A Survey of Bootstrapping Techniques in Natural Language Processing

Daniel Waegel
Department of Computer Science
University of Delaware
Newark, DE 19711
danielw@udel.edu

Abstract

This paper surveys a selection of seminal bootstrapping articles and present an in-depth discussion of the algorithms and the relative strengths and weaknesses of the techniques used. We also examine the problems these authors attempt to solve and analyze when it is appropriate to utilize a bootstrapping algorithm. Finally, we discuss some of the limitations inherent in any bootstrapping approach.

1 Introduction

Statistical approaches to information extraction and natural language processing tasks require vast amounts of information in order to perform acceptably and produce reliable results. Training empirical algorithms requires huge corpora. A corpus of unannotated, unmarked natural language text is not often suitable for training purposes on most tasks; algorithms typically need their training data to be annotated in some fashion in order to be able to extract and learn the salient textual features.

Unfortunately, annotated corpora are in short supply. Manually annotating even a small corpus is incredibly time-consuming and is infeasible for larger ones. Automatically tagging a corpus with the necessary information is usually not possible, since the annotations needed for training are typically the information we are trying to find in the first place (Agichtein and Gravano, 2000).

For example, an empirical algorithm that attempts to identify whether a given article is written in a subjective or objective fashion would typically train on a corpus that is marked up with annotations indicating subjectivity and objectivity in some fashion. The algorithm would then learn from this annotated training corpus and then (hopefully!) be able to effectively perform this task on unannotated natural language text.

Bootstrapping provides an alternative to painstaking manual annotation. The techniques reviewed herein are all trained on unannotated corpora. The implementers provide their algorithm with a small number of carefully chosen **seeds**. These seeds are then used as a starting point to gather other, similar terms from an unannotated training corpus, which are in turn then used to gather even more terms, and so on. In essence, the algorithm "pulls itself up from its bootstraps", hence the name.

In the following section, we discuss in detail a selection of widely cited bootstrapping methods and the problems they are attempting to solve. First, we discuss the *Snowball* algorithm used by Agichtein and Gravano (2000) to extract relations from unannotated text. Next, we investigate the *Basilisk* algorithm (Thelen and Riloff, 2002) used for the task of semantic categorization. We also examine the *NOMEN* bootstrapping system and its application to two tasks: Semantic categorization (Lin et al., 2003) and learning generalized names (Yangarber et al., 2002). Finally, we cover how the bootstrapping of extraction patterns can identify elusive subjective language using two different methodologies (Riloff et al., 2003; Riloff and Wiebe, 2003).

We then examine the corpora used in these papers and evaluate the results reported by the authors. We strive to identify the strengths and weaknesses of the various bootstrapping approaches and their suitability for the different applications.

In the last sections we discuss and draw conclusions about the merits of bootstrapping more generally. We demonstrate that it is a powerful and flexible technique that can extract many different types of information with minimal human interaction. We also muse upon the limitations and drawbacks of bootstrapping techniques and what can be done to mitigate such concerns.

2 Bootstrapping Algorithms

While there is great variation between implementations, bootstrapping approaches in natural language processing all follow the same general format:

1. Start with an empty *list* of *things*.[1]
2. Initialize this *list* with carefully chosen *seeds*.
3. Leverage the *things* in the *list* to find more *things* from a *training corpus*.
4. Score those newly found *things*; add the best ones to the *list*.
5. Repeat from step 3. Stop after a set number of iterations or some other stop condition. The intuition for bootstrapping stems from the observation that words from the semantic category tend to appear in similar patterns and similar contexts. For example, the words "water" and "soda" are both in the semantic BEVERAGE category, and in a large text they will likely both be found in phrases containing *drank* or *imbibed*. The core intuition is that by searching for patterns like that we can find other BEVERAGEs, and then by finding patterns that contain *those*, we can find even more.

The core feature of a bootstrapping algorithm is that each iteration is fed the same type of data as its input that it produces as its output. The output of the first iteration is used as the input of the second iteration, and so on. It should not be surprising that choosing the initial set of data (the *seeds*) from which all other data is "grown" is a critical factor (arguably the *most* critical factor) in the performance of the algorithm.

2.1 Snowball

This paper by Agichtein and Gravano (2000) outlines an attempt to identify and extract structured *relations* between named entities from unstructured text. In this context, a relation is defined as a table or listing that maps one entity onto another (as in a relational database). The example used throughout the article is mapping an organization entity (e.g., *Microsoft* or *Boeing*) onto their headquarters (a location entity such as *Redmond* for Microsoft). These are represented as a *tuple* in the form of ¡O, L¿. The goal of this research is to allow much more nuanced responses to questions such as "Where is Microsoft headquartered?".

Snowball is built on top of the DIPRE (**D**ual **I**terative **P**attern **R**elation **E**xpansion) method by Brin (1999). The central idea behind both of these exploits the idea, termed *pattern relation duality*, that a "good" extraction pattern will produce good tuples given a large enough body of text and a "good" set of tuples can have good extraction patterns deduced from it. The only user-provided input to the Snowball system is the training corpus and a small handful of manually compiled relations (the *seeds*). The initial seeds provided by Agichtein and Gravano (2000) were:

ORGANIZATION	HEADQUARTERS
MICROSOFT	REDMOND
EXXON	IRVING
IBM	ARMONK
BOEING	SEATTLE
INTEL	SANTA CLARA

Table 1: Seed tuples provided to Snowball.

