CLASSIFICATION OF STRUCTURED DESCRIPTIONS

Srinivas Bangalore

AT&T Labs–Research
180 Park Avenue
Florham Park, NJ 07932
srini@research.att.com

Owen Rambow

Center for Computational Learning Systems
Columbia University
New York, NY
rambow@cs.columbia.edu

ABSTRACT

Many speech and language processing problems have been successfully cast as classification problems– associating a token with a label from a prespecified label set. However, in all these applications the set of labels is regarded as a flat list of symbols with no inherent internal structure and no co-constraints among the labels. In this paper, we present a classification task using structured word labels called Supertags and discuss methods that exploit the structure of these labels in the context of natural language generation. Supertags encode predicate-argument information along with syntactic ordering constraints that can be exploited for realizing a sentence. We report the accuracy of supertagging models for a node using features derived from its local tree context and attributes of the supertags.

1. INTRODUCTION

Many problems in Speech and Natural Language Processing can be encoded as classification problems – associating a token with a label from a prespecified label set. Examples of such problems include assigning part-of-speech labels to words (POS tagging), identifying names of people, places etc. (named entity tagging), routing an incoming call to the appropriate destination (call routing), and assigning a label for a text (text classification). Although the same classification technique can be used for all these tasks, the token of classification and the size of the label set differs across tasks. The token to be classified varies from entire documents as in text classification, to substrings of a string as in named entity classification to individual words as in POS tagging. Also, the size of the label set varies from two classes for spam filtering to tens of classes for text classification and POS tagging. However, most of these classification tasks have viewed the set of classes as a flat list of symbols with no significant structure for the members and no relationships among the members of the class set. Furthermore, during decoding the classification techniques have not exploited hard constraints imposed by a class on the presence or absence of other classes.

In this paper, we review the problem of associating words with highly structured classes called *supertags* that encode syntactic information and predicate-argument structure. Supertags have been shown to be a useful level of representation for a number of NLP tasks [1], including parsing, language modeling and natural language generation. Supertags not only are structured labels, but also define relations between members of the supertag set and coconstrain other supertags. Supertagging, the problem of assigning the correct supertag for a word has been previously modeled using a generative model [1]. In this paper, we provide a classification model for supertagging in the context of natural language generation task; a generative model for the same task has been explored in [2]. The input tree to the natural language generation component is transformed into a string using information encoded in supertags. We show that exploiting the structure and constraints of supertags improves supertagging accuracy.

The paper is organized as follows. In Section 2, we discuss the details of the supertag representation, and in Section 3, we review the components of a natural language generation system. We present the model for supertagging in Section 4 and the results from experiments are presented in Section 5.

2. SUPERTAGS

Supertags are elementary trees of a Tree-Adjoining Grammar (TAG) [3]. In a lexicalized TAG, each word of a grammar is associated with one or more supertags. The word that is associated with the supertag is called the *anchor* of the supertag. Supertags are combined with each other using two operations, *substitution* (which appends one supertag at the frontier of another) and *adjunction* (which inserts one supertag into the middle of another). The combining of supertags results in two structures – a derived tree and a derivation tree. The derived tree is the tree resulting from substituting and adjoining supertags while the derivation tree records the process of constructing the derived tree.

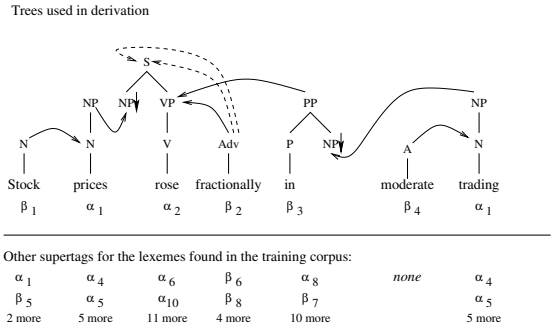

Fig. 1. The substitution and adjunction operations needed to derive *Stock prices rose fractionally in moderate trading*; dotted lines show possible adjunctions that were not made

For example, to derive the sentence *Stock prices rose fractionally in moderate trading* from the grammar in Figure 1, we adjoin the tree for *stock* into the tree for *prices*. We then substitute the tree for *prices* and adjoin the tree for *fractionally* into the tree for *rose* at the NP and VP nodes respectively. We then adjoin the tree for *moderate* into the tree for *trading* and substitute it into the tree for the preposition *in*. Finally, the tree for the preposition *in* is adjoined into the tree for *rose*, resulting in a complete derivation.

