Large Scale Experiments with Naive Bayes and Decision Trees For Function Tagging

MIHAI LINTEAN

Department of Computer Science, Institute for Intelligent Systems
The University of Memphis, Memphis, TN 38152, USA

M.Lintean@memphis.edu

VASILE RUS

Department of Computer Science, Institute for Intelligent Systems
The University of Memphis, Memphis, TN 38152, USA

vrus@memphis.edu

Received (Day Month Year)
Revised (Day Month Year)
Accepted (Day Month Year)

This paper describes the use of two machine learning techniques, naive Bayes and decision trees, to address the task of assigning function tags to nodes in a syntactic parse tree. Function tags are extra functional information, such as logical subject or predicate, that can be added to certain nodes in syntactic parse trees. We model the function tags assignment problem as a classification problem. Each function tag is regarded as a class and the task is to find what class/tag a given node in a parse tree belongs to from a set of predefined classes/tags. The paper offers the first systematic comparison of the two techniques, naive Bayes and decision trees, for the task of function tags assignment. The comparison is based on a standardized data set, the Penn Treebank, a collection of sentences annotated with syntactic information including function tags. We found out that decision trees generally outperform naive Bayes for the task of function tagging. Furthermore, this is the first large scale evaluation of decision trees based solutions to the task of functional tagging.

Keywords: natural language processing; function tags; machine learning; naive Bayes; decision trees.

1. Introduction

Syntactic information is an important processing step to many language processing applications such as Anaphora Resolution [1], Machine Translation [2], and Question Answering [3].

We illustrate the importance of syntactic information with an example from Question Answering. In Question Answering, given a question, e.g. *What is the capital of Italy?*, the task is to retrieve a short answer to the question, e.g. *Rome*. The length of the answer must be a sentence or less. The answer should be retrieved from a large collection of text, in the order of millions of documents (gigabytes of text). A typical Question Answering system[4] first locates a set of candidate answer sentences from the collection based on shallow techniques that mainly use keywords from the question. A deeper analysis is then performed on the candidate sentences. The deep analysis relies on various types of linguistic information: lexical, syntactic, and semantic. To show the importance of the syntactic information

to distinguish between correct and incorrect candidate answer sentences, we pick question 481 from the NIST's QA track[a]: *Who shot Billy the Kid?* The two candidate answers that can be retrieved from the 5 gigabytes collection of text provided by NIST are: *Billy the Kid shot the rope* and *Billy the Kid was shot and killed by Sheriff Pat Garrett*. These candidate sentences are extracted as possible answers because they contain the two keywords from the question: *"Billy the Kid"* and *"shot"*. However, only the second sentence contains the correct answer to the question. To determine the sentence with the correct answer from the two candidate answer sentences, we can use syntactic information. The question inquires about the logical subject of a *shooting* event whose object is *Billy the Kid*. Thus, a correct answer sentence should talk about a *shooting* event that has *Billy the Kid* as its direct object. The logical subject of the *shooting* event in the correct answer sentence would be the answer to the question. In the first candidate sentence, the object of the verb *shoot* is *the rope* which does not match the object of the verb *shoot* in the question, *Billy The Kid*. The second candidate answer sentence meets the syntactic constraints imposed by the question and thus it is a correct answer sentence. The logical subject of the *shooting* event in this sentence, *Sheriff Pat Garrett*, is the correct answer. In this paper, we use naive Bayes and decision trees based classifiers to build a functional tagger that can retrieve syntactic information of the type needed in the above example. More specifically, we augment syntactic parse trees, the output of state-of-the-art syntactic parsers, with functional tags such as logical subject. Function tags are extra labels attached to nodes, which represent phrases, in a parse tree. Function tags augment the syntactic and semantic roles of the nodes/phrases.

Syntactic parsing in its most general definition may be viewed as discovering the underlying syntactic structure of a sentence. The specificities include the types of elements and relations that are retrieved by the parsing process and the way in which they are represented. For example, Treebank-style parsers[5] retrieve a hierarchical organization (tree) of smaller elements (called phrases, e.g. noun phrases - NP, verb phrases - VP, sentence - S), while Grammatical-Relations(GR)-style parsers explicitly output relations together with elements involved in the relation (e.g., subj(John,walk) explicitly indicates a subject relation between *John* and *walk*).

We work in this paper in the realm of Treebank-style parsers, i.e., parsers producing an output conforming to the Penn Treebank annotation guidelines [5]. The Penn Treebank project defined a tag set and bracketed form to represent syntactic trees that became a standard for parsers developed/trained on Penn Treebank. It also produced a treebank, a collection of hand-annotated texts with syntactic information, containing over one million words of text taken from 1989 Wall Street Journal materials.

Given a sentence as input, Penn Treebank-style parsers output parse trees. Examples of such parse trees are shown in Figure 1 for the sentence *Mr. Hahn rose swiftly through the ranks.*[b] The most successful Treebank-style parsers are based on statistical models [6,7,8]. They use Penn Treebank to derive the parameters of the models. These parsers focus on identifying the major phrases in a sentence such as NP, VP or S, although Penn Treebank

[a]National Institute of Standards and Technology - Question Answering Track
[b]This sentence is from Wall Street Journal portion of Penn Treebank.

