RC22967 (W0311-030) November 5, 2003
Computer Science

IBM Research Report

Towards Populating Knowledge Bases Using Text Analysis with Bootstrapping: Extended Abstract

Roy Byrd
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598

Towards Populating Knowledge Bases using Text Analysis with Bootstrapping: Extended Abstract

Roy J. Byrd
IBM T. J. Watson Research Center
Yorktown Heights, New York

Abstract: A challenge for building the Semantic Web is to capture the knowledge contained in large document collections without having to perform massive manual knowledge engineering. This paper presents Breeder, a system designed to semi-automatically generate lists of "similar" items based on their contexts in a large corpus. To breed a single item list, the system uses a small set of user-provided seed items as the starting point in a bootstrapping operation which can either proceed automatically until a stopping condition is met or be guided by interactive user feedback as the bootstrapping proceeds. The bootstrapping relies on the "duality" of two orthogonal views of the items and their features, as obtained by text analysis. We also present MultiClassBreeder, which further constrains breeding of the classes as they compete for members with one another.

We have experimented using Breeder to build lexical classes of nominal expressions, a result which is useful for populating a knowledge base with instances of ontological classes. We will describe Breeder's implementation, including the underlying text analysis and measures taken to make real-time, interactive use feasible. We will discuss our preliminary evaluation of Breeder. We also describe our plans to use Breeder to generate pairs of nominal expressions, referring to class instances, which can be used to populate knowledge bases with the extensions of ontological relations.

Introduction

In order to realize the promise of the Semantic Web, large portions of the knowledge that is currently contained in text documents on the World-Wide Web must be represented and stored as the contents of knowledge bases that are consistent with underlying ontologies. Once that is done, computer processing involving various forms of reasoning over that knowledge can be designed to automate tasks – such as document search and retrieval, document categorization and routing, and transaction processing – which now require mainly human effort. Since we also don't want to build these knowledge bases manually, a prerequisite task for obtaining them is to design, build, and execute other automated processes for extracting the required knowledge from the text in documents. This paper presents a text analysis approach to accomplishing parts of the knowledge base building task.

Specifically, the paper discusses methods for populating classes of an ontology by expanding sets of "seed" items for the class into much larger collections of items which are semantically similar to the seeds and to one another. The full paper will also describe how the same methods can be used to find instances of ontological relations, based on sample relation instances. These methods are based on the intuition, common to much information extraction work, that items whose textual mentions are in similar lexical, syntactic, and semantic contexts must themselves be semantically similar. We take this semantic similarity to be prerequisite for sets of items or relations to be assigned to the same ontological types.

Our approach to using these methods acknowledges that they are heuristic and that their results are not guaranteed to be perfect or to meet the particular requirements of any knowledge based application. Thus, they are embedded within an interactive system framework which gives human knowledge engineers flexible control over their operation and over the results they produce. An important difference from other types of knowledge engineering tools, however, is that these methods allow the human user to guide the system as it populates the knowledge base rather than force him to generate the knowledge base contents himself. The processing uses iterative bootstrapping – during each iteration, the system attempts to generate new items that are similar to the ones that are already in the set. Our approach is thus called "Breeder", a name intended to suggest that the knowledge engineer behaves like a horticulturist, designing the final outcome he wants to achieve by selecting natural specimens (i.e., "seeds") having the characteristics he wants and then monitoring the growth of the desired result by inspecting and pruning intermediate generations (i.e., the results of multiple iterations).

The plan of the paper is as follows. The "Breeder Methods" section gives a detailed explanation of the Breeder system and its component technologies. In this section, we describe the component configuration used in the experiments performed to date. In the "Evaluation" section, we briefly discuss issues related to the evaluation of Breeder-like systems and we present mechanisms built into Breeder for evaluating its performance. A full evaluation has not yet been performed, but preliminary results will be given. The next section, "Related Work," describes other information extraction research aimed at extracting lexical classes and relations from text. Finally, we conclude with a description of work remaining in the Breeder project.

