Counter-Training in Discovery of Semantic Patterns

Roman Yangarber
Courant Institute of Mathematical Sciences
New York University
roman@cs.nyu.edu

Abstract

This paper presents a method for unsupervised discovery of semantic patterns. Semantic patterns are useful for a variety of text understanding tasks, in particular for locating events in text for information extraction. The method builds upon previously described approaches to iterative unsupervised pattern acquisition. One common characteristic of prior approaches is that the output of the algorithm is a continuous stream of patterns, with gradually degrading precision.

Our method differs from the previous pattern acquisition algorithms in that it introduces competition among several scenarios simultaneously. This provides natural stopping criteria for the unsupervised learners, while maintaining good precision levels at termination. We discuss the results of experiments with several scenarios, and examine different aspects of the new procedure.

1 Introduction

The work described in this paper is motivated by research into automatic pattern acquisition. Pattern acquisition is considered important for a variety of "text understanding" tasks, though our particular reference will be to Information Extraction (IE). In IE, the objective is to search through text for entities and events of a particular kind—corresponding to the user's interest. Many current systems achieve this by pattern matching. The problem of recall, or coverage, in IE can then be restated to a large extent as a problem of acquiring a comprehensive set of good patterns which are relevant to the *scenario* of interest, i.e., which describe events occurring in this scenario.

Among the approaches to pattern acquisition recently proposed, *unsupervised* methods[1] have gained some popularity, due to the substantial reduction in amount of manual labor they require. We build upon these approaches for learning IE patterns.

The focus of this paper is on the problem of *convergence* in unsupervised methods. As with a variety of related iterative, unsupervised methods, the output of the system is a *stream* of patterns, in which the quality is high initially, but then gradually degrades. This degradation is inherent in the trade-off, or tension, in the scoring metrics: between trying to achieve higher recall vs. higher precision. Thus, when the learning algorithm is applied against a reference corpus, the result is a ranked list of patterns, and going down the list produces a curve which trades off precision for recall.

Simply put, the unsupervised algorithm does not know when to stop learning. In the absence of a good stopping criterion, the resulting list of patterns must be manually reviewed by a human; otherwise one can set *ad-hoc* thresholds, e.g., on the number of allowed iterations, as in (Riloff and Jones, 1999), or else to resort to supervised training to determine such thresholds—which is unsatisfactory when our

[1] As described in, e.g., (Riloff, 1996; Riloff and Jones, 1999; Yangarber et al., 2000).

goal from the outset is to try to limit supervision. Thus, the lack of natural stopping criteria renders these algorithms less unsupervised than one would hope. More importantly, this lack makes the algorithms difficult to use in settings where training must be completely automatic, such as in a general-purpose information extraction system, where the topic may not be known in advance.

At the same time, certain unsupervised learning algorithms in other domains exhibit inherently natural stopping criteria. One example is the algorithm for word sense disambiguation in (Yarowsky, 1995). Of particular relevance to our method are the algorithms for semantic classification of names or NPs described in (Thelen and Riloff, 2002; Yangarber et al., 2002).

Inspired in part by these algorithms, we introduce the *counter-training* technique for unsupervised pattern acquisition. The main idea behind counter-training is that several identical simple learners run *simultaneously* to compete with one another in *different* domains. This yields an improvement in precision, and most crucially, it provides a natural indication to the learner when to stop learning—namely, once it attempts to wander into territory already claimed by other learners.

We review the main features of the underlying unsupervised pattern learner and related work in Section 2. In Section 3 we describe the algorithm; 3.2 gives the details of the basic learner, and 3.3 introduces the counter-training framework which is super-imposed on it. We present the results with and without counter-training on several domains, Section 4, followed by discussion in Section 5.

2 Background

2.1 Unsupervised Pattern Learning

We outline those aspects of the prior work that are relevant to the algorithm developed in our presentation.

• We are given an IE scenario S, e.g., “Management Succession” (as in MUC-6). We have a raw general news corpus for training, i.e., an unclassified and un-tagged set of documents D_u. The problem is to find a good set of patterns in D_u, which cover events relevant to S.

We presuppose the existence of two general-purpose, lower-level language tools—a name recognizer and a parser. These tools are used to extract all potential patterns from the corpus.

