Real-time Chunking of Incomplete Sentences

Duy Nguyen
Department of Computer Science
Oregon State University
Corvallis, OR 97331
nguyend6@oregonstate.edu

Ga Wu
Department of Computer Science
Oregon State University
Corvallis, OR 97331
wug2@oregonstate.edu

Christopher Eriksen
Department of Mechanical, Industrial, and Manufacturing Engineering
Oregon State University
Corvallis, OR 97331
eriksenc@oregonstate.edu

Abstract

Patients diagnosed with ALS often need to use a speech assistance device (AAC) to communicate. However, long pauses produced as the AAC waits for all user input can cause listeners to get distracted. In this paper, we evaluate techniques for chunking, or determining comprehensible and sensible phrases within incomplete sentences such that the AAC can output phrases before all user input is received. We train HMM, SVM, and CRF chunking models on training sets of both complete and incomplete sentences and find that chunking performance on incomplete sentences improves when models are trained on incomplete sentences. Specifically, we find that F_1 performance increases on average by 3.86 percentage points for the HMM, 0.34 percentage points for the SVM, and 0.81 percentage points for the CRF. Additionally, we find that the SVM implementation slightly outperforms the CRF model, which greatly outperforms the HMM model.

1 Introduction

According to the ALS (Amyotrophic lateral sclerosis) Association[1], there are roughly 6,400 people diagnosed with ALS each year in the United States. These patients eventually lose their ability to speak and many of them have to use some type of AAC (Augmentative Alternative Communication) device to communicate. With the rapid growth of technology, there has been great progress in creating better AAC devices. However, these devices still retain some significant drawbacks. For instance, when the AAC users type their responses on the device, they unintentionally produce long and awkward pauses. These pauses can cause their conversational partners to lose attention or interest in the conversation.

In order to combat this problem, we first assume that there is a general time instance at which listeners will lose attention amidst a long pause during a conversation (this time is currently being investigated in a concurrent psychology study). For our solution, we propose that the AAC device vocalize some portion of the user's written response at the time when listeners will start to lose attention. However, we want the spoken units to be comprehensible and sensible. To achieve this, we investigate methods for "chunking" sentences into understandable chunks or phrases. Since the sentences will not be fully constructed when the AAC device needs to vocalize understandable chunks, we specifically investigate methods for chunking incomplete sentences.

There exist many mature methods for chunking complete sentences, such as the work proposed by Abney[2] and Ramshaw and Marcus[3]. Sha and Pereira[4] note that most previous approaches

use generative probabilistic models such as Hidden Markov Models (HMM) or treat chunking as a series of classification problems which can be solved by discriminative classifiers. In their paper, the authors report competitive performance with previous approaches using a Conditional Random Field (CRF) chunking model. The advantage of using a CRF over a basic HMM is that it takes the entire observation sequence into consideration when making predictions at each time step instead of looking at a single observation.

In this project, we investigate how well previous chunking approaches to complete sentence chunking translate to the incomplete sentence domain. Following the pattern set by Sha and Pereira[4], we consider 1.) a HMM implementation to represent classical, generative probabilistic models, 2.) a Support Vector Machine (SVM) implementation to represent the best of classical, discriminative classifiers, and 3.) a CRF implementation to represent more recent approaches as proposed by Sha and Pereira. While chunking prediction needs to be performed in real-time for our purposes, we note that the models can be pre-trained. While model training often takes a considerable amount of time for the approaches we consider, prediction is performed significantly faster, and the prediction time for a single example as found in our context is negligible. Additionally, we note that the chunking task often involves a pre-processing step of tagging the sentence tokens with part of speech (POS) markers. Since POS tagging is an easier task with many robust existing implementations, we assume POS tagging has already been performed in our experiments (in fact, our training data includes the tokens' POS tags).