Breeder Methods

Overview: Breeder's general architecture is shown in Figure 1. The overall goal is to create an environment, based on the information found in a large document collection, in which small sets of seed examples for target lexical class can be augmented, through bootstrapping, into large sets of class members. In the full paper, we will expand this description to show how Breeder can also be applied to relations.

Processing begins by applying text analysis to a large document collection representing the domain of current interest. The goal of text analysis is to derive the full set of lexical items existing in the document collection and to associate them with features that characterize the document contexts in which the items occur. This association is recorded into one or more "pairs files" which are then loaded into a feature-by-item matrix. The matrix is then used by the main Breeder bootstrapping process to generate new items that are similar to items in the seed set. After each iteration, a knowledge engineer can be given the opportunity to inspect and filter the list of new items. When iteration stops, the current set of class members is emitted as the final result.

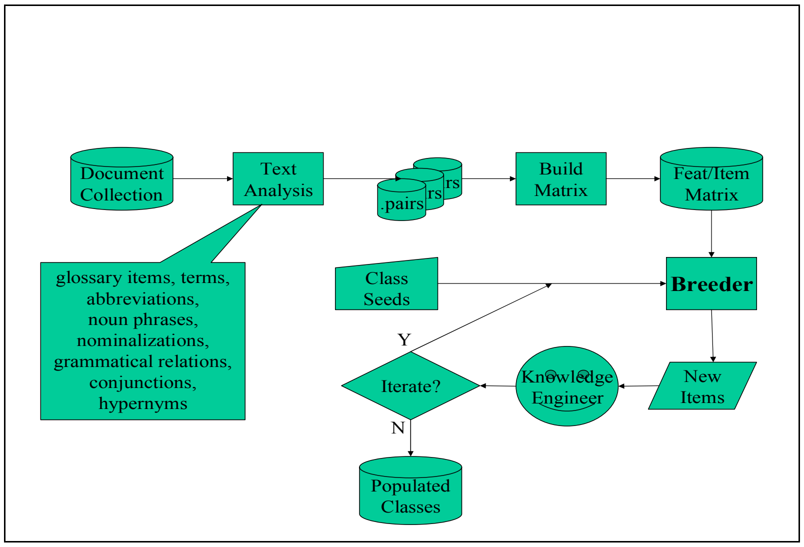


Figure 1. Breeder System Architecture

We now describe this processing in more detail.

Text Analysis. We use a number of the annotators in the Talent environment [Neff, et al. (2003)] to extract the lexical items and contextual features for use in the pairs file. For the lexical items, all of which are nominal expressions in this work, we first use Terminator and Abbreviator to extract technical vocabulary items. We could also use Nominator, a proper name extractor, to identify vocabulary items. However, the presence of long lists of non-textual proper names in our experimental document collection produced many spurious coordination features (see below) which degraded our results; hence, we didn't use Nominator for our experiments. Second, we use our glossary extraction tool, GlossEx [Park, et al. (2002)], to identify additional lexical items. In an off-line run of GlossEx against the experimental document collection, we produce a list of glossary items. Text analysis for Breeder uses that list to annotate lexical items in the texts. A benefit we derive from our vocabulary and glossary processing is that aggregation allows us to normalize variants of lexical items that occur in the text to their canonical form. See Neff, et al. (2003) for details. This enriches the set of contexts that Breeder considers for any lexical item having multiple variants. For our third source of lexical items, we use noun phrases produced by our shallow parser [Boguraev (2000)]. Specifically, we use the heads plus selected prenominal adjectives. These are similar to the items recognized by Terminator and GlossEx, but allow us to identify items which don't satisfy their criteria and yet are still useful for Breeder.

