# Textual Data

## What is Textual Data?

**Textual Data** refers to *sytematically collected material consisting of written, printed, or electronically published words, typically either purposefully written or transcribed from speech.*[2]

**Document.** A **text document** is a single unit of **textual data**: one element of the systematically collected material acessible in a holistic way (either as a single file, or as a single logical web page).

**Document collection.** A set of **text documents** that was *systematically collected*.

**Note:** What constitutes a document may vary depending on the document collection and its purpose. A book represents a single document in the collection of books (e.g., The Guttenberg Project). At the same time, a single book may often be considered a document collection itself, in which case the documents with in it may be its chapters, or even, individual paragraphs.

### Formats for text documents

Text documents are usually stored electronically in the following data formats:

- **Plain text files.** The simplest of all text formats, usually labeled by the `.txt` file extenstions, these files contain verbatim the text of the document *and nothing else*.
- **Rich Text Format.** Rich text format file, usually carrying the `.rtf` extension is an extension of plain text format with a number of special purpose annotations and commands that a text processor can parse and render. RTF files can add different fonts and other simple editorial features to the text documents. RTF is a proprietary Microsoft format, but it has been used widely as a document interchange format.
- **HTML.** Most text documents available on-line are stored inside HTML containers, just as about anything else on the World Wide Web. HTML adds a variety of tags that specify formatting of the text, as well as may apply Cascaded Style Sheets and some other tools to properly render the text. At the same time, the actual text content of the document stored in HTML is usually readable.

  One other important thing that HTML is able to add to text documents is **hyperlinking** - one text in a collection of documents can point directly to another text.
- **SGML and XML.** Standard General Markup Language and Extensible Markup Language emerged in late 1990s as data formats for annotating text documents. SGML/XML text documents often look similar to HTML documents, but unlike HTML, where the names of the annotation tags are

fixed, SGML and XML allow the author of the documents to use their own tags/elements to annotate the text.

- **E-book formats.** Since late 1990s, *electronic books* became popular. Over the years a large number of proprietary e-book formats was created: Fb2 and Mobipocket (one of Amazon's Kindle formats), KF8 (another Kindle format) are the most popular of them. An open e-book format called ePub also exists[1]. Fb2 and ePub are based on XML, while Mobipocket is not directly readable, KF8 is somewhat compatible with HTML5 – the latest standard for HTML.
- **Word processing files.** Microsoft's .doc and .docx data formats, despite being widely different dominate this space. There are also open formats coming from the Open Office community. .doc format is not human readable. .docx encodes documents in XML.
- Postscript and PDF files. While different, these two formats share common features, as they are largely created so that complex documents are properly rendered on screen and in print. Postscript is a programming language for processing and rendering text.
- Documents as parts of other objects. Textual data may often be embedded inside other objects as one of the attribute among many. In this case, textual data can come in any known format and will require the appropriate tools for extraction.

In this course we will primarily deal with data formats that are easy to read and easy to process - plain text, HTML, with some XML and JSON (see **Document as parts of other objects** above).

## Processing Text Document Collections

In data science, text collections can be used for a wide range of purposes, and the text documents from those collections can be analyzed in a diverse number of ways. Understanding what text documents will be used for will guide the initial processing of text documents.

**Note.** Strictly speaking, text processing steps described below correspond to three steps in the data science process:

- Data acquisition. On this step the text collection is formed and the format for the original text documents is selected.
- Data cleaning. On this step some preliminary processing of the text documents is performed. In case of text document collections, this step is often called preprocessing and may consist of a variety of activities (see below).
- Data modeling. In order to be conveniently analyzed, text data needs to be "unpacked" from its original format and representations of text documents that are easy to use in analysis need to be built. This step is often called text indexing, the data in the text documents is getting *indexed* for easier access.

[1] See https://en.wikipedia.org/wiki/Comparison_of_e-book_formats for a detailed list of e-book formats.