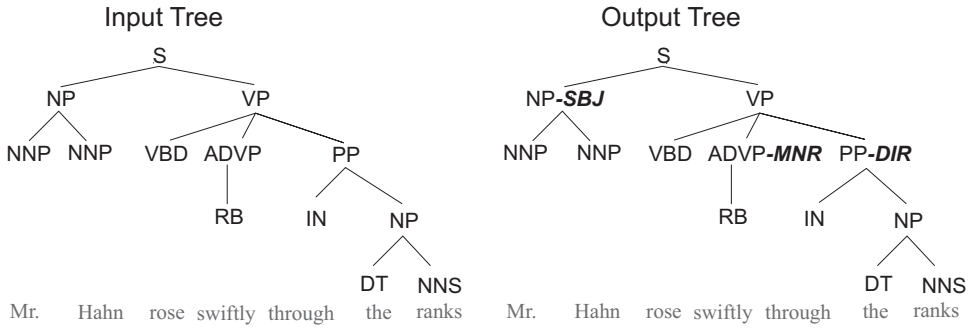

Fig. 1. Examples of syntactic parse trees with (left) and without (right) function tags

contains annotations for other types of information such as function tags and traces [c]. Function tags have been added in Penn Treebank to augment the syntactic and semantic roles for the nonterminal labels. These tags are necessary to distinguish words or phrases that belong to one syntactic category and are used for some other function or when it plays a role that is not easily identified without special annotation. The current state-of-the-art parsers usually ignore the extra information,i.e. function tags and traces, available in Penn Treebank in order to make the estimation of the parameters of their underlying statistical models more reliable. This paper presents a method to augment the output of Treebank-style syntactic parsers with functional information in the form of function tags. We build functional taggers that can be used with a state-of-the-art syntactic parser to produce parse trees with richer syntactic information.

The paper describes two techniques, one based on naive Bayes and another based on decision trees, to assign function tags to nodes in parse trees. The function tags assignment problem is viewed as a classification problem, where the task is to find for a given node in a parse tree the correct function tag/class from a list of candidate tags/classes. Classifiers assign a class from a predefined set of classes to an instance based on the values of attributes used to describe the instance. For function tagging, each node in a parse tree corresponds to an instance. We define a set of linguistically motivated attributes/features based on which we characterize the instances. Instances are automatically generated from Penn Treebank for each internal node. The instances are then used to derive naive Bayes and decision trees classifiers as solutions to the function tags assignment problem. The paper presents a systematic comparison of the two techniques based on evaluations conducted on a standard data set. The two techniques use the same underlying model and same data set to derive and test the classifiers, which allows for a fair comparison.

We chose naive-Bayes and decision trees for their simplicity and user-friendliness, respectively. Naive-Bayes classifiers make strong assumptions about how the data is generated and use a probabilistic model that reflects these assumptions. They use a collection of

[c]Traces are remote dependencies between words that are far apart in a sentence such as the direct object relationship between *call* and *John* in the following sentence: *John is the person I called yesterday*.

labeled training examples to estimate the parameters of the generative model. Each training example is described by a set of attributes, also called features, and the corresponding class label. Classification of new examples is performed with Bayes' rule by selecting the class that is most likely to have generated the example. The naive Bayes classifier assumes that all attributes of the examples are independent of each other given the context of the class. This is the so-called *naive Bayes assumption*. This assumption is wrong in many real-world tasks, yet naive Bayes classifiers often perform very well. This paradox is explained by the fact that classification estimation is only a function of the sign (in binary cases) of the function estimation; the function approximation can still be poor while classification accuracy remains high [9]. Because of the independence assumption, the parameters for each attribute can be learned separately, and this greatly simplifies learning, especially when the number of attributes is large [10]. Many studies have focused on the performance of the naive Bayes classifiers: McCallum and Nigam[10], Irina[11], and more recently Zhang[12] who proposes a novel explanation for the apparently unreasonable efficacy of the naive Bayes classifiers.

Decision tree induction has been studied in detail both in the area of pattern recognition and in the area of machine learning.[13] In the vast literature concerning decision trees, also known as classification trees or hierarchical classifiers, at least two seminal works must be mentioned: Breiman et al.[14] and Quinlan[15]. The former originated in the field of statistical pattern recognition and describes a system, named CART (Classification And Regression Trees), which was mainly applied to medical diagnosis and mass spectra classification. The latter synthesizes the experience gained by people working in the area of machine learning and describes a computer program, called ID3, which has evolved into a new system, named C4.5 [16]. Decision trees are mainly used for classification purposes, but they are also helpful in uncovering features of data that were previously unrecognizable to the eye. Thus, they can be very useful in data mining activities as well as data classification. They work well in many areas from typical business scenarios to airplane autopilots to medical diagnoses. One major advantage of decision trees is their generation of a tree model that can be easily interpreted by humans.

We present in this article the first large scale evaluation of decision trees based solutions to the task of functional tagging. We used the full data set that Penn Treebank makes available in order to train and test a decision trees based functional tagger. In Blaheta[17] (see section 5.2 *Why we abandoned decision trees*), the decision trees approach was abandoned due to memory limitations. We addressed the memory issue by using a set of preprocessing steps applied to training and test data and by using a High-Performance Computer (IBM AIX System with 64GB of RAM). The preprocessing was necessary in order to reduce the large number of distinct values some features/attributes have. Too many distinct values for these features led to very large decision trees that would not fit even in the memory of the High-Performance Computer.

The rest of the paper is organized in the following five sections. First, in the *Related Work* section, we analyze previous efforts related to the task of function tags assignment and to comparing naive Bayes and decision trees classifiers. *The Problem* section defines the problem we addressed. Details on how we approached the problem are presented in *The Model* section. The *Experimental Setup and Results* section presents the evaluation we

conducted on a standardized data set. The *Conclusion and Future Work* section ends the paper.

2. Related Work

Previous work can be categorized in two groups: work related to the task of function tag assignment and work related to comparing decision trees and naive Bayes classifiers. We present an overview of each group, starting with the former.

