POS Disambiguation and Unknown Word Guessing with Decision Trees

Giorgos S. Orphanos
Computer Engineering & Informatics Dept.
and Computer Technology Institute
University of Patras
26500 Rion, Patras, Greece
georfan@cti.gr

Dimitris N. Christodoulakis
Computer Engineering & Informatics Dept.
and Computer Technology Institute
University of Patras
26500 Rion, Patras, Greece
dxri@cti.gr

Abstract

This paper presents a decision-tree approach to the problems of part-of-speech disambiguation and unknown word guessing as they appear in Modern Greek, a highly inflectional language. The learning procedure is tag-set independent and reflects the linguistic reasoning on the specific problems. The decision trees induced are combined with a high-coverage lexicon to form a tagger that achieves 93,5% overall disambiguation accuracy.

1 Introduction

Part-of-speech (POS) taggers are software devices that aim to assign unambiguous morphosyntactic tags to words of electronic texts. Although the hardest part of the tagging process is performed by a computational lexicon, a POS tagger cannot solely consist of a lexicon due to: (i) morphosyntactic ambiguity (e.g., 'love' as verb or noun) and (ii) the existence of unknown words (e.g., proper nouns, place names, compounds, etc.). When the lexicon can assure high coverage, unknown word guessing can be viewed as a decision taken upon the POS of open-class words (i.e., Noun, Verb, Adjective, Adverb or Participle).

Towards the disambiguation of POS tags, two main approaches have been followed. On one hand, according to the linguistic approach, experts encode handcrafted rules or constraints based on abstractions derived from language paradigms (usually with the aid of corpora) (Green and Rubin, 1971; Voutilainen 1995). On the other hand, according to the data-driven approach, a frequency-based language model is acquired from corpora and has the forms of n-grams (Church, 1988; Cutting *et al.*, 1992), rules (Hindle, 1989; Brill, 1995), decision trees (Cardie, 1994; Daelemans *et al.*, 1996) or neural networks (Schmid, 1994).

In order to increase their robustness, most POS taggers include a *guesser*, which tries to extract the POS of words not present in the lexicon. As a common strategy, POS guessers examine the endings of unknown words (Cutting *et al.* 1992) along with their capitalization, or consider the distribution of unknown words over specific parts-of-speech (Weischedel *et al.*, 1993). More sophisticated guessers further examine the prefixes of unknown words (Mikheev, 1996) and the categories of contextual tokens (Brill, 1995; Daelemans *et al.*, 1996).

This paper presents a POS tagger for Modern Greek (M. Greek), a highly inflectional language, and focuses on a data-driven approach for the induction of decision trees used as disambiguation/guessing devices. Based on a high-coverage[1] lexicon, we prepared a tagged corpus capable of showing off the behavior of all POS ambiguity schemes present in M. Greek (e.g., Pronoun-Clitic-Article, Pronoun-Clitic, Adjective-Adverb, Verb-Noun, etc.), as well as the characteristics of unknown words. Consequently, we used the corpus for the induction of decision trees, which, along with

[1] At present, the lexicon is capable of assigning full morphosyntactic attributes (i.e., POS, Number, Gender, Case, Person, Tense, Voice, Mood) to ~870.000 Greek word-forms.

the lexicon, are integrated into a robust POS tagger for M. Greek texts.

The disambiguating methodology followed is highly influenced by the Memory-Based Tagger (MBT) presented in (Daelemans *et al.*, 1996). Our main contribution is the successful application of the decision-tree methodology to M. Greek with three improvements/customizations: (i) injection of **linguistic bias** to the learning procedure, (ii) formation of **tag-set independent** training patterns, and (iii) handling of **set-valued** features.

