

## Chapter 2

# Text Analysis Pipelines

*I put my heart and my soul into my work, and have lost my mind in the process.*

– Vincent van Gogh

**Abstract** The understanding of natural language is one of the primary abilities that provide the basis for human intelligence. Since the invention of computers, people have thought about how to operationalize this ability in software applications (Jurafsky and Martin 2009). The rise of the internet in the 1990s then made explicit the practical need for automatically processing natural language in order to access relevant information. Search engines, as a solution, have revolutionized the way we can find such information ad-hoc in large amounts of text (Manning et al. 2008). Until today, however, search engines excel in finding relevant texts rather than in understanding what information is relevant in the texts. Chapter 1 has proposed text mining as a means to achieve progress towards the latter, thereby making information search more intelligent. At the heart of every text mining application lies the analysis of text, mostly realized in the form of text analysis pipelines. In this chapter, we present the basics required to follow the approaches of this book to improve such pipelines for enabling text mining ad-hoc on large amounts of text as well as the state of the art in this respect.

Text mining combines techniques from information retrieval, natural language processing, and data mining. In Sect. 2.1, we first provide a focused overview of those techniques referred to in this book. Then, we define the text analysis processes and pipelines that we consider in our proposed approaches (Sect. 2.2). We evaluate the different approaches based on texts and pipelines from a number of case studies introduced in Sect. 2.3. Finally, Sect. 2.4 surveys and discusses related existing work in the broad context of ad-hoc large-scale text mining.

## 2.1 Foundations of Text Mining

In this section, we explain all general foundations of text mining the book at hand builds upon. After a brief outline of text mining, we organize the foundations along the three main research fields related to text mining. The goal is not to provide a formal and comprehensive introduction to these fields, but rather to give exactly the information that is necessary to follow our discussion. At the end, we describe how to develop and evaluate approaches to text analysis. Basic concepts of text analysis processes are defined in Sect. 2.2, while specific concepts related to our overall approach are directly defined where needed in Chaps. 3–5.<sup>1</sup>

### 2.1.1 Text Mining

Text mining deals with the automatic or semi-automatic discovery of new, previously unknown information of high quality from large numbers of unstructured texts (Hearst 1999). Different than sometimes assumed, the types of information to be inferred from the texts are usually specified manually beforehand, i.e., text mining tackles given tasks. As introduced in Sect. 1.1, this commonly requires to perform three steps in sequence, each of which can be associated to one field (Ananiadou and McNaught 2005):

1. **Information retrieval.** Gather input texts that are potentially relevant for the given task.
2. **Natural language processing.** Analyze the input texts in order identify and structure relevant information.<sup>2</sup>
3. **Data mining.** Discover patterns in the structured information that has been inferred from the texts.

Hearst (1999) points out that the main aspects of text mining are actually the same as those studied in empirical computational linguistics. Although focusing on natural language processing, some of the problems computational linguistics is concerned with are also addressed in information retrieval and data mining, such as text classification or machine learning. In this book, we refer to all these aspects with the general term text analysis (cf. Sect. 1.1). In the following, we look at the concepts of the three fields that are important for our discussion of text analysis.

---

<sup>1</sup>Notice that, throughout this book, we assume that the reader has a more or less graduate-level background in computer science or similar.

<sup>2</sup>Ananiadou and McNaught (2005) refer to the second step as information extraction. While we agree that information extraction is often the important part of this step, also other techniques from natural language processing play a role, as discussed later in this section.

### 2.1.2 Information Retrieval

Following Manning et al. (2008), the primary use case of information retrieval is to search and obtain those texts from a large collection of unstructured texts that can satisfy an information need, usually given in the form of a query. In ad-hoc web search, such a query consists of a few keywords, but, in general, it may also be given by a whole text, a logical expression, etc. An information retrieval application assesses the relevance of all texts with respect to a query based on some similarity measure. Afterwards, it ranks the texts by decreasing relevance or it filters only those texts that are classified as potentially relevant (Manning et al. 2008).

Although the improvement of ad-hoc search denotes one of the main motivations behind this book (cf. Chap. 1), we hardly consider the retrieval step of text mining, since we focus on the inference of information from the potentially relevant texts, as detailed in Sect. 2.2. Still, we borrow some techniques from information retrieval, such as filtering or similarity measures. For this purpose, we require the following concepts, which are associated to information retrieval rather than to text analysis.

**Vectors.** To determine the relevance of texts, many approaches map all texts and queries into a *vector space model* (Manning et al. 2008). Such a model defines a common vector representation  $\mathbf{x} = (x_1, \dots, x_k)$ ,  $k \geq 1$ , for all inputs where each  $x_i \in \mathbf{x}$  formalizes an input property. A concrete input like a text  $D$  is then represented by one value  $x_i^{(D)}$  for each  $x_i$ . In web search, the standard way to represent texts and queries is by the frequencies of words they contain from a set of (possibly hundreds of thousands) words. Generally, any measurable property of an input can be formalized, though, which becomes particularly relevant for tasks like text classification.

**Similarity.** Given a common representation, similarities between texts and queries can be computed. Most word frequencies of a search query will often be 0. In case they are of interest, a reasonable similarity measure is the cosine distance, which puts emphasis on the properties of texts that actually occur (Manning et al. 2008). In Chap. 5, we compute similarities of whole texts, where a zero does not always mean the absence of a property. Such scenarios suggest other measures. In our experiments, we use the *Manhattan distance* between two vectors  $\mathbf{x}^{(1)}$  and  $\mathbf{x}^{(2)}$  of length  $k$  (Cha 2007), which is defined as:

$$\text{Manhattan distance}(\mathbf{x}^{(1)}, \mathbf{x}^{(2)}) = \sum_{i=1}^k |\mathbf{x}_i^{(1)} - \mathbf{x}_i^{(2)}|$$

**Indexing.** While queries are typically stated ad-hoc, the key to efficient ad-hoc search is that all texts in a given collection have been indexed before. A query is then matched against the *search index*, thereby avoiding to process the actual texts during search. Very sophisticated indexing approaches exist and are used in today's web search engines (Manning et al. 2008). In its basic form, a search index contains one entry for every measured property. Each entry points to all texts that are relevant with respect to the property. Some researchers have adapted indexing to information

extraction by building specialized search indexes based on concepts like entities, such as Cafarella et al. (2005). We discuss in Sect. 2.4 in how far they reduce the need for ad-hoc large-scale text mining that we tackle in this book.

**Filtering.** While the ranking of texts by relevance is not needed in this book, we filter relevant portions of texts in Chap. 3. Filtering is addressed in information retrieval on two levels: *Text filtering* classifies complete texts as being relevant or irrelevant (Sebastiani 2002), whereas *passage retrieval* aims to determine the passages of a text that are relevant for answering a given query (Cui et al. 2005). We investigate the difference between our and existing filtering approaches in Sect. 2.4 and their integration at the end of Chap. 3. Filtering is usually seen as a classification task (Manning et al. 2008) and, thus, addressed as a text analysis. We cover text classification as part of natural language processing, which we describe next.

### 2.1.3 Natural Language Processing

Natural language processing covers algorithms and engineering issues for the understanding and generation of speech and human-readable text (Tsujii 2011). In the book at hand, we concentrate on the analysis of text with the goal of deriving structured information from unstructured texts.

In text analysis, algorithms are employed that, among others, infer lexical information about the words in a text, syntactic information about the structure between words, and semantic information about the meaning of words (Manning and Schütze 1999). Also, they may analyze the discourse and pragmatic level of a text (Jurafsky and Martin 2009). In Chaps. 3–5, we use lexical and syntactic analyses as preprocessing for information extraction and text classification. Information extraction targets at semantic information. Text classification may seek for both semantic and pragmatic information. To infer information of certain types from an input text, text analysis algorithms apply rules or statistics, as we detail below.

Generally, natural language processing faces the problem of *ambiguity*, i.e., many utterances of natural language allow for different interpretations. As a consequence, all text analysis algorithms need to resolve ambiguities (Jurafsky and Martin 2009). Without sufficient context, a correct analysis is hence often hard and can even be impossible. For instance, the sentence “*SHE’S AN APPLE FAN.*” alone leaves undecidable whether it refers to a fruit or to a company.

Technically, natural language processing can be seen as the production of *annotations* (Ferrucci and Lally 2004). An annotation marks a text or a span of text that represents an instance of a particular type of information. We discuss the role of annotations more extensively in Sect. 2.2, before we formalize the view of text analysis as an annotation task in Chap. 3.

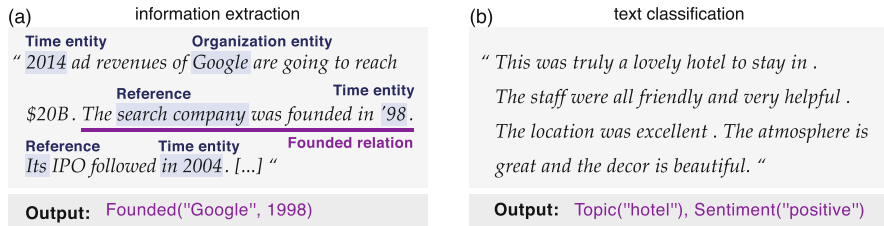
**Lexical and Syntactic Analyses.** For our purposes, we distinguish three types of lexical and syntactical analyses: The *segmentation* of a text into single units, the *tagging* of units, and the *parsing* of syntactic structure.

Mostly, the smallest text unit considered in natural language processing is a *token*, denoting a word, a number, a symbol, or anything similar (Manning and Schütze 1999). Besides the *tokenization* of texts, we also refer to *sentence splitting* and *paragraph splitting* as segmentation in this book. In terms of tagging, we look at *part-of-speech*, meaning the categories of tokens like nouns or verbs, although much more specific *part-of-speech tags* are used in practice (Jurafsky and Martin 2009). Also, we perform *lemmatization* in some experiments to get the *lemmas* of tokens, i.e., their dictionary forms, such as “be” in case of “is” Manning and Schütze (1999). Finally, we use shallow parsing, called *chunking* (Jurafsky and Martin 2009), to identify different types of phrases, and *dependency parsing* to infer the dependency tree structure of sentences (Bohnet 2010). Appendix A provides details on all named analyses and on the respective algorithms we rely on. The output of parsing is particularly important for information extraction.

**Information Extraction.** The basic semantic concept is a named or numeric *entity* from the real world (Jurafsky and Martin 2009). Information extraction analyzes usually unstructured texts in order to recognize references of such entities, *relations* between entities, and *events* the entities participate in Sarawagi (2008). In the classical view of the MESSAGE UNDERSTANDING CONFERENCES, information extraction is seen as a template filling task (Chinchor et al. 1993), where the goal is to fill entity slots of relation or event templates with information from a collection or a stream of texts **D**.

The set of information types **C** to be recognized is often predefined, although some recent approaches address this limitation (cf. Sect. 2.4). Both rule-based approaches, e.g. based on regular expressions or lexicons, and statistical approaches, mostly based on machine learning (see below), are applied in information extraction (Sarawagi 2008). The output is structured information that can be stored in databases or directly displayed to the users (Cunningham 2006). As a matter of fact, information extraction plays an important role in today’s database research (Chiticariu et al. 2010a), while it has its origin in computational linguistics (Sarawagi 2008). In principle, the output qualifies for being exploited in text mining applications, e.g. to provide relevant information like GOOGLE in the example from Fig. 1.2 (Sect. 1.1). However, many information types tend to be domain-specific and application-specific (Cunningham 2006), making their extraction cost-intensive. Moreover, while some types can be extracted accurately, at least from high-quality texts of common languages (Ratinov and Roth 2009), others still denote open challenges in current research (Ng 2010).