Once lexical items have been identified, they are associated with features that occur in their contexts. We capture a wide variety of information about the textual contexts for use as features. We collect about ten feature types, with the specific features containing the specific word or phrase (often another lexical item) that fills the argument slot of the chosen feature type. The feature types we collect are:

- AdjMod – an adjective that modifies the lexical item
- NounMod – a noun (or nominal expression) that modifies the lexical item
- QuantMod – a quantifier that modifies the lexical item
- VerbMod – a (typically participial) verb that modifies the lexical item
- ObjOfPrep – a noun which takes this lexical as the object of a prepositional argument
- PrepArg – a preposition that this lexical item uses to introduce its own argument
- PrepArgNP – a nominal expression that this lexical item takes as a prepositional argument
- ObjOfVerb – a verb, including nominalization bases, that takes this lexical item as a direct object
- SubjOfVerb – a verb that takes this lexical item as a subject
- Hypernym – another lexical item of which this lexical item is a hyponym
- Conjunct – another lexical item with which this lexical item is conjoined

This list of feature types can easily be expanded to include others, including simpler ones such as words to the immediate left and right, part-of-speech labels from a tagger, upper/lower case patterns, etc. We did not use such features in our experiments.

From their descriptions, above, it is clear that our features all exploit the syntactic structures that the Talent shallow parser derives for the document sentences. Two feature types deserve special comment. The Hypernym feature results from an elaboration of the hypernym/hyponym extraction heuristics proposed in Hearst (1992). In our version, the extraction patterns include constraints on the syntactic structure in which the potentially multi-word lexical items occur, rather than Hearst's linear patterns over single-word concepts. Similarly, the Conjunct feature exploits the conjunction subgrammar in the shallow parser to identify cases where lexical items appear in coordinate conjunctions and extracts the conjuncts as one another's features. The intuition that supports the use of this feature type for lexical class population is that conjoined nominal expressions often belong to the same semantic class [cf. Roark and Charniak (1998)]. Figure 2 shows an example of four feature-item pairs extracted from the sentence fragment "… for life-threatening diseases like cancer and aids, …".

```
(Hypernym:life-threatening disease; cancer)
(Hypernym:life-threatening disease; AIDS)
(Conjunct:AIDS; cancer)
(Conjunct:cancer; AIDS)
```

Figure 2. Pairs Extracted from the Text "... for life-threatening diseases like cancer and aids, ..."

Feature-Item Matrix: All of the feature item pairs extracted by text analysis are stored in flat “pairs files,” using the format shown in Figure 2. As can be seen, the inventories of features and items are open-ended. In fact, any other dataset that can be represented as a pairs file can also be used for bootstrapping with Breeder. The pairs files are then loaded into a feature-item matrix. The rows of the matrix represent all extracted features; the columns represent all extracted lexical items; and each cell contains the frequency with which its item occurred with its feature.

The feature-item matrix supports two views of the frequency information extracted by text analysis: the FeatureView and the ItemView. From each view, it is possible to gain a perspective on elements of the other view. For example, given a handful of items in the ItemView, we can extract and rank the set of features that co-occur with any of those items. Using the matrix, we identify the features to extract by looking for non-zero frequencies in the columns representing the starting items. Ranking is done in various ways, to be discussed under “Bootstrapping”, below. Similarly, given any set of features, we can use the matrix rows to identify and rank items that occur with those features.

Using the terminology introduced by Brin (1998), we say that the feature and item views are in “duality” with one another. We exploit this duality to breed classes of lexical items, using a bootstrapping algorithm.

Bootstrapping: Single-class bootstrapping uses the feature-item matrix in an iterative algorithm. The operation of a single iteration is illustrated in Figure 3. For the first iteration, the ItemView’s repository is initialized with a set of seed items supplied by the user. Using that set of items, the ItemView extracts, from the matrix, the list of features that occur with any of the items currently in its repository, yielding a set of candidate features for the FeatureView. Those candidates are processed by one of several FeatureView “evaluators”, which (1) assigns a score to each feature, usually based on the features’ effectiveness in accounting for the current set of items in the ItemView repository, (2) ranks the filters by their scores, and (3) filters the ranked list according to user-supplied parameters, and (4) adds (or replaces) the filtered features to the FeatureView repository. Next, the new set of features in the repository are used to extract a set of candidate items. Then an ItemView evaluator is used to score, rank, and filter those candidates, yielding a new set of items to add to the repository.

