Survey on Three Reranking Models for Discriminative Parsing

Kaidi Ma
Department of Computer and Information Science
University of Delaware
101 Smith Hall, Newark, DE 19716, USA
kdma@udel.edu

Abstract
This survey is inspired by the so-called reranking techniques in natural language processing (NLP). The aim of this survey is to provide an overview of three main reranking tasks particularly for discriminative parsing. We will focus on the motivation for discriminative reranking, on the three models, boosting model, support vector machine (SVM) model and voted perceptron model, on the procedures and on the comparisons between different reranking approaches.

Key Words: reranking, machine learning, natural language processing

1 Introduction
Reranking techniques have been successfully applied to enhance the performance of NLP parsing tasks. Reranking models for parsing usually rely on structures generated within the baseline parser. A baseline generative model produces n-best candidates, which are then reranked using a rich set of local and global features that are not computable or intractable within the baseline system in order to select the best analysis.

Various supervised learning algorithms are introduced to the task of reranking for parsing, such as RankBoost (Collins, [2000]; Collins and Koo, [2003]), MaxEnt-Rank (Charniak and Johnson, [2005]), Voted Perceptron (Collins, [2002]; Collins and Duffy, [2002]; Shen and Joshi, [2004]), and SVMRank (Shen and Joshi, [2003]).

This survey contains three supervised ranking techniques. Boosting approach (Collins [2000]) has been shown successful for NLP reranking tasks. Voted Perceptron (Collins [2002]) shows the similar F-score compared with Boosting model (Collins [2000]) tested on the same dataset. But it provided a fast and easy way to implement. Using SVMRank, Shen and Joshi (2003) also achieved similar performance.

This survey is organized as follows. In section 2, it describes generative model and history-based model. Then the limitation of these models will be discussed in section 2.2. In Section 3, we will show the motivation of discriminative reranking and its purpose. We then summarize different approaches on reranking. Main procedures of reranking including boosting, feature selection, voted perceptron, SVM will be discussed in section 3.1, 3.2, 3.3 and 3.4 respectively.

2 History-based Model and Generative Model

2.1 Background
In NLP processing, the task of a parser is to produce a grammatical interpretation of a sentence which represents the syntactic and semantic intent of the sentence. The parser must have a mechanism that estimates the coherence of an interpretation,
both in isolation and in context. According to Collins (2000), parsing problem can be also regarded as a supervised learning task. It could be described as to find a function \( f : X \rightarrow Y \), given training instances \((x_i, y_i)\) where \( x_i \in X \), \( y_i \in Y \). \( X \) is the set of sentences and \( Y \) is the set of parsing trees. Define \( \text{GEN}(x) \subseteq Y \) to be the set of candidates for a given input \( x \). \( x \) is a sentence in the parsing problem and \( \text{GEN}(x) \) is a set of candidate parsing trees for sentence \( x \). One problem of this model is that the complexity of \( \text{GEN}(x) \) may be very large and internal structure of each candidate parsing tree may be very complicate. It makes the parsing problem too complicated to be solved as a simple classification task which has a fixed, small set of classes.

Generative probabilistic models provide a different way to achieve the same goal. In probabilistic approaches, a model is defined to assign a probability \( P(x, y) \) to each \((x, y)\) pair. The most likely parsing tree for each sentence \( x \) is then got by using \( \arg \max_{y \in \text{GEN}(x)} P(x, y) \). Probabilistic language models, however, need strong independence assumptions. Contextual information is very limited in probabilistic models because of the difficulty in estimating rich probabilistic models of context. Black et al. (1992) adopted history-based models which takes the advantage of contextual information in the history and showed their importance in NLP parsing. There are several good performing history-based model parsers on the WSJ treebank including Ratnaparkhi (1997); Charniak (2000); Collins (1999); Henderson (2003). Conditional history-based models define a one-to-one mapping between each pair \((x, y)\) and a decision sequence \((d_1, ..., d_{i-1})\). This sequence can be thought of as the sequence of moves that build \((x, y)\) in order. Given this mapping, the probability of a tree can be written as

\[
P(x, y) = \prod_{i=1}^{n} P(d_i | \Phi(d_1, ..., d_{i-1}))
\]

Where \( d_1, ..., d_{i-1} \) are the decisions or history made in building a parse tree for the \( i \)th iteration, \( \Phi \) is a function which sets histories into equivalence classes.

