# Edit Detection and Parsing for Transcribed Speech

**Eugene Charniak and Mark Johnson**
Brown Laboratory for Linguistic Information Processing (BLLIP)

**Abstract**

We present a simple architecture for parsing transcribed speech in which an edited-word detector first removes such words from the sentence string, and then a standard statistical parser trained on transcribed speech parses the remaining words. The edit detector achieves a misclassification rate on edited words of 2.2%. (The NULL-model (which marks everything as not edited) has an error rate of 5.9%.) To evaluate our parsing results we introduce a new evaluation metric, the purpose of which is to make evaluation of a parse tree relatively indifferent to the exact tree position of EDITED nodes. By this metric the parser achieves 85.3% precision and 86.5% recall.

## 1 Introduction

While there has been significant effort expended on the parsing of written text, parsing speech has received very little attention. The comparative neglect of speech (or transcribed speech) is understandable as parsing transcribed speech presents several problems absent in regular text: "um"s and "ah"s (or more formally, filled pauses), frequent use of parentheticals (e.g., "you know"), ungrammatical constructions, and speech repairs (e.g., " Why didn't he, why didn't she stay home?").

In this paper we present and evaluate a simple two-pass architecture for handling the problems of parsing transcribed speech. The first pass tries to identify which of the words in the string are edited ("why didn't he," in the above example). These words are removed from the string given to the second pass, an already existing statistical parser trained on a transcribed speech corpus. (In particular, all of the research in this paper was performed on the parsed "Switchboard" corpus as provided by the Linguistic Data Consortium).

This architecture is based upon a fundamental assumption; the semantic and pragmatic content of an utterance is based solely on the unedited words in the word sequence. This assumption is not completely true, corrected Freudian slips would be a counter example. However, it seems so close to true as to make worrying about the lost content seem perverse, particularly given our limited ability at extracting semantic content in the best of circumstances.

In order to evaluate the parser's output we compare it with the gold standard parse trees. For this purpose a very simple third pass is added to the architecture; the hypothesized edited words are inserted into the parser output (see Section 3 for details). To the degree that our fundamental assumption holds, a "real" application would ignore this last step.

This architecture has several things to recommend it. First it allows us to treat the editing problem as a pre-process, keeping the parser unchanged. Second, the major clues for detecting edited words in transcribed speech seem to be relatively shallow phenomena, such as repeated word and part-of-speech sequences. The kind of information that a parser would add, e.g., what is the node dominating the EDITED node, seems much less critical.

Note that of the major problems associated with transcribed speech, we chose to deal with only one of them, speech repairs, in a special fashion. Our reasoning here is based upon what one might and might not expect from a second-pass statistical parser. For example, ungrammaticality in some sense is relative, so if the training corpus contains the same kind of ungrammatical examples as the testing corpus, one would not expect ungrammaticality itself to be a show stopper. Furthermore, the best statistical parsers [3,6] do not use grammatical rules, but rather define probability distributions over all possible rules.

Similarly, parentheticals and filled pauses already exist in the newspaper text these parsers currently handle, albeit at a much lower rate. Thus there is no particular reason to believe that these constructions should have a major impact.[1] This leaves speech repairs as the one major phenomena not present in the text that would seem to pose a major problem for our parser. It is for that reason we have chosen to handle it separately.

The organization of this paper follows the architecture we have just described. Section 2 describes the first pass. We present therein a boosting model for learning to detect edited nodes (Sections 2.1 – 2.2), and an evaluation of the model as a stand-along edit detector (Section 2.3). Section 3 describes the parser. As the parser is that already reported in [3] this section simply describes the parsing metrics used (Section 3.1), the details of the experimental setup (Section 3.2), and the results (Section 3.3).

## 2 Identifying EDITED words

The Switchboard corpus annotates disfluencies such as restarts and repairs. These include repetitions and substitutions, shown italicized in (1a) and (1b) respectively.

