Effective Use of Cross-Domain Parsing in Automatic Speech Recognition and Error Detection

Marius Alexandru Marin

A dissertation submitted in partial fulfillment of the
requirements for the degree of

Doctor of Philosophy

University of Washington

2015

Reading Committee:

Mari Ostendorf, Chair

Luke Zettlemoyer

Hannaneh Hajishirzi

Program Authorized to Offer Degree:
Electrical Engineering

University of Washington

Abstract

Effective Use of Cross-Domain Parsing in Automatic
Speech Recognition and Error Detection

Marius Alexandru Marin

Chair of the Supervisory Committee:
Professor Mari Ostendorf
Electrical Engineering

Automatic speech recognition (ASR), the transcription of human speech into text form, is used in many settings in our society, ranging from customer service applications to personal assistants on mobile devices. In all such settings it is important for the system to know when it is making errors, so that it may ask the user to rephrase or restate their previous utterance. Such errors are often syntactically anomalous. The primary goal of this thesis is to find novel uses of parsing for automatic detection and correction of ASR errors.

We start by developing a framework for ASR rescoring and automatic error detection leveraging syntactic parsing in conjunction with a maximum entropy classifier, and find that parsing helps with error detection, even when the parser is trained on out-of-domain data. In particular, features capturing parser reliability are used to improve the detection of out-of-vocabulary (OOV) and name errors. However, parsers trained on out-of-domain treebanks do not provide any benefit to ASR rescoring.

This observation motivates our work on domain adaptation of parsing, with the objective of directly improving both transcription accuracy and error detection. We develop two weakly supervised domain adaptation methods which use error labels, but no hand-annotated parses: a self-training approach to directly improve the probabilistic context-free grammar (PCFG) model used in parsing, as well as a novel model combination method using a discriminative log-linear model to augment the generative PCFG. We apply both

methods to ASR rescoring and error detection tasks. We find that self-training improves the ability of our parser to select the correct ASR hypothesis. The log-linear adaptation improves both OOV and name error detection tasks, and self-training performed after log-linear adaptation further improves the reliability of the parser, while producing smaller, faster models.

Finally, motivated by empirical observations that the presence of names in an utterance is often indicated by words located far apart from the names themselves, we develop a general long-distance phrase pattern learning algorithm using word-level semantic similarity measures, and apply it to the problem of name error detection. This novel feature learning method leads to more robust classification, both when used independently of parsing, and in conjunction with parse features.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

Many people have contributed to my success during the long journey towards a Ph.D. While I hope to have acknowledged everyone, my apologies to everyone I may have inadvertently left out.

First and foremost, I would like to thank my advisor, Mari Ostendorf. Her guidance and support during my time at the University of Washington have been irreplaceable. I have learned from Mari more things than I can count, and likely already forgot much more than I ever should, but a few lessons I hope I will never lose. In particular, her integrity, dedication, and relentless work ethic are qualities that I will always aspire to match in my own career.

I would also like to thank Maya Gupta, who was my sole advisor during my first quarter as a graduate student, and then became my co-advisor for the remainder of her career at UW. Her technical expertise and enthusiasm were invaluable while I was learning to become a researcher.

Many thanks are due to the rest of my supervisory committee. Luke Zettlemoyer was involved as a co-author in the early stages of the system presented in this thesis, and provided invaluable feedback to my research throughout my time at UW. I am extremely grateful to Emily Bender, who took the time to provide careful advice on my dissertation despite not being one of the official readers. Hanna Hajishirzi and Eve Riskin gracefully agreed to sit on my committee at the last minute; for this I am very thankful as well. Meliha Yetizgen took the time to provide invaluable advice on future career paths.

Many faculty members of the EE department have contributed to my academic career, though I would like to single out Radha Poovendran, who acted as a mentor throughout my time here; I appreciate all of his advice over the years, and am thankful for the opportunity to have served as his teaching assistant and informal co-instructor during one quarter.

I will always value my time in the SSLI lab, and later its sub-lab, TIAL. I have learned a lot from my past and present labmates. Chief among them is Jon Malkin, who was my student mentor during our shared time in grad school; even after he graduated, he has been my main sounding board when I needed advice about research or system design. Jeremy Kahn and Dustin Hillard provided useful thoughts on research ideas, and conversation partners about all things NLP, during my first few years. I am particularly grateful to the rest of my cohort - Amittai Axelrod, Sangyun Hahn, Ji He, Brian Hutchinson, Yuzong Liu, Anna Margolis, Julie Medero, Nicole Nichols, Wei Wu, and Bin Zhang, with whom I spent many long days (and sleepless nights) working on projects or preparing for system evaluations. Amittai and Nicole shared my love of pineapple cakes; Julie and Nicole, my interest in obscure teas; Brian, my love for vinyl records; Amittai, my too-dark humor. They all kept me sane during the tough times. I am also grateful to my newer labmates – Hao Cheng, Hao Fang, Yi-Chin Huang, Aaron Jaech, Kevin Lybarger, Yi Luan, and Victoria Zayats, for invaluable feedback on my dissertation and presentations.

Special thanks are due to our lab's system administrator, Lee Damon, whose tireless efforts have kept our network stable and allowed me to complete my experiments, sometimes despite my own best efforts to the contrary, and to the EE graduate advisors, Brenda Larson and Bryan Crockett, who have helped me navigate the treacherous world of academic paperwork. I want to give special thanks also to Stephen Graham, who always has the answers to any and all questions, however obscure, about academics, the department, or the university as a whole.

Much of the work in my dissertation was developed as part of the DARPA BOLT project. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of DARPA. I would like to thank my collaborators at SRI International, Columbia University, and Aix-Marseille University for all the help and feedback. In particular, Necip Fazil Ayan, Arindam Mandal, Colleen Richey, Yik-Cheung Tam, and Wen Wang from SRI have graciously provided much of the training data used in my experiments. Thanks are also due to my mentor at Microsoft,

Roman Holenstein, and our manager there, Ruhi Sarikaya, who allowed me to use some of the algorithms developed during my internship as part of my dissertation work.

At long last, I would like to thank my entire family. My mother, Mihaela, believes in me whenever I lose hope, and always supports me, no matter the cost to herself. My late step-father, Robert, always believed I would become an engineer despite my attempts to convince him that *Mathematics* was my true calling – he proved me right, as he so often did. My grandmother, Dumitra, taught me to treasure books and seek knowledge. My father, Marian, and my grandmother, Elena, whose love from afar supports me every day. My extended family here in Washington, Scott and Carol Bowen, opened their lives to me, and treat me as their second son. Finally, Jayson, who stood by me all through my graduate studies and encouraged me to keep going when I felt like giving up, to find the way forward when it seemed impossible – I could never have done it without him.

DEDICATION

This thesis is dedicated to the memory of my grandfather, Alexandru Gh. Savu, who instilled in me the love of learning and of telling stories. He will always be missed.

Chapter 1

INTRODUCTION

Automatic speech recognition (ASR) has long been a goal of signal processing research. Over the last few decades, automatic transcription has become increasingly reliable, to the point where the technology could be transferred into commercial, educational, and government applications. Such applications leverage ASR to provide a service, often in the context of an information-gathering or information-dispensing system.

An important early application of speech technology was the automation of various telephone customer service tasks, such as performing automatic call center routing or providing automated airline reservation systems. Such systems benefited from a relatively small set of commands the user had to provide. Furthermore, utterance structure was relatively rigid; thus, even when a larger vocabulary was necessary (for instance in flight reservation systems, where thousands of destinations may be available), most of the available vocabulary would show up in only specific parts of a user's utterance.

The improvement of ASR technology has led to more open-domain systems, such as voice search applications on mobile devices. While the user *intents* that systems handle are few, the way these intents are expressed may vary significantly from one user to the next. Thus, natural language processing (NLP) techniques are required in order for the system to interpret a user's request and produce the desired information in return.

Traditionally, most search applications (whether text or voice search) considered user interactions as being stateless, at least from the *user's* perspective; that is, if a user wished to refine an earlier query, the query would have to be repeated, with the additional constraints inserted in the appropriate string. More recently, however, more and more applications have been developed that preserve state across such requests. Chief among them are *personal assistant* applications, such as Cortana, Siri, or Google Now. Key in such systems is the avoidance of recognition errors, especially in the beginning of an interaction (i.e. *user*

session), since early mistakes may often be propagated all through the course of the session, leading to significant degradation in user experience. If errors cannot be avoided, it is important for the system to detect that an error was made, even if the correct transcript cannot be generated; in such cases, the dialog manager component of the system can ask the user for a rephrase or clarification of the errorful segment of the utterance before continuing with the interaction.

1.1 Handling ASR Errors

Speech recognition errors tend to manifest themselves in a variety of ways. Often, short words tend to be combined with their neighboring words, whereas longer misrecognized words are split into a sequence of shorter lexical units; the latter phenomenon can also cause errors to spread to neighboring words, particularly those immediately preceding or following the word causing such problems. For example, the phrase *at Litanfeeth*, which contains a foreign name, gets misrecognized as a sequence of short words *it leaks on feet*. The problem is particularly severe when the utterance contains long words not present in the recognizer vocabulary, since by definition they will always result in errors. However, other rare words, particularly names, can also behave in a similar manner.

Whether due to out-of-vocabulary (OOV) words, rare names, or just general misrecognitions, erroneous regions in ASR output often result in a syntactically-anomalous transcript, particularly when one long word (perhaps together with its neighbors) is replaced by a sequence of shorter words. It follows therefore that detecting such syntactic anomalies can help identify errors before they are presented to the user. Shallow syntactic constructs, such as part-of-speech (POS) tags, can help, but modeling the richer underlying syntactic structure can serve to further guide the system in its attempt to resolve recognition errors; for instance, correctly identifying the syntactic category covering the errorful region would allow for the appropriate *wh*-question to be asked.

When the ASR output contains only the top hypothesis obtained from the decoder, even if an error is detected, little can be done other than asking the user for a rephrase. On the other hand, if more than one hypothesis is obtained, parsing can be of use beyond simply detecting errors, by helping to disambiguate between transcripts for potentially-errorful

regions of the output.

Parsing performance is susceptible to domain mismatch between the training and evaluation sets. Both topic and style mismatch can lead to problems; topic differences mean that many of the topic words in the test set will be OOV to the parser, and thus more likely to be tagged with the incorrect POS; this can lead to additional errors in the higher level structure. However, the style of the data causes much more severe problems; the rate (and type) of disfluencies, contractions, or even a mismatch in the rate of use of first, second or third person language in the different datasets, or the rate of using active vs. passive voice, can cause many mistakes in the parse output. With treebank data being expensive to hand-generate, domain mismatch is a likely issue in many practical scenarios when parsing is used to process speech transcripts. Even for English, treebanked data is available only for formal, written and read news data, and for a small amount of informal telephone conversations. On the other hand, data from the target domain[1] that has been hand-transcribed but not hand-labeled with parse trees is often readily available. The main goal of this dissertation is to make more effective use of parsing in both automatic rescoring of ASR output and detection of different types of ASR errors by using unlabeled (for parsing) data from the target domain to improve a parser trained using treebanked out-of-domain data.

Even when labeled in-domain training data is available, such as for the error detection tasks, if the amount of available training data is small, the learned models may not be robust. This is particularly an issue when building classification systems with a large feature dimensionality, such as when using lexical n-gram features for error detection tasks. A second goal of this thesis is to develop algorithms for learning more effective features for error detection, improving the generalizability of the models to unseen data.

1.2 General Framework and Contributions

The work presented in this thesis is designed as a component of a speech-to-speech translation system, i.e. a human-computer interaction system designed to mediate the communication between two users speaking in different languages. In such a system, the users speak

[1]Data from the target domain is often referred to as *in-domain* data, with data from any domain other than the target domain being referred to as *out-of-domain* data.

in alternating turns. Each user's utterance is transcribed automatically, then translated into their conversation partner's language; text-to-speech technology is used to generate an audio signal which is then played back to the second user. Previous systems, such as [6], focused on providing a smooth translation experience, with the computer acting as a transparent interface. In our current setup, the interaction between the two users is mediated by a hands-free, eyes-free human-computer dialog interface that requests clarifications or restatements from each user, in particular to resolve potential ASR or machine translation (MT) errors prior to the final translation to the opposite language, as described in [14]. In such a system, due to the hands-free, eyes-free requirement, the user cannot be shown the ASR output, and therefore the user cannot detect mistakes made by the system without further speech prompts from the system itself. One approach to do this would be to have the system recite back to the user its transcription; however, such an approach is prone to causing unnecessary delays in communication and an increase in user frustration, in particular when the recognition accuracy is good. On the other hand, it is important to handle ASR errors before the translation component is used, as errors in the source language will likely be compounded during translation. Thus, if speech recognition errors can be detected, the system may ask the user to rephrase or restate their utterance in such a way as to alleviate the recognition problems and improve the chance of successful communication.

This work focuses on improving automatic handling of ASR errors in two ways. First, we work on rescoring the ASR output to produce a new hypothesis with lower error, for cases when the correct alternatives exist in a richer form of the ASR output, such as a word lattice. Second, for errors that cannot be corrected automatically with the available information, we attempt to detect their location and extent (within the ASR hypothesis), as well as the type, in order to handle them effectively during the subsequent clarification request made by the dialog system component. In particular we are interested in two types of ASR errors. If the incorrect region of the transcript corresponds to a word which does not appear in the recognizer vocabulary (i.e. an OOV word), then that word will never be transcribed correctly and thus the user should be asked to rephrase the utterance while avoiding the use of the OOV. If the incorrectly-transcribed sequence corresponds to a name, translation may be a matter of simply transliterating the word into the target language,

or playing back the audio corresponding to that name, as provided within the speaker's utterance.

The main intellectual contributions of this dissertation include building a novel system for detecting and, when possible, automatically correcting speech recognition errors using parsing; developing two approaches for automatically adapting parsing models with the goal of improving the performance of speech error processing; and learning novel long-distance lexical features to address data sparsity issues while reducing overtraining.

1.3 Dissertation Overview

Chapter 2 introduces the scientific underpinnings of this work. We discuss past work on speech recognition, focusing on rescoring and error detection tasks, and present the most relevant work on parsing. We also discuss feature learning methods and general domain adaptation approaches.

Chapter 3 describes the general framework into which this work is integrated. We also introduce the data we use for all the experiments presented in this thesis, discussing style characteristics of each corpus and the challenges in making effective use of this data, leading to the methods we pursued.

Chapter 4 introduces a new system for rescoring and error detection. We describe in detail a method for integrating parse information into the rescoring and error detection pipelines, and introduce a novel set of features intended to capture parser reliability. We also present a set of baseline experiments for rescoring and error detection, with a focus on motivating the new domain adaptation methods and feature learning approaches presented in this thesis.

Chapter 5 introduces the two proposed domain adaptation approaches, as applied to the problem of ASR rescoring. We explore both self-training methods and a model combination approach using log-linear models to extend standard PCFG models, and investigate their effectiveness in improving recognition performance over un-adapted models as well as baseline ASR output.

The two parser adaptation approaches are repurposed in chapter 6, where we investigate their effectiveness in improving OOV and name error detection. We present results for each

method in isolation as well as when used together, and discuss which methods lead to more generalizable results across evaluation sets with different characteristics.

In chapter 7, we present a novel phrase pattern feature learning algorithm using semantic representations of words. We apply the proposed algorithm to the problem of name error detection. We discuss the effect of the new features when used separately as well as in combination with parsing, and discuss how different sources of semantic similarity affect the overall performance as well as the generalizability of the learned features.

Finally, chapter 8 includes a summary of the domain adaptation approaches as applied to error detection and rescoring. We discuss several practical observations related to integration of this work into the larger speech-to-speech translation system, and discuss future directions to further advance this work.

Chapter 2

BACKGROUND

In this chapter, we discuss methods and algorithms related to this work. We start with a brief introduction to automatic speech recognition in section 2.1, which provides the foundation of this work and represents the first task that we address. We introduce parsing and parsing language models, key components of this work, in section 2.2. Section 2.3 introduces error detection as the second task of interest to this thesis, including its relationship to the general speech recognition framework, and discusses prior methods. In section 2.4 we present previous approaches to learning complex lexical features for text classification tasks, which is important for error detection. In section 2.5, we discuss domain adaptation approaches related to both tasks. Section 2.6 includes a short summary of the background to motivate the approach taken to build the systems discussed in this thesis.

2.1 Automatic Speech Recognition Systems

A typical speech recognition system takes as input an audio file containing the acoustic signal and produces a sequence of words as the most likely lexical string corresponding to the acoustic observation. Most modern systems employ statistical approaches to obtain the text output given the acoustic signal. The basic mathematical framework for a typical single pass ASR system is:

$$W^* = \text{argmax}_W \, p(W|X, \theta) \tag{2.1}$$

$$= \text{argmax}_W \, p_A(X|W, \theta) p_L(W|\theta) \tag{2.2}$$

where Q represents the predicted word sequence, X is the acoustic signal, and p_A and p_L represent the acoustic model and language model, respectively (with parameters θ). We discuss some typical methods for acoustic models in section 2.1.1 and language models in section 2.1.2. Multi-pass systems may include rescoring intermediate ASR representations,

as well as various adaptation approaches; the latter are out of the scope of this dissertation. We discuss ASR rescoring in subsection 2.1.4.

2.1.1 Acoustic Modeling and ASR Dictionaries

In most state-of-the-art speech recognition systems, the acoustic model is composed of two separate systems: a subword-level acoustic model, which is used to map acoustic observations to phone or other subword-level units, and a pronunciation model, which maps sequences of subword units to words. We discuss each of these components briefly here.

Most acoustic models use Hidden Markov Models (HMM) [17] as the primary modeling tool [61]. An HMM is a generative sequence model which makes a (typically first-order) Markov assumption about the conditional distribution of a state given past states. Various representations for the observation space are used; among the most common are context-dependent Gaussian Mixture Models (GMM), where each state is modeled as a mixture of Gaussian distributions. More recently, much success has been obtained from replacing the GMM approach with a deep neural network (DNN) architecture, with the DNN layers trained discriminatively; details can be found in a recent survey paper [59]. In both the GMM and the DNN case, various acoustic features are used to model the input signal; two of the most common are Mel-Frequency Cepstral Coefficient (MFCC) features [93] and Perceptual Linear Prediction (PLP) features [56].

Pronunciation models are most often dictionary-based, with one or more sequences of subword units provided for each word. More sophisticated approaches weight the possible pronunciations probabilistically. To cover words not seen in the pronunciation model training data, a grapheme-to-phoneme (G2P) system can be used to generate a pronunciation given the orthography of a word.

2.1.2 Language Modeling

A language model computes the probability of a word sequence, which can be expressed using the chain rule as

$$p(w_1^N) = \prod_{i=1}^{N} p(w_i | w_1^{i-1}).$$

N-gram language models use an $(n-1)$-order Markov assumption so that we can write $p(w_i|w_1^{i-1}) \approx p(w_i|w_{i-n+1}^{i-1})$. For simplicity we often write $w_{i-n+1}^{i-1} = h_i$, the history of the ith word in the sequence.

One major shortcoming of simple n-gram models is their inability to estimate low-frequency events. To address such issues a number of approaches have been developed, roughly falling into two categories: i) backoff methods that resort to using a shorter history when insufficient observations are present in the training data for the full history context, with heuristic weights set to assign non-zero probability mass to each event and to ensure a valid probability distribution is generated; ii) interpolation approaches that linearly combine distributions with different length histories. In practice, the two methods are often combined, with one of the most successful approaches being the modified Knesser-Ney method [32]. More details of various smoothing and backoff methods can be found in [32].

An alternative method for dealing with low frequency events involves pooling words with similar function or behavior into word classes. Different methods of learning word classes exist; one of the most common involves using word co-occurrence statistics with an agglomerative clustering algorithm, merging classes to maximize likelihood [27]. This potentially allows for estimating probabilities involving infrequent words more reliably; on the other hand, the predictive power of the model is reduced, since there are fewer free parameters. Thus, there is a trade-off between improving the reliability of the model estimation and reducing the prediction ability of the model. Word classes, such as those described in [27], are also used in text classification tasks; we will discuss that usage (together with forms of feature clustering) in section 2.4.

More recently, there has been renewed interest in neural network language models. Originally introduced by Bengio et al. [20], the most successful current methods generally use a recursive neural network structure, such as the architecture introduced by Mikolov et al. [97] for use in ASR decoding tasks. In such a language model, the one-hot representation of each word is first mapped onto (or embedded into) a continuous space; various methods are used to compute the output layer given the word embeddings. We note that word embeddings can be considered as a form of soft-clustering; such word embeddings have been used in a variety of tasks to reduce the dimensionality of the feature set and improve performance

(for example, named entity recognition or chunking [144, 55]).

Of the language models that are more complex than n-grams, those most related to the methods we will explore are syntactic language models, which we will review in detail after we discuss parsing, at the end of section 2.2.

2.1.3 ASR Output Formats

Many standard ASR decoding tasks, such as voice dictation, use as output format a single string containing the "best" (i.e. "most likely") word sequence according to the ASR models. We refer to this as the *(ASR) 1-best* (or *1-best hyp*). Sometimes the 1-best hyp also contains word posteriors and time alignment information; we do not distinguish such formats as different, instead referring to them collectively as the *1-best*.

When the ASR output is used in further downstream processing, such as machine translation or NLP applications, intermediate ASR output formats which provide more information are often used. The simplest such representation is an *n-best* list, where instead of storing only the top hypothesis produced by the system, we store the top n hypotheses according to their overall probability. This representation has the advantage of simplicity, in that each possible utterance can be processed independently of the others. However, in practice, there is significant overlap between the top n hypotheses, thus causing unnecessary space to be wasted during the processing. For large vocabulary tasks, the n selected must be very large, in order to get broad coverage of the space of possible utterances to generate.

To alleviate the problems posed by n-best lists, a popular alternate representation is a word *lattice*, which is generated during the standard HMM decoding process. The word lattice is a directed acyclic graph (DAG). Words can be represented either at the node level or by each edge in the graph; additionally, each word may also have a probability, start time, and end time associated with it. Additional symbols are used to encode silence and noise regions in the input signal. We present an example of word lattice in figure 2.1. In this example, words are encoded in each edge, and only the word start time is specified, with the start symbol (not pictured) assumed to be at time $t = 0$.

Word confusion networks (WCN) represent a special case of general word lattices, with

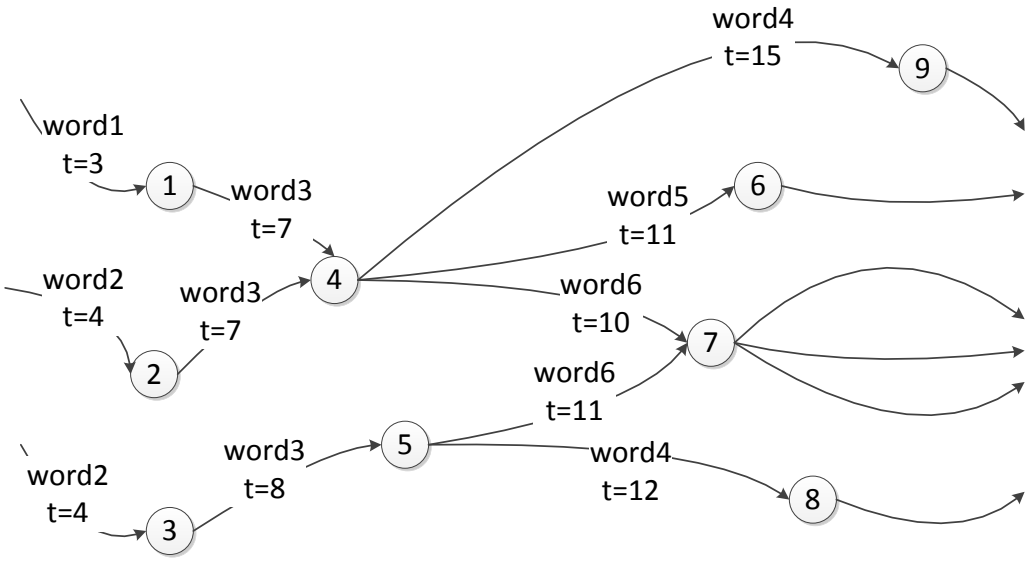

Figure 2.1: Example of a section of a word lattice.

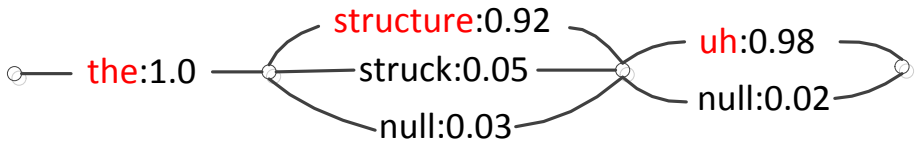

Figure 2.2: Example of a section of a word confusion network.

a more compact, regular structure. A confusion network contains a sequence of slots, with each slot containing one or more word arcs. Selecting a path through the confusion network therefore reduces to selecting which arc in each slot is being used. This representation has been shown to provide similar performance to general lattices in ASR rescoring tasks [82]. An example of a confusion network for a short fragment of an utterance is included in figure 2.2. The highest-scoring path through the network (i.e. the best single hypothesis produced by the ASR system) is marked in red. The reference transcript for this utterance fragment is "the structure", thus the last slot in the WCN corresponds to an ASR insertion.

2.1.4 ASR Rescoring

Many state-of-the-art ASR systems use a multipass approach, with an initial decoding followed by potential rescoring of the intermediate ASR output; additionally, unsupervised adaptation of the acoustic models may also be used, with a second decoding pass with the adapted models. In this discussion, we focus primarily on rescoring methods; acoustic model adaptation and other multipass techniques are out of the scope of this dissertation.

Rescoring techniques are developed for a variety of reasons. Frequently, more expensive models may be used in a rescoring pass than in the full decoding configuration. This is often the case with language models, where a higher order n-gram LM is used in rescoring [61]. Such a method is used by Chelba et al. [29]; their system is built using Google n-grams (230B words from Google queries selected randomly). They find that such a large model is important for web-scale systems, such as Voice Search and YouTube transcription tasks, even when the domains are mismatched (e.g. in YouTube transcription). More complex LMs may also be used. For example, Jeon et al. [64] describe a novel architecture for n-best rescoring using a Trie DB-based language model. Additional methods using syntactic LMs are discussed in section 2.2.2.

An alternative reason to perform ASR rescoring is to optimize the ASR performance for an objective other than WER. For example, in [69], the authors examine joint optimization of ASR rescoring and parsing, with the goal of improving parsing performance. Morbini et al. [102] work with a natural language understanding system; in this case, the goal of ASR

rescoring is to improve the NLU processing rather than rescoring for WER improvements. They integrate the rescoring of ASR hypotheses with the NLU hypothesis reranking algorithm, finding that the integrated reranking approach outperforms the pipeline approach. In this work, we will also investigate this approach for error detection.

2.1.5 ASR Evaluation

The most common metric for ASR performance is word error rate (WER), which measures the number of errors of various types in the top hypothesis produced by the ASR system. WER distinguishes between three types of errors:

- **insertions** (*ins*) - extra words added in the recognized sentence
- **deletions** (*del*) - correct words removed from the recognized sentence
- **substitutions** (*sub*) - incorrect words which replaced correct words in the recognized sentence

We compute word error rate as:

$$\text{WER} = 100 \times \frac{\#ins + \#del + \#sub}{\#\text{words in reference}} \tag{2.3}$$

The naive alignment (comparing the words in the reference and the 1-best hypothesis one by one, starting from the left) does not lead to the optimal solution. Instead, we generate the list of insertions, deletions, and substitutions using a dynamic programming algorithm to solve the maximum substring matching problem.

Figure 2.3 presents a simple example of a reference string and its corresponding 1-best output from the recognizer. Erroneous words in both the reference (top string) and the 1-best (bottom string) are indicated with red. In this case, the errors consist of one substitution, one insertion, and two deletions, for a WER of $100 \times \frac{4}{9} = 44.4\%$.

The WER formula indicated in equation 2.3 does not assign any specific word higher or lower value than the rest. In practice, however, that is not the case. For example, in the above example, one of the deletions was the filled pause “uh”. For some applications, such as

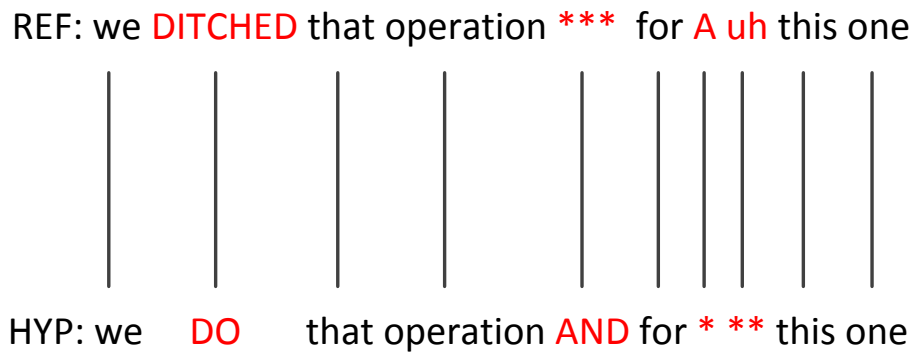

Figure 2.3: Word error rate computation example

topic detection, machine translation, or summarization, missing the filled pause (or other) disfluencies would likely not affect the end result.[1] In such cases, using a modified version of WER scoring, designed to ignore certain differences between the reference and hypothesis strings, provides a better measure of the likely performance of downstream applications. A standard modification is to allow simple substitutions, such as negation-related contractions and their fully-expanded form (e.g. *don't* → *do not*), and not counting insertions or deletions of filled pauses (e.g. *uh, um*). We will use this approach for scoring the ASR rescoring systems presented in this work, using the NIST SCLITE [2] toolkit, which is a standard tool in the speech community.

Other approaches weigh different components of the WER computation unevenly (for example, giving more weight to insertions over deletions), or may use certain characteristics of the error words to determine a per-word weight (e.g. using the words' POS tags). Hillard [57] discusses using SParseval [123] as an alternative metric for ASR performance when the goal is to improve translation accuracy. We will use a modified WER metric for measuring the effect of the error detection methods, as described in chapter 4.

[1] Not all tasks benefit from disfluency removal; for example, for tasks involving extraction of paralinguistic information, a missed disfluency in the decoder output would be a problem.

2.2 Parsing

At its core, syntax describes the structure of a sentence, both in terms of what "kind" of words are used, and how they relate to each other. As such, syntax is widely used in the speech and language community as a modeling tool for both text and spoken transcript data. Syntactic models may be applied to many other tasks, such as providing features for machine translation or summarization, or assessing the "goodness" of an automatically-transcribed utterance. Key to describing the syntactic structure of a sentence is the process of (syntactic) parsing, i.e. the automatic generation of a structure that describes what words are used and how they are organized together within the sentence, for example in terms of a hierarchical constituent structure, or in terms of dependence relations between words. Most state-of-the-art methods in syntactic parsing are statistical in nature, though many use rules as part of the process. We will focus on two such methods in this discussion, and briefly discuss methods of evaluating parsing accuracy, then discuss how syntax is used in language modeling.

2.2.1 Syntactic Parsing Methods

We focus the discussion on two methods of syntactic parsing, intimately tied to the linguistic framework used to describe the structure of a sentence. In constituency parsing, we model a sentence as a nested set of progressively smaller units (*constituents*). Each constituent can be thought of as a distinct unit, with its own characteristics and structure. The constituent *parse tree* of a sentence thus describes how the constituents are arranged and fit together within the sentence.

Constituent parsing often uses a probabilistic context free grammar (PCFG) to express the relationship between constituents. A PCFG is composed of a set of unary and binary context-free rewrite rules [68], such as those shown in figure 2.4, with the grammar either defined explicitly with hand-labeled data or induced automatically from an unlabeled set. While unsupervised grammar induction is sometimes used, most often the grammar is specified a priori by linguists. Supervised parser training is generally done using hand-annotated trees (i.e. a *treebank*), in which case estimating the model is simply a matter of counting

$$S \rightarrow NP\ VP$$

$$NP \rightarrow DET\ NN$$

$$NN \rightarrow cat$$

Figure 2.4: PCFG rule examples.

all the rules to estimate their probabilities using a maximum likelihood criterion. Treebank resources exist for both text (e.g. newswire) data [4] and conversational (mostly conversational telephone speech, or CTS) domains [5, 1]. An example of a PCFG parse is shown in figure 2.5.

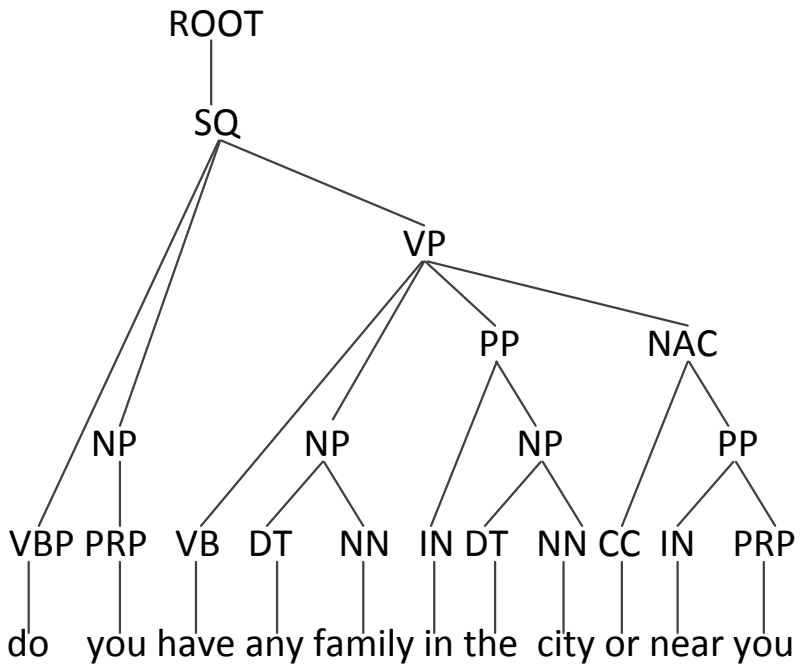

Figure 2.5: Constituent parse example.

Given an existing grammar (whether hand-specified, induced over unlabeled data, or learned using supervised data) and a corresponding PCFG model, a number of algorithms for constituency parsing exist. One of the most common PCFG parsing algorithms is the

probabilistic Cocke-Kasami-Younger (PCKY) algorithm [107], a bottom-up algorithm which uses dynamic programming to induce possible derivations over all subsequences of a sentence in an efficient (polynomial-time) way.

Performance of PCFG parsing is improved by removing the constraint of using context-free rules, allowing for some amount of dependence on context, primarily through lexicalization and specialization of rules. A number of lexicalized PCFG algorithms have been developed over the last decade [36, 119, 70].

Dependency Type	Dependency Label Examples
argument	subject (subj), object (obj), indirect object (iobj)
modifier	determiner (det), noun modifier (nmod), verbal modifier (vmod)

Table 2.1: Examples of different dependency types and labels.

An alternative method to constituency parsing focuses on only binary relations between the words in a sentence, rather than the constituent units used in PCFG parsing. We refer to this as *dependency* parsing, in that for each sentence, its parse is a tree structure where the nodes correspond to the words in the sentence, and the links represent dependency (governor → modifier) relations [68]. Dependencies may be untyped or typed, in the latter case the type representing various kinds of relations, as shown in table 2.1. An example of a dependency parse is shown in figure 2.6, using the same sentence used in figure 2.5. Various algorithms for dependency parsing have been developed in recent years [31, 105].

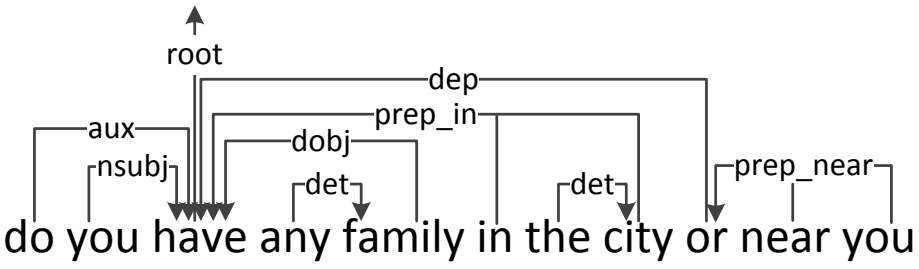

Figure 2.6: Dependency parse example.

Hybrid methods also exist. For example, the fast factored parser model [71] uses a PCFG model to generate the n-best candidate parses for a sentence, then rescores these parses with a dependency model, where the dependencies for the candidate constituent parses are extracted automatically using a standard head-finding algorithm [34].

In this work, we use an implementation of the fast factored parser model [71] in the Stanford parser, with modifications to the PCKY algorithm as described in section 4.1.2. We select the fast factored parser approach because it allows us to obtain syntactic categories for subphrases, which we believe to be useful in downstream processing, such as user adjudication of possible ASR errors. The rescoring with dependency information provides a framework both for extracting features from the parses (with dependency information already provided) and for improving the rescoring of the n-best parses (by adapting the dependency-based reranker).

Evaluation of parsing methods tends to depend on the parsing framework employed. With constituency parsing, a standard evaluation method is bracketed precision/recall/F-score. In this method, a list of constituents is generated for both the reference tree and the parser-predicted tree. Each constituent in the list is represented as a triplet containing its label, start position, and end position. The overlap between the two lists is computed in terms of the precision, recall, and F-score metrics. For dependency parsing, we use attachment scores, i.e. the number of dependencies in the candidate tree which appear in the reference tree, either not counting labels (i.e. *unlabeled*) or requiring labels to match (i.e. *labeled*). Label accuracy, i.e. just the number of correctly-detected labels, may also be used as a metric.

While the metrics mentioned above are useful when measuring the intrinsic performance of a particular parser, when the parser is only a tool used to improve a downstream application or task, metrics more directly associated with the downstream system may be used. This is often the case in syntactic language modeling, when we evaluate the parsing LMs using perplexity or (if used for ASR decoding or rescoring) word error rate over the rescored transcripts. We will use a similar approach in this work, using WER as measure of parsing performance when the parser is used as a rescoring language model. For error detection tasks we will use a modified WER metric as well as error detection F-score, as described in

more detail in section 4.1.4. In either case, by not focusing on measuring the performance of the parser itself, we do not need any in-domain treebank data, only data annotated with the labels of interest (ASR errors and their types).

2.2.2 Syntactic Language Models

In this section we discuss in more detail language models that incorporate syntactic information, with an emphasis on their use in ASR. Syntactic language models have been used successfully for both first pass decoding and various forms of ASR rescoring. Chelba et al. [30], in one of the earliest such works, develop two extensions of standard n-gram LMs by integrating dependencies on preceding head words as well as local history words, and chaining one level of such dependency structures to build more complicated hierarchies. Rastrow et al. [121] improve upon [30] for read news, using a state-of-the-art probabilistic shift-reduce parser. They use hierarchical interpolation of the syntactic model parameters to avoid increasing the model complexity, and prune the model using a relative entropy pruning technique.

Wang and Harper [145] use Constraint Dependency Grammars (CDG) to build a new almost-parsing language model, using lexical features and syntactic constraints integrated in a new linguistic structure called *SuperARV* (*super abstract role value*), providing a mechanism for lexicalizing CDG parse rules. Among the syntactic information they use in the SuperARV tags they include the lexical category of the word, feature vectors related to the word morphology and syntax (e.g. **case** for nouns, **mood** and **voice** for verbs), and the structure of the dependencies linking the word with each of the words it modifies. They evaluate this model on read speech, in both direct decoding and rescoring settings. A follow-up paper [146] investigates the performance of the SuperARV-based almost-parsing LM when the training data may be inconsistent, as well as cross-domain, applying it to broadcast news domain data using only automatically-generated parses from newswire-trained parsers.

Syntax is also used in some discriminative language models. For example, Collins et al. [37] build a discriminative language model, extending the work done on discriminative n-best rescoring in [124] by also using syntactic features. In this work, the n-gram LM is

augmented with features extracted from sequences of words with their syntactic information, which include POS tags, chunking, and head word structure. They also experiment with removing subtrees corresponding to disfluencies from the syntactic sequence features. This model is applied to Switchboard data.

While much of the parsing LM work was done with a goal of improving ASR, this is not always the case. For example, additional investigation of the connection between disfluencies and parsing performance in the context of ASR output is done in [52], with focus on parsing performance and other downstream applications, rather ASR than itself. Kahn et al. [69] perform a study using joint parsing and speech recognition for conversational speech, with focus once again on parsing quality, not improving ASR. In this thesis, a similar approach is taken for error detection, with parsing performance measured in terms of optimizing error detection rates in addition to a WER metric.

Much of the parsing language model work focuses on adding parsing information into an existing language model framework, either through conditioning on parsing structures (such as the SuperARV work) or building discriminative LMs with parse features. In this work, we use the parser directly as a language model, taking the parse score as an indicator of hypothesis rank. We will also extract parse features similar to those used in [69], though use them in error-detection classifiers instead. More importantly, in most of the works reviewed here, the parser training domain matches the application domain. This is different from the setup used in this thesis. We will address parsing domain mismatch issues in section 2.5.

2.3 ASR Error Detection

We discuss error detection approaches next. Since many error detection methods are set up as classification tasks, we first give a brief overview of text classification, with emphasis on the approaches used in this work. We then discuss confidence estimation methods from the perspective of error detection in ASR. Finally, we discuss methods targeting specific types of ASR errors, such as misrecognition of OOVs or misrecognized names.

2.3.1 Text Classification Approaches

In a typical classification task, a model is learned from training data and used to predict a label for each test sample, with labels selected from the same set as those used in training. Typical classification systems are composed of a feature extraction component, which takes the raw input data and composes a vector of relevant features for classification, and a decision function, which will generate a classification decision given the feature vector for that test sample.

ASR output is inherently structured, in that there is a temporal dependency between the current word and those "preceding" it (for example, in the form of lattice arcs or confusion network slots leading to the current word). Nevertheless, often a simplifying assumption is made, that each word can be treated independently for purposes of classification, with information from neighboring words or regions of the output being represented in the form of features associated with the target word.

Many classifiers have been used for error detection tasks when such simplifying assumptions are made. One category of such algorithms is that of generative classifiers, which model the joint distribution of the data and class labels:

$$p(y, x) = p(y)p(x|y) \tag{2.4}$$

So that the conditional distribution of the class given the observations, $p(y|x)$, is obtained through application of Bayes' rule:

$$p(y|x) = \frac{p(x, y)}{p(x)} = \frac{p(y)p(x|y)}{p(x)} \tag{2.5}$$

The most popular among these models is the naïve Bayes [43] classifier, which assumes that features are conditionally independent given the class label.

