Better Training for Function Labeling

Grzegorz Chrupała
Dublin City University
Dublin, Ireland
gchrupala@computing.dcu.ie

Nicolas Stroppa
Dublin City University
Dublin, Ireland
nstroppa@computing.dcu.ie

Josef van Genabith
Dublin City University
Dublin, Ireland
josef@computing.dcu.ie

Georgiana Dinu
Eberhard Karls Universität
Tübingen, Germany
gdinu@sfs.uni-tuebingen.de

Abstract

Function labels enrich constituency parse tree nodes with information about their abstract syntactic and semantic roles. A common way to obtain function-labeled trees is to use a two-stage architecture where first a statistical parser produces the constituent structure and then a second component such as a classifier adds the missing function tags.

In order to achieve optimal results, training examples for machine-learning-based classifiers should be as similar as possible to the instances seen during prediction. However, the method which has been used so far to obtain training examples for the function labeling classifier suffers from a serious drawback: the training examples come from perfect treebank trees, whereas test examples are derived from parser-produced, imperfect trees.

We show that extracting training instances from the reparsed training part of the treebank results in better training material as measured by similarity to test instances. We show that our training method achieves statistically significantly higher f-scores on the function labeling task for the English Penn Treebank. Currently our method achieves 91.47% f-score on the section 23 of WSJ, the highest score reported in the literature so far.

Keywords

function labeling, machine-learning

1 Introduction

Treebanks such as the English or Chinese Penn Treebanks are collections of syntactic parse trees. The trees include extra information in addition to constituent bracketing and labeling. In this paper we focus on the function labels (also known as function tags). The function labels used in the Penn treebanks fall into several types. Grammatical labels are used to encode the grammatical function of the constituent. Form-function labels are used to indicate the semantic class of adjuncts and discrepancies between form and function. There is also a label used for topicalization, and several other miscellaneous labels. Detailed information about the label sets can be found in the annotation guidelines for the respective treebanks (Bies, 1995; Xue and Xia, 2000). Table 1 provides a summary of labels used in the English and Chinese treebanks.

Widely used statistical parsers, such as those of (Collins, 1999; Charniak, 2000), which use treebanks as training data to parse unseen sentences, do not include function labels in the parse trees they produce. However, pure constituency trees may be insufficient for many NLP tasks - often something closer to semantic information is required. Grammatical functions and semantic roles such as those encoded in form-function labels are a step towards this deeper, abstract representation. Thus an important task is to be able to produce parses which include the richer annotations provided by function labels.

In this paper we review approaches to producing parse trees with function labels and present our research on the impact of different training methods in a two-stage processing architecture where we use machine learning techniques to train classifiers which add function labels to bare constituent trees such as those output by Charniak's or Collins' parsers.

In a multi-stage processing pipeline the optimal training input for the downstream stages is important. Ideally the training at stage $n+1$ should be performed on input from stage n: e.g. a parsing model which uses automatically POS-tagged input should be trained on tags produced by the POS tagger used to preprocess the raw input, rather than gold tags. In practice pipeline architectures this has been violated.

For example, in the case of function labeling, the two-stage models used in previous work have all used "perfect" treebank trees to train the function labeler even though the labeler operates on "imperfect" trees output by the parser. This is presumably due to the fact that the function labels we want to learn are attached to nodes in the treebank trees. Unfortunately, those nodes do not necessarily correspond to constituents in the trees produced by the parser.

The main contribution of our paper consists in presenting a theoretically sound method of training on parser output rather than treebank trees for the function labeling task and investigating the effect of several versions of this approach on the results as compared against the baseline method which uses perfect tree-

Label	Meaning	ETB	CTB
Clause types			
IMP	imperative		√
Q	question		√
Syntactic labels			
LGS	logical subject	√	√
PRD	predicate	√	√
PUT	complement of put	√	
SBJ	surface subject	√	√
IO	indirect object		√
OBJ	direct object		√
FOC	focus		√
Miscellaneous labels			
CLF	it-cleft	√	
HLN	headline	√	√
TTL	title	√	√
CLR	closely related	√	
APP	appositive		√
PN	proper noun		√
SHORT	short form		√
WH	WH-phrases		√
Semantic (form-function) labels			
ADV	adverbial	√	√
BNF	benefactive	√	√
DIR	direction	√	√
EXT	extent	√	√
LOC	locative	√	√
MNR	manner	√	√
NOM	nominal	√	
PRP	purpose or reason	√	√
TMP	temporal	√	√
CND	condition		√
IJ	interjective		√
VOC	vocative	√	√
Topicalization			
TPC	topicalized	√	√

Table 1: *Function labels in the English and Chinese Penn Treebanks*

bank trees. We show that using the better-motivated methods helps to improve the quality and quantity of training material available to the machine-learning algorithm.

