tm Text Mining Environment

Ingo Feinerer

Technische Universität Wien, Austria

SNLP Seminar, 22.10.2010

Text Mining Package and Infrastructure

I. Feinerer
tm: Text Mining Package, 2010
URL http://CRAN.R-project.org/package=tm
R package version 0.5-4

I. Feinerer, K. Hornik, and D. Meyer
Text mining infrastructure in R
Journal of Statistical Software, 25(5):1–54, March 2008
ISSN 1548-7660
URL http://www.jstatsoft.org/v25/i05

What is a Corpus?

Definition

A *corpus* represents a collection of documents and is often augmented with meta data annotations.

Collections can be implemented by arbitrary data structures, like lists or multi-sets.

Documents can be in any file format, like plain text or XML.

Meta Data can annotate the corpus, the documents, or higher level entities (like classifications).

Corpus Representation in R

The **tm** Package

The **tm** extension package for R provides a sophisticated text mining infrastructure.

Corpora	are lists containing documents.
Documents	are abstract containers with instantiations for each file format.
Meta Data	is stored in attached data frames and lists.

Meta Data Representation

Conceptually we have corpus and document meta data.

Meta Data in **tm**

Corpus	meta data is stored in a list holding information only relevant for the whole corpus.
Document	meta data is stored locally at each document.
Mixed	meta data either forms an own entity (like classifications) or is aggregated for performance reasons. Stored as data frame.

Conceptual Layers and Packages

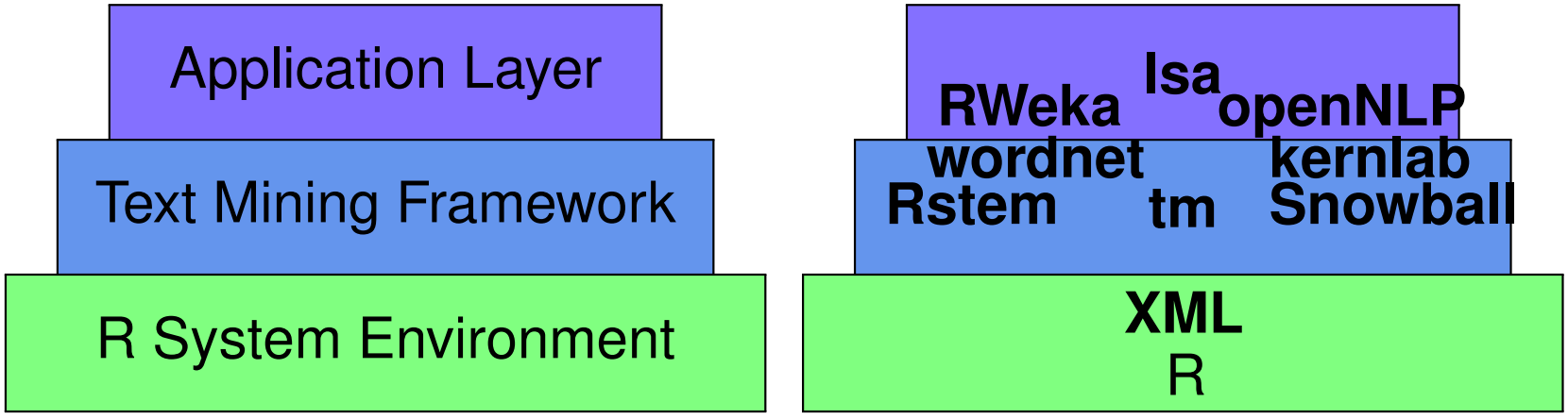

Figure: Conceptual Layers and Packages.

