Discriminative Reranking

Ziqian Luo
Carnegie Mellon University
ziqianlu@andrew.cmu.edu

Abstract

The project implements two kinds of parsing re-ranker based Perceptron learning algorithm and Primal SVM algorithm. The base parser presents a set of candidate parsers for each sentence, with associated log probabilities that define the initial ranking of those parsers. The re-ranker is going to rearrange the initial ranking by learning from specified features (David Hall, 2014), from the training dataset that has 36765 trees each paired with a gold tree as reference. The ranking performance is evaluated by F1 score and my awesome reranker is the SVM reanker, which has 85.82 F1 score, outperforms the 1-best baseline by 2.16 and my basic reranker is the Perception reranker and it has 85.05 F1 score.

1 Implementation details

Machine learning has shown great success in semantic parsing when we want to identify the best syntactic structure in a natural language sentence. The project trains two reranker to learn the best parse of a sentence by reranking the existing ranked parsers. This method requires a tree to be represented as one row of vectors, say features, to be fed into the reranking model to learn weights from the training corpus. The two reranker will be discussed in 1.1 and 1.2. Feature extraction will be discussed in 1.3.

1.1 Perceptron Learner

The score function used in my perceptron learning algorithm is:

$$s(y) = \mathbf{w}^\top \mathbf{f}(y)$$

where we do the vector multiplication with the weights and feature vector of a certain tree in a k-best tree list to get the new score of that tree. In order to get the tree that has the highest score among all k trees, the following selecting criteria is used: where we compute all the scores and simply store

$$\hat{y} = \arg\max_{y \in \mathcal{L}} \mathbf{w}^\top \mathbf{f}(y)$$

the index of the tree that has the max score among k trees. Before do these jobs, at the very start, we need to define the weight size, which is determined by the feature vector size of a tree. I did this initialization in the first step of my feature extraction loop. After I got the index of the tree that has the highest score, I also need to get the index of the tree that has the highest F1 score before I want to update the weights. A potential problem is that the gold tree might not exist in the k-best tree so a plausible method to solve this is to find the tree that has the highest F1 score and regard it as the new gold tree. The updating formula is shown below:

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

where y hat is the tree has the maximum score and y start is the tree has the maximum F1 score. The true gold tree is only used in calculating F1 scores. Averaging technique is not used here since I found final results will be better and the speed will not be very slower without averaging.

1.2 Primal Subgradient SVM Learner

The primal SVM learner uses AdaGrad as its optimizer. The step size, regularization constant, learning rate and batch size will be discussed in part 3.

The score function is the same as the score function used in perceptron learner and optimization problem is formulated below:

where the loss function is zero one loss and I only used this loss function since it can already

$$\min_{\mathbf{w}} \; k\|\mathbf{w}\|^2 - \sum_i \left(\mathbf{w}^\top \mathbf{f}(y_i^*) - \max_{y \in \mathcal{L}_i} (\mathbf{w}^\top \mathbf{f}(y) + l_i(y)) \right)$$

exceeds the baseline by more than 2 in F1 score. Like discussed in part 1.1, in the SVM learner, the true gold true is also used in calculating F1 scores and the new gold true is the one that has the highest F1 score. Loss is zero if and only if the predicted tree, say the tree that has the highest score among the k best trees, has the same index with the new gold tree. The loss-augmented guess is similar to the guess in perceptron model except for a loss part, which is shown below:

$$\hat{y} = \arg\max_{y \in \mathcal{L}} (\mathbf{w}^\top \mathbf{f}(y) + l(y))$$

A Data class is used to store the best F1 score index and IntCounter class is used to calculate the dot product score.

1.3 Feature Extraction

There are many features that can add more context or grammar information to capture a better semantic parser like position features, rule features, span features, head features, sibling n-gram features, tree n-gram features, neighbours features, heavyness features, right branch features and word features.

There is a lot of work need to be done to select which feature may make more positive contribution to the final result and oceans of combination experiments together with ablation study should be carried out.

