Semantic Lexicon Expansion using Bootstrapping and Syntax-based, Contextual Extraction Patterns

Casper Kuijjer
9810579

August 21, 2007

Abstract

We describe the task of Semantic Lexicon Expansion, in which a small seed lexicon is expanded into a semantic lexicon that encompasses all terms in a given document-set. For this task we introduce a bootstrapping algorithm, that uses simple context-based extraction patterns to iteratively expand a seed lexicon. This algorithm is compared to other bootstrapping algorithms and its implementation is evaluated on a document-set of movie reviews. In this evaluation we pay special attention to finding the minimum seed lexicon and document-set size needed to start iterating.

Acknowledgements

First of all I would like to thank my supervisors, Maarten van Someren and Victor de Boer, for their valuable input and their tremendous effort in helping me finish this thesis. Without their help and guidance I would not have been able to complete this on time.

Thanks to the people at Textkernel, especially to Jakub and Remko, for introducing me to the field of Information Extraction and to my colleagues at Exact Software for giving me a quiet workplace, and keeping me focused while I was running experiments and writing my thesis.

I would also like to thank Wouter Driessen, for proofreading earlier versions of this thesis, and pointing out most of the glaring grammatical mistakes I made. And last but not least, Esther Andeweg, for being very supportive and never showing any sign of frustration during all those evenings I spent writing.

Contents

Chapter 1

Introduction

In the field of Information Extraction techniques are developed to automatically retrieve structured information from unstructured documents. Machine Learning is often used to create extraction models from a set of example documents. Such an extraction model, for example a set of extraction patterns or a Hidden Markov Model, gives computer programs the ability to process unstructured documents. For example this ability can then be used to provide structured search, e.g. a job search website using a back-end that is filled with user uploaded resumes in Microsoft Word format, or to create a database from different data sources, e.g. by scraping websites of local real-estate agencies to create a national database of the housing market.

In both these examples Information Extraction is used to perform extraction on the document level: the extraction model is used to retrieve information from single documents. For other tasks the extraction can be performed on the corpus level. The resumes contained in a job search website can be used to create reports on the amount of education the average person has obtained. Another task might be to create a dictionary of domain specific terms, a semantic lexicon. Such a semantic lexicon, e.g. one containing instances of the semantic category *Weapon* (see Riloff [1996]), could then for example be used to show the most commonly used weapons in a set of newspapers, by looking at how often each of the terms in the semantic lexicon occur in them.

The techniques used to create extraction models for tasks that work on the document level might not be optimal for tasks on the corpus level. An extraction model used to make a semantic lexicon does not for example need to make all possible extractions in each of the documents, it only needs to extract each lexical entry once. For tasks on the corpus-level we can use the fact that there is a lot of redundant information in a large enough document-set, i.e. the same terms occurs in multiple locations, in different contexts. The redundancy allows for reuse of extractions, since information about the other contexts in which the extraction occurs can be used in the extraction model. This use of redundancy in the document-set is used in our bootstrapping algorithm.

1.1 Focus of the Research

We are interested in Information Extraction tasks at the corpus-level, and want to show that it is possible to create algorithms for these tasks that work on unannotated documents. With our research we design an algorithm for Semantic Lexicon Expansion. This algorithm uses a bootstrapping approach, and extraction patterns that make only use of the syntax in a small context window. Our research focuses on finding out how the seed lexicon and the document-set on which this algorithm is applied influence the resulting expanded seed lexicon. This question is also answered for the scoring function, and other parameters of our algorithm.

1.2 Organisation of the Thesis

In the remaining parts of this chapter we describe the differences in approaches to Information Extraction on the document and on the corpus-level. We also describe some other algorithms that use bootstrapping. Chapter 2 covers our bootstrapping method and algorithm, and ends with a small example. In Chapter 3 we apply our bootstrapping algorithm to a set of movie reviews. Chapter 4 extends our algorithm to make use of lexical- and semantic categories to create more general extraction patterns. Chapter 5 gives conclusions and pointers on further research.

1.3 Information Extraction

Most Information Extraction systems are document-based, they are designed for tasks that need to extract all relevant pieces of information from each single document that is given to them. Information Extraction systems can be created manually, e.g. by creating hand written rules, or by Machine Learning. The machine learning approach to create a document-based system is to collect a set of documents from the domain for which the system is being build, to annotate them and then to use machine learning on the annotated document-set to create an extraction model. It is important that the annotation is done as completely and correctly as possible since the learning algorithm uses the annotation layer to create an extraction model.

For example, to create an extraction model that can be used to automatically insert resumes into a database, we would collect a set of resumes, annotate every piece of information to be extracted in those resumes and then apply the machine learning algorithm.

A machine learning algorithm, such as Hidden Markov Models, Conditional Random Fields or Wrapper Induction (see Kushmerick and Thomas [2003] for an overview) is then applied to the annotated training set to create an extraction model. This extraction model is designed to correctly extract as many annotated tokens in the training set as possible (in order to obtain a high recall) and to make as few other extractions as possible (in order to keep precision high). The annotation layer is assumed to be complete, so that non annotated tokens, or tokens that are annotated as belonging to another semantic category can be seen as negative examples.

There are some problems with this approach, and it might not be the best approach for tasks at the corpus-level. One problem is that it needs large amounts of annotated training documents in order to learn an extraction model. Annotating these training documents has to be done manually, and is thus time consuming and costly.

Two techniques techniques that are used to make the annotation task easier are active learning and semi-supervised learning. When active learning techniques are used, the extraction model is learned during the annotation phase instead of after it. The extraction model is applied to the unannotated documents, and the document for which extraction model is the least certain of its extractions is presented to the user for annotation. This enables the machine learning algorithm to use less annotated documents to come to an extraction model of the same quality, since it keeps documents that do not contain much information for the machine learning algorithm from being annotated.

Semi-supervised learning combines the set of annotated documents with a larger set of unannotated documents to come to an extraction model. For example co-training (see Blum and Mitchell [1998]) uses two machine learning algorithms, that use a different view on the annotated document-set, to create two document classifiers. Each document classifier is then used to classify a few of the unannotated documents, after which these, now annotated, documents are appended to the training-set of the other classifier. This is repeated for a few iterations, allows the resulting classifiers to be based on more annotated documents, and improves their quality.

Another problem is that annotation is an error-prone process. When learning an extraction model, the annotated portions of the text are seen as positive examples, the rest of the text as negative examples. Therefore incorrect or missed annotations have an impact on the quality of the extraction model. A lot of semantic categories are quite simple to annotate (for example annotating names or addresses), but other categories might need domain experts (for example to annotate computer skills in resumes of a computer scientist, one might need to know that J2EE is about the programming language Java and a Cisco Certification is about Networking). Domain experts are hard to come by.

Although the approach of annotating a document-set and training an extraction model works for information extraction from single documents, i.e. for information extraction tasks at the document-level, there are other information extraction tasks for which it is either too simple or too complex. For example the approach is too simple to extract information from several linked documents (e.g. to extract a list of computer scientists, their publications and contact information from a document-set consisting of homepages and some bibliography databases). For information extraction tasks at the corpus-level, such as semantic lexicon expansion and ontology population the approach may be too complex. For example to create a semantic lexicon it is not necessary to find every occurrence of every possible term in the document-set, it is only necessary to find enough occurrences of each term to be able to determine that the term is an element of the lexicon, i.e. it is not necessary to make every correct extraction.

1.4 Semantic Lexicon Expansion

Semantic categories are used to group certain terms, consisting of one or more words, together. The terms that are grouped together refer to instances of a certain class, for example a semantic category for the class *Weapon* could be used to describe terms like *M-60, Knife, AK-47, Pistol* and perhaps the less prototypical *Bare Hands*. A semantic lexicon is a dictionary of terms, labeled with information about the semantic categories to which each term belongs to. A small example semantic lexicon is displayed in Table 1.1.

Term	**Semantic Category**
Car Bomb	Weapon
AK-47	Weapon
Suicide Bomb	Weapon
IED	Weapon
Kalashnikov	Weapon
Ansar al-Islam	Terrorist Organization
Jama'at al-Tawhid wa'al-Jihad	Terrorist Organization
Jaish Ansar al-Sunna	Terrorist Organization

Table 1.1: Example of a Semantic Lexicon

Multiple terms in a semantic category can refer to the same instance. For example *AK-47* and *Kalashnikov* are two different terms that refer to the same sort of weapon. It is also possible for the same term to be used in several semantic categories, terms are allowed to be ambiguous. For example the term *Batman* might be used as to refer to the instance batman the movie, the character batman and the role to play batman, and therefore belongs to three different semantic categories, *Movie Title*, *Super Hero* and *Role* respectively.

Semantic Lexicons can be used in different ways. For example, to facilitate Information Retrieval by expanding a search query with related terms, or by giving an overview of the terms used in a domain. Automatically extracting a semantic lexicon is useful for creating domain dependent semantic lexicons, and is an Information Extraction task at the corpus-level.

Semantic Lexicon Expansion techniques try to create an expanded semantic lexicon using a small example seed lexicon and a document-set. The expanded semantic lexicon should encompass all terms, for all the semantic categories defined in the seed lexicon, that are used in the document-set. Semantic Lexicon Expansion is a subtask of Information Extraction that works on the corpus-level. An algorithm used for semantic lexicon expansion does not need to be able to extract every term from the expanded semantic lexicon used in a single document. The algorithm only needs to be able to extract each term in enough documents to be able to determine that it exists in the expanded semantic lexicon. It should however be careful not to expand a category with ambiguous terms that are only used in the document-set to refer to instances of other categories. For example, the category *Super Hero* from the example above should not be expanded with the term *Batman* if *Batman* is only used to refer to movie titles.

A task closely related to Semantic Lexicon Expansion is Ontology Popula-

tion. Ontology Population techniques are used to find the instances belonging to a certain class, or in its subtask Relation Instantiation to instantiate relations between classes.

1.5 Bootstrapping Information Extraction

Bootstrapping is a form of semi-supervised learning. It uses a small list of positive examples, called the seed lexicon, and a set of unannotated documents and tries to find an expansion, the expanded seed lexicon, for this list. Bootstrapping uses multiple iterations to gradually find all terms of the expanded semantic lexicon that are used in the document-set. To do this in each iteration the algorithm seeks a small and reliable expansion of the seed lexicon, the seed expansion, allowing the algorithm to use more positive examples in later iterations.

An iteration starts with finding all the occurrences of the terms in the seed lexicon in the unannotated document-set. These occurrences are then used to create a set of extraction patterns. By applying the set of extraction patterns to the document-set, extractions are found. An iteration ends with selecting a few of the best extractions as the seed expansion to be used in the next iteration.

A few other algorithms that use bootstrapping techniques to perform information extraction exist. Our algorithm is mainly modeled on techniques described in Multi-level bootstrapping and Basilisk. We describe two other algorithms, that use bootstrapping for Relation Instantiation instead of Semantic Lexicons Expansion. While describing these bootstrapping algorithms we will choose what parts are to be used in our algorithm, and for which parts we have to devise our own methods.

1.5.1 Multi-level Bootstrapping

Multi-level bootstrapping (see Riloff and Jones [1999]) uses a set of unannotated training documents and a seed lexicon to learn both a set of extraction patterns and an expanded seed lexicon. The use of bootstrapping is based on the observation that extraction patterns can lead to new extractions, which in turn can lead to new extraction patterns.

Before bootstrapping begins the algorithm exhaustively generates extraction patterns for all the noun phrases in the document-set. This is done using AutoSlog (see Riloff [1993]), an algorithm that uses CIRCUS, a sentence analyser, and 15 heuristic rules to create extraction patterns. The resulting extraction patterns are then stored in combination with their extractions. See Table 1.2 for a few of these heuristics and example extraction patterns.

Heuristic	**Example Extraction Pattern**
<subj>passive-verb	`<EXTRACTION> was shown`
verb infin. <dobj>	`tried to direct <EXTRACTION>`
active-verb prep <np>	`starred alongside <EXTRACTION>`

Table 1.2: Some of the heuristics used by AutoSlog plus example extraction patterns

Extractions are anchored by either a specific left or a specific right context. This can be done because there exists a restriction on the extraction field: an extraction should be recognized as a noun phrase by CIRCUS, and gives the advantage of creating better patterns near the beginning and end of sentences. We want our algorithm to be independent from sentence analysis, and strict requirements on the extraction field, so our algorithm has to require both a specific left and right context.

The advantage of creating all extraction patters in advance is that it can be done offline, which keeps the main part, the bootstrapping loops, of the algorithm fast. An disadvantage is that extractions are also made for extraction patterns that can be known not to be helpful. As we do not want our extraction patterns to be limited to patterns found by a tool such as CIRCUS, the number of possible extraction patterns make it prohibitive to create all extraction patterns in advance. Our algorithm will therefore create its extraction patterns during the bootstrapping loop.

Multi-level bootstrapping uses two bootstrapping loops. An inner bootstrapping loop, called mutual bootstrapping, in which a set of extraction patterns and the seed lexicon are expanded. And an outer bootstrapping loop, called meta bootstrapping, that restarts the inner bootstrapping loop to improve the quality of the set of extractions.

The inner bootstrapping loop starts with a seed lexicon, and an empty set of extraction patterns. In each iteration the best extraction pattern is sought and added to the set of extraction patterns, and all its iterations are used as a seed expansion for the next iteration. Each extraction is stripped of leading articles, common adjectives and numbers and a stop list is used to discard overly general extractions.

To find the best extraction pattern, all patterns are scored according to the $RlogF$ metric

$$RlogF = R_i * log_2(F_i)$$

with $R_i = \frac{F_i}{N_i}$, F_i the number of different terms in the seed lexicon that pattern i extracts, and N_i the total number of different terms that pattern i extracts.

$$PatternScore(P_i) = \frac{F_i}{N_i} log_2(F_i)$$

The $RlogF$ metric is designed to find a balance between being reliable, R is larger if a larger portion of its extractions are in the seed lexicon, and being general, F is larger for extraction patterns that extract more terms in the seed lexicon.

The inner bootstrapping loop is normally ended after 10 iterations, i.e. after 10 extraction patterns have been added to the set of extraction patterns. Two thresholds on the pattern score exist that make it possible to stop before, or continue after iteration 10.

The inner bootstrapping loop selects as the seed expansion all the extractions of the best extraction pattern. This assumes that all the extractions of that extraction pattern are correct. A meta-bootstrapping loop of 50 iterations is used to correct this often violated assumption by restarting the inner bootstrapping loop using only the most reliable lexicon expansions. The best five extractions, found during the inner bootstrapping loop, are added to the permanent seed

lexicon that is then used as input to restart the inner bootstrapping loop. To select the best five extractions the extractions are scored according to:

$$ExtractionScore(E_i) = \sum_{k=1}^{M_i} 1 + (0.01 * PatternScore(P_k))$$

with M_i the subset of the set of extraction patterns found in the inner bootstrapping loop that extracts E_i. This scoring function gives a higher score to extractions if they are made by more extraction patterns, the Pattern Score is used for tie-breaking.

In our algorithm we only want to focus on finding the best set of extractions, finding a set of high quality extraction patterns is of secondary concern. This allows us to use all extraction patterns to select the best extraction. To be able to do so a different extraction scoring function is needed, simply being based on the number of extraction patterns that make the extraction is not enough. Taking the quality of the extraction patterns into consideration in the extraction scoring function is more important in our algorithm, since we also want to use extraction patterns of lesser quality.