The system then searches the training corpus looking for documents where both terms in a seed tuple occur in close proximity to one another. It analyzes the connecting text and surrounding context and generates an extraction pattern represented by a *Snowball pattern*, a 5-tuple of the form *<left, tag1, middle, tag2, right>*. The *left*, *middle*, and *right* terms are weighted vectors of words that represent the respective left, middle, and right contexts. The other two terms in the 5-tuple, *tag1* and *tag2*, are named-entity tags such as *ORGANIZATION* or *LOCATION*. Since the patterns can only possibly match named entities of the proper type, a preprocessing step is required to tag the corpus with a named-entity tagger. An example *Snowball pattern* generated by the system is the 5-tuple *<{<the, 0.2>}, LOCATION, {<-, 0.5>, <based, 0.5>}, ORGANIZATION, { }>* which match text in the form "the ¡LOCATION¿-based ¡ORGANIZATION¿".

In addition to the *Snowball pattern* 5-tuples, each document snippet *S* that contains two named entities with the correct named-entity tags (ORANGANIZATION and LOCATION in our examples) are also assigned a corresponding 5-tuple. Three weighted vectors l_S, r_S, and m_S are

[1] Usually words or phrases, but it can be any representation of language (such as regular expressions, tuples, etc.)

created from the text surrounding the two named entities. The size of the l_S and r_S vectors is limited by a window of size *w*. The vector weights indicate the relative importance, measured as a function of the frequency, of each term in the context. The vectors are scaled so their norm is 1, and then the vectors are multiplied by a scaling factor to indicate each vector's relative importance. [2]

Snowball generates a 5-tuple for each string that contains a co-occurring pair of named entities found in one of the seed tuple. Once all of these are gathered, they are clustered using a simple single-pass clustering algorithm. Their proximity is measured by a similarity threshold τ_{sim} for the following similarity metric, given two 5-tuples t_1 and t_2:

$$Match(t_1, t_2) = \begin{cases} l_1 \cdot l_2 + m_1 \cdot m_2 + r_1 \cdot r_2 & \text{if } the\ tags\ match \\ 0 & \text{Otherwise} \end{cases} \quad (1)$$

After clustering, *Snowball patterns* are formed by collapsing the *left*, *middle*, and *right* vectors of 5-tuples in the clusters via their centroid: The *left* vectors are represented by their centroid $\bar{l_s}$, the *middle* vectors by centroid $\bar{m_s}$, and the *right* vectors by $\bar{r_s}$. Since the similarity metric $Match(t_1, t_2)$ guarantees that only 5-tuples with the same tags will be similar, the *Snowball pattern* for the cluster is built as $< \bar{l_s}, t_1, \bar{m_s}, t_2, \bar{r_s} >$.

After the *Snowball patterns* are generated, the corpus can now be scanned to find to locate additional relations (tuples of the form <*O, L*> in our examples). The algorithm for doing so is outlined in Algorithm 1.

For every sentence S containing an organization and location tagged by the named-entity tagger, a 5-tuple $t_S =< l_s, t_1, m_s, t_2, r_s >$ is generated. If there exists a pattern P represented by 5-tuple t_P such that $Match(t_S, t_P) \geq \tau_{sim}$, the snowball system generates a candidate tuple $< O, L >$.

Thus, after the algorithm is done, each candidate tuple will have one or more patterns associated with it. Each candidate-pattern association has a similarity score associated with it.

The final step is to measure the system's confidence that the patterns and tuples produced are likely to be "good". The confidence in a pattern p is simply the ratio of its $positive$ matches (those matched tuples that agree with known data) to positive and negative matches (those matched tuples that disagree with known data). The confidence in a candidate tuple $Conf(T)$ is defined as

$$Conf(T) = 1 - \prod_{i=0}^{|P|} 1 - Conf(P_i) \cdot Match(C_i, P_i), \quad (2)$$

where $P_i \in P$ is each pattern that generated T and $Match(C_i, P_i)$ is the similarity score between the candidate T and the pattern P_i.

Any candidates with a low confidence score are discarded, and the rest are added to the pool of accepted relations. This newly enlarged pool, which originally starts with the manually chosen seeds, forms the basis for the next cycle of bootstrapping.

foreach *snippet* $\in$ *corpus* **do**
  $\{< O, L >, < l_s, t_1, m_s, t_2, r_s >\}$
  $\leftarrow CreateOccurrence(snippet)$;
  T_C = $< O, L >$;
  Sim_{Best} = 0;
  foreach *p* $\in$ *Patterns* **do**
    $sim \leftarrow Match(< l_s, t_1, m_s, t_2, r_s >, p)$;
    if $sim \geq \tau_{sim}$ **then**
      UpdatePatternSelectivity(p, T_C);
      if $sim \geq Sim_{Best}$ **then**
        $Sim_{Best} \leftarrow sim$;
        $P_{Best} \leftarrow p$;
      end
    end
  end
  if $Sim_{Best} \geq \tau_{sim}$ **then**
    $CandidateTuples[T_C].Patterns[P_{Best}] \leftarrow Sim_{Best}$;
  end
end
return CandidateTuples;