(1) a. *I really,* I really like pizza.

b. *Why didn't he,* why didn't she stay home?

Restarts and repairs are indicated by disfluency tags '[', '+' and ']' in the disfluency POS-tagged Switchboard corpus, and by EDITED nodes in the tree-tagged corpus. This section describes a procedure for automatically identifying words corrected by a restart or repair, i.e., which are dominated by an EDITED node in the tree-tagged corpus.

This method treats the problem of identifying EDITED nodes as a word token classification problem, where each word token is classified either as edited or not. The classifier applies to words only; punctuation inherits the classification of the preceeding word. A linear classifier trained by a greedy Boosting algorithm [11] is used to predict whether a word token is edited. Our Boosting classifier is directly based on the greedy Boosting algorithm described by Collins [4] which contains important implementation details that are not repeated here. We chose Collins' algorithm because it offers good performance and scales to hundreds of thousands of possible feature combinations.

### 2.1 Boosting estimates of linear classifiers

This section describes the kinds of linear classifiers that the Boosting algorithm infers. Abstractly, we regard each word token as an event characterized by a finite tuple of random variables

$$(Y, X_1, \ldots, X_m).$$

$Y$ is the the *conditioned variable* and ranges over $\{-1, +1\}$, with $Y = +1$ indicating that the word is not edited. $X_1, \ldots, X_m$ are the *conditioning variables*; each $X_j$ ranges over a finite set $\mathcal{X}_j$. For example, $X_1$ is the orthographic form of the word and $\mathcal{X}_1$ is the set of all words observed in the training section of the corpus. Our classifiers use $m = 18$ conditioning variables. The following subsection describes the conditioning variables in more detail; they include variables indicating the POS tag of the preceding word, the tag of the following word, whether or not the word token appears in a "rough copy" as explained below, etc.

The goal of the classifier is to predict the value of $Y$ given values for $X_1, \ldots, X_m$. The classifier makes its predictions based on the occurence of combinations of conditioning variable/value pairs called features. A *feature* $\mathcal{F}$ is a set of variable-value pairs $\langle X_j, x_j \rangle$, with $x_j \in \mathcal{X}_j$. Our classifier is defined in terms of a finite number $n$ of features $\mathcal{F}_1, \ldots, \mathcal{F}_n$, where $n \approx 10^6$ in our classifiers.[2] Each feature $\mathcal{F}_i$ de-

[1] Indeed, [12] suggests that filled pauses tend to indicate clause boundaries, and thus may be a *help* in parsing.

[2] It turns out that many pairs of features are extensionally equivalent, i.e., take the same values on each word token in our training data. We developed a method for quickly identifying such extensionally equivalent feature pairs based on hashing XORed random bitmaps, and deleted all but one of each set of extensionally equivalent features (we kept a feature with the smallest number of conditioning variables).

fines an associated random boolean variable

$$F_i = \prod_{\langle X_j, x_j \rangle \in \mathcal{F}_i} (X_j{=}x_j),$$

where $(X{=}x)$ takes the value 1 if $X = x$ and 0 otherwise. That is, $F_i = 1$ iff $X_j = x_j$ for all $\langle X_j, x_j \rangle \in \mathcal{F}_i$.

Our classifier estimates a *feature weight* $\alpha_i$ for each feature $\mathcal{F}_i$, which is used to define the *prediction variable* $Z$.

$$Z = \sum_{i=1}^{n} \alpha_i F_i.$$

The prediction made by the classifier is $\text{sign}(Z) = Z/|Z|$, i.e., $-1$ or $+1$ depending on the sign of $Z$.

Intuitively, our goal is to adjust the vector of feature weights $\vec{\alpha} = (\alpha_1, \ldots, \alpha_n)$ to minimize the expected *misclassification rate* $\text{E}[(\text{sign}(Z) \neq Y)]$. This function is difficult to minimize, so our Boosting classifier minimizes the expected *Boost loss* $\text{E}[\exp(-YZ)]$. As Singer and Schapire [11] point out, the misclassification rate is bounded above by the Boost loss, so a low value for the Boost loss implies a low misclassification rate.

