Technical Whitepaper

Natural-language processing toolkit

Date May 2018

Author Fionnuala Carr

Contents

Natural-language processing

Natural-language processing (NLP) can be used to answer a variety of questions about unstructured text data, as well as facilitating open-ended exploration.

It can be applied to datasets such as emails, online articles and comments, tweets, or novels. Although the source is text, transformations are applied to convert this data to vectors, dictionaries and symbols which can be handled very effectively by q. Many operations such as searching, clustering, and keyword extraction can all be done using very simple data structures, such as feature vectors and bag-of-words representations.

Preparing text

Operations can be pre-run on a corpus, with the results cached to a table, which can be persisted.

Operations undertaken to parse the dataset:

operation	*effect*
Tokenization	splits the words; e.g. `John's` becomes `John` as one token, and `'s` as a second
Sentence detection	characters at which a sentence starts and ends
Part of speech tagger	parses the sentences into tokens and gives each token a label e.g. `lemma`, `pos`, `tag` etc.
Lemmatization	converts to a base form e.g. `ran` (verb) to `run` (verb)

.nlp.newParser

Creates a parser

Syntax: `.nlp.newParser[spacymodel;fields]`

Where

- `spacymodel` is a model or language[1] (symbol)
- `fields` is the field/s you want in the output (symbol atom or vector)

returns a function to parse the text.

The optional fields are:

field	*type*	*content*
`text`	list of characters	original text
`tokens`	list of symbols	the tokenized text
`sentChars`	list of lists of longs	indexes of start and end of sentences
`sentIndices`	list of integers	indexes of the first token of each sentences
`pennPOS`	list of symbols	the Penn Treebank tagset
`uniPOS`	list of symbols	the Universal tagset
`lemmas`	list of symbols	the base form of the word
`isStop`	boolean	is the token part of the stop list?
`likeEmail`	boolean	does the token resembles an email?

1. https://spacy.io/usage/models

field	*type*	*content*
`likeURL`	boolean	does the token resembles a URL?
`likeNumber`	boolean	does the token resembles a number?
`keywords`	list of dictionaries	significance of each term
`starts`	long	index that a token starts at

The resulting function is applied to a list of strings.

Parsing the novel *Moby Dick*:

```
/ creating a parsed table
fields:`text`tokens`lemmas`pennPOS`isStop`sentChars`starts`sentIndices`keywords
myparser:.nlp.newParser[`en;fields]
corpus:myparser mobyDick
cols corpus
`tokens`lemmas`pennPOS`isStop`sentChars`starts`sentIndices`keywords`text
```

Finding part-of-speech tags in a corpus

This is a quick way to find all of the nouns, adverbs, etc. in a corpus. There are two types of part-of-speech (POS) tags you can find: Penn Tree tags[2] and Universal Tree tags[3].

.nlp.findPOSRuns

Runs of tokens whose POS tags are in the set passed

Syntax: `.nlp.findPOSRuns[tagtype;tags;document]`

Where

- `tagtype` is `uniPos` or `pennPos`
- `tags` is one or more POS tags (symbol atom or vector)
- `document` is parsed text (dictionary)

returns a general list:

1. text of the run (symbol vector)
2. indexes of the first occurrence of each token (long vector)

Importing a novel from a plain text file, and finding all the proper nouns in the first chapter of *Moby Dick*:

2. https://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html

3. http://universaldependencies.org/docs/en/pos/all.html

```
fields:`text`tokens`lemmas`pennPOS`isStop`sentChars`starts`sentIndices`keywords
q)myparser:.nlp.parser.i.newParser[`en;fields]
q)corpus:myparser mobyDick

q).nlp.findPOSRuns[`pennPOS;`NNP`NNPS;corpus 0][;0]
`loomings`ishmael`november`cato`manhattoes`circumambulate`sabbath`go`corlears`hook`coenties
```

Feature vectors

We can generate a dictionary of descriptive terms, which consist of terms and their associated weights. These dictionaries are called *feature vectors* and they are very useful as they give a uniform representation that can describe words, sentences, paragraphs, documents, collections of documents, clusters, concepts and queries.

Calculating feature vectors for documents

The values associated with each term in a feature vector are how significant that term is as a descriptor of the entity. For documents, this can be calculated by comparing the frequency of words in that document to the frequency of words in the rest of the corpus.

