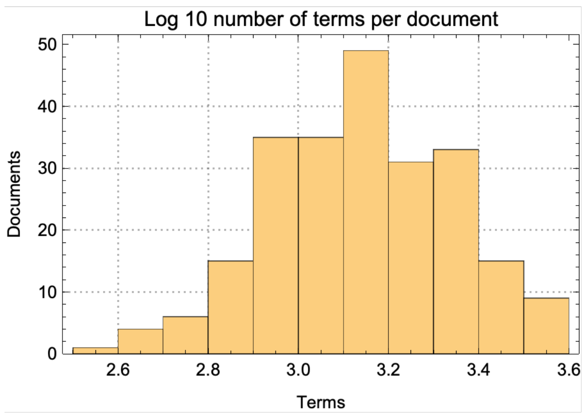
In[∘]:=
```
url =
  "https://github.com/antononcube/MathematicaVsR/blob/master/Data/MathematicaVsR-Data-StateOfUnionSpeeches.JSON.zip?raw=
    true";
str = Import[url, "String"];
filename = First@Import[StringToStream[str], "ZIP"];
aStateOfUnionSpeeches = Association@ImportString[Import[StringToStream[str],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

In[◦]:=
```
lsaObj =
  LSAMonUnit[aStateOfUnionSpeeches] ⟹ LSAMonMakeDocumentTermMatrix ⟹ LSAMonEchoDocumentTermMatrixStatistics["LogBase" → 10];
```

» Context value "documentTermMatrix":

Dimensions:{233, 25323}   Density:0.0611426






```
Log 10 number of
documents per term
summary
 1 # documents
 1st Qu  0.
 Min     0.
{Median  0.477121}
 Mean    0.649259
 3rd Qu  1.07918
 Max     2.36736
```

In[◦]:=
```
TakeLargest[ColumnSumsAssociation[lsaObj ⟹ LSAMonTakeDocumentTermMatrix], 12]
```

Out[◦]=
```
<|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[◦]:=
```
stopWords = Complement[DictionaryLookup["*"], DeleteStopwords[DictionaryLookup["*"]]];
```

In[◦]:=
```
Short[stopWords]
```

Out[◦]//Short=
```
{a, about, above, across, add-on, after, again, <<290>>, you'll, your, you're, yours, yourself, yourselves, you've}
```

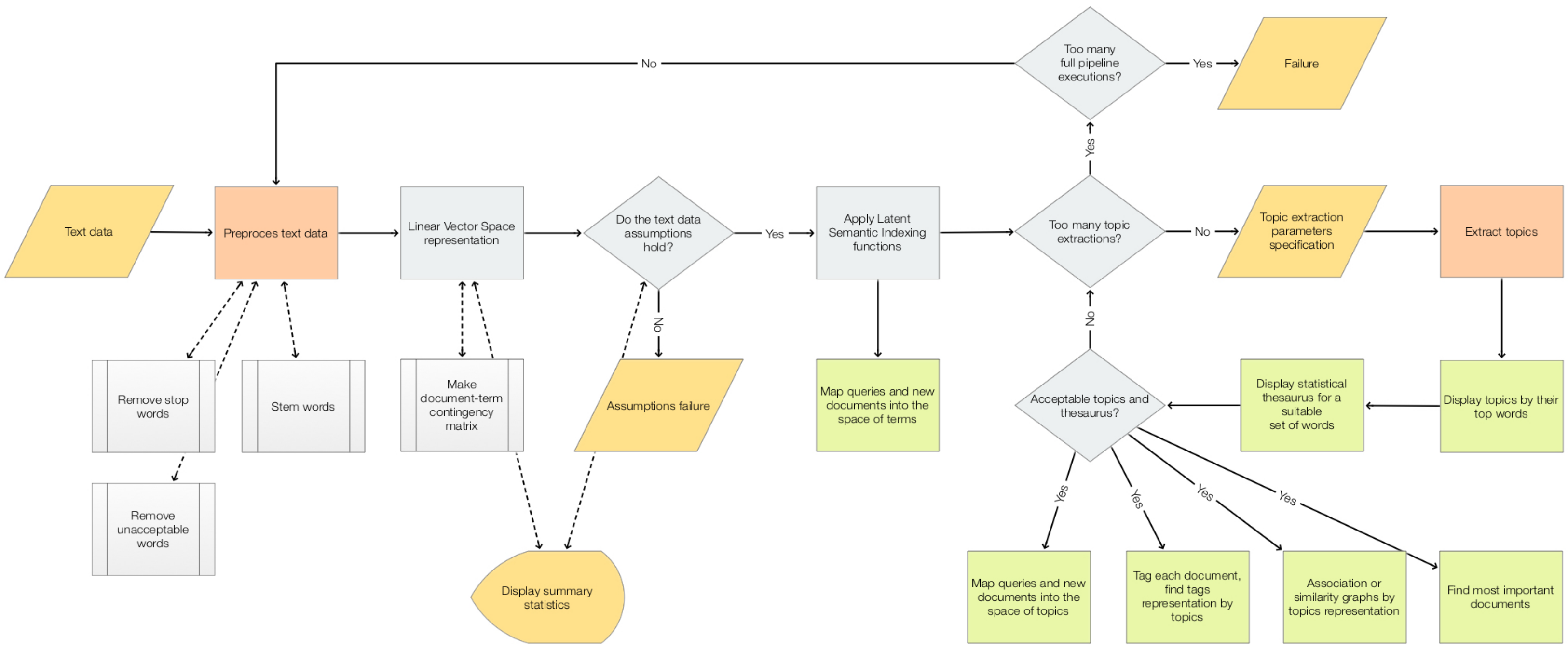
Design considerations

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.

Out[◦]=



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 **Mon**ad”.

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

$$\texttt{LSAMon[\_]} \xRightarrow[\texttt{LSAMonBind[LSAMon[\_],f\_]}]{} f_1 \xRightarrow[\texttt{LSAMonBind[LSAMon[\_],f\_]}]{} f_2 \xRightarrow[\texttt{LSAMonBind[LSAMon[\_],f\_]}]{} \ldots \xRightarrow[\texttt{LSAMonBind[LSAMon[\_],f\_]}]{} f_k \qquad (1)$$

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

$$\texttt{LSAMon[pval\_, context\_]} \xRightarrow[\texttt{QRMonBind[m\_,f\_]}]{} \ldots \left(\begin{cases} f_i[\texttt{\$LSAMonFailure}] & m \equiv \texttt{\$LSAMonFailure} \\ f_i[\texttt{x\_, c\_Association}] & m \textit{ is } \texttt{LSAMon[x\_, c\_Association]} \\ \texttt{\$LSAMonFailure} & \text{otherwise} \end{cases} \right) \xRightarrow[\texttt{LSAMonBind[m\_,f\_]}]{} \ldots \qquad (2)$$

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 NNMF 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].

