Distinguishing Easy and Hard Instances

Yuval Krymolowski
Department of Computer Science
Bar-Ilan University
52900 Ramat Gan, Israel

Abstract

Error analysis is a key step in developing statistical parsers. In doing this, we manually discover typical cases by examining parser output. In this paper we argue that the process can be speeded up by considering the output from an ensemble of parsers. We do this by resampling small proportions (10% and up) from the training data, and exploiting the high diversity of the resulting parsers - resulting from the sparseness of natural-language data. Varying the sample size, we can trace the gradual learning of each instance and classify instances into a few types. This division helps in distinguishing instances which are hard for the system, from instances which may be learned in principle. We suggest that such analysis can yield a qualitative approach to evaluation of statistical parsers.

1 Introduction

The task of parsing can be viewed as a detection task, where the goal of the parser is to detect instances of structures witnin the sentence. Accordingly, recall and precision are used in measuring the parser performance.

When we say "Recall is 80%", do we mean:

i. in the given test set, the chance of a single instance to be detected is 80%? or

ii. in an arbitrary sample, 80% of the instances will be detected?

In general, we have a single test data set at hand. Assuming it is representative, we adopt the second interpretation and conjecture that the recall carries to other data sets. There still remains, however, the question from the first interpretation: what are the chances of an instance to be detected? Or in more general terms: how easy or hard each instance is?

Some instances in natural language (NL) data are very easy to detect, or to classify correctly. For example, words with only one possible part-of-speech (POS) are easy to tag, prepositional phrases that include "of" are easy to attach, and NPs with a very common structure are easy to detect. On the other hand, some instances may be very hard to detect with statistical methods due to the inherent sparseness of NL data.

When analyzing the errors made in a single run of a supervised parser, we can discover problematic cases by observing recurring errors. But results from a single run show a partial picture, because they are sensitive to peculiarities in the training set. Such results may reflect failures or successes that are not due to properties of the model, but to a random balance of positive and negative evidence for certain types of instances.

If we knew, for a certain learning system, which types of instances are easy and which are hard to detect, and how "easy" or "hard" the individual instances are, this would provide us with means to qualitatively analyze the system's performance. This, in turn, would enable us to compare systems that are similar according to recall and precision. In such cases, it is possible that the systems differ in their ability to handle certain types of instances in a way which recall and precision do not reflect. Such a comparison would be qualitative in nature, unlike statistical significance tests such as McNemar's test whose goal is different.

In this paper, we propose a method which allows to study the easiness and hardness of test instances. The method relies on training the model on samples from the training data, thereby creating multiple parsers, and recording the instances detected each time.

We use the number of parsers that detected the instance as a measure of easiness. Due to the Zipfian nature of NL data, training samples are likely to differ in low-count instances. As each parser uses the features that were present in its training set, the parsers will differ with respect to the set of features they use. Each parser will provide a different viewpoint of the target structures. It is expected that easy instances will be detected by all or most of the parsers, while hard instances – missed by most of the parsers. Moreover, as we increase the sample size, we can trace the learning process at the instance level and characterize instances according to the variation of their easiness.

The idea of using an ensemble of supervised systems, trained on different samples, for making observations regarding individual instances is common in the frameworks of classifier combination and selective sampling. The boosting approach (Freund and Schapire, 1995) creates a classifier ensemble by training a system on samples drawn with preference to hard instances. Skalak (1997) suggests a more elaborated taxonomy of easiness and hardness levels, derived from leave-one-out results. He uses this information in order to remove uncharacteristic instances from the training set.

Query by committee (Seung et al., 1992) approaches select training instances for which the disagreement between classifiers is the highest. Abe and Mamitsuka (1998), proposes to obtain the classifier collection by sampling from the training set with a uniform distribution (as in bagging (Breiman, 1996)) or with preference to hard instances as in boosting.

We propose to use the ensemble of supervised parsers for *error analysis*. We demonstrate our method with a statistical memory-based shallow parsing algorithm and the task of NP detection.

2 Learning Algorithm

The experiments were carried out using the memory-based sequence learning algorithm presented by Dagan and Krymolowski (2001, hereafter MBSL). The system considers POS sequences at the beginning or end of target instances, NPs in our case. For each POS sequence, it records the number of times it appears in the beginning or end of an NP, as well as anywhere else in the corpus. For compositional NPs, the embedded NPs are treated in the same way as POS. This provides an additional level of abstraction. Figure 1 presents sample training data.

