Text Mining with Graph Databases:
Traversal of Persisted Token-Level Representations for Flexible On-Demand Processing

Thomas Efer
Natural Language Processing Group, Universität Leipzig, Germany
efer@informatik.uni-leipzig.de

Abstract: This contribution introduces a new text modelling paradigm that allows for on-demand Text Mining approaches without the need for pre-calculated statistics or data structures other than generic edge-typed adjacency information in a graph database. A simple use cases is presented, for which these new text storing and querying methods can prove beneficial. Furthermore a prototypical implementation is presented and analyzed, showing an adequate runtime behavior for interactive queries.

1 Introduction

Text Mining has a truly broad spectrum of applications, ranging from Business Intelligence to Plagiarism Detection, from Translation Memories to Topic Tracking. It has proven to be a reliable companion in the context of Information Retrieval (IR) tasks: Text Mining methods are employed to extract and structure various bits of knowledge from textual sources in order to enable the IR user and system to better find relevant information.

The World Wide Web has long become a central mass medium and Information Retrieval – a discipline dating back to the 1950es – has suddenly become a key technology. One, that has to rapidly adapt to new challenges posed by the growth of the Internet – both in terms of the amount of data, as well as its complexity. The following remarks from 2010 by Waitelonis et al. show how a change in usage patterns slowly became apparent:

Today, searching the World Wide Web in most cases turns out in looking for a specific item, which means that the user should know the item in advance. In the future internet, searching for information comes closer to the notion of ‘window shopping’ by means of exploratory and semantic search technologies.

[8]
Google’s introduction of their “Knowledge Graph”[7] not even two years later proved that an entity-centric knowledge base with a comprehensive set of modelled relations can become the flagship of search-augmenting technology at the world’s largest search engine company.

Structured exploration of heterogeneous sources gets more and more important, not least in Social Networks. What makes them so useful is that they give the user the freedom to define an individual context by accumulating lists of their friends, former employers, hobbies, interests, etc. and then deliver a custom-tailored stream of news with relevancy relative to that context. The user profiles work as filter but also as a way to let others discover new friends, job opportunities or hobbies without requiring an active search.

On the technological side, such systems have to deal with large data collections, quite frequent data insertions and many complex queries to create detailed web pages. Here a lot of innovation has taken place, mainly to exploit the extremely local scopes of interest: To satisfy the information need of any given user there is usually only a tiny (somehow coherent) fraction of the database to consider. Using a technology that relies on global operations (e.g by performing set intersection of all entries in order to obtain such a small fraction as a result) is not possible. This mindset of weighing local versus global aspects of data management and querying is not yet fully present in Text Mining.

Many methods still rely on global pre-calculations that make them inflexible with regard to changes and additions but also prevent ad-hoc queries on only portions of the data. Local contexts can not be easily defined - neither as a filter nor as a means of calculation speedup. This is where more flexible technologies need to be adopted so that Text Mining can hold pace with Information Retrieval developments.

2 Property Graphs as Data Model for Text

2.1 What is Text?

What is needed is a flexible, locality-aware representation of text with all its important aspects: sequence, structure, annotations and metadata. Layout and formatting of documents is not (yet) important for Text Mining concerns and can be left aside.

When DeRose et al. asked in 1990 “What is Text really?”[1], they came up with the most progressive model to that time, the “Ordered Hierarchy of Content
Objects” (OHCO), which made excellent use of the then-upcoming SGML standard. Also since the 1980es – so for decades now, the Text Encoding Initiative (TEI) is working on best practices for the detailed, (if wished for, even philological and scholarly deep) representation of documents. Their “TEI Guidelines P5 2.0” are an impressive 1636 pages long documentation of all possible considerations and caveats when dealing with digital representations of text of all sorts – alongside an XML vocabulary and grammar to deal with all those cases.

For XML data there are specialized databases, most prominently eXistdb\(^1\) and BaseX\(^2\) which enable an optimized structured access via XQuery. Yet they cannot make up for the fundamental flaws of the OHCO idea: The text being trapped in a single, fixed hierarchy. Pages have lines, paragraphs have sentences. Both hierarchies do overlap. Both should be treated equal but cannot since the document is a tree. Stand-off annotations via XPointers or lists with token offsets cannot solve this either and make for an annoying media disruption. And the overlap problem gets more and more pressing the more annotation layers are added to a text.

### 2.2 Graph Databases

In graph databases the relations between entries are first-class modelling constructs. They are called edges or relationships, while the records are called nodes or vertices. There are several kinds of graph databases. This paper follows the O’Reilly book “Graph Databases”\(^5\) in that it focuses on the “most popular variant of graph model”, the Property Graph. It is built from only a few very simple building blocks and separates the (extendable) schema from the data, what helps to make efficient queries.

In this model, so-called properties exist, which are key-value-pairs that can be attached to nodes and edges. Edges always connect exactly two nodes. They have a direction and a so-called label, that is used to distinguish between semantically different types of edges.