Subsequent iterations are like the first one, except that the ItemView repository retains its current contents from the preceding iteration. When the iterations are stopped, the final contents of the ItemView repository are returned as the desired lexical class.

Breeder offers choices of evaluators for the two views. For ItemViews, the currently defined set of evaluators is:

- TupleProbability – used in Agichtein and Gravano (2000) and Yangarber, et al. (2002)
- AvgLog – used in Thelen and Riloff (2002)
- MetaBootstrapping – used in Riloff and Jones (1999)
- DecisionList – used in Yarowsky (1995)
- CentroidCosine – adapted from Schütze (1998)

For FeatureViews, the choices are:

- RLogF – used in Thelen and Riloff (2002)
- LogLikelihood – based on Dunning (1993)
- LogProbRatio – used in Yarowsky (1995)

As noted, these evaluators reflect a range of scoring strategies from the current literature on bootstrapping and word-sense disambiguation. By making these choices flexibly available, Breeder offers a laboratory for comparing their performance on various tasks and datasets.

When using Breeder, the user must specify a number of operational parameters. Among them are controls for the evaluators and the conditions for stopping the iteration. The major evaluator controls are the number of features or items to let through the filter on each iteration and an update policy, stating whether the filtered items are added to the current repository contents or replaces them. Stopping conditions offer a choice of stopping either after a certain number of iterations, or when no new items are added to the item repository, or when the interactive user is satisfied with the current class members. Again, some of these choices reflect various practices in the literature and support experimentation with Breeder.

