Discriminative Reranking

Samridhi Shree Choudhary
sschoudh@cs.cmu.edu

Abstract

This document presents an implementation of a parsing reranker trained with two learning algorithms: Primal SVM and Perceptron. The base parser produces a set of candidate parses for each sentence, with associated log probabilities that define the initial ranking of these parses. A second model attempts to improve on this initial ranking, using additional features extracted from the trees in the training set. The *features extracted* from the training trees is explained along with the *scores(F1) achieved* for each learning model. An analysis of the features and an overview of which features work together is also presented.

1 Introduction

Identifying the syntactic structure in natural language sentences is the first step towards achieving a good semantic interpretation. Machine learning approaches to natural language parsing have shown some success in this direction. This document presents an approach which learns a meaningful parse of a sentence by reranking the output of an existing base probabilistic parser. This approach allows a tree to be represented as a vector of features which is then passed to a reranking model to learn the weights over all the features extracted from the training corpus. The motivation behind extracting the features is to add more context to the symbols of the grammar to capture better semantic behavior. One example where such context might be helpful is that *'to'* PPs usually take verbs as their object whereas a *'of'* PPs usually take nouns. Context free grammars can attach PP equally well to each other.
The base parser used in this assignment is a simplified version of the Berkeley parser. Each input (k-best list of parsing trees and the gold tree) is converted into a feature vector using a feature extractor. The features used for the reranking purposes and the intuition behind them is detailed. This is followed by results and analysis.

2 Learning Algorithms

This assignment uses two learning algorithms to learn the weights of the features extracted from the training trees : ***Primal SVM sub-gradient descent*** *(uses Adagrad)* and ***Perceptron***. The objectives and minimization rule for each of them is deatiled below.

2.1 Primal SVM sub-gradient descent

The primal SVM implementation in this assignment uses Adagrad for its minimization objective. Its takes the step size, the regularization constant and the batch size as the inputs. It also takes a customized definition of a loss-augmented model which is its basic unit of operation. The minimization objective is simplified by reducing the number of constraints to be satisfied as follows:
Find the most violated constraint

$$\hat{y} = arg \max_{y \epsilon Y(x)} (w^T f_i(y) + l_i(y))$$

Then use this constraint while calculating the gradients, which is given by:

$$\bigtriangledown_w \leftarrow w + C \sum_i (f_i(\hat{y}) - f_i(y^*))$$

Where $f_i(y^*)$ is the feature vector of the gold tree. The hyper parameters used in the final implementation were as follows:

- Number of iterations/epochs = 30
- Batch-size = 10
- Step size = 0.001
- Regularization constant = 0.0001

- Loss function used ($l_i(y)$) = 1-F1 (higher the F1 lower the loss)

2.2 Perceptron

A basic implementation of perceptron was used as the second learning model in this assignment. The objective function and the update rule for the perceptron is as follows:
Try to classify:

$$\hat{y} = arg \max_{y \epsilon Y(x)} w^T f_i(y)$$

If it is incorrect (not equal to the gold tree), update the weights:

$$w \leftarrow w + (f_i(y^*) - f_i(\hat{y}))$$

Where $f_i(y^*)$ is the feature vector of the gold tree and $f_i(\hat{y})$ is the feature vector of the misclassified example. The perceptron implementation in this assignment returned an average of the weights taken at different points of the iteration instead of returning the final weight vector. The defining parameters were:

- Number of iterations/epochs = 30
- Batch-size/number of training samples after which the average was taken for the current weight and the previous average weight vector = 10000

3 Features

A number of features are added to improve the performance of the reranker over the baseline grammar. The features are described in the sections below.

3.1 Position in the k-best list - Class 1

This rule specifies the position of the parse tree in the k-best list returned by the base parser. For the gold tree, we find if it is present in the k-best list and use that position for retrieving the feature. However, if the tree is not present in the k-best list, we do not add that feature for the gold-tree. The features take the form: ***Posn = i***
Where *i* is the position of the tree in the list.

