Active Learning for Information Extraction via Bootstrapping

Andrew Carlson, Sue Ann Hong, Kevin Killourhy, Sophie Wang

10-709 Read the Web
Carnegie Mellon University

1 Problem definition

Text learning algorithms are reasonably successful when provided with enough labeled or annotated training examples. For instance, text classifiers [13, 10, 21, 4, 18] reach high accuracy from large sets of class-labeled documents; information extraction algorithms [3, 15, 19, 8] perform well when given many tagged documents or large sets of rules as input. However, creating these training sets becomes tedious and expensive, since typically they must be labeled by a person.

To learn from a large amount of unlabeled content, bootstrap learning algorithms have been proposed. Bootstrap learning is a form of semi-supervised learning where the algorithm starts with a small set of labeled points commonly referred to as *seeds* and performs a task such as classification with minimal intervention or labeling onward.

In our work, we consider extracting facts from a corpus; given a relation or a predicate, the task is to find entities that satisfy the relation or predicate. For example, for predicate *is-a-nation*, we would like to extract strings representing nations such as *United States* or *Japan*. We use the bootstrap setting introduced in the KnowItAll system by Etzioni et al.[7], where they employ an iterative approach that alternates between extracting rules from a set of entities, and finding entities from the newly extracted set of rules. Although a rule can be of any form, e.g., an incremental bag-of-words classifier that uses the words around an entity, or even a pre-trained entity classifier, for simplicity we examine only rules matching an exact sequence of tokens. We believe more sophisticated or general forms of rules can increase the performance of the bootstrapping algorithm, but this issue is outside our scope of problem.

One major weakness of such bootstrap algorithms is the tendency to decrease in precision over time. Because there is no oracle labeling examples after the initial seeding, the algorithm can rapidly diverge from the target relation. In our setting, it is typical for a bootstrapper to extract many entities that do not satisfy the given predicate or relation, or to become unable to extract any entities or rules. We identified the main cause of this deviation problem to be the equal treatment of every rule and entity; the plain bootstrap algorithm uses all entities and rules for further iterations and assumes all extractions to be true, without assigning any confidence scores. If a rule is too general, it will extract many entities not satisfying the predicate or relation; for example, *I flew to* will not only extract names of nations but also of other locations such as states and cities, thereby lowering the precision of extracted examples. If a rule is too specific, it will not extract many entities other than the entity that extracted the rule; *I love the beautiful snow-covered slopes of the Rockies in* will not extract any nations apart from *Canada* and *United States*.

There exists a handful of previous work addressing the deviation problem by scoring rules and entities. [7] presents multiple scoring metrics for the KnowItAll system, such as pointwise mutual information (PMI), the Noisy-Or model, and the Urns model. We revisit these methods and compare their influence on the quality of extractions by bootstrapping. Furthermore, we study the value of feedback in bootstrap extraction using active learning. We conjecture that active learning can prevent divergence of the bootstrapper with a small amount of user feedback – far less than the effort required to label a large pool of examples.

Hence our work involves two main goals. The first is to compare the precision and recall of a set of scoring methods for rules and entities, and to examine the effect of active learning on those methods. The second is to examine the sensitivity of active learning by investigating issues such as how many questions are required

for active learning to be useful, at what point in the life of a bootstrapper can active learning prove helpful, and what kinds of active learning strategies provide the most improvement.

2 Related work

Bootstrap learning is closely related to co-training [2], which alternately learns using two orthogonal views of the data. Jones et al.[9] gives a good overview of methods used in bootstrap learning. Information extraction systems that use bootstrapping include [12, 1, 5, 17, 14]. These systems begin with a set of hand-tagged seed entities, then alternately learn rules from seeds, and further seeds from rules.

Furthermore, Etzioni et al. proposed the KnowItAll system, which does not require hand-tagged seeds, but instead begins with a domain-independent set of generic extraction patterns from which it induces a set of seed entities [7]. For example, the generic pattern "NP1 such as NPList2" indicates that the head of each simple noun phrase (NP) in the list NPList2 is a member of the class named in NP1. By instantiating the rule for the class *is-a-city*, KnowItAll extracts three candidate cities from the sentence, "We provide tours to cities such as Paris, London, and Berlin."

Previous work has explored inactive methods of scoring facts extracted by bootstrap learning systems. The KnowItAll system used pointwise mutual information, a method of assessing the truth of a relation using Web queries, to find confidence scores for extracted facts. Downey et al.[6] expanded upon this work by using probabilistic models such as Noisy-Or and the Urns model.

Jones [11] explores methods of incorporating active learning in a bootstrap learning system in the co-testing setting. Her work finds that active learning can help improve the precision of a bootstapped named entity recognizer.

3 Approach

In order to investigate the effect of active learning on the bootstrapping process, we conduct an evaluation of different scoring methods and measure the effect that active learning has on each. Four different scoring algorithms were developed. Each was adapted to use one or more different active learning strategies. A standard bootstrapping algorithm was developed to act as a common environment for testing these scoring algorithms, and they were evaluated on their performance in a simple bootstrapping task.

3.1 Confidence scoring methods

The role of scoring methods in bootstrap learning is twofold. First, as the bootstrapping algorithm iteratively uses the extraction rules and entities that it has already found to learn new entities and rules, the scores assigned to each rule and entity drives the selection of new ones. High scoring rules and entities are assumed to be better at finding other high-scoring ones. Second, after the bootstrapping process has finished, a list is generated of those entities most likely to belong to the target relation according to their scores. Since active learning may provide different benefits to different scoring methods, we did not confine our investigation to the effect on a single method. Instead, a diverse sample of four methods was chosen, some taken from existing work, others designed and developed for this application. Specifically, they are pointwise mutual information, Basilisk, Noisy-Or, and Pagerank. Additionally, in order that these methods might be compared against a common baseline, a fifth "straw man" scoring method called Uniform Scorer was developed.

3.1.1 Pointwise mutual information (PMI)

Etzioni et al.[7] demonstrated that the PMI of an entity and a relation can be approximated using the statistics maintained by vast Web indexing systems such as Google. Specifically, the PMI between an entity

e and a relation R is approximately proportional to a statistic involving the estimated hit counts returned by Google.

$$\text{PMI}(e, R) = \frac{\text{HitCount(e + R)}}{\text{HitCount}(e)}$$

The ratio of the number of hits for both the entity and the relation to the total number of hits for the entity is a measure of the strength of association between the two. Note that using a strategy suggested by Etzioni et al., we treat the expression $e + R$ to indicate the composite search string "e is a R" (e.g., "Australia is a Nation").