Information extraction often involves several subtasks, including *coreference resolution*, i.e., the identification of references that refer to the same entity (Cunningham 2006), and the *normalization* of entities and the like. In this book, we focus mainly on the most central subtasks, namely, named and numeric *entity recognition*, binary *relation extraction*, and *event detection* (Jurafsky and Martin 2009). Concrete analyses and algorithms that realize the analyses are found in Appendix A. As an example, Fig. 2.1(a) illustrates instances of different information types in a sample text. Some refer to a relation of the type *Founded(Organization, Time)*.



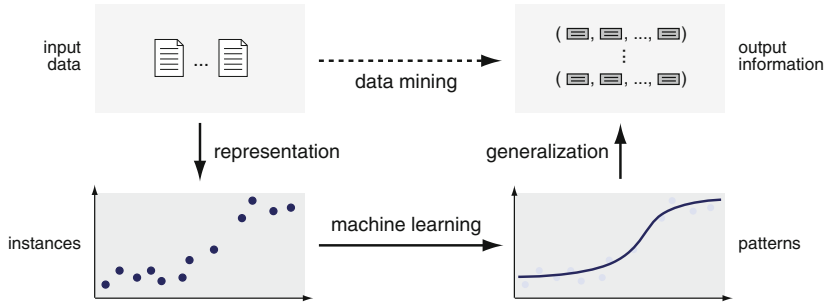
**Fig. 2.1** **a** Illustration of an information extraction example: Extraction of a relation of the type *Founded*(*Organization*, *Time*) from a sample text. **b** Illustration of a text classification example: Classification of the topic and the sentiment polarity of a sample text.

In Sect. 2.3, we introduce the more sophisticated extraction of financial events from news articles that we consider in many experiments in this book. Without exception, we extract information only *within* but not *across* texts unlike e.g. Li et al. (2011). Also, we restrict our view to unstructured texts and, hence, omit to present approaches that target at structured or semi-structured texts like wrapper induction (Kushmerick 1997).

**Text Classification.** Text classification (or text categorization) denotes the task of assigning each text from a collection or a stream of texts  $\mathbf{D}$  to one of a set of predefined classes  $C = \{c_1, \dots, c_k\}$ ,  $k \geq 2$  (Jurafsky and Martin 2009). We call  $C$  the *classification scheme* here. The standard approach to text classification is to statistically learn an assignment based on a set of training texts with known classes (Manning et al. 2008). To this end, every text is converted into a vector representation consisting of a number of (typically lexical or shallow syntactic) features of the text (Sebastiani 2002). We introduce this representation as part of the data mining foundations below.

Several text classification tasks are studied in natural language processing (Manning and Schütze 1999) and information retrieval (Manning et al. 2008). The classic *topic detection* (Lewis et al. 2004) targets at the main topic of a text, whereas in *non-standard text classification tasks*, classes go beyond the subjects of texts (Lipka 2013). Examples are the identification of the *genre* of a text in terms of the form, purpose, and/or intended audience of the text (Stein et al. 2010), or *authorship attribution* where the author of a text is to be determined (Stamatatos 2009). In *automatic essay grading*, the goal is to assign ratings from a usually numeric classification scheme to texts like essays (Dikli 2006), and *stance recognition* seeks for the stance of a person with respect to some topic (Somasundaran and Wiebe 2010). All these and some related tasks are discussed more or less detailed in this book.

Of highest importance in our experiments is *sentiment analysis*, which has become one of the most widely investigated text classification tasks in the last decade. By default, sentiment analysis refers to the classification of the *sentiment polarity* of a text as being positive or negative (Pang et al. 2002). An example is shown in Fig. 2.1(b). Sometimes, also an objective (or neutral) “polarity” is considered, although this class rather refers to *subjectivity* (Pang and Lee 2004). Moreover, sentiment can also be assessed on numeric scales (Pang and Lee 2005), which we call



**Fig. 2.2** Illustration of a high-level view of data mining. Input data is represented as a set of instances, from which a model is derived using machine learning. The model is then generalized to infer new output information.

*sentiment scoring* here. We employ a number of sentiment analysis algorithms in Sect. 5. They are listed in Appendix A.

As passage retrieval (see above), text classification does not always deal with complete texts. In Chap. 5, we classify the subjectivity of single discourse units, where objective units can be seen as *facts* and subjective units as *opinions*. In *opinion mining*, such techniques are combined with information extraction techniques to find opinions on certain topics (Popescu and Etzioni 2005), as done in one of our case studies (cf. Sect. 2.3). Sentiment analysis and opinion mining are of high practical relevance, because they can be used in text mining applications that analyze the people’s opinions on products and brands in social media, online review sites, and similar (Pang and Lee 2008).<sup>3</sup> For this purpose, data mining needs to be performed on the output of the respective algorithms.

### 2.1.4 Data Mining

Data mining primarily aims at the inference of new information of specified types from typically huge amounts of input data, already given in structured form (Witten and Frank 2005). To address such a *prediction problem*, the data is first converted into instances of a defined representation and then handed over to a *machine learning* algorithm. The algorithm recognizes statistical patterns in the instances that are relevant for the prediction problem. This process is called *training*. The found patterns are then generalized, such that they can be applied to infer new information from unseen data, generally referred to as *prediction*. In this regard, machine learning can be seen as the technical basis of data mining applications (Witten and Frank 2005). Figure 2.2 shows a high-level view of the outlined process.

<sup>3</sup>Unlike us, some researchers do not distinguish between sentiment analysis and opinion mining, but they use these two terms interchangeably (Pang and Lee 2008).

Data mining and text mining are related in two respects: (1) The structured output information of text analysis serves as the input to machine learning, e.g. to train a text classifier. (2) Many text analyses themselves rely on machine learning algorithms to produce output information. Both respects are important in this book. In the following, we summarize the basic concepts relevant for our purposes.<sup>4</sup>

**Machine Learning.** Machine learning describes the ability of an algorithm to learn without being explicitly programmed (Samuel 1959). An algorithm can be said to learn from data with respect to a given prediction problem and to some quality measure, if the measured prediction quality increases the more data is processed (Mitchell 1997).<sup>5</sup> Machine learning aims at prediction problems where the *target function*, which maps input data to output information, is unknown and which, thus, cannot be (fully) solved by following hand-crafted rules. In the end, all non-trivial text analysis tasks denote such prediction problems, even though many tasks have been successfully tackled with rule-based approaches.<sup>6</sup>

A machine learning algorithm produces a *model*  $\mathcal{Y} : \mathbf{x} \rightarrow C$ , which generalizes patterns found in the input data in order to approximate the target function.  $\mathcal{Y}$  defines a mapping from represented data  $\mathbf{x}$  to a *target variable*  $C$ , where  $C$  captures the type of information sought for. In text analysis, the target variable may represent classes of texts (e.g. topics or genres), types of annotations (e.g. part-of-speech tags or entity types), etc. Since machine learning generalizes from examples, the learned prediction of output information cannot be expected to be correct in all cases. Rather, the goal is to find a model  $\mathcal{Y}$  that is optimal with respect to a given quality measure (see below). Besides the input data, the quality of  $\mathcal{Y}$  depends on how the data is represented and how the found patterns are generalized.

**Representation.** Similar to information retrieval, most machine learning algorithms rely on a vector space model. In particular, the input data is represented by a set  $X$  of *feature vectors* of the form  $\mathbf{x}$ .  $\mathbf{x}$  defines an ordered set of *features*, where each feature  $x \in \mathbf{x}$  denotes a measurable property of an input (Hastie et al. 2009). In text mining, common features are e.g. the frequency of a particular word in a given text or the shape of a word (say, capitalized or not). Representing input data means to create a set of instances of  $\mathbf{x}$ , such that each instance contains one *feature value* for every feature in  $\mathbf{x}$ .<sup>7</sup> In many cases, hundreds or thousands of features are considered in combination. They belong to different *feature types*, like *bag-of-words* where each feature means the frequency of a word (Manning et al. 2008).

---

<sup>4</sup>Besides the references cited below, parts of the summary are inspired by the COURSERA machine learning course, <https://www.coursera.org/course/ml> (accessed on June 15, 2015).

<sup>5</sup>A discussion of common quality measures follows at the end of this section.

<sup>6</sup>The question for what text analysis tasks to prefer a rule-based approach over a machine learning approach lies outside the scope of this book.

<sup>7</sup>Throughout this book, we consider only features whose values come from a metric scale. Other features are transformed, e.g. a feature with values “red”, “green”, and “blue” can be represented by three 0/1-features, one for each value. All values are normalized to the same interval, namely [0,1], which benefits learning (Witten and Frank 2005).



The feature representation of the input data governs what patterns can be found during learning. As a consequence, the development of features, which predict a given target variable  $C$ , is one of the most important and often most difficult steps in machine learning.<sup>8</sup> Although common feature types like bag-of-words help in many text analysis tasks, the most discriminative features tend to require expert knowledge about the task and input. Also, some features generalize worse than others, often because they capture domain-specific properties, as we see in Chap. 5.

**Generalization.** As shown in Fig. 2.2, generalization refers to the inference of output information from unseen data based on patterns captured in a learned model (Witten and Frank 2005). As such, it is strongly connected to the used machine learning algorithm. The training of such an algorithm based on a given set of instances explores a large space of models, because most algorithms have a number of parameters. An important decision in this regard is how much to bias the algorithm with respect to the complexity of the model to be learned (Witten and Frank 2005). Simple models (say, linear functions) induce a high bias, which may not fit the input data well, but regularize noise in the data and, thus, tend to generalize well. Complex models (say, high polynomials) can be fitted well to the data, but tend to generalize less. We come back to this problem of *fitting* in Sect. 5.1.<sup>9</sup>

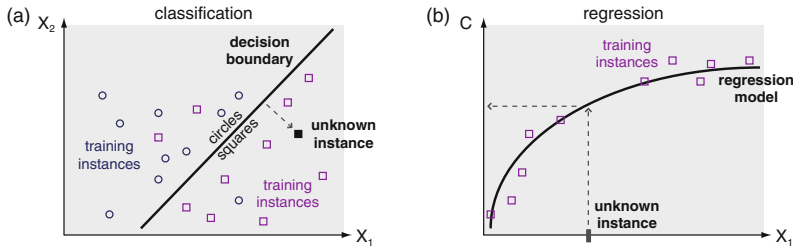
During training, a machine learning algorithm incrementally chooses a possible model and evaluates the model based on some cost function. The choice relies on an optimization procedure, e.g. *gradient descent* stepwise heads towards a local minimum of the cost function until convergence by adapting the model to all input data (Witten and Frank 2005). In large-scale scenarios, a variant called *stochastic gradient descent* is often more suitable. Stochastic gradient descent repeatedly iterates over all data instances in isolation, thereby being much faster while not guaranteeing to find a local minimum (Zhang 2004). No deep understanding of the generalization process is needed in this book, since we focus only on the question of how to address text analysis tasks with existing machine learning algorithms in order to then select an adequate one. What matters for us is the type of learning that can or should be performed within the task at hand. Mainly, we consider two very prominent types in this book, supervised learning and unsupervised learning.