3.2 Grammar Rules - Class 2

This feature represents the grammar rules used in each of the k-parse trees. The rule takes the form: $Rule = [rule_i]$

3.3 Span Length - Class 3

This features captures the length of the span for each subtree in the parse trees. Since there might be many unique values for the lengths, the values are binned into the buckets: $1, 2, 3, 4, 5, 10, 20, >= 21$. The features take the form: $Rule + spanLength \rightarrow [BucketNum]$

3.4 First and last word/POS tags for a span - Class 4

This feature represents the first and the last word (as well as their POS tags) of the span. Intuitively, these set of features capture the lexical properties of the heads to some extent without performing the expensive lexical annotations. They take the following form:

- $subtreeRule \rightarrow firstword = word_{first}$
- $subtreeRule \rightarrow lastword = word_{last}$
- $subtreeRule \rightarrow posfirst = postag_{first}$
- $subtreeRule \rightarrow poslast = postag_{last}$

3.5 Right branching length - Class 5

This feature captures the number of non-terminal nodes that lie on the path from root node to the right-most *non-punctuation* word in the given parse tree. This makes the reranker prefer right-branching trees. The features take the form: $rbLength =< rightLength >$.

3.6 Head NNGram - Class 6

This feature attaches the headlabel of a subtree with the trigrams and bigrams of its child labels. The features in this category take the form:
$label_{parent} \rightarrow [label_{child1}][label_{child2}][label_{child3}]$
$label_{parent} \rightarrow [label_{child1}][label_{child2}]$
for all trigram and bigram sequences of children labels.

3.7 Previous and next context for a span - Class 7

This features adds context to the headwords of the spans. As the name suggests the previous context identifies the the word that occurs before the first word of the span and the next context does the same for the word that occurs after the last word of the span. The features in this set take the following form:

- $subtreeRule \rightarrow firstcontext = word_{prev}$
- $subtreeRule \rightarrow lastcontext = word_{next}$

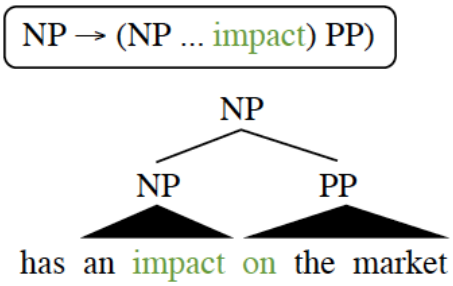

Figure 1: A split point feature example showcasing disambiguation of PP attachment. The feature captures that the word 'impact' is likely to take a PP.

3.8 Split Point for Binary Rules - Class 8

This feature captures the words at and around the split point of a binary rule in the parse tree. This features helps resolve PP attachment ambiguity or the right attachments to noun phrase. An example for this is shown in Figure 1.

3.9 Span Shape for all the parse subtrees - Class 9

This span feature captures the overall shape of the span of a subtree in the given parse tree. For each word in the span, we write their representations as follows:

- If the word starts with a capital letter: represent it as **"X"**
- If the word starts with a small letter: represent it as **"x"**
- If the word is a digit: represent it as **"0"**
- If the word is a punctuation: copy it as it is.

The rules are prefixed with the subtree head label. The features in this schema take the form: $NP \rightarrow X, x"x"0$

3.10 Syntactic and Lexical heads - Class 10

This feature captures the surface head annotations and attaches a syntactic or a lexical head to the root of a subtree in the parse tree. However, no improvement in the F1 score was seen for this feature along with the features detailed above and hence the feature was removed from the final implementation.

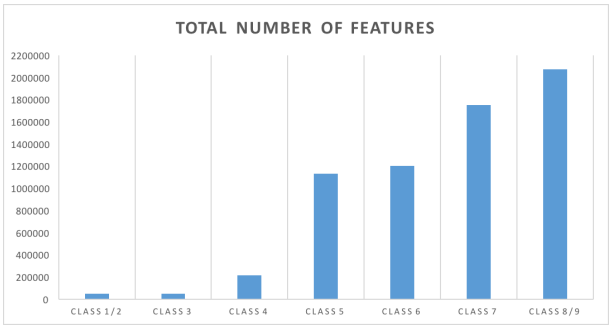

Figure 2: The graph shows the total number of features for the reranker to learn as each class of features (described previously) is added