• The user provides a small number of seed patterns for S. The algorithm uses the corpus to iteratively bootstrap a larger set of good patterns for S.

• The algorithm/learner achieves this bootstrapping by utilizing the duality between the space of documents and the space of patterns: good extraction patterns select documents relevant to the chosen scenario; conversely, relevant documents typically contain more than one good pattern. This duality drives the bootstrapping process.

• The primary aim of the learning is to train a strong *recognizer* R for S; R is embodied in the set of good patterns. However, as a result of training R, the procedure also produces the set D^S_{res} of documents that it deems relevant to S—the documents selected by R.

• Evaluation: to evaluate the quality of discovered patterns, (Riloff, 1996) describes a direct evaluation strategy, where precision of the patterns resulting from a given run is established by manual review. (Yangarber et al., 2000) uses an automatic but *indirect* evaluation of the recognizer R: they retrieve a test sub-set $D^S_{test} \subset D_u$ from the training corpus and manually judge the relevance of every document in D^S_{test}; one can then obtain standard IR-style recall and precision scores for D^S_{res} relative to D^S_{test}.

In presenting our results, we will discuss both kinds of evaluation.

The recall/precision curves produced by the indirect evaluation generally reach some level of recall at which precision begins to drop. This happens because at some point in the learning process the algorithm picks up patterns that are common in S, but are not sufficiently specific to S alone. These patterns then pick up irrelevant documents, and precision drops.

Our goal is to prevent this kind of degradation, by helping the learner stop when precision is still high, while achieving maximal recall.

2.2 Related Work

We briefly mention some of the unsupervised methods for acquiring knowledge for NL understanding, in particular in the context of IE. A typical architecture for an IE system includes knowledge bases

(KBs), which must be customized when the system is ported to new domains. The KBs cover different *levels*, viz. a lexicon, a semantic conceptual hierarchy, a set of patterns, a set of inference rules, a set of logical representations for objects in the domain. Each KB can be expected to be domain-specific, to a greater or lesser degree.

Among the research that deals with automatic acquisition of knowledge from text, the following are particularly relevant to us. (Strzalkowski and Wang, 1996) proposed a method for learning concepts belonging to a given semantic class. (Riloff and Jones, 1999; Riloff, 1996; Yangarber et al., 2000) present different combinations of learners of patterns and concept classes specifically for IE.

In (Riloff, 1996) the system AutoSlog-TS learns patterns for filling an individual slot in an event template, while simultaneously acquiring a set of lexical elements/concepts eligible to fill the slot. AutoSlog-TS, does not require a pre-annotated corpus, but does require one that has been split into subsets that are relevant vs. non-relevant subsets to the scenario.

(Yangarber et al., 2000) attempts to find extraction patterns, without a pre-classified corpus, starting from a set of seed patterns. This is the basic unsupervised learner on which our approach is founded; it is described in the next section.

3 Algorithm

We first present the basic algorithm for pattern acquisition, similar to that presented in (Yangarber et al., 2000). Section 3.3 places the algorithm in the framework of counter-training.

3.1 Pre-processing

Prior to learning, the training corpus undergoes several steps of pre-processing. The learning algorithm depends on the fundamental redundancy in natural language, and the pre-processing the text is designed to reduce the sparseness of data, by reducing the effects of phenomena which mask redundancy.

Name Factorization: We use a name classifier to tag all proper names in the corpus as belonging to one of several categories—*person*, *location*, and *organization*, or as an *unidentified* name. Each name is replaced with its category label, a single token.

The name classifier also factors out other out-of-vocabulary (OOV) classes of items: dates, times, numeric and monetary expressions. Name classification is a well-studied subject, e.g., (Collins and Singer, 1999). The name recognizer we use is based on lists of common name markers—such as personal titles (Dr., Ms.) and corporate designators (Ltd., GmbH)—and hand-crafted rules.

Parsing: After name classification, we apply a general English parser, from Conexor Oy, (Tapanainen and Järvinen, 1997). The parser recognizes the name tags generated in the preceding step, and treats them as atomic. The parser's output is a set of syntactic dependency trees for each document.

