Automatic Prediction of Parser Accuracy

Sujith Ravi and Kevin Knight
University of Southern California
Information Sciences Institute
Marina del Rey, California 90292
{sravi,knight}@isi.edu

Radu Soricut
Language Weaver, Inc.
4640 Admiralty Way, Suite 1210
Marina del Rey, California 90292
rsoricut@languageweaver.com

Abstract

Statistical parsers have become increasingly accurate, to the point where they are useful in many natural language applications. However, estimating parsing accuracy on a wide variety of domains and genres is still a challenge in the absence of gold-standard parse trees.

In this paper, we propose a technique that automatically takes into account certain characteristics of the domains of interest, and accurately predicts parser performance on data from these new domains. As a result, we have a cheap (no annotation involved) and effective recipe for measuring the performance of a statistical parser on any given domain.

1 Introduction

Statistical natural language parsers have recently become more accurate and more widely available. As a result, they are being used in a variety of applications, such as question answering (Hermjakob, 2001), speech recognition (Chelba and Jelinek, 1998), language modeling (Roark, 2001), language generation (Soricut, 2006) and, most notably, machine translation (Charniak et al., 2003; Galley et al., 2004; Collins et al., 2005; Marcu et al., 2006; Huang et al., 2006; Avramidis and Koehn, 2008). These applications are employed on a wide range of domains and genres, and therefore the question of how accurate a parser is on the domain and genre of interest becomes acute. Ideally, one would want to have available a recipe for precisely answering this question: "given a parser and a particular domain of interest, how accurate are the parse trees produced?" The only recipe that is implicitly given in the large literature on parsing to date is to have human annotators build parse trees for a sample set from the domain of interest, and consequently use them to compute a PARSEVAL (Black et al., 1991) score that is indicative of the intrinsic performance of the parser. Given the wide range of domains and genres for which NLP applications are of interest, combined with the high expertise required from human annotators to produce parse tree annotations, this recipe is, albeit precise, too expensive. The other recipe that is currently used on a large scale is to measure the performance of a parser on existing treebanks, such as WSJ (Marcus et al., 1993), and assume that the accuracy measure will carry over to the domains of interest. This recipe, albeit cheap, cannot provide any guarantee regarding the performance of a parser on a new domain, and, as experiments in this paper show, can give wrong indications regarding important decisions for the design of NLP systems that use a syntactic parser as an important component.

This paper proposes another method for measuring the performance of a parser on a given domain that is both cheap and effective. It is a fully automated procedure (no expensive annotation involved) that uses properties of both the domain of interest and the domain on which the parser was trained in order to measure the performance of the parser on the domain of interest. It is, in essence, a solution to the following prediction problem:

Input: (1) a statistical parser and its training data, (2) some chunk of text from a new domain or genre

Output: an estimate of the accuracy of the parse trees produced for that chunk of text

Accurate estimations for this prediction problem will allow a system designer to make the right decisions for the given domain of interest. Such decisions include, but are not restricted to, the choice of the parser, the choice of the training data, the choice of how to implement various components such as the treatment of unknown words, etc. Altogether, a correct estimation of the impact of such decisions on the resulting parse trees can guide a system designer in a hill-climbing scenario for which an extrinsic metric (such as the impact on the overall quality of the system) is usually too expensive to be employed often enough. To provide an example, a machine translation engine that requires parse trees as training data in order to learn syntax-based translation rules (Galley et al., 2006) needs to employ a syntactic parser as soon as the training process starts, but it can take up to hundreds and even thousands of CPU hours (for large training data sets) to train the engine before translations can be produced and measured. Although a real estimate of the impact of a parser design decision in this scenario can only be gauged from the quality of the translations produced, it is impractical to create such estimates for each design decision. On the other hand, estimates using the solution proposed in this paper can be obtained fast, before submitting the parser output to a costly training procedure.

2 Related Work and Experimental Framework

There have been previous studies which explored the problem of automatically predicting the task difficulty for various NLP applications. (Albrecht and Hwa, 2007) presented a regression based method for developing automatic evaluation metrics for machine translation systems without directly relying on human reference translations. (Hoshino and Nakagawa, 2007) built a computer-adaptive system for generating questions to teach English grammar and vocabulary to students, by predicting the difficulty level of a question using various features. There have been a few studies of English parser accuracy in domains/genres other than WSJ (Gildea, 2001; Bacchiani et al., 2006; McClosky et al., 2006), but in order to make measurements for such studies, it is necessary to have gold-standard parses in the non-WSJ domain of interest.