In Section 2 we describe previous approaches to the function labeling task. In Section 3 we present our improved method of obtaining appropriate training material for function labeling. In Section 4 we present experimental results for English and Chinese, and in Section 5 we conclude and suggest possible future research.

2 Previous Work

There are two main approaches to obtaining parse trees with function label information:

- Two-stage systems, where "bare" parse trees are enriched with function labels in a postprocessing step (Blaheta and Charniak, 2000; Jijkoun and de Rijke, 2004; Chrupała and van Genabith, 2006),
- Modifying the parser's internals to output function labels (Musillo and Merlo, 2005; Gabbard et al., 2006).

Blaheta and Charniak (2000) use a probabilistic model with feature dependencies encoded by means of feature trees to add English Penn II Treebank function labels to the output of Charniak's parser. They report an f-score of 87.277% on correctly parsed constituents, and 88.472% on original treebank trees from WSJ section 23.

Jijkoun and de Rijke (2004) describe a method of learning function labels, empty nodes and coindexations from the English Penn II Treebank trees. They transform trees to dependencies and use memory-based learning to transform the dependecy graphs. One of the transformations is node renaming, which adds function labels to parser output. They report an f-score of 88.5% for the task of function tagging on correctly parsed constituents on WSJ section 23.

Chrupała and van Genabith (2006) compare the performance of three machine learning algorithms on function labeling of the Spanish Cast3LB treebank (Civit and Martí, 2004) against the baseline which uses a modified version of Bikel's parser (Bikel, 2002) to directly learn and output function-labeled nodes. They evaluate their results in a task-based setting by using the resulting function-labeled trees to produce LFG f-structures, and report a 2.67% improvement in f-score over the baseline for this task.

Musillo and Merlo (2005) extend the Henderson parser (Henderson, 2003) and model function labels as both expressions of the lexical semantics properties of a constituent and as syntactic elements whose distribution is subject to structural locality constraints. This improves their parsing score and function labeling score on the grammatical and semantic label classes in the English Penn II Treebank.

Gabbard et al. (2006) describe a two stage parser which builds Penn Treebank analyses including both function labels and empty categories and coindexations. Function labeling is performed during the first stage: they modify Bikel's implementation of Collins' parsing model to enable it to output function labels. They report 88.96% f-score on correctly parsed constituents on WSJ section 23.

3 Methods

We use the two-stage architecture, in which the first stage consists of bare constituency parsing using a statistical parsing model and the second stage decorates constituent labels with function labels. The labeler is a machine-learning classification model. Our focus is to investigate ways of improving the performance of the classifier by extracting more and better quality training examples from the available resources.

By improving the quality of training material we mean making it more similar to the instances that the model has to classify during prediction, i.e. we will try to better approximate the standard assumption made in most machine-learning research that instances (in training and test) are *independently and identically distributed* (i.i.d.), in particular, they should be drawn from the same probability distribution.

In the previous two-stage approaches (Blaheta and Charniak, 2000; Jijkoun and de Rijke, 2004; Chrupała

and van Genabith, 2006) this assumption is violated in that the training instances are feature vectors extracted from nodes in the "perfect" parse trees from the treebank, whereas at prediction time the model has to classify instances extracted from nodes in imperfect parser output, which can and does contain a certain proportion of errors (incorrect bracketings or incorrect constituent labels).

We propose to alleviate this issue by using training material which is extracted from the trees obtained by reparsing the training portion of the treebank and using the (imperfect) trees output by the parser rather than the original treebank trees. We still need the original treebank trees in order to assign classes (function labels) to the training instances extracted from parser output. We do this by matching node-spans between automatically parsed trees and gold trees in the training set. We only extract training instances from those nodes in the automatically parsed tree for which there is a node with the same span in the gold tree, from which we can obtain the function label.