History-based approaches lead to models in which the log-probability of a parse tree can be written as a linear sum of parameters \( \alpha_k \) multiplied by features \( h_k \). Each feature \( h_k(x, y) \) is the count of a different event within the tree. Using probabilistic context-free grammars (PCFG) as an example, consider a PCFG with rules \( A_k \rightarrow \beta_k \) for \( 1 \leq k \leq m \). If \( h_k(x, y) \) is the number of times \( A_k \rightarrow \beta_k \) is seen in the tree, and \( \alpha_k = logP(\beta_k | A_k) \) is the weight associated with that rule, thus

\[
logP(x, y) = \sum_{k=1}^{m} \alpha_k h_k(x, y)
\]

Usually, the weight \( \alpha_k \) represents the influence of each feature on the score of a tree and its value is trained from training instances.

### 2.2 Limitation

It is important to study history-based models and learn their limitations in the parsing task. One important issue is about the efficiency of the history-based model. Usually the parsing task is a fairly large problem, involving around one million parse trees and tremendous features.

Another problem with history-based models is that it is difficult to specify features as arbitrary predicates of the entire tree. In particular, previous work (Johnson et al. [1999]; Riezler et al. [2002]) has investigated the use of Markov random fields (MRFs) or log-linear models as probabilistic models with global features for parsing and other NLP tasks. (Log-linear models are often referred to as maximum-entropy models in the NLP literature.). It has been found that the choice of derivation has an intense influence on the parameterization of the parsing tree.

When designing a parsing model, it would be desirable to have a mechanism in which features can be easily added to the model. Unfortunately, with history-based models adding new features often requires a modification of the underlying derivations in the model. Modifying the derivation to include a new feature type can be a painstaking task. In an ideal situation, arbitrary features \( h_k \) could be encoded into the model without worrying about modifying a sequence of derivations to fit in the new features.
3 Discriminative Reranking Approaches

Discriminative models have the advantages of updating the weights assigned to each features instead modifying the global derivations. In addition, the use of global features is critical to the performance of many NLP applications. However, the introduction of global features results in difficulty for the dynamic programming of the generative history-based models. On the other hand, a rich set of local and global features are not computable or intractable within the baseline system. Therefore the generative model often output a set of parse trees with probabilities. The main idea of the reranking is as follows: based on the n-best candidates generated from a baseline generative model, these candidates can be reranked by using the set of local and global features which would improve the performance. Usually, only the top candidate of the reranked results is regarded as final parsing tree.

Several reranking approaches have been done since the year of 2000. Charniak (2000) reranked the n-best parsing trees by re-estimating a language model on a large number of features. Collins (2000) used Boost Loss and Log-Likelihood Loss to do a boosting based reranking task. The Boost Loss model generated better result and we will show the Boosting approach in section 3.1. In Collins and Duffy (2002), the voted is applied in parse reranking. Similar to Collins (2000), pairwise samples were used as training samples, although implicitly. In section 3.3, we will show how these perceptron algorithms work. Shen and Joshi (2003) adopted SVM and Tree kernels to their model and gain good result. The SVM was used to maximize the margin between positive samples and negative samples, which in turn was proportional to the margin between the best parse of each sentence and the rest of the n-best parses. The details of SVM approach will be shown in section 3.4.

3.1 Boosting

Collins (2000) introduces a boosting-based approach for the reranking task based on previous boosting approach described in Freund et al. (1998). The algorithm can be viewed as a feature selection method, optimizing a particular loss function (the exponential loss function in Collins [2000]’s work). The baseline model (Collins [1999]) achieved 88.2% F-measure on this task. The reranking model achieves 89.75% F-measure, a 13% relative decrease in F-measure error.

Loss function is the central idea in the boosting model. In Collins (2000), the loss function is defined as:

Let $x_{i,j}$ be the feature vector of the $j$th parse of the $i$th sentence. Let $\tilde{x}_i$ or $x_{i,1}$ be the best parse for the $i$th sentence. A score function $F(\alpha, x_{i,j})$ is defined as

$$F(\alpha, x_{i,j}) = \alpha \cdot x_{i,j},$$

where $\alpha$ is a weight vector.

A margin $M_{\alpha,i,j}$ is defined on the $i$th training example, given weight vector $\alpha$ as

$$M_{\alpha,i,j} = F(\alpha, \tilde{x}_i) - F(\alpha, x_{i,j}).$$

Finally the Boost Loss function is defined as:

$$\text{BoostLoss}(\alpha) = \sum_i \sum_j e^{-M_{\alpha,i,j}} = \sum_i \sum_j e^{-(F(\alpha, \tilde{x}_i) - F(\alpha, x_{i,j}))}$$

The Boosting algorithm was used to search the weight vector $\alpha$ to minimize the Boost Loss function.