However, it is very likely that I will get the same secrets shared in the recitation that span features and word features may be the best. So in this project, I want to investigate in another way,that is how the word feature, one of the best features, will influence my model, if it can remember a different number of its parents.

The word features, shown in Figure 1, will always contain a terminal and a pre-terminal. In the simplest case, it can only remember one parent. In the Figure 1, it can remember three level of its parents.

2 Performance

After a lot of experiments, my awesome SVM reranker can achieve a 85.82 F1 score, which exceeds the 1-best baseline by 2.16. The parameters

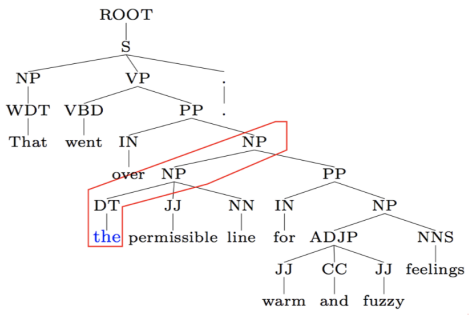

Figure 1: Word Features

are shown below:

* Learning rate(Or step size): 0.01
* Regularization Constant: 0.1
* Batch size: 30

The running screen shot is shown below:

```
Training took 171966 millis
1-best baseline
 [Average]  P: 84.13 R: 83.19 F1: 83.66 EX: 22.31
Decoding took 32 millis
Your system
 [Average]  P: 86.27 R: 85.37 F1: 85.82 EX: 28.68
Decoding took 1623 millis
[luoziqiandeMBP:assign_rerank luoziqian$ ./run.sh
```

In the perceptron model, it takes a much longer time to train the model. I used the best feature combination (will be discussed in 3.1) recorded in SVM model and find this combination will also be the best in perceptron. It can reach a 85.05 F1 score in 17 iterations of training on those features.

3 Experiments

3.1 Experiment results

As discussed in the previous section, I used a new gold tree selected from k-best list instead of the true gold tree to define the loss and it has much more better results.

In order to investigate how the word features will influence my model, I design six kinds of word features in MyFeatureExtractor class. The performance is shown in table 1.

In Table 1, I used Arabic numerals to represent how many levels of parents the terminal can remember. The plus symbol means I combined several kinds of word feature at the same time.

Note that position features and rule features are considered in the feature size.

Combination	F1	Feature size
1	84.6	93711
2	85.04	117465
3	85.57	200560
4	85.7	368692
5	85.52	583856
6	85.33	809375
1+2+3	85.54	325250
3+4	85.74	519521
2+3	85.55	264485
2+3+4	**85.82**	**574684**
1+2+3+4	85.7	644211
2+3+4+5	85.6	1093556

Table 1: Results on SVM Learner

3.2 Experiment illumination

The idea of this experiment is inspired by the query expansion algorithm broadly used in the web search engine. In the web search engine situation, the original query can return some ranked documents and new query terms are learnt from the top k documents retrieved by the original query. The new learnt query consists of the new learnt query terms and uses them to do the query again. Similarly, in this project, I regard the learnt word features as my learnt query terms and use them to rerank the ”documents”.

3.3 Error analysis

However, when the learnt query terms are too long, it can hardly match any documents since the query terms are becoming more and more irrelevant to the documents we want. Here, if the word feature is too large, each rule is too unique so that there will be many unseen features in the test dataset that I can not add them to my TrainFeatureIndexer. And that's why the result will become poor when the feature size is too large.

4 Discussion

4.1 Future Work

I will implement MaxEnt model and compare is with these two models and also try to investigate how different feature will influence the final result. Besides, how to combine those useful features mentioned in 1.3 is also a topic that need much experiments to be done.

References

Dan Klein David Hall, Greg Durrett. 2014. Less grammar, more features. *52nd Annual Meeting of the Association for Computational Linguistics*, 1(14):228–237.