Scaling up Pattern Induction for Web Relation Extraction through Frequent Itemset Mining

Sebastian Blohm and Philipp Cimiano

Institute AIFB, Universität Karlsruhe (TH)

Abstract. In this paper, we address the problem of extracting relational information from the Web at a large scale. In particular we present a bootstrapping approach to relation extraction which starts with a few seed tuples of the target relation and induces patterns which can be used to extract further tuples. Our contribution in this paper lies in the formulation of the pattern induction task as a well-known machine learning problem, i.e. the one of determining frequent itemsets on the basis of a set of transactions representing patterns. The formulation of the extraction problem as the task of mining frequent itemsets is not only elegant, but also speeds up the pattern induction step considerably with respect to previous implementations of the bootstrapping procedure. We evaluate our approach in terms of standard measures with respect to seven datasets of varying size and complexity. In particular, by analyzing the extraction rate (extracted tuples per time) we show that our approach reduces the pattern induction complexity from quadratic to linear (in the size of the occurrences to be generalized), while mantaining extraction quality at similar (or even marginally better) levels.

1 Introduction

A problem which has received much attention in the last years is the extraction of (binary) relations from the Web. Automatic extraction of relations is useful whenever the amount of text to analyze is not manageable manually. As an example, a car manufacturer may want to monitor upcoming market developments by analyzing news and blogs on the Web. Relation extraction can extract the *presentedAt* relation in order to compile a list of upcoming car models and where they will be presented (e.g *presentedAt(Audi Q7, Detroit Motor Show)*). To address this problem, several supervised approaches have been examined which induce a classifier from training data and then apply it to discover new examples of the relation in question. These approaches typically work on a closed corpus and rely on positive and (implicit) negative examples provided in the form of annotations [18, 8] or a handful of positive and negative examples [5]. The obvious drawback of such methods is that they can inherently not scale to the Web as they would require the application of the classifier to the whole textual data on the Web, thus being linear in its size.

Alternative approaches to address the problem of extracting relations from the Web have been presented (we discuss a couple of systems below). These approaches rely on the induction of patterns on the basis of occurrences of a few examples of the relation in question. Such explicit textual patterns allow to take a shortcut to linearly scanning the whole Web by relying on standard index structures to evaluate the string patterns

as standard search engine queries using off-the-shelf search engine APIs. This circumvents the need to linearly process the whole Web (see e.g. [3]). Some approaches perform pattern induction in an iterative fashion in a cyclic approach which uses the new examples derived in one iteration for the induction of new patterns in the next iteration [4, 1]. In this paper we follow this latter approach and in particular examine more in detail the empirical complexity of the pattern induction step. As in these approaches the induction of patterns proceeds in a bootstrapping-like fashion, the complexity of the pattern induction step crucially determines the time complexity of the whole approach. Earlier implementations of the approach have used greedy strategies for the pairwise comparison of the occurrences of seed examples. In this paper we show how the Apriori algorithm for discovering frequent itemsets can be used to derive patterns with a minimal support in linear time. Our empirical evaluation shows that with this approach pattern induction can be reduced to linear time while maintaining extraction quality comparable (and even marginally better) to earlier implementations of the algorithm.

The remainder of this paper is organized as follows. In the next section we describe the approach of pattern-based relation extraction using Web search engines in more detail. In section *Pattern Induction as Frequent Itemset Mining*, we give a brief introduction to Frequent Itemset Mining before describing how it is applied in order to induce patterns for relation extraction. We describe our experimental results in section *Experimental Results*, before discussing related work and giving some concluding remarks.

2 Iterative Pattern Induction

The goal of pattern induction is, given a set of seed examples (pairs) S of a relation R as well as occurrences $Occ(S)$ in the corpus (the Web in our case) of these seeds, to induce a set of patterns P which are general enough to extract many more tuples standing in the relation R (thus having a good coverage) and which at the same time do not overgenerate in the sense that they produce too many spurious examples. The challenging issues here are on the one hand that the hypothesis space is huge, corresponding to the power set of the set of possible patterns P representing abstractions over the set of occurrences $Occ(S)$. We will denote this hypothesis space as 2^P. On the other hand, the complete extension ext_R of the relation R is unknown (it is the goal of the whole approach to approximate this extension as closely as possible at the end of the cycle), such that we cannot use it to compute an objective function: $o : 2^P \rightarrow \mathbb{R}$ to determine the patterns' accuracy with respect to the extension ext_R.