Gildea (2001) studied how well WSJ-trained parsers do on the Brown corpus, for which a gold standard exists. He looked at sentences with 40 words or less. (Bacchiani et al., 2006) carried out a similar experiment on sentences of all lengths, and (McClosky et al., 2006) report additional results. The table below shows results from our own measurements of Charniak parser[1] (Charniak and Johnson, 2005) accuracy (F-measure on sentences of all lengths), which are consistent with these studies. For the Brown corpus, the test set was formed from every tenth sentence in the corpus (Gildea, 2001).

Training Set	**Test Set**	**Sent. count**	**Charniak accuracy**
WSJ sec. 02-21 (39,832 sent.)	WSJ sec. 24	1308	90.48
	WSJ sec. 23	2343	91.13
	Brown-test	2186	86.34

Here we investigate algorithms for predicting the accuracy of a parser $\mathcal{P}$ on sentences, chunks of sentences, and whole corpora. We also investigate and contrast several scenarios for prediction: (1) the predictor looks at the input text only, (2) the predictor looks at the input text and the output parse trees of $\mathcal{P}$, and (3) the predictor looks at the input text, the output parse trees of $\mathcal{P}$, and the outputs of other programs, such as the output parse trees of a different parser $\mathcal{P}_{ref}$ used as a reference. Under none of these scenarios is the predictor allowed to look at gold-standard parses in the new domain/genre.

The intuition behind what we are trying to achieve here can be compared to an analogous task—trying to assess the performance of a median student from a math class on a given test, without having access to the answer sheet. Looking at the test only, we could probably tell whether the test looks hard or not, and therefore whether the student will do well or not. Looking at the student's answers will likely give us an even better idea of the performance. Finally, the answers of a second student with similar proficiency will provide even better clues: if the students agree on every answer, then they probably both did well, but if they disagree frequently, then they (and hence our student) probably did not do as well.

Our first experiments are concerned with validating the idea itself: can a predictor be trained such

[1] Downloaded from *ftp.cs.brown.edu/pub/nlparser/rerankingparserAug06.tar.gz* in February, 2007.

that it predicts the same F-scores as the ones obtained using gold-trees? We first validate this using the WSJ corpus itself, by dividing the WSJ treebank into several sections:

1. *Training* (WSJ section 02-21). The parser $\mathcal{P}$ is trained on this data.

2. *Development* (WSJ section 24). We use this data for training our predictor.

3. *Test* (WSJ section 23). We use this data for measuring our predictions. For each test sentence, we compute (1) the PARSEVAL F-measure score using the test gold standard, and (2) our predicted F-measure. We report the correlation coefficient (r) between the actual F-scores and our predicted F-scores. We will also use a root-mean-square error (rms error) metric to compare actual and predicted F-scores.

Section 3 describes the features used by our predictor. Given these features, as well as actual F-scores computed for the development data, we use supervised learning to set the feature weights. To this end, we use SVM-Regression[2] (Smola and Schoelkopf, 1998) with an RBF kernel, to learn the feature weights and build our predictor system.[3] We validate the accuracy of the predictor trained in this fashion on both WSJ (Section 4) and the Brown corpus (Section 5).

3 Features Used for Predicting Parser Accuracy

3.1 Text-based Features

One hypothesis we explore is that (all other things being equal) longer sentences are harder to parse correctly than shorter sentences. When exposed to the development set, SVM-Regression learns weights to best predict F-scores using the values for this feature corresponding to each sentence in the corpus.

Does the predicted F-score correlate with actual F-score on a sentence by sentence basis? There was a positive but weak correlation:

Feature set	dev (r)	test (r)
Length	0.13	0.19

Another hypothesis is that the parser performance is influenced by the number of UNKNOWN words in the sentence to be parsed, i.e., the number of words in the test sentence that were never seen before in the training set. Training the predictor with this feature produces a positive correlation, slightly weaker compared to the Length feature.

Feature set	dev (r)	test (r)
UNK	0.11	0.11

Unknown words are not the only ones that can influence the performance of a parser. Rare words, for which statistical models do not have reliable estimates, are also likely to impact parsing accuracy. To test this hypothesis, we add a language model perplexity–based (LM-PPL) feature. We extract the yield of the training trees, on which we train a trigram language model.[4] We compute the perplexity of each test sentence with respect to this language model, and use it as feature in our predictor system. Note that this feature is meant as a refinement of the previous UNK feature, in the sense that perplexity numbers are meant to signal the occurrence of unknown words, as well as rare (from the training data perspective) words. However, the correlation we observe for this feature is similar to the correlation observed for the UNK feature, which seems to suggest that the smoothing techniques used by the parsers employed in these experiments lead to correct treatment of the rare words.