**Supervised Learning.** In *supervised learning*, a machine learning algorithm derives a model from known *training data*, i.e., from pairs of a data instance and the associated correct output information (Witten and Frank 2005). The model can then be used to predict output information for unknown data. The notion of being supervised refers to the fact that the learning process is guided by examples of correct predictions. In this book, we use supervised learning for both statistical classification and statistical regression.

---

<sup>8</sup>The concrete features of a feature type can often be chosen automatically based on input data, as we do in our experiments, e.g. by taking only those words whose occurrence is above some threshold. Thereby, useless features that would introduce noise are excluded.

<sup>9</sup>Techniques like feature selection and dimensionality reduction, which aim to reduce the set of considered features to improve generalizability and training efficiency among others (Hastie et al. 2009), are beyond the scope of this book.



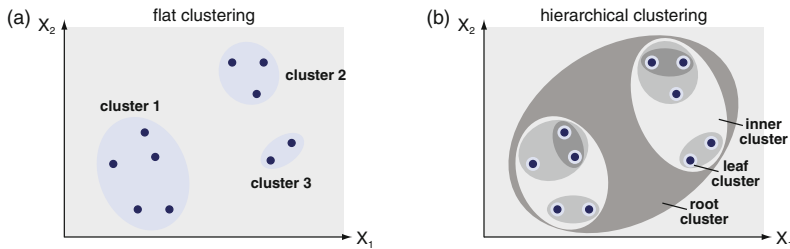
**Fig. 2.3** Illustration of supervised learning: **a** In classification, a decision boundary can be derived from training instances with known classes (open circles and squares) based on their feature values, here for  $x_1$  and  $x_2$ . The boundary decides the class of unknown instances. **b** In regression, a regression model can be derived from training instances (represented by the feature  $x_1$ ) with known value for the target variable  $C$ . The model decides the values of all other instances.

*Classification* describes the task to assign a data instance to the most likely of a set of two or more predefined discrete classes (Witten and Frank 2005). In case of binary classification, machine learning algorithms seek for an optimal *decision boundary* that separates the instances of two classes, as illustrated in Fig. 2.3(a). Multi-class classification is handled through approaches like one-versus-all classification (Hastie et al. 2009). The applications of classification in text mining are manifold. E.g., it denotes the standard approach to text classification (Sebastiani 2002) and it is also often used to classify candidate relations between entities (Sarawagi 2008). In all respective experiments below, we perform classification with a *support vector machine* (Witten and Frank 2005). Support vector machines aim to maximize the margin between the decision boundary and the training instances of each class. They have been shown to often perform well (Meyer et al. 2003) while not being prone to adapt to noise (Witten and Frank 2005).<sup>10</sup>

In case of *regression*, the task is to assign a given data instance to the most likely value of a metric and continuous target variable (Witten and Frank 2005). The result of learning is a *regression model* that can predict the target variable for arbitrary instances (cf. Fig. 2.3(b)). We restrict our view to *linear regression* models, which we apply in Chap. 4 to predict the run-times of pipelines. In our experiments, we learn these models with stochastic gradient descent for efficiency purposes.

**Unsupervised Learning.** In contrast to supervised learning, *unsupervised learning* is only given data instances without output information. As a consequence, it usually does not serve for predicting a target variable from an instance, but merely for identifying the organization and association of input data (Hastie et al. 2009). The most common technique in unsupervised learning is *clustering*, which groups a set of instances into a possibly but not necessarily predefined number of *clusters* (Witten and Frank 2005). Here, we consider only hard clusterings, where each instance belongs to a single cluster that represents some class. Different from

<sup>10</sup>Some existing text analysis algorithms that we employ rely on other classification algorithms, though, such as *decision trees* or *artificial neural networks* (Witten and Frank 2005).



**Fig. 2.4** Illustration of unsupervised learning: **a** Flat clustering groups a set of instances into a (possibly predefined) number of clusters. **b** Hierarchical clustering creates a binary hierarchy tree structure over the instances.

classification, the meaning of a class is usually unknown in clustering, though. Clustering learns patterns in the similarities of instances based on similarity measures like those used in information retrieval (see above). The resulting model can assign arbitrary instances to one of the given clusters. In text mining, clustering is e.g. used to detect texts with similar properties.

Conceptually, two basic types of clustering exist, as shown in Fig. 2.4. While *flat clustering* partitions instances without specifying associations between the created clusters, *hierarchical clustering* organizes instances in a hierarchical tree (Manning et al. 2008). Each node in the tree represents a cluster of a certain size. The root cluster consists of all instances and each leaf refers to a single instance. A flat clustering can be derived from a hierarchical clustering through cuts in the tree. The tree is incrementally created by measuring distances between instances and clusters. To this end, a cluster is e.g. represented by its *centroid*, i.e., the average of all instances in the cluster (Manning et al. 2008). In general, both clustering types have certain advantages with respect to efficiency and cluster quality. We rely on hierarchical clustering in Chap. 5 for reasons discussed there. In particular, we perform *agglomerative hierarchical clustering* where the hierarchy is created bottom-up, beginning with the single instances (Manning et al. 2008).

**Further Learning Types.** Some other machine learning types are used more or less frequently in text mining, part of which are variations of supervised learning. Sporadically, we talk about *semi-supervised learning* in this book, which targets at tasks where much input data is available, but little known training data. Intuitively, semi-supervised learning first derives patterns from the training data and then applies knowledge about these patterns to find similar patterns in the other data (Chapelle et al. 2006). Some research in information extraction proposes *self-supervised learning* approaches that aim to fully overcome the need for known data by generating training data on their own (Banko et al. 2007). This can work when some output information is accessible without uncertainty. We present an according approach in Chap. 5. Also, we employ an entity recognition algorithm that relies on *sequence labeling* (cf. Appendix A). Sequence labeling classifies each of a sequence of instances, exploiting information about the other instances.

While there are more learning types, like reinforcement learning, recommender systems or one-class classification, we do not apply them in this book and, so, omit to introduce them here for brevity.

### 2.1.5 Development and Evaluation

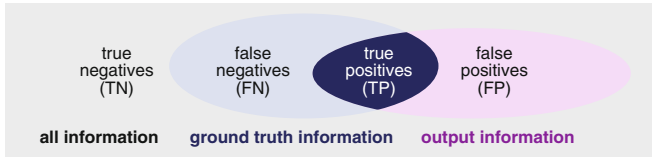
As discussed, text analysis aims to approximate unknown target functions, which map input texts to output information. To this end, both rule-based and statistical text analysis approaches are usually developed based on a collection of input texts with known properties. Still, the output information they produce will, in general, not always be correct. The reasons behind relate to the ambiguity of natural language (see above) and to the incompleteness and inexactness of the input data (Witten and Frank 2005).

As a consequence, an empirical evaluation of the quality of a text analysis approach is of high importance and mostly closely connected to its development. Here, with quality we primarily refer to the effectiveness and efficiency of an approach, as outlined in Sect. 1.2.<sup>11</sup> While the concrete quality measures that are adequate for evaluation partly differ between the mentioned tasks from information retrieval, natural language processing, and data mining, in principle all three fields rely on similar methods (Manning et al. 2008; Jurafsky and Martin 2009; Witten and Frank 2005). In particular, experiments are performed, in which an approach is first developed on a collection of input texts. Then, its quality is measured on previously unseen input texts and compared to alternative approaches. In the following, we detail the outlined concepts for text analysis.

**Text Corpora.** In this book, we approach text analysis in a *corpus linguistics* manner, i.e., we address all tasks based on the analysis of samples of real-world texts, called text corpora. A *text corpus* is a principled collection of texts that has been compiled to analyze a problem related to language (Biber et al. 1998). Here, we consider corpora that serve for the development of text analyses (e.g. for sentiment analysis). Text corpora often contain annotations, especially annotations of the target variable that represents the output information to be inferred (e.g. the sentiment polarity of a text). In contrast to the annotations produced by text analysis algorithms, corpus annotations have usually been created manually in a cost-intensive *annotation process*. To avoid such a process, they can sometimes be derived from existing metadata (such as the author or star rating of a review). In both cases, they are seen as *ground truth* annotations (Manning et al. 2008).

---

<sup>11</sup>Besides effectiveness and efficiency, we also investigate the *robustness* and *intelligibility* of text analysis in Chap. 5. Further details are given there.



**Fig. 2.5** Venn diagram showing the four sets that can be derived from the ground truth information of some type in a collection of input texts and the output information of that type inferred from the input texts by a text analysis approach.

To allow for generalization, the compilation of texts in a text corpus usually aims to be *representative* for some target variable  $C$ , i.e., it includes the full range of variability of texts with respect to  $C$  (Biber et al. 1998). We discuss representativeness at the beginning of Chap. 5. For evaluation, also the distribution of texts over the values of  $C$  should be representative for the real distribution. In machine learning, though, a *balanced* distribution, where all values of  $C$  are evenly represented, is favorable according to statistical learning theory (Batista et al. 2004).

**Effectiveness.** Text analysis approaches are mostly evaluated with respect to their effectiveness, which quantifies the extent to which output information is correct. Given a collection of input texts with ground truth annotations for the target variable  $C$  of a text analysis approach, the effectiveness of all approaches relevant in this book can be evaluated in the sense of a two-class classification task, i.e., whether the decision to produce each possible instance of  $C$  is correct or not.

We call the output instances of an approach the *positives* and all other instances the *negatives*. On this basis, four different sets can be distinguished (Witten and Frank 2005): *True positives (TP)* are all positives that belong to the ground truth, *true negatives (TN)* are all negatives that do not belong to the ground truth, *false negatives (FN)* are all negatives that belong to the ground truth, and *false positives (FP)* are all positives that do not belong to the ground truth. Figure 2.5 illustrates all sets.

Once the four sets are given, effectiveness can directly be quantified with different measures whose adequateness depends on the task at hand. One measure is the *accuracy*  $a$ , which denotes the ratio of correct decisions:

$$a = (|TP| + |TN|) / (|TP| + |TN| + |FP| + |FN|)$$

The accuracy is an adequate measure, when all decisions are of equal importance. This holds for many text classification tasks as well as for other text analysis tasks, in which every portion of an input text is annotated and, thus, requires a decision, such as in tokenization. In contrast, especially in information extraction tasks like entity recognition, the output information usually covers only a small amount of the processed input texts. As a consequence, high accuracy can be achieved by simply producing no output information at all. Thus, accuracy is inadequate if the true negatives are of low importance. Instead, it seems more suitable to measure effectiveness in terms of the *precision*  $p$  and the *recall*  $r$  (Manning and Schütze 1999):

$$p = |TP| / (|TP| + |FP|) \quad r = |TP| / (|TP| + |FN|)$$

Precision quantifies the ratio of output information that is inferred correctly, while recall refers to the ratio of all correct information that is inferred. In many cases, however, achieving either high precision or high recall is as easy as useless. E.g., perfect recall can be obtained by producing all possible output information. If both high precision and high recall are desired, their harmonic mean can be computed, called the *F<sub>1</sub>-score* (or *F<sub>1</sub>-measure*), which rewards an equal balance between *p* and *r* (van Rijsbergen 1979):

$$f_1 = 2 \cdot p \cdot r / (p + r)$$

The four defined effectiveness measures are used in a number of experiments in this book. In addition, we compute the mean *regression error* of numeric predictions in Chap. 5, which is defined as the average difference between a predicted and a correct value. Also, we talk about the *labeled attachment score* in Chap. 3, which denotes the proportion of correctly classified tokens in dependency parsing (Bohnet 2010). Other effectiveness measures are left out here for lack of relevance.