An alternative approach is to use discriminative classification, in which case the conditional distribution $p(y|x)$ is modeled directly. Examples of discriminative classifiers include maximum entropy (MaxEnt) (also referred to as logistic regression in some literature) [21], support vector machines (SVM) [38], or boosting [130]. These classifiers tend to have similar loss functions (the hinge loss in the case of SVMs, the logistic function for MaxEnt, or the exponential loss for boosting with decision tree stumps). Advantages of the various

approaches have to do with whether the classifier is inherently multi-class (as in the case of boosting or MaxEnt) or requires voting to select the final classification decision from among a number of binary (for example, pair-wise or 1-against-all) classifiers (as is the case with SVMs), as well as robustness to overfitting when large feature sets are used.

If the simplifying assumption of label independence is dropped, more complex sequential models must be used. As with non-sequential models, a variety of techniques have been developed. Among the most popular generative sequence models are HMMs. Discriminative models used most often are all members of the exponential family. Many systems employ Maximum Entropy Markov Models (MEMM) [76], which extend MaxEnt models assuming a Markov structure in the label space; such models are effective at capturing dependencies between labels, but suffer from label bias issues when the number of out transitions from a particular state is small. Conditional Random Field (CRF) models [76] address this label bias issue by performing the normalization over the entire sequence instead of separately for each time slot.

Throughout this dissertation we use maximum entropy models for error detection tasks, since they scale well to using large numbers of features relative to the training set size. The maximum entropy framework treats each slot in a confusion network as independent, which does not take advantage of the sequential dependence of errors. However, maximum entropy models are relatively easy to train, in comparison to sequential models such as CRFs, requiring fewer labeled samples to achieve good performance. We attempt to capture the sequential nature of the data through the use of more complex features, particularly those derived from parse information, as we will discuss in section 4.1.2. We will discuss the labeling strategies used in this thesis, as well as issues related to label independence, in section 4.1.4.

2.3.2 Word Confidence Prediction

While speech recognition accuracy has improved significantly over the years, errors are still common, in particular in low resource scenarios or when having to deal with noisy data. Thus, estimating when the output of the recognizer can be trusted is an important metric

for any downstream applications which process spoken language data.

An approach commonly used to generate error confidence measures is to use the word posteriors obtained during the decoding process; however, due to pruning of the search space during the decoding process, ASR systems are often overconfident. Various approaches have been proposed to deal with this issue over the years, such as using lattices instead of N-best lists [151, 152], or directly optimizing the acoustic and language model scores to improve the word posteriors generated during decoding [136].

A second approach in confidence estimation involves post-processing the output of the recognizer with a parametric model trained on binary (*true*/*false*) labels indicating whether each word corresponds to an error or not; the posterior of this classifier is taken as the confidence prediction. Various classifier paradigms have been used, such as linear models [46], generalized linear models [132], neural networks [149], decision trees [106], boosting [103], maximum entropy models [153], or SVMs using confusion networks [58]. A more recent approach combines a deep neural network-based system and a GMM-HMM system using confusion network and RNNLM [97] features [142]. A survey of many of these methods can be found in [65].

2.3.3 OOV and Name Error Detection

There have been a variety of approaches explored for OOV detection, which can be roughly characterized in terms of whether they use explicit OOV models in decoding and/or post-processing of ASR hypotheses. The explicit decoding approach often incorporates a "garbage" (e.g. phone loop) model or a hybrid word and subword vocabulary that has a high expected OOV coverage [13, 26, 18, 129, 22, 122]. Such an approach can be effective in a constrained-domain application, but it tends to have a high false detection rate in open-domain or large vocabulary systems. Including sub-word fragments is also particularly useful for highly inflected languages. Other approaches include aligning independently-generated word- and phone-based lattices, and then associating the poorly-aligned regions with OOVs [81], and encoding OOV words as phonetic lattices in a weighted finite state transducer (WFST) framework for spoken term detection queries [116].

Another approach is to build upon methods for generating word confidences as a post-processing step. Such methods identify regions of ambiguity in the ASR output and take advantage of the fact that OOV errors often result in anomalous word sequences or lattice structure. Earlier work often used lattice information [157, 141, 111, 78] or a combination of one-best word and phone sequences [72], together with language model and semantic context features. More recent work typically uses word confusion networks to represent the ASR hypotheses [75]. These approaches are combined in [114], where the OOV detection is performed using a CRF with contextual features representing structural, local lexical, and global lexical and syntactic information.

The work presented in this thesis distinguishes itself from previous methods in a number of ways. First, while explicit decoding approaches perform well in small vocabulary settings or with spoken term detection tasks, they are less effective in large vocabulary, open domain settings. The work in this thesis builds upon confidence estimation approaches but makes more use of syntactic information. Among the works discussed here, Parada et al. [114] take an approach closest to that of this thesis, integrating parsing information as well as structural and local lexical features, but use a CRF. We instead use a simpler exponential model (MaxEnt) to avoid the difficulty of training a CRF in sparse data conditions, and model the sequence information through features extracted from syntactic structure; we also employ a novel method of learning a parser that acts as classifier. Furthermore, we directly address domain differences between the treebank used to train our parser and the target domain for our tasks by exploring domain adaptation methods.

Work on name error detection follows approaches similar to those used in OOV detection. Early work on name error detection incorporated ASR word confidence estimates in a named entity recognition (NER) system [112, 111, 139], taking advantage of local contextual cues to names (e.g. titles for person names). Parada et al. [115] extend this work by applying the NER tagger to a word confusion network based on a hybrid word/fragment ASR system, similar to their work on OOV detection [114]. Hatmi et al. [54] integrate NER directly into the ASR decoder via a generic NE token in the language model. Kumar et al. [74] experiment with different methods of integrating ASR and NER detection with a maximum entropy classifier and part-of-speech features. Chen et al. [33] propose a variable-span approach to

directly tagging longer spans of consecutive words instead of individual words.

While much of the recent work addresses name error detection through a combination of confusion network and structural and lexical features, similar to this work, there has been less emphasis on parsing. Thus, the parsing-based approaches discussed here introduce novelty in this context, too. Additionally, name errors are different from non-name OOVs in that lexical cues indicating the presence of a name may not be co-located with the name, lexically *or* syntactically. We address this issue using novel methods for learning long-span lexical features while addressing overfitting problems caused by data sparsity.

2.4 Feature Learning

Many kinds of features have been used in text classification tasks. Text classification systems often use lexical features, such as n-grams, and lexicon containment features (i.e. binary features, holding positive value if at least one word in the lexicon occurs in the text), with the lexicons either designed by human experts [117, 155] or automatically-generated [84, 85]. Various word cluster features, such as part-of-speech (POS)-based or automatically-learned (e.g. [27]) have also been used.

We focus the review of feature learning methods relevant to the work in this dissertation. Two approaches are of interest to us. First, we discuss a number of different lexical features aimed at capturing long-distance context beyond that expressed in fixed-length n-grams. Second, we discuss a number of feature clustering approaches aimed at improving generalizability, as well as reducing the dimensionality of the space and thus diminishing the risk of overfitting and improving performance in sparse training scenarios.

2.4.1 Long Distance Lexical Features

Among the most common lexical features used in text classification are n-gram features, with the value of each feature being binary or real-valued (i.e. the frequency of the n-gram in a document, either unweighted or weighted, perhaps using TF-IDF scores [67]). n-gram features are by necessity local and do not capture information about the structure of the sentence. While sufficient for simpler tasks, such as topic classification, more complex problems, such as intent classification or slot-filling for dialog systems, or error detection

in ASR systems, often require some longer distance context to discriminate between the possible labels. One approach to derive such context is to use syntactic features. Recent papers using syntactic information for classification tasks include our past work on using dependency parse tuples for detecting authority claims [85] and the targets of alignment moves in multi-party discussions [89]. Syntactic features have also been used for error detection tasks, such as OOV detection [114].

Another approach is to design lexical features that can span longer word contexts, often allowing gaps between words. We refer to these as *phrase patterns*. Some phrase patterns can be hand-crafted regular expressions, which require human effort but are cheaper to apply; they can also be used to implement system constraints, which could be difficult to model in a statistical system. Statistical methods to learn phrase pattern features are often unsupervised to improve feature performance by allowing the use of unlabeled data. Such methods have been used successfully in a variety of text processing tasks [63, 154, 140, 143]. More recent work makes use of class labels to learn more effective phrase patterns, either in a purely-supervised scenario as done by collaborators at the University of Washington [158], or in a semi-supervised setting using unlabeled data and knowledge graph information, as done in our work with collaborators at Microsoft [86].

2.4.2 Word Classes

Feature clustering (i.e. generating new features by combining statistics for those observed in data) for classification is often done to improve generalizability, for example by using clustering methods which extend to unseen words. They also help improve the reliability of model estimation techniques in the absence of large amounts of (in-domain) training data. The methods most often focus on word features, though they can be extended to other kinds of features.

Much of the work on learning feature clustering methods for classification tasks has been done in the context of topic classification, either supervised or unsupervised. This is partially due to the availability of topic-labeled corpora, as well as the strong ties to information retrieval and document organization. In contrast, work on other high-level

NLP classification tasks, such as sentiment analysis, or classification of social acts in multi-party conversations, is relatively recent.

The clustering methods are primarily data-driven, though we can draw a distinction between methods based on distributional clustering. Other methods use external information, such as knowledge graphs, rule-based systems, and semantic representations. We can further distinguish between agglomerative and divisive approaches, as well as techniques using k-means or graph clustering algorithms.

We first discuss data-driven methods. One fairly popular method is based on work performing distributional clustering of English words, as introduced by Brown et al. [27] for language modeling (often referred to in the literature as *Brown classes* or *Brown clustering*). Much work has been done to extend Brown clustering for purposes of improving language modeling; we will not discuss those methods here. However, Brown classes have also been used in many NLP applications, including in our work on OOV detection [87] and with collaborators on name error detection [55].

A related method to Brown clustering is introduced by Pereira et al. [118] for classification. Distributional clustering uses the relative distribution of the context of the words to cluster them. The method described in [118] clusters nouns using agglomerative clustering based on their distributions as direct objects of verbs and vice-versa. Baker and McCallum [15] apply the same method to classification, using the class labels of the training documents as context.

Extensions, such as by Dhillon et al. [41, 42] instead use divisive clustering and a global mutual information-based objective, finding the clustering that minimizes the drop in mutual information between classes and words. Wenliang et al. [150] also use a global objective, adjusting the original pairwise class comparison objective proposed by Baker to account for the sizes of the clusters in the pair relative to the sizes of the other clusters accumulated thus far during the process.

In another derivative of the original distributional clustering work, Slonim and Tishby [133] define the Information Bottleneck method as a principled way of choosing the "right" similarity measure between word distributions, in the sense of preserving the most information about the class variable during the clustering process. A follow-up paper by the same

authors [134] applies this method to the problem of word clustering for topic classification of text documents. They use a greedy agglomerative hierarchical clustering algorithm and a naïve Bayes classifier. Bekkerman et al. [19] present a version of distributional word clusters computed using the Information Bottleneck method with a top-down clustering approach with annealing stages. Slonim and Tishby [135] discuss using word clusters obtained via agglomerative clustering with the Information Bottleneck criterion to derive document-level clusters. A double clustering procedure that uses the document clusters to improve the word clusters is also outlined.

A semi-supervised approach based on distributional clustering, used for word-sense disambiguation rather than topic classification, is used by Niu et al. [110]. They use a label propagation approach to assign labels to features observed only in unlabeled data based on their similarity with features in the labeled data. Their method is transductive in that the unlabeled examples are in fact the test set.

Other work has used external information as part of the clustering approach. Han et al. [51] introduce a rule-based word clustering method using hand-crafted rules, e.g. driven by static regular expressions. They use different word information, such as orthographic properties and domain databases relevant to some feature types. They use entropy methods to select representative words for particular word classes from training data. Gabrilovich and Markovitch [44, 45] use semantic analysis to generate high level concept features for text categorization and for computing semantic relatedness between document fragments. They map words onto concepts derived from various ontologies, either obtained from domain experts (as in [44]) or from Wikipedia using automated methods (as in [45]).

Finally, a few recent papers use language modeling techniques. Bassiou and Kotropoulos [16] first introduce a long distance language model, which explicitly models bigrams with skips between the predicted word and the history word. The long-distance language model compensates for the loss of syntactic and semantic information from distant words, relative to using standard n-gram models, while reducing the number of free parameters. The long-distance bigram LM is used in a variant of Brown-style clustering that uses a Mahalanobis distance-based objective, and in a variant of PLSA that uses the latent topic variables as word classes.

Mikolov et al. [94, 95] employ RNNLM methods to learn embeddings. In this work, two distinct RNN modifications are proposed. In the first version, each word is predicted given its preceding and succeeding context. In the second architecture, the procedure is flipped, so that each context word is predicted given the value of the current word; this latter architecture is referred to as a skip-gram, though it is distinct from the bigrams with skips in [16]. Training is optimized by noting that a good model should be able to differentiate data from noise; thus, negative (i.e. noise) class training data is generated via sampling, with a log-linear (logistic regression) model used to replace the softmax layer in the RNN. A paper by Levy and Goldberg [79] extends the skip-gram model proposed in [95] by replacing local lexical context with longer-distance context extracted from a dependency parse model.

2.5 Domain Adaptation

2.5.1 Terminology

Before we can talk about domain adaptation it is useful to define exactly what we mean by "domain" (as well as other related terms, such as "genre" and "style"). We use the term *domain* to refer to some human-defined scenario of interest whose data has a specific set of properties (possibly shared with other domains). For example, we consider "broadcast news" and "newswire" to be two distinct domains, since there are significant differences between them—chiefly that newswire is written text, whereas broadcast news usually includes audio segments of spoken utterances, often with Q&A involving multiple speakers in each segment or story. One or more *corpora* might exist that belong to the same domain, in that they all relate to the same human-defined scenario. For instance, a number of newswire collections exist, such as the various editions of the Gigaword corpus [49].

While a domain describes properties of the data, we use the term *task* to refer to language processing activities in support of particular applications of interest. For example, given the speech-to-speech translation domain TRANSTAC [3], tasks of interest will include ASR and machine translation, but also, potentially, parsing, named entity recognition, and coreference resolution. Subtasks may also be defined, refining the general ASR problem to

that of OOV or name error detection within the ASR transcripts, as discussed before.

Many researchers use the term *genre* to be synonymous with domain. Here, we use genre as being closer to *style*, which we define broadly as how people talk or write. Style includes lexical and syntactic characteristics of the data as well as formatting, capitalization, disfluencies (if spoken form), and other specifics of the medium and authors of the text [73]. On the other hand, *topic* describes what people talk about, and thus it, too, influences lexical choice, primarily related to content words. Domain characteristics thus are determined by a combination of topic and style (or genre) attributes.

2.5.2 Domain Adaptation Methods

Given the definition of a domain in the previous section, we take *domain adaptation* to mean, loosely, making use of data from one domain to improve a system targeting a different domain. In this case, too, a number of different but related terms are used by researchers in the field. We distinguish domain adaptation from *multi-task learning*, in which case the aim is to improve the performance on multiple tasks (or domains) concurrently. The term *transfer learning* is often used interchangeably with either domain adaptation or multi-task learning, depending on the context. Within the domain adaptation framework we can further distinguish between *unsupervised domain adaptation*, in which case the target domain data has no labels, and *supervised domain adaptation*, in which case target domain labels exist. We focus on unsupervised domain adaptation methods here. We note furthermore that unsupervised domain adaptation is related to *semi-supervised learning* methods, though the focus of semi-supervised learning is on using unlabeled data which is assumed to be drawn from the same domain as the labeled training set. However, many semi-supervised methods can be extended to unsupervised domain adaptation settings.

More formally, given a source domain S and a target domain T, we let $D_S = \{(x_i, y_i)\}$ be the (labeled) source-domain training data, and $D_T = \{(x_i)\}$ be the (unlabeled) target-domain training data, where $x \in X$ are feature vectors and $y \in Y$ are class labels. The goal of this work is to design effective parametric models $F : X \rightarrow Y$ that label test samples x drawn from the target domain T as $\hat{y} = F(x)$.

Even restricting to the space of unsupervised domain adaptation methods, there are too many approaches to describe comprehensively here. We focus the discussion on two approaches of particular relevance to this dissertation, namely self-labeling methods and feature representation approaches.

Self-Labeling Approaches

Self-labeling methods all have a similar structure. An initial classifier is trained, which is then used to label the unlabeled data. The automatically-generated labels are then used in an iterative fashion to build a new model. A number of different approaches exist; we focus here on self-training and co-training [25].

Self-training is one of the simplest self-labeling semi-supervised training approaches. In a domain adaptation setting, the algorithm starts by using the labeled source-domain data D_S to train the initial version of the model, F^0. We generate labels $\hat{y}_i^0$ for each target domain test sample $x_i \in D_T$. We use the automatically-labeled set $D_T^0 = \{(x, \hat{y}^0) \ : x \in D_T\}$ together with the source-domain data D_S to generate a new model F^1, then repeat the procedure until the termination criterion is fulfilled.

Variations of the algorithm use different termination criteria and data selection strategies. The most common termination criteria use either a fixed number of iterations or convergence of the confidence in the automatically-generated labels. Among the most popular data selection strategies are using only the samples labeled with highest confidence, or using the entire dataset at each iteration. An alternative method adds unlabeled samples according to the class prior distribution [120]. Hybrid approaches also exist, for example using a "cooling schedule" similar to that used in simulated annealing [77]. Another distinction can be drawn between "hard" self-training, when samples are added with a single label (i.e. the label with highest confidence or posterior as given by the model at the current iteration) and "soft" self-training, in which case the confidence of the automatically-generated labels is taken into account during the model re-estimation procedure [90].

The Expectation-Maximization (EM) algorithm has been used in several studies as a form of self-training. For example, Nigam et al. [109] use the standard version of the EM

algorithm to label unlabeled data and retrain the model using the automatically-labeled data in an iterative fashion. Their algorithm is similar to self-training with soft labels and using the entire unlabeled dataset to retrain the model at each step. While the standard EM algorithm (as used in [109]) requires a generative model, discriminative versions also exist [7, 50], in each case the resulting algorithm being similar to self-training with convergence as a stopping criterion and with all unlabeled data being used at each iteration.

Self-training is a fairly popular algorithm in the NLP community, with applications to many tasks. In particular, it has been used successfully to improve part-of-speech tagging and named-entity recognition [66] as well as parsing [126, 92, 128, 62, 127]; in the latter case, evaluation is done either directly on parsing [126, 128, 62] or on a downstream task, such as semantic role labeling [127].

In [126], the authors use MAP adaptation with different prior choices (reducing to count merging or model interpolation), comparing results when the adaptation is supervised (i.e. with a small amount of labeled in-domain data) or unsupervised (using the output of a speech recognizer as unlabeled in-domain set). In the latter case, the parser is trained on the Brown corpus portion of the Penn treebank, while the speech domain is Wall Street Journal (WSJ). Wang et al. [146] apply a form of self-training in the sense that the parser for the target domain is trained on automatically-generated trees using a parser trained on the source domain.

In [92], the authors use a PCFG parser with a MaxEnt reranker and find that using the best reranking output yields some minor improvement in the performance of the two-stage setup, as well as that of the parser without reranking. In this work, both the in-domain and the out-of-domain data are news articles, though they use WSJ for the source domain and North American News Text corpus (NANC) as the target. They use only one round of self-training, and find that the adaptation helps the most with medium-length sentences (20-40 words) and with sentences with specific structure (such as improving the handling of coordinating conjunctions). They also conclude that performance of the parser as a language model is improved, though the conclusion appears to be influenced primarily by improvements in the structure of the most likely parse, as opposed to improving the ability to predict the next word in the sentence.

Huang et al. [62] focus on cross-language adaptation, with a parser trained on English treebank as the source domain being evaluated on Chinese data as the target; the Chinese data contains both newswire and broadcast news, whereas the English data is just news text (from the WSJ corpus). They perform a single round of self-training, obtaining improvements from the unlabeled data, weighting the in-domain vs. the out-of-domain treebank to avoid overfitting based on the unlabeled set.

Co-training [25] is another popular self-labeling approach. In co-training, two separate classifiers are trained, using two sets of features known to be independent conditioned on the class labels. The method alternates the automatic labeling of unlabeled data with each classifier, labeling the data with one classifier before adding it to the training set of the other. As with self-training, various stopping criteria can be used, such as a fixed number of iterations or convergence of labeling confidence. Unlike with self-training, where a single classifier is produced in the final stage, with co-training the final classification decisions are produced by a weighted combination of the decisions produced by the two classifiers. Another approach when no natural split of the feature set into two independent views exists is to use two different classification algorithms instead, as shown in [48].

Co-training has also been applied to a number of NLP tasks. For example, co-training has been used to improve the performance of a newswire-trained parser when applied to other domains, such as broadcast news [147] or broadcast news and broadcast conversations [148]. In both cases, a small amount of in-domain treebanked data is used to complement the unlabeled data, and the different views of the data are given by different parsers (the Charniak parser [28] and the Berkeley parser [119]), a method suggested in [48].

The relative performance of self-training and co-training has been discussed in a number of papers. In general, self-training approaches do not make guarantees of convergence, whereas if the two views of the data used in co-training are known to be conditionally independent given the class labels, arbitrarily close to oracle performance can be obtained [25]. In practice, co-training can outperform self-training when two independent feature sets exist, and even when no such split exists, co-training can outperform EM when the features are split randomly between the two views [109]. However, other researchers have found that in datasets with no natural feature split, self-training and EM tend to outperform

co-training, for example Ng and Cardie [108] on a coreference resolution task. Many other NLP researchers have used self-training for various tasks, though without comparing to co-training [66, 126, 92, 62, 127]. Since co-training setups are more complex to implement than self-training, we will focus on self-training approaches in this work.

Feature Representation Approaches

Many feature representation domain adaptation approaches exist; here we focus on methods which induce new features from unlabeled data in both the source domain and the target domain. Such methods often rely on distributional similarity properties of the features (e.g. feature co-occurrence statistics in each domain). Among the feature transformation methods used in NLP tasks, we discuss here singular value decomposition-based methods, such as latent semantic analysis (LSA) [40] and principal component analysis (PCA) [53], as well as methods employing auxiliary tasks, such as alternating structure optimization (ASO) [10] and structural correspondence learning (SCL) [23].

Both LSA and PCA use singular value decomposition (SVD) at their core, and thus they are often grouped together in discussions about change of feature representation methods. However, they learn different representations, and make different assumptions about the data. LSA (also called Latent Semantic Indexing (LSI) in some publications) [40] was originally used in the context of information retrieval, as a way to learn a more robust data for keyword searches. LSA learns a low-rank representation of the term-document matrix representation of the data, such that words which appear in similar contexts are grouped together, using the intuition that co-occurrence is a good indicator of semantic relatedness. On the other hand, PCA [53] applies SVD to the term covariance matrix, after mean subtraction. Thus, the top singular vectors computed by the decomposition represent the directions of highest variance.

Generally, both LSA and PCA are applied to single-domain data. However, a number of works have looked at using SVD-based methods in a domain transfer setting for NLP applications, for example [12] for POS tagging and [12, 60] for chunking, obtaining improvements both within-domain and in a domain transfer setting. Ando [8] uses a method for learning

features using unlabeled data and SVD, finding that such change-of-feature representations with a small amount of labeled data and unlabeled data can outperform fully-supervised methods. In [12], Ando applies the method to a cross-domain setting, where the unlabeled data comes from a different domain (WSJ) than the domain of interest (ACE). In [60], Huang et al. use a smoothed representation of the (labeled) training data using LSA, then apply the HMM trained over this smoothed representation to an external domain, finding significant improvements over the initial representation of the data in this cross-domain setting. Pan et al. [113] apply PCA to sentiment classification in a cross-domain setting. In this work, domain-specific words indicating sentiment polarity are grouped into domain-independent clusters, by using domain-independent words as a bridge. A spectral algorithm (spectral feature alignment) is developed to perform this mapping, using the distribution of links in a bipartite graph to model the co-occurrence between domain-specific and domain-independent words.

An alternative approach to SVD-based methods is learning feature representations using a set of auxiliary tasks for which generating labels is cheap, and which are correlated with the underlying task of interest. The auxiliary tasks are applied to the unlabeled data to extract "predictive structure"; the learned predictive structures are then used for the task of interest. This idea was introduced in [10] for general semi-supervised learning, with applications to text chunking [11] and word sense disambiguation [9]. Structural correspondence learning (SCL), introduced by Blitzer et al. [23] extends the approach of using auxiliary tasks to a domain adaptation setting; here, the auxiliary tasks are learned over both source and target data, using only auxiliary tasks labels (which are easy to compute automatically) but not requiring labels for the task of interest. While SCL does not require target-domain labeled data for the target task, later work applies SCL to sentiment classification for customer reviews [24], using a small amount of in-domain labeled data to correct erroneous feature matches generated during the application of SCL.

Some of the work in this dissertation is related to the auxiliary task-based methods of learning features. In such works (for example [10, 23]), the emphasis is on developing methods using in-domain data which are unlabeled with respect to the task of interest but labeled (often automatically) for a different set of tasks; the latter serve as a proxy for the

task of interest. In such a scenario, if features are developed across both the unlabeled in-domain data and the labeled (in-domain, in the case of ASO, or out-of-domain, in SCL) data that can be shown to improve the auxiliary tasks, then the same features can be used for training more robust systems for the task of interest as well. This is related to error detection methods using syntax presented in this thesis, where we use ASR error detection as an auxiliary task for improving parsing. However, in the approach of this thesis, error detection itself is the goal, not parsing; as such, we treat the parse tree labels as latent, and care about them only insofar as they help us improve error detection.

2.6 Discussion

A survey of previous approaches to improve ASR rescoring and error detection has shown that, while syntax is a component in many such systems, there is room to do more. In particular, most syntax-based approaches integrate syntactic information as features into a discriminative model or through conditioning in generative models, rather than using a parser directly to select among a set of hypotheses. Parse features have also been used in error detection, often with log-linear models, but without modeling errors explicitly in the grammar. The work in this thesis attempts to fill those gaps in the current body of work.

Another gap in previous work relates to the use of a parser trained on mismatched domain data. In particular, for ASR rescoring tasks, improvements have been reported primarily when the parser was trained in matched conditions. In this work, when domain adaptation is used to improve the parser, the objective used for adaptation is almost always parsing accuracy, which requires at least some in-domain parse-annotated data for evaluation. There is an opportunity for studying alternate, weakly task-supervised approaches which do not have such requirements and instead optimize directly for the task of interest.

Finally, while many methods to learn word class features have been used in prior work, comparatively little work has been done to combine long-distance phrase pattern features and word classes, particularly in resource-poor domains. There is, therefore, a gap in the space of existing methods, leveraging weakly-labeled data and injecting external knowledge (either from out-of-domain human labels or from unlabeled data), which will be exploited in this work.

Chapter 3

SYSTEM AND DATA OVERVIEW

In this chapter, we first discuss an overview of the architecture of the overall speech-to-speech translation project project, including components designed by our collaborators. We also indicate where and how work presented in this thesis contributes to the overall system. Next, we present the various datasets used to train and evaluate the new methods presented in this thesis. Finally, we conclude with a short discussion of some of the challenges imposed by the data, which will be addressed in the rest of this thesis.

3.1 Speech-to-speech Translation System Overview

Figure 3.1 shows a simplified version of the speech-to-speech translation system developed together with collaborators on the ThunderBOLT team [14].[1] Each system has two human users speaking different languages; the diagram shows only one direction of the communication, with the "source language" speaker communicating with the "target language" listener. In this project, the two languages of interest are English and Iraqi Arabic.

The acoustic signal from the speaker is transcribed automatically by the ASR component, which is an extension of the SRI Dynaspeak [159] speaker-independent speech recognition system. Two separate systems are run—a Gaussian mixture model (GMM)-based system and a deep neural network (DNN)-based system. Lattices from both systems are converted to confusion networks using the SRILM toolkit [137]. Each slot in the confusion network output by DNN system is augmented with arcs from the corresponding slot in the confusion network from the GMM-based system if the GMM arcs are missing, with the two confusion networks aligned using a dynamic programming alignment. The ASR system is

[1]The full ThunderBOLT team is composed of SRI International, Aix-Marseille University (AMU), Columbia University, University of Rochester, University of Arizona, and University of Washington. The system diagram shown in this thesis presents a simplified workflow which omits some of the system components not directly relevant to the work in this thesis, such as error segment detection and characterization on the merged utterances, word sense detection, and MT error detection.

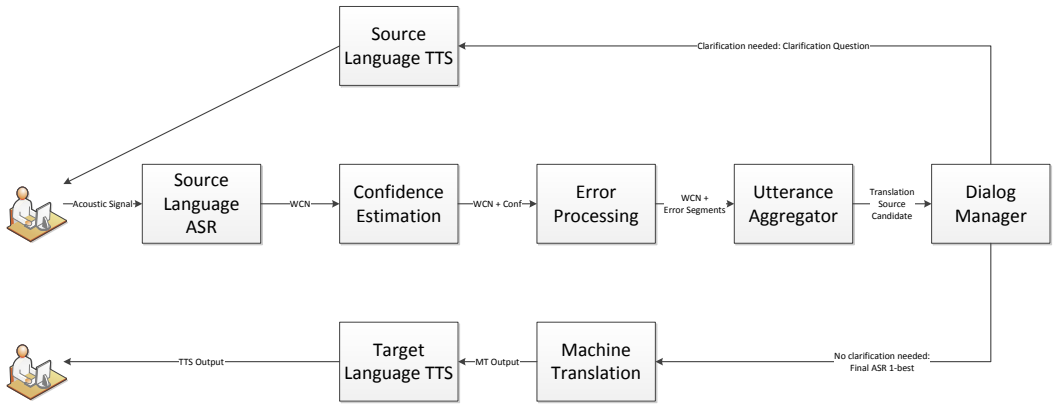

Figure 3.1: Speech-to-speech translation system diagram, one direction.

described in more detail in [142].

The WCN produced by the ASR component is annotated with slot-level confidences, which indicate the posterior likelihood that the top-ranked arc in the slot is in fact correct. The confidence prediction system is also implemented as a 3-layer neural network [142], using features related to the confusion network structure as well as features extracted from a recurrent neural network (RNN) language model [98]. Confidences for slots whose highest-posterior arc is a NULL arc have posteriors deterministically set to 1.0 by this system.

The confusion networks annotated with slot-level confidences are used as input to our component: the error processing system, which is described in more detail in chapter 4. The output of the error processing component is two-fold; we produce a new top hypothesis over the confusion network, as well as additional slot-level annotations, indicating which slots are likely to correspond to error regions. Two types of errors are marked, indicating slots which may correspond to an out-of-vocabulary (OOV) word, or to a name error, respectively. A slot may be marked as containing both error types (in which case the OOV word is taken to be a name OOV), just one, or neither.

The utterance aggregator component accumulates information from successive utterances according to the dialog state and combines them into a single "merged" utterance; the process is repeated until the aggregate ASR hypothesis generated in this way is determined by the system to be suitable for translation. The initial utterance is simply passed

along to the dialog manager component.

The dialog manager is responsible for determining whether to ask the speaker for a clarification, or translate the aggregate ASR hypothesis corresponding to the sequence of utterances obtained thus far. This decision is made using a set of deterministic rules, which take into account the types of errors detected and their confidences [14]. For example, if a name error is detected, the user may be asked to spell the name; if a non-name OOV is detected, the user may be asked to replace it with a different word or, if the error region is found to extend over a large portion of the utterance, to restate the entire utterance. If a clarification is requested, the user's response to the clarification is decoded and processed as a new utterance, which is merged with the previous utterances accumulated in the utterance aggregator. The new translation candidate is evaluated once again in the dialog manager, the process continuing until the system determines that a translation should be made.

Table 3.1 includes a short snippet of an actual dialog using our system from the perspective of the English-side speaker. For each turn of the English speaker (where we define a turn as one speaker's interactions until translation), we show the human speaker's reference transcript and the corresponding system hypothesis, as well as the system's response aimed at correcting any detected ASR errors (if any). The last sentence in a turn has no system response, coming right before the translation step. From the Arabic side we include only the translation at the end of the Arabic speaker's turn, to provide context for the next English-side interaction.

In this example, the OOV name *halloween* is misrecognized in four consecutive utterances. In each case it is replaced by a syntactically-anomalous sequence of shorter words, e.g. *how will we in*. In the first instance of this error, the system asks the user to rephrase it, indicating that the system correctly detected an error in that region, but not that the error type is a name. The second instance is not detected as an error and thus a translation is attempted. In the next two cases, the system asks the user to rephrase the entire sentence, indicating that an error region is detected, but that the extent of the error region is likely incorrect (either too big or not overlapping the true error region).

While the overall system developed for the BOLT project performs two-way translation with source utterances in both English and Iraqi Arabic, all the work presented in this thesis

is done using only English-side data.

3.2 Data

Our experiments use several evolving in-domain speech and text corpora with different train/test configurations, as well as out-of-domain data from two sources. The different in-domain datasets evolve with improving system components and strategies to address the issues of skewed class priors. The out-of-domain datasets are static, but not always well-matched to the target domain data. We describe below the main similarities and differences between the different corpora, as well as the experimental conditions in which they are used.

The target domain of this work is the domain of interest to the DARPA BOLT (Broad Operational Language Translation) project. The spoken data consists of short- to medium-length conversations between English and foreign language speakers. Topics include a variety of scenarios, such as medical aid, military training, and reconnaissance. The human-human interactions are mediated by a dialog system which is capable of asking questions of clarification, as described in section 3.1. In such dialogs with human-computer interactions (i.e. the responses to dialog system requests for clarification or rephrasing), we draw a distinction between *initial* interactions (i.e. human-initiated utterances) and *clarification* interactions (i.e. sentences responding to a computer prompt).

The only truly in-domain data available is the data from the most recent NIST-run evaluation of the project. This consists of complete dialogs from the BOLT Phase 2 dryrun and evaluation, with both initial and clarification responses. We refer to this data as the NIST-Eval set. This set consists of 1879 utterances. We use this set only as a "blind" evaluation set for our systems.

We also have available data from a collection performed early during the BOLT project. This data consists of a set of read utterances on various topics drawn from BOLT scenarios, but which are not part of any human-human dialog. The collection was designed to contain an artificially-high error rate, in particular targeting name errors and non-name OOVs. There are approximately 820 utterances available in this form. The audio for each sentence in the set was recorded by our collaborators at SRI.

Turn	Speaker	Language	Type	Trans	Text
1a	HUM	EN	INIT	REF	do you know about the american holiday halloween
1a	HUM	EN	INIT	HYP	do you know about the american holiday hall wing
1a	SYS	EN	CLAR	HYP	I didn't understand when you said [halloween] Please rephrase or try again
1b	HUM	EN	CLAR	REF	halloween
1b	HUM	EN	CLAR	HYP	hollow wing
2	SYS	IA	TRAN	HYP	do you have a holiday soon will now be
3a	HUM	EN	INIT	REF	the american holiday halloween
3a	HUM	EN	INIT	HYP	the american holiday how will we in
3a	SYS	EN	CLAR	HYP	Please rephrase the entire sentence
3b	HUM	EN	CLAR	REF	no i'm trying to tell you about an american holiday celebrating the dead named halloween
3b	HUM	EN	CLAR	HYP	no i'm trying to tell you about an american holiday celebrating the dead named how will we
3b	SYS	EN	CLAR	HYP	Please say that sentence using different words
3c	HUM	EN	CLAR	REF	no the american holiday is about celebrating the dead
3c	HUM	EN	CLAR	HYP	now the american holiday is a out celebrating the dead
3c	SYS	EN	CLAR	HYP	I'm having trouble Please rephrase the entire sentence using different words
3d	HUM	EN	CLAR	REF	it's american holiday for the dead
3d	HUM	EN	CLAR	HYP	it's american holiday for the dead

Table 3.1: Sample dialog snippet, including English human and system utterances, as well as Iraqi-side English translations.

The rest of the data that we consider "in-domain" comes from a previous DARPA project, TRANSTAC (Spoken Language Communication and Translation System for Tactical Use). As in the case of the BOLT project, the domain of interest to TRANSTAC was that of two-way speech-to-speech translation, with scenarios of interest similar to those used in BOLT. However, unlike BOLT, TRANSTAC interactions did not have the additional human-computer dialog component; the system simply translated each utterance in the human-human dialog. Therefore, each utterance from data recorded for the TRANSTAC project is an initial utterance according to the proposed definition.

Two types of data from TRANSTAC are available. The first source consists of dialogs originally collected for use in ASR training or tuning. Due to the way data was partitioned between ASR training and tuning partitions during the TRANSTAC project, we do not have entire sessions available from all dialogs recorded for ASR training purposes. The second source consists of data recorded during previous NIST-run evaluations for the TRANSTAC project. Such evaluations consisted of either full human-human conversations mediated by the TRANSTAC system, or single sentences of read speech on one of the topics used during the evaluation. We do have complete sessions available for any NIST evaluation consisting of actual dialog interactions (not just read speech). In all cases, we use only the English side of the interactions, allowing us to use conversations between English speakers and conversation partners speaking any of several foreign languages used in TRANSTAC. The TRANSTAC data available to us consists of approximately 8,000 utterances; the rest of the data is used for training and tuning other components of the BOLT system.

For all TRANSTAC and BOLT data, we have available both the ASR output and the hand-generated transcript.[2] Thus, we can evaluate the ASR error rate. One of the goals of the BOLT project is to develop better ways to handle ASR OOVs; thus, all ASR systems developed for the BOLT project have the same fixed, limited vocabulary. We label (and evaluate) the utterances with WCN slot-level ASR error and OOV labels (indicating, respectively, whether the highest-scoring arc in the slot matches the reference word, and whether the reference word corresponding to that slot is OOV).

[2]All older data was decoded with the latest ASR system available at the time when the training data partitioning was performed.

Out of the available TRANSTAC and BOLT data. the SRI BOLT team created two internal training sets, BOLT-P2 and BOLT-P3. Both sets are split 60-20-20 into *train* (used for model training), *dev* (used for hyperparameter tuning), and *eval* (used for reporting results on internal data and for making system selection decisions). The two different datasets reflect the changing nature of the overall project, in terms of available data and hand annotations, as well as team-wide changes in training methodology, as discussed below.

The BOLT-P2 set consists of 1585 utterances, consisting of all the read speech recorded for BOLT, as well as some of the TRANSTAC dialogs that had been used for ASR tuning during that project.[3] This data was processed using the Phase Two evaluation of the BOLT system, which includes older ASR and confidence estimation components. This data was not annotated for name errors. We use this data for experiments related to rescoring and OOV detection only. The 60-20-20 split was used for building the error detection systems as well as to train the confidence predictor described in [142].

The BOLT-P2 data was used for two publications using methods developed as part of this dissertation [87, 88]. The data used in those papers was decoded using an even earlier version of the ASR system; furthermore, the system in [87] did not use features extracted from the confidence estimation described in [142]. Results presented throughout this thesis are therefore not directly comparable to the results presented in those papers, even when the methods are otherwise very similar.

The BOLT-P3 set consists of 8781 utterances and is largely a superset of BOLT-P2; however, a relatively small number of utterances in BOLT-P2 which had also been included in the BOLT ASR system training were removed from BOLT-P3. Data in the BOLT-P3 set was split into five folds used for training a cross-validation setup of the confidence predictor described in [142], so that confidence prediction would be trained independently of OOV and name error prediction, where it is used as a feature. This is a difference from the BOLT-P2 setup. Additionally, two annotators labeled all names in the reference transcripts, with differences in annotations reconciled to generate a single, consistent set of name labels. A randomly selected set of 40% of the names present in the original recognizer

[3]There is no overlap with the ASR tuning data used for BOLT

vocabulary in each of the five folds were held out of the recognizer vocabulary and the data in each fold was decoded using the restricted vocabulary, to provide more examples of errors for training the OOV and name error detection modules.

Corpus	# session fragments	# utterances	# single utterances	# name tokens	# OOV tokens	# name tokens
BOLT-P2	306	1585	820	n/a	631	n/a
BOLT-P3	3646	8781	1450	562	1024	426
NIST-Eval	148	1879	0	85	91	68

Table 3.2: Training corpus session, utterance, and label statistics.

Additional statistics regarding each of the three datasets are included in table 3.2. In this table, by "session fragment" we mean a sequence of one or more contiguous utterances from within the same session; this may be a complete session, as in the case of TRANSTAC data from NIST evaluations, or a partial session, as in the case of TRANSTAC ASR training data. Single utterances are utterances which do not belong to a session, either because they originated in past NIST evaluations whose protocol did not include full dialogs, or because they consist of read speech recorded for the BOLT project targeting specific error types. We include the number of annotated names, rather than name errors, because the errors may change depending on the ASR system used.

We notice a number of differences between the three datasets. The BOLT-P2 data has a very large proportion of single (sessionless) utterances, compared to BOLT-P3; the external evaluation set NIST-Eval has none, since it is composed of complete sessions. The average length of a session fragment in the BOLT-P2 and BOLT-P3 data is around 2-3 utterances.

The OOV and name statistics also vary across the sets. We see that, in the BOLT-P2 set, the OOV rate is very high, with approximately 40% of utterances containing an OOV; this is by design, with many of the sentences recorded by our collaborators at SRI designed to contain OOVs. On the other hand, in BOLT-P3 data the OOV rate was reduced by

removing some of the read speech sentences with OOVs, and by using much more of the available TRANSTAC data, to better match the statistics of the external NIST EVAL set, in which only about 5% of sentences contain OOVs. The OOV rate was, to an extent, artificially increased, in BOLT-P3 data, too, though rather than generating additional read speech sentences, we achieve the higher OOV rate by removing names from the decoding vocabulary before ASR processing.

Looking at name statistics, we see that 75% of names in NIST-EVAL set are OOV; in the BOLT-P3 data, even with some of the names removed from the decoding vocabulary to increase the name OOV rate, only about 40% are marked OOV. Removing even more names from the decoding vocabulary would have brought the BOLT-P3 training set statistics more in line with the external eval set ones; however, this would lead to a higher risk of increasing the false positive rate (and thus decreasing precision).

We make use of three additional data sources for parser training and feature learning. We have at our disposition the entire set of TRANSTAC language model training data (LMTRAIN). This data, like the TRANSTAC speech data, is a close match to the target domain in terms of both style and topic, even if if is not fully "in-domain" because of the different focus between the TRANSTAC project (and thus data) and the BOLT project. The LMTRAIN dataset consists of 1.6 million sentences, representing a mixture of hand-transcribed references of the TRANSTAC ASR training data and written text. We apply uniform sampling to select a subset of about 96,000 sentences for use in parser domain adaptation experiments (LMTRAIN-SMALL), with the downsampling performed for experiment speed. The entire LMTRAIN set was used for training language models in the ASR system used to decode the speech data, and to learn word embeddings and word classes.