Our classifier estimates the Boost loss as $\widehat{\text{E}}_t[\exp(-YZ)]$, where $\widehat{\text{E}}_t[\cdot]$ is the expectation on the empirical training corpus distribution. The feature weights are adjusted iteratively; one weight is changed per iteration. The feature whose weight is to be changed is selected greedily to minimize the Boost loss using the algorithm described in [4]. Training continues for 25,000 iterations. After each iteration the misclassification rate on the development corpus $\widehat{\text{E}}_d[(\text{sign}(Z) \neq Y)]$ is estimated, where $\widehat{\text{E}}_d[\cdot]$ is the expectation on empirical development corpus distribution. While each iteration lowers the Boost loss on the training corpus, a graph of the misclassification rate on the development corpus versus iteration number is a noisy U-shaped curve, rising at later iterations due to overlearning. The value of $\vec{\alpha}$ returned by the estimator is the one which minimizes the misclassficiation rate on the development corpus; typically the minimum is obtained after about 12,000 iterations, and the feature weight vector $\vec{\alpha}$ contains around 8,000 non-zero feature weights (since some weights are adjusted more than once).[3]

### 2.2 Conditioning variables and features

This subsection describes the conditioning variables used in the EDITED classifier. Many of the variables are defined in terms of what we call a rough copy. Intuitively, a *rough copy* identifies repeated sequences of words that might be restarts or repairs. Punctuation is ignored for the purposes of defining a rough copy, although conditioning variables indicate whether the rough copy includes punctuation. A rough copy in a tagged string of words is a substring of the form $\alpha_1\beta\gamma\alpha_2$, where:

1. $\alpha_1$ (the *source*) and $\alpha_2$ (the *copy*) both begin with non-punctuation,

2. the strings of non-punctuation POS tags of $\alpha_1$ and $\alpha_2$ are identical,

3. $\beta$ (the *free final*) consists of zero or more sequences of a free final word (see below) followed by optional punctuation, and

4. $\gamma$ (the *interregnum*) consists of sequences of an interregnum string (see below) followed by optional punctuation.

The set of *free-final words* includes all partial words (i.e., ending in a hyphen) and a small set of conjunctions, adverbs and miscellanea, such as *and, or, actually, so*, etc. The set of *interregnum strings* consists of a small set of expressions such as *uh, you know, I guess, I mean,* etc. We search for rough copies in each sentence starting from left to right, searching for longer copies first. After we find a rough copy, we restart searching for additional rough copies following the free final string of the previous copy. We say that a word token is in a rough copy iff it appears in either the source or the free final.[4] (2) presents an example of a rough copy.

[3] We used a smoothing parameter $\epsilon$ as described in [4], which we estimate by using a line-minimization routine to minimize the classifier's minimum misclassification rate on the development corpus.

[4] In fact our definition of rough copy is more complex. For example, if a word token appears in an interregnum and the word immediately following the interregnum also appears in a (different) rough copy, then we say that the interregnum word token appears in a rough copy. This permits us to approximate the Switchboard annotation convention of annotating interregna as EDITED if they appear in iterated edits.

(2) I thought $\underbrace{\text{I}}_{\alpha_1}$ $\underbrace{\text{cou-,}}_{\beta}$ $\underbrace{\text{I mean,}}_{\gamma}$ $\underbrace{\text{I}}_{\alpha_2}$ would finish the work

Table 1 lists the conditioning variables used in our classifier. In that table, subscript integers refer to the relative position of word tokens relative to the current word; e.g. $T_1$ is the POS tag of the following word. The subscript $f$ refers to the tag of the first word of the free final match. If a variable is not defined for a particular word it is given the special value 'NULL'; e.g., if a word is not in a rough copy then variables such as $N_m$, $N_u$, $N_i$, $N_l$, $N_r$ and $T_f$ all take the value NULL. Flags are boolean-valued variables, while numeric-valued variables are bounded to a value between 0 and 4 (as well as NULL, if appropriate). The three variables $C_t$, $C_w$ and $T_i$ are intended to help the classifier capture very short restarts or repairs that may not involve a rough copy. The flags $C_t$ and $C_i$ indicate whether the orthographic form and/or tag of the next word (ignoring punctuation) are the same as those of the current word. $T_i$ has a non-NULL value only if the current word is followed by an interregnum string; in that case $T_i$ is the POS tag of the word following that interregnum.

