Using Probabilistic Tree Substitution Grammars

Ben Swanson

January 24, 2011

1 Abstract

Probabilistic Tree Substitution Grammars model the syntactic structure of natural language, and can be learned in an unsupervised manner from annotated sentences such as the Penn Treebank. In this work we use this grammar in a novel approach to Form Function Tagging, using a specialized data structure to efficiently implement the sum-product algorithm. We also extend the existing induction algorithm to a hierarchical form, which we use to simultaneously induce grammars from a corpus split into document classes. We evaluate this model on the task of document dating and show that it outperforms other models commonly employed in previous work. Furthermore, we use the data structure for efficient sum-product calculation in an implementation of a block sampler for grammar induction, and investigate the effects of grammar markovization on experimental performance.

2 Introduction

Due to the requirements of efficient parsing algorithms, most parsing research has focused on Probabilistic Context Free Grammars (PCFG). However, techniques developed for PCFG models can be leveraged on more expressive grammatical formalisms. We apply one such technique, hierarchical modeling, to Probabilistic Tree Substitution Grammars (PTSG), and evaluate its effect on the task of document classification.

Document classification includes such concrete tasks as authorship attribution, author verification, spam filtering, and sentiment analysis. The general formulation of the problem requires a corpus where each unit of text is associated with one or more classes. With this type of data, statistical machine learning techniques can be used to determine the most likely class of a piece of novel text.

One straightforward approach to text classification is language modeling. This involves the characterization of a block of natural language as a set of events and the definition of the joint probability function of those events. These models generally fall into one of two categories, n-gram or syntactic. Due to variability across data sets it is not clear if one type generally outperforms the other. However, the patterns of style captured by these models differ and have been shown to provide complementary information for classification [13]. This work focuses on the use of syntactic language models for document classification by using a PTSG, which can capture lexical dependancies within a syntactic framework.

PTSGs are similar to the PCFG model commonly employed in syntactic parsing in that they define probabilities of rewrite rules for each nonterminal symbol in a grammar. A PTSG is in fact a strict generalization of a PCFG in that it models a superset of rewrite rules. This model has become popular in natural language processing following the recent development of Gibbs sampling techniques for grammar induction [6] [12].

Language modeling with a PTSG involves computing the probability of all derivations of a parse tree. This is equivalent to calculating the inside probability of the root of the tree, and if implemented naively can lead to redundant computation. In Section 4, we present a data structure which allows for efficient determination of the PTSG rules used in all derivations of a given parse tree. We also use this data structure in an implementation of the block sampler presented in [5]. Our implementation avoids the full annotation burden of the grammar transform they employ, and has been packaged for easy third party use.

The recently popularized probabilistic model for PTSG induction is the Dirichlet Process (DP). This stochastic process exhibits a rich-get-richer dynamic which identifies commonly occurring patterns in parse tree data. In applications such as document classification, there is added information in the form of class labels. In order to incorporate these labels we modify the PTSG and its induction algorithm through the use of a Hierarchical Dirichlet Process. In this work we show that tying together the PTSG models of each class with a common prior distribution in this way leads to principled smoothing and better generalization.

We also apply the PTSG to the task of Form Function Tagging, which involves the classification of syntactic non-terminals into linguistically motivated groups. We construct and evaluate a novel PTSG based algorithm which performs this classification with near state of the art accuracy.

3 Related Work

This work combines three relatively distinct threads of research in natural language processing and machine learning. The first is authorship attribution, a common form of document classification. The second is recent work on induction of PTSGs using Gibbs Sampling [6] [12] , and the third is the Hierarchical Dirichlet Process (HDP) of [15].

3.1 Authorship Attribution

Authorship attribution has clear real world application, with such notable historical examples as the disputed authorship of the Federalist Papers and Bill Clinton's unofficial campaign "biography" Primary Colors. The models employed in authorship attribution seek to identify stylometric features which disambiguate one author from another. Examples include [14], which investigates a range of classification methods using n-gram lexical features, and [8], which compares n-gram features with stylometric features such as vocabulary richness and sentence length.

There is also a nascent trend in authorship attribution using syntactic features. Most recently [13] showed not only that syntactic PCFG features give competitive classification accuracy but that syntactic and lexical models can be used in conjunction to improve classification accuracy. It has also been shown by [7] that using syntactic features with longer dependencies (loosely equivalent to PTSG rules) leads to accuracies which outperform not only the simpler PCFG features but also function word and POS tag features. Our work is quite similar to this tree mining technique, but is preferable not only in that it provides a principled probabilistic framework (e.g., for comparing confidence in classification results) but also in the nonparametric formulation which does not require parameter selection for maximum tree size.