Sorting the terms in a feature vector by their significance, you get the keywords that distinguish a document most from the corpus, forming a terse summary of the document. This shows the most significant terms in the feature vector for one of Enron CEO Jeff Skilling's email's describing a charity bike ride.

TF-IDF is an algorithm that weighs a term's frequency (TF) and its inverse document frequency (IDF). Each word or term has its respective TF and IDF score. The product of the TF and IDF scores of a term is called the TF-IDF weight of that term.

.nlp.TFIDF

TF-IDF scores for all terms in the document

Syntax: `.nlp.TFIDF x`

Where `x` is a table of documents, returns for each document, a dictionary with the tokens as keys, and relevance as values.

Extract a specific document and find the most significiant words in that document:

```
q)queriedemail:jeffcorpus[where jeffcorpus[`text] like "*charity bike*"]`text;
q)5#desc .nlp.TFIDF[jeffcorpus]1928
bikers   | 17.7979
biker    | 17.7979
strenuous| 14.19154
route    | 14.11932
rode     | 14.11136
```

In cases where the dataset is more similar to a single document than a collection of separate documents, a different algorithm can be used. This algorithm is taken from Carpena, P., et al. 'Level statistics of words: Finding keywords in literary texts and

symbolic sequences.' . The idea behind the algorithm is that more important words occur in clusters and less important words follow a random distribution.

.nlp.keywordsContinuous

For an input which is conceptually a single document, such as a book, this will give better results than TF-IDF

Syntax: `.nlp.keywordsContinuous x`

Where `x` is a table of documents, returns a dictionary where the keys are keywords and the values are their significance.

Treating all of *Moby Dick* as a single document, the most significant keywords are *Ahab*, *Bildad*, *Peleg* (the three captains on the boat) and *whale*.

```
q)10#keywords:.nlp.keywordsContinuous corpus
ahab     | 65.23191
peleg    | 52.21875
bildad   | 46.56072
whale    | 42.72953
stubb    | 38.11739
queequeg | 35.34769
steelkilt| 33.96713
pip      | 32.90067
starbuck | 32.05286
thou     | 32.05231
```

Calculating feature vectors for words

The feature vector for a word can be calculated as a collection of how well other words predict the given keyword. The weight given to these words is a function of how much higher the actual co-occurrence rate is from the expected co-occurrence rate the terms would have if they were randomly distributed.

.nlp.findRelatedTerms

Feature vector for a term

Syntax: `.nlp.findRelatedTerms[x;y]`

Where

- `x` is a list of documents
- `y` is a symbol which is the token for which to find related terms

returns a dictionary of the related tokens and their relevances.

```
q).nlp.findRelated[corpus;`captain]
peleg | 1.653247
bildad| 1.326868
ahab  | 1.232073
ship  | 1.158671
cabin | 0.9743517
```

Phrases can be found by looking for runs of words with an above-average significance to the query term.

.nlp.extractPhrases

Runs of tokens that contain the term where each consecutive word has an above-average co-occurrence with the term

Syntax: `.nlp.extractPhrases[corpus;term]`

Where

- `corpus` is a subcorpus (table)
- `term` is the term to extract phrases around (symbol)

returns a dictionary with phrases as the keys and their relevance as the values.

Search for the phrases that contain `captain` and see which phrase has the largest occurrence; we find `captain ahab` occurs most often in the book: 31 times.

```
q).nlp.extractPhrases[corpus;`captain]
"captain ahab"         | 31
"captain peleg"        | 12
"captain bildad"       | 7
"captain sleet"        | 5
"stranger captain"     | 4
"said the captain"     | 3
"sea-captain"          | 2
"whaling captain"      | 2
"captain's cabin"      | 2
"captain ahab,\" said"| 2
"captain pollard"      | 2
"captain d'wolf"       | 2
"way, captain"         | 2
```

Comparisons

Comparing feature vectors

A vector can be thought of either as

- the co-ordinates of a point
- describing a line segment from the origin to a point

The view of a vector as a line segment starting at the origin is useful, as any two vectors will have an angle between them, corresponding to their similarity, as calculated by cosine similarity.

The *cosine similarity* of two vectors is the dot product of two vectors over the product of their magnitudes. It is a standard distance metric for comparing documents.

Comparing corpora

A quick way to compare corpora is to find words common to the whole dataset, but with a strong affinity to only one corpus. This is a function of how much higher their frequency is in that corpus than in the dataset.