As described above, the classifier's features are sets of variable-value pairs. Given a tuple of variables, we generate a feature for each tuple of values that the variable tuple assumes in the training data. In order to keep the feature set managable, the tuples of variables we consider are restricted in various ways. The most important of these are constraints of the form 'if $X_j$ is included among feature's variables, then so is $X_k$'. For example, we require that if a feature contains $P_{i+1}$ then it also contains $P_i$ for $i \geq 0$, and we impose a similiar constraint on POS tags.

### 2.3 Empirical evaluation

For the purposes of this research the Switchboard corpus, as distributed by the Linguistic Data Consortium, was divided into four sections (or sub-corpora). The training sub corpus consists of all files in the directories 2 and 3 of the parsed/merged Switchboard Corpus. Directory 4 is split into three approximately equal size sections. (Note that the files are *not* consecutively numbered.) The first of these (files sw4004.mrg to sw4153.mrg) is the testing corpus. All edit detection and parsing results reported herein are from this sub corpus. The files sw4154.mrg to sw4483.mrg are reserved for future use. The files sw4519.mrg to sw4936.mrg are the development corpus. In the complete corpus three parse trees were sufficiently ill formed in that our tree-reader failed to read them. These trees received trivial modifications to allow them to be read, e.g., adding the missing extra set of parentheses around the complete tree.

We trained our classifier on the parsed data files in the training and development sections, and evaluated the classifer on the test section. Section 3 evaluates the parser's output in conjunction with this classifier; this section focusses on the classifier's performance at the individual word token level. In our complete application, the classifier uses a bitag tagger to assign each word a POS tag. Like all such taggers, our tagger has a non-neglible error rate, and these tagging could conceivably affect the performance of the classifier. To determine if this is the case, we report classifier performance when trained both on "Gold Tags" (the tags assigned by the human annotators of the Switchboard corpus) and on "Machine Tags" (the tags assigned by our bitag tagger). We compare these results to a baseline "null" classifier, which never identifies a word as EDITED. Our basic measure of performance is the word misclassification rate (see subsection 2.1). However, we also report precision and recall scores for EDITED words alone. These results show that our classifier makes approximately only 1/3 of the misclassification errors made by the null classifier, and that using the POS tags produced by the bitag tagger does not have much effect on the classifier's performance.

## 3 Parsing Transcribed Speech

We now turn to the second pass of our two-pass architecture, using an "off the shelf" statistical parser to parse the transcribed speech after having removed the words identified as edited by the first pass. We first define the evaluation metric we use and then describe the results of our experiments.

### 3.1 Parsing Metrics

In this section we describe the metric we use to grade the parser output. As a first desiderata we want a metric that is a logical extension

| | |
|---|---|
| $W_0$ | Orthographic word |
| $P_0, P_1, P_2, P_f$ | Partial word flags |
| $T_{-1}, T_0, T_1, T_2, T_f$ | POS tags |
| $N_m$ | Number of words in common in source and copy |
| $N_u$ | Number of words in source that do not appear in copy |
| $N_i$ | Number of words in interregnum |
| $N_l$ | Number of words to left edge of source |
| $N_r$ | Number of words to right edge of source |
| $C_t$ | Followed by identical tag flag |
| $C_w$ | Followed by identical word flag |
| $T_i$ | Post-interregnum tag flag |

Table 1: Conditioning variables used in the EDITED classifier.