1.5.2 Basilisk

A continuation of the Multi-level bootstrapping algorithm is found in Basilisk and described in Thelen and Riloff [2002] and Thelen [2002]. Multi-level bootstrapping focusses on the quality of extraction patterns, all extractions of the best pattern are used as the seed expansion for the next iteration, while Basilisk focusses on the quality of extractions. Basilisk uses collective evidence over a large set of extraction patterns instead of one extraction pattern to determine the quality of an extraction. This removes the need of the correcting meta-bootstrapping loop used by Multi-level bootstrapping.

In each iteration all extraction patterns are scored using the *RlogF* metric. The best N extraction patterns, that are not yet depleted, are put into the pattern pool. N has an initial value of 20 and is incremented after each iteration to renew the pattern pool. The extraction patterns in the pattern pool are used to create extractions. Instead of using the entire noun phrase Basilisk collects the head noun (e.g. *film* in the noun phrase *My favourite film*) of each extraction into the candidate word pool. The terms in the candidate word pool are scored according to:

$$AvgLog(term_i) = \frac{\sum_{j=1}^{P_i} log_2(F_j + 1)}{P_i}$$

All extraction patterns, not only those in the pattern pool, are used to calculcate the extraction score. P_i is the number of extraction patterns able to extract term i, and F_j is the number of different terms in the seed lexicon that pattern j extracts. The best five terms are used as the seed expansion.

Basilisk also extends Multi-level bootstrapping with the ability to use information about other semantic categories in the semantic lexicon while looking for an expansion of a semantic category. Basilisk assumes that an extraction can only belong to a single semantic category, and keeps a semantic category from expanding into other semantic categories. Two methods for doing this are described:

- Simple Conflict Resolution discards extractions found for a category which already exist in the expanded seed lexicon of another category. If in the same iteration an extraction is found for two or more categories, it is discarded from the category for which it scored the least.

- Adapting the extraction scoring function to be based on all semantic categories instead of on only the semantic category for which an seed expansion is sought. The extraction scoring function is changed to:

$$diff(w_i, c_a) = AvgLog(w_i, c_a) - max_{b \neq a}(AvgLog(w_i, c_b))$$

  This makes an extraction being given a high score if there is a lot of evidence relating it to the semantic category that is to be expanded, and there is almost no evidence relating it to the other semantic categories in the semantic lexicon.

We find the assumption of a term belonging to a single semantic category too strict to be used in our algorithm. For example, the semantic category *Actor Name* has a considerable overlap with the semantic category *Director*.

1.5.3 Relation Instantiation Based Methods

Learning an ontology consists of two parts, learning a data model and learning a knowledge base. Ontology Population is used to fill the knowledge base for a given data model. This task consists of finding instances for the concepts described in the data model, and relating these instances using the relations described in the data model. This latter task, a subtask of Ontology Population called Relation Instantiation, tries to find, for a certain relation between two classes, instance-pairs for which the relation holds. A Relation Instantiation algorithm could for example be designed to find instances of the relation *directed by* by finding relations in a semantic lexicon containing *Movie Title*'s and *Director*'s.

In Relation Instantiation knowledge about the relation (e.g. is it a one-to-many relation?, how many relation instances are there?) can be used when finding new relations. For example if for a many-to-one relation such as *directed by* a possible *movie, director*-pair is considered it gives an enormous amount of confidence if another *movie, director* instance is already known for that particular director. On the other hand if another *movie, director*-pair was already known for a particular movie knowing that it is a many-to-one relation can be used to reject the relation instance under evaluation.

To expand a few example relation instances in an ontology alignment setting, where all the instances of subject- and object class of the relation are known in advance, Geleijnse and Korst [2006] uses the excerpts obtained by querying Google to come up with a set of patterns. Google is then used to search for combinations of a pattern and a subject instance. If an object instance is found in the excerpts, a relation instance is found. For tasks where not all instances are known in advance, Hearst patterns are used to check if a term in the excerpt occurs as an instance of the object class in the document-set.

In de Boer et al. [2007] Google is used to find a document-set by querying on an instance of the subject class. By applying an extraction method for

the instances of the object class, the algorithm finds which instances of the object class occur in each document, after which the documents are given a *DocumentScore*. This score is based on a few example relation instantiations and the assumption that the relation is either well represented in a document or not at all. For each of the candidate object instances a *InstanceScore* is calculated that is based on the collective evidence over all documents. To select which relation instances should be used de Boer et al. [2007] describe using a threshold on the *InstanceScore* and the use of a bootstrapping loop. The bootstrapping loop selects the single best relation instance, adds this to the set of example relation instantiations and recalculates the *DocumentScore* and the *InstanceScore*. Combined with a *DropFactor* that stops the bootstrapping loop if the *InstanceScore* of the relation instance under consideration is too low, bootstrapping is shown to eliminate the need for a domain- and task-specific threshold.

Two approaches that can be used in settings where not all instances of the subject- and object class of the relation are known in advance are described in more detail below. These algorithms both use a bootstrapping loop, and a set of context based extraction patterns to expand a set of relation instances.

1.5.3.1 DIPRE

DIPRE (see Brin [1998]) expands a seed set of *author, book title*-pairs by applying a bootstrapping algorithm to the World Wide Web. The algorithm works directly with HTML pages, no tokenization or other analysis is used.

Finding the set of occurrences is done by searching the entire World Wide Web for documents that contain an element of the seed set. An occurrence is found if both the *author* and the *book title* of a seed element occur in the same sentence. In addition to the context to the left, right and in the middle of the *author* and *book title*, the url of the document is also stored.

Patterns are created by clustering the occurrences. First all occurrences with the middle context and order of *author* and *book title* are grouped in a cluster. For each of those clusters an extraction pattern is created that consists of the longest common url-prefix, and the longest common context to the left and the right. If an extraction pattern does not meet a certain amount of specificity (a requirement based on the length of the pattern and a minimum seed support of two), the url-prefix is used to cluster its occurrences into smaller clusters, for which new extraction patterns are created. This continues until all clusters of occurrences lead to an extraction pattern that meets the minimum specificity requirement.

URL Pattern	Text Pattern
www.sff.net/locus/c.*	`<LI><B>`*title*`</B> by` *author* `(`
dns.city-net.com/~lmann/awards/hugos/1984.html	`<i>`*title*`</i> by` *author* `(`
dolphin.upenn.edu/~dcummins/texts/sf-award.htm	*author* `||` *title* `|| (`

Table 1.3: Extraction Patterns found by DIPRE, as shown in Brin [1998]

DIPRE creates highly website-specific extraction patterns (see Table 1.3 for a few examples), the url-prefix makes the pattern only apply to certain web-

pages. Web-pages that have the same url-prefix, are likely to have the same layout, e.g. the web-pages on a web-site are often generated from one template. This fact is used by DIPRE to make extraction patterns for specific layouts, and to apply extraction patterns only to those documents that use a certain layout. In Section 3.1 we show that each movie review, in our document-set, that is written by the same reviewer often uses the same layout. Unfortunately there is no feature, such as the url of a web-page, in our set of movie reviews that can be used to see which layout is used in a document. This possibly forces our algorithm to create less specific extraction patterns than DIPRE, which might have a negative effect on the precision of the algorithm.

DIPRE does not evaluate the quality of the extraction patterns, it uses all extractions of all extraction patterns as the seed expansion for the next iteration.

1.5.3.2 Snowball

Agichtein and Gravano [2000] describes Snowball, an algorithm based on the bootstrapping method used by DIPRE (see Brin [1998]). The algorithm is applied to a corpus of news paper articles to expand a seed set of *Organization, Headquarter*-pairs. Snowball expands the general method described in DIPRE with pattern and extraction scoring functions.

Snowball applies a named-entity tagger, that is able to tag organizations, locations and persons, to the document-set. An occurrence is found if in a sentence both the *Organization* and *Headquarter* of a seed element are found and the named-entity tagger tagged them accordingly.

The set of patterns is created by applying a clustering algorithm to the set of occurrences. A pattern in Snowball consists of two named-entity tags (one organization-tag and one location-tag) and a token weight vector for the left, middle and right context. All patterns with seed support less than τ_{sup} are discarded.

The sentences in which both a organization and a location tag occur are then matched to the set of patterns. An extraction is found if there exists a minimum similarity τ_{sim} between the sentence and at least one of the patterns.

Extractions are scored using a confidence measure that is based on the confidence of-, and similarity to each of the patterns. To calculate the pattern confidence the fact that the *organization, headquarter*-relation is a many-to-one relation is used. If the pattern extracts a *organization, location*-pair for an *organization* that was extracted in a previous iteration, the extraction is seen as a positive one if the *location* is equal to the *location* in the previous iteration. Otherwise it is seen as a negative extraction. The confidence of a pattern is then calculated by:

$$Conf(P) = \left(\frac{P}{P+N} log_2(P) \right) \cdot W_{updt} + Conf_{old}(P) \cdot (1 - W_{updt})$$

with P the number of positive extractions, N the number of negative extractions, $Conf_{old}(P)$ the confidence of the pattern in the previous iteration and W_{updt} as a parameter to control the learning rate.

All extractions with confidence above the user-definable threshold τ_t are used as the seed expansion.

We will not use a threshold such as τ_t in our algorithm to determine which extractions are to be used in the seed expansion. We will use a constant number of extractions as we are more interested in the behaviour of our algorithm, than we are in finding an optimum amount of extractions to add in each iteration.

Chapter 2

Algorithm & Implementation

We have created an algorithm that can be used for semantic lexicon expansion. The algorithm is capable of expanding semantic lexicons that consist of terms belonging to a single semantic category. If the semantic lexicon consists of terms belonging to multiple semantic categories, the semantic lexicon can be split into subsets of terms belonging to a single semantic category after which our algorithm is applied to each of the subsets.

The input to the algorithm is a small seed lexicon and a large set of unannotated documents. The algorithm has a few user definable parameters that can be used to fine-tune the algorithm. They are described in Section 2.2.

A seed lexicon is used to initiate a bootstrapping loop. The more often the terms in the seed lexicon occur in the document-set, the more likely it becomes that the algorithm finds a good set of extraction patterns and is able to make a large amount of iterations. A larger or more prototypical seed lexicon, or a larger document-set is therefore better than a smaller one. The use of bootstrapping allows us to simplify our semantic lexicon expansion task from Section 1.4. Instead of finding the entire set of correct expansions at once, we have to find at least one expansion in each iteration. If in each iteration the algorithm expands the seed lexicon with terms belonging to the semantic category, and if the algorithm is able to keep iterating, after a number of iterations it will have found the entire set of correct expansions. Another advantage of gradually expanding the semantic lexicon, is that we use the expansion of the previous iteration to find new patterns and extractions. This enables the algorithm to find more extractions, and thus a larger expansion, than is possible if it would try to find the expansion at once.

Our bootstrapping algorithm uses the same steps as the algorithms in Section 1.5, an overview of these steps is given in Figure 2.1. Each iteration tries to find a set of occurrences of the seed lexicon in the document-set, creates a set of extraction patterns from them, and uses these extraction patterns to come up with a set of extractions. At the end of an iteration, the seed lexicon gets expanded with one, or a few, of the best extractions.

As in DIPRE, but in contrast with the other bootstrapping methods from our background chapter, the extraction patterns only use the exact tokens in a

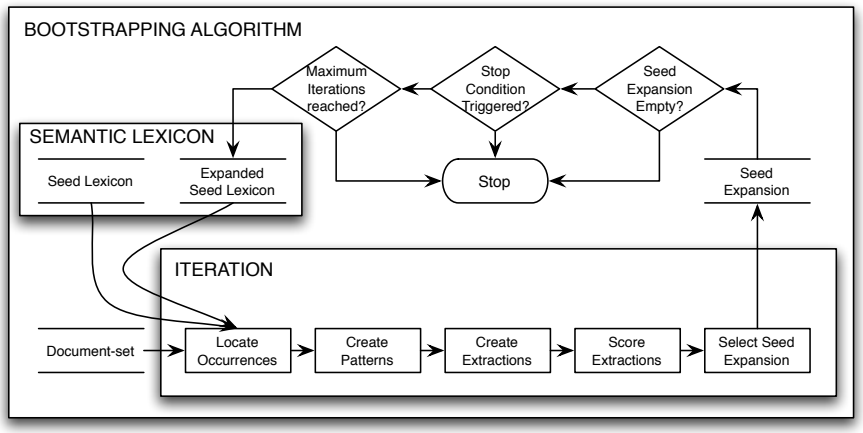

Figure 2.1: Overview of the steps in our bootstrapping algorithm.

small context around the extraction field. An advantage of doing this is that it removes the need for natural language processing of the document-set, such as done by the sentence analyser CIRCUS used in both Multi-level bootstrapping and Basilisk), and the named-entity tagger used in Snowball, which keeps the algorithm language independent. Another advantage is that it enables our algorithm to find extractions in semi-structured documents, something the other algorithms from Section 1.5 are not able to do due to the nature of their extraction patterns. As we describe Section in 3.1 the document-set used in our experiments contain semi-structured documents.

A disadvantage of not using natural language processing is that our extraction patterns need to consist of both a left and a right context. We cannot use the fact that an extraction should always be a noun phrase, as was done in Multi-level bootstrapping and Basilisk, to determine the size of the extraction. Demanding a specific left and right context, might make many of our extraction patterns too specific, and makes it harder to create extraction patterns near the begin or end of a sentence, as the extraction pattern also models a part of the previous or next sentence.

It is possible that after a few iterations, the algorithm starts to diverge from the semantic category that is to be expanded. This might happen for two reasons:

- The terms belonging to one semantic category also appear in another semantic category. For example there is a lot of overlap between people that are actors and movie directors. Since our algorithm creates extraction patterns for all occurrences of a term, regardless of the context in which the term is used, ambiguous terms will also lead to extraction patterns for another semantic category. If there are enough ambiguous terms in the seed lexicon, these extraction patterns for another semantic category might lead to their extractions getting used in the seed expansion. In effect such a seed expansion will increase the scores of the ambiguous extraction

patterns in the next iteration, and the chance of diverging into adding terms belonging to another semantic category even more.

- If a term that occurs very common in the document-set is used in the seed expansion, the algorithm is likely to diverge into adding other very common terms in the next iterations. Such a term might be an ambiguous term belonging to the semantic category that is expanded, such as for example *It* or *Go* when expanding the category *Movie Title*, or simply a term that occurs very often in the document-set. These very common terms might get selected in the seed expansion because they are simply so common that there are always some extraction patterns that extract them. In the next iteration so many new occurrences, extraction patterns and extractions are found that the algorithm almost certainly will add another very common term in the following iteration.

In our algorithm we introduce a Stop Condition to tackle the latter reason. The Stop Condition is triggered and the algorithm stopped, if the number of occurrences for one of the terms in the seed expansion surpasses a certain threshold. The first reason for diverging is harder to solve, some possible ways are described in Section 5.1 on future research.

Our algorithm assumes that the document-set has been tokenized in advance, and simply sees single spaces as token delimiters.

2.1 Steps in the Bootstrapping Loop

2.1.1 Locate Occurrences

The first step of our bootstrapping steps is to find a set of occurrences of the seed lexicon. For each of the terms in the seed lexicon we look through the entire document-set for an exact string match. Looking for the seeds is done case insensitive. For example uppercase if often used for layout in our document-set and we want our algorithm to find seeds regardless of case.

If a match is found, we create an occurrence by taking the match plus N tokens to its left and N tokens to its right. We do not strip case information from the occurrences, as matching a pattern to the document-set is done case sensitive.