In [6] Rodriguez and Neubauer describe so-called Graph Traversals which are operations that can move along edges in the graph and can be steered to take certain paths according to the local environment at a given point, e.g. the value of node properties or the number of outgoing edges with a given label. They conclude that “Graphs are a flexible modeling construct that can be used to model a

\(^1\)http://http://exist-db.org/
\(^2\)http://basex.org/
domain and the indices that partition that domain into an efficient, searchable space.” It is important to know that traversal is the key query mechanism for graphs, but also that it is supported by value-indices for properties, that allow to “jump” to nodes or edges where a certain value (or a value within a given range) is present.

2.3 The Text Model in a Nutshell

For several use cases a rather complex graph schema has been designed and tested. In the scope of this article only the main aspects shall be presented. The following general “layers” can be identified that mainly interact with their adjoining layers:

- Metadata (documents, authors, connected places, genres, …)
- Structure (chapters, pages, lines, sentences, stanzas, speeches, quotations, …)
- Text (Types, Tokens, normalized Tokens)
- Lexical and linguistic information (dictionary entries, thesauri, part-of-speech, …)
- Concepts (Named Entities, user-specified clusters of vocabulary)

Fig. 1: A schematic overview of multiple hierarchies, the chain of tokens and clusters of types and normalized types – the text level model from documents down to words.
Figure 1 shows the most important part: An example of how nodes are connected to present the vocabulary (Types and their normalized versions) and running words in the text (Tokens). Note how the string value is only stored on the type level so that token nodes have a very small footprint and only provide an anchor for the traversal of their neighbourhood. Edge labels have been omitted for clarity.

On the linguistic level several well-known annotation formats can be merged and unified with graph structure as demonstrated in [2]. Dictionary entries can be modelled as single nodes and connected to the type of their lemma (and of mentioned grammatical variants). Organizational aspects such as user accounts and rights management can also be directly modelled within the same graph database and can be applied very fine-grained (up to the rather ridiculous case where access to a single token can be granted or forbidden).

3 Co-Occurrence Analysis as an Example Use Cases

When trying to grasp the connections between known or unknown vocabulary items it is common practice to look at the places where those words occur. Words that are often close together (or more generally spoken used “in the same context”) tend to share traits of meaning – a consideration often credited to Ferdinand de Saussure.

In Text Mining co-occurrence analyses count how often words occur next to each other or together in a larger unit - document, sentence or fixed-width word window. Most often the simple counting is not enough. More frequent words occur together more often by chance, why usually their relative frequencies are considered and inserted into falsifiable statistical hypotheses on the independence of their occurrence. With this or different approaches, many significance measures for co-occurrences can be employed. Many of them need semi-global knowledge, like “In how many different documents does this word occur at all”? These queries can be answered with graph traversals jumping from types to tokens, to documents and counting the unique ones. But for many use-cases of exploratory vocabulary analysis, the significance analysis can be skipped.

Figure 2 shows how frequent other relevant terms appear near a set of “interesting” words. The proximity is set to at most 3 words before and after the searched term. 3 The visual representations of all co-occurrants is reflecting their absolute frequency through tag size, which is is enough to enable the human eye to spot

3For more information on the analyzed documents see section 5.
differences, anomalies and common words in the neighborhood of more than on word. The image shown does not do full justice to the colored interactive (mouse hover sensitive) version. The wildcard search of words is for now realized with an external index but it would very well be possible to implement a Trie data structure with logarithmic access times to all words with a common beginning directly in the graph database!

Fig. 2: “TagPie”[3] visualization of co-occurring vocabulary for financial pivot terms “trade”, “market”, “stock” and one of either “rise” or “fall” (with suffixes wildcards).

The traversal to determine co-occurrences can be extended with filtering steps: Only regard occurrences that lie within a document by a specific author or that is connected to a certain genre. Only consider those documents that do not contain the other search terms but the right one more than 3 times. The possibilities are endless and the query remains reasonably complex. The context for co-occurrences can also be freely adjusted: A fixed word proximity the enclosure in a structural element (sentence or paragraph), a flexible word window that spans the next and previous three round (if such an annotation is present). Again: The query can be varied without the need for any re-calculation or physical creation of a fitting “subcorpus”. All this helps to build an explorative environment where the users can gradually refine their context of interest.
4 Prototype Implementation

The prototype is a pure backend system that provides a restful JSON-based API through which frontend applications such as search interfaces and visualizations can access the data. The runtime environment is Java but most code is written in JRuby, a Java implementation of the Ruby language that allows for the transparent usage of Java objects and libraries within Ruby scripts. This allows for rapid prototyping since no separate recompilation of altered or added code is needed. Also, Ruby comes with many tools for web service creation and expressive data transformation features that tremendously facilitate the development process compared to a pure Java solution.

The graph database is embedded through the well-documented, interface of the Tinkerpop stack, an industry standard for Java Graph Databases. The chosen database is Titan, a flexible system with a focus on scalability that can work with many storage backends, among them Big-Data ready column stores that can work on distributed cluster systems. To interact with the Database, the graph traversal library Pacer is used. It provides an imperative domain-specific query language in Ruby.