3.2 Hierarchical Grammars

To explore new syntactic models for authorship attribution, we extend the PTSG model of [6] to the hierarchical case using the HDP. Such hierarchical grammars have been previously employed by [4]. Our approach is similar in this aspect, but we use the provided class labels to determine the grouping structure of the HDP.

3.3 Form Function Tagging

There are two items of previous work for Form Function tagging which are most relevant. [2] uses a feature based discriminative classifier, conditioning the decision on surrounding nodes or linguistic head words. [10] uses a generative parser to jointly model the parse tree and form function tags, and yields the current state of the art. Our work is similar to this latter technique, differing in our use of Tree Substitution Grammars instead of Context Free Grammars.

4 HDP-PTSG

The formal description of a Tree Substitution Grammar is a four-tuple $G = (T, N, S, R)$. N and T are the nonterminal and terminal symbols of the grammar, $S \in N$ is a unique start symbol, and R is a set of rewrite rules. We assume a familiarity with this model and refer the interested reader to [6] for a full exposition. The basic intuition for those familiar with PCFGs is that we now allow rewrite rules which contain any parse tree substructure.

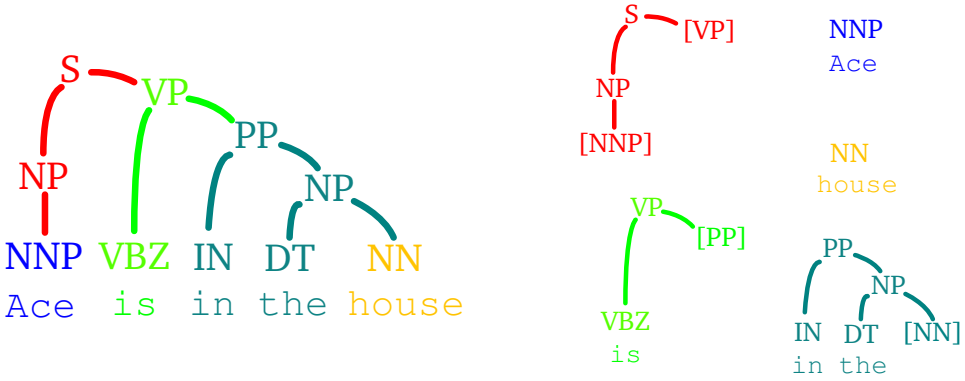

Figure 1: A tree substitution grammar derivation, with its constituent elementary trees on the right. Nodes with brackets around their symbol denote nonterminal substitution sites.

4.1 HDP Extension

The members of R are segments of full parse trees and are referred to as *elementary trees*. PTSG models define the probability of each elementary

tree $e \in R$ using a multinomial distribution $P_s(e)$ for each nonterminal $s \in N$ over rules which rewrite the nonterminal s.

We consider the case where elementary trees are additionally associated with a single discrete label. For document classification this association is determined by the class labels in the data and corresponds to information such as the sentence's author or the time period in which the sentence was written. Let there be K such labels, and associate each elementary tree with a label index k. The HDP-PTSG extends the PTSG by defining multinomial distributions $P_s^k(e)$ for all pairs (s, k).

The probability of a sample of text under a HDP-PTSG is defined as follows. The probability of a corpus C of parse trees is given as

$$P(C) = \prod_{t \in C} P(t)$$

where the corpus probability factorizes over the probability of the parse trees for each sentence. For a parse tree with n nonterminal nodes, there are 2^n derivations which could have produced the tree. The probability of a parse tree is given as the sum over the probabilities of the tree's derivations D_t, and the probability of a derivation factorizes over the multiset of elementary trees it employs as rewrite rules

$$P(t) = \sum_{d \in D_t} \prod_{e \in d} P_s^k(e)$$

where in document classification k is the class label of t.

4.2 Induction of $P_s^k(e)$

While the straightforward estimation algorithm for $P_s^k(e)$ would be to consider all possible derivations of a corpus of trees, the exponential number of derivations for each tree makes this computationally intense. This method has been employed through compressed data structures in [1], yielding state of the art parsing performance but large memory and computational costs. Instead we employ the algorithm of [6] which takes a parse tree corpus and efficiently induces derivations for each tree corresponding to a compact PTSG.

To sample these derivations we treat $\mathbf{e}$, the set of elementary trees in all derivations, as a collection of exchangeable items. We define a set of DP distributed G_s^k, one for each (s, k). The Chinese Restaurant Process (CRP) provides the probability of incremental local changes in the derivations.

Our technique is similar to that of Cohn et al, which induces $|N|$ DPs, one for each nonterminal $s \in N$. This work extends their model to the hierarchical case where we subdivide these DPs for each label $k \in \{1, ..., K\}$, where the K subdivided DPs are linked by a common DP prior H_s, as defined below.

