ScaleText: The Design of a Scalable, Adaptable and User-Friendly Document System for Similarity Searches

Digging for Nuggets of Wisdom in Text

Jan Rygl[1], Petr Sojka[2], Michal Růžička[2], and Radim Řehůřek[1]

[1] RaRe Technologies, {jimmy,radim}@rare-technologies.com
[2] Faculty of Informatics, Masaryk University, Brno, Czech Republic
sojka@fi.muni.cz, ORCID: 0000-0002-5768-4007
and mruzicka@mail.muni.cz, ORCID: 0000-0001-5547-8720

Abstract. This paper describes the design of a new ScaleText system aimed at scalable semantic indexing of heterogeneous textual corpora. We discuss the design decisions that lead to a modular system architecture for indexing and searching using semantic vectors of document segments – nuggets of wisdom. The prototype system implementation is evaluated by applying Latent Semantic Indexing (LSI) on the Enron corpus. And the Bpref measure is used to automate comparing the performance of different algorithms and system configurations.

Key words: ScaleText, vector space modelling, Latent Semantic Indexing, LSI, machine learning, scalable search, search system design, text mining

1 Introduction

Today's growing information overload dictates the need for effective semantic searching in custom datasets, such as emails, texts in corporation information systems and knowledge bases, Wikipedia, web browsing history, and in personal information space. Such a search service gives working professionals a competitive advantage, and allows them to have relevant information at their fingertips.

Content semantics indexing is the king for document indexing and filtering large volumes of textual data. Its relevance search goes beyond string, word or phrase indexing.

In this paper we describe the design of ScaleText, a system that aims to meet the demands of the working professional's information search needs. The design imperatives are:

Scalability: with the size of today's document collections, efficiency is a primary concern, allowing low latency responses.

Adaptability: since no size fits all, the system should be easily customizable and tunable for any given application purpose.

Relevance: search precision could be improved by clever semantic representations of the meanings of indexed texts. It is both necessary and desirable to find highly relevant document chunks.

Implementation **C**larity: the implementation should be written with ease of maintenance in mind.

Simplicity: keep it simple stupid, yet provide the functionality needed.

Having SARICS in mind during the design phase lays the foundation for a system capable of meeting the *big data* needs of many enterprises.

This paper is structured as follows: Section 2 evaluates state-of-the-art systems and approaches. In Sections 3 and 4 the top-level system architecture is described, with processing pipelines for indexing and searching, the main components of ScaleText. Section 5 describes automatic evaluation framework for human-unassisted comparison of implementations and configurations of our prototype system on the ground truth of Enron corpus. We conclude with an overview of our contributions along with our work plans for the future in Section 6.

2 The State of the Art in Semantic Document Processing

There has been a noticeable drift away from an emphasis on keyword-based statistics such as term frequency–inverse document frequency (TfIdf) weighting to semantic-based methods such as Latent Semantic Indexing LSI [4] or semantic vectors learned by deep learning [7]. Improving the relevance of search results and scalability are also current design imperatives, given that semantic searches are often performed during web browsing [10] and even at each keystroke as is the case with Google Instant.

Approaches to Semantics The key to better relevance is some sort of sense representation of words, phrases, sentences, paragraphs and documents. We can have either (i) *a discrete representation* of meaning, which can be based on knowledge-based representations such as WordNet, BabelNet, Freebase or Wikipedia, or (ii) *a smooth representation* based on a distributional hypothesis, e.g. representing meanings as word, phrase, sentence, . . . embeddings [7] which are learned from the language used in big corpora by unsupervised, deep learning approaches, or by topic modeling [1].

Existing Frameworks and Systems There are already frameworks that support building smooth semantic models such as Gensim [8]. One system that builds on semantic document models is Kvasir [10] which supports instant searching in the web browser, thereby stressing the need for speed and relevance.