As out-of-domain data used for parser training we make use of two treebanks, both drawn from conversational telephone speech (CTS) corpora. SWITCHBOARD is the full Switchboard [47] corpus treebank released by the Linguistics Data Consortium (LDC). We use two versions of this treebank; one where no preprocessing is applied (SWITCHBOARD-RAW) and another where some minimal preprocessing is performed, specifically with punctuation and casing removed (SWITCHBOARD-PREPROC). As an alternative to the full Switchboard treebank, we use a smaller treebank drawn from both Switchboard and the Fisher [39] cor-

pus, also released by the LDC. We refer to this corpus as CTS Small, to indicate that it is general conversational telephone speech (CTS), but smaller than the full Switchboard treebanks.

Corpus	Utterance Count	Ave. Utt. Length	Vocabulary Size	% Filled Pauses	% Repetitions
Switchboard	110505	7.7	16389	3.4	1.5
CTS Small	16519	6.9	5511	2.7	1.3
LMTrain Small	95916	18.1	17728	0.41	0.26
P2-Train	951	10.9	2062	2.3	0.5
P2-Eval	304	11.8	973	2.4	0.5
P3-Train	5306	10.4	3737	2.5	0.6
P3-Eval	1693	9.7	1987	1.9	0.5
NIST-Eval	1879	9.3	1899	0.3	0.1

Table 3.3: Statistics illustrating genre differences between the in-domain speech and text utterances and treebank data.

Some basic statistics related to the style of various corpora used in this thesis are shown in table 3.3. We examine each treebank and the LMTrain sets, as well as the train and eval components of each in-domain training dataset (denoted as P2-Train, P2-Eval, P3-Train, P3-Eval), and the external evaluation set NIST Eval. We observe that the in-domain utterances are, on average, about 9-11 words long. This is longer than the sentences in the two treebanks, but much shorter than the average utterance in the LMTrain data. Since, in general, longer sentences tend to have more complex syntactic structure, we expect that using the LMTrain set for parser adaptation may not be as useful as using the in-domain spoken utterances. We also observe that the disfluency rates are very different among the various datasets. In particular, the rate of filled pauses in Switchboard is much higher than all other corpora; on the other hand, there are very few disfluencies in the LMTrain corpus, and even fewer in the external evaluation set. The latter point is not entirely surprising, since it is known that human-computer interaction is less disfluent than human-

directed speech [131]. The BOLT dialogs, while intended as human-human interactions, are mediated by a computer, hence they are likely perceived as a form of human-computer interactions by the discussants. The rate of repetitions is also much higher in the two treebank corpora than in all the in-domain data, with the NIST Eval set once again displaying a much lower repetition rate than the internal train and eval sets.[4]

Corpus	% voc. overlap, P2-Train	% voc. overlap, P3-Train
LMTrain Small	75	84
CTS Small	53	50
Switchboard	70	73
NIST Eval	41	23

Table 3.4: Vocabulary overlap between the two in-domain speech training corpora and the LM training and treebank corpora.

Table 3.4 shows the vocabulary coverage of each of the two in-domain training sets by each of the treebanks, as well as the LMTrain corpus; we also include the vocabulary coverage for the external evaluation set here. The LMTrain set covers a significantly larger proportion of words in the two in-domain training sets than the smaller out-of-domain treebank; the advantage over the vocabulary coverage of the full Switchboard treebank is reduced, but still present. Here by vocabulary coverage we mean the proportion of words in each training partition (e.g. P3-Train) which is also present in each of the treebank or LM corpora (e.g. CTS Small). We hypothesize that reduced vocabulary coverage may lead to diminished parsing performance, due to the OOV words (with respect to the treebank vocabulary) in the in-domain spoken corpora receiving incorrect POS tags and thus skewing the distribution of higher-level constituents as well. With respect to the vocabulary coverage in the external evaluation set, the relatively low coverage shows that many of the lexical features learned on the two training partitions, P2-Train and P3-Train, may not even

[4]For purposes of this discussion, we consider only single repeated words, not repeated phrases.

be present in the test set NIST Eval, thus leading to poor classification performance if high-weighted features are missing from the test data.

Dataset	partition	1-best WER	oracle WER
BOLT-P2	dev	13.8	8.9
	eval	14.6	10.1
BOLT-P3	dev	12.7	7.8
	eval	11.6	7.3
NIST Eval	eval	5.8	3.7

Table 3.5: Baseline ASR performance.

We include statistics about the baseline ASR performance in table 3.5. We see that, in general, the performance of the two internal datasets BOLT-P2 and BOLT-P3 is similar, though P2 is slightly more difficult; this phenomenon holds for both the dev partition and the internal eval partition. On the other hand, the external NIST Eval set is much easier than both. Comparing the error rate of the original ASR hypothesis with the error rate of the best possible hypothesis through the confusion network (which we denote as *oracle WER*[5]), we find that in all cases, there is significant room for improvement, with 2.5-4% absolute WER gain possible. The absolute gain is smaller for the NIST Eval set, due to the lower starting point, though the relative gain is larger.

3.3 Discussion

In this chapter, we have introduced the overall speech-to-speech translation system aided by clarifications, designed with collaborators as part of the DARPA BOLT project, and discussed how the ASR rescoring and error detection system that will be presented in this thesis integrate within the full BOLT system. We also reviewed the available data sources. Both in-domain data sources and out-of-domain resources have been reviewed.

[5]In both cases, we compute the WER with errors due to contractions and common spelling variations in compound words ignored; for oracle WER the removal of such errors is performed after selection of the optimal path through the confusion network.

The constraints of the available data help direct and focus the scope of this thesis. In particular, we identify two challenges, related to the lack of in-domain hand-annotated parses and the limited amount of in-domain speech data available for system training. Both are discussed in more detail below.

3.3.1 Parser Domain Mismatch

Few conversational domain English treebanks are available, and even those that exist, such as the CTS SMALL and SWITCHBOARD treebanks, display major domain differences to the target domain data for this work (i.e. the BOLT data). These differences manifest themselves both in terms of topic (as evidenced by the vocabulary overlap statistics shown in table 3.4) and in terms of style (as shown in table 3.3). In particular, sentences in the NIST-EVAL corpus are longer than the sentences in the two treebanks, but contain a much smaller proportion of filled pauses and repetitions. An additional difference is that the parser is trained on reference text, whereas the test data is in the form of ASR confusion networks. It is expected, therefore, that the parser trained solely on out-of-domain data will have poor performance when applied to the in-domain NIST-EVAL data. This motivates the use of domain adaptation techniques to improve the parser, with use of both LM training data (which matches the target domain data in terms of topic coverage and disfluency rate) and speech training data (of which there is less, but which is better matched in terms of structure, including the presence of ASR NULL arcs).

3.3.2 Speech Training Data Sparsity

As shown in table 3.4, the vocabulary overlap between the NIST-EVAL set and each of the two speech training sets is limited. Thus, if error detection classifiers are trained using the speech training sets, most of the lexical features are unlikely to be found in the test data. Conversely, many of the lexical features found in the test data are likely to be absent from training and thus will not be used during classification. This motivates the work on learning class-based lexical features, with a dual goal of improving feature reliability during training and feature coverage at test time.

Chapter 4

INITIAL RESCORING AND ERROR DETECTION SYSTEMS

In this chapter, we introduce the overall architecture of the new ASR rescoring and error detection systems used in this thesis, and present a set of experiments that detail the baseline system performance. We discuss the architecture of the system in section 4.1. Baseline experiments are shown in section 4.2; we also comment on different error measures here. The questions we would like to answer with these experiments include:

- How much benefit to ASR rescoring do we obtain from using the parser as a language model?

- In error detection, is it more effective to use the parser as a classifier, or to use it as a feature generation step?

We will address these questions through various experiments, and comment about differences due to training set differences (in terms of size and domain match). Initial answers to these questions are discussed in section 4.3, motivating the subsequent improvements in this work.

4.1 System Overview

Our system takes as input the output of a speech recognizer, in the form of word confusion networks. We employ a three-stage architecture, as shown in figure 4.1. An initial slot-level classification stage augments the WCN with error arcs, where the arcs may represent an error of any type, or errors of specific types, such as OOVs or names. We henceforth refer to all such cases as error arcs for simplicity. A parser stage identifies the optimal path through the WCN according to a parser model without error rules (i.e. with a parser model which cannot select any path through the confusion network which contains error arcs), and refines the error predictions using a second parser model which explicitly models errors and thus can generate error arcs. The dotted line indicates that the error-aware parser model

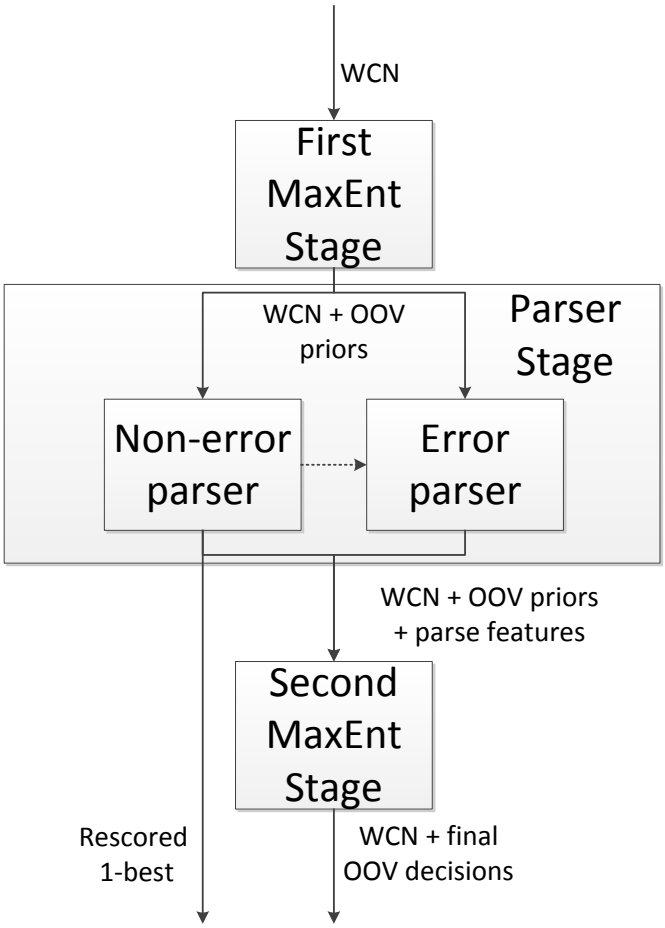

Figure 4.1: Error processing system diagram.

is a modified version of the error-free parser model, adding the ability to generate error regions. The last stage performs a second round of slot-level classification, with additional features obtained from the parser output. The output of the system is a WCN annotated with errors and rescored to obtain a new best path through the structure. Each stage is described in further detail below.

The system architecture is similar to an earlier version designed with collaborators Tom Kwiatkowski, Luke Zettlemoyer, and Mari Ostendorf [87] but it has been reimplemented with some modifications.[1] Results in this thesis are based on a more recent recognition system than those published in [87] and therefore are not directly comparable. The last stage classifier is also similar to the use of a conditional random field with syntactic language model features [114] for OOV detection, with the sequence information modeled implicitly by the syntactic features, instead of explicitly through the sequence of states in the CRF.

4.1.1 Initial Classification

The first-stage classifier predicts, for each slot in the WCN, whether that slot aligns with an OOV word. This stage adds an OOV arc $\bar{w}$ to each WCN slot, generates OOV confidence $p_i(\bar{w}|\theta_{\text{ME}})$ for slot i using the maximum entropy model with parameters θ_{ME}, and renormalizes the confidences of all the other arcs in slot i so that the set of posteriors sums to one. For maximum entropy classification we use a classifier based on the MALLET package [91]. We refer to this initial classification stage as the *first MaxEnt stage* or *first MaxEnt classification stage* throughout this thesis.

The MaxEnt classifier with parameters θ_{ME} uses features that describe the distribution of arcs in the confusion network slots, as summarized in table 4.1. Several features are motivated by the observation that the confusion network slots with more arcs tend to correspond to erroneous regions. The *nullArc* feature signals a slot where the ASR system assigns the highest probability to a NULL arc (i.e. a skip). The *highPost* feature is the

[1] I was primarily responsible for developing the three-stage process described in the paper, designing confusion network features, and running the experiments detailed in the paper. The initial version of the log-linear parsing adaptation algorithm developed by Tom Kwiatkowski was re-implemented and extended in this thesis. The grammar changes developed jointly with the other authors to support confusion network and error parsing were altered and improved upon in this thesis as well.

highest confidence ASR prediction. The *nullArc* and *highPost* features are extracted from the current slot as well as the previous and subsequent two slots, thus generating five distinct features for each slot, respectively. We also use the output of the confidence estimation system *nnConf* and a set of features indicating whether the DNN and GMM ASR systems disagree in the current or neighboring slots (*hasGMM*, *leftGMM*, *rightGMM*, respectively).

Feature	Description
sentLength	length of sentence (in words)
sentPos	position of slot in sentence
mean	mean of slot arc posteriors
stdev	standard deviation of slot arc posteriors
highPost	highest posterior in slot
highLength	length of highest posterior word in slot
nullArc	1 if highest posterior arc is NULL
nnConf	value of confidence estimation posterior output
hasGMM	1 if the DNN and GMM ASR disagree in the current slot
wordClass	1 for word class corresponding to highest posterior word

Table 4.1: First MaxEnt stage confusion network features.

In addition to the confusion network structural features, we also employ a set of features motivated by the observation that a single OOV word is often replaced by a sequence of shorter, common words. We capture the word length in the *highLength* feature. Lexical information could be captured through word identity features, at the expense of increasing the feature space dimension; to alleviate the dimensionality increase, we instead capture lexical information through a set of word classes. We learn 1000 Brown classes [27] using the full LMTrain corpus. After the 1000 classes are generated, we extract each of the 50 most frequent words in the LMTrain corpus into their own separate class, under the assumption that common words have more predictive power when used as individual features instead of within word classes. Then, for each slot, we find the word class of the highest

posterior word in that slot and set the binary feature for that word class equal to 1.

Feature selection is performed using the criterion of mutual information with the class variables. The decision threshold and the optimal number of features are selected to optimize the slot-level F-score on a development set.

4.1.2 Parsing Stage

We employ a factored parsing algorithm based on the work presented by Klein and Manning [71] and implemented in the Stanford parser, modified to parse word confusion networks rather than single sentences. In this parsing framework, a lexicalized parse tree L can be viewed as having two components - an unlexicalized constituent (i.e. phrase structure) tree T and a dependency tree D which captures the lexical information. Thus, the probability of a lexicalized tree L given a sentence s and model parameters $\theta = [\theta_c, \theta_d]$ is:

$$P(L|s,\theta) = P(T,D|s,\theta) = \frac{1}{K} f_c(T,s,\theta_c) f_d(D,s,\theta_d) \tag{4.1}$$

where we factor the distribution into separate components f_c and f_d that generate the unlexicalized constituent tree T and the lexicalized dependency tree D, respectively, with K as a normalizing constant to obtain a valid probability distribution. In practice, for a single sentence s, the fast factored parser implementation employed in this thesis first uses an unlexicalized PCFG model $f_c(T,s,\theta_c)$ to generate n-best phrase tree candidates $T_1, \ldots, T_n$. Then, each candidate tree is rescored using the lexicalized dependency model $f_d(D,s,\theta_d)$, where each dependency tree D_i is obtained from the corresponding constituent tree T_i using a standard head-finding algorithm [36]. The constituent parsing framework allows us to assign syntactic categories to error regions detected by the parser; the separation of the unlexicalized PCFG model and the lexicalized dependency model allows for easier adaptation of the unlexicalized PCFG model to add error region rules.

Parsing word confusion networks is similar to the approach taken by Chelba for parsing general lattices [30, 125]. However, where parsing a general lattice may produce sentences of different lengths, the regular structure of the confusion network simplifies the process somewhat. Each slot in the confusion network corresponds to a cell in the lowest diagonal of the chart, with any word (or arc) in the confusion network slot being able to fill that

cell. The n-best trees over the confusion network may use as terminal nodes any sequence of arcs in the confusion network, instead of just the 1-best path.

Parsing full word confusion networks adds additional complications not present in parsing single hypotheses (e.g. each hypothesis in an n-best list). In particular, the parser needs to account for NULL markers used to represent ASR hypotheses of different lengths. The base PCFG grammar G used in the constituent parser framework has as its basis a set of unary and binary context-free rewrite rules. To these we add a single pair of binary grammar rules:

$$\text{X} \rightarrow \text{X NULL} \tag{4.2}$$

$$\text{X} \rightarrow \text{NULL X} \tag{4.3}$$

which allow the NULL region to attach to any constituent X in the grammar. We note that maintaining both left- and right-attachment rules is required in order to handle NULL slots as the first or last slots in the WCN. Additionally, the ability to attach a NULL region to neighboring WCN slots on either side may potentially lead to a more robust model than using only left- or right-sided attachment rules, since now the parser could generate a tree with either structure, depending on which side yielded a more commonly-seen derivation.

To generate the terminal corresponding to the NULL arcs in the confusion network, we use a single rule:

$$\text{NULL} \rightarrow \mathsf{null} \tag{4.4}$$

We note that this set of rules does not generate a single NULL region if multiple consecutive null arc slots exist in a confusion network. Instead, we obtain a structure where each null arc is a branch of a recursive structure of the same non-terminal. We illustrate this in figure 4.2. As before, red indicates the best path through the WCN according to the WCN posteriors, and blocks indicate the words selected by the parser. We see that the subtree spanning the three slots has syntactic category NN, with the rule

$$\text{NN} \rightarrow \text{NN NULL} \tag{4.5}$$

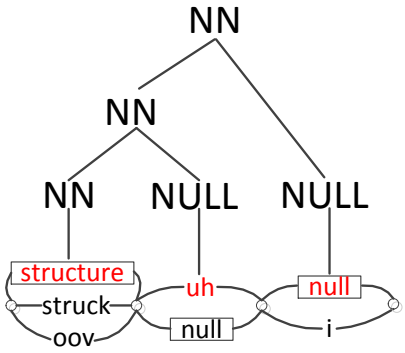

Figure 4.2: Subtree illustrating nested attachment of NULL rules

being invoked recursively to generate the two null arcs.

We employ a second grammar modification to explicitly model error regions. Each constituent X can thus generate an error region using a new rule:

$$\text{X} \rightarrow \text{ERROR} \tag{4.6}$$

Two additional rules are used to grow error regions using a Markov process:

$$\text{ERROR} \rightarrow \text{ERROR ERROR} \tag{4.7}$$

$$\text{ERROR} \rightarrow \text{ERR} \tag{4.8}$$

These rules allow us to generate error regions of arbitrary length with a corresponding arbitrary syntactic category. All the new rules that generate confusion network NULL and error arcs are given a small default score relative to the weight of the grammar rules learned from the treebank distribution.

We parse the confusion network output by the slot level classifier using a probabilistic context free grammar G. This generates the single most probable constituent parse T^* of a sequence of edges $\mathbf{w} = [w_0, \ldots, w_{|\mathbf{w}|-1}]$ representing a path through the confusion network. Each parse T is scored according to G and the first stage arc posteriors.

$$T^* = \arg_{\{T, \mathbf{w} \in \text{WCN}\}} \max f_c(T, \mathbf{w}, \theta_c) \prod_{i=0}^{|\mathbf{w}|-1} p(w_i|\theta_{\text{ME}}) \tag{4.9}$$

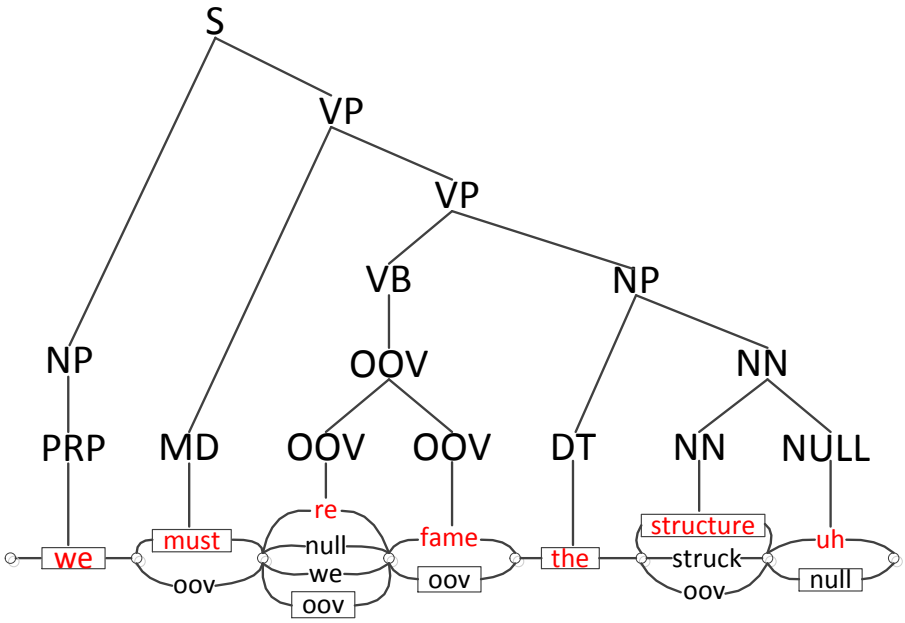

Figure 4.3: Example of the best parse of a WCN for the utterance "We must reframe the structure", where word "reframe" is OOV; words in boxes correspond to the path through the WCN selected by the parser.

The one-best parse T^* of the WCN is used as an error classifier by allowing a path that assigns error arcs as terminals in $\mathbf{w}$. In the factored parser framework, we first produce the n best constituent trees $T_1, \ldots, T_n$ according to the procedure above. Each tree T_k has a corresponding path $\mathbf{w}_k$ through the confusion network WCN. The best tree T^* then is that with the best score from the dependency model $f_d(D, \mathbf{w}, \theta_d)$, where D is the dependency tree corresponding to the constituent tree T, and $\mathbf{w}$ is the corresponding path through the confusion network.

An example parse generated using a parser model augmented with OOV rules is shown in figure 4.3. In this figure, error arcs on the WCN slots are represented by the string *oov*, and the error constituent is indicated with *OOV*. The path through the WCN selected by the parser is indicated by the boxed words, while the best path according to the WCN arc posteriors is indicated by the red words. This parse has correctly classified the second and third slots as representing an OOV region in the original string, and assigned to the

OOV region the syntactic category VB, thus producing a parse that is consistent with the grammar G.

We make use of two distinct parser models. The first, which contains the NULL arc rules but not the error region rule additions, is used to perform ASR rescoring. We will subsequently refer to this model as the *error-free* parser. The second model contains both NULL arc rules and the error rules, and is used to perform error region detection. We refer to this model as the *error-aware* parser. Together, the two parser models are used to generate new features for a separate classification stage, as described in section 4.1.3 below.

4.1.3 Classification with Parse Features

We employ a second round of maximum entropy classification to refine the error predictions, integrating additional features extracted from the parser output with the baseline features described in section 4.1.1. We refer to this second classifier as the *second MaxEnt stage*, to distinguish it from the initial MaxEnt classifier with no parse features.

We employ two sets of features which capture information about the reliability of the parse structure. In particular, we focus on extracting features which capture differences between a tree produced by the error-free parser and one from the error-aware parser, under the assumption that the local structure in error-free regions of the utterance should be similar in the two cases, but the structure in error regions should be different.

Dependency Tuples

We use dependency information to compare the local structure of the two trees. Given a constituent parse tree for an utterance, we obtain a dependency parse tree by running a standard head-finding algorithm [36]. The dependency tree structure allows us to construct a tuple for each slot in the confusion network. Each tuple thus contains the following elements:

- the word selected by the parser for that slot
- the word selected by the parser for the slot acting as its head

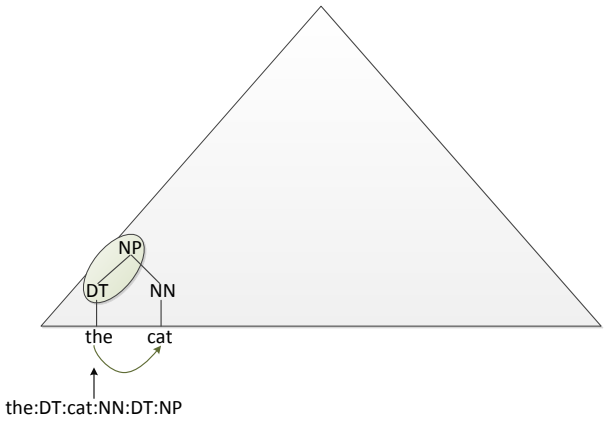

Figure 4.4: Example of the dependency tuple for one slot in a WCN, as extracted from an error-free tree.

- the syntactic category for each of the two words
- the root of the largest subtree spanning the slot but not its head
- the root of the smallest subtree containing both the slot and its head

For each slot, we compare the two dependency tuples extracted from the two parser models by counting the number of common elements.

An example of a dependency tuple is shown in figure 4.4. Here, the determiner *the* is governed by the noun *cat*. The relationship between them is obtained from the tree edge *NP–DT*, where the former is the root of the smallest subtree covering both the target slot and its head slot, and the latter is the largest subtree covering the target but not the head. As actual features we use various binned versions of the tuples, with the POS tags and syntactic categories at the attachment point, or the pair of constituents alone; in the example shown in figure 4.4, these features would be *DT_DT_NP_NN* and *DT_NP*, respectively. We also include two numeric features, counting how many of the tuples are different between the error-aware and error-free parse trees, *relOnlyDiff* (taking into account differences only in the constituent pair at the attachment point) and *POSRelDiff* (taking into account the attachment point constituents as well as the POS tags of the two words).

Inside Score Pairs

We use inside score information to capture the confidence of the parser regarding the structure surrounding a given WCN slot. For each slot, we are interested in measuring the performance of the error-free parser both in the vicinity of that slot (the local structure) and on the periphery of the error region it corresponds to, if any (the structure overlapping error region boundaries), with the intuition that a difference in confidence between the two parsers in corresponding regions will provide information about whether to trust the error predictions made by the error-aware parser.

We compute two inside score-based features:

- local score: the inside score of the smallest non-trivial subtree (i.e. with root higher than pre-terminals) in the error-free tree, with a word in the current slot as one terminal in the tree;

- boundary score: the inside score of the smallest subtree in the error-free tree that contains the current slot and an error region boundary slot, minimizing over all the boundary slots.

We present an example of inside score feature pairs in figure 4.5. The subtrees corresponding to the phrase "are condemnable", with the word "condemnable" being an OOV, are shown. The word "condemnable" corresponds to the last three slots in the confusion network, which the error-aware parser models as an OOV region; the error-free parser models that same region as a noun phrase followed by an adverbial phrase.

To compute the inside score features for the third last slot (corresponding to the word "condemn" in the error-free tree), we take the smallest non-trivial subtree containing it (in this case, the second-level "NN" node); this node has relatively low weight, thus indicating that the error-free tree may contain incorrect structure in this region. On the other hand, the boundary score for this slot is given by the inside score of the "VP" node covering the entire region; its higher value indicates increased confidence, thus suggesting that the extent of the subtree is more likely to be correct.

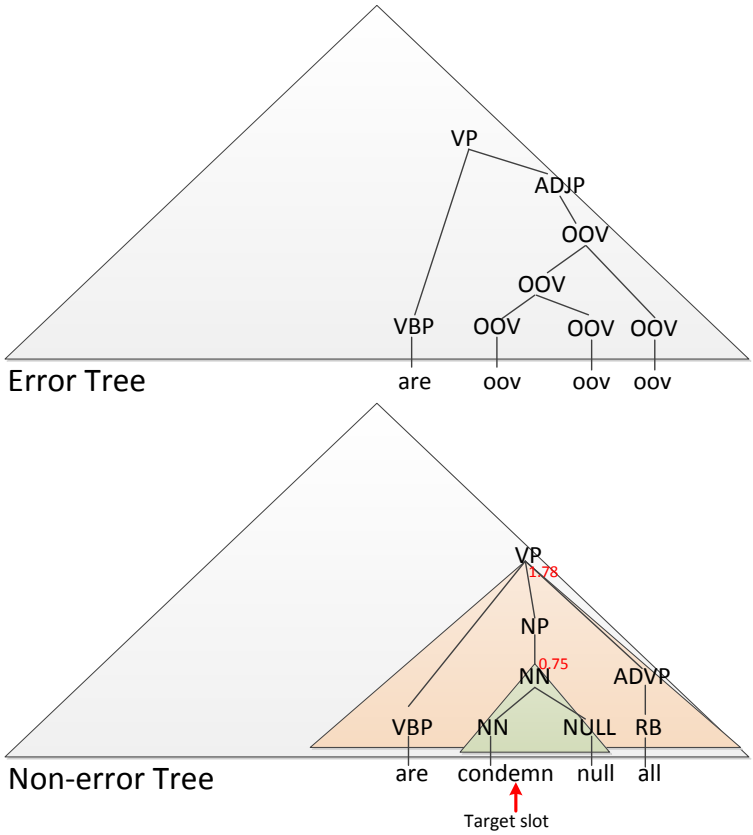

Figure 4.5: Example of the inside score features for one slot, as extracted from two parallel error-free and error-aware trees.

4.1.4 Error Detection Labeling and Evaluation

Marking error labels on ASR output is non-trivial when the output format is more complex than a single hypothesis. The labeling task is particularly difficult when dealing with compound entities, such as multi-word names, when one word of the name may be recognized correctly, but not the rest. There is no agreement in the community for how to label and score in such conditions, with methods ranging from scoring at the frame level [122] to scoring based on error region overlap, giving full credit when even a tiny amount of overlap exists [81] or taking into account the amount of overlap [75]. We will use a hybrid approach in this thesis, allowing for overlap in scoring while also taking into account the amount of overlap, as described below.

We perform ASR error labeling at the confusion network slot level. With general ASR errors, this requires alignment of references to the confusion networks, marking each slot whose highest-confidence arc does not match the corresponding reference word as being in error. OOV labeling is performed by marking each slot aligned to a reference word not present in the ASR vocabulary as being an OOV error. Names in the reference transcript are hand-labeled (taking into account the full utterance context) by two annotators with annotation decisions reconciled in the final version of the data. The name labels are propagated to their respective confusion network slots post-alignment with references. In both cases, slots which neighbor error label slots and whose highest-posterior arc is a NULL arc are also labeled as errors. Error regions thus grow to encompass all neighboring NULL arc slots. This approach violates to an extent the label independence assumption made by maximum entropy classifiers.

Scoring error detection is dependent on the needs of downstream applications; however, there is no single metric which provides optimal benefits to all downstream applications. For example, when the goal is spoken term detection, the focus is often on recall to the expense of precision. On the other hand, when detected errors drive system-initiated interactions with the system users, too-low precision can be just as important as too-low recall. Additionally, different ASR output formats lend themselves to different scoring strategies. We outline the approach used in this thesis here, including a new scoring strategy developed as part of

this dissertation.

In scoring error detection performance over confusion networks, it is of interest to evaluate both whether detected error regions are found to overlap with the true error regions in the confusion network, and whether their extent is accurate (i.e. how much of the detected error region is in fact an error). One approach to this is to score each confusion network slot independently. We refer to this as *slot-level* scoring. While very simple, this method has the disadvantage of giving potentially significant weight to NULL arc slots. As an alternative, we can score only non-NULL arc slots, which we refer to as *word-level* scoring (since the non-NULL arc slots are exactly the 1-best path through the confusion network). A third option is to combine adjacent words into a single error "region", thus counting compound names or OOVs as a single instance for purposes of scoring. This also combines adjacent incorrectly-labeled samples into a single false positive. We refer to this approach as *region-level* scoring.

We illustrate the difference between *slot-*, *word-*, and *region-level* scoring in figure 4.6. We see that at the slot level, there are 4 slots incorrectly labeled as containing an OOV error, three correctly identified as errors, and 6 correctly identified as not errors. Thus we obtain a slot-level F-score of 0.6 (with a recall of 1 and precision of 0.43). Moving to the word level, we see that two out of the four incorrectly-labeled error slots are NULL arc slots, as well as two of the three correctly-labeled ones. Thus we obtain word-level precision of 0.33, perfect recall, and thus F-score of 0.5. At the region level, the incorrectly-labeled slots are all grouped in a single region, as are the correctly-labeled slots, leading to precision of 0.5, perfect recall, and thus F-score of 0.75. However in this case we also take into account the modified word error rate metric, WER*, as described further below.

With region-level scoring we incorporate arbitrarily long sequences of confusion network slots into a single sample for scoring purposes, so we must separately evaluate the extent of the error regions. In particular, we note that the extent of the error regions detected also plays a role in the quality of the interactions between users and the system; for example, if the detected error region is too large, the speaker may be asked to rephrase a significant portion of the utterance; repeated instances of this phenomenon may lead to user fatigue or annoyance [138]. On the other hand, if parts of the true error region are left undetected,

Slot:	0	1	2	3	4	5	6	7	8	9	10	11	12
Ref:	they	fail	to	*NULL*	*NULL*	comprehend	that	their	acts	are	*NULL*	*NULL*	condemnable
1-best:	they	fail	to	company	*NULL*	him	that	their	acts	are	condemn	of	all
Label:	0	0	0	0	0	0	0	0	0	0	1	1	1
Prediction:	0	0	0	1	1	1	1	0	0	0	1	1	1

Figure 4.6: OOV error scoring example

when the user is asked to rephrase the portion of the utterance that is in error, the incorrect error region boundaries will lead to user confusion, and the correction provided by the user will likely be insufficient to address the underlying problem with the initial transcript.

To evaluate the error detection strategies, the main evaluation metric used is *F-score* (sometimes referred to as *F-1 measure*), the harmonic mean of precision and recall. The F-score captures the successful detection of error regions when there is overlap between a true error region (i.e. the slots labeled as error according to the reference transcript) and a sequence of slots that had been marked as errorful by the error detection systems. We treat a slot as a minimal unit. Thus, in *slot-level* scoring, a correct decision is made when both the predicted label and the true label are marked as positive. The same is true of *word-level* scoring, the only difference being that we do not count NULL-arc slots for purposes of scoring. On the other hand, in *region-level* scoring, a region can have variable length, and thus overlap between the reference error region and the predicted one can be partial. We allow such partial overlap to count as a true positive for purposes of computing precision, recall, and F-score metrics.

While F-score allows for comparisons of two different systems at their optimal operating point, it does not allow for comparisons of system performance over a range of possible operating points. Additionally, primarily in regions of low recall, precision (and thus F-score values) are often noisy, since the denominator in the precision computation can

change significantly even with only minor differences between two decision thresholds. Using receiver-operator characteristic (ROC) curves would allow us to compare the different systems at different operating points, but do not alleviate the noise in high-precision regions. Instead, detection error trade-off (DET) curves can be used, which do not suffer from the same potential noise problems. Thus, both methods are useful—F-score for tuning hyperparameters, such as regularization or feature selection thresholds, and DET curves to select the final operating point for a particular system configuration given an error type constraint. We can also use DET curves to manually select the best configuration among multiple systems (e.g. to compare different training data splits or different feature sets).

For *region-level* scoring, we measure the extent of error regions by employing word error rate as a secondary metric. We use a modified version of WER computation, with words belonging to error regions (both in the reference transcript according to the true labels, and the ASR hypothesis as per the detected error regions) being replaced with a single ERROR token. We call this modified WER metric WER*. In general, for a given utterance, WER $>$ WER*, since error region words will no longer be considered errors if the error region is detected correctly. However, for error regions which are either too large or too small, replacing the reference and hypothesized words with the ERROR token will maintain the errors, thus allowing us to measure, for instance, the impact of removing correctly-transcribed words when detected error regions are too large, or not removing incorrect transcriptions when the error span is too small.

4.2 Initial Experiments

We perform a set of experiments to evaluate the performance of the parser when used for ASR rescoring and in the multistage system for error detection. We are interested primarily in the effect of parsing when no adaptation is performed. We also perform a few analysis experiments to investigate the effectiveness of different hyperparameter and system tuning strategies.

4.2.1 Rescoring for In-Vocabulary Error Correction

We first perform experiments to gauge the impact of parsing for ASR rescoring when the parser is trained solely on out-of-domain data. We compare the best hypothesis predicted by a parser trained on each of the three treebanks, respectively, against the baseline hypothesis obtained directly from the highest-scoring path in the confusion network without any rescoring. Results for the BOLT-P2 and BOLT-P3 dev and eval sets, as well as the NIST Eval set are included in table 4.2. In all cases, we see that the parser hypotheses yield higher error than the WCN 1-best hypotheses. Among the three treebanks, we see that the CTS Small treebank outperforms the preprocessed as well as the original-form Switchboard treebanks, despite its smaller size, though the differences are relatively small when compared to the performance drop against the baseline 1-best hyps. We do not see any gains from the preprocessed version of the Switchboard treebank, which is surprising, since the pre-processing makes the data better matched to ASR output. In fact, we notice a performance degradation in four of the five test sets.

Treebank / Dataset	no parsing (WCN baseline)	CTS Small	Switchboard	
			no preproc.	preproc.
P2-Dev	13.8	15.1	15.6	15.7
P2-Eval	14.6	16.5	17.1	17.3
P3-Dev	12.7	15.0	15.3	15.6
P3-Eval	11.6	14.1	14.5	14.7
NIST Eval	5.8	7.1	7.4	7.3

Table 4.2: WER results for WCN rescoring using the baseline parsers.

4.2.2 OOV Detection

Since the CTS Small treebank outperformed the Switchboard treebanks for rescoring, we focus on that parser configuration for the OOV and name error detection experiments. In this section, we present experiments for OOV detection. We discuss tuning strategies in

experiment 1. Experiment 2 presents results for OOV detection at each stage of the error processing pipeline. A discussion of tuning for F-score vs. using DET curves is presented in experiment 3.

Experiment 1: OOV Detection Tuning

The MaxEnt classifiers generate slot-level decisions, with each slot being treated as a separate sample; thus, classifier hyperparameter tuning (e.g. to perform feature selection) and operating point selection is performed at the slot level as well. However, when we report final scores we have a choice of whether to generate scores independently for each slot, or whether to use a higher-level metric, such as region-level scoring, where contiguous regions of the WCN marked as OOV are combined into a single sample and NULL arcs are ignored. We are interested in the relationship between slot- and region-level scoring; if the two metrics perform similarly, we can use the finer-grained samples (i.e. slot-level) for tuning, to avoid the cost of generating error regions and computing overlap, while reporting results on the aggregate samples (i.e. region-level), which better match with downstream application uses, without losing performance from the mismatch in scoring strategies.

We include the results for the first MaxEnt classification stage for OOV detection using the BOLT-P2 data in table 4.3 (results using the BOLT-P3 data, not included, are similar). We see that, in all cases, scoring at the region level yields higher precision and recall than scoring at the slot level. Slot-level recall is significantly lower than region-level recall; this is primarily due to NULL arc slots near OOVs, which skew the slot-level statistics. The performance trends show that higher slot-level results correspond to higher region-level results for precision, recall, as well as F-score. In future experiments, we will report only region-level F-scores together with WER* results, but continue to tune the MaxEnt classifiers on slot-level scores.

Experiment 2: OOV Detection, Three-Stage Setup

In the next experiment, we compare the performance for the OOV detection task using the BOLT-P2 and BOLT-P3 data for training and each of the first MaxEnt stage classifier,

Fileset	Scoring Level	Precision	Recall	F-score
P2-DEV	Slot	41.1	54.4	46.8
	Region	49.4	72.9	58.9
P2-EVAL	Slot	45.5	52.2	48.6
	Region	57.0	75.0	64.7
NIST EVAL	Slot	19.1	50.4	27.7
	Region	20.7	59.6	30.7

Table 4.3: Different scoring levels, first MaxEnt stage for OOV detection.

the CTS SMALL-based parser with OOV rules with default weights, and the second MaxEnt stage classifier with parser features added for each training configuration, using both the WER* metric and F-score. We note that we use the same parser for both BOLT-P2 and BOLT-P3 configurations, since no adaptation is performed; however, the MaxEnt classifiers (both first and second stages) are trained separately for each dataset, with results on the shared NIST EVAL data reported for each training condition.

Results comparing the different stages are shown in table 4.4. In both training configurations, the parser performs worse than the first MaxEnt stage in terms of both WER* and F-score. The performance degradation is significant, with WER* differences of over 30%, and F-score drops of 7-10 points. However, the parser-derived features yield some slight improvement over the first MaxEnt stage, though the differences are not very large. We notice that the eval sets are in fact easier than the corresponding dev sets, for both BOLT-P2 and BOLT-P3 data. This suggests that test set variation is substantial. We do observe some overfitting when adding extra features to the MaxEnt classifier, given the larger improvement between the MaxEnt stages (for example, using the BOLT-P2 training configuration, F=58.9 to F=62.6 for P2-DEV vs. F=64.7 to F=65.7 for P2-EVAL). Comparing the performance on the NIST EVAL set when using the BOLT-P2-trained system vs. the BOLT-P3 version, we notice significant improvement from the larger training set in BOLT-P3, in both MaxEnt stages, as well as a smaller improvement in the parser stage

due to the improved initial OOV predictions. Results in table 4.4 will serve as baseline for parser adaptation experiments for OOV detection in chapter 6.

Stage / Dataset	first MaxEnt		parser		second MaxEnt	
	WER*	F-score	WER*	F-score	WER*	F-score
P2-Dev	12.8	58.9	14.4	51.4	11.4	62.6
P2-Eval	12.6	64.7	14.6	54.2	12.2	65.7
NIST Eval	6.3	30.7	10.0	22.6	6.3	31.5
P3-Dev	13.4	26.8	19.8	20.5	13.3	30.0
P3-Eval	12.2	33.6	18.2	21.8	12.2	36.0
NIST Eval	5.8	36.2	9.0	24.7	5.7	38.6

Table 4.4: OOV detection, CTS Small treebank with no parsing adaptation, with results showing the impact of each stage of processing.

To determine the impact of the parse features, we examine the features with the highest absolute weight in the first and second MaxEnt stages for the BOLT-P2 system. We find that, for the first stage system, the *nnConf* feature gets the highest weight, being correlated with the negative class; this is not surprising, since we expect the confidence estimation system to produce high confidence for correct words. Other features associated with the negative class are obtaining high posteriors from the ASR system in neighboring slots (*leftPost* and *rightPost*) and having a high mean arc posterior in the current slot (*mean*). On the other hand, features associated with the positive label (i.e. being part of an OOV region) include the GMM features (*rightGMM, leftGMM*) which indicate that the DNN and GMM ASR systems produced different results, and the sentence position feature *sentPos*.