An improved boosting algorithm is defined as follows:

Input:

- Examples $x_{i,j}$ for $i=1...n$, $j=1...ni$, drawn from set $\chi$
- Weights $S_{i,j}$ representing importance of examples where
  $$S_{i,j} = \text{Score}(x_{i,1}) - \text{Score}(x_{i,j}).$$
  Score($x_{i,j}$) is a measure of how good a parse is, could be use a F-measure in the implementation.
- Initial model log-likelihood $L(x_{i,j})$, for all examples $x_{i,j}$
- Feature functions $h_k: \chi \rightarrow \{0,1\}$ for $k=1...m$
- Smoothing parameter $\epsilon$ (chosen from optimization on development data)
- Number of iteration N (chosen from optimization on development data)

Initialize:

- Set $\alpha_0 = \text{argmin}_\alpha \sum_i \sum_{j=2}^n S_{i,j} e^{-\alpha(L(x_{i,1}) - L(x_{i,j}))}$
Set $\alpha_k=0$ for $k=1,\ldots,m$

For all $i$, $2 \leq j \leq n_i$, set margins $M_{ij}=\alpha_k[L(x_{i,j}) - L(x_{i,j})]$ for all $k=1,\ldots,m$

Set $A_k^+=\{(i,j): [h_k(x_{i,j})-h_0(x_{i,j})]=1\}$, $A_k^-$ is the set of training examples in which the $k$th feature is seen in the correct parse but not in the competing parse.

Set $\tilde{A}_k^+\tilde{A}_k^+=\{(i,j): [h_0(x_{i,j})-h_k(x_{i,j})]=1\}$, $\tilde{A}_k^-$ is the set in which the $k$th feature is seen in the incorrect parse.

For all $i$, $2 \leq j \leq n_i$

Set $B_k^+_{ij}=[k: [h_k(x_{i,j})-h_0(x_{i,j})]=1]$

Set $B_k^-_{ij}=[k: [h_0(x_{i,j})-h_k(x_{i,j})]=1]$

$B_k^+_{ij}$ and $B_k^-_{ij}$ are reverse indices from training examples.

- to features
  - Calculate $Z=\sum_i \sum_{j=2}^{n_i} S_{i,j} e^{-M_{ij}}$
  - For $k=1,\ldots,m$
    - Set $W_k^+=W_k^-=0$
    - For $(i,j)\in A_k^+$, $W_k^+=W_k^++S_{i,j} e^{-M_{ij}}$
    - For $(i,j)\in A_k^-$, $W_k^-=W_k^-+S_{i,j} e^{-M_{ij}}$
    - $G_k=[\sqrt{W_k^+}-\sqrt{W_k^-}]$
  - Repeat: for $t=1,\ldots,N$
    - Choose $k^*=\text{argmax}_k G_k$ and $\delta^*=1/2\log(1+\Delta)$
    - For $(i,j)\in A_k^+$
      - Set $\Delta = S_{i,j} (e^{-M_{ij}} - \delta^* - e^{-\delta^*})$
      - Set $M_{ij}=M_{ij} + \delta^*$
      - Set $Z=Z+\Delta$
    - For $k\in B_k^+_{ij}$, $W_k^+=W_k^++\Delta$
    - For $k\in B_k^-_{ij}$, $W_k^-=W_k^-+\Delta$
    - For $(i,j)\in \tilde{A}_k^+$
      - Set $\Delta = S_{i,j} (e^{-\delta^*} - e^{-M_{ij}})$
      - Set $M_{ij}=M_{ij} - \delta^*$
      - Set $Z=Z+\Delta$

- For $k\in B_k^+_{ij}$, $W_k^+=W_k^++\Delta$
- For $k\in B_k^-_{ij}$, $W_k^-=W_k^-+\Delta$
- For feature $k$ whose values of $W_k^+$ and/or $W_k^-$ have changed, update $G_k$

$\alpha^t = \text{Update}(\alpha^{t-1}, k^*, \delta^*)$

Where

$\text{Update}(\alpha^t, k^*, \delta^*) = \{\alpha_0, \alpha_1, \ldots, \alpha_k + \delta^*, \ldots, \alpha_m\}$

### 3.2 Feature Selection Methods

Feature selection methods have been proposed in the maximum-entropy literature by several researchers (McCallum [2003]; Zhou et al. [2003]; Riezler and Vasserman [2004]). The basic approach is to select a single feature at each iteration, then update the entire model, as follows:

Step 1: Throughout the algorithm which aims to minimize a loss function, maintain a set of active features. Initialize this set to be empty.