Each confidence scoring method can be reduced to two formulas: one scoring entities, the other scoring extraction rules. The first formula expresses the score associated with an entity e and a relation R. We directly use $\text{PMI}(e, R)$ for this score. While the PMI score can range between zero and one, it is usually notoriously close to zero[7]. However, we are not performing any arithmetic on the scores, simply comparing them to one another. As such, the raw score seems sufficient. Note that in the event $\text{HitCount}(e) = 0$, we consider the PMI to be zero. The second of the two formulas expresses the score associated with an extraction rule r and a relation R. We compute the score for a rule r and a relation R using the score for the entities $e_1, \ldots, e_n$ associated with the rule. Specifically, we compute the scores $s_1, \ldots, s_n$ for each entity $e_1, \ldots, e_n$ and relation R, and we take the average $\bar{s}$ over $s_1, \ldots, s_n$ as the score for the rule r and relation R. We say that an entity is associated with a rule whenever the two co-occur. A rule and entity can co-occur either because the rule extracts the entity or because, when searching for the entity, the rule occurs as a nearby phrase. (While seemingly arbitrary, the latter association exists because it is often used in the bootstrapping of new rules from existing entities as described in Section 3.3.)

3.1.2 Basilisk

Basilisk (Bootstrapping Approach to Semantic Lexicon Induction using Semantic Knowledge) is a weakly supervised mutual bootstrapping algorithm that automatically generates semantic lexicons [20]. Riloff's research group has done much previous work on bootstrap learning in Information Retrieval and the Basilisk algorithm is their most recent one. The Basilisk algorithm starts from one seed entity, which may be too weak for our data. To obtain a more robust algorithm, we begin the algorithm with a small set of seed entities. The first step in the Basilisk bootstrapping process is to extract a set of candidate rules from the Web using an entity pool (initialized with seed entities). Among all extracted rules, we select the n rules most likely to belong to the target class (e.g., nations) according to a predefined score function. They employ the $R \log F$ scoring metric which was previously proposed by AutoSlog-TS[16]. It is defined as

$$score(rule_i) = \frac{F_i}{N_i} log_2(F_i)$$

where F_i is the number of unique class members in the entity pool, and N_i is the total number of entities extracted by $rule_i$. Intuitively, the $R \log F$ metric tries to strike a balance between reliability and frequency: $\frac{F_i}{N_i}$ is high when the rules extractions are highly correlated with the class, and F_i is high when the rule extracts a large number of class members. After scoring, we select the top 5 rules and add them to a pool of all accepted extracted rules. The next step is to extract entities by the rules in the rule pool and score the entities using another score function. An entity is scored according to the number of unique rules that extracted it. Intuitively, an entity extracted by three different rules is more likely to belong to the class than an entity extracted by a single rule. For tie-breaking purposes, we also introduce a small factor to represent the strength of the rules that extracted it. The scoring metric for entities is formally defined as:

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

The main advantage of Basilisk bootstrapping comes from re-evaluating the extraction rules after each bootstrapping iteration. For example, after the first bootstrapping run, if five new entities are added to the permanent entity pool, bootstrapping is restarted from scratch with the original seed entities plus these five new entities. Now the best rule selected by bootstrapping might be different from the best rule selected last time. This procedure has a snowball effect because, in each iteration, new extractions are added to the temporary class used to choose rules for subsequent iterations. In particular, more general rules seem to float to the top as the permanent class member grows.

3.1.3 Noisy-Or

The Noisy-Or model [6] assumes that each time a rule extracts an entity, it is an independent event. Let p_r denote the precision of rule r. The precision is the probability that a random extraction from the rule will be a positive example of the target relation Let c_{er} denote the number of times entity e is extracted by rule r. The probability that an entity extracted by a single rule is a positive example of the target relation is $p(relation(e)|p_r, c_{er}) = 1 - (1 - p_r)^{c_{er}}$. It is the probability that rule r was incorrect c_{er} times. Seed rules are assumed to have a precision $p = 0.90$, while other rules are assumed to have an initial precision $p = 0.50$.

When we take multiple rules into account, the probability that an entity is a positive example of the target relation becomes $p(relation(e)|p, c) = 1 - \prod_{r \in Rules}(1 - p_r)^{c_{er}}$.

3.1.4 Pagerank

We chose to implement a variant of the Pagerank algorithm in order to address our intuition that the goodness of a node should reflect the goodness of its neighbors and of their neighbors and so on. The Pagerank scorer is an iterative algorithm that repeats the scoring scheme described below until a fixed point.

Consider the extraction data in the following graph form of $G = (V, E)$. Let each rule or entity be a node $v_i \in V$. Then, there exists an edge between nodes v_i and v_j if v_i extracted v_j or if v_j extracted v_i. For each independent event of extraction between v_i and v_j from a unique span, where span is defined to be a tuple of document and the location in the document, we add an edge between v_i and v_j. Hence the number of edges between v_i and v_j represents how many times the two strings co-occur in the corpus. Note that following the original Pagerank scheme one would employ the directed form of the graph (i.e., the edge from v_i to v_j is treated differently an edge from v_j to v_i). However, we consider the undirected version. Theoretically, rule and entity extraction must be symmetric; given the same corpus, if a rule extracts an entity then the entity must also extract the rule. When bootstrapping within the World Wide Web using Google, this assumption fails due to the asymmetry of top pages returned by Google given two different query strings. However, since we consider learning from any given corpus, not the top pages per query, the undirected version seems more adequate.

For each entity or rule i and relation R, the confidence score $S(i, R)$ is calculated by:

1. If the entity or rule i is a seed for relation R, $S(i, R) = 1$.
2. If the entity or rule i is labeled (answered) by the user to be true for the relation R, $S(i, R) = 1$.
3. If the entity or rule i is labeled (answered) by the user to be false for the relation R, $S(i, R) = 0$.
4. Otherwise,
$$S(i, R) = \sum_{j,(i,j)\in E} S(j, R) Pr(j \to i),$$
where
$$Pr(j \to i) = \frac{\text{Number of edges between i and j}}{\sum_{x \in V} \text{Number of edges between x and j}}.$$

Note here that the original Pagerank algorithm has an additional term in the scoring equation:

$$S(i, R) = a \sum_{j,(i,j)\in E} S(j, R) Pr(j \to i) + (1 - a)/|V|,$$

where $|V|$ is the number of nodes in the graph. One may interpret the additional term as giving a small confidence score to a node for existing in the graph (after all, being extracted at all is a stronger evidence of the entity's truthfulness to the predicate than not being extracted). However, we used $a = 1$ because the Markov random walk model does not fit the scoring problem, and we believe the additional term's smoothing of scores will make the scoring scheme less discriminating.

