Discriminative, Syntactic Language Modeling through Latent SVMs

Colin Cherry and Chris Quirk
Microsoft Research
One Microsoft Way
Redmond, WA, 98052
{colinc,chrisq}@microsoft.com

Abstract

We construct a discriminative, syntactic language model (LM) by using a latent support vector machine (SVM) to train an unlexicalized parser to judge sentences. That is, the parser is optimized so that correct sentences receive high-scoring trees, while incorrect sentences do not. Because of this alternative objective, the parser can be trained with only a part-of-speech dictionary and binary-labeled sentences. We follow the paradigm of discriminative language modeling with pseudo-negative examples (Okanohara and Tsujii, 2007), and demonstrate significant improvements in distinguishing real sentences from pseudo-negatives. We also investigate the related task of separating machine-translation (MT) outputs from reference translations, again showing large improvements. Finally, we test our LM in MT reranking, and investigate the language-modeling parser in the context of unsupervised parsing.

1 Introduction

A language model is an important tool in any task involving natural language generation. Language modeling has been especially useful in speech recognition and machine translation; by modeling the likelihood of generated sentences, language models greatly increase fluency, and provide contextual disambiguation between possible interpretations of the input.

Given the success of discriminative training in many machine learning tasks, there has been recent interest in discriminative language modeling. It is possible to train a language model that corrects errors in a particular task (Gao et al., 2006; Collins et al., 2005); however, if one wants to build a general-purpose, discriminative language model, the question becomes: between what should the system discriminate? What are the negative examples?

Okanohara and Tsujii (2007) propose an interesting answer to this question, where positive examples are taken from a corpus, while negative examples are sampled from an n-gram language model. In that work, they confirm that native speakers can easily discriminate between human-generated and machine-generated sentences, as sampled sentences include both syntactic disfluencies and semantic nonsense, such as:

- Basically, we are a fighter jet.
- The shortage of topsoil moisture in a personal basis, unlike his book, and extended last summer's semiconductor's partner in a big banks.

Discriminating between sampled and human sentences has the appeal of specifically addressing weaknesses of n-gram language modeling by building a system that selects the most human-looking sentence when all the candidates contain likely n-grams. It is also appealing as a challenging classification problem, as the obvious representation to capture surface structure, a bag of n-grams, has been rendered nearly useless by the careful construction of the negative set. Using an online training method to learn from 500k examples, Okanohara and Tsujii show that a large margin classifier built over bigrams of automatically-induced segment clusters can achieve a classification accuracy of 74.11, with

a standard bag-of-trigrams system built using a polynomial kernel not far behind at 73.65.

Okanohara and Tsujii dismiss parsing as a solution to this classification task after demonstrating that two parsers can successfully parse all negative examples. This seems uncharitable; to avoid brittleness, broad coverage parsers use highly permissive grammars, and can parse almost any input. They are not meant to reject ill-formed sentences. A more fair assessment would judge inputs according to the probabilities of the returned parses. Generative parsers calculate $max_z P(x, z)$ in their search for the best tree z for an input sentence x. This score can be viewed as a max approximation to the language model $P(x) = \sum_z P(x, z)$. This form of language model, with its syntactic hidden structure, might be able to outperform n-gram methods by capturing long-distance dependencies. Unfortunately, even when tested in domain, scores from generative Treebank parsers fail to consistently rank real sentences above pseudo-negatives, as we show in §5.2.

There have been other indications that parse scores do not always behave as useful language models. While investigating the poor performance of parser score as a re-ranking feature for statistical machine translation (SMT), Och et al. (2004) discovered that their Treebank parser tends to rank MT outputs above human reference translations. This is attributed to the fact that MT outputs use fewer unseen words, which causes the parser's terminal productions to dominate any syntactic distinctions. Attempts to normalize for word choice did not improve the performance of parser probability as a translation feature. Sentence length is another factor that could produce spurious differences in parser probability; however, SMT systems already include a length term in their discriminatively weighted linear models, so this does not explain the poor performance of the parser feature.