.nlp.compareCorpora

Terms' comparative affinities to two corpora

Syntax: `.nlp.compareCorpora[corpus1;corpus2]`

Where `corpus1` and `corpus2` are tables of lists of documents, returns a dictionary of terms and their affinity for `corpus2` over `corpus1`.

Enron CEO Jeff Skillings was a member of the Beta Theta Pi fraternity at Southern Methodist University (SMU). If we want to find secret fraternity code words used by the Betas, we can compare his fraternity emails (those containing *SMU* or *Betas*) to his other emails.

```
q)fraternity:jeffcorpus i:where (jeffcorpus[`text] like "*Betas*")|jeffcorpus[`text] like "*SMU*"
q)remaining:jeffcorpus til[count jeffcorpus]except i
q)summaries:key each 10#/:.nlp.compareCorpora[fraternity;remaining]
q)summaries 0  / summary of the fraternity corpus
`beta`homecoming`betas`smu`yahoo`groups`tent`reunion`forget`crowd
q)summaries 1  / summary of the remaining corpus
`enron`jeff`business`information`please`market`services`energy`management`company
```

Comparing documents

This function allows you to calculate the similarity of two different documents. It finds the keywords that are present in both the corporas, and calculates the cosine similarity.

.nlp.compareDocs

Cosine similarity of two documents

Syntax: `.nlp.compareDocs[dict1;dict2]`

Where `dict1` and `dict2` are dictionaries that consist of the document's keywords, returns the cosine similarity of two documents.

Given the queried email defined above, and a random email from the corpus, we can calculate the cosine similarity between them.

```
q)queryemail2:jeffcorpus[rand count jeffcorpus]
q).nlp.compareDocs[queryemail`keywords;email2`keywords]
0.1163404
```

Comparing documents to corpus

.nlp.compareDocToCorpus

Cosine similarity between a document and other documents in the corpus

Syntax: `.nlp.compareDocToCorpus[keywords;idx]`

Where

- `keywords` is a list of dictionaries of keywords and coefficients
- `idx` is the index of the feature vector to compare with the rest of the corpus

returns as a float the document's significance to the rest of the corpus.

Comparing the first chapter with the rest of the book:

```
q).nlp.compareDocToCorpus[corpus`keywords;0]
0.03592943 0.04720108 0.03166343 0.02691693 0.03363885 0.02942622 0.03097797 0.04085023 0.04321152 0
```

Clustering

The NLP library contains a variety of clustering algorithms, with different parameters and performance characteristics. Some of these are very fast on large data sets, though they look only at the most salient features of each document, and will create many small clusters. Others, such as bisecting k-means, look at all features present in the document, and allow you to specify the number of clusters. Other parameters can include a threshold for the minimum similarity to consider, or how many iterations the algorithm should take to refine the clusters. Some clustering algorithms are randomized, and will produce different clusters every time they are run. This can be very useful, as a data set will have many possible, equally valid, clusterings. Some algorithms will put every document in a cluster, whereas others will increase cluster cohesion by omitting outliers.

Clusters can be summarized by their centroids, which are the sum of the feature vectors of all the documents they contain.

Markox Cluster algorithm

MCL clustering, which takes document similarity as its only parameter other than the documents. This algorithm first generates an undirected graph of documents by classifying document pairs as related or unrelated, depending on whether their similarity exceeds a given threshold. If they are related, an edge will be created between these documents. Then it runs a graph-clustering algorithm on the dataset.

.nlp.cluster.MCL

Cluster a subcorpus using graph clustering

Syntax: `.nlp.cluster.similarity[document;min;sample]`

Where

- `document` is a table of documents
- `min` is the minimum similarity (float)
- `sample` is whether a sample of `sqrt[n]` documents is to be used (boolean)

returns as a list of longs the document's indexes, grouped into clusters.

Cluster 2603 of Jeff Skillings emails, creating 398 clusters with the minimum threshold at 0.25:

```
q)clusterjeff:.nlp.cluster.similarity[jeffcorpus;0.25;0b]
q)count clusterjeff
398
```

Summarizing Cluster algorithm

This clustering algorithm finds the top ten keywords in each document, finds the average of these keywords and determines the top keyword. This is set to be the centroid and therefore finds the closest document. This process is repeated until the number of clusters are found.