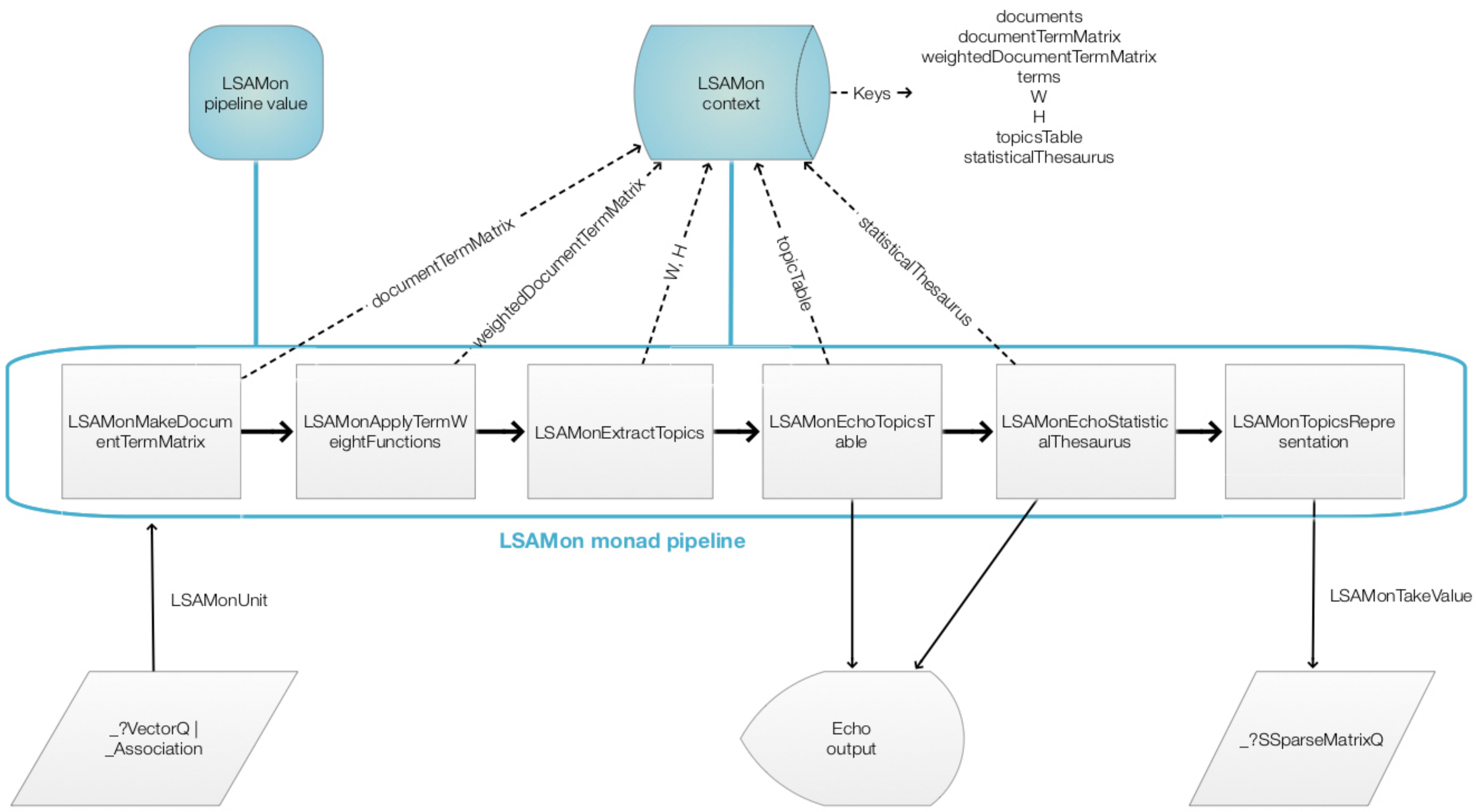
LSAMon overview

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:

Out[*]=



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	LSAMonDocumentCollectionQ	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

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].)

Out[•]=

#	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 withe 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.

Out[•]=

#	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[•]:=
```
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|>,
  ≪3≫, 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[•]:=
```
LSAMonUnit[textHamlet] ⟹ LSAMonMakeDocumentTermMatrix ⟹ LSAMonTakeDocumentTermMatrix
```

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

```
In[•]:= LSAMonUnit[] ⟹ LSAMonSetDocuments[textHamlet] ⟹ LSAMonMakeDocumentTermMatrix ⟹ LSAMonTakeDocumentTermMatrix
```

```
Out[•]= 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[•]:= WordData[#, "PorterStem"] & /@ {"consequential", "constitution", "forcing", ""}
Out[•]= {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[•]:= inds = {10, 19};
aTempText = AssociationThread[inds, textHamlet[[inds]]]
Out[•]= <|10 → ACT I. Scene I. Elsinore. A platform before the Castle., 19 → Scene II. Elsinore. A room of state in the Castle.|>
```

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

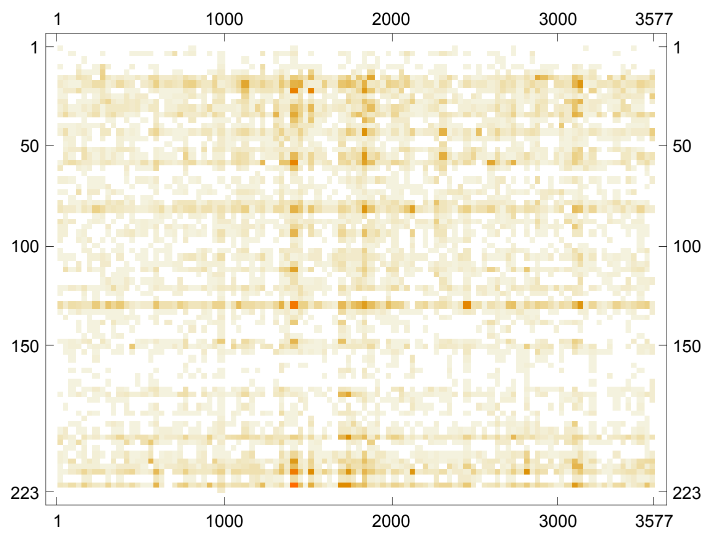
Out[•]//MatrixForm=

	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[•]:= 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[•]:= Short[lsaObj ⟹ LSAMonTakeStopWords]
Out[•]//Short= {a, about, above, across, add-on, after, again, <<290>>, 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[•]:= Short[lsaObj ⟹ LSAMonTakeStemmingRules]
Out[•]//Short= Automatic
```