No occurrences are found from matches that occur less than N tokens from the beginning or end of the document. We do this because it is impossible to create an occurrence with a window of N tokens from these matches, and we want to create patterns using a window-size of exactly N tokens.

2.1.2 Create Patterns

An Extraction Pattern consists of three parts: a middle part that is used to make the extraction and a left and a right part used for anchoring.

The left and the right part of a pattern both consist of N, the window-size, tokens. The part that makes the extraction, the middle part, can make extractions of one up to K tokens in size. This restriction on size exists for two reasons:

- If the left and right part of a pattern can be found in a document, but there exists a large amount of tokens between them, it is unlikely that the extraction is a term of the semantic category. The more the number of tokens moves away from the average number of tokens for terms of the semantic category, the more likely it becomes that the extraction is incorrect and the occurrence of the left and right part of the pattern in a document is merely accidental.

- In our implementation it is performance wise a lot cheaper to only make small extractions. That way it does not need to keep looking for a match of the right part to the end of the document, and can stop looking after K tokens.

How to set K depends on the semantic category that is expanded by the algorithm. It should be large enough to be able to extract the larger elements of the semantic category, and small enough not to make extractions from coincidental pattern matches. Usually the user of our algorithm should be able to give a fair estimate of the maximum number of tokens in the terms of the semantic category that is to be expanded. For example to be able to extract most cities a value for K of four tokens will suffice, but to be able to extract names of universities a larger value for K should be used (if only because they often contain a city).

Internally a pattern is represented as a regular expression. Extractions are made by applying this regular expression to a document and finding all matches. This regular expression is created by concatenating the left part of the pattern, a minimal matching of one up to K tokens, and the right part of the pattern.

Since we use fairly simple patterns, creating the set of patterns from the set of occurrences is straightforward. From each occurrence we create an extraction pattern, unless that extraction pattern is already in our set of extraction patterns. We do this because we do not want duplicate extraction patterns in our set. As duplicate extraction patterns will make the same extractions, they will not lead to more terms being found. In Subsection 2.1.4 we require that each extraction must be made by at least two patterns to ensure that only reliable extractions are used as the seed expansion. This requirement becomes useless if we allow duplicate extraction patterns to exist in our set of patterns.

2.1.3 Create Extractions

Finding all extractions of all of the extraction patterns is done by applying each extraction pattern it to each of the documents in the document-set. If both the left- and the right context of a pattern occur in a document, and there are maximum K tokens between them an extraction is found. Looking for extractions is done case sensitive.

Our implementation uses a few optimizations to speed up this process. First we skip finding extractions for patterns that are constructed from only one term in the seed lexicon. Since the extraction scoring function described in Section 2.1.4 below uses a seed support requirement of 2 terms, these patterns will never contribute to the extraction score.

For those extraction patterns that do meet the seed support requirement we try to decrease the number of documents in which we have to look for

extractions. This is done by an external Information Retrieval tool, Swish-e (see Swish-e) that retrieves only those documents in which the tokens of both the left and right part of the pattern occur.

These two optimizations give our algorithm a boost in speed.

2.1.4 Score Extractions

All extractions that we have created in the previous step are scored. This extraction score is used in the next step to select with which extractions we are going to expand the seed lexicon.

The extraction scoring function is of high importance to our algorithm, if it gives a high score to too many extractions that are not terms belonging to the semantic category that is expanded the algorithm might diverge, but it can only make use of very little information. Our algorithm first gives a pattern score to all the extraction patterns, and uses these scores to score each of the extractions.

Our algorithm, in contrast with Multi-level Bootstrapping (see Riloff and Jones [1999]) makes extractions using all the extraction patterns, not only a few high quality extraction patterns, which makes it important to take the quality of the extraction patterns into account in the extraction scoring function.

Determining the quality of an extraction pattern is not a trivial task, since it is not known which extractions are correct and which are incorrect. Only extractions that occur in the seed lexicon are known to be correct, which gives a minimum on the number of correct extractions. To determine the quality of an extraction pattern we use the $RlogF$ metric used in both Multi-level Bootstrapping and Basilisk.

$$PatternScore(P_i) = \frac{F_i}{N_i} log_2(F_i)$$

With F_i the number of different terms in the seed lexicon that the extraction pattern extracts, and N_i the total number of different extractions that the extraction pattern makes. Extraction patterns are given a higher score by this metric if:

- An extraction pattern is more reliable, i.e. if a larger portion of its extractions occur in the seed lexicon.
- An extraction pattern is more general, i.e. if it is able to make a larger number of correct extractions, or if more terms of the seed lexicon are extracted by the extraction pattern.

$log_2(F_i)$ in the pattern scoring function implicitly defines a seed support requirement of two seeds. Extraction patterns that only extract one of the terms in the seed lexicon are given a score of zero. Using extraction patterns that are able to extract only one term known to be correct is dangerous, and since the algorithm makes multiple iterations, it is still possible for the extraction pattern to get used in a later iteration (due to the seed lexicon being expanded).

To calculate the extraction score for an extraction, our extraction scoring function uses the pattern scores of all the extraction patterns that extract the extraction. The following function is used:

$$ExtractionScore(E_i) = \begin{cases} \sum_{k=1}^{M_i} PatternScore(P_k) & \text{if } |M_i| > 1 \\ 0 & \text{otherwise} \end{cases}$$

Or expanded:

$$ExtractionScore(E_i) = \begin{cases} \sum_{k=1}^{M_i} (\frac{F_k}{N_k} log_2(F_k)) & \text{if } |M_i| > 1 \\ 0 & \text{otherwise} \end{cases}$$

With M_i the set of extraction patterns that extract the extraction. The extraction scoring function basically takes the sum of the pattern scores of all the extraction patterns able to make the extraction, with the exception that extractions that are found by only one extraction pattern are scored zero.

We expect that this pattern support requirement of two extraction patterns will significantly lower recall, but is necessary to keep the algorithm from diverging in later iterations.

In Subsection 3.5.3 we experiment with dropping the seed support and the pattern support requirement.

2.1.5 Select Seed Expansion

Finding the seed expansion is done by appending the M unused extractions with the highest extraction score to the seed lexicon. If an expansion for the seed lexicon can be found we start with the next iteration. The algorithm stops if no expansion can be found.

Instead of selecting a constant number of extractions as our seed expansion, we could also select all extractions that score above a certain threshold, as is done in Snowball. Although this would be a certain improvement, we are more interested in being as careful as possible. In our experiments we use the most careful option of only adding one extraction to the seed lexicon of the next iteration.

Our extraction scoring function leads to the following requirements to start with the next iteration:

- The seed lexicon contains at least two elements, since extraction patterns should have a seed support of at least two.
- At least one extraction that does not exist in the seed lexicon has been made.
- This extraction has been made by at least two patterns, since extractions should have a pattern support of at least two.
- Each of these patterns also extracted two elements of the seed lexicon.

Especially during the first iterations of the algorithm, when the seed lexicon is still rather small, these requirements might not be met. This leads to the algorithm stopping, although there may be plenty of correct extractions to select an seed expansion from.

Although using a less strict extraction scoring function would alleviate this problem, it also makes it more probable for the algorithm to diverge from the

semantic category that is being expanded. We designed the algorithm such that it rather stops iterating prematurely rather than that it keeps iterating and possibly diverges.

Our algorithm does not require a minimum extraction score for an extraction to be used as the seed expansion. In later iterations, when the amount of unused extractions decreases and the extraction scores get relatively lower (compared to the extraction scores of the terms in the seed lexicon), such a minimum can be used to stop the algorithm before the remaining usable, but probably incorrect, extractions get used. Although we do not use such a minimum, the Stop Condition might end the algorithm or the set of usable extractions might get depleted before the problem of only having incorrect extractions left arises.

2.1.6 Stop Condition

Our algorithm uses a Stop Condition that is triggered if the algorithm starts diverging into adding only too common terms. If a term, for which a striking larger amount of occurrences than average can be found in the document-set, gets added to the seed lexicon a large amount of new extraction patterns will be found. Since its occurrences appear in more diverse contexts, the extractions made by the extraction patterns are more diverse. The new extraction patterns also have a large influence on the extraction scores, making it very likely for another too common term to get the highest extraction score. This effect on the occurrences, patterns and extractions only increases if that too common term is selected as the seed expansion for the next iteration.

Our Stop Condition is triggered if for more than O consecutive iterations, for at least one term in the seed expansion more than P times the average number of occurrences are found. By default O is set to 2 consecutive iterations to allow correct terms that are very common, such as the movie title *Go* or *It*, to be added to the seed lexicon if that does not make the iteration process diverge. P is by default set to a factor of 10.

Another option to stop the algorithm from diverging is to use a stop list of terms that may not exist in the seed expansion. In pilot experiments done during the implementation phase of our algorithm we found that it is hard to think of enough terms for such a list, every-time we expanded the stop list a new term appeared to be too common.

An obvious improvement to the Stop Condition is to skip too common words from the seed expansion instead of to stop when too common words are selected as the seed expansion. This and other options are described as future research in Section 5.1, as our implementation can not be easily modified for this.

2.2 User Definable Parameters

The following parameters can be set by the user. They are described in more detail in Section 2.1.

- N, the window-size. By default the algorithm uses a window-size of two tokens. A different window-size can be used but we found out that, at least for our document-set of movie reviews, using a window-size of only one token led to patterns that are to general. Using a window-size of three

tokens led to patterns that were too specific and extracted only the seed element for which it was made.

- M, the number of extractions (default 1) with which the seed lexicon is expanded after each iteration.
- K, the number of tokens (default 6) a pattern can make extractions up to.
- L, the maximum number of iterations (default 100) that the algorithm should attempt to make. This setting can be used to keep running a large number of experiments feasible.
- O and P, used by the Stop Condition. By default these are set to $O = 2$ and $P = 10$, so that the Stop Condition is triggered if for two consecutive iterations, there exists a term in the seed expansion that has over 10 times the average number of occurrences.

Apart from these parameters, the behaviour of the algorithm is also influenced by the seed lexicon and the document-set.

2.3 An Example

In this section we will illustrate our bootstrapping algorithm with an example that spans two iterations. We will start with the a seed lexicon of two movie titles (see Table 2.1) and a small document-set. The tokenized pieces of text from this document-set relevant to our example are shown in Table 2.2. It should be noted that each space splits two tokens, so that for example d_1 consists of 11 tokens ('s and " are also tokens). In our example we use the following user definable settings:

- $N = 2$, a window-size of two tokens.
- $M = 1$, after each iteration we expand our seed lexicon with the single best extraction.
- $K = 6$, the patterns will make extractions that are up to six tokens in length.
- $L = 2$, our example stops after two iterations.
- $O = 2$, $P = 10$, the Stop Condition is triggered if for two consecutive iterations the seed expansion consists of an extraction that has over 10 times the average number of occurrences.

#	**Term**
s_1	Titanic
s_2	The Blair Witch Project

Table 2.1: Initial Seed Lexicon

#	Text Snippet
d_1	... It 's similar to " Titanic " in that it realizes ...
d_2	... compared to " The Blair Witch Project " in that the terror ...
d_3	... I think that " the blair witch project " is a great film ...
d_4	... Thus begins The Blair Witch Project , one of the best ...
d_5	... Why do I say that " The Blair Witch Project " is a great film? ...
d_6	... states adamantly that " Titanic " is not a disaster film ...
d_7	... releases of " The Blair Witch Project " and " The Sixth Sense " ...
d_8	... real star of " The Matrix " and the film 's powerhouse ...
d_9	" Titanic " in my opinion ...
d_{10}	... writer of " The Blair Witch Project " and " Altered " ...
d_{11}	... second only to " The Matrix " in my opinion ...
d_{12}	... shows that " The Shawshank Redemption " is overrated ...
d_{13}	... probably think that " Episode 1 " is marvelous . ...
d_{14}	... paying homage to " titanic . " in casting ...
d_{15}	..., and that " The Matrix " is a pretty ...
d_{16}	... to learn that " The Motorcycle Diaries " is a ...
d_{17}	... is similar to " The Shawshank Redemption " in a lot ways ...
d_{18}	... out of " Minority Report " and " SimOne . " ...
d_{19}	... midnight screening of " The Texas Chainsaw Massacre " and that ...
d_{20}	... hope so . " The Matrix " may not have ...

Table 2.2: Relevant Snippets from our Example Document-set

2.3.1 First Iteration

The first iteration is started with a seed lexicon of two elements. After the first iteration we have created an expanded seed lexicon, that is three elements large, with which we start the second iteration.

2.3.1.1 Locate Occurrences

The set of occurrences in Table 2.3 is created by locating occurrences for all the elements of the seed lexicon. First the occurrences o_1 and o_2 are found for the seed element *Titanic* from text snippet d_1 and d_6 in Table 2.2. We cannot create an occurrence from d_9. There is only one token left from the seed text, and our window-size of two tokens makes it impossible to find occurrences with a context of less than two tokens.

The rest of the occurrences are then found for the seed element *The Blair Witch Project*. Because the search for seed elements is done case insensitive, we find occurrence o_4 from d_3.

2.3.1.2 Create Patterns

From the set of occurrences in Table 2.3 the set of patterns in Table 2.4 is created. The columns F_i, N_i and $\frac{F_i}{N_i} log_2(F_i)$ are used in the calculation of the extraction score. In this step we calculate F_i, the number of different elements of the seed lexicon that a pattern extracts.

For each occurrence we try to create a pattern. The first pattern, p_1, has

#	Left Context	Seed Element	Right Context
o_1	to "	Titanic	" in
o_2	that "	Titanic	" is
o_3	to "	The Blair Witch Project	" in
o_4	that "	the blair witch project	" is
o_5	Thus begins	The Blair Witch Project	, one
o_6	that "	The Blair Witch Project	" is
o_7	of "	The Blair Witch Project	" and
o_8	of "	The Blair Witch Project	" and

Table 2.3: The Set of Occurrences Found in Iteration 1

been created by occurrence o_1. When the algorithm tries to create a pattern for occurrence o_3 it notices that the pattern would be exactly the same as p_1 and links occurrence o_3 to pattern p_1. F_i for pattern p_1 is increased, since the seed element *The Blair Witch Project* is not yet known to be extracted by p_1.

For occurrences o_4, o_6 and o_8 such linking also occurs. For occurrences o_6 and o_8 F_i is not increased. Pattern p_2 and p_4 are already known to extract the seed element *The Blair Witch Project*.

#	Left	Right	F_i	N_i	$\frac{F_i}{N_i} log_2(F_i)$
p_1	to "	" in	2	5	0.40
p_2	that "	" is	2	6	0.33
p_3	thus begins	, one	1	-	0
p_4	of "	" and	1	-	0

Table 2.4: The Set of Patterns Found in Iteration 1

2.3.1.3 Create Extractions

From the set of patterns in Table 2.4 the set of extractions in Table 2.5 are created.

Pattern p_3 and p_4 are created from only one occurrence (p_3 was created from o_5), or from occurrences about only one seed element (p_4 is created from o_7 and o_8 both about *The Blair Witch Project*). For these patterns we do not need to find the extractions as they do not contribute to the extraction score.

#	Extraction	p_1	p_2	Score
e_1	the blair witch project	✓	✓	0.73
e_2	titanic	✓	✓	0.73
e_3	the matrix	✓	✓	0.73
e_4	the shawshank redemption	✓	✓	0.73
e_5	titanic.	✓		0
e_6	episode 1		✓	0
e_7	the motorcycle diaries		✓	0

Table 2.5: The Set of Extractions Found in Iteration 1

2.3.1.4 Score Extractions

The extraction score in Table 2.5 is calculated. Extractions e_1, e_2, e_3 and e_4 are extracted by pattern p_1 and p_2. They get scored $0.40 + 0.33 = 0.73$, the sum of the Pattern score of pattern p_1 and p_2.