Syntactic Normalization: To reduce variation in the corpus further, we apply a tree-transforming program to the parse trees. For every (non-auxiliary) verb heading its own clause, the transformer produces a corresponding active tree, where possible. This converts for passive, relative, subordinate clauses, etc. into active clauses.

Pattern Generalization: A "primary" tuple is extracted from each clause: the verb and its main arguments, subject and object.

The tuple consists of three literals `[s,v,o]`; if the direct object is missing the tuple contains in its place the subject complement; if the object is a subordinate clause, the tuple contains in its place the head verb of that clause.

Each primary tuple produces three *generalized* tuples, with one of the literals replaced by a wildcard. A pattern is simply a primary or generalized tuple. The pre-processed corpus is thus a many-many mapping between the patterns and the document set.

3.2 Unsupervised Learner

We now outline the main steps of the algorithm, followed by the formulas used in these steps.

1. Given: a *seed* set of patterns, expressed as primary or generalized tuples.

2. Partition: divide the corpus into relevant vs. non-relevant documents. A document d is *relevant*—receives a weight of 1—if some seed matches d, and non-relevant otherwise, receiving weight 0. After the first iteration, documents are assigned relevance weights between 0 and 1. So at each iteration, there is a distribution of relevance weights on the corpus, rather than a binary partition.

3. Pattern Ranking: Every pattern appearing in a relevant document is a *candidate* pattern. Assign a score to each candidate; the score depends on how accurately the candidate predicts the relevance of a document, with respect to the current weight distribution, and on how much *support* it has—the total wight of the relevant documents it matches in the corpus (in Equation 2). Rank the candidates according to their score. On the i-th iteration, we select the pattern p_i most correlated with the documents that have high relevance. Add p_i to the growing set of seeds $\{p_i\}$, and record its accuracy.

4. Document Relevance: For each document d covered by any of the accepted patterns in $\{p_i\}$, recompute the relevance of d to the target scenario S, $Rel^S(d)$. Relevance of d is based on the cumulative accuracy of patterns from $\{p_i\}$ which match d.

5. Repeat: Back to **Partition** in step 2. The expanded pattern set induces a new relevance distribution on the corpus. Repeat the procedure as long as learning is possible.

The formula used for scoring candidate patterns in step 3 is similar to that in (Riloff, 1996):

$$Score(p) = \frac{Sup(p)}{|H|} \cdot \log Sup(p) \qquad (1)$$

where $H = H(p)$ are documents where p matched, and the support $Sup(p)$ is computed as the sum of their relevance:

$$Sup(p) = \sum_{d \in H(p)} Rel(d) \qquad (2)$$

Document relevance is computed as in (Yangarber et al., 2000)

$$Rel(d) = 1 - \prod_{p \in K(d)} \Big(1 - Prec(p)\Big) \qquad (3)$$

where $K(d)$ is the set of accepted patterns that match d; this is a rough estimate of the likelihood of relevance of d, based on the pattern accuracy measure. Pattern accuracy, or precision, is given by the average relevance of the documents matched by p:

$$Prec(p) = \frac{Sup(p)}{|H|} = \frac{1}{|H|} \sum_{d \in H(p)} Rel(d) \qquad (4)$$

Equation 1 can therefore be written simply as:

$$Score(p) = Prec(p) \cdot \log Sup(p) \qquad (5)$$

3.3 Counter-Training

The two terms in Equation 5 capture the trade-off between precision and recall. As mentioned in Section 2.1, the learner running in isolation will eventually acquire patterns that are too general for the scenario, which will cause it to assign positive relevance to non-relevant documents, and learn more irrelevant patterns. From that point onward pattern accuracy will decline.

To deal with this problem, we arrange n different learners, for n different scenarios $\{S_i\}, i = 1..n$ to train simultaneously on each iteration. Each learner stores its own bag of good patterns, and each assigns its own relevance, $Rel^{S_i}(d)$, to the documents. Documents that are "ambiguous" will have high relevance in more than one scenario.