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

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

```
Out[•]= 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 $f_{i j}$ 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

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

where

g_j -- global term weight;

$l_{i j}$ -- 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 a table of weight functions formulas.

Out[∘]=

weight type	name	formula
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	$\frac{1}{\sqrt{\sum_i f_{i j}^2}}$
local	Binary	$\chi(f_{i j})$
local	Logarithmic	$\log(f_{i j} + 1)$
local	Term frequency	$f_{i j}$
normalization	None	1
normalization	Cosine	$\frac{1}{\sqrt{\sum_j g_j \, l_{i j}}}$

Computation specifications

LSAMon function `LSAMonApplyTermWeightFunctions` delegates the LSI weight functions application to the package "DocumentTermMatrixConstruction.m", [AAp4].

Here is an example.

In[∘]:=
```
lsaHamlet = LSAMonUnit[textHamlet]⟹LSAMonMakeDocumentTermMatrix;
```

In[∘]:=
```
wmat =
  lsaHamlet⟹
   LSAMonApplyTermWeightFunctions["IDF", "TermFrequency", "Cosine"]⟹
   LSAMonTakeWeightedDocumentTermMatrix;
```

In[∘]:=
```
TakeLargest[ColumnSumsAssociation[wmat], 6]
```

Out[∘]=
```
<|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[∘]:=
```
wmat2 =
  lsaHamlet⟹
   LSAMonApplyTermWeightFunctions["GlobalWeightFunction" → "IDF",
    "LocalWeightFunction" → "TermFrequency", "NormalizerFunction" → "Cosine"]⟹
   LSAMonTakeWeightedDocumentTermMatrix;
```

In[∘]:=
```
TakeLargest[ColumnSumsAssociation[wmat2], 6]
```

Out[∘]=
```
<|enter → 21.7403, ham → 9.8253, hamlet → 9.16829, polonius → 8.79678, scene → 8.56075, king → 8.1436|>
```

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.

Out[•]=

1 IDF, Binary, Cosine	
Min	0.0150041
1st Qu	0.0573778
Median	0.0760402
Mean	0.102338
3rd Qu	0.107593
Max	1.

1 IDF, None, RowStochastic	
Min	0.000811524
1st Qu	0.00332414
Median	0.00613544
3rd Qu	0.0114367
Mean	0.020181
Max	1.

1 GFIDF, None, None	
1st Qu	1.
Median	1.
Min	1.
3rd Qu	1.5
Mean	2.24491
Max	237.5

1 Binary, None, Cosine	
Min	0.0181339
1st Qu	0.0342393
Median	0.057735
3rd Qu	0.090167
Mean	0.091407
Max	1.

1 None, Log, RowStochastic	
Min	0.00204538
1st Qu	0.00362849
Median	0.00655579
3rd Qu	0.0113535
Mean	0.020181
Max	1.

1 ColumnStochastic, Log, None	
Min	0.00193617
1st Qu	0.0462098
Median	0.173287
Mean	0.271695
3rd Qu	0.549306
Max	0.693147

1 IDF, Binary, None	
Min	1.15868
1st Qu	2.84222
Mean	3.56758
Median	3.79773
3rd Qu	4.71402
Max	4.71402

1 GFIDF, Binary, Cosine	
Min	0.0336221
1st Qu	0.048543
Median	0.0642267
3rd Qu	0.093449
Mean	0.0965935
Max	1.

1 GFIDF, None, RowStochastic	
Min	0.00082284
1st Qu	0.00205098
Median	0.00444174
3rd Qu	0.00933166
Mean	0.020181
Max	1.

1 Binary, None, None	
1st Qu	1
3rd Qu	1
Median	1
Min	1
Mean	1.32217
Max	27

1 None, None, Cosine	
Min	0.0181339
1st Qu	0.0342393
Median	0.057735
3rd Qu	0.090167
Mean	0.091407
Max	1.

1 ColumnStochastic, Log, RowStochastic	
Min	0.0000298676
1st Qu	0.00128507
Median	0.00425429
3rd Qu	0.0112668
Mean	0.020181
Max	1.

1 IDF, Binary, RowStochastic	
Min	0.000762208
1st Qu	0.00373532
Median	0.0061038
3rd Qu	0.0115251
Mean	0.020181
Max	1.

1 GFIDF, Binary, None	
1st Qu	1.
Median	1.
Min	1.
Mean	1.32217
3rd Qu	1.33333
Max	13.

1 Binary, Binary, Cosine	
Min	0.0494468
1st Qu	0.0614295
Median	0.0811107
3rd Qu	0.105409
Mean	0.106594
Max	1.

1 Binary, None, RowStochastic	
Min	0.00168634
1st Qu	0.002849
Median	0.00595238
3rd Qu	0.0106383
Mean	0.020181
Max	1.

1 None, None, None	
1st Qu	1
3rd Qu	1
Median	1
Min	1
Mean	1.32217
Max	27

1 ColumnStochastic, None, Cosine	
Min	0.000411024
1st Qu	0.0124279
Median	0.0396686
Mean	0.0780288
3rd Qu	0.101774
Max	1.

1 IDF, Log, Cosine	
Min	0.0164353
1st Qu	0.0560995
Median	0.0738618
Mean	0.101692
3rd Qu	0.107561
Max	1.

1 GFIDF, Binary, RowStochastic	
Min	0.00198353
1st Qu	0.00338414
Median	0.00610167
3rd Qu	0.0105852
Mean	0.020181
Max	1.

1 Binary, Binary, None	
1st Qu	1
3rd Qu	1
Max	1
Mean	1
Median	1
Min	1

1 None, Binary, Cosine	
Min	0.0494468
1st Qu	0.0614295
Median	0.0811107
3rd Qu	0.105409
Mean	0.106594
Max	1.

1 None, None, RowStochastic	
Min	0.00168634
1st Qu	0.002849
Median	0.00595238
3rd Qu	0.0106383
Mean	0.020181
Max	1.

1 ColumnStochastic, None, None	
Min	0.0027933
1st Qu	0.0769231
Median	0.25
Mean	0.40181
3rd Qu	1.
Max	1.

1 IDF, Log, None	
Min	0.803133
1st Qu	2.15194
Mean	2.723
Median	2.78706
3rd Qu	3.26751
Max	12.9073

1 GFIDF, Log, Cosine	
Min	0.0132742
1st Qu	0.0262098
Median	0.0436552
3rd Qu	0.0748374
Mean	0.0810793
Max	1.

1 Binary, Binary, RowStochastic	
Min	0.00244499
1st Qu	0.00377358
Median	0.00657895
3rd Qu	0.0111111
Mean	0.020181
Max	1.

1 None, Binary, None	
1st Qu	1
3rd Qu	1
Max	1
Mean	1
Median	1
Min	1

1 ColumnStochastic, Binary, Cosine	
Min	0.000246694
1st Qu	0.0104195
Median	0.036084
Mean	0.0766439
3rd Qu	0.101229
Max	1.