Extractions e_5, e_6 and e_7 are scored 0, as they are only extracted by one pattern.

2.3.1.5 Select Seed Expansion

The seed lexicon is then expanded with the best M extractions. Since we use $M = 1$ in this example and our experiments, we add the single best extraction not yet in the seed lexicon.

In our example two extractions, *the matrix* and *the shawshank redemption*, are scored the highest. The extraction *the matrix* is randomly picked as the seed expansion.

2.3.1.6 Stop Condition

The Stop Condition is triggered if for two consecutive iterations the seed expansion consists of an extraction that has over 10 times the average number of occurrences. Since this is the first iteration, the Stop Condition will not be triggered.

2.3.2 Second Iteration

With the Seed Lexicon expanded with *the matrix* the second iteration gets started.

2.3.2.1 Locate Occurrences

Since an element never gets removed from the seed lexicon, the seed lexicon only gets expanded, and the document-set never changes we can reuse the set of occurrences found in iteration 1. By expanding this set of occurrences with the occurrences found for our new seed element *the matrix* (see Table 2.6) we find the set of occurrences for iteration 2.

#	Left Context	Seed Element	Right Context
⋮	⋮	⋮	⋮
o_9	of ”	The Matrix	” and
o_{10}	to ”	The Matrix	” in
o_{11}	that ”	The Matrix	” is
o_{11}	. ”	The Matrix	” may

Table 2.6: Additions to the Set of Occurrences in Iteration 2

2.3.2.2 Create Patterns

The new set of patterns in Table 2.7 is created from the expanded set of occurrences.

Occurrence o_9, o_{10} and o_{11} lead to patterns already known in iteration 1. For those patterns F_i gets increased, which in case of pattern p_4 means that it will be used to create extractions with.

Occurrence o_{11} leads to the new pattern p_5. Since this pattern only extracts one element of the Seed Lexicon it will not yet be used to create extractions with. Just as what happened with pattern p_4 in this iteration, pattern p_5 might get used for extraction in a later iteration.

#	Left	Right	F_i	N_i	$\frac{F_i}{N_i} log_2(F_i)$
p_1	to "	" in	3	5	0.95
p_2	that "	" is	3	6	0.79
p_3	thus begins	, one	1	-	0
p_4	of "	" and	2	4	0.5
p_5	. "	" may	1	-	0

Table 2.7: The Set of Patterns Found in Iteration 2

2.3.2.3 Create Extractions

In addition to pattern p_1 and p_2 used in iteration 1, we now also use pattern p_4 to create the set of extractions shown in Table 2.8. The set of extractions gets expanded with two new extractions: e_8 and e_9.

#	Extraction	p_1	p_2	p_4	Score
e_1	the blair witch project	✓	✓	✓	2.24
e_2	titanic	✓	✓		1.74
e_3	the matrix	✓	✓	✓	2.24
e_4	the shawshank redemption	✓	✓		1.74
e_5	titanic.	✓			0
e_6	episode 1		✓		0
e_7	the motorcycle diaries		✓		0
e_8	minority report			✓	0
e_9	the texas chainsaw massacre			✓	0

Table 2.8: The Set of Extractions Found in Iteration 2

2.3.2.4 Score Extractions

All the extraction scores are recalculated as shown in Table 2.8. Because patterns p_1 and p_2 can extract more elements of the Seed Lexicon than in iteration 1, their pattern score has been increased. This also means that the extraction scores of extractions e_1, e_2, e_3 and e_4 get increased.

Since extractions e_1 and e_3 now also get extracted by pattern p_4 their score is increased even more.

2.3.2.5 Select Seed Expansion

Only *The Shawshank Redemption*, extraction e_4, is left to be used for the expansion of the Seed Lexicon. The other extractions are either already in the Seed Lexicon or scored zero.

2.3.2.6 Stop Condition

The Stop Condition is triggered if for two consecutive iterations the seed expansion consists of an extraction that has over 10 times the average number of occurrences. In the first iteration the seed expansion consisted of *The Matrix* for which 4 occurrences could be found. The average number of occurrences before that iteration was also 4, so the Stop Condition will not trigger.

If in the next iteration no new extraction is found that is made by at least two patterns, either by finding a new pattern that makes e_5, e_6, e_7, e_8 or e_9 in Table 2.8 or by finding two new patterns that make the same new extraction, the algorithm will stop.

In this case the algorithm already stops after this iteration, L was set to make a maximum of two iterations. The final Seed Lexicon consists of the initial two elements of the Seed Lexicon, *The Blair Witch Project* and *Titanic* and got expanded with *The Matrix* and *The Shawshank Redemption*.

Chapter 3

Experiments & Results

With our experiments we try to describe the behaviour of our bootstrapping algorithm. By looking at the effect of varying the two most important factors, the seed lexicon and the document-set, we find a baseline setting for the algorithm. This baseline setting is then used in experiments where we vary other user definable parameters and find seed expansions for other semantic categories. In all our experiments we use a document-set of movie reviews.

3.1 Movie Review Document-set

The document-sets used in our experiments consist of movie reviews. We have picked this domain because there is an enormous amount of movie reviews available on the internet, and because the semantic categories in them have interesting properties.

The document-sets in our experiments are subsets from a collection of 40,000 movie reviews. This collection consists of articles that were posted to the Usenet newsgroup `rec.arts.movies.reviews` between 1987 and 2005. This newsgroup is archived by the Internet Movie Database (see IMDB, IMDB-Reviews). IMDB provides a link to the reviewed movie in its database, and the name of the reviewer next to a lot of the archived articles.

The collection used in our experiments is a copy of this archive, and has been partially scraped by Pang et al. [2002] in an experiment on sentimentality analysis. In order to restore the original newsgroup articles, we have removed the HTML markup added by IMDB, and converted the articles to plain text. To the plain text we applied a simple tokenization algorithm, to make the reviews usable to our bootstrapping algorithm. The tokenization algorithm is very simple (as we explain in Section 3.3.4 it is too simple), it first replaces all strings of whitespace with a single space character, and then places all non alphanumeric characters at the beginning or ending of a token into a new token.

The movie reviews are free text, with some parts being more semi-structured. Most of the reviewers use a distinctive layout style, such that the articles often consists an unstructured body that reviews a movie accompanied by a header and/or footer that is more structure. The header/footer is for example used to list the title, writers, directors and actors or to summarize the movie. For example one reviewer always appends his reviews with a footer that lists his

recently reviewed movies and how he rated them. Another reviewer almost always writes movie titles capitalised.

Several semantic categories can be found in this set of movie reviews. For example *Movie Title*, *Actor Name*, *Writer*, *Director*, *Certification* and *Running Time*. We will experiment with *Movie Title*, *Actor Name* and *Certification*.

To get an idea of the amount of information in our collection of movie reviews, we have manually annotated a document-set of 250 reviews. In these reviews we counted a total number of 871 different movie titles, 1131 different actors/actresses and 12 different certifications. These numbers will not increase linearly for larger document-set. Due to the fact that many movies are reviewed multiple times and that almost all actors play in more than one movie, a document-set of 500 reviews will contain less than twice the number of movie titles or actor names than a document-set of 250 reviews. This effect is even stronger for *Certification* than it is for *Movie Title* or *Actor Name*, as the number of different certifications is very low.

3.2 Baseline Experiments

The two most important factors in our bootstrapping algorithm are the seed lexicon and the document-set. In our baseline experiments, we try to determine the effect of using different seed lexicons and document-sets. The results of these experiments will tell us how the size of the seed lexicon and the size of the document-set influences the resulting seed expansion, and what sizes are the minimum for the algorithm to perform reasonable well. We will use these settings as the default setting for our later experiments. In all of our baseline experiments we let the algorithm expand a seed lexicon of *Movie Title*'s, since, as described in Section 3.2.1, for this semantic category we have a way of automatically evaluating the resulting seed expansion.

To see the effect of the seed lexicon we vary the size of the seed lexicon and which seed elements are in it. The same is done for our document-set, and for all combinations of seed lexicon and document-set we perform an experiment. We expect that the performance of the algorithm is influenced in the following way:

- Both the seed lexicon and document-set can be too small for the algorithm to be able to expand the seed lexicon in the first iterations. In those cases there are not enough occurrences, patterns and extractions found to get the algorithm to enlarge the set of patterns.

- Using a larger seed lexicon, or a larger document-set leads to higher precision of the expanded seed lexicon. Enlarging the seed lexicon or the document-set lets the algorithm find more occurrences, patterns and extractions. We expect this to allow our extraction scoring function to better differentiate correct extractions from incorrect extractions.

In our experiments we vary the size of the seed lexicon between 2 and 20 terms. The document-set consists of 125, 250, 500, 1000, 2000 or 4000 documents. For each of the combinations of seed lexicon size and document-set size we make 16 experiments (for each seed lexicon size there exists 4 different seed lexicons, and for each document-set size there exist 4 different document sets).

All seed lexicons are created from one of four randomly chosen sequences of the list of *Movie Title*'s in Appendix A.1. From each of these four sequences a seed lexicon of 20 elements is created. By keeping to remove the last element of the sequence we create the seed lexicons of 19, 18, etc. elements. This allows us to see the effect of expanding a seed lexicon with one more element. Four sequences of document sets are created in a similar fashion.

The other user definable parameters have been kept at their default values. This means that we use a window-size of two tokens, after each iteration expand the seed lexicon with the single best extraction, extractions can be up to six tokens in size, and we make up to 100 iterations. We have chosen to make 100 iterations, as a compromise between the experiments still being computably feasible and making enough iterations to obtain useful results. We determined this number from pilot experiments done during our implementation phase where we found out that later iterations take more time than the earlier iterations. In later iterations more occurrences, patterns and extractions have to be investigated than in earlier iterations. The time later iterations take can be decreased by implementing better caching strategies in our algorithm.

3.2.1 Evaluation Method

The standard approach of evaluation used for Information Extraction experiments cannot be used. We experiment on unannotated document-sets, and instead of calculating precision and recall over single documents, we need to calculate precision and recall over all extractions found in the entire document-set.

The standard approach calculates precision and recall for each single document using the annotation layer. In our semantic lexicon expansion task, we do not need to extract every instance from each of the documents, extracting every instance only once is enough. If we were to use the standard approach this leads to a too low recall.

Another problem is the evaluation of ambiguous terms, a term that is too describe instances of different semantic categories. For example in our collection of movie reviews, the term *Go* might be used as both a movie title and a verb. Using the annotation layer to calculate precision, the former is seen as a correct extraction while the latter is seen as an incorrect extraction. In our semantic lexicon expansion task both extractions should be considered correct. Even if a term, used to describe an instance of another semantic category, is extracted and used as a seed expansion, it is likely that the next iteration leads to that term being extracted from a context where it is used to describe an instance of the semantic category that is expanded.

As a solution to these problems, and to deal with the fact that we only have unannotated documents we base our precision on a list of correct extractions. We simply compare the extractions found by our algorithm to this list of correct extractions.

For the semantic category *Movie Title* we use a list of about 700,000 movie titles, all the movie titles and their alternative titles/synonyms known to the Internet Movie Database. This is also the reason why we have picked *Movie Title* for our baseline experiments, we do not have such a list for actor names or certifications.

Comparing extractions to this list has two problems:

- There are more movie titles on this list than exist in our collection of movie reviews. Terms that are never used to describe an instance of a movie title are still seen as a correct extraction, perhaps leading to an artificially high precision.

- There are more terms that refer to a particular instance of a movie title than exist on our list. For example the instance *Star Wars: Episode I - The Phantom Menace* is also known in our list by 27 different terms, but the obviously correct *Star Wars 1* is not one of them. Since this makes correct extractions seen as incorrect, it will lower precision.

In Section 3.1 we found that there are approximately 870 movie titles in a document-set of 250 reviews. Since in our baseline experiments we make a maximum of 100 iterations, the calculation of recall is problematic, but an indication can be found in the number of correct extractions found by the algorithm, and in Section 3.5 we experiment with continuing to iterate after iteration 100.

3.3 Results

In the beginning of Section 3.2 we anticipated that for too small seed lexicons or document sets, the algorithm will not get past the first iterations, and that after that point is reached using a larger set would have a positive influence on the precision of the resulting seed expansion. Our results do not completely reflect these anticipations.

If the algorithm is not able to get past the first iterations, it was not able to expand the set of usable extractions after the first iteration. The set of usable extractions are those extractions with extraction score higher than zero that do not yet exist in the seed lexicon. The algorithm does not get past the first iterations if it only finds a few extraction patterns, and is not able to expand this set using the seed expansions of the first iterations. As described in Section 2.1.4, for an extraction to be considered as a seed expansion it needs to be made by at least two patterns, which in turn are able to extract at least two different seeds.

Using our experiments we want to determine the minimum size of the seed lexicon and the minimum size of the document-set that enables the algorithm to find a real expansion for the set of extraction patterns and extractions. To determine these numbers we will look for those experiments where the algorithm was able to make at least 10 iterations. In our results we call such experiments as "started iterating". If we do not do this, and determine these numbers by the first experiment that made at least one iteration, our results would be clouded by experiments in which the algorithm finds only two extraction patterns which it can not expand. Demanding at least 10 iterations solves this.

In addition to not being able to get past the first iterations for too small document sets, we also find that it is possible for the document-set to become depleted in later iterations. This happens if the algorithm is not able to find any new extraction patterns that lead to new extractions.

3.3.1 Size of the Document-set

To see what effect the size of the document-set has on the expanded seed lexicon we used the experiments where the entire seed lexicon of 20 movie titles was used. In Table 3.1 we show in what percentage of these experiments the algorithm started iterating, the average number of iterations that the algorithm was able to reach, and the average precision of all the seed expansions found after the final iteration.

Document -set Size	% of the Experiments started	Average Iteration Reached	Avg. Prec.	Avg. Nr. of Unused Extractions	Avg. Prec. of Unused Extractions
125	0	-	-	-	-
250	0	-	-	-	-
500	0	-	-	-	-
1000	50	39	0.92	-	-
2000	100	100	0.94	45	0.94
4000	100	100	0.91	344.25	0.90

Table 3.1: Influence of the size of Document-set

In none of the experiments on document-sets of 500 or less reviews the algorithm started iterating, in no experiment did the algorithm make it to the 10th iteration. Only in half of the experiments on document-sets of size 1000 the algorithm started iterating, but the maximal amount of 100 iterations was not reached, the algorithm made either 38 or 40 iterations before stopping due to a lack of usable extractions. The experiments on document-sets of size 2000 and 4000 all started iterating and all reached iteration 100.

The table also shows the average amount and the average precision of the unused extractions. These extractions are the usable extractions found at iteration 100, i.e. the extractions with extraction score higher than zero that do not exist in the seed lexicon. Combined with the number of correct extractions in the expanded seed lexicon, the number of correct unused extractions can be used as an indication for the recall that can be achieved were the algorithm allowed to make it past iteration 100.

The results show that our algorithm needs a considerable amount of documents before it is able to start iterating. This is a result of our algorithm being conservative, but also leads to precision measures of about 90 percent. The larger the document-set becomes, the more likely it becomes for the algorithm to start iterating. The amount of unused extractions for the experiments of document-sets of 2000 or more reviews show us that there is still room for the algorithm to continue iterating after iteration 100. In Subsection 3.5.1 we experiment with continuing to iterate after iteration 100.