When a new rule or an entity is extracted, it is given a score of $1/N$ for each relation, where N is the number of relations considered in the bootstrapper. Since our experiments only consider predicates of the form *is-a-X*, the set of relations become {*is-a-X*, *is-not-a-X*}, and the initial score 0.5.

3.1.5 Uniform Scorer

The Uniform Scorer algorithm simply assigns a constant score, uniformly to all rules and entities. It was developed in order that the scoring methods described above could be compared against a common baseline. Its performance is a measure of what can be done by a bootstrapping algorithm without any useful scoring strategy.

3.2 Active learning methods

The user's time is valuable. To make good use of labeling effort, it is important to judiciously select examples presented to the user for labeling. We want to select examples which will steer the bootstrapper in the right direction, so that it does not diverge from the target relation. A good overview of strategies for active learning in a bootstrap setting is given by Jones[11].

1. *Random* selection chooses entities randomly. The intuition is that we want the user to give input on a wide variety of entities, including positive and negative examples.
2. *Density* selection chooses entities for labeling based on how many times they are extracted by rules. We might consider selecting the entities which occur most frequently across all rules, since the answers given by the user will tend to affect confidence scores more than entities which occur rarely. We might also consider the least frequently occurring entities, since they are more likely to be incorrect.
3. *Score-based* selection chooses entities for labeling based on their scores (as estimated by some scoring method). We might consider entities with the topmost or bottommost scores, because inaccuracies in their scores would be most detrimental to the scoring method's performance.

Each of our scoring methods was adapted to use these strategies. For each scoring method, several variants were created. In addition to the unsupervised scoring method described in the previous section (which we call the *passive* variant to distinguish it from the active variants), a *random* variant was developed for each strategy. A *most-frequent density selection* variant was developed for each of PMI, Noisy-Or, and Pagerank. A *least-frequent density selection* variant was developed for Noisy-Or and Pagerank. A *topmost score-based* variant was developed for Basilisk.

In addition to selecting which questions to ask a user, an active learning strategy must include a procedure for incorporating the labels provided by the user back into the scoring function. The reason a scoring function must approximate the association between an entity and a relation is because the actual association is unknown. If that uncertainty is removed, e.g., because the user informs the scorer that an association exists or it does not, then there is no need to approximate the score. In every case, if the user confirms that an entity belongs to a relation, we set the score to 1.0. If an entity does not belong to a relation, we set the score to 0.0.

Labeled data affect not just entity scores but rule scores as well. The PMI and Pagerank methods simply use the newly computed entity scores into their rule scores. (E.g., PMI calculates the score of a rule as the average score of the entities that are associated with it.) The Basilisk algorithm, where rule scores are based on those entities in the entity pool, ensures that entities with positive labels always appear in the pool while entities with negative labels are always excluded. Noisy-Or uses the answers of questions to estimate the precision for each rule. Let E_p denote the entities labeled positively by the user, and E_n denote those labeled negatively. Let c_{er} denote the count of times entity e is extracted by rule r. For rule r, the precision is estimated as $p_r = \left(\sum_{e \in E_p} c_{er}\right) / \left(\sum_{e \in E_p \cup E_n} c_{er}\right)$.

3.3 Experimental settings

An experiment was designed and conducted in order to measure the effectiveness of each of these active learning strategies and scoring methods. A simple bootstrapper was created to act as a common platform on which to evaluate each of the strategies previously described. While the World Wide Web would be the natural corpus on which such a bootstrapper would operate, for the purpose of evaluation, a static corpus was created. While it is impossible to completely encapsulate the Web in a fixed corpus, we believe that

procedure SIMPLEBOOT((a list of seed extraction rules R))

Create a new scoring object $\mathcal{S}$ which can be active or passive
while (1) the list R of unsearched extraction rules is not empty and
(2) the maximum number of iterations have not been run **do**

Stage 1: Extract new entities from rules
Assign score $\mathcal{S}_R(r)$ to all unsearched rules $r \in R$.
Sort the rules in R from highest to lowest score
for all the top five scoring rules $r \in R$ **do**
Search for the top ten occurrences of r.string.
Extract all noun phrases occurring on the r.hand side of the string in the title or "snippet" text
Add each noun phrase to the list E of potential entities
Remove r from list of unsearched extraction rules

Stage 2: Extract new rules from entities
Assign score $S_E(e)$ to all entities $e \in E$.
Sort the entities in E from highest to lowest score
Let e be the highest scoring entity in E
while e is not null and fewer than ten rules have been added to R **do**
Search for the top ten occurrences of e.string
Extract all 3–5 word phrases to the left and right of the string in the title or "snippet" text
If a phrase co-occurs next to two or more entities, create an extraction rule r and add it to R.

Stage 3: Active learning (when applicable)
Get five unlabeled entities from $\mathcal{S}$ to ask the user to label
Ask the user for labels for each entity
Report the user's responses back to the scoring object $\mathcal{S}$
Assign scores $\mathcal{S}_E(e)$ to all extracted entities
Return a list of the top scoring entities

Figure 1: The complete simple bootstrapping algorithm used in the evaluation

the corpus was compiled so as to appear indistinguishable from the Web to our bootstrapping algorithm. Consequently, our evaluation environment should be realistic and yet remain controllable, both desirable characteristics for an evaluation.

3.3.1 Simple bootstrapping algorithm

The goal of the bootstrapper is to perform unsupervised information extraction, much as was done by Etzioni et al.[7] with their KNOWITALL system. Given a target relation (e.g., *is-a-nation*), the bootstrapper compiles a list of entities belonging to a that relation. It does so by issuing search queries on a corpus and analyzing the results.

The complete bootstrapping algorithm is detailed in Figure 1. It uses an iterative process with new entities being extracted every iteration. An iteration proceeds in three stages. In the first stage, a set of extraction rules are used in search queries, and the returned results are parsed in order to extract new potential entities. In the second stage, the set of extracted entities are used to perform further queries, and phrases that co-occur with multiple entities are added as potential extraction rules for future iterations. In the third stage, scoring methods with an active learning component are asked to compose queries to a teacher (i.e., the user), the teacher (or an equivalent oracle) is consulted, and the feedback is provided to the scoring object (i.e., whether the entity belongs to the target relation).