Previous research to address the task of function tags assignment was mainly conducted by Don Blaheta and Eugene Charniak [18,17]. They used a statistical algorithm based on a set of features grouped in *trees*, rather than *chains*. The advantage is that features can better contribute to overall performance for cases when several features are sparse. When such features are conditioned in a chain model the sparseness of a feature can have a dilution effect of an ulterior (conditioned) one.

In Rus and Desai[19], preliminary work on function tagging with a naive-Bayes approach is reported. The major difference from our work is that they used different data preprocessing steps. Our data preprocessing steps were developed to reduce the number of values of certain features in our model. The reduction is necessary in order to generate decision trees that can fit in the computational resources available. Also, Rus and Desai used a slightly different set of features than ours.

There were other notable efforts related to function tags. Michael Collins[6] used function tags to define certain constituents as complements. The technique was used to train an improved syntactic parser. Also, there were attempts to enrich the output of syntactic parsers with additional information available in Penn Treebank, such as syntactic relations among empty nodes and their antecedents [20,21].

Before Blaheta[18] there were no previous attempts to recover function tags from Penn Treebank. However, there were several projects whose goal was to annotate grammatical functions and syntactic categories[22] or semantic roles of word phrases[23]. The grammatical functions used by Brants et al.[22] are similar to function tags. The set of tags is different from the Penn Treebank set because the target language is German. Semantic roles[23] indicate roles, e.g. agent, played by various phrases in a sentence. We discuss below in more detail the two projects that focused on grammatical functions and semantic tagging.

Brants et al.[22] studied the problem of grammatical function tagging on the NEGRA corpus. Like Penn Treebank, the NEGRA corpus is a syntactically annotated corpus of german newspapers texts. The latest version of NEGRA, version 2, contains over twenty thousand manually parsed sentences. The most important difference from Penn Treebank is that the NEGRA corpus has a grammatical function tag on every node in the syntactic tree, except the root node. Brants and colleagues developed a system to tag the NEGRA corpus using Markov models. The system is an annotation tool, which provides partial or complete parses of the text, and an intuitive graphical interface to assist annotators to verify and correct the parsed texts.

The FrameNet[23] project assigns semantic roles to phrases in a sentence. The function tags correspond here to the so-called ”frame elements”, which are subelements of some

predefined semantic frame. Typically each sense of a polysemous word belongs to a different semantic frame, a script-like conceptual structure that describes a particular type of situation, object, or event along with its participants and props (tools). For example the Apply-heat frame describes a common situation involving a COOK, some FOOD, and a HEATING-INSTRUMENT, and is evoked by words such as *bake, blanch, boil, broil, simmer, steam,* etc.[23] The number of possible semantic roles, as defined in FrameNet, is very large. Gildea and Jurafsky[24] developed a system to identify these semantic roles in the FrameNet corpus. They have constructed a mapping from the frame element tags into a more abstract set of eighteen *thematic roles*, which resemble function tags.

The other major group of prior work is about comparing naive Bayes and decision trees. There were many empirical comparisons between the naive Bayes and decision trees classifiers, which showed that both models predict equally well (Langley, Iba and Thomas[25]; Kononenko[26]; Pazzani[27]). However, there is no previous work, to our knowledge, that attempted to systematically compare naive Bayes or decision trees approaches to the task of function tagging.

We present in this paper a common framework to address the problem of function tagging and report how naive Bayes and decision trees perform within this framework. The framework is defined by a common underlying model and a common set of data preprocessing steps. The model and the preprocessing steps are described later.

3. The Problem

The task of function tagging is to add extra labels, called function tags, to certain nodes in a parse tree. As an illustrative example we use the sentence *Mr. Hahn rose swiftly through the ranks.* A state-of-the-art Treebank-style syntactic parser would generate the parse tree shown on the left hand side in Figure 1. Each word in the sentence has a corresponding leaf (terminal) node, representing that word's part of speech. For instance, the word *ranks* has NNS as its part of speech (NNS indicates a common noun in plural form). All the other nodes, called internal or non-terminal nodes, will be labeled with a syntactic tag that marks the grammatical phrase corresponding to the node, such as NP, VP, or S.

It is not obvious from such syntactic parse trees which node plays the role/function of logical subject. A user of these parse trees needs to develop extra procedures to detect roles played by various words or phrases. The task of function tagging, the problem addressed in this paper, is to add labels, i.e. function tags, to phrases that explicitly indicate the roles played by the phrases.

Function tags have been added in Penn Treebank[5] to augment the syntactic and semantic roles for the nonterminal labels. These tags are necessary to distinguish words or phrases that belong to one syntactic category and are used for some other function or when it plays a role that is not easily identified without special annotation. In Table 1 we give a complete list of the function tags, along with the percentage that they appear within all the nonterminal nodes in the current Penn Treebank corpus. Next, we briefly introduce the

most important function tags. The SBJ tag marks the subject phrase of every S or SINV [d] node. It is by far the most common function tag. On the other hand, the logical subject of a passive sentence is labeled with the LGS tag. The PRD function tag marks any predicative construction that is not a verb phrase, for example the noun and adjective phrases of *be* (e.g. *This is* `an old story`). Another common tag is the TMP tag and it marks the temporal constituents, which are phrases that answer questions such as *when?*, *how long?* or *how often?* Other tags are DIR, prepositional phrases that answer the questions *from where?* and *to where?*; MNR, which is used for phrases that indicate the manner in which some action was performed; LOC, a common tag used to mark phrases that denote the place where something takes place; and PRP, which marks phrases that annotate the purpose of or reason for an action. Some tags are very rare, such as BNF (Benefactive), CLF (It-cleft) or VOC (Vocative). The CLR tag is a special tag, which was treated differently by Blaheta in his study[17]. CLR stands for "closely related" and marks constituents that occupy some middle ground between argument and adjunct of the verb phrase [5]. This tag was omitted in Blaheta's reported results because they found out that the tag's original purpose was just to "relieve annotator frustration" and it should have been removed in the final version of the corpus. However, because the tag is present in the corpus, we decided to keep the tag in our tests and treat it the same as the other tags.