The *derivation tree* is constructed as follows: whenever we adjoin or substitute a supertag t_1 into a supertag t_2, we add a new daughter labeled t_1 to the node labeled t_2. We notate the links with role information which is derived from the node at which substitution or adjunction takes place. The derivation tree for our derivation is shown in Figure 2(a). As can be seen, this structure is a de-

pendency tree and resembles a representation of lexical argument structure.

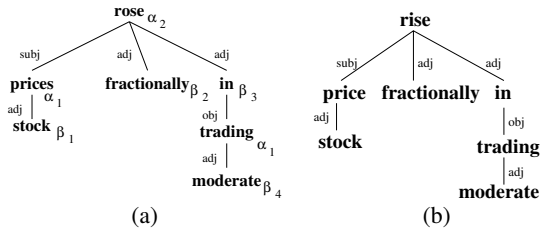

Fig. 2. (a): Derivation tree for LTAG derivation (b): Input for the surface realizer component for *Stock prices rose fractionally in moderate trading*

Unlike (un)lexicalized phrase structure rules which are single level trees, supertags are multi-level trees which encapsulate both predicate-argument structure of the anchor lexeme (by including nodes at which its arguments must substitute) and morpho-syntactic constraints such as subject-verb agreement within the supertag associated with the anchor. This property is referred to as TAG's *extended domain of locality* and a supertag is argued to be the domain over which linguistic dependency constraints can be specified. Thus, recursive structures (such as adjuncts) are factored out from these domains of dependencies and represented as separate supertags, known as auxiliary supertags. The non-recursive trees are called initial supertags.

As a result of the constraint that a supertag express all and only the predicate-argument dependencies, there are a number of supertags for each lexeme to account for the different syntactic transformations (relative clause, wh-question, passivization etc.). Most of the process of parsing a sentence can be viewed as disambiguating among the set of supertags associated with each lexeme, given the context of the sentence. This task of supertag disambiguation is termed as *supertagging*. In order to arrive at a complete parse, the only step remaining after supertagging is establishing the attachments among the supertags. Hence the result of supertagging is termed as an "almost parse" [1].

2.1. Supertag Set

The set of supertags is derived from a phrase-structure annotated TreeBank such as the Penn TreeBank [4] as illustrated in [5, 6]. These supertags are extracted by specifying certain patterns on the phrase structure trees so as to localize predicate-argument information and factor out recursive structures. Such an extraction procedure results in a supertag set of about 5000 supertags.

3. SUPERTAGS FOR NATURAL LANGUAGE GENERATION

Within natural language processing, natural language generation (NLG) is the task of determining a surface utterance (written or spoken) from some underlying representation. For example, the dialog manager in a dialog system can determine communicative goals for the next turn of the dialog system, and the NLG component then realizes these goals as a spoken utterance. NLG is often divided into three task groups: **text planning** is the task of determining communicative goals; **sentence planning** is the task of choosing abstract linguistic resources to achieve the communicative goals; and **surface realization** is the task of implementing the sentence planning choices in a well-formed surface string or spoken utterance specification in the target language. In this section we focus on the surface realization component and describe the input representation we work on and the choices that need to be made to produce an utterance.

The input representation we assume in this paper is an unordered deep dependency syntax tree, shown in Figure 2(b). Each node in the tree is labeled with a meaning-bearing ("autosemantic") lexeme, and the arcs are labeled with deep-syntactic relations (deep subject, deep object, deep indirect object, adjunct). The input thus encodes lexical choice and abstract syntactic choice. We also assume that each node is already annotated with semantic features. A semantic feature changes the meaning of a proposition, not just its linguistic realization. Semantic features which we do not determine include tense (for verbs), number and definiteness (for nouns), or degree of comparison (for adjectives and adverbs).

The surface realizer's task is to determine, for each node in the tree, the correct syntactic realization. This means determining which function words are to be inserted (such as determiners for nouns or auxiliaries for verbs), determining the word order among the node and its dependents, and determining the morphological inflection for the node's lexeme. We exploit the information encoded in the supertags for this purpose. As seen earlier, supertags encode the relative ordering information between the anchor and its dependents. Thus for each node, selecting a supertag that is compatible with the children nodes, provides linear ordering constraints for the nodes in that subtree. This information is used to construct the utterance as discussed in [2]. In this paper, we present discriminative models for selecting the appropriate supertag for each node in the input tree. We show that exploiting the structure of supertags in this selection process improves the accuracy of supertag assignment.