.nlp.cluster.summarize

A clustering algorithm that works like many summarizing algorithms, by finding the most representative elements, then subtracting them from the centroid and iterating until the number of clusters has been reached

Syntax: `.nlp.cluster.summarize[docs;noOfClusters]`

Where

- `docs` is a list of documents or document keywords (table or list of dictionaries)
- `noOfClusters` is the number of clusters to return (long)

returns the documents' indexes, grouped into clusters.

```
q).nlp.cluster.summarize[jeffcorpus;30]

0 31 47 127 361 431 513 615 724 786 929 933 1058..
1 40 44 189 507 514 577 585 746 805 869 1042..
2 3 4 6 7 9 10 13 16 17 19 20 22 23 24 28 33 34..
5 27 30 39 393 611 641 654 670 782 820 1358..
8 73 147 427 592 660 743 794 850
11 26 113 236 263 280 281 340 391 414 429 478..
12 14 38 43 49 52 89 173 232 278 325 328
15 18 21 25 32 45 100 119 168 202 285 298..
29 159 386 430 459 499 508 597 659 731
68 83 105 132 141 152 177 182 185 226 257..
78 91 219 225 231 239 244 255 401 477 524 551..
```

K-means clustering

Given a set of documents, K-means clustering aims to partition the documents into a number of sets. Its objective is to minimize the residual sum of squares, a measure of how well the centroids represent the members of their clusters.

.nlp.cluster.kmeans

K-means clustering for documents

Syntax: `.nlp.cluster.kmeans[docs;k;iters]`

Where

- `docs` is a table or a list of dictionaries
- `k` is the number of clusters to return (long)
- `iters` is the number of times to iterate the refining step (long)

returns the document's indexes, grouped into clusters.

Partition *Moby Dick* into 15 clusters; we find there is one large cluster present in the book:

```
q)clusters:.nlp.cluster.kmeans[corpus;15;30]
q)count each clusters
32 9 13 9 12 5 12 8 6 8 7 11 11 5 2
```

Bisecting K-means

Bisecting K-means adopts the K-means algorithm and splits a cluster in two. This algorithm is more efficient when *k* is large. For the K-means algorithm, the computation involves every data point of the data set and *k* centroids. On the other hand, in each bisecting step of Bisecting K-means, only the data points of one cluster and two centroids are involved in the computation. Thus the computation time is reduced. Secondly, Bisecting K-means produce clusters of similar sizes, while K-means is known to produce clusters of widely differing sizes.

.nlp.cluster.bisectingKmeans

The Bisecting K-means algorithm uses K-means repeatedly to split the most cohesive clusters into two clusters

Syntax: `.nlp.cluster.bisectingKmeans[docs;k;iters]`

Where

- `docs` is a list of document keywords (table or list of dictionaries)
- `k` is the number of clusters (long)
- `iters` is the number of times to iterate the refining step

returns, as a list of lists of longs, the documents' indexes, grouped into clusters.

```
q)count each .nlp.cluster.bisectingKMeans[corpus;15;30]
8 5 13 5 12 8 10 10 1 12 5 15 1 37 8
```

Radix algorithm

The Radix clustering algorithms are a set of non-comparison, binning-based clustering algorithms. Because they do no comparisons, they can be much faster than other clustering algorithms. In essence, they cluster via topic modeling, but without the complexity.

Radix clustering is based on the observation that Bisecting K-means clustering gives the best cohesion when the centroid retains only its most significant dimension, and inspired by the canopy-clustering approach of pre-clustering using a very cheap distance metric.

At its simplest, Radix clustering just bins on most significant term. A more accurate version uses the most significant *n* terms in each document in the corpus as bins, discarding infrequent bins. Related terms can also be binned, and documents matching some percent of a bins keyword go in that bin.

Hard Clustering

Hard Clustering means that each datapoint belongs to a cluster completely or not.

.nlp.cluster.fastradix

Uses the Radix clustering algorithm and bins by the most significant term

Syntax: `.nlp.cluster.fastradix[docs;numOfClusters]`

Where

- `docs` is a list of documents or document keywords (table or a list of dictionaries)
- `numOfClusters` is the number of clusters (long)

returns a list of list of longs, the documents' indexes, grouped into clusters.

Group Jeff Skilling's emails into 60 clusters:

```
q)count each .nlp.cluster.radix1[jeffcorpus;60]
15 14 10 9 8 13 9 8 8 6 5 6 6 8 5 6 5 4 4 4 4 4 4 8 4 5 4 4 5 4 4 4 3 3 3 3 3..
```

Soft Clustering

In Soft Clustering, a probability or likelihood of a data point to be in a clusters is assigned. This mean that some clusters can overlap.