| | Classifer | | |
|---|---|---|---|
| | Null | Gold Tags | Machine Tags |
| Misclassification rate | 0.059 | 0.021 | 0.022 |
| EDITED precision | – | 0.952 | 0.944 |
| EDITED recall | 0 | 0.678 | 0.668 |

Table 2: Performance of the "null" classifier (which never marks a word as EDITED), and Boosting classifiers trained on "Gold Tags" and "Machine Tags".

of that used to grade previous statistical parsing work. We have taken as our starting point what we call the "relaxed labeled precision/recall" metric from previous research (e.g. [6,Charniak 2000 Parsing ],). This metric is characterized as follows:

For a particular test corpus let $N$ be the total number of non-terminal (and non pre-terminal) constituents in the gold standard parses. Let $M$ the the number of such constituents returned by the parser, and let $C$ be the number of these that are correct (as defined below). Then precision $= C/M$, and recall $= C/N$.

A constituent $c$ is correct if there exists a constituent $d$ in the gold standard such that:

1. label($c$) = label($d$)[5]
2. begin($c$) $\equiv_r$ begin($d$)
3. end($c$) $\equiv_r$ end($d$)

In 2 and 3 above we introduce an equivalence relation $\equiv_r$ between string positions. We say $a \equiv_r b$ iff

1. $a = b$, or
2. the word separating $a$ and $a'$ is punctuation, and $a' \equiv_r b$.

The parsing literature uses $\equiv_r$ rather than $=$ because it is felt that two constituents should be considered equal if they disagree only in the placement of, say, a comma (or any other sequence of punctuation), where one constituent includes the punctuation and the other excludes it.

Our new metric, "relaxed *edited* labeled precision/recall" is identical to relaxed labeled precition/recall except for two modifications. First, in the gold standard all non-terminal subconstituents of an EDITED node are removed and the terminal constituents are made immediate children of a single EDITED node. Furthermore, two or more EDITED nodes with no separating non-edited material between them are merged into a single EDITED node. We call this version a "simplified gold standard parse." All precision recall measurements are taken with respected to the simplified gold standard.

Second, we replace $\equiv_r$ with a new equivalence relation $\equiv_e$. Assuming without loss of generality that $a \leq b$, then $a \equiv_e b$ iff:

1. $a \equiv_r b$, or
2. there is an EDITED node $c$ (in the simplified gold standard parse) such that $a \equiv_e$ begin($c$) and end($c$) $\equiv_e b$.[6]

[5] For some reason, starting with [8] the labels ADVP and PRT are considered to be identical as well

[6] We considered, but ultimately rejected, the possi-

| 1 | | 2 | 3 | | 4 | | 5 | | 6 | | 7 | | 8 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | E | | E | | E | | E | | | |
| | the | | , | | bagel | | with | | uh | | , | doughnut | |
| 1 | | 2 | 2 | | 4 | | 5 | | 2 | | 2 | | 8 |

Figure 1: Equivalent string positions as defined by $\equiv_e$

We give a concrete example in Figure 1. The first row indicates string position (as usual in parsing work, position indicators are between words). The second row gives the words of the sentence. Words that are edited out have an "E" above them. The third row indicates the equivalence relation by labeling each string position with the smallest such position with which it is equivalent.

There are two basic ideas behind this definition. First, we do not care where the EDITED nodes appear in the tree structure produced by the parser. Second, we are not interested in the fine structure of EDITED sections of the string, just the fact that they *are* EDITED. That we do care about which words are EDITED comes into our figure of merit in two ways. First, (non-contiguous) EDITED nodes remain, even thought their sub structure does not, and thus they are counted in the precision and recall numbers. Secondly (and probably more importantly), failure to decide on the correct positions of edited nodes can cause collateral damage to neighboring constituents by causing them to start or stop in the wrong place. This is particularly relevant because according to our definition, while the positions at the beginning and ending of an edit node are equivalent, the interior positions are not (unless related by the punctuation rule). See Figure EQIVFIG.