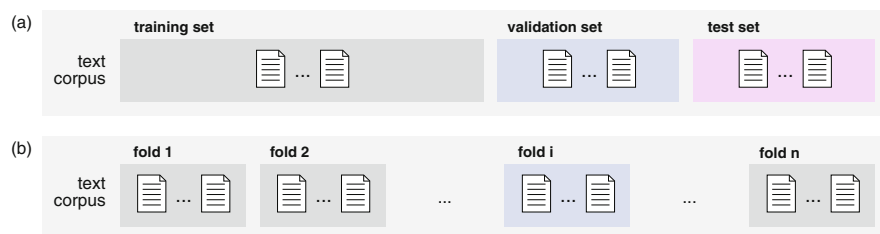
**Efficiency.** Since we aim to perform text analysis on large amounts of input texts, not only effectiveness is important in this book, but also efficiency. In general, efficiency quantifies costs in terms of the consumption of time or memory (Cormen et al. 2009). While we sporadically discuss the effects of high memory consumptions in the subsequent chapters, we always refer to efficiency here as the *run-time* (also called running time) an approach takes to process a given input. We use the terms efficiency and *run-time efficiency* interchangeably from here on.

We quantify the efficiency of each algorithm  $A_i$  and pipeline  $\Pi$  in terms of two measures, both specified in seconds or milliseconds: First, the absolute *overall run-times*  $t_i(D)$  and  $t_\Pi(D)$  on an input text  $D$ , and second, the average *run-time per portion of text* (mostly, per sentence),  $t(A_i)$  and  $t(\Pi)$ . All run-times are averaged over a defined number of runs (either 5 or 10) and complemented by their *standard deviation*  $\sigma$ . In some cases, we compute specific run-times (say, the training time), as defined where given.

**Experiments.** In corpus linguistics, the general method to develop and evaluate both rule-based and statistical text analysis approaches is to perform experiments using a split of a corpus into different *datasets*. One (or the union of some) of these datasets is analyzed manually or automatically for the development, and the others are processed to evaluate a developed approach (Jurafsky and Martin 2009).<sup>12</sup> We realize the process underlying this method in the following two ways in this book, both of which are very common in statistical evaluation (Witten and Frank 2005).

---

<sup>12</sup>The development of statistical approaches benefits from a balanced dataset (see above). This can be achieved through either undersampling minority classes or oversampling majority classes. Where needed, we mostly perform the latter using random duplicates.



**Fig. 2.6** Two ways of splitting a corpus for development and evaluation: **a** A training set is used for development, a validation set for optimizing parameters, and a test set for evaluation. **b** Each fold  $i$  out of  $n$  folds serves for evaluation in the  $i$ -th of  $n$  runs. All others are used for development.

In most cases, we split a given text corpus into a training set, a validation set, and a test set, as illustrated in Fig. 2.6(a).<sup>13</sup> After developing an approach on the *training set*, the quality of different configurations of the approach (e.g. with different feature vectors or learning parameters) is iteratively evaluated on the *validation set*. The validation set thereby serves for optimizing the approach, while the approach adapts to the validation set. The best configuration is then evaluated on the *test set* (also referred to as the held-out set). A test set represents the unseen data. It serves for estimating the quality of an approach in practical applications.<sup>14</sup>

The described method appears reasonable when each dataset is of sufficient size and when the given split prevents from bias that may compromise the representativeness of the respective corpus. In other cases, an alternative is to perform (stratified) *n-fold cross-validation* (Witten and Frank 2005). In *n*-fold cross-validation, a text corpus is split into  $n$  (e.g. 10) even folds, assuring that the distribution of the target variable is similar in all folds. The development and evaluation then consist of  $n$  runs, over which the measured quality of an approach is averaged. In each run  $i$ , the  $i$ -th fold is used for evaluation and all others for development. Such a split is shown in Fig. 2.6(b). We conduct according experiments once in Chap. 5.

**Comparison.** The measured effectiveness and efficiency results of a text analysis approach are usually compared to alternative ways of addressing the given task in order to assess whether the results are good bad. For many tasks, an upper-bound ceiling of effectiveness is assumed to be the effectiveness a human would achieve (Jurafsky and Martin 2009).<sup>15</sup> For simplicity, effectiveness is thus often measured with respect to the human-annotated ground truth. While there is no general upper-bound efficiency ceiling, we see in the subsequent chapters that optimal efficiency can mostly be determined in a given experiment setting. We call every

<sup>13</sup>Many text corpora already provide an according corpus split, including most of those that we use in our experiments (cf. Appendix C).

<sup>14</sup>In some of our efficiency experiments, no parameter optimization takes place. We leave out the use of validation set in these cases, as pointed out where relevant.

<sup>15</sup>Some exceptions to the truth of this assumption exist, of course. For instance, authorship attribution (see above) is expected to be often hard for humans.

upper-bound ceiling of a quality measure the *gold standard* and we define the gold standard accordingly where needed.

For interpretation, results are also checked whether they are significantly better than some lower bound *baseline* (Jurafsky and Martin 2009). E.g., an accuracy of 40% in a 5-class classification task may appear low, but it is still twice as good as the accuracy of guessing. The standard way to determine lower bounds is to compare an evaluated approach with one or more approaches that are trivial (like guessing), standard (like a bag-of-words approach in text classification), state-of-the-art or at least well known from the literature. We compare our approaches to according baselines in all our experiments in Chaps. 3–5. In these experiments, we mostly consider complex text analysis processes realized by pipelines of text analysis algorithms, as presented next.

## 2.2 Text Analysis Tasks, Processes, and Pipelines

In Sect. 1.2, we have roughly outlined that text mining requires task-specific text analysis processes with several classification, extraction, and similar steps. These processes are realized by text analysis pipelines that infer output information from input texts in order to satisfy a given information need. Since text analysis pipelines are in the focus of all approaches proposed in this book, we now explain the outlined concepts of text analysis more comprehensively and we illustrate them at the end. Thereby, we define the starting point for all discussions in Chaps. 3–5.

### 2.2.1 Text Analysis Tasks

As specified in Sect. 1.2, we consider tasks in which we are given input texts and an information need to be addressed. The goal is to infer output information from the input texts that is relevant with respect to the information need. Here, we detail basic concepts behind such tasks. An extension of these concepts by quality criteria to be met follows in Chap. 3 after discussing the optimality of text analysis pipelines.

**Input Texts.** In principle, the input we deal with in this book may be either given in the form of a *collection of texts* or a *stream of texts*. The former denotes a set of natural language texts  $\{D_1, \dots, D_n\}$ ,  $n \geq 1$ , usually compiled with a purpose, like a text corpus (see above). With the latter, we refer to continuously incoming natural language text data. We assume here that such data can be split into logical segments  $D_1, D_2, \dots$  (technically, this is always possible). Given that the speed of processing a stream can be chosen freely, we can then deal with collections and streams in the same way except for the constraint that streaming data must be processed in the order in which it arrives. We denote both a collection and a stream as **D**.



We see a single text  $D \in \mathbf{D}$  as the atomic input unit in text analysis tasks. While no general assumptions are made about the length, style, language, or other properties of  $D$ , we largely restrict our view to fully *unstructured texts*, i.e., plain texts that have no explicit structure aside from line breaks and comparable character-level formatings. Although text mining may receive several types of documents as input, such as HTML files in case of web applications, our restriction is not a limitation but rather a focus: Most text analysis approaches work on plain text. If necessary, some content extraction is, thus, usually performed in the beginning that converts the documents into plain text (Gottron 2008). Besides, some of our approaches in the subsequent chapters allow the input texts to already have annotations of a certain set of zero or more information types  $\mathbf{C}_0$ , which holds for many text corpora in computational linguistics research (cf. Sect. 2.1).

**Output Information.** In Sect. 2.1, different information types have been mentioned, e.g. tokens, part-of-speech tags, concrete types of entities and relations, certain text classification schemes, etc. In general, an information type  $C = \{c_1, c_2, \dots\}$  denotes the set of all pieces of information  $c \in C$  that represent a particular lexical, syntactic, semantic, or pragmatic concept. We postpone a more exact definition of information types to Chap. 3, where we formalize the expert knowledge for tackling text analysis tasks automatically. A concrete information type is denoted with an upper-case term in this book, such as the *Token* type or a relation type *Founded*. To signal that an information type is part of an event or relation type, we append it to that type in lower case, such as *Token.lemma* or *Founded.time*.

Now, in many tasks from information extraction and text classification, the goal is to infer output information of a specific set of information types  $\mathbf{C}$  from texts or portions of texts. Here, we use the set notation as in propositional logic (Kleine et al. 1999), i.e., a set  $\mathbf{C} = \{C_1, \dots, C_k\}$ ,  $k \geq 1$ , can be understood as a conjunction  $C_1 \wedge \dots \wedge C_k$ . In case of *Founded(Organization, Time)* from Sect. 2.1, for example, a text or a portion of text that contains an instance of this relation type must comprise an organization name and a time information as well as a representation of a foundation relation between them. Hence, the relation type implicitly refers to a conjunction  $\text{Founded} \wedge \text{Founded.organization} \wedge \text{Founded.time}$ , i.e., a set  $\{\text{Founded}, \text{Founded.organization}, \text{Founded.time}\}$ .

**Information Needs.** Based on the notion of information types, we can define what information is relevant with respect to an information need in that it helps to fulfill the need. The goal of text mining is to infer new information of specified types from a collection or a stream of input texts  $\mathbf{D}$  (cf. Sect. 2.1). From a text analysis perspective, addressing an information need hence means to return *all* instances of a given set of information types  $\mathbf{C}$  that are found in  $\mathbf{D}$ . In this regard,  $\mathbf{C}$  itself can be seen as a specification of an information need, a single information need in particular. Accordingly, a combination of  $k > 1$  information needs (say, the desire to get information of  $k = 2$  relation types at the same time) refers to a disjunction  $C_1 \vee \dots \vee C_k$ . In practical text mining applications, parts of an information need might be specified beforehand. E.g., *Founded(Organization, "1998")* denotes the request to extract all names of organizations founded in the year 1998.

We assume in this book that information needs are already given in a formalized form. Consequently, we can concentrate on the text analysis processes required to address information needs. Similar to information types, we actually formalize information needs later on in Chap. 3.

### 2.2.2 Text Analysis Processes

In real-world text analysis tasks, information needs refer to combinations of concrete information types from natural language processing. We have introduced the general analyses that can infer these information types from input texts in Sect. 2.1. However, even the inference of a single information type often requires several analysis steps, each of which refers to one text analysis. The reason is that many text analyses require as input the output of other text analyses, which in turn depend on further text analyses, and so forth. As a consequence, addressing an information need means the realization of a complex text analysis process. Common examples refer to the areas of information extraction and text classification, as sketched below. In general, also other natural language processing tasks entail a number of analysis steps, like *semantic role labeling*, which seeks for the associations between the verb in a sentence and its arguments (Gildea and Jurafsky 2002). Some processes in the intersection of the different areas comprise almost 30 steps (Solovyev et al. 2013).

