Disfluency Detection Using Multi-step Stacked Learning

Xian Qian and **Yang Liu**
Computer Science Department
The University of Texas at Dallas
{qx,yangl}@hlt.utdallas.edu

Abstract

In this paper, we propose a multi-step stacked learning model for disfluency detection. Our method incorporates refined n-gram features step by step from different word sequences. First, we detect filler words. Second, edited words are detected using n-gram features extracted from both the original text and filler filtered text. In the third step, additional n-gram features are extracted from edit removed texts together with our newly induced in-between features to improve edited word detection. We use Max-Margin Markov Networks (M^3Ns) as the classifier with the weighted hamming loss to balance precision and recall. Experiments on the Switchboard corpus show that the refined n-gram features from multiple steps and M^3Ns with weighted hamming loss can significantly improve the performance. Our method for disfluency detection achieves the best reported F-score 0.841 without the use of additional resources.[1]

1 Introduction

Detecting disfluencies in spontaneous speech can be used to clean up speech transcripts, which helps improve readability of the transcripts and make it easy for downstream language processing modules. There are two types of disfluencies: filler words including filled pauses (e.g., 'uh', 'um') and discourse markers (e.g., 'I mean', 'you know'), and edited words that are repeated, discarded, or corrected by the following words. An example is shown below that includes edited words and filler words.

I want a flight $\underbrace{\textit{to Boston}}_{\text{edited}}$ $\underbrace{\textit{uh I mean}}_{\text{filler}}$ to Denver

Automatic filler word detection is much more accurate than edit detection as they are often fixed phrases (e.g., "uh", "you know", "I mean"), hence our work focuses on edited word detection.

Many models have been evaluated for this task. Liu et al. (2006) used Conditional Random Fields (CRFs) for sentence boundary and edited word detection. They showed that CRFs significantly outperformed Maximum Entropy models and HMMs. Johnson and Charniak (2004) proposed a TAG-based noisy channel model which showed great improvement over boosting based classifier (Charniak and Johnson, 2001). Zwarts and Johnson (2011) extended this model using minimal expected F-loss oriented n-best reranking. They obtained the best reported F-score of 83.8% on the Switchboard corpus. Georgila (2009) presented a post-processing method during testing based on Integer Linear Programming (ILP) to incorporate local and global constraints.

From the view of features, in addition to textual information, prosodic features extracted from speech have been incorporated to detect edited words in some previous work (Kahn et al., 2005; Zhang et al., 2006; Liu et al., 2006). Zwarts and Johnson (2011) trained an extra language model on additional corpora, and used output log probabilities of language models as features in the reranking stage. They reported that the language model gained about absolute 3% F-score for edited word detection on the Switchboard development dataset.

[1] Our source code is available at http://code.google.com/p/disfluency-detection/downloads/list

Proceedings of NAACL-HLT 2013, pages 820–825,
Atlanta, Georgia, 9–14 June 2013.

In this paper, we propose a multi-step stacked learning approach for disfluency detection. In our method, we first perform filler word detection, then edited word detection. In every step, we generate new refined n-gram features based on the processed text (remove the detected filler or edited words from the previous step), and use these in the next step. We also include a new type of features, called in-between features, and incorporate them into the last step. For edited word detection, we use Max-Margin Markov Networks (M^{3}Ns) with weighted hamming loss as the classifier, as it can well balance the precision and recall to achieve high performance. On the commonly used Switchboard corpus, we demonstrate that our proposed method outperforms other state-of-the-art systems for edit disfluency detection.