The input for parsing is a POS sequence, representing a tagged sentence. MBSL tests each subsequence, with its context, as a candidate for being an NP instance. Single words are tested first, then two-word subsequences and so on in increasing order of length. This allows the algorithm to use embedded NPs when testing composite ones.

If a subseqeuence corresponds to a real instance, then ideally, the entire POS string with its context would appear a number of times in the training data as an instance, and not in other structures. For example, the sequence[1]

"`VBZ DT NNS IN NNP .`"

appears once in Figure 1, third sentence, as an NP instance with one word in each side:

"`VBZ [`$_{\texttt{NP}}$ `DT NNS IN NNP` $_{\texttt{NP}}$`] .`"

In the more general case, MBSL tries to reconstruct the POS sequence from prefixes and suffixes of NPs in the training data, which we term *tiles*. For example, the NP

"`[`$_{\texttt{NP}}$ `DT JJ NN NNS IN NNP` $_{\texttt{NP}}$`]`"

has support for the prefix tile "`[`$_{\texttt{NP}}$ `DT JJ NN`" from the two first NPs in sentences 2 and 3, and for the suffix tiles "`IN NNP` $_{\texttt{NP}}$`]`" "`NNS IN NNP` $_{\texttt{NP}}$`]`" from the first NP in sentence 2 and the second NP in sentence 3 respectively.

The algorithm takes negative evidence into account as well. For example, the prefix tile "`[`$_{\texttt{NP}}$ `JJ`" appears in sentence 1, but "`JJ`" appears twice in other positions, that is, not as the first word in an NP. These two appearances constitute a negative evidence for this tile, which overshadows the single positive appearance, therefore MBSL will not use this tile.

Each NP candidate is given a score, the algorithm uses this score in disambiguation. MBSL considers up to a specified number c of context words, before and after an NP, in tiles. Here we use $c = 2$.

3 Easiness

Let T denote the set of NPs in the test data, and S a set of n training-set samples used for

[1] We use the Penn Treebank set of POS tags: `DT`=determiner, `JJ`=adjective, `RB`=adverb, `VBD`=verb in past tense, `VBN`=passive verb, `VB`=verb, `IN`=preposition, `NN`=singular noun, `NNS`=plural noun, and `CC`=coordinating conjunction.

1. [$_{\text{NP}}$ NNS $_{\text{NP}}$] [$_{\text{VP}}$ VBZ RB [$_{\text{NP}}$ JJ NNS $_{\text{NP}}$] $_{\text{VP}}$] .
2. [$_{\text{NP}}$ DT JJ NN IN NNP $_{\text{NP}}$] [$_{\text{VP}}$ VBZ [$_{\text{NP}}$ DT NN $_{\text{NP}}$] $_{\text{VP}}$] .
3. [$_{\text{NP}}$ DT JJ NN NN $_{\text{NP}}$] [$_{\text{VP}}$ VBZ [$_{\text{NP}}$ DT NNS IN NNP $_{\text{NP}}$] $_{\text{VP}}$] .

Figure 1: An example of training data for MBSL

training a supervised parser. We refer to the parser trained on the i^{th} sample as "parser i". Each instance $a \in T$ can be characterized by a bit-vector

$$v^S(a) = (v_1(a), \ldots, v_n(a)) \ ,$$

where

$$v_i(a) = \begin{cases} 1 & a \text{ was detected by parser } i \\ 0 & a \text{ was not detected by parser } i \ . \end{cases}$$

The vector $v^S(a)$ is the *detection profile* of a according to the set of samples S.

We extract from the detection profile the proportion

$$\text{easiness}^S(a) = \frac{\text{number of '1's in } v^S(a)}{n} \ ,$$

which is the probability of detecting a by one of the parsers in the ensemble. For easy and hard instances, easiness$^S(a)$ will be close to 1 and 0 respectively. The easiness does not depend only on the instance, but also on the training samples. We will discuss this issue further in the experiment section. In this paper we restrict our study to NPs marked in an annotated corpus, all of these NPs are therefore correct. In the general case, an instance can have a high easiness but still not be correct.