Feature set	dev (r)	test (r)
LM-PPL	0.11	0.10

We also look at the possibility of automatically detecting certain "cue" words that are appropriate for our prediction problem. That is, we want to see if we can detect certain words that have a discriminating power in deciding whether parsing a sentence that contains them is difficult or easy. To this end, we use a subset of the development data, which contains the 200 best-parsed and 200 worst-parsed sentences (based on F-measure scores). For each word in the development dataset, we compute the information gain (IG) (Yang and Pedersen, 1997) score for that word with respect to the best/worst parsed

[2]Weka software (http://www.cs.waikato.ac.nz/ml/weka/)

[3]We compared a few regression algorithms like SVM-Regression (using different kernels and parameter settings) and Multi-Layer Perceptron (neural networks) – we trained the algorithms separately on dev data and picked the one that gave the best cross-validation accuracy (F-measure).

[4]We trained using the SRILM language modeling toolkit, with default settings.

dataset. These words are then ranked by their IG scores, and the top 100 words are included as lexical features in our predictor system. As expected, the correlation on the development set is quite high (given that these lexical cues are extracted from this particular set), but a positive correlation holds for the test set as well.

Feature set	dev (r)	test (r)
lexCount100	0.43	0.18

3.2 Parser $\mathcal{P}$–based Features

Besides exploiting the information present in the input text, we can also inspect the output tree of the parser $\mathcal{P}$ for which we are interested in predicting accuracy. We create a rootSYN feature based on the syntactic category found at the root of the output tree ("is it S?", "is it FRAG?"). We also create a puncSYN feature based on the number of words labeled as punctuation tags (based on the intuition that heavy use of punctuation can be indicative of the difficulty of the input sentences), and a labelSYN feature in which we bundled together information regarding the number of internal nodes in the parse tree output that have particular labels ("how many nodes are labeled with PP?"). In our predictor, we use 72 such labelSYN features corresponding to all the syntactic labels found in the parse tree output for the development set. The test set correlation given by the rootSYN and the labelSYN features are higher than some of the text-based features, whereas the puncSYN feature seems to have little discriminative power.

Feature set	dev (r)	test (r)
rootSYN	0.21	0.17
puncSYN	0.09	0.01
labelSYN	0.33	0.28

3.3 Reference Parser $\mathcal{P}_{ref}$–based Features

In addition to the text-based features and parser $\mathcal{P}$–based features, we can bring in an additional parser $\mathcal{P}_{ref}$ whose output is used as a reference against which the output of parser $\mathcal{P}$ is measured. For the reference parser feature, our goal is to measure how similar/different are the results from the two parsers. We find that if the parses are similar, they are more likely to be right. In order to compute similarity, we can compare the constituents in the two parse trees from $\mathcal{P}$ and $\mathcal{P}_{ref}$, and see how many constituents match. This is most easily accomplished by considering $\mathcal{P}_{ref}$ to be a "gold standard" (even though it is not necessarily a correct parse) and computing the F-measure score of parser $\mathcal{P}$ against $\mathcal{P}_{ref}$. We use this F-measure score as a feature for prediction.

For the experiments presented in this section we use as $\mathcal{P}_{ref}$, the parser from (Bikel, 2002). Intuitively, the requirement for choosing parser $\mathcal{P}_{ref}$ in conjunction with parser $\mathcal{P}$ seems to be that they are different enough to produce non-identical trees when presented with the same input, and at the same time to be accurate enough to produce reliable parse trees. The choice of $\mathcal{P}$ as (Charniak and Johnson, 2005) and $\mathcal{P}_{ref}$ as (Bikel, 2002) fits this bill, but many other choices can be made regarding $\mathcal{P}_{ref}$, such as (Klein and Manning, 2003; Petrov and Klein, 2007; McClosky et al., 2006; Huang, 2008). We leave the task of creating features based on the consensus of multiple parsers as future work.

The correlation given by the reference parser–based feature $\mathcal{P}_{ref}$ on the test set is the highest among all the features we explored.

Feature set	dev (r)	test (r)
$\mathcal{P}_{ref}$	0.40	0.36

3.4 The Aggregated Power of Features

The table below lists all the individual features we have described in this section, sorted according to the correlation value obtained on the test set.

