Harvesting Relations from the Web
– Quantifiying the Impact of Filtering Functions –

Sebastian Blohm and **Philipp Cimiano**
Institute AIFB, Knowledge Management Research Group
University of Karlsruhe
D-76128 Karlsruhe, Germany
{blohm, cimiano}@aifb.uni-karlsruhe.de

Egon Stemle
Institute of Cognitive Science
University of Osnabruück
D-49069 Osnabruück, Germany
egon.stemle@uni-osnabrueck.de

Abstract

Several bootstrapping-based relation extraction algorithms working on large corpora or on the Web have been presented in the literature. A crucial issue for such algorithms is to avoid the introduction of too much noise into further iterations. Typically, this is achieved by applying appropriate pattern and tuple evaluation measures, henceforth called *filtering functions*, thereby selecting only the most promising patterns and tuples. In this paper, we systematically compare different filtering functions proposed across the literature. Although we also discuss our own implementation of a pattern learning algorithm, the main contribution of the paper is actually the extensive comparison and evaluation of the different filtering functions proposed in the literature with respect to seven datasets. Our results indicate that some of the commonly used filters do not outperform a trivial baseline filter in a statistically significant manner.

Introduction

For many knowledge-intensive applications, the acquisition of background knowledge still forms a major bottleneck. Currently, the Web constitutes a major source of data for many applications, but manually scanning the Web for relevant data is typically neither feasible nor desireable. Acquiring relational facts like *interactsWith(Protein,Protein)* or *releaseInYear(Company,CarModel,Year)* in an automated fashion could have important impact on knowledge-intensive tasks like pharmaceutical and medical research or market analysis. Extracting such structured relations from unstructured sources like text seems thus a key challenge towards effectively tackling the knowledge acquisition bottleneck.

Current research in Relation Extraction (e.g. (Agichtein & Gravano 2000), (Downey *et al.* 2004) and (Pantel & Pennacchiotti 2006)) has focused so far mainly on pattern learning and matching techniques for extracting relational facts from large corpora or the Web. The Web as a source has the advantage of being highly redundant, not domain-restricted and freely available but lacks any control that could ensure coherence or correctness. Many systems have been presented that successfully address the task of Web-based relation extraction for a limited set of relations or a particular application. They typically build on the paradigm of iterative induction of patterns as proposed by Brin (Brin 1998). However, to achieve the goal of minimal supervision and automatic adaptation to new extraction tasks, further understanding of crucial design choices for such systems is required. In this line, we present experimental results obtained on a fairly generic implementation allowing to systematically compare different pattern evaluation functions presented in the literature. As opposed to previous studies we base our analysis on a large number of relations that vary in size, domain and presence on the Web. The extensive evaluation has, to our knowledge, no precedent in the literature and constitutes the main contribution of the paper.

The structure of the paper is as follows: in Section *The Pronto System* we discuss the basic algorithm and our implementation which we called*Pronto*. In Section *Filtering Functions* we discuss the different evaluation functions subject to analysis. In Section *Experiments* we describe in detail the settings and results of our experiments on seven relations. Before concluding, we discuss relevant related work.

The Pronto System

Figure 1 describes the generic pattern learning algorithm used in our experiments. It subsumes many of the approaches mentioned in the introduction and implementing similar bootstrapping-like procedures. 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. For the *locatedIn* relation the seed set may be $\{(Vancouver, Canada), (Karlsruhe, Germany), ...\}$ Further, patterns are learned by abstracting over the text occurrences of the tuples. The new patterns are then evaluated and filtered before they are matched. From these matches, new tuples are extracted, evaluated and filtered. Patterns may look like

“flights to ARG_1 , ARG_2 from ANY airport”