3.3.2 Size of the Seed Lexicon

To see the effect of the size of the seed lexicon on the expanded seed lexicon, we look at the results for those experiments where the size of the seed lexicon was just large enough to start iterating. In Table 3.2 we show the average minimum

Document -set Size	Minimum Seed Size Needed	% of the Experiments started	Average Iteration Reached	Avg. Prec.	Avg. Nr. of Unused Extractions	Avg. Prec. of Unused Extractions
1000	12.00	50	38.88	0.92	-	-
2000	8.81	100	100	0.94	43.63	0.94
4000	4.56	100	100	0.92	260.69	0.91

Table 3.2: Influence of the size of the seed lexicon

size of the seed lexicon, the average number of iterations that was reached, and the average precision of those experiments.

Our results show that a seed lexicon can be too small for the algorithm to really start iterating. The minimum size of the seed lexicon needed to start iterating decreases for larger document-sets. In a larger document-set there are more occurrences and patterns to be found, making it easier to find enough extraction patterns and extractions to keep expanding the seed lexicon.

There is a large amount of variation in the minimum size of the seed lexicon needed between the seed lexicons used in experiments on the same document-set. The variation can be explained by looking into the seed lexicons used. The last movie title added to the minimum seed lexicon is often a specific movie title. This movie titles makes it possible for an extraction pattern to be found that extracts at least two terms of the seed lexicon. For example, in three out of four of the minimum seed lexicons used on a document-set of 1000 movie reviews, *The Sixth Sense* was used as the last element of the seed lexicon. This variation decreases for larger document-sets, where it becomes easier to find enough extraction patterns and extractions to keep iterating the algorithm.

Comparing these results to the results we found in the experiments where the entire seed lexicon of 20 movie titles was used, see Table 3.1, we find that for the document-set of 1000 movie reviews, there is no effect in enlarging the seed lexicon. The iteration that the algorithm was able to reach, and the precision were about equal. For the experiments on document-sets of 2000 and 4000 reviews, there is no significant change in the precision found over the first 100 iterations. Comparing the average number of usable extractions at iteration 100, we find them to be about equal for document-sets of 2000 reviews, and a lot smaller for document-sets of 4000 reviews.

In Subsection 3.5.1 we experiment with continuing to iterate after iteration 100 for both the minimum seed lexicons we found in this section, and the entire seed lexicon of 20 movie titles used in the experiments in Subsection 3.3.1.

3.3.3 Analysis of the Iteration Process

To get a better feeling of the iteration process we have manually inspected one of our experiments. We have chosen to inspect one of the experiments on a document-set of 4000 reviews, as this will also show us how many extractions are left at iteration 100.

In Figure 3.1 we show the total number of occurrences found for all terms in the seed lexicon, and in Figure 3.2 the average number of occurrences found for a term in the seed lexicon. Overall the average number of occurrences decreases, indicating that the movie titles that occur more often in the document-set are

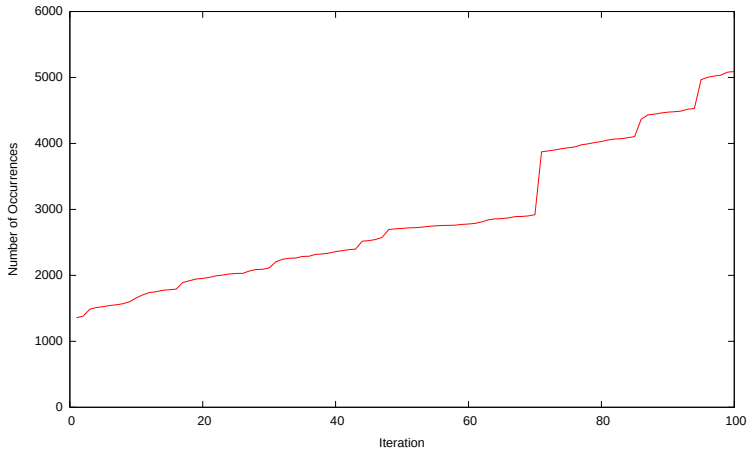

Figure 3.1: Total Number of Occurrences

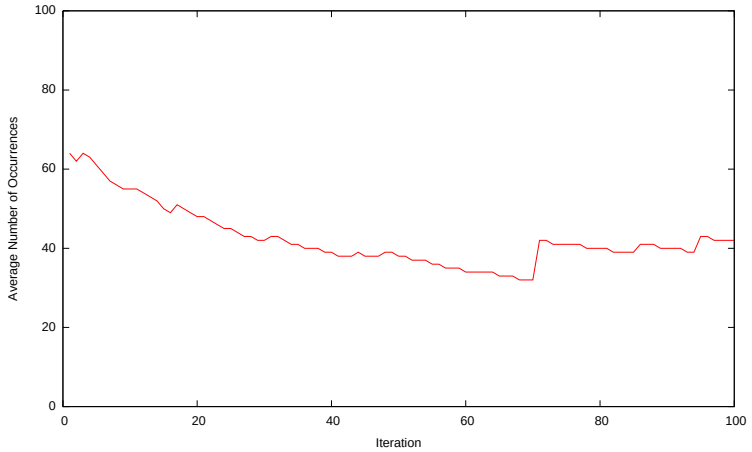

Figure 3.2: Average Number of Occurrences

more likely to be extracted in the earlier iterations of the algorithm than less often occurring movie titles.

In this figure there are a few points where there is a sharp increase in the number of occurrences. The three most significant increases correspond to the extractions *Sense*, *Spirit* and *2001* being added to the seed lexicon. These extractions are all made from a context where they are used to refer to a movie title, *The Sixth Sense*, *Spirit: Stallion of the Cimmarron* and *2001: A Space Odyssey* respectively. Although these three terms lead to a large amount of occurrences, most of them not representative of the context in which movie titles occurs, they do not lead to a large amount of new extraction patterns, which can be seen in Figure 3.3.

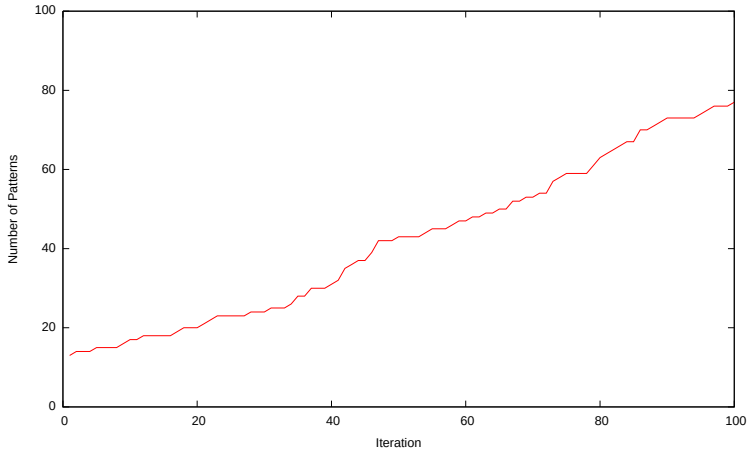

Figure 3.3: Total Number of Patterns

The number of extraction patterns increases linearly, and there are no sharp increases. This is a result of the seed support requirement in our algorithm. Even though adding a single term to the seed lexicon can lead to a large increase in the number of occurrences, these occurrences only lead to new extraction patterns if the occurrence can also be found for another term in the seed lexicon.

In Figure 3.4 we show the total number of extractions with extraction score greater than zero and the number of these extractions that can be used as the seed expansion in later iterations, i.e. the number of extractions that do not yet exist in the seed lexicon. There are still a lot of usable extractions left at iteration 100, making it likely that the algorithm is able to keep iterating after iteration 100. The slope of the number of usable extractions is either almost flat, or makes a sharp increase. Since the increase in the number of extraction patterns is fairly linear, this means that many of the extraction patterns lead to no, or only a few previously unknown extractions, while some of the extraction patterns enable the algorithm to reach a large amount of new extractions.

In this figure we can also see that the plateaus where almost no new usable extractions are found get larger, and the sharp increases less sharp in later iterations. This indicates that at some point after iteration 100 the number of usable extractions is likely to reach a maximum and that after this point it starts to decrease. This makes it possible to see the number of usable extractions that

are correct as an indication for the total number of correct extractions that the algorithm is able to reach, were it allowed to continue to iterate after iteration 100.

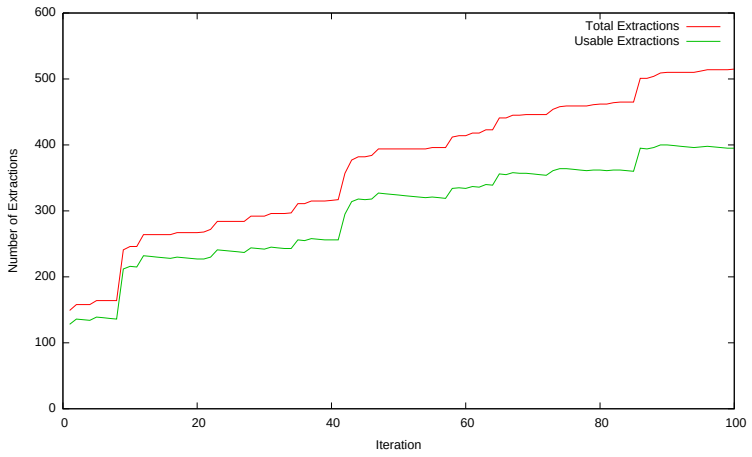

Figure 3.4: Total and Usable Number of Extractions

#	Extraction Pattern
p_1	about " <EXTRACTION> " is
p_2	) - <EXTRACTION> (8/10
p_3	, " <EXTRACTION> " has
p_4	Review... ’ <EXTRACTION> (2000
p_5	with " <EXTRACTION> " is
p_6	. " <EXTRACTION> " takes
p_7	, " <EXTRACTION> " doesn’t
p_8	. " <EXTRACTION> " also
p_9	, " <EXTRACTION> " offers
p_{10}	, " <EXTRACTION> " will

Table 3.3: Best 10 extraction patterns for one of the experiments on *Movie Title*

Table 3.3 shows the top 10 extraction patterns used at iteration 100. The best extraction patterns make only use of one word, the rest of the context consists of punctuation such as , or ".

3.3.4 Analysis of our Evaluation Method

In Subsection 3.2.1 we described some possible shortcomings of our evaluation method. We will see if these shortcomings occur by looking at the results for our Baseline Experiments.

The precision as calculated by our evaluation method can be too high, if terms are seen as correct extractions even though they are never used in a context to describe a movie title, or too low, if correct terms are seen as incorrect extractions due to a slightly different spelling.

By combining the results of all our Baseline Experiments we see that a total number of 631 different terms have been extracted. Our evaluation method considered 573 of these 631 terms as correct extractions. To see if these terms are indeed correct extractions, we can search in the document-sets for a context in which the term is used to describe a movie title. We will not do this due to being time-constraint. The other 58 terms are seen as incorrect extractions by our evaluation method and are shown in Appendix B. We have manually inspected these terms to see if they are indeed incorrect extractions.

Although there are some incorrect extractions on the list, the large part of the list consists of correct extractions. Our evaluated method saw these as incorrect extractions due to the following reasons:

- In 18 cases, a slightly different spelling than the one known for that movie was used. For example *3,000 miles to graceland* is extracted, while *3000 miles to graceland* is on our evaluation list or *starsky and hutch* while *starsky & hutch* is on our list.
- 26 cases happen because there is a difference in the tokenizer used for our document-sets, and the one used to tokenize our evaluation list. Apparently the tokenizer used on our document-sets is not able to tokenize `'s` or to correctly tokenize words that end in two or more punctuation characters. For example `what?)` gets tokenized to `what? )` instead of the anticipated `what ? )`. This difference unfortunately occurred to us in a too late stadium to re-tokenize the document-sets used.
- In 4 cases a referent to a movie title has been extracted. For example *jade scorpion* used in a review of *the curse of the jade scorpion* after the movie title has been introduced.

Taking into account that the large part of the extractions validated as incorrect are in fact correct extractions, we find that the precision found in Table 3.1 and 3.2 is too low.

Another shortcoming of our evaluation method lies in how recall is measured. Since we make a maximum of 100 iterations in our Baseline Experiments, it is hard to give a measure of recall. As described in Section 3.2.1 an indication of the possible recall that the algorithm is able to reach can be found by looking at the number of correct extractions found at iteration 100. Although we found in Subsection 3.3.3 that the speed at which new extractions are found decreases, this indicator is rather unreliable. For example the algorithm might stop in the 101th iteration due to the Stop Condition being triggered, or the algorithm might find a new extraction pattern of high quality, enabling the algorithm to reach a large amount of previously unknown correct extractions. In Subsection 3.5.1 we experiment with continuing to iterate after iteration 100, and compare this indicator to the total number of correct terms in the expanded seed lexicon.

3.4 Conclusions

From our experiments we found initial evidence that our hypothesis from Section 3.2 is incorrect. At least for the first 100 iterations there is no effect on precision in enlarging the seed lexicon or the document-set. We have found that there

exists a minimum size of the seed lexicon and document-set for the algorithm to start iterating. After this minimum has been reached, enlarging these sets has no effect on the precision of the expanded seed lexicon. The minimum needed for the seed lexicon can not be known in advance.

To keep the seed lexicon, and the amount of work a user needs to do, to a minimum our algorithm could be encapsulated. This encapsulating loop would ask the user to keep appending seeds to the seed lexicon until the algorithm starts and get past the first crucial iterations.

We have found that the maximum number of iteration is too low for document-sets that are 2000 movie reviews in size or larger. Experimenting with a larger L will show us how the algorithm continues iterating after the 100th iteration.

By analysing the manually annotated document-set we find that the default value for the maximum tokensize for an extraction, $K = 6$, is reasonable. This size covers an estimate of 93 percent of all movie titles. We will not experiment with other values for this user definable parameter.

3.5 Other User Definable Parameters

In Section 3.4 we have described the behaviour of our bootstrapping algorithm using different sizes for the seed lexicon and document-set. In this section we experiment with the other settings of our algorithm.

3.5.1 Maximal Iterations

Seed Size	**Document -set Size**	**% of the Experiments started**	**Average Iteration Reached**	**Avg. Prec.**
min	1000	50	38.88	0.92
max	1000	50	39	0.92
min	2000	100	164.88	0.93
max	2000	100	187.25	0.94
min	4000	100	527.25	0.90
max	4000	100	537.00	0.89

Table 3.4: Results for dropping the constraint on the maximal iteration

In our baseline experiments, time constraints forced us to use the rather low maximal number of iterations, L, of 100. This number proved to be large enough for experiments on document-sets of 1000 or less movie reviews, but was reached for each experiment on the document-sets of 2000 and 4000 reviews.

To see the effect of continuing after the 100th iteration, we have experimented with the constraint on L removed. These experiments will show us the precision and recall for expanding the semantic category *Movie Title* on document-sets of size 2000 and 4000, and tell us if the average number of usable extractions at iteration 100 is a reasonable indicator for the recall.

For each of the seed lexicons used in Subsection 3.3.1 and the minimum seed lexicons found in Subsection 3.3.2 combined with the four document-sets of size 2000, and the four of size 4000 used in our baseline experiments we start the

algorithm, leading to a total of 64 experiments. In Table 3.4 we show the results for these experiments. In this table we have also added the results found in our baseline experiments for the document-sets of 1000 reviews.

The precision found over the first 100 iterations is about the same as the precision found over all iterations. The average number of iterations reached is slightly smaller if the minimum seed lexicon is used instead of the seed lexicon of 20 movie titles.