The general algorithm for iterative induction of patterns is presented in Figure 1. It subsumes many of the approaches mentioned in the introduction which implement similar bootstrapping-like procedures. The key idea is to co-evolve P (which at the beginning is assumed to be empty) as well as a constantly growing set of examples S which at the beginning corresponds to the seed examples. The candidate patterns can be generated in a greedy fashion by abstracting over the occurrences $Occ(S)$. Abstracting requires finding common properties, which in principle is a quadratic task as it requires pairwise comparison between the different occurrences.

```
ITERATIVE PATTERN INDUCTION(PatternsP', TuplesS')
 1  S ← S'
 2  P ← P'
 3  while not DONE
 4  do Occ_t ← MATCH-TUPLES(S)
 5     P ← P ∪ LEARN-PATTERNS(Occ_t)
 6     EVALUATE-PATTERNS(P)
 7     P ← {p ∈ P | PATTERN-FILTER-CONDITION(p)}
 8     Occ_p ← MATCH-PATTERNS(P)
 9     S ← S + EXTRACT-TUPLES(Occ_p)
10     EVALUATE-TUPLES(S)
11     S ← {t ∈ S | TUPLE-FILTER-CONDITION(t)}
```

Fig. 1. Iterative pattern induction algorithm starting with initial tuples S' or (alternatively) patterns P'.

The algorithm starts with a set of initial tuples S' of the relation in question – so called *seeds* – and loops over a procedure which starts by acquiring occurrences of the tuples currently in S (e.g. by querying a search engine with `"Stockholm" "Sweden"`) for the relation $locatedIn$. Further patterns are then learned by abstracting over the text occurrences of the tuples. The new patterns are then evaluated and filtered before they are matched. A resulting pattern could be "flights to ARG_1 , ARG_2 from $*$ airport" and thus may contain wildcards and argument place holders. From these matches, new tuples are extracted, evaluated and filtered. The process is repeated until a termination condition DONE is fulfilled. The learning is thus inductive in nature, abstracting over individual positive examples in a bottom-up manner.

For our experiments we have used the implementation of the above algorithm as described in [3]. They have shown in previous work that in absence of an objective function to maximize, we can reasonably estimate the quality of the set P of patterns by a heuristic function. Among the different functions examined in the above mentioned work, a simple function which assesses the quality of a pattern on the basis of its support, i.e. the different occurrences which it was generated from and therefore covers, is shown to be a good choice compared to other more elaborate measures such as the pointwise mutual information used in the Espresso [12] and other systems (e.g. KnowItAll [9]). Therefore, a reasonable choice is to select those patterns which have a minimal support and meet some heuristic syntactic criteria to prevent too general patterns[1]. We describe in the following section how this problem can be formulated as the one of determining frequent itemsets using the well-known apriori algorithm. With this move, we also reduce the complexity of the pattern induction step from quadratic to linear in the number of occurrences.

3 Pattern Induction as Frequent Itemset Mining

In our approach, we translate textual occurrences of a certain relation into set representations and use the Apriori algorithm to find patterns in these occurrences that exceed a certain minimum support. This task is typically called *frequent itemset mining* (FIM).

[1] In particular, we ensure that the patterns have a minimal number of token constraints (and not only wildcards) as well as that they have been generated from at least two different tuples.

The mining for frequent itemsets is a subtask of Association Rule Mining. Association rules are used to derive statements like "Clients who bought product X also bought product Y" from transaction databases. A transaction $t \in DB$ constitutes a process with several items a from an alphabet of items A (e.g. products that have been jointly purchased). DB is thus a (multi) set of subsets of A.