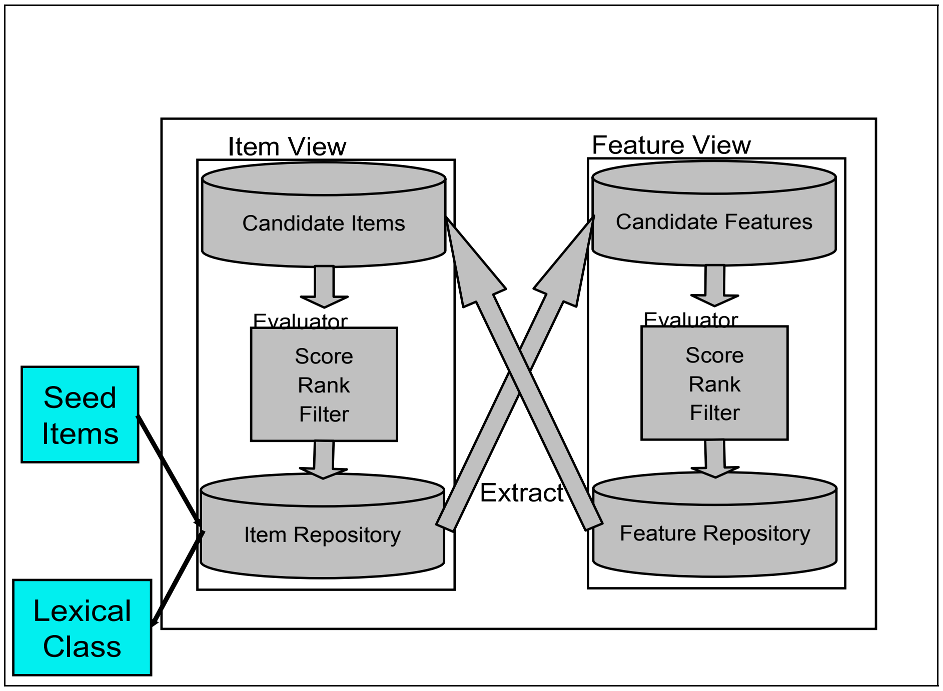
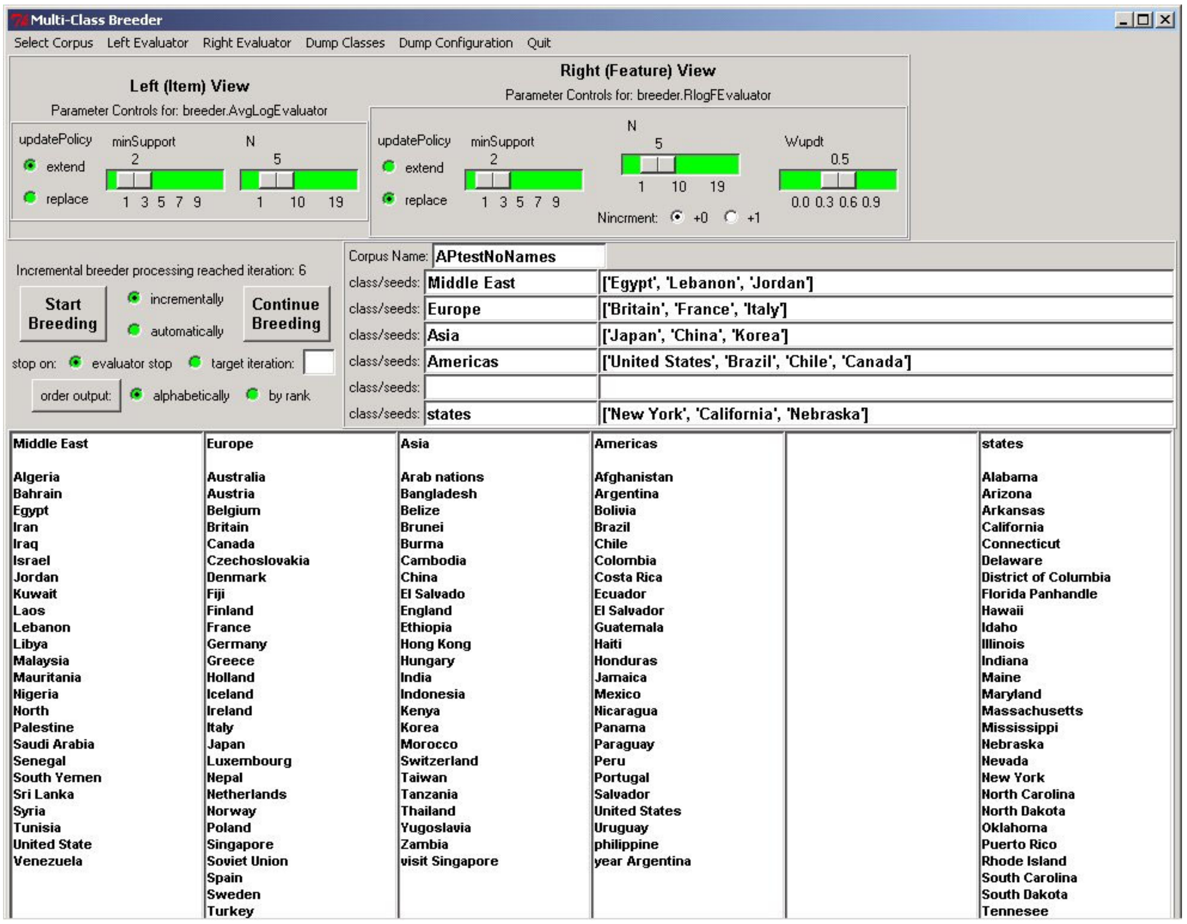