As a measure of recall we show the average number of correct seed expansions in Table 3.5. This table also shows an estimation to the average number of correct seed expansions. This estimation is done by looking at the precision found over the first 100 iterations, and the precision and number of usable extractions at iteration 100, and can be used if the algorithm makes a maximum number of 100 iterations. Using a larger document-set has a positive effect on the number of correct terms in the seed expansion, an effect that is stronger than the small positive effect found for using a larger seed lexicon.

The estimation to the number of correct seed expansions is rather crude, but is for example able to show the effect of using a larger document-set.

Seed Size	**Document -set Size**	**Average Number of Correct Seed Expansions**	**Average Estimated Number of Correct Seed Expansions**
min	1000	35.77	-
max	1000	35.88	-
min	2000	153.33	135.01
max	2000	176.02	136.30
min	4000	474.53	329.23
max	4000	477.93	400.83

Table 3.5: Number of correct terms in the seed expansion for *Movie Title*

In all experiments on the document-sets of 2000 reviews the algorithm stopped iterating due to a lack of usable extractions. All, but one, experiments on the document-sets of 4000 reviews stopped because the Stop Condition was triggered. In one of the experiments on a document-set of 4000 movie reviews and using a seed lexicon of 20 movie titles the algorithm unfortunately ran out of memory and crashed. This happened at iteration 600. Looking at the usable extractions at iteration 600, we find that there are several too common terms (*It*, *This Film*, *Film* and *The Film*) amongst the highest scored extractions. It is reasonable to assume that the Stop Condition would have been triggered in one of the next iterations for this experiment.

3.5.2 Window Size

In our baseline experiments we used the default window-size, N, of 2. This creates extraction patterns with two tokens in its left and right context. To see the effect of changing the window-size we have experimented with a window-size of one and a window-size of three tokens.

For each window-size combined with the four document-sets of size 2000, and the four of size 4000 used in our baseline experiments we start the algorithm,

leading to a total of 16 experiments.

Window Size	Document -set Size	Average Iteration Reached	Avg. Prec.
1	2000	2.50	0.81
1	4000	2.25	0.79
3	2000	0.75	0.17
3	4000	5.50	0.85

Table 3.6: Results when using different window-sizes

The algorithm is not able to start iterating, i.e. it is not able to reach at least iteration 10, in any of the experiments. Since we use the default value of M this means that in none of the experiments we were able to find 10 extractions. In Table 3.6 we show the average iteration that the algorithm was able to reach, and the average precision.

6 out of 8 experiments where a window size of one token was used where stopped due to the Stop Condition being triggered. In the first two iterations a too common term was chosen as the seed expansion. The other 2 experiments had to be halted manually. In these two experiments the first iteration chose *and* as the seed expansion, but the extraction chosen in the second iteration did not trigger the Stop Condition.

Using only a window-size of only one token creates too general extraction patterns. For example, the average number of extractions found in the first iteration is a lot higher than if a window-size of two tokens is used. For the document-sets of 2000 reviews an average of over 8000 extractions are found (for a window-size of two tokens about 100 extractions are found), for the document-sets this average is over 22000 extractions (for a window-size of two tokens about 200 extractions are found).

When we use a window-size of three tokens, the algorithm stops after only a few iterations due to a lack of usable extractions. The extraction patterns are too specific to keep the algorithm iterating.

That our bootstrapping algorithm does not start when a window-size of only one token is used was to be expected, there is only very little context to be used in the extraction patterns. This is especially true for the semantic category *Movie Title* where common punctuation characters, see Table 3.3, are often directly surrounding the movie title.

It is more of a surprise that our bootstrapping algorithm doesn't start when a window-size of three tokens is used. It will be possible to use a window-size larger than two tokens if the terms of the semantic category appear in a more structured context in the document-set, or if a larger document-set is used which makes it more likely to find extractions that reach both the seed- and pattern support requirement.

3.5.3 Extraction Scoring Function

Our extraction scoring function uses two requirements, the seed support and the pattern support requirement, on the extractions that might be too strict to

get the algorithm to start, but we expect to be essential to make sure that the algorithm does not diverge from the semantic category. To show the effect of removing these requirements we experiment with dropping one or two of these requirements.

3.5.3.1 Without the Seed Support requirement

Our bootstrapping algorithm uses a Seed Support requirement of two. Only extraction patterns that extract at least two terms of the seed lexicon are used to find extractions. This requirement is implicitly defined in our pattern scoring function, by the logarithm in the *RlogF* metric.

Dropping this requirement changes the extraction scoring function from Section 2.1.4 into:

$$ExtractionScore(E_i) = \begin{cases} \sum_{k=1}^{M_i} (\frac{F_k}{N_k} log_2(F_k + 1)) & \text{if } |M_i| > 1 \\ 0 & \text{otherwise} \end{cases}$$

For the four document-sets of size 2000, and the four of size 4000 used in our baseline experiments we start the algorithm, leading to a total of 8 experiments.

Document -set Size	**Average Iteration Reached**	**Avg. Prec.**
2000	2.5	0.63
4000	2	0.50

Table 3.7: Results for expanding the semantic category *Movie Title*, with the seed support requirement dropped

The results for these experiments are shown in Table 3.7. In all our experiments, the Stop Condition was triggered during the first iterations of the algorithm, because in each experiment the algorithm tried to expand the seed lexicon with too common terms.

Dropping the seed requirement makes the algorithm find an enormous amount of extraction patterns. The extraction patterns that extract more than one term of the seed lexicon are only very small part of them. Many of the extraction patterns that extract only one term of the seed lexicon are too specific and not representative of the context in which movie titles occur. They almost always apply to only one piece of text, the occurrence from which they have been made. If they also apply to another piece of text this is almost certainly an incorrect extraction.

Since our extraction scoring function uses the pattern scores of all extraction patterns, also these extraction patterns that are too specific, the effect of the extraction patterns that extract more than one term of the seed lexicon is diminished. This makes the extraction scoring function rank extractions mainly on how often they occur in the document-set.

These results show that a seed support requirement of two is necessary for the algorithm to start the bootstrapping process.

3.5.3.2 Without the Pattern Support requirement

The Pattern Support requirement of two, required that an extraction should be extracted by at least two extraction patterns to be considered as a seed expansion. Dropping this requirement changes the extraction scoring function from Section 2.1.4 into:

$$ExtractionScore(E_i) = \sum_{k=1}^{M_i}(\frac{F_k}{N_k}log_2(F_k))$$

For the four document-sets of size 2000, and the four of size 4000 used in our baseline experiments we start the algorithm, leading to a total of 8 experiments.

Document -set Size	% of the Experiments started	Average Iteration Reached	Avg. Prec.	Avg. Nr. of Unused Extractions	Avg. Prec. of Unused Extractions
2000	100	100	0.89	440	0.75
4000	100	100	0.83	1802.5	0.57

Table 3.8: Results for expanding the semantic category *Movie Title*, with the pattern support requirement dropped

In Table 3.8 we show the results for these experiments. In all of the experiments the algorithm started iterating, and all reached the maximum number of iterations of 100 to make. Comparing these results to the results obtained in our baseline experiments in Table 3.1 we see that there is a decrease of about 7 percent in the precision obtained over the first 100 iterations, or extractions.

The total number of usable extractions at iteration 100, an indication for the recall that can be achieved if the algorithm were allowed to continue after iteration 100, shows that dropping the pattern support requirement enlarges recall considerably, but also decreases precision. This drop in precision indicates that the algorithm is diverging from the semantic category that it is expanding.

Choosing which extraction scoring function to use depends on the task for which the bootstrapping algorithm is used. If the algorithm is used to autonomously expand a semantic lexicon the pattern support requirement should be used to guarantee high precision, but if the algorithm is used to expand a semantic lexicon that is reviewed by the user it might be useful to drop the requirement to obtain a higher recall.

3.5.3.3 Without the Seed- and the Pattern Support requirement

Initially we also wanted to conduct experiments in which both requirements are dropped. Dropping both requirements will lead to results that are worse than the experiments above, and dropping the seed support requirement already led to no results.

In conclusion our algorithm benefits from both requirements, the seed support requirement is an absolute requirement to start iterating and the pattern support requirement has a small positive effect on the precision.

3.6 Other Semantic Categories

In the previous section we have performed several experiments to determine the general behaviour of our algorithm. In all these experiments we let the algorithm expand the semantic category *Movie Title*. In this section we use our algorithm to find expansions for two other semantic categories, *Actor Name* and *Certification*. This will give us an indication of how our algorithm is applicable to other semantic categories. In these experiments we have used the same settings for the user definable parameters as used in our baseline experiments.

3.6.1 Actor Name

The terms that are used in the semantic category *Actor Name* have an enormous overlap with the terms used in the semantic category *Director* or *Writer*. Many persons working in the movie industry have held jobs as both an actor and a director. It will be interesting to see how our algorithm copes with semantic categories that overlap.

In Table 3.9 we show the results for our experiments on finding an expanded seed lexicon for the semantic category *Actor Name*. We initially experimented with a seed lexicon of 20 actor names, but since for that size the algorithm did not start iterating in any of the experiments we have also experimented with a seed lexicon of size 40 and 60. The three seed lexicons are taken from the list in Appendix A.2 with the seed lexicon of size 20 consisting of the first 20 actor names on that list, etcetera.

For each of these seed lexicons combined with the four document-sets of size 2000, and the four of size 4000 used in our baseline experiments we start the algorithm, leading to a total of 24 experiments.

The results show that none of the experiments using a seed size of 20 actor names started. This in contrast with the experiments we have done for the semantic category *Movie Title*. The average number of occurrences found in the document-sets used is about the same for the actor names and the movie titles in our seed lexicons. We think that actor names occur in less uniform contexts as movie titles, and that this fact causes it to be harder for the bootstrapping algorithm to find a good set of extraction patterns able to start expanding a seed lexicon of actor names than it is for movie titles.

None of the experiments were stopped due to the Stop Condition being triggered, they either stopped due to a lack of usable extractions or to the maximum number of iterations, $L = 100$, being reached.

We manually evaluated the resulting seed expansions. This was done by one annotator due to time constraints. An extraction is evaluated as being correct if we can manually locate the corresponding actor in the IMDB website and if he has been credited as an actor in at least one movie. A problem with this measure is that persons who are mostly a director, but have also acted in one our two movies (for example the director *George Lucas* played a small role in *Beverly Hills Cop III*). Although it is technically correct to see the extraction of such a person as correct, it is highly likely that it has been extracted from a context where it is used as a director. The precision found in our experiments should therefore be seen as an optimistic measure of the precision.

The precision found in our experiments is about 60 percent, which is considerably lower than the precision achieved for the semantic category *Movie Title*.

Seed Size	Document -set Size	% of the Experiments started	Average Iteration Reached	Avg. Prec.	Avg. Nr. of Unused Extractions	Avg. Prec. of Unused Extractions
20	2000	0	-	-	-	-
20	4000	0	-	-	-	-
40	2000	0	-	-	-	-
40	4000	25	100	0.65	129	0.57
60	2000	25	12	0.5	-	-
60	4000	75	100	0.61	150	0.59

Table 3.9: Results for expanding the semantic category *Actor Name*

Two reasons for this difference can be found by looking at the top 10 extraction patterns found in the final iteration of one of the experiments in Table 3.10:

- Since most of the incorrect extractions are person names, it appears that the algorithm has diverged into another semantic category containing person names. The patterns in Table 3.10 show that this has indeed happened, six out of ten of the patterns are used to extract directors, indicating that the algorithm has diverged into the semantic category *Director*.
- Although Table 3.10 shows the best extraction patterns, the four patterns that do not extract director are really specific. This shows us that the algorithm is not able to use more general extraction patterns, indicating that actor names indeed occur in less uniform contexts as movie titles. This has a negative effect on the precision achieved by our algorithm.

#	Extraction Pattern
p_1	`Directed by <EXTRACTION> . Screenplay`
p_2	`Director : <EXTRACTION> Writers :`
p_3	`Directed by <EXTRACTION> . Written`
p_4	`Directed by <EXTRACTION> . Rated`
p_5	`Cast : <EXTRACTION> , Alec`
p_6	`Actors : <EXTRACTION> as Jack`
p_7	`Starring : <EXTRACTION> , Gene`
p_8	`. Director <EXTRACTION> keeps the`
p_9	`directed by <EXTRACTION> , who`
p_{10}	`Cast : <EXTRACTION> , Billy`

Table 3.10: Best 10 extraction patterns for one of the experiments on *Actor Name*

These extraction patterns also hint to a possible place for improvements. Extraction pattern p_5, p_6, p_7 and p_{10} all use another actor name in the right context. This makes these patterns very specific, they are only able to make a few extractions. If we generalize the extraction patterns, i.e. by allowing the extraction patterns to use semantic categories as a feature, we are perhaps able to find extraction patterns that are more general, making the algorithm find more usable extractions. For example pattern p_5 can be generalized

to `Cast : <EXTRACTION> , <ACTOR NAME>`. In Chapter 4 we experiment with generalized patterns, see especially Subsection 4.2.2 where we experiment with expanding the semantic category *Actor Name* using extraction patterns where the semantic category *Actor Name* itself is used as a feature in the left and right part of the patterns.

3.6.2 Certification

Our bootstrapping algorithm is designed to expand semantic categories for which a large number of terms can be found. For such large semantic categories, using an algorithm for the semantic lexicon expansion task is useful, as finding all terms in a document-set is time consuming. Although for a small semantic category such as *Certification* there is no use in solving this task, as finding all terms can simply be done by looking through a few documents, we have chosen to apply our algorithm to this category.

These experiments are used to give us an indication of how our algorithm copes with semantic categories for which only a few terms exist. We are interested the precision and recall and in seeing if the algorithm stops after all terms are found or if it keeps iterating and diverges into other semantic categories.

Most of the reviews in our document-set are from the United States, and the rating system used in the United Stated currently uses five certifications (*G*, *PG*, *PG-13*, *R* and *NC-17*.

We have experimented with a seed lexicon of three of these five certifications (*PG*, *R* and *NC-17*) on the document-sets of size 125, 250, 500, 1000, 2000 and 4000 used in our baseline experiments. This leads to a total of 24 experiments. Since there only exist a few certifications we see an experiment as started to iterate if it reaches the first iteration.

Document -set Size	**% of the Experiments started**	**Average Iteration Reached**	**Avg. Prec.**	**Avg. Recall**
125	0	-	-	-
250	25	1	1	0
500	25	1.5	1	0.25
1000	100	2.5	1	0.75
2000	100	3.75	1	1
4000	100	5.75	0.93	1

Table 3.11: Results for expanding the semantic category *Certification*

In Table 3.11 we show for each document-set size which percentage of the experiments started iterating, which iteration was reached on average, the average precision and the average recall (total recall is achieved when both *G* and *PG-13* are extracted). The algorithm expands the seed lexicon with all five certifications used in the United Stated in all experiments on document-sets of 2000 or more reviews.

Table 3.12 shows the top 5 extraction patterns found in the final iteration of one of the experiments. Table 3.13 shows all the terms of the seed expansions that were found in our experiments. Only extraction e_6 and e_7 are incorrect

#	Extraction Pattern
p_1	`. Rated <EXTRACTION> . Running`
p_2	`Rating " <EXTRACTION> " Running`
p_3	`Rated : <EXTRACTION> ( Nudity`
p_4	`Classification : <EXTRACTION> ( Mature`
p_5	`RATED : <EXTRACTION> RELEASED :`

Table 3.12: Best 5 extraction patterns for one of the experiments on *Certification*

#	Extraction Text
e_1	pg-13
e_2	g
e_3	nr
e_4	no mpaa rating
e_5	not rated
e_6	pg-13 for violence
e_7	pg-13 . running time : approx
e_8	aa

Table 3.13: All terms used as a seed expansion for *Certification*

terms. Extraction e_3, e_4 and e_4 can be seen as correct, they are used to show that a movie has not been rated, extraction e_8 is a certification used by the MFCB, another motion picture rating system.