.nlp.cluster.radix

Uses the Radix clustering algorithm and bins are taken from the top 3 terms of each document

Syntax: `.nlp.cluster.radix[docs;numOfClusters]`

Where

- `docs` is a list of documents or document keywords (table or a list of dictionaries)
- `numOfClusters` is the number of clusters (long), which should be large to cover the substantial amount of the corpus, as the clusters are small

returns the documents' indexes (as a list of longs), grouped into clusters.

Group Jeff Skilling's emails into 60 clusters:

```
q)count each .nlp.cluster.radix2[jeffcorpus;60]
9 7 6 7 10 12 6 5 5 5 6 8 6 5 8 5 6 5 5 5 6 7 5 5 5 6 9 6 5 5 9 5 5 8 17 7 37.
```

Cluster cohesion

The cohesiveness of a cluster is a measure of how similar the documents are within that cluster. It is calculated as the mean sum-of-squares error, which aggregates each document's distance from the centroid. Sorting by cohesiveness will give very focused clusters first.

.nlp.cluster.MSE

Cohesiveness of a cluster as measured by the mean sum-of-squares error

Syntax: `.nlp.cluster.MSE x`

Where `x` is a list of dictionaries which are a document's keyword field, returns as a float the cohesion of the cluster.

```
q)/16 emails related to donating to charity
q)charityemails:jeffcorpus where jeffcorpus[`text] like "*donate*"
q).nlp.cluster.MSE charityemails`keywords
0.1177886

q)/10 emails chosen at random
q).nlp.cluster.MSE (-10?jeffcorpus)`keywords
0.02862244
```

Grouping documents to centroids

When you have a set of centroids and you would like to find out which centroid is closest to the documents, you can use this function.

.nlp.cluster.groupByCentroid

Documents matched to their nearest centroid

Syntax: `.nlp.cluster.matchDocswithCentroid[centroid;docs]`

Where

- `centroid` is a list of the centroids as keyword dictionaries
- `documents` is a list of document feature vectors

returns, as a list of lists of longs, document indexes where each list is a cluster.

Matches the first centroid of the clusters with the rest of the corpus:

```
q).nlp.cluster.groupByCentroids[[corpus clusters][0][`keywords];corpus`keywords]
0 23 65 137
1 5 14 45 81
2 6 7 13 15 16 17 19 20 21 26 27 31 40 44 47 48 49 50 54 57 58 62 63 66 67 68..
3 9 10
,4
8 51 55 95 96 108 112 117 129 132 136 146 148
11 12
,18
22 25
,24
28 53 61 72 82 83 86 91 113 130 147
,29
,30
32 33 79 98 104 105 107 131
34 97
35 37 38 39 41 42
36 133 149
43 60 64 74 106 115
```

Finding outliers, and representative documents

The *centroid* of a collection of documents is the average of their feature vectors. As such, documents close to the centroid are representative, while those far away are the outliers. Given a collection of documents, finding outliers can be a quick way to find interesting documents, those that have been mis-clustered, or those not relevant to the collection.

The emails of former Enron CEO Ken Lay contain 1124 emails with a petition. Nearly all of these use the default text, only changing the name, address and email address. To find those petitions which have been modified, sorting by distance from the centroid gives emails where the default text has been completely replaced, added to, or has had portions removed, with the emails most heavily modified appearing first.

.nlp.compareDocToCentroid

Cosine similarity of a document and a centroid, subtracting the document from the centroid

Syntax: `.nlp.compareDocToCentroid[centroid;document]`

Where

- `centroid` is the sum of all documents in a cluster which is a dictionary
- `document` is a document in a cluster which is a dictionary

returns the cosine similarity of the two documents as a float.

```
q)petition:laycorpus where laycorpus[`subject] like "Demand Ken*"
q)centroid:sum petition`keywords
q).nlp.compareDocToCentroid[centroid]each petition`keywords
0.2374891 0.2308969 0.2383573 0.2797052 0.2817323 0.3103245 0.279753 0.2396462 0.3534717 0.369767
q)outliers:petition iasc .nlp.compareDocToCentroid[centroid]each petition`keywords
```

Searching

Searching can be done using words, documents, or collections of documents as the query or dataset. To search for documents similar to a given document, you can represent all documents as feature vectors using TF-IDF, then compare the cosine similarity of the query document to those in the dataset and find the most similar documents, with the cosine similarity giving a relevance score.

