ABSTRACT

Title of dissertation: DECISION TREE-BASED SYNTACTIC LANGUAGE MODELING

Denis Filimonov, Doctor of Philosophy, 2011

Dissertation directed by: Dr. Mary Harper
Department of Computer Science
Dr. Philip Resnik
Department of Linguistics

Statistical Language Modeling is an integral part of many natural language processing applications, such as Automatic Speech Recognition (ASR) and Machine Translation. N-gram language models dominate the field, despite having an extremely shallow view of language—a Markov chain of words. In this thesis, we develop and evaluate a joint language model that incorporates syntactic and lexical information in a effort to "put language back into language modeling." Our main goal is to demonstrate that such a model is not only effective but can be made scalable and tractable. We utilize decision trees to tackle the problem of sparse parameter estimation which is exacerbated by the use of syntactic information jointly with word context. While decision trees have been previously applied to language modeling, there has been little analysis of factors affecting decision tree induction and probability estimation for language modeling. In this thesis, we analyze several aspects that affect decision tree-based language modeling, with an emphasis on syntactic language modeling. We then propose improvements to the decision tree

induction algorithm based on our analysis, as well as the methods for constructing forest models—models consisting of multiple decision trees. Finally, we evaluate the impact of our syntactic language model on large scale Speech Recognition and Machine Translation tasks.

In this thesis, we also address a number of engineering problems associated with the joint syntactic language model in order to make it tractable. Particularly, we propose a novel decoding algorithm that exploits the decision tree structure to eliminate unnecessary computation. We also propose and evaluate an approximation of our syntactic model by word n-grams—the approximation that makes it possible to incorporate our model directly into the `CDEC` Machine Translation decoder rather than using the model for rescoring hypotheses produced using an n-gram model.

DECISION TREE-BASED SYNTACTIC LANGUAGE MODELING

by

Denis Filimonov

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2011

Advisory Committee:
Dr. Philip Resnik, Chair
Dr. Mary Harper, Advisor/Co-Chair
Dr. Jimmy Lin
Dr. Hal Daume III
Dr. Jeff Foster
Dr. Armand Makowski, Dean's Representative

Acknowledgments

First of all, I must say that no words can begin to express the degree of gratitude I feel towards the people who helped me in various ways on the long way to this point. Therefore, if I might appear brief, it is for the lack of words not the appreciation.

First and foremost, I must thank my advisor Professor Mary Harper who has taught me everything I know about doing research (sorry, the mandatory course "How to do research" did not turn out to be very helpful). I also must thank her for being an uncompromising editor and for taking the pain of going through countless versions of this thesis.

I am very grateful to all people in the CLIP Lab, especially Philip Resnik, to whose passion about computational linguistics I owe my interest in the field. I am thankful to Doug Oard for support in the last year. I would like to thank many fellow UMD students, particularly Zhongqiang Huang (without his parser the work that has led to this dissertation would not have even begun!) and Vlad Eidelman, who has been enormously helpful with the machine translation experiments.

I am also indebted to the CLSP group at Johns Hopkins University, especially Sanjeev Khudanpur and Damianos Karakos, for discussions and criticism–many of the ideas that made it into this dissertation (and some that didn't) underwent first public scrutiny there. I also would like to thank JHU students: Ariya Rastrow, Anoop Deoras, Scott Novotney and others for their input and help with speech recognition experiments. I am also grateful to the HLTCOE at JHU for financial

support, and its staff for tolerating my constant [ab]use of the computing cluster, especially in the last months.

Finally, I would like to thank all my friends and my family: they are the reason I have made it thus far without going [completely] crazy.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction and Overview of the Thesis

1.1 Motivation

A number of natural language applications including Automatic Speech Recognition (ASR), Machine Translation (MT), and Optical Character Recognition (OCR) need to distinguish between word sequences that belong to the language in question and the sequences that do not. This necessity comes from the fact that the observed data, be that an acoustic signal or a scanned image, may have originated from many possible sources. For instance, "*flavored popcorn*" may be virtually indistinguishable from "*flavored popcom*" in some fonts, and yet we would expect an OCR system to choose the former sequence. "*The sky is cloudy*" may sound similar to "*this guy is cloudy*;" however, the former transcription appears to be more plausible. Note that in the last example, it is not that "*this guy is cloudy*" is an impossible sentence, but it simply is *less likely* than "*the sky is cloudy*." The subsystem that estimates which word sequence is the most plausible from a given set of candidates is called a *Language Model* (LM). If the language model employs statistical methods, such as estimating a probability mass function $p(W)$, where W is an arbitrary word sequence, such model is called a *Statistical Language Model*[1].

[1] Sometimes Statistical Language Model is abbreviated as SLM. We do not use this abbreviation to avoid confusion with Structured Language Model. Instead, since this work concerns only statistical models, whenever we say a language model, we mean a statistical language model.

A wide variety of methods have been developed for language modeling (see an extensive survey by Goodman [46]), some of which are quite complex, and still, the most widely used models are the *n-gram* models, based on a simple Markov chain. While many complex models do outperform the n-gram models, the improvement is often not large enough to justify the increased computational complexity, as argued by Goodman [46]. "All hope abandon, ye who enter here," quotes he. While it is true, that it is usually more effective to add more data to an n-gram model than to use a more complex model trained on a smaller amount of data (both computationally and performance-wise), additional data is often simply not available in some domains and languages, leaving the use of linguistic knowledge, such as syntax and morphology, as possibly the only way to improve the performance. The use of additional types of information necessitates the construction of more complexly structured models than the n-gram model.

Another stimulus for developing more complex models comes from the observation that a language model is never used by itself, rather it is always a part of a bigger system, such as an ASR or a MT system. Therefore, the potential impact of the language model strongly depends on other components of the system. Indeed, it is not uncommon for ASR hypotheses in low-resource languages and domains to have word error rates of 50% or more. Under these conditions, it would be hard even for a human judge to select the best candidate, as none of them would probably make sense. It is reasonable to expect that, as other subsystems improve, the impact of the language model will increase, making complex models more desirable.

In this work, we use syntactic tags to augment the language model as a means

to increase the performance of the model. This choice comes from the strong evidence that there is syntactic structure in the language. Although the sentences "cats meow" and "dogs bark" consist of different words, they share syntactic structure — both contain a plural noun as the subject and a present tense verb. There are restrictions to that structure, e.g., "cats dogs" is not a correct sentence. Thus, being able to recognize correct syntax should improve the performance of the language model[2] by making syntactically implausible sentences less likely; indeed, prior work in syntactic language modeling (e.g. [3, 51]) shows that syntax can improve the performance of a language model substantially. However, the scalability issues that arise in such models are often overlooked (e.g., Heeman [51] only uses up to 1 million words of training data–a tiny amount for modern language models), making the practicality of such models questionable. In this thesis, we not only create an effective syntactic model, but we also identify and address computational problems that are vital for a practically useful language model.

Adding syntactic information to a language model exacerbates the problem of sparse parameter estimation. We choose decision trees to address this problem because they are known to work well with sparse data, and they also simplify the addition of various information sources to the model (e.g., prosody or morphological features) when they are available. Although there are some other methods that can address the problem of sparsity (e.g., log-linear models, neural networks), their computational complexity makes their application challenging even for large-scale

[2] From the syntactic perspective, "cats bark" is a perfectly good sentence. Therefore, we cannot rely *solely* on syntax.

word models. Applying these methods to joint syntactic language models does not appear to be feasible at present, because in these models, the computational cost depends on the vocabulary size linearly, and in joint models the vocabulary size can be extremely large. Decision trees are widely used for classification and regression, and they have been applied to language modeling. However, prior work on decision tree-based language modeling lacks an analysis of methods for decision tree induction and probability estimation specifically for language modeling. One of the main goals of this thesis is to analyze these specifics and devise decision tree-related methods that are more effective for the language modeling task. Additionally, rather than stopping at the proof-of-concept stage, we also aim to create a model that can utilize large amounts of data for training a syntactic decision tree-based LM, which is a daunting engineering task because of the computational complexity involved.

1.2 Structure of the Thesis

The thesis is organized as follows:

- Chapter 2 reviews prior work on statistical language modeling, concentrating on the problems that are most essential to this thesis: methods used for context clustering and the use of syntactic information in language modeling.
- Chapter 3 describes experimental setups that we will use throughout this thesis to evaluate the impact of our ideas.
- Chapter 4 describes in detail the construction of a decision tree-based language model, from decision tree induction to estimation of probability distribution,

as well as the combination of multiple decision tree models. We review prior work on decision tree-based language modeling and point out the existence of a bias in the decision tree induction algorithm. We propose a remedy that utilizes prior research in decision tree induction applied to classification; this research seems to have been largely overlooked in the literature on decision tree-based language modeling. We also investigate the problem of backoff in decision tree models and show that methods adopted from the literature on n-gram models do not apply well to decision tree-based models. Additionally, we investigate various methods for constructing forests of decision trees.

- Chapter 5 describes a number of engineering solutions that enable our model to scale to large amounts of data. Additionally, we introduce an approximation that allows us to apply our syntactic model to tasks for which such a model would be intractable otherwise.

- In Chapter 6, we present experiments demonstrating the impact of our model on large scale speech recognition and machine translation tasks, in addition to the experiments in Chapters 4 and 5 that are designed to test specific ideas.

- Finally, Chapter 7 presents the contributions of this thesis and discusses directions for future work.

1.3 Summary of Contributions

- We analyze the influence of the specifics of the language modeling (particularly syntactic language modeling) task on the decision tree construction algorithms in Section 4.2.

- We discover a bias in the metric for decision tree construction used in all prior work on decision tree-based language modeling, and we propose a novel metric that addresses this bias in Section 4.7.

- We investigate methods for interpolation of higher- and lower-order decision tree models, which in prior work were adopted from n-gram models. In Section 4.8, we discover that these interpolation methods imply a certain relation between the context clustering used in the higher- and the lower- order models. We formulate this property as the Backoff Property; and we observe that in decision tree-based models, this property is not typically satisfied, leading to a poor performance of the interpolation methods. We propose a generalized linear interpolation method that combines decision trees as a forest, i.e., all trees are treated in the same way, and we show that this interpolation method performs well even when the Backoff Property is not satisfied. Additionally, we investigate a number of techniques for constructing small (tractable) forests of decision trees.

- In Section 5.1, we tackle a number of engineering problems in order to make the model computationally tractable. Particularly, we propose a novel decoding

algorithm that exploits the decision tree structure to eliminate unnecessary computation.

- In Section 5.2, we show that a syntactic model can be accurately approximated by sufficiently long word sequences (n-grams). This approximation enables us to incorporate our model into the `CDEC` machine translation decoder [36].

- Finally, in Section 6.1, we demonstrate that our syntactic model significantly outperforms a standard n-gram model on a speech recognition task; and in Section 6.2, we observe on a machine translation task that the improvement of our syntactic model over a comparable n-gram model strongly correlates with the performance of the translation model, leading us to conclude that the impact of our model on machine translation will only increase as the translation models improve.

Chapter 2

Language Modeling Review

In this chapter, we formulate the problem of language modeling and review prior work in this area. We put a particular emphasis the analysis of two aspects of language modeling: context clustering and the use of syntactic information, because these are the main problems we address in this thesis.

2.1 Background

In this section, we present the definition of a statistical language model, and describe some of the applications of language models. We also describe metrics that are used for evaluation of language models.

2.1.1 What is Language Modeling?

Canonically formulated, a *Language Model* (LM) is a probability mass function that assigns a probability to any sequence of words.

$$LM = p(w_1 w_2 \dots w_n) \equiv p(w_1^n)$$

It is difficult to normalize a function with an infinite set of values, without some form of factorization; therefore, this equation is usually factored as follows[1]:

[1] Some models, notably Charniak [16], use a different factorization (see Section 2.1.3).

$$p(w_1^n) = \prod_{i=1}^{n} p(w_i|w_1^{i-1}) \tag{2.1}$$

Note that in this factorization, we only need to define a function that computes the probability of one word (drawn from a finite vocabulary) given prior words. Incidentally, the lack of future context is a requirement in some applications, such as Automatic Speech Recognition (ASR) or predictive text input[2], where the full word sequence is simply not available.

In the literature, the distribution of word probabilities given prior words $p(w_i|w_1^{i-1})$ is often called the *language model*, thus refining the definition of the problem. Henceforth, we will generally use this formulation for the language modeling task. In the function $p(w_i|w_1^{i-1})$, we will call w_1^{i-1} the *context* and w_i the *future* as in [17].

2.1.2 Applications of Language Modeling

Perhaps the most important application of language modeling stems from the *noisy channel* framework. For instance, the goal of an Automatic Speech Recognizer is to find the most likely sequence of words $\hat{w}_1^n$ given the acoustic input A. This probability cannot be estimated directly; therefore, it is factored using Bayes rule as follows:

[2] Predictive text input uses a language model to offer a user suggestions for subsequent words or characters as the user types. It is often used in handheld devices, where typing is difficult; and recently, it has become popular in web applications such as Google search.

$$\begin{aligned} \hat{w}_1^n &= \underset{w_1^n}{argmax}\ p(w_1^n|A) \\ &= \underset{w_1^n}{argmax}\ p(A|w_1^n) \cdot p(w_1^n) \end{aligned} \tag{2.2}$$

where the probability $p(A|w_1^n)$ is called the *acoustic model* and $p(w_1^n)$ is the language model in its canonical form.

In Machine Translation (MT), language modeling plays a similar role:

$$\begin{aligned} \hat{w}_1^n &= \underset{e}{argmax}\ p(e|f) \\ &= \underset{e}{argmax}\ p(f|e) \cdot p(e) \end{aligned}$$

where $e \equiv w_1^n$ is target language phrase, f is a source phrase, $p(f|e)$ is called the *translation* model and $p(e) \equiv p(w_1^n)$ is the language model.

ASR and MT are the most high profile applications that utilize LMs, but not the only ones; others include predictive text input, information retrieval, optical character recognition, among others. As Eq. 2.2 shows, the language model plays a vital role in applications using the noisy channel framework, and the quality of the language model directly affects the performance of the whole system, motivating research to improve the effectiveness of language models.

2.1.3 Types of Language Models

There is no widely accepted classification of language models. This is not surprising as language modeling is a complex task encompassing a number of problems, each

of which allows a variety of solutions. Thus, how we divide the existing models into classes depends on the purpose of the classification. From the perspective of this thesis, we consider the following types to be important:

1. **Generative word models**

   Models of this type predict the word w_i based on its *history* context h_i, and this history is a function of w_1^{i-1}, the words preceding w_i:

$$p(w_i|w_1^{i-1}) \approx p(w_i|h_i) \ ; \ h_i(w_1^{i-1}) \tag{2.3}$$

   How the history h_i is defined and how the distribution $p(w_i|h_i)$ is estimated varies from one modeling approach to another, but what is shared among language models of this type is the fact that the context h_i is *observed*, i.e., there is no ambiguity in the context.

   To produce a proper distribution over all word sequences w_1^n using the factorization in Eq. 2.1, the model in Eq. 2.3 must have a proper distribution $p(w_i|h_i)$ for any given history context h_i:

$$\forall_{h_i} \ : \ \sum_{w \in V} p(w|h_i) = 1$$

   where V is the *vocabulary* of the model. Although natural languages do not have a limited vocabulary (indeed, new words are invented daily), for the purpose of language modeling there is an assumption that the model's vocabulary V is fixed. Words outside V are typically mapped to a special token `<unk>`,

which is a token in V that is modeled as any other word. Such models are called *open vocabulary* models. In *closed vocabulary* models, words not included in V are simply ignored or assigned probability of 0. Typically, the vocabulary V includes tens of thousand to hundreds of thousand words, although in some specific tasks much smaller vocabularies can be used.

2. **Joint Models** utilize latent stochastic variables t_i along with the observed words w_i:

$$p(w_i t_i | w_1^{i-1} t_1^{i-1}) \approx p(w_i t_i | h_i) \ ; \ h_i(w_1^{i-1} t_1^{i-1}) \tag{2.4}$$

Note that, unlike words, these stochastic variables (*tags*) are not present in the input; therefore, the language model must hypothesize them, and in order to compute the probability of the word sequence, sum over all tag assignments:

$$p(w_1^n) = \sum_{t_1 \ldots t_n} \prod_{i=1}^{n} p(w_i t_i | w_1^{i-1} t_1^{i-1}) \tag{2.5}$$

Although the number of possible tag assignments is exponential in the length of the word sequence, some independence assumptions together with dynamic programming reduce the complexity to polynomial in the size of the tagset (Manning and Schütze [73]). Probability $p(w_i | w_1^{i-1})$ can be estimated as follows:

$$p(w_i|w_1^{i-1}) = \frac{p(w_1^i)}{p(w_1^{i-1})} = \frac{\sum_{t_1 \ldots t_i} \prod_{j=1}^{i} p(w_j t_j | w_1^{j-1} t_1^{j-1})}{\sum_{t_1 \ldots t_{i-1}} \prod_{j=1}^{i-1} p(w_j t_j | w_1^{j-1} t_1^{j-1})} \tag{2.6}$$

We will refer to this type of model as *joint model* since they estimate the joint distribution $w_i t_i$.

3. **Whole sentence language models**

   This type of model does not use the factorization in Eq. 2.1 and includes models that are based on a parser developed by Charniak [16] or exponential models with whole sentence features developed by Rosenfeld et al. [95]. Because this type of language model requires the whole sentence to be available before calculating the probability, the applicability of whole sentence language models is significantly limited.

   Charniak's approach [16] is based on a syntactic parser. Generally, a parser computes the probability $p(w_1^n \mathcal{T})$, where $\mathcal{T}$ is a grammar derivation of the word string w_1^n. Normally, the parser's task is to find the most likely parse derivation $\hat{\mathcal{T}}$, given the word sequence:

$$\begin{aligned} \hat{\mathcal{T}} &= \underset{\mathcal{T}}{argmax}\, p(\mathcal{T}|w_1^n) \\ &= \underset{\mathcal{T}}{argmax}\, p(w_1^n \mathcal{T}) \end{aligned}$$

   However, it is also possible to marginalize the probability of parse trees, yielding the word sequence probability:

$$p(w_1^n) = \sum_{\mathcal{T}} p(w_1^n \mathcal{T})$$

Parser-based models can benefit from deep syntactic structure; however, it is unclear how effective this approach would be with errorful input. In ASR and MT rescoring tasks, it is often the case that none of the hypotheses is syntactically well-formed, and in fact, the parser can fail on many of them.

Another type of whole sentence model was proposed by Rosenfeld et al. [95]. They used an exponential model in the following form:

$$p(w_1^n) = \frac{1}{Z} p_0(w_1^n) e^{\sum_k \lambda_k f_k(w_1^n)}$$

where Z is a normalization constant, p_0 is a baseline probability for the sentences (estimated using another model), and f_k are the features involving arbitrary information from the entire sentence w_1^n. Rosenfeld et al. [95] report very small improvements over a baseline n-gram model (Section 2.2.1), which suggests that the approach, although very interesting, requires further work before it is practical.

2.1.4 Evaluation Metrics

As in many fields of science, formal evaluations are pivotal for comparing the quality of language models. In this section, we describe metrics used to evaluate language models: one intrinsic metric based on likelihood of a test set and several extrinsic

metrics where the quality of a language model is evaluated by the impact on an application-specific metric.

2.1.4.1 Perplexity

Language models are typically evaluated using an intrinsic metric called *perplexity* (PPL), described by the following formula:

$$\text{PPL} = p(w_1^n)^{-\frac{1}{n}}$$

where w_1^n is a representative test set. This formula applies to all types of language models that compute $p(w_1^n)$, although they may use very different formulae to compute $p(w_1^n)$ (see Section 2.1.3).

In practice, all models that are factored as in Eq. 2.1 add a special special symbol (usually denoted `</s>`) to the end of each sentence (or utterance). This special symbol is needed to ensure that the probability of all word sequences sum to 1; moreover, without, the probability of a sentence "*I see the*" would be higher than the probability of "*I see the sun*" because according to Eq. 2.1, $p(I\ see\ the\ sun) = p(sun|I\ see\ the)p(I\ see\ the)$, which is counter-intuitive. While all models compute the probability $p(w_1^n)$ this way, they differ in whether to count the added symbol `</s>` as a word for perplexity computation. In our example "*I see the sun*," some models would calculate the perplexity as $\text{PPL} = p^{-\frac{1}{4}}$, while in others, $\text{PPL} = p^{-\frac{1}{5}}$.

While arguments can be made to support each of the ways of computing perplexity, this choice is not essential (in fact, the SRILM toolkit developed by

Stolcke [102] computes both). It is important, however, when comparing different models to ensure that the perplexities are computed in the same way because their difference is significant. In this work, we do not include `</s>` as a word for perplexity computation.

Researchers have observed, however, that perplexity of different models does not always correlate well with their relative performance in the task the LMs are applied to (Iyer et al. [56]); therefore, task-specific evaluations often accompany perplexity numbers, and the quality of language models is judged by their effect on the performance of an entire system.

2.1.4.2 Metrics for Speech Recognition

In speech recognition, models are usually evaluated by their ability to reduce the word error rate (WER) [55]. Word error rate is the Levenshtein distance between the reference word sequence and the hypothesis, i.e.,

$$WER = \frac{I + D + S}{|R|} \tag{2.7}$$

where I, D, and S are the numbers of word insertions, deletions, and substitutions, respectively, while $|R|$ is the number of words in the reference. WER is usually presented as a percentage; note that because of insertions, a WER higher than 100% is possible.

2.1.4.3 Metrics for Machine Translation

Metrics used for machine translation are more complex than WER used for speech recognition, partly because they must account for the fact that there is no *single* correct translation, the same idea can be expressed in many ways. Therefore, multiple reference translations are often used for evaluation. BLEU [86], the most commonly used metric, is defined as follows:

$$\begin{aligned} BLEU &= BP \cdot \prod_{n=1}^{N} (pr_n)^{w_n} \\ BP &= \min(1, e^{1-\frac{|R|}{|C|}}) \\ pr_n &= \frac{\sum_{C} \sum_{\text{n-gram} \in C} Count_{clip}(\texttt{ngram})}{\sum_{C} \sum_{\texttt{ngram} \in C} Count(\texttt{ngram})} \end{aligned} \tag{2.8}$$

where pr_n is the n-gram precision computed from clipped counts $Count_{clip}(\texttt{ngram})$, which is the maximum number of times the n-gram `ngram` appears in any single reference corresponding to the candidate translation C, BP is the *brevity penalty* intended to compensate for the tendency of precision metrics to favor shorter hypotheses, and $|R|$ and $|C|$ are the lengths of the best-match and candidate translations, respectively. Typically, the maximum n-gram order is 4 and the weights are equal: $w_n = \frac{1}{N}$.

Translation Error Rate (TER) [99] is similar to WER, except that in addition to insertions, deletions, and substitutions, phrasal shifts are allowed. Phrasal shift helps to account for variability in word order present in many languages. For example, if the reference is "I watched a movie last night" and the candidate translation

is "Last night I watched a movie," the phrasal shift will allow this perfectly good candidate translation to have only one error, while WER and BLEU would be quite poor.

HTER is similar to TER, except that editing is performed by a human expert, creating a new reference translation in the process. Other metrics exist (e.g., METEOR [33], F-measure [79]), although they are less commonly reported in the literature.

2.2 Review of Language Modeling Technology

In this section, we review the prior work on language modeling, beginning with n-gram models, which constitute by far the most widely used class of models because of their simplicity, computational effectiveness, and freely available implementations [26, 102, 52].

2.2.1 N-gram Language Models

An *n-gram* model is a Markov chain, i.e., it makes the assumption that the word w_i only depends on the immediately preceding $n-1$ words, where n is called the *order* of the model:

$$p(w_i|w_1^{i-1}) \approx p(w_i|w_{i-n+1}^{i-1}) \tag{2.9}$$

These models are usually estimated from counts of *n-grams*, which are tuples of n consecutive words, as follows:

$$p_{ML}(w_i|w_{i-n+1}^{i-1}) = \frac{c(w_{i-n+1}^{i})}{\sum_{w_i \in V} c(w_{i-n+1}^{i})} \tag{2.10}$$

where $c(w_{i-n+1}^{i})$ is the number of times the n-gram $w_{i-n+1}w_{i-n+2}\dots w_i$ occurs in a collection of text called the *training data*. Eq. 2.10 is a *maximum likelihood* (ML) estimation because it maximizes the likelihood of the training data. However, this estimation does not generalize well; indeed, any n-gram that was not observed in the training data would have zero probability under ML estimation[3]. Therefore, in practice, n-gram models *discount* probabilities of observed n-grams and distribute the discounted probability mass to unseen n-grams. This process is also often referred to as *smoothing* the probability distribution.

Many discounting algorithms have been developed for n-gram models. In the next three subsections, we describe some of the most frequently used methods. We refer to Chen and Goodman [20] for a comprehensive survey of discounting algorithms.

2.2.1.1 Katz Smoothing

Katz smoothing utilizes Good-Turing discounting for observed n-grams while unseen n-grams are estimated using a lower order *backoff* model:

$$p_{katz}(w_i|w_{i-n+1}^{i-1}) = \begin{cases} \frac{d_r r}{\sum_{w_i} c(w_{i-n+1}^{i})} & \text{if } r > 0 \\ \alpha(w_{i-n+1}^{i-1}) p_{katz}(w_i|w_{i-n+2}^{i-1}) & \text{if } r = 0 \end{cases} \tag{2.11}$$

[3] Unobserved n-grams are quite common: After training on 1.5 million words, 23% of the trigrams drawn from new text in the same collection of text are unseen (Manning and Schütze [73]).

where $r = c(w_{i-n+1}^{i})$ and d_r is the discounting ratio defined as follows:

$$d_r = \begin{cases} 1 & \text{if } r > k \\ \frac{\frac{r^*}{r} - \frac{(k+1)n_{k+1}}{n_1}}{1 - \frac{(k+1)n_{k+1}}{n_1}} & \text{if } 1 \leq r \leq k \end{cases}$$

where n_r is the number of n-grams occurring r times in the training data[4] and r^* is the Good-Turing estimate for n-grams occurring r times.

$$r^* = (r+1)\frac{n_{r+1}}{n_r}$$

Frequent counts $r > k$ (Katz proposed $k = 5$) are considered reliable and are not discounted, while infrequent n-grams with $1 \leq r \leq k$ are discounted proportionately to the Good-Turing estimate, and this ensures that the total probability mass assigned to unseen n-grams also matches that estimate. $\alpha(w_{i-n+1}^{i-1})$ in Eq. 2.11 is the backoff weight, chosen to ensure that the distribution $p_{katz}(w_i|w_{i-n+1}^{i-1})$ sums to 1.

Note that Eq. 2.11 contains a recursion: the same equation is used to estimate the backoff model $p_{katz}(w_i|w_{i-n+2}^{i-1})$. The base case for the recursion is the maximum likelihood estimation for the unigram model:

$$p_{katz}(w_i) = p_{ML}(w_i) = \frac{c(w_i)}{\sum_{w_i \in V} c(w_i)}$$

where V is the word vocabulary.

[4] Counts n_r are often also smoothed.

2.2.1.2 Jelinek-Mercer Smoothing

Jelinek-Mercer [58] smoothing utilizes linear interpolation of the maximum likelihood estimation with a lower order backoff model:

$$\begin{aligned} p_{interp}(w_i|w_{i-n+1}^{i-1}) &= \lambda(w_{i-n+1}^{i-1}) \cdot p_{ML}(w_i|w_{i-n+1}^{i-1}) \\ &\quad +(1-\lambda(w_{i-n+1}^{i-1})) \cdot p_{interp}(w_i|w_{i-n+2}^{i-1}) \end{aligned} \tag{2.12}$$

The base case for the recursion is the 0-th order model, which has a uniform distribution: $p_{uniform} = \frac{1}{|V|}$. Coefficients $\lambda(w_{i-n+1}^{i-1})$ are optimized on a heldout set using Expectation Maximization (EM) approach. Because the number of parameters λ can be very large, Jelinek and Mercer proposed partitioning them into a relatively small number of buckets, and then forcing all values of λ in one bucket to have the same value.

