# Discriminative reranking of top-K parses of a sentence

**Harsh Jhamtani**
Language Technology Institue
Carnegie Mellon University
jharsh@cs.cmu.edu

## Abstract

This report discusses discriminative reranking of top-K parses of a sentence. Various features based on syntactic and lexical properties of trees are analyzed. Discriminative models are trained using algorithms like SVM, perceptron, and Maxent. Best model selected based on validation data set (F1 score of $\mathbf{87.10}\%$) achieves F1 score of $\mathbf{86.36}\%$ on test. For the set of features and model configurations explored, best score using **Averaged weights Perceptron** (following a certain strategy for averaging) outperforms best F1 scores from SVM and Maxent learning algorithms. Report will conclude with discussion on the relative performance on using the different algorithms, and possible explanation for the observed behavior.

## 1 Introduction

Algorithms to produce parse tree of a sentence, like CKY algorithm, can be easily extended to provide top-K possible parses. Thereafter, discriminative models can be trained using algorithms like SVM, perceptron, and Maxent, to predict the best parse out of the K-best parses. Due to the nature of these algorithms, it is easier to consider a large number of features, which are difficult to consider in the parsing algorithm. In this report, I discuss a Java implementation of discriminative reranking of parse trees with model training based on SVM, perceptron, and Maxent algorithms.

## 2 Algorithms

Following algorithms are considered:

### 2.1 SVM

Support Vector Machines aim to learn weights (Cortes and Vapnik, 1995) such that the desired class of data points are separated from the undesired class data points by a maximum possible margin. Since data sets are often inseparable, the formualtion allows for misclassifications at a penalty.

### 2.2 Maximum entropy

We would want to maximize the log likelihood given as follows:

$$\log(P) = \sum_x \frac{exp(w^T x_c)}{\sum_i exp(w^T x_i)}$$

, where $w$ represents the weight vector. Training is done using LBFGS (Andrew and Gao, 2007) batch updates with regularization on weights.

### 2.3 Perceptron

Perceptron (Freund and Schapire, 1999) is a simple artificial neural network which learns a linear separator. In case of data not being linearly separable, perceptron will not converge. Using number of epochs as a hyperparamter, accuracy on validation set can be used to obtain favorable model. Moreover, it has been argued by prior works that averaged perceptron might perform better. So average perceptron model is also tried.

**Note on using 'proxy' gold tree :** Note that in the given data set, the gold tree ($x_c$) may not appear in the k-best list always. In fact, my analysis suggests that about $30\%$ times it is not present in k-best list. For such cases, formulation of Maxent model becomes ill-defined. A workaround for this to use $oracle_{x_c}$, which would be the tree in the k-best list with highest F1-score as compared to the gold tree. Then we can *throw away* the gold tree, and consider $oracle_{x_c}$ as gold for training purposes.

## 3 Features

In this section I describe various features. There are prior works like (Charniak and Johnson, 2005), (Johnson and Ural, 2010), and (Hall et al., 2014) which discuss a variety of features relevant for the task.

### 3.1 Position in k-best list

The k-best list is sorted as per parser provided score. So intuitively, higher ranked parses are the ones which have more parser confidence. Simple indicator features representing position in kbest list are used. During learning, it may happen that gold tree is not present in k-best list, and in that case, none of the features is fired.

### 3.2 Parser score

Parse score provided by the parsing algorithm are intuitively supposed to be informative. Both raw scores and binned values of scores are tried.

### 3.3 Right branching

One feature is the number of nodes from root upto right most non-punctuation terminal. Another feature is the number of nodes not in the rightmost branch of the tree. Yet another type of feature is indicator of the non-terminals present in the rightmost branch of the tree.

### 3.4 Span features

All the features described here are considerd in conjunction with the rule in question.

#### 3.4.1 Span length features

Binned length of the span. The intuition is that these features will capture the spreads with which different constituents occur in gold trees as compared to lesser accurate trees.

#### 3.4.2 Span lexical features

- Word at the beginning of the span
- Word at the end of the span
- Word just before beginning of the span
- Word just after the end of the span

#### 3.4.3 Span POS features

These indicator features are to characterize the span byt the POS of the words present on its edges.

- POS(Part-of-speech) tag at the beginning of the span
- POS tag at the end of the span

#### 3.4.4 Span split point feature

Marking the split position of the binary rules. These features can help disambiguating prepositional phrase attachment. Some words have more chance to be followed by specific prepositions. Specifically, the feature is binary rule coupled with the the last word under the first child.

#### 3.4.5 Span shape feature

These features capture capitilization or presence of punctuations in words of the every span. These features, for instance, can help in detecting noun phrases, as many proper nouns begine with capital alphabet.

### 3.5 Ngram Siblings

For a rule $R > R_1R_2R_3...R_m$, features are generated as per following criteria: An indicator function is added for each pattern as follows: $R > R_{i-1}R_iR_{i+1}$

### 3.6 Head features

For a rule $R > R_1R_2R_3...R_m$, all the $R_i$'s which are prepositional phrases are contextualized with their head words. The motivation being to tackle problems with prepositional phrase attachment.