Where ARG_1 , ARG_2 represent the argument slots and ANY is a wildcard.

```
ITERATIVE PATTERN INDUCTION($PatternsP', TuplesS'$)
  $S \leftarrow S'$
  $P \leftarrow P'$
  while not DONE
  do $Occ_t \leftarrow$ MATCH-TUPLES($S$)
      $P \leftarrow P \cup$ LEARN-PATTERNS($Occ_t$)
      EVALUATE-PATTERNS($P$)
      $P \leftarrow \{p \in P \mid$ PATTERN-FILTER-CONDITION($p$)$\}$
      $Occ_p \leftarrow$ MATCH-PATTERNS($P$)
      $S \leftarrow S +$ EXTRACT-TUPLES($Occ_p$)
      EVALUATE-TUPLES($S$)
      $S \leftarrow \{t \in S \mid$ TUPLE-FILTER-CONDITION($t$)$\}$
```

Figure 1: Iterative pattern induction algorithm starting with initial patterns P' and tuples S'

The process is stopped when the termination condition DONE is fulfilled (typically, a fixed number of iterations is set). A crucial issue for such an approach is an appropriate filtering function on the basis of which the patterns can be ranked and selected for matching new tuples. Our focus in this article is on an extensive analysis of different filtering functions to be applied within the filtering steps. Below, we describe each of the steps of the algorithm in Figure 1 in more detail.

To ensure the generality of our results, we have refrained from integrating specific additional knowledge in our implementation. Common forms of background knowledge applied in the literature are thesauri, filters for part-of-speech or syntactic criteria and knowledge about the type of relation in question (e.g. part-of-speech tags in (Pantel & Pennacchiotti 2006) or named entity classification in (Agichtein & Gravano 2000)).

Matching Tuples

In order to identify occurrences of the current seed set on the Web, the search index is accessed via Google's Java API querying for pages on which all words present in both arguments of the tuple can be found. A fixed number $num_{matchTuples}$ of results is retrieved. From those, only the result headers and text snippets are kept which contain all arguments within a distance of at most $max_{argDist}$. For the experiments presented in this paper we set $max_{argDist} = 4$ and decrease $num_{matchTuples}$ from 200 to 20 in steps of 45 over 5 iterations. Note that all parameters chosen for the experiments have been determined experimentally to ensure stable performance across typical configurations and target relations.

Learning and Filtering Patterns

LEARN-PATTERNS aims at finding representative abstractions of as many valid occurrences of relation instances as possible. Patterns are expressed as a set of constraints on the tokens. There are two types of constraints: the *surface string* of individual words and their corresponding *capitalization.*

Our learning algorithm essentially merges groups of occurrences on a token by token basis. Constraints that are shared by all occurrences within a group are kept while the others are eliminated. An unoptimized version of the algorithm for merging is given in Figure 2 for illustration purposes. Basically, it ensures that all subsets of the set of found occurrences Occ are merged, if they share a certain minimum number of constraints. The above pattern example may have been generated by the following example occurrences:

```
LEARN-PATTERNS($Occ$)
  $Queue \leftarrow Occ$
  $P' \leftarrow \emptyset$
  while NON-EMPTY($Queue$)
  do
      $o =$ FIRST($Queue$)
      for $o' \in Occ \cup P'$
      do
          $p \leftarrow$ MERGE($o, o'$)
          if CONSTRAINTS($p$) $\geq min_{common}$
            then
                $P' \leftarrow P' \cup p$
                ADD($Queue, p$)
  OUTPUT($P'$)
```

Figure 2: The algorithm that learns patterns from a set Occ of occurrences.

"... flights to Athens , Greece from Heathrow airport..."

"... flights to Paris , France from JFK airport..."

The procedure MERGE(p, p') takes the patterns p and p', aligns them by their arguments and generates a pattern containing only the constraints that p and p' share for any of their token positions. The function CONSTRAINTS(p) counts the number of non-empty constraints in p as we ensure that at least min_{common} constraints are shared. To reduce the algorithm's time complexity, an index data structure is used to avoid the $|Occ|^2$ comparisons required otherwise.

In its pure form, the algorithm generates much more candidate patterns than could reasonably be processed further. We heuristically filter out too specific patterns by eliminating those that were generated from merging occurrences from one tuple and too general patterns by eliminating those consisting mostly of stop words.