1 ColumnStochastic, None, RowStochastic	
Min	0.0000294977
1st Qu	0.00135327
Median	0.0042479
3rd Qu	0.0111001
Mean	0.020181
Max	1.

1 IDF, Log, RowStochastic	
Min	0.000952831
1st Qu	0.00360635
Median	0.00633097
3rd Qu	0.0111849
Mean	0.020181
Max	1.

1 GFIDF, Log, None	
1st Qu	0.693147
Median	0.693147
Min	0.693147
3rd Qu	1.03972
Mean	1.15397
Max	37.4467

1 Binary, Log, Cosine	
Min	0.0386884
1st Qu	0.0502109
Median	0.0752375
Mean	0.102772
3rd Qu	0.102919
Max	1.

1 None, Binary, RowStochastic	
Min	0.00244499
1st Qu	0.00377358
Median	0.00657895
3rd Qu	0.0111111
Mean	0.020181
Max	1.

1 ColumnStochastic, Binary, None	
Min	0.0027933
1st Qu	0.0625
Median	0.2
Mean	0.381212
3rd Qu	0.5
Max	1.

1 IDF, None, Cosine	
Min	0.0115357
1st Qu	0.0440683
Median	0.065596
Mean	0.0958
3rd Qu	0.0993633
Max	1.

1 GFIDF, Log, RowStochastic	
Min	0.00144179
1st Qu	0.00270264
Median	0.00517701
3rd Qu	0.0099619
Mean	0.020181
Max	1.

1 Binary, Log, None	
Log[2]	9355
Log[3]	1031
Log[4]	323
Log[5]	127
Log[6]	76
Log[7]	40
(Other)	98

1 None, Log, Cosine	
Min	0.0386884
1st Qu	0.0502109
Median	0.0752375
Mean	0.102772
3rd Qu	0.102919
Max	1.

1 ColumnStochastic, Binary, RowStochastic	
Min	0.0000163371
1st Qu	0.00115157
Median	0.00419756
3rd Qu	0.0112038
Mean	0.020181
Max	1.

1 IDF, None, None	
Min	1.15868
1st Qu	3.20995
Median	4.02088
Mean	4.36558
3rd Qu	4.71402
Max	81.8626

1 GFIDF, None, Cosine	
Min	0.00335833
1st Qu	0.0113981
Median	0.026682
3rd Qu	0.0548158
Mean	0.06707
Max	1.

1 Binary, Log, RowStochastic	
Min	0.00204538
1st Qu	0.00362849
Median	0.00655579
3rd Qu	0.0113535
Mean	0.020181
Max	1.

1 None, Log, None	
Log[2]	9355
Log[3]	1031
Log[4]	323
Log[5]	127
Log[6]	76
Log[7]	40
(Other)	98

1 ColumnStochastic, Log, Cosine	
Min	0.000413022
1st Qu	0.0115956
Median	0.0380513
Mean	0.0775075
3rd Qu	0.102593
Max	1.

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.

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

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

In[•]:= 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		3		5		7		9		11	
1.000	player	1.000	laert	1.000	state	1.000	hamlet	1.000	rosencrantz	1.000	answer
0.063	plai	0.248	king	0.963	room	0.167	ghost	0.830	guildenstern	0.072	sir
0.045	welcom	0.067	attend	0.915	castl	0.041	father	0.164	king	0.069	mother
0.044	present	0.037	lord	0.781	elsinor	0.037	denmark	0.146	poloniu	0.031	mad
0.042	tell	0.027	gertrud	0.040	fear	0.023	gentlemen	0.091	queen	0.027	night
0.040	poloniu	0.026	shall	0.024	welcom	0.018	call	0.025	lord	0.025	good
0.040	work	0.021	ophelia	0.019	mark	0.017	come	0.012	thank	0.017	bed
0.039	hear	0.018	world	0.018	love	0.014	daughter	0.008	command	0.017	make
0.039	queen	0.017	young	0.015	like	0.012	mother	0.008	gentlemen	0.017	look
0.032	make	0.017	come	0.015	make	0.009	rosencrantz	0.008	hath	0.016	matter
2		4		6		8		10		12	
1.000	ro	1.000	end	1.000	daughter	1.000	father	1.000	ophelia	1.000	horatio
0.724	lord	0.111	inde	0.718	pass	0.816	thou	0.372	queen	0.146	attend
0.364	sir	0.106	make	0.604	love	0.815	king	0.200	poloniu	0.132	gentleman
0.332	bring	0.059	king	0.561	fair	0.702	shall	0.122	mark	0.057	queen
0.300	king	0.034	passion	0.226	follow	0.685	good	0.103	nai	0.021	night
0.256	bodi	0.028	fortun	0.185	nai	0.665	let	0.089	prai	0.017	present
0.179	shall	0.028	love	0.184	lord	0.584	come	0.065	denmark	0.017	hast
0.177	know	0.027	thought	0.155	old	0.544	like	0.045	lord	0.016	work
0.160	come	0.021	grief	0.136	right	0.539	night	0.044	mother	0.014	hear
0.157	sai	0.020	dead	0.132	call	0.501	queen	0.034	daughter	0.012	sir

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

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

4. The interpretability provided by NNMF comes at a price.
5. NNMF is prone to get stuck into local minima, so several topic extractions (and corresponding evaluations) have to be done.
6. We would get different results with different NNMF runs using the same parameters. (NNMF uses random numbers initialization.)
7. The NNMF topic vectors are not orthogonal.
8. SVD is much faster than NNMF, but it topic vectors are hard to interpret.
9. Generally, the topics derived with SVD are stable, they do not change with different runs with the same parameters.
10. The SVD topics vectors are orthogonal, which provides for quick to find representations of documents not in the monad’s document collection.

The document-topic matrix *W* has column names that are automatically derived from the top three terms in each topic.

```
In[•]:= ColumnNames[lsaHamlet⟹LSAMonTakeW]
Out[•]= {player-plai-welcom, ro-lord-sir, laert-king-attend, end-inde-make, state-room-castl, daughter-pass-love, hamlet-ghost-father,
  father-thou-king, rosencrantz-guildenstern-king, ophelia-queen-poloniu, answer-sir-mother, horatio-attend-gentleman}
```

Of course the row names of *H* have the same names.

```
In[•]:= RowNames[lsaHamlet⟹LSAMonTakeH]
Out[•]= {player-plai-welcom, ro-lord-sir, laert-king-attend, end-inde-make, state-room-castl, daughter-pass-love, hamlet-ghost-father,
  father-thou-king, rosencrantz-guildenstern-king, ophelia-queen-poloniu, answer-sir-mother, horatio-attend-gentleman}
```

Extracting statistical thesauri

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

Here is an example over the State of Union speeches.