This document is structured as follows. Section 2 presents our general framework and contains background information on the chunking models we consider. In Section 3, we briefly describe the dataset we use and discuss our experimental approach. Results and discussion is presented in Section 4. Finally, we end with some conclusions and considerations for future work in Section 5.

2 Methodology

In this section, we present our approach for investigating the efficacy of popular chunking models in the incomplete sentence domain. Specifically, we test whether chunking performance on incomplete sentences improves when the proposed models are trained on incomplete rather than complete sentences. We also compare the performance from the different models to see which performs best for incomplete sentences. For our models, we test the classical approaches of using a generative probabilistic model or a discriminative classifier as well as a more modern approach of using a CRF. To represent classical approaches, we use a HMM chunker implementation found in the LingPipe toolkit[5] and a SVM chunker implementation presented by Kudo and Matsumoto[6]. For the more modern approach, we use a CRF chunker implementation found in the OpenNLP toolkit[7].

In the following subsections, we first propose our mathematical formalization for measuring chunking performance on incomplete sentences. Subsequently, we describe how the HMM, SVM, and CRF models are adapted to the chunking domain. The details of our experimental evaluation are presented in Section 3.

2.1 Formalization

In this subsection, we formalize the learning model for our chunking task. Generally, we try to maximize the following likelihood function:

$$\arg\max_{ChunkingSequence} P(ChunkingSequence|ObservedSentence) \tag{1}$$

However, this objective must be adapted to fit the model for each approach that we investigate.

Classical classification algorithms can only output a single classification label. Therefore, direct inference and sequential prediction is difficult. We modify the objective as shown below:

$$\arg\max_{y} \prod_{y=1}^{T} P(y_i|\phi(\mathbf{x})) \tag{2}$$

Note that the best prediction sequence is not equivalent to the product of best predicted labels, y_i. Such an approach is limited as compared to a structured prediction approach.

Since HMM's support exact inference of the joint likelihood, we use the joint likelihood as our objective. The joint likelihood is proportional to the conditional likelihood as shown in equation 3.

$$\arg\max_{ChunkingSequence} P(ChunkingSequence, ObservedSentence) \tag{3}$$

Due to increased complexity in the model, CRF's on the otherhand are able to maximize the likelihood function found in equation 1 directly.

2.2 SVM

We use a classical Support Vector Machine (SVM) model adapted for the chunking context, as proposed by Kudoh and Matsumoto[6]. The model was introduced for the CoNLL 2000 Shared Task on Chunking[8].

Generally, the goal of a SVM is to maximize the margin of support vectors close to the decision boundary. The input format for a SVM is a vector and the output is a single label. In this implementation, chunk classification for each token is performed sequentially, and the input features for each classification include local word tokens, POS tags, and previous classifications provided by the model. The input vector is defined as follows:

$$\mathbf{x} = \begin{bmatrix} x_1 : \text{token: } -2 \\ x_2 : \text{token: } -1 \\ \vdots \\ x_5 : \text{token: } +2 \\ x_6 : \text{pos: } -2 \\ x_7 : \text{pos: } -1 \\ \vdots \\ x_{10} : \text{pos: } +2 \\ x_{11} : \text{chunk: } +1 \end{bmatrix} \tag{4}$$

The SVM implementation provided by the authors is an open-source package named Yet Another Multipurpose CHunk Annotator (YamCha). We note that this implementation can be trained with any polynomial kernel and can use either pairwise comparison or a one verses all approach for its multi-class classification. This model also performed best in the CoNLL 2000 Shared Task.

2.3 HMM

One argument for using a structured model is that the most likely sequence of predictions is not equivalent to the best set of predictions for each latent variable. Classical approaches ignore the correlation between latent variables.

A Hidden Markov Model (HMM) is a generative structured graphic model, which, similarly to Naive Bayes, predicts the joint probability of observations and a latent sequence. One serious limitation of this model is its strong assumption of conditional independence amongst the observations, which is often not true.