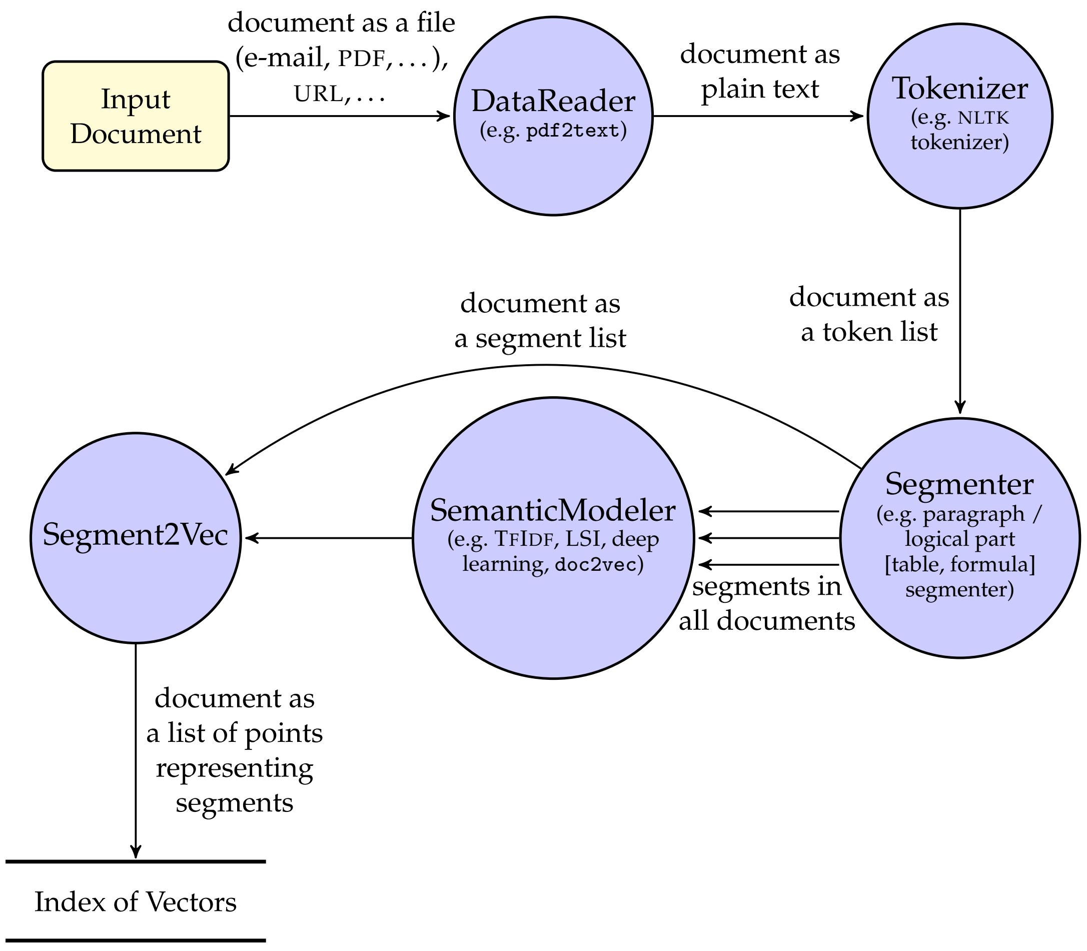
Input Document
document as a file (e-mail, PDF, ...), URL, ...
DataReader (e.g. pdf2text)
document as plain text
Tokenizer (e.g. NLTK tokenizer)
document as a token list
Segmenter (e.g. paragraph / logical part [table, formula] segmenter)
segments in all documents
SemanticModeler (e.g. TFIDF, LSI, deep learning, doc2vec)
document as a segment list
Segment2Vec
document as a list of points representing segments
Index of Vectors

Fig. 1: Data flow diagram of document indexing in ScaleText

As there are still many open questions to be clarified in text semantics representation, our ScaleText system architecture has been designed to be modular, based on a set of components with a defined purpose and communication interfaces. Different module implementations give the system extra flexibility. Indexing and searching, as main components of ScaleText, are outlined in the following sections.