4. SUPERTAG DISAMBIGUATION

The objective is to assign supertags for the nodes of the input tree. This can be formulated as in Equation 1, where the T_w is the input tree with lexical items and T_s is the tree with supertags assigned to the nodes of the input tree. In previous work, we have used a generative model to disambiguate the supertags [2]. In this paper, we present a conditional probability model which is trained discriminatively. This model allows us to incorporate richer conditioning context without the problem of sparsity of data – a problem with generative models.

$$T_s^* = \underset{T_s}{argmax}\ Pr(T_w, T_s) \tag{1}$$

4.1. Discriminative Supertagging models

We determine the labeling (T_s^*) for a tree T_w by maximizing the posterior probability as shown in Equation 2. We use a large set of features of T_w and its context to determine the posterior probability and hence resort to a classification algorithm that is robust to large set of features for this purpose.

$$T_s^* = \underset{T_s}{argmax}\ Pr(T_s \mid T_w) \tag{2}$$

We use the AdaBoost classifier [7] wherein a highly accurate classifier is built by combining many "weak" or "simple" base classifiers f_i, each of which may only be moderately accurate. The selection of the weak classifiers proceeds iteratively, picking in each iteration the weak classifier that correctly classifies the examples that are misclassified by the previously selected weak classifiers. Each weak classifier is associated with a weight (w_i) that reflects its contribution towards minimizing the classification error. The posterior probability of $Pr(T_s \mid T_w)$ is computed as in Equation 3.

$$Pr(T_s \mid T_w) = \frac{1}{(1 + e^{-2 * \sum_i w_i * f_i(T_w)})} \tag{3}$$

4.2. Features used for Supertagging

We use the local tree context of a node to determine its supertag label. In general, the local context for a node includes the lexeme

and part-of-speech (POS) label of the node's mother and the lexeme, POS and supertag labels for all its daughters in addition to the lexeme and POS of the current node. All these attributes of the local context are regarded as features to the boosting algorithm. Since a prior bound cannot be set for the number of daughters for a given node, we model the lexeme, POS, supertag attributes of the set of daughters as an unordered bag of lexeme features, POS, supertag features respectively. This results in a fixed size feature vector for each node. From these feature vectors, we create a set of weak learners that test the values of these features. For the features that are bags, the weak learners test the presence of a particular value in the bag.

4.3. Greedy Decoding

The assignment of labels to an input dependency tree using these models is completely straightforward. The boosting model is queried with the local context that is appropriate for that particular model and the tag with the highest score is assigned to the node. Thus, we have a greedy decoder. The direction of decoding — top-down or bottom-up — depends on the particular model being used. In this paper, we use bottom up decoding. For the full model, the bottom-up order of decoding is the natural order of decoding, since the children's supertag would have been determined by the time the supertag for the current node needs to be determined.

5. EXPERIMENTS

In this section, we describe a set of experiments and present results for supertagging an input tree using discriminative models discussed previously. The input dependency trees were created by transforming the phrase structure tree annotations in the Penn TreeBank of the Wall Street Journal corpus. The conversion process produces input representations that are described in detail in Section 3. We use the Sections 01-22 as our training set (38332 sentences) and Section 00 as our test set (a total of 1745 sentences). The baseline of selecting the most frequent supertag for a lexeme results in an accuracy of 60.9% on the test set.

5.1. Results from Discriminative Model

We partitioned the training data based on the lexeme and for infrequent lexeme, we used the POS information to partition the data and train classifiers. Infrequent lexemes were those that occurred less than 5 times in the training corpus. Thus, the classifier was factored on the lexeme information for frequent lexemes and POS information for infrequent lexemes.

$$P(S\,|lex, lexPOS, \ldots) \\ = P_{lex=w}(S|lexPOS, \ldots)\ if\ \ count(w) >= 5 \\ = P_{lexPOS=p}(S|childPOS\ldots)\ if\ \ count(w) <= 5$$

In Table 1 we present the results of supertagging classification models for frequent and infrequent lexemes. In these experiments, we supertag the current node assuming the features that are used for classification are correct. In the next section, we report results on true decoding where we relax this assumption. We have investigated a number of models with different features but we report here only the best performing model due to space limitations. The best performing model used the lexeme and POS information of the mother, the current node and children nodes in addition to the children supertags and their role information as features. The overall performance for the entire test data set is 86.9%.