In a database DB of transactions the frequent itemsets $F \subset 2^A$ are defined as those sets that occur at least $freq_{min}$ times as subset of a transaction, i.e. $F = \{f \in 2^A || \{t \in DB | f \subset t\}| \geq freq_{min}\}$.

3.1 The Apriori Algorithm

Apriori [2] is an algorithm for finding all frequent itemsets given a database and a frequency threshold. It is based on the observation that an itemset f of size $|f| = n$ can only be frequent in DB if all its subsets are also frequent in DB. Apriori thus significantly reduces the amount of itemsets for which the frequency has to be counted by first deriving all frequent itemsets of size $n = 1$ and then progressively increasing n so that the above subset condition can be checked when generating the candidates for $n + 1$ as all subsets of size n are known. The Apriori algorithm looks as follows in pseudocode:

```
APRIORI(Alphabet A, Database DB ⊂ 2^A, Threshold freq_min)
1  C ← {{a}|a ∈ A}
2  n ← 1
3  while C ≠ ∅
4  do
5     ∀c ∈ C : COUNTSUPPORT(c, DB)
6     F_n ← {c ∈ C|SUPPORT(c) >= freq_min}
7     C ← {f ∪ g|f, g ∈ F_n ∧ MERGABLE(f, g)}
8     C ← PRUNE(C, F_n)
9     n ← n + 1
```

The algorithm stores all frequent itemsets of size n in a set F_n after verifying for each itemset that it occurrs at least $freq_{min}$ times in DB. The set of candidates for the first iteration is given by all elements of the alphabet. For the following iterations it is then generated by taking all elements of F_n and combining them if the condition MERGABLE(f, g) is fulfilled, which makes sure that f and g overlap in $n - 1$ elements. PRUNE(C, F_n) removes all itemsets c from C (which all have length $n + 1$) for which one or more of all possible subsets of c of size n are not contained in F_n which is the above-mentioned necessary condition for c to be frequent.

The performance of the Apriori algorithm depends on the efficient implementation of the operations COUNTSUPPORT(c, DB), MERGABLE(f, g) and PRUNE(C, F_n). It is common to use a Trie data structure (also called Prefix Tree) for this purpose. Given an arbitrary total order on A, one can represent the itemsets as ordered sequences with respect to that sequence. Tries are trees that represent sequences as paths in the tree along with their frequency counts. After constructing a Trie from the DB, one can find and count non-continuous subsequences of DB entries very efficiently, which is the task of COUNTSUPPORT. Similarly, MERGABLE and PRUNE can be implemented as traversal operations on the Trie (as described in [11]).

3.2 Mining for Text Patterns with Apriori

The general idea of applying frequent itemset mining for text pattern induction is that a text pattern `"flights to *, *"` can be considered the frequent itemset of the set of text occurrences it has been generated from (e.g. DB = {"We offer flights to London, England.","I look for flights to Palo Alto, CA."}). In order to ensure that, in spite of the set character of itemsets, word order is preserved, a special encoding is used, allowing at the same time to express additional constraints over words. While sequence mining algorithms such as the one used by Jindal and Liu [10] can be applied, it is not straightforward to encode multiple constraints per token. Thus, in our approach we exploit the more general model of unordered itemsets and encode word order and other constraints as described below.

We use the notion of constraints for describing the textual occurrences and patterns. Each constraint has a type, a position and a value. A constraint is fulfilled for a given text segment if the value is present at the given position in a way described by the constraint type. The positions are the token numbers (aligned by the positions of the arguments). Types can be for example surface string, capitalization and part-of-speech with their obvious sets of possible values. The pattern `"We offer flights to *, *"` may be represented as the following set of constraints:

$$surface_1 = \text{we}, capitalization_1 = true$$
$$surface_2 = \text{offer}, capitalization_2 = false$$
$$surface_3 = \text{flights}, capitalization_3 = false$$
$$surface_4 = \text{to}, capitalization_4 = false$$
$$surface_6 = \text{COMMA}, capitalization_6 = false$$

Note that no constraints are posed for positions 5 and 7 because those are the argument positions (reflected by the $*$ wildcard above). In our implementation we ensure that all occurrences are aligned such that the position numbers are always the same relative to the argument positions.