Although in each of the experiments the set of usable extractions was depleted before the algorithm started to diverge, and the algorithm stopped this should not be generalized as the behaviour to be expected when expanding any semantic category with a limited number of members. More experiments on semantic categories with a small number of terms are needed to show this.

Chapter 4

Lexical and Semantic Categories as Features

The extraction patterns that are created by our bootstrapping algorithm demand a specific left and right context. Doing this keeps our extraction patterns highly specific, the patterns have high precision but also low recall. As we use multiple extraction patterns to come up with a set of extractions, having low recall over a single extraction pattern is ok, but it might be possible to generalize extraction patterns in a way that enlarges their recall without decreasing their precision. We expect this to also increase the quality of the resulting expanded seed lexicon in terms of precision and recall. Using extraction patterns that are better tuned to the contexts in which the seed lexicon occurs will lead to higher precision, and enlarging the recall of the single extraction patterns will have a positive effect on the recall of the entire algorithm.

As an example of a pattern that can be generalized we encountered the pattern `Review... ' <EXTRACTION> ( 2000` while analysing one of our baseline experiments (see Table 3.3 in Section 3.3.3). This pattern uses a year in the second token of its right context. The generalized pattern `Review... ' <EXTRACTION> ( <YEAR>` that applies to all years instead of only to the year 2000 is likely to find more extractions while keeping the ratio between correct and incorrect extractions equal.

To create generalized extraction patterns, we allow our algorithm to make use of lexical- and semantic categories as a feature in the left and right context of an extraction pattern. The part of the extraction pattern where such a feature is used will match every element of that category instead of a specific token.

The lexical categories, and the elements belonging to them can be defined by the user. For example a lexical category can be defined to group together years, or to group together certain punctuation characters. In the same fashion, the semantic category that is being expanded, or another semantic category in the semantic lexicon, can be used to create generalized extraction patterns.

To implement the use of these features in our algorithm we have chosen to change how patterns (see Subsection 2.1.2) and extractions (see Subsection 2.1.3) are created. In addition to the set of non-generalized extraction patterns, we let our algorithm create a second set of extraction patterns, consisting of generalized extraction patterns.

The set of generalized extraction patterns is created from the set of non-generalized extraction patterns. For each non-generalized extraction pattern in which an element of one of the lexical- or semantic categories exists, a copy is made. This copy is then maximally generalized, each of the possible generalizations are made. For example if two lexical categories were defined, one that groups together the quotation characters ’ and ", and one that groups together years, next to the pattern `Review... ’ <EXTRACTION> ( 2000` the algorithm will now create the pattern `Review... <QUOTATION> <EXTRACTION> ( <YEAR>`.

Because we create our generalized extraction patterns next to our non-generalized extraction patterns, we in effect create two extraction patterns for each occurrence that can be generalized. In our extraction scoring function we used a pattern support requirement of 2 patterns. If we do not use a different extraction scoring function in our experiments with generalized patterns, every pattern that can be generalized will directly pass our pattern support requrement. So that we can compare the results for these experiments to those obtained in our baseline experiments, we have changed the pattern support requirement to 2 non-generalized extraction patterns.

Another Information Extraction algorithm that generalizes a set of extraction patterns is RAPIER (see Califf and Mooney [1998] and Califf [2003]). RAPIER randomly selects two extraction patterns and generalizes these into a new extraction pattern. If this extraction pattern is acceptable, the new pattern is added and the patterns subsumed by it are removed from the set of extraction patterns. To generalize two patterns, RAPIER first generalizes the middle part of the pattern, used to make extractions, and then specializes the left and right part until the pattern stops making incorrect extractions.

Although the approach of RAPIER is a better method of generalizing, we are more interested in seeing if generalizing patterns enables the algorithm to achieve better recall measures than we are in finding the best method of generalization. Apart from this motivation, RAPIER uses negative examples to find the right amount of generalization and assumes that all positive examples existing in the document-set are known. Since our algorithm works on unannotated document-sets and uses bootstrapping, it does not have negative examples and only a subset of the positive examples at its disposal, making it hard to use this approach.

4.1 Lexical Features

Our initial experiments with using features are focused on finding out the behaviour of our algorithm using lexical categories as features. We have used a list of four different lexical categories:

- *Quotation*, which matches ’ and ".
- *Punctuation*, which matches `.!?,;` and `:`
- *Bracket*, which matches `()[]{}<` and `>`
- *Year*, which matches all strings of 4 decimals between 1900 and 2006.

4.1.1 Movie Title

For the four document-sets of size 2000, and the four of size 4000 used in our baseline experiments we start the algorithm, leading to a total of 8 experiments.

Document -set Size	% of the Experiments started	Average Iteration Reached	Avg. Prec.	Avg. Nr. of Unused Extractions	Avg. Prec. of Unused Extractions
2000	100	100	0.94	51	0.91
4000	100	100	0.91	333	0.91

Table 4.1: Results for expanding the semantic category *Movie Title* using lexical categories as features

In Table 4.1 we show the results for these experiments. Compared to the results obtained for the baseline experiments in Table 3.1 we see no significant change.

This happens because we made a mistake in changing our pattern support requirement. We changed this requirement from requiring two extraction patterns into requiring two non-generalized extraction patterns. Although this makes it possible to compare our results to those obtained in the baseline experiments, the requirement makes it impossible for a generalized extraction pattern to be really used. For example if an extraction e was not extracted by p_α but was extracted by p_β and the generalized version of p_α the pattern support requirement is not reached, although we would like our algorithm to use extraction e. Generalized extraction patterns will therefore not lead to new extractions being found, the only effect they will have is an increase in the extraction score of extractions for which the pattern support requirement is met. Due to this mistake in changing our patterns support requirement we do not see the anticipated positive effect on the recall of our algorithm. Time-constraints unfortunately force us not to correct this mistake, as both the implementation process of changing the extraction scoring function and re-running the experiments in this chapter will take too much time.

#	Extraction Pattern
p_1	`<PUNCTUATION> <QUOTATION> <EXTRACTION> <QUOTATION> is`
p_2	`Review... <QUOTATION> <EXTRACTION> <BRACKET> <YEAR>`
p_3	`<PUNCTUATION> <QUOTATION> <EXTRACTION> <QUOTATION> does`
p_4	`<BRACKET> <QUOTATION> <EXTRACTION> <QUOTATION> does`
p_5	`of <QUOTATION> <EXTRACTION> <QUOTATION> <BRACKET>`

Table 4.2: A few generalized extraction patterns using Lexical Categories found for expanding *Movie Title*

We can however manually inspect the generalized extraction patterns to see if it is plausible that these extractions will have a positive effect on the recall. In Table 4.2 we show a few of the highest scored generalized extraction patterns found in these experiments. Ranking all extraction patterns by their pattern score, we find that the generalized version of an extraction pattern generally

appears slightly below the non-generalized version. This happens because a generalized extraction pattern makes more extractions than a non-generalized extraction pattern, and because the pattern score is mostly based on the ratio between terms in the seed lexicon and the total number of extractions. If the seed lexicon is small, it is likely that generalizing a pattern will make it find more new extractions unknown to the seed lexicon than new extractions known to the seed lexicon.

4.1.2 Actor Name

The experiments in expanding the semantic category *Actor Name* using Lexical Features in the generalized extraction patterns use the same seed lexicons and document-sets that were used in Subsection 3.6.1.

Seed Size	Document -set Size	% of the Experiments started	Average Iteration Reached	Avg. Prec.	Avg. Nr. of Unused Extractions	Avg. Prec. of Unused Extractions
20	2000	0	-	-	-	-
20	4000	0	-	-	-	-
40	2000	0	-	-	-	-
40	4000	0.25	100	0.65	130	0.57
60	2000	0.25	12	0.50	-	-
60	4000	0.75	100	0.64	149.67	0.59

Table 4.3: Results for expanding the semantic category *Actor Name* using lexical features

In Table 4.3 we show the results for these experiments. Compared to the results obtained for the baseline experiments in Table 3.9 we see no significant change. This happens for the same reason as described in Subsection 4.1.1. In Table 4.4 we show a few examples of generalized extraction patterns.

#	Extraction Pattern
p_1	`Director <PUNCTUATION> <EXTRACTION> Writers <PUNCTUATION>`
p_2	`Director <PUNCTUATION> <EXTRACTION> Starts <PUNCTUATION>`
p_3	`Starring <PUNCTUATION> <EXTRACTION> <PUNCTUATION> Gene`
p_4	`Thurman <PUNCTUATION> <EXTRACTION> <PUNCTUATION> Tim`
p_5	`Tyler <PUNCTUATION> <EXTRACTION> <PUNCTUATION> Will`

Table 4.4: A few generalized extraction patterns using Lexical Categories found for expanding *Actor Name*

4.2 Using Semantic Categories as Features

Semantic categories can also be used as a feature in the extraction patterns. In this section we use the semantic category that the algorithm is expanding as a feature. If a term of the seed lexicon occurs in the left or right context of an

occurrence, the algorithm creates a generalized extraction pattern that can be applied to each term of the seed lexicon. In later iterations these generalized extraction patterns are likely able to make more extractions than in earlier iterations, since having more terms in the seed lexicon makes these patterns more general.

Especially for the semantic category *Actor Name* we think that using semantic categories as a feature can lead to improvements. In Subsection 3.6.1 we noticed that the extraction patterns used in the experiments for the semantic category *Actor names* often used the name of another actor in the left and/or right context. This happens because in our document-sets actor names often occur in lists (for example *... Starring Tobey Maguire, Reese Witherspoon*). Generalized patterns that use *Actor Name* as a feature can be helpful to our algorithm, as the order and elements on these lists vary.

4.2.1 Movie Title

For the four document-sets of size 2000, and the four of size 4000 used in our baseline experiments we start the algorithm, leading to a total of 8 experiments.

Document -set Size	**% of the Experiments started**	**Average Iteration Reached**	**Avg. Prec.**	**Avg. Nr. of Unused Extractions**	**Avg. Prec. of Unused Extractions**
2000	100	100	0.94	45	0.94
4000	100	100	0.91	344.25	0.90

Table 4.5: Results for expanding the semantic category *Movie Title* using *Movie Title* as a Feature

As in Subsection 4.1.1 we find no significant change compared to the results obtained for the baseline experiments in Table 3.1. This happens for the same reason as described in that section. Where in the results where Lexical Categories where used as a feature generalized extraction patterns occurred in each of the experiments, here we find only a few in only one of the experiments. Apparently movie titles do not often occur in close context to each other in our document-sets. In Table 4.6 we show the three generalized extraction patterns found in this single experiment.

#	**Extraction Pattern**
p_1	`<SEMANTIC_CATEGORY> , <EXTRACTION> and <SEMANTIC_CATEGORY>`
p_2	`<SEMANTIC_CATEGORY> , <EXTRACTION> , and`
p_3	`<SEMANTIC_CATEGORY> , <EXTRACTION> , The`

Table 4.6: The three generalized extraction patterns found for expanding *Movie Title* while using the semantic category *Movie Title* as a feature

4.2.2 Actor Name

The experiments in expanding the semantic category *Actor Name* using Lexical Features in the generalized extraction patterns use the same seed lexicons and document-sets that were used in Subsection 3.6.1.

Seed Size	Document -set Size	% of the Experiments started	Average Iteration Reached	Avg. Prec.	Avg. Nr. of Unused Extractions	Avg. Prec. of Unused Extractions
20	2000	0	-	-	-	-
20	4000	0	-	-	-	-
40	2000	0	-	-	-	-
40	4000	0.25	100	0.64	150	0.58
60	2000	0.25	12	0.5	-	-
60	4000	0.75	100	0.62	155.33	0.59

Table 4.7: Results for expanding the semantic category *Actor Name* using *Actor Name* as a Feature

Compared to the results obtained for the baseline experiments in Table 3.9, the results in Table 4.7 show no significant change. This happens for the same reason as described in Subsection 4.1.1.

In Table 4.8 we show a few examples of generalized extraction patterns. In contrast to the experiments the semantic category *Movie Title* was used as a feature, here we find generalized extraction patterns in each of the experiments. Combined these generalized extraction patterns make it possible to extract the actor names in lists like *Starring: Joseph Gordon-Levitt, Danny Glover, Christopher Lloyd, Brenda Fricker, and Tony Danza.* Pattern p_5 is an example of a generalized extraction pattern that will be generalized into p_3 once the actress, probably *Liv Tyler*, gets used as a seed expansion.

#	Extraction Pattern
p_1	`Cast : <EXTRACTION> , <SEMANTIC_CATEGORY>`
p_2	`Starring : <EXTRACTION> , <SEMANTIC_CATEGORY>`
p_3	`<SEMANTIC_CATEGORY> , <EXTRACTION> , <SEMANTIC_CATEGORY>`
p_4	`<SEMANTIC_CATEGORY> , <EXTRACTION> , and`
p_5	`<SEMANTIC_CATEGORY> , <EXTRACTION> , Liv`

Table 4.8: A few generalized extraction patterns found for expanding Actor Name while using the semantic category Actor Name as a feature

Instead of using the semantic category that is expanded as a feature, another possibility would be to use a different semantic category as a feature. If the algorithm is first used to expand a seed lexicon for one semantic category, the expanded seed lexicon can be used as a feature while expanding another semantic category. If the terms of the two semantic categories appear close together in the document-set, the algorithm will find generalized extraction patterns. We have chosen not to experiment with this, as the mistake made in designing the extraction scoring function will surely make these experiments show no significant change compared to the baseline experiments.

In none of the experiments using lexical- or semantic categories as features were we able to show a significant change in precision or recall compared to the results found in our baseline experiments. Although, especially when using the semantic category *Actor Name* as a feature while expanding a seed lexicon of actor names, the algorithm is able to find interesting and promising generalizations a mistake in designing the extraction scoring function makes it impossible for these generalizations to be of real use.

Chapter 5

Discussion & Conclusion

In the previous chapters we have introduced and evaluated our bootstrapping algorithm for semantic lexicon expansion. Our bootstrapping algorithm differs from the other bootstrapping algorithm described in Section 1.5 in that it uses simple extraction patterns, which only make use of the syntax in a small context window. Our algorithm does not use named-entity recognition or other sentence analysation. This keeps our algorithm language independent and applicable to semi-structured texts. Instead of using only a small number of extraction patterns, our algorithm uses the entire set of extraction patterns to find the expanded seed lexicon.

To be able to do this we have introduced an extraction scoring function that is able to use the entire set of extraction patterns. Our extraction scoring function is based on the proven pattern scoring function used by both Riloff and Jones [1999] and Thelen and Riloff [2002].

Our research describes the implicit seed support requirement the pattern scoring function of the algorithm, and introduces a pattern support requirement. Instead of using a stop list of common words that are not to be used in the seed expansion, our algorithm uses a stop condition.

With the results of our experiments on expanding the semantic category *Movie Title* and *Actor Name* we have given further evidence that it is possible to perform corpus-based Information Extraction tasks, such as semantic lexicon expansion, using a document-set consisting solely of unannotated documents. The use of bootstrapping allows our algorithm to find more extraction patterns and extractions than that are possible to find if only a single iteration is made.