In the second MaxEnt stage, with parse features added, we find that the various baseline features with high weight in the first stage system also get high weight; however, some of the parse features are also of relatively strong importance. A number of dependency tuple features associate strongly with the negative class: *VBZ_S_NP_NN, NNS_NP_NN_NN, NN_NP_NP_NN*. The *local_inside_score* feature gets relatively high weight, though it is not

among the top 10 features used. Somewhat surprisingly to us, in the parse structure difference feature *POSRelDiff* in fact gets the highest weight, being found to associate strongly with the negative class. For features associated with the positive label (i.e. the current slot being an OOV), we find that the most informative features are all parse features, primarily tuples containing OOV tags: *OOV_VP_VP_OOV, NN_NP_NN_OOV, VB_VP_NP_OOV*. The aggregate parse structure difference feature *relOnlyDiff* also gets high weight.

Experiment 3: Region-level F-score vs. DET Curves

While using OOV detection F-scores allows us to evaluate different systems in terms of their performance at a particular operating point, it is also valuable to gauge the performance of the systems over a range of operating points. Such an evaluation will allow us to better select a system according to downstream application needs, for example when operating at a set (required) precision or false alarm rate.

Figure 4.7 shows the performance of the first and second stage MaxEnt systems built using the BOLT-P2 data, evaluated on the P2-Eval set. We are primarily concerned with the region of low false alarm rate, $0.05 < FA < 0.25$. In this region, we find that the second MaxEnt stage performs better than the first MaxEnt stage. Performance of the two stages is tied in regions of very high and very low FA. Performance of the two stages on the P3-Eval set is similar (curve not included).

We cannot compare performance on the P2-Eval and P3-Eval datasets directly, since they contain different samples. However, we can compare performance on the shared external evaluation set, NIST Eval. The performance differences for the NIST Eval set when training using BOLT-P2 and BOLT-P3 data, respectively, are shown in figure 4.8. In regions of very low false alarm, the P2 system is better, whereas in higher FA regions the P3 system outperforms. Thus, depending on the needs of the overall system our component integrates into, we could pick one configuration over the other.

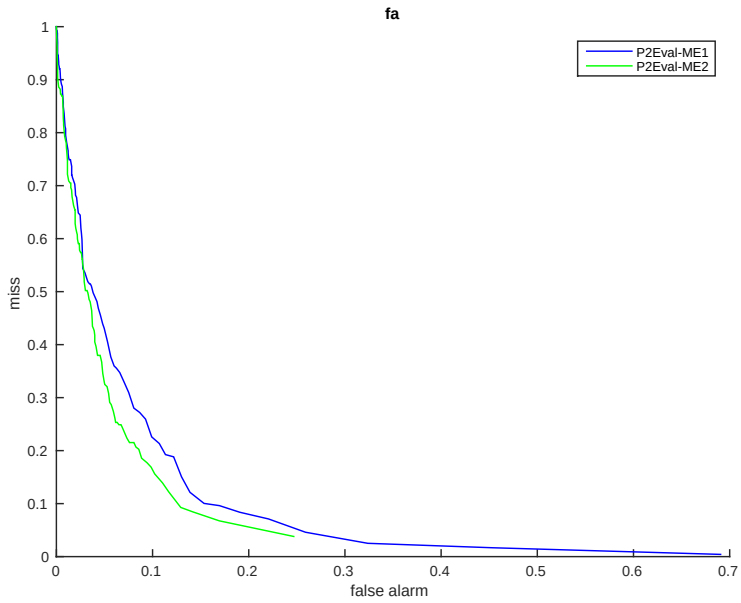

Figure 4.7: Miss vs. false alarm curve of OOV detection on the P2-Eval set using the first and second MaxEnt systems.

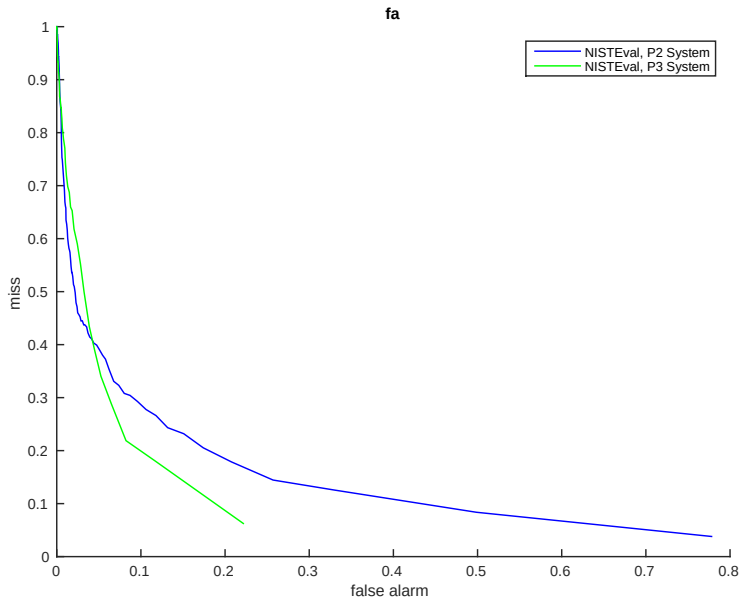

Figure 4.8: Miss vs. false alarm curve of OOV detection on the NIST-Eval set using the second MaxEnt systems built using BOLT-P2 and BOLT-P3 systems.

4.2.3 Name Error Detection

We repeat the OOV detection experiment 2 for the name error detection task. As in the case of OOV detection, we use parsers trained on the CTS SMALL treebank with no adaptation, using the default weights for NULL and error arc rules. Here we focus on the BOLT-P3 data only, for which we have reliable name error labels.

Results for each of the three stages in the error detection system are shown in table 4.5. We see that performance for this system is similar to that obtained for the OOV detection task on the BOLT-P3 datasets, though name error detection absolute F-scores are slightly higher than those for OOV detection (shown in table 4.4). As with OOV detection, parsing on its own hurts, with the parser performing significantly worse than the first MaxEnt stage; however, the second MaxEnt stage, which uses parse features as well, yields significantly improved results on all three datasets. Results presented in table 4.5 will serve as a baseline for name error adaptation experiments in chapter 6.

Stage / Dataset	first MaxEnt		parser		second MaxEnt	
	WER*	F-score	WER*	F-score	WER*	F-score
P3-DEV	12.8	31.8	17.1	16.4	12.5	42.7
P3-EVAL	11.7	38.1	14.7	17.4	11.6	43.5
NIST EVAL	5.7	36.0	7.8	17.2	5.6	42.2

Table 4.5: Name Error Detection, CTS SMALL Treebank with no parsing adaptation, with results showing the impact of each stage of processing.

4.3 Discussion

The initial experiments lead to mixed performance when it comes to the impact of parsing. We obtain no improvement in rescoring with the parser used as a language model, with WER degradation of 1-2.5% (absolute) depending on the test set and treebank used. Perhaps surprisingly, we find that a larger treebank is no guarantee of performance improvements; in fact, the smaller, mixed source CTS SMALL treebank outperforms the SWITCHBOARD

treebank-based parsers in all cases, though often by only a small amount. We hypothesize that a large factor in this outcome is the domain mismatch, both in terms of lexical use (e.g. imperfect coverage of the in-domain vocabulary by the available treebanks) and, perhaps more importantly, in terms of sentence structure (with a significant contributor to the degradation being the rules for generating WCN NULL arcs). We will investigate this issue further in the next chapter, when we discuss the proposed domain adaptation strategies for ASR rescoring.

Parsing fares much better when used in the context of error detection; we obtain consistent improvements from the parser-derived features when they augment a MaxEnt classifier in the second MaxEnt stage. We find, however, that parsing alone does not help, with the classification results from the parser stage often being worse than those obtained from the baseline MaxEnt system. This effect holds for both OOV and name error detection. For OOV detection in particular, we also find that using the larger, better-matched BOLT-P3 dataset configuration yields better results on the final evaluation set, NIST Eval. This is consistent with having more training data available in the BOLT-P3 dataset as compared to the BOLT-P2 dataset.

Chapter 5

PARSER DOMAIN ADAPTATION FOR ASR ERROR CORRECTION

In chapter 4, we investigated the use of parsing as an ASR rescoring language model when the parser is trained on mismatched domain data (with respect to the target domain). This chapter presents two methods for adapting a parser model trained on out-of-domain data, using in-domain data that we labeled with locations of OOV and name errors but not with hand-labeled syntactic structure, with the goal of improving ASR rescoring. The first method employs self-training techniques to automatically generate additional in-domain training data. The second method augments the standard PCFG model with a log-linear model. In both cases, the objective is given by an auxiliary task (word error rate improvements over ASR output), rather than optimizing parsing directly; this is a form of weak task supervision.

The work in this chapter will answer a number of questions regarding the performance of the domain adaptation methods proposed. Specifically, we would like to know:

- What is the relative impact of the different domain adaptation approaches proposed for ASR rescoring?
- What is the impact of the amount of in-domain data available?
- How robust are the adaptation algorithms to tuning (in terms of hyperparameters as well as system configuration choices)?

Section 5.1 includes a short discussion that motivates the domain adaptation approach in more detail. The self-training algorithm and the associated experiments are presented in section 5.2. The log-linear adaptation algorithm, and its associated experiments, are presented in section 5.3. We conclude with a comparison of the two methods and a discussion of the final results in 5.4.

5.1 Motivation

A parser trained on out-of-domain data suffers from mismatch both at the lexical level (with lexical distribution of the target domain being different from the distribution of words in the treebank data, due to task-related as well as topical differences) and in higher-level syntactic structures (in which case style also plays a significant role in the type of mismatch present, in addition to task-related differences, such as information-seeking vs. unstructured conversation). Even when studying conversational domains, differences in style may manifest themselves, for example in terms of the rate of disfluencies or the average sentence length (which correlates with syntactic complexity). Human-human conversations mediated by a computer, as is the case in the BOLT system, tend to be more carefully planned and somewhat more formal than direct conversations [131]; in this sense, the language in the target domain data is more closely aligned to human-computer interactions than to standard human-human dialogs.

An extra source of differences, in addition to the lexical and sentence structure mismatch between the source and target domains, is the presence of NULL arcs in the confusion network structure used as input to the parser. A parser trained on reference or even 1-best hypothesis output will not learn any rules to handle such arcs. The proposed adaptation approach therefore must generate a parser capable of better handling NULL arc rules.

5.1.1 Data Analysis

In chapter 3 we showed (in table 3.5) that the WER of the ASR one-best hypothesis is significantly higher than the WER of the oracle path through the confusion network, with differences as high as 30% relative for our internal test sets. Thus, selecting a better path through the confusion network can lead to significant performance gains. In chapter 4 we showed that parsing does not help improve ASR performance when the parser is trained on mismatched domain data. Differences between domains include both vocabulary mismatch (with only 50-70% of the words in the speech training corpora being covered by the out-of-domain treebanks) and sentence structure differences (in particular, the in-domain data is much more formal and more careful, with fewer filled pauses or repetitions).

To address these issues we investigate domain adaptation using two data sources of different modality. The language model training dataset LMTrain Small was shown (in table 3.4) to have much better coverage of the vocabulary in the in-domain sets. Thus, we expect to gain some benefit from using it for adaptation, even though sentences in this corpus are much longer than those in the in-domain data (on average 18 words vs. 9-11 for the in-domain speech transcripts); this addresses the topic mismatch between the out-of-domain treebanks and the in-domain test sets. On the other hand, by using the training portions of the in-domain speech data, P2-Train or P3-Train, we can target more directly the style of the conversations in that data, both in terms of formality level and the NULL arcs in confusion networks.

We saw in table 3.3 that there are few stylistic differences between P2-Train and P3-Train data. However, the latter corpus has five times more utterances available; it also has a lower baseline word error rate. Thus, we are interested in using both datasets for adaptation, to investigate effects of word error rate and training set size on the proposed adaptation approaches.

5.1.2 Adaptation Objective

Previous parsing adaptation focused on methods which optimized parser performance itself. In this work, we instead focus on optimizing the performance of the downstream task of interest. For the rescoring task, we focus on building an adapted parser model which can be used to yield a better path through confusion network data. We use WER as the objective function, using the WER of the training set to optimize model parameters and the WER of the development set to optimize hyperparameters of the training procedure, as required by each algorithm.

5.2 Parser Self-Training

5.2.1 Algorithm

We present the basic self-training process in algorithm 1. For each sentence in the target domain, we obtain the 1-best parse tree (i.e. the parse tree over the WCN for that utterance

Algorithm 1 Parser self-training algorithm

INPUT: Base treebank TB

INPUT: Unlabeled set $S_{\text{unlabeled}}$

INPUT: Dev set S_{dev}

OUTPUT: Best adapted parser M_n

Train parser on treebank TB, obtaining model M_0

Base score: WER of dev set from initial parse tree

for each iteration i **do**

  for each sentence s_j in unlabeled set $S_{\text{unlabeled}}$ **do**

    Parse s_j, obtaining parse tree p_{ij} with score sc_{ij}

  end for

  for each threshold $T_{ik} \in \{0.05, \ldots, 0.95\}$ **do**

    Augment treebank TB with $T_{ik}\%$ of unlabeled set parse trees p_{ij}

      with highest scores sc_{ij}

    Retrain parser using augmented treebank, obtaining model M_{ik}

    Parse dev set S_{dev} and score, obtaining new WER score WER_{ik}

  end for

  find best threshold $\hat{k} = \text{argmax}_k\, WER_{ik}$

  current iteration score: $WER_i = WER_{i\hat{k}}$

  if $WER_i >= WER_{i-1}$ **then**

    stop

  end if

  Current iteration parser: $M_i = M_{i\hat{k}}$

end for

Return parser M_n for the last iteration n

with highest score assigned by the factored parser algorithm), and add it to the treebank if its inside score is in the top $T\%$. The model is retrained after traversing the entire unlabeled corpus (i.e. a full iteration). The threshold T for each iteration is tuned over a range of values using the dev set. The procedure terminates when either no additional trees are added to the treebank, or the improvement obtained on the held-out set drops below a pre-set threshold.

A variation on the method allows us to use additional information related to the task objective to select the best automatically-generated trees to add to the treebank. Since the 1-best parse tree may not correspond to the optimal path through the confusion network, we instead select the best parse tree (out of the N-best parses produced by the factored parser algorithm) with the restriction that the path is optimal relative to the reference string. We call such trees the "oracle" trees, though we do not require that the tree structure itself be optimal, only the terminals selected as part of the parsing process. Thus, the "oracle" aspect refers to the optimal performance according to performance on the ASR rescoring task and relative to the WER metric.

We experiment with another variation on the method, related to whether automatically-generated trees are preserved across iterations or regenerated at each iteration. In the base algorithm, we re-label the entire unlabeled set at each iteration. Thus, improvements in the parser in each iteration may be used to improve the trees used to augment the treebank in the previous iteration. As an alternative, we consider not regenerating trees which were already used in a previous iteration, thus gradually reducing the unlabeled set. This approach allows for faster processing (and potentially faster convergence).

5.2.2 Experiments

We perform a series of experiments to assess the efficacy of the self-training approach. The main outcome we are interested in is determining whether a self-trained parser, when used as a rescoring language model, can yield a new hypothesis that has lower error than the best hypothesis according to the confusion network arc posterior. However, with a number of algorithm decisions to consider, we also investigate which algorithm configurations yield

the best results. Also of interest are metrics of practical interest, such as the model size (in terms of number of rules learned) and thus parsing speed.

We report results separately for BOLT-P2 and BOLT-P3 data, to determine the impact on two datasets obtained from slightly different recognizer and confidence prediction systems. In each case, we use the development subsets P2-DEV and P3-DEV to tune hyperparameters of the algorithm (specifically, the number of iterations and subset of trees to be used for retraining at each iteration), and to report results for the different training procedure choices. The evaluation partitions P2-EVAL and P3-EVAL are reserved for reporting the final results.

Experiment 1: Treebank and Unlabeled Set Selection

In the first experiment, we investigate the effect of performing self-training using the in-domain text corpus LMTRAIN and the in-domain speech training partitions of BOLT-P2 and BOLT-P3, respectively, using each of the three initial treebank configurations available.

Results for the smaller BOLT-P2 set are presented in table 5.1. We present the WER for each treebank using only the baseline parser with no self-training as well as with self-training using just in-domain text data, just in-domain speech data, or both (with the speech data being added after self-training with text). The baseline WER for the P2-DEV set, obtained from the ASR 1-best path through the WCN, is 13.8. As discussed in section 4.2.1, all three treebanks yield an unadapted parser which severely underperforms the baseline.

Adding LMTRAIN data does not result in any improvements over the baseline, and in fact yields a small performance degradation when added to the CTS SMALL treebank. In all cases, the performance degrades as more data is being added, with the best (in this case, least-bad) results being obtained when only the highest-scoring 5% of the trees are used.

Using just the P2-TRAIN data results in significant improvements over the baseline as well as the no self-train case. No major differences are observed across the three treebank choices (the preprocessed version yields a minor 0.1% performance loss relative to the other two choices). This shows that using the speech data provides consistent advantage over using the text data from the language modeling training set.

Using the speech data in addition to text data again leads to consistent improvements over the no self-train case. Comparing to the speech-adapted parsers alone, we see negligible decrease in performance in two cases and minor improvement in the third.

Unlabeled Dataset	CTS Small	Switchboard (no preproc.)	Switchboard (preproc.)
None (no self-train)	15.1	15.6	15.7
LMTrain	15.3	15.6	15.7
P2-Train	**13.2**	**13.2**	13.3
Both	13.4	13.3	**13.2**

Table 5.1: Self-training WER results, P2-dev, compared to a baseline of 13.8% with no rescoring (best results for each treebank are in bold).

Unlabeled Dataset	CTS Small	Switchboard (no preproc.)	Switchboard (preproc.)
None (no self-train)	15.0	15.3	15.6
LMTrain	15.2	15.4	15.5
P3-Train	**13.8**	13.8	13.9
Both	**13.8**	**13.7**	**13.8**

Table 5.2: Self-training WER results, P3-dev, compared to a baseline of 12.7 with no rescoring (best results for each treebank are in bold).

Results for the equivalent experiment performed on the BOLT-P3 data are presented in table 5.2 (baseline WER from the ASR 1-best path through the WCN: 12.7). We see that many of the same conclusions hold here, too. The LMTrain corpus alone does not improve over the no self-training baseline, whereas using the speech P3-Train corpus does; the combination never hurts, consistent with the observation that the LMTrain corpus effectively gets lower weight in the combination, due to the larger size of the speech corpus P3-Train relative to P2-Train. However, in this configuration self-training no longer

yields a win over the baseline (no rescoring) case, with the best result from self-training (WER=13.7) unable to beat the WER of the original ASR 1-best hyps (WER=12.7).

We include two figures indicating the performance of the self-trained parsers at different iterations of the self-training algorithm and with different amounts of data at each iteration. Figure 5.1 shows the behavior of the self-training algorithm for the adaptation process starting with the CTS SMALL treebank, with adaptation using the P2-TRAIN data. We see that performance improves after each of the first two iterations; the third and fourth iterations perform about the same, with the fourth iteration (red line) performing slightly worse towards the end. We therefore select the 5% case in the third iteration as the best result, to avoid possible overfitting in the later iteration.

Results for the analogous condition using the LMTRAIN set are shown in figure 5.2. In this case, during the first iteration the parser results never improve over the baseline, so we terminate the algorithm. Results are reported using the best configuration on the single iteration, which are obtained with the smallest amount of data considered (5%).

Experiment 2: Tree Selection Method

In the second experiment, we explore two distinct tree selection approaches for the speech data. In the first method, we select the most confident trees over the training set confusion networks, regardless of whether any given tree produces a correct path through the confusion network (relative to the reference string for that utterance); only the parser confidence is taken into account. As an alternate method, we consider only trees which produce the correct strings. Our hypothesis is that the second method may produce better results by reducing the number of incorrectly-used rules (in particular of those rules specific to parsing confusion networks. such as those involving NULL arc productions). This method may also be slightly faster, with each iteration over the unlabeled data yielding a potentially smaller addition to the labeled treebank.

Results for the BOLT-P2 data are presented in table 5.3. We present results for the parser adapted using only the speech corpus using the two methods. Unexpectedly, we observe a slight (0.1%) performance degradation using the SWITCHBOARD treebanks, and

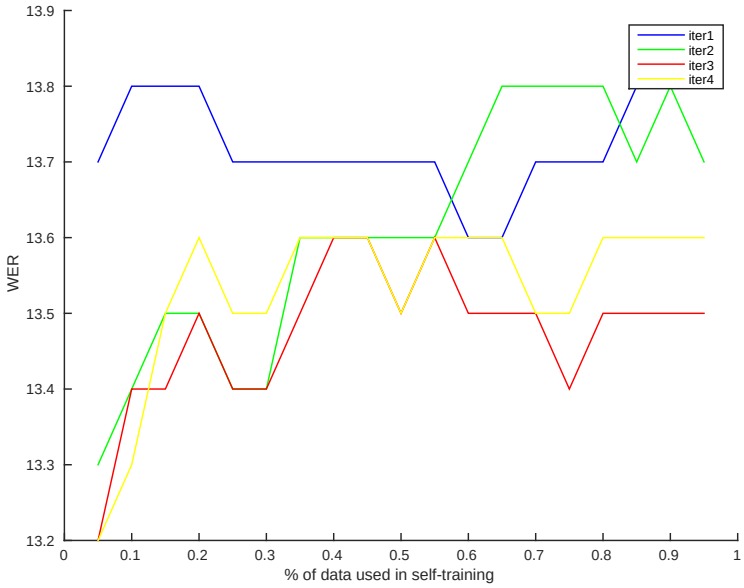

Figure 5.1: Self-training results at different iterations, CTS Small treebank adapted with P2-Train data.

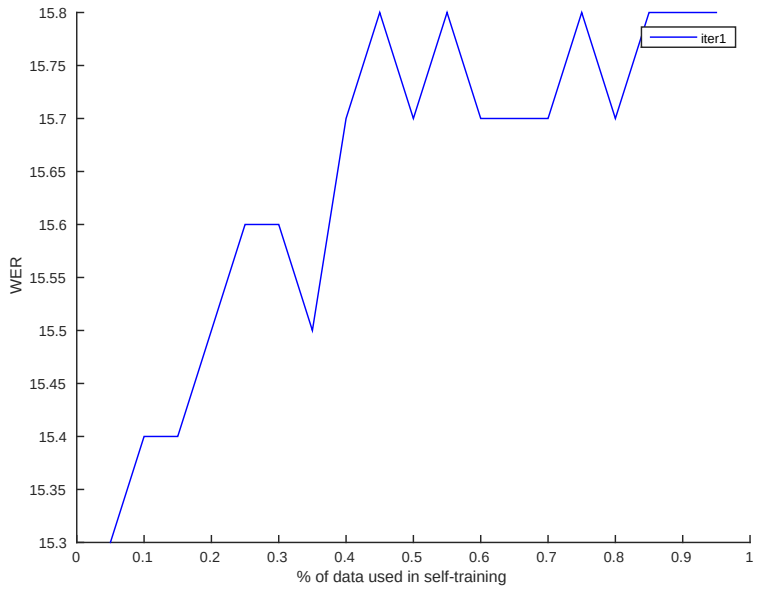

Figure 5.2: Self-training results at different iterations, CTS Small treebank adapted with the LMTrain data.

no change using the CTS SMALL treebank, showing that in fact there is no benefit from restricting the adaptation to those trees over the optimal paths through confusion networks. The last two rows in table 5.3 show the number of trees used in the final adaptation round. We observe that, for the CTS SMALL data, we are using about the same number of trees in each configuration. For the two versions of the SWITCHBOARD treebank, the best configuration obtained using the oracle string approach uses considerably more trees than the best configuration obtained using the highest confidence method. This suggests that using the oracle tree selection approach makes it slightly more difficult to obtain the optimal adaptation, though the differences are always small.

Tree Selection	CTS SMALL	SWITCHBOARD (no preproc.)	SWITCHBOARD (preproc.)
Highest score	13.2	13.2	13.3
Oracle	13.2	13.3	13.4
# trees selected			
Highest score	113	47	47
Oracle	103	427	475

Table 5.3: Self-training tree selection method, P2-DEV.

Experiment 3: Relabeling Approach

In the third experiment, we explore two different relabeling strategies, as discussed in section 5.2.1. The default method of relabeling (automatically) each unlabeled sample at each iteration is compared with the approach of relabeling only utterances which have not already been selected for inclusion in the treebank. We denote these methods as **Full** and **Least Confident** in the results tables, respectively. In all experiment conditions we use the highest score selection strategy.

Results for the BOLT-P2 data are presented in table 5.4. We present results for the parser adapted using only the speech corpus using the two methods. We find that the **Full**

method outperforms the **Least Confident** method when starting with the SWITCHBOARD treebank, though the differences are small. The **Least Confident** approach requires one fewer iteration of self-training to no longer achieve any improvements. Results for the CTS SMALL treebank are tied, though again convergence is faster by one self-training iteration for the **Least Confident** relabeling approach. Behavior for self-training under the other training conditions (using the LMTRAIN for adaptation testing on the BOLT-P2 data, or using either LMTRAIN or P3-TRAIN for adaptation testing on the BOLT-P3 data) also did not result in any performance differences; those results are not included here for brevity.

Relabeling Set	CTS SMALL	SWITCHBOARD (no preproc.)	SWITCHBOARD (preproc.)
Full	13.2	13.2	13.3
Least Confident	13.2	13.4	13.5

Table 5.4: Self-training relabeling set tuning, P2-DEV.

Experiment 4: Oracle Experiments

We perform two oracle experiments to further assess the impact of the self-training approach. In the first experiment, we explore using each dev set, P2-DEV and P3-DEV, respectively, as input to the self-training algorithm (replacing the training partitions, P2-TRAIN and P3-TRAIN, respectively). This will allow us to eliminate the mismatch due to vocabulary and syntactic differences between the training and dev partitions. Results are presented in table 5.5. In all cases, we use the CTS SMALL treebank as the out-of-domain treebank, and use the full relabeling method and the highest score tree selection method. We find that, for BOLT-P2 data, we in fact do not get any gain from using the dev set as self-training source instead of the (3x larger) training partition. This suggests that much of the benefit in adaptation is provided by having parse trees over confusion network data, rather than specific syntactic structures or missing vocabulary entries.

To further analyze the hypothesis that having confusion networks, rather than a closer

Self-Train Data Source	P2-DEV	P3-DEV
Train	13.2	13.7
Dev	13.2	13.6

Table 5.5: Self-training oracle experiment, using dev sets for training.

vocabulary or syntactic match, is the key to adaptation success, we design a second experiment, contrasting the effect of using the speech corpus (which leads to improvements in performance) and the LMTRAIN corpus (which does not in most cases). As an additional, oracle condition, we also use as training the reference transcripts corresponding to the P2-TRAIN and P3-TRAIN corpora, using the CTS SMALL treebank for the BOLT-P2 case, and the SWITCHBOARD treebank without preprocessing for the BOLT-P3 case, respectively. Results are shown in table 5.6. In both cases, using speech references instead of confusion networks leads to significant performance degradation; in the BOLT-P3 case, we no longer get a win from the adaptation, just as in the case of using only the LMTRAIN text data. This shows that in fact a significant factor in the success of the self-training for this task comes from using the speech data. In particular, we infer that having ASR output in the form of confusion networks allows us to learn effective weights for rules involved in the generation of NULL arcs, while the text data (both LM training and speech reference transcripts) cannot help in that respect.

Self-Train Data Source	P2-DEV	P3-DEV
LMTRAIN	15.3	15.4
Speech WCNs	13.2	13.8
Speech references	13.8	15.4

Table 5.6: Self-training oracle experiment, using references for adaptation.

5.3 Log-linear Models

The self-training approach generated an adapted model by adding entire parse trees to the training data, with a focus on adding the trees which lead to the highest improvement in word error rate when rescoring the development set. However, this approach does not allow for targeting of specific rules or sets of rules. To address this issue, we develop a second approach based on a log-linear framework, specifically targeting adaptation at a more granular level. As the best results using self-training adaptation all involve the speech corpora, with the LMTrain data found to provide little benefit, we focus in this approach on making more effective use of confusion network data, and do not use the LM text at all.

5.3.1 Model Description

We augment the unlexicalized PCFG model portion of the factored parser model discussed in chapter 4 with a new model component, not present in the factored parser infrastructure. The new component is a log-linear model learned discriminatively, so that the score of a constituent parse T of the WCN w given model parameters θ_c becomes:

$$f_c(T, w, \theta_c) = f_{\text{PCFG}}(T, w, \theta_{\text{PCFG}}) f_{\text{LL}}(T, w, \theta_{\text{LL}}) \tag{5.1}$$

$$= \left(\prod_{a \to b \in T} \rho_{\text{PCFG}}(a, b, \theta_{\text{PCFG}}) \right) \exp(\phi(T) \cdot \theta_{\text{LL}}) \tag{5.2}$$

where T is a constituent parse tree with terminals forming a path through the WCN w, f_{PCFG} denotes the inside score of a parse computed by the base PCFG model, f_{LL} denotes the score obtained from the log-linear model, and model parameters $\theta_c = [\theta_{\text{PCFG}}, \theta_{\text{LL}}]$ consist of both the PCFG model parameters θ_{PCFG} and the log-linear model parameters θ_{LL}. Each rule of the form $a \to b$ is a rule used in the derivation of T, though we make no restrictions of b being a single constituent or terminal, and we denote by $\rho_{\text{PCFG}}(a, b, \theta_{\text{PCFG}})$ the score for using rule $a \to b$ in the derivation. We use ϕ to denote log-linear model features.

The log-linear model framework is very flexible, allowing for many different types of features to be used. In this work, we use rule-level binary features, so that $\phi_{a \to b} = 1$ for all rules $a \to b$ used in the derivation, and 0 otherwise. Thus, we obtain:

$$f_c(T, w, \theta) = \prod_{a \to b \in T} \rho_{\text{PCFG}}(a, b, \theta_{\text{PCFG}}) \exp(\theta_{\text{LL}}(a, b)) \tag{5.3}$$

Converting to log space, we can rewrite the expression as:

$$\log f_c(T, w, \theta) = \sum_{a \to b \in T} (\log \rho(a, b, \theta_{\text{PCFG}}) + \theta_{\text{LL}}(a, b)) \tag{5.4}$$

where $\theta_{\text{LL}}(a, b)$ indicates the weight of the feature corresponding to the grammar rule $a \to b$ in the log-linear model LL. Since the PCFG tree score can be written in the form $\prod_{a \to b \in T} \exp(\theta_{\text{PCFG}}(a, b))$, we can rewrite equation 5.4 as

$$\log f_c(T, w, \theta) = \sum_{a \to b \in T} (\theta_{\text{PCFG}}(a, b) + \theta_{\text{LL}}(a, b)) \tag{5.5}$$

so the LL model is learning an additive correction term for the PCFG rules.

In practical terms, the behavior of the LL model as an additive correction over the PCFG rules means that we can apply the standard CKY algorithm to fill the chart, with the only difference being in the computation of the score for each rule, using the sum of the PCFG score and the log-linear rule weight. Rules which do not get any log-linear model adaptation simply receive a default $\log \theta_{\text{LL}}(a, b) = 0$.

The log-linear model feature weights θ_{LL} are computed using an averaged online perceptron learning algorithm [35]. For each utterance, we first parse the reference string, and increment the weights for all rules used in the parse tree over the reference; then, we parse the full confusion network, and decrement the weights for all rules used in the best top parse tree. Thus, rules used (correctly) in both cases retain their original weight; rules used (correctly) in the reference tree, but not the tree over the confusion network, have their weight incremented; and rules used (incorrectly) in the tree over the confusion network, but not in the reference tree, have their weight decremented. We output the model after each iteration and select the configuration that yields the best result on the held-out development set. We initialize the feature weights uniformly before the first iteration of the algorithm. The version of the model with uniform log-linear model weights is used as a second baseline when comparing the impact of different adaptation approaches.

5.3.2 Experiments

We perform two sets of experiments aimed at determining the effectiveness of our log-linear model adaptation approach. As in the self-training case, we investigate the performance of the log-linear adapted models separately for the BOLT-P2 and BOLT-P3 data, and evaluate each out-of-domain treebank separately to determine the effect of the initial labeled tree set. We investigate separately whether adapting the weights of just the NULL arcs is more effective than adapting the weights for all grammar rules, with the former producing less adaptation but being less prone to overfitting.

We perform the adaptation procedure for 10 iterations and use the best results on the P2-DEV or P3-DEV sets, respectively, to perform comparisons across treebanks and rule set configurations. We treat the perceptron update weight as a hyperparameter, selected from among three different values $\{0.25, 0.5, 1.0\}$.

Experiment 5: Log-linear Model Adaptation, NULL *Arc Rules*

We examine the impact of adaptation using a log-linear model when only weights of the NULL arc rules are being modified. Table 5.7 includes results using the BOLT-P2 dataset. We see that, for each treebank, the initial model (using the NULL arc rules with default weights) performs worse than the adapted models with each perceptron weight. However, no configuration outperforms the baseline (WER=13.8). The CTS SMALL treebank yields better results than either version of the SWITCHBOARD treebank, both when using the initial model and after adaptation. The perceptron update weight makes little difference, with only minor differences observed in a couple of configurations.

Results for the BOLT-P3 dataset are shown in table 5.8. Again, adaptation helps over the models with default weights, though no log-linear adapted model outperforms the baseline (WER=13.1). The CTS SMALL treebank still outperforms the SWITCHBOARD treebank configurations. This time, however, the differences between models obtained with the various perceptron update weights are larger, in particular for the CTS SMALL case, which yields a significantly better-performing model than the SWITCHBOARD configurations.

We illustrate the behavior of the log linear-adapted model training procedure by showing

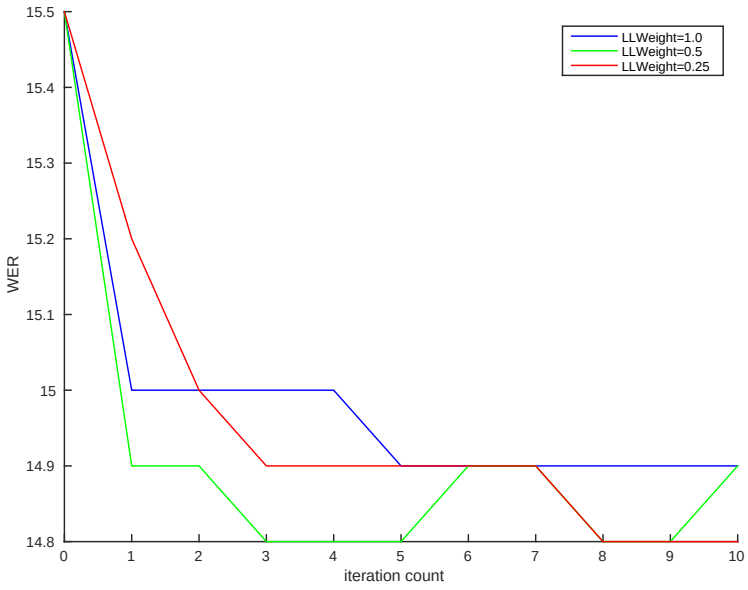

Figure 5.3: Log-linear adaptation results at different iterations, CTS Small treebank adapted with P2-Train data, adaptation of NULL rules only.

Perceptron Weight	CTS Small	Switchboard (no preproc.)	Switchboard (preproc.)
n/a (default weight)	15.1	15.6	15.7
1.0	14.9	15.3	15.3
0.5	14.8	15.3	15.3
0.25	14.8	15.3	15.4

Table 5.7: Log-linear adaptation treebank comparison, NULL rules, P2-dev.

Perceptron Weight	CTS Small	Switchboard (no preproc.)	Switchboard (preproc.)
n/a (default weight)	15.1	15.4	15.7
1.0	15.0	15.4	15.7
0.5	15.0	15.2	15.6
0.25	14.9	15.2	15.4

Table 5.8: Log-linear adaptation treebank comparison, NULL rules, P3-dev.

results for each iteration of the perceptron algorithm. Results for adaptation of the NULL rules using different perceptron update weights, using the CTS Small treebank and the P2-Train data for adaptation, are shown in figure 5.3. We see that the algorithm improves results for each perceptron update weight, with only minor differences in results between the different weights.

Experiment 6: Log-linear Model Adaptation, All Rules

Next, we repeat the previous experiment, this time with all rules being included in adaptation, instead of adapting weights for only NULL arc rules. Results for the BOLT-P2 dataset are shown in table 5.9. The general trends are similar to the case of adapting only the NULL arc rules shown in table 5.7: the CTS Small treebank models perform better than Switchboard models, and the differences between models based on different

thresholds are small; the baseline WER is still lower than any of the adapted models, and some adaptation generally helps over the default weights. However, the improvement due to weight adaptation (over the default weights baseline) is lower than when using only NULL arc rules.

Perceptron Weight	CTS Small	Switchboard (no preproc.)	Switchboard (preproc.)
n/a (default weight)	15.1	15.6	15.7
1.0	14.6	14.9	14.8
0.5	14.8	15.0	15.0
0.25	14.4	15.1	15.2

Table 5.9: Log-linear adaptation treebank comparison, all rules, P2-dev.

Perceptron Weight	CTS Small	Switchboard (no preproc.)	Switchboard (preproc.)
n/a (default weight)	15.1	15.4	15.7
1.0	14.4	13.9	13.8
0.5	14.5	14.8	13.0
0.25	14.0	13.5	15.0

Table 5.10: Log-linear adaptation treebank comparison, all rules, P3-dev.

Table 5.10 shows results for the BOLT-P3 dataset. This time, the results are rather different from the corresponding experiments with just NULL arc rules. The Switchboard treebanks outperform the CTS Small treebank during adaptation, with one configuration coming closes to matching the WER=12.7 no-rescoring baseline (preprocessed Switchboard with 0.5 perceptron update weight, with WER=13.0). On the other hand, the performance seems much more difficult to predict, in particular the perceptron update weight tuning appearing very sensitive to local optima.

We perform additional analysis of the model yielding the best results on the P3-dev

data. Table 5.11 shows the WER scores for the P3-DEV dataset after each log-linear model iteration. We observe a significant performance improvement after the first couple of iterations of the perceptron algorithm, with another minor improvement before the performance levels off. Performing more analysis on the trees produced by each model, we see a significant jump in the number of $X \rightarrow XX$ productions (which are used in the Penn Treebank [83] to indicate unknown, uncertain, or unbracketable constructions) each time the WER decreases. The last few iterations use the production in over 90% of the trees. In these configurations, the XX constituent generates the vast majority of terminals, at the expense of other POS tags present in the treebank.

Iteration	WER	# trees with $X \rightarrow XX$ rules
initial	15.7	1
0	15.2	1
1	15.1	5
2	13.3	215
3-4	13.1	1690
5-9	13.0	1720

Table 5.11: Number of trees using the $X \rightarrow XX$ rule obtained at each iteration of the log-linear adaptation using BOLT-P3 data which gives best results on the P3-DEV dataset.

5.4 Analysis

5.4.1 Final Results

The tuning process for each adaptation method used the dev partitions in the BOLT-P2 and BOLT-P3 datasets. We evaluate the best systems obtained using self-training and log-linear model adaptation on the internal eval partitions, P2-EVAL and P3-EVAL, as well as the external NIST EVAL set. For BOLT-P2, we obtained significant improvements from self-training, so we include multiple configurations to assess the possibility of overfitting; we also include the best configuration from log-linear model adaptation, even though this

configuration did not yield any gain over the ASR 1-best. For BOLT-P3, we include only one configuration from self-training, since we did not obtain any improvement over the ASR 1-best; we also include two configurations from log-linear adaptation, to compare the effects of the degenerate parser model that yielded good results against a configuration which did not result in gains, but led to a less degenerate model.

Parser Config	P2-EVAL	NISTEVAL
Baseline (WCN hyp)	14.6	5.8
SelfTrain-CTS-Small	14.3	6.2
SelfTrain-Swbd-NoPreproc	14.3	6.1
SelfTrain-Swbd-Preproc	14.2	6.1
LLAdapt-CTS-Small-All	15.8	6.5

Table 5.12: Best adaptation results for P2-EVAL and NISTEVAL using the P2-TRAIN data for adaptation.

Results using the BOLT-P2 data are shown in table 5.12. We see that the three self-training configurations perform similarly on both the internal (P2-EVAL) and the external (NISTEVAL) sets, with only minor differences for each model. On the other hand the log-linear adapted model performs significantly worse than either model, consistent with the performance degradation observed on the P2-DEV set.

Results using the BOLT-P3 data are shown in table 5.13. We see that, for the internal evaluation dataset, neither the self-training configuration nor the log-linear adapted models improve upon the baseline of using the best confusion network hyps, though the degenerate log-linear adapted model using adaptation over all the rules comes close to matching the baseline. On the external evaluation set, that model, and the self-trained model, also come closest to beating the confusion network baseline, resulting in performance that matches the BOLT-P2-based adaptation.

We perform a more detailed analysis of the results obtained from the best configurations using adaptation with each methods, examining not only the final WER scores for each configuration, but also the rate of insertions, substitutions, and deletions. We select one self-

Parser Config	P3-EVAL	NISTEVAL
Baseline (WCN hyp)	11.7	5.8
SelfTrain-CTS-Small	12.9	6.0
LLAdapt-CTS-Small-NULL	13.7	7.5
LLAdapt-Swbd-NoPreproc-NULL	14.0	7.9
LLAdapt-Swbd-Preproc-NULL	14.3	7.9
LLAdapt-Swbd-Preproc-All	11.9	6.1

Table 5.13: Best adaptation results for P3-EVAL and NISTEVAL, using the P3-TRAIN data for adaptation.

trained model and one model with log-linear adaptation for each (BOLT-P2 and BOLT-P3) training condition. For self-training with BOLT-P2 data we select the best model starting with CTS-SMALL treebank, **SelfTrain-Swbd-Preproc**. For log-linear adaptation we use the CTS-SMALL configuration with adaptation of all rules, **LLAdapt-CTS-Small-All**. For BOLT-P3-based training, we use **SelfTrain-CTS-Small** for self-training, and **LLAdapt-Swbd-Preproc-All** for log-linear adaptation.

Results are summarized in table 5.14. We see that the two self-training configurations have similar effect on each component of the WER computation; the insertion rate drops, but at the expense of an increase in the deletion rate, with the substitution rate remaining relatively unchanged (or occasionally improved). For the log-linear adaptation-based systems, we see that results are slightly more mixed; with the BOLT-P3-based adaptation, which gives us results as good as or better than the self-training systems, we notice a worsening of the substitution rate as well as the deletion rate. Comparing performance on the NIST EVAL set under the two different training set conditions, we find that performance of the two self-training systems is similar, with only minor differences between the insertion, deletion, and substitution rates across the two configurations. On the other hand, performance of the BOLT-P3 log-linear adapted system is better than that of the BOLT-P2 log-linear adapted system, though this improvement comes at the cost of degenerate POS tags being produced in the vast majority of cases. The differences are primarily manifested

through a much larger insertion rate in the case of the BOLT-P2 system. We will discuss the effect this behavior has on further uses of the BOLT-P3-based log-linear adapted parser model in chapter 6.