3 Indexing: Storing Document Chunks as Points in Vector Space

ScaleText introduces a flexible data processing pipeline for document indexing, leading to semantic document representations in a vector space. The overall scheme of document transformations in the indexing workflow is depicted in Figure 1.

In the course of our SARICS imperatives, ScaleText is flexible what format of document is accepted on its input. Raw input documents are read from their primary resources outside ScaleText by the `DataReader` module. Different implementations of the `DataReader` component can be used to provide data from arbitrary data sources such as log files, data files, binary streams, etc. and in various formats such as plain text documents, PDF documents, emails with attachments. Every raw input document is transformed into plain full text, and given a unique document identifier (`doc_id`). All data are made persistent in the `Storage` module.

The `Tokenizer` module processes plain text with a standard linguistic pipeline: (i) tokenization, part of speech tagging, and (ii) phrase detection all take place at this stage. The next stage in the document processing is segmentation in the `Segmenter` module. Plain text is cut into a list of small, meaningful segments, called *nuggets*. Nuggets usually take the form of a triplet consisting of a document identifier (`doc_id`), a segment identifier (`seg_id`) and a list of tokens (`tokens`) comprising a paragraph or equation or another logic element with semantic meaning, extracted from the document plain text. Segmenting a document into nuggets is one of the key ScaleText design ideas that facilitates the semantic indexing of textual data.

Having nuggets of all documents enables us to build a semantic model of an indexed dataset. To represent the semantics of the documents we can use distributional semantics modeling, topic modeling methods, deeply learned representations or LSI. The semantic document model can be rebuilt and retrained at any time from the currently indexed nuggets. In our default implementation, the TfIdf (Term frequency–Inverse document frequency) matrix is computed from the `tokens` of all nuggets, followed by LSI, which results in the projection of nuggets into a latent semantic subspace.

Thus, every document is represented as a list of semantic vectors of nuggets. Both the model and the vectors are persistently stored in a database and used during the search.

4 Document Similarity Search: Digging for Nuggets of Wisdom

The indexed dataset is used for similarity searching. To pursue the gold mining metaphor, gold nuggets are washed with different gold mining techniques. The overall schema of the search procedure is depicted in Figure 2.

Query representation as nuggets The query document is segmented into nuggets.

Querying and scoring We build a set of hit lists between the query and all database nuggets. The hit lists are in the form of triplets of `doc_id` and `seg_id`, referring to a database nugget, and a `score` – a numerical represention of its semantic similarity to the query nugget. Cosine similarity is the implicit similarity measure.