As a response to the above issues, we train a parser specifically so that its parse score reflects the quality of its input. We adopt latent support vector machines, or LSVMs (Felzenszwalb et al., 2008), to train our parser with a classification objective. The LSVM is tasked with altering the weights on a context-free grammar to transform our parser into a classifier, where positive sentences produce high-scoring parse trees, while negative sentences do not. In this setting, the classification objective over-rides Treebank accuracy as a parsing goal. One can view our approach as a form of unsupervised parsing, in the sense that we are not training the parser to match Treebank annotations. In fact, we are able to train our parser-classifier with only a part-of-speech dictionary and a binary-labeled training set. However, we show that the classifier does benefit substantially from the use of a Treebank-derived grammar. We demonstrate superior accuracy when discriminating human sentences from n-gram samples. Furthermore, we apply our models to the related tasks of recognizing SMT output, and SMT reranking.

We begin by reviewing related research areas, including unsupervised parsing and structured language modeling. We then describe the latent SVM as a general-purpose classifier, followed by the particulars of our parser-classifier. We present results in a number of tasks related to discriminative language modeling: sample detection, SMT output detection, and SMT reranking. We also examine the Treebank accuracy of the latent trees learned during classifier training. Finally we conclude and speculate on future work.

2 Background

2.1 Unsupervised parsing

Our method is similar to unsupervised parsing in that we train a parser using sentences that are not annotated with parse trees. The main difference between our approach and unsupervised parsing is the motivation behind inducing trees. In unsupervised parsing, the tree is the end product and progress is measured by comparing to the Treebank. In our work, the tree is a means to an end; improved classification is our true goal, and the tree is useful to the extent that it can improve accuracy. We also differ from past work in unsupervised parsing in terms of scale. We train over tens of thousands of sentences, with no sentence-length limit, and leave punctuation intact. However, our work is influenced by unsupervised parsing. We briefly review a number of relevant methods below.

Chen (1995) describes a Bayesian model for grammar induction, where the resulting grammar is employed for language modeling. This approach

shares our motivation, but employs a generative technique to maximize the likelihood of a corpus; it does not make use of negative examples. A number of attempts have been made to induce context-free constituency structures, with the goal of matching Treebank annotations (Carroll and Charniak, 1992; Klein and Manning, 2004). These generative models share little in common with our discriminative approach, but we do borrow their basic grammar structure for our uninformed G_x grammar in §4. Our method is similar to the contrastive estimation approach of Smith and Eisner (2005), which also employs automatically-generated negative examples. However, they construct negative examples by mutating their positive sentences in ways designed to improve Treebank accuracy, while we sample incorrect, but locally likely, sentences to improve language modeling.

2.2 Structured Language Modeling

Since we are parsing to judge a sentence, the subfield of structured language modeling is highly related to this work. Structured language models begin with the intuition we presented in §1, namely that generative parsing models can also serve as language models; however, they generally modify the parser to account for left-to-right incremental parsing (Chelba and Jelinek, 1998; Roark, 2001), and the use of lexical heads to capture long-distance language-model contexts (Chelba and Jelinek, 1998; Charniak, 2001). With the increasing popularity of parser-like syntactic translation systems (Galley et al., 2004), we feel that left-to-right processing is no longer a primary concern, and following Charniak et al. (2003), we make no special accommodations for it. We also do not lexicalize our grammar, eliminating the possibility of conditioning one token on another directly in a language-model-like fashion. Instead, we test how well we can achieve our classification objective by re-weighting unlexicalized context-free productions. One could always hope to extend our approach by re-introducing lexicalization. Collins et al. (2005) employ similar parse features to our own in a discriminative language model for speech recognition, but they do not dynamically alter the grammar as we do.

2.3 Machine Translation Evaluation

Recognizing pseudo-negative sentences sampled from an n-gram language model is similar to the task of recognizing machine-translated sentences, which is is explored by Kulesza and Shieber (2004) in the context of MT evaluation. In that work, both a response sentence and a reference translation are provided as input, and an SVM is trained to recognize those responses generated by SMT. They use features commonly employed in automatic MT evaluation, such as n-gram precision and word-error-rate. We explore the translation recognition task as well, with a drastically different feature set and without the benefit of a reference translation, in order to test our language models' discriminative power.