5.4.2 Discussion

We find that both adaptation methods improve the ability of the parser to act as a rescoring language model; when comparing parsers trained on out-of-domain data alone with the best adapted models, we find that adaptation gives significant gains, on the order of 1-2% absolute WER. On the other hand, we find that even the best adapted models do not always improve upon the top recognizer hypothesis, though they always come close. The trade-off often appears to be caused by a drop in insertion rate at the cost of an increase in deletions. We obtain better results on the BOLT-P2 internal test sets than on the BOLT-P3 internal test sets, though the best-performing systems trained using each set yield similar results on the external NIST Eval test set.

Comparing the two approaches for adaptation, we find that in general self-training is more consistently likely to improve upon the baseline. In fact, in all but one case, the best models came from a self-training approach. The single log linear-adapted model which does produce an improvement comes at the cost of a degenerate model, which is incapable of generating correct part-of-speech tag output. The self-training models are also much more robust to hyperparameter tuning, with many different training configurations yielding the same results. On the other hand, the log-linear adaptation seems much more sensitive to the parameters—most parameter configurations do not yield improvements, and those which do generate unstable models.

A somewhat surprising result is that in fact the choice of treebank, too, has little effect on the ultimate performance of the self-training models. Conventional wisdom suggests that larger treebanks are better, even when the data is mismatched; we find otherwise, and thus prefer smaller treebanks, which lead to much faster models. Furthermore, the minor treebank preprocessing performed to reduce the mismatch between reference and automatic transcripts seems to have little effect on the eventual results, though the best result using

log-linear adaptation does make use of the preprocessed treebank. Of far more importance, we find, is the use of even a small amount of in-domain ASR output data, even if such data does not contain any hand-generated parses.

Dataset	Configuration	P2-Eval				P3-Eval				NIST Eval			
		Sub.	Del.	Ins.	WER	Sub.	Del.	Ins.	WER	Sub.	Del.	Ins.	WER
	baseline (no rescore)	7.9	2.3	4.4	14.6	6.3	3.1	2.3	11.7	3.7	0.7	1.4	5.8
P2	**SelfTrain-Swbd-Preproc**	7.7	2.9	3.6	14.2	n/a	n/a	n/a	n/a	4.0	1.2	0.9	6.1
	LLAdapt-CTS-Small-All	8.0	2.3	5.7	15.8	n/a	n/a	n/a	n/a	4.0	0.7	1.9	6.5
P3	**SelfTrain-CTS-Small**	n/a	n/a	n/a	n/a	6.2	5.0	1.7	12.9	3.7	1.3	1.0	6.0
	LLAdapt-Swbd-Preproc-All	n/a	n/a	n/a	n/a	6.5	3.0	2.4	11.9	4.0	0.9	1.1	6.1

Table 5.14: Best adaptation results, WER component analysis

Chapter 6

PARSER DOMAIN ADAPTATION FOR ASR ERROR DETECTION

In chapter 5, we explored more effective usage of parsing in rescoring the output of a speech recognition system, developing two adaptation frameworks (one based on self-training, the other on log-linear model adaptation). Here we reuse the infrastructure built in that chapter, repurposing it for the goal of correctly identifying regions of errors in the ASR transcripts. While the methods are general and can be applied to different kinds of ASR errors, we focus in particular on two types: out-of-vocabulary (OOV) errors and misrecognized names (whether in-vocabulary or OOV). We also explore briefly some changes required to better apply the algorithms to the specific task of interest, instead of reusing the infrastructure completely out-of-the-box.

The main questions we answer in this chapter are:

- How much does error-aware parser adaptation help improve the performance of the parser and the impact of parser-derived features when used in error detection?
- What is the impact on error detection of initializing parser adaptation with parser models adapted for rescoring?
- What is the impact of domain adaptation under different training dataset conditions?
- What are the advantages of combining adaptation methods, if any?

In chapter 5 we introduced two domain adaptation methods, self-training and log-linear adaptation, and applied them to the ASR rescoring task. For error detection, intuitively, it seems that log-linear model adaptation may be better suited than self-training, because it is possible to focus on adapting only the grammar rules involved in generating error regions. Therefore, we explore log-linear model adaptation first, in section 6.1.

Next, in section 6.2, we discuss modifications to the self-training algorithm when applied to error detection. We present two sets of experiments, for self-training alone as well as for self-training following log-linear adaptation, thus showing the effect of combining the two adaptation methods.

We summarize the findings in section 6.3, discussing results obtained with each adaptation method separately as well as in combination, and assessing the performance of the different methods under different training dataset scenarios. We also address some practical issues here, including the model sizes obtained from each adaptation method.

6.1 Log-Linear Model Adaptation

In this section, we discuss the log-linear model adaptation method as applied to error detection. The model and adaptation algorithm are substantially similar to the version used for parser-based rescoring, described in section 5.3. Here, we first discuss specific differences as they pertain to applying the algorithm for error region detection instead of rescoring. Next, we present a series of experiments to assess the impact of the log-linear adaptation approach.

6.1.1 System Implementation Decisions

We review system implementation decisions related to three aspects of the log-linear adaptation algorithm: the objective optimized during the adaptation procedure, the initial conditions, and a proposed modification to the grammar specifically aimed at improving name error detection.

Objective

With rescoring, using word error rate as an objective was natural, allowing us to upweight rules contributing to correct derivations, and downweight rules which do not. Such a strategy is also useful when the model is updated with a goal of improving error detection. However, for error detection we use the modified word error rate metric, WER*, which captures correctly-detected error regions as well as mistakes due to detected error regions

which are too large or too small.

Note that we draw a distinction between the objective used in the update rule and the model evaluation. While we use WER* as an update rule metric, for model evaluation we continue to use both WER* and the error detection F-score. By taking into account all mistakes made by the parser in the path which was selected through the confusion network, not just those in an error region, we have the ability to reweight other rules that may affect the correct parsing around error regions.

Initial Model

When applying log-linear adaptation to rescoring, we started from an initial model trained on only the out-of-domain treebank data. For error detection, we adapt starting with a no-error model augmented with error rules with default weights. Thus, we have a choice of which non-error model to use as a starting point: the initial treebank-only model, or a model that has already been adapted for rescoring. Intuitively, it seems that starting with a better rescoring model will help improve error detection. Though the tasks are different, the WER* objective used in error detection is similar to the WER objective used in adaptation for rescoring. Starting with a better rescoring model will likely make fewer mistakes in non-error regions, thus allowing the adaptation to focus on learning weights for only the rules of interest.

Task-Dependent Grammar Modifications

The error rules added to the treebank-learned grammar, as described in section 4.1.2, allow for modeling a broad range of error types, with error regions that can be generated by any grammar constituent. In practice, however, domain knowledge about a particular error type can help guide the adaptation process.

We employ one instance of human knowledge injection for the name error detection task. A correct name will almost certainly be a noun (NN/NNS/NNP/NNPS); however, a name error region may spread beyond the name itself if the name error corrupted a neighboring word. In such a case, the error region should take the syntactic category of the smallest

constituent containing the error. With names, such an error region would most likely be a noun phrase (NP), adjectival phrase (AdjP), or adverbial phrase (AdvP). We therefore consider a variation on the log-linear parser adaptation process for name error detection, where the weight of an error rule will be updated only if it is generated by a syntactic category listed above.

6.1.2 Experiments

We perform a series of experiments to assess the effectiveness of the log-linear model adaptation approach when aimed at error detection, rather than error correction as we did in section 5.3.2. We are interested in both OOV detection and name error detection, with OOV detection assessed on BOLT-P2 and BOLT-P3 data, allowing us to study the effect of the methods under different training conditions. We assess name error detection only for the BOLT-P3 data. As in section 5.3.2, we evaluate adaptation strategies of different subsets of parser rules. For name error detection, we also experiment with whether to utilize the full set of error rules or just those targeted at generation of names.

The baseline for all error detection tasks is the output of the first stage MaxEnt classifier with confusion network structural and lexical features, as described in section 4.1.1. Each parser model is used for error detection both directly, using the parser as an error detector, and indirectly, using the second MaxEnt stage classifier with features derived from parse output. As in chapter 4, we consider the question of whether the parser is best used directly or to extract features for the second MaxEnt stage. As a second set of baselines, for each adaptation experiment we also use the results from the parser and second MaxEnt stage without any rescoring adaptation.

We introduce four sets of experiments, which are summarized briefly below.

- Experiment 1 explores the problem of log-linear adaptation tuning. Two objectives are used: WER* and F-score, with the discussion focusing on the trade-offs in optimizing each. Results are presented for only one system configuration, though they generalize over many different system design choices.

- Experiment 2 presents results for both OOV and name error detection using log-linear adaptation starting with the baseline parser models introduced in chapter 4.

- Experiment 3 discusses a set of initial results using log-linear adaptation starting with different rescoring models. One rescoring model is selected for each training dataset condition, to use in future experiments.

- Experiment 4 presents an in-depth analysis of log-linear adaptation starting from the models selected in the third experiment.

We conclude this section with a discussion of the findings.

Experiment 1: Log-Linear Adaptation Tuning and Model Selection

Each parser configuration yields four scores: the parser stage WER* and F-score, and the corresponding second MaxEnt stage WER* and F-score. Therefore, when we discuss the "best" system configuration, the question arises of how to select the best system configuration when different metrics yield different performance comparisons. To investigate this issue, we perform an experiment using one model adaptation approach, and present results for each iteration of the perceptron training, for both the parser stage and the corresponding second MaxEnt stage. We present results for one typical configuration of log-linear adaptation for OOV detection using adaptation of error rules (corresponding to the "Error Only, 1.0" line in table 6.1 in a later experiment); many other configurations yield similar results.

WER* results for both the parser stage and the second MaxEnt stage are included in figure 6.1. We see that the behavior of the WER* metric is very stable for the parser stage, the metric varying only slightly at iteration 6. On the other hand, for the second MaxEnt stage the behavior is less stable, with the minimum being attained at a few different iterations. None of the best second stage MaxEnt configurations match the best parser stage configuration, but the difference between the best parser stage configuration and the rest of the scores are negligible.

F-score results for both the parser stage and the second MaxEnt stage are shown in figure 6.2. The parser stage results are again stable, though the best configurations are

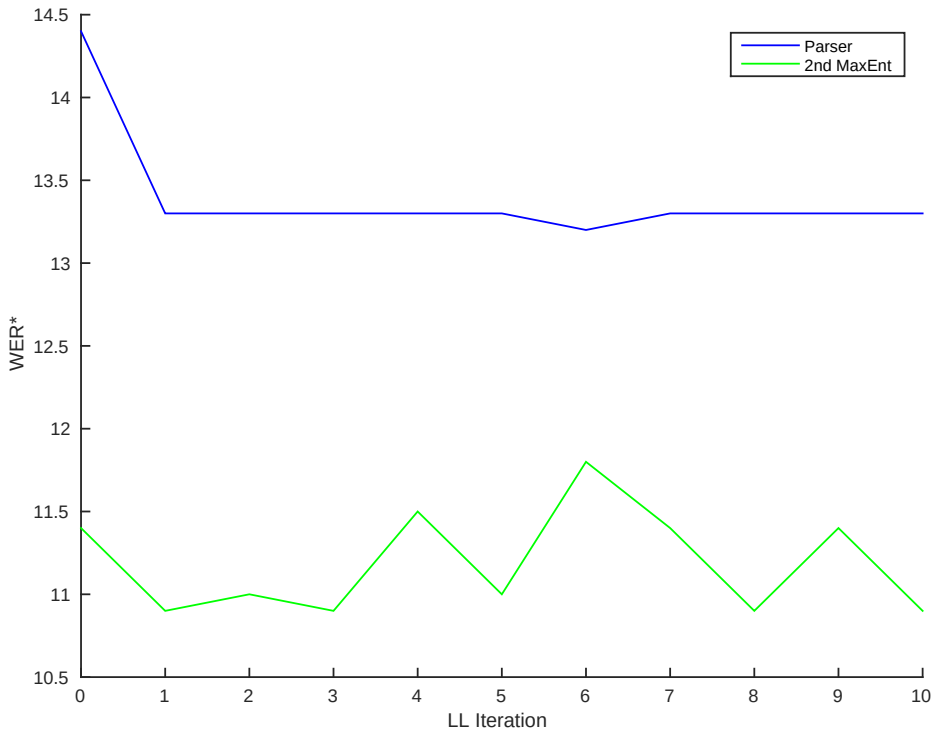

Figure 6.1: Sample tuning of log-linear adaptation for OOV detection, P2-DEV WER* scores for the parser and second MaxEnt stages.

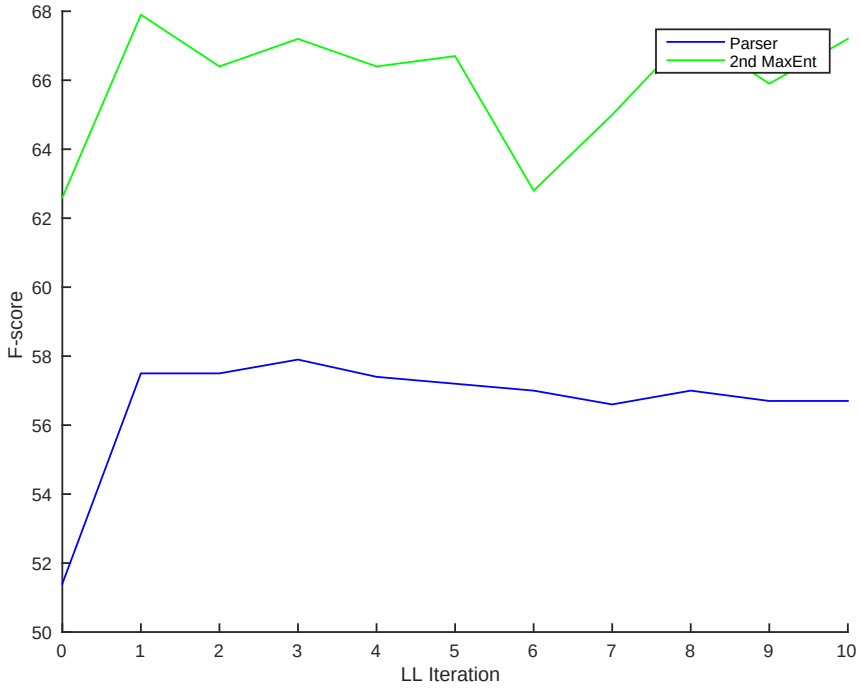

Figure 6.2: Sample tuning of log-linear adaptation for OOV detection, P2-DEV F-scores for the parser and second MaxEnt stages.

attained at iteration 3 and 1, earlier than the single best WER* score for that stage. The second MaxEnt stage results are less stable, just as in the WER* case; the highest score is obtained in iteration 1, which is one of the best, but not the absolute highest, F-score result for the parser stage. The best parser stage result corresponds to a high, but again not maximal, second MaxEnt stage result.

Examining the two plots together gives us some insight into how to tune the adaptation process. First, we see that for either metric, choosing the best system according to the second MaxEnt stage yields a parser that is pretty good; on the other hand, choosing according to parser stage performance may yield a not-very-good second MaxEnt stage. We will continue to report results based on tuning either stage, to highlight differences in performance between the two stages as they occur. Second, comparing results obtained when tuning for either metric, we see that the relative error difference over different iterations is lower when measuring WER* (about 8%) vs. measuring F-score (about 10-13%). Furthermore, choosing the iteration number based on F-score yields a WER* score that is close to optimal, more so than the reverse. This observation holds for both the parser stage and the 2nd MaxEnt stage. Thus, we will use the F-score results for tuning, but continue to report both WER* and F-score for system evaluation.

In all future experiments, we refer to results obtained from tuning on the dev set parser stage F-scores as "best parser", and obtained from tuning on the dev set second MaxEnt stage F-scores as "best 2nd MaxEnt". We will compare these against results from the first MaxEnt stage (the "Baseline MaxEnt") and parser and second MaxEnt stages without parser adaptation ("No-Adapt Parser"), all from chapter 4. Tuning results will also indicate which iteration of the log-linear adaptation procedure yields the best results for each stage, allowing comparisons of consistency of results across stages throughout the experiments.

Experiment 2: Log-linear Adaptation Starting with No-Rescoring Parser

The next set of experiments illustrates the impact of log-linear adaptation for error detection, starting with the baseline CTS Small parser model without any adaptation for rescoring (henceforth referred to as the "no-rescoring" parser model). We investigate three

data-task pairs independently:

- BOLT-P2, OOV detection,
- BOLT-P3, OOV detection, and
- BOLT-P3, name error detection.

For each dataset-task pair, we examine the performance of the parser and second MaxEnt stage when using different perceptron update weights in the log-linear model adaptation procedure, and when adapting different subsets of the parser rules. For each dataset, we present the tuning of different hyperparameters of the system on the respective dev sets. Final results are presented on the internal and external eval sets. For OOV detection only, we present results using both the BOLT-P2 and the BOLT-P3 datasets, side by side, comparing the effect of different training dataset conditions separately on the internal eval sets P2-EVAL and P3-EVAL, as well as on the shared NIST-EVAL external testset. We will reuse this same experiment structure in Experiment 4, where we discuss log-linear adaptation starting with a rescoring parser instead.

Experiment 2a: OOV detection

We discuss OOV experiments first. We present two sets of results, for experiments using the BOLT-P2 dataset and BOLT-P3 dataset for training, respectively.

Results for OOV detection using the BOLT-P2 data are shown in table 6.1. With both adaptation using only error rules, and adaptation using error and NULL arc rules, we get fairly similar results. The adapted parsers improve upon the no-adaptation baselines, with fairly consistent results across the three log-linear adaptation weight choices. Unsurprisingly, the smaller the adaptation weight, the more iterations it takes to achieve the optimal result. Performance for the corresponding second MaxEnt stage classifiers also consistently improve over the second MaxEnt stage in conjunction with the baseline parser without adaptation, though the improvement tends to be higher when only error rules are adapted.

Rule Set	Perceptron	Best Parser			Best MaxEnt		
	Update Weight	WER*	F-score	Iter. #	WER*	F-score	Iter. #
Baseline MaxEnt	n/a	n/a	n/a	n/a	12.8	58.9	n/a
No-Adapt Parser	n/a	14.4	51.4	n/a	11.4	62.6	n/a
Error Only	1.0	**13.3**	**57.9**	**3**	10.9	67.9	1
	0.5	13.2	57.8	8	**10.6**	**70.1**	**8**
	0.25	13.3	57.5	9	10.8	68.4	7
Error & NULL	1.0	13.1	57.7	4	11.2	67.4	3
	0.5	13.3	57.0	6	11.1	67.4	1
	0.25	13.2	56.6	9	11.4	64.8	1
All	1.0	13.1	47.4	2	12.2	61.3	10
	0.5	13.4	40.4	10	11.9	62.9	10
	0.25	**13.0**	**57.8**	**1**	12.3	63.3	1

Table 6.1: Log-Linear Adaptation for OOV detection starting from the no-rescoring parser, P2-DEV (best results in bold),

Rule Set	Perceptron	Best Parser			Best MaxEnt		
	Update Weight	WER*	F-score	Iter. #	WER*	F-score	Iter. #
Baseline MaxEnt	n/a	n/a	n/a	n/a	13.4	26.8	n/a
No-Adapt Parser	n/a	19.8	20.5	n/a	13.3	30.0	n/a
Error Only	1.0	18.1	25.4	4	**12.7**	**38.0**	**5**
	0.5	18.0	25.5	9	12.7	37.3	10
	0.25	18.1	25.1	10	13.7	33.7	4
Error & NULL	1.0	17.5	27.6	3	**12.7**	**38.0**	**4**
	0.5	17.2	27.6	8	12.8	37.2	7
	0.25	17.5	27.5	10	**12.8**	**40.0**	**8**
All	1.0	12.8	3.4	1	13.3	28.7	8
	0.5	**15.5**	**28.5**	**1**	13.3	31.1	1
	0.25	**15.8**	**28.7**	**1**	13.8	33.2	1

Table 6.2: Log-Linear Adaptation for OOV detection starting from the No-rescoring parser, P3-DEV (best results in bold).

Results when all rules are included in adaptation are significantly different from the other two configurations. We see that in this case, the perceptron update weight has a much bigger impact on the results. Best results are obtained with the smallest update weight; in this case, the parser performance is similar to configurations when smaller sets of rules are used in adaptation, whereas the second MaxEnt stage performs worse than the other two rule adaptation configurations. In both cases, the higher perceptron update weight results are significantly worse, no longer outperforming the no-adaptation baseline. For the best configuration (using update weight = 0.25), the best result is obtained after only one round of training, with subsequent results getting progressively worse.

Results for OOV detection using the BOLT-P3 data are shown in table 6.2. As in the analogous case using the BOLT-P2 data, we notice a few trends across all three rule adaptation strategies. We find that the higher perceptron update weights tend to give the best results after fewer iterations, though in general the best performance using each set of rules in adaptation is pretty consistent, obtaining significant gains over the no-adaptation parser baseline. The parser alone either does not outperform the first stage MaxEnt baseline (as seen in the adaptation using error rules only) or only barely outperforms it (for error and NULL arc rules adaptation). The second MaxEnt stage systems perform better than all the baseline configurations, with or without parsing, though we get slightly better results (in terms of F-score) when using the error + NULL arc rule adaptation strategy.

When we perform adaptation using all rules, compared to adaptation using one of the two filtered sets of rules, the best results are obtained after fewer iterations, with performance declining as the model is adapted further. Overall, parser-only results are no worse from adaptation using fewer rules, though the second MaxEnt stage classifiers perform slightly worse in this configuration.

Table 6.3 summarizes the results using both the BOLT-P2 and the BOLT-P3 datasets for OOV detection, presenting results for the two internal eval sets, P2-EVAL and P3-EVAL, respectively, for each training condition, as well as for NIST EVAL for both training conditions. We see that parser adaptation leads to improvement over the baselines using all configurations in the parser stage; in the second MaxEnt stage, we obtain improvements using the BOLT-P2 data, but results using the BOLT-P3 data are more mixed.

Dataset	Adaptation Configuration	P2-EVAL		P3-EVAL		NIST EVAL	
		Parser	MaxEnt	Parser	MaxEnt	Parser	MaxEnt
P2	Baseline MaxEnt	n/a	64.7	-	-	n/a	30.7
	No-Adapt Parser	54.2	65.7	-	-	22.6	31.5
	Best Parser	**59.1**	**71.9**	-	-	25.6	33.9
	Best MaxEnt	58.3	70.2	-	-	26.1	35.2
P3	Baseline MaxEnt	-	-	n/a	33.6	n/a	36.2
	No-Adapt Parser	-	-	21.8	**36.0**	24.7	38.0
	Best Parser	-	-	**30.0**	33.8	**41.6**	**41.1**
	Best MaxEnt	-	-	28.5	35.5	31.8	28.6

Table 6.3: Log-Linear Adaptation for OOV detection starting from the no-rescoring parser: eval sets, F-score only (best results in bold).

Comparing results across stages, we see that, in general, the second MaxEnt stage outperforms the parser; the exception is in the BOLT-P3 training case using the NIST EVAL testset, where results are mixed, with one configuration showing tied results between the two stages, and the other showing minor degradation in performance.

In terms of whether to select the best system based on performance in the parser stage vs. the second MaxEnt stage, results are also inconclusive; with the BOLT-P2 data, we observe more degradation in performance when comparing the internal and external test sets when we select based on parser stage; when training using BOLT-P3 data the results are reversed, in that we obtain a win from selecting based on the parser stage, whereas we do not in the best system selected using second MaxEnt stage results.

No single approach consistently works best in all cases; however, adaptation almost always yields some improvement over the no-adaptation cases. The best MaxEnt-based system using BOLT-P2 data outperforms all the BOLT-P2 baselines. The only configuration that outperforms the baseline for NIST EVAL set using BOLT-P3 data for training does not improve performance on the P3-EVAL internal set, but other configurations yield improvements on P3-EVAL as well.

Experiment 2b: Name Error Detection

Next, we discuss experiments related to the name error detection task. We present a set of experiments based on the baseline (no adaptation) CTS SMALL-trained no-error parser first. In addition to the three rule adaptation configurations used for OOV detection (error rules only, error + NULL rules, and all rules), we add a fourth configuration, using the reduced set of error rules most likely (based on human judgment) to generate name errors as described in section 6.1.1.

Rule Set	Perceptron	Best Parser			Best MaxEnt		
	Update Weight	WER*	F-score	Iter. #	WER*	F-score	Iter. #
Baseline MaxEnt	n/a	n/a	n/a	n/a	12.8	31.8	n/a
No-Adapt Parser	n/a	17.1	16.4	n/a	12.5	42.7	n/a
Error Only	1.0	14.5	26.1	10	12.5	44.3	10
	0.5	14.5	26.0	10	12.5	41.0	10
	0.25	14.6	25.7	4	12.5	40.0	2
Error & NULL	1.0	14.4	27.0	2	12.5	42.6	3
	0.5	14.4	27.3	6	12.5	41.7	10
	0.25	14.4	26.2	3	12.5	42.2	6
All	1.0	14.7	25.2	1	12.6	35.3	1
	0.5	14.7	24.1	2	12.5	43.4	1
	0.25	13.7	32.0	4	**12.4**	**48.8**	**5**
Name Error & NULL	1.0	**13.3**	**34.7**	**10**	12.5	44.9	6
	0.5	13.5	34.2	3	12.5	44.5	3
	0.25	13.5	31.4	6	12.5	41.5	1

Table 6.4: Log-Linear Adaptation for name error detection starting from the no-rescoring parser, P3-DEV.

Results for all adaptation conditions are presented in table 6.4. Using just error rules, the parser stage results show consistent (and large) improvement over the no-adaptation

baseline parser, though no configuration outperforms the MaxEnt baselines. Unlike in OOV detection, here we find that even the second MaxEnt stage classifiers, after adaptation, rarely outperform the second MaxEnt stage built on top of the parser with no-adaptation, with only one configuration yielding a better result. The same observations apply to the adaptation using NULL arc rules in addition to error rules; using that configuration, no system beats the no-adaptation second MaxEnt stage.

We obtain better results when adapting all parser rules. As before, the best results are obtained using the smallest perceptron update weight. This configuration in fact outperforms the baseline (i.e. without parsing adaptation) second MaxEnt stage classifier, both in its parser stage and the subsequent second MaxEnt stage. Results using the hand-filtered set of error rules include the configuration which yields the best parser stage-only results, both in terms of WER* and F-score. While results using the second MaxEnt stage do not outperform the best configurations obtained with adaption for all the rules, we find that, in general, performance is more consistent over the range of perceptron update weights.

Dataset	Adaptation Configuration	P3-EVAL		NIST EVAL	
		Parser	MaxEnt	Parser	MaxEnt
P3	Baseline MaxEnt	n/a	38.1	n/a	36.0
	No-Adapt Parser	17.4	43.5	17.2	42.2
	Best Parser	37.6	45.4	36.8	43.6
	Best MaxEnt	**37.8**	**51.8**	**41.7**	**46.2**

Table 6.5: Log-Linear Adaptation for name error detection starting from the no-rescoring parser, eval sets, F-score only.

Results comparing the two best adapted systems selected based on tuning over the parser and second MaxEnt stages are shown in table 6.5. We compare these against the first MaxEnt stage, as well as the parser stage and second MaxEnt stage where the parser is trained on only the out-of-domain treebank. In this case, we find significant gain from adaptation when comparing the parser stages alone, with F-scores that more than double in most cases. When comparing to the baseline first MaxEnt stage, all the parser stages from

the parser-adapted configurations perform better. We also obtain improvements from the MaxEnt stage in each parser adaptation condition. Comparing against the baseline second MaxEnt stage, we obtain wins with each adaptation condition, though the improvements are sometimes small; the best system is the one selected using the second MaxEnt stage results for tuning, which yields the best results in both parser and MaxEnt stage on both P3-EVAL and NIST EVAL data.

Experiment 3: Selection of Best Rescoring Models

In chapter 5, we performed model adaptation for ASR rescoring using both self-training and log-linear adaptation. Any of the rescoring models can be used as a starting point for the adaptation process targeting error detection. However, it is not necessarily the case that the best rescoring model may lead to the best error detection model. To investigate this issue before performing a full suite of adaptation experiments for error detection, we perform an initial experiment using log-linear adaptation for OOV detection, starting from the top rescoring models obtained using self-training and log-linear adaptation for each training dataset condition, as discussed in chapter 5 and shown in table 5.14. For BOLT-P2 data, we use:

- self-training: **SelfTrain-Swbd-Preproc**, the self-trained model starting from the preprocessed SWITCHBOARD treebank with adaptation using both the LMTRAIN corpus and the P2-TRAIN corpus;

- log-linear adaptation: **LLAdapt-CTS-Small-All**, the log-linear adapted model using the CTS SMALL treebank with adaptation for all rules in the grammar.

For BOLT-P3 data, we use:

- self-training: **SelfTrain-CTS-Small**, the self-trained model using the CTS SMALL treebank with adaptation using both the LMTRAIN corpus and the P3-TRAIN corpus;

- log-linear adaptation: **LLAdapt-Swbd-Preproc-All**, the log-linear adapted model using the preprocessed version of the SWITCHBOARD treebank with adaptation for all

rules in the grammar.

In all four cases we use adaptation using the error and NULL arc rules with a perceptron update weight of 1.0. Results for these four configurations are shown in table 6.6, with the baseline no-adaptation parser configurations from table 4.4 added for reference.

Dataset	Rescoring Adaptation	Best Parser WER*	Best Parser F-score	Best MaxEnt WER*	Best MaxEnt F-score
BOLT-P2	No-Adapt Parser	14.4	51.4	**11.4**	62.6
	self-training	**11.6**	**69.3**	**11.4**	**66.7**
	log-linear adaptation	13.3	55.6	11.8	65.4
BOLT-P3	No-Adapt Parser	19.8	20.5	13.3	30.0
	self-training	14.1	**36.3**	**12.8**	**35.2**
	log-linear adaptation	**12.8**	0.0	13.1	29.3

Table 6.6: Comparison of rescoring models used to initialize log-linear adaptation for the OOV detection task, P2-DEV and P3-DEV.

For the BOLT-P2 dataset, we find that starting with the best self-training rescoring model yields much better results than starting with the best log-linear adaptation rescoring model at the parser stage, when measuring both F-score and WER*. For MaxEnt, the differences are smaller, though the best self-training model still outperforms the best log-linear model. Both adaptation cases outperform the no-adaptation baseline.

For the BOLT-P3 dataset, we find that starting with the best self-training rescoring model yields better results than starting from the best log-linear adaptation rescoring model, even though in the rescoring task we obtained better results from the log-linear adaptation approach. We note that the log-linear adapted model suffers from degenerate performance. In particular, the extremely skewed distribution of pre-terminal rules, with most words being assigned the POS tag XX as discussed in section 5.3.2, leads to the model not generating any OOV regions (thus the OOV region detection F-score being 0); additionally, the MaxEnt classifier trained using parse features also suffers from performance degradation, relative

to the system trained starting with the best self-training-based rescoring model as well as compared to the no-adaptation baseline.

In all subsequent experiments with log-linear adaptation for error detection, we use the two best self-training models obtained for rescoring, SelfTrain-Swbd-Preproc for BOLT-P2 and SelfTrain-CTS-Small for BOLT-P3, respectively. We refer to these two configurations as the "best rescoring" models during subsequent experiments.

Experiment 4: Log-linear Adaptation Starting with Best Rescoring Parser

Using the best rescoring parsers obtained in experiment 3 as starting point, we now perform a set of experiments to determine the full impact of the log-linear adaptation procedure. The experiment structure is essentially the same as that used in experiment 2, though starting from the best rescoring parser instead of the no-rescoring CTS Small parser.

Experiment 4a: OOV detection

As in the no-rescoring case, we include two sets of results for OOV detection, using the BOLT-P2 data and BOLT-P3 data for adaptation and MaxEnt stage training, respectively.

Results using the BOLT-P2 data are shown in table 6.7. Using only error rules, we see that we obtain a small, but consistent, gain from most configurations, with the parser stage improving both over the baseline parser stage (in all configurations) and the baseline second MaxEnt stage (in one configuration). The best adapted second MaxEnt stage configuration slightly outperforms the best parser-only configuration, though the results are close.

We observe similar behavior when adapting both NULL arc rules and error rules. This time, the best parser stage result actually outperforms all the second MaxEnt stage configurations, though the performance of the second MaxEnt stage configurations is more consistent across the range of perceptron update weights than the results from the parser stage alone.

When all rules are included in the log-linear adaptation, we find that the parser stage results are slightly more consistent across the range of perceptron update weights, though the second MaxEnt stage results are overall generally better.

Rule Set	Perceptron	Best Parser			Best MaxEnt		
	Update Weight	WER*	F-score	Iter. #	WER*	F-score	Iter. #
Baseline MaxEnt	n/a	n/a	n/a	n/a	12.8	58.9	n/a
No-Adapt Parser	n/a	12.9	59.6	n/a	11.3	**67.4**	n/a
Error Only	1.0	11.8	66.4	5	**10.9**	**67.4**	0
	0.5	12.0	62.9	7	11.2	65.9	2
	0.25	12.4	60.9	1	11.8	65.5	1
Error & NULL	1.0	**11.6**	**69.3**	**8**	11.4	66.7	8
	0.5	12.2	63.3	1	11.7	66.2	5
	0.25	12.0	64.2	8	11.3	66.2	5
All	1.0	11.7	66.9	1	11.4	67.2	3
	0.5	12.0	64.2	1	11.9	63.1	1
	0.25	12.0	65.6	4	11.7	67.2	5

Table 6.7: Log-linear adaptation for OOV detection starting from the best rescoring parser, P2-DEV (best results bold).

Rule Set	Perceptron	Best Parser			Best MaxEnt		
	Update Weight	WER*	F-score	Iter. #	WER*	F-score	Iter. #
Baseline MaxEnt	n/a	n/a	n/a	n/a	13.4	26.8	n/a
No-Adapt Parser	n/a	15.0	24.9	n/a	13.4	31.0	n/a
Error Only	1.0	14.4	36.9	2	13.3	36.2	4
	0.5	14.1	36.1	3	12.8	37.1	1
	0.25	14.0	36.0	1	**13.3**	**38.8**	**7**
Error & NULL	1.0	14.1	36.3	1	12.8	35.2	1
	0.5	**13.9**	**37.2**	**1**	12.8	35.4	1
	0.25	13.9	34.4	3	12.8	37.0	5
All	1.0	14.3	32.7	1	12.7	36.0	1
	0.5	14.2	36.9	1	12.8	37.2	2
	0.25	14.0	36.0	6	**12.7**	**38.0**	**3**

Table 6.8: Log-linear adaptation for OOV Detection starting from the best rescoring parser, P3-DEV (best results bold).

Among the configurations using all rules for adaptation, we find that best results are obtained with the highest perceptron update weight (1.0). When all rules are used in adaptation, the best configurations are obtained after only a few iterations, after which point performance begins to degrade; this is consistent with the observations presented in experiment 2. Also consistent is the observation that in this case, the smallest perceptron update does require more iterations to obtain the best result. Best results are obtained when both error rules and NULL arc rules are adapted.

Results using the BOLT-P3 data are shown in table 6.8. We note that, compared to starting from the baseline model with no rescoring-oriented adaptation, we already obtain a significant win (from F=20.5 to F=24.9) in the parser stage, with a smaller win (from F=30.0 to F=31.0) in the second MaxEnt stage, as seen comparing to results in table 6.2.

Using only error rules in adaptation, we find that both the parser stages and the subsequent second MaxEnt stages improve upon all baseline results across the range of perceptron update weights tests, though the best parser stage result is from the highest update weight, whereas the best second MaxEnt stage result is from the lowest perceptron update weight. We also note that the best parser stage results are obtained after a small number of iterations, whereas the MaxEnt stages require more tuning, and thus are more likely to overfit.

When using both error rules and NULL arc rules, we find that all configurations improve over the baselines. However, F-score results are on average lower than when using only error rule adaptation, even though the WER* results are slightly better with this rule configuration. In this configuration, most of the best configurations are obtained after only one iteration of the perceptron algorithm, for both parser and second MaxEnt stages.

With adaptation using all rules, again, all configurations beat the baselines, though not significantly in some of the parser stages, and only the lowest perceptron update weight case benefits from more than a couple of rounds of perceptron training.

Table 6.9 summarizes the results using both the BOLT-P2 and the BOLT-P3 datasets for OOV detection, presenting results for the two internal eval sets, P2-EVAL and P3-EVAL, respectively, for each training condition, as well as for NIST EVAL for both training conditions. Parser adaptation once again helps; we improve upon the parser stage baselines in both training conditions, with particularly large gains in the BOLT-P3 case. For the

Dataset	Adaptation Configuration	P2-EVAL		P3-EVAL		NIST EVAL	
		Parser	MaxEnt	Parser	MaxEnt	Parser	MaxEnt
P2	Baseline MaxEnt	n/a	64.7	-	-	n/a	30.7
	No-adapt Parser	54.2	65.7	-	-	22.6	31.5
	Best Parser	**70.8**	66.2	-	-	40.0	31.6
	Best MaxEnt	69.6	**70.6**	-	-	40.0	**44.6**
P3	Baseline MaxEnt	-	-	n/a	33.6	n/a	36.2
	No-Adapt Parser	-	-	21.8	36.0	24.7	38.0
	Best Parser	-	-	**41.9**	38.7	**40.6**	35.5
	Best MaxEnt	-	-	40.8	**39.2**	38.2	37.8

Table 6.9: Log-linear adaptation for OOV detection starting from the best rescoring parser: eval sets, F-score.

second MaxEnt stage performance, we see that we obtain smaller gains (or, in the BOLT-P3 training condition, no gains). Performance on the NIST EVAL set using the second MaxEnt stage is harder to improve using the BOLT-P3 training condition, as in the case of adaptation starting with the no-rescoring models.

Comparing across stages, we find that in three out of the four adaptation cases, we actually obtain better results from the parsing stage than the corresponding second MaxEnt stage. The exception is the BOLT-P2 system selected based on the best second MaxEnt stage in the dev set, which in fact gives us the best performance on the NIST EVAL stage. However, in general, we find that we obtain more reliable results from the parser stage; all four adaptation systems' parser stages improve upon all the baseline second MaxEnt stages, using both training conditions and on all three test sets, as appropriate. Thus, to select a system, we would pick either of the training conditions, and use only the parser stage to provide the final OOV error region decisions.

Experiment 4b: Name Error Detection

Next, we present results for name error detection with adaptation starting from the best BOLT-P3 rescoring model. As in the previous name error experiment, we add an extra configuration, using the hand-selected set of error rules described in section 6.1.1 as well as the standard three sets from previous experiments as targets of the log-linear adaptation.

Rule Set	Perceptron	Best Parser			Best MaxEnt		
	Update Weight	WER*	F-score	Iter. #	WER*	F-score	Iter. #
Baseline MaxEnt	n/a	n/a	n/a	n/a	12.8	31.8	n/a
No-Adapt Parser	n/a	17.1	16.4	n/a	12.5	42.7	n/a
Error Only	1.0	**12.7**	**38.8**	**9**	12.5	42.5	3
	0.5	12.7	37.2	8	12.6	43.1	10
	0.25	12.8	34.2	10	**12.6**	**43.6**	**10**
Error & NULL	1.0	12.7	38.2	5	12.5	40.8	5
	0.5	12.7	38.4	9	12.5	39.2	9
	0.25	12.8	32.6	9	12.6	36.9	3
All	1.0	13.0	29.5	1	12.5	40.9	3
	0.5	13.1	31.6	10	12.5	41.1	9
	0.25	13.2	31.7	8	12.5	42.5	5
Name Error & NULL	1.0	12.6	33.5	6	12.6	39.4	9
	0.5	12.7	31.6	8	12.5	38.8	5
	0.25	12.7	29.5	4	12.6	36.4	5

Table 6.10: Log-linear adaptation for name error detection starting from the best rescoring parser, P3-DEV (best results bold).

Results for all adaptation conditions are shown in table 6.10. Comparing results using only error rules and error + NULL arc rules with the corresponding baseline parser results from table 6.4, we see that the parser stage results when starting from the adapted parser outperform the results when starting with a parser with no adaptation. The improvement

is particularly significant in terms of F-score, though we also see improvement of up to 2 WER* points between comparable configurations.

The performance improvements are more limited when adapting all the rules or only the hand-selected name error rules. For adaptation using all rules, the parser stage still gives an improvement in most configurations, primarily in terms of WER*, but results in the second MaxEnt stage are lower in the systems starting from an adapted parser. For the hand-selected name error rules, neither the parser nor the MaxEnt stage give wins in terms of F-score, though we still get a small win in terms of WER* from starting with the adapted parser, when comparing equivalent configurations.

Dataset	Adaptation Configuration	P3-EVAL		NIST EVAL	
		Parser	MaxEnt	Parser	MaxEnt
P3	Baseline MaxEnt	n/a	38.1	n/a	36.0
	No-Adapt Parser	17.4	43.5	17.2	**42.2**
	Best Parser	**43.2**	**44.8**	**37.4**	38.1
	Best MaxEnt	38.6	42.5	33.8	39.6

Table 6.11: Log-linear adaptation for name error detection starting from the best rescoring parser, Eval sets, F-score only.

Evaluation of name error experiments starting from the best rescoring parser is performed using the P3-EVAL and NIST EVAL sets, as shown in table 6.11. As before, we compare the best systems according to the parser stage and second MaxEnt stage results, respectively, with the first MaxEnt stage, as well as the parser and second MaxEnt stages where the parser is trained on only the supervised out-of-domain treebank.

In this case, we find significant gain from adaptation when comparing the parser stages alone, with F-scores that more than double in most cases. When comparing to the baseline first MaxEnt stage, all the parser stages from the parser-adapted configurations perform better. Looking at just the parser adaptation cases, we see that in all cases the parser stage does not perform as well as the second MaxEnt stage, though the differences are sometimes small. Comparing to the baseline second MaxEnt stage, we obtain only a small gain on

the P3-EVAL set, and no gain on NIST EVAL, though the performance degradation in the latter case is also small.

6.1.3 Experiment Summary

We conclude this section with a discussion of the overall findings. We have investigated using log-linear adaptation of the parser models to perform error detection for two kinds of errors, OOVs and name errors, and found that in all cases *some* log-linear adaptation gives a significant boost. For OOV detection, we further find that using the parser stage alone tends to yield more consistent performance across the internal and external evaluation sets than using the subsequent second MaxEnt stage; for name errors, while the parser boosts performance significantly as a result of adaptation, the best results are obtained when incorporating parser features into the second MaxEnt stage classifier. Results from the name error task are also different from those in OOV detection in another way; this is the only case where starting from a parser model adapted for rescoring using self-training did not help, even when the parser log-linear adaptation was constrained to a hand-selected set of rules.