Despite this, HMM model is often used to perform sequential prediction because it is easy to implement. Training a HMM is straightforward using the Expectation Maximization algorithm, and model inference is similarly easy using the Viterbi algorithm, or more generally the max-product algorithm.

The joint likelihood of a HMM can be expressed as follows:

$$P(O, S) = P(S_1) \prod_{i=2}^{T} P(S_t|S_{t-1}) \prod_{j=1}^{T} P(O_t|S_t)$$

The HMM chunking task is described with two line variables as shown in Figure 1. The observations are POS tags and the latent variables are chunking tags. Here, in Figure 1, to simplify, each latent

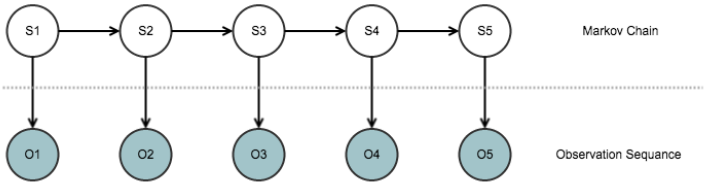

Figure 1: HMM MODEL

variable has only one observation. However, it is possible to have multiple observations for single latent variable.

Training this model is normally achieved by using the EM algorithm. First we need to guess an initial parameter sequence that determines the transition probability and fix it to maximize latent variables. Then, we fix learned latent variables to optimize parameters.

The flexibility of the HMM model is strictly limited by Markov assumptions, which means it is unable to take more than one observation into consideration at each time step. It is thus more of a baseline algorithm for our work.

For our analysis, we use a HMM chunking implementation provided in the LingPipe Java Natural Language Processing package[5] .

2.4 CRF

In the Natural Language Processing field, Conditional Random Fields (CRF) have become more common in tackling POS tagging and chunking problems. The relation between a CRF and HMM is similar to that of Logistic Regression and a Naive Bayes Classifier.

A CRF is a deterministic model which tries to maximize the conditional probability of $P(Y|X)$ directly rather than assuming the probability of observations.

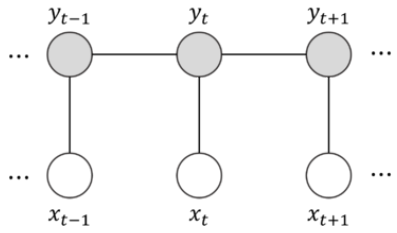

Figure 2: CRF MODEL

Factorization of an undirected graphical model is more focused on compatibility among variables. Therefore, the complexity of the CRF model depends on the number of variables considered.

$$P(\mathbf{y}, \mathbf{x}) = \frac{1}{Z} \prod_{t=1}^{T} \exp\{\sum_{k=1}^{K} \theta_k f_k(y_t, y_{t-1}, x_t)\} \tag{5}$$

Computing the partition function of the CRF is difficult, which makes using the undirected graphical model to compute the joint likelihood intractable. However, computing the conditional probability $P(Y|X)$ will directly eliminate the partition function.

$$
\begin{aligned}
P(\mathbf{y}|\mathbf{x}) &= \frac{P(\mathbf{y},\mathbf{x})}{\sum_{y'} P(\mathbf{y}',\mathbf{x})} \\
&= \frac{\prod_{t=1}^{T} \exp\{\sum_{k=1}^{K} \theta_k f_k(y_t, y_{t-1}, x_t)\}}{\sum_{y'} \prod_{t=1}^{T} \exp\{\sum_{k=1}^{K} \theta_k f_k(y'_t, y'_{t-1}, x_t)\}}
\end{aligned} \tag{6}
$$

In this project, we use a Java implementation of a CRF chunker provided in the OpenNLP toolset[7]. The package provides code for training and testing a CRF chunker.

3 Experiment

3.1 Data

We decide to use data provided by the CoNLL 2000 Shared Task on Chunking[8]. This task provides a standard metric for assessing the performance of chunkers on complete sentences. This data is furthermore useful since the YamCha and OpenNLP implementations we use are constructed to accept data following its format. The task provides a corpus of 211,727 token-POS-chunk instances used for training and a corpus of 47,377 instances for testing.