2 Tagger Architecture

Figure 1 illustrates the functional components of the tagger and the order of processing:

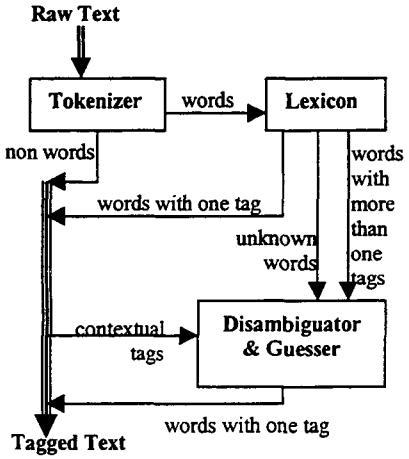

Figure 1. Tagger Architecture

Raw text passes through the Tokenizer, where it is converted to a stream of tokens. Non-word tokens (e.g., punctuation marks, numbers, dates, etc.) are resolved by the Tokenizer and receive a tag corresponding to their category. Word tokens are looked-up in the Lexicon and those found receive one or more tags. Words with more than one tags and those not found in the Lexicon pass through the Disambiguator/Guesser, where the contextually appropriate tag is decided/guessed.

The Disambiguator/Guesser is a 'forest' of decision trees, one tree for each ambiguity scheme present in M. Greek and one tree for unknown word guessing. When a word with two or more tags appears, its ambiguity scheme is identified. Then, the corresponding decision tree is selected, which is traversed according to the values of morphosyntactic features extracted from contextual tags. This traversal returns the contextually appropriate POS. The ambiguity is resolved by eliminating the tag(s) with different POS than the one returned by the decision tree. The POS of an unknown word is guessed by traversing the decision tree for unknown words, which examines contextual features along with the word ending and capitalization and returns an open-class POS.

3 Training Sets

For the study and resolution of lexical ambiguity in M. Greek, we set up a corpus of 137.765 tokens (7.624 sentences), collecting sentences from student writings, literature, newspapers, and technical, financial and sports magazines. We made sure to adequately cover all POS ambiguity schemes present in M. Greek, without showing preference to any scheme, so as to have an objective view to the problem. Subsequently, we tokenized the corpus and inserted it into a database and let the lexicon assign a morphosyntactic tag to each word-token. We did not use any specific tag-set; instead, we let the lexicon assign to each known word all morphosyntactic attributes available. Table 1 shows a sample sentence after this initial tagging (symbolic names appearing in the tags are explained in Appendix A).

Table 1. An example-sentence from the tagged corpus

Sentence #	Token #	Token	English translation	Automatic Tag (assigned by the lexicon)	Manual Tag
2638	1	Οι	*The*	Art(MscFemSglNom)	
2638	2	απαντήσεις	*answers*	Vrb(_B_SglActPstSjv + _B_SglActFutInd) + Nnn(FemPlrNomAccVoc)	Nnn
2638	3	του	*of*	Prn(_C_MscNtrSngGen) + Clt + Art(MscNtrSngGen)	Art
2638	4	κ.	*Mr.*	Abr	
2638	5	Παπαδόπουλου	*Papadopoulos*	"ου" Cap Nnn + Vrb + Adj + Pcp + Adv	Nnn
2638	6	ήταν	*were*	Vrb(_C_SglPlrIctInd)	
2638	7	σαφείς	*clear*	Adj(MscFemPlrNomAccVoc)	
2638	8	.	.	'.'	