The following example searches using `.nlp.compareDocs` for the document most similar to the below email where former Enron CEO Jeff Skilling is discussing finding a new fire chief.

```
q)queryemail:first jeffcorpus where jeffcorpus[`text] like "Fire Chief Committee*"
q)-1 queryemail`text;
q)mostsimilar:jeffcorpus first 1_idesc .nlp.compareDocs[queryemail`keywords]each jeffcorpus`keywords

Select Comm AGENDA - Jan 25-Febr 1

Houston Fire Chief Selection Committee Members: Jeff Skilling - Chairperson,
Troy Blakeney, Gerald Smith, Roel Campos and James Duke.

Congratulations selection committee members! We have a very important and
exciting task ahead of us.

On the agenda for the next week are two important items - (1) the Mayor's
February 1 news conference announcing the Houston Fire Chief selection
committee and its members; and (2) coordination of an action plan, which we
should work out prior to the news conference.

News Conference specifics:
speakers - Mayor Brown and Jeff Skilling
in attendance - all selection committee members
location - Fire Station #6, 3402 Washington Ave.
date - Thursday, February 1, 2001
time - 2pm
duration - approximately 30 minutes

I'd like to emphasize that it would be ideal if all selection committee
members were present at the news conference.

I will need bios on each committee member emailed to me by close of business
Monday, January 29, 2001. These bios will be attached to a press release the
Mayor's Office is compiling.

Coordination of action plan:
Since we have only 1 week between now and the news conference, Jeff has
proposed that he take a stab at putting together an initial draft. He will
then email to all committee members for comments/suggestions and make changes
accordingly. Hope this works for everyone - if not, give me a call
(713)-345-4840.

Thanks,
Lisa
```

Explaining similarities

For any pair of documents or centroids, the list of features can be sorted by how much they contribute to the similarity.

This example compares two of former Enron CEO Jeff Skilling's emails, both of which have in common the subject of selecting Houston's next fire chief.

.nlp.explainSimilarity

Syntax: `.nlp.explainSimilarity[doc1;doc2]`

Where `doc1` and `doc2` are dictionaries consisting of their associated documents' keywords, returns a dictionary of how much of the similarity score each token is responsible for.

```
q)10#.nlp.explainSimilarity . jeffcorpus[`keywords]568 358
fire     | 0.2588778
roel     | 0.1456685
committee| 0.1298068
mayor    | 0.1295087
station  | 0.09342764
chief    | 0.06948782
select   | 0.04325209
important| 0.03838308
members  | 0.03530552
plan     | 0.02459828
```

Sentiment analysis

Using a pre-built model of the degrees of positive and negative sentiment for English words and emoticons, as well as parsing to account for negation, adverbs and other modifiers, sentences can be scored for their negative, positive and neutral sentiment. The model included has been trained on social-media messages.

.nlp.sentiment

Sentiment of a sentence

Syntax: `.nlp.sentiment x`

Where `x` is string or a list of strings, returns a dictionary or table containing the sentiment of the text.

An run of sentences from *Moby Dick*:

```
q).nlp.sentiment("Three cheers,men--all hearts alive!";"No,no! shame upon all cowards-shame upon the
compound   pos       neg       neu
-----------------------------------------
0.7177249  0.5996797 0         0.4003203
-0.8802318 0         0.6910529 0.3089471
```

Importing and parsing MBOX files

The MBOX file is the most common format for storing email messages on a hard drive. All the messages for each mailbox are stored as a single, long, text file in a string of concatenated e-mail messages, starting with the *From* header of the message. The NLP library allows the user to import these files and creates a kdb+ table.