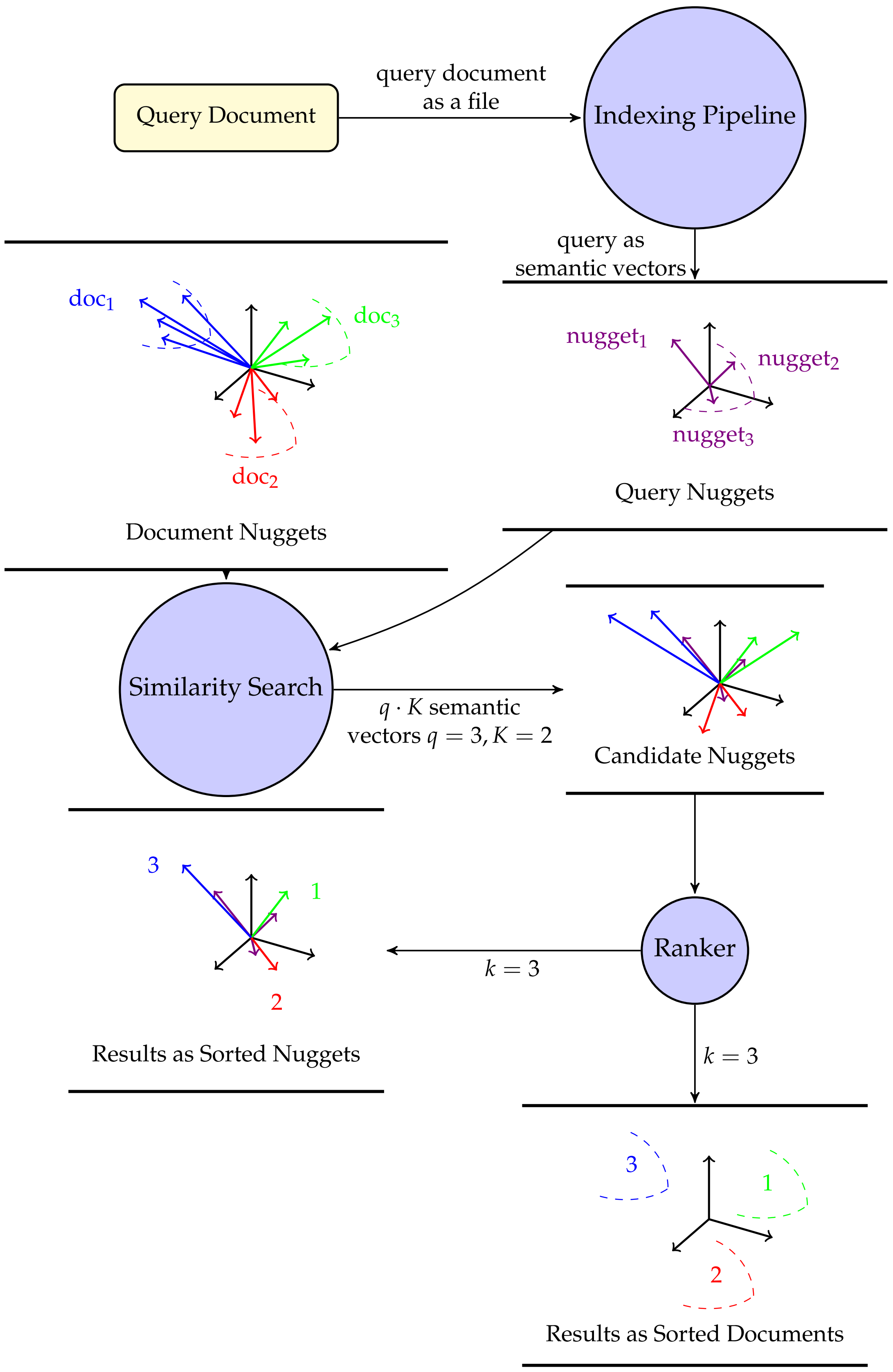

Fig. 2: Data flow diagram of document similarity search in ScaleText. q is the number of query nuggets, K is the number of best nugget candidates for each query nugget, and k is the number of desired final results

Hit merging and sorting strategies The final step of the search procedure is merging and sorting the nuggets found.

The implicit strategy sorts the nuggets by the value of the similarity score only. More advanced strategies can boost scores of the nuggets, for example, based on the number of matching nuggets belonging to the same document. In this case, the score of document nuggets also depends on the overall coverage of the document by the query nuggets.

Document-based result sorting When users prefer whole documents to individual nuggets, the results can be the documents sorted by an aggregation of matched nugget scores per document. There are various aggregation possibilities, such as arithmetic mean, maximum, or sum normalized by the document length.

Scoring and sorting depend on data and application goals but ScaleText provides a flexible architecture to achieve them: a separate `Ranker` module for reordering the results – found nuggets – allows a suitable results sorting strategy to be implemented, tailored to a particular user's needs. The variability of nugget mining strategies that ScaleText design offers provides an opportunity to fine-tune the system with respect to the needs and specifics of a particular project and dataset.

5 Automatic Evaluation Framework for System Modules

As Figures 1 and 2 show, different implementations of the modules in ScaleText data processing workflow can be used. It is necessary to evaluate the system performance of different configurations. In order to achieve a quick verification of ScaleText design ideas and rapid prototyping, we needed a fully automatic, fast and human-unassisted evaluation procedure to compare exchangeable modules in the architecture. We consequently built a ScaleText prototype on top of existing libraries such as Gensim [8] and Spotify Annoy[3], using agile development techniques.

To measure and compare performance of the system modules we needed a dataset with ground truth for a set of queries.