Feature set	dev (r)	test (r)
$\mathcal{P}_{ref}$	0.40	0.36
labelSYN	0.33	0.28
lexCount500	0.56	0.23
lexBool500	0.58	0.20
lexCount1000	0.67	0.20
lexBool1000	0.58	0.20
Length	0.13	0.19
lexCount100	0.43	0.18
lexBool100	0.43	0.18
rootSYN	0.21	0.17
UNK	0.11	0.11
LM-PPL	0.11	0.10
puncSYN	0.09	0.01

Note how the lexical features tend to over-fit the development data—the words were specifically chosen for their discriminating power on that particular set. Hence, adding more lexical features to the predictor system improves the correlation on development (due to over-fitting), but it does not produce consistent improvement on the test set. However,

Method (using 3 features: Length, UNK, $\mathcal{P}_{ref}$)	**# of random restarts**	**dev (r)**
SVM Regression		**0.42**
Maximum Correlation Training (MCT)	1	0.138
	5	0.136
	10	0.166
	25	0.178
	100	0.232
	1000	0.27
	10,000	0.401

Table 1: Comparison of correlation (r) obtained using MCT versus SVM-Regression on development corpus.

there is some indication that the counts of the lexical features are important, and count-based lexical features tend to have similar or better performance compared to their boolean-based counterparts.

Since these features measure different but overlapping pieces of the information available, it is to be expected that some of the feature combinations would provide better correlation that the individual features, but the gains are not strictly additive. By taking the individual features that provide the best discriminative power, we are able to get a correlation score of 0.42 on the test set.

Feature set	**dev (r)**	**test (r)**
$\mathcal{P}_{ref}$ + labelSYN + Length + lexCount100 + rootSYN + UNK + LM-PPL	0.55	0.42

3.5 Optimizing for Maximum Correlation

If our goal is to obtain the highest correlations with the F-score measure, is SVM regression the best method? Liu and Gildea (2007) recently introduced Maximum Correlation Training (MCT), a search procedure that follows the gradient of the formula for correlation coefficient (r). We implemented MCT, but obtained no better results. Moreover, it required many random re-starts just to obtain results comparable to SVM regression (Table 1).

4 Predicting Accuracy on Multiple Sentences

The results for the scenario presented in Section 3 are encouraging, but other scenarios are also important from a practical perspective. For instance, we are interested in predicting the performance of a particular parser not on a sentence-by-sentence basis, but for a representative chunk of sentences from the new domain. In order to predict the F-measure on multiple sentences, we modify our feature set to generate information on a whole chunk of sentences

Sentences in chunk (n)	**WSJ-test (r)**	**WSJ-test (rms error)**
1	0.42	0.098
20	0.61	0.026
50	0.62	0.019
100	0.69	0.015
500	0.79	0.011

Table 2: Performance of predictor on n-sentence chunks from WSJ-test (Correlation and rms error between actual/predicted accuracies).

rather than a single sentence. Predicting the correlation at chunk level is, not unexpectedly, an easier problem than predicting correlation at sentence level, as the results in the first two columns of Table 2 show.

For 100-sentence chunks, we also plot the predicted accuracies versus actual accuracies for the WSJ-test set in Figure 1. This scatterplot brings to light an artifact of using correlation metric (r) for evaluating our predictor's performance. Although our objective is to improve correlation between actual and predicted F-scores, the correlation metric (r) does not tell us directly how well the predictor is doing. In Figure 1, the system predicts that on an average, most sentence chunks can be parsed with an accuracy of 0.9085 (which is the mean predicted F-score on WSJ-test). But the range of predictions from our system [0.89,0.92] is smaller than the actual F-score range [0.86,0.95]. Hence, even though the correlation scores are high, this does not necessarily mean that our predictions are on target. An additional metric, *root-mean-square (rms) error*, which measures the distance between actual and predicted F-measures, can be used to gauge the quality of our predictions. For a particular chunk-size, lowering the *rms error* translates into aligning the points of a scatterplot as the one in Figure 1, closer to the x=y line, implying that the predictor is getting better at exactly predicting the F-score values. The third column in Table 2 shows the rms error for our predictor at different chunk sizes. The results using this metric also show that the prediction problem becomes easier as the chunk size increases.

Assuming that we have the test set of WSJ section 23, but without the gold-standard trees, how can we get an approximation for the overall accuracy of a parser $\mathcal{P}$ on this test set? One possibility, which we use here as a baseline, is to compute the F-score on a set for which we do have gold-standard trees. If we use our development set (WSJ section

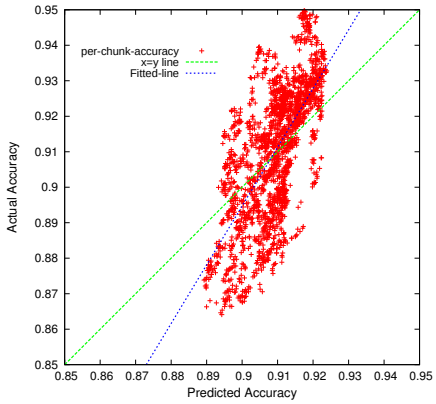

Figure 1: Plot showing Actual vs. Predicted accuracies for WSJ-test (100-sentence chunks). Each plot point represents a 100-sentence chunk. *(rms error = 0.015)*

