Sample Selection for Statistical Parsing

Rebecca Hwa* University of Pittsburgh

Corpus-based statistical parsing relies on using large quantities of annotated text as training examples. Building this kind of resource is expensive and labor-intensive. This work proposes to use sample selection to find helpful training examples and reduce human effort spent on annotating less informative ones. We consider several criteria for predicting whether unlabeled data might be a helpful training example. Experiments are performed across two syntactic learning tasks and within the single task of parsing across two learning models to compare the effect of different predictive criteria. We find that sample selection can significantly reduce the size of annotated training corpora and that uncertainty is a robust predictive criterion that can be easily applied to different learning models.

1 Introduction

Many learning tasks for natural language processing require supervised training; that is, the system successfully learns a concept only if it was given annotated training data. For example, while it is difficult to induce a grammar with raw text alone, the task is tractable when the syntactic analysis for each sentence is provided as a part of the training data (Pereira and Schabes, 1992). Current state-of-the-art statistical parsers (Collins, 1999; Charniak, 2000) are all trained on large annotated corpora such as the Penn Tree-

* Computer Science Department, Pittsburgh, PA 15260

bank (Marcus, Santorini, and Marcinkiewicz, 1993). However, supervised training data is difficult to obtain; existing corpora might not contain the relevant type of annotation, and the data might not be in the domain of interest. For example, one might need lexical-semantic analyses in addition to the syntactic analyses in the treebank, or one might be interested in processing languages, domains, or genres for which there are no annotated corpora. Because supervised training demands significant human involvement (e.g., annotating the syntactic structure of each sentence by hand), creating a new corpus is a labor-intensive and time-consuming endeavor. The goal of this work is to minimize a system's reliance on annotated training data.

One promising approach to mitigate the annotation bottleneck problem is to use *sample selection*, a variant of active learning. It is an interactive learning method in which the machine takes the initiative in selecting unlabeled data for the human to annotate. Under this framework, the system has access to a large pool of unlabeled data, and it has to predict how much it can learn from each candidate in the pool if that candidate is labeled. More quantitatively, we associate each candidate in the pool with a *Training Utility Value* (TUV). If the system could accurately identify the subset of examples with the highest TUV, it would have located the most beneficial training examples, thus freeing the annotators from having to label less informative examples.

In this paper, we apply sample selection to two syntactic learning tasks: training a prepositional phrase attachment (PP-attachment) model, and training a statistical parsing model. We are interested in addressing two main questions. First, what are good predictors of a candidate's training utility? We propose several predictive criteria and define evaluation functions based on them to rank the candidates' utility. We have performed experiments comparing the effect of the evaluation functions on the size of the training corpus. We find that, with a judiciously chosen evaluation function, sample selection can significantly reduce the size of the training corpus. The second main question is: are the predictors consistently effective for different types of learners? We compare the predictive criteria both across tasks (between PP-attachment and parsing) and within a single task (applying the criteria to two parsing models: an EM-trained parser and a count-based parser). We find that the learner's uncertainty is a robust predictive criterion that can be easily applied to different learning models.

2 Learning with Sample Selection

Unlike traditional learning systems that receive training examples indiscriminately, a sample selection learning system actively influences its progress by choosing new examples to incorporate into its training set. There are two types of selection algorithms: *committee-based* or *single learner*. A committee-based selection algorithm works with multiple learners, each maintaining a different hypothesis (perhaps pertaining to different aspects of the problem). The candidate examples that lead to the most disagreements among the different learners are considered to have the highest TUV (Cohn, Atlas, and Ladner, 1994; Freund et al., 1997). For computationally intensive problems, such as parsing, keeping multiple learners may be impractical.

In this work, we focus on sample selection using a single learner that keeps one working hypothesis. Without access to multiple hypotheses, the selection algorithm can nonetheless estimate the TUV of a candidate. We identify the following three classes of predictive criteria:

Problem-space: Knowledge about the problem-space may provide information about the type of candidates that are particularly plentiful or difficult to learn. This criterion focuses on the general attributes of the learning problem, such as the distribution of the input data and properties of the learning algorithm, but it ignores the current state of the hypothesis.

Performance of the hypothesis: Testing the candidates on the current working hypothesis shows the type of input data on which the hypothesis may perform weakly. That is, if the current hypothesis is unable to label a candidate or is uncertain about it, then it might be a good training example (Lewis and Catlett, 1994). The underlying assumption is that an uncertain output is likely to be wrong.

Parameters of the hypothesis: Estimating the potential impact that the candidates will have on the parameters of the current working hypothesis. This locates those examples that will change the current hypothesis the most.

Figure 1 outlines the single-learner sample selection training loop in pseudo-code. Initially, the training set, L, consists of a small number of labeled examples, based on which the learner proposes its first hypothesis of the target concept, C. Also available to the learner is a large pool of unlabeled training candidates, U. In each training iteration, the selection algorithm, $Select(n, U, C, f)$, ranks the candidates of U according to their expected TUVs and returns the n candidates with the highest values. The algorithm predicts the TUV of each candidate, $u \in U$, with an evaluation function, $f(u, C)$. This function may rely on the hypothesis concept C to estimate the utility of a candidate u. The set of the n chosen candidates are then labeled by human experts and added to the existing training set. Running the learning algorithm, $Train(L)$, on the updated training set, the system proposes a new hypothesis that is the most compatible with the examples seen thus far. The loop continues until one of three stopping conditions is met: the hypothesis is considered to perform well enough, all candidates are labeled, or an absolute cut-off point is reached (e.g., no more resources).

U is a set of unlabeled candidates.
L is a set of labeled training examples.
C is the current hypothesis.
Initialize:
  $C \leftarrow Train(L)$.
Repeat
  $N \leftarrow Select(n, U, C, f)$.
  $U \leftarrow U - N$.
  $L \leftarrow L \cup Label(N)$.
  $C \leftarrow Train(L)$.
Until (C is good enough)**or** ($U = \emptyset$) **or** (cut-off).

Figure 1

The pseudo-code for the sample selection learning algorithm

3 Sample Selection for Prepositional Phrase Attachment

One common source of structural ambiguities arises from syntactic constructs in which a prepositional phrase might be equally likely to modify the verb or the noun preceding it. Researchers have proposed many computational models for resolving PP-attachment ambiguities. Some well-known approaches include rule-based models (Brill and Resnik, 1994), backed-off models (Collins and Brooks, 1995), and a maximum-entropy model (Ratnaparkhi, 1998). Following the tradition of using learning PP-attachment as a way to gain insight into the parsing problem, we first apply sample selection to reduce the amount of annotation used in training a PP-attachment model. We use the Collins-Brooks model as the basic learning algorithm, and experiment with several evaluation functions based on the types of predictive criteria described earlier. Our experiments show that the best evaluation function can reduce the number of labeled examples by nearly half without loss of accuracy.

3.1 A Summary of the Collins-Brooks Model

The Collins-Brooks model takes prepositional phrases and their attachment classifications as training examples, each is represented as a quintuple of the form $(v, n, p, n2, a)$, where

```
subroutine Train(L)
  foreach ex ∈ L do
    extract (v, n, p, n2, a) from ex
    foreach tuple ∈ {(v, n, p, n2), (v, p, n2), (n, p, n2), (v, n, p), (v, p), (n, p), (p, n2), (p)} do
      Count(tuple) ← Count(tuple) + 1
      if a = noun then
        Count_NP(tuple) ← Count_NP(tuple) + 1

subroutine Test(U)
  foreach u ∈ U do
    extract (v, n, p, n2) from u
    if Count(v, n, p, n2) > 0 then
      prob ← Count_NP(v,n,p,n2) / Count(v,n,p,n2)
    elsif Count(v, p, n2) + Count(n, p, n2) + Count(v, n, p) > 0 then
      prob ← (Count_NP(v,p,n2)+Count_NP(n,p,n2)+Count_NP(v,n,p)) / (Count(v,p,n2)+Count(n,p,n2)+Count(v,n,p))
    elsif Count(v, p) + Count(n, p) + Count(p, n2) > 0 then
      prob ← (Count_NP(v,p)+Count_NP(n,p)+Count_NP(p,n2)) / (Count(v,p)+Count(n,p)+Count(p,n2))
    elsif Count(p) > 0 then
      prob ← Count_NP(p) / Count(p)
    else prob ← 1
    if prob ≥ .5 then
      output noun
    else output verb
```

Figure 2

The Collins-Brooks PP-Attachment Classification Algorithm.

v, n, p, and $n2$ are the head words of the verb phrase, the object noun phrase, the preposition, and the prepositional noun phrase, respectively; and a specifies the attachment classification. For example, (*"wrote a book in three days", attach-verb*) would be annotated as $(wrote, book, in, days, verb)$. The head words can be automatically extracted using a heuristic table-lookup in the manner described by Magerman (1994). For this learning problem, the supervision is the one-bit information of whether p should attach to v or to n. In order to learn the attachment preferences of prepositional phrases, the system builds attachment statistics for each the *characteristic tuple* of all training examples. A characteristic tuple is some subset of the four head words in the example, with the condition that one of the elements must be the preposition. Each training example forms eight characteristic tuples: $(v, n, p, n2), (v, n, p), (v, p, n2), (n, p, n2), (v, p), (n, p), (p, n2), (p)$. The attachment statistics are a collection of the occurrence frequencies for all the characteristic tuples in the training set and the occurrence frequencies for the characteristic tuples of those examples determined to attach to nouns. For some characteristic tuple t, $Count(t)$ denotes the former and $Count_{NP}(t)$ denotes the latter. In terms of the sample selection algorithm, the collection of counts represents the learner's current hypothesis (C in Figure 1). Figure 2 provides the pseudo-code for the **Train** routine.