Now, given multiple learners, we can refine the measure of pattern precision in Eq. 4 for scenario S_i, to take into account the *negative* evidence—i.e., how much weight the documents matched by the pattern received in other scenarios:

$$Prec(p) = \frac{1}{|H|} \sum_{d \in H(p)} \Big(Rel^{S_i}(d) - \sum_{j \neq i} Rel^{S_j}(d) \Big) \qquad (6)$$

If $Prec(p) \leq 0$ the candidate is not considered for acceptance. Equations 6 and 5 imply that the learner will disfavor a pattern if it has too much opposition from other scenarios.

The algorithm proceeds as long as two or more scenarios are still learning patterns. When the number of surviving scenarios drops to one, learning terminates, since, running unopposed, the surviving scenario is may start learning non-relevant patterns which will degrade its precision.

Scenarios may be represented with different density within the corpus, and may be learned at different rates. To account for this, we introduce a parameter, N: rather than acquiring a single pattern on each iteration, each learner may acquire up to N patterns (3 in this paper), as long as their scores are near (within 5% of) the top-scoring pattern.

4 Experiments

We tested the algorithm on documents from the Wall Street Journal (WSJ). The training corpus consisted of 15,000 articles from 3 months between 1992 and

Table 1: Scenarios in Competition

Scenario	*Seed Patterns*	*# Documents*	*Last Iteration*
Management Succession	[Company *appoint* Person] [Person *quit*]	220	143
Merger&Acquisition	[*buy* Company] [Company merge]	231	210
Legal Action	[*sue* Organization] [bring/settle *suit*]	169	132
Bill/Law Passing	[*pass bill*]	89	79
Political Election	[run/win/lose election/campaign]	42	24
Sports Event	[run/win/lose competition/event]	25	19
Layoff	[expect/*announce* layoff]	43	15
Bankruptcy	[file/declare bankruptcy]	7	4
Natural Disaster	[*disaster* kill/damage people/property]	16	0
Don't Care	[cut/raise/lower rate] [report/post earning]	413	—

1994. This included the MUC-6 training corpus of 100 tagged WSJ articles (from 1993).

We used the scenarios shown in Table 1 to compete with each other in different combinations. The seed patterns for the scenarios, and the number of documents initially picked up by the seeds are shown in the table.[2] The seeds were kept small, and they yielded high precision; it is evident that these scenarios are represented to a varying degree within the corpus.

We also introduced an additional "negative" scenario (the row labeled "Don't care"), seeded with patterns for earnings reports and interest rate fluctuations.

The last column shows the number of iterations before learning stopped. A sample of the discovered patterns[3] appears in Table 2.

For an *indirect* evaluation of the quality of the learned patterns, we employ the *text-filtering* evaluation strategy, as in (Yangarber et al., 2000). As a by-product of pattern acquisition, the algorithm acquires a set of relevant documents (more precisely, a distribution of document relevance weights). Rather than inspecting patterns $\{p_i\}$ on the i-th iteration by hand, we can judge the quality of this pattern set based on the quality of the documents that the patterns $\{p_i\}$ match. Viewed as a categorization task on a set of documents, this is similar to the text-filtering task in the MUC competitions. We use the text-filtering power of the set $\{p_i\}$ as a *quantitative* measure of the goodness of the patterns.

Management Succession
demand/announce resignation
Person succeed/replace person
Person continue run/serve
Person continue/serve/remain/step-down chairman
Person retain/leave/hold/assume/relinquish post
Company hire/fire/dismiss/oust Person
Merger&Acquisition
Company plan/expect/offer/agree buy/merge
complete merger/acquisition/purchase
agree sell/pay/acquire
get/buy/take-over business/unit/interest/asset
agreement creates company
hold/exchange/offer unit/subsidiary
Legal Action
deny charge/wrongdoing/allegation
appeal ruling/decision
settle/deny claim/charge
judge/court dismiss suit
Company mislead investor/public

Table 2: Sample Acquired Patterns

To conduct the text-filtering evaluation we need a *binary* relevance judgement for each document. This is obtained as follows. We introduce a cutoff threshold θ_{rel} on document relevance; if the system has *internal* confidence of more than θ_{rel} that a document d is relevant, it labels d as relevant *externally*

[2] Capitalized entries refer to Named Entity classes, and italicized entries refer to small classes of synonyms, containing about 3 words each; e.g., *appoint* = {appoint, name, promote}.