We encode each constraint as a positive integer value using a bijective function $encode : Type \times Position \times Value \rightarrow \mathbb{N}$: $encode(con, pos, value) = value * maxCon * maxPos + (pos + maxPos * (con - 1))$. where con is the number of the constraint type, pos the position and $value$ a numerical value reflecting frequency. The remaining variables reflect the respective maximal values with respect to the given database. One can think of this as the process of first "flattening" the structured information contained in the constraints to items like:

$$\{surface\_1\_we, capitalization\_1\_true,$$
$$surface\_2\_offer, capitalization\_2\_false,$$
$$surface\_3\_flights, capitalization\_3\_false,$$
$$surface\_4\_to, capitalization\_4\_false,$$
$$surface\_6\_COMMA, capitalization\_6\_false\}$$

and subsequently translated to integer values: {987, 435, 656634, 4235, 234, 6453, 64, 242, 786, 89}. During the application of Apriori, only those subsets are retained that reflect a frequently occurring textual pattern: {6453,64,242,786,89}= "flights to *, *".

Apriori generates all patterns that exceed a given frequency threshold. Inevitably, this yields multiple patterns that are subsumed by each other (e.g. if `" * was born`

Relation	Size	Dataset Description	P_{manual}	$P_{classic}$	ΔP_{FIM}	$\Delta P_{FIMtuned}$
albumBy	19852	Musicians and their musical works	80.8%	**27.4%**	-11.6%	-18%
bornInYear	172696	persons and their year of birth	40.7%	19.5%	**+48.4%**	+17%
currencyOf	221	countries and their official currency	46.4%	22.8%	-17.6%	**+10.9%**
headquarteredIn	14762	companies and the country of their headquarter	3%	9.8%	**+2.2%**	-5.2%
locatedIn	34047	cities and their corresponding country	73%	**56.5%**	-8.4%	-0.5%
productOf	2650	product names and their manufacturers.	64.6%	42.2%	-0.9%	**+12%**
teamOf	8307	sportspersons and their team or country	30%	8.0%	**+1.4%**	+0.8%
average			48.3	26.6%	+1.9%	**+4.7**%

Table 1. Relations with precision scores obtained by the classic system (manual evaluation) and differences (Δ) measured with the two FIM conditions.

`in * "` is frequent, then `" * was * in * "` is frequent as well). In order to avoid such too general patterns and at the same time avoiding too specific ones (e.g. `"Wolfgang Amadeus * was born in * "`), we introduce the following rule for removing more general patterns: if pattern a has all constraints also present in b and one more, b is removed unless SUPPORT(b) is at least 20% higher than SUPPORT(a). This rule is applied starting with the smallest patterns. We experimentally determined that the threshold of 20% leads to a generally rather appropriate set of patterns. The remaining unwanted patterns are left to be eliminated by further filtering.

4 Experimental Evaluation

The goal of our experimental evaluation is to demonstrate the advantages of modeling the pattern abstraction subtask of iterative pattern induction as a frequent itemset mining (FIM) problem. We do so by comparing the performance achieved by our itemset-based implementation with the abstraction algorithm used in previous implementations (compare [3]). We do not intend to show the superiority of the approach based on Frequent Itemset Mining to those from the literature as this would require a common benchmark for large-scale Web Relation Extraction or at least a common basis of implementation. Such a standard does not exist due to the diversity of applications and pattern representation formalisms in the literature. Yet, we evaluate our results on a fairly diverse set of non-taxonomic relations to ensure generality. The datasets we use have already been used in [3] and are provided for download by the authors. As in these experiments, we have also used the same 10 seeds selected by hand and the same automatic evaluation procedure.