Once trained, the system can be used to classify the test cases based on the statistics of the most similar training examples and back off as necessary. For instance, to determine the PP-attachment for a test case, the classifier would first consider the ratio of the two frequency counts for the 4-word characteristic tuple of the test case. If the tuple never occurred in the training example, the classifier would then back off to look at its three 3-word characteristic tuples. It would continue to back off further, if necessary. In the case that the model has no information on any of the characteristic tuples of the test case, it would, by default, classify the test case as an instance of noun-attachment. Figure 3 shows using the back-off scheme on a test case. We describe in the **Test** pseudo-code routine in Figure 2 the model's classification procedure for each back-off level.

3.2 Evaluation Functions

Based on the three classes of predictive criteria discussed in Section 2, we propose several evaluation functions for the Collins-Brooks model.

3.2.1 The Problem Space One source of knowledge to exploit is our understanding of the PP-attachment model and properties of English prepositional phrases. For instance, we know that the most problematic test cases for the PP-Attachment model are those for which it has no statistics at all. Therefore, those data that the system has not yet encountered might be good candidates. The first evaluation function we define,

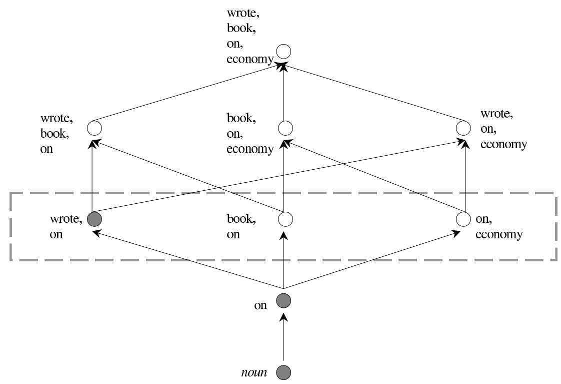


Figure 3

In this example, the classification of the test case preposition is backed-off to the two-word tuple level. In the diagram, each circle represents a characteristic tuple. A filled circle denotes that the tuple has occurred in the training set. The grey rectangular box indicates the back-off level on which the classification is made.

$f_{novel}(u, C)$, equates the TUV of a candidate u with its degree of novelty, the number of its characteristic tuples that currently have zero counts.[1]

$$f_{novel}(u, C) = \sum_{t \in Tuples(u)} \begin{cases} 1 & : & Count(t) = 0 \\ 0 & : & \text{otherwise} \end{cases}$$

This evaluation function has some blatant defects. It may distort the data distribution so much that the system would not be able to build up a reliable collection of statistics. The function does not take into account the intuition that those data that rarely occur, no matter how novel, probably have overall low training utility. Moreover, the scoring scheme does not make any distinction between the characteristic tuples of a candidate.

We know, however, that the PP-attachment classifier is a back-off model that makes its decision based on statistics of the characteristic tuple with the most words first. A more sophisticated sampling of the data domain should consider not only the novelty of

1 Note that the current hypothesis C is ignored in evaluation functions of this class because they depend only on the knowledge about the problem space.

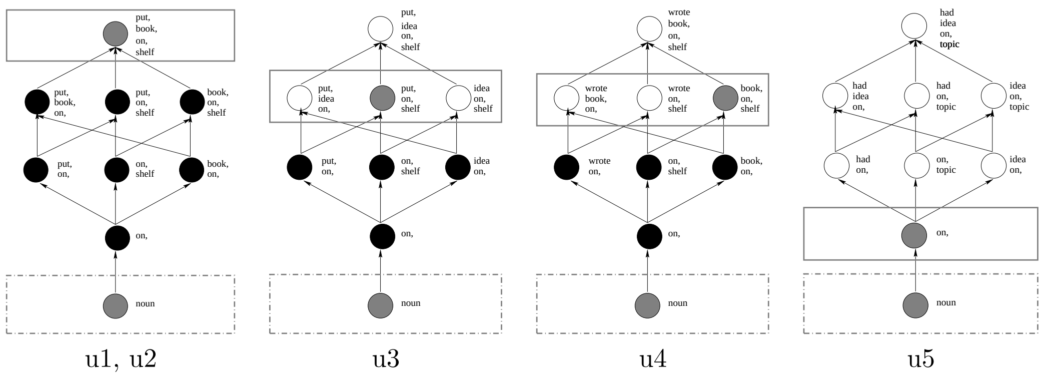


Figure 4

If candidate u_1 were selected, a total of 22 tuples can be ignored. The dashed rectangles show the classification level before training and the solid rectangles show the classification level after the statistics of u_1 have been taken. The obviated tuples are represented by the filled black circles.

the data, but also the frequency of its occurrence as well as the quality of its characteristic tuples. We define a back-off model-based evaluation function, $f_{backoff}(u, C)$, that scores a candidate u by counting the number of characteristic tuples obviated in all candidates if u were included in the training set. For example, suppose we have a small pool of 5 candidates, and we are about to pick the first training example:

u_1 = (put, book, on, shelf),

u_2 = (put, book, on, shelf),

u_3 = (put, idea, on, shelf),

u_4 = (wrote, book, on, shelf),

u_5 = (had, idea, on, topic)

According to $f_{backoff}$, either u_1 or u_2 would be the best choice. By selecting either as the first training example, we could ignore all but the four-word characteristic tuple for both u_1 and u_2 (a saving of 7 tuples each); since u_3 and u_4 each has three words in common with the first two candidates, they would no longer depend on their lower 4 tuples; and although we would also improve the statistics for one of u_5's tuple: *(on)*, nothing could be pruned from u_5's characteristic tuples. Thus, $f_{backoff}(u_1, C) = f_{backoff}(u_2, C) = 7 + 7 + 4 + 4 = 22$ (See Figure 4).

Under $f_{backoff}$, if u_1 were chosen as the first example, u_2 loses all its utility because we cannot prune any extra characteristic tuples by using u_2. That is, in the next round of selection, $f_{backoff}(u_2, C) = 0$. Candidate u_5 would be the best second example because it now has the most tuples to prune (7 tuples).

The evaluation function $f_{backoff}$ improves upon f_{novel} in two ways. First, novel candidates that occur frequently are favored over those that rarely come up. As we have seen in the above example, a candidate that is similar to other candidates can eliminate more characteristic tuples all at once. Second, the evaluation strategy follows the working principle of the back-off model and discounts lower-level characteristic tuples that do not affect the classification process, even if they were "novel." For instance, after selecting u_1 as the first training example, we would no longer care about the two-word tuples of u_4 such as *(wrote, on)* even though we have no statistics for it.

A potential problem for $f_{backoff}$ is that after all the obvious candidates have been selected, the function would not be very good at differentiating between the remaining candidates that have about the same level of novelty and occur infrequently.

3.2.2 The Performance of the Hypothesis The evaluation functions discussed in the previous section score candidates based on prior knowledge alone, independent of the current state of the learner's hypothesis and the annotation of the selected training examples. To attune the selection of training examples to the learner's progress, an evaluation function might factor in its current hypothesis in predicting a candidate's TUV.

One way to incorporate the current hypothesis into the evaluation function is to score each candidate using the current model, assuming its hypothesis is right. An error-driven evaluation function, f_{err}, equates the TUV of a candidate with the hypothesis' estimate of its likelihood to mis-classify that candidate (i.e., one minus the probability of the most-likely class). If the hypothesis predicts that the likelihood of a prepositional phrase to attach to the noun is 80%, and if the hypothesis is accurate, then there is a 20% chance that it has mis-classified.

A related evaluation function is one that measures the hypothesis' *uncertainty* across all classes, rather than focusing on only the most likely class. Intuitively, if the hypothesis classifies a candidate as equally likely to attach to the verb as the noun, it is the most uncertain. If the hypothesis assigns a candidate to a class with a probability of 1, then it is the most certain of its answer. For the binary class case, the uncertainty-based evaluation function, f_{unc}, can be expressed in the same way as the error-driven function, as a function that is symmetric about 0.5 and monotonically decreases if the hypothesis prefers one class over another[2]:

$$\begin{aligned} f_{unc}(u, C) &= f_{err}(u, C) \\ &= \begin{cases} 1 - P(noun \mid u, C) & : \quad P(noun \mid u, C) \geq 0.5 \\ P(noun \mid u, C) & : \quad \text{otherwise} \end{cases} \\ &= 0.5 - abs(0.5 - P(noun \mid u, C)). \end{aligned} \tag{1}$$

In the general case of choosing between multiple classes, f_{err} and f_{unc} are different. We shall return to this point in Section 4.1.2 when we consider training parsers.

