Implementation of Discriminative Reranking

Wei Peng
University of Pittsburgh
wpeng1@andrew.cmu.edu

Abstract

Discriminative reranking is one method constructing high-performance statistical parsers. A discriminative reranker takes a source of candidate parses for each sentence as the input and picks out the "best" one as the prediction. In the work, we implement two rerankers, Perceptron reranker (BASIC) and SVM reranker (AWESOME). The best model we obtained achieves a F1 score of **85.68**.

1 Discriminative Reranking

Typically, there are two main approaches to learn classifier weights. Generative builds a probabilistic model of the data, and then get weights according to local conditional probabilities, while Discriminative sets weights based on some error-related criterion. The main ideas of training a discriminative reranker are representing parses as feature vectors, scoring by linear functions, and learning (the score functions) by optimization. Let $\mathcal{L}$ be the set of candidate parses and $F_1(\mathrm{T}', \mathrm{T}'')$ be the F1 score of trees T' and T'', then the mechanism of discriminative reranker can be illustrated as following:

- Feature Map: $\phi : \mathrm{T} \in \mathcal{L} \to \phi(\mathrm{T}) \in \mathrm{R}^d$.
- Linear Scoring Function: $s(\mathrm{T}) = \mathrm{w}^T \phi(\mathrm{T})$ where $\mathrm{w} \in \mathrm{R}^d$ are learnable parameters.
- Learning w:
  - Perceptron
  - Primal SVM - Subgradient Descent

2 Learning Algorithms

We highlight the main steps of the above two learning algorithms in this section.

2.1 Perception

- Make a guess: $\hat{\mathrm{T}} = \mathrm{argmax}_{\ \mathrm{T}\in\mathcal{L}}\ \mathrm{w}^T \phi(\mathrm{T})$.
- Adjust weights:

$$\mathrm{w} \to \mathrm{w} + \phi(\mathrm{T}^*) - \phi(\hat{\mathrm{T}}), \tag{1}$$

  where T^* is the target tree (could be the gold tree $\mathrm{T}^\dagger$ or $\mathrm{T}^{\dagger\dagger} = \mathrm{argmax}_{\ \mathrm{T}\in\mathcal{L}} F_1(\mathrm{T}, \mathrm{T}^\dagger)$.)
- Decision rule: $\mathrm{argmax}_{\ \mathrm{T}\in\mathcal{L}} \mathrm{w}^T \phi(\mathrm{T})$.

2.2 SVM

- Optimization problem:

$$\min_{\mathrm{w}} k||\mathrm{w}||^2 - \sum_i \left[\mathrm{w}^T\phi(\mathrm{T}^*) - \max_{\mathrm{T}\in\mathcal{L}} \left(\mathrm{w}^T\phi(\mathrm{T}) + l_i(\mathrm{T})\right)\right] \tag{2}$$

  where $l_i(\mathrm{T}) = \begin{cases} 0, & \mathrm{T} = \mathrm{T}^* \\ 1, & \mathrm{T} \neq \mathrm{T}^* \end{cases}$.
- Decision rule: $\mathrm{argmax}_{\ \mathrm{T}\in\mathcal{L}} \mathrm{w}^T \phi(T)$.

See (Shalev-Shwartz et al., 2011) for more details about the subgradient solver for SVM.

3 Experiment

3.1 Feature Extraction

This section describes how each parse tree T is mapped to a feature vector $\phi(T) = (f_1(\mathrm{T}), f_2(\mathrm{T}), \ldots, f_d(\mathrm{T}))$. We outline the feature schemata used in the experiment below. Most of them are inspired by the features and descriptions provided by (Charniak and Johnson, 2005), (Johnson and Ural, 2010) and (Hall et al., 2014). Also, as pointed out by those authors, developing feature schemata is such more of an art than a science, as adding or deleting a single schema usually does not have a significant effect on performance, yet overall impact of many well-chosen schemata can be dramatic. In total, we obtain **2,801,193** unique features.

- **Position**: the position in the k-best list.
- **Rule**: Parent, FirstWord + Rule, LastWord + Rule, Parent + Spanlength + FirstWord + LastWord
- **Span Shape**: Parent + Span of Words, replacing word with first letter being uppercase by X, and word with first letter being lowercase by x.
- **Span context**: Parent + Word Before Span + Categories of Children
- **Split Point**: Rule + Splitting Words.
- **Heavyness**: Parent + SpanLength + Closeness to the End of Sentence.
- **Word + 3 Ancestors**: Lexical items together with categories of 3 of their immediate ancestor nodes.
- **Neighbors**: Parent + SpanLength + Preterminal Before Span + Preterminal After Span
- **RightBranch**: the number of nonterminal nodes that lie on the path from the root node to the right-most non-punctuation preterminal node.

3.2 Implementation Details

Currently, files 200-2199 of the Penn Treebank with 36,765 sentences were used as preliminary training data, and the remaining sentences of files 2300-2399 were used as preliminary test data. We use *featureExtraction* to generate feature vectors for each parse tree T, and store those vectors in `List<List<int[]>> data` temporally. Latter to make things faster, we use *InterCounter* to store the learning weight w and feature vectors, such that for each training instance (or training batch) we only have to touch weight indices corresponding to the non-zero features on that instance (or batch).