4.1 Experimental Setup

In our experiments, we rely on the widely used precision and recall measures to evaluate our system's output with respect to the full extension of the relation[2]. To give an

[2] Note that this is different from the evaluation of other similar systems which calculate these measures with respect to a specific corpus, thus yielding higher scores. Also due to the abscence of a closed corpus in our Web scenario, our notion of recall is is not comparable. We use "relative recall" in the sense that it reflects extractions compared to the highest yield count obtained over all experimental settings we applied.

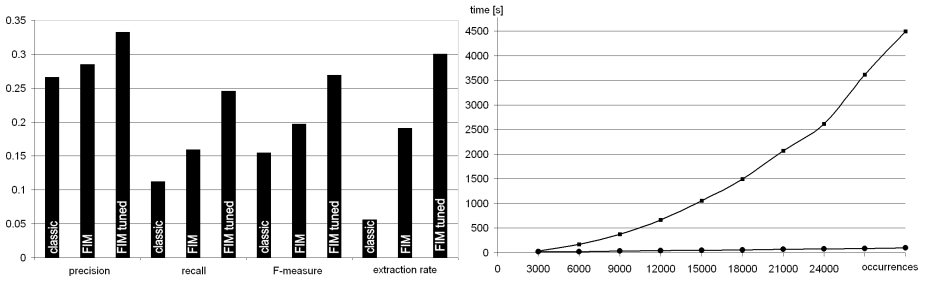

Fig. 2. Precision, recall, F-measure and extraction rate for the individual configurations averaged over all relations (left); Time (sec.) taken by a run of the classical induction algorithm (squares) and the FIM-based algorithm (circles) over the numbers of sample occurrences. (right)

objective measure for temporal performance, we use the *Extraction Rate*, that is, the number of correctly extracted tuples TP over the duration D of the extraction process in seconds (on a dual core machine with 4GB of RAM): $Ex = \frac{TP}{D}$

Figure 2 shows precision, recall and F-measure for three configurations of the system: the *classic* configuration, the *FIM* configuration which uses the proposed modeling of the learning problem with all parameters unchanged and *FIM tuned* for which the parameters have been optimized for the new learning algorithm. In particular, as FIM is more efficient than the classic merge procedure, we can process a higher number of tuples, such that we set the number of occurrences downloaded to 200 (versus a decreasing number as used in [3]). All the other parameters of the algorithm have been chosen as described there. Overall, there is a small superiority of *FIM* over the classic version in terms of precision and recall (29% vs. 27% and 15% vs. 11%). Most importantly, there is a clear superiority in terms of extraction rate (0.19 vs. 0.05 occurrences/second). This difference is statistically significant (two-sides paired Student's t-test with an α-Level of 0.05).

Table 1 shows the different relations together with the size of their extension, the precision yielded by a manual evaluation of a sample of 100 tuples of each relation (P_{manual}), the precision yielded by the classic pattern induction algorithm $P_{classic}$ as well as the relative improvements yielded by our formulation of the problem as a frequent itemset mining (FIM) task relative to the precision $P_{classic}$ calculated automatically with respect to the relation's extension[3]. The best results for each relation are highlighted. In general, we see that while the results vary for each relation, overall the FIM version of the algorithm does not deteriorate the results, but even slightly improves them on average (+1,9% for the FIM version and +4.7% for the tuned FIM version).

4.2 Discussion

In principle, there are no reasons for any of the abstraction algorithms to show better precision and recall because they both explore all possible frequently occurring patterns

[3] Note here that the precision $P_{classic}$ calculated automatically with respect to the datasets is much lower than the precision obtained through sampled manual evaluation (P_{manual}). This is due to the in some cases unavoidable in-completeness of the datasets and orthographic differences in test data and extraction results.

in a breadth-first-search manner. Differences are due to minor modeling issues (see below), the slightly different evaluation of patterns based directly on support counts produced by apriori and, most importantly, the fact that learning is cut off after one hour per iteration. Indeed the standard implementation frequently reached this time limit of an hour, thus leading to better results for the FIM version of the algorithm which does not suffer from this time limit.