Step 2: Choose a feature from outside of the set of active features which has the largest estimated impact in terms of reducing the loss function, and add this to the active feature set.

Step 3: Minimize loss function with respect to the set of active features; allow only the active features to take nonzero parameter values when minimizing a Log Loss function. Return to Step 2.

Log Loss function is defined as:

$\text{LogLoss}(\alpha) = \sum_i \sum_j \log(e^{F_\alpha(x_{i,j})})$

$= \sum_i \sum_j \log(1 + \sum_j e^{-M_{i,j}})$

Collins (2000) boosting method can be regarded as feature selection methods of the following form:

Step 1: Start with all parameter values set to zero.

Step 2: Choose a feature which has largest estimated impact in terms of reducing the loss function $\text{BoostLoss}(\alpha)$.

Step 3: Update the parameter for the feature chosen at Step 2 in such a way as to minimize $\text{BoostLoss}(\alpha)$ with respect to this one parameter. All other parameter values are left fixed. Return to Step 2.
The maximum-entropy feature selection method might be quite inefficient, due to the reason that updating entire model at each step. The boosting approach only adjusts one parameter value within iteration, and does not update the entire model at each round of feature selection.

### 3.3 Perceptron

Using perceptron-based algorithms is an alternative way to do the reranking task. In (Collins and Duffy, [2002a]), the use of Voted Perceptron for the parse reranking problem has been described. In that paper, Collins and Duffy (2002a) used a tree kernel (Collins and Duffy, [2001]) to count the number of common subtrees. The voted perceptron and the Tree kernel were applied to rerank the candidate parses. Similar to Collins (2000), pairwise samples were used as training samples. The perceptron updating step was defined as

\[ w^{t+1} = w^t + x_{i,j} - x_{i,1} \]

where \( w^t \) is the weight vector at \( t \)th iteration and \( x_{i,j} \) is the feature vector of the \( j \)th rank parse of the \( i \)th sentence. \( x_{i,1} \) is the best parse for the \( i \)th sentence.

In Collins (2002a), the perceptron training algorithm is defined as:

Define: \( F(x, w) = w \cdot h(x) \)

Input: Examples \( x_{i,j} \) with feature vectors \( h(x_{i,j}) \).

Initialization: Set parameters \( w^0 = 0 \)

For \( i = 1 \ldots n \)

\[ j = \text{argmax}_{j=1 \ldots m} F(x_{k}, w_{i-1}) \]

If \( j = 1 \) Then \( w^i = w^{i-1} \)

Else \( w^i = w^{i-1} + h(x_{i,j}) - h(x_{i,1}) \)

Output: Parameter vectors \( w^i \) for \( i = 1 \ldots n \)

The training algorithm maintains a parameter vector \( w \), which is set to all zeros at the beginning. The algorithm then makes a go-through the training set, at each training example storing a weight vector \( w^t \) for \( i = 1 \ldots n \). The parameter vector is only modified when a mistake is made on an example. In this case, weight update is very simple, involving adding the difference of the offending examples’ representations:

\[ w^{t+1} = w^t + x_{i,1} - x_{i,j} \]

In the most basic form of the perceptron, the parameter values \( w^0 \) are taken as the final parameter settings, and the output on a new test example with \( x_j \) for \( j = 1 \ldots m \) is simply the highest scoring candidate under these parameter values, i.e., \( x_k \) where \( k = \text{argmax}_j w^n \cdot h(x_j) \).