The potential drawback of the performance-based evaluation functions is that they assume that the hypothesis is correct. Selecting training examples based on a poor hy-

2 As long as it adheres to these criteria, the specific form of the function is irrelevant, since the selection is not determined by the absolute scores of the candidates, but by their scores relative to each other.

pothesis is prone to pitfalls. On the one hand, the hypothesis may be overly confident about the certainty of its decisions. For example, the hypothesis may assign *noun* to a candidate with a probability of 1 based on a single previous observation in which a similar example was labeled as *noun*. Despite the unreliable statistics, this candidate would not be selected since the hypothesis considers this a known case. Conversely, the hypothesis may also direct the selection algorithm to chase after undecidable cases. For example, consider PP's with *in* as the head. These PP's occur frequently and about half of those PP's should attach to the object noun. Even though training on more labeled *in* examples does not improve the model's performance on future *in*-PP's, the selection algorithm will keep on requesting more *in* training examples because the hypothesis remains uncertain about this preposition.[3] With an unlucky starting hypothesis, these evaluation functions may select uninformative candidates initially.

3.2.3 The Parameters of the Hypothesis The potential problems with the performance-based evaluation function stem from the evaluation functions' trust in the model's diagnosis of its own progress. Another way to incorporate the current hypothesis is to determine how good it is and what type of examples will improve it the most. In this section we propose an evaluation function that scores candidates based on their utilities in increasing the confidence about the parameters of the hypothesis (i.e., the collection of statistics over the characteristic tuples of the training examples).

Training the parameters of the PP-attachment model is similar to empirically determining the bias of a coin. We measure the coin's bias by repeatedly tossing it and keeping track of the percentage of times it landed on heads. The more trials we perform,

3 This phenomenon is particularly acute in the early stages of refining the hypothesis because most decisions are based on statistics of the head preposition alone; in the later stages, the hypothesis can usually rely on higher-ordered characteristic tuple that tend to be better classifiers.

the more confident we become about our estimation of the bias. Similarly, in estimating p, the likelihood of a PP to attach to its object noun, we are more confident about the classification decision based on statistics with higher counts. A quantitative measurement of our confidence in a statistic is the *confidence interval.* It is a region around the measured statistic, bounding the area where the true statistic might lie. More specifically, the confidence interval for p, a binomial parameter, is defined as:

$$conf\_int(\bar{p}, n) = \frac{1}{1+\frac{t^2}{n}}\left(\bar{p} + \frac{t^2}{2n} \pm t\sqrt{\frac{\bar{p}(1-\bar{p})}{n} + \frac{t^2}{4n^2}}\right),$$

where $\bar{p}$ is the expected value of p based on n trials, and t is a *threshold value* that depends on the number of trials and the level of confidence we desire. For instance, if we want to be 90% confident that the true statistic p lies within the interval, and $\bar{p}$ is based on $n = 30$ trials, then we set t to be 1.697.[4] Applying the confidence interval concept to evaluating candidates for the back-off PP-attachment model, we define a function f_{conf} that scores a candidate by taking the average of the lengths of the confidence interval of each back-off level. That is,

$$f_{conf}(u, C) = \frac{\sum_{l=1}^{4} |conf\_int(\bar{p}_l(u, C), n_l(u, C))|}{4},$$

where $\bar{p}_l(u, C)$ is the probability that model C would attach u to *noun* at back-off level l, and $n_l(u, C)$ is the number of training examples upon which this classification is based.

The confidence-based evaluation function has several potential problems. One of its flaws is similar to that of f_{novel}. In the early stage, f_{conf} picks the same examples as f_{novel} because we have no confidence in the statistics of novel examples. Therefore,

4 For $n \leq 120$, the values of t can be found in standard statistic textbooks; for $n \geq 120$, $t = 1.6576$. Because the derivation for the confidence interval equation makes a normality assumption, the equation does not hold for small values of n (cf pp. 277-278, (Larsen and Marx, 1986)). When n is large, the contributions from the terms in $\frac{t^2}{n}$ are negligible. Dropping these terms, we would have the t statistic, $\bar{p} \pm t\sqrt{\bar{p}(1-\bar{p})/n}$.

f_{conf} is also prone to chase after examples that rarely occur to build up confidence of some unimportant parameters. A second problem is that f_{conf} ignores the output of the model. Thus, if candidate A has a confidence interval around $[0.6, 1]$ and candidate B has a confidence interval around $[0.4, 0.7]$, then f_{conf} would prefer candidate A even though training on A will not change the hypothesis's performance since the entire confidence interval is already in the *noun* zone.

3.2.4 Hybrid Function The three categories of predictive criteria discussed above are complementary, each focusing on a different aspect of the learner's weakness. Therefore, it may be beneficial to combine these criteria into one evaluation function. For instance, the deficiency of the confidence-based evaluation function described in the previous section can be avoided if the confidence interval covering the region around the uncertainty boundary (Candidate B in the example) is weighed more heavily than one around the end points (Candidate A).

In this section, we introduce a new function that tries to factor in both the uncertainty of the model performance and the confidence of the model parameters. First, we define a quantity, called $area(\bar{p}, n)$, that computes the area under a Gaussian function $N(x, \mu, \sigma)$ with a mean of 0.5 and a standard deviation of 0.1 [5] that is bounded by the confidence interval as computed by $conf\_int(\bar{p}, n)$ (see Figure 5). That is, suppose $\bar{p}$ has a confidence interval of $[a, b]$, then

$$area(\bar{p}, n) = \int_a^b N(x, 0.5, 0.1)dx.$$

Computing *area* for each back-off level, we define an evaluation function, $f_{area}(u, C)$, as their average. This function can be viewed as a product of f_{conf} and f_{unc}. [6]

5 The standard deviation value for the Gaussian is chosen so that more than 98% of the mass of the distribution is between 0.25 and 0.75.

6 Note that we can replace the function in Equation (1) with the $N(x, 0.5, \sigma)$ without affecting f_{unc}

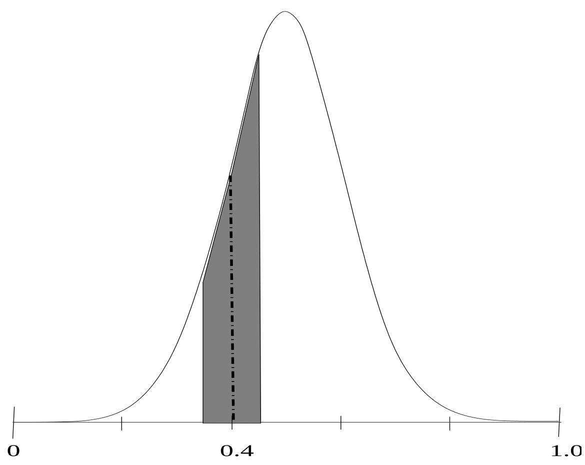


Figure 5

An example: suppose that the candidate has a likelihood of 0.4 for noun attachment and a confidence interval of width 0.1. Then *area* computes the area bounded by the confidence interval and the Gaussian curve.

3.3 Experimental Comparison

To determine the relative merits of the proposed evaluation functions, we compare the learning curve of training with sample selection according to each function against a baseline of random selection in an empirical study. The corpus is a collection of phrases extracted from the WSJ Treebank. We used Section 00 as the development set, Sections 2-23 as the training and test sets. We performed 10-fold cross validation to ensure the statistical significance of the results. For each fold, the training candidate pool contains about 21,000 phrases, and the test set contains about 2,000 phrases.

As shown in Figure 1, the learner generates an initial hypothesis based on a small set of training examples, L. These examples are randomly selected from the pool of unlabeled candidates and annotated by a human. Random sampling ensures that the initial trained set reflects the distribution of the candidate pool, and thus the initial hypothe-

---

because it is also symmetric about 0.5 and monotonically decreasing as the input value moves further.

sis is unbiased. Starting with an unbiased hypothesis is important for those evaluation functions whose scoring metrics are affected by the accuracy of the hypothesis. In these experiments, L initially contains 500 randomly selected examples.

In each selection iteration, all the candidates are scored by the evaluation function, and n examples with the highest TUV's are picked out from U to be labeled and added to L. Ideally, we would like to have $n = 1$ for each iteration. In practice, however, it is often more convenient for the human annotator to label data in larger batches rather than one at a time. In these experiments, we use a batch size of $n = 500$ examples.

We make note of one caveat to this kind of n-best batch selection. Under a hypothesis-dependent evaluation function, identical examples would receive identical scores. Because identical (or very similar) examples tend to address the same deficiency in the hypothesis, adding n very similar examples to the training set is unlikely to lead to big improvement in the hypothesis. To diversify the examples in each batch, we simulate single-example selection (whenever possible) by re-estimating the scores of the candidates after each selection. Suppose we have just chosen to add candidate x to the batch. Then, before selecting the next candidate, we *estimate* the potential decrease in scores of candidates similar to x once it belongs to the annotated training set. The estimation is based entirely on the knowledge that x is chosen, but not on the classification of x. Thus, only certain types of evaluation functions are amenable to the re-estimation process. For example, if scores were assigned by f_{conf}, then we know that the confidence intervals of the candidates similar to x must decrease slightly after learning x. On the other hand, if scores were assigned by f_{unc}, then we could not perceive any changes to the scores of similar candidates without knowing the true classification of x.