UML Class Diagram

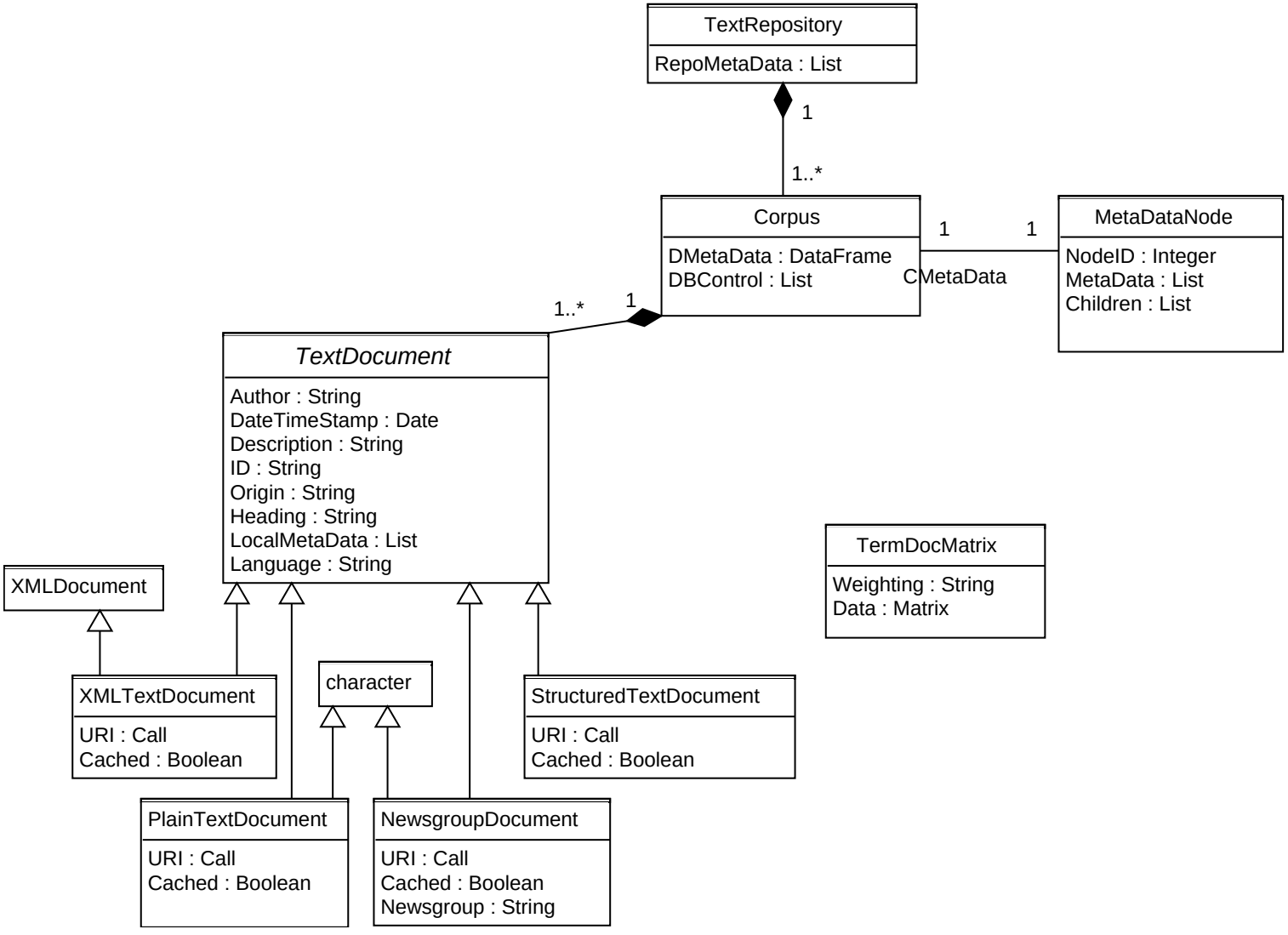

Figure: UML class diagram of the **tm** package.

Corpus Construction

1. Fetch documents from sources (disk, Internet)
2. Parse document structure (HTML, PDF, `getReaders()`)
3. Extract text and meta information
4. Dynamically create corpus
5. Fill corpus
   - immediately
   - delayed (load on demand)
   - referentially (using pointers to a database)