**Information Extraction.** As discussed in Sect. 2.1, information extraction often aims at filling complex event templates whose instances can be stored in databases. Therefore, information extraction processes are made up of possibly tens of analysis steps, covering the whole spectrum from lexical and syntactic preprocessing over entity recognition, relation extraction, and event detection to coreference resolution and normalization. While we investigate processes with up to 11 distinguished analysis steps in the experiments of the subsequent chapters, for brevity we here exemplify only that even binary relation extraction may already require several steps.

In particular, assume that instances of the above-mentioned relation type *Founded* shall be extracted from the sentences of an input text using supervised classification (cf. Sect. 2.1). Before features can be computed for classification, both organization and time entities need to be recognized in the sentences. Entity recognition often relies on the output of a chunker, while relation extraction benefits from information about the positions of candidate entities in a dependency parse tree (Sarawagi 2008). These analyses are usually based on part-of-speech tags and lemmas, which mostly makes a preceding tokenization and sentence splitting necessary.

**Text Classification.** In terms of the number of distinguished analysis steps, text classification processes tend to be shorter than information extraction processes, because the focus is usually on the computation of feature values the class of an input text is inferred from. Still, many features rely on the existence of previously produced instances of information types, especially those resulting from lexical and shallow syntactic analyses (cf. Sect. 2.1). In sentiment analysis, for example, some

baseline approaches derive features from the output of tokenization and part-of-speech tagging only (Pang et al. 2002), while others e.g. also perform chunking, and extract relations between recognized domain-specific terms (Yi et al. 2003). Moreover, some text classification approaches rely on fine-grained information from semantic and pragmatic text analyses, such as the sentiment analysis in our case study ARGUANA that we introduce in Sect. 2.3.

**Realization.** The complexity of common text analysis processes raises the question of how to approach a text analysis task without losing the mind in the process, like van Gogh according to the introductory quote of this chapter. As the examples above indicate, especially the dependencies between analysis steps are not always clear in general (e.g. some entity recognition algorithms require part-of-speech tags, while others do not). In addition, errors may propagate through the analysis steps, because the output of one step serves as input to subsequent steps (Bangalore 2012). This entails the danger of achieving limited overall effectiveness, although each single analysis step works fine. A common approach to avoid error propagation is to perform joint inference, where all or at least some steps are performed concurrently. Some studies indicate that joint approaches can be more effective in tasks like information extraction (cf. Sect. 2.4 for details).<sup>16</sup>

For our purposes, joint approaches entail limitations, though, because we seek to realize task-specific processes ad-hoc for arbitrary information needs from text analysis. Moreover, joint approaches tend to be computationally expensive (Poon and Domingos 2007), since they explore larger search spaces emanating from combinations of information types. This can be problematic for the large-scale scenarios we target at. Following Buschmann et al. (1996), our requirements suggest the resort to a sequence of small analysis steps composed to address a task at hand. In particular, small analysis steps allow for an easy recombination and they simplify the handling of interdependencies. Still, a joint approach may be used as a single step in an according sequence. We employ a few joint approaches (e.g. the algorithm ENE described in Appendix A.1) in the experiments of this book. Now, we present the text analysis pipelines that realize sequences of analysis steps.

### 2.2.3 Text Analysis Pipelines

Pipelines denote the standard approach to realize text analysis processes. Although the application of pipelines is ubiquitous in natural language processing (Hollingshead and Roark 2007), rarely their design and execution are defined formally. As sketched in Sect. 1.2, a text analysis pipeline processes a collection or a stream of

---

<sup>16</sup>A simple example is the interpretation of periods in tokenization and sentence splitting: Knowing sentence boundaries simplifies the determination of tokens with periods like abbreviations, but knowing the abbreviations also helps to determine sentence boundaries.

input texts with a sequence of algorithms in order to stepwise produce a set of output information types.<sup>17</sup> We model a text analysis pipeline in the following way.<sup>18</sup>

**Text Analysis Pipeline.** A text analysis pipeline  $\Pi$  is a 2-tuple  $\langle \mathbf{A}, \pi \rangle$  where

1. **Algorithm Set.**  $\mathbf{A} = \{A_1, \dots, A_m\}$  is a set of  $m \geq 1$  text analysis algorithms, and
2. **Schedule.**  $\pi \subset \{(A_i < A_j) \mid A_i, A_j \in \mathbf{A}\}$  is a strict total order on  $\mathbf{A}$ .

For a concise presentation, we sometimes shorten the notation of a text analysis pipeline with  $m$  algorithms in this book as  $\Pi = (A_1, \dots, A_m)$  and we often refer to text analysis pipelines as *pipelines* only. Also, we discuss some special pipeline cases, namely, *empty pipelines* with  $m = 0$  algorithms, *partial pipelines* that employ a subset of the algorithms in  $\mathbf{A}$  only, and *partially ordered pipelines* that have a *partial schedule*, defining a partial order only.

**Text Analysis Algorithms.** According to our motivation from Chap. 1, we consider pipelines in a *universe*  $\Omega$  where the set  $\mathbf{A}_\Omega$  of all available text analysis algorithms is arbitrary but fixed. Each algorithm from  $\mathbf{A}_\Omega$  employed in a pipeline  $\Pi = \langle \mathbf{A}, \pi \rangle$  realizes one analysis step of a text analysis process, performing any text analysis like those outlined in Sect. 2.1. By that, such an algorithm can be seen as the atomic processing unit in text analysis.

While an algorithm may perform several analyses, feature computations, and similar, we handle all algorithms in a black-box manner, not considering their internal operations. Instead, we describe each algorithm  $A_i \in \mathbf{A}$  by its input and output behavior. In particular,  $A_i$  requires a text and instances of a (possibly empty) set of information types  $\mathbf{C}_i^{(in)}$  as input and  $A_i$  produces instances of a set  $\mathbf{C}_i^{(out)}$  as output. A more formal definition is provided in Chap. 3, where we also talk about the effects of language and other input properties on the applicability and quality of an algorithm. Technically, algorithms produce annotations of texts or portions of texts, as discussed in Sect. 2.1. In some parts of this book, we assume that no text analysis is performed by more than algorithm in a pipeline (as mentioned where relevant). In this case, an algorithm adds annotations to a text, but it never deletes or overwrites annotations given already.

<sup>17</sup>Some related work speaks about *workflows* rather than pipelines, such as (Shen et al. 2007). The term workflow is more general, also covering cascades where the input can take different paths. Indeed, such cascades are important in text analysis, e.g. when the sequence of algorithms to be executed depends on the language of the input text. From an execution viewpoint, however, we can see each taken path as a single pipeline in such cases.

<sup>18</sup>While named differently, the way we represent pipelines and the algorithms they compose here largely conforms to their realization in standard software frameworks for text analysis, like APACHE UIMA, <http://uima.apache.org>, accessed on June 15, 2015.

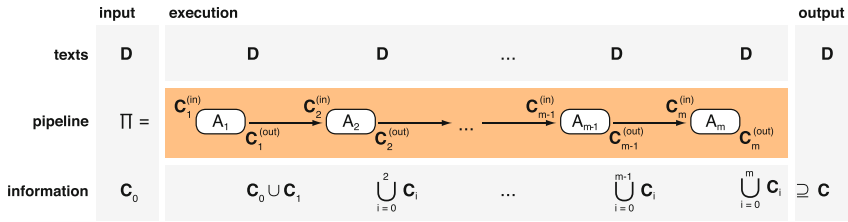
**Schedules.** The schedule  $\pi$  of a pipeline  $\Pi = \langle \mathbf{A}, \pi \rangle$  prescribes the order in which the algorithms in  $\mathbf{A}$  are applied to every input text. As such,  $\pi$  rules whether the input requirements of each algorithm  $A_i \in \mathbf{A}$  are fulfilled, i.e., whether all information types in  $\mathbf{C}_i^{(in)}$  are produced by the algorithms preceding  $A_i$ . While some algorithms will fail completely if their requirements are not met (e.g. entity recognition must precede the extraction of respective relations), others will behave unpredictably, usually degrading significantly in effectiveness. Schedules play an important role in the approaches proposed in Chaps. 3 and 4. We formalize desired properties of schedules in Chap. 3. Afterwards, we reveal that *scheduling*, i.e., the definition of a schedule for a given algorithm set, impacts pipeline efficiency.

**Pipeline Design.** The general design style of text analysis pipelines is fixed. According to their definition above, it largely corresponds to the architectural pattern *pipes and filters*. Pipes and filters divides a process into several sequential processing steps that are connected by the data flow of the process (Buschmann et al. 1996). In particular, the output data of one processing step is the input to the subsequent step. Different from pipes and filters architectures in areas like computer graphics (Angel 2008), however, usual text analysis algorithms do not really transform their input, but they add information in the sense of annotations to their input only. In terms of data management, text analysis pipelines hence rather follow a *blackboard architecture* (Hayes-Roth 1985), i.e., they have a shared knowledge base (with texts and annotations) that all algorithms can access.

In principle, text analysis processes may also be realized with so called *tees and joins*, i.e., with algorithms that have more than one predecessor or successor (Buschmann et al. 1996). Without parallelization, a respective pipeline needs to be linearized for execution anyway. In the mentioned area of computer graphics, different orderings of processing steps like transformations can lead to different possibly useful results (Angel 2008). In contrast, as long as annotations are only added to a text, either the ordering of two algorithms does not matter (in case they are independent) or there is exactly one correct ordering (otherwise). We discuss limitations of the pipeline architecture in Sect. 2.4 and we clarify both the notion of correct orderings and the effects of parallelization in Chap. 3.

Since the analysis steps to be performed depend on the text analysis task to be tackled, pipelines are task-specific. Traditionally, a pipeline  $\Pi = \langle \mathbf{A}, \pi \rangle$  is designed manually by a human expert by when given a task by selecting and scheduling an appropriate subset  $\mathbf{A}$  of the set of available text analysis algorithms  $\mathbf{A}_\Omega$  (Ferrucci and Lally 2004).

**Pipeline Execution.** Figure 2.7 illustrates how a pipeline  $\Pi$  is traditionally executed on a collection or a stream of input texts  $\mathbf{D}$ , for which a set of information types  $\mathbf{C}_0$  is already provided in advance (see above). For a clear presentation, the type level is shown for most of the concepts explained on the previous pages. Actually, each text from  $\mathbf{D}$  runs sequentially through the  $m$  algorithms in  $\Pi$ , and each algorithm  $A_i$  in  $\Pi$  produces instances of a set of information types  $\mathbf{C}_i^{(out)}$  as output by adding annotations to the text. After the execution of  $A_i$ , the union  $\bigcup_{j=0}^i \mathbf{C}_j$  of information



**Fig. 2.7** Abstract view of executing a text analysis pipeline  $\Pi = (A_1, \dots, A_m)$  on a collection or a stream of input texts  $\mathbf{D}$  in order to produce a set of output information types  $\mathbf{C}$ . For every text in  $\mathbf{D}$ , each algorithm  $A_i$ ,  $1 \leq i \leq m$ , adds instances of a set of information types  $\mathbf{C}_i^{(out)}$  to the instances of all inferred information types  $\bigcup_{j=0}^i \mathbf{C}_j$ . This set is initialized with instances of a possibly empty set of information types  $\mathbf{C}_0$ .

types inferred so far is given. The union of information types inferred by all algorithms employed in  $\Pi$  is supposed to be a superset of the set of information types  $\mathbf{C}$ , which represents the information need to be addressed. So, we observe that the pipeline controls the process of creating all information sought for.