3.3.1 Results and Discussions This section presents the empirical measurements of the model's performances using training examples selected by different evaluation func-

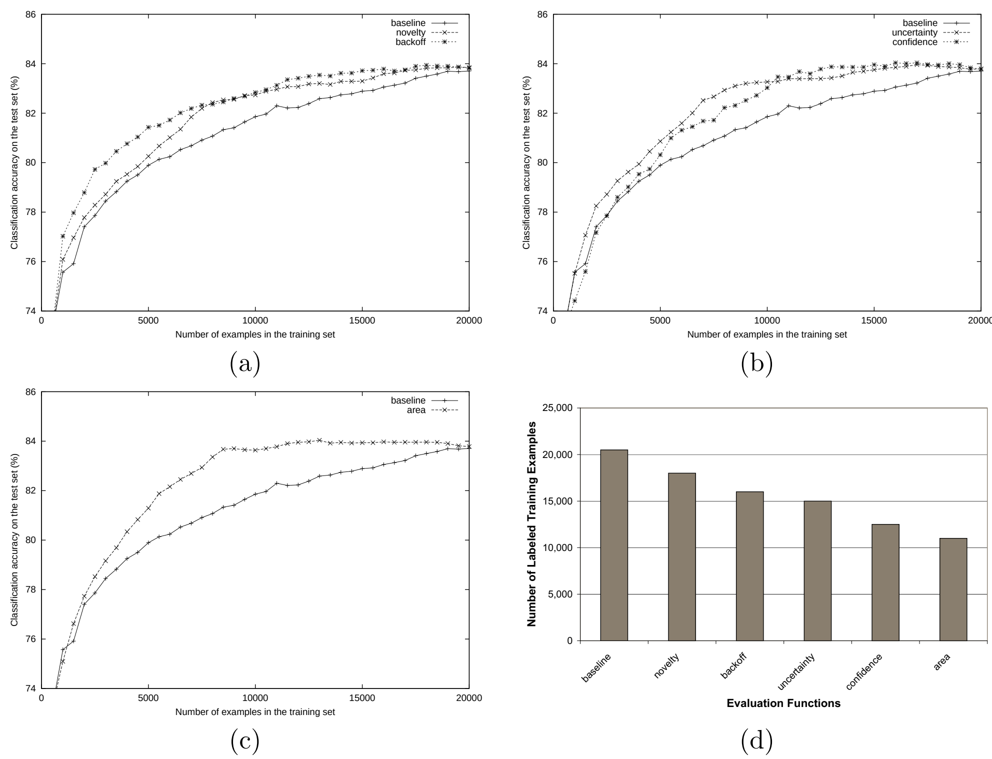


Figure 6

A comparison of the performance of different evaluation functions (a) compares the learning curves of the functions that uses knowledge about the problem space: f_{novel} and $f_{backoff}$ with that of the baseline; (b) compares the learning curves of performance-based functions: f_{unc} and f_{conf} with the baseline; (c) compares the learning curve of f_{area}, which combines uncertainty and confidence with f_{unc}, f_{conf}, and the baseline; (d) compares all the functions for the number of training examples selected at the final performance level (83.8%) compares $f_{backoff}$ to the baseline.

tions. We compare each proposed function with the baseline of random selection (f_{rand}). The results are graphically depicted from two perspectives. One (e.g., Figure 6(a)-(c)) plots the learning curves of the functions, showing the relationship between the number of training examples (x-axis) and the performance of the model on test data (y-axis). We deem one evaluation function to be better than another if its learning curve envelopes the other's. An alternative way to interpret the results is to focus on the reduction in training size offered by one evaluation function over another for some particular performance level. Figure 6(d) is a bar graph comparing all the evaluation functions at the highest performance level. The graph shows that in order to train a model that attaches PP's with an accuracy rate of 83.8%, sample selection with f_{novel} requires 2500 fewer examples than the baseline.

Compared to f_{novel}, $f_{backoff}$ selects more helpful training examples in the early stage. As shown in Figure 6(a), the improvement rate of the model under $f_{backoff}$ is always at least as fast as f_{novel}. However, the differences between these two functions become smaller for higher performance level. This outcome validates our predictions. Scoring candidates by a combination of their novelty, occurrence frequencies, and the qualities of their characteristic tuples, $f_{backoff}$ selects helpful early (the first 4,000 or so) training examples. Then, just as in f_{novel}, the learning rate remained stagnant for the next 2,000 poorly selected examples. Finally, when the remaining candidates all have similar novelty values and contain mostly characteristic tuples that occur infrequently, the selection became random.

Figure 6(b) compares the two evaluation functions that score candidates based on the current state of the hypothesis. Although both functions suffer a slow start, they are more effective than $f_{backoff}$ at reducing the training set when learning high quality models. Initially, because all the unknown statistics are initialized to 0.5, selection based f_{unc} is essentially random sampling. Only after the hypothesis became sufficiently accurate (after training on about 5,000 annotated examples) did it begin to make informed selections. Following a similar but more exaggerated pattern, the confidence-based function, f_{conf}, also improved slowly at the beginning before finally overtaking the baseline. As we have noted earlier, because the hypothesis is not confident about novel candidates, f_{conf} and f_{novel} tend to select the same early examples. Therefore, the early learning rate of f_{conf} is as bad as that of f_{novel}. In the later stage, while f_{novel} continues to flounder, f_{conf}

can select better candidates based on a more reliable hypothesis.

Finally, the best performing evaluation function is the hybrid approach. Figure 6(c) shows that the learning curve of f_{area} combines the earlier success of f_{unc} and the later success of f_{conf} to always outperform the other functions. As shown in Figure 6(d), it requires the least number of examples to achieve the highest performance level of 83.8%. Compared to the baseline, f_{area} requires 47% fewer examples. From these comparison studies, we conclude that involving the hypothesis in the selection process is a key factor in reducing the size of training set.

4 Sample Selecting for Statistical Parsing

In applying sample selection to training a PP-attachment model, we have observed that all effective evaluation functions made use of the model's current hypothesis in estimating the training utility of the candidates. Although knowledge about the problem space seemed to help sharpening the learning curve initially, overall, it was not a good predictor. In this section, we investigate whether these observations hold true for training statistical parsing models as well. Moreover, in order to determine whether the performances of the predictive criteria are consistent across different learning models within the same domain, we have performed the study on two parsing models: one based on a context-free variant of Tree Adjoining Grammars (Joshi, Levy, and Takahashi, 1975), the Probabilistic Lexicalized Tree Insertion Grammar (PLTIG) formalism (Schabes and Waters, 1993; Hwa, 1998), and Collins's Model 2 Parser (1997). Although both models are lexicalized, statistical parsers, their learning algorithms are different. The Collins Parser is a fully-supervised, history-based learner that models the parameters of the parser by taking statistics directly from the training data. In contrast, PLTIG's EM-based induction algorithm is partially-supervised; the model's parameters are estimated indirectly from the training data.

As a superset of the PP-attachment task, parsing is a more challenging learning problem. Whereas a trained PP-attachment model is a binary classifier, a parser must identify the correct syntactic analysis out of all possible parses for a sentence. This classification task is more difficult since the number of possible parses for a sentence grows exponentially with respect to its length. Consequently, the annotator's task is more complex. Whereas the person labeling the training data for PP-attachment reveals one unit of information (always choosing between *noun* or *verb*), the annotation needed for parser training is usually greater than one unit[7] and the type of labels varies from sentence to sentence. Because the annotation complexity differs from sentence to sentence, the evaluation functions must strike a balance between maximizing the potential informational gain and minimizing the expected amount of annotation exerted. We propose a set of evaluation functions similar in spirit to those for the PP-attachment learner, but extended to accommodate the parsing domain.

4.1 Evaluation Functions

4.1.1 Problem Space Similar to scoring a PP candidate based on the novelty and frequencies of its characteristic tuples, we define an evaluation function, $f_{lex}(\mathbf{w}, G)$ that scores a sentence candidate, $\mathbf{w}$, based on the novelty and frequencies of word pair co-occurrences:

$$f_{lex}(\mathbf{w}, G) = \frac{\sum_{w_i, w_j \in \mathbf{w}} new(w_i, w_j) \times coocc(w_i, w_j)}{length(\mathbf{w})},$$

7 We consider each pair of brackets in the training sentence to be one unit of supervised information, assuming that the number of brackets correlates linearly with the amount of effort spent by the human annotator. This correlation is an approximation, however; in real life, adding one pair of brackets to a longer sentence may require more effort than adding to a shorter one. To capture bracketing interdependencies at this level, we would need to develop a model of the annotation decision process and incorporate it as an additional factor in the evaluation functions.

where $\mathbf{w}$ is the unlabeled sentence candidate; G is the current parsing model (which is ignored by problem-space-based evaluation functions); $new(w_i, w_j)$ is an indicator function that returns 1 if we have not yet selected any sentence in which w_i and w_j co-occurred; and $coocc(w_i, w_j)$ is a function that returns the number of times that w_i co-occurs[8] with w_j in the candidate pool. We expect these evaluation functions to be less relevant for the parsing domain than the PP-attachment domain for two reasons. First, because we do not have the actual parses, the extraction of lexical relationships are based on co-occurrence statistics, not syntactic relationships. Second, because the distribution of words that form lexical relationships is wider and more uniform than words that form PP characteristic tuples, most word pairs will be novel and appear only once.