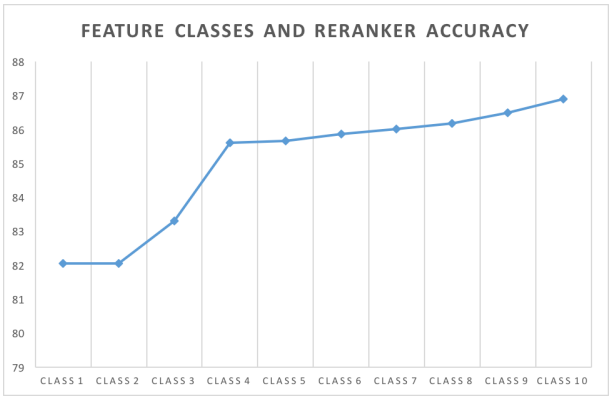

Figure 3: The graph shows the final reranker F1 score after features belonging to a particular feature class are added to the extractor.

4 Results

The re-ranker trained with SVM performed better than the baseline parser achieving an F1 score of **87.09**. The total number of features after adding the above feature classes was around **2 million**. Figure 2 shows the increase in the total number of features as each feature class was added to the feature extractor of the system. The classes have been marked in the 'Features' section above. The numbers for the graphs shown were performed for 30 iterations for SVM. However, the final F1 score of **87.09** was achieved for 40 iterations with some increase in the training time. The results shown are for the validation set.

Each feature class added had its own contribution towards improving the final F1 score of the reranker. Figure 3 shows the change in F1 score with each feature class added.

As the number of features increased, the

training time also increased. The following graph shows the total training time in seconds for each class of features added (30 epochs for SVM):

The final accuracy results for both SVM and perceptron is displayed in the tables below:

Learning Model	**Total Features**	**P**	**R**	**F1**	**EX**
Primal SVM	2076476	87.34	86.84	**87.09**	32.31
Perceptron	2076476	84.7	84.75	84.73	18.25
Baseline (on both)	2076476	84.6	83.65	84.12	22.94

Table 1: Accuracy values of the reranker for SVM and Perceptron on the Validation Set

Learning Model	**Total Features**	**P**	**R**	**F1**	**EX**
Primal SVM	2076476	87.03	86.24	**86.63**	29.71
Perceptron	2076476	84.11	83.86	83.99	16.65
Baseline (on both)	2076476	84.13	83.19	83.66	22.31

Table 2: Accuracy values of the reranker for SVM and Perceptron on the Test Set

SVM was able to converge better on the training set provided than Perceptron and hence achieved better F1 score. Feature pruning was also tried, where any feature which appeared less than a certain number of times (5 was used as a threshold) was discarded from the final training set. Though this decreased the total number of features by almost half there was no improvement seen in the F1 score for the reranker.

5 Conclusion

This assignment considered 2 alternative learning models for reranking the initial output of a baseline parser using a large set of features with good improvement in the accuracy. These show that reranking methods are generally helpful in improving the parse outputs of a baseline parser without going through the process of making the underlying grammar more complex. The features added noisily capture the lexical features of the grammar and hence select a better parse of a sentence. The accuracy can be improved by using better models to learn the weights of the features extracted.

We also noticed that certain combinations of features provided a good increase in accuracy while others did not. For example, we added both the position feature (position of the tree in the k-best list) and the binned log-score feature (10 buckets for the log-scores of each parse). Both these features essentially capture the same information, however when used together it reduces the F1 score. When used exclusively (one or the other) it provided the same increase in F1. Similarly, after adding the features related to span context, adding surface head annotations (for the syntactic and lexical head) did not provide any significant improvement in the F1.

References

Jurafsky, Dan, and James H. Martin. *Speech and language processing.* Pearson, 2014.

Manning, Christopher D., and Hinrich Schtze. *Foundations of statistical natural language processing. Vol. 999.* Cambridge: MIT press, 1999.

Collins, Michael, and Terry Koo. *"Discriminative reranking for natural language parsing."* Computational Linguistics 31.1 (2005): 25-70.

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

Charniak, Eugene, and Mark Johnson. *"Coarse-to-fine n-best parsing and MaxEnt discriminative reranking."* Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics. Association for Computational Linguistics, 2005.