Column	*Type*	*Content*
`sender`	list of characters	The name and email address of the sender
`to`	list of characters	The name and email address of the reciever/recievers
`date`	timestamp	The date
`subject`	list of characters	The subject of the email
`text`	list of characters	The original text of the email
`contentType`	list of characters	The content type of the email
`payload`	list of characters or dictionaries	The payload of the email

.nlp.loadEmails

An MBOX file as a table of parsed metadata

Syntax: `.nlp.loadEmails x`

Where `x` is a string of the filepath, returns a table.

```
q)email:.nlp.email.getMbox["/home/kx/nlp/datasets/tdwg.mbox"]
q)cols email
`sender`to`date`subject`contentType`payload`text
```

.nlp.email.getGraph

Graph of who emailed whom, with the number of times they emailed

Syntax: `.nlp.email.getGraph x`

Where `x` is a table (result from `.nlp.email.i.parseMbox`), returns a table of to-from pairing.

```
q).nlp.email.getGraph[emails]

sender                          to                                volume
-------------------------------------------------------------------------
```

Donald.Hobern@csiro.au	tdwg-img@lists.tdwg.org	1
Donald.Hobern@csiro.au	tdwg@lists.tdwg.org	1
Donald.Hobern@csiro.au	vchavan@gbif.org	1
RichardsK@landcareresearch.co.nz	tdwg-img@lists.tdwg.org	1
Robert.Morris@cs.umb.edu	Tdwg-tag@lists.tdwg.org	1
Robert.Morris@cs.umb.edu	tdwg-img@lists.tdwg.org	1
mdoering@gbif.org	lee@blatantfabrications.com	1
mdoering@gbif.org	tdwg-img@lists.tdwg.org	1
morris.bob@gmail.com	tdwg-img@lists.tdwg.org	1
ram@cs.umb.edu	RichardsK@landcareresearch.co.nz	1
ram@cs.umb.edu	tdwg-img@lists.tdwg.org	2
ricardo@tdwg.org	a.rissone@nhm.ac.uk	3
ricardo@tdwg.org	leebel@netspace.net.au	3
ricardo@tdwg.org	tdwg-img@lists.tdwg.org	3
ricardo@tdwg.org	tdwg-lit@lists.tdwg.org	3
ricardo@tdwg.org	tdwg-obs@lists.tdwg.org	3
ricardo@tdwg.org	tdwg-process@lists.tdwg.org	3
ricardo@tdwg.org	tdwg-tag@lists.tdwg.org	3
ricardo@tdwg.org	tdwg-tapir@lists.tdwg.org	3
roger@tdwg.org	Tdwg-img@lists.tdwg.org	1

Parsing emails from a string format

.nlp.email.parseMail

Parses an email in string format

Syntax: `.nlp.email.parseMail x`

Where `x` is an email in a string format, returns a dictionary of the headers and content.

```
q)table:.nlp.email.parseMail emailString
q)table
`headers`content
```

Useful functions

These are functions that extract elements of the text that can be applied to NLP algorithms, or that can help you with your analysis.

.nlp.findTimes

All the times in a document

Syntax: `.nlp.findTimes x`

Where `x` is a string, returns a general list:

1. time
2. text of the time (string)
3. start index (long)
4. index after the end index (long)

```
q).nlp.findTimes "I went to work at 9:00am and had a coffee at 10:20"
09:00:00.000 "9:00am" 18 24
10:20:00.000 "10:20"  45 50
```

.nlp.findDates

All the dates in a document

Syntax: `.nlp.findDates x`

Where `x` is a string

returns a general list:

1. start date of the range
2. end date of the range
3. text of the range
4. start index of the date (long)
5. index after the end index (long)

```
q).nlp.findDates "I am going on holidays on the 12/04/2018 to New York and come back on the 18.04.20
```

```
2018.04.12 2018.04.12 "12/04/2018" 30 40
2018.04.18 2018.04.18 "18.04.2018" 74 84
```

.nlp.getSentences

A document partitioned into sentences.

Syntax: `.nlp.getSentences x`

Where `x` is a dictionary or a table of document records or subcorpus, returns a list of strings.

```
/finds the sentences in the first chapter of MobyDick
q) .nlp.getSentences corpus[0]
```

.nlp.loadTextFromDir

All the files in a directory, imported recursively

Syntax: `.nlp.loadTextFromDir x`

Where `x` the directory's filepath as a string, returns a table of filenames, paths and texts.

```
q).nlp.loadTextFromDir["./datasets/maildir/skilling-j"]

fileName path                                              text                ..
-------------------------------------------------------------------------------..
1.       :./datasets/maildir/skilling-j/_sent_mail/1.   "Message-ID: <1461010..
10.      :./datasets/maildir/skilling-j/_sent_mail/10.  "Message-ID: <1371054..
100.     :./datasets/maildir/skilling-j/_sent_mail/100. "Message-ID: <47397.1..
101.     :./datasets/maildir/skilling-j/_sent_mail/101. "Message-ID: <2486283..
```