3.1 Baseline Method

Our baseline method uses a simple two-stage architecture: constituency parsing, followed by function labeling. The first stage is performed by the constituency parsing model, obtained by training a statistical parser on the training portion of the treebank. The output of this stage, sentences parsed into bare constituency trees, are the input to the second stage component, i.e. the function labeler. The labeler is trained, in the baseline method, on the original "perfect" trees from the training portion of the treebank.

3.1.1 Features

Each node to label is represented as a fixed-length vector of features encoding categorial, configurational and lexical information about the node and its context. We use the following features:

1. Node constituent label
2. Node head word's part of speech tag
3. Node head word
4. Node's head-sister's constituent label
5. Node's head-sister's head word's part of speech
6. Node's head-sister's head word
7. Node's alternative head word's part of speech tag (alternative head is the head of the second child for PPs)
8. Node's alternative head word
9. Node's yield length
10. Node's mother's constituent label
11. Node's grandmother's constituent label
12. Offset to node's head sister

Plus the following:

- Features 1,2,3,7,8,9 for the preceding sister node
- Features 1,2,3,7,8,9 for the following sister node

Those features are binarized (i.e. each feature:value pair is mapped to a new boolean feature), and the examples (i.e. the feature vectors) are used to train a classifier. There is one minor complication: in principle a node can be decorated with more than one function label (although labels belonging to the same group are (usually) mutually exclusive). Thus we could train a separate classifier for each label, or a separate classifier for each label group, or simply treat the label set on the node as an atomic class. In the experiments reported below we used the first method, i.e. we train a separate binary classifier for each function label, and combine their output to add a set of function labels to each node.

3.2 Evaluation Metrics

Evaluating the performance of a function labeling system is not entirely straightforward. Since the constituency trees output by the parser are not identical to the gold-standard treebank trees, one cannot report simple labeling accuracy. Blaheta and Charniak (2000) decide to measure accuracy (for their with-null metric) and f-score (for the no-null metric) over the *correctly parsed* subset of nodes, i.e. those nodes that subtend the correct portion of the string and have the correct constituent label. In this work we use the same metric.

3.3 Training on Parser Output

Using the metric described above, since we are evaluating only the correctly parsed subset of nodes, one might naively expect that the score should be the same for labeling both the parser output and the perfect treebank trees. However, the results reported in (Blaheta and Charniak, 2000) show that the performance is over 1% better for the treebank trees. The authors convincingly explain that the likely cause is that although the focus node to be labeled is correctly parsed, the neighouring context nodes that some features depend on may be incorrect.

This fact serves as our motivation for extracting training examples from treebank sentences parsed by the same parser that is used to parse unseen test data. Our hypothesis is that training instances obtained in this way are going to be more similar to test instances than the ones extracted from perfect treebank trees and thus will better approximate the i.i.d assumption. We expect that the machine learning algorithm will perform better on test instances which are more similar to those used for training; for example it might be able to weight down features which depend on incorrect characteristics of the parse trees, as such features will be less reliable as class predictors.

Our improved training example extraction procedure is as follows: sentences in the training portion of the treebank are reparsed. Then we follow the algorithm presented in Figure 1 to extract training instances. The function INSTANCES returns training instances from a parse tree T given the reference treebank gold tree T' for the same sentence. For each node n in T we check whether there exist one or more nodes with the same span and constituent label in the corresponding T' (line 3)[1]. The function INSTANCE takes the union of the function label sets (FUNCLABELS(n')) found on the nodes in the gold tree T' and returns this set (as a class $\mathcal{C}$) together with the feature vector FEATURES(n) corresponding to node n.

[1] The square bracket notation denotes multisets.

```
1  INSTANCES(T, T') =
2    N ← { NODESPEC(n) | n ∈ NODESET(T') }
3    I ← [ INSTANCE(n, T') | n ∈ NODEMULTISET(T) ∧ NODESPEC(n) ∈ N ]
4    return I
5  INSTANCE(n, T') =
6    C ← ⋃ { FUNCLABELS(n') | n' ∈ NODESET(T') ∧ NODESPEC(n') = NODESPEC(n) }
7    return ⟨FEATURES(n), C⟩
8  NODESPEC(n) = ⟨NODESPAN(n), NODECONSTITUENTLABEL(n)⟩
```

Fig. 1: *Algorithm for extracting training instances from a parser tree T and gold tree T'*