Evaluation dataset We used TREC 2010 Legal track version [3, chapter 2] of the Enron dataset [9]: 455,449 messages plus 230,143 attachments form the 685,592 documents of the TREC 2010 Legal Track collection.

Ground truth dataset To build our ground truth we exploited the availability of 2,720 documents out of the Enron dataset which had an assessed relevance to the Learning task [3, chapter 4.1] that consisted of 8 topics. Every ground truth document was labeled as relevant or irrelevant to each of the topics.

[3] `https://github.com/spotify/annoy`

Query dataset ScaleText uses whole documents as queries for similarity searches. For automatic prototype evaluation we used our ground truth documents, i.e. documents with known relevance to the topics, as the queries.

Bpref@*k* evaluation metric We used the Bpref measure to evaluate the instance performance of the ScaleText system. It is a cheap and rigorous measure of the performance effect of module changes which does not need to assume the completeness of the relevance judgments in the evaluation dataset.

However, the original Bpref proposal [2] was found [6] not to correspond to the actual `trac_eval`[4] implementation of Bpref. Furthermore, the `trac_eval` implementation still does not work correctly on result lists where the number k of inspected results (Bref@k) is lower than the number of relevant results by ground truth. To cope with this we finally implemented Bpref@k as follows:

$$\text{Bpref@}k = \frac{1}{\min(R,k)} \sum_{r} \left(1 - \frac{\min\left(\text{number of } n \text{ ranked higher than } r, R\right)}{\min(N,R)}\right),$$

where R is the number of documents relevant to the topic, N is the number of documents irrelevant to the topic, k is the maximal number of inspected results, and "number of n ranked higher than r" is the number of irrelevant documents (according to the judgment) ranked higher than the relevant (according to the judgment) document r that is being processed in the step.

Evaluation procedure For a given query, every document in the result list is automatically classified as (i) *relevant*, if the document is in our ground truth and the relevance assessment for the topic is the same as for the query document, (ii) *irrelevant*, if it is in our ground truth but has a different relevance assessment from the query, (iii) *unknown* otherwise.

Table 1 provides examples of the evaluations of different ScaleText configurations and ranking strategies. It is important to note that unsupervised machine learning techniques have been used, i.e. Enron seed set [2] was *not* used to train the model on labeled data. This evaluation procedure is useful for a fast, automatic, human-unassisted evaluation of system configuration effectiveness, as it shows the differences among configurations. E.g., from the Bpref@100 values it is clear that setting the number of LSI features as high as 500 already degrades performance. Also, TfIdf+LSI configuration gives significantly better results than using only TfIdf weighting.

6 Conclusion and Future Work

We have designed a ScaleText system for scalable semantic searching and indexing. The system has been designed with SARICS (Scalability, Adaptability, Relevance, Implementation, Clarity and Simplicity) in mind. Its architecture

[4] `http://trec.nist.gov/trec_eval/`

Table 1: ScaleText prototype evaluation on the Enron dataset via Bpref. The single metric value is the average of Bpref@100 over all the queries

document model	document ranking strategy	#features	avg. Bpref@100
TfIdf	maximum nugget score	100	0.0451
TfIdf+LSI	maximum nugget score	50	0.0460
TfIdf+LSI	maximum nugget score	100	0.0565
TfIdf+LSI	maximum nugget score	500	0.0358
TfIdf	average nugget score	100	0.0451
TfIdf+LSI	average nugget score	50	0.0460
TfIdf+LSI	average nugget score	100	0.0548
TfIdf+LSI	average nugget score	500	0.0358
TfIdf	normalized sum of nugget scores	100	0.0451
TfIdf+LSI	normalized sum of nugget scores	50	0.0460
TfIdf+LSI	normalized sum of nugget scores	100	0.0534
TfIdf+LSI	normalized sum of nugget scores	500	0.0358

allows for massive parallelization of both crucial operations – indexing and searching with low latency, yet allowing easy maintenance and pluggable modules for semantic indexing (LSI, distributive semantics) and searching (*k*-NN search, new techniques for searching in feature vector spaces).