Table 2. A fragment from the training set Verb-Noun

Example ID	Tag_{i-1}	Tag_i	Tag_{i+1}	Manual Tag_i
1	Adj(FemSglNomAcc)	Vrb(_B_SglPntActImv) + Nnn(FemSglNomAccVoc)	Prn(_C_FemSglGen) + Clt + Art(FemSglGen)	Nnn
2	Adj(FemSglNomAcc)	Vrb(_C_SglPntFcsActIndSjv) + Nnn(FemSglNomAccVoc)	Nnn(NtrSglNomAccVoc)	Nnn
3	Nnn + Vrb + Adj + Pcp + Adv	Vrb(_B_SglFutPstActIndSjv) + Nnn(FemPlrNomAccVoc)	','	Nnn
4	Prn(_A_SglGenAcc) + Pps	Vrb(_B_SglFutPstActIndSjv) + Nnn(NtrSglPlrNomGenAccVoc)	Adj(FemSglNomAccVoc)	Vrb
5	Art(FemPlrAcc)	Vrb(_B_SglFutPstActIndSjv) + Nnn(FemPlrNomAccVoc)	Prn(_C_MscNtrSglGen) + Clt + Art(MscNtrSglGen)	Nnn
6	Pcl	Vrb(_B_SglPntFcsFutPstActIndSjv) + Nnn(MscSglNom)	Prn(_A_SglGenAcc) + Pps	Vrb
7		Vrb(_B_SglFutPstActIndSjv) + Nnn(FemPlrNomAccVoc)	Vrb(_C_PlrPntFcsActIndSjv)	Nnn
8	Pcl	Vrb(_B_SglFutPstActIndSjv) + Nnn(NtrSglPlrNomGenAccVoc)	'»'	Vrb
9	Adj(FemSglNomAcc)	Vrb(_C_SglPntFcsActIndSjv) + Nnn(FemSglNomAccVoc)	Art(MscSglAcc + NtrSglNomAcc)	Nnn
10	Pcl + Adv	Vrb(_B_SglPntFcsFutPstActIndSjv) + Nnn(MscSglNom)	','	Vrb

To words with POS ambiguity (e.g., tokens #2 and #3 in Table 1) we manually assigned their contextually appropriate POS. To unknown words (e.g., token #5 in Table 1), which by default received a disjunct of open-class POS labels, we manually assigned their real POS and declared explicitly their inflectional ending.

At a next phase, for all words relative to a specific ambiguity scheme or for all unknown words, we collected from the tagged corpus their automatically and manually assigned tags along with the automatically assigned tags of their neighboring tokens. This way, we created a training set for each ambiguity scheme and a training set for unknown words. Table 2 shows a 10-example fragment from the training set for the ambiguity scheme Verb-Noun. For reasons of space, Table 2 shows the tags of only the previous (column Tag_{i-1}) and next (column Tag_{i+1}) tokens in the neighborhood of an ambiguous word, whereas more contextual tags actually comprise a training example. A training example also includes the manually assigned tag (column *Manual* Tag_i) along with the automatically assigned tag[2] (column Tag_i) of the ambiguous word. One can notice that some contextual tags are missing (e.g., Tag_{i-1} of Example 7; the ambiguous word is the first in the sentence), or some contextual tags may exhibit POS ambiguity (e.g., Tag_{i+1} of Example 1), an incident implying that the learner must learn from incomplete/ambiguous examples, since this is the case in real texts.

If we consider that a tag encodes 1 to 5 morphosyntactic features, each feature taking one or a disjunction of 2 to 11 values, then the total number of different tags counts up to several hundreds[3]. This fact prohibits the feeding of the training algorithms with patterns that have the form: (Tag_{i-2}, Tag_{i-1}, Tag_i, Tag_{i+1}, Manual_Tag_i), which is the case for similar systems that learn POS disambiguation (e.g., Daelemans *et al.*, 1996). On the other hand, it would be inefficient (yielding to information loss) to generate a simplified tag-set in order to reduce its size. The 'what the training patterns should look like' bottleneck was surpassed by assuming a set of functions that extract from a tag the value(s) of specific features, e.g.:

Gender(Art(MscSglAcc + NtrSglNomAcc)) = Msc + Ntr

With the help of these functions, the training examples shown in Table 2 are interpreted to patterns that look like:

(**POS**(Tag_{i-2}), **POS**(Tag_{i-1}), **Gender**(Tag_i), **POS**(Tag_{i+1}), **Gender**(Tag_{i+1}), Manual_Tag_i),

[2] In case the learner needs to use morphosyntactic information of the word being disambiguated.

[3] The words of the corpus received from the lexicon 690 different tags having the form shown in Table 2.

that is, a sequence of feature-values extracted from the previous/current/next tags along with the manually assigned POS label.