2 Balancing Precision and Recall Using Weighted M^{3}Ns

We use a sequence labeling model for edit detection. Each word is assigned one of the five labels: *BE* (beginning of the multi-word edited region), *IE* (in the edited region), *EE* (end of the edited region), *SE* (single word edited region), *O* (other). For example, the previous sentence is represented as:

I/*O* want/*O* a/*O* flight/*O* to/*BE* Boston/*EE* uh/*O* I/*O* mean/*O* to/*O* Denver/*O*

We use the F-score as the evaluation metrics (Zwarts and Johnson, 2011; Johnson and Charniak, 2004), which is defined as the harmonic mean of the precision and recall of the edited words:

$$\begin{aligned} P &= \frac{\#\text{correctly predicted edited words}}{\#\text{predicted edited words}} \\ R &= \frac{\#\text{correctly predicted edited words}}{\#\text{gold standard edited words}} \\ F &= \frac{2 \times P \times R}{P + R} \end{aligned}$$

There are many methods to train the sequence model, such as CRFs (Lafferty et al., 2001), averaged structured perceptrons (Collins, 2002), structured SVM (Altun et al., 2003), online passive aggressive learning (Crammer et al., 2006). Previous work has shown that minimizing F-loss is more effective than minimizing log-loss (Zwarts and Johnson, 2011), because edited words are much fewer than normal words.

In this paper, we use Max-margin Markov Networks (Taskar et al., 2004) because our preliminary results showed that they outperform other classifiers, and using weighted hamming loss is simple in this approach (whereas for perceptron or CRFs, the modification of the objective function is not straightforward).

The learning task for M^{3}Ns can be represented as follows:

$$\begin{aligned} \min_{\alpha} \quad & \frac{1}{2} C \| \sum_{x,y} \alpha_{x,y} \Delta f(x,y) \|_2^2 + \sum_{x,y} \alpha_{x,y} L(x,y) \\ s.t. \quad & \sum_{y} \alpha_{x,y} = 1 \qquad \forall x \\ & \alpha_{x,y} \geq 0, \qquad \forall x, y \end{aligned}$$

The above shows the dual form for training M^{3}Ns, where x is the observation of a training sample, $y \in \mathcal{Y}$ is a label. α is the parameter needed to be optimized, $C > 0$ is the regularization parameter. $\Delta f(x,y)$ is the residual feature vector: $f(x,\tilde{y}) - f(x,y)$, where $\tilde{y}$ is the true label of x. $L(x,y)$ is the loss function. Taskar et al. (2004) used un-weighted hamming loss, which is the number of incorrect components: $L(x,y) = \sum_t \delta(y_t, \tilde{y}_t)$, where $\delta(a,b)$ is the binary indicator function (it is 0 if $a = b$). In our work, we use the weighted hamming loss:

$$L(x,y) = \sum_t v(y_t, \tilde{y}_t) \delta(y_t, \tilde{y}_t)$$

where $v(y_t, \tilde{y}_t)$ is the weighted loss for the error when $\tilde{y}_t$ is mislabeled as y_t. Such a weighted loss function allows us to balance the model's precision and recall rates. For example, if we assign a large value to $v(O, \cdot E)$ ($\cdot E$ denotes SE, BE, IE, EE), then the classifier is more sensitive to false negative errors (edited word misclassified as non-edited word), thus we can improve the recall rate. In our work, we tune the weight matrix v using the development dataset.

3 Multi-step Stacked Learning for Edit Disfluency Detection

Rather than just using the above M^{3}Ns with some features, in this paper we propose to use stacked learning to incorporate gradually refined n-gram features. Stacked learning is a meta-learning approach (Cohen and de Carvalho, 2005). Its idea is to use two

(or more) levels of predictors, where the outputs of the low level predictors are incorporated as features into the next level predictors. It has the advantage of incorporating non-local features as well as non-linear classifiers. In our task, we do not just use the classifier's output (a word is an edited word or not) as a feature, rather we use such output to remove the disfluencies and extract new n-gram features for the subsequent stacked classifiers. We use 10 fold cross validation to train the low level predictors. The following describes the three steps in our approach.

3.1 Step 1: Filler Word Detection

In the first step, we automatically detect filler words. Since filler words often occur immediately after edited words (before the corrected words), we expect that removing them will make rough copy detection easy. For example, in the previous example shown in Section 1, if "uh I mean" is removed, then the reparandum "to Boston" and repair "to Denver" will be adjacent and we can use word/POS based n-gram features to detect that disfluency. Otherwise, the classifier needs to skip possible filler words to find the rough copy of the reparandum.