**Processing text documents.** The following processing steps are usually considered.

1. **Preprocessing.** Preprocessing involves a number of, typically, model-independent procedures that turn input documents from text into representations that **model-based indexer** can use to construct the **document collection index**. Typical preprocessing steps include:
   (a) **Parsing and tokenizing** input. On this stage individual keywords/terms, sentences, phrases and other constructs are identified. Also, parts of the input, not needed in the indexing process (e.g., HTML tags) are filtered out.
   (b) **Stopword removal.** Some words are ubiquitous, and their value carriers of meaningful data is negligible. These words are identified and removed from the document representations.
   (c) **Stemming.** Each keyword is replaced with its **stem** - a substring that represents the *unchangable portion of the keyword*. This allows the IR system to treat words like "computer", "computers", "computing", "computation" as the same keyword. [2]
2. **Modeling and Indexing.** The **modeling** part of this process produces a *representation* for each text document that is needed for the analytical methods.

   The **indexing** part creates a searchable *secondary data structure* that allows analytical methods discover documents posessing specific features *without having to access all document representations*.

## Preprocessing

**Preprocessing** involves a number of **model-independent** activities: i.e., tasks that need to be performed in order to represent documents from $D$ as **bags of words**.

These tasks include **parsing**, **stopword removal** and **stemming**.

### Parsing

**Parsing** is the only preprocessing activity that is dependent on the input format.

The **parsing** process reads one-by-one each document $d_j$ from the document collection $D$. The following tasks are typically performed:

- The input document is **tokenized**. That is, the **parser** detects word boundaries, punctuation and other features of the document.
- The input document is **filtered** if necessary. Some documents are in a form of `HTML` or `XML` files. Other documents may contain formatting instructions, tables, images, and other features that either require **deep parsing** or need to be/ can be ignored by the rest of the preprocessing and indexing components.

[2] This is not always desirable. Google, for example does not stem keywords. However, in smaller document collections stemming can significantly improve the quality of retrieval.

### Stopword Removal

**Stopwords.** A **stopword** is a word that is found in colloquial speech/literature/specific document collection *so often, that it does not carry any specific meaning.*

Various lists of stopwords exist. Typical stopwords are pronouns ("I", "you", "they", "them", "his"), common verbs ("do", "be", "am", "are", "have" "had"), propositions and connectives: ("in", "onto", "and", "or", "of", "from").

Some document collection acquire stopwords that are specific to them. E.g., a collection of Computer Science articles may need to declare "computer" to be a stopword.

**Stopword removal.** For many analytical tasks stopwords are not needed. At the same time, documents may contain a significant percentage of stopwords, which may slow down the analysis. In such cases it may be important to get rid of stopwords. The traditional mechanism is as follows:

- Select a list of stopwords appropriate for the document collection and the analytical tasks considered for the document collection.
- For each token of type word read from the document, check if the word belongs to the list of known stopwords.
- If it belongs to the list of stopwords, remove it from the stream, consider next word.
- If is **does not belong** to the list of stopwords, pass the token onto the next step of processing.

#### Stemming

**Stemming** is the process of replacing a word with its **stem** or **root**[3].

**Stemming procedures** are unique for each language. In English, **Porter Algorithm**[1] is considered to be the traditional **stemming technique**. The specifics of the algorithm can be found in [1], as well as at

http://tartarus.org/∼martin/PorterStemmer/

The algorithm itself is described at

http://tartarus.org/∼martin/PorterStemmer/def.txt

The algorithm is organized as a series of rule applications. Given a word, on each step of the series the algorithm checks whether a rule is applicable. A rule identifies the ending of the current word, and replaces it (occasionally, simply removes it) with a shorter ending.

## Feature Extraction from Text Documents

The modeling process for text document collections represents text documents as data structures based on document features. The following types of features can be found in text documents.

[3] Technically, a stem of a word is more than just a root. Stemming algorithms keep prefixes intact.

**Words and keywords.** The most basic unit of information extractable from a text is a single word. Often, only some of the words in the document are important enough to keep in the document model. Such words are called keywords (while all other words are called stopwords).

Some keywords are longer than a single word. For example `"Cal Poly"` is a keyword consisting of two words.

**Sentences.** Often, the units of analysis of text documents are individual sentences. Sentences may be extracted in raw format, as well as in a parsed form, together with the *annotated parse tree* showing the grammatical structure of the sentence.