This gives $|N|K$ total DPs, and each elementary tree e in our sampled derivation is mapped to one such DP. This mapping is uniquely defined by the class label k of the section of the corpus in which e is found and e's root nonterminal s. The generative model is now given by

$$\begin{aligned} H_s \mid \gamma_s, P_0 &\sim DP(\gamma_s, P_0) \\ G_s^k \mid \alpha_s^k, H_s &\sim DP(\alpha_s^k, H_s) \\ e &\sim G_s^k \end{aligned}$$

where P_0 is a PCFG distribution with added parameters β_s for each nonterminal symbol, as described in [6]. This serves to bias the induction algorithm towards smaller elementary trees.

The DP is an infinite extension of the finite mixture model, and is therefore a distribution over mixture components, not data items. It is a useful feature of this induction model that these mixture components are delta distributions which give mass to single elementary trees. Such a model creates a one to one mapping between customers and possible dishes which allows us to simplify the integral over tables when calculating the likelihood of a data item.

After performing this grammar induction procedure, we can estimate the multinomial distribution $P_s^k(e)$ by truncating the infinite mixture models, as a mixture of delta functions is equivalent to a multinomial distribution. Since our mixture components are delta distributions, truncating this distribution is equivalent to including or excluding elementary trees in our grammar. We include all trees which have non-zero counts in the sampled derivations, as well as all PCFG rules which appear in the training data but do not appear in a derivation. The remaining mass is distributed equally among all members of the multinomial.

4.3 Sampler Implementation

[6] describe a Gibbs sampling algorithm for PTSG induction, and [5] provides an alternative algorithm which exhibits better convergence. These algorithms make use of certain modeling techniques which allow efficient sampling. Unfortunately, one of these techniques does not generalize to the HDP-PTSG. To explain this we use the metaphors of the CRP, assuming a

familiarity with the concept of restaurants, customers, dishes, and tables as outlined in [15].

In the induction algorithm of [6] dishes are defined to be delta distributions, which allows the use of the CRP for sampling. The CRP defines the probability of an elementary tree e which maps to G_s^k as outlined above. If this G_s^k is drawn from a DP with concentration parameter α and base distribution H, the probability of e is given as

$$P(e|\mathbf{e}^-, \alpha, H) = \frac{n_e + \alpha H(e)}{n_\bullet + \alpha}$$

where $\mathbf{e}^-$ is the set of previous elementary trees which map to (s, k), n_e is the count of e in $\mathbf{e}^-$, and $n_\bullet$ is the size of $\mathbf{e}^-$. When sampling derivations as outlined in existing PTSG sampling algorithms, we iteratively choose between possible modifications of our current multiset of elementary trees by considering ratios of these probabilities. We omit the specifics as they follows identically from previous work.

This probability function is the sum of two terms which have allegorical equivalents in the CRP. The first, $\frac{n_e}{n_\bullet+\alpha}$, is the probability that the customer e sits at a table which is already occupied. The second, $\frac{\alpha H(e)}{n_\bullet+\alpha}$, is the probability that the customer sits at a new table.

The subtlety which arises when using another DP as the prior distribution H is that in the second case, when a customer sits at a new table, we must seat a customer in the restaurant associated with H. This involves an additional sampling step in which we sample against the ratio $\frac{\alpha H(e)}{n_e}$ to decide if a customer is seated in H's restaurant. When removing a customer from a restaurant, we must decide if this leaves a table empty, in which case the base distribution must also have a customer removed. The simplest approach is to keep a table index for each customer, but this is not ideal. This subtlety has been studied in previous work by [3] and we employ their technique of maintaining a histogram of table counts for each restaurant in the CRP, which allows for a smaller memory footprint and less book-keeping.

5 Language Modeling

Probabilistic models of natural language such as the HDP-PTSG can be used to calculate the likelihood of an arbitrary block of text under a class specific model. If this computation is performed for each of a set of classes, document classification can be done by choosing the class which gives the highest likelihood.

Full language modeling with a grammatical model would require consideration of all possible parse trees for a sentence. For the sake of computational efficiency, in our work we approximate the language model score by scoring only the most probable parse of a PCFG parser.

With the HDP-PTSG, the likelihood of a set of parse trees $\mathbf{t}$ given a class k is

$$P(\mathbf{t}|k) = \prod_{t \in \mathbf{t}} \sum_{d \in D_t} \prod_{e \in d} P_s^k(e)$$

The exponential number of possible derivations in D_t makes this value computationally intensive to calculate. Fortunately, the sum-product algorithm can be used to perform this calculation efficiently using dynamic programming, as the probability of a parse tree is the marginal probability of the root node. This is equivalent to running the inside portion of the inside-outside algorithm and reading off the inside probability of the tree's root.