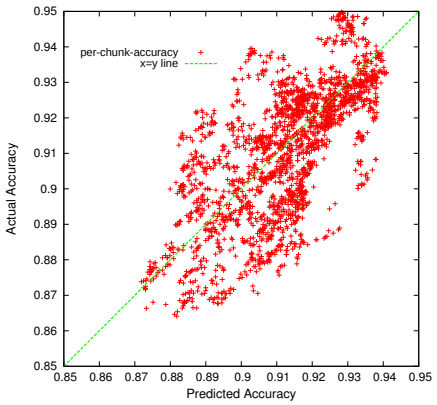

Figure 2: Plot showing Actual vs. Adjusted Predicted accuracies *(shifting with $\alpha = 0.757$, skewing with $\beta = 1.0$)* for WSJ-test (100-sentence chunks). *(rms error = 0.014)*

System	F-measure
Charniak F-measure on WSJ-dev (baseline)	90.48 (f_d)
Predictor (feature weights set with WSJ-dev)	**90.85** (f_p)
Actual Charniak accuracy	91.13 (f_t)

Table 3: Comparing Charniak parser accuracy (from different systems) on entire WSJ-test corpus

24) for this purpose, and (Charniak and Johnson, 2005) as the parser $\mathcal{P}$, the baseline is an F-score of 90.48 (f_d), which is the actual Charniak parser accuracy on WSJ section 24. Instead, if we run our predictor on the test set (a single chunk containing all the sentences in the test set), it predicts an F-score of 90.85 (f_p). These two predictions are listed as the first two rows in Table 3. Of course, having the actual gold-standard trees for WSJ section 23 helps us decide which prediction is better: the actual accuracy of the Charniak parser on WSJ section 23 is an F-score of 91.13 (f_t), which makes our prediction better than the baseline.

4.1 Shifting Predictions to Match Actual Accuracy

We correctly predict (in Table 3) that the WSJ-test is easier to parse than the WSJ-dev (90.85 $>$ 90.48). However, our predictor is too conservative—the WSJ-test is actually even easier to parse (91.13 $>$ 90.85). We can fix this by shifting the mean predicted F-score (which is equal to f_p) further away from the dev F-measure (f_d), and closer to the actual F-measure (f_t). This is achieved by shifting all the individual predictions by a certain amount as shown below.

Let p be an individual prediction from our system. The shifted prediction p' is given by:

$$p' = p + \alpha(f_p - f_d) \quad (1)$$

We can tune α to make the new mean prediction (f'_p) to be equal to the actual F-measure (f_t).

$$f'_p = f_p + \alpha(f_p - f_d) \quad (2)$$

$$\alpha = \frac{f_t - f_p}{f_p - f_d} \quad (3)$$

Using the F-score values from Table 3, we get an $\alpha = 0.757$ and an exact prediction of 91.13. Of course, this is because we tune on test, so we need to validate this idea on a new test set to see if it leads to improved predictions (Section 5).

4.2 Skewing to Widen Prediction Range

Our predictor is also too conservative about its *distribution* (see Figure 1). It knows (roughly) which chunks are easier to parse and which are harder, but its range of predictions is lower than the range of actual F-measure scores.

We can skew individual predictions so that sentences predicted to be easy are re-predicted to be even easier (and those that are hard to be even harder). For each prediction p' (from Equation 1), we compute

$$p'' = p' + \beta(p' - f'_p) \quad (4)$$

We simply set β to 1.0, doubling the distance of each prediction p' (in Equation 1) from the (adjusted) mean prediction f'_p, to obtain the skewed prediction p''.

Figure 2 shows how the points representing 100-sentence chunks in Figure 1 look after the predictions have been shifted ($\alpha = 0.757$) and skewed ($\beta = 1.0$). These two operations have the desired effect of changing the range of predictions from [0.89,0.92] to [0.87,0.94], much closer to the actual

Sentences in chunk (n)	WSJ-test (rms error)	Brown-test Prediction (rms error)	Brown-test Adjusted Prediction (rms error)
1	0.098	0.129	0.139
20	0.026	0.039	0.036
50	0.019	0.032	0.029
100	0.015	0.025	0.020
500	0.011	0.038	0.024

Table 4: Performance of predictor on n-sentence chunks from WSJ-test and Brown-test (rms error between actual/predicted accuracies).

range of [0.86,0.95]. The points in the new plot (Figure 2) also align closer to the "x=y" line than in the original graph (Figure 1). The rms error also drops from 0.015 to 0.014 (7% relative reduction), showing that the predictions have improved.