Algorithm 1: Extracting new tuples from *Snowball patterns*

2.2 Basilisk

The Basilisk algorithm (**B**ootstrapping **A**pproach to **S**emant**I**c **L**exicon **I**nduction using **S**emantic **K**nowledge), developed by Thelen and Riloff (2002), utilizes a bootstrapping approach to developing a semantic lexicon of nouns. This is a categorization task: Search for nouns in an unan-

[2] In the implementation, Agichtein and Gravano (2000) assigns the middle vector higher weights than the left or right vectors with the intuition that the middle vector is the most important.

notated corpus and assign them to one of six semantic categories.[3] These ad hoc categories were manually selected by the authors because the corpus is comprised of articles written about terrorism.[4]

Knowledge of semantic class information is integral to the success of a huge assortment of natural language processing tasks, such as question answering, information extraction, (Thelen and Riloff, 2002) and summarization (Lin et al., 2003). Large semantic dictionaries such as WordNet provide a robust and readily accessible semantic lexicon, but Thelen and Riloff (2002) argue that these are insufficient for a number of reasons: They can never be fully comprehensive (due to neologisms, idioms, esoteric verbiage, etc), and also that a general-purpose lexicon will not usually contain domain-specific jargon and vocabulary.

Basilisk, like all bootstrapping algorithms, must be seeded with carefully selected terms in order to be effective. Thelen and Riloff (2002) seeded their algorithm by sorting the words in the corpus by frequency and manually identifying the 10 most frequent words for each of the six semantic categories.

Prior to the actual bootstrapping process, the algorithm runs an *extraction pattern learner*[5] on the corpus to extract every noun phrase in one of three forms: subject ("*<subject>was arrested*"), direct object ("*murdered <direct object>*"), or prepositional phrase ("*collaborated with <pp object>*").

The bootstrapping algorithm then proceeds as per Algorithm 2.

In step 1, the extraction patterns must be evaluated and scored. Basilisk uses the ***RlogF*** metric for this task. This is defined as:

$$RlogF(pattern_i) = \frac{F_i}{N_i} \cdot log_2(F_i) \qquad (3)$$

where F_i is the number of nouns extracted by $pattern_i$ known to belong to a semantic category and N_i is the total number of nouns extracted by $pattern_i$. This essentially scores those patterns which are highly selective (have a high precision) or which are moderately selective but are very prolific (have moderate precision but a high recall).

[3]The semantic categories are: BUILDING, EVENT, HUMAN, LOCATION, TIME, and WEAPON.

[4]See section 3 for more details regarding the corpus.

[5]The learner, the Autoslog system by Riloff, is briefly mentioned for completeness. A full description is outside the scope of this survey.

Generate all extraction patterns in the corpus and record their extractions.
$lexicon \leftarrow \{\text{seed words}\}$
$i \leftarrow 0$
BOOTSTRAPPING
1. Score all extraction patterns
2. $patternpool \leftarrow$ top $20 + i$ patterns
3. $candidate\ word\ pool \leftarrow$ extractions from $pattern\ pool$
4. Score $candidate \in candidate\ word\ pool$
5. Add top 5 $candidates$ to $lexicon$
6. $i \leftarrow i + 1$
7. Go to Step 1

Algorithm 2: Pseudo-code for Basilisk algorithm

In step 2, the best-scoring patterns are placed in the $pattern\ pool$. In the first cycle, Thelen and Riloff (2002) set this parameter to the top 20 patterns. During each subsequent cycle, the pool size is expanded by one. This prevents the pool from becoming "stale"; witha constant pool size, eventually all valid candidates would be extracted by the same top-scoring 20 patterns and only invalid candidates would remain.

In step 3, every extraction produced by the patterns in the $pattern\ pool$ is added to the $candidate\ word\ pool$, unless it is already present in the $lexicon$. Next, these $candidates$ need to be scored, which is done using the scoring metric:

$$score(word_i) = \frac{\sum_{j=1}^{P_i} log_2(F_j + 1)}{P_i} \qquad (4)$$

where P_i is the number of patterns that extract $word_i$, and F_j is the number of unique, known, category members extracted by $pattern_j$. After scoring, the best candidates are added to the lexicon, the pools are flushed, and the bootstrapping cycle starts over with the enlarged lexicon.

2.3 NOMEN

Two of the articles we survey use the NOMEN algorithm, although the goals of each paper were very different. In much the same vein of research as the previous section, Lin et al. (2003) investigates the suitability of bootstrapping for unsupervised learning of semantic classification of terms. We survey this research along with the previously discussed (Thelen and Riloff, 2002) examine whether dissimilar bootstrapping approaches

to the same problem will have similar results. This gives us some insight into the robustness of the bootstrapping paradigm.[6]

The NOMAN algorithm was first formally described in the literature by Yangarber et al. (2002), where it is used to address the problem of learning generalized names in a biomedical context. Generalized names, such as *mad cow disease* or *Ebola hemorrhagic fever*, often lack the (full) capitalization cues afforded to proper names and detection methods must rely on other metrics to attain reasonable performance. The authors particularly focus on the confounding factor of semantic ambiguity on the bootstrapping process. It is common for names in the biomedical context to refer to both a disease and a symptom, resulting in ambiguity and greatly increasing the difficulty of distinguishing between the two semantic classes. *E. coli* and *encephalitis* are two examples of this provided by Yangarber et al. (2002).

Pseudo-code for NOMAN can be found in Algorithm 3. The following details are taken from Lin et al. (2003) as we feel that description was more readable. There are only two small differences between the implementations, which we note below.