One example of slight modeling differences which influenced performance is the treatment of multi-word instances. The learner has to decide whether to insert one wildcard $*$ in an argument position (nearly always matching exactly one word) or two (allowing for two or more words). The classic version heuristically takes the number of words in the argument of the first occurrence used for pattern creation as sample for the wildcard structure. The FIM version encodes the fact that an argument has more than one word as an additional constraint. If this item is contained in a learned frequent itemset, a double wildcard is inserted. The stronger performance with the *bornInYear* (+48%), *currencyOf* (+10.9%) and *productOf* (+12%) relations can be explained in that way (compare Table 1). For example, the FIM version learns that person names have typically length 2 and birth years always have length 1 while the classic induction approach does not allow this additional constraint. This explains the decreased performance of the classic approach for the relations mentioned above for which at least one argument has a rather fixed length (e.g. years).

As indicated in Figure 2, the clear benefit of the FIM abstraction step lies in its runtime behavior. The duration of a pattern generation process is plotted over the number of sample instances to be generalized. To measure these times, both learning modules were provided with the same sets of occurrences isolated from the rest of the induction procedure. The FIM shows a close to linear increase of processing duration for the given occurrence counts. Even though implemented with a number of optimizations (see [3]), the classic induction approach clearly shows a quadratic increase in computation time w.r.t. the number of input occurrences.

5 Related Work

The iterative induction of textual patterns is a method widely used in large-scale information extraction. Sergey Brin pioneered the use of Web search indices for this purpose [4]. Recent successful systems include KnowItAll which has been extended to automatic learning of patterns [9] and Espresso [12]. The precision of Espresso on various relations ranges between 49% and 85%, which is comparable to our range of precisions P_{manual}. Concerning the standard restriction to binary relations, Xu et al. [17] have shown how approaches used for extracting binary relations can be applied to n-ary relations in a rather generic manner by considering binary relations as projections of these. These and the many other related systems vary considerably with respect to the representation of patterns and in the learning algorithms used for pattern induction. The methods used include Conditional Random Fields [16], vector space clustering [1], suffix trees [14] and minimizing edit distance [13]. In this paper, we have proposed to model different representational dimensions of a pattern such as word order, token at a certain position, part-of-speech etc. as constraints. Our approach allows straightfor-

wardly to represent all these dimensions by an appropriate encoding. Given such an encoding, we have shown how frequent itemset mining techniques can be used to efficiently find patterns with a minimal support.
Apart from pattern-based approaches, a variety of supervised and semi-supervised classification algorithms have been applied to relation extraction. The methods include kernel-based methods [18, 8] and graph-labeling techniques [6]. The advantage of such methods is that abstraction and partial matches are inherent features of the learning algorithm. In addition, kernels allow incorporating more complex structures like parse trees which cannot be reflected in text patterns. However, such classifiers require testing all possible relation instances while with text patterns extraction can be significantly speeded up using search indices. From the point of view of execution performance, a pattern-based approach is superior to a classifier which incorporates a learned model which can not be straightforwardly used to query a large corpus such as the web. Classification thus requires linear-time processing of the corpus while search-patterns can lead to faster extraction. Recently, the

A similar approach to ours is the one by Jindal and Liu [10]. They use Sequential Pattern Mining – a modification of Frequent Itemet Mining – to derive textual patterns for classifying comparative sentences in product descriptions. While, like our approach, encoding sequence information, their model is not able to account for several constraints per word. Additionally, the scalability aspect has not been focus of their study as mining has only be performed on a corpus of 2684 sentences with a very limited alphabet. Another approach orthogonal to ours is presented by [7]. Each occurrence is abstracted over in a bottom up manner which saves pairwise occurrence comparison at the expense of evaluating the large amounts of pattern candidates with respect to the training set. The algorithm seems thus more appropriate for fully supervised settings of limited size.