[3] The algorithm learns hundreds of patterns; we present a sample to give the reader a sense of their shape and content.

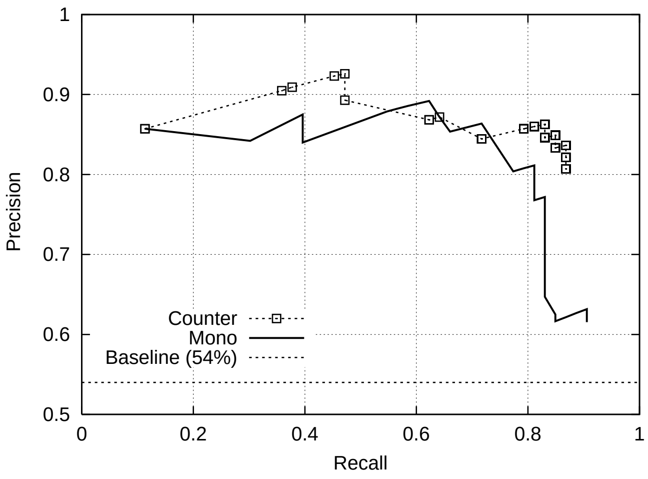

Figure 1: Management Succession

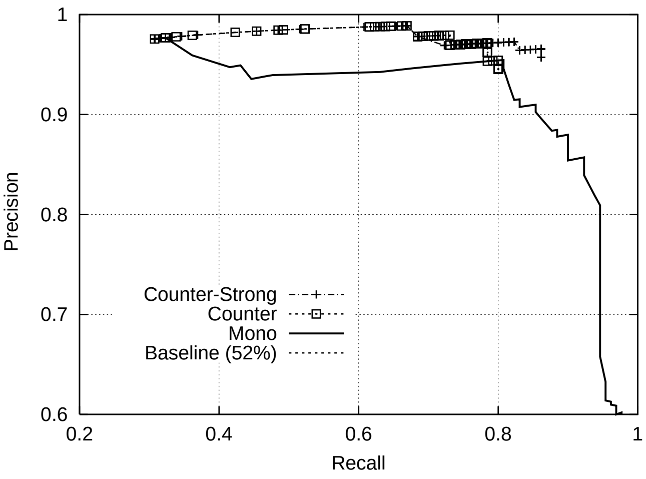

Figure 2: Legal Action/Lawsuit

for the purpose of scoring recall and precision. Otherwise it labels d as non-relevant.[4]

The results of the pattern learner for the "Management Succession" scenario, with and without counter-training, are shown in Figure 1. The test sub-corpus consists of the 100 MUC-6 documents.

The initial seed yields about 15% recall at 86% precision. The curve labeled *Mono* shows the performance of the baseline algorithm up to 150 iterations. It stops learning good patterns after 60 iterations, at 73% recall, from which point precision drops.

The reason the recall appears to continue improving is that, after this point, the learner begins to acquire patterns describing *secondary* events, derivative of or commonly co-occurring with the focal topic. Examples of such events are fluctuations in stock prices, revenue estimates, and other common business news elements.

The *Baseline* 54% is the precision we would expect to get by randomly marking the documents as relevant to the scenario.

The performance of the Management Succession learner counter-trained against other learners is traced by the curve labeled *Counter*. It is important to recall that the counter-trained algorithm *terminates* at the final point on the curve, whereas the mono-trained case it does not.

We checked the quality of the discovered patterns by hand. Termination occurs at 142 iterations. We observed that after iteration 103 only 10% of the patterns are "good", the rest are secondary. However, in the first 103 iterations, over 90% of the patterns are good Management Succession patterns.

In the *same* experiment the behaviour of the learner of the "Legal Action" scenario is shown in Figure 2. The test corpus for this learner consists of 250 documents: the 100 MUC-6 training documents and 150 WSJ documents which we retrieved using a set of keywords and categorized manually. The curves labeled *Mono*, *Counter* and *Baseline* are as in the preceding figure.

We observe that the counter-training termination point is near the mono-trained curve, and has a good recall-precision trade-off. However, the improvement from counter-training is less pronounced here than for the Succession scenario. This is due to a subtle interplay between the combination of scenarios, their distribution in the corpus, and the choice of seeds. We return to this in the next section.