We have implemented ScaleText in Python for easy prototyping and maintenance. Semantic modeling algorithms use a high-performance implementation of Gensim.

We have designed a mechanism for evaluating pluggable system modules. The ground truth Enron database with Bpref metric has allowed us to quickly and automatically measure the performance of the system and compare the module effectiveness of different system configurations.

In subsequent work we will evaluate different vector space representations of documents and prepare a methodology for configuring ScaleText to meet document system search demands of both the research community and industry.

We have several research questions in our sights:

– **Word disambiguation in context:** current methods represent a word in the vector space as the centroid of its different meanings. We want to evaluate an approach based on random walks through texts so as to distinguish the representation of words in context.
– **Compositionality of segment representation:** semantic vectors representing the meaning of segments should reflect compositionality of meaning of its parts, e.g. words, phrases and sentences.
– **Representation of narrativity:** we may represent narrative text qualities [5] as a *trajectory* of words or nuggets in vector space, e.g. document representation may be a trajectory instead of a point.

Acknowledgments. The authors thank all proofreaders for their detailed commentary and suggestions for improvements. Funding by TA ČR Omega grant TD03000295 is gratefully acknowledged.

References

1. Blei, D.M.: Probabilistic topic models. Commun. ACM 55(4), 77–84 (Apr 2012), `http://doi.acm.org/10.1145/2133806.2133826`
2. Buckley, C., Voorhees, E.M.: Retrieval evaluation with incomplete information. In: Proc. of the 27th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval. pp. 25–32. SIGIR '04, ACM, New York, NY, USA (2004), `http://doi.acm.org/10.1145/1008992.1009000`
3. Cormack, G.V., Grossman, M.R., Hedin, B., Oard, D.W.: Overview of the TREC 2010 legal track. In: Proc. 19th Text REtrieval Conference. pp. 1–45. National Institute of Standards and Technology, Gaitherburg, MD (2010), `http://trec.nist.gov/pubs/trec19/papers/LEGAL10.OVERVIEW.pdf`
4. Deerwester, S.C., Dumais, S.T., Landauer, T.K., Furnas, G.W., Harshman, R.A.: Indexing by latent semantic analysis. Journal of the American Society of Information Science 41(6), 391–407 (1990)
5. Hoenkamp, E., Bruza, P., Song, D., Huang, Q.: An Effective Approach to Verbose Queries Using a Limited Dependencies Language Model. In: Azzopardi, L., Kazai, G., Robertson, S.E., Rüger, S.M., Shokouhi, M., Song, D., Yilmaz, E. (eds.) ICTIR. Lecture Notes in Computer Science, vol. 5766, pp. 116–127. Springer (2009), `http://dx.doi.org/10.1007/978-3-642-04417-5_11`
6. Kdorff: Bpreftreceval2006 (2007), `http://icb.med.cornell.edu/wiki/index.php/BPrefTrecEval2006`, Accessed: 2016-10-29
7. Mikolov, T., Sutskever, I., Chen, K., Corrado, G., Dean, J.: Distributed representations of words and phrases and their compositionality (2013), `http://arxiv.org/abs/1310.4546`, arXiv:1310.4546
8. Řehůřek, R., Sojka, P.: Software Framework for Topic Modelling with Large Corpora. In: Proceedings of LREC 2010 workshop New Challenges for NLP Frameworks. pp. 45–50. Valletta, Malta (2010), software available at `http://nlp.fi.muni.cz/projekty/gensim`
9. TREC version of EDRM Enron Dataset, version 2, Accessed: 2016-10-29, `http://www.edrm.net/resources/data-sets/edrm-enron-email-data-set-v2`
10. Wang, L., Tasoulis, S., Roos, T., Kangasharju, J.: Kvasir: Seamless integration of latent semantic analysis-based content provision into web browsing. In: Proc. of the 24th International Conference on World Wide Web. pp. 251–254. WWW '15 Companion, ACM, New York, NY, USA (2015), `http://doi.acm.org/10.1145/2740908.2742825`