3 Latent SVM

We are interested in training a parser to improve the accuracy of our sentence classification task. This is an instance of classification with latent structure. We adopt a solution, the latent SVM, that has been successful in the task of object detection in images (Felzenszwalb et al., 2008). In that case, the latent SVM is used to train an image classifier with a latent part-model. The classifier benefits from the notion that objects have parts (people have heads, arms and legs; cars have windows and wheels), but the system designers do not need to specify or label these parts. Instead, a part model is learned during classifier training, guided by the classification objective. Similarly, we want our classifier to benefit from the notion that sentences have parse trees, and to learn a parser suited to our classification task.

3.1 Training Algorithm

In a latent SVM, the structure prediction algorithm (in our case, a parser) produces a classification while simultaneously searching for a latent structure (a parse tree). One can abstract away the search process by stating that the parser solves the problem:

$$f_{\beta,Z}(x) = \max_{z \in Z(x)} \beta \cdot \Phi(x, z) \qquad (1)$$

where x is the input sentence, and $Z(x)$ is the set of all parse trees for x. $\Phi(x, z)$ is a feature vector for a sentence-tree pair, which decomposes over dynamic programming, and β is a parameter vector, providing

a weight for each feature. Note the use of $\max$, as opposed to the argmax usually used in parsing. The search for the highest-scoring latent tree z is conducted only to find a score for the input sentence x. That is, each input is scored according to its best latent structure, and the classification is determined by the sign of that score.

Given the training set $(\langle x_1, y_1\rangle, \ldots, \langle x_n, y_n\rangle)$, $y \in \{-1, 1\}$, we wish to train the parser parameters β so that positive examples receive positive $f_{\beta,Z}$ scores, while negative examples receive negative scores. Adopting a large-margin formalism, we desire a β that minimizes:

$$\lambda||\beta||^2 + \sum_{i=1}^{n} \max\left(0, 1 - y_i \cdot f_{\beta,Z}(x_i)\right) \quad (2)$$

where the second term implements a soft-margin. Each misclassification incurs a penalty proportional to the degree with which it violates the margin.

The objective in (2) is not convex, due to the inner max over Z in $f_{\beta,Z}$. However, Felzenszwalb et al. (2008) observe that (2) can be made convex by fixing the latent structure z for each positive example. This implies a coordinate descent algorithm, alternating between optimizing z for positive examples, and optimizing β. The descent is guaranteed to arrive at a local minimum in (2). Optimizing β with fixed structures for only the positive examples is not entirely straightforward, so Felzenszwalb et al. implement an approximation, which fixes z for all examples. This results in an alternative coordinate descent, iterating over the following two steps until convergence:

1. Given the current β, find the $\max$ structure for each training example:

$$\forall i : z_i \leftarrow \operatorname{argmax}_{z \in Z(x_i)} \beta \cdot \Phi(x_i, z)$$

2. Using features derived from $z_{[1\ldots n]}$, find new parameters β, which minimize the standard SVM classification objective:

$$\lambda||\beta||^2 + \sum_{i=1}^{n} \max\left(0, 1 - y_i \cdot \beta \cdot \Phi(x_i, z_i)\right)$$

The above algorithm can be implemented easily, using a parser to find $\max$ structures, turning those structures into labeled feature vectors $\langle \Phi(x_i, z_i), y_i\rangle$, and then training a linear SVM to classify those vectors. The resulting parameters β provide weights for the next round of parsing.

We have found that this approximation does not necessarily arrive at a local minimum; the coordinate descent is prone to oscillation on negative examples. Consider a training pair $\langle x_i, y_i\rangle$ where $y_i = -1$. In the first iteration, the SVM reacts to the parser by reducing the score for the max structure $z_{i,1}$. In the next iteration, the parser finds a $z_{i,2}$, which receives a high score because it shares few productions with $z_{i,1}$. The SVM reduces the score for $z_{i,2}$, forgetting $z_{i,1}$ and leaving the parser free to return to $z_{i,1}$ on the next iteration. We altered the approximation by maintaining a structure memory $\hat{Z}$ for negative examples. In each SVM training phase, we include the training points derived from positive structures from the last parsing phase, and from negative structures seen in any parsing phase. In this way, we maintain a growing list of negative structures that must receive low scores, while ensuring that each positive example has at least one high-scoring structure. This can be justified as an approximation to constraint generation, which has been used effectively in the past for similar structured optimization problems (Tsochantaridis et al., 2004). In development, we found that maintaining this negative structure memory reduced classification error rates by up to 30%.