The NOMEN algorithm requires its training set to undergo significant pre-processing: The training corpus is run through a zoner, a sentence splitter, a tokenizer/lemmatizer, and a part-of-speech tagger. The zoner is necessary to extract the actual content because the training corpus is a collection of mailing list correspondence with mailing headers and footers. The splitter and tokenizer break the corpus into word tokens, the lemmatizer converts words to their base form, which are finally tagged by the part-of-speech tagger.

After pre-processing, the first step is choosing the seeds. Yangarber et al. (2002) chose the 10 most common names for each semantic category from an "IE database of more than 10,000 records". This source is different from the corpora used in training and testing. Lin et al. (2003) runs multiple experiments using different seeds and often lists them without explaining their origin. We presume they are manually selected. The seeds are then placed in the pool of *accepted names*.

NOMAN BOOTSTRAPPING
1. Tag $accepted\ names$ in corpus
2. For each tag, generate a pattern
$p = [l_{-3}l_{-2}l_{-1} < t > l_{+1}l_{+2}l_{+3}]$
3. For every learner, generate $pos(p)$, $neg(p)$, and $unk(p)$
4. For every pattern, compute $acc(p)$ and $conf(p)$

$$\begin{cases} Discard(p) & \text{if } acc(p) < \theta_{prec} \\ Score(p) & \text{Otherwise} \end{cases}$$

Accept the n top scoring patterns 5. For each accepted pattern p, add $unk(p)$ to $candidates$
6. For every learner, score candidate name t
$M_t \leftarrow$ set of accepted patterns matching any instance of t

$$\begin{cases} Discard(t) & \text{If } M_t\ lacks\ Mass\ or\ Balance \\ Rank(t) & \text{Otherwise} \end{cases}$$

Learner accepts top-scoring k percent of names (maximum of m)
7. Repeat until no more names can be learned

Algorithm 3: Pseudo-code for NOMEN algorithm

The next step is to tag each instance of every *accepted name* occurring in the corpus, e.g., <disease>*mad cow disease*</disease>. Then for both the opening and closing tags, an extraction pattern is generated around the tag using a *context window*. This window size was set to 3 in this work. The pattern is then generalized in some way by replacing some elements in the context window with wildcards. It is completely unclear from either paper what the $generalized\ pattern$ would look like or what kinds of text it would match.

There is one $learner$ per semantic category. For each $learner$, the $generalized\ patterns$ are then matched against the training corpus. Since each pattern was created from either the opening or closing tag, the pattern will only match either the left or the right boundary of a name. NOMEN uses a regular expression of the form $[Adj * Noun+]$ to determine where the other boundary of the word ends. Each learner then checks whether the matched name is a $positive$ (already in this learner's pool of *accepted names*),

[6] We use the term "robust" here not in the software sense of handling abnormal situations gracefully, but rather to indicate a broad capacity to solve problems despite varying approaches and implementations. This is contrasted with a weak or fragile paradigm, which can only adequately solve problems under a specific set of artificially chosen parameters and/or conditions. Assessing the strengths of classes of algorithms is a major issue in Computer Science, as failed approaches go unpublished in the literature.

$negative$ (already in another learner's pool), or $unknown$ (in no pool) name.

Each pattern, for each learner, is then scored using the number of $unique$ matches of each type: $pos(p)$, $neg(p)$, and $unk(p)$. The pattern is then scored using $accuracy$ and $confidence$ metrics:

$$acc(p) = \frac{|pos(p)|}{|pos(p)| + |neg(p)|} \quad (5)$$

$$conf(p) = \frac{|pos(p)| - |neg(p)|}{|pos(p|) + |neg(p)| + |unk(p)|} \quad (6)$$

where patterns with an accuracy of less than Θ_{prec} are removed from consideration. The rest are given a final score of $Score(p) = conf(p) \cdot log|pos(p)|$. The top n patterns for each learner are then accepted.

Candidate names are then drawn from the union of the unknown names of accepted patterns ($unk(p)$). Each candidate t must have an acceptable $mass$ M_t (defined as the # of accepted patterns that generated t } of 2 or greater and be $balanced$ (at least one accepted pattern matched both left and right name boundaries). Names that meet this criteria are scored:

$$rank(t) = 1 - \prod_{p \in M_t} (1 - conf(p)) \quad (7)$$

and each learner then finally accepts up to top-scoring k percent, but no more than m in any given cycle.[7] The cycle is then repeated until no more acceptable names exist.

2.4 AutoSlog-TS

Riloff and Wiebe (2003) uses bootstrapping to identify subjective expressions. Empirical methods have previously been leveraged to create lists of words and n-grams statistically associated with subjective language. Previous work has also resulted in lists of manually compiled subjective words.[8]

However, subjective ideas are expressed in myriad forms, many of which are very hard to detect using conventional natural language processing approaches. Sarcastic, metaphorical, and idiomatic phrases are cases where the expression of subjectivity is easily overlooked by empirical methods; both utilize words which may be objective when isolated but may become strongly subjective when placed in context.[9] While some idioms, such as *kicked the bucket* may be easily captured by an n-gram of sufficient length, many others contain pronouns or proper nouns (as in *dealt John a mortal blow*).