Another simple evaluation function based on the problem space is one that estimates the TUV of a candidate by its sentence length:

$$f_{len}(\mathbf{w}, G) = length(\mathbf{w})$$

The intuition behind this function is based on the general observation that longer sentences tend to have complex structures and introduce more opportunities for ambiguous parses. Although these evaluation functions may seem simplistic, they have one major advantage: they are easy to compute and require little processing time. Because inducing parsing models demands significantly more time than inducing PP-attachment models, it becomes more important that the evaluation functions be as efficient as possible.

4.1.2 The Performance of the Hypothesis We previously defined two performance-based evaluation functions: f_{err}, the model's estimate of how likely it made a classification error, and f_{unc}, the model's estimate of its uncertainty in making the classification. We have shown the two functions to have similar performance for the PP-attachment task.

8 We consider two words as co-occuring if their log-likelihood ratio is greater than some threshold value determined with heldout data.

This is not the case for statistical parsing because the number of possible classes (parse trees) differ from sentence to sentence. For example, suppose we wish to compare one candidate for which the current parsing model generated four equally likely parses with another candidate for which the model generated one parse with probability of 0.2 and ninety-nine other parses with a probability of 0.01 (such that they sum up to 0.98). The error-driven function, f_{err}, would score the latter candidate higher because its most likely parse has a lower probability; the uncertainty-based function, f_{unc}, would score the former candidate higher because the model does not have a strong preference for one parse over any other. In this section, we provide a formal definition for both functions.

Suppose that a parsing model G generates a candidate sentence $\mathbf{w}$ with probability $P(\mathbf{w} \mid G)$, and that the set $\mathcal{V}$ contains all possible parses that G generated for $\mathbf{w}$. Then, we denote the probability of G generating a single parse, $v \in \mathcal{V}$ as $P(v \mid G)$ such that $\sum\limits_{v \in \mathcal{V}} P(v \mid G) = P(\mathbf{w} \mid G)$. The parse chosen for $\mathbf{w}$ is the most likely parse in $\mathcal{V}$, denoted as v_{max}, where

$$v_{max} = ArgMax_{v \in \mathcal{V}} P(v \mid G).$$

Note that $P(v \mid G)$ reflects the probability of one particular parse tree, v, out of all possible parse trees for all possible sentences that G can generate. To compute the likelihood of a parse being the correct parse out of the possible parses of $\mathbf{w}$ according to G, denoted as $P(v \mid \mathbf{w}, G)$, we need to normalize the tree probability by the sentence probability. So according to G, the likelihood that v_{max} is the correct parse for $\mathbf{w}$ is[9]:

$$\begin{aligned} P(v_{max} \mid \mathbf{w}, G) &= \frac{P(v_{max} \mid G)}{P(\mathbf{w} \mid G)} \\ &= \frac{P(v_{max} \mid G)}{\sum_{v \in \mathcal{V}} P(v \mid G)}. \end{aligned} \tag{2}$$

9 Note that $P(\mathbf{w}|v, G) = 1$ for any $v \in \mathcal{V}$, where $\mathcal{V}$ is the set of all possible parses for $\mathbf{w}$, because v only exists when $\mathbf{w}$ is observed.

Therefore, the error-driven evaluation function is defined as:

$$f_{err}(\mathbf{w}, G) = 1 - P(v_{max} \mid \mathbf{w}, G).$$

Unlike the error-driven function, which focuses on the most likely parse, the uncertainty-based function takes the probability distribution of all parses into account. To quantitatively characterize its distribution, we compute the entropy of the distribution. That is,

$$H(V) = -\sum_{v \in \mathcal{V}} p(v) \lg(p(v)), \quad (3)$$

where V is a random variable that can take any possible outcome in set $\mathcal{V}$, and $p(v) = Pr(V = v)$ is the density function. Further details about the properties of entropy can be found in textbooks on information theory (Cover and Thomas, 1991).

Determining the parse tree for a sentence from a set of possible parses can be viewed as assigning a value to a random variable. Thus, a direct application of the entropy definition to the probability distribution of the parses for sentence $\mathbf{w}$ in G computes its *tree entropy*, $TE(\mathbf{w}, G)$, the expected number of bits needed to encode the distribution of possible parses for $\mathbf{w}$. However, we may not wish to compare sentences with different numbers of parses by their entropy directly. If the parse probability distributions for both sentences are uniform, the sentence with more parses would have a higher entropy. Because longer sentences typically have more parses, using entropy directly would place bias towards selecting long sentences. To normalize for the number of parses, the uncertainty-based evaluation function, f_{unc}, is defined as a measurement of similarity between the actual probability distribution of the parses and a hypothetical uniform distribution for that set of parses. In particular, we divide the tree entropy by the log of the number of parses:[10]

$$f_{unc}(\mathbf{w}, G) = \frac{TE(\mathbf{w}, G)}{\lg(\|\mathcal{V}\|)}.$$

We now derive the expression for $TE(\mathbf{w}, G)$. Recall from Equation (2) that if G produces a set of parses, $\mathcal{V}$, for sentence $\mathbf{w}$, the set of probabilities $P(v \mid \mathbf{w}, G)$ (for all $v \in \mathcal{V}$) defines the distribution of parsing likelihoods for sentence $\mathbf{w}$:

$$\sum_{v \in \mathcal{V}} P(v \mid \mathbf{w}, G) = 1.$$

Note that $P(v \mid \mathbf{w}, G)$ can be viewed as a density function $p(v)$ (i.e., the probability of assigning v to a random variable V). Mapping it back into the entropy definition from Equation (3), we derive the tree entropy of $\mathbf{w}$ as follows:

$$\begin{aligned}
TE(\mathbf{w}, G) &= H(V) \\
&= -\sum_{v \in \mathcal{V}} p(v) \lg(p(v)) \\
&= -\sum_{v \in \mathcal{V}} \frac{P(v \mid G)}{P(\mathbf{w} \mid G)} \lg(\frac{P(v \mid G)}{P(\mathbf{w} \mid G)}) \\
&= -\sum_{v \in \mathcal{V}} \frac{P(v \mid G)}{P(\mathbf{w} \mid G)} \lg(P(v \mid G)) + \sum_{v \in \mathcal{V}} \frac{P(v \mid G)}{P(\mathbf{w} \mid G)} \lg(P(\mathbf{w} \mid G)) \\
&= -\frac{1}{P(\mathbf{w} \mid G)} \sum_{v \in \mathcal{V}} P(v \mid G) \lg(P(v \mid G)) + \frac{\lg(P(\mathbf{w} \mid G))}{P(\mathbf{w} \mid G)} \sum_{v \in \mathcal{V}} P(v \mid G) \\
&= -\frac{1}{P(\mathbf{w} \mid G)} \sum_{v \in \mathcal{V}} P(v \mid G) \lg(P(v \mid G)) + \lg(P(\mathbf{w} \mid G))
\end{aligned}$$

Using the bottom-up, dynamic programming technique (see Appendix A for details) of computing Inside Probabilities (Lari and Young, 1990), we can efficiently compute the probability of the sentence, $P(\mathbf{w} \mid G)$. Similarly, the algorithm can be modified to

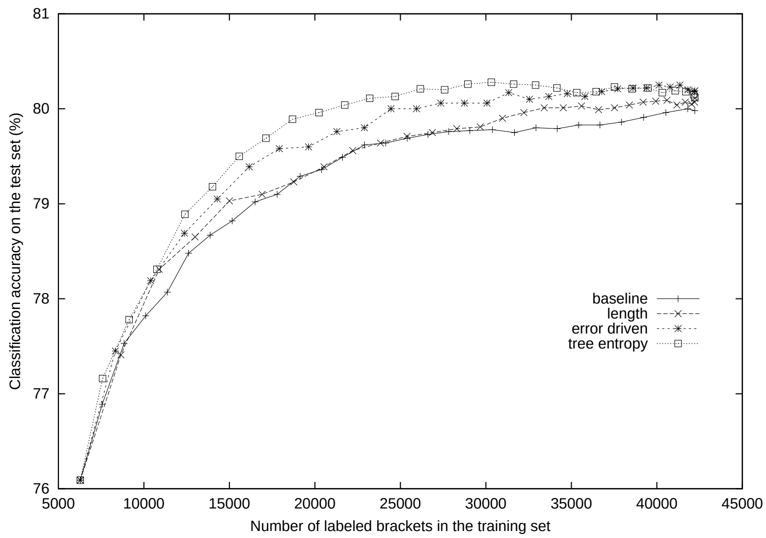
10 When $f_{unc}(\mathbf{w}, G) = 1$, the parser is considered to be the most uncertain about that sentence. Instead of dividing tree entropies, one could have computed the Kullback Leibler distance between the two distributions (in which case a score of zero would indicate the highest level of uncertainty). Because the selection is based on relative scores, as long as the function is monotonic, the exact form of the function should not have much impact on the outcome.

compute the quantity $\sum_{v \in \mathcal{V}} P(v \mid G) \lg(P(v \mid G))$.