5.2. True Classification and Accuracy

In the set of experiments presented in the previous section, the supertags for the current node in a tree was determined assuming the children supertags are correct. The accuracy numbers are thus upper bounds of supertagging accuracy since in reality the children supertags have to be determined by the model. In order to arrive at true decoding we classify the nodes of the tree in a bottom-up scheme, beginning at the leaves and use the supertag predictions of the children as features for classifying the current node. We use the models that include role information from the Table 1 for frequent and infrequent lexemes respectively for true bottom-up decoding. The supertag for the current node is however chosen greedily at every node. Decoding in this manner, results in a accuracy of 85.8%, a drop in accuracy of about 1% (86.9% to 85.8%).

Schema used to represent a node	Accuracy(%)
Full Model: Lexeme,Lexeme POS, Children Lexeme(s), Children POS(s), Children Supertags, Children Roles, Mother Lexeme, Mother POS	86.8%
PosFull Model: Full Model without Lexeme	87.4%
Overall	86.9%

Table 1. Performance of supertagging models using different feature sets for frequent lexemes (Full Model) and infreuqent lexemes (PosFull Model)

5.3. Exploiting compatibility constraints during decoding

The result of classification is a local supertag assignment for each node of the input tree. However, the supertag assignments must be compatible in order to form a globally consistent tree. These compatibility constraints from the supertags can be used to weed out incompatible supertag configurations. After the classification of a node is done, the assigned supertag is checked for compatibility constraints with respect to its children supertags. For example, if the assigned supertag to a node has no substitution nodes (in its tree representation) and if one of the children's assigned supertag is meant for substitution, in the mother supertag, then this particular configuration of supertags for the mother and children is regarded as not compatible. In such cases, the assigned supertag for the mother node is discarded and the next best configuration is examined. Similar checks involving the grammatical type and role information of the children nodes are used to filter out incompatible configurations. Exploiting these compatibility constraints boosts the supertagging accuracy to 86.4% (from 85.8%).

5.4. Exploiting compatibility constraints during training

In the previous set of models, the training data was partitioned on the lexeme and a classifier was trained for each individual lexeme. In those models, the deep role information (subject, object, indirect object) provided in the input dependency tree in conjunction with the grammatical category (NP, S, AdvP, AdjP) is used to construct the subcategorization information (DSUBCAT) of a node and this feature is used during supertagging of a lexeme. For example, the verb *like* has a DSUBCAT=(NP_0,NP_1) in *John likes peanuts* and has a DSUBCAT=(NP_0,S_1) in *John likes to shop*. In order to select supertags that ensure the role constraints from the input are satisfied, we use the compatibility check described previously. However, the boosting training procedure does not guarantee the selection of DSUBCAT as an active weak learner during the training process. Thus by treating the DSUBCAT as a feature, the strict constraint is transformed into a weak constraint.

In order to maintain the strict constraint imposed by the DSUBCAT feature, we factor the model on the DSUBCAT feature of a supertag of a node instead of the lexemes, and train classifiers for each partition of the training data. This way of factoring the training, ensures that the DSUBCAT feature is used as a strict constraint.

The training process is further factored to take into account the lexeme information for frequent lexemes, POS information for less

frequent ones in conjunction with the DSUBCAT information. For the infrequent lexemes, we use the model factored only on the POS information. Thus:

$$
\begin{aligned}
&P(S|dsubcat, lex, lexPOS, \ldots) \\
&= P_{dsubcat=d,lex=w}(S|lexPOS, \ldots) \;\; if \;\; count(w) > 40 \\
&= P_{dsubcat=d,lexPOS=p}(S|childPOS, \ldots) \\
&\qquad\qquad if \;\; 5 < count(w) <= 40 \\
&= P_{lexPOS=p}(S|childPOS \ldots) \;\; if \;\; count(w) <= 5
\end{aligned}
$$

While decoding, the DSUBCAT information for a node is derived from role information specified in the input in conjunction with the grammatical category of the children's supertags. For example, if the two children of a node *like* are specified with roles as *0* (subject) and *1* (object) and during bottom-up decoding it is determined that the grammatical category of the root node of the children's supertags are *NP* and *S* respectively, then the DSUBCAT feature for the current node is *(NP_0,S_1)*. Now, if the lexeme *like* happens to be a frequent lexeme, then the classifier indexed on *(dsubcat=(NP_0,S_1),lex=like)* is used to classify the node.

By factoring the model as described above, we have ensured that the compatibility constraint is always satisfied. The import of this constraint is attested by the improved performance on supertagging: an accuracy of 89.5% (from 86.4%).