Table 1. Penn Treebank Function Tags

Label	Name	Coverage	Label	Name	Coverage
ADV	Non-specific adverbial	0.76%	MNR	Manner	0.41%
BNF	Benefactive	0.00%	NOM	Nominal	0.48%
CLF	It-cleft	0.00%	PRD	Predicate	2.24%
CLR	Closely related	0.98%	PRP	Purpose	0.39%
DIR	Direction	0.45%	PUT	Locative complement of 'put'	0.04%
DTV	Dative	0.06%	SBJ	Subject	8.62%
EXT	Extent	0.19%	TMP	Temporal	2.73%
HLN	Headline	0.04%	TPC	Topic	0.44%
LGS	Logical subject	0.36%	TTL	Title	0.08%
LOC	Location	2.16%	VOC	Vocative	0.01%

To sum up, the task of function tags assignment is to generate a parse tree that has function tags attached to certain nodes in the parse tree. The task is to generate a parse tree, similar to the one on the right hand side in Figure 1, from an input parse tree that has no function tags, such as the parse tree on the left hand side in the figure. The tree on the right hand side has three functional tags: SBJ attached to the node NP, MNR attached to ADVP (Adverbial Phrase), and DIR attached to PP (Prepositional Phrase).

[d] SINV - Inverted declarative sentence, i.e. one in which the subject follows the tensed verb or modal.

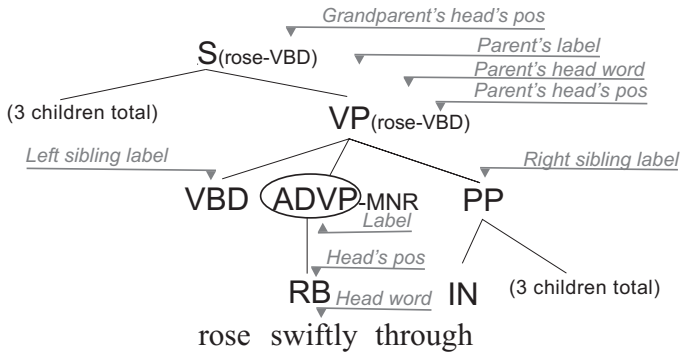

Fig. 2. Example of a Treebank Node and its Features

4. The Model

We model the problem of assigning function tags as a classification problem. Classifiers are methods that assign a class from a predefined set of classes to an instance based on the values of attributes used to describe the instance. We define a set of linguistically motivated attributes/features based on which we characterize the instances. Instances are automatically generated from Penn Treebank. The instances are then used to derive naive Bayes and decision trees classifiers as solutions to the function tags assignment problem.

We describe below the set of features and classes we used to build the classifiers.

Features. We used a set of features inspired from Blaheta and Charniak[18] that includes the following features associated with a nonterminal node: label, parent's label, right sibling label, left sibling label, head word, parent's head word, head's pos (part-of-speech), parent's head's pos, grandparent's head's pos. In Figure 2, we show these features for the ADVP node marked with a circle. The function tag of this node is MNR, which means that the adverbial phrase represented by the node indicates the manner of the action represented by the parent verb phrase.

The meaning of the first four features listed above is self explanatory. To explain the rest of the features we must define what a head word is. The *head word* of a phrase is the most important word in the phrase. There is a set of deterministic rules to detect the head word of a phrase (see Magerman[28] and Collins[29]). They are simply called *the head rules*. An example of a head rule is the following: the head word of a noun phrase is its rightmost noun [e]. For instance, the noun phrase *goat cheese* has *cheese* as its head. To find the head word for a phrase, corresponding to a node in a parse tree, we used the head rules, with a slight modification described below. The head words for each node are recursively found in the tree starting from bottom to top. The head word of an internal node in the parse tree is the head word of one of its children. The child is chosen based on the head rules. If the chosen child node is terminal, then the word in the terminal node becomes the head

[e]This rule applies to non-recursive noun phrases. A non-recursive noun phrase is a noun phrase that does not contain another noun phrase as one of its children.

word. If the child is an internal node, then it will have a head word already assigned. This node becomes the head word of the current phrase. The head word and parent's head word features in our model are represented by the head words assigned to the current node and the parent's node, respectively.

The head's pos mentioned in the description of the last three features is nothing else than the part of speech tag of the head word. This is stored in the label of the terminal node associated with the head word.

The only change we made to the head word rules was due to the *alternative head* feature mentioned by Blaheta[17]. This feature is only valid for prepositional phrases that are the head of a prepositional object. We did not use the alternative head's pos and alternative head as explicit features but rather modified the head word rules so that the same effect is captured in the head's pos and head word features.

Classes. The set of classes we used in our model corresponds to the set of function tags in Penn Treebank. In the annotation guidelines for Treebank II [5], the 20 function tags listed in the previous section are grouped in four categories: form/function discrepancies, grammatical role, adverbials, and miscellaneous. Up to 4 function tags can be added to the standard syntactic label (NP, VP, S, etc.) of each node. For instance, a node can have a composed tag such as NP-LGS-TPC to indicate a noun phrase (NP) that has attached to it two function tags, logical subject (LGS) and topicalisation (TPC).

Table 2. Categories of Function Tags.