4 Experiments

4.1 Data

The training data used in our experiments consisted of Penn Treebank WSJ (Marcus et al., 1993) Sections 15-18, with Section 20 as test. These data sets were used by Ramshaw and Marcus (1995) and CoNLL-2000 shared task (Tjong Kim Sang and Buchholz, 2000) and have become a common testbed for shallow parsing tasks. Table 1 shows the number of sentences and NPs in the training and test data. We counted compositional NPs separately, as their structure is more complicated than that of base NPs.

	Training	Test
Dataset	WSJ 15-18	WSJ 20
Sentences	8936	2012
NPs:		
Base	50860	11401
Compositional	18472	4398
Total	69332	15799

Table 1: Training and test data

p_{samp}	10%	25%	50%	80%	95%
Avg. rec.%	81	83	84	85	85
Easiness:	proportion in test data				
0	3	4	5	8	10
0–0.1	9	9	10	12	14
0.1–0.2	3	2	2	1	1
0.2–0.3	2	2	1	1	0
0.3–0.4	2	2	1	1	0
0.4–0.5	2	2	1	0	0
0.5–0.6	2	2	2	1	0
0.6–0.7	2	2	1	1	0
0.7–0.8	3	2	2	2	0
0.8–0.9	5	3	2	2	1
0.9–1	70	74	78	81	84
1	52	61	67	74	78

Table 2: A summary of five resampling experiments: sample size, average recall, and distribution of easiness. Instances with easiness of 0 and 1 are counted within the corresponding ranges as well as separately.

4.2 Estimating Easiness

We conducted a few resampling experiments, each with a different sampling proportion $10\% \leq p_{\text{samp}} \leq 99.75\%$ of the training data. 1000 samples were taken in each experiment, resulting in an ensemble of $n = 1000$ parsers.

Table 2 presents a summary of five resampling experiments. For reference, the recall of the parser trained on the complete data is 85%. We see that the average recall rises as samples grow.

As p_{samp} is increased, there is more overlap between individual training samples. For example, two samples of 90% each will share at least 80% of the training data, while at the other extreme, two 10% samples are typically disjoint. This effect results in more similar parsers when the samples are larger. We therefore see an increase in the number of instances with extreme easiness values near 0 and 1 for larger p_{samp}. Ultimately, with the full training sample, each instance is either detected or missed, therefore, for each $a \in T$

$$\lim_{p_{\mathrm{samp}} \to 100\%} \mathrm{easiness}^{S}(a) = 0 \text{ or } 1 \,.$$

where S represents a set of training samples drew with proportion p_{samp}.

When the training samples are small, only very easy instances will have easiness $\simeq 1$, but instances with easiness $\simeq 0$ may not necessarily be hard. The converse happens when the samples are large: hard instances will have lower easiness values but instances with easiness $\simeq 1$ need not be very easy. In order to get a detailed view, we examined, for each $a \in T$, how its easiness varies with the sample size.

Figure 2 shows four patterns of easiness change. The curves can be viewed as "learning curves" for individual instances. As table 3 shows, most of the instances start with easiness values above 0.5 for small samples, which increase as the sample size grows. The higher the initial easiness, the faster it gets to the value of 1. As for the instances with initial easiness below 0.5 – they end up mostly undetected by the parser trained on the full training set. Such a behaviour coincides with our usual view of learning.

This is not, however, the whole story. Some instances start with very low easiness and yet end with a value of 1 ("learned" instances), while others start with high easiness but eventually are not detected ("forgotten" instances). We explain both cases by a borderline phenomenon for which there is little negative evidence in the first case, and little positive evidence in the second. For example, the construction "`NP IN NP`" may or may not be a compositional noun. When most such constructions in the training sample are not NPs, incidental balance of features in a small sample can still result in detecting such instances, whereas larger samples are less noisy and therefore less susceptible to this effect.