3.2 General Applicability

The latent SVM could potentially become a powerful tool for the entire NLP community. It applies whenever a hidden structure would be useful in a classification task. In fact, a similar technique has been previously developed ad-hoc, to detect entailment in sentence pairs by building a latent matching structure between parse trees (Haghighi et al., 2005). The technique is most useful when no labeled data exists for the latent structure, or when the latent structure is not well-defined. As such, we see applications for latent SVMs in a variety of areas, including classifying multilingual word-pairs as cognates (with a latent character alignment) and biomedical relation detection (with a latent semantic parse).

3.3 An Alternative Approach

Throughout this paper, we adopt a parser-classifier approach to training our discriminative language model. That is, a single parameter vector β is learned for both tree construction and scoring. Alternatively, one could parse the input sentence with a fixed Treebank parser, and use the resulting tree to generate features for classification. This approach involves two parameter vectors, one to parse, and one to classify. Preliminary tests with this alternative indicate that it works equally well on this language modeling task. However, we focus on the parser-classifier approach here because we feel that it is academically interesting to weight a grammar to classify parser inputs. Furthermore, the parser-classifier can still be trained in languages and domains where Treebank annotations are not available.

4 Implementation

This section describes the details of our latent SVM parser-classifier, which can be used for any sentence classification task. We describe the data used to train our parser as a discriminative language model in §5.1. A latent SVM has three components:

1. An SVM training algorithm to update β
2. A feature representation $\Phi(x, z)$
3. A search to find $\max_{z \in Z(x)} \beta \cdot \Phi(x, z)$

For SVM training, we select the freely available package SVM$^{\text{light}}$ (Joachims, 1999) for its speed and scalability. The remaining two components must be designed in tandem, since the feature representation needs to decompose into the search. We adopt an unlexicalized, context-free parse tree as our latent structure z. For this structure, a natural, decomposable feature set is the set of indicators on productions. If $G = \{g_i\}$ is the set of productions in our context-free grammar, then we define $\Phi_i(x, z)$ to be the number of times the production g_i occurs in the parse tree z. Thus, $\beta \in \mathbb{R}^{|G|}$ provides a discriminatively-trained weight for each production. Our structure search is provided by a standard CKY parser. Since our features are on productions, we need to provide a fixed grammar G to the training process. We describe our options for G next.

Our primary grammar G_t is extracted from the Penn Treebank. Following Klein and Manning (2003), we perform a series of operations on the Treebank before extraction, to make the productions more useful for context-free parsing. Because Treebank accuracy is not our primary objective, we use a small, easily characterized subset of those operations; we perform only Markovization and parent-annotation. Markovization is performed head-out, after identifying production heads using a small set of rules. Next, each node in the tree is augmented with the label of its parent. To facilitate feature parsing, we perform one additional operation: chains of unary productions are replaced with a single non-terminal whose label is the concatenation of the original labels (Taskar et al., 2004). This prevents cycles in the grammar, which can cause problems when parsing with arbitrary weights. We also extract terminal productions on broad back-off classes of words, defined by character classes similar to those described in (Klein and Manning, 2003).[1]

We can create a standard probabilistic context-free grammar by assigning parameters to G_t according to the maximum likelihood probabilities of its productions, as observed in the Treebank: $\beta_i = \log P(rhs(g_i)|lhs(g_i))$. Because it is a generative model, we use this *MLE parser* as a baseline syntactic language model. We also use it as an initializer for our latent SVM training process.