Table 1: Data Sample

Confidence	NN	B-NP
in	IN	B-PP
the	DT	B-NP
pound	NN	I-NP
is	VBZ	B-VP
widely	RB	I-VP
expected	VBN	I-VP
to	TO	I-VP
take	VB	I-VP
another	DT	B-NP
sharp	JJ	I-NP

The structure for CoNLL 2000 data is shown in 1. The data is contained in a single file and includes three columns: word token, POS tag, and chunk tag. Chunk tags begin with either a "B" or "I", specifying whether the given token begins a new chunk or continues an existing chunk. The prefix is followed by the chunk identifier. A separate "O" tag is used to identify punctuation. For our purposes, we say that a word is chunked correctly if it is correctly assigned the correct tag (prefix and chunk identifier) as found in the test set.

We use the CoNLL training and test sets as bases for generating datasets with incomplete sentences. Specifically, we bootstrap examples from the original corpus and randomly select a word in the sentence to split on. The first half of the sentence is then treated as an incomplete sentence example. We take this bootstrapping approach to randomly generate incomplete sentences from the original data. In order to construct similar training sets for complete and incomplete sentences, for every bootstrapped example, we save the unaltered version of the example in a complete sentence training set and the randomly split version in an incomplete sentence training set. We construct a testing set of incomplete sentences in a similar manner. We choose to bootstrap a number of examples equal to a chosen multiplier multiplied by the size of the original data set.

3.2 Approach

To test the performance of our proposed models on chunking incomplete sentences, we first generate two separate training sets containing complete or incomplete sentences as described in the previous subsection. For each of the proposed models, we train a separate version on each of the training sets. To assess the robustness of each model to the size of the training set, we use a variety of bootstrap multipliers for training.

The test set was generated by bootstrapping the CoNLL 2000 Shared Task test data with a bootstrap multiplier of 5, splitting each example at a random index to produce incomplete sentences. This produced a test set of 139,084 token-POS-chunk instances in a similar format to that of the CoNLL 2000 shared task, but with incomplete rather than complete sentences. We chose a bootstrap multiplier of 5 since it was high enough such that we are expected to sample almost all (99%) of the examples at least once.

3.3 Parameter Tuning

While the HMM and CRF chunker tools did not have tunable parameters, the SVM implementation had the option of specifying the polynomial order of the kernel, C value for slack weighting, and multi-class classification strategy (pairwise or one versus rest). To perform this tuning, we split off 80% of the original CoNLL training data as training data and the other 20% as validation data for testing the effects of parameter variation. From each set, we then bootstrapped training examples using a bootstrap multiplier of 5. After determining the best parameters by comparing performance on the bootstrapped validation set, we retrained the SVM model using the optimal parameters on data bootstrapped from the whole CoNLL training dataset for our model comparison evaluation. We note that we do not perform cross-validation due to time constraints and the large time cost of training the SVM models.

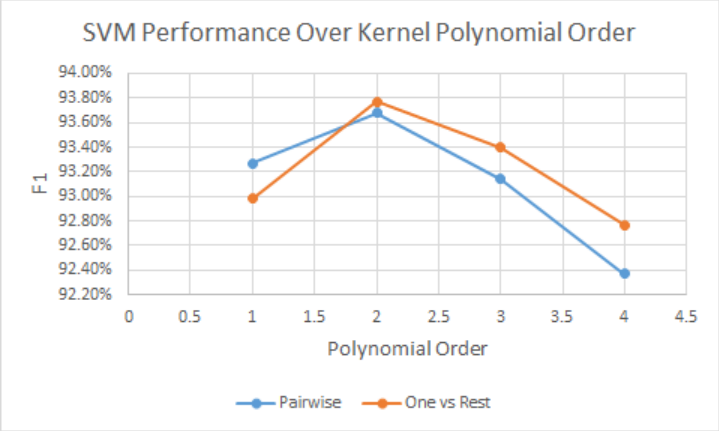

Figure 3: F_1 performance of SVM when trained with a polynomial kernel of various orders. Performance using both pairwise and one verses rest multi-class classification strategy is shown.