Prior to merging, the occurrences are stripped off the text more than t_{prefix} words before the first and t_{suffix} words after the last argument. When comparing the occurrences in which the arguments stand at different distances, only the first t tokens are considered, where t is the minimum distance encountered between arguments. For our experiments we chose $t_{prefix} = t_{suffix} = 2$ and $min_{common} = 2$.

In our experiments, we define PATTERN-FILTER-CONDITION(p) to always retain the top 100 best scoring patterns according to filtering functions, i.e. $|P| = 100$. Thus, newly learned patterns compete against those kept from previous iterations and may replace them. Filtering is important so as to exclude too specific (e.g. " the Acropolis in ARG_1 , ARG_2") or too general patterns (e.g. "... ARG_1 is in ARG_2...").

Matching Patterns and Filtering Tuples

MATCH-PATTERNS(P') matches each pattern in P' by running a set of queries to the Google API. For this purpose, patterns are translated to queries. For each query, a fixed number of search results $num_{matchPatterns}$ (60 in our experiments) is retrieved. The queries are generated by taking the surface string constraint for each token in a blank-separated manner. Tokens with empty surface string constraints are represented by a $*$ wild card, which - when used in quotes - will be replaced with any word or very few words in this position in the Google results. This sequence is stripped from leading and closing $*$ wild cards and surrounded by quotes. For instance, the above pattern example would be translated into a Google-query as follows:

```
"flights to * * from * airport"
```

A $*$ wildcard is filled in for each argument slot and each token with eliminated surface string constraint (above marked by "ANY"). The comma in the pattern is represented as an individual token with the comma surface string. During querying, however, it is omitted as Google discards punctuation characters in queries. More exact matching is done in a subsequent analysis step prior to tuple generation (EXTRACT-TUPLES(Occ_p)).

For the purpose of our experiments we compute the confidence of a tuple by averaging over the confidence that Pronto assigns to the patterns that extracted the tuple.

TUPLE-FILTER-CONDITION(t) is implemented to be true for the top $p_{filterTuples}$ % of the newly generated tuples (we use 50% if not otherwise specified). In the following section, we present several pattern filtering functions and evaluate the impact of the choice of the filtering function in Section *Experiments*.

Filtering Functions

We model both the assessment of patterns in P and tuples S as scoring functions $score_P : P \rightarrow \mathbb{R}$ and $score_T : S \rightarrow \mathbb{R}$ and then filter out potentially erroneous tuples and weakly performing patterns.

As the focus of this paper is to compare the performance of different pattern filtering functions, we present those used for comparison in more detail. Tuple filtering is kept constant and across all experiments (cf. Section), i.e. the top 50% of the ranked tuples are selected and carried over to the next iteration as seeds.

We identified five general types of pattern quality assessment.

- *Syntactic assessment.* Filtering purely based on syntactic criteria is for example done in (Brin 1998), where a pattern's length is used to predict its specificity.
- *Inter-pattern comparison.* If there is a set of patterns that is known to be good, it may be worth-while to rate a new pattern based on how similar its output is to the output of those patterns.
- *Support-based assessment.* The iterative nature of the extraction allows to estimate quality of patterns based on the set of occurrences that contributed to the generation of this pattern. An analogous filtering step was suggested in (Brin 1998).
- *Performance-based assessment.* The most straightforward way to assess a pattern's quality is to judge the rate of correctly produced output typically by comparing it to output of previous iterations.
- *Instance-Pattern correlation.* A further indicator for the quality of a pattern is whether its presence correlates strongly with the presence of instances of the target relation. Estimating this by counting occurrences of patterns, seed instances and patterns instantiated with seed instances allows controlling both precision and potential recall of a pattern within one value. Such measures are used for example in the Espresso (Pantel & Pennacchiotti 2006) and the KnowItAll (Etzioni *et al.* 2005) system.

We present here the filtering functions compared in our experiments, which are partly taken from the literature of state-of-the-art pattern induction systems.