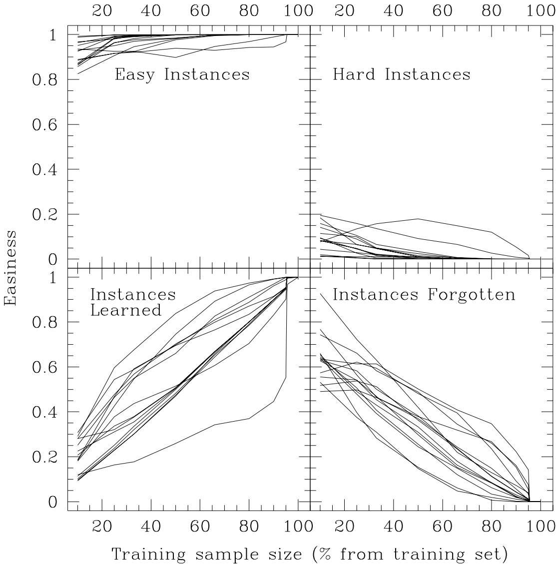

Figure 2: Four patterns of the easiness change with sampling proportion p_{samp}.

Easiness	Full Sample	
$p_{\mathrm{samp}} = 10\%$	Missed	Detected
0-50%	13% Hard	5% Learned
50-100%	1.5% Forgotten	80.5% Easy

Table 3: Frequencies of instances exhibiting the four types of easiness change

4.3 Easiness and Bagging

Taking small samples has the advantage of producing parsers which disagree more with one another, and can provide finer distinctions between instances. In memory-based learning, where it is important to keep all the available evidence from the training data (Daelemans et al., 1999), small samples have an advantage in requiring less space - with a tradeoff in performance.

In order to study whether we can still get a good performance with small samples, we ran bagging (Breiman, 1996) experiments. We used $p_{\mathrm{samp}} = 10\%$ and tried a range of thresholds. For each threshold θ, we selected the instances with easiness greater than θ, and calculated the recall, precision, and F_β with $\beta = 1$. The results are presented in Table 4 along with the performance of the parser trained on the full training set. As we see, it is possible to achieve a performance similar to that of the full model by bagging parsers trained on small samples of the training set.

As we use a single data set and a single method for this work, it is hard to say whether this holds for other tasks as well. This result may indicate that a small training material is sufficient for analyzing the particular test set (as also implied by the high recall values for low p_{samp} in table 2), we leave that for future research. In particular, while thresholds of 30% and 40% yielded recall and precision similar to those obtained by training on the full data set, it is not possible to say at present whether this is due to the task, the system, or the data sets in use.

Threshold	Recall	Precision	F_β
20%	88.3	65.5	75.2
30%	86.1	70.1	77.2
40%	84.0	73.4	78.3
50%	81.9	76.5	79.1
60%	79.5	79.0	79.3
70%	77.1	81.4	79.2
Full	85.0	71.4	77.6

Table 4: Results of bagging experiment with 10% samples

5 Discussion

Analyzing the results of a single run of a supervised parser, we can find frequent errors. This information is, however, limited to that run and does not always reflect why an instance was not detected. Possible reasons can be that the system cannot find supporting evidence in the training data, or due to a random balance of supporting and contradicting evidence for that instance. Distinguishing between errors of these types and, more generally, tracing the learning or "forgetting" patterns of instances, are important steps in analyzing the performance of a supervised parser. Combined with a clustering approach (Krymolowski and Marx, 2002), we may be able to group together instances with similar behaviour and structure, and speed up the process of error analysis.

When a model is probabilistic (e.g., DOP (Scha et al., 1999)), we can intuitively observe that instances that get a high probability are easy while those with low probability are hard. For models represented as a separator in an abstract numeric feature space (e.g., SNoW (Roth, 1998)), the distance from the separator can be an indication of easiness. Assuming the separator fluctuates within a bounded area of space for different training samples, the easy instances are those within a safe distance from that area, while the hard ones are more sensitive to errors resulting from noise in the training samples.

In this work, we proposed a method for estimating the easiness of instances which is suitable even for a non-probabilistic model or model which is not represented in an abstract numeric feature space. The method relies on generating an ensemble of parsers by resampling from the training data. We studied the effect of sample size on the distribution of easiness in a test sample, and presented learning curves for individual instances. The curves can help in finding easy instances that are being learned in the common way, as well as instances affected by noise, or for which little evidence does exist in the data although the full model misses them.