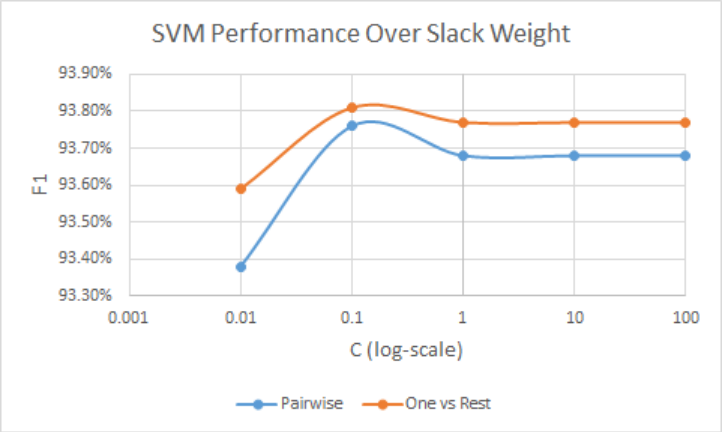

Figure 4: F_1 performance of SVM when trained with different C values. Performance using both pairwise and one verses rest multi-class classification strategy is shown.

Figures 3 and 4 show the results of performing parameter tuning on the SVM model. Due to the large training time overhead, we used a greedy approach by first tuning the polynomial order and then tuning the C value using the optimal polynomial parameter value. We found that using a polynomial order of 2 and a C value of 0.1 was optimal for both multi-class classification strategies. Furthermore, we chose to use the one versus rest approach since it performed optimally in the best case.

4 Results and Analysis

Figure 5 shows the test performance of each of the proposed models after training on either complete or incomplete sentences. To evaluate the robustness of each model to the size of the training data, we vary the bootstrap multiplier used to generate the training data. We choose to represent performance using the F_1 score, a standard metric in natural language processing for combining the information provided by precision and recall. For each model, we see that the version trained on incomplete sentences performs better than the version trained on complete sentences in almost all cases. This breaks down slightly for the HMM model, where the version trained on complete sentences performs better for smaller amounts of data. Still, on average, we see a 0.34 percentage point increase in performance for the SVM, 3.86 percentage point increase for the HMM, and 0.81 percentage point increase for the CRF when the model is trained on incomplete rather than complete sentences. This suggests that training a chunking model on incomplete sentences will lead to increased performance in chunking incomplete sentences over a model trained on complete sentences.

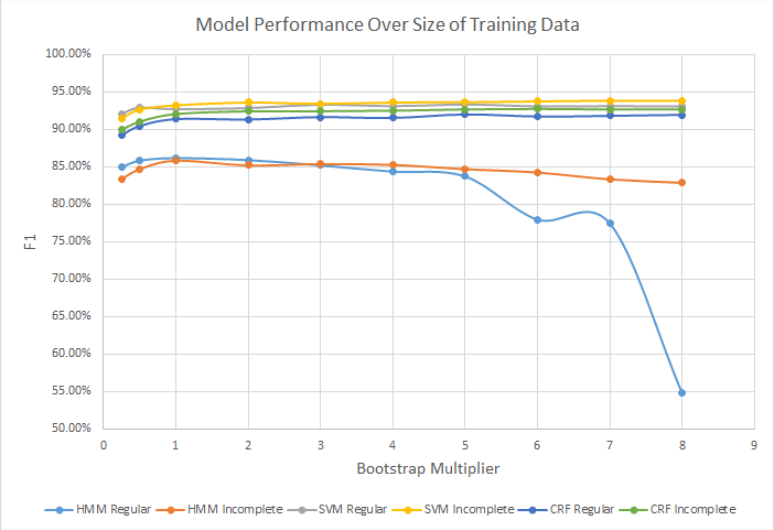

Figure 5: F_1 performance on incomplete sentences for HMM, SVM, and CRF models. A version of each model was trained on complete sentences and a separate version was trained on incomplete sentences.