For filler word detection, similar to edited word detection, we define 5 labels: BP, IP, EP, SP, O. We use un-weighted hamming loss to learn M^{3}Ns for this task. Since for filler word detection, our performance metric is not F-measure, but just the overall accuracy in order to generate cleaned text for subsequent n-gram features, we did not use the weighted hamming hoss for this. The features we used are listed in Table 1. All n-grams are extracted from the original text.

3.2 Step 2: Edited Word Detection

In the second step, edited words are detected using M^{3}Ns with the weighted-hamming loss. The features we used are listed in Table 2. All n-grams in the first step are also used here. Besides that, word n-grams, POS n-grams and logic n-grams extracted from filler word removed text are included. Feature templates $I(w_0, w'_i)$ is to generate features detecting rough copies separated by filler words.

3.3 Step 3: Refined Edited Word Detection

In this step, we use n-gram features extracted from the text after removing edit disfluencies based on the previous step. According to our analysis of the errors produced by step 2, we observed that many errors occurred at the boundaries of the disfluencies, and the word bigrams after removing the edited words are unnatural. The following is an example:

- **Ref:** *The new type is prettier than what their/SE they used to look like.*
- **Sys:** *The new type is prettier than what/BE their/EE they used to look like.*

Using the system's prediction, we would have bigram *than they*, which is odd. Usually, the pronoun following *than* is accusative case. We expect adding n-gram features derived from the cleaned-up sentences would allow the new classifier to fix such hypothesis. This kind of n-gram features is similar to the language models used in (Zwarts and Johnson,

unigrams	w_0, w_{-1}, w_1, w_{-2}, w_2 p_0, p_{-1}, p_1, p_{-2}, p_2, w_0p_0
bigrams	$w_{-1}w_0$, w_0w_1, $p_{-1}p_0$, p_0p_1
trigrams	$p_{-2}p_{-1}p_0$, $p_{-1}p_0p_1$, $p_0p_1p_2$
logic unigrams	$I(w_i, w_0)$, $I(p_i, p_0)$, $\quad -4 \leq i \leq 4$
logic bigrams	$I(w_{i-1}w_i, w_{-1}, w_0)$ $I(p_{i-1}p_i, p_{-1}p_0)$ $I(w_iw_{i+1}, w_0w_1)$ $I(p_ip_{i+1}, p_0p_1)$, $\quad -4 \leq i \leq 4$
transitions	$y_{-1}y_0$

Table 1: Feature templates for filler word detection. w_0, p_0 denote the current word and POS tag respectively. w_{-i} denotes the i^{th} word to the left, w_i denotes the i^{th} word to the right. The logic function $I(a, b)$ indicates whether a and b are identical (eigher unigrams or bigrams).

All templates in Table 1	
unigrams	w'_1, w'_2, w'_3, w'_4
bigrams	$p_0p'_1$, $p_0p'_2$, $p_0p'_3$, $p_0p'_4$ $w_0p'_1$, $w_0p'_2$, $w_0p'_3$, $w_0p'_4$ w_0p_1, w_0p_2, w_0p_3, w_0p_4
logic unigrams	$I(w_0, w'_i)$, $\quad 1 \leq i \leq 4$
transitions	$p_0y_{-1}y_0$

Table 2: Feature templates for edit detection (step 2). w'_i, p'_i denote the i^{th} word/POS tag to the right in the filler words removed text. If current word w_0 is removed in step 1, we use its original n-gram features rather than the refined n-gram features.

2011). They have the benefit of measuring the fluency of the cleaned text.