We also employ a secondary grammar G_x, which uses the Treebank only to determine a part-of-speech dictionary for types and back-off classes. The remaining productions allow all possible binary structures, similar to dependency-style grammars for unsupervised parsing (Carroll and Charniak, 1992):

- $\text{PoS} \rightarrow w$
- $\text{PoS}' \rightarrow \text{PoS Y} \mid \text{Y PoS} \mid \text{PoS}' \text{ Y} \mid \text{Y PoS}'$

where PoS is a part-of-speech symbol, w is a terminal symbol and Y is any non-terminal in the grammar. This grammar has minimal input from the Treebank, and therefore may be useful for languages

[1] Our grammar G is not completely fixed; when a back-off terminal production is used in training to build a non-terminal NT over a type w not observed in the Treebank, the SVM adds a production $\text{NT} \rightarrow w$ to G, enabling a weight for the previously unseen type. In this way, we can adapt our parser-classifier to non-Treebank-annotated domains by providing positive and negative sentences for the new domain.

with more modest resources. Unlike recent approaches to unsupervised parsing (Klein and Manning, 2004; Smith and Eisner, 2005), we do not part-of-speech tag our sentences before parsing, so as to not bias the parser-classifier by layering statistically reasonable parts-of-speech over the lexical items.

5 Experiments

Our primary experiments test various baselines and parser-classifiers in the task of discriminating between human and machine-generated sentences. We describe our training set for this task, our evaluation criteria, and each system to be tested. We then describe related experiments in machine translation recognition and machine translation reranking. Finally, we examine the quality of the SVM-learned latent trees in terms of the Treebank standard.

5.1 Experimental Design

We extract our training and test data from the BLLIP Wall Street Journal Corpus, ignoring the provided parse trees, but maintaining its Treebank-style tokenization. Following previous work (Okanohara and Tsujii, 2007), we randomly split our data into four parts, with sentence counts listed in parentheses:

- Negative Generating Data (450k)
- Positive Train (50k)
- Positive Development and Test (3k each)

We use the negative generating data to train a KN-smoothed (Kneser and Ney, 1995) trigram language model, from which we sample our negative examples using a straight-forward algorithm (Okanohara and Tsujii, 2007). We sample 50k negative training examples, and 3k for both development and test. The complete train, development and test sizes are 100k, 6k and 6k respectively. Following (Okanohara and Tsujii, 2007), we constrain all sentences to be more than 4 words in length.

Our primary evaluation metric is classification accuracy, though we also report 11-point average precision (**AvgPrec**), the average of precision measured at recall levels $0, 10, \ldots, 100$, which measures the system's ability to rank positives above negatives.

We test six systems, three baselines and three latent SVM parsers. The parsers differ in the amount of Treebank knowledge used in their grammars and initial parameters:

- **MLE**: Sentences are scored with the log probability of the tree found by our MLE Treebank parser. For classification, we empirically find the probability threshold with the highest accuracy on the development set.
- **Trigram-Linear**: A linear SVM that uses a bag of all 1, 2 and 3-grams as features. Feature vectors are L2-normalized.[2]
- **Trigram-Poly**: As Trigram-Linear, but trained with a third-order polynomial kernel. This corresponds roughly to the state-of-the-art in this task (Okanohara and Tsujii, 2007).
- **LSVM-MLE**: A latent SVM parser, using our G_t grammar, with initial parameters provided by MLE production probabilities.
- **LSVM-G_t**: Like LSVM-MLE, but all parameters are initially set to 0. Tree selection in the first round of training is left to our tie-breaker, which favors right-branching trees.
- **LSVM-G_x**: A latent SVM parser using the G_x grammar. Parameters are initially set to 0.

The soft-margin trade-off parameter λ is optimized through grid search on a development set. For the latent SVM, the λ parameter is tuned according to the accuracy of the SVM during the first iteration. The same development set is used to select a stopping point for iterative latent SVM training.

5.2 Classification Accuracy

We report test set performance of all six of our systems in the discriminative language modeling classification task. Results are shown in the first two columns of Table 1. The first thing to note is that our Trigram-Poly scores are in line with the scores reported by Okanohara and Tsujii (2007) with a similar amount of training data, indicating that it is a reasonable stand-in for their approach. All three LSVM parsers exceed the Trigram-Poly system, with LSVM-MLE producing a relative reduction in error rate of 37%.

[2] During development, vector normalization improved results for the trigram SVM by up to 5 percentage points, and was crucial to learning a polynomial SVM in a reasonable amount of time (less than 12 hours).