We also see that both versions of the SVM model perform better than the CRF models, which in turn perform significantly better than the HMM models. While it is surprising that the SVM model outperforms the CRF model, we note that this particular implementation of a SVM chunker was developed specifically for the CoNLL 2000 shared task (it in fact won the competition), and thus was probably tuned specifically for this data set. We also note Sha and Perier mention in their paper that their CRF chunker did not outperform the YamCha implementation on complete sentences[4]. In any case, we note that both the SVM and CRF significantly outperform the HMM models, suggesting that these more advanced models are preferred for the chunking task. We also note that HMM performance (especially when trained on complete sentences) drops off with too much data. This might be because of the fact that the HMM assumes conditional independence between the features, which breaks down with too much data (particularly since all examples are generated from the same original data set).

5 Conclusions

In this project, we examined how three different approaches (namely SVM, HMM, and CRF) of complete sentence chunking translated to the incomplete sentence domain. The experimental results suggest that training on incomplete rather than complete sentences leads to improved performance for all models. We also find that the CRF and SVM implementations have comparable performance, with the SVM chunker performing slightly better. Both models, however, significantly outperform the HMM chunker.

We do, however, note a number of limitations in our approach. First, the SVM tool that we used was tailored specifically for our data set and, due to time constraints and the large time cost of model training, we used only that data set for our experiments. This might have led to unfair bias in favor of the SVM model. In any case, testing on multiple datasets or even generating multiple training sets from the source text would lead to an increase in the validity of our results. Second, for each method, we only used one implementation in our experiments. There might be other SVM, HMM, or CRF tools that outperform our models, but we only considered available open-source implementations. Additionally, we note that advanced structured prediction models such as a structured SVM (which had been used to great effect for tasks like parsing) were not considered in our experiments. Adapting such advanced models could lead to improved performance in the incomplete chunking domain.

For future work, this project will serve as a baseline for our development of an improved real-time chunking program for incomplete sentences. We plan to apply further testing using new tools, datasets, and models and hope to develop novel extensions that are better suited for our task. Additionally, once we determine the time that people tend to lose attention from the concurrent psychological research, we can apply the insights gained from our experiments toward constructing better AAC devices.

References

[1] The ALS Association. Facts you should know.

[2] Steven P Abney. *Parsing by chunks*. Springer, 1992.

[3] Lance A Ramshaw and Mitchell P Marcus. Text chunking using transformation-based learning. *arXiv preprint cmp-lg/9505040*, 1995.

[4] Fei Sha and Fernando Pereira. Shallow parsing with conditional random fields. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology-Volume 1*, pages 134–141. Association for Computational Linguistics, 2003.

[5] Breck Baldwin and Bob Carpenter. Lingpipe. *Available from World Wide Web: http://alias-i. com/lingpipe*, 2003.

[6] Taku Kudoh and Yuji Matsumoto. Use of support vector learning for chunk identification. In *Proceedings of the 2nd workshop on Learning language in logic and the 4th conference on Computational natural language learning-Volume 7*, pages 142–144. Association for Computational Linguistics, 2000.

[7] Jason Baldridge. The opennlp project. *URL: http://opennlp. apache. org/index. html,(accessed 2 February 2012)*, 2005.

[8] Erik F Tjong Kim Sang and Sabine Buchholz. Introduction to the conll-2000 shared task: Chunking. In *Proceedings of the 2nd workshop on Learning language in logic and the 4th conference on Computational natural language learning-Volume 7*, pages 127–132. Association for Computational Linguistics, 2000.