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

» statistical thesaurus:

term	statistical thesaurus entries
bank	{bank, deposit, specul, currenc, speci, paper, distribut, charter, surplu, embarrass, credit, institut}
economi	{economi, respons, strengthen, earn, elimin, higher, plan, better, creat, comprehens, save, dedic}
enemi	{enemi, victori, command, savag, british, lake, prison, militia, superior, confin, warfar, volunt}
health	{health, invest, coverag, work, middl, care, lower, hard, stai, economi, lot, start}
nuclear	{nuclear, technolog, ensur, strateg, energi, oil, middl, inflat, percent, basic, leader, proud}
port	{port, commenc, instruct, amiti, intercours, london, vessel, dispatch, extinguish, princip, arrang, harbor}
school	{school, child, children, teacher, drug, parent, thank, famili, centuri, environ, student, medic}
war	{war, forc, peac, right, power, place, great, success, good, provid, continu, secur}

In the code above: (i) the options signature style is used, (ii) the statistical thesaurus entry words are stemmed first.

We can also call `LSAMonEchoStatisticalThesaurus` directly without calling `LSAMonExtractStatisticalThesaurus` first.

```
In[•]:= lsaSpeeches⟹
   LSAMonEchoStatisticalThesaurus["Words" → Map[WordData[#, "PorterStem"] &, entryWords], "NumberOfNearestNeighbors" → 12];
```

» statistical thesaurus:

term	statistical thesaurus entries
bank	{bank, deposit, specul, currenc, speci, paper, distribut, charter, surplu, embarrass, credit, institut}
economi	{economi, respons, strengthen, earn, elimin, higher, plan, better, creat, comprehens, save, dedic}
enemi	{enemi, victori, command, savag, british, lake, prison, militia, superior, confin, warfar, volunt}
health	{health, invest, coverag, work, middl, care, lower, hard, stai, economi, lot, start}
nuclear	{nuclear, technolog, ensur, strateg, energi, oil, middl, inflat, percent, basic, leader, proud}
port	{port, commenc, instruct, amiti, intercours, london, vessel, dispatch, extinguish, princip, arrang, harbor}
school	{school, child, children, teacher, drug, parent, thank, famili, centuri, environ, student, medic}
war	{war, forc, peac, right, power, place, great, success, good, provid, continu, secur}

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

» value: SparseArray[Specified elements: 5 Dimensions: {1, 3569}]

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

```
In[•]:= obj⟹LSAMonEchoFunctionValue[ Function[...] ];
```

»

	denmark	hamlet	kill	king	princ
1	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[•]:= inds = {6, 10};
queries = Part[lsaHamlet⟹LSAMonTakeDocuments, inds];
queries
```

```
Out[•]= ⟨|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[•]:= 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[•]:= 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[•]:= inds = {6, 10};
queries = Part[lsaHamlet⟹LSAMonTakeDocuments, inds];
queries
```

```
Out[•]= ⟨|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[•]:= lsaHamlet⟹
  LSAMonRepresentByTopics[queries]⟹
  LSAMonEchoFunctionValue[Function[...]];
```

»

	player-plai-welcom	state-room-castl	daughter-pass-love	hamlet-ghost-father	rosencrantz-guildenstern-king	ophelia-queen-polo
id.0006	0.00295518	-0.000452681	0.210033	0.0889422	0.0116542	0.239248
id.0010	0.0015422	0.304837	-0.00184199	0.000254639	0.000101572	0.000131886

```
In[•]:= lsaHamlet⟹
  LSAMonEchoFunctionContext[Function[...]];
```

»

	player-plai-welcom	state-room-castl	hamlet-ghost-father	ophelia-queen-poloniu	answer-sir-mother
id.0006	0.00349852	0.000671018	0.0919758	0.199069	0.0182967
id.0010	0.	0.	0.	0.	0.

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.

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

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

$m_i \approx w_i.H$

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

$q_0 \approx x.H$

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

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

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

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

The vector x obtained with `LSAMonRepresentByTopics`.

Tags representation

Sometimes we want to find the topics representation of tags associated with monad's documents and the tag-document associations are one-to-many. See [AA3].

Let us consider a concrete example -- we want to find what topics correspond to the different presidents in the collection of State of Union speeches.

Here we find the document tags (president names in this case.)

In[∘]:=
```
tags = StringReplace[RowNames[lsaSpeeches⟹LSAMonTakeDocumentTermMatrix],
   RegularExpression[".\\d\\d\\d\\d-\\d\\d-\\d\\d"] → ""];
Short[
 tags]
```

Out[∘]//Short=
```
{George.Washington, George.Washington, George.Washington,
 George.Washington, ≪225≫, Barack.Obama, Donald.Trump, Donald.Trump, Donald.Trump}
```

Here is the number of unique tags (president names.)

In[∘]:=
```
Length[Union[tags]]
```

Out[∘]= 42

Here we compute the tag-topics representation matrix using the function `LSAMonRepresentDocumentTagsByTopics`.

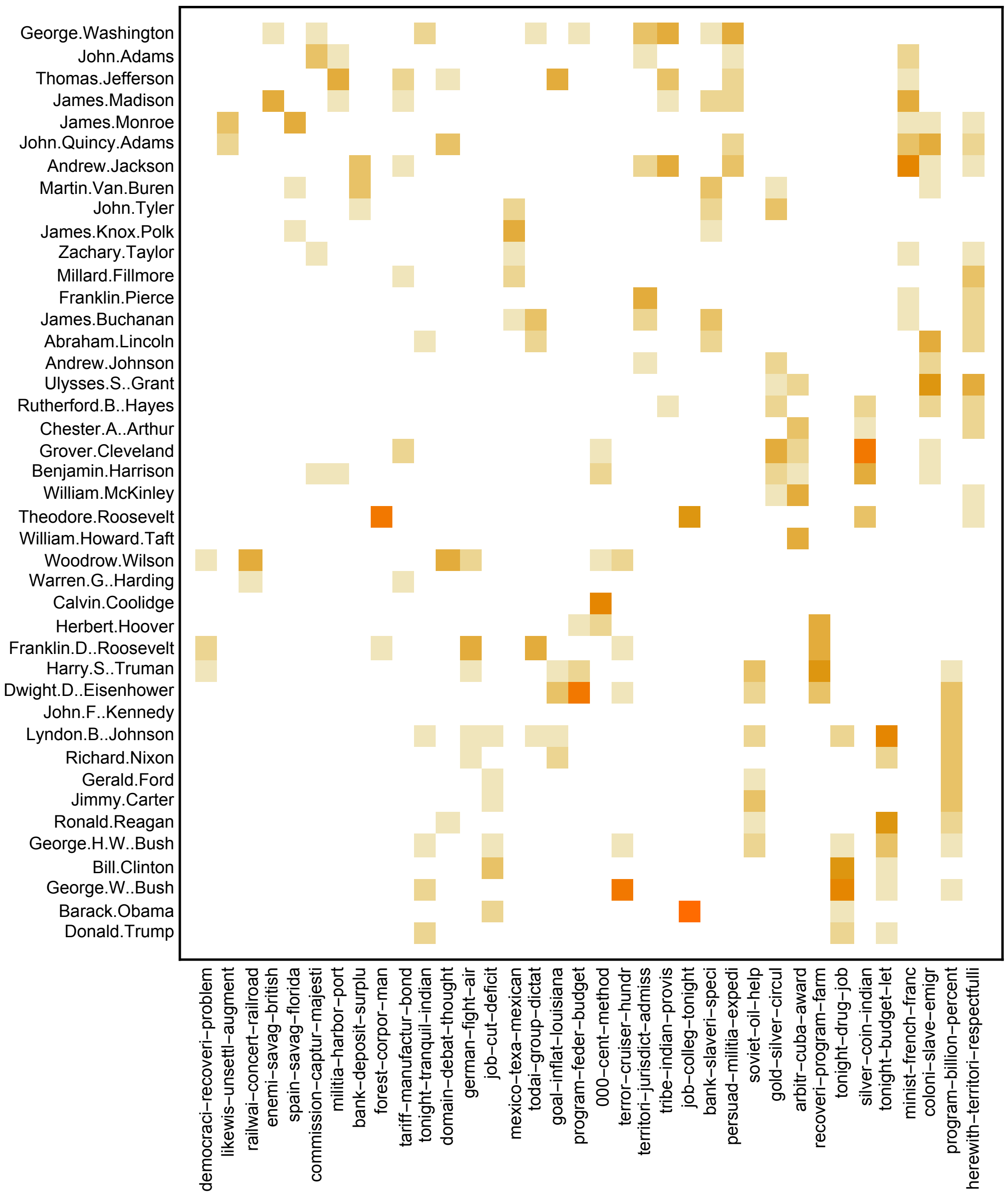
In[∘]:=
```
tagTopicsMat =
 lsaSpeeches⟹
  LSAMonRepresentDocumentTagsByTopics[tags]⟹
  LSAMonTakeValue
```

Out[∘]= SparseArray[Specified elements: 192 Dimensions: {42, 36}]

Here is a heatmap plot of the tag-topics matrix made with the package "HeatmapPlot.m", [AAp11].

In[∘]:=
```
HeatmapPlot[tagTopicsMat⟦All, Ordering@ColumnSums[tagTopicsMat]⟧, DistanceFunction → None, ImageSize → Large]
```

Out[∘]=



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 sub-section 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[∘]:= focusText = First@Pick[textHamlet, StringMatchQ[textHamlet, ___ ~~ "to be" ~~ __ ~~ "or not to be" ~~ ___, IgnoreCase → True]];
Short[focusText]
```

```
Out[∘]//Short= Ham. To be, or not to be- that is the question: Whether 'tis nobler
 … 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[∘]:= 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[∘]:= p =
  LSAMonUnit[textHamlet] ⟹ LSAMonMakeDocumentTermMatrix[Automatic, Automatic];
```

```
In[∘]:= p ⟹ LSAMonTakeMatrix
```

```
Out[∘]= SparseArray[ Specified elements: 10 759
                     Dimensions: {223, 3577} ]
```