Definition 1 $score_{prec}(p)$*:*
Following (Agichtein & Gravano 2000) we use the output of previous iterations to approximate a performance-based *precision. We define $m(p)$ to be the tuples matched by pattern p, and S to be the seeds of the current iteration. Approximating the precision amounts to calculating:*

$$score_{prec}(p) = \frac{|m(p) \cap S|}{|m(p)|}$$

Filtering functions based on instance-pattern correlation rely on counts of Web search matches of patterns with or without filling their argument slots with particular relation instances. The Web frequencies derived in this manner are used to assess the correlation between patterns and certain tuples via pointwise mutual information (PMI).

Definition 2 $score_{pmi}(p)$*: Once such a coherence value $pmi(p,t)$ is available, pattern confidence values can be computed by averaging over a random subset of the currently accepted tuples S' (whereby sampling is done for efficency reasons).*

$$score_{pmi}(p) = \frac{1}{|S'|} \sum_{t \in S'} pmi(p,t)$$

The $pmi(p,t)$ is approximated in different ways two of which are given below[1].

Definition 3 $pmi_{KnowItAll}(p,t)$*: The KnowItAll (Etzioni* et al. *2005) information extraction system uses PMI in the following way to assess coherence of a pattern-tuple pair (p,t) in*[2]*:*

$$pmi_{KnowItAll}(p,t) = \frac{|t_1, p, t_2|}{|t_1, *, t_2|}$$

[1] Note that in both versions PMI is not applied in line with its original definition. See (Pantel & Pennacchiotti 2006) for a PMI definition in this context

[2] Following (Pantel & Pennacchiotti 2006), we write $|t_1, p, t_2|$ to denote the number of search engine matches of a query generated by filling the components of tuple $t = (t_1, t_2)$ into the argument slots of pattern p, while $*$ means allowing arbitrary values for the pattern or the argument replaced.

Definition 4 $pmi_{Espresso}(p,t)$*: In the Espresso system (Pantel & Pennacchiotti 2006), PMI is used in a different way aiming at relating the event of the pattern occurring in the corpus and the event of the tuple occurring in the corpus:*

$$pmi_{Espresso}(p,t) = log\frac{|t_1,p,t_2|}{|*,p,*|\;|t_1,*,t_2|}$$

In addition to the above filtering functions, we further present a simple filtering function based on the count of distinct tuples from which a pattern was generated:

Definition 5 $score_{merge}(p)$*: Given the number of distinct tuples present in the occurrences from which a pattern was generated, i.e.* $distinct\_generators(p)$*, we define*

$$score_{merge}(p) = |distinct\_generators(p)|$$

Thus, $score_{merge}$ evaluates patterns by the number of different seed tuples from which they have been produced, hence favoring more general patterns and penalizing patterns which just hold for a few examples.

Definition 6 $score_{random}(p)$*: As a baseline condition, a pattern evaluator has been implemented that assigns random confidence values* $score_{random}(p)$ *to all patterns.*

Definition 7 $score_{gold}(p)$*: In order to estimate the upper limit of the potential of performance-based pattern evaluation, we introduce a scoring function that is based on the full knowledge of the extension G of the target relation. This extension is made available externally from large datasets we produced for that purpose (compare Section Experiments):*

$$score_{gold}(p) = \frac{|m(p) \cap G|}{|m(p)|}$$

We use the term gold standard for this measure even though the measure may still be out-performed due to limitations inherent to performance-based filtering.

Experiments

In order to assess the potential of the approach taken, we have performed experiments with various target relations and configurations of Pronto. The goal of our experiments is to explore the strengths and weaknesses of different filtering functions from the literature, comparing these results to our baseline $score_{random}(p)$ as well as the informed upper bound $score_{gold}(p)$.

All other parameters have been chosen as described in Section *The Pronto System.*

The experiments consisted of running the extraction algorithm for five iterations, starting with a seed set S of size 10.