2.2.1.3 Kneser-Ney Smoothing

Chen and Goodman [20] formulate Kneser-Ney (KN) smoothing using the interpolated form, which, they argue, outperforms its original backoff form.

$$p_{KN}(w_i|w_{i-n+1}^{i-1}) = \frac{\max(c(w_{i-n+1}^{i}) - D, 0)}{\sum_{w_i} c(w_{i-n+1}^{i})} + D\frac{N_{1+}(w_{i-n+1}^{i-1}\bullet)}{\sum_{w_i} c(w_{i-n+1}^{i})} p_{KN}(w_i|w_{i-n+2}^{i-1}) \tag{2.13}$$

where D is the discount coefficient, and $N_{1+}(w_{i-n+1}^{i-1}\bullet)$ is the number of unique words following context w_{i-n+1}^{i-1}:

$$N_{1+}(w_{i-n+1}^{i-1}\bullet) = |\{w_i : c(w_{i-n+1}^{i}) > 0\}|$$

Unlike previously described smoothing methods, in KN discounting, the lower order distribution $p_{KN}(w_i|w_{i-n+2}^{i-1})$ is estimated from *modified* counts, where number of observed n-grams $c(w_i|w_{i-n+2}^{i-1})$ is replaced with the number of *unique* words preceding w_{i-n+2}^{i}. D can be optimized using deleted interpolation, although commonly the following heuristic value is used:

$$D = \frac{n_1}{n_1 + 2n_2}$$

where n_1 and n_2 are the numbers of n-grams that occur in the training data exactly one and two times, respectively.

Chen and Goodman [20] also propose a variant of Eq. 2.13 where D is a function of count $c(w_{i-n+1}^{i})$, which takes the following values:

$$D(c(w_{i-n+1}^{i})) = \begin{cases} 0 & \text{if } c(w_{i-n+1}^{i}) = 0 \\ D_1 & \text{if } c(w_{i-n+1}^{i}) = 1 \\ D_2 & \text{if } c(w_{i-n+1}^{i}) = 2 \\ D_{3+} & \text{if } c(w_{i-n+1}^{i}) \geq 3 \end{cases}$$

As in the standard KN discounting, the coefficients D_1, D_2, D_{3+} can be optimized, but in practice the following values are used:

$$
\begin{aligned}
Y &= \frac{n_1}{n_1 + 2n_2} \\
D_1 &= 1 - 2Y\frac{n_2}{n_1} \\
D_2 &= 2 - 3Y\frac{n_3}{n_2} \\
D_{3+} &= 3 - 4Y\frac{n_4}{n_3}
\end{aligned}
$$

Modified KN smoothing outperforms most of the other discounting techniques [20]; therefore, we will use it for smoothing our baseline n-gram models.

2.2.2 Class-based Models

Class-based language models were defined in Brown et al. [10] as follows:

$$p(w_i|w_1^{i-1}) = p(w_i|c_i)p(c_i|c_1^{i-1})$$

where c_i is the *word class* drawn from the class vocabulary C, and each word belongs to exactly one word class[5]. Similarly to n-gram models, the context is typically limited to $n-1$ immediately preceding classes:

$$p(w_i|w_1^{i-1}) \approx p(w_i|c_i)p(c_i|c_{i-n+1}^{i-1}) \tag{2.14}$$

Models of this type aim at reducing the parameter space of the model (thus, reducing the model's memory footprint, as well as alleviating the sparsity) by grouping words that behave similarly into one class. The n-gram model in Eq. 2.9 has at most $|V|^n$

[5] Sometimes models allowing multiple class membership are also called class-based; however, they require a very different probability computation (see Eq. 2.5). Therefore, we will describe them as a special case of joint language models in Section 2.2.3.

probabilities to estimate, where V is the word vocabulary. The class-based model in Eq. 2.14, on the other hand, has at most $|V| + |C|^n$ parameters. Typically $|C| << |V|$, which results in a significant reduction of the model size.

Class-based models exploit the observation that some words tend to appear in similar contexts, e.g., *Thursday* and *Friday*, and therefore, it might be beneficial to combine these words together for modeling purposes, i.e., assign them to one word class. Word classes are generally created using one of the following clustering approaches:

- *Bottom-up*: Each word starts in its own class, then the algorithm proceeds by merging classes until the desired number is reached, as in Brown et al. [10].
- *Top-down*: The algorithm starts with all words in one class and then proceeds by splitting classes, as was done by Zitouni [115].
- *Non-hierarchical*: Words assigned to classes randomly, and then the algorithm moves words from one class to another, trying to optimize some metric, as in Martin et al. [77].

While class-based models are able to reduce the size of the model, they generally fail to outperform n-gram models significantly [10, 115], and often perform more poorly unless interpolated with an n-gram model.

One of the shortcomings of class-based models is that they fail to distinguish different syntactic roles a word can play. For instance, the word *can* in the previous sentence is a modal verb; however, it can also be a singular noun as in "a *can* of

soup", or a verb as in "to *can* goods." The distributions of words following *can* are very different in each of these cases, but if the model does not distinguish them, it effectively unifies these distributions into one, "blurring" these uses at the expense of the less frequent meanings[6]. A clustering algorithm is likely to group the word *can* with other modal verbs since it is the predominant usage, further suppressing other meanings of this word.

Another phenomenon that class-based models do not handle well is *collocation*, i.e., a sequence of words that is used together far more frequently than a random chance would allow, for example, "white house" or "new york stock exchange." Indeed, "stock" and "share" are synonymous in the financial domain; however, "stock exchange" is common, but "share exchange" is not. If a class-based model groups these words into one class, it would lose the ability to make that distinction and would result in a poorer quality probability estimation for $p(\text{exchange}|\text{stock})$ than an n-gram model.

2.2.3 Joint Models

Heeman [50] points out that distributions of words that follow the word "loads" would be quite different when this word is a noun than when it is a verb. He argues that in those cases we have, in fact, two *different* words which happen to have the same *surface form*. In order to disambiguate the unknown "true" word, we can represent it as a tuple $w_i t_i$, where w_i is the surface word, as used in n-gram models,

[6] N-gram models suffer from this problem as well, but in higher order models the issue is largely remedied by other words in the context.

and t_i is a *tag* assigned to the word to help to distinguish its different roles. Tags can be thought of as word classes, but since in every particular case the true tag is hidden and often ambiguous, we must assume stochastic class membership:

$$\sum_{t_i \in T} p(t_i|w_i) \equiv 1$$

where T is the vocabulary of tags. This seemingly simple change from deterministic to stochastic classes complicates the probability computation significantly. In order to compute the probability of a word sequence, one must sum over all possible tag sequences, as shown in Eq. 2.5. If we apply Markovian independence assumptions similar to Eq. 2.9, we get:

$$p(w_1^m) = \sum_{t_1 \ldots t_m} \prod_{i=1}^{m} p(w_i t_i | w_{i-n+1}^{i-1} t_{i-n+1}^{i-1}) \tag{2.15}$$

The complexity of direct computation of Eq. 2.15 is prohibitive: $O(m \cdot |T|^m)$. However, this model is very similar to a Hidden Markov Model (HMM)[7], so a dynamic programming algorithm can be used to reduce the computation to $O(m \cdot |T|^n)$, where n is the order of the model (a fixed parameter). This may still be too expensive for a large tagset T; therefore, other methods, similar to beam search, are often used to reduce the complexity further, e.g., Heeman [50] used the following approximation:

$$p(w_i|w_1^{i-1}) \approx \frac{\sum_{t_i \in T} \sum_{t_1 \ldots t_{i-1} \in P_i} p(w_i t_i | w_1^{i-1} t_1^{i-1}) p(w_1^{i-1} t_1^{i-1})}{\sum_{t_1 \ldots t_{i-1} \in P_i} p(w_1^{i-1} t_1^{i-1})}$$

where P_i is a limited set of the most likely tag sequences $t_1 \ldots t_{i-1}$.

[7] The standard HMM uses a simpler model, shown in Eq. 2.16, but the decoding algorithm is essentially the same.

Note that the parameter space of this model is larger than that of an n-gram model: $O(|T|^n|V|^n)$ vs. $O(|V|^n)$. In order to reduce the model size, some models (e.g., [83]) make further approximations, similar to Eq. 2.14:

$$p(w_i t_i | w_{i-n+1}^{i-1} t_{i-n+1}^{i-1}) \approx p(w_i|t_i) p(t_i | t_{i-n+1}^{i-1}) \tag{2.16}$$

When the set of tags T is small, compared to the vocabulary V, the number of parameters in this model, $O(|T|^n + |V||T|)$, can be significantly smaller than that of an n-gram model. That reduction, however, comes at a price; models with reduced context often perform worse than the n-gram model, while models utilizing full context show significant improvement over n-gram models [83, 50, 40]. Niesler and Woodland [83] use a model similar to Eq. 2.16 and report an 11% and a 49% *increase* in perplexity compared to a word n-gram model in experiments on different corpora, although their model is much more compact—their model has an order of magnitude fewer parameters than the word n-gram model. In contrast, Heeman [50] shows that using full joint context ($p(w_i t_i | w_{i-n+1}^{i-1} t_{i-n+1}^{i-1})$) provides a 10% reduction of perplexity over a word n-gram model, while the approximation in Eq. 2.16 produces a model with 70% higher perplexity than the word n-gram model.

Another interesting model that can be loosely classified as a joint model was developed by Chelba [17]. This model is based on a parser that, unlike Charniak's model [16], constructs the parse trees using the following model:

$$p(w_1^n \mathcal{T}) = \prod_{k=1}^{n+1} p(w_k | w_1^{k-1} \mathcal{T}_{k-1}) p(\mathcal{T}_{k-1}^k t_k | w_1^k \mathcal{T}_{k-1})$$

where $\mathcal{T}_k$ is a parse derivation covering w_1^k, t_k is a POS tag, and $\mathcal{T}_{k-1}^k$ is a parser operation that transforms $\mathcal{T}_{k-1}$ into $\mathcal{T}_k$. The number of words is $n+1$ in the equation because of the end-of-sentence symbol. Similarly to Charniak's model [16], the probability of the word sequence is obtained by marginalization:

$$p(w_1^n) = \sum_{\mathcal{T}} p(w_1^n \mathcal{T})$$

Chelba [17] reports significant improvements compared to an n-gram model in perplexity and WER when his model is interpolated with the n-gram model.

2.2.4 Maximum Entropy Models

The maximum entropy principle states that a model should satisfy some explicitly specified constraints but should make no assumptions beyond that, i.e., it should have maximum entropy. The model is represented in the exponential form:

$$p_{ME}(w_i|w_1^{i-1}) = \frac{1}{Z(w_1^{i-1})} e^{\sum_j \lambda_j f_j(w_i, w_1^{i-1})} \tag{2.17}$$

where f_j are *features*, λ_j are their weights, and Z is the normalization term: $Z(w_1^{i-1}) \equiv \sum_{w_i} e^{\sum_j \lambda_j f_j(w_i, w_1^{i-1})}$. Typically boolean features are used. The model is usually constrained to have the same expected value for each feature as in the training data:

$$\forall_{f_j} : \sum_{(w_i, w_1^{i-1}) \in \mathcal{T}} p(w_i|w_1^{i-1}) f_j(w_i, w_1^{i-1}) = \sum_{(w_i, w_1^{i-1}) \in \mathcal{T}} p_{ME}(w_i|w_1^{i-1}) f_j(w_i, w_1^{i-1})$$

where $\mathcal{T}$ is the training data set and $p_{ME}(w_i|w_1^{i-1})$ is the empirical distribution on $\mathcal{T}$.

Advantages of maximum entropy models include the ease of incorporating arbitrary features and resilience to data fragmentation. Training and using a maximum entropy model is computationally costly, although there have been efforts to reduce this complexity. Wu and Khudanpur [110] proposed optimizations to the training algorithm that reduce the training time by over an order of magnitude. In addition, Wu and Khudanpur [109] proposed using an approximation for the normalization factor Z in Eq. 2.17, that could be precomputed during the training, reducing the computational cost of using the model by two orders of magnitude.

2.2.5 Other Models

2.2.5.1 Caching LM

If a person uses a word in the recent past, he or she is more likely to use that word again. Caching language models [69, 57] exploit this observation by interpolating a (static) language model estimated from training data using any of the previously discussed methods with a small LM estimated from recently observed test data, usually based on a running window of 1000 words.

$$p_{cache}(w_i|w_{i-n+1}^{i-1}) = \lambda p_{static}(w_i|w_{i-n+1}^{i-1}) + (1-\lambda)p_{dynamic}(w_i|w_{i-n+1}^{i-1})$$

Since $p_{dynamic}$ has to be updated after every word, sophisticated smoothing techniques are not used for its estimation. Jelinek et al. [57] propose using a linear interpolation of maximum likelihood estimated distributions:

$$p_{dynamic}(w_i|w_{i-n+1}^{i-1}) = \lambda_1 p_{ML}(w_i|w_{i-1}w_{i-2}) + \lambda_2(1-\lambda_1)p_{ML}(w_i|w_{i-1}) + (1-\lambda_2)(1-\lambda_1)p_{ML}(w_i)$$

Kuhn and de Mori [69] and Jelinek et al. [57] report significant improvements over an n-gram model; however, Goodman in his survey of language modeling techniques [46] was unable to observe improvement in WER, despite a significant reduction of perplexity. The likely explanation for this behavior is that in many real-world applications, such as speech recognition, the true transcript is not available and a caching model caches the output of a speech recognizer, thus reinforcing the errors it makes.

2.2.5.2 Neural Network LM

Neural networks have also been used in language modeling, e.g., by Bengio et al. [5] and Schwenk and Gauvain [96]. Relatively simple feedforward neural networks (NNs) are often used, consisting of one *projection* layer (in which discrete words are mapped to a continuous representation), one *hidden* layer, and an *output* layer (at which each node corresponds to a word in the output vocabulary). Note that in order to reduce computational complexity, both in training and in applying the model, the output vocabulary is often reduced to a *short list* of the most frequent words, while a simpler model (such as an ngram) is used to compute probabilities of rare words.

Recently, Mikolov et al. [80] proposed a language model based on a *recurrent*

neural network. Unlike feedforward NNs, which have limited context (similar to n-gram models), recurrent LMs are capable of capturing long-distance context by using a recurrent context layer. Mikolov et al. [80] report strong improvements over a KN n-gram LM. However, neural networks are very slow to train and therefore are rarely trained with more than a few million words of training data.

2.2.5.3 Discriminative Language Models

Unlike generative language models, discriminative models do not estimate the probability distribution over sequences of words $p(w_1^n)$. Instead, they rank a given set of hypotheses without computing their probabilities, sometimes utilizing additional information (e.g., the acoustic input [94]).

Stolcke et al. [100] constructed an n-gram model from misrecognized n-grams (obtained from n-best lists) and used this model with a negative weight to discourage likely errors. Roark et al. [94] used a linear model:

$$\hat{W} = \underset{W \in GEN(A)}{\arg\max}\ \Phi(W, A) \cdot \overline{a}$$

where A is the acoustic input, W is a word sequence from the set of hypotheses $GEN(A)$; $\Phi(W, A)$ is a vector of features, and $\overline{a}$ is the vector of corresponding weights. A perceptron was used to optimize the weights vector $\overline{a}$. Note that unlike generative models, which are optimized to increase likelihood of an ostensibly representative development text, discriminative models are often optimized to reduce recognition error directly.

Discriminative models are designed to complement generative LMs rather than replace them; therefore, despite their success, they do not reduce the importance of improving generative language models.

2.3 Context Clustering in Language Modeling

Despite strong independence assumptions, e.g., restricting the context to $n-1$ immediately preceding words in n-gram models, the context space of a language model is still far too large in all but the most trivial language models, necessitating further approximations. Such approximations generally come in the form of context clustering:

$$p(w_i|w_{i-n+1}^{i-1}) \approx p(w_i|\Phi(w_{i-n+1}^{i-1}))$$

where Φ is the clustering function $\Phi(w_{i-n+1}^{i-1}) \rightarrow \{\Phi^1, \ldots, \Phi^N\}$, N is the number of clusters, and clusters Φ^k are disjoint subsets of the context space. This clustering function may be specified either explicitly or implicitly. In this section, we discuss various types of context clustering methods used in current models.

2.3.1 N-gram Models

In an n-gram model (Section 2.2.1), when a context w_{i-n+1}^{i-1} has not been observed in training data, all probability mass is usually assigned to the lower order model, because $\forall_{w_i} c(w_{i-n+1}^{i}) = 0$. That is, the distribution $p(w_i|w_{i-n+1}^{i-1})$ is defined by the longest *observed* suffix substring of the context w_{i-n+1}^{i-1}, and all contexts with the

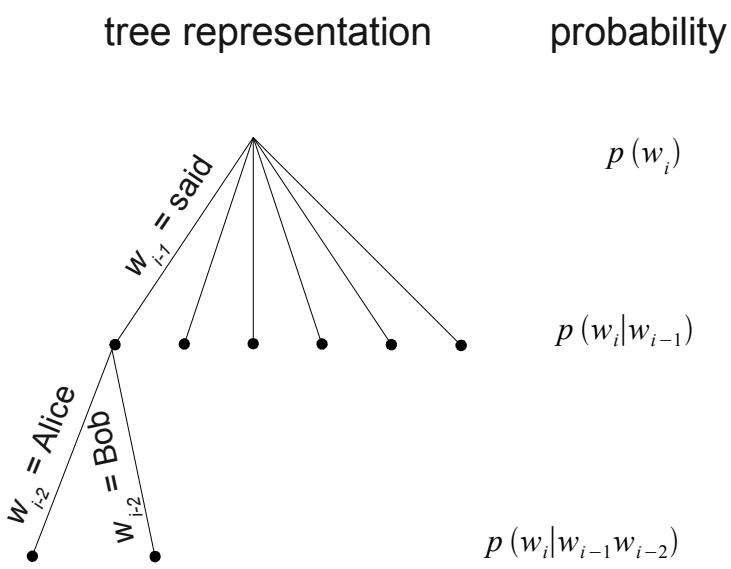


Figure 2.1: Hierarchical clustering in n-gram models

same suffix will have the same probability distribution, i.e., they belong to one cluster. This type of clustering function can be represented by a k-ary decision tree, as depicted in Figure 2.1. In this figure, a trigram model using at most two context words is shown. For a known context, e.g, "Alice said," the distribution $p(w_i|\text{Alice said})$ is used (shown at the bottom of the tree), and for an unseen context, e.g., "Carol said," the bigram backoff $p(w_i|\text{said})$ will be used.

Note that this is a very constrained form of a decision tree, and is probably suboptimal. Indeed, it is likely that some of the clusters predict very similar distributions of words. In the example in Figure 2.1, one would expect the distributions $p(w_i|\text{Alice said})$ and $p(w_i|\text{Bob said})$ to be very similar; hence, the model could benefit from merging these two clusters together, as it would lead to more robust probability estimation.

Several modifications to this clustering scheme have been proposed. Kneser [64] and Stolcke [101] pruned least informative n-grams reducing the number of parameters without degrading the performance appreciably. They also showed that increasing the order of the n-gram model while keeping the number of parameters constant through pruning can improve performance. Niesler and Woodland [83] proposed a somewhat similar idea, but instead of pruning non-informative n-grams, they added more context where it was beneficial[8].

The n-gram history clustering approach makes several poor assumptions. First, it makes the assumption that the immediately preceding word is more informative than the second-most, which is not always true[9]. Second, it makes the assumption that all siblings in the clustering tree in Figure 2.1 are "equivalently alike" in the sense that they back off to the same distribution $p(w_i|w_{i-1})$. Particularly, this implies that the distribution $\tilde{p}(w_i|w_{i-2}^{i-1}) = \tilde{p}(w_i|w_{i-1})$ if $C(w_{i-2}^{i-1}) = 0$, i.e., it depends only on w_{i-1} for all w_{i-2} for which $C(w_{i-2}^{i-1}) = 0$; this is not always true (consider the words that are likely to follow "The can" and "I can").

Zitouni [115], instead of backing off from the n-gram right to the $(n-1)$-gram, $w_{i-n+1}^{i-1} \rightarrow w_{i-n+2}^{i-1}$, used a more gradual backoff:

$$w_{i-n+1}^{i-1} \rightarrow F^1(w_{i-n+1}), w_{i-n+2}^{i-1} \rightarrow F^2(w_{i-n+1}), w_{i-n+2}^{i-1} \rightarrow \ldots \rightarrow w_{i-n+2}^{i-1}$$

where F^j is a sequence of clustering functions with F^{j+1} being coarser than F^j. This

[8] Niesler and Woodland [83] used n-grams of POS tags (Eq. 2.16) rather than words as in Kneser [64] and Stolcke [101], but for the purpose of context clustering this is not essential.

[9] *Skip* models, such as Martin et al. [76], mitigate this problem by interpolating models in which different context words are "skipped," i.e., $p'(w_i|w_{i-1}w_{i-2}w_{i-3}) = \lambda_1 p(w_i|w_{i-1}w_{i-2}) + \lambda_2 p(w_i|w_{i-1}w_{i-3}) + (1-\lambda_1-\lambda_2)p(w_i|w_{i-2}w_{i-3})$

type of clustering can also be represented as a decision tree similar to Figure 2.1. The only transformation is that at each node, instead of the flat k-ary structure, siblings are organized into a *binary* subtree with the same leaves and the root. Note that this binarization of the decision tree is performed based on *global* similarity between words, rather than similarity within a specific context, which, as we discussed in Section 2.2.2, has the drawback of mixing secondary usages of words into the same cluster. Zitouni [115] reported a 6% decrease of perplexity on *unseen* n-grams, although the overall perplexity reduction was only 2% compared to a Katz backoff word n-gram model.

2.3.2 Class-based Models

From the context clustering perspective, class-based models (Eq. 2.14) are structured similarly to n-gram models shown in Figure 2.1, except that instead of words, word classes are used.

On the one hand, merging words into one class enables more robust probability estimation. On the other hand, as we pointed out in Section 2.2.3, since many surface words have multiple uses, it is likely that they will be merged based on their most common use, resulting in an odd distribution when a word is used in a less common meaning. For example, the words "may" and "can" are most frequently used as modal verbs and, based on that usage, they tend to have very similar words surrounding them. Therefore, they are very likely to be clustered together by any word classification algorithm. This results in an improved probability estimation

when the words are used as modal verbs, but it also forces the n-grams "a can of" and "a may of" to have the same probability.

2.3.3 Decision Tree Models

In contract to the clustering methods discussed above, unconstrained decision trees may be grown to produce *any* partitioning of the context space. This implies that an *optimal* partitioning, i.e., the partitioning that minimizes some loss function, can be constructed. Unfortunately, construction of an optimal decision tree has been proven to be NP-complete [30]. Hence, in practice, we can only approximate an optimal decision tree by using greedy tree construction methods. Unless otherwise specified, we will utilize *binary* decision trees because they are simpler than k-ary trees, and they are equally powerful.

Strengths of decision trees include:

- Optimal partitioning is theoretically achievable. As we have pointed out above, any context clustering can be represented by a decision tree, unlike, for example, n-gram models, which impose constraints on how the context can be clustered and there is no theoretical justification that the optimal clustering would satisfy those constraints. While construction of an optimal decision tree is likely to remain unattainable, we can hope to approach it by improving greedy tree construction algorithms.

- Any number of attributes can be easily added to the model. Although attributes can be added to n-gram models as well, the type of clustering uti-

lized in n-gram models requires a sequence of backoff models. In the n-gram models described in Section 2.2.1, such a sequence is straightforward; it is based on the observation that more distant words carry less information about the future word. However, if additional attributes, such as tags in joint models (Section 2.2.3), or morphological features such as a word stem, or a suffix are added, it becomes unclear which model should be the backoff for $p(w_i t_i | w_{i-n+1}^{i-1} t_{i-n+1}^{i-1})$: $p(w_i t_i | w_{i-n+1}^{i-1} t_{i-n+2}^{i-1})$, $p(w_i t_i | w_{i-n+2}^{i-1} t_{i-n+1}^{i-1})$, or $p(w_i t_i | w_{i-n+2}^{i-1} t_{i-n+2}^{i-1})$? Sometimes the choice of the backoff path is made by trial and error [105]. Bilmes and Kirchhoff [8] propose a generalized backoff model where multiple backoff paths can be specified. However, both of these approaches quickly become unwieldy as the number of attributes grows. Decision trees, on the other hand, have no limitations on the number of attributes. They rely on information theoretic metrics to make partitioning decisions.

- There is a natural hierarchy of clusters. Because of sparsity, reliance on backoff models, i.e., coarser clustering, is essential in language modeling. Decision trees provide a natural hierarchy of increasingly coarse clusters from a leaf to the root.

Weaknesses:

- Decision tree construction is computationally intensive (see Section 4.1).
- There is potential for data fragmentation. As the decision tree construction algorithm progresses, it partitions the training data, thus consequent decisions

are based on increasingly smaller amounts of data, leading to the tendency to overfit the training data.

2.4 Syntax in Language Modeling

It is generally accepted that human languages have structure. For example, in a typical English sentence, the subject is followed by the main verb, and then optionally by the object and other verb arguments. Therefore, knowing the syntactic roles of each of the words w_1^{i-1} would certainly help to predict the role of the following word, as well as the word itself, and thus should help to improve the language model.

A number of formalisms has been used to capture the structure of a sentence, such as context free grammars [24], dependency grammars [49], constraint dependency grammars [78, 48], and link grammars [98]. Additionally, parsing tools has been developed to automatically generate syntactic structure for a given sentence. These tools can be broadly divided into two categories: *parsers* and *taggers*. Parsers generate *deep* syntactic structure that encompasses the entire sentence, while taggers produce *shallow* syntactic information, assigning syntactic tags to individual words without defining the relations among all of the words in the sentence [60, 73]. In the remainder of this section, we will review some of the methods used to represent syntactic structure.

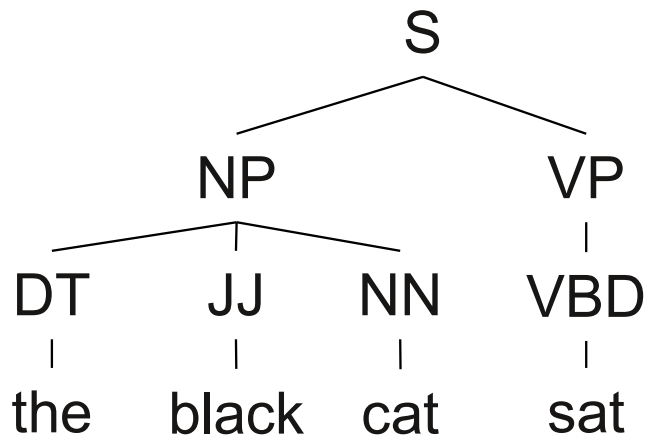


Figure 2.2: A parse tree example

2.4.1 Parse Trees

In our work, we use two types of parse trees: *constituency* trees and *dependency* trees. Constituency trees are based on the observation that some phrases (constituents) play certain roles in a sentence and can be replaced by other constituents of the same type.

Figure 2.2 illustrates a constituency parse tree for the sentence "the black cat sat." Note that the noun phrase (NP) constituent "the black cat" can be replaced with another noun phrase, e.g., "a bird with a long tail," and the entire sentence would remain syntactically correct.

Relationships among different constituents are typically described using *context free grammars* (CFG) [24]. Formally, a context free grammar is defined as a tuple of finite sets of *terminals*, *non-terminals*, and *productions*, with one of non-terminals designated as the start state. Productions are rewrite rules with one non-terminal on the left hand side and any sequence of terminals and non-terminals

on the right hand side. However, in natural language processing, a slightly more restrictive form of productions is used; there is a special subset of non-terminals called *pre-terminals*[10], such that productions with a pre-terminal on the left-hand side always rewrite to a single terminal; whereas, productions with a non-terminal (excluding pre-terminals) on the left-hand side can only produce non-terminals (including pre-terminals) on the right-hand side. It is common to refer to non-terminals as the set of non-terminals excluding pre-terminals; we will follow this convention in subsequent discussion. For example, the following productions are necessary to produce the parse tree in Figure 2.2:

$$
\begin{aligned}
S &\rightarrow NP\ \ VP \\
NP &\rightarrow DT\ \ JJ\ \ NN \\
VP &\rightarrow VBD \\
DT &\rightarrow the \\
JJ &\rightarrow black \\
NN &\rightarrow cat \\
VBD &\rightarrow sat
\end{aligned}
$$

In a probabilistic context free grammar (PCFG) [9], probabilities are assigned to each production in such a way that the probabilities of productions with the same left hand side sum to 1. The probabilities can be estimated from hand-annotated treebanks [15, 27].