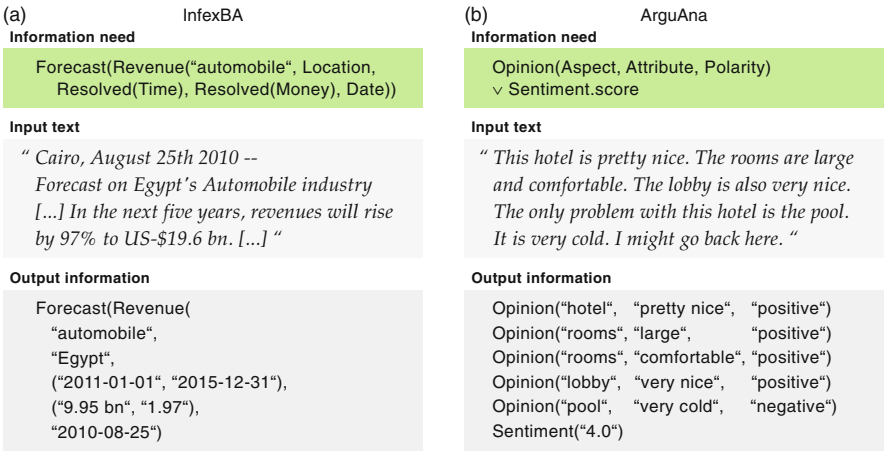
At the same time, the processed input texts themselves do not change at all within the realized text analysis process, as emphasized in the upper part of Fig. 2.7. I.e., each algorithm traditionally processes each input text completely. We present an enhancement of such an execution in Chap. 3 after summarizing existing approaches in Sect. 2.4. Before, we introduce the case studies we examine in order to evaluate all of our approaches.

## 2.3 Case Studies in This Book

In this book, we aim to improve the design, efficiency, and robustness of the text analysis pipelines defined in Sect. 2.2. All developed approaches are evaluated in empirical experiments with text analysis tasks that refer to a selection of scientifically and/or industrially relevant case studies. Some of these case studies are associated to two of our research projects, INFEXBA and ARGUANA, whereas the others are known from related research. We briefly outline all of them in this section.

### 2.3.1 InfexBA – Information Extraction for Business Applications

INFEXBA is a research project that was funded by the GERMAN FEDERAL MINISTRY OF EDUCATION AND RESEARCH (BMBF) from 2008 to 2010 under contract number 01IS08007A. The primary goal of INFEXBA was to develop text mining applications



**Fig. 2.8** Example for output information inferred from an input text to address the main information needs in **a** the INFEXBA project and **b** the ARGUANA project.

for an automatic market analysis in the sense of a focused web search engine. Given an organization or market name, the search engine retrieves a set of candidate web pages, extracts, normalizes, and aggregates information about financial forecasts for the given subject from the web pages, and visualizes the aggregated information. More details about the project and its results can be found at <http://infexba.upb.de>.

Within the project, we proposed to perform text mining in a sequence of eight information retrieval, natural language processing, and data mining stages (Wachsmuth et al. 2010), but the focus was on those stages related to information extraction. In particular, information extraction begins after converting all retrieved web pages into plain texts and ends after normalization. We can view the set of these plain texts as a collection of input texts **D**. One of the main tasks tackled in INFEXBA was to extract each revenue forecast for the given organization or market at a certain location (including the date it was published) from **D** and to bring the time and money information associated to the forecast into resolved and normalized form. We can specify the underlying information need as follows:

*Forecast(Revenue(Subject, Location, Resolved(Time), Resolved(Money), Date))*

Given the market “*automobile*” as the subject, Fig. 2.8(a) exemplifies for one input text what information is meant to satisfy the specified need. Some information is directly found in the text (e.g. “*Egypt*”), some must be computed (e.g. the money amount of “*9.95 bn*” at the end of 2010).

We refer to the INFEXBA project in the majority of those experiments in Chaps. 3 and 4 that deal with pipeline design and optimization. For a focused discussion, we evaluate different simplifications of the presented information need, though. One such need, for instance, targets at all related pairs of time and money information that belong to revenue forecasts. Accordingly, the text analysis pipelines that we



evaluate still perform several preprocessing, entity recognition, relation extraction, event detection, and normalization steps in our experiments. In total, 21 different text analysis algorithms are employed in the evaluated pipelines, namely, all those described in Appendix A that are not used for classification or discourse parsing.

For the development and evaluation of approaches related to an automatic market analysis, a manually annotated text corpus with German online business news articles was created within INFEXBA, called the REVENUE CORPUS. We describe the corpus in Appendix C.1. Besides, some existing corpora are used, especially the CONLL-2003 dataset (cf. Appendix C.4).

### 2.3.2 *ArguAna – Argumentation Analysis in Customer Opinions*

As INFEXBA, ARGUANA is a research project that was funded by the GERMAN FEDERAL MINISTRY OF EDUCATION AND RESEARCH (BMBF). It ran from 2012 to 2014 under contract number 01IS11016A. The project aimed at the development of novel text analysis algorithms for fine-grained opinion mining from customer product reviews. In particular, a focus was on the analysis of the sequence of single arguments in a review in order to capture and interpret the review’s overall argumentation. From the results of a set of reviews, a text mining application cannot only infer collective opinions about different aspects of a product, but also provide a precise classification of the sentiment of customers with respect to the product. More information about ARGUANA is given at <http://www.arguana.com>.

In the project, we developed a complex text analysis process to tackle the underlying text classification and information extraction tasks: First, the body of each review text from a collection of input texts **D** is segmented into its single subsentence-level discourse units. Every unit is classified as being either an objective fact, a positive, or a negative opinion. Discourse relations between the units are then extracted as well as products and aspects the units are about, together with their attributes. Finally, a sentiment score in the sense of the review’s overall rating is predicted. The output information helps to address the following information need:

*Opinion(Aspect, Attribute, Polarity)  $\wedge$  Sentiment.score*

As for this information need, the disjunctions defined in Sect. 2.2 should not be misunderstood in the sense that addressing one of the two connected conjunctions suffices. Rather, it states that instances of either of them is relevant. Figure 2.8(b) shows the output information for a sample hotel review. The sentiment score comes from a scale between 1 (worst) and 5 (best).

We refer to the ARGUANA project mostly in the evaluation of pipeline robustness in Chap. 5. There, we omit the recognition of products, aspects, and attributes, but we focus on text classification approaches based on the extracted facts, opinions, and discourse relations. The remaining text analysis process is realized with the following



pipeline:  $\Pi_{\text{ARGUANA}} = (\text{SSE}, \text{STO}_2, \text{TPO}_1, \text{PDU}, \text{CSB}, \text{CSP}, \text{PDR}, \text{CSS})$ . For details on the algorithms, see Appendix A.

The experiments related to the ARGUANA project are based on two English collections of texts, consisting of reviews from the hotel domain and the film domain, respectively. In particular, we rely on our own ARGUANA TRIPADVISOR CORPUS developed within the project (cf. Appendix C.2) as well as on the widely used SENTIMENT SCALE DATASET (cf. Appendix C.4).

### 2.3.3 Other Evaluated Text Analysis Tasks

Most of the concrete text analysis tasks in this book are at least loosely connected to the presented projects INFEXBA and ARGUANA. In some cases, though, we provide complementary results obtained in other experiments in order to achieve more generality or to analyze the generalizability of our approaches. All noteworthy results of this kind are associated to the following three text analysis tasks.

**Genia Event Extraction.** GENIA denotes one of the main evaluation tasks of the BIONLP SHARED TASK (Kim et al. 2011). While the latter deals with the general question of how text mining can help to recognize changes of states of bio-molecules described in the biomedical literature, the former specifically targets at the extraction of nine different event types that relate a number of proteins, other entities, or other events. For instance, a *Phosphorylation* event refers to an entity of the *Protein* type as well as to some information that denotes a binding site of the protein (Kim et al. 2011). In the evaluation of automatic pipeline design in Chap. 3, we consider the formal specifications of several entity recognition and event detection algorithms that infer information types relevant in the GENIA task.

**Named Entity Recognition.** A *named entity* is an entity that refers to a unique concept from the real world. While numerous types of named entities exist, most of them tend to be rather application-specific (Jurafsky and Martin 2009). Some types, though, occur in diverse types of natural language texts, of which the most common are person names, location names, and organization names. They have been in the focus of the CONLL-2003 shared task on *named entity recognition* (Tjong et al. 2003). In Chap. 4, we analyze the distribution of the three entity types in several text corpora from Appendix C in the context of influencing factors of pipeline efficiency. There, we rely on a common sequence labeling approach to named entity recognition (cf. Sect. 2.1), using the algorithm ENE (cf. Appendix A) in the pipeline  $\Pi_{\text{ENE}} = (\text{SSE}, \text{STO}_2, \text{TPO}_1, \text{PCH}, \text{ENE})$ .

**Language Function Analysis.** Finally, we address the text classification task *language function analysis* in this book. We introduced this task in Wachsmuth and Bujna (2011). As argued there, every text can be seen as being predominantly expressive, appellative, or informative. These *language functions* define an abstract classification scheme, which can be understood as capturing a single aspect of genres (Wachsmuth

and Bujna 2011). In Chap. 5, we concretize the scheme for product-related texts in order to then outline how much text classification depends on the domain of the input texts. Moreover, in Chap. 3 we integrate CLF in the information extraction pipelines from INFEXBA (see above). In particular, we employ CLF to filter possibly relevant candidate input texts, which can be seen as one of the most common applications of text classification in text mining.

## 2.4 State of the Art in Ad-Hoc Large-Scale Text Mining

With the approaches developed in this book, we seek to enable the use of text analysis pipelines for ad-hoc large-scale text mining (cf. Sect. 1.3). Several other approaches have been proposed in the literature that tackle similar problems or that tackle different problems but pursue similar goals. In this section, we survey the state of the art in these respects, focusing on text analysis to a wide extent, and we stress how our approaches extend the state of the art. From an abstract viewpoint, our discussion follows the overall structure of this book. It reuses content from the related work sections of most of our publications listed in Table 1.1 (Sect. 1.4).

### 2.4.1 Text Analysis Approaches

As defined in Sect. 2.2, we consider the classic realization of a text analysis process in the form of a pipeline, where each algorithm takes as input the output of all preceding algorithms and produces further output. Pipelines represent the most widely adopted text analysis approach (Bangalore 2012). The leading software frameworks for text analysis, APACHE UIMA and GATE, target at pipelines (cf. Sect. 1.3). Some of our approaches assume that no analysis is performed by more than one algorithm in a pipeline. This is usual, but not always the case (Whitelaw et al. 2008). As a consequence, algorithms can never make up for errors of their predecessors, which may limit the overall effectiveness of pipelines (Bangalore 2012). In addition, the task dependency of effective text analysis algorithms and pipelines (cf. Sects. 2.1 and 2.2) renders their use in the ad-hoc search scenarios we focus on problematic (Etzioni 2011). In the following, we describe the most important approaches to tackle these problems, grouped under the topics *joint inference*, *pipeline enhancement*, and *task independence*.