And the algorithm for testing is defined as follows:

Define: \( F(x, w) = w \cdot h(x) \)

Input: A set of candidates \( x_j \) for \( j = 1 \ldots m \) with A sequence of parameter vectors \( w^i \) for \( i = 1 \ldots n \)

Initialization: Set parameters \( V[j] = 0 \) where \( V[j] \) stores the number of votes for \( x_j \)

For \( i = 1 \ldots n \)

\[ j = \text{argmax}_{k=1 \ldots m} F(x_k, w^i) \]

\[ V[j] = V[j] + 1 \]

Output: \( x_j \) where \( j = \text{argmax}_{k=1 \ldots m} V[k] \)

For the voted perceptron, all parameter vectors \( w^i \) for \( i = 1 \ldots n \) are stored. Thus the training phase can be regarded as a way of constructing \( n \) different weight settings. Each of these weight settings will have its own highest ranking candidate, \( x_k \), where \( k = \text{argmax}_j w^n \cdot h(x_j) \). The idea behind the voted perceptron is to take each of the \( n \) parameter settings to vote for a candidate, and the candidate which gets the most votes is returned as the most likely candidate.

### 3.4 SVM

Shen, Sarkar, and Joshi (2003) describe a SVM based approach for reranking. Unlike the Loss function based algorithms, SVM approaches search for the hyperplane that separates a set of training samples that contain two distinct classes and then maximizes the margin between these two classes. A good hyperplane can separate the best candidate from the other not-so-good candidates. One reason for choosing SVM is that SVM has the ability to maximize the margin. In addition, SVM can achieve high performance even with input data of high dimensional feature space, by using of kernel.

Let \( x_{i,j} \) be the \( j \)th candidate parse for the \( i \)th sentence in training data. Let \( x_{i,1} \) is the parse with the highest f-score among all the parses for the \( i \)th sentence. Instead of using the parse tree as a training sample, Shen et al (2003) use a pair of
parse trees as a sample. For any \(i\) and \(j > 1\), \((x_{i,j}, x_{k,1})\) is positive sample, and \((x_{i,j}, x_{i,1})\) is negative sample.

Similar idea was employed in the boosting algorithm of (Collins, 2000): for each parse \(x_{i,j}\), its margin is defined as \(F(x_{i,1}) - F(x_{i,j})\). In (Collins and Duffy, 2002), for each offending parse, the parameter vector updating function is in the form of \(w^{t+1} = w^t + h(x_{i,1}) - h(x_{i,j})\).

The SVM approach, the kernel on pairs of parses is defined as:

Let \((t_1, t_2)\), \((v_1, v_2)\) denote two pairs of parses. Let \(K\) denote any kernel function on the space of single parses. The preference kernel \(P_k\) is defined on \(K\) as follows.

\[
P_k((t_1, t_2), (v_1, v_2)) = K(t_1, v_1) - K(t_1, v_2) - K(t_2, v_1) + K(t_2, v_2)
\]

The preference kernel of this form was previously used in the context of ordinal regression in (Herbrich et al., 2000). The decision function is then defined as:

\[
f((x_j, x_k)) = \sum_{i=1}^{N_s} \alpha_i y_i P_k((s_{i,1}, s_{i,2}), (x_j, x_k)) + b
\]

where \(x_j\) and \(x_k\) are two distinct parses of a sentence, \((s_{i,1}, s_{i,2})\) is the \(i\)th support vector, and \(N_s\) is the total number of support vectors. Given the definitions, the training samples are symmetric with respect to the origin in the space. Therefore, for any hyperplane that does not pass through the origin, there is always a parallel hyperplane that crosses the origin and makes the margin larger. Hence, the outcome hyperplane has to pass through the origin, which means that it is unbiased and \(b = 0\).

In this approach, for each test parse \(x\), score can be computed by given the definition as:

\[
\text{score}(x) = \sum_{i=1}^{N_s} \alpha_i y_i (K(s_{i,1}, x) - K(s_{i,2}, x))
\]

then

\[
f((x_j, x_k)) = \text{score}(x_j) - \text{score}(x_k)
\]

The SVM was used to maximize the margin between positive samples and negative samples, which in turn was proportional to the margin between the best parse of each sentence and the rest of the N-best parses.

The performance of SVM approaches (Shen et al [2003]) is a little lower than the results of the boosting approach in (Collins, 2000) tested on the same dataset.

### 4 Conclusion

This survey introduced three basic discriminative models for reranking task, including boosting, voted perceptron and SVM approaches. All three approaches are using pairwise samples in training and updating the weight of features. These three approaches share almost similar performance on the same dataset. Boosting and voted-perceptron are slightly higher than SVM approach in F-score. Voted-perceptron is every quick in training and easy to implement. Although there are many newer versions of these approaches, the basic ideas remain the same.

### 5 Acknowledgement

I would like to thank Prof. Shanker for offering the course CISC889 in which I learned a lot of new stuff in NLP area.

### References