[10] Pre-terminals include the nodes labeled DT, JJ, NN, and VBD in the example in Figure 2.2.

Note that a PCFG makes an extremely strong independence assumption, namely that the probability of each constituent is independent of the context. For example, the probability of production $S \rightarrow NP \;\; VP$ does not depend on the NP and VP constituents despite the fact that in natural languages these dependencies are very strong. For example, inanimate objects are unlikely to be the subject of some verbs, e.g., "eat." For this reason, simple PCFG grammars perform poorly. To overcome this problem, state-of-the-art PCFG parsers condition the production probabilities on context either explicitly [15, 63] or implicitly by assigning latent variables to non-terminals [87, 53].

Dependency trees are another way of representing syntactic relations [49, 75]. An example of a dependency parse tree is shown in Figure 2.3. A dependency tree is a directed graph where each edge represents a relation between two words: the *head* (or *governor*) and its *dependent* (or *modifier*). A word can have multiple modifiers. The edges can be labeled to denote the specific role of the relation. In the example in Figure 2.3, the dependency parse produced by the Stanford Dependency Parser [75] indicates that "the" and "black" are the determiner and adjectival modifier for "cat," which in turn is the nominal subject for "sat." Sometimes additional dependencies are added to a dependency tree (which becomes a more general form of graph) to represent the fact that a word can be related to multiple words in a sentence. For example, in Figure 2.4, the word "that" is the subject for the main verb "driving," the head of the relative clause "that were driving fast before," but at the same time it is a referent to the word "ones."

Note that a dependency structure can be extracted from a constituency parse

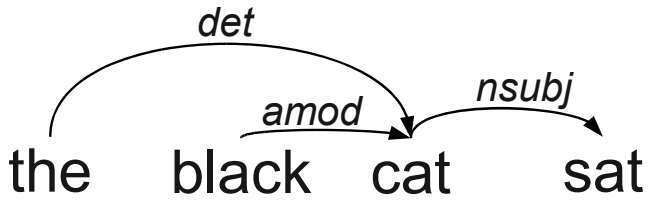


Figure 2.3: A dependency parse tree example

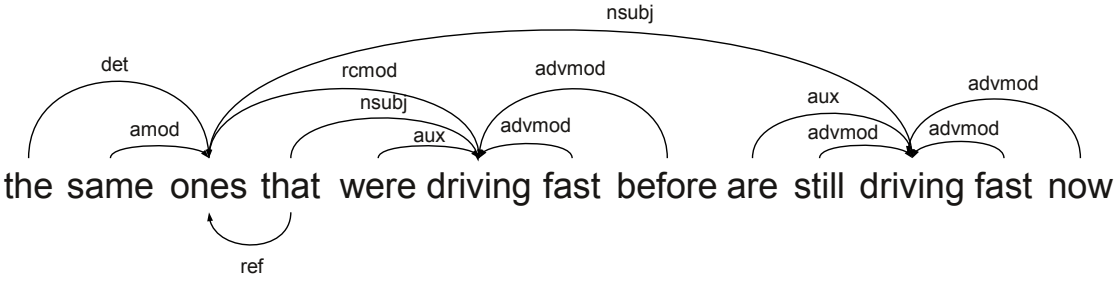


Figure 2.4: Non-tree dependency graph

tree by applying a set of deterministic rules, often referred to as *headword percolation* rules. However, it is often not trivial to infer a constituency tree from a dependency parse.

2.4.2 Syntactic Tags

Unlike parse trees, which cover the entire sentence in a single coherent structure, tags are assigned to individual words and therefore express roles that words play in the sentence without defining inter-word relations. Although tags contain only shallow syntactic information, they are more suitable as the source of syntactic information for language modeling than parse trees because language models are often required

to operate in a left-to-right manner, without having access to the entire sentence[11]. Additionally, sentence boundaries are often not available to a language model, which further complicates deriving a full parse structure. In the remainder of this section, we discuss various types of tags, some of which have been used in prior work and some are novel. Finally, in Section 2.4.2.6, we propose information theoretic metrics to estimate the usefulness of tagsets for language modeling.

2.4.2.1 Part-of-Speech (POS) Tags

The number of POS tags can vary greatly depending on the language and the purpose of the tagset. For example, Penn treebank English tagset contains 36 tags excluding punctuation (see Table 2.1), while the Urdu tagset proposed by Rabbi et al. [90] has 280 tags (the increase in the number of tags was largely to express gender and number agreement).

The amount of information POS tags typically provide is very limited. For example, while it is helpful to know whether *fly* is a verb or a noun, knowing that *you* is a personal pronoun does not carry information about whether it is a subject or an object (given the Penn Treebank tagset), which would certainly help to predict the following word. Most words can also have more than one tag assignment (*to* is a notable exception – it has a dedicated tag TO). For instance, the word *can* can be a modal verb (MD), a singular nominal noun (NN), a singular proper noun (NNP), a verb in the base form (VB), or present tense singular non-3rd person verb (VBP).

[11] Parsers that parse sentences left-to-right do exist, e.g., [17], but their performance is substantially worse than whole-sentence state-of-the-art parsers.

Tag	Description	Tag	Description
CC	Coordinating conjunction	PRP$	Possessive pronoun
CD	Cardinal number	RB	Adverb
DT	Determiner	RBR	Adverb, comparative
EX	Existential *there*	RBS	Adverb, superlative
FW	Foreign word	RP	Particle
IN	Preposition or subordinating conjunction	SYM	Symbol
JJ	Adjective	TO	*to*
JJR	Adjective, comparative	UH	Interjection
JJS	Adjective, superlative	VB	Verb, base form
LS	List item marker	VBD	Verb, past tense
MD	Modal	VBG	Verb, gerund or present participle
NN	Noun, singular or mass	VBN	Verb, past participle
NNS	Noun, plural	VBP	Verb, non-3rd person singular present
NNP	Proper noun, singular	VBZ	Verb, 3rd person singular present
NNPS	Proper noun, plural	WDT	Wh-determiner
PDT	Predeterminer	WP	Wh-pronoun
POS	Possessive ending	WP$	Possessive wh-pronoun
PRP	Personal pronoun	WRB	Wh-adverb

Table 2.1: Penn Treebank part-of-speech tags

2.4.2.2 SuperARV

The SuperARV [106] essentially organizes information concerning one consistent set of dependency links for a word that can be directly derived from its syntactic parse based on knowledge of head dependencies, case roles, and lexical features in English. SuperARVs encode lexical information, as well as syntactic and semantic constraints, in a uniform representation that is much more fine-grained than POS. A SuperARV is a four-tuple $(C; F; R+; D)$, where C is the lexical category of the word, F is a vector of lexical features for the word, $R+$ is a set of governor and need labels that indicate the function of the word in the sentence and the types of words it needs, and D represents the relative position of the word and its dependents. SuperARVs can be produced from parse trees by applying deterministic rules. An example of SuperARV tag assignments for a sentence is shown in Table 2.2. In the row labeled **C**, the lexical categories are assigned to the words. Note that these categories are coarser than POS tags, e.g., *verb* includes verbs of all types. Features are assigned in row **F** to distinguish among different types of verbs, nouns, and other lexical categories. The row **R+** contains one governor G and zero or more need roles N with labels. The row **DC** contains positional constraints, e.g., P<G means that the current word (P) is in a location before its governor (G). Note that the verb *fell* has only one need role filled, N1; the missing dependencies (G, N2, and N3) point to the word itself: P=G=N2=N3. We refer the reader to the literature

	corn	futures	also	fell
C	noun	noun	adverb	verb
F	behavior: count case: common number: 3s type: common	behavior: count case: common number: non-3s type: common	behavior: normal type: common	gapp: none inverted: no mood: none number: all voice: active vtype: past
R+	G=np	G=np	G=vmod	G=vp N1=S N2=S N3=S
DC	P<G	P<G	P<G	N1<P=G=N2=N3

Table 2.2: SuperARV tags assigned to the sentence "*corn futures also fell.*" Each column (without the word) represents a SuperARV tag.

for further details on SuperARVs [105].

2.4.2.3 Parent Constituent

This tag type is a tuple of three elements: (P, NT, pos), where P is the word's POS tag NT is its immediate parent in the parse tree (non-terminal), and *pos* is the POS tag's relative position among its siblings in NT. We refer to this type of tag as *parent*. Using this convention, the example in Figure 2.2 would be tagged as shown in Table 2.3. This tagset is designed to represent constituency information and is

word	tag
the	DT-NP-start
black	JJ-NP-mid
cat	NN-NP-end
sat	VB-VP-single

Table 2.3: "the black cat sat" tagged using the *parent* tagset

word	tag
the	DT-NN
black	JJ-NN
cat	NN-VBD
sat	VBD-root

Table 2.4: "the black cat sat" tagged using the *head* tagset

similar to the tagset used in [29].

2.4.2.4 Head Tag

This tag type is a combination of the word's POS tag and the POS tag of its *governor*. The governor roles are obtained from the dependency parses such as the example depicted in Figure 2.3. The dependency parse in Figure 2.3 is generated from the constituency parse tree in Figure 2.2 by applying a set of deterministic headword percolation rules developed by Charniak for his lexicalized parser [15]. For example, the sentence in Figure 2.2 would be tagged as shown in Table 2.4. Henceforth, we will refer to this kind of tag as *head*.

word	tag
the	det,2,nsub
black	amod,1,nsubj
cat	nsubj,1,root
sat	root,0,root

Table 2.5: "the black cat sat" tagged using the *dep* tagset

2.4.2.5 Dependency Tag

We also use a tagset extracted from typed dependency trees produced from constituency trees by the Stanford Dependency parser [75].

The dependency tag that we utilize is a three-tuple (H, D, GH), where H is the type of the dependency between the word and its head, D is the distance and direction from the word to its head, bucketed as follows: $D \in \{\leq -3, -2, -1, 0, 1, 2, \geq 3\}$, with 0 indicating that the word does not have a head. GH is the type of the dependency between the word's head and the word's head head (grand-head). Thus, the example in Figure 2.3 would be tagged as shown in Table 2.5. We refer to this tagset as *dep*. It represents another way of capturing dependency information; note that unlike the *head* tagset (Section 2.4.2.4), in the *dep* tagset, we capture the relative positions of the word and its head.

It is reasonable to believe that different tasks may favor different tagsets; indeed, for languages with free word order, tagsets based on dependencies may perform better, while tasks with a very small amount of training data are likely to favor tagsets that utilize external linguistic knowledge, such as the SuperARV.

Therefore, the ability to easily utilize a variety of different tagsets is one of the main goals for the design of our language model.

2.4.2.6 Information-theoretic Comparison of Tags

Note that the tags described in the previous sections can easily be modified (augmented or reduced), thus producing new tagsets. For example, the *head* tag can be augmented with the relative position of the word's head. However, it is unclear whether the additional information in the tag would lead to a better syntactic language model. To create an effective tagset, we need a method to estimate the potential suitability of a tagset without having to train a full model (which would be computationally expensive). In this section, we propose a simple information-theoretic approach to estimate the usefulness of a tagset for syntactic language modeling.

There are two conflicting intuitions for tag design: on the one hand they should be specific enough to be helpful in the language model's task; on the other hand, they should be easy for the LM to predict. We propose the following metrics:

- To quantify how hard it is to predict a tag, we can compute the conditional entropy:

$$
\begin{aligned}
H_p(t_i|w_i) &= H_p(t_i w_i) - H_p(w_i) \\
&= -\sum_{w_i t_i} p(t_i w_i) \log p(t_i w_i) + \sum_{w_i} p(w_i) \log p(w_i) \\
&= -\sum_{w_i t_i} p(t_i w_i) \log p(t_i w_i) + \sum_{w_i t_i} p(t_i w_i) \log p(w_i) \\
&= -\sum_{w_i t_i} p(t_i w_i) \log p(t_i|w_i)
\end{aligned}
$$

- To measure how helpful a tagset is in the LM task, we can compute the reduction of the conditional cross entropy:

$$
\begin{aligned}
&H_{\tilde{p},q}(w_i|w_{i-1}) - H_{\tilde{p},q}(w_i|w_{i-1}t_{i-1}) = \\
&\quad -\sum_{w_{i-1}^i} q(w_{i-1}^i) \log \tilde{p}(w_i|w_{i-1}) + \sum_{w_{i-1}^i t_{i-1}} q(w_{i-1}^i t_{i-1}) \log \tilde{p}(w_i|w_{i-1}t_{i-1}) \\
= &\quad -\sum_{w_{i-1}^i t_{i-1}} q(w_{i-1}^i t_{i-1}) \log \frac{\tilde{p}(w_i|w_{i-1}t_{i-1})}{\tilde{p}(w_i|w_{i-1})}
\end{aligned}
$$

Note that in this case, we use conditional *cross* entropy because conditional entropy has the tendency to overfit the data as we select more and more fine-grained tags. Indeed, $H_p(w_i|w_{i-1}t_{i-1})$ can be reduced to zero if the tags are specific enough (meaning that w_i is precisely predicted by the context). This is not a problem for the former metric, because the context there, w_i, is the same for all tagsets. To eliminate zero probabilities in the cross entropy calculation, we use a smoothed distribution $\tilde{p}$ computed on the training set and the heldout distribution q. To smooth the distribution, we use *one-count* smoothing proposed by Chen [19]:

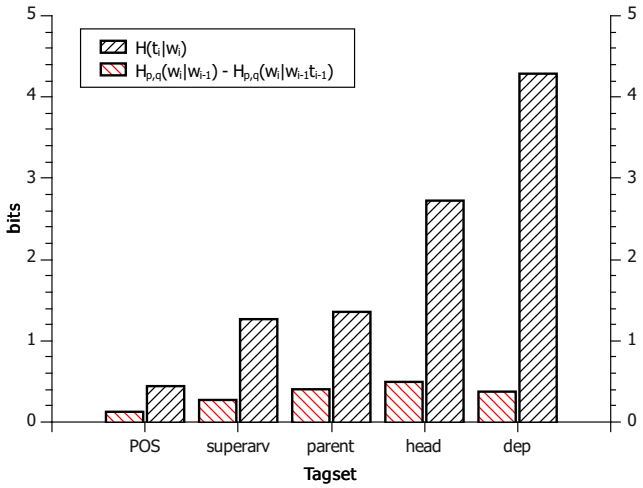


Figure 2.5: Tagset selection metrics for different tagsets

$$\tilde{p}(w_i|w_{i-1}t_{i-1}) = \frac{c(w_i w_{i-1} t_{i-1}) + \alpha \cdot p(w_i)}{\sum_{w_i} c(w_i w_{i-1} t_{i-1}) + \alpha}$$

where

$$\alpha = \gamma \left[n_1(w_{i-1}t_{i-1}) + \beta \right]; \quad n_1(w_{i-1}t_{i-1}) = |\{w_j : c(w_j w_{i-1} t_{i-1}) = 1\}|$$

i.e., $n_1(w_{i-1}t_{i-1})$ is the number of words that appear exactly once (hence the name) after the context $w_{i-1}t_{i-1}$, and γ and β are constants[12]. This smoothing algorithm is very simple to implement and yet, according to Chen [19], it performs better than most other n-gram smoothing methods.

In Figure 2.5, we present the results of these measurements obtained on training and heldout data selected from NYT '94-'95 section of the English Gigaword corpus (approximately 70M words) which was parsed using [53] and different tags

[12] Chen does not make any suggestions about how to select γ and β; we found that the values of 15 and 1, respectively, work well in this case.

were extracted from parse trees as discussed in Sections 2.4.2.1 through 2.4.2.5. POS tags, albeit easy to predict, provide very little additional information about the following word; and therefore, we would not expect them to perform very well. The *parent* tagset seems to perform better than SuperARVs – it provides 0.13 bits more information while being only 0.09 bits harder to predict based on the word. The *head* tagset is interesting: it provides 0.2 bits more information about the following word (which would correspond to 15% perplexity reduction if we had perfect tags), but on the other hand, the model is less likely to predict these tags accurately. The *dep* tagset has almost as much information about the following word as the *parent* tagset does, but it is much harder to predict, i.e., it has a lot of information irrelevant for word prediction.

This approach is only a crude estimate (it uses only unigram and bigram context), but it is potentially quite useful for designing tagsets, e.g., for a new language, because it allows us to assess the relative performance of tagsets without having to train a full model. To confirm that this approach is in fact predictive of the performance of a language model, for each tagset, we trained joint syntactic language models on the data described above using methods presented in Chapter 4.

In Figure 2.6, we present the perplexity results and the storage usage for the syntactic model utilizing various tagsets. A comparison with the metrics in Figure 2.5 shows that a larger difference in conditional cross entropy $H_{\tilde{p},q}(w_i|w_{i-1}) - H_{\tilde{p},q}(w_i|w_{i-1}t_{i-1})$ results in a better (lower) perplexity, while a larger conditional entropy $H_p(t_i|w_i)$ results in an increase in the size of the model. Note that the large amount of extraneous information in the *dep* tag does not impact the performance

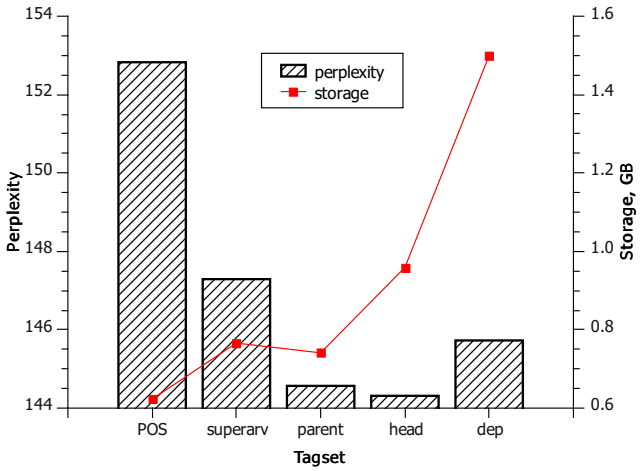


Figure 2.6: Perplexity on PTB WSJ Section 23 and the storage usage of the syntactic model utilizing different tagsets

of the model[13]. In future experiments in this thesis, we will use *head* and *parent* tagsets because they are relatively compact, perform well, and are easy to extract.

2.5 Summary

In this chapter, we have formulated the problem of language modeling and have described its purpose, and we have reviewed prior literature on language modeling. In Section 2.3, we have reviewed the methods for context clustering used in language models; we have observed that some language models, such as n-gram models, impose constraints on the way the context can be clustered, while other models, such as decision tree-based models, do not have such constraints. In Section 2.4, we have described various forms of syntactic information, particularly shallow syntactic information in the form of tags, which can be used in language modeling. We have also

[13] This is probably due to the fact that decision trees are very good at ignoring irrelevant information.

proposed and evaluated metrics that allow us to estimate the potential usefulness of a tagset for syntactic language modeling.

Chapter 3

Experimental Setups

In this chapter, we describe the experimental setups that are used throughout this thesis to compare various language models and to evaluate the impact of our research. Whenever we compare against an n-gram model, unless explicitly specified, an open vocabulary LM will be employed with interpolated modified Kneser-Ney (Section 2.2.1.3) smoothing, trained using the SRILM toolkit [102] with default cutoff parameters[1] as follows:

```
ngram-count -unk -kndiscount -interpolate ...
```

The order of n-gram models will vary from one experiment to another and will be specified explicitly for each experiment. All models compared in a single experiment will utilize the same vocabulary, unless otherwise specified.

3.1 Hub4 Setup

The Hub4 setup is used in perplexity experiments in Sections 4.7.3, 4.8.3, and in word lattice rescoring experiments in Section 6.1.

Acoustic Model. The ASR system used in this setup is based on the 2007 IBM Speech transcription system for the GALE Distillation Go/No-go Evaluation [18]. The acoustic model is a state-of-the-art discriminatively trained model that was

[1] By default, the SRILM toolkit discards singleton n-grams of order 3 and above.

trained on Broadcast News (BN) Hub4 acoustic training data as in [116]. The pronunciation vocabulary for the acoustic model contains approximately 90,000 pronunciations of 84,000 distinct words.

LM training data. Language model training data consists of the TRAIN section of Hub4 CSR 1996 [44] (130M words). The vocabulary has approximately 84,000 words (the same vocabulary that is used in the ASR system).

In order to produce tags for the syntactic models, we parsed the corpus using a self-training version of Berkeley parser [53]. To train the parser, we utilized the Broadcast News treebank from Ontonotes [107] together with the WSJ Penn Treebank (PTB) for supervised training [74], and Hub4 CSR 1996 utterances [44] for self-training.

The treebanks were preprocessed as follows: empty nodes and function labels were deleted, auxiliary verbs were replaced with AUX, and symbolic expressions with verbal forms (e.g., "$5.3" was replaced with "five point three dollars"), and punctuation and case were removed. The Hub4 self-training data was segmented into utterances, punctuation was removed, words were down-cased, and contractions and possessives were tokenized for parsing.

In order to match the tokenization of the output produced by the ASR system, after parsing and tag extraction, we recombine tokens that were split, namely, possessives and contractions, combining the tags as well.

Test data. For perplexity evaluations, the reference utterances for the NIST RT04 Broadcast News dataset, consisting of 1,709 utterances and 44,965 tokens are used. Additionally, the RT04 test set is used to compute WER in the word lattice rescoring

task. The word lattices are produced by the ASR system described above.

3.2 WSJ '94-'96 Setup

This setup is used in perplexity and n-best list rescoring experiments in Sections 4.8.3, 4.8.4, 5.1.6.1, and 5.2.1.

Acoustic Model. We utilize the same acoustic model as in the Hub4 setup (Section 3.1).

LM Training data. Models are trained on 35M words of WSJ '94-'96 from LDC2008T13. The text was converted into speech-like form to match the type of output produced by the ASR system, namely numbers and abbreviations were verbalized. We then parsed the text using a latent variable PCFG parser [53]. After parsing, we extracted tags using the *head* tagset (see Section 2.4.2.4). Then words were downcased, punctuation was removed, and contractions and possessives were joined, combining the associated tags. All models use the same vocabulary of approximately 50k words obtained by intersecting the set of most frequent words from the training data with the 84k word vocabulary of the acoustic model described in Section 3.1.

Test data. For perplexity evaluations, the PTB WSJ Section 23 [74], preprocessed in the same way as the training data, is used.

For ASR n-best list rescoring experiments, we use a test set consisting of 4,088 utterances of WSJ0 (LDC93S6A). We optimize the weights for the combination of acoustic and language model scores on a separate set comprised of 1,243

utterances from Hub2 5k closed vocabulary and the WSJ1 5k open vocabulary sets (LDC94S13A).

To produce word lattices, the same acoustic model as in the HUB4 setup (described in Section 3.1) is used; however, the language model is a trigram LM trained on 35M words of WSJ '94-'96 preprocessed as described above. One thousand best hypotheses were extracted from the lattices.

3.3 GALE MT Setup

Translation Model. In the Arabic-to-English machine translation experiments reported in Chapter 6, we use a hierarchical phrase-based decoder `CDEC` (Dyer et al. [36]), which is formally based on synchronous context-free grammar (SCFG) (Chiang [21]). `CDEC` can utilize word segmentation lattices as input. The word segmentation lattices were built using both the standard ATBv3 Arabic segmentation (ATB) [71] and an untokenized (tok) version of the data, containing only end-of-sentence periods.

The parallel data for the translation model training consists of approximately 6M sentences from the NIST OpenMT evaluation, which includes mostly LDC releases plus approximately 1.5M sentences provided to participants in the GALE program. The complete list of corpora can be found in the GALE Catalog [42]. In addition, approximately 8M sentences of automatic translations of the Arabic gigaword corpus (LDC2007T40) with ATB segmentation were used.

The corpus was then filtered based on length ratio; "waw" tokens were removed from the beginning of Arabic sentences whose English translation did not begin with

"and." Then GIZA++ was used to produce alignment, using the "grow-diag-final-and" method of Koehn et al. [66]. Finally, `<s>` and `</s>` tokens were added to the beginning and the end of each sentence, respectively, as in [37, 39].

Feature weights in the MT system (including the LM weight(s)) were optimized using minimum error rate training (MERT) algorithm [84, 36] on GALE DEV10-tune data set.

LM Training data. Language model training data consists of the English side of bitext data: Speech transcripts (BN/BC) with 3.4M words[2], Xinhua newswire (XIN) with 45M words, France Press newswire (AFP) with 94M words, and assorted Arabic newswire corpora (NW) with 210M words. Note that speech transcripts had punctuation automatically inserted using a `hidden-ngram` tool from the SRILM toolkit [102] trained on newswire and weblog bitext data as in [35].

As a baseline, we use a fourgram n-gram model trained on the combination of the four datasets (BN/BC, XIN, AFP and NW) using the SRILM toolkit (*baseline* system). The vocabulary of the model contains all observed words (approximately 300,000 unique words). Additionally, we trained four separate fourgram n-gram models on each dataset (*ngram-4* system). Each model has a vocabulary containing all words observed in its respective dataset.

In order to produce tags for syntactic language modeling, the training data was parsed using a latent variable PCFG parser [53]. Note that the parser was trained on PTB WSJ preprocessed to match the bitext data. Specifically, words were

[2] Due to the small amount of speech transcript data available, we did not separate the Broadcast News (BN) and Broadcast Conversations (BC) genres.

lowercased, numbers were verbalized, hyphenated words were split (e.g., "context-free" would be split into three tokens "context - free"), and a new tag `HH` was assigned for the hyphen token.

After parsing, tags were extracted using the *parent* tagset (see Section 2.4.2.3). We then trained four separate models on each data set (BN/BC, XIN, AFP and NW) independently. Similarly to the n-gram models, the vocabulary of each syntactic model consists of all words from the model's respective training data set. Additionally, we utilize word-type feature, a feature that determines (using regular expressions) whether the word is is a number, a punctuation, an Arabic word, or a regular English word (*joint-4* system).

Test data. For evaluation, we use on the following datasets from GALE: DEV09-dev, DEV09-tune, and DEV10-dev. Each tuning and evaluation dataset consists of four portions of different genres: Newswire (NW), Broadcast News (BN), Broadcast Conversations (BC), and Weblogs (WB), and the genre of the test set is presumed to be known. Therefore, we conduct experiments on each genre separately, e.g., in Newswire experiments, we optimize on the Newswire portion of DEV10-tune, and evaluate on the Newswire portions of DEV09-dev, DEV09-tune, and DEV10-dev.

Chapter 4

Decision Trees (DTs) in Language Modeling: Methods, Problems, and Solutions

Decision trees are often used for a classification task, which is formulated as follows: Given a tuple of *attributes* $X = (x_1, x_2, \ldots, x_n)$, assign a *class* $C \in \{c_1, c_2, \ldots, c_k\}$. In a variant of this problem, instead of winner-takes-all classification (a *hard* class assignment), *soft* classification can be used to determine the probability of each class assignment, i.e., compute the distribution $p(C|X)$. Decision trees solving this problem are often referred to as *class probability trees*. Much effort has been invested in research on various aspects of applying decision trees to this problem [82, 81]. Generative language models (Eq. 2.3 and 2.4) can also easily be formulated as a soft classification task, where C is the future word w_i from vocabulary V, and attributes $(x_1, x_2, \ldots, x_n)$ are features extracted from the context w_1^{i-1}.