Figure 3. Bootstrapping Architecture for a Single-Class Breeder

MultiClassBreeder: We expect that in knowledge base building applications, users will need to populate multiple classes. In such applications, we can gain an advantage by breeding the multiple classes simultaneously, using MultiClassBreeder. Figure 4 shows an example in which the user wants to differentiate countries from different continents and from U.S. states. Among the advantages of multi-class breeding is the fact that, under control of an ambiguity parameter (not shown in Figure 4), as each class absorbs items that are closest to it, those items become unavailable to the other classes. In fact, by defining a catch-all “other” class, users can help ensure the purity of the classes they are really interested in. A further advantage, exploited by some of the evaluators listed above, is that knowledge of lexical items which are definitely not in a lexical class – because they are known to belong to other classes – can be used to improve the scoring of feature.

Inspection of Figure 4 reveals that some of the item assignments to classes seem like errors. Indeed, putting Venezuela into the Middle East class is patently wrong. However, we are less certain about assigning Mauritania (an African country) to the Middle East. In fact, it is not clear from the seed list whether the user wants "Middle East" countries classed by geography, culture, religion, language, or anything else. MultiClassBreeder allows us to deal with such uncertainty by offering an interactive mode in which the user can monitor the items added to the ItemView repository at the end of each iteration. In this mode, the user may delete an item from any class, move it to another class, or add an entirely new seed to one of the classes. Using this capability, which admittedly requires more work than fully automatic breeding, the user can obtain results that are arbitrarily close to his intentions.



Left (Item) View
Parameter Controls for: breeder.AvgLogEvaluator
updatePolicy: [✓ extend] replace
minSupport: 2
N: 5

Right (Feature) View
Parameter Controls for: breeder.RlogFEvaluator
updatePolicy: extend [✓ replace]
minSupport: 2
N: 5
Wupdt: 0.5
Nincrment: [✓ +0] +1

Incremental breeder processing reached iteration: 6
[✓ incrementally] automatically
stop on: [✓ evaluator stop] target iteration:
order output: [✓ alphabetically] by rank

Corpus Name: APtestNoNames

class/seeds:	
Middle East	['Egypt', 'Lebanon', 'Jordan']
Europe	['Britain', 'France', 'Italy']
Asia	['Japan', 'China', 'Korea']
Americas	['United States', 'Brazil', 'Chile', 'Canada']
states	['New York', 'California', 'Nebraska']

Middle East	Europe	Asia	Americas		states
Algeria	Australia	Arab nations	Afghanistan		Alabama
Bahrain	Austria	Bangladesh	Argentina		Arizona
Egypt	Belgium	Belize	Bolivia		Arkansas
Iran	Britain	Brunei	Brazil		California
Iraq	Canada	Burma	Chile		Connecticut
Israel	Czechoslovakia	Cambodia	Colombia		Delaware
Jordan	Denmark	China	Costa Rica		District of Columbia
Kuwait	Fiji	El Salvado	Ecuador		Florida Panhandle
Laos	Finland	England	El Salvador		Hawaii
Lebanon	France	Ethiopia	Guatemala		Idaho
Libya	Germany	Hong Kong	Haiti		Illinois
Malaysia	Greece	Hungary	Honduras		Indiana
Mauritania	Holland	India	Jamaica		Maine
Nigeria	Iceland	Indonesia	Mexico		Maryland
North	Ireland	Kenya	Nicaragua		Massachusetts
Palestine	Italy	Korea	Panama		Mississippi
Saudi Arabia	Japan	Morocco	Paraguay		Nebraska
Senegal	Luxembourg	Switzerland	Peru		Nevada
South Yemen	Nepal	Taiwan	Portugal		New York
Sri Lanka	Netherlands	Tanzania	Salvador		North Carolina
Syria	Norway	Thailand	United States		North Dakota
Tunisia	Poland	Yugoslavia	Uruguay		Oklahoma
United State	Singapore	Zambia	philippine		Puerto Rico
Venezuela	Soviet Union	visit Singapore	year Argentina		Rhode Island
	Spain				South Carolina
	Sweden				South Dakota
	Turkey				Tennesee

Figure 4. MultiClassBreeder used to Breed Classes of Countries and States.