For the purpose of evaluation, ground truth is supplied to the bootstrapper in the form of an exhaustive list of those entities that belong to the target relation. Consequently, the bootstrapper automates the role of the teacher by looking for the entity in the table of true entities and providing the scoring object with the answer. Note that the scoring object's query-and-answer framework must allow the teacher to indicate that he or she does not know the answer, but this evaluation does not exercise that capability. When the scoring object does not have an active component, the third stage is omitted.

3.3.2 Building the corpus

The bootstrapping algorithm would be most suitable with a vast corpus such as that provided by the World Wide Web. It would naturally use Google's Java API or a similar mechanism to issue search queries. However, running in such an environment would compromise our attempt to conduct a controlled evaluation. Experience has shown that Google's Java API will return different results for the same query when the queries are made only minutes apart. Transient failures are frequent, e.g., server busy errors, connection timeouts.[1] Operating in such a chaotic environment, a fair comparison of two scoring algorithms is impossible. Intrinsic differences between methods could not be distinguished adequately from differences due to environmental changes between runs of the bootstrapper.

Given these issues, the decision was made to assemble a fixed corpus on which to conduct our evaluation. However, the corpus was assembled so that the observed behavior of the bootstrapper would be *realistic*, i.e., no different than if it had been running directly on the Web (except with reproducible results). It was discovered that a simple modification to the bootstrapping algorithm would turn it into a corpus-building algorithm that assembles a realistic corpus. This modification required only one key insight. Specifically, the role of the scoring algorithm in the discovery of new rules and entities is analogous to that of a search heuristic in graph traversal (in the A^* sense of "search" rather than the information retrieval sense).

During any particular iteration, a subset of the extraction rules and entities will be searched (in stages 1 and 2 respectively). Consequently, if we modify the bootstrapper slightly so that it searches all unsearched rules and all entities, and cache the results, then any query the bootstrapper might make during that iteration has already been cached. Further, these modifications do not stop the processing of the results returned by the queries, i.e., the extraction of entities and discovery of new extraction rules. In fact, by allowing this processing to proceed, we obtain a list of all the extraction rules and entities *that could be searched in the next iteration*. Informally, we are making an inductive argument that the results cached by this corpus-builder contain any result that the bootstrapper would encounter had it used the Web directly. In effect, this modification has made what was a form of heuristic search algorithm (where the heuristic was encoded in the scoring method) into a standard breadth-first search algorithm. Practically, since this corpus-builder must issue an (exponentially) increasing number of queries each iteration, we cannot run the corpus-builder forever. However, if we run the corpus-builder for n iterations, we ensure that the behavior of a bootstrapper on that corpus is realistic if it runs for no more than n iterations.

Continuing to view the role of the scoring object as that of a heuristic in graph traversal, we made one additional modification by actually encoding the corpus as a directed, bipartite graph. At a high level, there is a mapping from each extraction rule to the entities that it extracts, e.g., there might be an edge from the extraction rule *RH nations such as* to the entity *Australia*. This edge denotes the fact that in one of the snippets returned by Google, *Australia* appears as a noun phrase directly to the right of the string "nations such as." but such details are hidden from the scoring function and hence do not matter in the evaluation. Similarly, there is a mapping from an entity to an extraction rule that co-occurs with it in search results for the entity, e.g., an edge from *Australia* to the extraction rule *RH CIA The World Factbook* indicates that a search for *Australia* returns a title or snippet that contains the phrase "CIA The World Factbook"[2] Both the extraction of entities from rules and the co-occurrence of potential rules with entities can be encoded in a graph. After the graph is created (by the corpus-builder), the simple bootstrapping algorithm need only consult the graph, an operation much more efficient than loading cached web pages, parsing them, and performing part-of-speech tagging and noun phrase extraction. The initial investment of building the graph in the corpus-builder pays off both in the time the bootstrapping algorithm avoids in duplicated effort and in the actual size of the corpus. It also presents the opportunity to craft synthetic corpora and test our scoring algorithms under a range of operating conditions not easily produced in the real world.

[1] In addition to those Google API characteristics that made it undesirable, it is worth mentioning other issues that made it infeasible even to use for assembling our corpus. Queries often took several seconds to complete, and frequently raised exceptions. Some queries repeatably raised an undocumented, nonspecific exception, making it impossible to obtain results for those search queries even through multiple attempts. Results including reported hit counts were often different from those returned through Google's web page. Accessing more than 10 results required additional queries. The 1000-query-per-day restriction was an unexpected impediment because queries that resulted in an exception still counted against the daily limit.

[2] Actually, the full title is "CIA – The World Factbook – Australia" but punctuation is ignored by Google and consequently stripped by our parser.

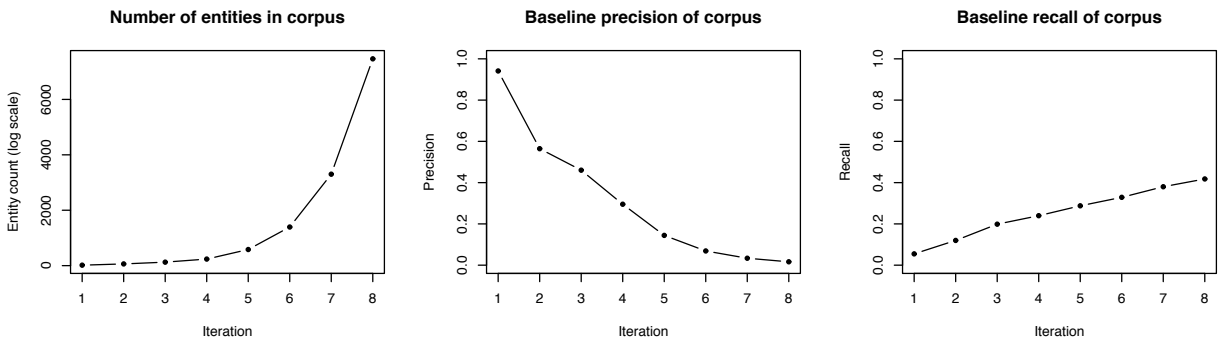

Figure 2: The number of entities in the corpus grows exponentially with the number of iterations; the precision drops super-linearly; and the recall increases linearly.

3.3.3 Nation-extraction corpus

The actual information extraction task chosen for the evaluation was the extraction of nation names. While the scoring objects support the learning and scoring of binary relations as well as unary ones, unary relations were chosen for their simplicity in this first evaluation. Further, a vetted list of actual country names was available and ready for our use,[3] another factor in favor of building a corpus around nation extraction.