Due to this transformation, two issues automatically arise: (a) A feature-extracting function may return more than one feature value (as in the **Gender(...)** example); consequently, the training algorithm should be capable of handling set-valued features. (b) A feature-extracting function may return no value, e.g. **Gender(**`Vrb(_C_PlrPntActIndSjv)`**)** = `None`, thus we added an extra value –the value `None`– to each feature[4].

To summarize, the training material we prepared consists of: (a) a set of training examples for each ambiguity scheme and a set of training examples for unknown words[5], and (b) a set of features accompanying each example-set, denoting which features (extracted from the tags of training examples) will participate in the training procedure. This configuration offers the following advantages:

1. A training set is examined only for the features that are relative to the corresponding ambiguity scheme, thus addressing its idiosyncratic needs.
2. What features are included to each feature-set depends on the linguistic reasoning on the specific ambiguity scheme, introducing this way **linguistic bias** to the learner.
3. The learning is **tag-set independent**, since it is based on specific features and not on the entire tags.
4. The learning of a particular ambiguity scheme can be fine-tuned by including new features or excluding existing features from its feature-set, without affecting the learning of the other ambiguity schemes.

4 Decision Trees

4.1 Tree Induction

In the previous section, we stated the use of linguistic reasoning for the selection of feature-sets suitable to the idiosyncratic properties of the corresponding ambiguity schemes.

Formally speaking, let **FS** be the feature-set attached to a training set **TS**. The algorithm used to transform **TS** into a decision tree belongs to the TDIDT (Top Down Induction of Decision Trees) family (Quinlan, 1986). Based on the divide and conquer principle, it selects the *best* $\mathbf{F_{best}}$ feature from **FS**, partitions **TS** according to the values of $\mathbf{F_{best}}$ and repeats the procedure for each partition excluding $\mathbf{F_{best}}$ from **FS**, continuing recursively until all (or the majority of) examples in a partition belong to the same class **C** or no more features are left in **FS**.

During each step, in order to find the feature that makes the best prediction of class labels and use it to partition the training set, we select the feature with the highest *gain ratio*, an information-based quantity introduced by Quinlan (1986). The gain ratio metric is computed as follows:

Assume a training set **TS** with patterns belonging to one of the classes $\mathbf{C_1}$, $\mathbf{C_2}$, ... $\mathbf{C_k}$. The average information needed to identify the class of a pattern in **TS** is:

$$info(\mathbf{TS}) = -\sum_{j=1}^{k} \frac{freq(\mathbf{C_j}, \mathbf{TS})}{|\mathbf{TS}|} \times \log_2 \left(\frac{freq(\mathbf{C_j}, \mathbf{TS})}{|\mathbf{TS}|}\right)$$

Now consider that **TS** is partitioned into $\mathbf{TS_1}$, $\mathbf{TS_2}$, ... $\mathbf{TS_n}$, according to the values of a feature **F** from **FS**. The average information needed to identify the class of a pattern in the partitioned **TS** is:

$$info_F(\mathbf{TS}) = \sum_{i=1}^{n} \frac{|\mathbf{TS_i}|}{|\mathbf{TS}|} \times info(\mathbf{TS_i})$$

The quantity:

$$gain(\mathbf{F}) = info(\mathbf{TS}) - info_F(\mathbf{TS})$$

measures the information relevant to classification that is **gained** by partitioning **TS** in accordance with the feature **F**. *Gain ratio* is a normalized version of information *gain*:

$$gain\ ratio(\mathbf{F}) = \frac{gain(\mathbf{F})}{split\ info(\mathbf{F})}$$

Split info is a necessary normalizing factor, since *gain* favors features with many values, and represents the potential information generated by dividing **TS** into **n** subsets:

$$split\ info(\mathbf{F}) = -\sum_{i=1}^{n} \frac{|\mathbf{TS_i}|}{|\mathbf{TS}|} \times \log_2 \left(\frac{|\mathbf{TS_i}|}{|\mathbf{TS}|}\right)$$

[4] e.g.: Gender = {**Masculine**, **Feminine**, **Neuter**, **None**}.