Evaluation

Even though it is difficult to evaluate a bootstrapping solution for lexical class population, we have several reasons for wanting to do so. First, we need to be able to compare our results to those reported in the literature. Second, given the number of user-specified parameters (evaluators, repository increment sizes, etc.), we need a way to assess the effectiveness of various settings. The third reason has to do with the need to operate in different application environments, with different corpora of different sizes, and with different user requirements. For example, breeding broad classes, like "Person", will be very different from breeding fine-grained classes, such as "weapons carried on ships". For this reason, as we develop Breeder, we need to understand how to choose suitable parameter settings appropriately for the different operational environments.

It is easier to evaluate precision for a tool like Breeder than it is to evaluate recall. We can do so by either (A) manually inspecting Breeder's output for the generated classes, or by (B) having an exhaustive list of desired class members, for example from a gazetteer. Of course, depending on the class, obtaining an exhaustive list may itself be difficult. Even if we possess exhaustive lists, however, evaluating recall remains hard. This is so because we can't just measure the proportion of a list that Breeder's output covers; in addition, we must know what portion of that list exists in the corpus in extractable form. Knowing that requires annotation of the entire corpus for the classes we want to evaluate, which is expensive.

For these reasons, in the preliminary evaluations we have done so far, we have usually used method (A) to generate lists of class members from the output of multiple runs of Breeder with multiple parameter settings. The resulting lists (as well as lists obtained by method (B)) can be used with a built-in assessment tool in Breeder to produce graphs of output performance. While we have not done extensive evaluations to date, Figure 5 shows an example of the kinds of assessment that may be done. The graph shows the precision obtained after six iterations over the class specifications shown in Figure 4. In this case, using method (A) to check the correctness of the output was fairly easy, modulo the issue such as whether Mauritania is a [Arabic/Islamic/Middle-eastern] country.