**Joint Inference.** We have already outlined joint inference as a way to avoid the problem of error propagation in classic pipelines in Sect. 2.2. Joint approaches infer different types of information at the same time, thereby mimicking the way humans process and analyze texts (McCallum 2009). Among others, tasks like entity recognition and relation extraction have been said to benefit from joint inference (Choi et al. 2006). However, the possible gain of effectiveness comes at the cost of lower

efficiency and less reusability (cf. Sect. 2.2), which is why we do not target at joint approaches in this book, but only integrate them when feasible.

**Pipeline Enhancement.** Other researchers have addressed the error propagation through iterative or probabilistic pipelines. In case of the former, a pipeline is executed repeatedly, such that the output of later algorithms in a pipeline can be used to improve the output of earlier algorithms (Hollingshead and Roark 2007).<sup>19</sup> In case of the latter, a probability model is built based on different possible outputs of each algorithm (Finkel et al. 2006) or on confidence values given for the outputs (Raman et al. 2013). While these approaches provide reasonable enhancements of the classic pipeline architecture, they require modifications of the available algorithms and partly also significantly reduce efficiency. Both does not fit well to our motivation of enabling ad-hoc large-scale text mining (cf. Sect. 1.1).

**Task Independence.** The mentioned approaches can improve the effectiveness of text analysis. Still, they have to be designed for the concrete task at hand. For the extraction of entities and relations, Banko et al. (2007) introduced *open information extraction* to overcome such task dependency. Unlike traditional approaches for predefined entity and relation types (Cunningham 2006), their system TEXTRUNNER efficiently looks for general syntactic patterns (made up of verbs and certain part-of-speech tags) that indicate relations. Instead of task-specific analyses, it requires only a keyword-based query as input that allows identifying task-relevant relations. While Cunningham (2006) argues that high effectiveness implies high specificity, open information extraction targets at web-scale scenarios. There, precision can be preferred over recall, which suggests the exploitation of redundancy in the output information (Downey et al. 2005) and the resort to highly reliable extraction rules, as in the subsequent system REVERB (Fader et al. 2011).

Open information extraction denotes an important step towards the use of text analysis in web search and big data analytics applications. Until today, however, it is restricted to rather simple binary relation extraction tasks (Mesquita et al. 2013). In contrast, we seek to be able to tackle arbitrary text analysis tasks, for which appropriate algorithms are available. With respect to pipelines, we address the problem of task dependency in Chap. 3 through an automatic design of text analysis pipelines.

### 2.4.2 Design of Text Analysis Approaches

In Sect. 2.2, we have discussed that text analysis processes are mostly realized manually in regard of the information need to be addressed. Also, the resulting text analysis approaches traditionally process all input texts completely. Not only APACHE UIMA and GATE themselves provide *tool support* for the construction and execution of

---

<sup>19</sup>Iterative pipelines are to a certain extent related to compiler pipelines that include feedback loops (Buschmann et al. 1996). There, results from later compiler stages (say, semantic analysis) are used to resolve ambiguities in earlier stages (say, lexical analysis).

according text analysis pipelines, as outlined below. In order to address information needs ad-hoc on large numbers of texts, a number of approaches have been proposed that, similar to us, aim for an *automatic construction* of text analysis approaches as well as for optimizing their execution by *filtering* relevant texts and portions of text. In Chap. 3, we detail that the key to make the approaches successful is the existence of a pool of reusable and formally specified text analysis algorithms and the like (Wimalasuriya and Dou 2010).

**Tool Support.** Kano et al. (2010) introduced U-COMPARE, which supports an easy but manual construction of text analysis pipelines. U-COMPARE targets at the automatic evaluation of pipelines on text corpora. Similarly, Yang et al. (2013) describe a framework for the comparison of different pipelines for the same task. Conversely, the tool WEBLICHT, associated to the project CLARIN-D on interoperable and scalable infrastructure for language research, allows setting up pipelines for automatic corpus annotation (Hinrichs et al. 2010). In contrast to these works, we realize pipeline construction fully automatically in order to enable ad-hoc text mining.

**Automatic Construction.** For automation, we rely on the artificial intelligence technique *planning* (Russell and Norvig 2009). Dezsényi et al. (2005) have proposed planning for composing information extraction algorithms. Unlike us, however, the authors neither realize nor evaluate planning and they disregard the quality of the composition. In related areas, approaches exist that plan knowledge discovery workflows of minimum length given an ontology of data mining algorithms (Žáková et al. 2011) or that sum up the costs of a planned sequence of data stream processing steps (Riabov and Liu 2006). While these approaches generally seem transferrable to text analysis, their quality functions do not apply to the efficiency and effectiveness criteria relevant here (cf. Sect. 2.1). Recently, Kano (2012) presented a first glance of the software platform KACHAKO, which composes and executes a defined set of algorithms largely automatically based on the standard algorithm descriptions of APACHE UIMA. While KACHAKO appears to be similar to our expert system for ad-hoc pipeline design described in Chap. 3, it is still not available yet, rendering an exact comparison hard.

An alternative to the automatic design of text analysis pipelines is implemented in SYSTEMT, which seeks to address the needs of enterprise analytics applications, such as scalability and usability (Chiticariu et al. 2010b). SYSTEMT follows the paradigms of *declarative information extraction* (Krishnamurthy et al. 2009): a user defines analysis steps with logical constraints in the form of a query, while the system manages the workflow (Doan et al. 2009). We do not adopt the declarative approach here, as it is restricted to rule-based text analyses (Reiss et al. 2008). Still, we rely on similar concepts. E.g., SYSTEMT restricts some analyses to scopes of a text based on location conditions in the given query (Shen et al. 2007), which resembles the filtering of the input control that we develop in Chap. 3.

**Filtering.** Our input control filters only relevant portions of an input text in each analysis step. The idea of filtering relevant texts and portions of texts is well-known in text analysis. Traditionally, filtering is performed based on word statistics or pre-defined patterns (Cowie and Lehnert 1996). Lewis and Tong (1992) analyze how

the filtering of complete texts at different positions in a pipeline impacts the effectiveness in complex extraction tasks. Other researchers observe that also classifying the relevance of sentences can help to improve effectiveness (Patwardhan and Riloff 2007; Jean-Louis et al. 2011). Nédellec et al. (2001) stress the importance of such filtering for all extraction tasks where relevant information is sparse. According to Stevenson (2007), a restriction to sentences may also limit effectiveness in event detection tasks, though. While we use filtering to improve efficiency, we provide evidence that our approach maintains effectiveness. Still, we allow specifying the sizes of filtered portions to trade efficiency for effectiveness.

Filtering approaches for efficiency often target at complete texts, e.g. using fast text classification (Stein et al. 2005) or querying approaches trained on texts with the relations of interest (Agichtein and Gravano 2003). A technique that filters portions of text is passage retrieval (cf. Sect. 2.1). While many text mining applications do not incorporate filtering until today, passage retrieval is common where information needs must be addressed in real-time, e.g. in question answering (Krikon et al. 2012). Cardie et al. (2000) compare the benefit of statistical and linguistic knowledge for filtering candidate passages, and Cui et al. (2005) propose a fuzzy matching of questions and possibly relevant portions of text. Sarawagi (2008) sees the efficient filtering of relevant portions of input texts as a main challenge of information extraction in large-scale scenarios. She complains that existing techniques are still restricted to hand-coded heuristics. Common heuristics aim for high recall in order not to miss relevant information later on, whereas precision can be preferred on large collections of texts under the assumption that relevant information appears redundantly (Agichtein 2005).

Different from all the outlined approaches, our filtering approach does not *predict* relevance, relying on vague models derived from statistics or hand-crafted rules. In contrast, our approach *infers* the relevant portions of an input text formally from the currently available information. Moreover, we discuss in Chap. 3 that the input control can be integrated with common filtering approaches. At the same time, it does not prevent most other approaches to improve the efficiency of text analysis.

### 2.4.3 *Efficiency of Text Analysis Approaches*

Efficiency has always been a main aspect of algorithm research (Cormen et al. 2009). For a long time, most rewarded research on text analysis focused on effectiveness as did the leading evaluation tracks, such as the MESSAGE UNDERSTANDING CONFERENCES (Chinchor et al. 1993) or the CONLL SHARED TASK. In the latter, efficiency has at least sometimes been an optional evaluation criterion (Hajič et al. 2009). In times of big data, however, efficiency is getting increasing attention in both research and industry (Chiticariu et al. 2010b). While the filtering techniques from above denote one way to improve efficiency, the filtered texts or portions of texts still often run through a process with many expensive analysis steps (Sarawagi 2008).

Other techniques address this process, ranging from *efficient algorithms* over an optimization through *scheduling* to *indexing* and *parallelization*.

**Efficient Algorithms.** Efficient algorithms have been developed for several text analyses. For instance, Al-Rfou' and Skiena (2012) present how to apply simple heuristics and caching mechanisms in order to increase the velocity of segmentation and tagging (cf. Sect. 2.1). Complex syntactic analyses like dependency parsing can be approached in linear time by processing input texts from left to right only (Nivre 2003). Bohnet and Kuhn (2012) show how to integrate the knowledge of deeper analyses in such transition-based parsing while still achieving only quadratic complexity in the worst case. van Noord (2009) trades parsing efficiency for effectiveness by learning a heuristic filtering of useful parses. For entity recognition, Ratnov and Roth (2009) demonstrate that a greedy search (Russell and Norvig 2009) can compete with a more exact sequence labeling (cf. Sect. 2.1). Others offers evidence that simple patterns based on words and part-of-speech tags suffice for relation extraction, when given enough data (Pantel et al. 2004). In text classification tasks like genre identification, efficiently computable features are best practice (Stein et al. 2010). Also, the feature computation itself can be sped up through unicode conversion and string hash computations (Forman and Kirshenbaum 2008).

All these approaches aim to improve the efficiency of *single* text analyses, mostly at the cost of some effectiveness. We do not compete with these approaches but rather complement them, since we investigate how to improve pipelines that realize *complete* processes consisting of different text analyses. In particular, we optimize the efficiency of pipelines without compromising effectiveness through scheduling.

**Scheduling.** Some approaches related to text mining optimally schedule different algorithms for the same analysis. For instance, Stoyanov and Eisner (2012) effectively resolve coreferences by beginning with the easy cases, and Hagen et al. (2011) efficiently detect sessions of search queries with the same information need by beginning with the fastest detection steps. The ordering in which information is sought for can also have a big influence on the run-time of text analysis (Sarawagi 2008). In Chap. 4, we seize on this idea where we optimize the schedules of pipelines that filter only relevant portions of texts. However, the optimal schedule is input-dependent, as has been analyzed by Wang et al. (2011) for rule-based information extraction. Similar to the authors, we process samples of input texts in order to estimate the efficiency of different schedules.

In this regard, our research is in line with approaches in the context of the above-mentioned SYSTEMT. Concretely, Shen et al. (2007) and Doan et al. (2009) exploit dependencies and distances between relevant text regions to optimize the schedules of declarative information extraction approaches, yielding efficiency gains of about one order of magnitude. Others obtain comparable results through optimization strategies such as the integration of analysis steps (Reiss et al. 2008).