In using training samples, we took advantage of the Zipfian distribution of natural-language data. This yields samples that differ in the low-count instances they contain which, in turn, increases the difference between parsers trained on different samples. Given a sampling proportion, we sampled randomly from the data. It might be possible to increase the diversity among parsers by sampling chunks of sentences. As the style within a chunk is more uniform than within a collection of texts, this could focus each parser on a smaller number of instance types. This will sharpen the coverage differences between parsers (in expense of the coverage of each one).

As further work, we plan to investigate the relation between easiness and concepts like typicality (Zhang, 1992) and class prediction strength (e.g. Hoste and Daelemans (2000)). We also plan to study the extent to which easiness depends on the feature set used by the supervised parser, and compare a number of systems in order to find instances which are hard or easy for most of the methods (cf. Pedersen (2002) for the word-sense disambiguation task). We hope this would provide a means for *qualitative* comparison between systems. Further yet, we hope this would contribute to a more focused use of the individual learning methods, possibly in combination with hand-coded rules, saving learning effort for the cases where it is more needed.

Acknowledgements
The author thanks Zvika Marx for very helpful discussions, as well as Adam Kilgarriff, Miles Osborne, and the anonymous reviewers for their interesting and important comments.

References

N. Abe and H. Mamitsuka. 1998. Query learning strategies using boosting and bagging. In *Proceedings of ICML'98*, pages 1–10.

L. Breiman. 1996. Bagging predictors. *Machine Learning*, 24:123–140.

Walter Daelemans, Antal van den Bosch, and Jakub Zavrel. 1999. Forgetting exceptions is harmful in language learning. *Machine Learning*, 34:11–44.

I. Dagan and Y. Krymolowski. 2001. Compositional memory-based partial parsing. In R. Bod, R. Scha, and K. Sima'an, editors, *Data-Oriented Parsing*, chapter II.6. CSLI Publications, Stanford. to appear, http://turing.wins.uva.nl/∼rens/dopbook.html.

Y. Freund and R. E. Schapire. 1995. A decision-theoretic generalization of on-line learning and an application to boosting. In *Proceedings of the* 2^{nd} *European Conference on Computational Learning Theory*, pages 23–37. Springer-Verlag.

Véronique Hoste and Walter Daelemans. 2000. Comparing bagging and boosting for natural language processing tasks: a typicality approach. In Ad Feelders, editor, *Proceedings of Benelearn 2000*, pages 101–108.

Yuval Krymolowski and Zvika Marx. 2002. Clustering the space of phrases identified by an ensemble of supervised shallow parsers. In *Proceedings of ICML'02 Workshop on Text Learning (TextML-2002)*, Sydney, Australia, July.

M. P. Marcus, B. Santorini, and M. Marcinkiewicz. 1993. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330, June.

Ted Pedersen. 2002. Assessing system agreement and instance difficulty in the lexical sample tasks of senseval-2. In *Proceedings of the Workshop on Word Sense Disambiguation: Recent Successes and Future Directions*, July.

L. A. Ramshaw and M. P. Marcus. 1995. Text chunking using transformation-based learning. In *Proceedings of the Third Workshop on Very Large Corpora*.

D. Roth. 1998. Learning to resolve natural language ambiguities: A unified approach. In *proc. of the Fifteenth National Conference on Artificial Intelligence*, pages 806–813, Menlo Park, CA, USA, July. AAAI Press.

Remko Scha, Rens Bod, and Khalil Sima'an. 1999. A memory-based model of syntactic analysis: Data-oriented parsing. *Journal of Experimental and Theoretical AI*, 11:409–440.

H. Sebastian Seung, Manfred Opper, and Haim Sompolinsky. 1992. Query by committee. In *Proceedings of the ACM Workshop on Computational Learning Theory*, Pittsburgh, PA.

David Skalak. 1997. *Prototype Selection for Composite Nearest Neighbor Classifiers*. Ph.D. thesis, University of Massachusetts, Amherst, Massachusetts.

Erik. F. Tjong Kim Sang and Sabine Buchholz. 2000. Introduction to the conll-2000 shared task: Chunking. In *Proceedings of CoNLL-2000 and LLL-2000*, pages 127–132, Lisbon, Portugal.

J. Zhang. 1992. Selecting typical instances in instance-based learning. In *Proceedings of ICML'92*, pages 470–479.