5 Discussion

Although the results we presented here are encouraging, there remains much research, experimentation and theoretical work to be done.

Ambiguity and Document Overlap

When a learner runs in isolation, it is in a sense undergoing "mono-training": the only evidence it

[4] The relevance cut-off parameter, θ_{rel} was set to 0.3 for mono-trained experiments, and to 0.2 for counter-training. These numbers were obtained from empirical trials, which suggest that a lower confidence is acceptable in the presence of negative evidence. Internal relevance measures, $Rel(d)$, are maintained by the algorithm, and the external, binary measures are used only for evaluation of performance.

has on a given iteration is derived from its own guesses on previous iterations. Thus once it starts to go astray, it is difficult to set it back on course.

Counter-training provides a framework in which other recognizers, training in parallel with a given recognizer R, can label documents as belonging to their own, other categories, and therefore as being *less likely* to belong to R's category. This likelihood is proportional to the amount of anticipated ambiguity or overlap among the counter-trained scenarios.

We are still in the early stages of exploring the space of possibilities provided by this methodology, though it is clear that it is affected by several factors. One obvious contributing factor is the choice of seed patterns, since seeds may cause the learner to explore different parts of the document space first, which may affect the subsequent outcome.

Another factor is the particular combination of competing scenarios. If two scenarios are very close—i.e., share many semantic features—they will inhibit each other, and result in lower recall. This closeness will need to be qualified at a future time.

There is "ambiguity" both at the level of documents as well as at the level of patterns. Document ambiguity means that some documents cover more than one topic, which will lead to high relevance scores in multiple scenarios. This is more common for longer documents, and may therefore disfavor patterns contained in such documents.

An important issue is the extent of overlap among scenarios: Management Succession and Mergers and Acquisitions are likely to have more documents in common than either has with Natural Disasters.

Patterns may be pragmatically or semantically ambiguous; "*Person died*" is an indicator for Management Succession, as well as for Natural Disasters. The pattern "*win race*" caused the sports scenario to learn patterns for political elections.

Some of the chosen scenarios will be better represented in the corpus than others, which may block learning of the under-represented scenarios.

The scenarios that are represented well may be learned at different *rates*, which again may inhibit other learners. This effect is seen in Figure 2; the Lawsuit learner is inhibited by the other, stronger scenarios. The curve labeled *Counter-Strong* is obtained from a separate experiment. The Lawsuit learner ran against the same scenarios as in Table 1, but some of the other learners were "weakened": they were given smaller seeds, and therefore picked up fewer documents initially.[5] This enabled them to provide sufficient guidance to the Lawsuit learner to maintain high precision, without inhibiting high recall. The initial part of the curve is difficult to see because it overlaps largely with the *Counter* curve. However, they diverge substantially toward the end, above the 80% recall mark.

We should note that the objective of the proposed methodology is to learn good patterns, and that reaching for the maximal document recall may not necessarily serve the same objective.

Finally, counter-training can be applied to discovering knowledge of other kinds. (Yangarber et al., 2002) presents the same technique successfully applied to learning names of entities of a given semantic class, e.g., diseases or infectious agents.[6] The main differences are: a. the data-points in (Yangarber et al., 2002) are instances of names in text (which are to be labeled with their semantic categories), whereas here the data-points are documents; b. the intended product there is a list of categorized names, whereas here the focus is on the patterns that categorize documents.

(Thelen and Riloff, 2002) presents a very similar technique, in the same application as the one described in (Yangarber et al., 2002).[7] However, (Thelen and Riloff, 2002) did not focus on the issue of convergence, and on leveraging negative categories to achieve or improve convergence.

Co-Training

The type of learning described in this paper differs from the co-training method, covered, e.g., in (Blum and Mitchell, 1998). In co-training, learning centers on labeling a set of data-points in situations where these data-points have multiple disjoint and redundant *views*.[8] Examples of spaces of such data-points are strings of text containing proper names, (Collins and Singer, 1999), or Web pages relevant to a query

[5] The seeds for Management Succession and M&A scenarios were reduced to pick up fewer than 170 documents, each.