Our algorithm is able to reach a precision of about 90 percent for expanding the semantic category *Movie Title* on a document-set of movie reviews. For the semantic category *Actor Name* we find a precision of about 60 percent. This difference can be explained by movie titles occurring in more structured contexts than actor names, which makes the precision of our algorithm dependent on the amount of structure in the contexts around the terms of the semantic category that is expanded.

The recall of our algorithm is relatively low. Using the default values for the user definable parameters, we were able to recall only about 475 different movie titles whereas we found that there already appear over 850 different movie titles in a document-set of 250 reviews. We were unable to recall a larger amount of movie titles by tweaking our algorithm, the default values are the optimal

settings for our document-set. Due to being time-constrained we were only able to let our algorithm make 100 iterations in many of our experiments. For these experiments we had to resort to an estimation of the number of correct terms in the seed expansion as a rather crude indication of the recall. As with precision, we also found a lower recall for actor names than for movie titles.

By applying our bootstrapping algorithm to increasing sizes of seed lexicons we found that there exists a minimum amount of terms needed to start iterating. After this minimum number is reached, there is no effect on the precision and only a slight positive effect on the recall in enlarging the seed lexicon. This effect was shown for both the semantic category *Movie Title* and *Actor Name*. The effect on the recall can be explained by the new terms in the seed lexicon leading to extraction patterns that otherwise could not have been found. The stark difference in behaviour of our algorithm between using a seed lexicon just below the minimum size and a seed lexicon at the minimum size can be explained by the use of our pattern support requirement. For too small seed lexicons, this requirement makes it impossible to find enough reliable extractions to expand the seed lexicon.

It is not possible to determine the minimum amount of terms in the seed lexicon needed to get the algorithm to start. The number depends on the amount of redundancy in both those the terms of the semantic category that is to be expanded, the context in which those terms occur and on which exact terms are chosen in the seed lexicon. The minimum amount of terms can be decreased by using prototypical examples of the semantic category (which are more likely to occur in a document-set) or by using a larger document-set.

Encapsulating our algorithm in a loop that asks a user to keep adding seeds to the seed lexicon until the minimum size is reached, allows a user to expand a semantic lexicon with as little effort as possible. We can do this because we are able to tell if the minimum size is reached or not when running our algorithm, and since there is no effect on the precision, and only a small effect on the recall in expanding the seed lexicon after this point.

The document-set also has to have a minimum-size for the algorithm to start iterating. In our experiments we found that for our experiments on expanding the semantic category *Movie Title* we had to use at least a document-set of 1000 movie reviews. Increasing the size of the document-set has no effect on the precision of the resulting expanded seed lexicon. There is a strong positive effect on the recall that appears to be linear in our experiments. We anticipate however that this effect starts to slow down for larger document-sets, in larger document-sets it becomes less likely for each document to contain a movie title, or actor name that was not yet encountered in previous documents. The implementation of our algorithm however makes it too time consuming to experiment with our algorithm on significantly larger document-sets.

Changing the size of the context window used by our extraction patterns has an enormous effect. Only for a window-size of two tokens is our algorithm able to find an expanded seed lexicon.

Diverging from the semantic category that is expanded can happen with our algorithm. We have seen that our Stop Condition is able to stop the algorithm when it starts to diverge due to too common terms, but diverging into an overlapping semantic category will occur as we saw for the semantic category *Actor Name* and its overlapping category *Director*.

The problem of only having incorrect extractions left to use as the seed

expansion, described in Subsection 2.1.5, did not occur in our experiments. There is no real decrease in precision in later iterations, which is probably due to our pattern support requirement. This requirement is so strict that there are simply not that many incorrect extractions in the lower regions of the extraction score. The algorithm either stops due to being the extractions being depleted or due to the Stop Condition being triggered.

Being able to stop iterating when all the terms in the semantic category are found was shown for the semantic category *Certification*, but further research has to show how general this behaviour is.

If we drop the seed support requirement from our extraction scoring function, the algorithm immediately diverges and is stopped by our Stop Condition. The pattern support requirement can be dropped to get an increase in recall at the cost of a decrease in precision.

In Chapter 4 we experimented with generalizing our extraction patterns. Although, due to a faulty designed pattern support requirement, we were unable to find the expected higher recall or any other significant results, the generalized extraction patterns that were found do however show promise for future research.

The document-set of movie reviews used in our experiments has, to our knowledge, not been used in other research on expanding semantic lexicons. Comparing our algorithm to the results of other algorithms has not been done. We see it as future research to make this comparison, either by applying our algorithm to the document-sets used in other research, or by implementing other algorithms and applying them to our set of movie reviews.

5.1 Further Research

Currently there is no generic method for keeping the algorithm from diverging, only a Stop Condition is used to keep it from diverging into adding too common terms. This Stop Condition can be improved by letting the algorithm skip over too common terms instead of stopping the algorithm if too common terms are selected as the seed expansion. Since this allows the algorithm to make more iterations, it will possibly lead to a higher recall. To implement this using the Stop Condition currently used by our algorithm, the algorithm should be allowed to backtrack O iterations, and disallowed to use the too common terms found by our Stop Condition in the seed expansion.

Another possibility is to look at the occurrences and patterns that will be found in the next iteration, if a certain extraction is used as the seed expansion. Doing this allows the algorithm to skip extractions for which too many occurrences and patterns will be found, an indication for seed expansion that will make the algorithm diverge into adding too common terms.

If this is done for all extractions found by the algorithm, it can be incorporated into the extraction scoring function and used to keep the algorithm from diverging into overlapping semantic categories. The extraction scoring should be modified to give a lower score to more ambiguous terms. A possible way of doing this is to look at the extraction patterns found for the extraction, a more ambiguous extraction will lead to extraction patterns that have less overlap with the extraction patterns found up to that point. Doing this will significantly decrease the speed of our algorithm.

In our research we used the default value for M of using one extraction as the seed expansion. If increasing M does not hamper the quality of the resulting expanded seed lexicon, it can be used to get the same recall using less iterations, and speed up our algorithm. A better option would be to use a threshold on the extraction score instead of a constant number to determine which extractions should be used as the seed expansion.

Our algorithm is currently able to apply two extraction patterns to the same tokens. For example `. Starring <EXTRACTION> , and` and `Starring : <EXTRACTION> , and`, applying this pattern to `. Starring : Ben Stiller, and` would let the former pattern extract `: Ben Stiller` while the latter extracts `Ben Stiller`. Only allowing the more specific pattern, in this case the latter pattern since it extracts less tokens, to make the extraction might improve the quality of the expanded seed lexicon. The implementation of our algorithm does not allow us to easily implement this.

Our extraction patterns make extractions up to K tokens, a user definable number. Perhaps this number can be set automatically by setting K to one more than the number of tokens of the largest term in the seed lexicon of the iteration. This technique is employed by RAPIER (see Califf and Mooney [1998] and Califf [2003]).

In our experiments on generalized extraction patterns we were only able to show that the algorithm indeed found generalized extraction patterns that used lexical- and semantic categories as features. Further research on changing the extraction scoring function to let the algorithm make use of these generalized extraction patterns needs to be done to see if generalizing extraction patterns has a positive effect on the quality of the expanded seed lexicon.

The evaluation method used in our experiments had to make use of external lists of correct terms to calculate precision and recall. We had to do this since the document-set of movie reviews was unannotated. If our document-set was an annotated document-set, we could run our algorithm on the document-set with the annotation layer removed, and use the annotation layer to evaluate the algorithm. This makes it possible to create lists of correct terms, that only contain terms if they are really used in the document-set in the context of the semantic category, and will give use more correct measures of precision and recall. The annotation layer can also help in evaluating the extraction patterns found-, and pattern scoring function used by our algorithm, since it tells us where terms are used in the context of our semantic category and where they are used in a different context.

Appendix A

Master Seed Lexicons

A.1 Movie Title

All experiments that expand the semantic category *Movie Title* start with a semantic lexicon that is a subset of the list in Table A.1 below. The movies on this list are chosen on the fact that they are blockbusters from the years in which most of the reviews in our document-set have been written.

To create this selection of movie titles we have used the IMDB movie pages to which most of the reviews are linked. First we sorted all those linked IMDB movie pages in decreasing number of reviews to find out the movies that were reviewed the most. The best 40 movie titles were then sorted according to decreasing number of occurrences (in the movie review set of 40,000 reviews), after which the best 20 movie titles were used for the master seed lexicon. This was done to remove movie titles are written in many different ways (such as *star wars : episode 1 - the phantom menace*).

This method of selecting a seed lexicon leads to a seed lexicon of often occurring movie titles, just as would probably happen if a user is asked to create a seed lexicon. A user will most likely come up with prototypical examples of movie titles, which will often occur in our document-set.

A.2 Actor Names

The list of 60 actor names in Table A.2 is created by taking all the actor names in our annotated movie review document-set. These actor names were then sorted according to decreasing number of occurrences (in the movie review set of 40,000 reviews) after which the top 60 were used.

s_1	contact
s_2	the game
s_3	godzilla
s_4	titanic
s_5	the matrix
s_6	armageddon
s_7	saving private ryan
s_8	fight club
s_9	the truman show
s_{10}	american beauty
s_{11}	the sixth sense
s_{12}	the blair witch project
s_{13}	men in black
s_{14}	the mummy
s_{15}	deep impact
s_{16}	starship troopers
s_{17}	dark city
s_{18}	eyes wide shut
s_{19}	lost in space
s_{20}	l.a . confidential

Table A.1: Master Seed Lexicon for the *Movie Title* category

s_1	bruce willis	s_{21}	george clooney	s_{41}	nicole kidman
s_2	tom hanks	s_{22}	adam sandler	s_{42}	tim robbins
s_3	robin williams	s_{23}	robert de niro	s_{43}	kevin costner
s_4	jim carrey	s_{24}	james woods	s_{44}	spike lee
s_5	julia roberts	s_{25}	bob thornton	s_{45}	liam neeson
s_6	tom cruise	s_{26}	sean connery	s_{46}	john goodman
s_7	ben affleck	s_{27}	morgan freeman	s_{47}	dustin hoffman
s_8	samuel l . jackson	s_{28}	billy bob thornton	s_{48}	bill paxton
s_9	anthony hopkins	s_{29}	meg ryan	s_{49}	ewan mcgregor
s_{10}	harrison ford	s_{30}	jack nicholson	s_{50}	ed harris
s_{11}	eddie murphy	s_{31}	sandra bullock	s_{51}	uma thurman
s_{12}	kevin spacey	s_{32}	al pacino	s_{52}	hugh grant
s_{13}	denzel washington	s_{33}	gwyneth paltrow	s_{53}	mike myers
s_{14}	matt damon	s_{34}	drew barrymore	s_{54}	sharon stone
s_{15}	john cusack	s_{35}	steve buscemi	s_{55}	russell crowe
s_{16}	keanu reeves	s_{36}	robert duvall	s_{56}	richard gere
s_{17}	johnny depp	s_{37}	william h . macy	s_{57}	harvey keitel
s_{18}	gene hackman	s_{38}	cameron diaz	s_{58}	winona ryder
s_{19}	christopher walken	s_{39}	danny devito	s_{59}	kenneth branagh
s_{20}	nicolas cage	s_{40}	edward norton	s_{60}	geoffrey rush

Table A.2: Master Seed Lexicon for the *Actor Name* category

Appendix B

Extractions evaluated as incorrect

1	2001	30	it was always me
2	3,000 miles to graceland	31	jade scorpion
3	48 hrs.	32	kids in the hall
4	a bug's life	33	little boy blue,
5	a.i.	34	manny and lo
6	a.i. : artificial intelligence	35	no man's land
7	arthur 2	36	nutty professor ii
8	big momma's house	37	o brother , where art thou?
9	blade 2	38	penn & teller : bullshit!
10	blood simple	39	peter's friends
11	bridget jones's diary	40	ricidule
12	charlie's angels	41	schindler's list
13	charlie's angels : full throttle	42	scooby-doo 2 : monsters unleashed
14	child's play	43	she's all that
15	city	44	shiloh 2
16	day after	45	smell of camphor
17	disney's the kid	46	starsky and hutch
18	dungeons & dragons,	47	the black cat,
19	felicia's journey	48	the blue dog
20	forces	49	the godfather part ii
21	geniuses	50	the imposters
22	goodbye , lover	51	the razor's edge
23	happy texas	52	the x-files : fight the future
24	harrison's flowers	53	tupac resurrection
25	harry potter and the sorcerer's stone	54	ulfc
26	hart's war	55	what?
27	hey arnold! the movie	56	who killed atlanta's children?
28	hfd	57	win a date with tad hamilton
29	i married a strange person!	58	your friends and neighbors

Table B.1: Extractions evaluated as incorrect extractions

Bibliography

E. Agichtein and L. Gravano. Snowball: Extracting relations from large plain-text collections. In *Proceedings of the Fifth ACM International Conference on Digital Libraries*, 2000. 1.5.3.2

A. Blum and T. Mitchell. Combining labeled and unlabeled data with co-training. In *Proceedings of the 11th Annual Conference on Computational Learning Theory*, pages 92–100, 1998. 1.3

S. Brin. Extracting patterns and relations from the world wide web. *WebDB Workshop at 6th International Conference on Extending Database Technology, EDBT'98*, 1998. 1.5.3.1, 1.3, 1.5.3.2

M. Califf. Bottom-up relational learning of pattern matching rules for information extraction. *Journal of Machine Learning Research*, 4:177–210, 2003. 4, 5.1

M. E. Califf and R. J. Mooney. Relational learning of pattern-match rules for information extraction. In *Working Notes of AAAI Spring Symposium on Applying Machine Learning to Discourse Processing*, pages 6–11, Menlo Park, CA, 1998. AAAI Press. 4, 5.1

V. de Boer, M. van Someren, and B. Wielinga. A redundancy-based method for the extraction of relation instances from the web. *International Journal of Human-Computer Studies*, 65(9):816–831, 2007. 1.5.3

G. Geleijnse and J. Korst. Learning effective surface text patterns for information extraction. In *2006 workshop on Adaptive Text Extraction and Mining*, 2006. 1.5.3

IMDB. The internet movie database (imdb). URL `http://www.imdb.com`. 3.1

IMDB-Reviews. Imdb's archive for the rec.arts.movies.reviews newsgroup. URL `http://www.imdb.com/Reviews`. 3.1

N. Kushmerick and B. Thomas. Core technologies for information agents. Technical report, Universität Koblenz-Landau, 2003. 1.3

B. Pang, L. Lee, and S. Vaithyanathan. Thumbs up? Sentiment classification using machine learning techniques. In *Proceedings of the 2002 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 79–86, 2002. 3.1

E. Riloff. Automatically constructing a dictionary for information extraction tasks. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, pages 811–816. AAAI Press / MIT Press, 1993. 1.5.1

E. Riloff. Automatically generating extraction patterns from untagged text. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, pages 1044–1049, 1996. 1

E. Riloff and R. Jones. Learning dictionaries for information extraction by multi-level bootstrapping. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence*, 1999. 1.5.1, 2.1.4, 5

Swish-e. Swish-e: Simple web indexing system for humans - enhanced. URL `http://swish-e.org/`. 2.1.3

M. Thelen. Simultaneous generation of domain-specific lexicons for multiple semantic categories. Master's thesis, University of Utah School of Computing, 2002. 1.5.2

M. Thelen and E. Riloff. A bootstrapping method for learning semantic lexicons using extraction pattern contexts. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 214–221. Association for Computational Linguistics, July 2002. 1.5.2, 5