In subsequent discussion, we will use the following notation:

- We denote the context of a language model as *ctx*. The context is comprised of a sequence of words $ctx \equiv w_{i-n+1}^{i-1}$ for word models $p(w_i|w_{i-n+1}^{i-1})$ and words and tags $ctx \equiv w_{i-n+1}^{i-1}t_{i-n+1}^{i-1}$ for joint models $p(w_it_i|w_{i-n+1}^{i-1}t_{i-n+1}^{i-1})$. We use $\mathcal{C}$ to denote the context space (the set of possible contexts).

- While our main focus is on the joint syntactic model $p(w_it_i|w_{i-n+1}^{i-1}t_{i-n+1}^{i-1})$, in order to decouple the effects of decision tree modeling from syntactic modeling,

we will compare results to a *word-tree* model $p(w_i|w_{i-n+1}^{i-1})$ constructed using the same decision tree-based algorithms as the joint model.

- *Training data* D is a set of tuples (ctx, w_i, c) for word-tree models (or $(ctx, w_i t_i, c)$ for joint models), where c is the number of times w_i (or $w_i t_i$) is observed following the context ctx in the training corpus.

- A *question* is a function Q that maps the context space to a finite set of values: $Q : \mathcal{C} \rightarrow \{q_1, \ldots, q_k\}$. We say that a question *partitions* $\mathcal{C}$ into k partitions. We use boolean values $\{true, false\}$ for the range of *binary* questions, and say that a context ctx *matches* the binary question q if $q(ctx) = true$.

- ϕ_k denotes a node in the decision tree (a context cluster) and D_k^t is the portion of the training data that is associated with that node. D^t is the entire training data set.

The remainder of this chapter is organized as follows: we introduce the recursive partitioning algorithm used for decision tree induction in Section 4.1. In Section 4.2, we outline the major differences between decision trees for language modeling and decision trees for a typical classification task (such as those used in ID3 [89] and similar algorithms). In Sections 4.3, 4.4, and 4.5, we describe specific steps of the decision tree induction process, review the techniques used in prior work on decision trees for language modeling and classification, and present the methods we will use in this thesis. Section 4.6 describes the method of estimating smooth probabilities from observed counts. In Sections 4.7 and 4.8, we present our contributions to

decision tree language modeling. In Section 4.7, we point out a bias in decision tree induction methods used in prior language models; and in Section 4.8, we investigate the problem of combining multiple decision trees into a single language model: we propose a novel interpolation method for decision tree models and evaluate a number of techniques for constructing forests of decision trees.

4.1 DT induction algorithm

One step look-ahead top-down greedy recursive partitioning [13], which is outlined in Algorithm 1, is the most commonly used algorithm to build a decision tree from training data. The algorithm starts with a tree consisting of one node, the root, from which the tree is grown. The entire training data D^t is associated with the root. The algorithm consists of the following steps:

- Given a decision tree node ϕ_k and the data D^t_k associated with it, the algorithm first checks if the *stopping rule* applies. The purpose of the stopping rule is to prevent the tree from *overfitting* the training data. We discuss the stopping rule in detail in Section 4.3.

- Then a set of possible questions is generated. Since it is infeasible to consider all possible partitioning questions of a dataset[1], we must select a relatively small number of questions to evaluate, as discussed in Section 4.4.

- The metric used to evaluate partitionings is often referred to as a *splitting rule*, denoted as metric $\mathcal{M}$, which is discussed in Section 4.5. Without loss of

[1] There are k^N different k-ary partitionings of a training set with N unique contexts.

generality, we choose to interpret the metric as *cost*, and therefore, minimize it. Sometimes the metric is interpreted as *gain* and maximized (e.g. Quinlan [89]), however, one interpretation can be transformed into the other essentially by changing the sign (see Section 4.7.2.2).

- Once the best question q is selected, new children nodes of ϕ_k are created and the training data set $(ctx, w_i, c) \in D_k^t$ is partitioned according to $q(ctx)$. The number of children nodes is the size of the range of the question q, thus boolean questions produce binary trees.

- The algorithm proceeds recursively with each of the newly created leaf nodes, and the algorithm terminates when each leaf triggers the stopping rule.

- Sometimes after a tree is grown, it is pruned using a *pruning rule*; however, pruning can also be applied during tree construction, thus preventing creation of branches that would be ultimately pruned, similarly to the stopping rule.

4.2 Differences between DT for Classification and Language Modeling

Although language modeling is an instance of a soft classification problem, there are a number of differences in the way the problems are typically set up that lead to substantial differences in the details of the tree growing algorithm:

- In classification problems, there is typically a large number of attributes, but each attribute tends to take on a small number of distinct values. Therefore in

Input: training data D^t

Result: decision tree $ROOT$

$ROOT \leftarrow$ new-node

`GrowTree(`$ROOT, D^t$`)`

Procedure `GrowTree(`ϕ_k, D^t_k`)`

begin

  if `stopping-rule(`ϕ_k, D^t_k`)` **then return**

  $Q \leftarrow$ `create-set-of-questions(`ϕ_k, D^t_k`)`

  $\hat{q} \leftarrow \arg\min_{q \in Q} \ \mathcal{M}(\phi_k, q, D^t_k)$

  foreach $v \in$ `range(`$\hat{q}$`)` **do**

    $\phi_{k_v} \leftarrow$ `new-child-of(`ϕ_k`)`

    $D^t_{k_v} \leftarrow \{(ctx, w_i, c) \in D^t_k : \hat{q}(ctx) = v\}$

    `GrowTree(`$\phi_{k_v}, D^t_{k_v}$`)`

end

Algorithm 1: Generic decision tree growing algorithm

classification, a question simply queries the value of an attribute, with its range being the number of observed values of that attribute (e.g., ID3 algorithm [89]). In language modeling, on the other hand, attributes are usually extracted from a small window of previous words, but each attribute can take on a large number of distinct values. If language models were to use the same strategy for question construction as classification, it would produce a decision tree similar to the n-gram model shown in Figure 2.1. More sophisticated methods of question construction are required to overcome the limitations of the n-gram type of clustering discussed in Section 2.3.1.

- Splitting rules are affected by the aforementioned differences in the type of attributes. In Section 4.7, we discuss the issue of a bias in splitting rules towards attributes with larger vocabularies, which is a significant problem for language models (particularly syntactic language models) because the disparity in vocabulary sizes of different attributes may be very large. Additionally, the large number of classes[2] results in sparse observations, which also affects the choice of the splitting rule as some approximate methods (e.g., χ^2-based) become inaccurate [41, 113].

- Decision trees for classification are often optimized for minimum size in addition to minimum error because smaller trees are easier for humans to understand. While smaller trees are always desirable due to the reductions in storage and computation, in language modeling, trees tend to be too large

[2] Recall that in language modeling the set of classes is the word vocabulary, which often contains tens of thousand to hundreds of thousand words.

(millions of nodes) to be understandable; therefore, language modeling performance is the sole criterion for decision tree induction used in this thesis. In Chapter 5, however, we describe methods used to reduce storage and computational requirements of the model to make the model practical on existing computers.

4.3 Stopping and Pruning Rules

Like most supervised machine learning approaches, decision trees have the tendency to overfit the training data. Stopping and pruning rules are used to prevent the decision tree from becoming too faithful to the training data. While the purpose of the stopping and pruning rules is similar, they are applied at different points in the tree construction algorithm:

- The stopping rule answers the question of whether to split a node *at all*. The rule is usually very simple and consists of checking that the training data associated with the node has a certain minimum number of examples.

- The pruning rule is applied after the best question for the node is chosen (or alternatively, after the tree is fully grown). Pruning rules usually involve verifying that the proposed question performs sufficiently well on a heldout set, independent of the training data.

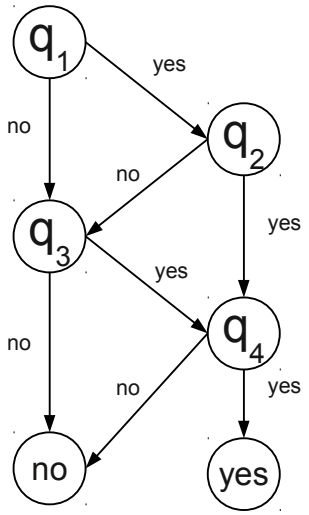


Figure 4.1: A schematic of a pylon which is the structure used in [2, 50] to construct multivariate questions from univariate questions q_1, q_2, q_3, and q_4.

4.4 Question Selection

All decision tree language models have used binary questions [2, 50, 112] because, as noted in Section 4.2, splitting on all attribute values results in an n-gram-like decision tree.

Questions can be univariate, i.e., involving a single attribute, or multivariate. Multivariate questions are constructed as Boolean combinations of univariate questions. Although arbitrary Boolean combination is possible, Bahl et al. [2] argue that it would be impractical from the computational perspective to consider all possible combinations. They proposed instead a restricted combination called a *pylon* (Figure 4.1). Below we describe the types of questions used in prior work on language modeling:

- Bahl et al. [2] used a large window of words (19 prior words) to construct multivariate questions in the form of pylons, for which elemental questions are

of the form "$w_{i-x} \in S$," where S is a subset of the vocabulary and $x \in [1, 19]$. The subset S is determined using a greedy iterative algorithm similar to the Exchange Algorithm (Algorithm 2).

- In his joint model, Heeman [50] views words as an extension of the POS tag; therefore, the identity of the word w_{i-x} is allowed to be questioned only when its tag t_{i-x} is fully resolved by previous questions. In order to construct questions about tags, Heeman [50] first creates a hierarchical classification for tags in the form of a binary tree, in which each leaf is a tag (see Figure 4.2). Each tag can be encoded as a binary path from the root to the respective leaf in that tree, while the inner nodes represent subsets of tags that share the same *prefix* of the path. The questions about tags are expressed in the form $prefix(t_{i-x}) = 11001$, which essentially means "does the tag t_{i-x} belong to the set of tags dominated by the node in the tag tree whose path from the root is 11001?"

- Xu and Jelinek [112] used the Exchange Algorithm (Algorithm 2) to construct univariate questions about words, but unlike Bahl et al. [2], they did not combine them into pylons.

Note the key difference between the approaches to question construction taken by Heeman [50] on the one side and Bahl et al. [2] and Xu and Jelinek [112] on the other. Heeman constructs questions *up front* based on the similarity between tags and words in the entire corpus, while Bahl et al. and Xu and Jelinek create questions for each individual node based on the training data associated with that

Input: Context attribute: x, training data: D_k^t
Output: Split vocabulary of x: $S, \overline{S}$

$Vocabulary \leftarrow \{x(ctx) : (ctx, w_i, c) \in D_k^t\}$
$(S, \overline{S}) \leftarrow$ `random-split(`$Vocabulary$`)`
$queue_1 \leftarrow S$
$queue_2 \leftarrow \overline{S}$
`updated(`S`)` $\leftarrow$ **False**
`updated(`$\overline{S}$`)` $\leftarrow$ **False**
entropy $\leftarrow$ `compute-entropy(`S, $\overline{S}$, D_k^t`)`
repeat
    iteration $\leftarrow$ iteration $+ 1$
    if $queue_1 \neq \emptyset$ **then**
        $w \leftarrow$ `take-first(`$queue_1$`)`
        if entropy $>$ `compute-entropy(`$S - \{w\}$, $\overline{S} + \{w\}$, D_k^t`)` **then**
            entropy $\leftarrow$ `compute-entropy(`$S - \{w\}$, $\overline{S} + \{w\}$, D_k^t`)`
            $S \leftarrow S - \{w\}$
            $\overline{S} \leftarrow \overline{S} + \{w\}$
            `updated(`S`)` $\leftarrow$ **True**
            `updated(`$\overline{S}$`)` $\leftarrow$ **True**

    else
        if `updated(`S`)` **then**
            $queue_1 \leftarrow S$
            `updated(`S`)` $\leftarrow$ **False**

    if $queue_2 \neq \emptyset$ **then**
        $w \leftarrow$ `take-first(`$queue_2$`)`
        if entropy $>$ `compute-entropy(`$S + \{w\}$, $\overline{S} - \{w\}$, D_k^t`)` **then**
            entropy $\leftarrow$ `compute-entropy(`$S + \{w\}$, $\overline{S} - \{w\}$, D_k^t`)`
            $S \leftarrow \overline{S} + \{w\}$
            $\overline{S} \leftarrow \overline{S} - \{w\}$
            `updated(`S`)` $\leftarrow$ **True**
            `updated(`$\overline{S}$`)` $\leftarrow$ **True**

    else
        if `updated(`$\overline{S}$`)` **then**
            $queue_2 \leftarrow \overline{S}$
            `updated(`$\overline{S}$`)` $\leftarrow$ **False**

until $(queue_1 = \emptyset \wedge queue_2 = \emptyset) \vee$ iteration $= MAX\_ITERATIONS$
return S, $\overline{S}$

Algorithm 2: Exchange Algorithm used to construct binary questions for multivalued attributes

node. Both approaches have strengths and weaknesses. Creation of questions for individual nodes takes advantage of the knowledge of the context. Words are often ambiguous and clustering them together introduces noise and obscures rare uses of words as we argued in Section 2.3. Answers to the questions that lead to the current node help to disambiguate the word-uses and thus create a more accurate clustering. However, only a fraction of the training data is assigned to a node, leading to an increase of sparsity. Heeman's approach partially addresses the problem of word-use ambiguity by considering words with the same surface form but different POS tags to be different words. This, however, imposes a strong limitation on the tagset because finer-grained tags would increase the sparsity. Therefore, we choose to use a mixed approach and construct questions about words and tags differently.

- Questions about words (as well as other attributes, such as morphological features) are constructed using the Exchange Algorithm (Algorithm 2), similar to Xu and Jelinek [112]. As we described above, this allows us to benefit from implicit word-use disambiguation based on the previously asked questions.

- Questions about tags are constructed using Heeman's method, namely tags are organized in a binary tree (an example for POS tags is shown in Figure 4.2), individual tags are addressed by their binary paths in that tree, and tag questions query whether a tag's path has a certain binary prefix. This form of question about tags enables an efficient decoding algorithm, which we discuss in Section 5.1.6.

The Exchange Algorithm [77] is outlined in Algorithm 2, where D_k^t is the training

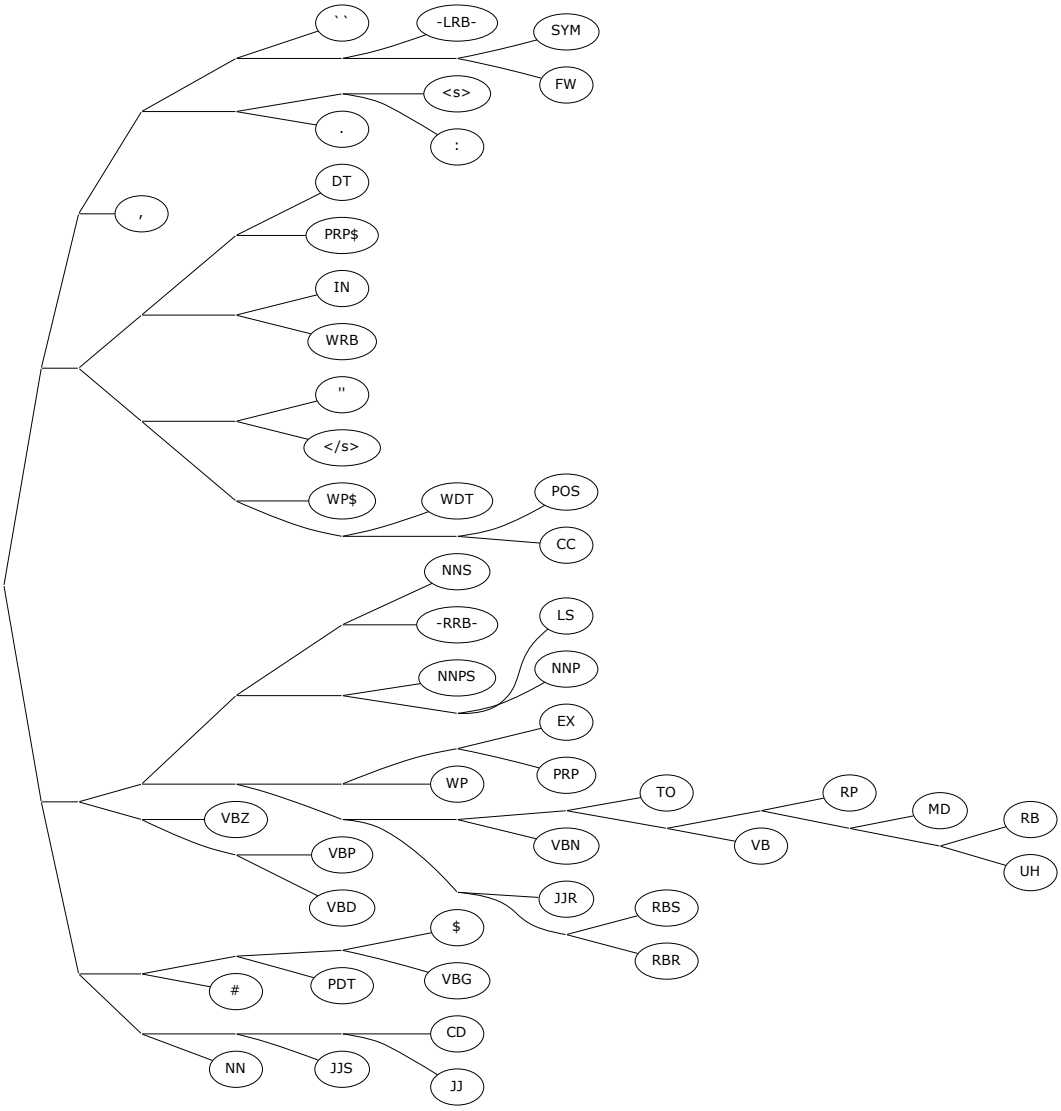


Figure 4.2: Example of a POS tag tree

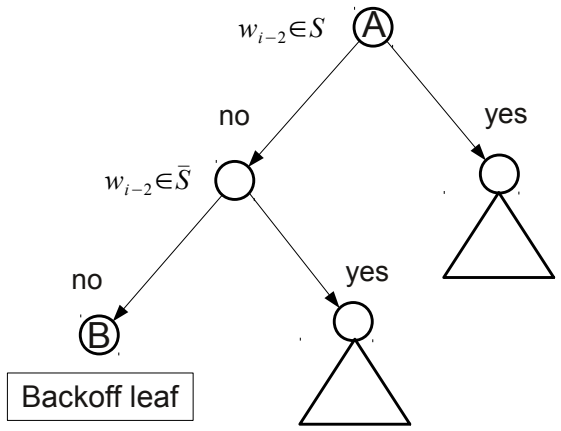


Figure 4.3: The structure for representing questions constructed using the Exchange algorithm

data associated with the current node and x denotes the attribute in the form of a function of context, i.e., when $x = w_{i-1}$, $x(ctx)$ extracts the value of w_{i-1} from the context. Note that the algorithm partitions only the values (words) *observed* in the training data D_k^t (line 1). In order to account for the unseen words, it utilizes the structure depicted in Figure 4.3, where the set S with its complement $\overline{S}$ comprise the set of words observed in the training data of the node A. The backoff leaf B is designated for the contexts with unseen words.

4.5 Splitting Rule

The purpose of the splitting rule is the selection of the decision tree $\hat{\Phi}$ that minimizes some some metric $\mathcal{M}$:

$$\hat{\Phi} = \arg\min_{\Phi} \mathcal{M}(\Phi) \tag{4.1}$$

All prior decision tree language models [2, 112, 50] have used entropy of the training

data set as the metric $\mathcal{M}$:

$$\mathcal{M}(\Phi) = -\frac{1}{C} \sum_{(ctx,w_i,c)\in D^t} c \cdot \log p(w_i|\Phi(ctx)) \tag{4.2}$$

where C is the total number of events in the training data, and $p(w_i|\Phi(ctx))$ is the maximum likelihood probability, estimated from events observed at the leaf $\Phi(ctx) \equiv \phi_k$ as follows:

$$p(w_i|\phi_k) = \frac{\sum_{(ctx,w_i',c)\in D_k^t \wedge w_i'=w_i} c}{\sum_{(ctx,w_i',c)\in D_k^t} c}$$

where D_k^t is the training data associated with the cluster ϕ_k: $\{(ctx, w_i', c) \in D^t : ctx \in \phi_k\}$. We can rewrite Eq. 4.2 in the following form:

$$\mathcal{M}(\Phi) = \sum_{k:\phi_k\in\Phi} \left(-\frac{1}{C} \sum_{(ctx,w_i,c)\in D_k^t} c \cdot \log p(w_i|\phi_k) \right) \tag{4.3}$$

Note that in this representation, the entropy contribution of each leaf ϕ_k is computed independently and therefore, the reduction of global entropy by splitting a leaf can also be calculated using only the training data associated with that leaf, which is very convenient computationally. Thus, the reduction of entropy by splitting the leaf ϕ_k into $\phi_{k'}$ and $\phi_{k''}$ is as follows:

$$\Delta\mathcal{M}(\Phi) = -\frac{1}{C}\left(\sum_{(ctx,w_i,c)\in D_k^t} c\cdot\log p(w_i|\phi_k) - \sum_{(ctx,w_i,c)\in D_{k'}^t} c\cdot\log p(w_i|\phi_{k'}) - \sum_{(ctx,w_i,c)\in D_{k''}^t} c\cdot\log p(w_i|\phi_{k''})\right) \tag{4.4}$$

Although reduction of entropy on the training data is certainly a desirable goal (as it tends to create trees that best explain the training corpus), there are some problems that arise from the properties of entropy. It has been shown [61, 111] that the reduction of entropy $\Delta\mathcal{M}(\Phi)$ is always non-negative, regardless of how the leaf ϕ_k is split. In practice, it is always positive, barring such cases as $D_{k'}^t = \emptyset$ or $D_{k''}^t = \emptyset$, or $p(w_i|\phi_{k'})$ and $p(w_i|\phi_{k''})$ being identical. As a result, even arbitrary splits lead to reductions of entropy, which is counter-intuitive and undesirable because it leads to overfitting of the decision tree to the training data. Moreover, we observe that this metric has a bias towards attributes with larger vocabularies, and in Section 4.7, we will discuss this bias at length and propose a remedy.

4.6 Probability estimation in a DT Language Model

A decision tree partitions training data and, as a result, each leaf contains only a small number of events, necessitating smoothing of probability estimates. In this section, we describe the methods that can be used to smooth probability distributions: smoothing in-tree by exploiting the hierarchy of clusters in the decision tree and smoothing by combining multiple decision trees into one model.

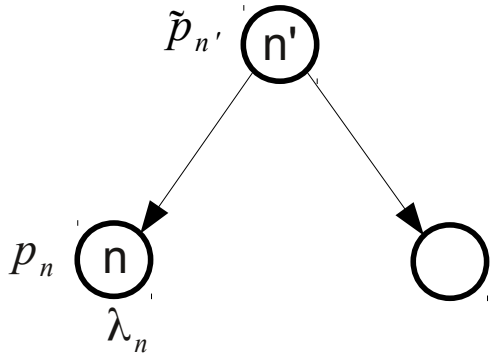


Figure 4.4: Recursive smoothing: $\tilde{p_n} = \lambda_n p_n + (1 - \lambda_n)\tilde{p_{n'}}$

4.6.1 In-tree Interpolation

In order to smooth probability distributions at the leaves of the decision tree, the following recursive formula can be used:

$$\tilde{p}_n(w_i) = \lambda_n p_n(w_i) + (1 - \lambda_n)\tilde{p}_{n'}(w_i) \tag{4.5}$$

where n' is the n-th node's parent, and $p_n(w_i)$ is the maximum likelihood distribution at node n (see Figure 4.4). The root of the tree is interpolated with the distribution $p_{unif}(w_i) = \frac{1}{|V|}$. To estimate interpolation parameters $\Lambda = \{\lambda_1, \lambda_2, \ldots\}$, we use a variant of Forward-Backward algorithm described in the rest of this section.

Note that since a decision tree is a hierarchical clustering function, a context ctx belongs to the sequence of clusters that constitutes the path from the root of the tree to a leaf. Let $l(ctx)$ denote this leaf, and $\mathcal{P}(ctx)$ denote the path from $l(ctx)$ to the root of the tree. Eq. 4.5 can be viewed as a graphical model where, given that we are at the state (node) n, a transition to the emission state will be taken with probability λ_n, and a transition to the next state n' will be taken with

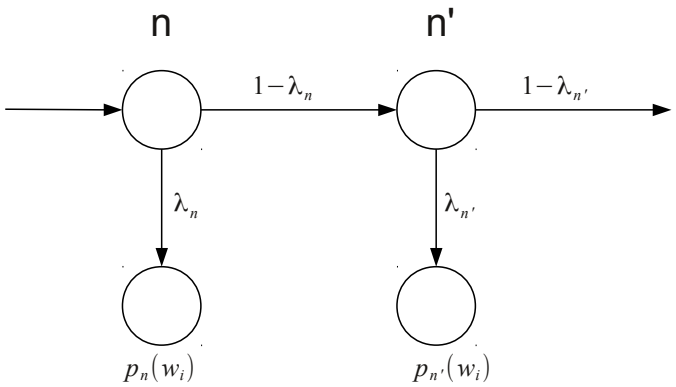


Figure 4.5: Graphical model representation for in-tree smoothing

probability $1 - \lambda_n$ (see Figure 4.5). Hence, $\alpha_n(ctx)$, the probability of reaching the state $n \in \mathcal{P}(ctx)$, can be defined recursively:

$$\begin{aligned} \alpha_{l(ctx)}(ctx) &= 1 \\ \alpha_{n'}(ctx) &= (1 - \lambda_n)\alpha_n(ctx) \end{aligned}$$

States *not* on the path $\mathcal{P}(ctx)$ cannot be reached, thus:

$$\forall_{n \notin \mathcal{P}(ctx)} \alpha_n(ctx) \equiv 0$$

The probability of generating the word w_i from the state n is:

$$Pr_\Lambda(w_i, n|ctx) = \alpha_n(ctx)\lambda_n p_n(w_i)$$

Let $\beta_n(w_i|ctx)$ be the probability of generating w_i from the state n or any of its ancestors:

$$\beta_n(w_i|ctx) = \lambda_n p_n(w_i) + (1 - \lambda_n)\beta_n(w_i|ctx)$$

Note that $\beta_{l(ctx)}(w_i|ctx)$ is the total probability of generating w_i given the context ctx:

$$\beta_{l(ctx)}(w_i|ctx) = \sum_{n \in \mathcal{P}(ctx)} Pr_\Lambda(w_i, n|ctx)$$

Let $Pr_\Lambda(n|w_i, ctx)$ be the probability that w_i is generated from the state n given the context ctx:

$$Pr_\Lambda(n|w_i, ctx) = \frac{Pr_\Lambda(w_i, n|ctx)}{\sum_{n \in \mathcal{P}(ctx)} Pr_\Lambda(w_i, n|ctx)} = \frac{\alpha_n(ctx)\lambda_n p_n(w_i)}{\beta_{l(ctx)}(w_i|ctx)}$$

The maximization step gives us the following update equation for parameters Λ:

$$\lambda'_n = \frac{\sum_{(ctx, w_i, c) \in D^h} c \cdot Pr_\Lambda(n|w_i, ctx)}{\sum_{(ctx, w_i, c) \in D^h} c \cdot \frac{\alpha_n(ctx)\beta_n(w_i|ctx)}{\beta_{l(ctx)}(w_i|ctx)}} \tag{4.6}$$

where D^h is the heldout dataset.

This method for smoothing probability in a decision tree model using a heldout set has been utilized by Bahl et al. [2], Magerman [72], and Heeman [50][3]. Rather than setting aside a separate development set for optimizing λ_n, we chose to use 4-fold cross validation and calculate the geometric mean of the resulting coefficients[4]. We chose this approach because a small development set often does not overlap with the training set for low-count nodes, leading the EM algorithm to set $\lambda_n = 0$ for

[3] Magerman [72] attributes the first publication of this algorithm to J. Lucassen's doctoral dissertation in 1983; however, we were unable to find the citation.