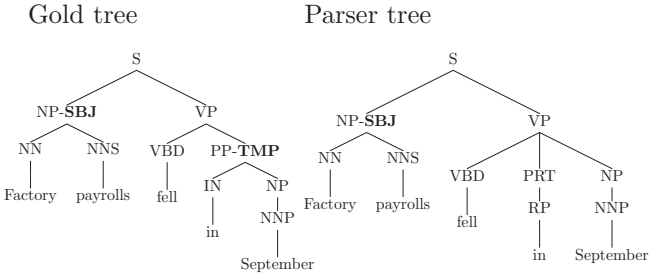

Fig. 2: *Example gold and parser tree*

Figure 2 illustrates this algorithm: in effect we transfer function labels from nodes in the gold tree to matching nodes in the parser tree. Matching nodes are those whose constituent label and span are the same. In the example tree the SBJ function label is tranferred but the TMP is not since there is no matching node in the parser tree due to a parsing error.

A problem with our method as described so far is that we use a constituency parsing model trained on sections 2-21 of WSJ to reparse those same sections so that we can extract training material from them. Obviously it is very likely that the resulting parse trees will be closer to gold trees than will be the case for test sentences taken from WSJ section 23. It would be advisable to extract input for our labeling model from the treebank trees reparsed with parsing models trained on material from which those trees are excluded. We did not do this for the experiments on the English data with Charniak's parser, due to technical difficulties encountered when attempting to retrain this parser. However, for the experiments on the Chinese data with Bikel's parser we did 10-fold-cross-training, that is we divided the training material into 10 parts and parsed each part in turn with the model trained on the remaining 9 parts. We report the results on the Chinese data in section 4.

3.3.1 Instance Similarity

We tried to verify our prediction that the instances extracted using our method would be more similar to test instances. As a simple metric of similarity, we compare instance overlap between the training set and the test set. Instance overlap is the cardinality of the intersection of the multiset of instance feature vectors used for training and the multiset of instance feature vectors used for testing. For multisets defined as tuples (A, f) with the underlying set A and the multiplicity function $f : A \rightarrow \mathbb{N}$ which assigns to each element the number of times it occurs, multiset cardinality is defined as:

$$|(A, f)| = \sum_{a \in A} f(a),$$

and multiset intersection as:

$$(A, f) \cap (B, g) = (A \cap B, a \mapsto min(f(a), g(a))).$$

	Instance count	Overlap
Test	44,113	—
Baseline	741,833	9,067
Reparse	712,973	10,022

Table 2: *Instance counts and instance overlap against test for the English Penn Treebank training set*

We use both the baseline method where examples are extracted from gold trees, and our improved training method to obtain training examples from sections 2-21 of the Wall Street Journal part of the English Penn Treebank and compare both against instances extracted from the parsed sentences taken from section 23. For parsing the test sentences and the training sentences we used the Charniak parser.

Table 2 summarizes the comparison. Even though our method produces a lower total number of instances than the baseline (since we only extract instances from correctly spanning nodes) it still shares 955 instances more with the test set than the baseline.

To further test our conjecture about our method giving better training examples we calculated mean Hamming distance between training examples and test examples. Hamming distance counts the number of features at which two vectors differ:

$$d^h(\mathbf{v}, \mathbf{w}) = \sum_{i=1}^{|\mathbf{v}|} \mathbf{v}_i \neq \mathbf{w}_i \ . \tag{1}$$

We calculate the mean distance between the collection of test instances $\mathbf{T}$ and the collection of training instances $\mathbf{U}$ as:

$$\bar{d^h}(\mathbf{T}, \mathbf{U}) = \frac{1}{|\mathbf{T}| \times |\mathbf{U}|} \sum_{\mathbf{t} \in \mathbf{T}} \sum_{\mathbf{u} \in \mathbf{U}} d^h(\mathbf{t}, \mathbf{u}) \ . \tag{2}$$

As shown in Table 3, against the test set derived from section 23 of WSJ we get mean Hamming distance of 15.1483 for the baseline method and 15.1283 for our method (for comparison the mean distance of the test set against itself is 15.099). According to this metric examples obtained by our method are more similar to test examples.