Table 1: Classification accuracy

Method	Sample AvgPrec	Sample Acc	SMT Pairwise
MLE	50.72	55.75	49.30
Trigram-Linear	56.15	58.72	49.90
Trigram-Poly	81.97	70.50	55.70
LSVM-G_x	76.79	73.47	63.15
LSVM-G_t	**87.53**	79.78	71.65
LSVM-MLE	87.22	**81.42**	**71.85**

The MLE entry indicates that parser probabilities alone are not particularly useful for this task, with accuracy only slightly above chance. We also tested scores from the Stanford Parser (Klein and Manning, 2003), and received similar results. Using parser probability and sentence length as the only features of an SVM classifier produced a development accuracy of 65%, significantly better than the parser alone, but still well below the baseline set by Trigram-Poly. Though MLE parser scores do not beat the baseline, the Treebank appears to be a useful resource for LSVM training; each addition of Treebank information to our grammar or initializer increases accuracy. It also shortens training time: LSVM-MLE reaches its peak in 4 iterations, while LSVM-G_t does so in 16 and LSVM-G_x in 25. The gap between LSVM-MLE and LSVM-G_t demonstrates that initial parameter choices do matter, as the LSVM objective is not convex. Yet the gap is small, indicating that this application is not overly sensitive to initial conditions.

5.3 Machine Translation Recognition

Previous reports of Treebank parser scores preferring SMT output over human references were a major motivation for this work (Och et al., 2004). Thus, we construct an alternate test-set from MT output to see if our parser-classifiers reverse this trend. We assemble a new test set using 2k German-English sentence pairs extracted from the Europarl corpus, corresponding to the devtest set from the 2006 WMT Translation Shared Task (Koehn and Monz, 2006). We translate the German sentences using a competitive phrase-based translation system (Koehn et al., 2003) trained with the shared task resources. Our test set consists of 2k English reference translations (positive) and 2k system translations (negative). We report pairwise accuracy: the classifier is correct only if, for a given German input, the reference receives a higher score than the system output.

Results are shown the final column of Table 1. The MLE parser indeed shows a slight preference for MT output; we confirmed this trend with the Stanford parser, which received a more dramatic 42.40. The remaining systems prove more effective, and their relative performance mirrors our previous results; in fact, the gap between the LSVM methods and Trigram-Poly has grown. This indicates that the syntactically-driven classifiers are adapting better to the change in domain and task.

5.4 Machine Translation Reranking

We have also conducted preliminary experiments on using our discriminative language models in SMT reranking. We use our phrasal decoder to produce unique 100-best lists for both the WMT 2006 devtest and test sets. Both sets consist of 2k sentences. Minimum error-rate training (Och, 2003) is conducted on devtest to learn feature weights after augmenting the 100-best list with a new language model. Results are reported after applying the learned weights on the held-out test set.

We have focused this paper on training with negative examples sampled from n-gram language models, with the hope that they are generally useful for language modeling. However, we can also use application-specific negative examples when they are available. We create a new 100k-sentence LSVM training set using 50k reference translations and 50k corresponding SMT outputs, which are created by a version of our SMT engine that has not seen our 50k references or their German source sentences during training. We train SMT versions of LSVM-MLE and Trigram-Linear on this data.

Results are shown in Table 2. As expected, the inclusion of the MLE parse scores has no impact on performance. Meanwhile, LSVM-MLE does show a very slight improvement. The task-specific models show further improvement, with the trigram model performing surprisingly well. We suspect that this is for two reasons: first, the language model is not so influential in phrasal SMT so as to affect the usefulness of n-gram features; and second, we suspect that the nature of phrasal decoding makes the sorts

Table 2: SMT Reranking results

Language Model	BLEU
Baseline	26.21
+ MLE	26.19
+ LSVM-MLE	26.30
+ SMT Trigram	26.45
+ SMT LSVM-MLE	26.38
+ SMT Trigram + SMT LSVM-MLE	26.52

Table 3: Treebank Unlabeled F-measure

Method	≤ 10	No limit
Right-branching	53.93	32.87
LSVM-G_x	50.71	38.62
LSVM-G_t	80.13	66.19
LSVM-MLE	83.15	71.53
MLE	95.46	83.60

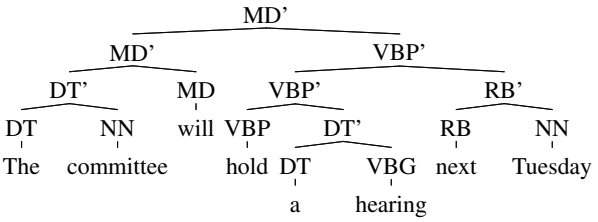
of local improvements that would be detected by a trigram language model more likely to appear in the 100-best list than those that would be detected by a syntactic language model.