Another common error we noticed is caused by the ambiguities of coordinates, because the coordinates have similar patterns as rough copies. For example,

- **Coordinates:** *they ca n′t decide which are the good aspects and which are the bad aspects*
- **Rough Copies:** *it/BE ′s/IE a/IE pleasure/IE to/EE it s good to get outside*

To distinguish the rough copies and the coordinate examples shown above, we analyze the training data statistically. We extract all the pieces lying between identical word bigrams *AB* ... *AB*. The observation is that coordinates are often longer than edited sequences. Hence we introduce the in-between features for each word. If a word lies between identical word bigrams, then its in-between feature is the log length of the subsequence lying between the two bigrams; otherwise, it is zero (we use log length to avoid sparsity). We also used other patterns such as *A* ... *A* and *ABC* ... *ABC*, but they are too noisy or infrequent and do not yield much performance gain.

Table 3 lists the feature templates used in this last step.

All templates in Table 1, Table 2	
word n-grams	w_1'', $w_0 w_1''$
in-between	L_{AB}, $w_0 b_{AB}$, b_{AB}

Table 3: Feature templates for refined edit detection (step 3). w_i'' denotes the i^{th} word tag to the right in the edited word removed text. L_{AB} denotes the log length of the sub-sequence in the pattern *AB*...*AB*, b_{AB} indicates whether the current word lies between two identical bigrams.

4 Experiments

4.1 Experimental Setup

We use the Switchboard corpus in our experiment, with the same train/develop/test split as the previous work (Johnson and Charniak, 2004). We also remove the partial words and punctuation from the training and test data for the reason to simulate the situation when speech recognizers are used and such kind of information is not available (Johnson and Charniak, 2004).

We tuned the weight matrix for hamming loss on the development dataset using simple grid search. The diagonal elements are fixed at 0; for false positive errors, $O \rightarrow \cdot E$ (non-edited word mis-labeled as edited word), their weights are fixed at 1; for false negative errors, $\cdot E \rightarrow O$, we tried the weight from 1 to 3, and increased the weight 0.5 each time. The optimal weight matrix is shown in Table 4. Note that we use five labels in the sequence labeling task; however, for edited word detection evaluation, it is only a binary task, that is, all of the words labeled with $\cdot E$ will be mapped to the class of edited words.

truth \ predict	*BE*	*IE*	*EE*	*SE*	*O*
BE	0	1	1	1	2
IE	1	0	1	1	2
EE	1	1	0	1	2
SE	1	1	1	0	2
O	1	1	1	1	0

Table 4: Weighted hamming loss for M^{3}Ns.

4.2 Results

We compare several sequence labeling models: CRFs, structured averaged perceptron (AP), M^{3}Ns with un-weighted/weighted loss, and online passive-aggressive (PA) learning. For each model, we tuned the parameters on the development data: Gaussian prior for CRFs is 1.0, iteration number for AP is 10, iteration number and regularization penalty for PA are 10 and 1. For M^{3}Ns, we use Structured Sequential Minimal Optimization (Taskar, 2004) for model training. Regularization penalty is $C = 0.1$ and iteration number is 30.

Table 5 shows the results using different models and features. The baseline models use only the n-grams features extracted from the original text. We can see that M^{3}Ns with the weighted hamming loss achieve the best performance, outperforming all the other models. Regarding the features, the gradually added n-gram features have consistent improvement for all models. Using the weighted hamming loss in M^{3}Ns, we observe a gain of 2.2% after deleting filler words, and 1.8% after deleting edited words. In our analysis, we also noticed that the in-between fea-

	CRF	AP	PA	M^3N	w. M^3N
Baseline	78.8	79.0	78.9	79.4	80.1
Step 2	81.0	81.1	81.1	81.5	82.3
Step 3	82.9	83.0	82.8	83.3	**84.1**

Table 5: Effect of training strategy and recovered features for stacked learning. F scores are reported. AP = Averaged Perceptron, PA = online Passive Aggresive, M^3N = un-weighted M^3Ns, w. M^3N = weighted M^3Ns.

tures yield about 1% improvement in F-score for all models (the gain of step 3 over step 2 is because of the in-between features and the new n-gram features extracted from the text after removing previously detected edited words). We performed McNemar's test to evaluate the significance of the difference among various methods, and found that when using the same features, weighted M^3Ns significantly outperforms all the other models (p value < 0.001). There are no significant differences among CRFs, AP and PA. Using recovered n-gram features and in-between features significantly improves all sequence labeling models (p value < 0.001).