[5] The training examples for unknown words, except contextual tags, also include the capitalization feature and the suffixes of unknown words.

Taking into consideration the formula that computes the *gain ratio*, we notice that the best feature is the one that presents the minimum **entropy** in predicting the class labels of the training set, provided the information of the feature is not split over its values.

The recursive algorithm for the decision tree induction is shown in Figure 2. Its parameters are: a node **N**, a training set **TS** and a feature set **FS**. Each node constructed, in a top-down left-to-right fashion, contains a default class label **C** (which characterizes the path constructed so far) and if it is a non-terminal node it also contains a feature **F** from **FS** according to which further branching takes place. Every value v_i of the feature **F** tested at a non-terminal node is accompanied by a pattern subset TS_i (i.e., the subset of patterns containing the value v_i). If two or more values of **F** are found in a training pattern (set-valued feature), the training pattern is directed to all corresponding branches. The algorithm is initialized with a root node, the entire training set and the entire feature set. The root node contains a dummy[6] feature and a blank class label.

```
InduceTree( Node N , TrainingSet TS , FeatureSet FS )
Begin
  For each value vi of the feature F tested by node N Do
  Begin
    Create the subset TSi and assign it to vi;
    If TSi is empty Then continue; /* goto For */
    If all patterns in TSi belong to the same class C Then
      Create under vi a leaf node N' with label C;
    Else
    Begin
      Find the most frequent class C in TSi;
      If FS is empty Then
        Create under vi a leaf node N' with label C;
      Else
      Begin
        Find the feature F' with the highest gain ratio;
        Create under vi a non-terminal node N' with
        label C and set N' to test F';
        Create the feature subset FS' = FS – {F'};
        InduceTree( N' , TSi , FS' );
      End
    End
  End
End
```

Figure 2. Tree-Induction Algorithm

[6] The dummy feature contains the sole value None.

4.2 Tree Traversal

Each tree node, as already mentioned, contains a class label that represents the 'decision' being made by the specific node. Moreover, when a node is not a leaf, it also contains an ordered list of values corresponding to a particular feature tested by the node. Each value is the origin of a subtree hanging under the non-terminal node. The tree is traversed from the root to the leaves. Each non-terminal node tests one after the other its feature-values over the testing pattern. When a value is found, the traversal continues through the subtree hanging under that value. If none of the values is found or the current node is a leaf, the traversal is finished and the node's class label is returned. For the needs of the POS disambiguation/guessing problem, tree nodes contain POS labels and test morphosyntactic features. Figure 3 illustrates the tree-traversal algorithm, via which disambiguation/guessing is performed. The lexical and/or contextual features of an ambiguous/unknown word constitute a testing pattern, which, along with the root of the decision tree corresponding to the specific ambiguity scheme, are passed to the tree-traversal algorithm.

```
ClassLabel TraverseTree( Node N , TestingPattern P )
Begin
  If N is a non-terminal node Then
    For each value vi of the feature F tested by N Do
      If vi is the value of F in P Then
      Begin
        N' = the node hanging under vi;
        Return TraverseTree( N' , P );
      End
  Return the class label of N;
End
```

Figure 3. Tree-Traversal Algorithm

4.3 Subtree Ordering

The tree-traversal algorithm of Figure 3 can be directly implemented by representing the decision tree as nested if-statements (see Appendix B), where each block of code following an if-statement corresponds to a subtree. When an if-statement succeeds, the control is transferred to the inner block and, since there is no backtracking, no other feature-values of the same level are tested. To classify a pattern with a set-valued feature, only one value

from the set steers the traversal; the value that is tested first. A fair policy suggests to test first the most important (probable) value, or, equivalently, to test first the value that leads to the subtree that gathered more training patterns than sibling subtrees. This policy can be incarnated in the tree-traversal algorithm if we previously sort the list of feature-values tested by each non-terminal node, according to the algorithm of Figure 4, which is initialized with the root of the tree.

```
OrderSubtrees( Node N )
Begin
  If N is a non-terminal node Then
  Begin
    Sort the feature-values and sub-trees of node N
    according to the number of training patterns each
    sub-tree obtained;
    For each child node N' under node N Do
      OrderSubtrees( N' );
  End
End
```