Since we use the WSJ-test corpus to tune the parameter values for shifting and skewing, we need to apply our predictor on a different test set to see if we get similar improvements by using these techniques, which we do in the next section.

5 Predicting Accuracy on the Brown Corpus

The Brown corpus represents a genuine challenge for our predictor, as it presents us with the opportunity to test the performance of our predictor in an out-of-domain scenario. Our predictor, trained on WSJ data, is now employed to predict the performance of a WSJ-trained parser $\mathcal{P}$ on the Brown-test corpus. As in the previous experiments, we use (Charniak and Johnson, 2005) trained on WSJ sections 02-21 as parser $\mathcal{P}$. The feature weights for our predictor are again trained on section 24 of WSJ, and the shifting and skewing parameters ($\alpha = 0.757$, $\beta = 1.0$) are determined using section 23 of WSJ.

The results on the Brown-test, both the original predictions and after they have been adjusted (shifted/skewed), are shown in Table 4, at different level of chunking. For chunks of size n $>$ 1, the shifting and skewing techniques help in lowering the rms error. On 100-sentence chunks from the Brown test, shifting and skewing ($\alpha = 0.757$, $\beta = 1.0$) leads to a 20% relative reduction in the rms error.

In a similar vein with the evaluation done in Section 4, we are interested in estimating the overall accuracy of a WSJ-trained parser $\mathcal{P}$ given an out-of-domain set such as the Brown test set (for which, at least for now, we do not have access to gold-standard trees). If we use (Charniak and Johnson, 2005) as parser $\mathcal{P}$, a cheap and readily-available answer is to approximate the performance using the Charniak parser performance on WSJ section 23, which has an F-score of 91.13. Another cheap and readily-available answer is to take the Charniak parser performance on WSJ section 24 with an F-score of 90.48. Table 5 lists these baselines, along with the prediction made by our system when using a single chunk containing all the sentences in the Brown test set (both base predictions and adjusted predictions, i.e. shifting using $\alpha = 0.757$). Again, having gold-standard trees for the Brown test set helps us decide which prediction is better. Our predictions are much closer to the actual Charniak parser performance on the Brown-test set, with the adjusted prediction at 86.96 compared to the actual F-score of 86.34.

System	F-measure
Baseline1 (F-measure on WSJ sec. 23)	91.13
Baseline2 (F-measure on WSJ sec. 24)	90.48
Predictor (base)	88.48
Adjusted Predictor (shifting using $\alpha = 0.757$)	**86.96**
Actual accuracy	86.34

Table 5: Charniak parser accuracy on entire Brown-test corpus

6 Ranking Parser Performance

One of the main goals for computing F-score figures (either by traditional PARSEVAL evaluation against gold standards or by methods such as the one proposed in this paper) is to compare parsing accuracy when confronted with a choice between various parser deployments. Not only are there many parsing techniques available (Collins, 2003; Charniak and Johnson, 2005; Petrov and Klein, 2007; McClosky et al., 2006; Huang, 2008), but recent annotation efforts in providing training material for statistical parsing (LDC, 2005; LDC, 2006a; LDC, 2006b; LDC, 2006c; LDC, 2007) have compounded the difficulty of the choices ("Do I parse using parser X?", "Do I train parser X using the treebank Y or Z?"). In this section, we show how our predictor can provide guidance when dealing with some of these choices, namely the choice of the training material to use with a statistical parser, prior to its application in an NLP task.

For the experiments reported in this paper, we use as parser $\mathcal{P}$, our in-house implementation of the Collins parser (Collins, 2003), to which various

speed-related enhancements (Goodman, 1997) have been applied. This choice has been made to better reflect a scenario in which parser $\mathcal{P}$ would be used in a data-intensive application such as syntax-driven machine translation, in which the parser must be able to run through hundreds of millions of training words in a timely manner. We use the more accurate, but slower Charniak parser (Charniak and Johnson, 2005) as the reference parser $\mathcal{P}_{ref}$ in our predictor (see Section 3.3). In order to predict the Collins-style parser behavior on the ranking task, we use the same predictor model (including feature weights and adjustment parameters) that was used for predicting Charniak parser behavior on the Brown corpus (Section 5).

We compare three training scenarios that make for three different parsers:

(1) P_{WSJ} - trained on sections 02-21 of WSJ.

(2) P_{News} - trained on the union of the English Chinese Translation Treebank (LDC, 2007) (news stories from Xinhua News Agency translated from Chinese into English) and the English Newswire Translation Treebank (LDC, 2005; LDC, 2006a; LDC, 2006b; LDC, 2006c) (An-Nahar new stories translated from Arabic into English).

(3) $\mathrm{P}_{WSJ-News}$ - trained on the union of all the above training material.