Using the sum-product algorithm on a tree t with TSG rules requires a subroutine which runs for each node s in the tree. This subroutine must return all possible TSG rules which could have been used to rewrite s in a derivation of t. The naive algorithm simply performs a traversal of each elementary tree with probability in $P_s^k(e)$. While this algorithm works relatively well for evaluating small blocks of text, it does not scale well to large test documents.

5.1 Compact Overlay Dictionary

We present a data structure which we call the Compact Overlay Dictionary (COD) which efficiently calculates the subroutine necessary for computing the inside probability of a parse tree under a PTSG. Credit for the creation of algorithm is due to the illustrious Micha Elsner. The input is a reference parse tree t and a nonterminal node n in t with symbol s. The output is $\mathbf{e}' \subset \mathbf{e}$, the elementary trees which could have been used to rewrite n in a derivation of t, and the nodes in t which correspond to the non-terminal leaves of each member of $\mathbf{e}'$. A useful way to visualize $\mathbf{e}'$ is as the members of $\mathbf{e}$ which we could overlay consistently on top of t.

A COD is precomputed for s using the set $\mathbf{e}$ of elementary trees in the PTSG. Figure 1 shows an example COD computed for the symbol S given the grammar consisting of the three elementary trees pictured on the right.

The COD is a tree structure where each node keeps track of information which can be used to confirm or deny the ability to overlay grammar rules.

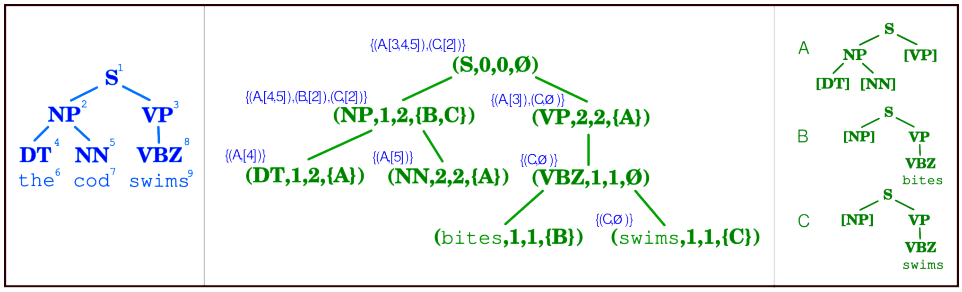

Figure 2: A Compact Overlay Dictionary in the center constructed from the set of elementary trees A, B, and C on the right. When the tree on the left queries this COD for node 1, the COD reveals that only A and C overlay the tree consistently. It also returns the indexes of the nodes where these trees' substitution sites overlay, allowing efficient implementation of the inside-outside algorithm. The intermediate recursive results of the lookup algorithm are shown in upper-left superscript, with the return value of the lookup as the superscript of the COD's root node. The node containing *bites* does not have a superscript because the lookup algorithm does not visit that node.

Formally, a COD node is a four-tuple (N, i, j, E).

As a motivating example, consider the first left child in the COD pictured in Figure 1. This node encodes NP symbols produced with index 1 in rules of arity 2, corresponding to the values of N, i, and j respectively. Its value for E contains all of the elementary trees which have leaves at this location, which for our small grammar is the set consisting of trees B and C.

5.1.1 COD Construction

To construct a COD we require only $\mathbf{e}$, the domain of one of our P_s^k multinomial distributions. All of these elementary trees share the same root nonterminal symbol s.

The recursive algorithm for COD construction is initialized with a one node tree consisting of $(s, 0, 0, \emptyset)$. Construction proceeds by iteratively adding each elementary tree $e \in \mathbf{e}$, all of which are necessarily rooted at a node labeled s. This insertion traverses e top down, updating the COD for each node in e.

5.1.2 COD Lookup

The purpose of the COD data structure is to allow efficient determination of overlays for a query tree t. This is done by a postorder traversal of the COD tree over nodes which are consistent with t. This defines a subgraph of the COD tree over which the lookup algorithm progresses. For the example shown in Figure 1, this includes all nodes except the one containing the terminal symbol *bites*.

The COD lookup algorithm outputs a set of elementary trees along with the overlay sites of their non-terminal leaves. For clarity we limit our description to the computation of the returned elementary trees. The determination of leaf overlay points is demonstrated by Figure 2.

Assume a set query tree t, and a subgraph of the COD tree which is consistent with t. There is a one to one mapping from nodes in the COD subgraph to nodes in t. We use a prime notation to refer to the fact that a COD node n maps to a node n' in t.

An *overlay set* of a COD node n is the set of elementary trees which are consistent with the subtree rooted at n'. Let P be the fourth tuple element of n. Given the overlay sets C_i of this node's k children, the complete list of elementary trees which can overlay the subtree rooted at n' is given as

$$O = P \cup (\bigcap_{i=0}^{k} C_i)$$

This captures the rule that to overlay consistently at n, an elementary tree must either have a non-terminal leaf at n or be consistent with all of the children of n.