As for Perceptron, we train the data with 10 epochs using early stopping as regularization. For SVM, we built *SVMLossAugmentedlinearModel* to return an updateBundle object, i.e. ($\phi(\mathrm{T}^*)$, $\phi(\hat{\mathrm{T}}), l(\hat{\mathrm{T}})$) by implementing the *LossAugmentedLinearModel* interface. Both Perceptron and SVM use $\mathrm{T}^* = \mathrm{T}^{\dagger\dagger}$ in training. Additionally, we printed out the top 10 features in both cases with the assistant of *quicksort*. Lastly, the hyper-parameters in SVM are set as below:

iter	batchSize	regConstant	stepSize
30	700	0.5	1e-2

Table 1: SVM Hyper-parameters

3.3 Experiment Results

The parsing reranker performance we obtained with **2,801,193** unique features are listed below:

Reranker	P	R	F1	EX
1-best baseline	84.13	83.19	83.66	22.31
BASIC	84.73	83.72	**84.22**	18.04
AVESOME	86.14	85.22	**85.68**	26.68

Table 2: Parsing Rerankers Performance

Our AWESOME reranker outperforms the 1-best baseline by **2.2** (>**1.5**) F1 score.

3.4 n-best parses vs. F1 score

The resulting n-bests are fairly good, as shown in Table 3.

n	numFeatures	F1 score
1	1,247,171	83.66
2	1,396,773	84.65
5	1,753,437	**85.5**
10	2,169,010	85.67
15	2,507,750	85.62
25	2,801,193	85.68

Table 3: F1 Score as a function of number n of n best parses

From the 1-best result we see that the base accuracy of the parser is 83.66%. 5-best shows dramatic improvement. After that the things start to slow down, and we achieve the best rate of 85.68% at 20-best. This observation is shown in Figure 1.

3.5 Important Feature Schemata

Additionally, we displayed the top 10 features of Perceptron and SVM in Table 4 and Table 5 correspondingly. Note that Posn=-1 is always the top one feature since it is the unique feature only owned by target parse. We should ignore it since it is unknown during testing.

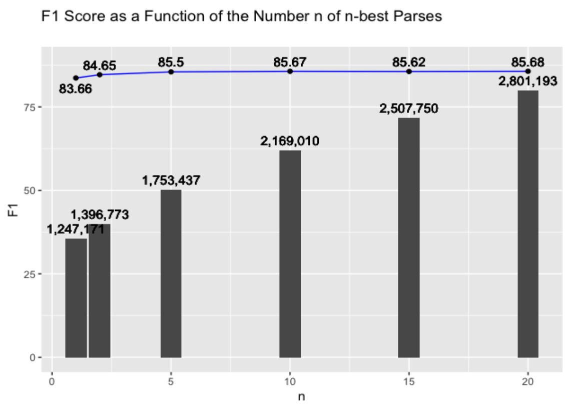

Figure 1: F1 score v.s. n of n-best parses

Top	Feature Schemata (Perceptron)
~~1~~	~~Posn=-1~~
2	NUM_NODES_RIGHTMOST=-2
3	ANCESTOR=. . FRAG ROOT
4	LASTWORD=.@RULE=ROOT → FRAG
5	ANCESTOR=? . FRAG ROOT
6	LASTWORD=?@RULE=ROOT → FRAG
7	RULE=PP → IN NP
8	RULE=VP → VP CC VP
9	SPAN_SHAPE=FRAG → X.
10	ANCESTOR=yesterday NN NP FRAG

Table 4: Perceptron: Top 10 Features

Top	Feature Schemata (SVM)
~~1~~	~~Posn=-1~~
2	ANCESTOR=because IN PP VP
3	SPLIT_POINT=NP → NP...’sNN
4	SPLIT_POINT=NP → NP...’NN
5	RULE=VP → MD ADVP VP
6	ANCESTOR=while IN SBAR VP
7	FIRSTWORD=to@RULE=S → VP
8	ANCESTOR=’s POS NP NP
9	SPLIT_POINT=NP → NP...’sNNS
10	RULE=VP → VP CC VP

Table 5: SVM: Top 10 Features

We found that LAST_WORD + RULE, RULE, WORD + ANCESTOR and SPAN_SHAPE are important feature schemata for Perceptron, whereas for SVM, FIRST_WORD + RULE, RULE, WORD + ANCESTOR, and SPLIT_POINT seem to be more important. And most of those are SPAN related features except for WORD + ANCESTOR. HEAVYNESS, SPAN_CONTEXT, NEIGHBORS and POSTION seem to be less important for both cases.

References

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.

David Hall, Greg Durrett, and Dan Klein. 2014. Less grammar, more features. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 228–237, Baltimore, Maryland. Association for Computational Linguistics.

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.

Shai Shalev-Shwartz, Yoram Singer, Nathan Srebro, and Andrew Cotter. 2011. Pegasos: Primal estimated sub-gradient solver for svm. *Mathematical programming*, 127(1):3–30.