Figure 4. Subtree-Ordering Algorithm

This ordering has a nice side-effect: it increases the classification speed, as the most probable paths are ranked first in the decision tree.

4.4 Tree Compaction

A tree induced by the algorithm of Figure 2 may contain many redundant paths from root to leaves; paths where, from a node and forward, the same decision is made. The tree-traversal definitely speeds up by eliminating the tails of the paths that do not alter the decisions taken thus far. This compaction does not affect the performance of the decision tree. Figure 5 illustrates the tree-compaction algorithm, which is initialized with the root of the tree.

```
CompactTree( Node N )
Begin
  For each child node N' under node N Do
  Begin
    If N' is a leaf node Then
    Begin
      If N' has the same class label with N Then
        Delete N';
    End
    Else
    Begin
      CompactTree( N' );
      If N' is now a leaf node And
      has the same class label with N Then
        Delete N';
    End
  End
End
```

Figure 5. Tree-Compaction Algorithm

Table 3. Statistics and Evaluation Measurements

POS Ambiguity Schemes	(1) % occurrence in the corpus	(2) % contribution to POS ambiguity	(3) %error most frequent POS	(4) %error decision trees
Pronoun-Article	7,13	34,19	14,5	1,96
Pronoun-Article-Clitic	4,70	22,54	39,1	4,85
Pronoun-Preposition	2,14	10,26	12,2	1,35
Adjective-Adverb	1,53	7,33	31,1	13,4
Pronoun-Clitic	1,41	6,76	38,0	5,78
Preposition-Particle-Conjunction	1,02	4,89	20,8	8,94
Verb-Noun	0,52	2,49	12,1	6,93
Adjective-Adverb-Noun	0,51	2,44	51,0	30,4
Adjective-Noun	0,46	2,20	38,2	18,2
Particle-Conjunction	0,39	1,87	1,38	1,38
Adverb-Conjunction	0,36	1,72	22,8	18,1
Pronoun-Adverb	0,34	1,63	4,31	4,31
Verb-Adverb	0,06	0,28	16,8	1,99
Other	0,29	1,39	30,1	12,3
Total POS Ambiguity	20,85		24,1	5,48
Unknown Words	2,53		38,6	15,8
Totals	**23,38**		**25,6**	**6,61**

5 Evaluation

To evaluate our approach, we first partitioned the datasets described in Section 3 into training and testing sets according to the 10-fold cross-validation method[7]. Then, (a) we found the most frequent POS in each training set and (b) we induced a decision tree from each training set. Consequently, we resolved the ambiguity of the testing sets with two methods: (a) we assigned the most frequent POS acquired from the corresponding training sets and (b) we used the induced decision trees.

Table 3 concentrates the results of our experiments. In detail: Column (1) shows in what percentage the ambiguity schemes and the unknown words occur in the corpus. The total problematic word-tokens in the corpus are 23,38%. Column (2) shows in what percentage each ambiguity scheme contributes to the total POS ambiguity. Column (3) shows the error rates of method (a). Column (4) shows the error rates of method (b).

To compute the total POS disambiguation error rates of the two methods (24,1% and 5,48% respectively) we used the contribution percentages shown in column (2).