6 Form Function Tagging

Form Function Tagging is the task of assigning class labels to nonterminal nodes which indicate semantic characteristics of the text in that subtree. For example, a prepositional phrase PP may be tagged as PP-LOC or PP-TMP to signify locative or temporal semantics of its subtree. This information is potentially useful in constructing a representation of the meaning of a sentence, as shown in Figure 3.

The classifier is learned by simply running the PTSG induction algorithm of Cohn et al on Form Function Tag (FFT) annotated trees. We will refer to this model as the FFT-PTSG. To perform classification, we take as input a sentence with its syntactic parse, and outputs a tag for each non-terminal

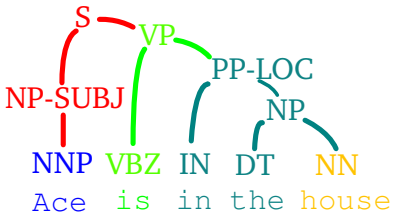

Figure 3: This FFT-PTSG derivation represents intuitively that Ace is the subject of this sentence, and that the house is a location that something is in.

or tells us that no tag should be present.[1] Following previous work [2] we used the gold standard parse trees of the Penn Treebank, which come with human annotated FFT labelings for evaluation.

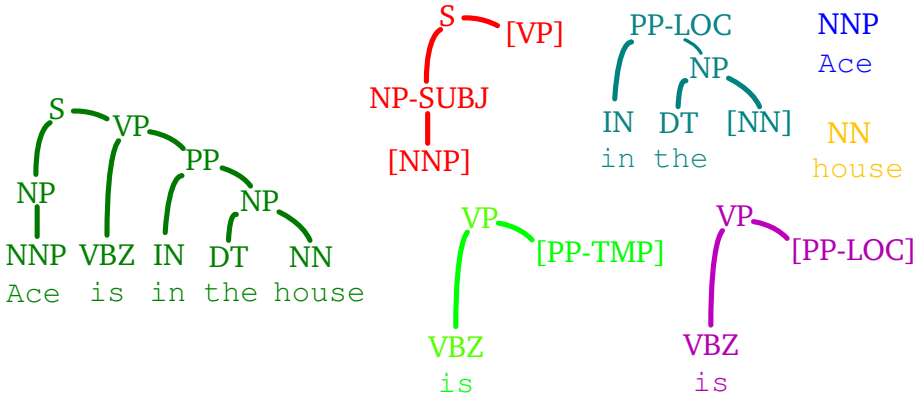

Figure 4: An FFT-PTSG can decide a tagging by using evidence from multiple elementary trees. While the VP trees give evidence for two differnt tags of the PP, the PP tree knows that "in the ____" is most often locative and not temporal.

We use the input parse tree to define the set of derivations D in our FFT-PTSG that if stripped of their tags would be identical to the input tree. In Figure 4, several elementary trees are shown that could derive the parse tree on the left. We say that an elementary tree can *overlay* the tree at a node n if the elementary tree is consistent with the structure of the subtree rooted at that node with the exception of tag suffixes. For example, both VP elementary trees can overlay the tree in Figure 4 at the VP node,

[1] For clarity's sake, in the following discussion when we refer to a node as tagged, we are implicity including the possibility of a "No Tag" option unless otherwise stated.

as both PP-TMP and PP-LOC are consistent with the PP tag.

Given an input tree T which we refer to as the *reference tree*, for each node N and its set of possible tags S_N for that symbol, we want to find

$$argmax_{N' \in S_N} P(N = N'|T)$$

where we use the equality operator to denote the event that N is tagged as N'. We have that

$$P(N = N'|T) = \frac{P(T, N = N')}{P(T)} \propto P(T, N = N')$$

We use the PTSG to calculate this joint probability. Specifically, our grammar defines the probability of all derivations, and we must compute the portion of this probability provided by derivations for which $N = N'$

$$P(T, N = N') = \sum_{d \in D} P(d, N = N') = \sum_{d \in D'} P(d)$$

where D' is the set of derivations which have $N = N'$. D' can alternatively be represented as a list of elementary trees paired with the sites where the root node and nonterminal leaf nodes of that tree overlay the reference parse tree. We rephrase the calculation above in terms of this list $E' = List((e, r, L))$ where e is an elementary tree, r is the node in the reference tree where the root of e overlays, and L is a list of the nodes where the nonterminal substitution sites of e overlay the reference tree.

$$\sum_{d \in D'} P(d) = \sum_{e \in E'} \alpha(r) P(e) \prod_{l \in L} \beta(l)$$

where α and β are the outside and inside probabilities respectively, and the PTSG provides $P(e)$ directly.