The shared NIST EVAL test set allows us to compare the effects of log-linear adaptation with different amounts of data for the same task. We find, somewhat surprisingly, that more data does not in fact result in better performance on the shared NIST EVAL set. If anything, the results are slightly worse. Both the BOLT-P2 and the BOLT-P3 sets have an artificially-increased rate of errors; however, where in BOLT-P2 there are more OOVs, in BOLT-P3 the increased error count is due to names held out from the recognizer vocabulary at an artificially high rate. We speculate that, even though the errors in BOLT-P3 are in some sense more "natural", the artificially high rate of OOVs in BOLT-P2 tends to help the parser learn the behavior of OOVs more effectively. This may be a useful observation for designing training sets for related tasks in this or other low-resource domains.

Finally, a note on selecting hyperparameters and other system building decisions. We find that, in general, the best systems trained on BOLT-P2 data benefit from a higher perceptron update weight, and fewer rules used in adaptation. On the other hand, with

the BOLT-P3 data, the opposite is often the case. We speculate that the denser error rate (in particular the rate of OOVs) in BOLT-P2 has to do with this phenomenon, too; in this configuration, we start with a pretty good parser and adapt it further to only the rules directly involved the phenomena of interest. On the other hand, the no-error parsers adapted for rescoring using BOLT-P3 data do not perform as well, and thus more adaptation is needed to improve results on the error detection tasks. This leads to adapting a larger set of rules; hence, a smaller update weight is needed to avoid missing optimal configurations during the adaptation process.

6.2 Self-Training

As with log-linear adaptation, the setup for parser self-training applied to error detection is very similar to the approach used for rescoring. We detail specific differences from the setup used in self-training for rescoring below. The changes include using different objectives for adaptation, as well as different strategies for selecting trees for addition to the treebank. We also discuss differences from the log-linear adaptation approach as necessary.

6.2.1 Approach

Objective

The objective used in parser adaptation for rescoring was WER, both when adaptation was performed using self-training and with the log-linear model adaptation approach. For adaptation targeting error detection tasks, the objective used in parameter updates during the log-linear adaptation training is the modified word error rate, WER*; however, both WER* and F-score are used in evaluating the models, with an experiment performed to compare the effectiveness of each approach. With self-training, the objective is not used during parameter estimation, only in evaluating models on an unseen dataset, and selecting the best model to be used during the next training iteration. We therefore experiment with both WER* and F-score as adaptation objectives.

Initial Model

Just as with log-linear adaptation, we have a choice of whether to start from the baseline model or the best model trained for rescoring. In either case, the weights of error rules are initialized to their default, hardcoded values. However, for each condition (no rescoring or best-rescoring), we can alternatively use the best model obtained through log-linear adaptation as a starting point, thus starting from a better initialization point for the set of error rule weights. In the latter case, not only the weights of the error rules may be different from the no log-linear adaptation condition, but also various other rules could be given an adaptation weight, too, depending on which adaptation strategy was used. Experiments with self-training starting from both the no-error model with default error rules weights added, and starting from the error model adapted using log-linear model adaptation, will be presented.

Tree Selection Method

In chapter 5 we experimented with two selection methods for rescoring, allowing adaptation using either the highest-scoring tree for each sentence, or only those trees with minimal possible error selected from the n-best trees produced by the parser for rescoring; we deemed the latter set the "oracle" trees.

For error detection tasks the highest-scoring tree selection method is also used. As an alternative, instead of using the oracle trees, we experiment with a different tree selection method, using only the highest-scoring 1-best trees which contain error regions, to steer the adaptation towards adding more trees that will affect the error rule statistics. We refer to this method as using the "error" trees.

6.2.2 Experiments

We perform a series of experiments to assess the effectiveness of the self-training approach for error detection. As with log-linear model adaptation, we examine the effects of self-training separately for both OOV and name error detection, with OOV detection assessed using both the BOLT-P2 and the BOLT-P3 data. This allows the study of the adaptation

strategy under different training conditions.

In the experiments using log-linear adaptation, we found mixed results when performing adaptation for error detection starting with the best rescoring parser vs. starting with the no-rescoring parser. We will therefore try both configurations as a starting point for self-training, too. Additionally, we experiment with two options, starting either from a parser without any log-linear adaptation (thus, the error rules have default weights), or from the best parser after log-linear adaptation for error detection. As configuration choices we consider whether to relabel all trees or just the ones not selected in the previous iteration, and whether to restrict the adaptation to using only trees containing error regions.

We introduce three sets of experiments, which are summarized briefly below.

- Experiment 1 discusses the issue of self-training tuning in more detail. As with log-linear adaptation, two objectives are used: WER* and F-score. We experiment with different adaptation strategies for one task (OOV detection using the BOLT-P2 data), though results generalize over the other tasks and datasets.

- Experiment 2 discusses results for both OOV and name error detection using self-training starting with the models not adapted for rescoring (i.e. **no-rescoring** models).

- Experiment 3 discusses results for both OOV and name error detection using self-training starting with the best models adapted for rescoring (i.e. **best rescoring** models).

We conclude this section with a discussion of the findings.

Experiment 1: Tuning on WER vs. F-score*

We first perform an experiment to address the question of whether to tune the parser adaptation procedure using WER* or F-score, in both cases using the parser stage results to select the best configuration. We use as an experimental condition the OOV detection

target and BOLT-P2 data, starting from the best rescoring parser model with no log-linear model adaptation for OOV detection. Results are presented in table 6.12. Here we report the best parser-stage result for each system configuration ("best parser") and the corresponding second MaxEnt stage result ("corresp. MaxEnt"). We find that, as with log-linear adaptation in the parsing stage, WER* is very stable, with most system configurations yielding very similar results. This behavior also holds for the second MaxEnt stage, though with more variation in the scores. The F-score results are also stable in the parser stage; many of the best parser configurations according to F-score yield the best F-score results in the second MaxEnt stage as well. Thus, we will continue to perform model selection according to F-score results, but report both metrics for the evaluation sets. If multiple system configurations yield the best parser stage F-score results on P2-Dev, we will select the system with the best performance in the MaxEnt stage as a tie-breaker.

Settings			Best Parser		Corresp. MaxEnt	
Tuning Criterion	Trees	Relabeling	WER*	F-score	WER*	F-score
Uniform Weight Baseline			12.8	60.3	11.0	65.2
WER*	all	full	**12.6**	64.6	11.4	64.2
F-score	all	full	12.7	**64.9**	**10.9**	**68.1**
WER*	all	partial	12.7	**64.9**	**10.9**	**68.1**
F-score	all	partial	12.7	**64.9**	**10.9**	**68.1**
WER*	error	full	**12.6**	**64.9**	11.2	65.7
F-score	error	full	**12.6**	**64.9**	11.2	65.7
WER*	error	partial	12.7	64.2	11.8	66.4
F-score	error	partial	12.7	64.4	11.0	67.4

Table 6.12: Self-training for OOV detection: comparing different tuning strategies, evaluated on the P2-dev dataset.

We investigate the issue of tuning the self-training algorithm using either F-score or the WER* metric further. We present results for a single self-train configuration—using the BOLT-P2 data for OOV detection, selecting only trees containing OOV arcs, and

relabeling the entire P2-TRAIN set at each iteration. We tune two parameters, the number of iterations and the percentage of data being used for adaptation in each iteration, the latter selected in 5% increments over the range (5%–95%).

Results for the WER* metric are shown in figure 6.3. We see that, in general, we obtain better results in the second iteration than the first or third iterations. We also obtain better results with less data being added. A similar observation is made about results using the F-score metric, shown in figure 6.4, with best results here, too, being obtained in the second iteration. Comparing results between the two metrics, we find that both metrics select the same best systems for each iteration, with the best model being the one obtained in the second iteration, using 30% of the data. This supports the decision to select the best system according to F-score, but report WER* as well to indicate the effect of the system on error region extent detection.

Experiment 2: Self-Training Starting with the No-Rescoring Parser

In this experiment, we discuss self-training results starting with the no-error model that has not been adapted for rescoring, using two configurations: the configuration using baseline error rule weights (in experiment 2a), and the configuration using the best log-linear adapted model (in experiment 2b), for each respective error detection task and training dataset. In each case, results for both OOV detection and name error detection are presented.

Experiment 2a: Starting from the no-rescoring model with uniform error weights

First, we discuss experiments which use as starting point for self-training the no-error model with no adaptation for rescoring or error detection, using the default error rule weights.

Results for the OOV detection task are shown in tables 6.13 (using the BOLT-P2 dataset for training) and 6.14 (using the BOLT-P3 dataset for training). In each case, self-training adaptation improves upon the baseline parser with uniform weights. More improvement is obtained in the parser stage for each configuration, with improvements of over 10% absolute F-score for both P2-DEV and P3-DEV. On the other hand, the improvement in the second MaxEnt stage is smaller; for P2-DEV we still obtain improvements of up to 5% absolute F-

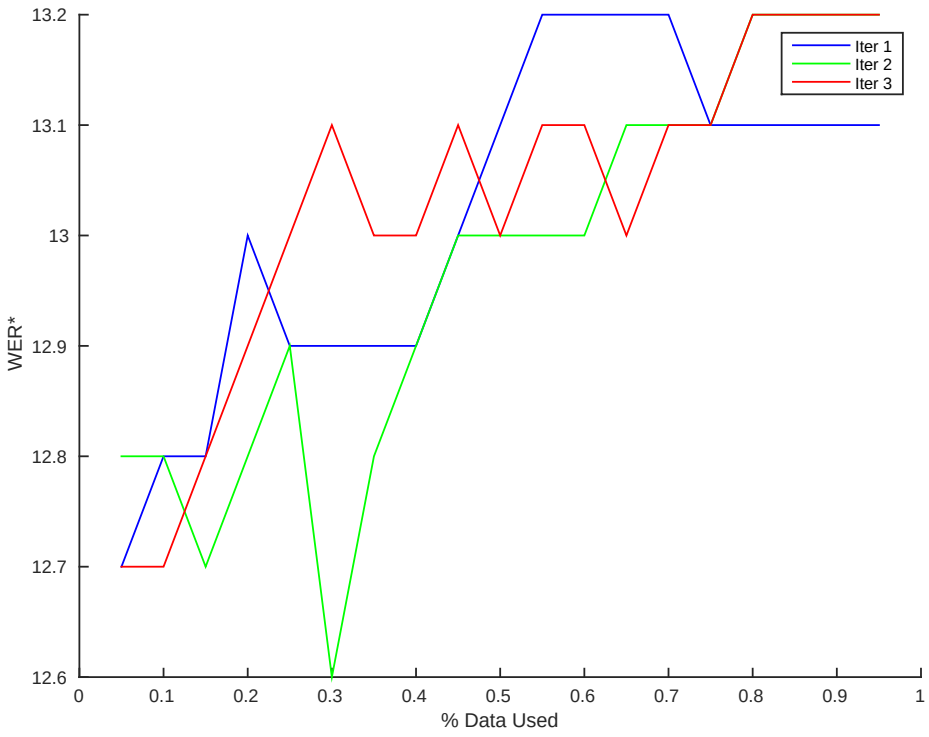

Figure 6.3: Self-training tuning results for the OOV detection task on P2-DEV, WER* results for different iterations and different amounts of data used at each iteration.

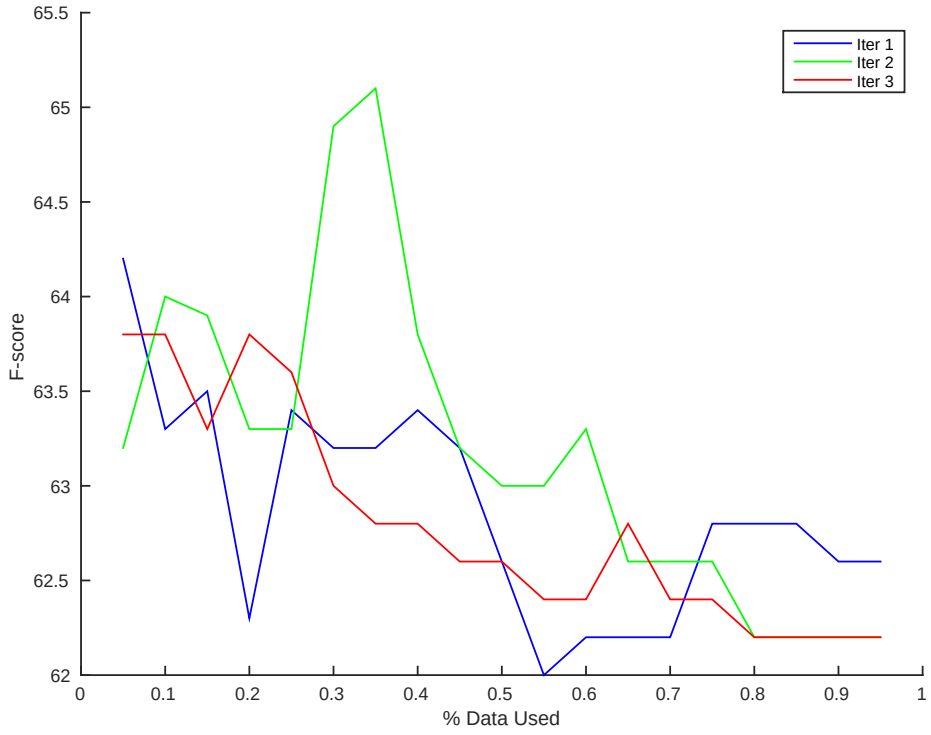

Figure 6.4: Self-training tuning results for the OOV detection task on P2-DEV, F-score results for different iterations and different amounts of data used at each iteration.

Configurations		Best Parser		Corresp. MaxEnt	
Trees	Relabeling	WER*	F-score	WER*	F-score
Uniform Weight Baseline		14.4	51.4	11.4	62.6
all	full	13.2	62.9	11.1	67.6
all	partial	13.2	62.9	11.1	67.6
error	full	12.7	64.1	11.1	66.2
error	partial	12.7	64.1	11.1	66.2

Table 6.13: Self-training for OOV detection: no-rescoring model uniform weight initialization, P2-DEV.

Configurations		Best Parser		Corresp. MaxEnt	
Trees	Relabeling	WER*	F-score	WER*	F-score
Uniform Weight Baseline		19.8	20.5	13.3	30.0
all	full	16.7	31.3	13.3	31.5
all	partial	16.7	31.3	13.3	31.5
error	full	17.3	29.0	13.2	31.2
error	partial	17.3	29.0	13.2	31.2

Table 6.14: Self-training for OOV detection: no-rescoring model with uniform weight initialization, P3-DEV.

score, whereas for P3-DEV the benefit from self-training is more minor, of only about 1.5% F-score. In both cases, no additional benefit is obtained from relabeling the entire dataset vs. only the data not already selected, with the self-training procedure yielding best results in the first iteration. For BOLT-P3, the best configuration is obtained using only the error trees; for BOLT-P2 the behavior is mixed, with error trees giving the best configuration in the parser stage but all trees giving better second MaxEnt stage results.

Configurations		Best Parser		Corresp. MaxEnt	
Trees	Relabeling	WER*	F-score	WER*	F-score
Uniform Weight Baseline		17.1	16.4	12.5	42.7
all	full	14.2	24.1	12.6	38.3
all	partial	14.2	24.1	12.6	38.3
error	full	14.3	25.2	12.4	45.3
error	partial	14.3	25.2	12.4	45.3

Table 6.15: Self-training for name error detection: no-rescoring model with uniform weight initialization, P3-DEV.

Results for name error detection are shown in table 6.15. Again, significant benefit is obtained in the parser stage using both WER* and F-score metrics, in particular with F-score improvements of almost 9% absolute. This improvement is obtained using only the error trees, again with no difference between relabeling strategies, due to the best performance being obtained in the first iteration. This approach also gives a (more modest) improvement in the second MaxEnt stage in terms of F-score, with performance essentially tied when measuring WER*.

Experiment 2b: Starting from the no-rescoring model with best log-linear adaptation for error weights

Next, we examine the performance of self-training when starting with the no-error model adapted for error detection using the best log-linear adaptation configuration for each task.

Configurations		Best Parser		Corresp. MaxEnt	
Trees	Relabeling	WER*	F-score	WER*	F-score
Uniform Weight Baseline		12.9	59.6	11.3	67.4
all	full	12.5	63.8	11.5	65.3
all	partial	12.5	63.8	11.5	65.3
error	full	12.5	64.8	11.2	64.8
error	partial	12.5	64.8	11.2	64.8

Table 6.16: Self-training for OOV detection: no-rescoring model with error rule weights initialized using log-linear adaptation, P2-DEV.

Configurations		Best Parser		Corresp. MaxEnt	
Trees	Relabeling	WER*	F-score	WER*	F-score
Uniform Weight Baseline		15.0	25.9	13.4	31.0
all	full	16.3	29.2	13.6	30.9
all	partial	16.3	29.2	13.6	30.9
error	full	16.6	28.6	13.4	31.3
error	partial	16.6	28.6	13.4	31.3

Table 6.17: Self-training for OOV detection: no-rescoring model with error rule weights initialized using log-linear adaptation, P3-DEV.

Results for OOV detection are presented in tables 6.16 (for the configuration using the BOLT-P2 dataset) and 6.17 (for the configuration using the BOLT-P3 dataset). In both cases, self-training provides a slight benefit in the parser stage, in terms of F-score (with improvements of 3-4% absolute). In terms of WER*, self-training gives a small win for BOLT-P2 training, but no win for BOLT-P3 training. Performance of the second MaxEnt stage is also mixed. For the BOLT-P2 case, there is a slight degradation of performance using all self-training configurations, while performance using WER* is at best tied. For BOLT-P3, only a slight improvement in terms of F-score is obtained, with WER* performance again tied. In all cases, no benefit is obtained from the different relabeling strategies, with the best configurations all reached in the first self-training iteration; performance between the different tree selection strategies is mixed, though the differences are never large.

Configurations		Best Parser		Corresp. MaxEnt	
Trees	Relabeling	WER*	F-score	WER*	F-score
Uniform Weight Baseline		17.1	16.4	12.5	42.7
all	full	14.5	25.7	12.6	37.9
all	partial	14.5	25.7	12.6	37.9
error	full	15.0	22.7	12.6	36.3
error	partial	15.0	22.7	12.6	36.3

Table 6.18: Self-training for name error detection: no-rescoring model with error rule weights initialized using log-linear adaptation, P3-DEV.

Results for name error detection are presented in table 6.18. As in the case of OOV detection, self-training yields significant improvement in the parser stage, with the best performance obtained using all trees as candidates for addition to the treebank. However, this improvement in the parser stage does not translate into an improvement in the second MaxEnt stage; on the contrary, performance degradation is observed in the second MaxEnt stage when measuring F-score, with WER* performance tied to the baseline. In all cases, the best performance is obtained in the first self-training iteration of each training condition, so changing the relabeling strategy does not provide any additional benefit.

Experiment 3: Self-Training Starting with Best Rescoring Parser

In the next experiment, we perform self-training starting with the best rescoring parser and either default weights for error rules (in experiment 3a) or the log-linear adapted models (in experiment 3b). We examine results for OOV detection as well as name error detection.

Experiment 3a: Starting from the best rescoring model with uniform error weights

For OOV detection we examine results using both the BOLT-P2 and the BOLT-P3 datasets. Results using the BOLT-P2 data are shown in table 6.19. We find that the parser stage results across all the configurations are very close. For the second MaxEnt stage, using all the trees at each iteration tends to yield a slight performance drop vs. relabeling only the low-confidence trees, though the differences are small. We also notice that, in general, using all the trees (regardless of whether or not they contain error regions) gives better results than using just trees with error regions, though again the results are close. The best configurations outperform the baseline no-adaptation parser by 3-4 F-score points in both parser stage and second MaxEnt stage.

Table 6.20 summarizes results for self-training adaptation using the BOLT-P3 dataset starting from the baseline parser model. We see that all configurations improve upon the baseline in the parser stage, both in terms of WER* and F-score, with only minor differences between the configurations; slightly better results are obtained when all the trees are considered for adaptation, rather than just those containing error regions. In both cases only one iteration of self-training suffices to reach the best results. Examining the second MaxEnt stage results, we see more variation, though all configurations tie or slightly outperform the baseline; here, we obtain best results from using just trees containing error regions, with slightly higher F-score results when relabeling only unused utterances.

Examining results cross-dataset, we see that the improvements in the parser stage are higher in the BOLT-P3 dataset case than when using BOLT-P2 data, though the baseline is much lower. The best absolute improvement in the second MaxEnt stage is similar in each case, though different configurations yield best results in each dataset (using all the trees with either full or partial relabeling for BOLT-P2, and using only the error trees with

Configurations		Best Parser		Corresp. MaxEnt	
Trees	Relabeling	WER*	F-score	WER*	F-score
Uniform Weight Baseline		12.8	60.3	11.0	65.2
all	full	12.7	64.9	10.9	68.1
all	partial	12.7	64.9	10.9	68.1
error	full	12.6	64.9	11.2	65.7
error	partial	12.7	64.4	11.0	67.4

Table 6.19: Self-training for OOV detection: best rescoring model with uniform weight initialization, P2-DEV.

Configurations		Best Parser		Corresp. MaxEnt	
Trees	Relabeling	WER*	F-score	WER*	F-score
Uniform Weight Baseline		19.8	20.5	13.3	30.0
all	full	16.2	31.9	13.4	29.9
all	partial	16.2	31.9	13.4	29.9
error	full	16.4	31.6	13.3	31.7
error	partial	16.4	31.7	13.4	32.2

Table 6.20: Self-training for OOV detection: best rescoring model with uniform weight initialization, P3-DEV.

partial relabeling for BOLT-P3).

Configurations		Best Parser		Corresp. MaxEnt	
Trees	Relabeling	WER*	F-score	WER*	F-score
Uniform Weight Baseline		17.1	16.0	12.5	42.7
all	full	14.0	27.1	12.5	38.5
all	partial	13.8	26.9	12.6	38.0
error	full	13.7	26.2	12.5	41.1
error	partial	13.7	26.2	12.5	41.1

Table 6.21: Self-training for name error detection: best rescoring model with uniform weight initialization, P3-DEV.

Results for name error detection starting from the baseline parser model are shown in table 6.21. As before, we compare the baseline system (with the parser used in rescoring adaptation) with four systems, obtained from self-training adaptation with all trees or only trees containing error regions, and full relabeling or partial, respectively. We find that all configurations improve over the baseline parser performance in the parsing stage, with only slight differences between the various configurations. In the second MaxEnt stages, we find that the error tree-only configurations outperform the configurations using all the trees as candidates, though no configuration performs better than the baseline.

Experiment 3b: starting from the best rescoring model with best log-linear adaptation for error detection

Results for self-training for OOV detection when the baseline parser is the best log-linear adapted parser model and using the BOLT-P2 data are shown in table 6.22. As before, we compare the baseline parser-based results against a variety of self-training approaches. All systems achieved their best performance within a single self-training iteration, thus the relabeling strategy did not affect results. Only minor differences were obtained from the choice of using only trees with errors vs. all trees. In no case do we outperform the baseline

Configurations		Best Parser		Corresp. MaxEnt	
Trees	Relabeling	WER*	F-score	WER*	F-score
Best LL Adapted Baseline		11.6	69.3	11.4	66.7
all	full	12.0	67.5	10.7	66.7
all	partial	12.0	67.5	10.7	66.7
error	full	12.0	67.3	11.0	66.9
error	partial	12.0	67.3	11.0	66.9

Table 6.22: Self-Training for OOV detection: best rescoring model with error rule weights initialized using log-linear adaptation, P2-DEV.

Configurations		Best Parser		Corresp. MaxEnt	
Trees	Relabeling	WER*	F-score	WER*	F-score
Best LL Adapted Baseline		14.2	36.9	12.8	36.9
all	full	16.0	31.8	12.8	37.2
all	partial	16.0	31.8	12.8	37.2
error	full	15.9	32.9	13.1	32.8
error	partial	15.9	32.9	13.1	32.8

Table 6.23: Self-training for OOV detection: best rescoring model with error rule weights initialized using log-linear adaptation, P3-DEV.

parser, though the self-training systems match or very slightly improve upon the second MaxEnt stage baseline.

Results for the equivalent condition using the BOLT-P3 data are shown in table 6.23. We present two sets of results, using all trees or only trees with errors; both configurations required only one iteration to attain their maximum performance, thus the relabeling strategy did not impact performance in either configuration. No self-training configuration outperforms the baseline parser stage; on the other hand, in their respective second MaxEnt stages, one of the two configurations (using all the high-scoring trees) leads to minor F-score improvements, with no degradation in terms of the WER* metric.

Looking at performance across training dataset configurations, we obtain no improvement in the parser stages of either condition, and only minor improvements in the MaxEnt stages. The F-score improvements are similar, though the BOLT-P2 condition does yield WER* improvements, indicating that the extent of the error regions detected was improved. This improvement is not observed in the BOLT-P3 condition.

Configurations		Best Parser		Corresp. MaxEnt	
Trees	Relabeling	WER*	F-score	WER*	F-score
Best LL Adapted Baseline		12.7	38.8	12.5	42.5
all	full	13.2	29.8	12.6	37.5
all	partial	13.2	29.8	12.6	37.5
error	full	13.4	31.9	12.4	41.5
error	partial	13.4	31.9	12.4	41.5

Table 6.24: Self-training for name error detection: best rescoring model with error rule weights initialized using log-linear adaptation, P3-DEV.

Results for self-training for name error detection starting with the best log linear-adapted parser model are presented in table 6.24. In all configurations, performance of the parser stage is significantly degraded vs. the model trained using log-linear adaptation, so the adaptation stops after only one iteration. There are no differences between relabeling strategies. In the second MaxEnt stage, the self-trained configuration using only high-scoring error

trees is essentially tied, with minor performance degradation when comparing F-scores, but a very slight improvement in terms of the WER* metric.

6.3 Analysis

6.3.1 Final Results

We performed the tuning of the various adaptation methods using the dev partitions of the BOLT-P2 and BOLT-P3 datasets, P2-DEV and P3-DEV, respectively. We perform a final assessment of the different methods on the internal P2-EVAL and P3-EVAL sets, as well as the external NIST EVAL set, for each of the tasks. For each error detection task-dataset pair, we compare the effect of adapting the error parser starting from two distinct no-error parser configurations:

- **no resc-adapt**: a no-error parser trained on only out-of-domain treebank;
- **resc.-ST**: a parser trained on out-of-domain treebank, with self-training applied for rescoring (as done in chapter 5);

In both cases, the error parser is obtained by adding error rules to the no-error parser, with error rule weights set to default values as described in section 4.1.2.

For each no-error parser configuration as described above, we examine the performance using four different configurations for adaptation of the error parser only, as follows:

- **no <Err>-adapt**: no adaptation of the error model is performed, so all error rules have default weights;
- **<Err>-LL**: the error model is adapted using log-linear adaptation, using the corresponding **no <Err>-adapt** model as the starting point;
- **<Err>-ST**: the error model is adapted using self-training, using the corresponding **no <Err>-adapt** model as the starting point;
- **<Err>-LL + <Err>-ST**: the error model is adapted using self-training, using the corresponding**<Err>-LL** model as the starting point.

The adaptation configurations correspond to the best configurations obtained from tuning on the respective development sets in previous sections. In all cases, "<**Err**>" refers to the task of interest, i.e. either **OOV** (for OOV detection) or **NErr** (for name error detection).

The final results for OOV detection using the BOLT-P2 dataset are shown in table 6.25. In general, parser adaptation helps, regardless of which no-error model is used as a starting point, in terms of both WER* and F-score. The improvements are larger in the parser stage, with consistent gains of 5-10% absolute in terms of F-score over the no-adaptation scenarios; in the second MaxEnt stages, improvements from adaptation are smaller, though consistent under most adaptation scenarios. We also find that the best parser stage performance comes close to matching the best second MaxEnt stage performance.

Comparing different configurations, we find that self-training tends to produce more consistent improvements across the two test sets, though we do not obtain a consistent win from starting self-training with a model already adapted using log-linear adaptation vs. the model without adaptation for OOV detection. Starting from the no-error model adapted for rescoring generally yields some improvement over models starting from a no-error model without adaptation for rescoring, consistent with the improvements obtained from adaptation for the rescoring task, as shown in chapter 5.

Similar behavior is observed for OOV detection using the BOLT-P3 dataset. Results for this configuration are shown in table 6.26. Adaptation consistently helps both stages, with F-score improvements of 8-10% absolute in both datasets for the parser stage, and 3-5% in the second MaxEnt stage. Performance measuring WER* is more mixed, though generally the best adaptation configuration leads to a slight improvement. Comparing the performance across stages, again, the best parser stage results are close, though never quite outperform, the best MaxEnt stage results, particularly in terms of F-score.

Comparing behavior of different adaptation approaches, self-training is competitive with log-linear adaptation, primarily in the second MaxEnt stage, with results more consistent across the two different datasets. This result holds true for configurations starting from either no-error model. In this case, performance between configurations using the two starting point no-error models is evenly matched, consistent with the observation that we did not obtain an improvement from adaptation of the no-error model for rescoring.

Rescoring Adaptation	OOV Adaptation	P2-Eval Best Parser WER*	P2-Eval Best Parser F-score	P2-Eval Best MaxEnt WER*	P2-Eval Best MaxEnt F-score	NIST Eval Best Parser WER*	NIST Eval Best Parser F-score	NIST Eval Best MaxEnt WER*	NIST Eval Best MaxEnt F-score
first MaxEnt stage		n/a	n/a	12.6	64.7	n/a	n/a	6.3	30.7
none	none	14.6	54.2	12.2	65.7	10.0	22.6	6.3	31.5
	OOV-LL	14.1	59.1	11.6	71.9	9.2	25.6	5.8	33.9
	OOV-ST	13.1	65.8	12.1	70.0	9.0	24.8	5.9	37.1
	OOV-LL + OOV-ST	13.5	63.0	12.3	68.6	7.9	26.0	6.3	33.2
resc.-ST	none	13.3	61.9	12.1	67.9	7.6	27.7	6.0	36.9
	OOV-LL	12.4	70.9	12.7	66.2	6.2	40.0	7.0	31.6
	OOV-ST	13.3	65.3	11.7	72.5	8.0	33.2	6.1	38.3
	OOV-LL + OOV-ST	12.6	68.9	11.2	73.4	7.4	36.5	6.0	40.1

Table 6.25: OOV detection results using the best systems, BOLT-P2.

Rescoring Adaptation	OOV Adaptation	P3-Eval Parser Stage WER*	P3-Eval Parser Stage F-score	P3-Eval MaxEnt Stage WER*	P3-Eval MaxEnt Stage F-score	NIST Eval Parser Stage WER*	NIST Eval Parser Stage F-score	NIST Eval MaxEnt Stage WER*	NIST Eval MaxEnt Stage F-score
first MaxEnt stage		n/a	n/a	12.2	33.6	n/a	n/a	5.8	36.2
none	none	18.2	21.8	12.2	36.0	9.0	24.7	5.7	38.6
	OOV-LL	14.3	30.0	12.6	33.8	6.1	41.6	5.8	41.1
	OOV-ST	15.5	31.1	12.2	39.1	7.0	36.2	5.6	42.6
	OOV-LL + OOV-ST	15.2	28.9	12.3	35.7	7.0	32.7	5.8	38.4
resc.-ST	none	13.9	22.7	12.3	34.4	6.5	33.5	5.8	40.2
	OOV-LL	12.5	41.9	11.9	38.7	5.9	40.6	5.6	35.5
	OOV-ST	15.9	31.9	12.3	35.7	6.6	39.7	5.6	42.2
	OOV-LL + OOV-ST	14.8	32.5	12.0	37.8	6.7	39.5	5.6	43.3

Table 6.26: OOV detection results using the best systems, BOLT-P3.

A few trends hold across the two training set (BOLT-P2 and BOLT-P3) configurations: adaptation helps; self-training gives more consistent behavior than log-linear adaptation; the parser stage results are competitive with the second MaxEnt stage results. The main differences between training configurations are in terms of the effect of the adaptation for rescoring, which helps in the BOLT-P2 case but not for BOLT-P3, with the effect carrying over to OOV detection as well. Comparing the best results from each configuration on the shared NIST Eval set, the BOLT-P3 training configuration gives a slight win, with F-score results consistently higher than in analogous configurations using BOLT-P2 training, in both stages.

Results for name error detection using the BOLT-P3 data are shown in table 6.27. Some of the trends observed with OOV detection also hold in name error detection: all adaptation methods targeting the name error task directly help in the parser stage in terms of F-score, and they almost always yield a small gain in terms of WER*. Adaptation generally yields a win in the second MaxEnt stage, too, though this time performance of the self-training configurations is not always competitive with the log-linear adaptation. Since at least in some cases the best log-linear adaptation results are obtained using adaptation of the reduced set of rules hand-picked to target name errors, we hypothesize that these results show log-linear adaptation can outperform self-training when the adaptation can be focused on specific rules.

For OOV detection using the BOLT-P3 data, the choice of initial no-error model did not have a significant impact in the best results for OOV detection, despite not obtaining a win from adaptation using rescoring (as shown in chapter 5). For name errors, the effect of the lack of improvement in the rescoring task appears to be stronger, with name error-targeted adaptation results starting from the no-error model without adaptation almost always outperforming the name error adaptation results starting from the error model with adaptation in the second MaxEnt stage. For both OOV and name error detection, adding the second MaxEnt stage provides some benefit, though the benefit is larger in the name error detection task, perhaps because syntactic features are not sufficient to capture long-distance lexical context. The issue of capturing lexical context that is useful for name error detection is addressed in chapter 7.

Rescoring Adaptation	Name Error Adaptation	P3-Eval Best Parser WER*	P3-Eval Best Parser F-score	P3-Eval Best MaxEnt WER*	P3-Eval Best MaxEnt F-score	NIST Eval Best Parser WER*	NIST Eval Best Parser F-score	NIST Eval Best MaxEnt WER*	NIST Eval Best MaxEnt F-score
first MaxEnt Stage		n/a	n/a	11.7	38.1	n/a	n/a	5.7	36.0
none	none	14.7	17.4	11.6	43.5	7.8	17.2	5.6	42.2
	NErr-LL	12.5	37.8	11.5	51.8	5.7	41.7	5.6	46.2
	NErr-ST	13.1	24.9	11.5	50.5	6.1	27.7	5.6	40.9
	NErr-LL + NErr-ST	13.3	26.9	11.6	47.2	6.1	31.2	5.6	41.4
resc.-ST	none	12.5	21.3	11.6	48.3	6.0	24.9	5.6	38.7
	NErr-LL	11.6	43.2	11.5	44.8	5.6	37.4	5.6	38.1
	NErr-ST	13.0	25.2	11.6	46.2	6.0	32.1	5.6	41.7
	NErr-LL + NErr-ST	12.6	29.0	11.5	45.7	5.9	32.4	5.6	40.0

Table 6.27: Name error detection results using the best systems, BOLT-P3.

6.3.2 Effect of Parse Features

Section 4.2.2 presented a discussion of features used in the first MaxEnt stage vs. the second MaxEnt stage, to analyze the effect of parse features when the parser was trained on only out-of-domain treebank data. As a follow-up, we examine the effect of domain adaptation for parsing on the effectiveness of the parse features. Table 6.28 shows the highest-weight features for two second MaxEnt stage classifiers built for OOV detection using the BOLT-P2 datasets: the second MaxEnt stage built upon a parser with no adaptation (i.e. the configuration in 4.2.2) and the best adaptation configuration, using log-linear adaptation followed by self-training. In both cases, most of the highest (positive) weight features are parse tuples, with three features appearing in both configurations (including one of the dependency tuple difference features). Conversely, the feature with the lowest negative weight (i.e. most strongly correlated with the negative label) is the *POSRelDiff* parse tuple difference feature, comparing differences both in the attachment constituents and POS tags (thus, in regions predicted by the parser as OOV, this feature is guaranteed to be non-zero). A few dependency tuple features also have low negative weight in each model, with the remaining features with strong negative weights all coming from the confusion network structure.

Examining the differences between the parse features used in the two models, it can be observed that, in the unadapted model, most dependency tuple features relate to nouns and verbs (i.e. NP and VP constituents), whereas the structure of the relationships captured in features from the best model using adaptation tends to be more mixed. The mixed behavior is observed both in the highest-weight features and in the lowest-weight features; in the former, the best adaptation model uses features related to adjectives, adverbs, particles, and NULL arcs. The NULL arc rules, in particular, are probably useful because they may correct the extent of OOV regions. Among the lowest negative weight features, the adapted parser also uses features containing sentence-level constituents, perhaps indicating high confidence in words that attach close to the top-level nodes in the parse tree.

It is interesting to note that in neither case do the inside score features figure very prominently; in the unadapted model, the *local_inside_score* feature is in the top 20 features

most associated with the negative class, while in the adapted model, it has lower weight, appearing in the top 5% of the features with negative weight (i.e. associated with the negative class). The *global_inside_score* feature is in the bottom 20% of the features in each model, ranked by weight.

No-adaptation		Best adaptation	
Feature	Weight	Feature	Weight
OOV_VP_VP_OOV	1.98	VB_VP_NP_OOV	1.27
NN_NP_NN_OOV	1.67	OOV_PRT_PRT_OOV	1.23
__VP_OOV	1.67	PRT_PRT	1.23
IN_SBAR_S_VB	1.58	DEL_PP_PP_DEL	1.15
VB_VP_NP_OOV	1.46	ADVP_JJS	1.15
relOnlyDiff	1.44	OOV_S_S_OOV	1.11
OOV_NP_NP_OOV	1.44	IN_PP_NP_DEL	1.05
TO_S_NP_OOV	1.39	CLASS-24750	1.02
CLASS-24750	1.38	OOV_NP_NP_OOV	0.98
CLASS-33639	1.35	relOnlyDiff	0.97
that	-1.06	that	-0.73
NP_INTJ	-1.06	VP_S_NP_NNS	-0.79
NN_NP_NP_NN	-1.10	_NP	-0.79
NNS_NP_NN_NN	-1.13	leftPost	-0.82
VBZ_S_NP_NN	-1.39	VB_S_NP_NN	-0.91
leftPost	-1.41	nnConf	-1.05
nnConf	-1.42	SBAR_S	-1.07
stdDev	-1.56	rightPost	-1.09
rightPost	-1.87	stdDev	-1.38
POSRelDiff	-2.24	POSRelDiff	-1.86

Table 6.28: Comparison of the highest-weight features in the second MaxEnt stage systems built using a parser without domain adaptation vs. the best domain-adapted parser, BOLT-P2 OOV detection.

6.3.3 Comparison of Model Sizes and Parsing Speed

Finally, we present a short discussion about model sizes. We focus this discussion on models used for OOV detection using the BOLT-P3 data, though the conclusions hold for the other task-dataset pairs as well. For each of the error models used in table 6.26 we present the size of the lexicon, the number of constituents, as well as the number of unary and binary rules. The statistics are displayed in table 6.29. Statistics for two sets of models are displayed, starting with the no-error model built using the CTS-Small treebank with no adaptation (i.e. rescoring adaptation set to **none**), as well as starting with the no-error model built using the CTS-Small treebank with self-training adaptation using the LMTrain-Small and P3-Train data (i.e. **resc.-ST**).

For each model used as a starting point (the "no error rules" conditions), adding error rules to generate the no-OOV adaptation models increases significantly the size of the unary and binary rule set; the number of constituents increases only to add the error markers while the vocabulary remains fixed. Performing log-linear model adaptation does not change the model size, adjusting only the rule weights for any unary or binary rules included in the adaptation set.

Unlike log-linear adaptation, self-training does change the model statistics. For each self-training configuration, the lexicon size increases, reflecting the addition of new trees (potentially containing previously-unseen words) to the treebank. The change is biggest comparing the no-OOV-adaptation conditions, with self-training for rescoring more than doubling the size of the lexicon while also increasing the number of unary rules (by about 10%) and the number of binary rules (by about 40%); the latter changes primarily due to automatic constituent splitting.

Examining the effect of self-training for OOV detection, the lexicon size once again increases, though by a smaller amount, reflecting that only the speech training data (i.e. P3-Train) is used for adaptation in this case. This results in an increase in lexicon size by about 20% for the model trained only on hand-labeled CTS-Small treebank. For the model already adapted for rescoring, the lexicon increase is negligible, reflecting the fact that the model already included many of the words used in BOLT data but not present in

the treebank data.

While performing self-training for OOV detection still increases the lexicon size, the process has the opposite effect on the rule set sizes. After self-training, the unary rule set is only about 20% larger than the no-error condition (as opposed to a 15x increase when all the default error and NULL arc rules are added); the binary rule set increases by about 3-5% (instead of doubling when all the default rules are added). The number of rules being added varies slightly depending on the starting model used during the self-training procedure, reflecting the differences in the best adaptation configurations.

Rescoring Adaptation	OOV Adaptation	# words	# constituents	# unary rules	# binary rules
none	no error rules	12514	1807	134	5901
	none	12514	1810	1943	9516
	OOV-LL	12514	1810	1943	9516
	OOV-ST	15977	1810	167	6240
	OOV-LL + OOV-ST	15621	1810	174	6118
resc.-ST	no error rules	29334	1835	134	6166
	none	29334	1837	1970	9835
	OOV-LL	29334	1837	1970	9835
	OOV-ST	29359	1837	138	6169
	OOV-LL + OOV-ST	29752	1837	168	6215

Table 6.29: Statistics for the best parser models used for OOV detection using the BOLT-P3 data.

We perform an experiment to assess the impact of the size of different components of the parser in practical terms, measuring wall clock time of parsing the P3-Train and NIST Eval datasets. Four parser configurations are examined, using log-linear adaptation and self-training for OOV detection and starting with either a no-error model with no adaptation or the no-error model adapted for rescoring using self-training. These four sets of configurations give a good coverage of the different model sizes discussed in table 6.29.

All four parsers were run on the same machine: a 6-core Intel Core i7 980 @3.33GHz with 24GB of RAM. Each parser was run as a single-threaded application.

Results for the parser speed experiment are shown in table 6.30. In all cases, the self-training models are faster than the corresponding log-linear adapted models; the improvement from self-training appears more prominent in the configurations starting with a no-error model already adapted for rescoring. These two models are in fact slightly faster, despite the larger lexicon, showing that the lexicon size is not a factor in the parser speed. Of more importance, likely, is the fact that better models may allow for a faster generation of the constituent trees using a more effective search through the parse chart. Parsing the longer sentences in P3-Train ends up taking, on average, twice as long as parsing the sentences in NIST Eval, consistent with the superlinear algorithm used in filling the chart; with the longer P3-Train sentences, performance of both self-training models is in fact better than performance of each log-linear model, as expected from the smaller model sizes in the two self-training configurations.