Category	Function Tags
Grammatical	DTV, LGS, PRD, PUT, SBJ, VOC
Form/Function	NOM, ADV, BNF, DIR, EXT, LOC, MNR, PRP, TMP
Topicalisation	TPC
Miscellaneous	CLR, CLF, HLN, TTL

We rearranged the four categories into four new categories based on corpus evidence, similar to Blaheta and Charniak[18]. The new four categories are given in Table 2 and were derived so that no two labels from same new category can be attached to the same node in a parse tree.

The above features and classes are used to derive naive Bayes and decision trees classifiers. This common underlying model allows for a crisp comparison of the two techniques for the task of assigning function tags. The next section describes the experiments we conducted to derive and evaluate the classifiers.

5. Experimental Setup And Results

We trained the classifiers on sections 1-21 from Wall Street Journal (WSJ) corpus of Penn Treebank and used section 23 from the same corpus to evaluate the generated classifiers. This split is standard in the syntactic parsing community[7]. The evaluation follows a gold standard approach in which the output of classifiers is automatically compared with the correct values, also called gold values.

The performance measures reported are accuracy and kappa statistic. The *accuracy* is defined as the number of correctly tagged instances divided by the number of attempted instances. *Kappa statistic*[31] measures the agreement between predicted and actual/gold classes, while correcting for agreement that occurs by chance. Kappa can have values between -1 and 1 with values greater than 0.60 indicating substantial agreement and values greater than 0.80 showing almost perfect agreement (see WolframMathworld[30] for more details).

The evaluation measures that we report here are different from the ones reported by Blaheta[18] (they used precision, recall and f-measure) and thus a direct comparison is not possible. In Blaheta and Charniak[18], because they considered both nodes with and without function tags using precision and recall made more sense. Our experimental setup is different. Due to a large number of nodes with no function tags ($\sim$90%), we conducted two types of experiments. In the first type of experiments, we considered only nodes with function tags to get a better idea of how well naive Bayes and decision trees are at detecting true function tags. In this case, using recall and precision would not have made sense because there are no instances with no function tag. We could have used some form of precision and recall but the resulting figures would have not been comparable anyway with Blaheta and Charniak's. For the second type of experiments, we considered nodes with and without function tags, similar to Blaheta and Charniak. Here, using the precision and recall measures would have made more sense, but we chose to keep the measures we used in our initial set of experiments for consistency and comparison purposes. In a quick analysis, the figures reported in Blaheta[18] are close to our results for similar experiments.[f]

Another reason for which a direct comparison between the work presented in this paper and Blaheta's work is not possible is because Blaheta[17] was not able to report full results on decision trees due to memory limitations. Blaheta explains why they have abandoned decision trees, although "improving the performance of the decision trees was unquestionably a valid possible research direction"[17] (page 32). Conversely, we do report here performance figures on decision trees. As stated earlier, we present the first large scale experiments with decision trees for the task of function tagging.

Furthermore, a direct comparison is not possible because our data instances for both training and testing were obtained from perfect parse trees in Treebank, while in Blaheta and Charniak[18] they parsed the test data using a state-of-the-art parser and considered only correct constituents in the output when reporting per-category results of the function tags assignment classifier.

A direct comparison with Rus and Desai[19] for naive Bayes is also not possible because they used different data preprocessing steps and a slightly different set of features.

To build the classifiers, we used the implementations of naive Bayes and decision trees

[f]The performance figures we used to compare the results are the reported *with-null* accuracy from Blaheta and Charniak and the accuracy reported in our last experiment. As another indication that a direct comparison is not possible are the baseline results. The baseline figures are slightly different in this article from Blaheta and Charniak's although we use the same baseline of guessing the most frequent function tag. The slight difference can be explained by the differences in the data preprocessing steps.

in WEKA. WEKA[31] (Waikato Environment for Knowledge Analysis) is a comprehensive, open-source (GPL) machine learning and data mining toolkit written in Java. We have chosen WEKA because it offers the same environment for naive Bayes and decision trees, and thus the comparison would be more reliable. WEKA requires a great deal of memory to build the models from large training sets, especially for decision trees. A regular machine with 2GB of memory is not sufficient even after applying the data preprocessing steps aimed at reducing the size of the data set (see details below about data preprocessing). We used instead an AIX High Performance Computer (HPC) system with 64GB of RAM.

5.1. *Data Collection*

Before we build naive Bayes or decision trees based classifiers, there is need to collect training data. The training data is a set of problem instances. Each instance consists of values for each of the defined features of the underlying model and the corresponding class, i.e. function tag. The values of the features and class are automatically extracted from Penn Treebank parse trees by simply traversing the trees and for each node extracting values for the defined features. When considering also the nodes that are not labeled with function tags, then a default class (called NON-F) is assigned to those nodes. Only internal nodes from parse trees are considered because only these nodes can be labeled with a function tag. In general, if the data preprocessing is done well, a good predictive model would correctly classify instances of terminal or root nodes because these nodes are clearly associated with one class. For instance, the terminal (leaf) nodes in our problem belong to the NON-F class. However, the purpose of the predictive models is to correctly label the hard examples. This is one reason why we filtered out some of the nodes in the trees.

We collected two sets of training data, one for each type of experiment we conducted.

First type of experiments. For the first type of experiments, we only processed nodes with function tags, ignoring nodes unlabeled with such tags in Penn Treebank. This allowed for a more detailed comparison of the two approaches, naive Bayes and decision trees, on nodes with true function tags. Nodes without function tags greatly outnumber those with. The bottom line is that including the no-function-tag nodes would lead to a less clear picture of how well the two approaches could detect true function tags.