To perform tagging efficiently, we note that every elementary tree which is used in a derivation will appear in several such sums, one for each of its constituent nodes. We use the COD to calculate inside and outside probabilities for each node in the reference tree, and then consider each elementary tree in turn. We compute its contribution to the above sum and accumulate this value in a vote for each tagging it defines. When all elementary trees have been considered in this manner, we have the unnormalized values for $P(T, N = N')$ for all nodes and all tags and can easily find the maximum probability tag.

7 Experiments

7.1 Document Dating

	UNI-G	BI-G	CHAR-N	PCFG	PTSG	HDP-PTSG
Acc	22.11	40.7	36.15	28.68	42.55	44.14
MSE	3.62	1.96	3.37	36.76	1.73	1.73

Figure 5: Experimental Results for document dating. The generative models employed, from left to right in the table, are a unigram model, a bigram model, a character 8-gram model, a simple Probabilistic Context Free Grammar estimated directly from the data with add alpha smoothing, a set of PTSGs trained for each era independently, and the HDP-PTSG developed in this work. Each model trains a era specific generative model and chooses the era whose model gives the highest likelihood to a test text. The exception is the bigram model which uses KL-Divergence, as we found that this method outperformed language modeling for this task. We report both classification accuracy (Acc) and mean squared error (MSE). MSE is calculated with the error function defined in Section 5.

We apply the HDP-PTSG to the task of determining the date of publication of a piece of text, and frame the problem as a variant of authorship attribution. We use a corpus of books downloaded from the Gutenberg project which are tagged with their publication dates and place them into bins of 25 year eras. Each era is equivalent to the traditional concept of an author, and so the experimental task is to determine the correct era assignment for a block of novel text.

7.1.1 Experimental Setup

Our corpus consists of 96 English books by 77 authors, using 71 texts for training and 25 for test. The eras range from 1575 to 1900, with each 25 year era except 1625 represented in our data set. The mapping is performed such that any book published between year X and year $X + 24$ inclusive is included in the bin labeled X. We randomly sample 5000 sentences from each era in the training set leading to approximately 60K training example sentences. The full texts of the test set are chunked into continuous blocks of 100 sentences.

To prepare the data we used MXTerminator to segment the sentences and the self-trained Charniak parser to provide maximum probability parses. Lexical items which appear less than five times in the training set were

deterministically mapped to unknown word categories using an algorithm extracted from the open source split merge parser of [11]. Lexical items in the test set which did not appear in this new training set were then mapped to unknown word categories with the same algorithm.

We then performed Head Out Trinarization, which is a markovization scheme. This tree transformation begins by using lexical head finding rules to determine for each node n the child node that provides n's head word. This parent-child relationship is maintained, and the remaining child nodes are expanded in a binary branching structure, as illustrated in Figure 3. This process expands the number of grammar symbols by a fator of 3, and gives the property that each PCFG rule has a maximum arity of 3.

We have found in several independent ongoing experimental applications of PTSGs that Head Out Trinarization improves performance, as it mitigates the negative effect of sparse data. The grammar induction procedure provides a complementary role of finding commonly occurring substructures which often reintroduce the dependencies removed by the markovization, but only when such dependencies are merited by the data.

For both the HDP-PTSG and PTSG we use unsegmented trees for initialization and run for 1000 training iterations with no smoothing using the Gibbs sampling algorithm of [6]. We resample the concentration parameters of all DPs at every 10th iteration using the technique in [9] which uses the histogram of extant dishes to provide the posterior probabilty of α. We propose α from a lognormal distribution and use Metropolis Hastings as our sampling algorithm. The β parameters of the prior P_0 described in [6] are also resampled every 10 iterations, using a binomial distribution with a uniform beta prior.

7.1.2 Discussion

We compare the HDP-PSTG to a PTSG trained independently for each era, and also against several baseline language models. Accuracies are lower than many published document classification experiments, which is partially due to our binning approach; under our formulation, a book published in 1824 is as different from a book from 1825 as it is from one published in 1849. To show classification ability more clearly we report not only era assignment accuracy but also mean squared error (MSE) with a loss function equal to the integer distance between the indexes of the correct and estimated eras. For example, a book whose true era is 1800 and is classified into the 1850 bin is off by two eras and would incur a squared error of 4.