[4] To avoid a large number of zeros due to the product, we set a minimum for λ to be 10^{-7}.

those nodes, and because setting aside a large heldout set just for optimization is a waste of precious data. We also observed that bucketing of parameters λ_n, as was suggested by Magerman [72], did not improve the performance, and therefore, we chose not to use it.

Since backoff nodes (see Figure 4.3) have no observed events, the distributions for these nodes can be estimated using their grandparents (node A in Figure 4.3), or using a lower order model (see Section 4.6.2), or a combination of these methods. Our preliminary experiments found that none of these methods have a noticeable benefit over the others in performance, probably because only a small fraction of probability is estimated from these leaves (i.e., the probability of getting into a backoff leaf is small). Therefore, we will use the probability of the grandparent node as the method for estimating the probability of the backoff leaves due to its simplicity.

4.6.2 Multiple Decision Trees

Although a decision tree model can produce smooth probability estimates using a single decision tree by utilizing the smoothing method described in Section 4.6.1, additional decision trees may be used for more robust probability estimation. Xu [112] utilizes *lower order* decision trees (i.e., decision trees that were grown using a reduced context) as backoff models[5]. To interpolate with the lower order model, he used a discounting technique inspired by KN discounting for n-gram models (dis-

[5] Alternatively, an n-gram model could be used in his model for backoff instead of a lower order decision tree.

cussed in Section 2.2.1.3). In addition to interpolation with the lower order model, he utilized a combination of *same order* decision tree models (a forest) constructed using randomization techniques. Assuming that all models in the forest are a priori equal, he used a simple equally weighted combination of decision tree models:

$$p_{forest}(w_i|w_{i-n+1}^{i-1}) = \frac{1}{M}\sum_{m=1}^{M} p_m(w_i|w_{i-n+1}^{i-1})$$

where M is the number of models in the forest and p_m is the probability function of the m-th model in the forest.

Note that these two types of model combinations (the combination with lower order models and the combination with same order models) are substantially different, and Xu [112] utilizes different interpolation methods for these types of combination; for lower order models he uses a backoff technique borrowed from n-gram models, while same order decision trees are combined by simple weighting.

Similar to Xu [112], we also utilize lower order decision trees and use an n-gram model-inspired backoff technique; however, we use linear interpolation (see Eq. 4.7) similar to Jelinek-Mercer smoothing (see Section 2.2.1.2), because unlike Xu [112], our individual decision tree models are smoothed as described in Section 4.6.1, thus count based discounting methods, such as Kneser-Ney, are not applicable.

$$\begin{aligned}\tilde{p}_n(w_i t_i|w_{i-n+1}^{i-1} t_{i-n+1}^{i-1}) &= \lambda_n(\phi_n)\cdot p_n(w_i|\phi_n) + \\ &\quad (1-\lambda_n(\phi_n))\cdot \tilde{p}_{n-1}(w_i t_i|w_{i-n+2}^{i-1} t_{i-n+2}^{i-1})\end{aligned} \tag{4.7}$$

where $\phi_n \equiv \Phi_n(w_{i-n+1}^{i-1} t_{i-n+1}^{i-1})$ is the cluster in the decision tree Φ_n of order n, to

which the context $w_{i-n+1}^{i-1} t_{i-n+1}^{i-1}$ belongs, and $\lambda_n(\phi_n) \in [0,1]$ are assigned to each cluster and are optimized on a heldout set using EM. $p_n(w_i t_i | \phi_n)$ is the probability distribution at the cluster ϕ_n in the tree of order n. In Section 4.8, we will explore the problem of combining multiple decision tree models in depth.

4.7 Bias in Splitting Rules

As we mentioned in Section 4.5, all prior decision tree language models in the literature have utilized reduction of entropy (Eq. 4.4) as the metric for the splitting rule, i.e., the metric that is used to select the best partitioning question of the current node among the set of candidate questions. In Section 4.7.1, we show that the entropy metric has a bias towards attributes with larger vocabularies. To address this problem, in Section 4.7.2, we propose a two-step splitting rule that allows us to take advantage of the large body of research on metrics for growing decision trees for classification; this research has largely been ignored in prior work on decision tree language modeling. In Section 4.7.3, we evaluate the impact of different metrics on quality a language model.

4.7.1 Entropy Bias

The hypothesis that entropy is biased towards attributes with larger vocabularies is based on the following observation. Suppose that we have an attribute x with vocabulary $X = \{x_1, x_2, \ldots, x_{|X|}\}$ and training data D^t. Now suppose we split a value x_k of the attribute x into two new values $x_{k'}$ and $x_{k''}$, and distribute observations

(x_k, w_i) between them randomly. We then can apply the Exchange algorithm (Algorithm 2) to partition the vocabulary X into two sets X_1 and X_2. Two outcomes are possible:

1. If both $x_{k'}$ and $x_{k''}$ end up in the same set (say X_1), the resulting reduction of entropy (Eq. 4.4) will be the same as if we did not split x_k at all;

2. $x_{k'}$ and $x_{k''}$ are in different sets. Note that if this outcome is chosen, it means that splitting x_k into $x_{k'}$ and $x_{k''}$ results in a larger reduction of entropy, because otherwise the Exchange algorithm would have moved either $x_{k'}$ or $x_{k''}$, yielding outcome 1.

Thus, the attribute with split value x_k will tend to have larger reduction entropy of the split and will be preferred, despite having no additional information.

To examine this bias, we have conducted a simple simulation. We first computed counts of events (t_{i-1}, w_i) from varying amounts of tagged text (10, 20, and 40 thousand words) from the PTB WSJ [74], where t_{i-1} is the part-of-speech tag preceding the word w_i. Then we split each distinct POS tag t_{i-1} into a number of subtags t^x_{i-1}, distributing the observed events (t_{i-1}, w_i) randomly among the subtags of t_{i-1}. We then applied the Exchange algorithm (Algorithm 2) to obtain a binary split of the attribute t^x_{i-1} and measured the reduction of entropy (Eq. 4.4). By doing so, we create new context attributes with different vocabulary sizes, but with exactly the same amount of information about w_i; therefore, any significant increase in entropy reduction indicates a bias that is affected by the vocabulary size.

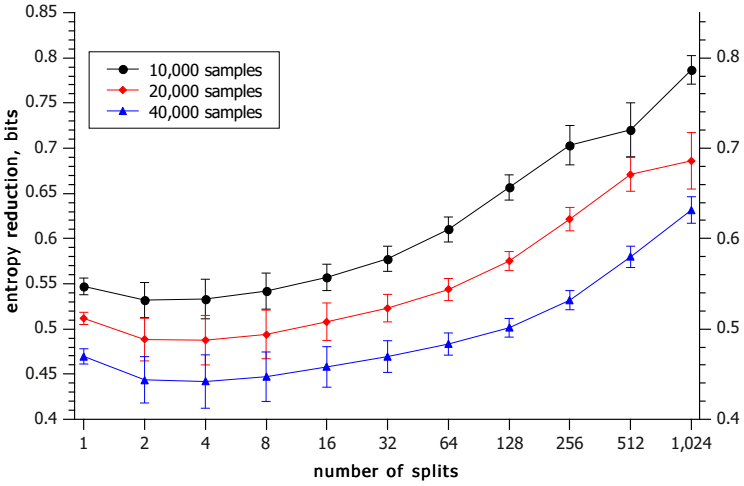


Figure 4.6: Entropy bias: the Exchange algorithm tends to achieve larger reductions in entropy for attributes with larger vocabularies (larger number of splits), despite the fact that the splits are random (i.e., no additional information is introduced by splitting).

The results of the simulation can be seen in Figure 4.6, where we varied the number of subtags for each tag (on the x-axis), from 1 to 1024, with 1 meaning that the original POS tag is used. The plot shows the average entropy reduction (y-axis) over 1000 experiments with error bars indicating one standard deviation. Note that as the number of splits increases beyond 32, the reduction in entropy grows dramatically, which confirms our hypothesis. Thus, if we treat all attributes equally and select questions solely based on their entropy reduction, there will be a bias in selection towards the attributes with larger vocabularies.

4.7.2 Attribute Selection Metrics

To address this issue, we propose to view question selection as a two-step process: (1) selection of an attribute and (2) selection of the best binary split of the attribute.

This perspective allows us to analyze attribute selection using contingency tables where the two random variables are the attribute values x and the word w_i. We also propose to use two different metrics for these steps:

- The first metric uses contingency tables to evaluate *attributes* rather than binary splits. This type of metric has been used in decision trees for classification, but not for language modeling. The purpose of this metric is to support a more *unbiased* selection of the most informative attribute.

- The second metric selects the best binary split of the attribute selected by the first metric. We will use the reduction in entropy for this metric because the lowest entropy is a reasonable goal for decision tree induction. Since the bias has been eliminated (because we compare splits of the same attribute), there is no reason not to utilize this metric.

Below we describe some of the other metrics used in the literature on decision trees for classification and discuss how they can be used in language modeling.

4.7.2.1 Time Ordered

Recall that the n-gram model is a k-ary decision tree (Figure 2.1), where the attributes are ordered by the time offset: first w_{i-1}, then w_{i-2}, etc. We can use the same heuristic for the attribute selection metric, which we call *time ordered*. Note that this metric does not distinguish different types of attributes (i.e., questions about all attributes at the same time offset will be pooled together); therefore, for the joint model, we will add a variant of this metric that orders attributes by time

and type, preferring tags over words to counter the bias due to the larger word vocabulary.

4.7.2.2 Information Gain (IG)

This metric was initially proposed by Quinlan [88][6].

$$\begin{aligned}\mathcal{M} &= H(w_i|x) = -\sum_{x,w_i} p(x,w_i)\cdot\log p(w_i|x) \\ &= -\left[\frac{1}{C}\sum_{(x,w_i,c)\in D_k^t} c\cdot\log\left(\sum_{(x',w_i',c')\in D_k^t \wedge x=x'} c'\right)\right] + \log C\end{aligned}$$

However, later several researchers found that the metric has a strong bias towards attributes with larger vocabularies [68, 89]. Recall that decision trees for classification are k-ary. Quinlan [89] observed that if, from an attribute x with k values we create the attribute x' with $k+1$ values by splitting one of values of x in two arbitrarily, then the latter would have a lower or equal $H(w_i|x)$ due to non-negative entropy reduction in Eq. 4.4, and therefore will tend to be preferred, despite the fact that it has no additional information. The bias can also be demonstrated on the following example: Suppose we have an attribute x such that it has a random unique value for each observed pair (x, w_i) in the training data. Then the Information Gain metric will always prefer this attribute because $H(w_i|x) = 0$.

Note that this bias has a different nature from the bias we describe in Section 4.7.1: The bias that was observed in [68, 89] was due to the fact that in a k-ary

[6] Originally, the metric was $H(w_i) - H(w_i|x)$, hence the name *information gain*, but we change the sign and add a constant $H(w_i)$ because, unlike Quinlan, we wish to *minimize* metrics.

decision tree, an attribute with a larger vocabulary gets split into more branches, while we showed that even when all attributes are split into the same number of branches, there is a bias towards attributes with larger vocabularies because the ability of the partitioning algorithm to better (over)fit the observed data tends to increase with the size of the attribute vocabulary. We will not use Information Gain in our experiments since its similarity to the entropy reduction metric makes it redundant.

4.7.2.3 Information Gain Ratio (IGR)

In order to mitigate the bias towards attributes with larger vocabularies, Quinlan [89] proposed to use information gain relative to the information of the attribute:

$$\mathcal{M} = 1 - \frac{H(w_i) - H(w_i|x)}{H(x)} = 1 - \frac{I(x; w_i)}{H(x)}$$

Attributes x with larger vocabularies tend to have higher entropy $H(x)$, thus this corrects the bias. We refer to this metric as Information Gain Ratio (IGR).

4.7.2.4 Distance-based Metric

A distance-based metric was proposed by de Mántaras [31].

$$\mathcal{M} = \frac{H(w_i|x) + H(x|w_i)}{H(w_i, x)}$$

Unlike the metrics discussed above, this metric satisfies the mathematical properties of a distance metric, namely:

order	n-gram	entropy	time	distance	IGR
3	250.2	246.4	246.7	251.1	**245.5**
4	234.8	237.2	235.5	245.8	**234.0**

Table 4.1: Word-tree model: Perplexity on the RT04 dataset

1. $\mathcal{M}(x, y) \geq 0$ and $\mathcal{M}(x, y) = 0$ iff $x = y$

2. $\mathcal{M}(x, y) = \mathcal{M}(y, x)$

3. $\mathcal{M}(x, y) + \mathcal{M}(y, z) \geq \mathcal{M}(x, z)$

However, it is unclear why properties such as symmetry or triangular inequality would be important for the task of question selection. Since we only measure the distance from an attribute x to w_i, and never between two attributes or from w_i to an attribute, we are not interested in the isotropy of the space. Note however that the metric was shown not to have the bias of the Information Gain metric.

4.7.3 Evaluation of Attribute Selection Metrics

To evaluate the impact of the proposed two-step splitting rule (as discussed in Section 4.7.2) on the performance of language models, we trained several different models (both joint syntactic and word-tree) using the metrics described in Section 4.7.2. The experiments were conducted on the Hub4 corpus (see Section 3.1). For joint syntactic models, we used the *parent* tagset (Section 2.4.2.3) containing 1,429 distinct tags.

In Table 4.1, we present the perplexity of the word-tree models using *entropy,*

order	n-gram	entropy	time	time-type	distance	IGR
3	250.2	239.0	239.2	235.8	240.2	**233.5**
4	234.8	233.1	229.6	**224.0**	236.5	224.3

Table 4.2: Joint model: Perplexity on the RT04 dataset

time, distance, and *IGR* metrics on the RT04 test set, along with standard n-gram models. The *entropy* column refers to the method currently used by all decision tree language models: questions are selected solely based on the entropy reduction, regardless of the attribute. When we use trigram context, all metrics perform comparably[7] and slightly better than the state-of-the-art n-gram model. Indeed, the trigram model has only two attributes: w_{i-1} and w_{i-2}, with w_{i-1} being by far more informative about w_i. It would require a really bad metric to pick the wrong attribute. When we increase the size of the context, the differences between the metrics become more pronounced, with the *IGR* metric being the only one with perplexity lower than the n-gram model's, although by a small margin. The *entropy* metric begins to suffer from the bias described in Section 4.7.2 and has 2% higher perplexity than the *IGR* metric. Although, in the word-tree model the differences between different metrics are small, they become more evident in a model with different types of attributes, such as the joint syntactic model.

We present perplexity results for the joint syntactic model in Table 4.2. The metrics *time* and *time-type* are both heuristic, and the only difference between them is that *time-type* explicitly favors tags over words, while *time* relies on entropy

[7] The exception is the *distance* metric, which despite its mathematical beauty, performed poorly in all experiments.

reduction (Eq. 4.4) to select questions about words or tags at the same time offset. The bias that we described in Section 4.7.1 causes the *time*, as well as *entropy*, metrics to perform more poorly than the *time-type* metric. *IGR* performs similarly to *time-type* in this experiment. Note that, although we were able to develop a simple heuristic metric (*time-type*) that performed well, this may not be easy to do when more attributes are added, e.g., word suffix; therefore, Information Gain Ratio (*IGR*) is a more appealing metric, and so we will use it in all subsequent experiments.

4.8 Interpolation of Multiple Decision Trees

In Section 2.2.1, we described methods used for smoothing n-gram models by backing off to lower order models. All those methods can be described by the following generic formula:

$$\begin{aligned} \tilde{p}(w_i|w_{i-n+1}^{i-1}) &= \rho(w_i|w_{i-n+1}^{i-1}) + \\ &\quad \gamma(w_{i-n+1}^{i-1}) \cdot \tilde{p}(w_i|w_{i-n+2}^{i-1}) \end{aligned} \tag{4.8}$$

where ρ is a *discounted* probability and $\gamma(w_{i-n+1}^{i-1})$ is chosen to normalize the distribution. In n-gram models, the lower order model is a necessity because the discounted distribution ρ is not smooth (all words w_i that were not seen after the context w_{i-n+1}^{i-1} in the training data would have zero probability). However, even in models where ρ *is* smooth (such as a decision tree model estimated as described in Section 4.6.1),

interpolation is beneficial because the lower order model adds robustness[8]. In the remainder of this section, we argue that, although superficially similar to n-gram models, interpolation of decision tree models, unlike n-gram models, does not satisfy a fundamental backoff property (formulated in Section 4.8.1) because of the differences in context clustering utilized by n-gram models and decision trees (see Sections 2.3.1 and 2.3.3); and therefore, interpolation of decision trees necessitates a different approach to be effective. In Section 4.8.2, we propose a generalization of the Jelinek-Mercer interpolation method for decision tree models and evaluate its effectiveness in Section 4.8.3. We then discuss and evaluate methods for selection of decision trees for combining in a *forest* model in Section 4.8.4.

4.8.1 Backoff Property

Let us rewrite the interpolation in Eq. 4.8 in a more generic way:

$$\begin{aligned} \tilde{p}(w_i|w_1^{i-1}) &= \rho_n(w_i|\Phi_n(w_1^{i-1})) + \\ &\quad \gamma(\Phi_n(w_1^{i-1})) \cdot \tilde{p}(w_i|BO_{n-1}(w_1^{i-1})) \end{aligned} \tag{4.9}$$

where ρ_n is a *discounted* distribution, Φ_n is a clustering function of order n, and $\gamma(\Phi_n(w_1^{i-1}))$ is the backoff weight chosen to normalize the distribution. BO_{n-1} is the *backoff* clustering function of order $n-1$, representing a reduction of the context size. In the case of an n-gram model, $\Phi_n(w_1^{i-1})$ is the set of word sequences where the last $n-1$ words are w_{i-n+1}^{i-1}, similarly, $BO_{n-1}(w_1^{i-1})$ is the set of sequences

[8] Reduced context implies less sparse and therefore more reliable estimation.

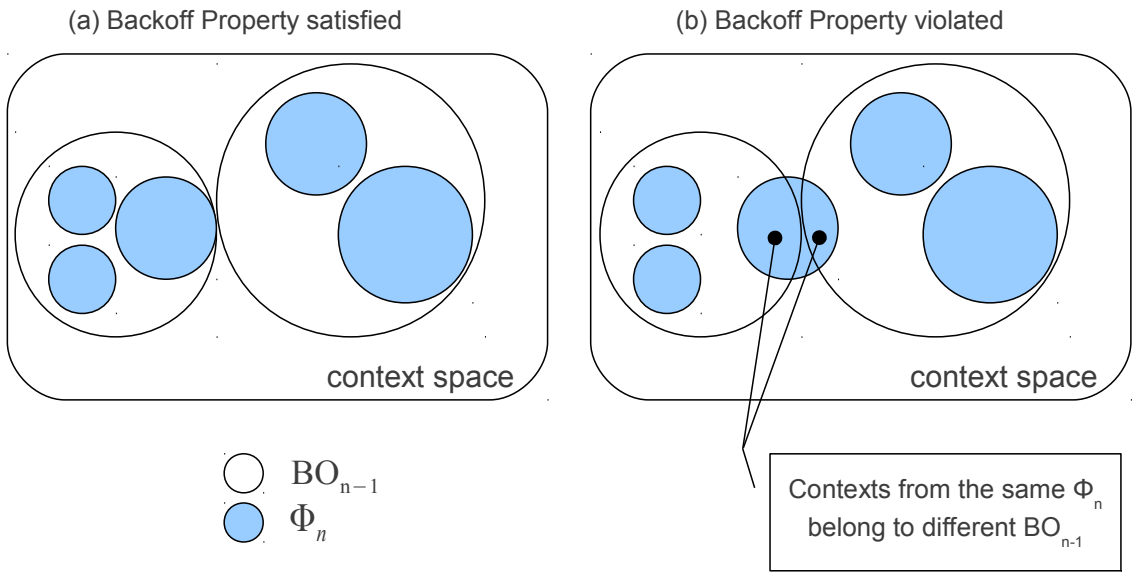


Figure 4.7: Backoff Property (Eq. 4.10)

ending with w_{i-n+2}^{i-1}. In the case of a decision tree model, the same backoff function is typically used; however, the clustering function can be arbitrary.

The intuition behind Eq. 4.9 is that the backoff context $BO_{n-1}(w_1^{i-1})$ allows for more robust (but less informed) probability estimation than the higher order context cluster $\Phi_n(w_1^{i-1})$. More precisely, any two sequences ($W = w_1^{i-1}$ and $W' = w'^{i-1}_1$) that belong to one cluster in Φ_n must belong to the same backoff cluster BO_{n-1}:

$$\forall_{W,W'} \ : \ W' \in \Phi_n(W) \Rightarrow W' \in BO_{n-1}(W) \tag{4.10}$$

thus, the sequences W and W' (and by extension all sequences in $\Phi_n(W)$) have the same backoff distribution (Figure 4.7 (a)). For n-gram models, Property 4.10 trivially holds since $BO_{n-1}(w_1^{i-1})$ and $\Phi_n(w_1^{i-1})$ are defined as sets of sequences ending with w_{i-n+2}^{i-1} and w_{i-n+1}^{i-1}, respectively, with the former clearly being a superset of the latter. However, when Φ can be arbitrary, e.g., a decision tree, that is not

necessarily so.

Let us consider what happens when we have two context sequences W and W' that belong to the same cluster $\Phi_n(W) = \Phi_n(W')$ but different backoff clusters $BO_{n-1}(W) \neq BO_{n-1}(W')$ (Figure 4.7 (b)). For example, suppose we have a higher order cluster $\Phi' = \Phi(w_{i-2} \in \{\text{on}\}, w_{i-1} \in \{\text{may,june}\})$ and two corresponding backoff clusters: $BO' = BO(w_{i-1} \in \{\text{may}\})$ and $BO'' = BO(w_{i-1} \in \{\text{june}\})$. Following *on*, the word *may* is likely to be a month rather than a modal verb, although the latter is more frequent and will dominate in BO'. Therefore, as the backoff for Φ', we have much less faith in $\tilde{p}(w_i|BO')$ than in $\tilde{p}(w_i|BO'')$ and would like a much smaller weight γ assigned to BO', but it is not possible in the backoff scheme in Eq. 4.9, because the backoff weight γ depends only on the higher order cluster Φ'; thus, we will have to settle on a compromise value of γ, resulting in suboptimal performance.

We would expect this effect to be more pronounced in higher order models because violations of Property 4.10 are less frequent in lower order models. Indeed, in a 2-gram model, the property is never violated since its backoff, the unigram, contains the entire context in one cluster. The 3-gram example above, $\Phi(w_{i-2} \in \{\text{on}\}, w_{i-1} \in \{\text{may,june}\})$, although illustrative, is not likely to occur because *may* in w_{i-1} position will likely be split from *june* very early on since it is very informative about the following word. However, in a 4-gram model, $\Phi(w_{i-3} \in \{\text{on}\}, w_{i-2} \in \{\text{may,june}\}, w_{i-1} \in \{\texttt{<unk>}\})$ is quite plausible.

Arbitrary clustering (an advantage of decision trees) leads to violation of Property 4.10, which, we argue, may lead to a degradation of performance *if* backoff

interpolation Eq. 4.9 is used. In the next section, we generalize the interpolation scheme which, as we show in Section 4.8.3, allows us to find a better solution in the face of the violation of Property 4.10.

4.8.2 Linear Interpolation

As we mentioned in Section 4.6.2, we utilize linear interpolation as the baseline in our experiment in Section 4.8.3, represented recursively in Eq. 4.7, shown below:

$$\begin{aligned}\tilde{p}_n(w_i|ctx) &= \lambda_n(\phi_n) \cdot p_n(w_i|\phi_n) + \\ &\quad (1 - \lambda_n(\phi_n)) \cdot \tilde{p}_{n-1}(w_i|ctx)\end{aligned}$$

where $\phi_n \equiv \Phi_n(ctx)$, and $\lambda_n(\phi_n) \in [0, 1]$ are assigned to each cluster and are optimized on a heldout set using EM as was discussed in Section 4.6.2. $p_n(w_i|\phi_n)$ is the probability distribution at the cluster ϕ_n in the tree of order n.

4.8.2.1 Generalized Linear Interpolation

We can unwind the recursion in Eq. 4.7 and make substitutions:

$$\begin{aligned}\lambda_n(\phi_n) &\rightarrow \hat{\lambda}_n(\phi_n) \\ (1 - \lambda_n(\phi_n)) \cdot \lambda_{n-1}(\phi_{n-1}) &\rightarrow \hat{\lambda}_{n-1}(\phi_{n-1}) \\ &\vdots\end{aligned}$$

$$\tilde{p}_n(w_i|ctx) = \sum_{m=1}^{n} \hat{\lambda}_m(\phi_m) \cdot p_m(w_i|\phi_m) \quad (4.11)$$

$$\sum_{m=1}^{n} \hat{\lambda}_m(\phi_m) = 1$$

Ideally we should be able to assign a different set of interpolation weights for every eligible combination of clusters $\phi_n, \phi_{n-1}, \ldots, \phi_1$. However, not only is the number of such combinations extremely large, but many of them will not be observed in the training data, making parameter estimation cumbersome. Therefore, we propose the following parameterization for the interpolation of decision tree models:

$$\tilde{p}_n(w_i|ctx) = \frac{\sum_{m=1}^{n} \lambda_m(\phi_m) \cdot p_m(w_i|\phi_m)}{\sum_{m=1}^{n} \lambda_m(\phi_m)} \quad (4.12)$$

Note that this parameterization has the same number of parameters as in Eq. 4.11 (one per cluster in every tree), but it has more degrees of freedom because the parameters are not constrained to sum to 1, hence the denominator.

In Eq. 4.12, there is no explicit distinction between higher order and backoff models. Indeed, it acknowledges that lower order models are *not* backoff models when Property 4.10 is not satisfied. However, it can be shown that Eq. 4.12 reduces to Eq. 4.7 if Property 4.10 holds.

Before we proceed to the proof of this statement, let us note that a cluster ϕ_m has two meanings: On the one hand, it is a node in the m-th decision tree, and on the other hand, it represents a set of contexts that belong this cluster. Given the latter meaning, ϕ_m can be used outside the scope of the m-th decision tree. Thus, for some function Ψ with a domain in the context space $\mathcal{C}$ and some set $\phi_m \subset \mathcal{C}$,

$\Psi(\phi_m)$ is defined as follows:

$$\forall_{c,c' \in \phi_m} \ \Psi(c) = \Psi(c') \equiv \Psi(\phi_m) \tag{4.13}$$

i.e., the function Ψ has a constant value on the set ϕ_m, and that value is denoted $\Psi(\phi_m)$.

Using this notation we will prove by construction that, given a set of decision trees $\{\Phi_1, \ldots, \Phi_n\}$, for any model represented by Eq. 4.12:

$$\tilde{p}_n(w_i|ctx) = \frac{\sum_{m=1}^{n} \lambda_m(\phi_m) \cdot p_m(w_i|\phi_m)}{\sum_{m=1}^{n} \lambda_m(\phi_m)} \ , \text{ where } \phi_m \equiv \Phi_m(ctx),$$

that if Property 4.10 holds (i.e., Φ_{m-1} is a backoff decision tree for Φ_m, $m = 2, \ldots, n$), then:

1. $\tilde{p}_n(w_i|ctx) = \tilde{p}_n(w_i|\phi_n)$, using definition 4.13,

   i.e., $\forall_{ctx,ctx' \in \phi_n} \ \tilde{p}_n(w_i|ctx) = \tilde{p}_n(w_i|ctx')$

2. There exists a set of parameters $\hat{\lambda}$ such that:

$$\forall_{\phi_n} : \hat{p}_n(w_i|\phi_n) = \tilde{p}_n(w_i|\phi_n)$$

   where

$$\hat{p}_n(w_i|\phi_n) = \begin{cases} \hat{\lambda}_n(\phi_n) \cdot p_n(w_i|\phi_n) + (1 - \hat{\lambda}_n(\phi_n)) \cdot \hat{p}_{n-1}(w_i|\phi_{n-1}) & \text{if } n > 1 \\ p_1(w_i|\phi_1) & \text{if } n = 1 \end{cases} \tag{4.7}$$

and

$$\hat{\lambda}_m \in [0,1],\ m = 1 \ldots n$$

That is, if Property 4.10 holds, for any model utilizing the proposed interpolation method (Eq. 4.12) of individual decision trees $p_1, \ldots, p_n$, there exists an equivalent model utilizing the recursive interpolation (Eq. 4.7) of the same individual decision tree models $p_1, \ldots, p_n$. Therefore, the proposed interpolation of Eq. 4.12 can be thought of as a *generalization* of the recursive linear interpolation that naturally extends to the case when Property 4.10 does *not* hold.