**Bigrams, trigrams, multigrams.** A bigram is defined as a succession of two simple features. In text analysis, the simple features may be either individual letters or individual words in the document.

For example, a word `"science"` consists of the following *letter bigrams*:

```
sc, ci, ie, en, nc, ce
```

A sentence `"A bigram is defined as a succession of two simple features"` consists of the following *word bigrams*:

```
a bigram, bigram is, is defined, defined as, as a, a succession
succession of, of two, two simple, simple features
```

If stopwords are removes, the same sentence may be represented as the following sequence of tokens: `"bigram defined succession two simple features"`, in which case, the list of bigrams for the sentence will be

```
bigram defined, defined succession, succession two, two simple,
simple features
```

Trigrams are defined as successions of three simple features, and so on. The list of letter trigrams in the word `"science"` is

```
sci, cie, ien, enc, nce
```

The list of word trigrams in our sample sentence above is

```
a bigram is, bigram is defined, is defined as, defined as a
as a succession, a succession of, succession of two, of two simple,
two simple features
```

**Entities and notions.** Words, bigrams, etc are *syntactic* units of information contained in the text documents, i.e., they simply represent parts structural parts of the document. Entities and *notions* are usually the units of *semantical* information contained in a text document.

For example, consider the sentence

`Mitchell and Stephanie had a great time in San Francisco in June.`

A reader can recognize the following information in this sentence:

- People. The words `"Mitchell"` and `"Stephanie"` refer to two different people mentioned in the sentence.
- Geography. The pair of words `"San Francisco"` refers to a city (most likely) in California.
- Calendar. The word `"June"` refers to a calendar month in this particular setting [4]

Thus, it is possible to extract four features:

```
Person("Mitchell")
Person("Stephanie")
City("San Francisco")   [or GeographicName("San Francisco")]
Month("June")
```

from this sentence. Unlike simply noticing that words `"Mitchell", "Stephanie", "San", "Francisco"` and `"June"` should be stored in a representation of this sentence, determining the types of entities or notions they correspond to allows us to stockpile for future the information about the actual *meaning* of the sentence.

**Relationships and associations.** The final (for now) and the most complex type of information that can be extracted from text is associations (otherwise often called relationships) between various entities and notions observed in text.

For example, from analyzing the sentence above, one can conclude that Mitchell visited San Francisco. One can also conclude that Stephanie visited San Francisco. A number of associations between Mitchell and Stephanie are also possible: e.g., we may conclude that Mitchell and Stephanie may be friends.

## Modeling Document Collections

Most document collection models share the following features.

**Document collection.** The input data for each model is a document collection $D = \{d_1, \ldots d_n\}$ of documents.

**Vocabulary (corpus).** The collection of **non-stop word** words (a.k.a. terms) found in the documents from $D$ is called the **vocabulary** or **corpus** of $D$. Given $D$, we denote the vocabulary of $D$ as

$$V_D = \{t_1, \ldots t_M\}.$$

(where $D$ is unique, we denote the vocabulary of $D$ as simply $V$.) Each $t_i$ is a **distinct term** (keyword) found in at least one document in $D$.

[4] Notice that `"June"` is also a possible name.

**Vector of features representation.** Each document $d_j inD$ is represented as a vector $d_j = (f_1, \ldots, f_n)$ of features selected by the model. Each $f_i$ in $d_j$ represents the *value* of *feature* $i$ of the model.

### Bag of Word Models

**Bag of words representation.** Each document $d_j \in D$ is represented as a **bag of words**, i.e., as an **unordered** collection of terms found in each document (**bag** means that the number of occurrences of each term may be taken into account).

The standard representation of **bag of words** is a **vector of keyword weights**: a vector which assigns each term $t_i \in V$ a **weight** based on its occurrence/non-occurrence in $d_j$.

As such, we view $d_j$ as the vector

$$d_j = (w_{1j}, w_{2j}, w_{3j}, \ldots, w_{Mj}).$$

Here $w_i j$ is the weight of term $t_i$ in document $d_j$.

Different **models** represent **term weights** in different ways.