6 Discussion and Future Goals

We have shown a uniform approach to the dual problem of POS disambiguation and unknown word guessing as it appears in M. Greek, reinforcing the argument that "machine-learning researchers should become more interested in NLP as an application area" (Daelemans *et al.*, 1997). As a general remark, we argue that the linguistic approach has good performance when the knowledge or the behavior of a language can be defined explicitly (by means of lexicons, syntactic grammars, etc.), whereas empirical (corpus-based statistical) learning should apply when exceptions, complex interaction or ambiguity arise. In addition, there is always the opportunity to bias empirical learning with linguistically motivated parameters, so as to meet the needs of the specific language problem. Based on these statements, we combined a high-coverage lexicon and a set of empirically induced decision trees into a POS tagger achieving ~5,5% error rate for POS disambiguation and ~16% error rate for unknown word guessing.

The decision-tree approach outperforms both the naive approach of assigning the most frequent POS, as well as the ~20% error rate obtained by the n-gram tagger for M. Greek presented in (Dermatas and Kokkinakis, 1995).

Comparing our tree-induction algorithm and IGTREE, the algorithm used in MBT (Daelemans *et al.*, 1996), their main difference is that IGTREE produces oblivious decision trees by supplying an a priori ordered list of best features instead of re-computing the best feature during each branching, which is our case. After applying IGTREE to the datasets described in Section 3, we measured similar performance (~7% error rate for disambiguation and ~17% for guessing). Intuitively, the global search for best features performed by IGTREE has similar results to the local searches over the fragmented datasets performed by our algorithm.

Our goals hereafter aim to cover the following:

- Improve the POS tagging results by: a) finding the optimal feature set for each ambiguity scheme and b) increasing the lexicon coverage.
- Analyze why IGTREE is still so robust when, obviously, it is built on less information.
- Apply the same approach to resolve Gender, Case, Number, etc. ambiguity and to guess such attributes for unknown words.

[7] In this method, a dataset is partitioned 10 times into 90% training material and 10% testing material. Average accuracy provides a reliable estimate of the generalization accuracy.

References

Brill E. (1995). Transformation-Based Error-Driven Learning and Natural Language Processing: A Case Study in Part of Speech Tagging. *Computational Linguistics*, 21(4), 543-565.

Cardie C. (1994). Domain-Specific Knowledge Acquisition for Conceptual Sentence Analysis. *Ph.D. Thesis*, University of Massachusetts, Amherst, MA.

Church K. (1988). A Stochastic parts program and noun phrase parser for unrestricted text. In

Proceedings of 2nd Conference on Applied Natural Language Processing, Austin, Texas.

Cutting D., Kupiec J., Pederson J. and Sibun P. (1992). A practical part-of-speech tagger. In *Proceedings of 3rd Conference on Applied Natural Language Processing*, Trento, Italy.

Daelemans W., Zavrel J., Berck P. and Gillis S. (1996). MBT: A memory-based part of speech tagger generator. In *Proceedings of 4th Workshop on Very Large Corpora*, ACL SIGDAT, 14-27.

Daelemans W., Van den Bosch A. and Weijters A. (1997). Empirical Learning of Natural Language Processing Tasks. In W. Daelemans, A. Van den Bosch, and A. Weijters (eds.) *Workshop Notes of the ECML/Mlnet Workshop on Empirical Learning of Natural Language Processing Tasks*, Prague, 1-10.

Dermatas E. and Kokkinakis G. (1995). Automatic Stochastic Tagging of Natural Language Texts. *Computational Linguistics*, 21(2), 137-163.

Greene B. and Rubin G. (1971). Automated grammatical tagging of English. Department of Linguistics, Brown University.

Hindle D. (1989). Acquiring disambiguation rules from text. In *Proceedings of ACL'89*.

Quinlan J. R. (1986). Induction of Decision Trees. *Machine Learning*, 1, 81-106.

Mikheev A. (1996). Learning Part-of-Speech Guessing Rules from Lexicon: Extension to Non-Concatenative Operations. In *Proceedings of COLING'96*.

Schmid H. (1994) Part-of-speech tagging with neural networks. In *Proceedings of COLING'94*.

Voutilainen A. (1995). A syntax-based part-of-speech analyser. In *Proceedings of EACL'95*.