	Mean distance to Test
Test	15.0999
Baseline	15.1483
Reparse	15.1283

Table 3: *Mean Hamming distance scores for the English Penn Treebank training set*

4 Experimental Results

In this section we present evaluation results on the function labeling task for two datasets:

- Section 23 of the WSJ portion of the English Penn II Treebank, with models trained on data extracted from sections 2-21. Section 22 was used for development. The Charniak parser[2] was used for constituency parsing.
- Articles 271 to 300 of the Penn Chinese Treebank 5, with models trained on data extracted from articles 26 to 270. Articles 1-25 were used for development. We follow (Levy and Manning, 2003) in adopting this test/training/development split. The Bikel parser [3] was used for constituency parsing.

For both datasets we used the LIBSVM library (Chang and Lin, 2001) which implements the Support Vector Machines algorithm (Vapnik, 1998).

4.1 Experiments with the English Penn Treebank

Table 4 summarizes evaluation results for the function labeling task on the English Penn II Treebank. There is a clear increase in f-score over the baseline for our method, which gives a relative error reduction of almost 8.5% over the baseline. The approximate randomization test (Noreen, 1989) with 10^6 shuffles obtained a p-value of 10^{-7} for the baseline versus our method, showing that the improvement is statistically significant.

Our results (91.47% f-score) are the best scores published to date on the function labeling task evaluated on parser output on the section 23 of WSJ: 87.27% in (Blaheta and Charniak, 2000), 88.5% in (Jijkoun and de Rijke, 2004) and 88.96% in (Gabbard et al., 2006)[4]

Table 5 shows the performance broken down per function label. Although performance on three labels (LOC, LGS and PRP) drops, the rest of the labels show the same score or benefit from our training method.

4.2 Experiments with the Penn Chinese Treebank

For the Chinese Treebank we performed experiments evaluating the impact of using our basic method and also the variation with cross-training on the function labeling task.

The results we obtained are somewhat contradictory: we saw an improvement in performance using both on the development set (articles 1-25), but on the test set (articles 271-300) the basic method shows practically no improvement whereas cross-training actually leads to results worse than for the baseline.

Table 6 shows the results for the development set which are consistent with our findings so far: our method outperforms the baseline by 0.18%. Additionally, we observe that adding cross-training produces a further increase in the f-score of 0.3%.

However, as can be seen in Table 7, for the test set our predictions are not borne out: with cross-training we actually obtain a lower score than the baseline (−0.32%); without cross-training the score is only marginally better than the baseline (+0.01%).

We performed an approximate randomization test for both the development set and the set, testing the baseline against our method with cross-training. For the development set we obtained a p-value of 0.13; for the test set the p-value was 0.08 — this suggests that neither the improvement for the development set nor the decrease in f-score for the test set are statistically significant.

It would be interesting to repeat our experiments for Chinese using larger data sets. There are two reasons why we want to do that. First, testing on a larger test set would offer a higher confidence in the significance of the observed performance scores. Second, we suspect that one reason that our approach did not

	Precision	Recall	F-score
Baseline	92.28	89.14	90.68
Reparse	93.07	89.92	91.47

Table 4: *Function labeling evaluation on parser output for WSJ section 23*

Label	Freq. in test	Baseline	Reparse
SBJ	4148	98.27	98.27
TMP	1303	91.19	91.52
PRD	1025	68.35	91.26
LOC	1024	89.45	89.06
CLR	635	68.98	68.93
ADV	419	85.98	89.36
DIR	293	68.98	71.20
TPC	267	86.50	96.02
PRP	207	68.35	67.95
NOM	199	95.02	95.58
MNR	178	76.12	77.62
LGS	166	88.10	88.10
EXT	105	87.72	88.24
TTL	61	74.42	74.42
HLN	52	18.18	26.23
DTV	19	66.67	66.67
PUT	10	66.67	66.67
CLF	3	—	—
BNF	2	—	—
VOC	1	—	—

Table 5: *Per-tag performance of baseline and when training on reparsed trees*

[2] Available at `ftp://ftp.cs.brown.edu/pub/nlparser/`

[3] Available at `http://www.cis.upenn.edu/~dbikel/software.html#stat-parser`

[4] Not all of those scores are exactly comparable to ours or to each other. The score in (Jijkoun and de Rijke, 2004) is on trees transformed into dependencies. Gabbard et al. (2006) use Bikel's parser to produce the trees whereas we use Charniak's.