When comparing the performance of these three parsers on a development set from WSJ (section 0), we get the following F-scores.[5]

Parser	**WSJ (sec. 0) Accuracy (F-scores)**
P_{WSJ}	88.25
P_{News}	83.00
$\mathrm{P}_{WSJ-News}$	88.00

Consider now that we are interested in comparing the parsing accuracy of these parsers on a domain completely different from WSJ. The ranking $\mathrm{P}_{WSJ}>\mathrm{P}_{WSJ-News}>\mathrm{P}_{News}$, given by the evaluation above, provides some guidance, but is this guidance accurate? The intuition here is that the information that we already have about the new domain of interest (which implicitly appears in texts extracted from this domain), can be used to better guide this decision. Our predictor is able to capitalize on this information, and provide domain-informed guidance for choosing the most accurate parser to use with the new data, which in this case relates to choosing the best training strategy for the parser $\mathcal{P}$. If we consider as our domain of interest, news stories from Xinhua News Agency, then using our predictor on a chunk of 1866 sentences from this domain gives the F-scores shown in the second column of Table 6.

Parser	**Xinhua News Prediction (F-scores)**	**Xinhua News Accuracy (F-scores)**
P_{WSJ}	85.1	79.14
P_{News}	87.0	84.84
$\mathrm{P}_{WSJ-News}$	89.4	85.14

Table 6: Performance of predictor on the Xinhua News domain, compared with actual F-scores.

As with the previous experiments, we can compute the actual PARSEVAL F-scores (using gold-standard) for this particular 1866-sentence test set, as it happens to be part of the English Chinese Translation Treebank (LDC, 2007). These F-score figures are shown in the third column of Table 6. As these results show, for this particular domain the correct ranking is $\mathrm{P}_{WSJ-News}>\mathrm{P}_{News}>\mathrm{P}_{WSJ}$, which is exactly the ranking predicted by our method, without the aid of gold-standard trees.

We observe that even though the system predicts the ranking correctly, the predictions in the Xinhua News domain might not be as accurate in comparison to the predictions on Brown corpus (predicted F-score = 86.96, actual F-score = 86.34). One possible reason for this lower accuracy is that we use the same prediction model without optimizing for the particular parser on which we wish to make predictions. Still, the model was able to make distinctions between multiple parsers for the ranking task correctly, and decide the best parser to use with the given data. We believe this to be useful in typical NLP applications which use parsing as a component, and where making the right choice between different parsers can affect the end-to-end accuracy of the system.

7 Conclusion

The steady advances in statistical parsing over the last years have taken this technology to the point

[5]Because of tokenization differences between the different treebanks involved in these experiments, we have to adopt a tokenization scheme different from the one used in the Penn Treebank, and therefore the F-scores, albeit in the same range, are not directly comparable with the ones in the parsing literature.

where it is accurate enough to be useful in a variety of natural language applications. However, due to large variations in the characteristics of the domains for which these applications are developed, estimating parsing accuracy becomes more involved than simply taking for granted accuracy estimates done on a certain well-studied domain, such as WSJ. As the results in this paper show, it is possible to take into account these variations in the domain characteristics (encoded in our predictor as text-based, syntax-based, and agreement-based features)—to make better predictions about the accuracy of certain statistical parsers (and under different training scenarios), instead of relying on accuracy estimates done on a standard domain. We have provided a mechanism to incorporate these domain variations for making predictions about parsing accuracy, without the costly requirement of creating human annotations for each of the domains of interest. The experiments shown in the paper were limited to readily available statistical parsers (which are widely deployed in a number of applications), and certain domains/genres (because of ready access to gold-standard data on which we could verify predictions). However, the features we use in our predictor are independent of the particular type of parser or domain, and the same technique could be applied for making predictions on other parsers as well.

There are many avenues for future work opened up by the work presented here. The accuracy of the predictor can be further improved by incorporating more complex syntax-based features and multiple-agreement features. Moreover, rather than predicting an intrinsic metric such as the PARSEVAL F-score, the metric that the predictor learns to predict can be chosen to better fit the final metric on which an end-to-end system is measured, in the style of (Och, 2003). The end-result is a finely-tuned tool for predicting the impact of various parser design decisions on the overall quality of a system.

8 Acknowledgements

We wish to acknowledge our colleagues at ISI, who provided useful suggestions and constructive criticism on this work. We are also grateful to all the reviewers for their detailed comments and suggestions.

References

Joshua Albrecht and Rebecca Hwa. 2007. Regression for sentence-level mt evaluation with pseudo references. In *Proc. of ACL*.

Eleftherios Avramidis and Philipp Koehn. 2008. Enriching morphologically poor languages for statistical machine translation. In *Proc. of ACL*.