Riloff and Wiebe (2003) use an unnamed variant of their previously defined AutoSlog-TS algorithm to address some of these issues. The work is notable in that it bootstraps *extraction patterns* instead of words. An extraction pattern is essentially a regular expression to capture some lexico-syntactic pattern in text, such as *¡x¿ drives ¡y¿ up the wall* , where x and y can match any arbitrary part-of-speech (a noun or noun phrase would make sense in this case). Riloff and Wiebe (2003) hypothesize that extraction patterns are better suited for capturing non-compositional meaning than other representations like words or n-grams.

Particularly worth noting is that this process is the only one that does not use manually selected seeds. The original set of extraction patterns are instead created by what is termed a *High-precision subjective classifier* (HP-Subj). This classifier flags sentences as subjective or objective using a large list of lexical items. These items are words, N-grams, and other subjective clues compiled from a wide variety of prior works which showed them to be good indicators of subjectivity. There is also a corresponding *High-precision objectivity classifier* (HP-Obj) that labels sentences it believes are objective. It functions somewhat differently: It labels sentences that have a dearth of subjective clues within them *and* in the surrounding textual context.

The Autoslog-TS algorithm takes as input the set of relevant (subjective) and irrelevant (objective) sentences flagged by the HP-Subj and HP-Obj classifiers. First, syntactic templates generated from prior work are exhaustively applied to the training corpus, so that extraction patterns are generated for *every possible* instantiation of the templates that appears in the training corpus. The list of templates and examples of learned extraction patterns are shown below.

Then, these extraction patterns are matched against every sentence in the training corpus. The

[7] Both implementations used a k of 5% and an m of 5.

[8] A well-known example is the "Verbs of Desire" class composed by Beth Levin.

[9] This is called non-compositional meaning as it defies the linguistic *Principle of Compositionality*.

SYNTACTIC FORM	EXAMPLE PATTERN
<subj> passive-verb	<subj> was satisfied
<subj> act-verb	<subj> complained
<subj> act-verb dobj	<subj> dealt blow
<subj> verb infinitive	<subj> appear to be
<subj> aux noun	<subj> has position
active-verb <dobj>	endorsed <dobj>
infinitive <dobj>	to condemn <dobj>
verb infinitive <dobj>	get to know <dobj>
noun aux <dobj>	fact is <dobj>
noun prep <np>	opinion on <np>
active-verb prep <np>	agrees with <np>
pass-verb prep <np>	was mad about < np>
infinitive prep <np>	to resort to <np>

Table 2: Syntactic templates and example patterns instantiated from them (Riloff and Wiebe, 2003)

frequency with which each extracted pattern occurs in known subjective and objective sentences (labeled earlier by the HP-Subj and HP-Obj classifiers) is calculated and this information is used to rank each pattern using the scoring function:

$$Pr(subj|pattern_i) = \frac{subjfreq(pattern_i)}{freq(pattern_i)} \quad (8)$$

Where $subjfreq(pattern_i)$ is the number of times the pattern matched part of a known subjective sentence and $freq(pattern_i)$ is the total number of times it matched any sentence. This is essentially the "precision" of the pattern.

After all patterns are scored, two threshold measures are used to select the patterns deemed most subjective. Patterns are selected for which $freq(pattern_i) \geq \Theta_1$ and $Pr(subj|pattern_i) \leq \Theta_2$. The new sentences are then mixed in with the original sentences and the extraction pattern learner is re-run to complete the bootstrapping cycle.

2.5 MetaBoot + Basilisk

Previously mentioned approaches have used a single algorithm to perform bootstrapping. Another approach employed by Riloff et al. (2003) is to use multiple algorithms simultaneously to achieve a synergistic effect. This can perform well if the combined algorithms each find a different set of terms. The goal is again to identify subjective language. Specifically, it is framed as a classification problem: Classify sentences in the corpus as either subjective or objective. The bootstrapping is performed on subjective nouns rather than on extraction patterns as is the case in Riloff and Wiebe (2003).

As the *Basilisk* algorithm was already discussed in Section 2.2 and *MetaBoot* is not the core focus of this approach, we will only provide a brief description of how the *MetaBoot* algorithm functions.[10] Like *Basilisk*, it was originally designed for semantic categorization tasks. It is re-purposed in this paper for learning subjective nouns. It is seeded with words belonging to the desired class ($subjective$ in this case), and then creates extraction patterns by instantiating templates that extract every noun phrase in the corpus. The patterns are scored based upon the number of seeds extracted. The best pattern is saved and then the head noun from $every$ extracted noun phrase is accepted into the category. The patterns are then re-scored and the cycle repeats.

What differentiates MetaBoot from previously discussed algorithms is that it has a second level of bootstrapping. After each cycle completes, all nouns put into the dictionary during the cycle are scored based on the number of patterns that extracted it. Only the five best are allowed to remain.

Both algorithms were used in tandem and run for 400 iterations. Two were used because they captured different sets of subjective nouns while being provided with the same seeds. Since the objective is to produce a $lexicon$ of $subjective$ $nouns$, this was a desirable feature. The authors would have had to come up with different seed words in order to produce more results from a single system. The final lexicon was a union of the output of the two algorithms.

3 Bootstrapping Corpora

All the corpora used in these experiments were unannotated. This is because bootstrapping is a process that needs no additional markup and is therefore ideal for tasks on unannotated natural language text. Furthermore, many of the tasks are such that annotations would be unhelpful, e.g., building lexicons. Agichtein and Gravano (2000), for the *Snowball* algorithm, used the *North American News Text Corpus* spanning from 1995 to mid-1997. They split this corpus up into a training and a testing set: The training set was the 178,000 documents from 1996, and the testing set was the 142,000 documents from 1995 and 1997.