In these works, the authors provide only heuristic hints on the reasons behind their empirical results. While some algebraic foundations of SYSTEMT are established in Chiticariu et al. (2010a), these foundations again reveal the limitation of declarative information extraction, i.e., its restriction to rule-based text analysis. In contrast, we

approach scheduling for arbitrary sets of text analysis algorithms. While we achieve similar gains as SYSTEMT through an optimized scheduling, our adaptive scheduling approach is, to our knowledge, the first that maintains efficiency on heterogeneous input texts. In addition, we show that the theoretically optimal schedule can be found with dynamic programming (Cormen et al. 2009) based on the run-times and filtered portions of text of the employed algorithms.

In the database community, dynamic programming is used since many years to optimize the efficiency of *join* operations (Selinger et al. 1979). However, the problem of filtering relevant portions of text for an information need corresponds to processing *and*-conditioned queries (cf. Sect. 2.2). Such queries select those tuples of a database table whose values fulfill some attribute conjunction, as e.g. in `SELECT * FROM forecasts WHERE (time>2011 AND time<2015 AND organization=IBM)`. Different from text analysis, the optimal schedule for an *and*-conditioned query is obtained by ordering the involved attribute tests (e.g. `time>2011`) according to the numbers of expected matches (Ioannidis 1997), i.e., without having to consider algorithm run-times.

**Indexing.** An alternative to optimizing the efficiency of text analysis is to largely avoid the need for efficient analyses by indexing relevant information for each input text beforehand (cf. Sect. 2.1). For instance, Cafarella et al. (2005) have presented the KNOWITNOW system, which builds specialized index structures using the output of information extraction algorithms. Their approach has then been adopted in the open information extraction systems discussed above. Also, the GOOGLE KNOWLEDGE GRAPH is operationalized in an index-like manner as far as known.<sup>20</sup>

In the best case, indexing renders text analysis unnecessary when addressing information needs (Agichtein 2005). In the database analogy from above, the run-times of the tests (that correspond to the text analysis algorithms) drop out then. By that, indexing is particularly helpful in scenarios like ad-hoc search. However, it naturally applies only to anticipated information needs and to input texts that can be preprocessed beforehand. Both cannot be assumed in the tasks that we consider in this book (cf. Sect. 1.2).

**Parallelization.** With the goal of efficiency finally arises the topic of parallelization. As discussed, we concentrate on typical text analysis algorithms and pipelines, which operate over each input text independently, making many parallelization techniques easily applicable (Agichtein 2005). This might be the reason for the limited literature on parallel text analysis, despite the importance of parallelization for practical text mining applications. Here, we focus on process-related approaches as opposed to distributed memory management (Dunlavy et al. 2010) or algorithm schemes like MAPREDUCE for text analysis (Lués and de Matos 2009).<sup>21</sup>

<sup>20</sup>GOOGLE KNOWLEDGE GRAPH, <http://googleblog.blogspot.co.uk/2012/05/introducing-knowledge-graph-things-not.html>, accessed on June 15, 2015.

<sup>21</sup>Accordingly, we omit to talk about infrastructural technologies for distributed computing, such as APACHE HADOOP, <http://hadoop.apache.org>, accessed on June 15, 2015.



Text analysis can be parallelized on various levels: Different algorithms may run distributed, both to increase the load of pipelines (Ramamoorthy and Li 1977) and to parallelize independent analyses. Pokkunuri et al. (2011) run different pipelines at the same time, and Dill et al. (2003) report on the parallelization of different algorithms. The two latter do not allow interactions between the parallelized steps, while others also consider synchronization (Egner et al. 2007). A deep analysis of parallel scheduling strategies was performed by Zhang (2010). Apart from these, different texts can be processed in parallel (Gruhl et al. 2004), and the execution of analysis steps like parsing is commonly parallelized for different portions of text (Bohnet 2010). Kalyanpur et al. (2011) even run different pipelines on the same text in parallel in order to provide results as fast as possible in ad-hoc question answering.

At the end of Chap. 4, we see that input-based parallelization is always applicable to the pipelines that we employ. The same holds for the majority of other approaches, as discussed there. Because of filtering, synchronization entails new challenges with respect to our approaches, though.

#### 2.4.4 Robustness of Text Analysis Approaches

In Chap. 5, we seek to improve the domain robustness of text analysis in order to produce high-quality information in applications where the domain of input texts cannot be anticipated, like ad-hoc web search. Most text analysis approaches at least partly rely on features of texts that are specific to a domain of application (Blitzer et al. 2007) and, hence, significantly drop in effectiveness when being applied in a new domain. Early work in this context often aimed to reduce the cost of adapting to new domains by exploiting machine learning techniques for obtaining training data automatically, surveyed by Turmo et al. (2006). However, the predominant approach today is to tackle domain dependence through *domain adaptation* (Daumé and Marcu 2006), as explained below. Some approaches also strive for *domain independence* based on generally valid features. From these approaches, we adopt the idea of focusing on structure, especially on the *argumentation structure* of texts, which in turn relates to *information structure* and *discourse structure*. Since robustness does not mean perfect effectiveness, we end with existing work on how to increase the *user acceptance* of erroneous results.

**Domain Adaptation.** The scenario usually addressed in domain adaptation is that many training texts are given from some source domain, but only few from a target domain (Blitzer et al. 2008). The goal is to learn a model on the source texts that works well on unknown target texts. In information extraction, most domain adaptation approaches share that they choose a representation of the source texts that makes them close the distribution of the target texts (Gupta and Sarawagi 2009). Similarly, domain adaptation is often tackled in text classification by separating the domain-specific from the domain-independent features and then exploiting knowledge about



the latter (Daumé and Marcu 2006). Also, structural correspondences can be learned between domains (Blitzer et al. 2007; Prettenhofer and Stein 2011). In particular, domain-specific features are aligned based on a few domain-independent features, e.g. “*Stay away!*” in the hotel domain might have a similar meaning as “*Read the book!*” in the film domain.

Domain adaptation, however, does not really apply to ad-hoc search and similar applications, where it is not possible to access texts from all target domains in advance. This also excludes the approach of Gupta and Sarawagi (2009) who derive domain-independent features from a comparison of the set of all unknown target texts to the set of known source texts.

**Domain Independence.** Since domains are often characterized by content words and the like (cf. Chap. 5 for details), most approaches that explicitly aim for domain independence try to abstract from content. Glorot et al. (2011), for instance, argue that higher-level intermediate concepts obtained through the non-linear input transformations of deep learning help in cross-domain sentiment analysis of reviews. While we evaluate domain robustness for the same task, we do not presume a certain type of machine learning algorithms. Rather, we work on the features to be learned. Lipka (2013) observes that style features like character trigrams serve the robustness of text quality assessment. Similar results are reported for authorship attribution in Sapkota et al. (2014). The authors reveal the benefit of mixed-domain training sets for developing robust text analysis algorithms.

Some experiments that we perform in Chap. 5 suggest that style features are still limited in their generalizability. We therefore propose features that model the structure of texts. This resembles the idea of open information extraction, which avoids the resort to any domain-dependent features, but captures only generally valid syntactic patterns in sentences (see above). However, we seek for domain independence in tasks, where complete texts have to be classified. For authorship attribution, Choi (2011) provide evidence that structure-based features like function word  $n$ -grams achieve high effectiveness across domains. We go one step further by investigating the argumentation structure of texts.

**Argumentation Structure.** Argumentation is studied in various disciplines, such as logic, philosophy, and artificial intelligence. We consider it from the linguistics perspective, where it is pragmatically viewed as a regulated sequence of speech or text (Walton and Godden 2006). The purpose of argumentation is to provide persuasive arguments for or against a decision or claim, where each argument itself can be seen as a claim with some evidence. Following the pioneer model of Toulmin (1958), the structure of an argumentation relates a claim to facts and warrants that are justified by backings or countered by rebuttals. Most work in the emerging research area of *argumentation mining* relies on this or similar models of argumentation (Habernal et al. 2014). Concretely, argumentation mining analyzes natural language texts in order to detect different types of arguments as well as their interactions (Mochales and Moens 2011).

Within our approach to robustness, we focus on texts that comprise a monological and positional argumentation, like reviews, essays, or scientific articles. In such a

text, a single author collates and structures a choice of facts, pros, and cons in order to persuade the intended recipients about his or her conclusion (Besnard and Hunter 2008). Unlike *argumentative zoning* (Teufel et al. 2009), which classifies segments of scientific articles according to their argumentative functions, we aim to find argumentation patterns in these texts that help to solve text classification tasks. For this purpose, we develop a shallow model of argumentation structure in Chap. 5.

**Information Structure.** Our model captures sequences of task-specific information in the units of a text as well as relations between them. By that, it is connected to information structure, which refers to the way information is packaged in a text (Lambrecht 1994). Different from approaches like (Bohnet et al. 2013), however, we do not analyze the abstract information structure within sentences. Rather, we look for patterns of how information is composed in whole texts (Gylling 2013). In sentiment-related tasks, for instance, we claim that the sequence of subjectivities and polarities in the facts and opinions of a text represents the argumentation of the text. While Mao and Lebanon (2007) have already investigated such sequences, they have analyzed the positions in the sequences only separately (cf. Chap. 5 for details). In contrast, we develop an approach that learns patterns in the complete sequences found in texts, thereby capturing the overall structure of the texts. To the best of our knowledge, no text analysis approach to capture overall structure has been published before.

**Discourse Structure.** The information structure that we consider is based on the discourse structure of a text (Gylling 2013). Discourse structure refers to organizational and functional relations between the different parts of a text (Mann and Thompson 1988), as presented in Chap. 5. There, we reveal that patterns also exist in the sequences of discourse relations that e.g. cooccur with certain sentiment. Gurevych (2014b) highlight the close connection between discourse structure and argumentation, while Ó Séaghdha and Teufel (2014) point out the topic independence of discourse structure. The benefit of discourse structure for sentiment analysis, especially in combination with opinion polarities, has been indicated in recent publications (Villalba and Saint-Dizier 2012; Chenlo et al. 2014). We use according features as baselines in our domain robustness experiments.

**User Acceptance.** Even a robust text mining application will output erroneous results occasionally. If users do not understand the reasons behind, their acceptance of such an application may be limited (Lim and Dey 2009). While in some technologies related to text mining much attention is paid to the transparency of results, like recommender systems (Sinha and Swearingen 2002), according research for text analysis is limited. We consider the explanation of text classification, which traditionally outputs only a class label, possibly extended by some probability estimate (Manning et al. 2008). Alvarez and Martin (2009) present an explanation approach to general supervised classification that puts the decision boundary in the focus (cf. Sect. 2.1). Kulesza et al. (2011) visualize the internal logic of a text classifier, and Gabrilovich and Markovitch (2007) stress the understandability of features that correspond

to real-world concepts. At the end of Chap. 5, we sketch explanation approaches that follow the intuitions of the two latter using knowledge about the employed pipelines and information about the developed features. We believe that the user acceptance of erroneous results is decisive for the success of ad-hoc large-scale text mining applications.

Text Analysis Pipelines

Towards Ad-hoc Large-Scale Text Mining

Wachsmuth, H.

2015, XX, 302 p. 74 illus. in color., Softcover

ISBN: 978-3-319-25740-2