Note that nation extraction is not an ideal task. From the standpoint of external validity, nation extraction is an unlikely task for a bootstrap learner. Since the number of nations is comparatively small and well known, it differs from the types of tasks for which one might naturally employ a bootstrap learner. From the standpoint of internal validity, evaluating whether an entity is a nation or not is not as black-and-white as initially believed. For instance, because lists of noun phrases are split into separate nations, conjoined nations such as *Antigua and Barbuda* will never be extracted, but *Antigua* and *Barbuda* will be extracted separately and counted against the algorithm. Likewise, typos such as *Vatican Cit* (missing a -y) do not count as nations. Further, nationhood can be a transient concept. Our list of nations contains both *Burma* (the name changed to *Myanmar*) and *Taiwan*, whose nationhood status remains a sensitive political issue.

For this evaluation, we considered the list of nations to be ground truth. We imagined it as an expression of the intentions of a particular user who did not, for instance, consider a misspelled nation to be a nation. While other users might be more lenient, our user was not. Future studies might relax some of these restrictions or, alternatively, demand that the nation be recognized as such today.

The corpus-builder was used to build a corpus for nation extraction. Because the number of nations is so small, a single seed rule was supplied to the corpus-builder, specifically *RH nations such as*. The corpus-builder was run for eight iterations. The decision was made to stop at eight simply because of time and memory constraints. Summary statistics about the growth of the corpus appear in Figure 2. As expected, the number of entities possible to extract does appear to grow exponentially in the number of iterations. The corpus-wide precision drops steeply in the first few iterations, indicating how quickly a bootstrapper with a poor scoring algorithm might diverge from the target concept. The recall is calculated as the number of nations appearing as entities in the corpus divided by the total number of nations in the ground truth list. It increases steadily and seemingly linearly in the number of iterations. The corpus-wide recall provides an upper bound on the performance of any scoring algorithm. Since the corpus only contains 47% of the nations in the ground-truth list, no scoring algorithm can extract more than 47% of the desired nation names.

3.3.4 Evaluation procedure

Our evaluation proceeds in two parts, each part designed to accomplish one of the two goals of this work. The first goal is to assess the effect of active learning on the performance of the bootstrap scoring object. In the first part of our evaluation, the bootstrap algorithm was run using each scoring strategy in turn, with all

[3]We are grateful to Mohit Kumar for providing us with this list of names.

```
RELATION            =   Nation

SCORER TYPE         =   NoisyOrScorer-MostFrequent
VERBOSE LEVEL       =   2

SEED RULE           =   RH nations such as

ENTITIES RETURNED   =   293
ITERATION MAX       =   8
RULES PER RUN       =   5
NEW RULES PER RUN   =   10
ENTITIES PER RULE   =   2
QUERIES PER RUN     =   5

CORPUS FILE         =   Experiments/nations2-corpus.txt
GROUND TRUTH        =   Experiments/nations-key.txt
RESULTS FILE        =   Experiments/nations2-PMIScorer-random/result-qnum-05.txt
```

Figure 3: Sample configuration file for the simple bootstrapping algorithm

other configuration options held fixed. Figure 3 shows the configuration file for the Noisy-Or scoring method with the most-frequent density-selection active learning strategy. All configuration options other than the scorer type and the result filename were constant across all runs. The precision and recall of the top scoring entities is calculated at the end of each iteration and recorded. While these scores alone provide us with a comparison of the relative merits of the algorithms, it is desirable to also measure their performance against a common baseline. To this end, an additional scoring object called Uniform Scorer was implemented and included in these runs. The Uniform Scoring algorithm returns a constant score for every entity and every rule. As such, its performance is a measure of what can be done by the bootstrap algorithm alone, without any useful scoring strategy.

The second goal is to determine the sensitivity of the the active learning strategies to the number of questions asked. In the second part of our evaluation, the the bootstrap algorithm was run eleven times each scoring method, each of the eleven runs used a slightly different configuration. Specifically, the number of questions that the scorer was allowed to ask in each run varied from zero to ten. In the first run, the scoring strategy was allowed to ask no questions (and as such should behave no differently than the passive strategy). In the second run, the scoring strategy was allowed to ask one question. In the third run, two questions were asked, and so forth. In the eleventh run, ten questions were asked. The precision and recall of the top-scoring entities were computed at the end of the final iteration, and the scores were recorded. Again, the Uniform Scoring algorithm was run through the same procedure to provide a common baseline.

4 Experimental results

Figure 4 shows the results of the first part of the evaluation, assessing the effect of active learning on each scoring method. Each row of the table compares the performance of each variant of the scoring method (e.g., active, passive, and random) to one another and to the baseline performance of Uniform Scorer. The results for the PMI-based strategies are in the top row, the Basilisk-based strategies in the second, Noisy-Or in the third, and Pagerank in the fourth.

The left column presents the precision results. The percentage of top-scoring results that are actually nations is plotted as a function of the number of iterations. Initial precision is high because the extractions from the seed rule are almost all true nations. The right column presents the recall results. The percentage of true nations that are among the top-scoring results are plotted as a function of the number of iterations.

Figure 5 shows the results of the second part of the evaluation, examining the sensitivity of each active learning strategy to the number of questions asked in each iteration. Again, each row focuses on the performance of all variants of a single scoring method (with Uniform Scorer's performance as a baseline). The left column presents precision results while the right column presents recall results.