It is difficult to compare these results with previous authorship attribu-

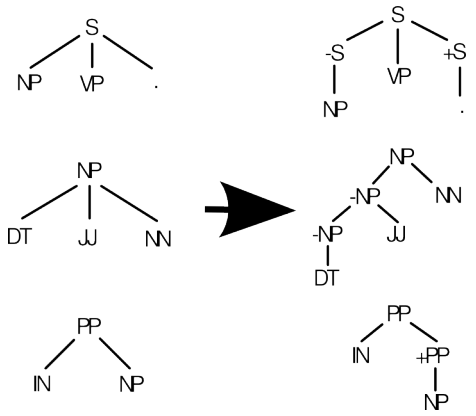

Figure 6: Head Out Trinarization

tion tasks, as different corpora exhibit drastically different levels of difficulty [8]. This disparity in difficulty arises from many factors such as topic similarity and corpus size. In the task of document dating there is the added problem that we are attempting to perform stylometric modeling of a set of authors rather than one single author. We leave the discussion of the facets of the document dating task to a separate publication, and use it here only as a motivating example for the comparative ability of the HDP-PTSG against other text classification models.

The HDP-PTSG outperforms a broad range of classifiers which have been used in previous work for authorship attribution. Of these, the PCFG, PTSG, and HDP-PTSG are syntactic models which use the parsed corpus, while the remaining models use only the lexical items. The difficulty of document dating yields low accuracies, but all models greatly outperform a random assignment baseline which would yield an accuracy of 7.7%. As mentioned above, the MSE criteria gives a better picture of the ability of these models to perform document dating.

The highest performing models are the PTSG and HDP-PTSG, which demonstrates the superiority of the richer Tree Substitution Grammar model over the simple PCFG. However, the difference in accuracy is relatively small and they give the same mean squared error.

To further motivate the advantage of the HDP-PTSG we investigated experimentally the benefits of the smoothing it provides. By taking snapshots of the PSTG at intervals of 100 iterations, we observed that the accuracy of the resultant classifier decreased over time. This is evidence of overfitting,

as the PTSG induction system for a single class is unaware of the grammars learned for other classes. The HDP-PTSG, on the other hand, shows steady improvement in accuracy over the same sequence of snapshots, which illustrates the smoothing effect of hierarchical models discussed in [16].

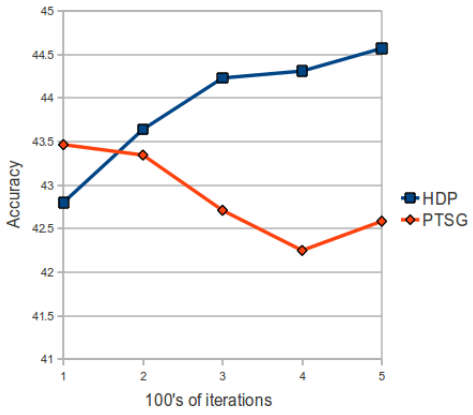

Figure 7: Dating Accuracy vs Sampling Iterations

Another benefit of the HDP-PTSG is that it constrains the class specific PTSGs to share the same vocabulary of elementary trees. When the PTSGs are trained separately they often find different subsets of the exponentially large space of possible PTSG rules to use in derivations, as the rich-get-richer sampling scheme can easily become trapped in local optima. While our experiments show that these optima are roughly equivalent in terms of classification performance, it makes it difficult to perform automatic stylometric comparisons of different eras. This can be easily observed by examining the induced PTSG grammars for the symbol S, where we observe that different classes find similar but distinct representations of high-level sentence patterns.

7.2 Form Function Tagging

We evaluated our Form Function Tagging algorithm on section 23 of the Penn Treebank. All words which occured less than five times were mapped deterministically to unknown word categories. We experimented with head out trinarization as well, and found it to improve performance by a small amount. Accuracy is calculated by only considering nodes which have tags in the gold data; including all nodes and giving credit for a correct choice

	PTSG	PTSG-TRI	BLAHETA	GABBARD
Acc	85.86	86.38	88.28	90.78

Figure 8: Experimental accuracies for Form Function Tagging. This work comprises the PTSG and PTSG-TRI models.

of no tag would give much higher results. This is done to compare with previous work, and is a more illuminating metric overall. We were unable to outperform previous work, but our scores provide clear evidence that our algorithm is capable at this supervised node tagging problem.

There are linguistically motivated categories into which the different form function tags may be classified. When our system is compared with other models, the category with the most room for improvement is the semantic category. To understand why this class is troublesome for a TSG, consider the case of a prepositional phrase (PP) whose preposition is "in". In order the decide between the TMP and LOC tags, the disambiguating information comes from the semantics of the noun phrase following the word "in". If the phrase is "in the house" then the locative tag (LOC) is correct, while "in an hour" would indicate that the prepositional phrase is temporal (TMP). A PTSG must model this information by learning specialized rules which include the lexical items "house" and "hour". This is unlikely given that each of these specific words will not be common enough to seed the rich get richer process of the induction algorithm. We have investigated several methods for incorporating this information into the grammar, but have not been able to effect a performance increase.