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

```
In[∘]:= Short@(p ⟹ QRMonTakeContext)["documents"]
```

```
Out[∘]//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[∘]:= Keys[p ⟹ LSAMonDropDocuments ⟹ QRMonTakeContext]
```

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

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

The utilization of SSparseMatrix objects

The LSAMon monad heavily relies on `SSparseMatrix` objects, [AAp6, AA5], for internal representation of data and computation results.

A `SSparseMatrix` object is a matrix with named rows and columns.

Here is an example.

```
In[∘]:= n = 6;
rmat = ToSSparseMatrix[SparseArray[{{1, 2} → 1, {4, 5} → 1}, {n, n}],
   "RowNames" → RandomSample[CharacterRange["A", "Z"], n], "ColumnNames" -> RandomSample[CharacterRange["a", "z"], n]];
MatrixForm[
 rmat]
```

Out[∘]//MatrixForm=

	c	l	a	v	t	f
F	0	1	0	0	0	0
S	0	0	0	0	0	0
P	0	0	0	0	0	0
G	0	0	0	0	1	0
X	0	0	0	0	0	0
C	0	0	0	0	0	0

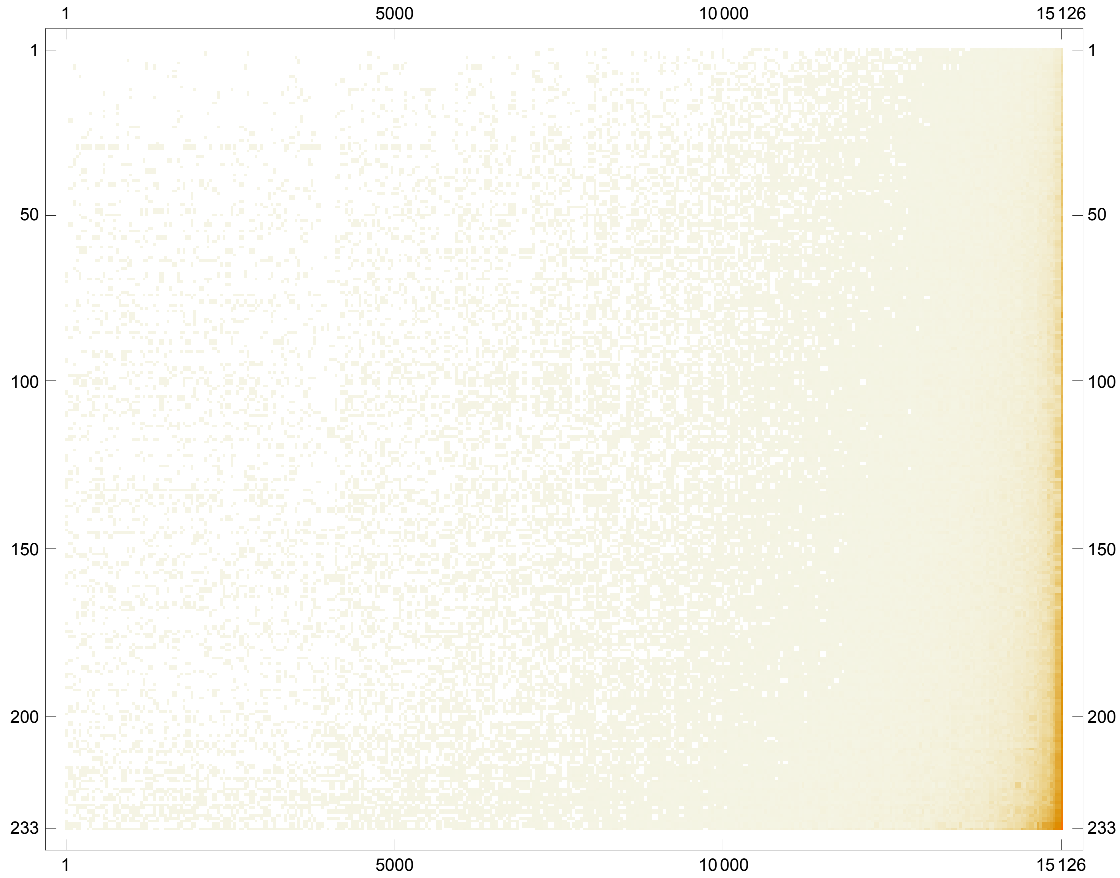
In this section we look into some useful `SSparseMatrix` idioms applied within `LSAMon`.