[10] For a full description, see (Riloff and Jones, 1999).

Thelen and Riloff (2002), with the *Basilisk* algorithm, utilized the *MUC-4 Proceedings* corpus from 1992. This contained 1,700 documents focusing on terrorism.

Yangarber et al. (2002) and Lin et al. (2003) both used the *ProMED* corpus for their experiments with the *NOMEN* algorithm. This corpus contained articles written in the biomedical field. Yangarber et al. (2002) used 5,100 *ProMED* articles spanning the years from 1999 to 2001, while Lin et al. (2003) used a smaller subset containing only 1,400 articles. They also performed some experiments on learning proper names in Chinese. Since there are no case distinctions in Chinese, this bears some similarity to the task of learning generalized names by Yangarber et al. (2002). They used a training corpus consisting of approximately 700,000 words from the *Beijing University Institute of Computational Linguistics* corpus.

Riloff and Wiebe (2003) and Riloff et al. (2003), using *AutoSlog-TS* and *MetaBoot* + *Basilisk* respectively, both utilized the *U.S. Foreign Broadcast Information Service* (FBIS) corpus. This corpus contains news stories in multiple languages, paired with their translations in English (e.g., pairs of English-Chinese stories). From this corpus, 302,163 sentences were used.

4 Experimental Results

As discussed earlier, Agichtein and Gravano (2000) performed the task of attempting to extract *¡ORGANIZATION, LOCATION¿* relations, specifically a company and its headquarters. The precision and recall of their algorithm, as compared to a baseline, the *DIPRE* system that Snowball is based on, and an alternate version of Snowball that discards punctuation (Snowball-plain) is detailed in Figure 1. The baseline is computed by tabulating each co-occurrence of an organization and a location and assuming the most frequently co-occurring location is the headquarters of the organization. The x-axis in these charts represents successive runs, where the number of times a tuple needs to be found before it is accepted is increased. As we can see, the baseline does quite well, but *Snowball* still comes out on top in both precision and recall. The results are reported after 3 iterations because Snowball's results quickly stabilize (see Figure 4). Thelen and Riloff (2002) utilized their *Basilisk* algorithm to build a semantic lexicon for six categories. The authors manually

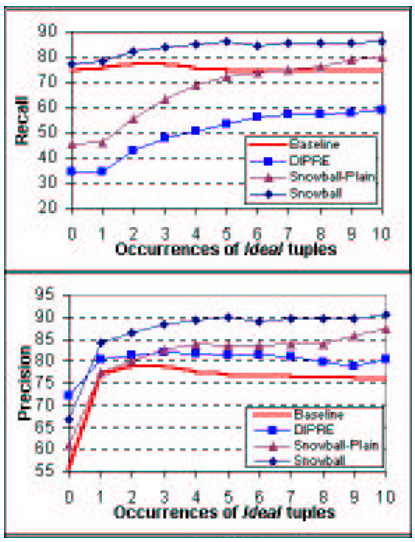

Figure 1: Precision and recall of the Snowball system Agichtein and Gravano (2000)

scored each word in the resulting lexicon for correctness and did the same thing for an implementation of *MetaBoot* that they used as a baseline. Results were compared at various points between 100 and 1000 bootstrapping iterations. *Basilisk* substantially outperformed *MetaBoot* at every iteration, especially when it learned its semantic categories in tandem instead of sequentially.

Yangarber et al. (2002) compared their implementation of the NOMEN algorithm to several manually constructed reference lists for precision and recall that they used as their gold standard. Figure 2 shows the results of their best score (*Dis + Loc + Sym + Other*) for the semantic category of disease names.

Riloff and Wiebe (2003) and Riloff et al. (2003) This score was obtained by simultaneously boot-

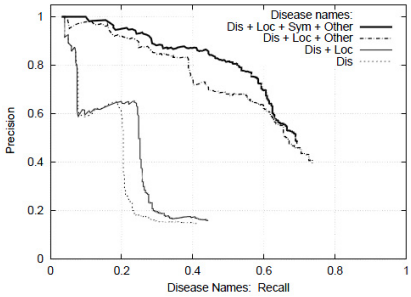

Figure 2: NOMAN precision/recall
Yangarber et al. (2002)

strapping multiple semantic categories at the same time. Without a baseline to compare to, it is dif-

ficult to evaluate success (see Section 5.1 for a discussion of this problem). However, Lin et al. (2003) performed some similar experiments but greatly increased the number of semantic categories being simultaneously learned. We can see the resulting heightened precision in Figure 3 confirms the trend.

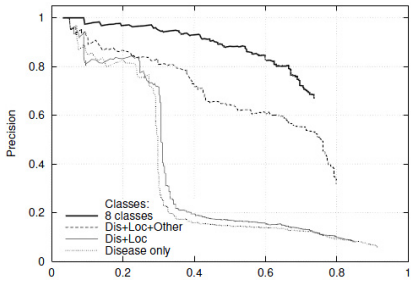

Figure 3: NOMAN precision/recall
Lin et al. (2003)

The *AutoSlog-TS* system detailed in Riloff et al. (2003) performed quite well according to the metrics reported. Their system learned 17,073 patterns that tended to extract subjective expressions. To evaluate these, they ran different subsets of the patterns on a manually annotated test set that contained 54% subjective sentences. The resulting precision when extracting subjective sentences ranged between 71% and 85%.