We also list the state-of-the-art systems evaluated on the same dataset, as shown in Table 6. We achieved the best F-score. The most competitive system is (Zwarts and Johnson, 2011), which uses extra resources to train language models.

System	F score
(Johnson and Charniak, 2004)	79.7
(Kahn et al., 2005)	78.2
(Zhang et al., 2006)†	81.2
(Georgila, 2009)*	80.1
(Zwarts and Johnson, 2011)$^{+}$	83.8
This paper	**84.1**

Table 6: Comparison with other systems. † they used the re-segmented Switchboard corpus, which is not exactly the same as ours. * they reported the F-score of *BE* tag (beginning of the edited sequences). $^{+}$ they used language model learned from 3 additional corpora.

5 Conclusion

In this paper, we proposed multi-step stacked learning to extract n-gram features step by step. The first level removes the filler words providing new ngrams for the second level to remove edited words. The third level uses the n-grams from the original text and the cleaned text generated by the previous two steps for accurate edit detection. To minimize the F-loss approximately, we modified the hamming loss in M^3Ns. Experimental results show that our method is effective, and achieved the best reported performance on the Switchboard corpus without the use of any additional resources.

Acknowledgments

We thank three anonymous reviewers for their valuable comments. This work is partly supported by DARPA under Contract No. HR0011-12-C-0016 and FA8750-13-2-0041. Any opinions expressed in this material are those of the authors and do not necessarily reflect the views of DARPA.

References

Yasemin Altun, Ioannis Tsochantaridis, and Thomas Hofmann. 2003. Hidden markov support vector machines. In *Proc. of ICML*.

Eugene Charniak and Mark Johnson. 2001. Edit detection and parsing for transcribed speech. In *Proc. of NAACL*.

William W. Cohen and Vitor Rocha de Carvalho. 2005. Stacked sequential learning. In *Proc. of IJCAI*.

Michael Collins. 2002. Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In *Proc. of EMNLP*.

Koby Crammer, Ofer Dekel, Joseph Keshet, Shai Shalev-Shwartz, Yoram Singer, and Yoram Singer. 2006. Online passive-aggressive algorithms. *Journal of Machine Learning Research*.

Kallirroi Georgila. 2009. Using integer linear programming for detecting speech disfluencies. In *Proc. of NAACL*.

Mark Johnson and Eugene Charniak. 2004. A TAG-based noisy-channel model of speech repairs. In *Proc. of ACL*.

Jeremy G. Kahn, Matthew Lease, Eugene Charniak, Mark Johnson, and Mari Ostendorf. 2005. Effective use of prosody in parsing conversational speech. In *Proc. of HLT-EMNLP*.

John D. Lafferty, Andrew McCallum, and Fernando C. N. Pereira. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proc. of ICML*.

Yang Liu, E. Shriberg, A. Stolcke, D. Hillard, M. Ostendorf, and M. Harper. 2006. Enriching speech recognition with automatic detection of sentence bound-

aries and disfluencies. *IEEE Transactions on Audio, Speech, and Language Processing*, 14(5).

Ben Taskar, Carlos Guestrin, and Daphne Koller. 2004. Max-margin markov networks. In *Proc. of NIPS*.

Ben Taskar. 2004. *Learning Structured Prediction Models: A Large Margin Approach*. Ph.D. thesis, Stanford University.

Qi Zhang, Fuliang Weng, and Zhe Feng. 2006. A progressive feature selection algorithm for ultra large feature spaces. In *Proc. of ACL*.

Simon Zwarts and Mark Johnson. 2011. The impact of language models and loss functions on repair disfluency detection. In *Proc. of ACL-HLT*.