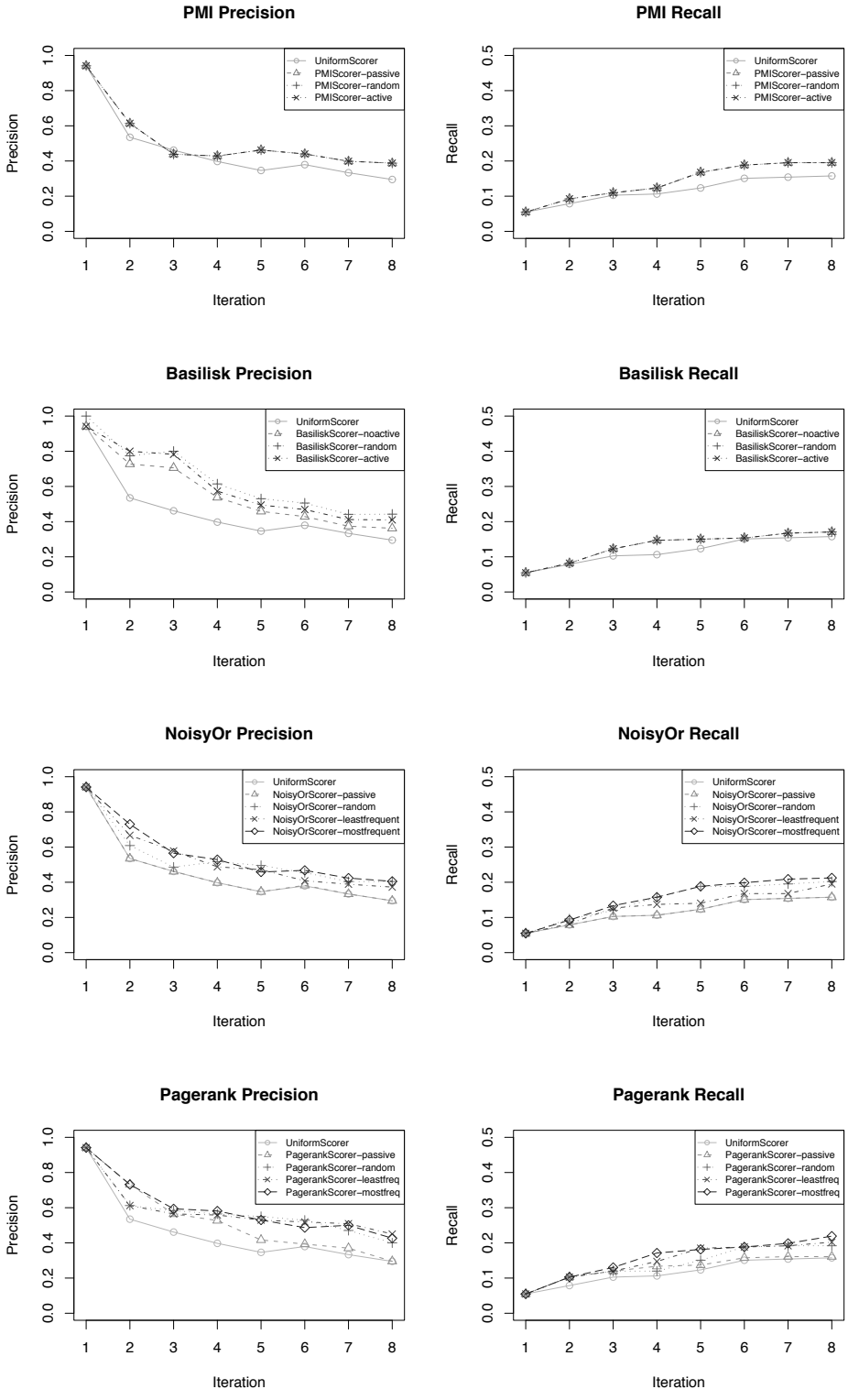

Figure 4: Comparison of the precision and recall of each of the scoring methods as the number of iterations increases.

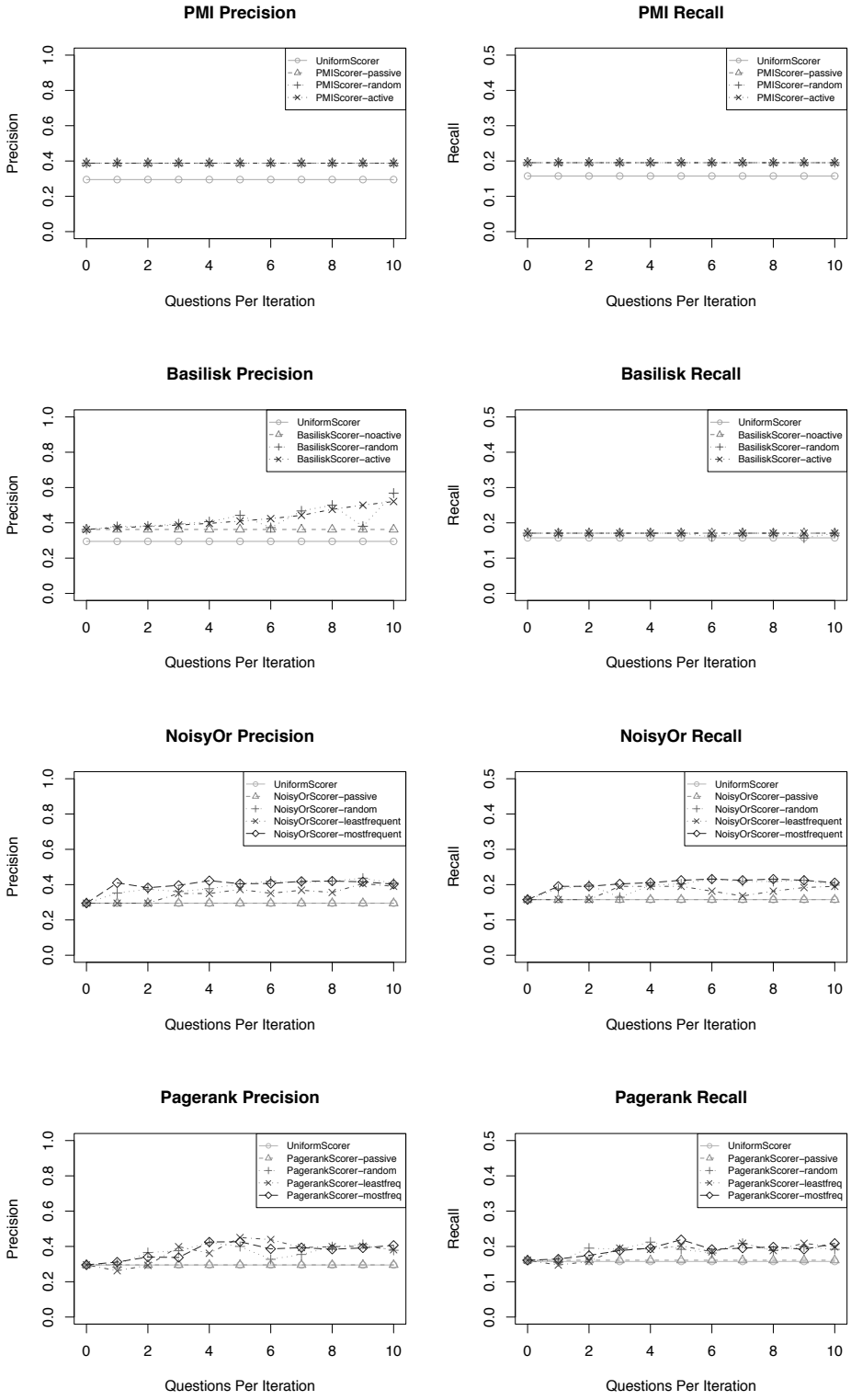

Figure 5: Comparison of the final precision and recall of each of the scoring methods as a function of the number of questions asked in each iteration.