The *MetaBoot + Basilisk* goal pursued by Riloff et al. (2003) was also building a semantic lexicon. In Table 3 we see that the resulting lexicon was manually reviewed and tagged by the authors as either strongly or weakly subjective words. Here we see how the merging of lexicons produced by each algorithm produces a final lexicon that is significantly larger. To give some insight about the accuracy of the system, after 1000 generations *Basilisk* had a lexicon containing approximately 55% subjective words. *MetaBoot* contained approximately 29%. After 2000 generations, this had fallen to roughly 44% and 25%, respectively.

	B	M	B ∩ M	B ∪ M
StrongSubj	372	192	110	454
WeakSubj	453	330	185	598
Total	825	522	295	1052

Table 3: Subjective words in lexicon
Riloff et al. (2003)

5 Discussion

The previous discussions have discussed the philosophy behind bootstrapping and provided a high-level overview of how several seminal bootstrapping algorithms are implemented. We then evaluated what kind of data they were trained on and how they performed. In the following sections, we discuss the strengths, weaknesses, and idiosyncrasies of the bootstrapping model.

5.1 Evaluation Methodologies

The bootstrapping tasks surveyed here were trained and tested on unannotated corpora. This posed difficulties when attempting to evaluate the performance of the bootstrapper, as there was often no gold standard against which to compare the bootstrapping results.

Agichtein and Gravano (2000) used their *Snowball* system to extract $Organization \rightarrow Headquarters$ relations from an unannotated corpus, but then they had no gold standard against which to check. They ultimately solved their lack of an ideal set for their extracted relations by downloading a publicly available directory of company information and parsing out the requisite values. Lin et al. (2003) manually compiled two lists of gold standards themselves in order to have something to compare their results against. Riloff et al. (2003) and Riloff and Wiebe (2003) both had to manually annotate their resulting lexicons in order to ascertain whether they even contained any results. Even after manually finding the subjective words in their results, they still did not have any kind of baseline to compare it to. These difficulties in evaluating the performance are a hindrance to the development and adoption of bootstrapping algorithms.

Evaluation is further confused by the fact that some bootstrapping algorithms are evaluated using different standards than others. Yangarber et al. (2002) note when presenting their results that algorithms can have their recall and precision scored for each $instance$ of a word correctly tagged in the corpus (termed *instance-based or token-based* evaluation), or they can be scored only once per word correctly found (*type-based*). He noted that which method was used often depends on the end goal: If the goal is to tag each instance in the corpus with a semantic category, then *token-based* makes sense. If the goal is to build a lexicon, then *type-based* makes sense. However,

since both objectives can be achieved by the same algorithm just keeping track of different things, this can making comparison efforts difficult.

5.2 Role of Bootstrapping

As broad, all-encompassing hierarchical semantic databases like WordNet continue to mature, it is reasonable to wonder if techniques for identifying semantic categories will become obsolete. However, there are many reasons why this is unlikely, although it may be relegated to niche positions. General semantic databases like WordNet are not designed for domain-specific lingo/jargon and techniques like bootstrapping may be the only solution to such tasks (Thelen and Riloff, 2002).

Additionally, specific tasks may always need bootstrapping. *Snowball* bootstraps specific relations; it maps a semantic class onto a related semantic class. *Basilisk* bootstraps nouns into arbitrary semantic classes. We saw it used for both identification of subjective words and the semantic categorization of other words. There are a (practically unlimited) number of potential ad hoc semantic classes and relations between the two. At their core, bootstrapping algorithms are not limited to learning general semantic categories. They are useful for learning any category or relationship for which words appear in similar linguistic phrases or contexts.

5.3 Weaknesses of Bootstrapping

5.3.1 Semantic Drift

A very well known flaw in bootstrapping is a phenomenon known as *semantic drift* or *creep*. This occurs when, after multiple iterations, the bootstrapper wanders away from the original semantic meaning of the seeds and begins to accept incorrect or undesirable entities.

This can occur if entities have more than one semantic word sense; Yangarber et al. (2002) note that bootstrapping disease names may also begin to pick up symptoms because some symptoms and diseases share names (i.e., *encephalitis*). It can also occur if words similar meaning and usage but have different undertones.

Consider a hypothetical bootstrapper that was seeded with $discussion$. On further iterations, it might pick up $debate \rightarrow disagreement \rightarrow argument \rightarrow altercation \rightarrow brawl$, if they all appeared in similar contexts in the training corpus (for example, "I had a(n) $<x>$ with him during dinner"). While each iteration may be semantically similar to the last word added ($discussion$ and $debate$, $argument$ and $altercation$, etc.), they share less and less semantic meaning with the original seed word. By the end, $brawl$ is hardly related at all to $discussion$.