5.5. Classification accuracy using attribute-value representation of supertags

In the preceeding experiments, supertags were treated as a single label for classification models. However, supertags can be characterized by a fixed set of attribute value pairs. These attributes can be viewed as dimensions of a linguistic space in which individual supertags are unique points. We represent each supertag with the set of attributes shown in Table 2. We regard each attribute as a dimension for classification and their values as the classes. Thus supertagging, in this view, amounts to obtaining the class values for each of the fixed set of attributes.

We have as many classifiers as the number of attributes (except for those that are derivable from the input dependency tree). The features used for training these classifiers are the same that we have used for supertag classification. Table 2 shows the accuracy of two baseline models and the accuracy of the boosted classifier model on the same test data. The first baseline is using the most frequent value of an attribute as the result class. This is in contrast to the accuracy of the boosted classifiers which use all the information from the tree context as features to arrive at the class. It is interesting to note that the accuracy on certain attributes such as *Modification Category*, *Modification Category*, *Predicative Category of Root node* improve significantly as the conditioning context increases.

During bottom-up decoding on the tree, at each node, we predict the value for each attribute using the corresponding classifier. The predicted values of the attributes in conjunction with the attribute values derivable from the input (such as DSUBCAT, POS etc.) is used to construct the *decoded* attribute-value (AV) representation. This decoded AV is mapped into a supertag label. In case there are multiple supertags with the same AV representation, the supertag with the highest frequency is selected. For cases where there is no supertag with the decoded AV representation, then the AV with the smallest edit distance from the decoded AV is used to retrieve the supertags. The overall supertagging accuracy with this method of decoding rises to 90.7%.

Table 3 summarizes the previous results and attests to the fact that exploiting the structural constraints encoded in supertags allows us to improve supertagging accuracy beyond models that treat supertags as a flat list of labels.

Attribute	Values	Baseline	Boosting
Dative Shift	+/nil	0.99	0.994
Direction of Modification	Left/Righ/nil	0.43	0.962
Empty Subject	+/nil	0.98	0.99
Modification Category	S,NP,VP,...	0.44	0.955
Particle Shift	+/-/nil	0.99	0.999
Predicative	+/-/nil	0.64	0.998
Predicative Auxiliary	+/nil	0.99	0.999
Argument Position of Relativization	+/0/1/2/nil	0.98	0.995
Category of Root node	S,NP,...	0.35	0.975
Voice	active/passive/ passive_by/nil	0.87	0.993
Argument Position of Wh-Extraction	0/1/2/nil	0.99	0.997

Table 2. List of attributes used to represent a supertag and their accuracies under different models.

Model		Supertagging Accuracy
Supertags as labels		85.8%
Exploiting compatibility constraints	during decoding	86.4%
	during training	89.5%
Attribute-value representation		90.7%

Table 3. Improvement in supertagging accuracy by exploiting the supertag representation.

6. CONCLUSIONS

Supertags encode predicate-argument structure and have been shown to be useful for parsing. In previous work, we have used supertags with a generative probabilistic model for surface realization, a subtask of Natural Language Generation. Given the limitations of generative models in terms of sparseness to large feature sets, we have presented, in this paper, a discriminatively trained model for selecting supertags. We have discussed several models that exploit the linguistic constraints encoded in the supertags and show the improvement in accuracy over models that ignore these constraints.

7. REFERENCES

[1] S. Bangalore and A. K. Joshi, "Supertagging: An approach to almost parsing," *Computational Linguistics*, vol. 25, no. 2, 1999.

[2] S. Bangalore and O. Rambow, "Exploiting a probabilistic hierarchical model for Generation," in *COLING*, Saarbucken, Germany, 2000.

[3] A. K. Joshi, "An introduction to tree adjoining grammars," in *Mathematics of Language*, A. Manaster-Ramer, Ed. John Benjamins, Amsterdam, 1987.

[4] M. Marcus, B. Santorini, and M.A. Marcinkiewicz, "Building a Large Annotated Corpus of English: The Penn Treebank," *Computational Linguistics*, vol. 19.2, pp. 313–330, June 1993.

[5] F. Xia, M. Palmer, and A.K. Joshi, "A uniform method of grammar extraction and its applications," in *Proceedings of Empirical Methods in Natural Language Processing*, 2000.

[6] J. Chen and K. Vijay-Shanker, "Automated extraction of tags from the penn treebank," in *Proceedings of the 6th International Workshop on Parsing Technologies*, Trento, Italy, 2000.

[7] R.E. Schapire, "A brief introduction to boosting," in *Proceedings of IJCAI*, 1999.