Datasets and Evaluation Measures

We obtained large relation sets using (i) a DAML version of the CIA World Factbook (for currency), (ii) lineup data from 50 years of FIFA soccer games provided by the SmartWeb project[3] and (iii) exploiting Wikipedia categories in a semi-automatic manner using the CatScan tool by Daniel Kinzler[4]. Some facts about the relation sets acquired are given in Table 1.

albumBy	Musicians and their albums (occasionally other types of musical works) $n = 19852$
bornInYear	Persons and their year of birth $n = 172696$
currencyOf	Countries and their official currency $n = 221$
headquarteredIn	Companies and the country of their headquarter $n = 14762$
locatedIn	Cities and the Country they lie in. $n = 34047$
productOf	Product names and the brand names of their makers. $n = 2650$
teamOf	Sportspersons and the team or country they are playing for $n = 8307$

Table 1: Relations used for evaluation.

The extraction output has been evaluated automatically based on the data sets described above. Approximate matches are admitted by allowing the omission of stop words and respecting WordNet synonyms.

In our experiments, we rely on the widely used precision and recall measures to evaluate Pronto's output. These measures compute the ratio of correctly found instances to overall tuples extracted (precision) or all tuples to be found (recall). As the fixed number n of iterations in our experiments poses a fixed limit on the number of possible extractions we use a notion of *relative recall* assuming the maximally extracted number of tuples by any configuration as the overall number of possible extractions. The F-measure (more precisely F_1-measure) is a combination of precision and recall by the harmonic mean.

Analysis of Filtering Functions

The impact of the choice of filtering function on the precision of the output can be observed in Figure 3. The precision of the output of the last iteration has been plotted over the relations examined and the filtering function chosen. The results of a two-sided paired Student's t-test given in Table 2 show the significance of the observed differences. The null hypothesis is that the results for all relations of the two filtering functions compared originate from the same distribution. A '+' indicates that the null hypothesis could be rejected at an α-level of 0.10 and a '++' indicates rejection at an α-level of 0.05.

The significance tests show that $score_{gold}$ outperforms all other strategies as expected and that $score_{merge}$ and $score_{prec}$ are superior to $score_{random}$. However, the PMI-based evaluation measures implemented in KnowItall and Espresso do not perform significantly better than the baseline, while at the same no significance differ-

[3] http://www.smartweb-project.de

[4] http://tools.wikimedia.de/~daniel/WikiSense

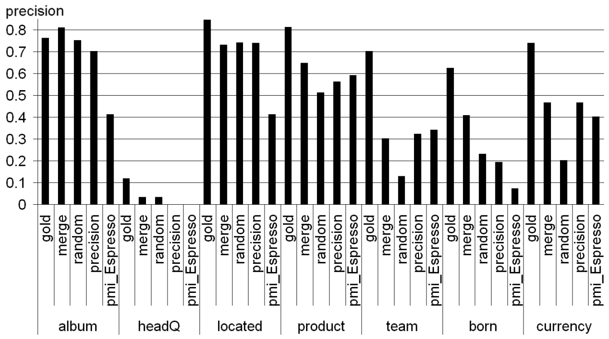

Figure 3: Overall output precision after 5 iterations for the 7 different relations and evaluation strategies based on automatic evaluation.

ence could be in fact observed between the filtering functions $score_{merge}$, $score_{pmi_{Espresso}}$, $score_{pmi_{KnowItAll}}$ and $score_{prec}$.

	gold	merge	Espr.	Prec.	Rand.	Know.
gold	-	++	++	++	++	++
merge	++	-	-	-	+	-
Espresso	++	-	-	-	-	-
Precision	++	-	-	-	++	-
Random	++	+	-	++	-	-
KnowItAll	++	-	-	-	-	-

Table 2: Results of a significance test on the difference of output distributions.

While $score_{random}(p)$ incorporates no information whatsoever into its selection, $score_{gold}(p)$ incorporates complete information about the extension of the relation and thus represents a ’fully informed’ evaluation strategy. Such complete information would actually never be available, such that it is important to stress that this evaluation strategy has to be regarded merely to assess the impact of pattern filtering.

As can be observed in Figure 3 and has been shown with the significance tests in Table 2, most filtering functions presented here perform better in terms of precision than $score_{random}$ and worse than the informed filter $score_{merge}$, which is an expected result. Although the evaluation strategies based on performance and instance-pattern correlation, as implemented in Espresso and KnowItAll do not perform significantly better than our baseline, it is important to emphasize that the baseline we have compared with, which consists in eliminating all patterns generated from only one instance, provides already a non-trivial baseline difficult to outperform.