Proposition 1 ($\tilde{p}_n(w_i|ctx) = \tilde{p}_n(w_i|\phi_n)$) trivially holds by the definition of $\tilde{p}_n(w_i|ctx)$:

$$\tilde{p}_n(w_i|ctx) = \frac{\sum_{m=1}^{n} \lambda_m(\phi_m) \cdot p_m(w_i|\phi_m)}{\sum_{m=1}^{n} \lambda_m(\phi_m)} \tag{4.12}$$

and because Property 4.10 implies that $\phi_n \subset \phi_{n-1} \subset \ldots \subset \phi_1$.

Suppose that we have the parameterization of Eq. 4.12 and that Property 4.10 holds. Let us transform this parameterization into Eq. 4.7:

$$\hat{p}_n(w_i|\phi_n) = \begin{cases} \hat{\lambda}_n(\phi_n) \cdot p_n(w_i|\phi_n) + (1 - \hat{\lambda}_n(\phi_n)) \cdot \hat{p}_{n-1}(w_i|\phi_{n-1}) & \text{if } n > 1 \\ p_1(w_i|\phi_1) & \text{if } n = 1 \end{cases} \tag{4.7}$$

by induction. Let us first define a partial model $\tilde{p}_m(w_i|ctx)$ as follows:

$$\tilde{p}_m(w_i|ctx) = \frac{\sum_{k=1}^{m} \lambda_k(\phi_k) \cdot p_k(w_i|\phi_k)}{\sum_{k=1}^{m} \lambda_k(\phi_k)}$$

Note that $\tilde{p}_m(w_i|ctx) = \tilde{p}_m(w_i|\phi_m)$ for all $m = 1 \ldots n$ because $\phi_m \subset \phi_{m-1} \subset \ldots \subset \phi_1$.

For the induction base case, the lowest order distribution p_1 is not interpolated with anything, hence:

$$\begin{aligned} \tilde{p}_1(w_i|\phi_1) &\equiv \frac{\lambda_1 p_1(w_i|\phi_1)}{\lambda_1} \\ &= p_1(w_i|\phi_1) \\ &= \hat{p}_1(w_i|\phi_1) \end{aligned}$$

Now the induction step. Suppose that the distribution $\tilde{p}_{m-1}(w_i|ctx)$ can be represented recursively as $\hat{p}_{m-1}(w_i|\phi_{m-1})$:

$$\begin{aligned} \tilde{p}_{m-1}(w_i|ctx) &\equiv \frac{\sum_{k=1}^{m-1} \lambda_k(\phi_k) \cdot p_k(w_i|\phi_k)}{\sum_{k=1}^{m-1} \lambda_k(\phi_k)} \\ &= \hat{p}_{m-1}(w_i|\phi_{m-1}) \end{aligned}$$

We need to show that $\exists \hat{\lambda}_m(\phi_m) \in [0, 1]$, s.t.:

$$\begin{aligned} \tilde{p}_m(w_i|ctx) &\equiv \frac{\sum_{k=1}^{m} \lambda_k(\phi_k) \cdot p_k(w_i|\phi_k)}{\sum_{k=1}^{m} \lambda_k(\phi_k)} \\ &= \hat{\lambda}_m(\phi_m) \cdot p_m(w_i|\phi_m) + (1 - \hat{\lambda}_m(\phi_m)) \cdot \hat{p}_{m-1}(w_i|\phi_{m-1}) \\ &= \hat{p}_m(w_i|\phi_m) \end{aligned}$$

We use the following transformations:

$\tilde{p}_m(w_i|ctx) =$

$$
\begin{aligned}
&= \tilde{p}_m(w_i|\phi_m) \\
&= \frac{\sum_{k=1}^{m} \lambda_k(\phi_k) \cdot p_k(w_i|\phi_k)}{\sum_{k=1}^{m} \lambda_k(\phi_k)} \\
&= \frac{1}{\sum_{k=1}^{m} \lambda_k(\phi_k)} \left(\lambda_m(\phi_m) \cdot p_m(w_i|\phi_m) + \sum_{k=1}^{m-1} \lambda_k(\phi_k) \cdot p_k(w_i|\phi_k) \right) \\
&= \frac{1}{\sum_{k=1}^{m} \lambda_k(\phi_k)} \left(\lambda_m(\phi_m) \cdot p_m(w_i|\phi_m) + (\sum_{k=1}^{m-1} \lambda_k(\phi_k)) \cdot \frac{\sum_{k=1}^{m-1} \lambda_k(\phi_k) \cdot p_k(w_i|\phi_k)}{\sum_{k=1}^{m-1} \lambda_k(\phi_k)} \right) \\
&= \frac{1}{\sum_{k=1}^{m} \lambda_k(\phi_k)} \left(\lambda_m(\phi_m) \cdot p_m(w_i|\phi_m) + (\sum_{k=1}^{m-1} \lambda_k(\phi_k)) \cdot \tilde{p}_{m-1}(w_i|\phi_{m-1}) \right)
\end{aligned}
$$

using the induction assumption $\hat{p}_{m-1}(w_i|\phi_{m-1}) = \tilde{p}_{m-1}(w_i|\phi_{m-1})$, we have:

$$
\begin{aligned}
&= \frac{1}{\sum_{k=1}^{m} \lambda_k(\phi_k)} \left(\lambda_m(\phi_m) \cdot p_m(w_i|\phi_m) + (\sum_{k=1}^{m-1} \lambda_k(\phi_k)) \cdot \hat{p}_{m-1}(w_i|\phi_{m-1}) \right) \\
&= \frac{\lambda_m(\phi_m)}{\sum_{k=1}^{m} \lambda_k(\phi_k)} \cdot p_m(w_i|\phi_m) + \frac{\sum_{k=1}^{m-1} \lambda_k(\phi_k)}{\sum_{k=1}^{m} \lambda_k(\phi_k)} \cdot \hat{p}_{m-1}(w_i|\phi_{m-1}) \\
&= \frac{\lambda_m(\phi_m)}{\sum_{k=1}^{m} \lambda_k(\phi_k)} \cdot p_m(w_i|\phi_m) + (1 - \frac{\lambda_m(\phi_m)}{\sum_{k=1}^{m} \lambda_k(\phi_k)}) \cdot \hat{p}_{m-1}(w_i|\phi_{m-1})
\end{aligned}
$$

substituting $\hat{\lambda}_m(\phi_m) \equiv \frac{\lambda_m(\phi_m)}{\sum_{k=1}^{m} \lambda_k(\phi_k)}$, we get:

$$
\begin{aligned}
&= \hat{\lambda}_m(\phi_m) \cdot p_m(w_i|\phi_m) + (1 - \hat{\lambda}_m(\phi_m)) \cdot \hat{p}_{m-1}(w_i|\phi_{m-1}) \\
&= \hat{p}_m(w_i|\phi_m)
\end{aligned}
$$

Additionally, since $\lambda_m(\phi_m) > 0$ for $m = 1 \ldots n$:

$$\hat{\lambda}_m(\phi_m) = \frac{\lambda_m(\phi_m)}{\sum_{k=1}^{m} \lambda_k(\phi_k)} \in [0, 1]$$

After n iterations, we have:

$$\tilde{p}_n(w_i|ctx) = \hat{p}_n(w_i|\phi_n)$$

Thus, we have constructed $\tilde{p}_n(w_i|\phi_n)$ using the same recursive representation as in Eq. 4.7, proving that the standard linear interpolation is a special case of the new interpolation scheme, which occurs when the backoff Property 4.10 holds.

The denominator $\sum_{m=1}^{n} \lambda_m(\phi_m)$ in Eq. 4.12 makes it difficult to use an EM approach[9] for estimating the parameters λ; therefore, we use L-BFGS-B method [14] for optimization. In this method, we maximize the log likelihood (LL) of a heldout dataset:

$$LL = \sum_{(w_i,ctx,c)\in D^h} c \cdot \log \tilde{p}_n(w_i|ctx)$$

where D^h is the heldout dataset and $\tilde{p}_n$ is defined in Eq. 4.12. L-BFGS-B requires the gradient of this function:

$$\frac{\partial LL}{\partial \lambda_m(\phi_m)} = \frac{1}{C} \cdot \sum_{(w_i,ctx,c)\in D^h \wedge ctx \in \phi_m} c \cdot \frac{1}{\sum_{m=1}^{n} \lambda_m(\phi_m)} \cdot \left(\frac{p_m(w_i|\phi_m)}{\tilde{p}_n(w_i|ctx)} - 1 \right) \quad (4.14)$$

where C is the total number of events in D^h. Since parameters $\lambda_m(\phi_m)$ can be interpreted as the confidence in the cluster ϕ_m of the model m, we choose the bounded version of the algorithm with the boundary $\forall_{m,\phi_m}\ \lambda_m(\phi_m) > 0$.

4.8.3 Perplexity Evaluation: Recursive vs. Generalized Interpolation

In Table 4.3, we compare the effectiveness of recursive interpolation (Eq. 4.7) and the proposed generalized interpolation (Eq. 4.12) by calculating the perplexity of PTB

[9] In the maximization step of an EM approach, we need to maximize the auxiliary function $Q(\lambda'|\lambda)$, which is a lower bound for the change in log likelihood (see [7]). The denominator in Eq. 4.12 makes it difficult to construct an auxiliary function that can be maximized analytically.

Model		2-gram	3-gram	4-gram
n-gram	Jelinek-Mercer	270.2	186.5 (31.0%)	177.1 (5.0%)
	Mod KN	261.0	174.3 (33.2%)	161.7 (7.2%)
DT: Eq. 4.7 (baseline)	word-tree	257.8	168.7 (34.6%)	164.0 (2.8%)
	joint	214.3	156.8 (26.8%)	156.5 (0.2%)
DT: Eq. 4.12 (generalized)	word-tree	258.1	168.4 (34.8%)	155.7 (7.5%)
	joint	214.6	155.3 (27.6%)	147.1 (5.3%)

Table 4.3: Perplexity results on PTB WSJ section 23. Percentage numbers in parentheses denote the reduction of perplexity relative to the lower order model of the same type.

WSJ Section 23 using WSJ '94-'96 corpus as the training data (see Section 3.2). We used the same decision trees for both interpolation methods to ensure that no other factors affect the measurement. For reference, we include the standard n-gram models using Jelinek-Mercer and Modified KN smoothing techniques. As expected, the benefit of the new interpolation becomes apparent at the 4-gram order, when Property 4.10 is most frequently violated. Note that both word-tree and joint syntactic models benefit from the generalized approach.

In order to evaluate how the generalized interpolation interacts with the improvements in the tree induction algorithm proposed in Section 4.7.2, we compare decision tree models that utilize various attribute selection metrics and different interpolation methods: recursive interpolation (this method was used for the models presented in Tables 4.1 and 4.2) and the generalized interpolation (the results for this method are presented in Tables 4.4 and 4.5). Note that the improvements

order	n-gram	entropy	time	distance	IGR
3	250.2	246.5 (0.0%)	247.6 (0.4%)	247.6 (-1.4%)	**244.5** (-0.4%)
4	234.8	228.8 (-3.5%)	230.2 (-2.3%)	234.3 (-4.7%)	**227.3** (-2.9%)

Table 4.4: Word-tree model with generalized interpolation: Perplexity on the RT04 dataset. The numbers in parentheses show the change in perplexity relative to the respective models using the baseline interpolation method in Table 4.1.

order	n-gram	entropy	time	time-type	distance	IGR
3	250.2	240.4 (0.6%)	241.3 (0.9%)	234.6 (-0.5%)	236.8 (-1.4%)	**231.0** (-1.1%)
4	234.8	225.4 (-3.3%)	225.2 (-1.9%)	220.5 (-1.6%)	225.3 (-4.7%)	**216.8** (-3.3%)

Table 4.5: Joint syntactic model with generalized interpolation: Perplexity on the RT04 dataset. The numbers in parentheses show the change in perplexity relative to the respective models using the baseline interpolation method in Table 4.2.

from utilizing a better metric and a better interpolation method are additive. Also note that metrics *time* and *time-type* have smaller reductions in perplexity than the other metrics. Recall that these metrics always select attributes by the time offset, nearest first (see Section 4.7.2.1). Thus, these metrics tend to create decision trees that have time-based hierarchy, similar to n-gram models; and therefore, the decision trees produced by these metrics are less prone to violate Property 4.10.

4.8.4 Selection of Decision Trees for Forest Modeling

Note that in Eq. 4.12 individual trees do not have explicit higher-lower order relations and are treated as a collection of trees, i.e., as a forest. Naturally, to benefit from the forest model, the component trees must differ in *some* way. Different

trees can be created based on differences in the training data, differences in the tree growing algorithm, or some non-determinism in the way the trees are constructed.

Xu [111] used randomization techniques to produce a large forest of decision trees that were combined as follows:

$$p(w_i|w_{i-n+1}^{i-1}) = \frac{1}{M}\sum_{m=1}^{M} p_m(w_i|w_{i-n+1}^{i-1}) \tag{4.15}$$

where M is the number of decision trees in the forest (Xu proposed $M = 100$) and p_m is the m-th tree model[10]. Note that this type of interpolation assumes that each tree model is "equal" a priori and therefore is only appropriate when the tree models are grown in the same way (particularly, using the same order of context). The generalized interpolation (Eq. 4.12) proposed in Section 4.8.2.1 not only assigns weights to models but conditions the weights on the context, and is therefore more suitable for combination of significantly different models. Note that Eq. 4.15 is a special case of Eq. 4.12 when all parameters λ are equal.

Xu [111] showed that, although each individual tree is a fairly weak model, their combination outperforms the n-gram baseline substantially. However, we find this approach impractical for online application of any sizable model. In our experiments, fourgram trees have approximately 1.8 million leaves, and the tree structure itself (without probabilities) occupies nearly 200MB of disk space after compression. It would be infeasible to apply a model consisting of more than a handful of such trees without distributed computing of some sort. Therefore, we pose the following question: If we can afford to have only a handful of trees in the model, what is the

[10] Note that Xu [111] used lower order models to estimate p_m.

best approach for constructing those trees?

In the remainder of this section, we will discuss and evaluate different ways of building decision tree forests for language modeling and also compare combination methods of Eq. 4.12 and Eq. 4.15 (when Eq. 4.15 is applicable). As in Section 4.8.3, we evaluate different combinations by comparing the perplexity results on PTB WSJ Section 23 using WSJ '94-'96 setup (described in Section 3.2). Additionally, in order to ensure significance of our findings, we evaluate some of the best performing models on the WSJ rescoring task (described in Section 3.2).

4.8.4.1 Methods of Constructing a Random Forest

Xu [111] evaluated a variety of randomization techniques that can be used to build trees. He used a word-only model, with questions constructed using the Exchange algorithm (Algorithm 2), similar to our model. He investigated two methods of randomization: selecting the positions in the history for question construction by a Bernoulli trial[11], and random initialization of the Exchange algorithm. He evaluated the performance of his random forest model with different values for the Bernoulli trial parameter r (between 0.01 and 0.99) and found that when the Exchange algorithm was initialized randomly, the Bernoulli trial parameter did not matter; however, when the Exchange algorithm was initialized deterministically; lower values for the Bernoulli trial parameter r yielded better overall forest performance. We implemented a similar method, namely, initializing the Exchange algorithm randomly

[11] In this method, positions in the history are ignored with probability $1 - r$, where r is the Bernoulli trial parameter.

and using $r = 0.1$ for Bernoulli trials[12]. Note however that Xu [111] utilized a different smoothing method: The distribution at a leaf was computed by discounting the maximum likelihood distribution using a method similar to Kneser-Ney discounting for n-gram models and interpolating the discounted distribution with either a lower order decision tree model or an n-gram model, while we used the recursive interpolation method described in Section 4.6.1 to compute smooth distributions at leaves.

There is a key difference between the two randomization methods. Since there is no a priori preference for choosing initializations for the Exchange algorithm, by using random initializations it is *hoped* that due to the greedy nature of the algorithm, the constructed trees, while being "undegraded,"[13] will be sufficiently different so that their combination improves over an individual tree. By introducing Bernoulli trials, on the other hand, there is a choice to purposely *degrade* the quality of individual trees in the hope that additional diversity would enable their combination to compensate for the loss of quality in individual trees. Another way of introducing randomness to the tree construction without apparent degradation of individual tree quality is by varying the data, e.g., using different folds of the training data (see Section 4.1).

Let us take a closer look at the effect of different types of randomization on individual trees and their combinations. In the first set of experiments, we compare

[12] Note that because in the joint model, the question about tags are deterministic, we use a lower value of r than Xu [111] to increase randomness.

[13] Here and henceforth, by "undegraded" we mean "according to the algorithm described in Section 4.1."

the performance of a single undegraded fourgram tree[14] with forests of fourgram trees grown randomly with Bernoulli trials. Having only same-order trees in a forest allows us to apply the interpolation of Eq. 4.15 (used by Xu [111]) and compare it with the generalized interpolation method presented in Eq. 4.12. By comparing forests of different sizes with the baseline from Table 4.3, we are also able to evaluate the effect of randomization in decision tree growing and assess the importance of the lower order trees.

The results are shown in Table 4.6. Note that, while an undegraded joint syntactic tree is better than the word tree, the situation is reversed when the trees are grown randomly. This can be explained by the fact that the joint model has a much higher dimensionality of the context space, and therefore is much more sensitive to the clustering method.

As we increase the number of random trees in the forest, the perplexity decreases as expected, with the interpolation method of Eq. 4.12 showing an improvement of a few percentage points over Eq. 4.15. Note that in the case of the word-tree model, it takes four random decision trees to reach the performance of a single undegraded tree, while in the joint syntactic model, even the forest of five trees is worse than a single decision tree constructed without randomization. Finally, compare the performance of single undegraded fourgram trees in Table 4.6 with the baseline fourgram models, which are constructed with lower order trees: both word-tree and joint models in Table 4.3 have over 20% lower perplexity compared to the corresponding

[14] Since each tree has a smooth distribution based on Eq. 4.5, lower order trees are not strictly required.

	word-tree		joint	
	Eq. 4.15	Eq. 4.12	Eq. 4.15	Eq. 4.12
1 × undgr	204.9		189.1	
1 × rnd	250.2		289.9	
2 × rnd	229.5	221.5	244.6	240.9
3 × rnd	227.5	214.5	226.2	220.0
4 × rnd	219.5	205.0	219.5	212.2
5 × rnd	200.9	184.1	216.5	209.0
baseline	N/A	155.7	N/A	147.1

Table 4.6: Perplexity numbers obtained using fourgram trees only. Note that "undgr" and "rnd" denote undegraded and randomly grown trees with Bernoulli trials, respectively, and the number indicates the number of trees in the forest. "Baseline" refers to the fourgram models with lower order trees (from Table 4.3, Eq. 4.12).

models consisting of a single fourgram tree.

In Table 4.7, we evaluate forests of fourgram trees produced using randomizations without degrading the tree construction algorithm. That is, we use random initializations of the Exchange algorithm and, additionally, variations in the training data folding. All forests in this table use the interpolation method of Eq. 4.12. Note that, while these perplexity numbers are substantially better than trees produced with Bernoulli trials in Table 4.6, they are still significantly worse than the baseline model from Table 4.3.

These results suggest that, while it is beneficial to combine *different* deci-

	word-tree		joint	
# trees	Exchange	+data	Exchange	+data
1	204.9		189.1	
2	185.9	186.5	174.5	173.7
3	179.5	179.9	168.8	167.2
4	176.2	176.4	165.1	164.0
5	173.7	172.0	163.0	162.0
baseline	155.7		147.1	

Table 4.7: Perplexity numbers obtained using fourgram trees using random initialization of the Exchange algorithm and, additionally, variations in training data folds (+data columns). "Baseline" refers to the fourgram models with lower order trees (from Table 4.3). All models use the interpolation method of Eq. 4.12.

sion trees, we should introduce differences to the tree construction process without degrading the trees when introducing randomness, especially for joint models. In addition, lower order trees seem to play an important role for high quality model combination.

4.8.4.2 Context-Restricted Forest

As we have mentioned above, a forest with a combination of higher and lower order decision trees produces much better results than a forest with same-order trees. A lower order decision tree is grown from a lower order context space, i.e., the context space where we purposely ignore some attributes. Note that in this case, rather

than *randomly* ignoring contexts via Bernoulli trials at every node in the decision tree, we discard some context attributes upfront in a principled manner (i.e., most distant context) and then grow the decision tree without degradation. Since the joint model, having more context attributes, affords a larger variety of *different* contexts, we use this model in the remaining experiments.

In Table 4.8, we present the perplexity numbers for our standard model with additional trees. We denote context-restricted trees by their Markovian orders (words w and tags t independently), so 3w2t indicates a decision tree implementing the probability function: $p(w_i t_i | w_{i-1} w_{i-2} t_{i-1})$. The fourgram joint model presented in Table 4.3 has four trees and is labeled with the formula "1w1t + 2w2t + 3w3t + 4w4t" in Table 4.8. The randomly grown trees (denoted "bernoulli-rnd") are grown utilizing the full context 4w4t using the methods described in Section 4.8.4.1. All models utilize the generalized interpolation method described in Section 4.8.2.1.

As can be seen in Table 4.8, adding undegraded trees consistently improves the performance of an already strong baseline, while adding random trees only increases the perplexity because their quality is worse than undegraded trees'. Trees produced by data randomization (denoted "data-rnd") also improve the performance of the model; however, the improvement is not greater than from addition of lower order trees, which are considerably smaller in size.

In Table 4.9, we present WER results along with the corresponding perplexity numbers from Tables 4.3 and 4.8. The interpolation method of Eq. 4.12 substantially improves performance over the interpolation of Eq. 4.7, reducing WER by 0.25% absolute ($p < 10^{-5}$). Adding four trees utilizing context restricted in different ways

Model	size	PPL
1w1t + 2w2t + 3w3t + 4w4t (*)	294MB	147.1
(*) + 4w3t + 3w2t	579MB	143.5
(*) + 4w3t + 3w4t	587MB	144.9
(*) + 4w3t + 3w4t + 3w2t + 2w3t	699MB	**140.7**
(*) + 1 × bernoulli-rnd	464MB	149.7
(*) + 2 × bernoulli-rnd	632MB	150.4
(*) + 3 × bernoulli-rnd	804MB	151.1
(*) + 1 × data-rnd	484MB	147.0
(*) + 2 × data-rnd	673MB	145.0
(*) + 3 × data-rnd	864MB	145.2

Table 4.8: Perplexity results using the standard joint syntactic model with additional trees. "Bernoulli-rnd" and "data-rnd" indicate fourgram trees randomized using Bernoulli trials and varying training data, respectively. The second column shows the combined size of decision trees in the forest.

further reduces WER by 0.12%, which is also a statistically significant ($p < 0.025$) improvement over the baseline models labeled (*). Altogether, the improvements over the n-gram baseline add up to 0.61% absolute (8% relative) WER reduction.

4.9 Contributions

- In Section 4.7, we have discovered a bias towards attributes with larger vocabulary in the splitting rule used in prior decision tree-based language models. In order to address this problem, we have introduced a novel two-step splitting

Model	PPL	WER
n-gram	161.7	7.81%
1w1t + 2w2t + 3w3t + 4w4t (Eq. 4.7)	156.5	7.57%
1w1t + 2w2t + 3w3t + 4w4t (*)	147.1	7.32%
(*) + 4w3t + 3w4t + 3w2t + 2w3t	140.7	**7.20%**

Table 4.9: Perplexity and WER results. Note that the last two rows are joint syntactic models using the interpolation method of Eq. 4.12.

rule and have demonstrated its efficiency.

- In Section 4.8.1, we have observed that in the backoff interpolation scheme, there is an implied relation between context clustering in the higher- and the lower-order models. We have formulated this relation as the Backoff Property and have pointed out that this property is not typically satisfied in decision tree-based models, resulting in a poor performance of backoff interpolation methods in such models.

- We have proposed a generalization of linear interpolation for the models where the Backoff Property is not satisfied in Section 4.8.2; and we have demonstrated that the proposed generalized interpolation method significantly outperforms the previously used interpolation method in Section 4.8.3.

- In Section 4.8.4, we have investigated a number of techniques for construction of small forests of decision trees. We have found that the combinations of decision trees constructed using context restricted in different ways yield significantly better results than the combinations of trees in which the variability

is introduced by Bernoulli trials or by varying inherently arbitrary parameters, such as initialization of greedy algorithms.

4.10 Summary

In this chapter, we have described various aspects of the decision tree construction process, which in the past was primarily investigated from the perspective of classification task. We have outlined the major differences between the tasks of classification and language modeling, and we have pointed out how these differences affect decision tree construction. We have reviewed prior approaches to decision tree language modeling and have observed that they suffer from bias in the tree induction. We have proposed to address the bias by utilizing methods that had been investigated in the literature on decision trees for classification, and our evaluations have demonstrated that the proposed method for correcting the bias consistently results in reduced perplexity of the language model.

We have also shown that combinations of higher- and lower-order decision tree models require approaches different from the methods developed for n-gram models, because the context clustering of decision tree-based models results in a violation of the backoff Property 4.10, upon which the n-gram model interpolation methods were implicitly built. We have proposed a generalization of linear interpolation suitable for decision tree models and have demonstrated its effectiveness. Additionally, we have evaluated a variety of techniques for induction of forests of decision trees that are effective without requiring a very large number of trees. We have concluded

that restricting context for decision tree induction in different ways produces better forests than randomization techniques.

Chapter 5

Making a Syntactic Decision Tree-based LM Tractable

The complexity of the joint syntactic model described in the previous chapter poses a number of engineering problems that must be addressed for the model to be able to utilize practically useful amounts of data and vocabularies. In Section 5.1, we describe the algorithms and data structures that are crucial for scalability of the model. In Section 5.2, we introduce and evaluate an n-gram approximation of the syntactic model. This approximation extends the applicability of the model to on-line applications, such as machine translation, and has enabled us to integrate our model into the `CDEC` decoder [36] for an experiment reported in Chapter 6.

5.1 Computational Considerations

Large-scale modeling implies that the model cannot rely on being able to load simultaneously all necessary data into the main memory during training or decoding process (or both). Instead, all memory-intensive algorithms of the model must either use a data pipelining approach, similar to the MapReduce framework [32], or be adjusted in such a way that most of data access is localized to a relatively small *working set* that can be kept in the main memory.

In the remainder of this section, we describe the approaches we have taken to ensure our model scales well for large amounts of data. In Sections 5.1.1 and 5.1.2, we

revisit the decision tree induction algorithm and in-tree interpolation, respectively, and describe the modifications to generic algorithms that we have implemented in order to improve scalability of the model. In Sections 5.1.3 through 5.1.6, we describe various various aspects of the model pertaining to the decoding process, including probability representation, on-disk format, and the decoding algorithm.

5.1.1 Tree Construction

In Algorithm 3, we introduce a modified version of tree induction algorithm (Algorithm 1 in Section 4.1). Unlike the generic tree induction algorithm (Algorithm 1), which grows the tree depth-first, this implementation uses a breadth-first approach by maintaining a list of active leaves and growing the tree layer-by-layer. This approach allows us to use a sequential data access pattern, which is much faster than random access; and since partitioning of a leaf depends only on the data associated with the leaf, this approach does not affect the quality of the produced decision tree. Additionally, this organization of data lends itself to a straightforward implementation within massively parallel frameworks, such as Hadoop [47]. Some of the data structures we employ for collecting statistics are similar to the structures used in prior literature on scalable decision tree induction, e.g., the SPRINT system [97]. The idea of decision trees from streaming data has also been investigated in the literature, e.g., by Jin and Agrawal [59]; however, they attack a much harder problem: each data record is allowed to be read only once, which imposes restrictions on the algorithm unnecessary in our model.