8 Conclusion

In this paper we present the HDP-PTSG, a hierarchical extension of the PTSG. We outline the modifications to the probabilistic model of grammar induction and the additional implementation considerations. The HDP-PTSG and a non-hierarchical PTSG are evaluated on the authorship attribution variant of document dating, and compared against several common authorship attribution models.

The use of Tree Substitution Grammar based models for document classification is novel, and outperforms models employed in previous work. This has added significance in that previous work has shown that an ensemble of models capturing different stylometric features is important for author iden-

tification, and syntactic models such as ours are relatively under-represented in existing published research.

Additionally, we describe a novel data structure called the Compact Overlay Dictionary which eliminates redundant computation in the inside-outside algorithm using Tree Substitution Grammars. This data structure is widely applicable as the inside-outside algorithm is a core technique of syntactic modeling.

We apply the PTSG to Form Function Tagging and demonstrate its capability for this task. While we do not read state of the art, we demonstrate that our algorithm is competative and identify a promising avenue for improvement.

References

[1] Mohit Bansal and Dan Klein. Simple, accurate parsing with an all-fragments grammar. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, ACL '10, pages 1098–1107, Morristown, NJ, USA, 2010. Association for Computational Linguistics.

[2] Don Blaheta and Eugene Charniak. Assigning function tags to parsed text. In *Proceedings of the First Conference of the North American chapter of the Association for Computational Linguistics (NAACL '00)*, pages 234–240, 2000.

[3] Phil Blunsom, Trevor Cohn, Sharon Goldwater, and Mark Johnson. A note on the implementation of hierarchical dirichlet processes. In *Proceedings of the ACL-IJCNLP 2009 Conference Short Papers*, pages 337–340, Suntec, Singapore, August 2009. Association for Computational Linguistics.

[4] Jordan Boyd-Graber and David M. Blei. Syntactic topic models. In *Neural Information Processing Systems*, 2008.

[5] Trevor Cohn and Phil Blunsom. Blocked inference in bayesian tree substitution grammars. In *Proceedings of the ACL 2010 Conference Short Papers*, ACLShort '10, pages 225–230, Stroudsburg, PA, USA, 2010. Association for Computational Linguistics.

[6] Trevor Cohn, Sharon Goldwater, and Phil Blunsom. Inducing compact but accurate tree-substitution grammars. In *Proc. NAACL*, 2009.

[7] Sangkyum Kim, Hyungsul Kim, Tim Weninger, and Jiawei Han. Authorship classification: a syntactic tree mining approach. In *Proceedings of the ACM SIGKDD Workshop on Useful Patterns*, UP '10, pages 65–73, New York, NY, USA, 2010. ACM.

[8] Kim Luyckx and Walter Daelemans. Authorship attribution and verification with many authors and limited data. In *Proceedings of the 22nd International Conference on Computational Linguistics - Volume 1*, COLING '08, pages 513–520, Stroudsburg, PA, USA, 2008. Association for Computational Linguistics.

[9] Steven N. Maceachern and Peter Müller. Estimating Mixture of Dirichlet Process Models. *Journal of Computational and Graphical Statistics*, 7(2):223–238, 1998.

[10] Ryan Gabbard Mitchell Marcus and Seth Kulick. Fully parsing the penn treebank. In *In Proceedings of HLT/NAACL 2006*, pages 184–191, 2006.

[11] Slav Petrov, Leon Barrett, Romain Thibaux, and Dan Klein. Learning accurate, compact, and interpretable tree annotation. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*, pages 433–440, Sydney, Australia, July 2006. Association for Computational Linguistics.

[12] Matt Post and Daniel Gildea. Bayesian learning of a tree substitution grammar. In *Proceedings of the ACL-IJCNLP 2009 Conference Short Papers*, pages 45–48, Suntec, Singapore, August 2009. Association for Computational Linguistics.

[13] Sindhu Raghavan, Adriana Kovashka, and Raymond Mooney. Authorship attribution using probabilistic context-free grammars. In *Proceedings of the ACL 2010 Conference Short Papers*, ACL '10, pages 38–42, Morristown, NJ, USA, 2010. Association for Computational Linguistics.

[14] Efstathios Stamatatos. A survey of modern authorship attribution methods. *J. Am. Soc. Inf. Sci. Technol.*, 60:538–556, March 2009.

[15] Y. W. Teh, M. I. Jordan, M. J. Beal, and D. M. Blei. Hierarchical Dirichlet processes. *Journal of the American Statistical Association*, 101(476):1566–1581, 2006.

[16] Yee Whye Teh. A hierarchical bayesian language model based on pitman-yor processes. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*, ACL-44, pages 985–992, Stroudsburg, PA, USA, 2006. Association for Computational Linguistics.