Sources

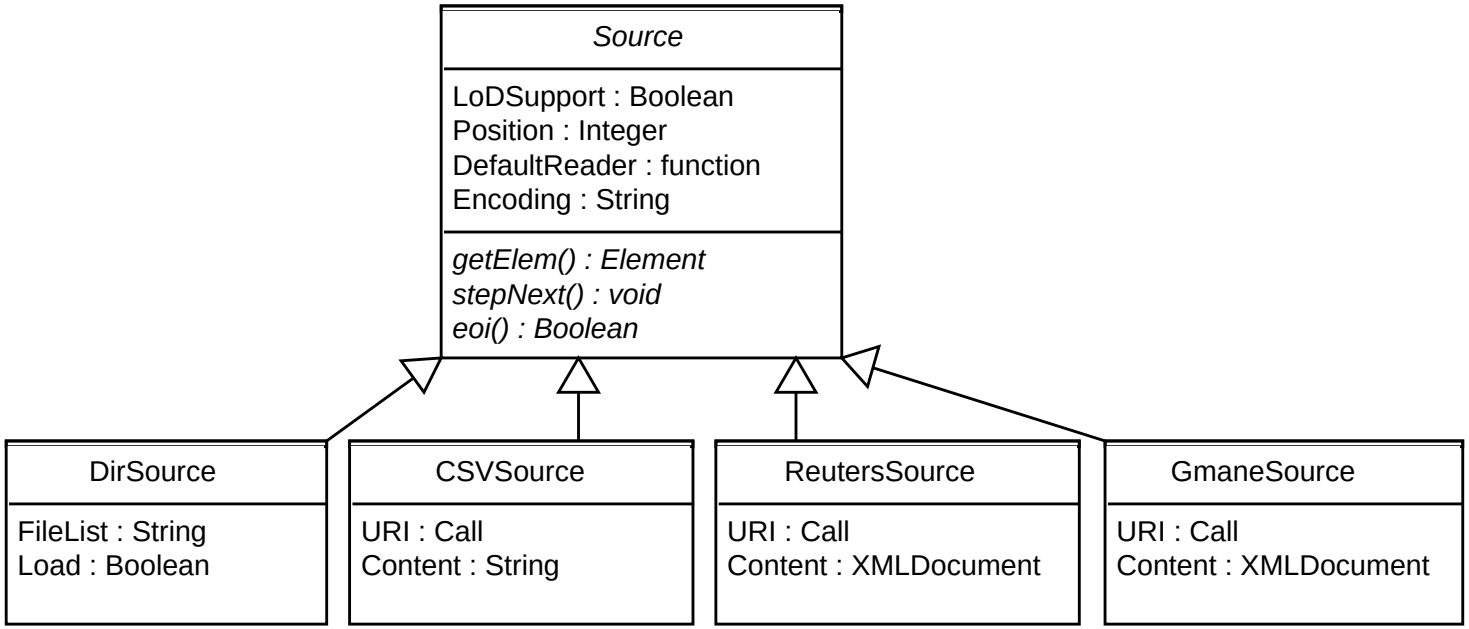

Figure: UML class diagram for Sources in the **tm** package.

Algorithms

- Document and Corpora Handling:
  - Constructors
  - Merging
  - Accessors and Extractors
- Transformations: Define mappings for corpora
  - Most preprocessing functions are transformations
  - Capture the concept of maps from functional programming
- Filters: Define predicate functions to extract documents from corpora
  - Full text search
  - Filters have full access to meta data

Predefined Functionality

- Preprocessing: data import, stemming, stopword removal, part of speech tagging, synonyms, ...
- Basis analysis techniques: count based evaluation, text clustering, text classification, ...
- Access to more advanced functionality: full integration with string kernels, latent semantic analysis, ...
- Export of term-document matrices: basically all methods in R working on matrices

Preprocessing

Definition

Preprocessing manipulates input data to generate output for later analysis steps.

- Conversion to plain text or lower case
- Stemming
- Part-of-speech tagging
- Punctuation, stopword, or whitespace removal

Transformations

Definition

Under a *transformation* we understand a (non-bijective) mapping between two states of the same corpus.

Example

Stemming maps words with suffixes to their stems.

- Very time consuming for large corpora
- Lazy mapping
- Parallel execution possible

Filters

Definition

A *filter* applies a predicate function on a corpus to extract patterns of interest.

Example

Full text search filters out documents matching specified terms.

Extensions

- Modular structure designed for easy extensibility
- Readers, sources, etc. define interfaces
- Implementations for these interfaces generate first-class objects (internal objects use the same mechanism)
- Easy plug-ins, we provide the infrastructure, the (advanced) user his custom functionality

Handling Big Corpora

- Some real world examples use multiple 100000 documents
- **tm** and R start getting problems beginning with 50000 documents, depending on RAM
- Consider a term-document matrix consisting of 100000 documents with 20000 unique terms:

$$\frac{100000 \cdot 20000 \cdot 32}{1024^3} \approx 60 \text{ GByte}$$

- Can be reasonably handled with sparse (**slam**) matrices (supported in **tm**)
- We consider the construction of a sparse matrix in **tm** as quite optimized now
- Problems arise when computing on such a matrix (e.g. correlation)