Because a node can have several function tags there are two possible setups for our classification task. We can define a class as a composed tag of all possible combinations of function tags that can be assigned to a node. A single classifier is generated in this case that would assign one composed tag to a node, i.e. one or more individual function tags at once. Alternatively, we can try to build four separate classifiers, one for each of the four functional tags categories. Based on the fact that no node can have more that one tag from a given category, each classifier will be used to predict a function tag from the corresponding category.

Second type of experiments. For the second type of experiments, both nodes with and without function tags are considered to generate training instances. We could use the same setups as in the first type of experiments: single classifier to predict composed tags versus four classifier, one for each category of function tags. The setup in which we would gener-

ate a single classifier for all function tag categories is not possible anymore. This challenge arises from two sources, which are related. First, nodes without function tags greatly outnumber nodes with function tags. The number of the training instances is much bigger. Second, there is a large number of distinct values for some of the features in the model. The problematic features are the head word based features. Due to lexical diversity, these features have tens of thousands of distinct values leading to extremely large decision trees. These decision trees are too large to fit in the memory of conventional or more powerful computers such as the High-Performance Computer we used in our experiments. The second setup in which we build four classifiers, one for each function tag category, greatly reduces the number of distinct values for the problematic features. Creating decision trees that handle nodes labeled with the newly introduced NON-F function tag, becomes possible.

In summary, we conducted the following three experiments:

(1) each training instance is assigned a *composed* tag, similar to Penn Treebank annotations; only nodes with function tags in Treebank are used to generate training instances; a single classifier is induced;
(2) each training instance is assigned a *single* function tag from one of the four function tags categories; only nodes labeled with function tags in Treebank are used to generate training instances; four classifiers are induced;
(3) each instance is assigned a *single* function tag from one of the four function tags categories; all the nodes, with and without function tags, in Penn Treebank are used to generate training instances; four classifiers are induced.

While the first experiment above is simpler, the last two experiments are similar to what a Treebank annotator would do, i.e. assigning a tag from one of the four categories at a time.

Data Preprocessing. Some simplifications were necessary to make the task feasible. First, punctuation was mapped to a unique token PUNCT and traces in parse trees replaced with TRACE. Second, we handled features with a large number of distinct values. Two head features, head word and parent's head word, have a great deal of distinct values, i.e. distinct words. There is need to further process the data to reduce the lexical diversity problem. We tried different transformations for reducing the number of possible values for head words based features. We applied lexical and morphological transformations, e.g. stemming and grouping some of the words in lexical categories. The full set of applied transformations are given below.

(1) Replace all the words that represent family, female or male names with labels indicating the following three generic classes: Family Name, Male Name and Female Name. We used three dictionaries to categorize proper nouns into one of the tree classes.
(2) Replace words denoting numbers, identified by the part of speech label CD, with the following class labels: Number, Fraction (e.g. 3/4), Time (e.g. 10:30) and Date Period (e.g. 1987-1995).
(3) Change everything to lower case. This is to prevent differences between words that are at the beginning of sentences and same words appearing somewhere in the middle of a

sentence.

(4) Replace numbers from complex words that are composed of a number and a word. For example words like *10%-used*, *80%-controlled* or *mid-1987*, were changed to *Percent-used*, *Percent-controlled*, and *mid-YearPeriod*.

(5) Eliminate special characters (dot or comma) from the words, because they are used as special characters by the classification program.

(6) Stem all the words, i.e. reduce all morphological variations of a word to the base form of the word. For instance, both *go* and *went* would be mapped to *go*. To stem the words we have used the Porter Stemmer's algorithm[32]. This stemmer is widely used in the information retrieval field. It is the de-facto standard algorithm for stemming English texts.

Table 3 shows the number of unique values for head words after applying the above transformations to the set of head words in Penn Treebank. We counted the distinct values for the *head word* and *parent's head word* features. The rows correspond to the transformations listed above, when they are gradually applied to the two sets of head words. We show numbers for the two types of experiments described earlier, that is when we consider only the nodes with function tags, or when we test all the internal nodes. The distinct values were counted from both the training and the test data.

Table 3. Number of distinct values for the head words.

Type of Transformation	Only nodes with function tags		All internal nodes	
	Head	Parent's Head	Head	Parent's Head
Untransformed	19731	14794	27374	23156
Step 1 - Names	18380	14056	25819	21811
Step 2 - Numbers	17552	13728	24376	20776
Step 3 - Lowercase	15581	12767	21631	18788
Step 4 - Numbers in words	15562	12756	21604	18761
Step 5 - Special Characters	15510	12733	21519	18717
Step 6 - Final-Set	**11430**	**8402**	**14975**	**12836**

Because by applying this set of transformations a property regarding the similarity between the **parent's head** and **head** features might be lost, we added a new feature to the set of features in our model. The new feature has a Boolean value that indicates if the parent's head is the same with the current node's head. In our experiments, we noticed that this new feature slightly improves the performance of the classifiers.

5.2. *Results*

The results of our three experiments are shown in Tables 4, 5, and 6. Table 4 shows the results for the first experiment in which we used as classes composed tags. A single classifier is generated from the training data. The top half of the table presents the results for the naive Bayes classifier. The bottom half shows the decision trees based results. For each classifier, the row labeled *All Categories* gives the overall performance figures as reported

by WEKA. The other rows, one for each function tags category, show figures for the performance of the classifiers for tags belonging to that category. We computed the per category figures by simply breaking the predicted composed tags into individual tags and checked if there was a matching individual tag in the correct composed tag. For instance, if the classifier predicted *LGS-DIR-TPC* and the correct composed tag is *LGS-TMP-DIR* then only the LGS and TPC individual tags that correspond to the Grammatical and Form/Function categories are hits. The Kappa statistic is reported only for the *All Categories* row since WEKA only reports it for the entire data set.