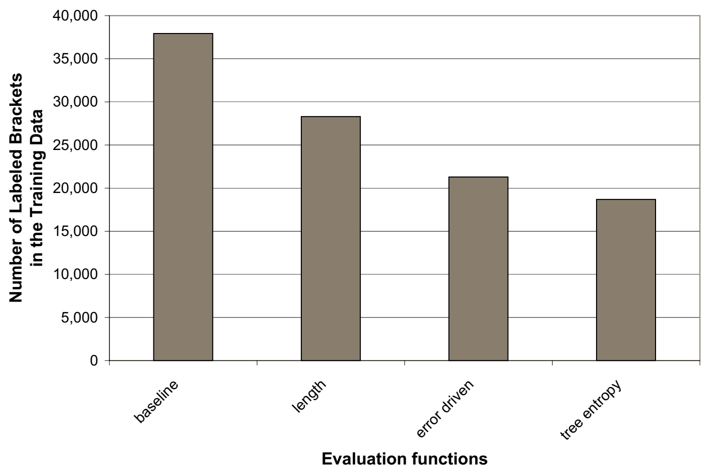
4.1.3 The Parameters of the Hypothesis Although the confidence-based function gave good TUV estimates to candidates for training PP-attachment model, it is not clear how a similar technique can be applied to training parsers. Whereas binary classification tasks can be described by binomial distributions, for which the confidence interval is well defined; a parsing model is made up of many multinomial classification decisions. We therefore need a way to characterize confidence for each decision as well as a way to combine them into an overall confidence. Another difficulty is that the complexity of the induction algorithm deters us from re-estimating the TUVs of the remaining candidates after selecting each new candidate. As we have discussed in section 3.3, re-estimation is important for batched annotation. Without some means of updating the TUVs after each selection, the learner would not realize that it had already selected a candidate to train some parameter with low confidence until the re-training phase that only occurs at the end of the batch selection; therefore, it may continue to select very similar candidates to train the same parameter. Even if we assume that the statistics can be updated, re-estimating the TUVs is a computationally expensive operation. Essentially, all the remaining candidates that share some parameters with the selected candidate would need to be re-parsed. For these practical reasons, we do not include an evaluation function measuring confidence for the parsing experiment.

4.2 Experiments and Results

We compare the effectiveness of sample selection using the proposed evaluation functions against a baseline of random selection ($f_{rand}(\mathbf{w}, G) = rand()$). Similar to previous experimental designs, the learner is given a small set of annotated seed data from the WSJ Treebank and a large set of unlabeled data (also from the WSJ Treebank but with the labels removed) to select new training examples from. All training data are from



(a)



(b)

Figure 7

(a) A comparison of the evaluation functions' learning curves. (b) A comparison of the evaluation functions for a test performance score of 80%

Sections 2-21. We monitor the learning progress of the parser by testing it on unseen test sentences. We used Section 00 for development and Section 23 for test. This study is repeated for two different models, the PLTIG parser and Collins's Model 2 parser.

4.2.1 An EM-based learner In the first experiment, we use an induction algorithm (Hwa, 2001a) based on the Expectation-Maximization (EM) principle that induces parsers for Probabilistic Lexicalized Tree Insertion Grammars (PLTIG). The algorithm performs heuristic search through an iterative re-estimation procedure to find the set of values for the grammar parameters that maximizes the grammar's likelihood of generating the training data. In principle, the algorithm supports unsupervised learning; however, because the search space has too many local optima, the algorithm tends to converge on a model that is unsuitable for parsing. Here, we consider a partially supervised variant in which we assume that the learner is given the phrasal boundaries of the training sentences but not the label of the constituent units. For example, the sentence "*Several fund managers expect a rough market this morning before prices stabilize.*" would be labeled as "((Several fund managers) (expect ((a rough market) (this morning)) (before (prices stabilize))).)" Our algorithm is similar to the approach taken by Pereira and Schabes (1992) for inducing PCFG parsers.

Because the EM algorithm itself is an iterative procedure, performing sample selection on top of an EM-based learner is an extremely computational intensive process. Here, we restrict the experiments for the PLTIG parsers to a smaller-scale study in the following two aspects. First, the lexical anchors of the grammar rules are backed off to part-of-speech tags; this restricts the size of the grammar vocabulary to be 48. Second, the unlabeled candidate pool is set to contain 3600 sentences, which is sufficiently large for inducing a grammar of this size. The initial model is trained on 500 labeled seed sentences. For each selection iteration, an additional 100 sentences are moved from the unlabeled pool to be labeled and added to the training set. After training, the updated parser is then tested on unseen sentences (backed off to their part-of-speech tags) and compared to the gold standard. Because the induced PLTIG produces binary branching parse trees, which have more layers than the gold standard, we measure parsing accuracy in terms of the crossing bracket metric. The study is repeated for ten trials, each using a different portions of the full training set, to ensure statistical significance (using pair-wise t-test at 95% confidence).

The results of the experiment are graphically shown in Figure 7. As with the PP-attachment studies, the graph on the left compares the learning curves between the proposed evaluation functions and the baseline. Note that even though these functions select examples in terms of entire sentences, the amount of annotation is measured in terms of the number of *brackets* rather than sentences in the graphs (x-axis). Unlike the PP-attachment case, the amount of effort from the annotators varies significantly from example to example. A short and simple sentence takes much less time to annotate than a long and complex sentence. We address this effect by approximating the amount of effort with the number of brackets the annotator needs to label. Thus, we deem one evaluation function more effective than another if, for the desired level of performance, the smallest set of sentences selected by the function contains fewer brackets than that of the other function. The graph on the right compares the evaluation functions at the final test performance level of 80%.

Qualitatively comparing the learning curves in the graphs, we see that with the appropriate evaluation function sample selection does reduce the amount of annotation. Similar to our findings in the PP-attachment study, the simple problem-space-based evaluation function, f_{len} offers only little savings; it is nearly indistinguishable from the baseline for the most part.[11] The evaluation functions based on hypothesis performances, on the other hand, do reduce the amount of annotation in the training data. Of the two that we proposed for this category, the tree entropy evaluation function, f_{unc}, has a slight edge over the error-driven evaluation function, f_{err}.

11 In this experiment, we have omitted the evaluation function for selecting novel lexical relationships, f_{lex}, because the grammar does not use actual lexical anchors.

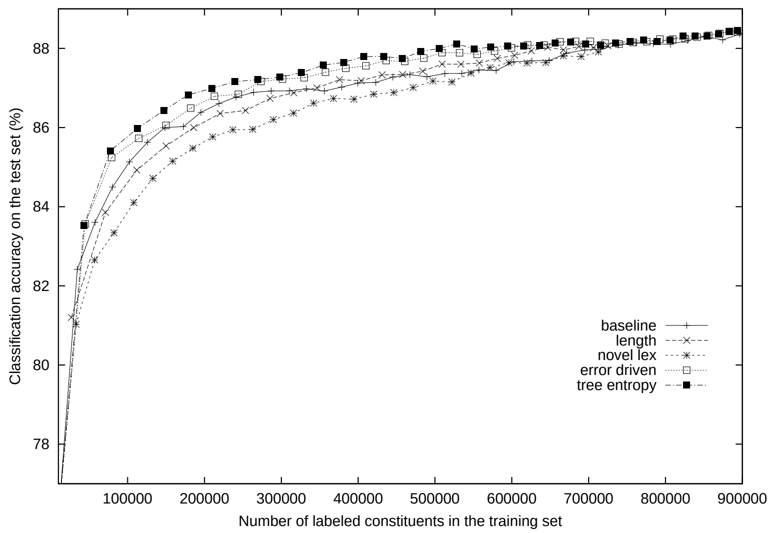
For a quantitative comparison, let us consider the set of grammars that achieve an average parsing accuracy of 80% on the test sentences. We consider a grammar to be comparable to that of the baseline if its mean test score is at least as high as that of the baseline and if the difference of the means is not statistically significant. The baseline case requires an average of about 38 thousand brackets in the training data. In contrast, to induce a grammar that reaches the same 80% parsing accuracy with the examples selected by f_{unc}, the learner requires, on average, 19 thousand training brackets. Although the learning rate of f_{err} is slower than that of f_{unc} overall, it seems to have caught up in the end; it needs 21 thousand training brackets, slightly more than f_{unc}. While the simplistic sentence length evaluation function, f_{len}, is less helpful, its learning rate still improves slightly faster than the baseline. A grammar of comparable quality can be induced from a set of training examples selected by f_{len} containing an average of 28 thousand brackets.[12]

4.2.2 A history-based learner In the second experiment, the basic learning model is Collins's Model 2 parser (1997), which uses a history-based learning algorithm that takes statistics directly over the treebank. As a fully-supervised algorithm, it does not have to iteratively re-estimate its parameters, and is computationally efficient enough for us to carry out a large-scale experiment. For this set of studies, the unlabeled candidate pool consists of around 39,000 sentences. The initial model is trained on 500 labeled seed sentences, and at each selection iteration, an additional 100 sentences are moved from the unlabeled pool into the training set. The parsing performance on the test sentences is measured in terms of the parser's f-score, the harmonic average of the labeled precision and labeled recall rates over the constituents (Van Rijsbergen, 1979).[13]

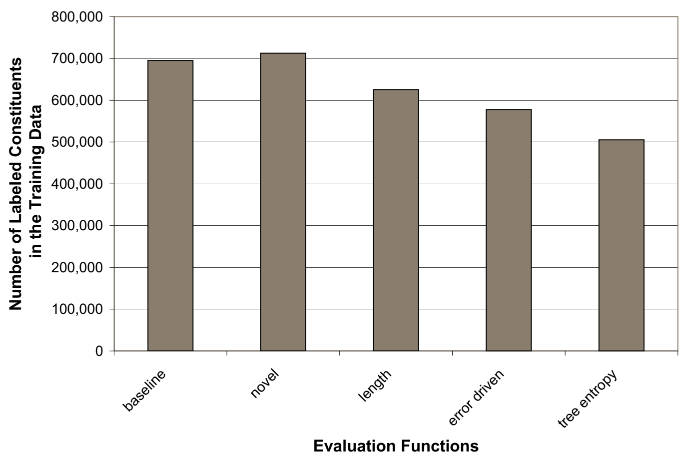
We plot the comparisons between different evaluation functions and the baseline for

12 In terms of the number of sentences, the baseline f_{rand} selected 2600 sentences; f_{len} selected 1300 sentences; and f_{err} and f_{unc} each selected 900 sentences.