Figure 4 shows precision, recall and F_1-measure values for different filtering functions averaged over runs on different relations. There is a clear superiority of $score_{gold}$ and $score_{merge}$ in terms of precision. In terms of recall, however, the other filtering functions $score_{random}$, $score_{pmi_{Espresso}}$ and $score_{prec}$ are slightly superior. This negative correlation between recall and precision can be observed in the output for all individual relations but is particularly apparent for the *productOf* relation which is the relation for which all scoring function achieve highest overall precision. The reasons for the lower recall with $score_{gold}$ and $score_{merge}$ lie in the fact that many of the patterns they generate contain individual tokens that make them too specific. Manual inspection of the patterns extracted for the *locatedIn* relation with $score_{gold}$ mention a city, person name or date in a position that should have been a wildcard in 48% of the cases (as opposed to 19% with $score_{random}$). Apparently these patterns do not harm extraction precision but reduce recall.

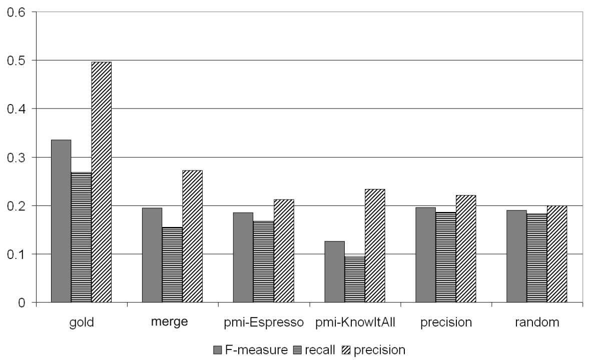

Figure 4: Precision, relative recall and F_1-measure by filtering strategy averaged over the 7 relations.

Finally, we have also asked human evaluators to manually evaluate the output of the experiments and presented them 100 randomly selected tuples of the learned relations. The precision was between 0.30 and 0.81 on average(with an exception of *headquarteredIn*), such that we can conclude that the automatic evaluation presented above underestimates the actual performance by a factor of 1.2 to 1.8 depending on the quality of the dataset.

Related Work

Several systems have addressed the task of learning instances of concepts, among them Know-It-All (Etzioni *et al.* 2005) and PANKOW (Cimiano, Handschuh, & Staab 2004). These approaches are based on a set of rigid patterns in the line of Hearst (Hearst 1992), which are matched on the Web and used to find instance-concept relations. The Know-It-All system has even been extended with pattern learning capabilities to discover new patterns (compare (Downey *et al.* 2004)). A similar system is that of (Snow, Jurafsky, & Ng 2005) which integrates syntactic dependency structure into pattern representation but has been only applied to the task of learning instance-of relations or isa-relations.

The seminal work of (Brin 1998) introduced the basic bootstrapping algorithm used by Snowball, Espresso as well as in Pronto as it is presented here. DIPRE was manually evaluated for (author,book) tuples reaching a precision of $\frac{19}{20} = 95\%$. As a novelty, Snowball (Agichtein & Gravano 2000) relies on annotation of named entities with their category which can be used in formulating the patterns. Parts of the pattern are represented as bag-of-words vectors and not plain strings, thus capturing the frequency of the words occurring around the arguments in diverse occurrences. Like Snowball, the system of Ravichandran and collegues (Ravichandran & Hovy 2001) has recently been applied and evaluated in question answering scenarios. An interesting feature of Ravichandran's system is the automatic detection of reasonable pattern borders with suffix trees as a data structure that allows retrieving occurrence counts for all sub-strings of the occurrence in linear time.

Espresso (Pantel & Pennacchiotti 2006) is a similar system to the ones described above, but mainly differs in its pattern evaluation strategy. Espresso relies on the Web to calculate a PMI-based association measure between tuples and patterns. Espresso is evaluated automatically on different text collections and various relations: the classical *is-a* and *part-of* relations, but also the *succession*, *(chemical) reaction* and *product-of* relations. The precision ranges from 49%-85% depending on the relation considered and thus performs similar to our results. The precision of both systems is thus smaller compared to the ones achieved by DIPRE and Snowball, but the latter ones have also been only evaluated on one relation.