Table 4. Performance of Naive Bayes and Decision Trees (Experiment 1).

Category	Training Data			Test Data		
	# Instances	Errors	Accuracy	# Instances	Errors	Accuracy
Naive Bayes	Kappa statistic = 0.8206			Kappa statistic = 0.8080		
All Categories	202311	26655	86.82%	11634	1644	85.87%
Grammatical	118483	3788	96.80%	6907	279	95.96%
Form/Function	66261	17940	72.92%	3902	1109	71.58%
Topicalisation	3715	131	96.47%	261	14	94.64%
Miscellaneous	16630	3966	76.15%	759	187	75.36%
Decision Trees	Kappa statistic = 0.8751			Kappa statistic = 0.8301		
All Categories	202311	18472	90.87%	11634	1444	87.59%
Grammatical	118483	2007	98.31%	6907	187	97.29%
Form/Function	66261	12689	80.85%	3902	1019	73.88%
Topicalisation	3715	180	95.15%	261	16	93.87%
Miscellaneous	16630	3911	76.48%	759	268	64.69%

Table 5 shows the results for the second experiment. In this experiment, we divided the training and test data per category of function tags. Each instance has an associated class in the form of an individual function tag. An instance from Treebank that has a composed tag such as *LGS-TMP* would lead to one instance for the Grammatical and Function/Form categories each. We generated four different classifiers, one for each category.

In Table 6, we show the results for the third experiment. The layout of this table is different from the previous tables for several reasons. First, the training data set used in this case is much larger than what we used in the previous experiments. Computing performance figures for this large training data set was too expensive. Thus, we report performance figures only for the test data. Second, because we are now using all the nodes in the syntactic trees, with and without function tags, the number of instances is the same in all of the four categories. There were a total of **827,193** instances used in the training data set, and **47,333** instances for the test data set. The first column in the table, called *F-Tags*, represents the number of instances that have a function tag from the category indicated by the corresponding row in the table. If we subtract this figure from the total number of instances, we find the number of instances which are treated as NON-F cases. The percentage of the NON-F instances is reported in the second column. We call this the *Baseline* column because it indicates the performance for a baseline classifier, which would simply label all instances

Table 5. Performance of Naive Bayes and Decision Trees (Experiment 2).

Category	Training Data				Test Data			
	# Instances	Errors	Accuracy	Kappa	# Instances	Errors	Accuracy	Kappa
Naive Bayes								
Grammatical	118483	546	99.54%	0.9841	6907	69	99.00%	0.9680
Form/Function	66261	13083	80.25%	0.7516	3902	798	79.55%	0.7366
Topicalisation	3751	0	100.00%	1.0000	261	0	100.00%	1.0000
Miscellaneous	16630	132	99.21%	0.9326	759	3	99.60%	0.9856
Decision Trees								
Grammatical	118483	421	99.64%	0.9877	6907	21	99.70%	0.9901
Form/Function	66261	8929	86.52%	0.8271	3902	639	83.62%	0.7841
Topicalisation	3751	0	100.00%	1.0000	261	0	100.00%	1.0000
Miscellaneous	16630	95	99.43%	0.9494	755	4	99.47%	0.9807

with the most frequent class, the NON-F class. The next columns indicate the number of errors, the classifier performance, and the kappa statistic for the naive Bayes and decision trees classifiers.

Table 6. Performance of Naive Bayes and Decision Trees (Experiment 3).

Category			Naive Bayes			Decision Trees		
	# F-Tags	Baseline	Errors	Accuracy	Kappa	Errors	Accuracy	Kappa
Grammatical	6907	86.41%	2120	95.52%	0.8357	729	98.46%	0.9370
Form/Function	3902	91.76%	4723	90.02%	0.5058	2294	95.15%	0.6405
Topicalisation	261	99.45%	185	99.61%	0.7139	61	99.87%	0.8849
Miscellaneous	759	98.40%	2356	95.02%	0.3093	692	98.54%	0.3318

From the tables we notice high values for Kappa which suggest that both naive Bayes and decision trees offer predictions that are in high agreement with the true, gold values. We also see that the Form/Function category has considerably lower performance values than the other categories. This is mainly because there are more classes/function tags that can be predicted and because of the complex nature of some of the tags (e.g. PRP), which makes them harder to predict.

While the performance figures for naive Bayes and decision trees are close to each other, decision trees based results are better for most of the function tags categories and types of experiments. One explanation why decision trees outperformed naive Bayes is that decision trees are more complex predictive models than naive Bayes. Nevertheless, their greater predicting power comes at the expense of more resources, mainly memory. This was the main issue that drove us to use the High Performance Computer system in our experiments. Blaheta[17] attempted to use decision trees on regular machines but gave up due to memory limitations. Thus, our experiments reported in this paper are the first successful large scale experiments with decision trees for function tagging.

The few cases where naive Bayes outperformed decision trees on the test data are the

topicalisation and miscellaneous categories in experiments 1 and 2. This is because of the sparseness of instances for the two categories in the data set and the overtraining of the decision trees classifiers.

6. Conclusion and Future Work

We presented in this paper a comparison of naive Bayes and decision trees techniques for the task of assigning function tags. The results reported are on perfectly parsed trees from the Penn Treebank corpus. Our experiments show that the two techniques deliver good performance. Although decision trees outperformed the naive Bayes approach, there is a trade-off in the amount of needed resources. Decision trees are more complex and require more resources, such as memory and processing time, than naive Bayes.