	Precision	Recall	F-score
Baseline	88.35	84.64	86.46
Reparse	88.54	84.82	86.64
Reparse + x-train	89.11	84.88	86.94

Table 6: *Function labeling evaluation for the CTB on the parser output for the development set*

	Precision	Recall	F-score
Baseline	91.46	90.13	90.79
Reparse	91.39	90.23	90.80
Reparse + x-train	91.53	89.43	90.47

Table 7: *Function labeling evaluation for the CTB on the parser output for the test set*

show consistent improvement across both the development set and the test set might be related to the relatively small amount of training material we used, for both training the parser and the function labeling model. Thus parse quality is rather low, and since we only exploit correctly parsed nodes in extracting training instances for labeling, the amount of training data available decreases even further. We also suspect that parse quality for Chinese may be lower than for English even while holding training set size constant, reflecting the smaller amount of work which has gone into research on parsing Chinese. Testing those conjectures remains an area for future investigation.

It remains to be seen whether using our approach with training sets comparable in size with the one we used for English would more show more consistent benefits for Chinese.

5 Conclusions and Future Work

We have presented a method to perform training in a sound manner in the two-stage function labeling model and investigated the impact of our proposal on the function labeling task. Our approach improves the similarity of the training material to the test instances as measured by instance overlap and mean Hamming distance.

We have consitently found substantial statistically significant improvements on the English Penn Treebank data, and a more mixed picture for the Chinese Penn Treebank sentences. We would like to better understand what factors influence the effect of our proposed training methods on function labeling performance: we plan to study this issue in more detail in future.

It should also be possible to apply our findings to other tasks where training examples are typically extracted from perfect trees whereas the test data is produced automatically and contains errors. Using parser output instead and exploiting several of the most probable trees could be beneficial in those situations.

Acknowledgements

We gratefully acknowledge support from Science Foundation Ireland grant 04/IN/I527 for the research reported in this paper.

References

Bies, Ann (1995). Bracketing guidelines for Treebank II style Penn treebank project. Technical report, University of Pennsylvania.

Bikel, Dan (2002). Design of a multi-lingual, parallel-processing statistical parsing engine. In *Human Language Technology Conference (HLT)*. San Diego, CA, USA.

Blaheta, Don and Charniak, Eugene (2000). Assigning function tags to parsed text. In *Proceedings of the first conference on North American chapter of the Association for Computational Linguistics*, pages 234–240. San Francisco, CA, USA.

Chang, Chih-Chung and Lin, Chih-Jen (2001). LIBSVM: a library for Support Vector Machines (version 2.31).

Charniak, Eugene (2000). A maximum-entropy-inspired parser. In *Proceedings of the first conference on North American chapter of the Association for Computational Linguistics*, pages 132–139. San Francisco, CA, USA.

Chrupała, Grzegorz and van Genabith, Josef (2006). Using machine-learning to assign function labels to parser output for Spanish. In *Proceedings of the COLING/ACL 2006 Main Conference Poster Sessions*, pages 136–143. Sydney, Australia.

Civit, Montserrat and Martí, Maria Antonia (2004). Building Cast3LB: A Spanish treebank. *Research on Language and Computation*, 2(4):549–574.

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

Gabbard, Ryan, Kulick, Seth, and Marcus, Mitchell (2006). Fully parsing the Penn treebank. In *Proceedings of the Human Language Technology Conference of the NAACL, Main Conference*, pages 184–191. New York City, USA.

Henderson, James (2003). Inducing history representations for broad coverage statistical parsing. In *NAACL '03: Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology*, pages 24–31. Morristown, NJ, USA.

Jijkoun, Valentin and de Rijke, Maarten (2004). Enriching the output of a parser using memory-based learning. In *Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics*, pages 311–318. Barcelona, Spain.

Levy, Roger and Manning, Christopher (2003). Is it harder to parse Chinese, or the Chinese treebank? In *ACL '03: Proceedings of the 41st Annual Meeting on Association for Computational Linguistics*, pages 439–446. Morristown, NJ, USA.

Musillo, Gabriele and Merlo, Paola (2005). Lexical and structural biases for function parsing. In *Proceedings of the Ninth International Workshop on Parsing Technology*, pages 83–92. Vancouver, British Columbia.

Noreen, Eric W. (1989). *Computer intensive methods for testing hypotheses*. A Wiley-Interscience Publication, New York.

Vapnik, Vladimir N. (1998). *Statistical Learning Theory*. Wiley-Interscience.

Xue, Nianwen and Xia, Fei (2000). The bracketing guidelines for the Penn Chinese treebank. Technical report, University of Pennsylvania.