Visualize with sorted rows and columns

In some situations it is beneficial to sort rows and columns of the (weighted) document-term matrix.

In[•]:=
```
docTermMat = lsaSpeeches⟹LSAMonTakeDocumentTermMatrix;
MatrixPlot[docTermMat⟦Ordering[RowSums[docTermMat]], Ordering[ColumnSums[docTermMat]]⟧,
 MaxPlotPoints → 300, ImageSize → Large]
```

Out[•]=



Finding the most and least popular terms

The most popular terms in the document collection can be found through the association of the column sums of the document-term matrix.

In[•]:=
```
TakeLargest[ColumnSumsAssociation[lsaSpeeches⟹LSAMonTakeDocumentTermMatrix], 10]
```

Out[•]=
```
⟨|state → 8852, govern → 8147, year → 6362, nation → 6182,
 congress → 5040, unit → 5040, countri → 4504, peopl → 4306, american → 3648, law → 3496|⟩
```

Similarly for the lest popular terms.

In[•]:=
```
TakeSmallest[ColumnSumsAssociation[lsaSpeeches⟹LSAMonTakeDocumentTermMatrix], 10]
```

Out[•]=
```
⟨|036 → 1, 027 → 1, _____________________ → 1, 0111 → 1, 006 → 1,
 0000 → 1, 0001 → 1, ______________________ → 1, ____ → 1, ____________________ → 1|⟩
```

Showing only non-zero columns

In some cases we want to show only columns of the data or computation results matrices that have non-zero elements.

Here is an example (similar to other examples in the previous section.)

In[•]:=
```
lsaHamlet⟹
  LSAMonRepresentByTerms[
   {"this country is rotten", "where is my sword my lord", "poison in the ear should be in the play"}]⟹
  LSAMonEchoFunctionValue[MatrixForm[#1⟦All, Keys[Select[ColumnSumsAssociation[#1], #1 > 0 &]]⟧] &];
```

»

	countri	ear	lord	plai	poison	rotten	sword
1	0.642798	0.	0.	0.	0.	0.766036	0.
2	0.	0.	0.367111	0.	0.	0.	0.930177
3	0.	0.529938	0.	0.454323	0.71607	0.	0.

In the pipeline code above: (i) from the list of queries a representation matrix is made, (ii) that matrix is assigned to the pipeline value, (iii) in the pipeline echo value function the non-zero columns are selected with by using the keys of the non-zero elements of the association obtained with `ColumnSumsAssociation`.

Similarities based on representation by terms

Here is way to compute the similarity matrix of different sets of documents that are not required to be in monad’s document collection.

```
In[∘]:= sMat1 =
         lsaSpeeches⟹
          LSAMonRepresentByTerms[aStateOfUnionSpeeches⟦Range[-5, -2]⟧]⟹
          LSAMonTakeValue
```

```
Out[∘]= SparseArray[ Specified elements: 4546 Dimensions: {4, 15126} ]
```

```
In[∘]:= sMat2 =
         lsaSpeeches⟹
          LSAMonRepresentByTerms[aStateOfUnionSpeeches⟦Range[-7, -3]⟧]⟹
          LSAMonTakeValue
```

```
Out[∘]= SparseArray[ Specified elements: 5831 Dimensions: {5, 15126} ]
```

```
In[∘]:= MatrixForm[sMat1.Transpose[sMat2]]
```

··· SSparseMatrix: The dimension names {1, 1} are the same; using {"1", "2"} instead.

Out[∘]//MatrixForm=

	Barack.Obama.2013-02-12	Barack.Obama.2014-01-28	Barack.Obama.2015-01-20	Barack.Obama.2016-01-12	D
Barack.Obama.2015-01-20	0.435536	0.46657	1.	0.438214	
Barack.Obama.2016-01-12	0.379678	0.404785	0.438214	1.	
Donald.Trump.2017-02-28	0.263294	0.23298	0.218087	0.203749	
Donald.Trump.2018-01-30	0.265816	0.25859	0.227366	0.217215	

Similarities based on representation by topics

Similarly to weighted Boolean similarities matrix computation above we can compute a similarity matrix using the topics representations. Note that an additional normalization steps is required.

```
In[∘]:= sMat1 =
          lsaSpeeches⟹
           LSAMonRepresentByTopics[aStateOfUnionSpeeches⟦Range[-5, -2]⟧]⟹
           LSAMonTakeValue;
        sMat1 = WeightTermsOfSSparseMatrix[sMat1, "None", "None", "Cosine"]
```

```
Out[∘]= SparseArray[ Specified elements: 144 Dimensions: {4, 36} ]
```

```
In[∘]:= sMat2 =
          lsaSpeeches⟹
           LSAMonRepresentByTopics[aStateOfUnionSpeeches⟦Range[-7, -3]⟧]⟹
           LSAMonTakeValue;
        sMat2 = WeightTermsOfSSparseMatrix[sMat2, "None", "None", "Cosine"]
```

```
Out[∘]= SparseArray[ Specified elements: 180 Dimensions: {5, 36} ]
```

```
In[∘]:= MatrixForm[sMat1.Transpose[sMat2]]
```

··· SSparseMatrix: The dimension names {1, 1} are the same; using {"1", "2"} instead.

Out[∘]//MatrixForm=

	Barack.Obama.2013-02-12	Barack.Obama.2014-01-28	Barack.Obama.2015-01-20	Barack.Obama.2016-01-12	D
Barack.Obama.2015-01-20	0.950843	0.974615	1.	0.875923	
Barack.Obama.2016-01-12	0.902992	0.903844	0.875923	1.	
Donald.Trump.2017-02-28	0.688586	0.696063	0.667645	0.531271	
Donald.Trump.2018-01-30	0.599199	0.596653	0.591386	0.386631	

Note the differences with the weighted Boolean similarity matrix in the previous sub-section -- the similarities that are less than 1 are noticeably larger.

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].