6 Conclusion

Our contribution in this paper lies in the formulation of the pattern induction step as a well-known machine learning problem, i.e. the one of mining frequent itemsets. On the one hand, this formulation is elegant and advantageous as we can import all the results from the literature on association mining for further optimization (an overview of which is given in and [15]). On the other hand, we have shown that this formulation leads to a significant decrease in the running time of the extraction. In particular, we have shown that the running time behavior decreases from quadratic to linear with the number of occurrences to be generalized with respect to previous implementations. Further, we have also shown that the quality of the generated tuples even slightly increases in terms of F-measure compared to the standard pattern induction algorithm. This increase is mainly due to the modeling of argument length as an additional constraint which can be straightforwardly encoded in our FIM framework. Overall, modeling the different representational dimensions of a pattern as constraints is elegant as it allows to straightforwardly add more information. In future work we plan to consider taxonomic as well as other linguistic knowledge.

Acknowledgements

This work was funded by Deutsche Forschungsgemeinschaft (MULTIPLA project, grant Nr 38457858) and the X-Media project (www.x-media-project.org) sponsored by the European Commission as part of the Information Society Technologies (IST) program under EC grant number IST-FP6-026978.

References

1. E. Agichtein and L. Gravano. Snowball: extracting relations from large plain-text collections. In *Proceedings of the fifth ACM conference on Digital Libraries (DL)*, 2000.
2. R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. In *Proceedings of the 20th International Conference on Very Large Data Bases (VLDB'94)*, pages 487–499, 1994.
3. S. Blohm, P. Cimiano, and E. Stemle. Harvesting relations from the web -quantifiying the impact of filtering functions. In *Proceedings of AAAI'07*, pages 1316–1323, 2007.
4. S. Brin. Extracting patterns and relations from the world wide web. In *Proceedings of the WebDB Workshop at the 6th International Conference on Extending Database Technology (EDBT)*, 1998.
5. R. Bunescu and R. Mooney. Learning to extract relations from the web using minimal supervision. In *Proceedings of ACL 2007*, 2007.
6. J. Chen, D. Ji, C. L. Tan, and Z. Niu. Relation extraction using label propagation based semi-supervised learning. In *Proceedings of COLING-ACL 2006*, pages 129–136, 2006.
7. F. Ciravegna. Adaptive information extraction from text by rule induction and generalisation. In *Proceedings of IJCAI 2001*, pages 1251–1256, 2001.
8. A. Culotta and J. Sorensen. Dependency tree kernels for relation extraction. In *Proceedings of the 42nd ACL*, pages 423–429, 2004.
9. D. Downey, O. Etzioni, S. Soderland, and D. Weld. Learning text patterns for web information extraction and assessment. In *Proceedings of the AAAI Workshop on Adaptive Text Extraction and Mining*, 2004.
10. N. Jindal and B. Liu. Mining comparative sentences and relations. In *Proceedings of AAAI'06*. AAAI Press, 2006.
11. A. Mueller. Fast sequential and parallel algorithms for association rule mining: a comparison. Technical report, College Park, MD, USA, 1995.
12. P. Pantel and M. Pennacchiotti. Espresso: Leveraging generic patterns for automatically harvesting semantic relations. In *Proceedings of COLING-ACL'06*, pages 113–120, 2006.
13. P. Pantel, D. Ravichandran, and E. H. Hovy. Towards terascale knowledge acquisition. In *Proceedings of COLING-04*, pages 771–777, 2004.
14. D. Ravichandran and E. Hovy. Learning surface text patterns for a question answering system. In *Proceedings of the 40th Annual Meeting of the ACL*, pages 41–47, 2001.
15. L. Schmidt-Thieme. *Assoziationsregel-Algorithmen für Daten mit komplexer Struktur*. PhD thesis, Universität Karlsruhe, 2007.
16. P. P. Talukdar, T. Brants, M. Liberman, and F. Pereira. A context pattern induction method for named entity extraction. In *Proceedings of the 10th CoNLL*, New York City, 2006.
17. F. Xu, H. Uszkoreit, and H. Li. A seed-driven bottom-up machine learning framework for extracting relations of various complexity. In *Proceedings of the 45th ACL*, pages 584–591, 2007.
18. D. Zelenko, C. Aone, and A. Richardella. Kernel methods for relation extraction. *Journal of Machine Learning Research*, 3:1083–1106, 2003.