Weischedel R., Meteer M., Schwartz R., Ramshaw L. and Palmucci J. (1993). Coping with ambiguity and unknown words through probabilistic models. *Computational Linguistics*, 19(2), 359-382.

Appendix A: Feature Values/Shortcuts

Part-Of-Speech = {**Article**/Art, **Noun**/Nnn, **Adj**ective/Adj, **Pronoun**/Prn, **Verb**/Vrb, **Participle**/Pcp, **Adv**erb/Adv, **Conj**unction/Cnj, **Preposition**/Pps, **Particle**/Pcl, **Clitic**/Clt}
Number = {**Sing**ular/Sng, **Plu**ral/Plu}
Gender = {**Masc**uline/Msc, **Fem**inine/Fem, **Neuter**/Ntr}
Case = {**Nom**inative/Nom, **Gen**itive/Gen, **Dat**ive/Dat, **Acc**usative/Acc, **Voc**ative/Voc}
Person = {First/_A_, Second/_B_, Third/_C_}
Tense = {**Present**/Pnt, **Future**/Fut, Future **Perfect**/Fpt, Future **Continuous**/Fcs, **Past**/Pst, **Present Perfect**/Pnp, **Past Perfect**/Psp}
Voice = {**Active**/Act, **Passive**/Psv}
Mood = {**Ind**icative/Ind, **Imp**erative/Imv, **Subj**anctive/Sjv}
Capitalization = {**Cap**ital/Cap}

Appendix B: A decision tree for the scheme Adverb-Adjective

```
/* 'disamb_AdvAdj.c' file, automatically generated from a
training corpus */

#include "../tagger/tagger.h"

int disamb_AdvAdj(void *TL) /* TL means 'Token List' */
{
 if(POS(TL, -1, Vrb)) /*-1: previous token */
  if(POS(TL, 1, Nnn)) return Adj; /*+1: next token */
  else return Adv;
 else if(POS(TL, -1, Prn))
  if(POS(TL, 1, None)) return Adv;
  else if(POS(TL, 1, Pps)) return Adv;
  else if(POS(TL, 1, Pcp)) return Adv;
  else return Adj;
 else if(POS(TL, -1, Art)) return Adj;
 else if(POS(TL, -1, None))
  if(POS(TL, 1, Nnn)) return Adj;
  else return Adv;
 else if(POS(TL, -1, Cnj))
  if(POS(TL, 1, Nnn)) return Adj;
  else return Adv;
 else if(POS(TL, -1, Adv))
  if(POS(TL, 1, Nnn)) return Adj;
  else if(POS(TL, 1, Adv)) return Adj;
  else return Adv;
 else if(POS(TL, -1, Adj))
  if(POS(TL, 1, Cnj)) return Adv;
  else if(POS(TL, 1, Pcp)) return Adv;
  else return Adj;
 else if(POS(TL, -1, Nnn))
  if(POS(TL, 1, Nnn)) return Adj;
  else if(POS(TL, 1, Exc)) return Adj;
  else return Adv;
 else if(POS(TL, -1, Pps))
  if(POS(TL, 1, Prn)) return Adv;
  else if(POS(TL, 1, None)) return Adv;
  else if(POS(TL, 1, Art)) return Adv;
  else if(POS(TL, 1, Pcl)) return Adv;
  else if(POS(TL, 1, Clt)) return Adv;
  else if(POS(TL, 1, Vrb)) return Adv;
  else if(POS(TL, 1, Pps)) return Adv;
  else if(POS(TL, 1, Pcp)) return Adv;
  else return Adj;
 else if(POS(TL, -1, Pcl))
  if(POS(TL, 1, Nnn)) return Adj;
  else if(POS(TL, 1, Adj)) return Adj;
  else return Adv;
 else if(POS(TL, -1, Pcp))
  if(POS(TL, 1, Nnn)) return Adj;
  else if(POS(TL, 1, Vrb)) return Adj;
  else return Adv;
 else return Adv;
}
```