## 4 Experiments and Results

### 4.1 Experiments with SVM

Primal SVM mode is used to solve the optmiization problem. Various model parameter configurations are tried:

- Penalty: $\{0.001, 0.01, 0.1\}$
- Step size $\{0.001, 0.01, 0.1\}$
- epochs: $\{20, 30, 40\}$

On testing with combinations of above configurations, best configuration was observed to be as follows: Penalty=0.01, Step size =0.1, and number of epochs to be 30.

Choice of loss function: Experiments are done using following two loss functions:
$-> 1 - F1$
$-> 0/1$ loss: Less inuitive than using above formulation, but a lot faster

#### 4.1.1 Impact of various features

Table 1 shows the incremental benefits from adding various types of features on using SVM model.

- Just having binned scores provides F1 score less than that of baseline.
- Adding all the rules as features provides an increas of 2.5 F1 points with just about 50K features. F1 score on validation goes above baseline.
- Adding span length, lexical and context features pushes F1 score to $85.46$
- Adding length of path from root to rightmost non punctuation terminal increases the F1 score by 0.66 F1 points.
- Adding position feature and head features descreases F1 score, and are therefore not reported in the table.

| Feature set | F1 | Features size |
|---|---|---|
| 1-best baseline | 84.12 % | - |
| + Score | 81.65 | 10 |
| + Rules | 84.17 | 49.71 K |
| + Span length | 84.79 | 121.30 K |
| + Span lexical | 85.36 | 1.05 M |
| + Span POS | 85.46 | 1.20 M |
| + Rightmost branch | 86.12 | 1.73 M |
| + Span split and shape | 86.17 | 2.40 M |
| + Ngram siblings | 86.20 | 2.43 M |

Table 1: F1 scores on addition of various feature sets [using SVM] ($K$ represents 1000 and $M$ represents million)

#### 4.1.2 Impact of loss function

Above results are for $0/1loss$. Using $1 - F1$ with the final configuration as above, F1 score on validation becomes $\mathbf{86.66}$

#### 4.1.3 Impact of using 'proxy' gold tree

On usnig proxy tree in place of gold tree, the F1 score reduces to 84.65.

#### 4.1.4 Discussion on results obtained using SVM model

The best F1 score on validation set using above set of features is $\mathbf{86.66}$. This model achieves F1 score of $86.08$ on test set. Using $1 - F_1$ as loss function is preferrable over $0/1$ loss. Using proxy gold tree does not help in case of SVMs.

### 4.2 Experiments with Maxent

Best F1 score on validation data set using Maxent model is $85.65\%$. Best F1 score without using oracle i.e. instead using gold tree provided, drops to below baseline of $84.12\%$. This shows that it is important to use a proxy gold tree in Maxent models.

### 4.3 Experiments with Perceptron

Perceptron algorithm is trained using all the features described in Table 1. Vanilla perceptron achieves F1 score of only 85.32

#### 4.3.1 Using 'proxy' gold tree

Instead of using provided gold trees, I switch to 'proxy' gold tree as per the methods described earlier. Figure 1 shows F1 measure on validation data when using perceptron algorithm as the number of epochs increases. Perceptron algorithm is able to achieve F1 score of $\mathbf{87.0}$ using number of epochs as 30. Thus, perceptron benefits from using 'proxy' gold tree. Section 5.2 discusses possible reasons for this behavior.

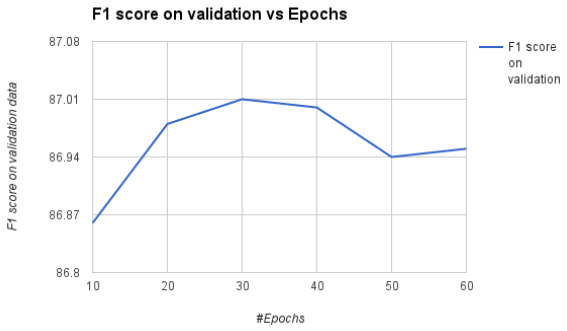


Figure 1: F1 score on validation data with respect to number of epochs for Perceptron learning

#### 4.3.2 Averaged perceptron

Since a perceptron does not converge in case of linearly inseparable data, after a decent number of epochs, weight vectors just thrashes around. So intuitively it makes sense to use a average of such

vectors to 'smooth' out weight vector estimate. The strategy used is as follows:

- From previous experiments it can be observed that just after 10 epochs, perceptron starts achieving F1 score close to $87\%$. So a weight vector $W'$ is maintained, and is updated once $M$ epochs have been processed. For experiments I tried with $M$ as 10 and 20.

- $W'^{(t+1)}$ is updated as $W'^{(t)} + W^{(t)}$ after processing every K data points. I trid with $K = 73$, $K = 7919$ $K = 10000$. The rationale behind using such large values of $K$ is only to save on processing time. The algorithm will probably benefit from smaller $K$, but it becomes very slow. Moreover, 7919 and 73 values may seem arbitrary - the reason for selection of these values is that they are prime numbers.

- Prediction for unseen data is made using $W'^{(T)}$, which is the final weight vector obtained.