### 3.2 Parsing Experiments

The parser described in [3] was trained on the Switchboard training corpus as specified in section 2.1. The input to the training algorithm was the gold standard parses minus all EDITED nodes and their children.

We tested on the Switchboard testing subcorpus (again as specified in Section 2.1). All parsing results reported herein are from all sentences of length less than or equal to 100 words and punctuation. When parsing the test corpus we carried out the following operations:

1. create the simplified gold standard parse by removing non-terminal children of an EDITED node and merging consecutive EDITED nodes.
2. remove from the sentence to be fed to the parser all words marked as edited by an edit detector (see below).
3. parse the resulting sentence
4. add to the resulting parse EDITED nodes containing the non-terminal symbols removed in step 2. The nodes are added as high as possible (though the definition of equivalence from Section 3.1 should make the placement of this node largely irrelevant).
5. evaluate the parse from step 4 against the simplified gold standard parse from step 1.

We ran the parser in three experimental situations, each using a different edit detector in step 2. In the first of the experiments (labeled "Gold Edits") the "edit detector" was simply the simplified gold standard itself. This was to see how well the parser would do it if had perfect information about the edit locations.

In the second experiment (labeled "Gold Tags"), the edit detector was the one described in Section 2 trained on the part-of-speech tags as specified in the gold standard trees. Note that the parser was *not* given the gold standard part-of-speech tags.

In the third experiment ("Machine Tags") we used the edit detector based upon the machine generated tags. It is this result that we consider the "true" capability of the detector/parser combination. We were interested in contrasting the results of this experiment with that of the second experiment to gauge what improvement one could expect from using a more

bility of defining $\equiv_e$ using the EDITED nodes in the returned parse rather than the simplified gold standard. We rejected this because the $\equiv_e$ relation would then itself be dependent on the parser's output, a state of affairs that might allow complicated schemes to improve the parser's performance as measured by the metric.

sophisticated tagger as input to the edit detector.

The results of the experiments are given in Table 3 The last line in the figure indicates the performance of this parser when trained and tested on Wall Street Journal text.

### 3.3 Discussion

The general trends of Table 3 are much as one might expect. Parsing the Switchboard data is much easer given the correct positions of the EDITED nodes than without this information. The difference between the Gold tags and the Machine tags parses is small, as would be expected from the relatively small difference in the performance of the edit detector reported in Section 2. This suggests that putting significant effort into an tagger for use by the edit detector is unlikely to produce much improvement. Also, as one might expect, parsing conversational speech is harder than Wall Street Journal text, even given the gold-standard EDITED nodes.

Probably the only aspect of the above numbers that is likely to raise any comment in the parsing community is the degree to which precision numbers are lower than recall. With the exception of the single pair reported in [3] and repeated above, *no* precision values have ever been lower than recall vales in the recent statistical-parsing literature [2,3,5,6,10]. and even this one exception is only by 0.1%, and not statistically significant.

We attribute the dominance of recall over precision primarily to the influence of edit detector mistakes. First, note that when given the gold standard edits the difference is quite small (0.3%). When using the edit detector edits the difference increases to 1.2%. Our best guess is that because the edit detector has high precision, and lower recall, many more words are left in the sentence to be parsed, and thus one finds more non-terminal constituents in the machine parses than in the gold parses. Thus the precision is lower than the recall.

## 4 Previous Research

While parsing edited text has been neglected, there is a significant body of work on finding edit positions [1,7,9,12]. It is difficult to make meaningful comparisons between the various research efforts as they differ in (a) the corpora used for training and testing, (b) the information available to the edit detector, and (c) the evaluation metrics used. For example, [9] uses a sub-section of the ATIS corpus, takes as input the actual speech signal (and thus has access to silence duration, but not to words), and uses as its evaluation metric the percentage of time the program identifies the start of the interregnum (see Section 2.2). On the other hand, [7] uses an internally developed corpus of sentences, works from a transcript enhanced with information from the speech signal (and thus uses words), but does use a metric that seems to be similar to ours. One work that also uses the transcribed Switchboard corpus that we use is [12]. However they report no independent measures of edit detection accuracy, but rather report the degree to which identifying and removing disfluencies helps with word perplexity. (Discouragingly, it raised perplexity).