5.5 Treebank Accuracy

To determine how much latent SVM training alters our Treebank-informed grammar G_t, and how much our uninformed LSVM-G_x parser conforms to linguistic intuitions, we parse a small held-out set consisting of the first 393 sentences of Section 22 of the Treebank, and report unlabeled bracketing F-measure using the *evalb* utility under its standard configuration. We also parse a set derived from all Treebank sentences with 10 or fewer words (ignoring punctuation), to more closely match the test set commonly used in unsupervised parsing (Klein and Manning, 2004; Smith and Eisner, 2005). Note that all three systems using the G_t grammar have benefited from seeing parts of the annotated ≤ 10 test set during grammar construction and initialization.

The resulting F-measures are shown in Table 3. In reconfiguring its parameters for classification, the LSVM-MLE system has diverged a fair amount from its MLE initializer. On the other hand, without the benefit of Treebank probabilities, the LSVM-G_t system begins with an F-measure of 38.14 (54.88 with ≤ 10), and LSVM training increases its F-measure by more than 25 absolute points on both sets. Of course, this is not truly unsupervised parsing, as LSVM-G_t employs productions derived from the Treebank, but it is still remarkable to see how well it recovers from beginning with uniform weights. LSVM-G_x scores substantially lower than the two Treebank-informed systems, and is unable to beat the right-branching baseline on the ≤ 10 set. On the unlimited set, it does climb from a right-branching baseline of 32.87 to a score of 38.62, deriving structure that is slightly in line with the Treebank while pursuing its classification objective. Its precision suffers because it produces only binary trees, while the G_t systems can produce n-array trees due to their Markovized grammar.

Figure 1: Example LSVM-G_x parse



Examining the trees

Our Treebank-informed systems, LSVM-MLE and LSVM-G_t, remain somewhat close to the Treebank standard. Although the grammar weights have been modified substantially, the resulting trees are not so different, as is demonstrated by their relatively high F-measures. Some preterminals have been repurposed during classifier training: JJ is used more often, for tokens that were previously tagged as nouns or determiners, while FW seems to have been held in reserve for unseen adjective-like words. It is difficult to characterize the changes to internal structure, but it is safe to say that trees for negative sentences deviate more drastically from the MLE baseline than those for positive sentences.

With minimal input from the Treebank, LSVM-G_x is substantially more interesting. As is often the case in unsupervised parsing, the learned latent structure may be different from gold-standard human annotations in systematic ways. For example, Figure 1 demonstrates a noun phrase headed by a

determiner, a phenomenon we observe frequently in the resulting parse trees. Other adjuncts select unusual heads: appositions and other adjunct modifiers are headed by the initial comma. The grammar has learned that many adjunct phrases, such as adverbs and appositions, are delimited by commas, and allows them to attach promiscuously to noun and verb phrases. Many prepositions are tagged as adverbs, and the resulting prepositional phrase becomes an adverbial phrase, a reasonable transformation since many play adverbial roles in the sentence.

Other systematic differences seem to arise from the classification objective. In the case of modal and auxiliary verb constructions, we often find verb phrases divided in half, as in Figure 1. The early part of the verb sequence absorbs its pre-modifying subject, perhaps attempting to capture subject-verb agreement, important for judging grammaticality. The later half of the verb sequence absorbs the remainder of the verbal arguments, only at the top level of the derivation are the two halves combined.

6 Conclusion

We have shown that latent SVMs can train a parser to classify sentences. By training to discriminate between human and machine-generated sentences, we have created a parser-classifier that behaves as a discriminative, syntactic language model. Even when trained without the benefit of a Treebank grammar, our parser-classifier outperforms the existing state-of-the-art in distinguishing real sentences from pseudo-negatives, and human reference translations from SMT output. We have shown promising initial results in reranking SMT outputs. In the future, we hope to incorporate our language model directly into a syntactic translation system.