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

Out[∘]=
```
{0.109397, TestReportObject[ Title: Test Report: MonadicLatentSemanticAnalysis-Unit-Tests.wlt
                              Success rate: Indeterminate   Tests run: 0 ]}
```

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

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

Out[∘]=
```
{{{2019, 9, 12, 10, 18, 35.886380}, TestReport
TestReport could not open the test file
~/MathematicaForPrediction/UnitTests/MonadicLatentSemanticAnalysis-Unit-Tests.wlt}[TestID]}
```

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:

Out[1151]=

<table>
<tr><th>#</th><th>pipeline</th></tr>
<tr><td>1</td><td>LSAMon[aStateOfUnionSpeeches, <| |>] ⟹ LSAMonSetDocuments[
hamletData] ⟹ LSAMonMakeDocumentTermMatrix[Automatic, Automatic] ⟹ LSAMonEchoDocumentsStatistics ⟹
LSAMonApplyTermWeightFunctions[IDF] ⟹ LSAMonExtractTopics[NumberOfTopics → 16,
Method → SVD] ⟹ LSAMonExtractStatisticalThesaurus[{life, countri}, 12] ⟹ LSAMonEchoStatisticalThesaurus[
Words → {health, friend}]</td></tr>
<tr><td>2</td><td>LSAMon[aStateOfUnionSpeeches, <| |>] ⟹ LSAMonSetDocuments[aStateOfUnionSpeeches] ⟹ LSAMonMakeDocumentTermMatrix[{},
stopWords] ⟹ LSAMonEchoDocumentsStatistics ⟹ LSAMonApplyTermWeightFunctions[] ⟹ LSAMonExtractTopics[
NumberOfTopics → -1, Method → NNMF, MaxSteps → 12] ⟹ LSAMonFindMostImportantDocuments ⟹ LSAMonEchoStatisticalThesaurus</td></tr>
<tr><td>3</td><td>LSAMon[None, <| |>] ⟹ LSAMonSetDocuments[
hamletData] ⟹ LSAMonMakeDocumentTermMatrix[] ⟹ LSAMonEchoDocumentsStatistics ⟹ LSAMonApplyTermWeightFunctions[
NormalizerFunction → Cosine] ⟹ LSAMonExtractTopics[NumberOfTopics → 16, Method → SVD,
MinNumberOfDocumentsPerTerm → -12] ⟹ LSAMonExtractStatisticalThesaurus[{life, countri}, 12] ⟹ LSAMonTakeValue</td></tr>
<tr><td>4</td><td>LSAMon[None, <| |>] ⟹ LSAMonSetDocuments[hamletData] ⟹ LSAMonMakeDocumentTermMatrix[{},
stopWords] ⟹ LSAMonApplyTermWeightFunctions[LocalWeightFunction → Binary] ⟹ LSAMonExtractTopics[
NumberOfTopics → 16, Method → SVD, MinNumberOfDocumentsPerTerm → -12]</td></tr>
<tr><td>5</td><td>LSAMon[aStateOfUnionSpeeches, <| |>] ⟹ LSAMonSetDocuments[
aStateOfUnionSpeeches] ⟹ LSAMonMakeDocumentTermMatrix ⟹ LSAMonEchoDocumentsStatistics ⟹
LSAMonApplyTermWeightFunctions[
IDF] ⟹ LSAMonExtractTopics[12] ⟹ LSAMonFindMostImportantDocuments ⟹ LSAMonTakeValue</td></tr>
<tr><td>6</td><td>LSAMon[None, <| |>] ⟹ LSAMonSetDocuments[
aStateOfUnionSpeeches] ⟹ LSAMonApplyTermWeightFunctions[GlobalWeightFunction → IDF] ⟹ LSAMonExtractTopics[12]</td></tr>
</table>

Here we run the pipelines as unit tests:

In[∘]:=
```
AbsoluteTiming[
 res = TestRunLSAMonPipelines[pipelines, "Echo" → False];
]
```

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

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

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

Future plans

Dimension reduction extensions

It would be nice to extend the Dimension reduction functionalities of LSAMon to include other algorithms like Independent Component Analysis (ICA), [Wk5]. Ideally with LSAMon we can do comparisons between SVD, NNMF, and ICA like the image de-nosing based comparison explained in [AA8].

Another direction is to utilize Neural Networks for the topic extraction and making of statistical thesauri.

Conversational agent

Since LSAMon is a DSL it can be relatively easily interfaced with a natural language interface.

Here is an example of natural language commands parsed into LSA code using the package [AAp13].

Out[=]=

```
1  command: create the document term matrix
    parsed: MakeDocumentTermMatrix[{document, {term, matrix}}]
  residual: {}
2  command: apply to the matrix entries idf
    parsed: LSICommand[LSIFuncsList[{LSIGlobalFunc[LSIGlobalFuncIDF[idf]]}]]
  residual: {}
3  command: extract 23 topics with the method NNMF and max steps 12
    parsed: TopicsExtractionCommand[{TopicsNumber[NumericValue[23]], TopicsParametersSpec[
              TopicsParametersList[{TopicsExtractionMethod[NNMF], {TopicsMaxIterations[NumericValue[12]]}}]]}]
  residual: {}
4  command: extract statistical thesaurus with 12 synonyms
    parsed: ThesaurusExtractionCommand[ThesaurusParametersSpec[ThesaurusNumberOfSynonyms[NumericValue[12]]]]
  residual: {}
```

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.

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 .

[AAp11] Anton Antonov, Heatmap plot Mathematica package, (2017), MathematicaForPrediction at GitHub.
URL: https://github.com/antononcube/MathematicaForPrediction/blob/master/Misc/HeatmapPlot.m .

[AAp12] Anton Antonov, Independent Component Analysis Mathematica package, (2016), MathematicaForPrediction at GitHub.
URL: https://github.com/antononcube/MathematicaForPrediction/blob/master/IndependentComponentAnalysis.m .

[AAp13] Anton Antonov, Latent semantic analysis workflows grammar in EBNF, (2018), ConverasationalAgents at GitHub.
URL: https://github.com/antononcube/ConversationalAgents/blob/master/EBNF/English/Mathematica/LatentSemanticAnalysisWorkflowsGrammar.m .

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/ .

[AA7] Anton Antonov, "Independent component analysis for multidimensional signals", (2016), MathematicaForPrediction at WordPress.
URL: https://mathematicaforprediction.wordpress.com/2016/05/23/independent-component-analysis-for-multidimensional-signals/ .

[AA8] Anton Antonov, "Comparison of PCA, NNMF, and ICA over image de-noising", (2016), MathematicaForPrediction at WordPress.
URL: https://mathematicaforprediction.wordpress.com/2016/05/26/comparison-of-pca-nnmf-and-ica-over-image-de-noising/ .

Other

[Wk1] Wikipedia entry, Monad,
URL: 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 .

[Wk5] Wikipedia entry, Independent component analysis,
URL: https://en.wikipedia.org/wiki/Independent_component_analysis .

[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 .