## 5 Conclusion

We have presented a simple architecture for parsing transcribed speech in which an edited word detector is first used to remove such words from the sentence string, and then a statistical parser trained on edited speech (with the edited nodes removed) is used to parse the text. The edit detector reduces the misclassification rate on edited words from the null-model (marking everything as not edited) rate of 5.9% to 2.2%. Due to differences in corpora and evaluation metrics, we have not attempted to compare this result to previous research. We would expect that it is very competitive.

To evaluate our parsing results we have introduced a new evaluation metric, relaxed edited labeled precision/recall. The purpose of this metric is to make evaluation of a parse tree relatively indifferent to the exact tree position of EDITED nodes, in much the same way that the previous metric, relax labeled precision/recall, made it indifferent to the attachment of punctuation. By this metric the parser achieved a 85.3% precision 86.5% and recall.

## References

| Experiment | Labeled Precision | Labeled Recall | F-measure |
|---|---|---|---|
| Gold Edits | 87.8 | 88.1 | 88.0 |
| Gold Tags | 85.4 | 86.6 | 86.0 |
| Machine Eags | 85.3 | 86.5 | 85.9 |
| WSJ | 89.5 | 89.6 | |

Table 3: Results of Switchboard parsing, sentence length $\leq 100$


1. Bear, J., Dowding, J. and Shriberg, E. *Integrating multiple knowledge sources for detection and correction of repairs in human-computer dialog.* In *Proceedings of the 3th Annual Meeting of the Association for Computational Linguistics*. 56–63.
2. Charniak, E. *Statistical parsing with a context-free grammar and word statistics.* In *Proceedings of the Fourteenth National Conference on Artificial Intelligence*. AAAI Press/MIT Press, Menlo Park, CA, 1997, 598–603.
3. Charniak, E. *A maximum-entropy-inspired parser.* In *Proceedings of the 2000 Conference of the North American Chapter of the Assocation for Computational Linguistics*. ACL, New Brunswick NJ, 2000.
4. Collins, M. *Discriminative reranking for natural language parsing* . In *Proceedings of the International Conference on Machine Learning(ICML 2000)*. 2000 .
5. Collins, M. J. *A new statistical parser based on bigram lexical dependencies.* In *Proceedings of the 34th Annual Meeting of the ACL*. 1996.
6. Collins, M. J. *Three generative lexicalised models for statistical parsing.* In *Proceedings of the 35th Annual Meeting of the ACL*. 1997, 16–23.
7. Heeman, P. A. and Allen, J. F. *Intonational boundaries, speech repairs and discourse markers: modeling spoken dialog.* In *35th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics*. 1997, 254–261.
8. Magerman, D. M. *Statistical decision-tree models for parsing.* In *Proceedings of the 33rd Annual Meeting of the Association for Computational Linguistics*. 1995, 276–283.
9. Nakatani, C. H. and Hirschberg, J. A corpus-based study of repair cues in spontaneous speech. *Journal of the Acoustical Society of America 953* (1994), 1603–1616.
10. Ratnaparkhi, A. Learning to parse natural language with maximum entropy models. *Machine Learning 34*1/2/3 (1999), 151–176.
11. Singer, Y. and Schapire, R. E. *Improved boosting algorithms using confidence-based predictions.* In *Proceedings of the Eleventh Annual Conference on Computational Learning Theory*. 1998, 80–91.
12. Stolcke, A. and Shriberg, E. *Automatic linguistic segmantation of conversational speech.* In *Proceedings of the 4th International Conference on Spoken Language Processing (ICSLP-96)*. 1996.