Rescoring Adaptation	OOV Adaptation	Average Sentence Parse Speed (ms) P3-Train	NIST Eval
none	OOV-LL	1814	906
	OOV-ST	1722	888
resc.-ST	OOV-LL	1767	885
	OOV-ST	1460	747

Table 6.30: Average parser speed (ms) for parsing one sentence using models trained using different configurations.

6.3.4 Discussion

We find that both adaptation methods significantly improve the error detection system when the parser itself is used as an error detector, with performance almost doubling in many cases. Results with the subsequent second MaxEnt stages using parse features do not

give as clear of a win from adaptation, though in all task-dataset pairs examined, at least one adaptation strategy outperforms the parser-based systems with no adaptation.

Comparing results when initializing the adaptation with the no-rescoring parser vs. the best rescoring parser, we find that, in most cases, the parser stage results benefit from the better starting point when applying log-linear adaptation. However, this improvement often does not carry through to the second MaxEnt stage for OOV detection. In fact, it is often the case that when starting with the best rescoring parser, the second MaxEnt stage no longer provides a win in OOV detection, whereas when starting with the no-rescoring parser the second MaxEnt stage helps. However, the differences between the best results over all stages of the no-rescoring and best rescoring setups are small for most datasets; thus, using the best rescoring setup is preferable when we already get a win from parsing for rescoring, especially since it may allow the elimination of the second MaxEnt stage, simplifying the system.

Comparing performance of the BOLT-P2 and BOLT-P3 based OOV detection systems, we find that, in almost all situations, the BOLT-P3 system slightly outperforms the corresponding BOLT-P2 system; this holds true with both adaptation methods used separately, as well as in combination. We conclude that having more in-domain training data for adaptation helps more than having data with a higher frequency of errors (or, more generally, positive samples), even despite the skew in the distribution of class labels.

Comparing the two approaches for adaptation, we find that, in general, self-training performance is more consistent than log-linear adaptation alone. This consistency tends to manifest itself in two ways; first, with self-training systems, the second MaxEnt stage generally improves upon the parser stage, even if the improvements are sometimes small; second, performance across test sets is somewhat predictable, with improvements carrying from the dev set to the internal and external eval sets. This consistency is often lacking with the log-linear adaptation systems. On the other hand, using log-linear adaptation outperforms self-training in some cases. Thus, the decision of when to use one method over the other is likely task- and dataset-dependent.

One advantage of the self-training approach over log-linear adaptation regardless of configuration is the relatively smaller size. In particular, the number of error rules is much

smaller when the final parser model is learned through self-training, rather than using log-linear adaptation to adjust the weights of automatically-generated rules. Since the systems which use self-trained parsers are competitive with those built using only log-linear adaptation, a reasonable strategy would be to first build the best possible parser using log-linear adaptation, then do a round of self-training to reduce the grammar size. At best this would risk a small performance degradation, but with the benefit of a significant drop in grammar size.

Chapter 7

LEARNING ROBUST TASK-INFORMED PHRASE PATTERN FEATURES USING SEMANTIC SIMILARITY

In the previous chapter, we showed that with weakly-supervised adaptation, parsing improves the ability of the system to automatically detect ASR errors, both using a parser as an error detector and by extracting features for use in a maximum entropy classification framework. We obtained improvements in detection performance for two types of errors, general OOVs as well as name errors (where the name may be either in-vocabulary or OOV). Thus, local syntactic structure is useful for detecting such errors. However, information related to whether a particular word is a name or not is sometimes located far away from the name itself; the types of syntactic features we use do not always capture such phenomena well, in particular when the parse output itself may have errors. The long distances separating a name from the relevant lexical cues may occur, for example, when multiple names occur in the same sentence. In conversational data, such phenomena may also occur due to disfluencies or other informal constructions. Consider the following example:

okay so we'll *start in* LOC_**Basra** *looking for* PER_**Ali** and i think you said PER_**Mathar** (7.1)

In this sentence, we have three names, one location (**Basra**) and two person names (**Ali** and **Mathar**). We also highlighted the lexical cues most relevant to name presence. The two person names belong to a list but are separated by the parenthetical phrase *I think you said*. Thus, if the word Mathar were misrecognized, local lexical cues would not suffice for identifying the error region as a name. So far we have addressed this problem using syntax. In this chapter, we explore an alternate approach, learning long-distance lexical context features (i.e. phrase patterns), and investigate their use in both the maximum entropy classification framework and directly during parsing, and compare results for each case.

A second focus of this chapter is the reliability of the classifiers. In the previous chapter, we briefly explored the issue of overfitting in the context of the different stages of processing, primarily in terms of which stage is more likely to lead to overfitting. Intuitively, lexical context features are expected to be more prone to overfitting. The following two sentences present a typical example of overfitting problems.

Train: hi my name is *lieutenant* PER_**Smith** (7.2)

Eval: you should talk with *colonel* PER_**Hassan** (7.3)

The lexical cues relevant to names in these two sentences are the two military ranks, *lieutenant* and *colonel*. However, if the word "colonel" is never (or rarely) seen in training, its corresponding feature will not be given a high (or any) weight in the classifiers. We will aim to learn robust word classes which can be used to alleviate this issue. The approach used in this thesis involves unsupervised expansion of word classes from seed words and phrases, which are either selected using domain knowledge or learned through discriminative methods. In recent prior work with collaborators on the same data [55], word classes were learned using word co-occurrence statistics using the Brown algorithm and using RNNLM embeddings and k-means clustering; these classes were used in n-gram features. The new feature learning method used in this thesis advances over the prior work, by explicitly using semantic similarity for learning word classes using a local neighborhood approach, as well as by employing phrase patterns, which are an extension of n-grams allowing skips between words used in a variety of text classification tasks [63, 154, 140, 143].

The organization of this chapter is as follows. Section 7.1 introduces the proposed approach to feature learning. We present a method for learning word class definitions using semantic similarity, starting from either a hand-labeled or a discriminatively-learned seed feature set. We also introduce an algorithm for learning long distance phrase patterns over the previously-learned word classes. The feature extraction process using the new features is presented in section 7.2. We present a brief experiment using only baseline lexical context features in section 7.3, to motivate the approach used in this thesis. Next, we describe experiments in name error detection to assess the impact of the proposed features when used in the first MaxEnt classification stage of the error detection pipeline, in section 7.4.

Section 7.5 discusses the impact of the phrase pattern features through experiments using the parsing and second MaxEnt stage configurations. We conclude with a summary of the findings in section 7.6.

We will answer a number of questions in this chapter. In particular, we are interested in the following issues:

- What are the relative benefits of local vs. long-distance context in the first MaxEnt stage classifier using words vs. word classes?

- What differences in performance do we obtain from the different methods for scoring semantic similarity in learning word classes?

- How much impact does the context-aware first MaxEnt stage have on parsing?

7.1 Learning Semantic Similarity-based Lexical Features

We design a new method (outlined in algorithm 2) to learn task-oriented lexical features based on word-level semantic similarity measures. Unlike purely supervised algorithms, which incorporate the task objective directly into the feature learning, we achieve the supervision in a first step of seeding the algorithm with discriminative words or phrase patterns learned or hand-specified from a small training set. If the seed set consists of only single words, an optional step is used to automatically extract phrase patterns from the training set; these automatically-extracted phrase patterns thus become the new seed set. The features defined by the seed set are generalized to improve coverage, with POS filtering applied to preserve the syntactic structure of the seeds. The feature space dimensionality remains unchanged; all words or phrase patterns obtained through the generalization process for one seed feature are mapped back to that same initial feature as members of that class during the run-time feature extraction step.

In practice, steps 2 and 3 are combined into a single algorithm, as we will discuss in the following subsections when we describe each step of the algorithm in more detail.

Algorithm 2 Phrase pattern learning algorithm.

INPUT: F_0 - seed set of features useful to task of interest

OUTPUT: F - feature class definitions

if F_0 is composed of unigram features, as an optional step **then**

$F_1 \leftarrow$ automatic phrase pattern extraction(F_0)

else

$F_1 = F_0$

end if

F_2 =Generalize feature set F_1 using semantic similarity (to improve coverage)

F_3 =Filter generalized feature set F_2 using POS information (to preserve syntactic class structure)

Return: $F = F_3$

7.1.1 Initial Feature Definition

We seed the algorithm with a set of features known to be of use to the task. This could be done in a variety of ways. We investigate two possible approaches, discussed below.

Hand-crafted Seed Features

The first approach uses a human-generated list of features likely to indicate the presence or absence of names in an utterance, based on inspection of a subset of the utterances in the P3-Train set. We use a set of about 80 features observed in the P3-Train data. The full set of features is included in appendix A. Most features follow one of a small number of syntactic templates. A few representative examples of the seed features, grouped according to the sequence of corresponding POS tags, are shown in table 7.1. We also include in the table the count of features belonging to each group.

Only a few hand-crafted seed features contain more than two words (e.g. *the people of*). We explicitly allow gaps in all multiword features, thus effectively treating them as phrase patterns. The observation that many of the hand-labeled features in fact share a common POS structure motivates the POS filtering process used in the generalization step of the

Syntactic Group	Count	Examples
VB* PP	34	start in, looking for, studying in, hurt by
NN	8	captain, lieutenant, tribe
PRP* NN	7	my name, your rank, your neighbor
DET NN	5	the school, the street, the area

Table 7.1: Examples of hand crafted seed features grouped by their corresponding POS sequences.

feature learning algorithm. Additionally, associated with each seed feature is an indication of which direction the matching process should apply (relative to the current slot) during the feature extraction process described in section 7.2.

Automatically-Generated Seed Features

As automatically-generated seed features we use the features learned by an auxiliary classifier trained on reference transcripts of the BOLT-P3 dataset, targeting the presence or absence of *names* (not name errors) in the sentence, as described in a paper with collaborators [55]. For this auxiliary classifier, each utterance is labeled as positive if it contains at least one name in the reference transcript. All name tokens are removed from the transcript post-sentence level labeling. As classification features the system uses unigram features extracted over the sentence. A feature selection step based on information gain (i.e. mutual information with the class variable), tuned on the P3-DEV set, led to this classifier using the top 170 words. Two configurations are explored in this thesis, using either all 170 words, or using the top 80 words according to their classifier weight, for matched conditions to the hand-crafted feature set.

An additional, optional step involves the generation of phrase patterns from the list of unigram features. For this we use an algorithm similar to the phrase pattern learning approach described in a paper with collaborators at Microsoft [86]. In that algorithm, phrase patterns were extracted separately over clusters of words, with the word clusters learned using agglomerative clustering and TF-IDF features. In this work, the phrase patterns are

extracted over all the seed words. Algorithm 3 below contains the outline of the procedure used in this thesis, which applies frequency filtering to remove rare patterns. The threshold T is set to 5, selected to obtain a set of features that is significantly larger than the initial unigram set while still being computationally tractable during the feature extraction process.

7.1.2 Semantic Feature Expansion

We expand the set of features using a neighborhood-based approach, as detailed in algorithm 4. For each individual word w_i, the corresponding expansion class $C[w_i]$ is defined using a neighborhood-based approach, using a word level, context-independent semantic similarity measure. The expansion class Q_p for a phrase pattern p is given by the Cartesian product of the word expansion classes Q_{pi}. For efficiency, the word-level expansion classes are learned while processing the phrases, and cached for use in future phrases. In the algorithm, the function *sim* is designed to take into account semantic relatedness between words. The size of Q_{pi} is restricted based on the distribution of the similarity scores, limiting to matches of high similarity, and optionally to a fixed size threshold to avoid triggering edge cases where a particular seed word has many more matches of high similarity than most others. We experiment with two sources of semantic relatedness, described in more detail next. The POS-based class filtering is described in subsection 7.1.3.

WordNet Similarity

Many researchers have used WordNet [100] as a source of semantic relatedness for various NLP tasks [156, 101, 89]. A number of different similarity metrics have been defined to take advantage of the WordNet ontology. In this work, we use a word-level similarity measure defined by Lin [80], which provides a score with range $[0, 1]$ where 1 corresponds to maximum similarity. As threshold we require the per-word semantic similarity score to be above 0.9, which was chosen because it leads to expansion classes that are intuitively meaningful, while ensuring that most classes have between 5 and 50 members. The size threshold was set to 50 but in practice it was rarely triggered.

Algorithm 3 Pattern extraction from automatically-generated seed words

Input: S - set of utterances $s = w_1^{n_s}$

Input: F - set of initial unigram features $f_1 \dots f_k$

Input: T - minimum number of times pattern must be seen for it to be selected

Output: P - dictionary of phrase patterns p_i extracted and their counts $P[p_i]$

Initialize $P = \phi$

for sentence $s = (w_1^{n_s})$ **do**

  Extract the longest subsequence $s_c = w_{i_1} \dots w_{i_k}$ such that each $w_{i_j} = f_l$ for some $f_l \in F$

  if $s_c \in P$ **then**

    increment $P[s_c]$

  else

    add s_c to P, $P[s_c] = 1$

  end if

  for each unique subsequence s_d of the sequence s_c, optionally also **do**

    increment $P[s_d]$

  end for

end for

for pattern $p_i \in P$ **do**

  if $P[p_i] <$ threshold T **then**

    remove p_i from P

  end if

end for

Optionally, remove from P any pattern s_d which is a substring of another pattern s_c (i.e. is never maximal)

return P

Algorithm 4 Class expansion from seed words or patterns

INPUT: P - dictionary of phrase patterns extracted

INPUT: V - recognizer vocabulary

INPUT: sim - similarity metric

INPUT: POS - word-to-POS mapping

INPUT: t_{sim} - similarity threshold, used to restrict to only high-similarity matches

INPUT: t_{size} - size threshold, used to restrict the number of matches to a predefined maximum

OUTPUT: Q - set of expanded phrase pattern definitions

initialize expansion set $Q = \phi$

initialize feature expansion cache $C = \phi$

for all patterns $p \in P$, where $p = (w_1^{k_p})$ **do**

  for $i = 1 : k_p$ **do**

    define Q_{pi} as the expansion set for w_i

    if $C[w_i]$ exists **then**

      $Q_{pi} = C[w_i]$

    else

      $Q_{pi} = \{w \in V : \text{sim}(w, w_i) > t_{\text{sim}}$ & $\text{POS}(w) = \text{POS}(w_i)$

      if $|Q_{pi}| > t_{\text{size}}\}$ **then**

        truncate Q_{pi} to t_{size} highest similarity words w_i

      end if

      $C[w_i] = Q_{pi}$

    end if

  end for

  $Q_p =$ cartesian product $Q_{p1} \times \ldots \times Q_{pk_p}$

  $Q = Q \cup Q_p$

end for

return Q

RNNLM-based Similarity

An alternative method to obtain semantic similarity information is through word embeddings [99]. We experiment with a set of word embeddings learned using an RNNLM trained on the LMTrain dataset, as described in a paper with collaborators [55].[1] The embeddings were learned using the RNNLM toolkit described in [98]. Training data is randomly shuffled and 10% sentences are used as a held-out set for language modeling tuning. For training, 10 levels of backpropagation through time were used, i.e. gradients of errors in the network were propagated back 10 times through the recurrent weights. The hidden layer was set to a size of 150 hidden units, based on suggestions from [96]. To obtain a similarity score from the word embeddings, we compute the Euclidean distance between the two embedding vectors, $d(x, y)$:

$$d(x, y) = \left(\sum_i (x[i] - y[i])^2 \right)^{\frac{1}{2}} \tag{7.4}$$

where $x[i]$ and $y[i]$ are the i^{th} components of the embedding vectors corresponding to words x and y, respectively. The distance is normalized to the range $[0, 1]$ using a sigmoid function:

$$d'(x, y) = \frac{1}{1 + \exp(-d(x, y))} \tag{7.5}$$

then take the similarity as

$$sim(x, y) = 2(1 - d'(x, y)) \tag{7.6}$$

with the extra factor needed to map back over the $[0, 1]$ range. As with the WordNet similarity approach, we select only matches with high similarity score. The similarity threshold set to 0.975, which was found to generate classes with between 5 and 50 members in most cases. The size threshold was also set to 50; in practice, the RNNLM-based semantic similarity expansion hit this threshold more often than the WordNet similarity did, though it was still rare.

7.1.3 POS Filtering

The analysis of the hand-crafted seed patterns shows that a large proportion of the features can be grouped into a small number of syntactic templates. To preserve the syntactic struc-

[1] We thank Ji He for generating the RNNLM embeddings.

ture of the seeds in the extended feature set, we apply POS filtering to the neighborhoods obtained using semantic similarity measures. Specifically, the words selected as possible class members are required to match the POS tag of the seed word. The match is done using a relaxed POS definition, with related POS tags from the Penn Treebank set grouped into a broader POS category as shown in table 7.2; POS tags not shown in the table are preserved as individual tags. While any given word may have multiple POS tags depending on context, the expansion classes are context-independent. Therefore, in this work, we assign each word the most frequent POS tag according to the distribution of tags in the LMTrain corpus, with the tags generated automatically.

POS Group	POS Members
noun	NN, NNS
proper noun	NNP, NNPS
pronoun	PRP, PRP$
adjective	JJ, JJR, JJS
adverb	RB, RBR, RBS
verb	MD, VB, VBD, VBG, VBN, VBP, VBZ
w-pronoun	WDT, WP, WP$, WRB
preposition	IN, TO

Table 7.2: POS tag groups and their POS members, as used in POS filtering.

There is no POS tagger available to us that is trained on in-domain data. Instead, we investigate two parsing approaches, in both cases leveraging work on domain adaptation for parsing to obtain reliable POS tags.

Rescoring-Adapted Constituent Parser

The first approach involves reusing the domain adaptation for parser-based ASR rescoring. We use the best self-trained parser model (CTS-ST-P2-Train) to parse the LMTrain data and use the POS tags from the 1-best parse trees to accumulate POS statistics for

each word in the LMTRAIN corpus. We refer to this approach as the *Stanford* POS tagger, after the name of the base package we are using for these experiments.

Dependency Parser

As an alternative to the rescoring-adapted constituent parser, we use the POS tagger [104] developed by collaborators at AMU. This POS tagger is developed as part of a dependency parser [105] which was also adapted to the BOLT domain using self-training, though with a focus of improving general parse quality rather than specific tasks. As in the previous case, we parse the entire LMTRAIN corpus, and use the POS tags obtained from the parse trees to accumulate POS statistics for each word. We refer to this approach as the *AMU* POS tagger.

7.2 Feature Extraction

Since one of the goals of this work is to reduce the likelihood of overfitting, the feature set size remains fixed (determined by the initial seed set). In the class representation of the feature space, we define each feature based on the word classes to which the corresponding initial seed set feature words belong. Each feature takes positive value if it is observed in the 1-best path of the confusion network ASR output. That is, during the feature extraction process, if any combination of words from the classes corresponding to a seed feature are present in the data in the order specified by the seed feature, the corresponding class pattern-based feature is activated. If no phrase pattern corresponding to a particular seed feature is observed in an utterance, the feature corresponding to that seed is not activated (i.e. gets value 0).

The feature extraction process is presented in algorithm 5. In this algorithm, s denotes the sentence for which feature extraction is performed, q is a phrase pattern consisting of pivot words $q_1, \ldots, q_{n_q}$ and p is the corresponding seed phrase pattern; it is also used to denote the feature which is actually activated if q matches s. The matching process is implemented as a standard string regular expression matching, with q being denoted by the

Algorithm 5 Extraction of phrase pattern features

INPUT: Q - dictionary of generalized class-based phrase pattern definitions

INPUT: s - ASR 1-best hypothesis for an utterance

OUTPUT: s_{annot} - ASR 1-best utterance annotations for phrase pattern features

define C - feature cache

initialize $C = \phi$

initialize $s_{\text{annot}} = \phi$, the annotation set for s

for each pattern $q = (q_1^{n_q}) \in Q$ **do**

  if $C[q]$ exists **then**

    set feature p corresponding to phrase pattern q as positive in s_{annot}

  else

    apply q as a regular expression match over s

    if q matches s **then**

      set $C[q] =$ true

      set feature p corresponding to q as positive in s_{annot}

    end if

  end if

end for

Return s_{annot}

regex $q_1(.*)q_2(.*)\ldots q_{n_q}$.[2] For efficiency, the algorithm caches seen phrase patterns, so that if multiple initial seed features yield the same phrase pattern as one of the generalized features, and that phrase pattern is found to match the current utterance, the regex matching process does not need to be repeated for each seed feature to which that phrase pattern maps.

7.3 Performance of Baseline Context Features

As an initial experiment, we augment the set of features used in the MaxEnt classifiers (both first and second stage) with a set of n-gram features designed to capture lexical context. We use n-grams of order up to three, over a window of ± 3 slots centered at the current slot, as described in a paper with collaborator Ji He [55].[3]

Table 7.3 shows results for the name error detection using lexical context features in both MaxEnt stages, with a parser without any adaptation trained on the CTS SMALL treebank. This configuration yields significant improvements over the first MaxEnt stage without lexical context features, for both internal test sets. However, the improvement in the NIST EVAL set is much smaller, likely due to the lexical mismatch between this dataset and the internal test sets. The improvement carries over to parsing, though parsing alone still does not lead to any gains over the MaxEnt systems. With lexical context added, however, parsing no longer helps improve MaxEnt-based detection, with the first MaxEnt stage results being essentially tied with the lexical context-based baseline for the internal test sets, and leading to performance degradation on the external NIST EVAL set.

7.4 No-Parser Experiments

We present a series of experiments that illustrate the effectiveness of the feature learning algorithms, both in terms of improving overall performance, and reducing the overfitting issue. We present results using the seed feature sets and the expanded feature set for each

[2]The matching process is applied over the entire sentence for automatically-generated phrase patterns; for the hand-labeled seed patterns and their extension classes, the matching is performed according to the direction indicated in the seed pattern, which can be L (left of current slot), R (right of current slot), C (enveloping the current slot), or a combination.

[3]An initial implementation of the code used to extract n-gram context features was provided by Wen Wang at SRI International.

Stage / Dataset	first MaxEnt		parser		second MaxEnt	
	WER*	F-score	WER*	F-score	WER*	F-score
P3-Dev	12.4	50.7	13.8	27.5	12.4	51.8
P3-Eval	11.5	53.7	12.8	26.2	11.5	50.5
NIST Eval	5.6	40.0	6.2	26.2	5.7	34.0

Table 7.3: Name error detection using n-gram lexical context and the CTS Small treebank.

set of seed features. In this section we focus on using the new features only in the first MaxEnt stage, to gauge their effect in isolation from parsing.

7.4.1 Initial Feature Definition

First, we discuss the impact of the different seed features used in the phrase pattern learning algorithms. We discuss results for a number of different seed feature sets, as follows:

- **Hand-crafted** - the set of 80 hand-annotated phrase pattern features.
- **Auto words, matched count** - a set of 80 single words with the highest weight in the sentence-level name detector, to compare the performance with the same number of features as the **Hand-crafted** features.
- **Auto words, full count** - all 170 words with non-zero weight in the sentence-level name detector.
- **Auto patterns, matched count** - patterns extracted from the 170 words used in the **auto words full count** configuration, using frequency pruning (requiring each pattern to be seen at least five times in the training data) to reduce the count to match the number of hand-crafted features. The optional step of also incrementing the count of all sub-patterns in algorithm 3 is not used in this case (about 80 features over 30 words).

- **Auto sub-patterns** > 5 - patterns extracted from the 170 words used in the **auto words full count** configuration as well as all possible sub-patterns, using a frequency pruning (requiring each pattern to be seen at least five times in the training data) to reduce the feature count (to about 1200 features).

- **Maximal patterns, matched count** - we filter the set of **Auto sub-patterns** to only those patterns which appear as the maximal pattern in at least one sentence in the training data; we further apply frequency pruning to drop the feature count to match the number of hand-annotated phrase pattern features (about 80 features).

- **Maximal patterns** > 5 - in this configuration, unlike the previous maximal patterns setting, we use a looser count threshold, requiring each pattern to appear at least five times in the training data (about 450 features).

Each of these configurations is added to the first MaxEnt stage feature set. We compare them with the original (no added features) configuration, as well as the configuration with all n-gram context features added.

Results are shown in table 7.4. We note that performance in terms of WER* is very consistent, with only minor variations (maximal difference between any two configurations of under 4%) across the different configurations and for each of the three test sets. The different systems yield bigger performance differences in terms of F-score, with each system using one of the new feature configurations outperforming the baseline first MaxEnt stage with no context features. Adding the word n-gram context features to the baseline feature set outperforms the new features on the two internal sets, but in most cases not on the NIST Eval set, due to overfitting to the BOLT-P3 data, which is not a perfect match to the NIST Eval data. Three configurations lead to consistent performance improvements (over the no-context baseline) in all three test sets: the **auto words, full count** features, the **auto sub-patterns,** > 5 features, and the **Max patterns,** > 5 features.

Feature	P3-DEV		P3-EVAL		NIST EVAL	
Set	WER*	F-score	WER*	F-score	WER*	F-score
original (no context)	12.8	31.8	11.7	38.1	5.7	36.0
local n-gram context	12.4	50.7	11.5	53.7	5.6	40.0
Hand-labeled	12.5	39.2	11.6	33.5	5.6	36.5
Auto words, matched count	12.6	37.7	11.5	43.5	5.6	39.7
Auto words, full count	12.5	44.1	11.5	47.2	5.5	45.0
Auto patterns, matched count	12.6	36.4	11.6	40.4	5.5	45.6
Auto sub-patterns > 5	12.5	43.1	11.5	46.9	5.6	42.9
Max patterns, matched count	12.6	35.3	11.6	34.1	5.5	45.2
Max patterns > 5	12.5	40.5	11.5	46.4	5.5	44.3

Table 7.4: First MaxEnt stage results using only seed features

7.4.2 The Role of Long-Distance Context

The next set of experiments is aimed at further examining the role of different types of lexical context. In particular, we examine whether the performance benefit obtained from pattern features is due to extracting the features over the entire sentence vs. locally (from neighboring words), as well as whether the benefit is due to phrase pattern features vs. regular n-grams. For sentence context, in addition to the **Auto words, full count** (shortened to **Auto words** here) and **Auto sub-patterns, >5** (shortened to **Auto sub-patterns** here) configurations, we add the **Auto n-grams** configuration, using only n-gram features extracted over the same starting set of words as the **Auto sub-patterns** configurations, but not allowing skips between words. We limit the value of n to 4, with only one n-gram found using the maximum length. For local context, we experiment with only the **Auto words** and **Auto n-grams** configurations, since phrase pattern features are by definition not restricted to local context in their use of arbitrary skips between pivot words. To these we add the baseline configuration using all local n-gram context features, not just those built over the **Auto words** vocabulary.

Context	Feature Set	Feature Count	P3-DEV		P3-EVAL		NIST EVAL	
			WER*	F-score	WER*	F-score	WER*	F-score
Sentence	Auto words	170	12.5	44.1	11.5	47.1	5.5	45.0
	Auto n-grams	850	12.5	42.6	11.6	44.8	5.5	45.2
	Auto sub-patterns	1200	12.5	43.1	11.5	46.9	5.5	42.9
Local	Auto words	170	12.5	41.7	11.5	49.1	5.5	41.2
	Auto n-grams	850	12.6	35.6	11.6	38.2	5.6	40.0
	all n-grams	146,000	12.4	50.7	11.5	53.7	5.6	40.0

Table 7.5: First MaxEnt stage results for automatically-learned seed features using different forms of context.

Results are presented in table 7.5. In all three cases, results using the local context are not as good as allowing for features to be extracted over the full utterance, when comparing matched feature sets. This shows that allowing the long-distance context to be captured is useful. Results comparing auto words and auto n-grams are mixed, though in general the single words do slightly better when used in local context. The auto sub-pattern features give the best results when using the full sentence as context, outperforming **Auto n-grams** on all test sets.

7.4.3 Semantic Feature Expansion

The next two experiments examine the effectiveness of the different seed feature sets when used to perform feature expansion with semantic similarity. The first of these experiments focuses on comparing the different sources of POS tags for POS filtering. The second experiment examines the performance of the different sources of semantic similarity when used with the different seed sets. In each case, we are interested in determining which seed set yields the best configuration, as well as determining any trends in terms of which similarity measure or POS tag source yields better results.

Experiment 1: Differences in performance due to different sources of POS tagging data

In the first experiment, we perform the semantic feature extraction starting from the hand-labeled seed set and use each source of semantic similarity together with each source of POS tagging data. Results are shown in table 7.6. In all but one case, using POS tags from the AMU parser leads to better performance than using the Stanford parser POS tags, when measuring F-score, with WER* performance the same or mixed. This behavior holds when performing semantic class expansion using other seed word or pattern sets (not included here). In the next experiments, we include only the AMU parser-based results for brevity.

Experiment 2: Semantic similarity word class expansion using different seed sets

Next, we discuss experiments using semantic similarity-based word class expansion starting with different seed feature sets. We report results for all the seed feature configurations

Semantic Similarity	POS Tagger	P3-DEV WER*	P3-DEV F-score	P3-EVAL WER*	P3-EVAL F-score	NIST EVAL WER*	NIST EVAL F-score
RNNLM	AMU	12.6	37.8	11.5	44.1	5.6	40.6
RNNLM	Stanford	12.7	35.6	11.7	36.5	5.6	38.7
WordNet	AMU	12.5	38.1	11.6	41.8	5.5	42.3
WordNet	Stanford	12.6	37.6	11.6	38.9	5.5	44.1

Table 7.6: First MaxEnt stage results using hand-labeled seed patterns

used in section 7.4.1. For each seed feature set, we include three configurations: using only the seed features with no class expansion, as well as with class expansion using either the RNNLM or WordNet similarities. For both class expansion setups we use the AMU POS tags. Results are shown in table 7.7.

First, we discuss performance separately for three different groups of seed features: using the hand-labeled seed set, using the auto-words seed sets, and using the various auto patterns seed sets. For the hand-labeled set, we find that both class expansion configurations perform as well as or better than the seed-only configuration for the two eval sets; performance is mixed between the RNNLM and the WordNet-based results, with the RNNLM performing worse on the NIST EVAL set, better on P3-EVAL, and essentially tied with the WordNet-based configuration on the P3-DEV set.

For the two auto-words configurations, we see that performance in general is more mixed. Neither class expansion approach outperforms the seed-only configuration on the NIST EVAL set. In general, the RNNLM-based expansion gives better results than the WordNet-based expansion on the internal dev and eval sets, but underperforms on the NIST EVAL set. The WordNet-based results are therefore more consistent, particularly in the **Auto-words, full** configuration, where there is a 10-point difference between P3-EVAL and NIST EVAL using the RNN configuration but only a 2-point difference using WordNet.

Moving to the four pattern-based configurations, performance is again mixed. The RNNLM-based class expansion generally does better than the WordNet-based expansion

Seed Set	Similarity	# Patterns	P3-DEV	P3-EVAL	NIST EVAL
Hand-labeled	Seed Only	80	39.2	33.5	36.5
	RNNLM	21k	37.8	44.1	40.6
	WordNet	4k	38.1	41.8	42.3
Auto-words, matched	Seed Only	80	37.7	43.5	39.7
	RNNLM	1.5k	40.5	43.4	34.6
	WordNet	1.3k	38.3	40.7	38.1
Auto-words, full	Seed Only	170	44.1	47.2	45.0
	RNNLM	3.3k	43.1	51.5	40.2
	WordNet	2.4k	39.8	46.9	44.6
Auto-patterns, matched	Seed Only	80	36.4	40.4	45.6
	RNNLM	38k	39.2	41.4	41.4
	WordNet	120k	35.6	40.8	42.3
Auto-subpatterns, > 5	Seed Only	1.1k	43.1	46.9	42.9
	RNNLM	6.4M	38.0	31.7	32.7
	WordNet	1.6M	41.2	43.3	37.4
Maximal patterns, matched	Seed Only	80	35.3	34.1	45.2
	RNNLM	7k	35.5	43.7	47.1
	WordNet	8.5k	36.4	39.0	42.2
Maximal patterns, > 5	Seed Only	460	40.5	46.4	44.3
	RNNLM	4.2M	39.0	41.6	38.8
	WordNet	320k	40.5	44.2	39.8

Table 7.7: F-score results for different seed set configurations and both semantic similarity sources, using the AMU parser-based POS tags in all cases.

on the P3-Dev and P3-Eval, but underperforms on the NIST Eval set; the exception is in the case of **Maximal patterns, matched** where the RNNLM in fact performs consistently better than both the seed-only set and the WordNet configuration on both eval sets, with the internal dev set results competitive. For the two configurations with the largest class expansion sets, **Auto-subpatterns,** > 5 and **Maximal patterns,** > 5 neither class expansion method yields a win.

A few trends emerge across all the different configurations. In general, the RNNLM expansion does better than WordNet expansion for the internal dev and eval sets, but underperforms for the NIST Eval set; this is consistent with the observation that the RNNLM embeddings are trained on data that is a good match to the BOLT-P3 training set, so features derived from these embeddings are more likely to overtrain. On the other hand, WordNet is a domain-independent resource, so features derived from it tend to be more robust. Second, we notice that most of the expansions that lead to very large pattern sets are likely to underperform, relative to their respective seed sets. Thus, it is likely that tuning the class expansion size for word-based features may not lead to the best configuration for pattern-based features. Third, while the class expansion configurations do not always lead to an improvement over the seed set, in a number of seed set configurations, at least one of the two class expansion methods leads to improvements; we will use some of these configurations for parsing experiments, as discussed in section 7.5.

Comparison of Word Classes Learned by the Best Semantic Expansion Configurations

We perform a qualitative evaluation of the class expansion using the two algorithms (RNNLM and WordNet) with AMU filtering. Table 7.8 contains some example seed words, together with the highest-similarity words found using each semantic similarity algorithm. Examining the classes learned by each algorithm, we find that, perhaps not surprisingly, the matches obtained from WordNet are like the seed word in terms of their *type*; for example, *street* is like other kinds of ontological objects which people use as land transportation surfaces; on the other hand, RNNLM matches are similar to the seed in terms of some intrinsic property; for example, for the seed word *street*, that property seems to be that all the matches are

other types of locations.

Seed Word	Similarity	Most Similar Matches
guarding	WordNet	held, defending, holding, patrolled, policing
	RNNLM	banish, appreciating, terrorizing, dismantled
street	WordNet	thoroughfare, routes, roads, way, highway, alley
	RNNLM	city, pharmacy, corner, restaurant, field, site

Table 7.8: Sample feature classes learned using each semantic similarity source.

These examples support our hypothesis about the behavior of the two class expansion approaches in terms of overall performance vs. generalizability. With the RNNLM trained on (mostly) domain-matched data, it follows that the semantic similarity derived from it would better capture the word uses in the training set, particularly with the hand-labeled features, which were also designed to match the training set. On the other hand, the WordNet ontology was designed as a general-purpose tool, and thus it generalizes well, particularly with a more diverse set of seed features learned in a data-driven way.

7.5 Parsing Experiments Using Phrase Pattern Features

In section 7.3 we found that the best phrase pattern features outperform the n-gram context features on the NIST Eval set, when used in the first MaxEnt stage in conjunction with the baseline set of features described in section 4.1.1. In this section, we perform a series of experiments to assess the impact of the feature learning algorithms on the parser and second MaxEnt stages of the name error detection pipeline. We expect the impact be two-fold, with improved priors on the error arcs seen by the parser improving the parser stage predictions, and the combination of parse features and name context features improving the second MaxEnt stage predictions as well. We experiment with four configurations for the first MaxEnt stage. Each configuration uses the baseline first MaxEnt stage set of features in conjunction with one of the following:

- n-gram word context with $n = 3$;
- the **hand-labeled** phrase patterns seed set using the RNNLM similarity for class expansion;
- the **auto-words, full** seed set using the WordNet similarity for class expansion;
- the **maximal patterns, matched** seed set using the RNNLM similarity for class expansion.

In all feature expansion cases, the AMU POS tags are used to perform filtering of the generalization classes.

The new first stage MaxEnt systems are used to provide input to the parsing stage, as before. The second MaxEnt stage uses the augmented feature set consisting of the baseline first MaxEnt stage features; the parse features; and the set of lexical context/phrase pattern features as used in the corresponding first MaxEnt stage for that system.

For parser adaptation, we experiment only with log-linear adaptation, which was shown in chapter 6 to give the best results on the name error detection task. We use two no-error models as starting points, again as we did in chapter 6: the baseline parser model trained on the CTS SMALL treebank with no adaptation for rescoring, and the best model obtained through self-training adaptation for rescoring (i.e. the two models used in chapter 6 as well). At the beginning of the adaptation process, all error rules have the default weight.

For each feature set we provide two sets of results: selecting the best configuration based on the parser stage results, or based on the second MaxEnt stage results, respectively, to illustrate the effect of tuning based on one stage or the other, both in terms of consistency of results across test sets, and in terms of the overall performance.

Experiment 1: Log-linear adaptation starting with no-rescoring parser

Results using the no-rescoring parser based on the CTS SMALL treebank are presented in table 7.9. We observe that, in general, the n-gram context configuration outperforms the two proposed feature sets on the P3-DEV and P3-EVAL sets; however, it significantly

underperforms on the NIST Eval set. This result is consistent with the first MaxEnt stage results using this configuration. On the other hand, the three proposed feature sets recover the performance drop we see when using n-gram context features, both when tuning on the parser stage and when tuning on the second MaxEnt stage. In general, better results are obtained when tuning with the second MaxEnt stage, though the results are less consistent across the three test sets. On the other hand, when tuning on the parser stage, the differences in performance between the three test sets are minor, particularly when using the hand-labeled seed set. All three new feature sets yield F-score results on the NIST Eval that improve over the corresponding first MaxEnt stage, as well as the first MaxEnt stage using n-gram context features. Performance of the both pattern seed features is competitive with, or slightly better than, the word-only seed features, with the hand-labeled seed set obtaining the best performance on the NIST Eval set.

Feature Set	Tuning Stage	P3-dev		P3-eval		NIST Eval	
		Parser	ME 2	Parser	ME 2	Parser	ME 2
n-gram context	Parser	**52.4**	53.0	**47.1**	51.1	32.4	34.8
n-gram context	MaxEnt II	44.8	**53.5**	46.4	49.0	**43.5**	37.6
hand-labeled	Parser	30.5	42.0	31.6	44.4	36.5	**48.5**
hand-labeled	MaxEnt II	26.5	47.1	30.9	46.1	34.7	43.4
auto-words, full	Parser	32.2	43.0	34.1	42.2	36.0	41.4
auto-words, full	MaxEnt II	31.0	47.3	37.2	**51.3**	36.0	46.4
auto-pats, matched	Parser	30.0	47.8	34.2	46.9	32.2	44.8
auto-pats, matched	MaxEnt II	30.0	47.8	34.2	46.9	32.2	44.8

Table 7.9: Parser F-score results for log-linear adaptation, various phrase pattern features and starting from no rescoring adaptation

Experiment 2: Log-linear adaptation starting with the best rescoring parser

Results when adaptation is performed starting with the best rescoring parser are presented in table 7.10. In this case, results are more mixed. Neither the n-gram context configuration

Feature Set	Tuning Stage	P3-DEV		P3-EVAL		NIST EVAL	
		Parser	ME 2	Parser	ME 2	Parser	ME 2
n-gram context	Parser	37.5	47.0	29.7	41.1	28.6	41.5
n-gram context	MaxEnt II	36.8	**50.4**	29.6	41.4	31.6	40.9
hand-labeled	Parser	37.3	43.6	36.4	41.4	35.1	39.1
hand-labeled	MaxEnt II	35.1	47.3	36.8	41.7	34.8	39.1
auto-words, full	Parser	**38.1**	39.1	**43.7**	**52.6**	**37.2**	40.5
auto-words, full	MaxEnt II	32.0	46.0	30.5	43.8	31.4	41.0
auto-pats, matched	Parser	35.7	39.5	35.9	45.6	36.9	42.2
auto-pats, matched	MaxEnt II	28.6	44.6	35.1	43.1	36.6	**42.8**

Table 7.10: Parser F-score results for log-linear adaptation, various phrase pattern features and starting from best rescoring adaptation

nor the three proposed feature sets yield improvements over the corresponding no-rescoring parser configurations on the internal dev and eval sets. The n-gram context features still cause significant overfitting on the dev set, though the performance this time is worse on both eval sets. We also notice significant overfitting in most cases when the best configuration is selected using the second MaxEnt stage dev set results. The **auto-words, full** configuration performs overall the best, but underperforms in the second MaxEnt stage on the NIST EVAL dataset. On the other hand, the **auto-pats, matched** configuration performs relatively consistent across all three test sets. This configuration also yields the best overall results on the NIST EVAL set, while performing competitively on the P3-DEV and P3-EVAL sets.

Experiment 3: Best parser configurations

In the previous two experiments, we included the best results using log-linear adaptation of the parser stage. For each feature set configuration, we performed the full set of parser adaptation experiments with different rule sets and perceptron update weights, as we had previously done in chapter 6. We provide another view into that tuning process here, to show the impact of the word context and phrase pattern features, respectively, on the parsing

Initial Parser	Feature Set	Tuning Stage	Adaptation Rule Set; Update Weight
No-rescoring	n-gram context	Parser	name-related error & NULL rules; 0.25
		MaxEnt II	all error & NULL rules; 1.0
	hand-labeled	Parser	name-related error & NULL rules; 0.5
		MaxEnt II	name-related error & NULL rules; 0.5
	auto-words, full	Parser	all rules; 0.5
		MaxEnt II	name-related error & NULL rules; 0.5
	auto-pats, matched	Parser	name-related error & NULL rules; 0.5
		MaxEnt II	name-related error & NULL rules; 0.5
Best rescoring	n-gram context	Parser	name-related error & NULL rules; 1.0
		MaxEnt II	name-related error & NULL rules; 1.0
	hand-labeled	Parser	error rules only; 1.0
		MaxEnt II	all error & NULL rules; 0.25
	auto-words, full	Parser	error rules only; 0.5
		MaxEnt II	all rules; 0.5
	auto-pats, matched	Parser	error rules only; 0.5
		MaxEnt II	all error and NULL rules; 0.5

Table 7.11: Best parser adaptation configurations

adaptation process.

Table 7.11 shows the adaptation configurations that led to the best results as presented in tables 7.9 and 7.10. We see that, in many configurations, the best results are obtained using adaptation of the same rule set: the set of error rules targeting name error regions, together with the NULL arc rules. In particular, this adaptation configuration achieves the best performance in most of the "no-rescoring" configurations for each feature set. This observation demonstrates the importance of focusing the adaptation process on rules most likely to be involved in generating name error regions. In a couple of cases, the same configuration gave the best results both when tuning based on the parser stage and when tuning on the second MaxEnt stage results, emphasizing the robustness of the system when context features are used in conjunction with parsing.