[6] These are termed *generalized names*, since they may not abide by capitalization rules of conventional proper names.

[7] The two papers appeared within two months of each other.

[8] A view, in the sense of relational algebra, is a sub-set of features of the data-points. In the cited papers, these views are exemplified by internal and external contextual cues.

(Blum and Mitchell, 1998).

Co-training iteratively trains, or refines, two or more n-way classifiers.[9] Each classifier utilizes only one of the views on the data-points. The main idea is that the classifiers can start out weak, but will strengthen each other as a result of learning, by labeling a growing number of data-points based on the mutually independent sets of evidence that they provide to each other.

In this paper the context is somewhat different. A data-point for each learner is a single document in the corpus. The learner assigns a binary label to each data-point: relevant or non-relevant to the learner's scenario. The classifier that is being trained is embodied in the set of acquired patterns. A data-point can be thought of having one view: the patterns that match on the data-point.

In both frameworks, the unsupervised learners help one another to bootstrap. In co-training, they do so by providing reliable positive examples to each other. In counter-training they proceed by finding their own weakly reliable positive evidence, and by providing each other with reliable *negative* evidence. Thus, in effect, the unsupervised learners "supervise" each other.

6 Conclusion

In this paper we have presented counter-training, a method for strengthening unsupervised strategies for knowledge acquisition. It is a simple way to combine unsupervised learners for a kind of "mutual supervision", where they prevent each other from degradation of accuracy.

Our experiments in acquisition of semantic patterns show that counter-training is an effective way to combat the otherwise unlimited expansion in unsupervised search. Counter-training is applicable in settings where a set of data points has to be categorized as belonging to one or more target categories.

The main features of counter-training are:

- Training several simple learners in parallel;
- Competition among learners;
- Convergence of the overall learning process;
- Termination with good recall-precision trade-off, compared to the single-trained learner.

[9] The cited literature reports results with exactly two classifiers.

Acknowledgements

This research is supported by the Defense Advanced Research Projects Agency as part of the Translingual Information Detection, Extraction and Summarization (TIDES) program, under Grant N66001-001-1-8917 from the Space and Naval Warfare Systems Center San Diego, and by the National Science Foundation under Grant IIS-0081962.

References

A. Blum and T. Mitchell. 1998. Combining labeled and unlabeled data with co-training. In *Proc. 11th Annl. Conf Computational Learning Theory (COLT-98)*, New York.

M. Collins and Y. Singer. 1999. Unsupervised models for named entity classification. In *Proc. Joint SIGDAT Conf. on EMNLP/VLC*, College Park, MD.

E. Riloff and R. Jones. 1999. Learning dictionaries for information extraction by multi-level bootstrapping. In *Proc. 16th Natl. Conf. on AI (AAAI-99)*, Orlando, FL.

E. Riloff. 1996. Automatically generating extraction patterns from untagged text. In *Proc. 13th Natl. Conf. on AI (AAAI-96)*.

T. Strzalkowski and J. Wang. 1996. A self-learning universal concept spotter. In *Proc. 16th Intl. Conf. Computational Linguistics (COLING-96)*, Copenhagen.

P. Tapanainen and T. Järvinen. 1997. A non-projective dependency parser. In *Proc. 5th Conf. Applied Natural Language Processing*, Washington, D.C.

M. Thelen and E. Riloff. 2002. A bootstrapping method for learning semantic lexicons using extraction pattern contexts. In *Proc. 2002 Conf. Empirical Methods in NLP (EMNLP 2002)*.

R. Yangarber, R. Grishman, P. Tapanainen, and S. Huttunen. 2000. Automatic acquisition of domain knowledge for information extraction. In *Proc. 18th Intl. Conf. Computational Linguistics (COLING 2000)*, Saarbrücken.

R. Yangarber, W. Lin, and R. Grishman. 2002. Unsupervised learning of generalized names. In *Proc. 19th Intl. Conf. Computational Linguistics (COLING 2002)*, Taipei.

D. Yarowsky. 1995. Unsupervised word sense disambiguation rivaling supervised methods. In *Proc. 33rd Annual Meeting of ACL*, Cambridge, MA.