- For above setting, total number of epochs used are $\{30, 40\}$

F1 score on validation data set reaches $\mathbf{87.1}$ for $M = 30$ and $K = 7919$

### 4.4 Feature importance

The results described in this section are with reference toaveraged perceptron algorithm using configuration as described in Section 4.3.2.

Feature importance cannot be inferred directly from the weights associated with the features. One approximate method is to one by one remocve features from the final model and see the decrease in the F1 score. However, given the large number of features, only practical way is to remove groups of related feature sets to observe above said values. Table 2 shows such values for the various feature sets. It can be observed that removing 'Span lexical' features leads to a decrease of $0.68$ F1 points - this means lexical features are important predictors. At the same time, it should be noted that 'span lexical' contributes a third of all features ( 800K).

| **Feature set removed** | **($\Delta$ F1)** |
|---|---|
| Score | -0.07 |
| Rules | -0.05 |
| Span length | -0.09 |
| Span lexical | **-0.68** |
| Span POS | -0.10 |
| Rightmost branch | -0.07 |
| Span split and shape | -0.13 |
| Ngram siblings | -0.15 |

Table 2: Change in F1 score on remove specific feature sets

## 5 Discussions and Conclusion

### 5.1 Discussion on relative performance of algorithms

Best performances observed for the three algorithms can be summarized as below:

- SVM: Best F1 score on validation data: $86.66\%$.

- Perceptron: Best F1 score on validation data: $\mathbf{87.10}\%$.

- Maxent: Best F1 score on validation data: $85.65\%$.

The three algorithms tried are SVM, Perceptron and Maxent. For the same set of features, and search over previously described configuration space, avergaed Perceptron surprisingly outperforms both Maxent and SVM. SVMs are powerful models and have been shown to work better than simpler algorithms like Perceptron in many classification tasks. However, for the feature set explored in this report, Perceptron works better. An important takeway is that using simpler models can often provide better results, and such models shouldn't be outrightly rejected just because of their simplicity. Moreover, it is tough to say if the same ranking among algorithms will hold if more features are added - it can only be verified experimentally.

### 5.2 Impact of using 'proxy' gold tree

From the experiments conducted, it was observed that using 'proxy' gold tree (oracle) provides benefit in case Maxent and Perceptron algorithms. In case of Maxent it is necessary to use this method, or resort to some other method, to keep the formulation well-defined. However, it was not neces-

sary to use such a mechanism in case of Perceptron. Therefore, such behavior may not be very intuitive. Moreover, using such mechanism led to decrease in accuracy for SVM.

An argument in favor of proxy or 'oracle' is that gold trees may contain certain features even though the parser, which feeds k-best list, may be unable to produce such trees in top-k lists. So in a way classifier 'learns' to give importance to those features which it wouldn't see during testing time anyway. Also note that our aim is to get as close as possible to gold tree, measure by F1 score. However, we penalize the best tree in k-best if gold tree is not present in the k-best. In other cases when gold is present in k-best, we favor the best tree in k-best list (which is gold tree). This has the potential to 'confuse' the classifier.

### 5.3 Feature set and configuration of the best performing model

The best model(*Awesome Reranker*) is an averged Perceptron with number of epochs as 30 and weights are sampled after visiting every 7919 data points. Features are as follows:
Score + Rules + Span length + Span lexical + Span POS + Rightmost branch + Span split and shape + Ngram siblings
F1 on validation: **87.10**
Training time: **99** seconds

The best SVM model(*Basic Reranker in code*) is SVM model with number of epochs as 30 and regularization factor of 0.01, step size of 0.02, and batch size of 10. It uses following features:
Score + Rules + Span length + Span lexical + Span POS + Rightmost branch + Span split and shape + Ngram siblings
F1 on validation: **86.66**
Training time: **498** seconds

### 5.4 Conclusion

Discriminative rerankers were trained using Maxent, SVM, and Perceptron algorithms. The best model (using avergaed perceptron with 'proxy' gold tree) achieves F1 score of **86.36** on test data set. Moreover, perceptron was about 5 times faster to train. A large number of features were explored for building the models. It was observed that removing span lexical features leads to largest decrease in accuracy, hinting at the importance of these features. Using 'proxy' gold tree and averaging of weights benefits models based on Perceptron learning algorithm.

## References


Galen Andrew and Jianfeng Gao. 2007. Scalable training of l 1-regularized log-linear models. In *Proceedings of the 24th international conference on Machine learning*, pages 33–40. ACM.

Eugene Charniak and Mark Johnson. 2005. Coarse-to-fine n-best parsing and maxent discriminative reranking. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, pages 173–180. Association for Computational Linguistics.

Corinna Cortes and Vladimir Vapnik. 1995. Support-vector networks. *Machine learning*, 20(3):273–297.

Yoav Freund and Robert E Schapire. 1999. Large margin classification using the perceptron algorithm. *Machine learning*, 37(3):277–296.

David Leo Wright Hall, Greg Durrett, and Dan Klein. 2014. Less grammar, more features. In *ACL (1)*, pages 228–237.

Mark Johnson and Ahmet Engin Ural. 2010. Reranking the berkeley and brown parsers. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 665–668. Association for Computational Linguistics.