13 $F = \frac{2 \times LR \times LP}{LR + LP}$, where LR is the labeled recall score and LP is the labeled precision score.



(a)



(b)

Figure 8

(a) A comparison of the learning curves of the evaluation functions. (b) A comparison of all the evaluation functions at the test performance level of 88%

the history-based parser in Figure 8. The examples selected by the problem-space-based functions do not seem to be helpful. Their learning curves are, for the most part, slightly worse than the baseline. In contrast, the parsers trained on data selected by the error-driven and uncertainty-based functions learned faster than the baseline; and as before, f_{unc} is slightly better than f_{err}.

For the final parsing performance of 88%, the parser requires a baseline training set of 30,500 sentences annotated with about 695,000 constituents. The same performance can be achieved with a training set of 20,500 sentences selected by f_{err}, which contains about 577,000 annotated constituents; or with a training set of 17,500 sentences selected by f_{unc}, which contains about 505,000 annotated constituents, reducing the number of annotated constituents by 27%. Comparing the outcome of this experiment with that of the EM-based learner, we see that the training data reduction rates are less dramatic than before. This may be because both f_{unc} and f_{err} ignore lexical items and chase after sentences containing words that rarely occur. Recent work by Tang, Luo, and Roukos (2002) suggests that a hybrid approach that combine features of the problem-space and the uncertainty of the parser may result in better performance for lexicalized parsers.

5 Related Work

Sample selection benefits problems in which the cost of acquiring raw data is cheap but the cost of annotating them is high. This is certainly the case for many supervised learning tasks in natural language processing. In addition to PP-attachment, as discussed in this paper, sample selection has been successfully applied to other classification applications. Some examples include text categorization (Lewis and Catlett, 1994), base noun phrase chunking (Ngai and Yarowsky, 2000), part-of-speech tagging (Engelson and Dagan, 1996), spelling confusion set disambiguation (Banko and Brill, 2001), and word-sense disambiguation (Fujii et al., 1998).

More challenging are learning problems whose objective is not classification, but generation of complex structures. One example in this direction is applying sample selection to semantic parsing (Thompson, Califf, and Mooney, 1999), in which sentences are paired with their semantic representation using a deterministic shift-reduce parser. A recent effort that focuses on statistical syntactic parsing is the work by Tang, Luo, and Roukos

(2002). Their results suggest that further reduction in training examples can be achieved by using a hybrid evaluation function that combines a hypothesis-performance-based metric such as tree-entropy ("word entropy" in their terminology) with a problem-space-based metric such as sentence clusters.

Aside from active learning, researchers have applied other learning techniques to combat the annotation bottleneck problem in parsing. For example, Henderson and Brill (2000) considered the case in which acquiring additional human annotated training data is not possible. They have shown that parser performance can be improved by using boosting and bagging techniques with multiple parsers. This approach assumes that there are enough existing labeled data to train the individual parsers. Another technique for making better use of unlabeled data is co-training (Blum and Mitchell, 1998), in which two sufficiently different learners helps each other learn by labeling training data for each other. The work of Sarkar (2001) and Steedman et al. (2003b) suggests that co-training can be helpful for statistical parsing. Pierce and Cardie (2001) have shown, in the context of base noun identification, that combining sample selection and co-training can be an effective learning framework for large-scale training. Similar approaches are being explored for parsing (Steedman et al., 2003a; Hwa et al., 2003).

6 Conclusion

In this paper, we have argued that sample selection is a powerful learning technique for reducing the amount of human labeled training data. Our empirical studies suggest that sample selection is helpful not only for binary classification tasks such as PP-attachment but also for applications that generate complex outputs such as syntactic parsing.

We have proposed several criteria for predicting the training utility of the unlabeled candidates and developed evaluation functions to rank them. We have conducted experiments to compare the functions' ability in selecting the most helpful training examples.

We have found that the uncertainty criterion is a good predictor that consistently finds helpful examples. In our experiments, evaluation functions that factor in the uncertainty criterion consistently outperform the baseline of random selection across different tasks and learning algorithms. For learning a PP-attachment model, the most helpful evaluation function is a hybrid that factors in the prediction performance of the hypothesis and the confidence on the values of the parameters of the hypothesis. For training a parser, we found that uncertainty-based evaluation functions that use tree entropy were the most helpful for both the EM-based learner and the history-based learner.

The current work points us in several future directions. First, we shall continue to develop alternative formulations of evaluation functions to improve the learning rates of parsers. Under the current framework, we did not experiment with any hypothesis-parameter-based evaluation functions for the parser induction task; however, hypothesis-parameter-based functions may be feasible under a multi-learner setting, using parallel machines. Second, while in this work we focused on selecting entire sentences as training examples, we believe that further reduction in the amount of annotated training data might be possible if the system could ask the annotators more specific questions. For example, if the learner were only unsure of a local decision within a sentence (such as a PP-attachment ambiguity), the annotator should not have to label the entire sentence. In order to allow for finer-grained interactions between the system and the annotators, we have to address some new challenges. To begin with, we must weigh in other factors in addition to the amount of annotations. For instance, the learner may ask about multiple substrings in one sentence. Even if the total number of labels were fewer, the same sentence would still need to be mentally processed by the annotators multiple times. This situation is particularly problematic when there are very few annotators as it becomes much more likely that a person would encounter the same sentence many times. Moreover, we must ensure that the questions asked by the learner are well-formed. If the learner were simply to present the annotator with some substring that it could not process, the substring might not form a proper linguistic constituent for the annotator to label. Third, we are interested in exploring the interaction between sample selection and other semi-supervised approaches such as boosting, re-ranking and co-training. Finally, based on our experience with parsing, we believe that active learning techniques may be applicable to other tasks that produces complex outputs such as machine translation.

Acknowledgments

We thank Joshua Goodman, Lillian Lee, Wheeler Ruml, and Stuart Shieber for helpful discussions; and Ric Crabbe, Philip Resnik, and the reviewers for their constructive comments on this paper. Portions of this work have appeared previously (Hwa, 2000; Hwa, 2001b); we thank the reviewers of those papers for their helpful comments. Parts of this work was carried out while the author was a graduate student at Harvard University, supported by the National Science Foundation under Grant No. IRI 9712068. The work is also supported by the Department of Defense contract RD-02-5700, and ONR MURI Contract FCPO.810548265.

References

Banko, Michele and Eric Brill. 2001. Scaling to very very large corpora for natural language disambiguation. In *Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics*, pages 26–33, Toulouse, France.

Blum, Avrim and Tom Mitchell. 1998. Combining labeled and unlabeled data with co-training. In *Proceedings of the 1998 Conference on Computational Learning Theory*, pages 92–100, Madison, WI.

Brill, Eric and Philip S. Resnik. 1994. A rule based approach to PP attachment disambiguation. In *Proceedings of the 15th International Conference on Computational Linguistics (COLING)*, pages 1198–1204, Kyoto, Japan.

Charniak, Eugene. 2000. A maximum-entropy-inspired parser. In *Proceedings of the First*

Meeting of the North American Association for Computational Linguistics, Seattle, Washington.

Cohn, David, Les Atlas, and Richard Ladner. 1994. Improving generalization with active learning. *Machine Learning*, 15(2):201–221.

Collins, Michael. 1997. Three generative, lexicalised models for statistical parsing. In *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics*, pages 16–23, Madrid, Spain.

Collins, Michael. 1999. *Head-Driven Statistical Models for Natural Language Parsing*. Ph.D. thesis, University of Pennsylvania.

Collins, Michael and James Brooks. 1995. Prepositional phrase attachment through a backed-off model. In *Proceedings of the Third Workshop on Very Large Corpora*, pages 27–38, Cambridge, MA.

Cover, Thomas M. and Joy A. Thomas. 1991. *Elements of Information Theory*. John Wiley.

Engelson, Sean P. and Ido Dagan. 1996. Minimizing manual annotation cost in supervised training from copora. In *Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics*, pages 319–326.

Freund, Yoav, H. Sebastian Seung, Eli Shamir, and Naftali Tishby. 1997. Selective sampling using the query by committee algorithm. *Machine Learning*, 28(2-3):133–168.

Fujii, Atsushi, Kentaro Inui, Takenobu Tokunaga, and Hozumi Tanaka. 1998. Selective sampling for example-based word sense disambiguation. *Computational Linguistics*, 24(4):573–598, December.

Henderson, John C. and Eric Brill. 2000. Bagging and boosting a treebank parser. In *Proceedings of the First Meeting of the North American Association for Computational Linguistics*, pages 34–41, Seattle, Washington.