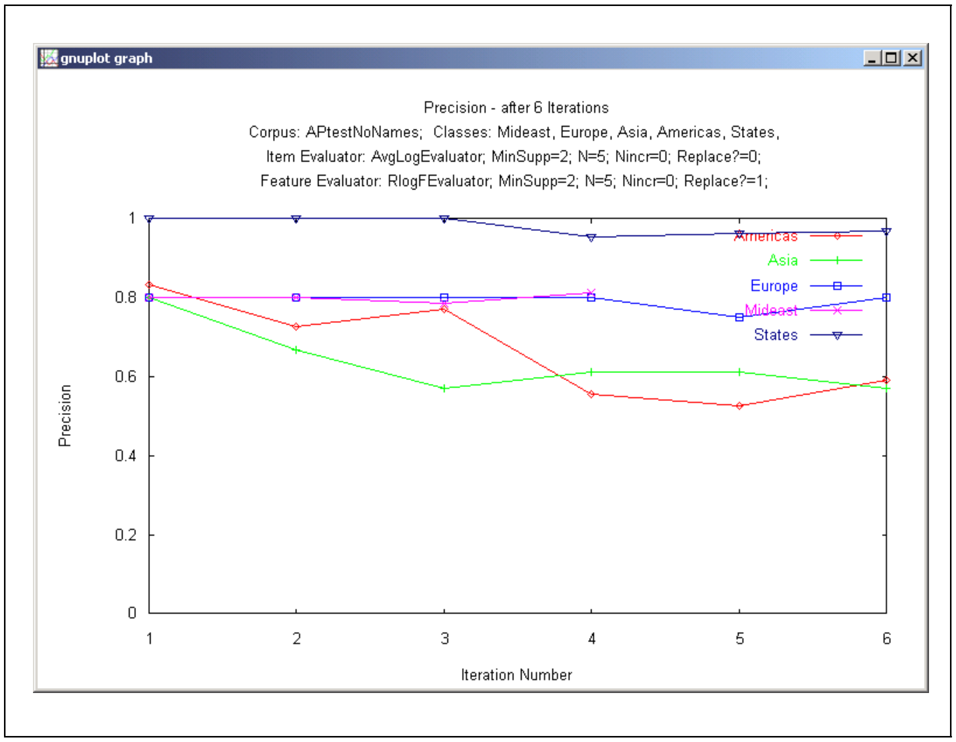

Precision - after 6 Iterations
Corpus: APtestNoNames; Classes: Mideast, Europe, Asia, Americas, States,
Item Evaluator: AvgLogEvaluator; MinSupp=2; N=5; Nincr=0; Replace?=0;
Feature Evaluator: RlogFEvaluator; MinSupp=2; N=5; Nincr=0; Replace?=1;

Figure 5. Plot of Precision against Iteration Number for Countries and States

When we consider interactive use of Breeder, there are further concerns about evaluation. Using recall and precision to evaluate the output of an interactive user is probably not the best thing to do. Rather, task-based evaluations seem more to the point. The question is whether the knowledge engineering user can be more productive with Breeder than without it; and, if so, by how much. Designing and carrying out such task-based evaluations remains for future work.

Related Work

In the final paper, we will discuss the relation of Breeder to other work on bootstrapping and word-sense disambiguation. Some of that discussion has already begun, in the section above about view evaluators.

Conclusions

In the final paper, we will detail our future work, including the use of Breeder with pairs of nominal expressions (as items) and contextual patterns (as features), in order to breed relations.

References

Agichtein, Eugene and Luis Gravano. Snowball: Extracting relations from large plain-text collections. In *Proceedings of the 5th ACM International Conference on Digital Libraries*. 2000.

Boguraev, Branimir K. Towards finite-state analysis of lexical cohesion", In *Proceedings of the 3rd International Conference on Finite-State Methods for NL*P, INTEX-3, Liege, Belgium. 2000.

Brin, Sergey. Extracting patterns and relations from the World-Wide Web. In *Proceedings of the 1998 International Workshop on the Web and Databases (WebDB '98)*. 1998.

Dunning, Ted. Accurate methods for the statistics of surprise and coincidence. In *Computational Linguistics*. 1993.

Hearst, Marti. Automatic acquisition of hyponyms from large text corpora. *In Proceedings of the 14th International Converence on Computational Linguistics*. 1992.

Neff, Mary, Roy Byrd, and Branimir Boguraev. The Talent System: TEXTRACT Architecture and Data Model. In *Proceedings of the Software Engineering and Architecture of Language Technology Systems (SEALTS) Workshop at NAACL 2003*. 2003

Park, Youngja, Roy Byrd, and Branimir Boguraev. Automatic Glossary Extraction: Beyond Terminology Identification. In *Proceedings of COLING 2002*. 2002

Riloff, Ellen and Rosie Jones. Learning Dictionaries for information extraction by multi-level bootstrapping. *In Proceedings of the 16th National Conference on Artificial Intelligence (AAAI-99)*. 1999.

Roark, Brian and Eugene Charniak. Noun-phrase co-occurrence statistics for semi-automatic semantic lexicon construction. In *Proceedings of COLING-ACL '98*. 1998

Schütze, Hinrich. Automatic word sense discrimination. In *Computational Linguistics*. 1998.

Thelen, Michael and Ellen Riloff. A bootstrapping method for learning semantic lexicons using extraction pattern contexts. In *Proceedings of the 2002 Conference on Empirical Methods in Natural Language Processing (EMNLP 2002)*. 2002.

Yangarber, Roman, Winston Lin, and Ralph Grishman. Unsupervised learning of generalized names. In *Proceedings of COLING*. 2002.

Yarowsky, David. Unsupervised word sense disambiguation rivaling supervised methods. In *Proceedings of ACL '95*. 1995.