Input: training data D^t

Output: binary decision tree $ROOT$

ActiveLeaves $\leftarrow \{(\text{ROOT}, D^t)\}$

repeat

  NewLeaves $\leftarrow \emptyset$

  `/* collect statistics and select the best partitioning question for each` ϕ_k `*/`

  forall $(\phi_k, D_k^t) \in$ ActiveLeaves **do**

    $Q \leftarrow$ `create-tag-questions(`ϕ_k, D_k^t`)`

    Stats $\leftarrow$ `collect-statistics(`D_k^t, Q`)`

    if `stopping-rule(`ϕ_k, D_k^t, Stats`)` **then** `continue`

    $Q \leftarrow Q \cup$ `create-word-questions(`ϕ_k, D_k^t, Stats`)`

    `question(`ϕ_k`)` $\leftarrow \arg\min_{q \in Q} \mathcal{M}(\phi_k, q, D_k^t, \text{Stats})$

  `/* grow the tree by splitting leaves and partition the training data */`

  forall $(\phi_k, D_k^t) \in$ ActiveLeaves **do**

    if *not* `question(`ϕ_k, D_k^t`)` **then** `continue`

    $\phi_{k'} \leftarrow$ `new-child-of(`ϕ_k`)`

    $\phi_{k''} \leftarrow$ `new-child-of(`ϕ_k`)`

    $((\phi_{k'}, D_{k'}^t), (\phi_k{}'', D_{k''}^t)) \leftarrow$ `partition-data(`$\hat{q}$, (ϕ_k, D_k^t)`)`

    NewLeaves $\leftarrow$ NewLeaves $\cup \{(\phi_{k'}, D_{k'}^t), (\phi_{k''}, D_{k''}^t)\}$

  ActiveLeaves $\leftarrow$ NewLeaves

until ActiveLeaves $= \emptyset$

Algorithm 3: Pipelined Decision Tree Growing Algorithm

Note that to grow the tree by one layer, Algorithm 3 makes two passes through the training data, see the loops in lines 4-9 and 11-16. In the first pass (lines 4-9), we select the best question for the each leaf ϕ_k (or decide that the leaf is final and not to be split further). In the second pass (lines 11-16), we partition the data for the next layer of leaves. These two passes cannot be easily combined, because each data set D_k^t (the set of training data associated with the node ϕ_k) is assumed to be too large to fit into memory, and therefore, is represented as a stream of data items (ctx, w_i, c). Thus, line 6 iterates over the training data D_k^t associated with ϕ_k and collects statistics for:

- Attribute Selection Metrics (see Section 4.7.2). Computation of metrics requires a contingency table for a context attribute x and the future word w_i. A compact representation of a sparse table requires space proportional to the number non-zero elements, i.e., the number of unique pairs (x, w_i) observed in the training data D_k^t. Words tend to have the largest vocabulary among all attributes (as discussed in Section 4.7); therefore, the amount of space is dominated by word attributes, which require $O(B(D_k^t))$, where $B(D_k^t)$ is the number of bigram types in D_k^t. The number of context attributes is proportional to the order of the model; thus, the overall space complexity is: $O(n \cdot B(D_k^t))$, where n is the order of the model.

- The Exchange algorithm (Algorithm 2 in Section 4.1) utilizes the same type of statistics as the Attribute Selection Metrics; therefore, it does not add to overall space complexity.

- Tag questions evaluation using Eq. 4.4:

$$\Delta\mathcal{M}(\Phi) = -\frac{1}{C}\left(\sum_{(ctx,w_i,c)\in D_k^t} c\cdot\log p(w_i|\phi_k) - \sum_{(ctx,w_i,c)\in D_{k'}^t} c\cdot\log p(w_i|\phi_{k'}) - \sum_{(ctx,w_i,c)\in D_{k''}^t} c\cdot\log p(w_i|\phi_{k''})\right)$$

  The space required for calculation of this equation is $O(|V|)$, where V is the vocabulary for w_i. Note that questions about words do not exist at this point (line 6) yet (they are produced later using the Exchange algorithm at line 8), however, the statistics collected for the Exchange algorithm contains sufficient information for computation of Eq. 4.4.

Thus, the overall space requirement for the pipelined decision tree construction algorithm (Algorithm 3) is as follows:

$$O(n\cdot B(D_k^t) + |Q||V|)$$

where Q is the set of possible questions.

5.1.2 In-tree Interpolation

In-tree interpolation described in Section 4.6.1 is also a memory intensive process as it involves observed distributions for every cluster in the tree (including the inner nodes). Recall that the smoothed probability distribution $\tilde{p}_n(w_i)$ at a leaf n is estimated as the observed distribution at the leaf $p_n(w_i)$ recursively interpolated with the smoothed distribution at the leaf's parent node n':

$$\tilde{p}_n(w_i) = \lambda_n p_n(w_i) + (1 - \lambda_n)\tilde{p}_{n'}(w_i) \tag{4.5}$$

The λ parameters are estimated using an EM approach similar to Forward-Backward algorithm, with the following EM update equation:

$$\lambda'_n = \frac{\sum_{(ctx,w_i,c)\in D^h} c \cdot Pr_\Lambda(n|w_i, ctx)}{\sum_{(ctx,w_i,c)\in D^h} c \cdot \frac{\alpha_n(ctx)\beta_n(w_i|ctx)}{\beta_{l(ctx)}(w_i|ctx)}} \tag{4.6}$$

where $Pr_\Lambda(n|w_i, ctx)$ is the probability of generating w_i from the node n given context ctx, $\alpha_n(ctx)$ is the probability of reaching the node n given context ctx climbing up from the leaf $l(ctx)$, $\beta_n(w_i|ctx)$ is the probability of generating w_i from the node n or any of its ancestors, and D^h is the heldout data set.

A close examination of the update equation Eq. 4.6 shows that any given tuple (ctx, w_i, c) of the heldout data D^h contributes only to the update of the λ parameters of the nodes n to which the context ctx belongs, i.e., $n \in \mathcal{P}(ctx)$, because $\forall_{n\notin\mathcal{P}(ctx)}\alpha_n(ctx) \equiv 0$, and therefore $\forall_{n\notin\mathcal{P}(ctx)} \Pr_\Lambda(n|w_i, ctx) = 0$. Hence, for any given context ctx in the heldout data D^h, only the observed distributions for the clusters that belong to $\mathcal{P}(ctx)$ must be loaded into the main memory, which is tractable even for large decision trees, since the memory usage is $O(|V| \cdot |\mathcal{P}(ctx)|)$. Per-cluster distributions $p_n(w_i)$ are stored in a database[1] and loaded on demand. Aggregation of the distributions $p_n(w_i)$ from the training data is performed using a layer-by-layer approach (Algorithm 4), similar to the pipelined decision tree induction algorithm (Algorithm 3).

[1] We utilized Berkeley DB Java Edition [85].

Additionally, the heldout data $(ctx, w_i, c) \in D^h$ is sorted by the cluster $\Phi(ctx)$. This allows us to minimize database access as we iterate through D^h computing the EM update Eq. 4.6.

Input: training data D^t

Output: binary decision tree *ROOT*

```
1 ActiveLeaves ← {(ROOT, D^t )}
2 repeat
3     NewLeaves ← ∅
4     forall (φ_k, D^t_k) ∈ ActiveLeaves do
5         p_k(w_i) ← collect-statistics(D^t_k)
6         store(p_k(w_i))
7         NewLeaves ← NewLeaves ∪ new-child-of(φ_k);
8     ActiveLeaves ← NewLeaves
9 until ActiveLeaves = ∅
```

Algorithm 4: Pipelined Data Aggregation Algorithm

5.1.3 Forest Interpolation

In Section 4.8.2.1, we described the interpolation method for combining multiple decision trees:

$$\tilde{p}_n(w_i|ctx) = \frac{\sum_{m=1}^{n} \lambda_m(\phi_m) \cdot p_m(w_i|\phi_m)}{\sum_{m=1}^{n} \lambda_m(\phi_m)} \tag{4.12}$$

where ϕ_m is the cluster in the m-th decision tree to which the context ctx belongs, $p_m(w_i|\phi_m)$ is the smoothed distribution of the model m for that cluster, and $\lambda_m(\phi_m)$ is the weight associated with the cluster ϕ_m in the model m. The $\lambda_m(\phi_m)$ parameters for this interpolation are estimated using L-BFGS-B approach, with the gradient given in Eq. 4.14:

$$\frac{\partial LL}{\partial \lambda_m(\phi_m)} = \frac{1}{C} \cdot \sum_{(w_i,ctx,c)\in D^h \wedge ctx \in \phi_m} c \cdot \frac{1}{\sum_{m=1}^{n} \lambda_m(\phi_m)} \cdot \left(\frac{p_m(w_i|\phi_m)}{\tilde{p}_n(w_i|ctx)} - 1 \right) \quad (4.14)$$

Note that this method assumes that, in every decision tree m, any context ctx belongs to exactly one cluster ϕ_m, which corresponds to a leaf in m. Recall, however, that a decision tree is a hierarchy clusters. Thus, the context ctx belongs to a sequence of clusters from a leaf to the root of the tree $\mathcal{P}(ctx)$; and any of the clusters from the sequence $\mathcal{P}(ctx)$ can be used instead of the leaf[2] in Eq. 4.12. We use an internal node for the cluster ϕ_m instead of a leaf in two cases:

1. When the leaf node for the context ctx is a backoff node, we use the backoff node's grandparent node for ϕ_m (see node A in Figure 4.3).

2. The leaf corresponding to the context ctx is resolved by applying decision tree questions to the context ctx starting from the root; we may choose to stop this process at an internal node and use the internal node as the cluster for ctx in order to reduce the amount of computation. The general idea behind this optimization is that in HMM decoding, the states that have a relatively small

[2] This would have the same effect as pruning the tree.

probability of reaching them will have a small effect on the overall probability, and therefore, a less accurate emission probability (afforded by a coarser internal node cluster) will have little impact on the quality of the model. We will discuss this optimization in detail in Section 5.1.6.1.

Due to the optimization described above, for any given context ctx, in the set of clusters $\phi_1, \ldots, \phi_n$ in Eq. 4.12, any number of the clusters can be internal nodes in their respective decision trees. Thus, for a given context ctx, the number of possible combinations of clusters $\phi_1, \ldots, \phi_n$ grows exponentially with the number of decision trees in the forest, which makes the estimation of the λ parameters difficult.

We solve this problem by estimating the λ parameters for the leaves and for the internal nodes separately, using a two-step procedure as follows:

1. In the first step, we optimize parameters $\lambda_m(\phi_m)$ assigned to the leaves as described in Eq. 4.14.

2. In the second step, for every decision tree m in the forest, and for every *internal* node ϕ_m in m, we optimize the combination of ϕ_m with the *leaf* nodes of other trees, while keeping the weights assigned to the leaf nodes constant.

While we do not optimize the λ parameters for every possible combination of internal nodes[3], this technique produces good estimations for the parameters, and the experiments in Section 5.1.6.1 show no noticeable degradation in performance due to the use of internal nodes compared to using leaves only.

[3] Note that we do not optimize every possible combination of leaves either; we only can optimize the combinations observed in the heldout data.

5.1.4 Probability Representation

In-tree interpolation (Section 4.6.1) produces a smooth distribution for each cluster k: $\tilde{p}_k(w_i)$ in the case of the word-tree model, and $\tilde{p}_k(w_i t_i)$ in the case of the joint model. However, storing the smoothed distributions would require $O(|\Phi| \cdot |V|)$ space, where $|\Phi|$ is the number of clusters in the decision tree Φ and V is the vocabulary for w_i. This is intractable even for moderately sized word-tree models with $|\Phi| \approx 10^6$ and $|V| \approx 50,000$. Therefore, instead of storing pre-computed smooth distributions $\tilde{p}_k(w_i)$, we store observed distributions $p_k(w_i)$ for all clusters (including inner nodes) along with interpolation weights λ_k and compute smooth distributions on demand.

In the joint model $p(w_i t_i | w_{i-n+1}^{i-1} t_{i-n+1}^{i-1})$, the computation of $p(w_i | w_1^{i-1})$ involves many leaves to which the contexts $w_{i-n+1}^{i-1} t_{i-n+1}^{i-1}$ belong (for a given w_{i-n+1}^{i-1} and varying t_{i-n+1}^{i-1}, as we sum over the tag sequences as shown in Eq. 2.6):

$$p(w_i | w_1^{i-1}) = \frac{p(w_1^i)}{p(w_1^{i-1})} = \frac{\sum_{t_1 \ldots t_i} \prod_{j=1}^{i} p(w_j t_j | w_1^{j-1} t_1^{j-1})}{\sum_{t_1 \ldots t_{i-1}} \prod_{j=1}^{i-1} p(w_j t_j | w_1^{j-1} t_1^{j-1})} \tag{2.6}$$

Thus, we require smoothed distributions $p(w_i t_i | \phi_k)$ for multiple leaves ϕ_k for the computation of Eq. 2.6. Many of these leaves share parts of their paths to the root, e.g., leaves A and B in Figure 5.1 share the path to the root starting from node C. Therefore, to eliminate duplicate computations, we cache smoothed distributions computed for internal nodes (C and D in Figure 5.1).

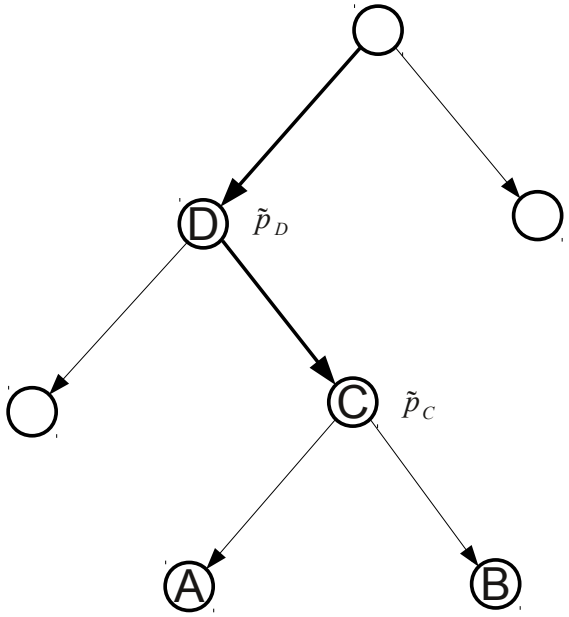


Figure 5.1: Caching interpolated distributions

5.1.5 On-disk Format

Large-scale language modeling demands that the model may not be loaded entirely into the memory, and since disk access is very expensive compared to main memory, the on-disk format plays an important role in overall model performance. There are several goals that the design of the on-disk format must achieve:

- Compactness. Reduction of the on-disk footprint reduces disk IO directly; and indirectly, it allows larger portions of the model to be cached by the operating system, further reducing disk IO.

- Favor sequential access pattern over random. In rotating hard drives, the slowest operation is the track seek. Sequential access reduces the number of seeks, resulting in dramatically faster transfer rates.

- Reduce unnecessary transfers. Due to the fact that hard drives are *block* devices (i.e., all operations are performed using fixed-size blocks), reading even one byte from the disk incurs the transfer of at least one block (typically several kilobytes) and often more, depending on the hard drive's and the operating system's read-ahead settings. Therefore, the on-disk format should keep related data close together.

In a decision tree forest model, we represent models of individual decision trees independently; therefore, in this section, we describe the format of one decision tree model that utilize the above goals in its design. A decision tree model consists of a decision tree Φ and observed probabilities $p_k(w_i t_i)$, and the interpolation weights λ_k, as described in Section 5.1.4. Even large decision trees with millions of leaves are small enough to be loaded into memory entirely; therefore, we only aim at optimizing the format of observed distributions $p_k(w_i t_i)$. Our design of the on-disk format is based on a simple yet ubiquitous phenomenon in natural language processing, i.e., "*the majority of events are rare*" [73]. Below are the specific consequences of this phenomenon that directly influenced the design of the on-disk format for our model:

1. In large vocabulary language modeling, only a fraction of the vocabulary is used in any given task due to the Zipfian nature of word frequencies: the majority of words are very rare.

2. The majority of observed *counts* of events $w_i t_i$ in clusters are small numbers.

3. For any given word w_i, many clusters contain the same number of events $w_i t_i$ for all t_i, i.e., for a given word w_i, many clusters k will have the same observed

distribution $p_k(t_i|w_i)$. Consider the following: In the decision tree induction process, the training data associated with some node k is divided among its children. If one of the children gets all events with w_i and the other gets none then the former child will have the same number of events $w_i t_i$ as the node k. This is likely because, firstly, this is essentially what the decision tree is optimized for, and secondly, the number of events with w_i is very small for the majority of nodes k and words w_i.

This leads us to the following design (depicted in Figure 5.2):

- We group distributions $p_k(w_i t_i)$ by the word w_i rather than by the cluster k. This way, the data blocks holding distributions for words (*model data file* in Figure 5.2) that are not used in the current task will not be loaded at all. And at the same time, observed distributions for many clusters required to compute the smoothed leaf distributions can be retrieved in one sequential disk access.

- We store integer counts $c_k(w_i t_i)$ and the total number of events in the cluster C_k (*cluster counts* in Figure 5.2), rather than probabilities $p_k(w_i t_i) \equiv \frac{c_k(w_i t_i)}{C_k}$. Since the majority of counts $c_k(w_i t_i)$ are small, most of the counts require only one byte in a variable-length coding[4], leading to an approximately 50% reduction in space usage compared to a fixed length (4 bytes) integer representation.

[4] We utilize variable length integer coding used in Berkeley DB Java Edition [85], which encodes integers between -119 and 119 in one byte, while adding an extra byte for larger numbers.

- As we pointed out above, for a given word w_i many clusters have identical counts[5] $c_k(w_i t_i)$, and the duplicates can be eliminated (*counts file* in Figure 5.2). In fact, our experiments showed that as many as 90% counts are duplicated, and elimination of the duplicates leads to further 50% reduction in disk space usage.

There are methods that achieve compact probability representation for language models by utilizing randomized approximate storage, e.g., [104, 6]. These methods have been successfully used for n-gram models, reducing on-disk footprint substantially without degrading the quality of the LM. We have evaluated the applicability of this approach to our decision tree-based joint language model by using an implementation of Van Durme and Lall [6] generously provided by Ben Van Durme. We have found that the amount of storage required for his method is comparable to the format we have outlined in Figure 5.2. However, random access is essential in his method, thus it requires the entire model to be loaded into the main memory, which significantly limits the scalability of the model.

5.1.6 The Decoding Algorithm

As in HMM decoding, in order to compute probabilities for i-th step, we need to sum over $|T|^{n-1}$ possible combinations of tags in the history, where T is the set of tags and n is the order of the model. With $|T|$ predictions for the i-th step, we have $O(|T|^n)$ computational complexity per word. Straightforward computation of these

[5] Note that although counts $c_k(w_i t_i)$ are often the same, the distributions $p_k(w_i t_i)$ are not, because clusters have different total number of events C_k.

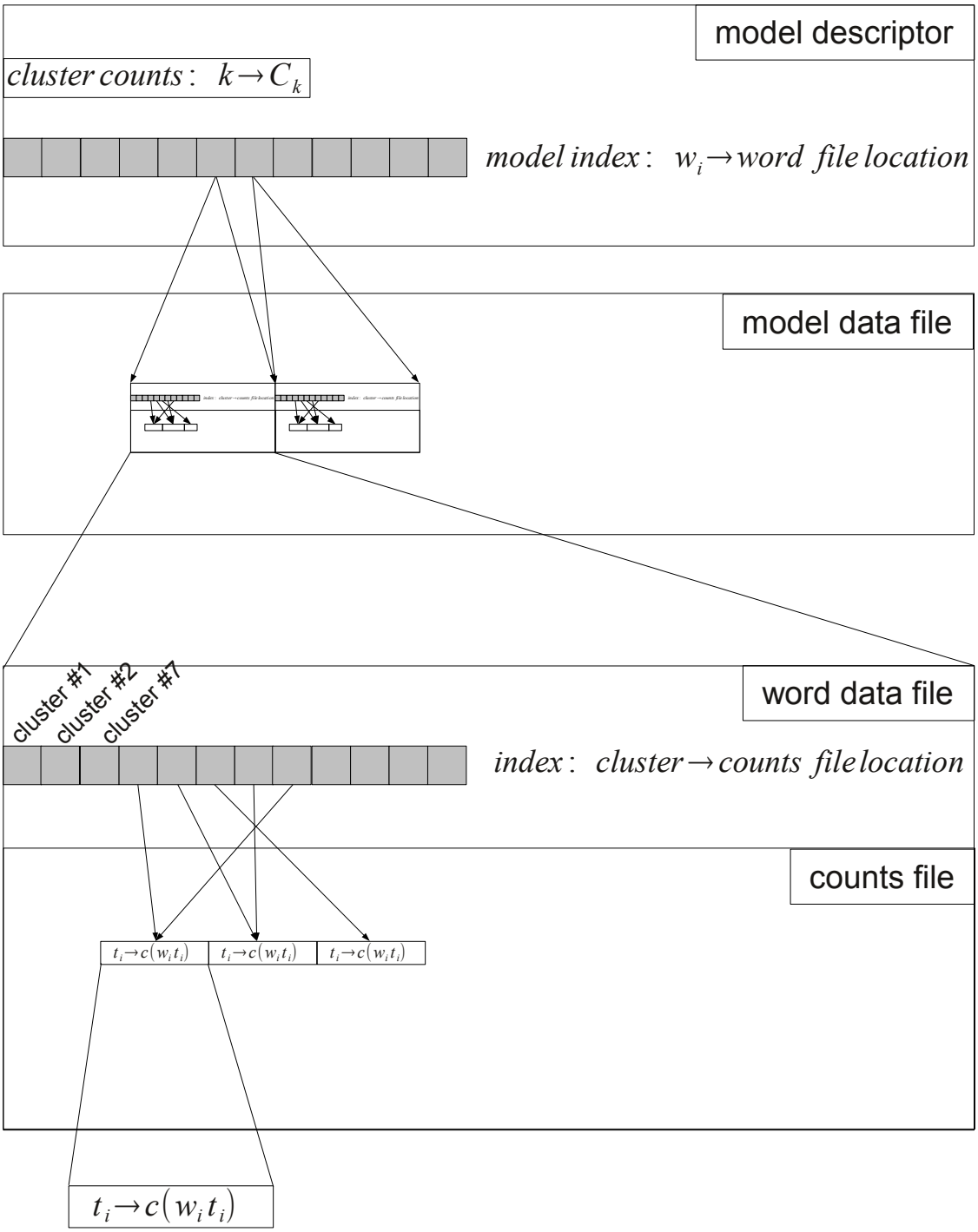


Figure 5.2: On-disk format for CLIP LM. "Model descriptor" contains the mappings of w_i to the location of the "word data file," as well as the total counts of events C_k for all clusters. "Word data file" consists of an index and a counts file, where the index maps a cluster k to the location of the counts $c(w_i t_i)$ in the counts file. Note that one set of counts $c(w_i t_i)$ may be shared among multiple clusters.

probabilities is costly even for a trigram model with POS tags, i.e., $n = 3, |T| \approx 40$. A standard approach to limit computational requirements is to use beam search where only $\mathcal{N}$ most likely paths are retained. However, with fine-grained tags where $|T| \approx 1,500$, a tractable beam size would cover only a small fraction of the entire space, leading to search errors such as pruning good paths.

Recall that a decision tree-based language model has a history clustering function $\Phi(w_{i-n+1}^{i-1} t_{i-n+1}^{i-1})$ represented by the decision tree, and in that tree, there are only $|\Phi|$ distinct clusters. For larger tagsets and order n, $|\Phi| \ll |T^{n-1}|$; moreover, any given word history $\mathcal{W}_1^{i-1}$ (we will use $\mathcal{W}$ to denote concrete values of word context, not to be confused with word context variables w) appears in only a subset of the clusters in Φ. $w_{i-n+1}^{i-1} = \mathcal{W}_{i-n+1}^{i-1}$ can be thought of as a plane in the space $w_{i-n+1}^{i-1} t_{i-n+1}^{i-1}$, the domain of clustering function Φ. The function Φ maps the plane $w_{i-n+1}^{i-1} = \mathcal{W}_{i-n+1}^{i-1}$ to a subset of clusters from its range.

$$\Phi(w_{i-n+1}^{i-1} t_{i-n+1}^{i-1})\big|_{w_{i-n+1}^{i-1} = \mathcal{W}_{i-n+1}^{i-1}} \;\Rightarrow\; \hat{\Phi}_{\mathcal{W}_{i-n+1}^{i-1}}(t_{i-n+1}^{i-1}) \tag{5.1}$$

The number of distinct clusters in $\hat{\Phi}_{i-n+1}^{i-1}$ depends on the decision tree configuration and can vary greatly for different words $\mathcal{W}_{i-n+1}^{i-1}$ in the history, but generally it is relatively small:

$$|\hat{\Phi}_{\mathcal{W}_{i-n+1}^{i-1}}(t_{i-n+1}^{i-1})| \;\ll\; |\Phi|$$

Thus, the number of probabilities that we need to compute $|\hat{\Phi}_{\mathcal{W}_{i-n+1}^{i-1}}| \cdot |T|$ is much smaller than $|T|^n$ for standard HMM decoding.

In order to achieve an overall complexity close to $O(|\hat{\Phi}_{\mathcal{W}_{i-n+1}^{i-1}}| \cdot |T|)$, the decoding algorithm must cluster the tag space t_{i-n+1}^{i-1} without having to iterate through every element of this space: Because the tag context space consists of $|T|^{n-1}$ elements, iterating over all t_{i-n+1}^{i-1} would result in the same complexity as the standard HMM decoding algorithm, which is much greater than would be practical. The only available partitioning approach that does not require iterating over all elements is top-down partitioning, in which we begin with the entire space t_{i-n+1}^{i-1} as a single cluster and then partition it recursively into $|\hat{\Phi}_{\mathcal{W}_{i-n+1}^{i-1}}|$ clusters. Note that the application of the decision tree clustering function Φ to the context space $\mathcal{W}_{i-n+1}^{i-1} t_{i-n+1}^{i-1}$ achieves exactly what we require. Specifically:

- The decision tree Φ recursively partitions the entire context $w_{i-n+1}^{i-1} t_{i-n+1}^{i-1}$ in a top-down manner.
- Questions about words in the context cut off elements of context space $w_{i-n+1}^{i-1} t_{i-n+1}^{i-1}$ that do not match the observed word history $\mathcal{W}_{i-n+1}^{i-1}$, effectively discarding clusters that do not belong to $\hat{\Phi}_{\mathcal{W}_{i-n+1}^{i-1}}$.
- Questions about tags partition the tag space into $|\hat{\Phi}_{\mathcal{W}_{i-n+1}^{i-1}}|$ clusters.

Note that a top-down partitioning does not automatically achieve partitioning of the tag space without iterating over each element; we need a special representation for the tag space to accomplish that. Recall that the set of tags T is represented as a binary tree (see Figure 4.2 in Section 4.4), thus, the entire space t_{i-n+1}^{i-1} can be represented as shown in Figure 5.3. Also recall that the only type of questions

about tags is in the form of a binary path prefix in the tag tree (as discussed in Section 4.4). Such a question dissects the tag tree into two parts as depicted in Figure 5.4. This partitioning does not iterate over all tags and the cost of this operation is approximately $O(\log |T|)$[6].

Let us illustrate how the decoding algorithm (Algorithm 5) works in the case of a bigram model. We represent states in the decoding lattice as shown in Figure 5.5, where p_{in}^{S} is the probability of reaching the state S:

$$p_{in}^{S} = \sum_{S' \in IN_S} \left[p_{in}^{S'} \; p_{S'}(w_{i-2}) \sum_{t \in T_{S'}} p_{S'}(t|w_{i-2}) \right] \tag{5.2}$$

and IN_S is the set of incoming links to the state S from the previous time index, and $T_{S'}$ is the set of tags generated from the state S' represented as a fragment of the tag tree. Note that since we maintain the property that the probability assigned to an inner node of the tag tree is the sum of probabilities of the tags it dominates, the sum $\sum_{t \in T_{S'}} p_{S'}(t|w_{i-2})$ is stored at the root of $T_{S'}$, and therefore it is an $O(1)$ operation.