#### Boolean Model

**Boolean Information Rertieval model** is the simplest document representation model. It has the following features:

**Term weights.** Given a document $d_j \in D$, it is represented as the vector of **binary keyword weights** $d_j = (w_{1j}, w_{2j}, w_{3j}, \ldots, w_{Mj})$, where $w_{ij} \in \{0, 1\}$ and

$$w_{ij} = \begin{cases} 1 & : \ t_i \text{ appears in } d_j; \\ 0 & : \ \text{otherwise}. \end{cases}$$

## Vector Space Model

**Overview.** **Vector Space Model** represents keyword weights on the scale from 0 to 1 and represents queries in a way, similar to documents. It uses **cosine similarity** to compute the relevance between a document and a query and uses the relevance value to rank the results.

**Term frequency.** Given a document $d_j \in D$ and a term $t_i \in V$, the **term frequency (TF)** $f_{ij}$ of $t_i$ in $d_j$ is the number of times $t_i$ occurs in $d_j$. For a document $d_j$, we can construct its vector of term frequencies

$$f_{d_j} = (f_{1j}, f_{2j}, \ldots, f_{Mj}).$$

**Normalized term frequency.** Term frequencies are commonly manipulated to provide for a better representation of the document. Two manipulation techniques used are **thresholding** and **normalization**.

Given a **threshold value** $\alpha$, we set **term frequency** $f'_{ij}$ to be

$$f'_{ij} = \begin{cases} f_{ij} & : \quad f_{ij} < \alpha; \\ \alpha & : \quad f_{ij} \geq \alpha \end{cases}$$

(i.e., we discount any further occurrences of the terms in document beyond a certain **threshold** $\alpha$ number of occurrences).

Given a vector $f_{d_j}$ of (possibly thresholded) term frequencies, we compute **normalized term frequencies** $tf_{ij}$ as follows:

$$tf_{ij} = \frac{f_{ij}}{\max(f_{1j}, f_{2j}, \ldots, f_{Mj})}.$$

**Document frequency (DF).** Given a term $t_i \in V$, its **document frequency**, $df_i$ is defined as the **number of documents in which $t_i$ occurs**:

$$df_i = |\{d_j \in D | f_{ij} > 0\}|.$$

**Inverse document frequency (IDF).** Given a term $t_i \in V$, its **inverse document frequency (IDF)** is computed as

$$idf_i = \log \frac{n}{df_i}.$$

**TF-IDF keyword weighting schema.** Given a document $d_j$ and a term $t_i$,

$$w_{ij} = tf_{ij} \cdot idf_i = \frac{f_{ij}}{\max(f_{1j}, \ldots, f_{Mj})} \cdot \log_2 \frac{n}{df_i}.$$

**Intuition.** The idea behind the **tf-idf weighting schema** is straightforward. The importance of a keyword to a document is measured using two rules:

1. The more often the keyword appears in a document, the more important it is.
2. The less frequently the keyword appears in the document collection, the more important its occurence is in each document.

**Term frequency** (or normalized term frequency) captures the first rule. **Inverse document frequency** captures the second rule.

**Query weighting schemas.** Given a query $q = (w_{1q}, \ldots, w_{Mq})$, the query term weights can be determined using the exact **TF-IDF weighting schema**:

$$w_{iq} = tf_{iq} \cdot idf_i.$$

However, if $q$ is short (contains relatively few terms in comparison with documents from $D$), some slight adjustments can be adopted:

$$w_{iq} = (0.5 + 0.5 \cdot tf_{iq}) \cdot idf_i.$$

**Other simple weighting schemas.** Some other simple term weighting schemas:

Normalized Term Frequency (TF): $w_{ij} = tf_{ij}$.

Inverse Document Frequency (IDF): $w_{ij} = idf_i$.

(note, when $d_j$ has no repeated keywords, TF-IDF weighting schema converges to simple IDF weighting. This is common in document collections that consist of small documents: e.g., lists of quotations.)

## References


[1] M.F. Porter, 1980, An algorithm for suffix stripping, Program, 14(3) pp 130–137.

[2] Kenneth Benoit, Data, Textual. In *International Encyclopedia of Political Science*, Bertrand Badie, Dirk Berg-Schlosser, Lenoardo Morlino (Eds.), DOI: http://dx.doi.org/10.4135/9781412959636.n127