Pointwise mutual information. The PMI scoring method is barely affected by active learning. The top rows in Figures 4 and 5 show that the PMI scoring methods dominate the Uniform Scorer in terms of both precision and recall, but they also show that the three PMI methods are nearly indistinguishable. While user labeling does change the score of an entity, it does not change it much relative to the other scores. Positively labeled data was already assigned a high PMI score and negatively labeled data a low PMI score prior to user labeling. This suggests that user labeling provided little information not already available to PMI. The results in Figure 5 support offer further support for this theory, since the number of questions seems to have little effect on the precision and recall of PMI scorers. One implication is that PMI might be used as a "poor man's substitute" for a real teacher for other learning methods.

Basilisk. Basilisk scoring schemes with Score-based selection and random selection outperform the passive approach. Basilisk scorer benefited the most from active learning in our experiments. Figure 5 shows the increase in precision for Basilisk algorithm using Score-based selection, which reaches near 60%, higher than any other combination of scoring and active learning methods we investigated in this work. Note here that precision does not show much change given different numbers of questions asked. This suggests that the Basilisk algorithm extracted the same number of positive examples in all runs (varying the number of questions asked each iteration) but extracted fewer negative examples when it was allowed more labels per iteration. Although the random selection curve seems to achieve a higher precision, its value shows a bigger variance during the bootstrapping iterations. The score-based selection strategy shows a more robust performance.

Noisy-Or. Among the Noisy-Or variants, the Most Frequent strategy, the algorithm that selects the most frequently extracted entities for labeling performs the best, with slightly higher recall than the Random strategy. The Least Frequent strategy does better than the Passive and Uniform Scorer strategies, but not as well as the Most Frequent and Random strategies. We suspect that asking about the Least Frequent entities does not provide as much information as the other two active methods. Noisy-Or algorithm estimates confidence scores of rules based on the how many of entities the rule is associated with are positive. We believe the Most Frequent strategy gives us the most positive information about rules since examples chosen by this strategy are extracted by many rules and/or many times. Also, because they were extracted many times or by many rules, we believe most of them would be positive examples. The Random strategy performs well because it asks about negative examples as well as positive examples, even though the examples may not be related to many rules and thus do not reveal the goodness level of many rules. However, the Least Frequent strategy will likely choose negative examples that reflect the badness of only few rules. The Passive strategy struggles due to the bootstrapper having only one seed rule, because non-seed, extracted rules are assumed to all have 50% precision. The number of questions asked each round does not affect performance much for any Noisy-Or algorithm. The largest difference appears to come from asking 1 question versus none, which indicates that the presence of active learning is more important than the degree of intervention.

Pagerank. For the Pagerank scorer, active strategies, especially Density Selection methods (both Most Frequent and Least Frequent) perform considerably better than the passive version, which initially outperforms but eventually converges to the performance of Uniform Scorer for both precision and recall, as suggested by Figure 4. Typically, the Random strategy performs similarly to either Density Selection strategies or the Passive strategy, but does not deviate much from them. Overall, strategies utilizing active learning perform much better than the passive scorer and Uniform Scorer for both precision and recall. The same observation holds when examining the performances with respect to the number of questions asked. Figure 5 shows that while obtaining labels for more examples is beneficial for small numbers of questions, after a certain point (5 in this case) the performance of active strategies plateau with respect to the number of questions asked. Hence some level of active learning improves bootstrap precision and recall, but there seems to be a limit to its usefulness. While this may suggest that we may not achieve higher precision and recall even by obtaining many labels, if proven generally it may show that whatever the achievable level of performance is can be obtained using just a small number of labeled examples.

General trends. Figures 6 and 7 compare the top performing variants of each scoring method. Figure 6 shows that after eight iterations, the top variants of each scoring method have similar precision. Note that

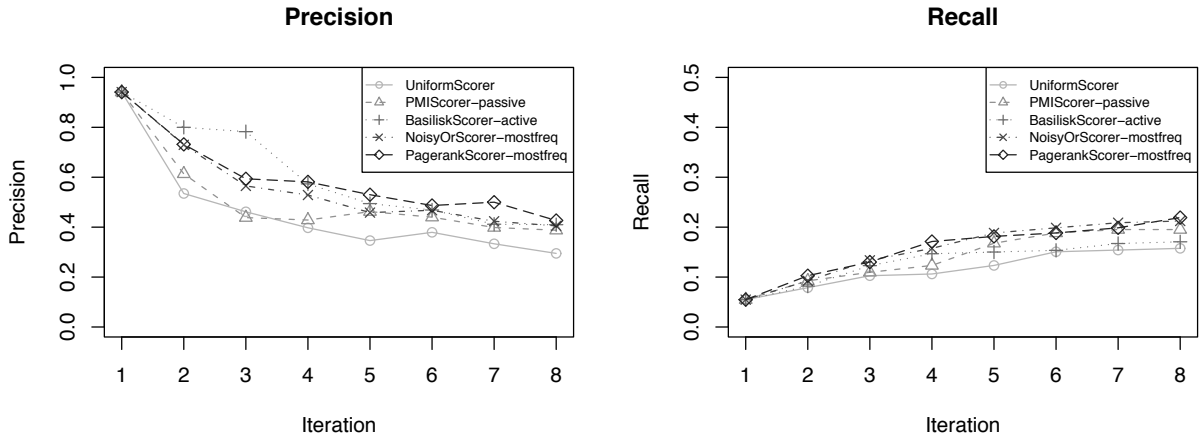

Figure 6: Comparison of the precision and recall of the best variant of each of the scoring methods as the number of iterations increases.