Michiel Bacchiani, Michael Riley, Brian Roark, and Richard Sproat. 2006. MAP adaptation of stochastic grammars. *Computer Speech & Language*, 20(1).

Daniel M. Bikel. 2002. Design of a multi-lingual, parallel-processing statistical parsing engine. In *Proc. of HLT*.

E. Black, S. Abney, D. Flickinger, C. Gdaniec, R. Grishman, P. Harrison, D. Hindle, R. Ingria, F. Jelinek, J. Klavans, M. Liberman, M. Marcus, S. Roukos, B. Santorini, and T. Strzalkowski. 1991. A procedure for quantitatively comparing the syntactic coverage of english grammars. In *Proc. of Speech and Natural Language Workshop*.

Eugene Charniak and Mark Johnson. 2005. Coarse-to-fine n-best parsing and MaxEnt discriminative reranking. In *Proc. of ACL*.

Eugene Charniak, Kevin Knight, and Kenji Yamada. 2003. Syntax-based language models for statistical machine translation. In *Proc. of MT Summit IX. IAMT*.

Ciprian Chelba and Frederick Jelinek. 1998. Exploiting syntactic structure for language modeling. In *Proc. of ACL*.

Michael Collins, Philipp Koehn, and Ivona Kucerova. 2005. Clause restructuring for statistical machine translation. In *Proc. of ACL*.

Michael Collins. 2003. Head-driven statistical models for natural language parsing. *Computational Linguistics*, 29(4).

Michel Galley, Mark Hopkins, Kevin Knight, and Daniel Marcu. 2004. What's in a translation rule? In *Proc. of HLT/NAACL*.

Michel Galley, Jonathan Graehl, Kevin Knight, Daniel Marcu, Steve DeNeefe, Wei Wang, and Ignacio Thayer. 2006. Scalable inferences and training of context-rich syntax translation models. In *Proc. of ACL*.

Daniel Gildea. 2001. Corpus variation and parser performance. In *Proc. of EMNLP*.

Joshua Goodman. 1997. Global thresholding and multiple-pass parsing. In *Proc. of EMNLP*.

Ulf Hermjakob. 2001. Parsing and question classification for question answering. In *Proc. of ACL Workshop on Open-Domain Question Answering*.

Ayako Hoshino and Hiroshi Nakagawa. 2007. A cloze test authoring system and its automation. In *Proc. of ICWL*.

Liang Huang, Kevin Knight, and Aravind Joshi. 2006. Statistical syntax-directed translation with extended domain of locality. In *Proc. of AMTA*.

Liang Huang. 2008. Forest reranking: Discriminative parsing with non-local features. In *Proc. of ACL*.

Dan Klein and Christopher D. Manning. 2003. Accurate unlexicalized parsing. In *Proc. of ACL*.

LDC. 2005. English newswire translation treebank. Linguistic Data Consortium, Catalog number LDC2005E85.

LDC. 2006a. English newswire translation treebank. Linguistic Data Consortium, Catalog number LDC2006E36.

LDC. 2006b. GALE Y1 Q3 release - English translation treebank. Linguistic Data Consortium, Catalog number LDC2006E82.

LDC. 2006c. GALE Y1 Q4 release - English translation treebank. Linguistic Data Consortium, Catalog number LDC2006E95.

LDC. 2007. English chinese translation treebank. Linguistic Data Consortium, Catalog number LDC2007T02.

Ding Liu and Daniel Gildea. 2007. Source-language features and maximum correlation training for machine translation evaluation. In *Proc. of NAACL-HLT*.

Daniel Marcu, Wei Wang, Abdessamad Echihabi, and Kevin Knight. 2006. Spmt: Statistical machine translation with syntactified target language phraases. In *Proc. of EMNLP*.

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

David McClosky, Eugene Charniak, and Mark Johnson. 2006. Reranking and self-training for parser adaptation. In *Proc. of COLING-ACL*.

Franz Joseph Och. 2003. Minimum error rate training in machine translation. In *Proc. of ACL*.

Slav Petrov and Dan Klein. 2007. Improved inference for unlexicalized parsing. In *Proc. of HLT/NAACL*.

Brian Roark. 2001. Probabilistic top-down parsing and language modelling. *Computational Linguistics*, 27(2).

A.J. Smola and B. Schoelkopf. 1998. A tutorial on support vector regression. *NeuroCOLT2 Technical Report NC2-TR-1998-030*.

Radu Soricut. 2006. *Natural Language Generation using an Information-Slim Representation*. Ph.D. thesis, University of Southern California,.

Y. Yang and J. Pedersen. 1997. A comparative study on feature selection in text categorization. In *Proc. of ICML*.