Hwa, Rebecca. 1998. An empirical evaluation of probabilistic lexicalized tree insertion grammars. In *Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics*, volume 1, pages 557–563, Montreal, Canada.

Hwa, Rebecca. 2000. Sample selection for statistical grammar induction. In *Proceedings of*

2000 Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora, pages 45–52, Hong Kong, China, October.

Hwa, Rebecca. 2001a. *Learning Probabilistic Lexicalized Grammars for Natural Language Processing*. Ph.D. thesis, Harvard University.

Hwa, Rebecca. 2001b. On minimizing training corpus for parser acquisition. In *Proceedings of the ACL 2001 Workshop on Computational Natural Language Learning (ConLL-2001)*, pages 84–89.

Hwa, Rebecca, Miles Osborne, Anoop Sarkar, and Mark Steedman. 2003. Corrected co-training for statistical parsers. In *Proceedings of the ICML Workshop on the Continuum from Labeled to Unlabeled Data in Machine Learning and Data Mining at the 20th International Conference of Machine Learning (ICML-2003)*, pages 95–102, Washington D.C., August.

Joshi, Aravind K., Leon S. Levy, and Masako Takahashi. 1975. Tree adjunction grammars. *Journal of Computer and System Sciences*, 10(1):136–163.

Lari, Karim and Steve J. Young. 1990. The estimation of stochastic context-free grammars using the inside-outside algorithm. *Computer Speech and Language*, 4:35–56.

Larsen, Richard J. and Morris L. Marx. 1986. *An Introduction to Mathematical Statistics and Its Applications*. Prentice-Hall, New Jersey.

Lewis, David D. and Jason Catlett. 1994. Heterogeneous uncertainty sampling for supervised learning. In *Proceedings of the Eleventh International Conference on Machine Learning*, pages 148–156, San Francisco, CA.

Magerman, David. 1994. *Natural Language Parsing as Statistical Pattern Recognition*. Ph.D. thesis, Stanford University University, February.

Marcus, Mitchell, Beatrice Santorini, and Mary Ann Marcinkiewicz. 1993. Building a large annotated corpus of English: the Penn Treebank. *Computational Linguistics*, 19(2):313–330.

Ngai, Grace and David Yarowsky. 2000. Rule writing or annotation: Cost-efficient resource usage for base noun phrase chunking. In *Proceedings of the 38th Annual Meeting of the Association for Computational Linguistics*, pages 117–125, Hong Kong, China, October.

Pereira, Fernando C. N. and Yves Schabes. 1992. Inside-Outside reestimation from partially bracketed corpora. In *Proceedings of the 30th Annual Meeting of the Association for Computational Linguistics*, pages 128–135, Newark, Delaware.

Pierce, David and Claire Cardie. 2001. Limitations of co-training for natural language learning from large datasets. In *Proceedings of the 2001 Conference on Empirical Methods in Natural Language Processing (EMNLP-2001)*, pages 1–9, Pittsburgh, PA.

Ratnaparkhi, Adwait. 1998. Statistical models for unsupervised prepositional phrase attachment. In *Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics*, volume 2, pages 1079–1085, Montreal, Canada.

Sarkar, Anoop. 2001. Applying co-training methods to statistical parsing. In *Proceedings of the Second Meeting of the North American Association for Computational Linguistics*, pages 175–182, Pittsburgh, PA, June.

Schabes, Yves and Richard Waters. 1993. Stochastic lexicalized context-free grammar. In *Proceedings of the Third International Workshop on Parsing Technologies*, pages 257–266.

Steedman, Mark, Rebecca Hwa, Stephen Clark, Miles Osborne, Anoop Sarkar, Julia Hockenmaier, Paul Ruhlen, Steven Baker, and Jeremiah Crim. 2003a. Example selection for bootstrapping statistical parsers. In *The Proceedings of the Joint Conference of Human Language Technologies and the Annual Meeting of the North American Chapter of the Association for Computational Linguistics*, pages 236–243, Edmonton, Canada.

Steedman, Mark, Miles Osborne, Anoop Sarkar, Stephen Clark, Rebecca Hwa, Julia Hockenmaier, Paul Ruhlen, Steven Baker, and Jeremiah Crim. 2003b. Bootstrapping statistical parsers from small datasets. In *The Proceedings of the Tenth Conference of the European Chapter of the Association for Computational Linguistics*, pages 331–338, Budapest, Hungary.

Tang, Min, Xiaoqiang Luo, and Salim Roukos. 2002. Active learning for statistical natural language parsing. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 120–127, Philadelphia, July.

Thompson, Cynthia A., Mary Elaine Califf, and Raymond J. Mooney. 1999. Active learning for natural language parsing and information extraction. In *Proceedings of the Sixteenth International Conference on Machine Learning (ICML-1999)*, pages 406–414, Bled, Slovenia.

Van Rijsbergen, Cornelis J. 1979. *Information Retrieval.* Butterworth.

A Efficient Computation of Tree Entropy

As discussed in Section 4.1.2, for learning tasks such as parsing, the number of possible classification would be so large that it may not be computationally efficient to calculate the degree of uncertainty using the tree entropy definition. In the equation, the computation requires summing over all possible parses, but the number of possible parses for a sentence grows exponentially with respect to the sentence length. In this section, we show that tree entropy can be efficiently computed using dynamic programming.

For illustrative purposes, we describe the computation process using a PCFG expressed in Chomsky Normal Form.[14] The basic idea is to compose the tree entropy of the entire sentence from the tree entropy of the subtrees. The process is similar to computing the probability of the entire sentence from the probabilities of substrings (called *Inside Probability*). We follow the notation convention of Lari and Young (1990).

The Inside Probability of a nonterminal X generating the substring $w_i \dots w_j$ is denoted as $e(X,i,j)$; it is the sum of the probabilities of all possible subtrees that have X as the root and $w_i \dots w_j$ as the leaf nodes. We define a new function $h(X,i,j)$ to represent the corresponding entropy for the substring.

$$h(X,i,j) = -\sum_{\mathbf{x} \in X \stackrel{*}{\Rightarrow} w_i \dots w_j} P(\mathbf{x} \mid G) \lg(P(\mathbf{x} \mid G)),$$

where G is the current model. Under this notation, the tree entropy of a sentence, $\sum_{v \in \mathcal{V}} P(v \mid G) \lg P(v \mid G)$ is denoted as $h(S,1,n)$.

Analogous to the computation of Inside Probabilities, we compute $h(X,i,j)$ recursively. The base case is when the nonterminal X generates a single token substring w_i.

14 That is, every production rule must be in one of two forms: a nonterminal expands into two more nonterminals, or a nonterminal expands into a terminal.

The only possible tree is for X to be at the root, immediately dominating the leaf node w_i. Therefore, the tree entropy is:

$$h(X,i,i) = e(X,i,i)\lg(e(X,i,i))$$

For the general case, $h(X,i,j)$, we must find all rules of the form $X \to YZ$, where Y and Z are nonterminals, that have contributed towards $X \stackrel{*}{\Rightarrow} w_i \dots w_j$. To do so, we consider all possible ways dividing up $w_i \dots w_j$ into two pieces such that $Y \stackrel{*}{\Rightarrow} w_i \dots w_k$ and $Z \stackrel{*}{\Rightarrow} w_{k+1} \dots w_j$:

$$h(X,i,j) = \sum_{k=i}^{j-1} \sum_{(X \to YZ)} h_{Y,Z,k}(X,i,j).$$

The function $h_{Y,Z,k}(X,i,j)$ is a portion of $h(X,i,j)$ that accounts for those parses in which the rule $X \to YZ$ is used and the division point is at word w_k. The nonterminals Y and Z may, in turn, generate their substrings with multiple parses. Let $\mathcal{Y}$ represent the set of parses for $Y \stackrel{*}{\Rightarrow} w_i \dots w_k$; let $\mathcal{Z}$ represent the set of parses for $Z \stackrel{*}{\Rightarrow} w_{k+1} \dots w_j$; and let x represent the parse step of $X \to YZ$. Then, there are a total of $\|\mathcal{Y}\| \times \|\mathcal{Z}\|$ parses, and the probability of each parse is $P(x)P(\mathbf{y})P(\mathbf{z})$, where $\mathbf{y} \in \mathcal{Y}$ and $\mathbf{z} \in \mathcal{Z}$. To compute $h_{Y,Z,k}$, we need to sum over all possible parses:

$$\begin{aligned} h_{Y,Z,k}(X,i,j) &= -\sum_{\mathbf{y}\in\mathcal{Y},\mathbf{z}\in\mathcal{Z}} P(x)P(\mathbf{y})P(\mathbf{z})\lg(P(x)P(\mathbf{y})P(\mathbf{z})) \\ &= -\sum_{\mathbf{y}\in\mathcal{Y},\mathbf{z}\in\mathcal{Z}} P(x)P(\mathbf{y})P(\mathbf{z})(\lg P(x) + \lg P(\mathbf{y}) + \lg P(\mathbf{z})) \\ &= -P(x)\lg(P(x))e(Y,i,k)e(Z,k+1,j) \\ &\quad +P(x)h(Y,i,k)e(Z,k+1,j) \\ &\quad +P(x)e(Y,i,k)h(Z,k+1,j). \end{aligned}$$

Thus, the tree entropy of the entire sentence can be recursively computed from the entropy values of the substrings.