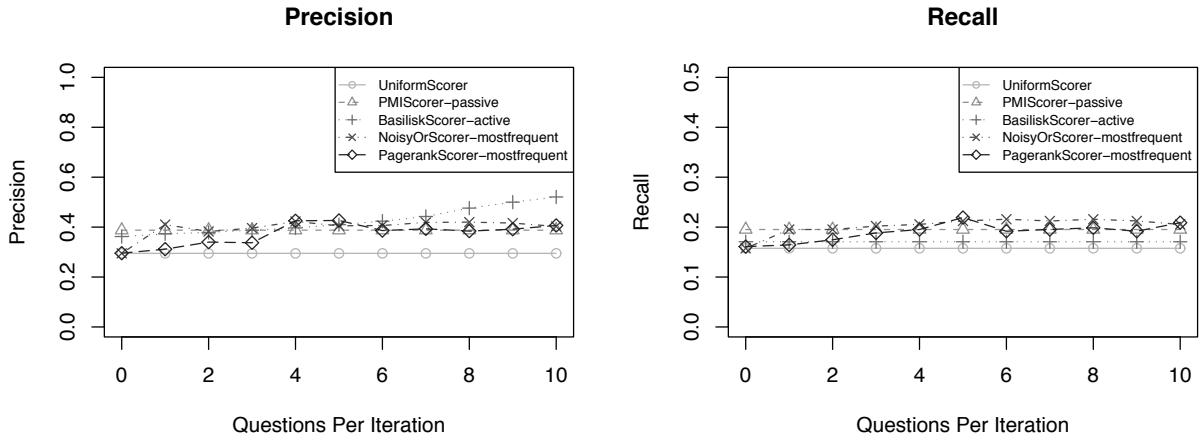

Figure 7: Comparison of the final precision and recall of the best variant of each of the scoring methods as a function of the number of questions asked in each iteration.

some methods maintain high precision early in the bootstrapping process (e.g., Basilisk) while others degrade quickly (e.g., PMI). Pagerank and NoisyOr reach the highest recall.

Figure 7 shows that as the number of questions increases, Basilisk sees a steady gain in precision. The behavior of Pagerank, PMI, and NoisyOr is quite similar, with PMI and NoisyOr faring better than Pagerank when only a few questions are asked per iteration. In general, the improvement due to active learning is seen most dramatically in the first four or five questions.

5 Conclusions

Our first goal was to assess the effect of active learning on various scoring methods used by bootstrap learners. All scoring methods demonstrated improved precision and recall over the baseline. All active learning variants except that of PMI showed improved performance over their passive counterparts.

Our second goal was to ascertain the sensitivity of active learning strategies to details such as the number of questions asked in each iteration. Again, except for PMI, the performance of all active learning strategies tends to improve as the number of questions increases. However, five or six questions are often sufficient to see those improvements. Interestingly, without any labeled data, PMI's performance (in terms of precision and recall) was competitive with respect to other scoring methods using labeled data.

These results suggest that unsupervised bootstrap learners can be greatly enhanced by a small amount of user input. While favorable, the results are preliminary. For instance, only a single bootstrapping task (e.g., the *is-a-nation* predicate) was considered in the evaluation. Such followup work would be needed to ensure that scoring methods and active learning can consistently fortify bootstrap learners to maintain a high precision and prevent divergence from the target concept.

References

[1] Eugene Agichtein and Luis Gravano. Snowball: Extracting relations from large plain-text collections. In *Proceedings of the Fifth ACM International Conference on Digital Libraries*, 2000.

[2] Avrim Blum and Tom Mitchell. Combining labeled and unlabeled data with co-training. In *COLT: Proceedings of the Workshop on Computational Learning Theory, Morgan Kaufmann Publishers*, 1998.

[3] Mary Elaine Califf. Relational learning techniques for natural language extraction. Technical Report AI98-276, 1, 1998.

[4] William W. Cohen. Learning trees and rules with set-valued features. In *AAAI/IAAI, Vol. 1*, pages 709–716, 1996.

[5] M. Collins and Y. Singer. Unsupervised models for named entity classification. 1999.

[6] Doug Downey, Oren Etzioni, and Stephen Soderland. A probabilistic model of redundancy in information extraction.

[7] Oren Etzioni, Michael Cafarella, Doug Downey, Ana-Maria Popescu, Tal Shaked, Stephen Soderland, Daniel S. Weld, and Alexander Yates. Unsupervised named-entity extraction from the web: an experimental study. *Artif. Intell.*, 165(1):91–134, 2005.

[8] Dayne Freitag. Multistrategy learning for information extraction. In *Proc. 15th International Conf. on Machine Learning*, pages 161–169. Morgan Kaufmann, San Francisco, CA, 1998.

[9] Rayid Ghani, Rosie Jones, Tom Mitchell, and Ellen Riloff. Active learning for information extraction with multiple view feature sets. In *20th International Conference on Machine Learning*, pages 21–24. Washington, DC, 2003.

[10] Thorsten Joachims. Text categorization with support vector machines: learning with many relevant features. In Claire Nédellec and Céline Rouveirol, editors, *Proceedings of ECML-98, 10th European*

Conference on Machine Learning, number 1398, pages 137–142, Chemnitz, DE, 1998. Springer Verlag, Heidelberg, DE.

[11] Rosie Jones. *Learning to Extract Entities from Labeled and Unlabeled Text*. PhD thesis, Carnegie Mellon University, 2005.

[12] Rosie Jones, Andrew McCallum, Kamal Nigam, and Ellen Riloff. Bootstrapping for text learning tasks. In *IJCAI-99 Workshop on Text Mining: Foundations, Techniques and Applications*, 1999.

[13] David D. Lewis and Marc Ringuette. A comparison of two learning algorithms for text categorization. In *Proceedings of SDAIR-94, 3rd Annual Symposium on Document Analysis and Information Retrieval*, pages 81–93, Las Vegas, US, 1994.

[14] Kamal Nigam, Andrew K. McCallum, Sebastian Thrun, and Tom M. Mitchell. Learning to classify text from labeled and unlabeled documents. In *Proceedings of AAAI-98, 15th Conference of the American Association for Artificial Intelligence*, pages 792–799, Madison, US, 1998. AAAI Press, Menlo Park, US.

[15] Ellen Riloff. Automatically constructing a dictionary for information extraction tasks. In *National Conference on Artificial Intelligence*, pages 811–816, 1993.

[16] Ellen Riloff. Automatically generating extraction patterns from untagged text. In *AAAI/IAAI, Vol. 2*, pages 1044–1049, 1996.

[17] Ellen Riloff and Rosie Jones. Learning dictionaries for information extraction by multi-level bootstrapping. In *AAAI/IAAI*, pages 474–479, 1999.

[18] Robert E. Schapire and Yoram Singer. BoosTexter: A boosting-based system for text categorization. *Machine Learning*, 39(2/3):135–168, 2000.

[19] Stephen Soderland. Learning information extraction rules for semi-structured and free text. *Machine Learning*, 34(1-3):233–272, 1999.

[20] M. Thelen and E. Riloff. A bootstrapping method for learning semantic lexicons using extraction pattern contexts, 2002.

[21] Yiming Yang. An evaluation of statistical approaches to text categorization. *Information Retrieval*, 1(1/2):69–90, 1999.