As future work, we plan to define more features for the underlying model. In addition, we plan to experiment with feature selection algorithms to find the best subset of features that deliver the best performance. We also plan to integrate function tagging in a state-of-the-art syntactic parser to provide full parsing for a given sentence. Such a full parser could be applied to real world tasks such as Question Answering.

References

1. S. Lappin and H. J. Leass, *An algorithm for pronominal anaphora resolution*, Computational Linguistics, (1994), 20(4):535–561.
2. E. Charniak, K.Knight and K. Yamada, Syntax-based language models for statistical machine translation, in *Proceedings of Machine Translation Summit IX*, (2003).
3. E. M. Voorhees, Overview of the trec 2002 question answering track, in *Proceedings of the Eleventh Text Retrieval Conference*, (2002).
4. Moldovan, D., Harabagiu, S., Pasca, M., Mihalcea, R., Girju, R., Goodrum, R., Rus, V. *The Structure and Performance of an Open-Domain Question Answering System*, in Proceedings of ACL 2000, Hong Kong, October, 2000.
5. A. Bies, M. Fergurson, K. Katz and R. MacIntyre, *Bracketing guidelines for treebank II style*, (Penn Treebank Project, 1995).
6. M. Collins, Three generative, lexicalised models for statistical parsing, in *Proc. 35th Annual Meeting of the Association for Computational Linguistics and Eight Conference of the European Chapter of the Association for Computational Linguistics*, (1997).
7. E. Charniak, A maximum-entropy-inspired parser, in *Proc. of North American Chapter of Association for Computational Linguistics*, (NAACL-2000).
8. D. Klein and C. Manning, Accurate Unlexicalized Parsing. Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics, Sapporo, Japan, (2003).
9. J. Friedman, *On bias, variance, 0/1 - loss, and the curse-of-dimensionality*, Data Mining and Knowledge Discovery, (1997), 1(1):55–77.
10. A. McCallum and K. Nigam, *A comparison of event models for naive bayes text classification*, Workshop on Learning for Text Categorization, (AAAI, 1998).
11. R. Irina, An empirical study of the naive Bayes classifier, in *Workshop on Empirical Methods in Artificial Intelligence*, (IJCAI, 2001).
12. H. Zhang, The Optimality of Naive Bayes, in *Proceeding of FLAIRS*, (2004).
13. F. Esposito, D. Malerba and G. Semeraro, A Comparative Analysis of Methods for Prunning Decision Trees, in *Transactions on Pattern Analysis And Machine Intelligence*, (IEEE,1997), 19(5):476–491.

14. L. Breiman, J. Friedman, R. Olshen and C. Stone, *Classification and Regression Trees*, (Wadsworth,1984).
15. J. R. Quinlan, *Induction of decision trees*, Machine Learning, (1986), 1(1):81–106.
16. J. R. Quinlan, *C4.5: Programs for Machine Learning*, (Morgan Kaufmann Publishers, 1993).
17. D. Blaheta, *Function tagging*, advisor E. Charniak, (Ph.D. Dissertation, Brown University, 2003).
18. D. Blaheta and E. Charniak, Assigning function tags to parsed text, in *Proc. 1st Annual Meeting of the North American Chapter of the Association for Computational Linguistics*, (2000), 234–240.
19. V. Rus and K. Desai, Assigning function tags with a simple model, in *Proc. of Conference on Intelligent Text Processing and Computational Linguistics*, (CICLing, 2005).
20. M. Johnson, A simple pattern-matching algorithm for recovering empty nodes and their antecedents, in *Proc. 40th Annual Meeting of the Association for Computational Linguistics*, (2002).
21. V. Jijkoun and M. De Rijke, Enriching the output of a parser using memory-based learning, in *Proceedings of the ACL 2004*, (2004).
22. T. Brants, W. Skut, and B. Krenn, Tagging Grammatical Functions, in *Proceedings of the Conference on Empirical Methods in Natural Language Processing*,(EMNLP, 1997).
23. J. Ruppenhofer, M. Ellsworth, M. Petruck, C. Johnson and J. Scheffczyk, *FrameNet II: Extended Theory and Practice*, International Computer Science Institute, (University of California at Berkley, 2006).
24. D. Gildea and D. Jurafsky, Automatic labeling of semantic roles, *Computational Linguistics*, (2002), 28(3):245–288.
25. P. Langley, W. Iba and K. Thomas, An analysis of Bayesian classifiers, in *Proceedings of the Tenth National Conference of Artificial Intelligence*, (AAAI, 1992), p223–228.
26. I. Kononenko, Comparison of inductive and naive Bayesian learning approaches to automatic knowledge acquisition, in Wielinga, B., ed., *Current Trends in Knowledge Acquisition*, (IOS Press., 1990).
27. M. J. Pazzani, Search for dependencies in Bayesian classifiers, in Fisher, D., and Lenz, H. J., eds., *Learning from Data: Artificial Intelligence and Statistics*, (V. Springer Verlag, 1996).
28. D. Magerman, *Natural Language Parsing as Statistical Pattern Recognition*, (Ph.D. Dissertation, Stanford University, 1994).
29. M. Collins, *Head-Driven Statistical Models for Natural Language Parsing*, (Ph.D. Thesis, University of Pennsylvania, 1999).
30. Eric W. Weisstein, *k-Statistic*, from MathWorld – A Wolfram Web Resource. http://mathworld.wolfram.com/k-Statistic.html
31. I. Witten and E. Frank, *Data Mining: Practical Machine Learning Tools and Techniques*, 2nd edn. (Morgan Kaufmann Publishers, 2005).
32. M.F. Porter, An Algorithm for Suffix Stripping, in *Program*, (1980), 14(3):130–137.