Conclusion and Future Work

We have investigated how the alternative approaches to pattern filtering affect the induction process. To our knowledge, this is the first systematic comparison of different evaluation techniques. A notable exception is the work of (Downey, Etzioni, & Soderland 2005), who show that a probabilistic model – URNS – significantly outperforms PMI. URNS however is specialized on tuple filtering requiring a precision estimate for each pattern as input to the model.

The conclusions of our experiments are in fact not only interesting for our own pattern-learning systems, but for all bootstrapping-based systems in the sense that it sheds light on the benefits and disadvantages of different pattern evaluation strategies. In particular, we have shown the influence of pattern filtering on extraction performance by comparing random and fully informed filtering ($score_{random}$ and $score_{gold}$, respectively) to various filtering strategies based on evaluation functions presented in the literature. Our results indicate that a relatively simple evaluation strategy, i.e. our simple merge evaluation strategy overall yields better results than more elaborate measures such as PMI, which relies on web occurrence counts. In fact, the PMI-based scores were unable to outperform the random baseline in a statistically significant way, while the merge strategy counting solely the number of distinct relation instances from which a pattern was generated does achieve statistical significance. This raises indeed doubts about the appropriateness of PMI and Web-based evaluation measures in general (compare also (Downey, Etzioni, & Soderland 2005)). However, we have also shown that PMI yields a higher recall than our straightforward merge strategy, which is biased towards precision.

The vision behind this research is developing a system that is relatively autonomously able to inform itself from Web sources choosing extraction parameters, appropriate filtering strategies and the level of pattern representation according to the task at hand. In future work we will consider if richer structural and linguistic information can be used in the description of patterns. Furthermore, it will be necessary to clarify those properties of relations which have a clear impact on their learnability of relations and therefore might turn out as crucial to guide the pattern induction algorithm.

Acknowledgements

The authors would like to thank Johanna Wenderoth, Mina Nikolova and Matthias Mantel for manually evaluating extraction output. This work was funded by 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. Thanks to Google for giving enhanced access to their API.

References

Agichtein, E., and Gravano, L. 2000. Snowball: extracting relations from large plain-text collections. In *Proceedings of the fifth ACM conference on Digital libraries (DL)*, 85–94.

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

Cimiano, P.; Handschuh, S.; and Staab, S. 2004. Towards the self-annotating web. In *WWW '04: Proceedings of the 13th international conference on World Wide Web*, 462–471.

Downey, D.; Etzioni, O.; Soderland, S.; and Weld, D. 2004. Learning text patterns for web information extraction and assessment. In *AAAI-2004 Workshop on Adaptive Text Extraction and Mining*.

Downey, D.; Etzioni, O.; and Soderland, S. 2005. A probabilistic model of redundancy in information extraction. In *Proceedings of IJCAI'05*.

Etzioni, O.; Cafarella, M.; Downey, D.; Popescu, A.-M.; Shaked, T.; Soderland, S.; Weld, D.; and Yates, A. 2005. Unsupervised named-entity extraction from the web: an experimental study. *Artificial Intelligence* 165(1):91–134.

Hearst, M. A. 1992. Automatic acquisition of hyponyms from large text corpora. In *Proceedings of the 14th conference on Computational linguistics*.

Pantel, P., and Pennacchiotti, M. 2006. Espresso: Leveraging generic patterns for automatically harvesting semantic relations. In *Proceedings of COLING/ACL-06*, 113–120.

Ravichandran, D., and Hovy, E. 2001. Learning surface text patterns for a question answering system. In *Proceedings of the 40th Annual Meeting of the ACL*, 41–47.

Snow, R.; Jurafsky, D.; and Ng, A. 2005. Learning syntactic patterns for automatic hypernym discovery. In *Proceedings of the 17th Conference on Advances in Neural Information Processing Systems (NIPS)*. MIT Press.