References

Glenn Carroll and Eugene Charniak. 1992. Two experiments on learning probabilistic dependency grammar from corpora. In *Working Notes of the Workshop on Statistically-based NLP Techniques*, pages 1–13.

Eugene Charniak, Kevin Knight, and Kenji Yamada. 2003. Syntax-based language models for machine translation. In *MT Summit IX*.

Eugene Charniak. 2001. Immediate-head parsing for language models. In *ACL*.

Ciprian Chelba and Frederick Jelinek. 1998. Exploiting syntactic structure for language modeling. In *COLING-ACL*, pages 225–231.

Stanley F. Chen. 1995. Bayesian grammar induction for language modeling. In *ACL*, pages 228–235.

Michael Collins, Brian Roark, and Murat Saraclar. 2005. Discriminative syntactic language modeling for speech recognition. In *ACL*, pages 507–514, Ann Arbor, USA, June.

Pedro Felzenszwalb, David McAllester, and Deva Ramanan. 2008. A disrcriminatively trained, multiscale, deformable part model. In *Computer Vision and Pattern Recognition*, Anchorage, USA, June.

Michel Galley, Mark Hopkins, Kevin Knight, and Daniel Marcu. 2004. What's in a translation rule? In *HLT-NAACL*, pages 273–280, Boston, USA, May.

Jianfeng Gao, Hisami Suzuki, and Bin Yu. 2006. Approximation Lasso methods for language modeling. In *COLING-ACL*.

Aria Haghighi, Andrew Y. Ng, and Christopher Manning. 2005. Robust textual inference via graph matching. In *EMNLP*.

Thorsten Joachims. 1999. Making large-scale SVM learning practical. In B. Schölkopf, C. Burges, and A. Smola, editors, *Advances in Kernel Methods - Support Vector Learning*. MIT Press.

Dan Klein and Chris Manning. 2003. Accurate unlexicalized parsing. In *ACL*.

Dan Klein and Chris Manning. 2004. Corpus-based induction of syntactic structure: Models of dependency and constituency. In *ACL*.

Reinhard Kneser and Hermann Ney. 1995. Improved backing-off for m-gram language modeling. In *International Conference on Acoustics, Speech, and Signal Processing (ICASSP-95)*, pages 181–184.

Philipp Koehn and Christof Monz. 2006. Manual and automatic evaluation of machine translation. In *HLT-NACCL Workshop on Statistical Machine Translation*, pages 102–121.

Philipp Koehn, Franz J. Och, and Daniel Marcu. 2003. Statistical phrase-based translation. In *HLT-NAACL*.

Alex Kulesza and Stuart M. Shieber. 2004. A learning approach to improving sentence-level MT evaluation. In *Proceedings of the 10th International Conference on Theoretical and Methodological Issues in Machine Translation*, Baltimore, MD, October 4–6.

F. J. Och, D. Gildea, S. Khudanpur, A. Sarkar, K. Yamada, A. Fraser, S. Kumar, L. Shen, D. Smith, K. Eng, V. Jain, Z. Jin, and D. Radev. 2004. A smorgasbord of features for statistical machine translation. In *HLT-NAACL*, pages 161–168.

Franz J. Och. 2003. Minimum error rate training for statistical machine translation. In *ACL*, pages 160–167.

Daisuke Okanohara and Jun'ichi Tsujii. 2007. A discriminative language model with pseudo-negative samples. In *ACL*, pages 73–80.

Brian Roark. 2001. Probabilistic top-down parsing and language modeling. *Computational Linguistics*, 27(2):249–276.

Noah A. Smith and Jason Eisner. 2005. Guiding unsupervised grammar induction using contrastive estimation. In *IJCAI Workshop on Grammatical Inference Applications*, pages 73–82.

Ben Taskar, Dan Klein, Mike Collins, Daphne Koller, and Christopher Manning. 2004. Max-margin parsing. In *EMNLP*, pages 1–8, Barcelona, Spain, July.

Ioannis Tsochantaridis, Thomas Hofman, Thorsten Joachims, and Yasemin Altun. 2004. Support vector machine learning for interdependent and structured output spaces. In *ICML*, pages 823–830.