Experiment 4: Comparison of Parsing vs. Lexical Context Feature Improvements

We conclude the experimental section of this chapter with a discussion of the effect of parse features and long-distance context features on the MaxEnt stages of the name error classification pipeline. Results for the first MaxEnt stage with baseline features, the best first MaxEnt stage results using context features, the best second MaxEnt stage results using only parse features, and the best second MaxEnt stage results using parse features and lexical context features are compared for each of the three datasets. For this analysis, the "best" configurations are selected based on the performance of each system on the external NIST Eval set.

Stage	Lexical	Parse	P3-Dev	P3-Eval	NIST Eval
ME I	no	n/a	31.8	38.1	36.0
ME I	yes	n/a	41.2	43.7	47.1
ME II	no	yes	**48.8**	**51.8**	46.2
ME II	yes	yes	42.0	44.4	**48.5**

Table 7.12: Comparison of the effect of parsing and lexical context on name error detection.

Results are shown in table 7.12. All three configurations adding lexical or parse information improve upon the first MaxEnt stage with no lexical or parse features in all three stages. Comparing lexical context alone vs. adding parse information, we find that the classifier using parse features is overall more consistent than the classifier using lexical context across the three different test sets, with performance on the NIST EVAL set almost matching that of the lexical context system. Combining parse features and lexical context features results in slight improvements over the lexical context alone, as well as over the parse-only features for NIST EVAL, though the parser-based system still performs better on the two internal datasets.

7.5.1 Analysis

Building upon the use of the proposed feature sets in the first MaxEnt stage of the error detection pipeline, we have found that the later pipeline stages also benefit from the additional context—the parser stage from the improved name error slot-level priors, and the second MaxEnt stage from the proposed features in conjunction with more reliable parse features. This led to improvements in log-linear adaptation of the parser. The parser adaptation was also found to be more robust, with less variation in the configurations leading to best performance using the parser stage vs. the second MaxEnt stage.

7.6 Discussion

In this chapter, we have developed new algorithms for learning phrase pattern features in a task-informed way, using semantic similarity measures and syntactic filtering.

Building upon the intuition that capturing lexical context is essential in detecting name errors, we have experimented with n-gram and phrase pattern features that capture local and long-distance context, and used them in the first MaxEnt stage for name error detection. The experiments presented in this chapter show that long-distance word-level context provides additional benefit over using only the local context captured by n-gram features. Capturing longer-span information through phrase patterns yields even better results in some configurations, in particular on the external NIST EVAL set.

Comparing the two sources of semantic similarity for learning word classes, we find that neither performs significantly better in all cases, though the WordNet-based classes tend to be more reliable than the RNNLM classes. There are a number of reasons why WordNet may be more effective as a source of semantic similarity than the RNNLM embeddings. WordNet is a domain-independent resource, whereas the RNNLM embeddings are trained on the LMTrain data; this may explain why RNNLM-based classes appear to overfit more. RNNLM classes also tend to be larger than the WordNet-based classes, which may lead to more spurious patterns being generated, thus affecting the quality of each word class when used as a feature.

Additionally, the semantic similarity classes were found to be less useful with large seed sets and phrase pattern features, when the product space approach leads to a very large number of patterns being generated, irrespective of the similarity source. In such cases, a more conservative expansion, tuned to the product space approach, may be needed.

Results using only lexical context were found to perform as well as, or in some cases better than, the best parser systems developed in chapter 6. Thus, we experimented with integrating the new features into the full parsing framework. We have found that using the more reliable priors from the improved first MaxEnt stage does yield some benefit to parsing. Additionally, the second MaxEnt stages combining the new class-based phrase pattern features and parse features yield additional improvements in the name error detection task.

Chapter 8

CONCLUSIONS

8.1 Summary

This thesis has presented a novel approach of leveraging parsing technology to improve speech recognition, both for rescoring and for error detection tasks. We developed a multi-stage error detection and rescoring pipeline for ASR output in the form of word confusion networks. To improve the effectiveness of the parsers, we applied domain adaptation techniques, with adaptation objectives aimed at directly optimizing performance of error detection and rescoring rather than parsing in and of itself; the domain-adapted parsers provided improvements in both sets of tasks over the unadapted parsers. Finally, we developed a new method for learning robust long-distance text features, which led to improvements at all stages of the error detection pipelines.

The specific contributions of this thesis include:

A new multi-stage ASR error processing pipeline

To address the first goal of this dissertation, we have developed a novel error processing pipeline as part of a larger speech-to-speech translation system. The standard PCFG model is augmented with rules to generate ASR NULL arcs and ASR errors. We use one parser model augmented with NULL arc rules to discriminate among the possible paths through the confusion network and produce a new ASR hypothesis. A three-stage pipeline for detecting errors integrates a parser augmented with both NULL arc rules and with error rules, both as a full-fledged error detector and to provide additional features for a maximum entropy classifier. We found that even when a parser trained on out-of-domain treebanks leads to no improvements over the ASR one-best when used in rescoring, it can still be useful for error detection, as a source of features.

Parser domain adaptation techniques for ASR rescoring

We present two domain adaptation approaches for improving ASR rescoring using parsing technology, including self-training and adaptation with a secondary, log-linear model augmenting the base PCFG model. The log-linear model adaptation is a novel contribution of this work. Another novel contribution is the use of word error rate as an objective for parser adaptation, which is a form of weak supervision that allows us to perform the adaptation using entirely unlabeled (for parsing) in-domain data. We have found that the self-training approach tends to outperform the log-linear adaptation approach in limited in-domain adaptation settings; the self-training approach is less sensitive to algorithm parameter tuning.

Parser domain adaptation improvements for ASR error detection

We apply the two parser domain adaptation approaches for ASR error detection, focusing on OOV detection and name error detection. Log-linear adaptation for parsing applied to the two error detection tasks leads to consistent improvements for both tasks, both in the parser stage and when parser-derived features are used in the subsequent MaxEnt classifier. Self-training provides additional gain when used subsequent to log-linear adaptation, resulting in smaller and more robust models.

A novel method for learning task-informed long-distance text features

Analysis of previous classification results lead to the observation that the presence of names in utterances is correlated with complex, long-distance lexical context; we model the lexical context in the form of phrase pattern features. We develop a new method for learning such features, starting from either a small set of hand-labeled patterns or from an automatically-learned set of single-word features. Both feature seed sets lead to improvements in name error detection in all stages of the pipeline, and are found to be more robust than local n-gram context features.

8.2 Future Directions

There are a number of possible directions for future research related to the methods presented in this thesis. We discuss some of them here. First, we suggest a number of improvements to the parser adaptation methods proposed in this thesis. Next, we discuss improvements to the proposed feature learning approach. Finally, we discuss some possibilities for tighter integration between the error detection tasks, as well as possible directions for use of parsing information in the downstream components of the full speech-to-speech translation pipeline.

8.2.1 Parser Domain Adaptation Enhancements

We envision a number of possible modifications to the parser domain adaptation strategies presented in this thesis. The self-training algorithm requires strict improvements at each iteration, otherwise the algorithm terminates. However, the various experiments using log-linear adaptation show that a weaker constraint can provide better results. We could experiment with weakening the self-training convergence constraint as well.

For self-training targeting error detection, we found that optimizing for F-score rather than the modified WER* metric yielded better results. The same method could be applied to the log-linear adaptation approach, too. A more complex modification to the log-linear adaptation algorithm involves modifying the training to adapt only the weights of those rules that are part of a derivation which has an error region contained in its yield.

Another direction of research involves using co-training, rather than self-training, to improve parsing performance. One such direction involves using the AMU dependency parser framework, modified to generate dependency trees over confusion networks instead of 1-best hypotheses, in addition to the factored parser approach used for self-training. This approach is particularly useful for rescoring; for error detection tasks, the dependency grammar would also require modifications to allow the generation of error regions.

A third direction of study involves adapting the POS tagging component of the parser separately from the higher-order rules in the model. Such an approach would allow for using more data that may be closely-matched to the BOLT domain in terms of topic, but

not style, and which may not have hand-annotated parses. This approach allows targeting the adaptation specifically to improve topic mismatch between the parser and the target domain (for example using the LMTrain data for the BOLT domain) without learning higher-level grammar rules that do not fit the style of the target domain.

8.2.2 Feature Learning Improvements

Much of the feature learning work focused on using existing semantic similarity resources, such as RNNLM embeddings or the WordNet hierarchy, as off-the-shelf components. However, by targeting the semantic similarity models at the tasks of interest, we hypothesize that we would learn more useful metrics. One direction of study involves learning a set of RNNLM embeddings over a dataset with all names mapped (automatically) to a single NAME token; such a modification would lead, we believe, to more robust connections between names and words likely to indicate their presence in an utterance. Further modifications to the RNNLM model involve adding parse structure information, for instance dependency parse structures, into the neural network model.

A second direction of research involves addressing some of the issues discovered during experiments with automatically-generated phrase patterns. In particular, tuning the class size may alleviate the performance degradation observed when using the class expansion methods with phrase patterns automatically-generated using a product space approach. Different tuning strategies may be employed, for example by sweeping the class size threshold and rebuilding the MaxEnt models for each threshold, using heuristics related to the distribution of the similarity scores, or hybrid methods.

In comparing the two sources of semantic similarity, it became apparent that RNNLM embeddings perform well on domain-matched data but lead to larger classes, whereas WordNet embeddings lead to small classes and tend to generalize better. To leverage the benefits of each method, one approach is to combine the classes generated from the two methods, for example by selecting as the class for each word the overlap of the two semantic similarity-based neighborhoods, or by computing the union while restricting the size of the RNNLM-based neighborhoods significantly.

Additional syntactic information could be leveraged to improve the definition of lexical context features. Syntax is already used to filter out neighborhood elements with mismatched POS tags; instead of using POS information alone, the classes could be filtered using dependency information instead. In such a setup, the neighborhood would be defined not only using semantic information, but also syntactic information, in the form of constituent or dependency neighbors.

Syntax is also useful in feature extraction. One direction of research involves limiting the extent of phrase pattern features to within particular types of constituents (for instance, not allowing a phrase pattern to cross boundaries of sentence subtrees, i.e. subtrees rooted at S or $SBAR$ nodes).

8.2.3 System-Level Issues

Finally, we discuss some possible future directions to improve the functionality of the ASR rescoring and error detection system as it is integrated into the larger BOLT speech-to-speech translation application. Two directions are considered, to improve parsing speed and to improve the handling of ASR errors in the dialog system component.

The error detection system as currently designed is instantiated separately for each error type of interest. However, the use of multiple constituent parsers is relatively expensive, even in distributed computation settings where multiple parsers are run in parallel. A number of approaches may be used to combine the processing and thus reduce some of the overhead of parsing. We envision a shared error detection pipeline using a first MaxEnt stage that detects all error types of interest jointly. The parser grammar would be modified to explicitly model all error types, with a separate set of rules for each type (instead of a single error type as we do now); the second MaxEnt stage would use a separate classifier for each error type to allow for additional specialization according to the different features extracted from the parser.

A second direction involves tighter integration with the dialog manager component of the larger BOLT system. The current implementation of the error processing system exposes the type (i.e. OOV or name error) and extent of an error region to downstream components,

such as the dialog manager; however, the syntactic category of the error region as detected by the parser is not exposed. Domain adaptation of the parser has led to improvements in the performance of the parser as an error detector in its own right; thus, it may be possible to use the syntactic category of the error regions detected by the parser as a label for the error regions detected in the (more accurate) second MaxEnt stage, if the overlap between the two regions is high enough, even if not complete.

8.3 Final Thoughts

The work performed in this thesis was geared towards speech-to-speech translation, but the problems of ASR error detection and automatic correction of ASR errors are of interest in a wide variety of contexts. Many other speech applications, ranging from personal assistants on mobile devices, to automatic generation of doctor notes for hospital patients, could benefit from the detection of incorrect regions of ASR output, particularly when the system may encounter rare terms, or when incorrect transcription has a high cost in terms of the system taking the wrong action. The methods used in this thesis are of interest to more than just speech applications, though; task-supervised parser domain adaptation may be applied equally successfully when the parser is used to provide features for sentiment analysis, or to study the differences of how people interact in various conversational settings, face-to-face or online. Long distance features, too, are useful for detecting more than just names; as long as a class of content words with specific semantic properties can be defined, their usage will be defined by the words around them. Such methods may be applied in keyword spotting tasks; they may also be of interest to intelligence-gathering applications. This thesis thus takes only a small step – of a likely-long journey – towards the larger goal of using both what people say, and how they say it, to help them achieve their intended tasks.

BIBLIOGRAPHY

[1] CTS treebank with structural metadata. https://catalog.ldc.upenn.edu/LDC2009T01.

[2] SCLITE Toolkit. ftp://jaguar.ncsl.nist.gov/pub/sctk-2.4.0-20091110-0958.tar.bz2.

[3] Translation technology: Breaking the language barrier. http://www.darpa.mil/workarea/downloadasset.aspx?id=2676.

[4] Treebank 2 corpus. https://catalog.ldc.upenn.edu/LDC95T7.

[5] Treebank 3 corpus. https://catalog.ldc.upenn.edu/LDC99T42.

[6] Murat Akbacak, Horacio Franco, Michael Frandsen, Sasa Hasan, Huda Jameel, Andreas Kathol, Shahram Khadivi, Xin Lei, Arindam Mandal, Saab Mansour, Kristin Precoda, Colleen Richey, Dimitra Vergyri, Wen Wang, Mei Yang, and Jing Zheng. Recent advances in SRI'S IraqComm. In *Proc. ICASSP*, pages 4809–4812, 2009.

[7] M.-R. Amini and P. Gallinari. Semi-supervised logistic regression. In *Proc. European Conference of Artificial Intelligence*, 2002.

[8] R. Ando. Semantic lexicon construction: learning from unlabeled data via spectral analysis. In *Proc. CoNLL*, 2004.

[9] R. K. Ando. Applying alternating structure optimization to word sense disambiguation. In *Proc. CoNLL-X*, 2006.

[10] R. K. Ando and T. Zhang. A framework for learning predictive structures from multiple tasks and unlabeled data. *Journal for Machine Learning Research*, 6:1817–1853, 2005.

[11] R. K. Ando and T. Zhang. A high-performance semi-supervised learning method for text chunking. In *Proc. ACL*, pages 1–9, Stroudsburg, PA, USA, 2005. Association for Computational Linguistics.

[12] R.K. Ando. Exploiting unannotated corpora for tagging and chunking. In *Companion volume to the Proc. ACL*, 2004.

[13] A. Asadi, R. Schwartz, and J. Makhoul. Automatic detection of new words in a large vocabulary continuous speech recognition system. In *Proc. ICASSP*, volume 1, pages 125–128, 1990.

[14] N.F. Ayan, A Mandal, M. Frandsen, Jing Zheng, P. Blasco, A Kathol, F. Bechet, B. Favre, A Marin, T. Kwiatkowski, M. Ostendorf, L. Zettlemoyer, P. Salletmayr, J. Hirschberg, and S. Stoyanchev. "Can you give me another word for hyperbaric?" Improving speech translation using targeted clarification questions. In *Proc. ICASSP*, pages 8391–8395, 2013.

[15] L. D. Baker and A. K. McCallum. Distributional clustering of words for text classification. In *Proc. SIGIR*, pages 96–103, New York, NY, USA, 1998.

[16] N. Bassiou and C. Kotropoulos. Long distance bigram models applied to word clustering. *Pattern Recognition*, 44(1):145–158, 2011.

[17] L. E. Baum and T. Petrie. Statistical inference for probabilistic functions of finite state markov chains. *The Annals of Mathematical Statistics*, 37(6):1554–1563, 1966.

[18] Issam Bazzi, James Glass, and Arthur C. Smith. Modeling out-of-vocabulary words for robust speech recognition. In *Proc. ICSLP*, 2000.

[19] R. Bekkerman, R. El-Yaniv, N. Tishby, Y. Winter, I. Guyon, and A. Elisseeff. Distributional word clusters vs. words for text categorization. *Journal of Machine Learning Research*, 3:1183–1208, 2003.

[20] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. A neural network probabilistic model. *Journal of Machine Learning Research*, 3:1137–1155, 2003.

[21] A. L. Berger, V. J. Della Pietra, and S. A. Della Pietra. A maximum entropy approach to natural language processing. *Computational Linguistics*, 22(1):39–71, March 1996.

[22] Maximilian Bisani and Hermann Ney. Open vocabulary speech recognition with flat hybrid models. In *Proc. Interspeech*, pages 725–728, 2005.

[23] J. Blitzer, R. Mcdonald, and F. Pereira. Domain adaptation with structural correspondence learning. In *Proc. EMNLP*, 2006.

[24] John Blitzer, Mark Dredze, and Fernando Pereira. Biographies, bollywood, boomboxes and blenders: Domain adaptation for sentiment classification. In *Proc. ACL*, pages 187–205, 2007.

[25] A. Blum and T. Mitchell. Combining labeled and unlabeled data with co-training. In *Proc. Conference on Computational Learning Theory (COLT)*, pages 92–100, 2008.

[26] Manuela Boros, Maria Aretoulaki, Florian Gallwitz, Elmar Noth, and Heinrich Niemann. Semantic processing of out-of-vocabulary words in a spoken dialogue system. In *Proc. Eurospeech*, pages 1887–1890, 1997.

[27] Peter F. Brown, Peter V. deSouza, Robert L. Mercer, Vincent J. Della Pietra, and Jenifer C. Lai. Class-based n-gram models of natural language. *Comput. Linguist.*, 18(4):467–479, December 1992.

[28] Eugene Charniak. A maximum-entropy-inspired parser. In *Proceedings of the 1st North American Chapter of the Association for Computational Linguistics Conference*, NAACL 2000, pages 132–139, Stroudsburg, PA, USA, 2000. Association for Computational Linguistics.

[29] Ciprian Chelba, Dan Bikel, Maria Shugrina, Patrick Nguyen, and Shankar Kumar. Large scale language modeling in automatic speech recognition. *arXiv preprint arXiv:1210.8440*, 2012.

[30] Ciprian Chelba and Frederick Jelinek. Structured language modeling. *Computer Speech & Language*, 14(4):283–332, 2000.

[31] Danqi Chen and Christopher D. Manning. A fast and accurate dependency parser using neural networks. In *Proc. EMNLP*, 2014.

[32] S. Chen and J. Goodman. An empirical study of smoothing techniques for language modeling. *Computer Speech and Language*, 13(4):359–394, 1999.

[33] Wei Chen, Sankaranarayanan Ananthakrishnan, Rohit Prasad, and Prem Natarajan. Variable-span out-of-vocabulary named entity detection. In *Proc. Interspeech*, pages 3761–3765, 2013.

[34] Michael Collins. *Head-Driven Statistical Models for Natural Language Parsing.* PhD thesis, University of Pennsylvania, 1999.

[35] Michael Collins. Discriminative training methods for hidden Markov models: theory and experiments with perceptron algorithms. In *Proc. ACL/EMNLP*, pages 1–8, 2002.

[36] Michael Collins. Head-driven statistical models for natural language parsing. *Comput. Linguist.*, 29(4):589–637, dec 2003.

[37] Michael Collins, Murat Saraclar, and Brian Roark. Discriminative syntactic language modeling for speech recognition. In *Proc. ACL*, pages 507–514, 2005.

[38] C. Cortes and V. Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, September 1995.

[39] Christopher Cieri David, David Miller, and Kevin Walker. The Fisher corpus: a resource for the next generations of speech-to-text. In *Proceedings 4th International Conference on Language Resources and Evaluation*, pages 69–71, 2004.

[40] S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, and R. Harshman. Indexing by latent semantic analysis. *Journal of the American Society for Information Science*, 41(6):391–407, 1990.

[41] I. S. Dhillon, S. Mallela, and R. Kumar. Enhanced word clustering for hierarchical text classification. In *Proc. KDD*, pages 191–200, New York, NY, USA, 2002.

[42] Inderjit S. Dhillon, Subramanyam Mallela, and Rahul Kumar. A divisive information-theoretic feature clustering algorithm for text classification. *Journal of Machine Learning Research*, 3:1265–1287, 2003.

[43] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification (2nd Edition)*. Wiley-Interscience, 2 edition, November 2001.

[44] E. Gabrilovich and S. Markovitch. Harnessing the expertise of 70,000 human editors: Knowledge-based feature generation for text categorization. *Journal of Machine Learning Research*, 8:2297–2345, December 2007.

[45] E. Gabrilovich and S. Markovitch. Wikipedia-based semantic interpretation for natural language processing. *Journal of Artificial Intelligence Research*, 34:443–498, March 2009.

[46] L. Gillick, Y. Ito, and J. Young. A probabilistic approach to confidence estimation and evaluation. In *Proc. ICASSP*, 1997.

[47] J. J. Godfrey, E. C. Holliman, and J. McDaniel. Switchboard: Telephone speech corpus for research and development. In *Proc. ACL*, volume I, pages 517–520, 1992.

[48] Sally Goldman and Yan Zhou. Enhancing supervised learning with unlabeled data. In *Proc. ICML*, 2000.

[49] David Graff and Christopher Cieri. English Gigaword. https://catalog.ldc.upenn.edu/LDC2003T05.

[50] Y. Grandvalet and Y. Bengio. Semi-supervised learning by entropy minimization. In *Advances in Neural Information Processing Systems 17*, 2005.

[51] H. Han, E. Manavoglu, C. L. Giles, and H. Zha. Rule-based word clustering for text classification. In *Proc. SIGIR*, pages 445–446, 2003.

[52] M. Harper, B. Dorr, B. Roark, J. Hale, I. Shafran, Y. Liu, M. Lease, M. Snover, L. Young, R. Stewart, and A. Krasnyanskaya. Parsing speech and structural event detection. Technical report, JHU Summer Workshop, 2005.

[53] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer Series in Statistics. Springer New York Inc., New York, NY, USA, 2001.

[54] Mohamed Hatmi, Christine Jacquin, Emmanuel Morin, Sylvain Meigner, et al. Incorporating named entity recognition into the speech transcription process. In *Proc. Interspeech*, 2013.

[55] Ji He, Alex Marin, and Mari Ostendorf. Effective data-driven feature learning for detecting name errors in automatic speech recognition. In *SLT*, 2014.

[56] Hynek Hermansky. Perceptual linear predictive (PLP) analysis of speech. *Journal of the Acoustical Society of America*, 87(4):1738–1752, 1990.

[57] Dustin Hillard. *Automatic Sentence Structure Annotation for Spoken Language Processing*. PhD thesis, Seattle, WA, USA, 2008.

[58] Dustin Hillard and Mari Ostendorf. Compensating forward posterior estimation bias in confusion networks. In *ICASSP*, 2006.

[59] Geoffrey Hinton, Li Deng, Dong Yu, Abdel-Rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara Sainath George Dahl, and Brian Kingsbury. Deep neural networks for acoustic modeling in speech recognition. *IEEE Signal Processing Magazine*, 29(6):82–97, November 2012.

[60] Fei Huang and Alexander Yates. Distributional representations for handling sparsity in supervised sequence-labeling. In *Proc. ACL*, 2009.

[61] Xuedong Huang, Alex Acero, and Hsiao-Wuen Hon. *Spoken Language Processing: A Guide to Theory, Algorithm, and System Development*. Prentice Hall PTR, 1 edition, 2001.

[62] Zhongqiang Huang and Mary Harper. Self-training PCFG grammars with latent annotations across languages. In *Proceedings of EMNLP*, 2009.

[63] S. Jaillet, A. Laurent, and M. Teisseire. Sequential patterns for text categorization. *Intell. Data Anal.*, 10(3):199–214, 2006.

[64] Euisok Chung Hyung-Bae Jeon Jeon, Gue Park, and Yun-Keun Lee. Lattice rescoring for speech recognition using large scale distributed language models. In *24th International Conference on Computational Linguistics*, page 217, 2012.

[65] Hui Jiang. Confidence measures for speech recognition: A survey. *Speech Communication*, 45(4):455 – 470, 2005.

[66] J. Jiang and C. Zhai. Instance weighting for domain adaptation in NLP. In *Proc. ACL*, 2007.

[67] Karen S. Jones. A statistical interpretation of term specificity and its application in retrieval. *Journal of Documentation*, 28:11–21, 1972.

[68] Daniel Jurafsky and James H. Martin. *Speech and Language Processing (2Nd Edition)*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2009.

[69] Jeremy G. Kahn and Mari Ostendorf. Joint reranking of parsing and word recognition with automatic segmentation. *Computer Speech and Language*, 26(1):1–19, January 2012.

[70] Dan Klein and Christopher D. Manning. Accurate unlexicalized parsing. In *Proc. ACL*, pages 423–430, 2003.

[71] Dan Klein and Christopher D. Manning. Fast exact inference with a factored model for natural language parsing. In *In Advances in Neural Information Processing Systems 15 (NIPS)*, pages 3–10. MIT Press, 2003.

[72] Stefan Kombrink, Lukás Burget, Pavel Matejka, Martin Karafiát, and Hynek Hermansky. Posterior-based out of vocabulary word detection in telephone speech. In *Proc. Interspeech*, pages 80–83, 2009.

[73] M. Koppel and J. Schler. Exploiting stylistic idiosyncrasies for authorship attribution. In *Proc. IJCAI Workshop on Computational Approaches to Style Analysis*, 2003.

[74] Rohit Kumar, Rohit Prasad, Sankaranarayanan Ananthakrishnan, Aravind Namandi Vembu, David Stallard, Stavros Tsakalidis, and Prem Natarajan. Detecting OOV named-entities in conversational speech. In *Proc. Interspeech*, 2012.

[75] Hong-Kwang Kuo, Ellen Eide Kislal, Lidia Mangu, Hagen Soltau, and Tomas Beran. Out-of-vocabulary word detection in a speech-to-speech translation system. In *Proc. ICASSP*, pages 7108–7112, 2014.

[76] J. Lafferty, A. McCallum, and F. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proc. ICML*, 2001.

[77] M. Lavielle and E. Moulines. A simulated annealing version of the EM algorithm for non-gaussian deconvolution. *Statist. Comput*, 1997.

[78] Benjamin Lecouteux, Georges Linars, and Benot Favre. Combined low level and high level features for out-of-vocabulary word detection. In *Proc. Interspeech*, pages 1187–1190, 2009.

[79] O. Levy and Y. Goldberg. Dependency-based word embeddings. In *Proc. ACL*, 2014.

[80] D. Lin. An information-theoretic definition of similarity. In *Proc. International Conference on Machine Learning*, 1998.

[81] Hui Lin, Jeff Bilmes, Dimitra Vergyri, and Katrin Kirchhoff. OOV detection by joint word/phone lattice alignment. In *Proc. ASRU*, pages 478–483. IEEE, 2007.

[82] Lidia Mangu, Eric Brill, and Andreas Stolcke. Finding consensus in speech recognition: word error minimization and other applications of confusion networks. *Computer Speech & Language*, 14(4):373–400, 2000.

[83] M. Marcus, G. Kim, M. A. Marcinkiewicz, R. MacIntyre, A. Bies, M. Ferguson, K. Katz, and B. Schasberger. The Penn treebank: Annotating predicate argument structure. In *In ARPA Human Language Technology Workshop*, pages 114–119, 1994.

[84] A. Marin, M. Ostendorf, B. Zhang, J.T. Morgan, M. Oxley, M. Zachry, and E.M. Bender. Detecting authority bids in online discussions. In *Proc. SLT*, 2010.

[85] A. Marin, B. Zhang, and M. Ostendorf. Detecting forum authority claims in online discussions. In *Proc. Workshop on Language in Social Media*, pages 48–57, 2011.

[86] Alex Marin, Roman Holenstein, Ruhi Sarikaya, and Mari Ostendorf. Learning phrase patterns for text classification using a knowledge graph and unlabeled data. In *Proc. Interspeech*, 2014.

[87] Alex Marin, Tom Kwiatkowski, Mari Ostendorf, and Luke S. Zettlemoyer. Using syntactic and confusion network structure for out-of-vocabulary word detection. In *Proc. SLT*, pages 159–164, 2012.

[88] Alex Marin and Mari Ostendorf. Domain adaptation for parsing in automatic speech recognition. In *Proc. ICASSP*, pages 6379–6383, 2014.

[89] Alex Marin, Wei Wu, Bin Zhang, and Mari Ostendorf. Detecting targets of alignment moves in multiparty discussions. In *Proc. ICASSP*, 2012.

[90] D. McAllester. Lecture on EM, 2007. http://ttic.uchicago.edu/dmcallester/ttic101-07/lectures/em/em.pdf.

[91] Andrew Kachites McCallum. MALLET: A machine learning for language toolkit. http://mallet.cs.umass.edu, 2002.

[92] D. McClosky, E. Charniak, and M. Johnson. Effective self-training for parsing. In *Proceedings of HLT-NAACL*, pages 152–159, 2006.

[93] Paul Mermelstein. Distance measures for speech recognition, psychological and instrumental. *Pattern Recognition and Machine Intelligence*, pages 374–388, 1976.

[94] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781, 2013.

[95] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *Proc. NIPS*, 2013.

[96] Tomáš Mikolov. *Statistical language models based on neural networks*. PhD thesis, Ph. D. thesis, Brno University of Technology, 2012.

[97] Tomáš Mikolov, Martin Karafiát, Lukas Burget, Jan Cernockỳ, and Sanjeev Khudanpur. Recurrent neural network based language model. In *Proc. Interspeech*, pages 1045–1048, 2010.

[98] Tomáš Mikolov, Stefan Kombrink, Anoop Deoras, Lukar Burget, and J Cernockỳ. RNNLM-recurrent neural network language modeling toolkit. In *Proc. ASRU*, pages 196–201, 2011.

[99] Tomáš Mikolov, Wen-Tau Yih, and Geoffrey Zweig. Linguistic regularities in continuous space word representations. In *Proc. NAACL-HLT*. Association for Computational Linguistics, 2013.

[100] G. A. et al. Miller. WordNet: An on-line lexical database. *Intl. Journal of Lexicography*, 3:235–244, 1990.

[101] Michael Mohler and Rada Mihalcea. Text-to-text semantic similarity for automatic short answer grading. In *Proc. EACL*, 1994.

[102] Fabrizio Morbini, Kartik Audhkhasi, Ron Artstein, Maarten Van Segbroeck, Kenji Sagae, Panayiotis Georgiou, David R. Traum, and Shri Narayanan. A reranking approach for recognition and classification of speech input in conversational dialogue systems. In *Proc. SLT*, 2012.

[103] P. J. Moreno, B. Logan, and B. Raj. A boosting approach for confidence scoring. In *Proc. Eurospeech*, 2001.

[104] A. Nasr, F. Béchet, J.F. Rey, B. Favre, and J. Le Roux. MACAON: an NLP tool suite for processing word lattices. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies: Systems Demonstrations*, pages 86–91. Association for Computational Linguistics, 2011.

[105] Alexis Nasr, Frederic Bechet, Benoit Favre, Thierry Bazillon, Jose Deulofeu, and Andre Valli. Automatically enriching spoken corpora with syntactic information for linguistic studies. In *Proc. LREC*, 2014.

[106] C. V. Neti, S. Roukos, and E. Eide. Word-based confidence measures as a guide for stack search in speech recognition. In *Proc. ICASSP*, 1997.

[107] Hermann Ney. Dynamic programming parsing for context-free grammars in continuous speech recognition. *IEEE Transactions on Signal Processing*, 2(39):336–340, 1991.

[108] V. Ng and C. Cardie. Weakly supervised natural language learning without redundant views. In *Proc. NAACL*, 2003.

[109] K. Nigam and R. Ghani. Analyzing the effectiveness and applicability of co-training. In *Proc. CIKM*, pages 86–93, 2000.

[110] Z.-Y. Niu, D.-H. Ji, and C. L. Tan. A semi-supervised feature clustering algorithm with application to word sense disambiguation. In *Proc. HLT EMNLP*, pages 907–914, Stroudsburg, PA, USA, 2005. Association for Computational Linguistics.

[111] David Palmer and Mari Ostendorf. Improving out-of-vocabulary name resolution. *Computer Speech and language*, 19(1):107–128, 2005.

[112] David Palmer, Mari Ostendorf, and John Burger. Robust information extraction from automatically generated speech transcriptions. *Speech Communication*, 32:95–109, 2000.

[113] S. J. Pan, X. Ni, J.-T. Sun, Q. Yang, and Z. Chen. Cross-domain sentiment classification with spectral feature alignment. In *Proc. WWW*, 2010.

[114] Carolina Parada, Mark Dredze, Denis Filimonov, and Frederick Jelinek. Contextual information improves OOV detection in speech. In *Proc. NAACL*, 2010.

[115] Carolina Parada, Mark Dredze, and Frederick Jelinek. OOV sensitive named-entity recognition in speech. In *Proc. Interspeech*, pages 2085–2088, 2011.

[116] Carolina Parada, Abhinav Sethy, and Bhuvana Ramabhadran. Query-by-example spoken term detection for OOV terms. In *Proc. ASRU*, 2009.

[117] J.W. Pennebaker, M.E. Francis, and R.J. Booth. *Linguistic Inquiry and Word Count: LIWC2001*. Erlbaum Publishers, 2001.

[118] F. Pereira, N. Tishby, and L. Lee. Distributional clustering of English words. In *Proc. ACL*, pages 183–190, Stroudsburg, PA, USA, 1993.

[119] S. Petrov, L. Barrett, R. Thibaux, and D. Klein. Learning accurate, compact, and interpretable tree annotation. In *Proc. COLING-ACL*, pages 433–440, 2006.

[120] David Pierce and Claire Cardie. Limitations of co-training for natural language learning from large datasets. In *Proc. EMNLP*, 2001.

[121] A. Rastrow, M. Dredze, and S. Khudanpur. Efficient structured language modeling for speech recognition. In *Proceedings of Interspeech*, 2012.

[122] A. Rastrow, A. Sethy, and B. Ramabhadran. A new method for OOV detection using hybrid word/fragment system. In *Proc. ICASSP*, pages 3953–3956, 2009.

[123] B. Roark, M. P. Harper, E. Charniak, B. Dorr, M. Johnson, J. Kahn, Y. Liu, M. Ostendorf, J. Hale, A. Krasnyanskaya, M. Lease, I. Shafran, M. Snover, R. Stewart, and L. Yung. Sparseval: Evaluation metrics for parsing speech. In *Proc. LREC*, 2006.

[124] B. Roark, M. Saraclar, and M. Collins. Corrective language modeling for large vocabulary ASR with the perceptron algorithm. In *Acoustics, Speech, and Signal Processing, 2004. Proceedings. (ICASSP '04). IEEE International Conference on*, volume 1, pages I–749–52 vol.1, 2004.

[125] Brian Roark. Probabilistic top-down parsing and language modeling. *Computational Linguistics*, 27(2):249–276, June 2001.

[126] Brian Roark and Michiel Bacchiani. Supervised and unsupervised PCFG adaptation to novel domains. In *Proc. NAACL*, 2003.

[127] K. Sagae. Self-training without reranking for parser domain adaptation and its impact on semantic role labeling. In *Proc. ACL Workshop on Domain Adaptation for Natural Language Processing*, 2010.

[128] K. Sagae and J. Tsujii. Dependency parsing and domain adaptation with LR models and parser ensembles. In *Proc. CoNLL Shared Task Session of EMNLP-CoNLL*, 2007.

[129] Thomas Schaaf. Detection of OOV words using generalized word models and a semantic class language model. In *Proc. Eurospeech*, pages 2581–2584, 2001.

[130] R. E. Schapire and Y. Singer. BoosTexter: A Boosting-based System for Text Categorization. *Machine Learning*, 39(2):135–168, May 2000.

[131] E. E. Shriberg. *Preliminaries to a Theory of Speech Disfluencies*. PhD thesis, University of California, Berkeley, 1994.

[132] Manhung Siu and Herbert Gish. Evaluation of word confidence for speech recognition systems. *Computer Speech & Language*, 13(4):299 – 319, 1999.

[133] N. Slonim and N. Tishby. Agglomerative information bottleneck. In *Proc. NIPS*, volume 12, pages 617–623, 1999.

[134] N. Slonim and N. Tishby. The power of word clusters for text classification. In *23rd European Colloquium on Information Retrieval Research*, 2001.

[135] Noam Slonim and Naftali Tishby. Document clustering using word clusters via the information bottleneck method. In *Proc. SIGIR*, pages 208–215, New York, NY, USA, 2000. ACM.

[136] Frank K. Soong and Wai kit Lo Satoshi. Optimal acoustic and language model weights for minimizing word verification errors. In *Proc. ICSLP*, 2004.

[137] Andreas Stolcke. SRILM – an extensible language modeling toolkit. In *Proc. ICSLP*, pages 901–904, 2002.

[138] S. Stoyanchev, A. Liu, and J. Hirschberg. Towards natural clarification questions in dialogue systems. In *AISB Symposium on "Questions, discourse and dialogue: 20 years after Making it Explicit*, 2014.

[139] Katsuhito Sudoh, Hajime Tsukada, and Hideki Isozaki. Incorporating speech recognition confidence into discriminative named entity recognition of speech. In *Proc. ACL*, 2006.

[140] Guihua Sun, Xiaohua Liu, Gao Cong, Ming Zhou, Zhongyang Xiong, John Lee, and Chin Y. Lin. Detecting erroneous sentences using automatically mined sequential patterns. In *Proc. ACL*, pages 81–88, 2007.

[141] H. Sun, G. Zhang, F. Zheng, and M. Xu. Using word confidence measure for oov words detection in a spontaneous spoken dialog system. In *Proc. Eurospeech*, pages 2713–2716, 2003.

[142] Yik-Cheung Tam, Yun Lei, Jing Zheng, and Wen Wang. ASR error detection using recurrent neural network language model and complementary ASR. In *Proc. ICASSP*, pages 2312–2316, 2014.

[143] Oren Tsur, Dmitry Davidov, and Ari Rappoport. ICWSM – A Great Catchy Name: Semi-Supervised Recognition of Sarcastic Sentences in Online Product Reviews. In *Proceedings of AAAI*, 2010.

[144] Joseph Turian, Lev Ratinov, and Yoshua Bengio. Word representations: a simple and general method for semi-supervised learning. In *Proc. ACL*, pages 384–394, Stroudsburg, PA, USA, 2010. Association for Computational Linguistics.

[145] W. Wang and M. P. Harper. The SuperARV language model: Investigating the effectiveness of tightly integrating multiple knowledge sources. In *Proc. EMNLP*, pages 238–247, 2002.

[146] W. Wang, M. P. Harper, and A. Stolcke. The robustness of an almost-parsing language model given errorful training data. In *Proc. ICASSP*, 2003.

[147] Wen Wang. Weakly supervised training for parsing mandarin broadcast transcripts. In *Proc. Interspeech*, 2008.

[148] Wen Wang. Combining discriminative re-ranking and co-training for parsing Mandarin speech transcripts. In *Proc. ICASSP*, pages 4705–4708, 2009.

[149] Mitch Weintraub, Franoise Beaufays, Ze'ev Rivlin, Yochai Konig, and Andreas Stolcke. Neural-network based measures of confidence for word recognition. In *Proc. ICASSP*, pages 887–890, 1997.

[150] C. Wenliang, C. Xingzhi, W. Huizhen, Z. Jingbo, and Y. Tianshun. Automatic word clustering for text categorization using global information. In S. Myaeng, M. Zhou, K.-F. Wong, and H.-J. Zhang, editors, *Information Retrieval Technology*, volume 3411 of *Lecture Notes in Computer Science*, pages 1–11. Springer Berlin / Heidelberg, 2005.

[151] F. Wessel, K. Maherey, and H. Ney. A comparison of word graph and n-best list based confidence measures. In *Proc. Eurospeech*, 1999.

[152] F. Wessel, R. Schluter, K. Macherey, and H. Ney. Confidence measures for large vocabulary continuous speech recognition. *Speech and Audio Processing, IEEE Transactions on*, 9(3):288–298, Mar 2001.

[153] C. White, J. Droppo, A. Acero, and J. Odell. Maximum entropy confidence estimation for speech recognition. In *Proc. ICASSP*, 2007.

[154] Janyce Wiebe and Ellen Riloff. Creating subjective and objective sentence classifiers from unannotated texts. *Computational Linguistics and Intelligent Text Processing*, 3406:486–497, 2005.

[155] Theresa Wilson, Janyce Wiebe, and Paul Hoffmann. Recognizing contextual polarity in phrase-level sentiment analysis. In *Proceedings of HLT-EMNLP*, pages 347–354, Stroudsburg, PA, USA, 2005.

[156] Zhibiao Wu and Martha Palmer. Verbs semantics and lexical selection. In *Proc. ACL*, 1994.

[157] S.R. Young. Detecting misrecognitions and out-of-vocabulary words. In *Proc. ICASSP*, volume ii, pages II.21–II.24, 1994.

[158] Bin Zhang, Alex Marin, Brian Hutchinson, and Mari Ostendorf. Learning phrase patterns for text classification. *IEEE Transactions on Audio Speech and Language Processing*, (6):1180–1189, 2013.

[159] Jing Zheng, A. Mandal, Xin Lei, M. Frandsen, N.F. Ayan, D. Vergyri, Wen Wang, M. Akbacak, and K. Precoda. Implementing SRI's Pashto speech-to-speech translation system on a smart phone. In *Proc. SLT*, pages 133 –138, 2010.

Appendix A

HAND-CRAFTED SEED PHRASE PATTERNS FOR NAME ERROR DETECTION

Pattern	Direction	Pattern	Direction	Pattern	Direction
back in	R	his name	R	study in	R
been in	R	house of	R	the area	C
been to	R	hurt by	R	the city	C
be in	R	i'm from	CR	the forces	C
be with	R	i'm	R	the mosque	C
bring to	CR	in city	C	the name	R
brought into	R	killed by	R	the people of	R
called	R	lieutenant	R	the school	C
came from	R	live in	R	the tribe	C
came to	R	lives in	R	toward	R
came with	R	lives	R	tribe	L
captain	R	living in	R	visit from	CR
coming from	R	looking for	R	visit in	CR
coming through	R	mister	R	visiting with	R
coming to	R	my name	R	visit	R
corporal	R	my name is	R	was in	R
doctor	R	named	R	went to	R
drive to	R	name is	R	work in	R
from to	CR	on street	C	works in	R
go back to	R	on the	R	works for	R
going down	R	party	L	works with	R

Pattern	Direction	Pattern	Direction	Pattern	Direction
going to	R	sergeant	R	your name	R
go to	R	start in	R	your neighbor	R
grew up in	R	stay in	R	your neighbor	R
have in	CR	street	L	your rank	R
here in	R	studying in	R		