There are several ways to ward against this kind of semantic creep. The chief line of defense is the evaluation of candidates before adding them to the system. All of the bootstrapping systems surveyed had some kind of candidate evaluation; most used a confidence measure. The effectiveness of this confidence measure is clarified by comparing the performance of the Snowball system to DIPRE (See Figure 4). The DIPRE system peaks and then quickly plummets since "it has no way to prevent unreliable tuples from being seed[s] for its next iteration". Another technique employed by

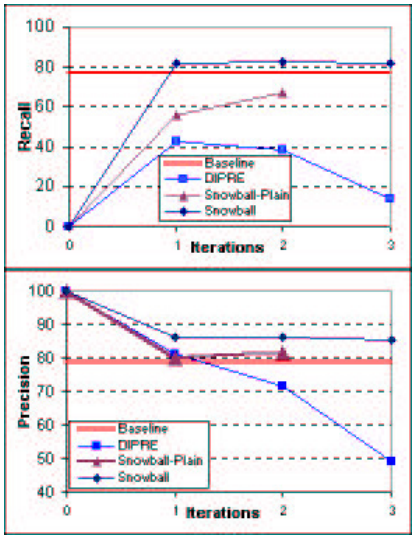

Figure 4: Effects of confidence measure, DIPRE vs Snowball

multiple authors ((Thelen and Riloff, 2002; Yangarber et al., 2002; Lin et al., 2003)) is the parallel or simultaneous learning of various semantic categories, instead of learning them serially. This offers an advantage that may not be immediately obvious: Each category's seeds will accept the most closely related terms to it during each iteration. When terms are forbidden from being shared between categories, this effectively partitions the search space and limits how far a category can grow before being constrained by the others. Another way simultaneous learning can help was discussed by Lin et al. (2003). Their system not only prohibited semantic categories from

including terms already included in another category, but also discouraged each semantic category from *learning patterns* that matched terms included in too many other categories. In this way, it provided a self-stabilizing or auto-correcting feature to the bootstrapping feature to help keep it "on track". These techniques, however, can only be utilized when the search space is partitionable.

5.3.2 Stop conditions

One major drawback that bootstrapping suffers is the difficulty in knowing when to stop the iterative cycles. Some (Riloff et al., 2003) stopped after a set and seemingly arbitrary number of iterations, while others (Lin et al., 2003) stopped when their algorithm had exhaustively added all candidates or rejected any remaining candidates. The metrics used to determine whether a bootstrapping algorithm should terminate are often of a dubious nature. For example, the confidence measures used by the majority of the bootstrapping algorithms we surveyed cannot make any guarantees about precision or recall. They can simply tell whether what they have is arbitrarily similar enough to what they are now finding.

5.3.3 Seeding Decisions

As we have seen, the choosing of seeds is arguably the most critical step in bootstrapping. However, most authors simply chose the most frequently occurring words in their corpus that they have quickly identified belong to the category they are interested in. While this ensures that the greatest amount of contextual information will be available to learn from, it does nothing to ensure the $quality$ of the contexts. It is easy to imagine improperly chosen seeds that would pick up tons of extraction patterns that, in turn, extract very poor additional words, ultimately producing poor results. None of the surveyed literature addressed how to find seed choices beyond taking the most frequent words. It would be interesting to read literature on how different seeding choices impacted the final performance of the system and provide guidelines or rules for making seeding decisions.

6 Conclusion

We examined in detail multiple approaches to bootstrapping for natural language processing tasks. We explored not only a variety of different goals but also multiple techniques.

We conclude that bootstrapping is a powerful method for extracting useful data from enormous volumes of unstructured text. However, bootstrapping systems must be carefully seeded and constrained to avoid growing in unwanted directions or becoming overly diluted with irrelevant data. The choice of seeds is pivotal to the success of the bootstrapping process and it is not at all clear how to determine what the "best" seed might be.

Bootstrapping is amenable to a wide variety of natural language processing tasks, such as semantic categorization, relation extraction, and subjectivity analysis. Bootstrapping systems are well suited to natural language tasks due to their ability to learn and navigate the syntactically rich, unstructured, and extremely complex nature of loosely structured natural languages.

References

Agichtein, Eugene and Gravano, Luis. 2000. Snowball: Extracting relations from large plain-text collections. In *Proceedings of the fifth ACM conference on Digital libraries*, pages 85–94. ACM.

Brin, Sergey. 1999. Extracting patterns and relations from the world wide web. In *The World Wide Web and Databases*, pages 172–183. Springer Berlin Heidelberg.

Thelen, Michael and Riloff, Ellen. 2002. A bootstrapping method for learning semantic lexicons using extraction pattern contexts. In *Proceedings of the ACL-02 conference on Empirical methods in natural language processing*, Volume 10, pages 214–221. Association for Computational Linguistics.

Yangarber, Roman, Lin, Winston, and Grishman, Ralph. 2002. Unsupervised learning of generalized names. In *Proceedings of the 19th international conference on Computational linguistics*, Volume 1. Association for Computational Linguistics.

Lin, Winston, Yangarber, Roman, and Grishman, Ralph. 2003. Bootstrapped learning of semantic classes from positive and negative examples. In *Proceedings of ICML-2003 Workshop on The Continuum from Labeled to Unlabeled Data*, Volume 4, No. 4.

Riloff, Ellen, Wiebe, Janyce and Wilson, Theresa. 2003. Learning subjective nouns using extraction pattern bootstrapping. In *Proceedings of the seventh conference on Natural language learning at HLT-NAACL*, Volume 4, pages 25–32. Association for Computational Linguistics.

Riloff, Ellen and Wiebe, Janyce. 2003. Learning extraction patterns for subjective expressions. In *Proceedings of the 2003 conference on Empirical meth-*

ods in natural language processing, pages 105–112. Association for Computational Linguistics.

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