The software is developed as a zero-downtime system, where the database stays fully operational while importing new texts. Single API requests are executed in their own thread allowing a parallel execution of different queries. The API is accessible via standard HTTP request as well as via EventSource. Via the latter option the API can send intermediate results and query status information in realtime within a single permanently open HTTP response. That can greatly improve the perceived responsivity of user interfaces.

The prototype is currently transformed into a stable software package named “Kadmos” that is developed in the context of the BMBF-funded project eX-Change and that will be released as Open Source Software.  

---

4 http://jruby.org/  
5 http://tinkerpop.com/  
6 http://titan.thinkaurelius.com/  
7 http://xnlogic.github.io/pacer/  
8 http://www.w3.org/TR/eventsource/, also known as “Server Sent Events”  
9 The Phoenician, who – according to ancient myths – brought the alphabet to the Greeks.  
10 http://exchange.e-humanities.net/
5 Evaluation

For evaluation purposes the freely available corpus “Reuters-21578, Distribution 1.0” \(^{11}\) was used. It contains 21578 news articles (hence the name) and sums up to almost 2.6 million running tokens from a vocabulary of around 110.000 different types after naive whitespace-based tokenization and special character removal. The most frequent word, “the” occurs almost 120.000 times while about half of all words occur only once, which is consistent with Zipf’s law of exponential word frequency distributions.

Titan was configured to use the local Berkley DB\(^ {12}\) backend that can run in the same Java Virtual Machine and therefore minimizes communication latency influences on the measured response times. In this setup the single-threaded corpus import takes about half an hour on commodity hardware. This includes the assembling of the graph structure as well as the creation of a full text index for the embedded string values.

Figure 3 shows how the processing times needed to find the co-occurrences of a given word grow linearly with the number of its co-occurring words. Keeping in mind the word frequency distribution, this means that there are many words in the corpus where such a query is very fast and only few words where the query is relatively slow (with a maximum of 20 seconds for “the” in the experiment).\(^ {13}\)

In Text Mining it is often not required to process omnipresent words since they do not yield statistically useful information - sometimes the most frequent words are excluded from analyses, both for efficiency as well as for quality reasons – a process known as Vocabulary Pruning. For real-time systems the measured linearity could result in a simple formula: If the query response does not arrive within a fixed reasonable time period \(t\), it is probably too unspecific to yield the desired domain insights and can be discarded. Should the allowed time budget need further be constrained, one can then at least estimate how much of the corpus is lost as “inqueryable” in the filtering process.

\(^{11}\)Obtained from https://github.com/fergiemcdowall/reuters-21578-json
\(^{13}\)Stating an “average” response time is not feasible since it would at least require a symmetric distribution.
6 Conclusion and Outlook

This paper presented first steps towards new (or adapted) Text Mining methods that work on graph-structured text representations down to the token level. The availability and good performance of graph databases make their employment feasible which allows to keep such a representation persisted, preventing loss or recalculation expenses. They allow for live manipulations without downtime and an immediate reflection of the new state in statistics and query results.

Aside from the simple use-case presented here, also other areas have already been investigated, such as a concept-based search that goes beyond a Boolean keyword-based query but allows to group vocabulary items and other concepts into coherent sets of meaning and to retrieve documents and sections that best represent a mixture of all the aspects included in the searched-for concept. But all in all, only few relatively basic methods have been confronted with the new graph-paradigm yet.

The model itself could also turn out to be helpful in more complex scenarios. To give just one example: In 2002 Montes-y Gómez et al. obtained “Conceptual Graphs” after an in-depth (local!) language-specific processing of text [4]. They aimed to generalize them as a way of indexing and clustering documents. These analysis artifacts could easily be modelled on top of the token chain and would in this configuration give access to a more fine-grained structure model.

Fig. 3: Double logarithmic view of response times (in seconds) dependent on the number of neighboring words in a word co-occurrence search.
than just documents. In Kadmos all “infrastructural” essentials for a fresh take on this approach are already in place. It could be seen as an NLP specialist’s workbench that is suitable for quick re-implementation tests which could maybe revive ideas from a 13 years old publication.

Even if ultimately just a small portion of Text Mining tasks could fully benefit from a switch to a graph representation, it would nevertheless be worthwhile to thoroughly think about the necessity of global knowledge and queries to approach them. This thought process could lead to efficient heuristics that can perform nearly as good in real-world scenarios without leaving a more local scope - increasing the processing efficiency.

A Graph model of text may offer great flexibility but it necessarily shows a poor runtime behavior when a global scan or ranking is needed. For future use it is important to identify such cases and see, if external indices or secondary structures anchored directly in the database can circumvent the need to query the global scope.

And even if that turns out to be impossible, new technological advancements make it easy to drop the real-time access patterns and replace the Graph Database with a Graph Processor that performs in batch-mode and can scale out to the global scope (with the help of computer clusters). Ideally both technologies will grow together and provide a unified query interface and transparent data distribution from the database to the analytics cluster.

References