Now given the state S at time $i-1$, in order to generate tag predictions for i-th word, we apply questions from the history clustering tree, starting from the top. As we mentioned above, questions about words always follow either a *true* or *false* branch, implicitly computing $\hat{\Phi}_{\mathcal{W}_{i-n+1}^{i-1}}(t_{i-n+1}^{i-1})$. Questions about tags can split the state S into two states S_{true} and S_{false}, each retaining a part of T_S as shown in the Figure 5.4. This process continues until each fragment of each state at the time

[6] $O(\log |T|)$ is the estimate for a balanced tree. In practice, we do not require the tag tree to be balanced, but as Figure 4.2 shows, the tree does not degrade into a chain to require $O(|T|)$ estimate.

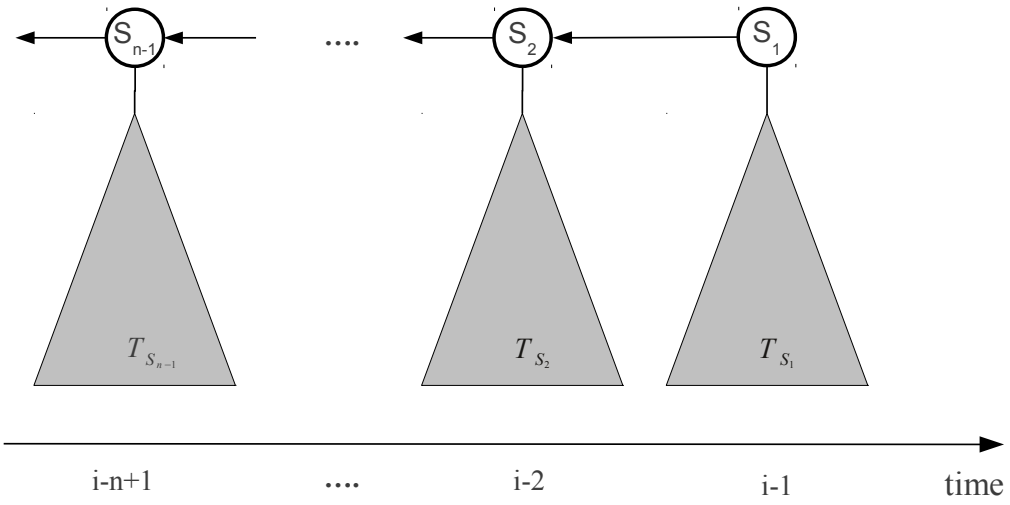


Figure 5.3: The representation of the tag context space t_{i-n+1}^{i-1}. Each state $S_1, S_2, \ldots, S_{n-1}$ is associated with a set of tags $T_{S_1}, T_{S_2}, \ldots, T_{S_{n-1}}$, respectively. A connection between states S_1 and S_2 ($S_2 \longleftarrow S_1$) denotes all combinations of tags $\{(t_1, t_2) \ : \ t_1 \in T_{S_1}, t_2 \in T_{S_2}\}$ in the connected states. Thus, the sequence of $n-1$ states above represents $|T_{S_1}| \cdot |T_{S_2}| \cdot \ldots \cdot |T_{S_{n-1}}|$ elements of the tag space.

$i-1$ reaches the bottom of the decision tree, at which point new states for time i are generated from the clusters associated with leaves. The states at $i-1$ that generate the cluster $\Phi_{\tilde{S}}$ become the incoming links to the new state $\tilde{S}$ at time i.

In Figure 5.6, we illustrate how the decoding algorithm works on a simple example. We depict the partitioning of the context for both types of questions (about words and tags) in a bigram model, as well as the generation of probabilities for the following word, i.e., we illustrate an entire decoding cycle.

- **Step (a)** Let us assume that at time $i-1$, we have one state in the decoding lattice with two incoming links from time $i-2$. We associate each state at time $i-1$ with the root of the decision tree and place these state into the agenda. We repeat the following steps until there are no states left in the

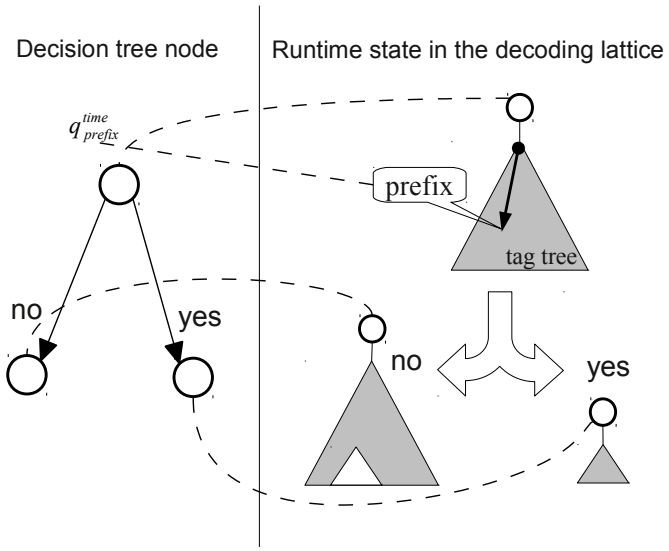


Figure 5.4: Questions about tags split states (see Figure 5.5) in the decoding lattice represented by tag trees.

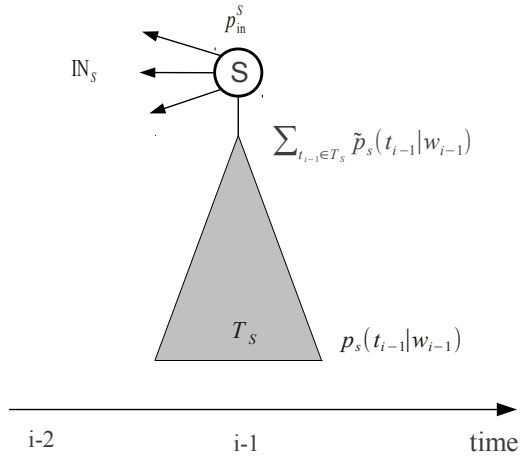


Figure 5.5: A state S in the decoding lattice is associated with the probability p^S_{in} of reaching the state through the links IN_S, as well as the word emission probability $p_S(w_{i-1})$ and the set of tags T_S emitted from the state S. The tag emission probability $p_S(t_{i-1}|w_{i-1})$ for all $t_{i-1} \in T_S$ is represented in the form of the tag tree where each internal node contains the sum of emission probabilities for the tags it dominates.

```
1  S ← new-state
2  word(S) ← <s>
3  P_IN(S) ← 1.0
4  node(S) ← ROOT
5  agenda ← { S }
6  for i = 1 ... N do
7      predictions(i) ← ∅
8      repeat
9          S ← pop-item(agenda)
10         if is-leaf(node(S)) then
11             predictions(i) ← predictions(i) ∪ { new-prediction(S) }
12             continue
13         (trueS,falseS) ← partition-state(S, question(node(S)))
14         if trueS ≠ None then agenda ← agenda ∪ { trueS }
15         if falseS ≠ None then agenda ← agenda ∪ { falseS }
16     until agenda = ∅
17     agenda ← predictions(i)
18 return sum-probabilities(predictions(N))
```

Algorithm 5: Decoding algorithm for decision tree syntactic LM

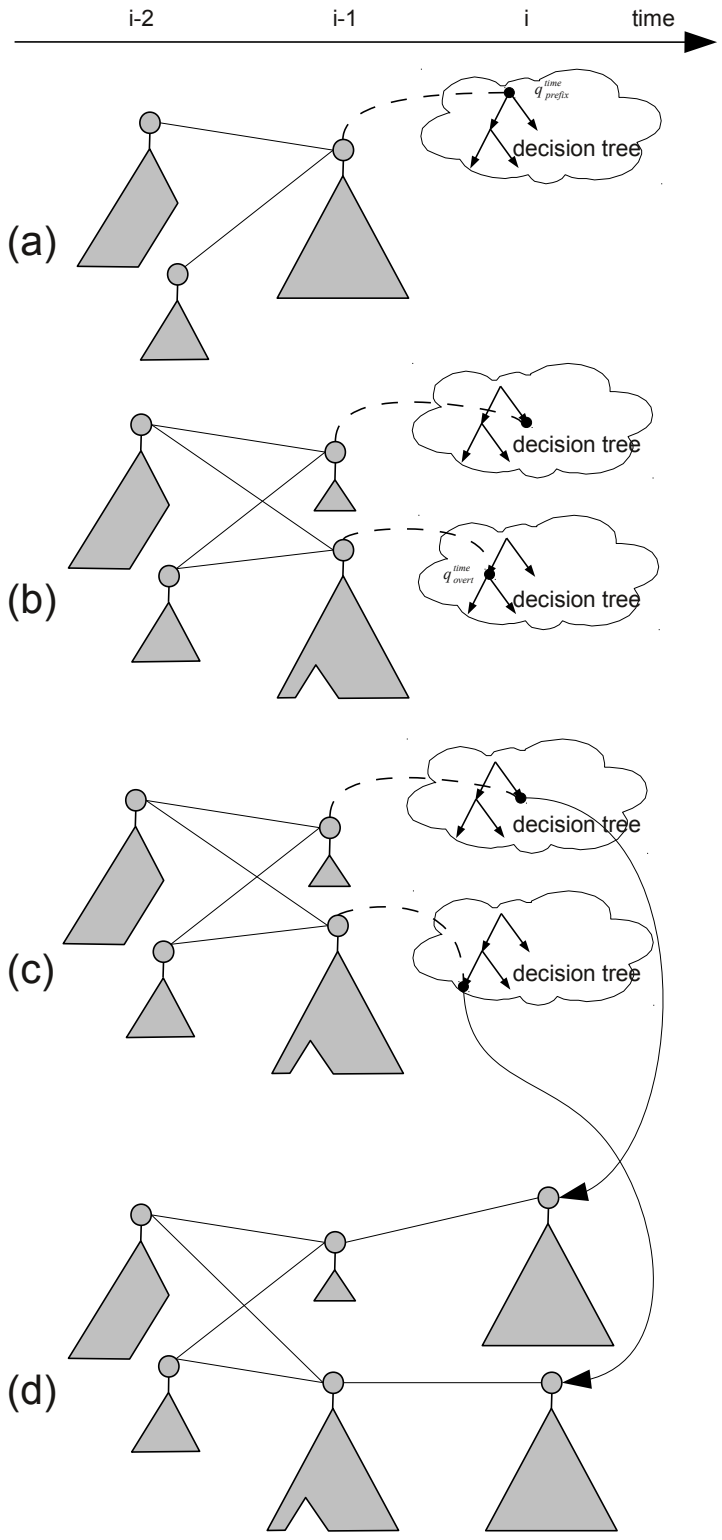


Figure 5.6: An example of the decoding process

agenda.

- **Step (b)** Suppose that the question at the root node of the decision tree is a question about tags that splits the state into two states, as was shown in Figure 5.4. Now the two new states are associated with the *true* and the *false* branches of the root node. The state on the top in Figure 5.6 (b) has reached a leaf of the decision tree, ending the partitioning of this state (the state is removed from the agenda at this point). Note that both of the new states can be reached from the same states at time $i-2$; and the probability of reaching these new states is the same as the probability of reaching the original, unsplit state.

- **Step (c)** Suppose that the question we apply to the state remaining in the agenda is about words. In this case, the state is not split, because answer to the question depends on the words in the history $\mathcal{W}_{i-n+1}^{i-1}$ but not on the tags associated with the state. Thus, the algorithm simply takes either the *true* or the *false* branch. The remaining state also reaches a leaf of the decision tree, ending the partitioning step of the algorithm.

- **Step (d)** Now that we know which leaves (clusters) the states at time $i-1$ belong to, we can generate probabilities for the new states at time index i. The newly generated states have incoming links from the states they were generated from. Note that there could be more than one if there were several states at $i-1$ before partitioning. The incoming probability for the new states is computed similarly to Eq. 5.2. This ends a cycle of the decoding algorithm,

and we proceed to splitting the states at time i.

Higher order models work similarly, except when a question concerns time $i-x$, we apply it all states at time $i-x$ that are connected with the state at time $i-1$ under consideration. A forest model with multiple decision trees is decoded similarly, instead of a single decision tree node, we associate with each state a tuple of nodes, where each node corresponds to one decision tree in the forest. Steps (b) and (c) are applied to a state until all decision trees associated with the state have reached their respective leaves.

5.1.6.1 Coarse and Fine Decoding

Note that in the decoding algorithm, a state S with a small (compared to other states at the same time index) incoming probability p_{IN}^S tends to have small impact on the overall probability of the sentence. In decoding algorithms that use beams of fixed width, such states are typically pruned to save computation. However, the specifics of our model and the decoding algorithm allow us to reduce computational cost by using a *coarser* probability estimation for such states instead of pruning them entirely.

Note that a decision tree represents a hierarchy of clusters, i.e., the inner nodes are also clusters, albeit coarser than leaves. Also note that the smoothed distributions $\tilde{p}_k(w_i t_i)$ are readily available for the inner nodes, as well as for the leaves, because the smoothed distribution at a leaf is computed recursively as an interpolation of the observed distribution at the leaf with the smoothed distribution

of its parent node (see Section 5.1.4).

The only change in the decoding algorithm (Algorithm 5) necessary to affect this approximation is at line 10, where we can add an alternative termination condition, such as a threshold:

$$\begin{aligned} p_{in}^{S} \cdot p_S(w_{i-2}) \sum_{t \in T_S} p_S(t|w_{i-1}) &< \theta \cdot \sum_{S'} \left(P_{in}^{S'} \cdot p_{S'}(w_{i-1}) \sum_{t \in T_{S'}} p_{S'}(t|w_{i-1}) \right) \\ &< \theta \cdot p(w_1^{i-1}) \end{aligned}$$

A coarse probability estimation for the state S is used when its relative contribution to the total probability is smaller than the threshold parameter θ.

To evaluate the trade-off between accuracy and computational efficiency, we conducted an experiment on the WSJ task (see setup in Section 3.2). For this experiment, we compute the perplexity of the PTB WSJ Section 23 and measure CPU time for different values of the threshold parameter θ. This experiment was conducted on a dual Intel Xeon E5430 machine with 16 gigabytes of RAM. The results are presented in Figure 5.7. Note that with $\theta \leq 10^{-3}$, the model does not suffer a noticeable degradation in performance (less than a 1% increase in perplexity), while the increases in speed is manifold.

Even with all improvements we have described in this section, our model is still approximately two orders of magnitude slower than the SRILM implementation of the n-gram model[7]: Computing perplexity of the MT08-NW data set (29,341 words) takes 1,152 seconds (CPU time) using our model with $\theta = 10^{-3}$ threshold,

[7] Not surprisingly, since we have to sum over all possible tag assignments!

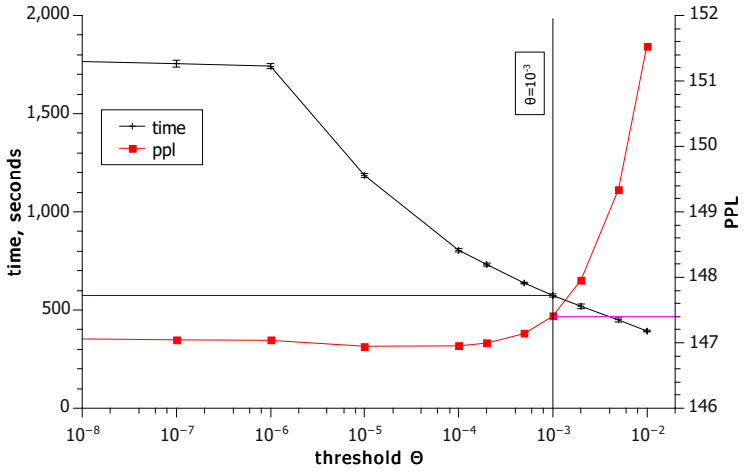


Figure 5.7: Perplexity and decoding time for different threshold parameters θ estimated on the WSJ task. Time is the sum of *user* and *system* time as measured by *time* utility, with model initialization time subtracted. Error bars on the time plot indicate one standard deviation computed from a sample of 10 runs.

and only 13 seconds using the n-gram model (both models are fourgram models trained on the 210M words of Newswire data, described in Section 3.3). However, we did reduce asymptotic computational complexity of the standard HMM decoding ($O(|T|^n) = O(1,397^4) \approx 3.8 \cdot 10^{12}$ *per word*) to something that can be feasibly computed. We believe there is room for substantial increase in speed of our model by careful optimization of algorithms and data structures, we intend to pursue this direction of work in the future.

5.2 N-gram Approximation

Unlike n-gram models, in which the observed history is limited to $n-1$ words, in a joint syntactic model, the entire word sequence w_1^{i-1} is required to predict the word w_i, as can be seen in Eq. 2.6:

$$p(w_i|w_1^{i-1}) = \frac{p(w_1^i)}{p(w_1^{i-1})} = \frac{\sum_{t_1 \ldots t_i} \prod_{j=1}^{i} p(w_j t_j | w_1^{j-1} t_1^{j-1})}{\sum_{t_1 \ldots t_{i-1}} \prod_{j=1}^{i-1} p(w_j t_j | w_1^{j-1} t_1^{j-1})} \tag{2.6}$$

Note that this is the case even if the model limits its context to $n-1$ words and tags: $w_{i-n+1}^{i-1} t_{i-n+1}^{i-1}$. This is because the distribution of tags t_{i-n+1}^{i-1} depends on the prior context; in a sense, the model carries information from the beginning of the sentence through the distribution of hidden states (tags).

The necessity of having the entire word sequence w_1^{i-1} to predict the word w_i limits the applicability of the model to rescoring n-best lists, or with special search algorithms, confusion networks (e.g., Deoras and Jelinek [34]) and word lattices (e.g., Rastrow et al. [91]). On-line application of such a model in ASR or MT decoding is infeasible because the number of different word sequences grows exponentially with the length of the input. However, it is likely that remote words have very little influence on the probability of w_i. This is due in part to the fact that the hidden states can carry a limited amount of information, and partly, because in most cases, a few previous words are sufficient to disambiguate syntactic and semantic roles of a word. Given these assumptions, we propose the following approximation for the probability of a word-tag sequence:

$$p(w_{i-n+1}^i t_{i-n+1}^i) \approx \left(\prod_{k=i-n+2}^{i} p(w_k t_k | w_{i-n+1}^{k-1} t_{i-n+1}^{k-1}) \right) \cdot p(w_{i-n+1} t_{i-n+1})$$

That is, the first word-tag $w_{i-n+1} t_{i-n+1}$ in the n-gram $w_{i-n+1}^i t_{i-n+1}^i$ is predicted using a unigram distribution $p(w_{i-n+1} t_{i-n+1})$ with no context, the second word-tag

$w_{i-n+2}t_{i-n+2}$ is predicted using a bigram model $p(w_{i-n+2}t_{i-n+2}|w_{i-n+1}t_{i-n+1})$, and so on. Marginalizing tag distributions out, similarly to Eq. 2.6, we get the following n-gram probability estimation:

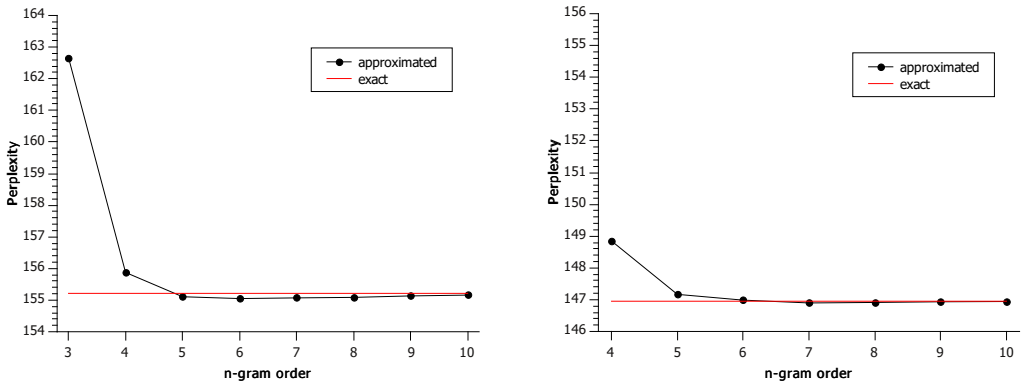
$$p(w_i|w_{i-n+1}^{i-1}) \approx \frac{\sum_{t_{i-n+1},\ldots,t_i} p(w_i t_i|w_{i-n+1}^{i-1}t_{i-n+1}^{i-1}) \cdot p(w_{i-1}t_{i-1}|w_{i-n+1}^{i-2}t_{i-n+1}^{i-2}) \cdot \ldots \cdot p(w_{i-n+1}t_{i-n+1})}{\sum_{t_{i-n+1},\ldots,t_{i-1}} p(w_{i-1}t_{i-1}|w_{i-n+1}^{i-2}t_{i-n+1}^{i-2}) \cdot \ldots \cdot p(w_{i-n+1}t_{i-n+1})} \tag{5.3}$$

With this approximation, a joint syntactic model can be applied in any application where regular n-gram models are used.

5.2.1 Accuracy of N-gram Approximation

N-gram probability estimation in Eq. 5.3 is an approximation of the Eq. 2.6, because in Eq. 5.3, we use lower order models (down to a unigram!) to estimate probabilities of the first words in an n-gram. Before applying this approximation, we need to make sure that it does not degrade the quality of the model significantly. In this section, we evaluate the accuracy of this approximation and its dependency on the size of the n-gram and the order of the model.

We conducted two experiments using WSJ '94-'96 setup discussed in Section 3.2. We measured the perplexity of the test set using both Eq. 2.6 and its approximation Eq. 5.3, and compared probabilities of individual n-grams. In the first experiment, we used a trigram joint syntactic model and varied the size of the n-gram from 3 to 10; and in the second experiment, we used a fourgram model



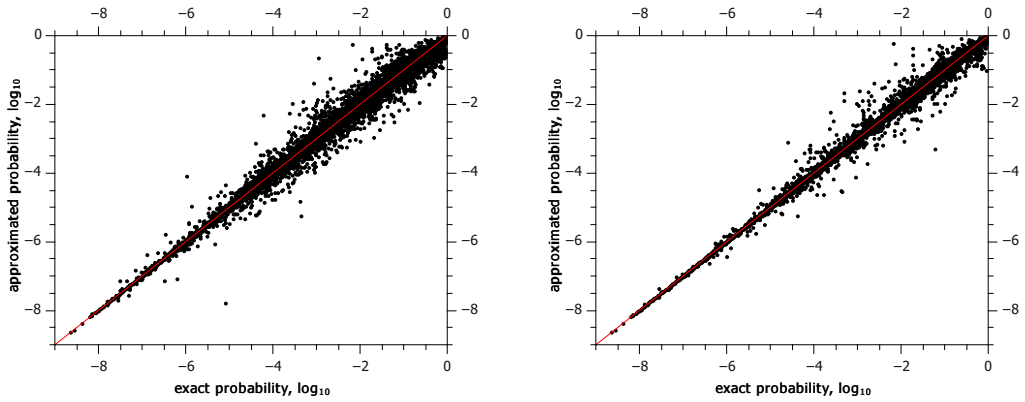
(a) Trigram (b) Fourgram

Figure 5.8: Perplexity on PTB WSJ Section 23 using exact (Eq. 2.6) and n-gram approximated (Eq. 5.3) probability estimation for trigram (a) and fourgram (b) joint syntactic model.

and varied the n-gram size from 4 to 10. Naturally, we expect that as the size of the n-gram grows, the approximated probability estimation should become closer to Eq. 2.6, however, the most important question is just how large an n-gram is needed for the loss in accuracy due to the approximation to be negligible.

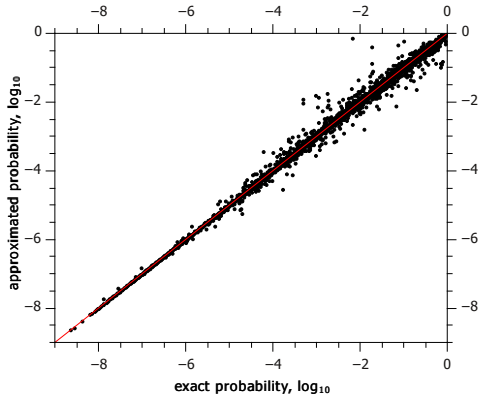
In Figure 5.8, we present the perplexity results for trigram and fourgram models. Note that for both models, approximating probability estimation with n-grams larger than a fourgram results in very a slight difference from the exact estimation, while using fourgrams produces approximately a 1% increase in perplexity.

In Figures 5.9 and 5.10 we plot probabilities $p(w_i|w_1^{i-1})$ in the test set (54,470 words) estimated using Eq. 2.6 (exact estimation) and Eq. 5.3 (approximated estimation). As predicted, as the order of n-gram increases, the approximated estimation becomes very close to the exact. Interestingly, the approximation with lower order n-grams, shown in Figures 5.9 (a) and Figure 5.10 (a), tends to underesti-

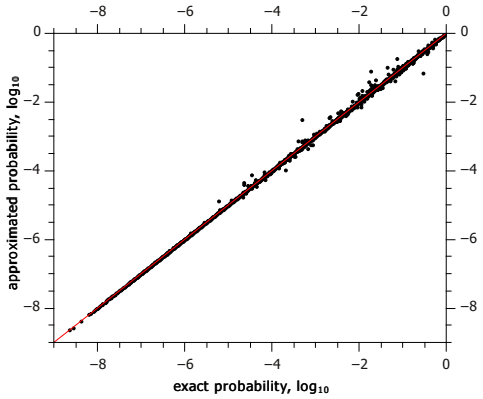


(a) n=3

(b) n=4

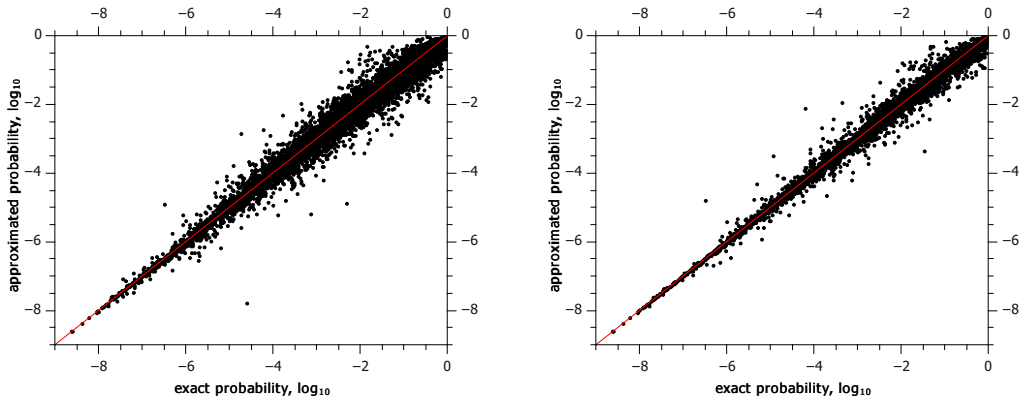


(c) n=5



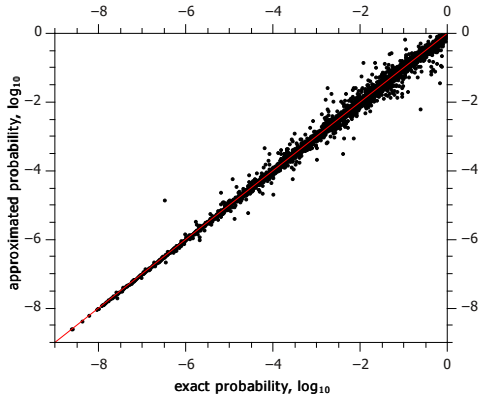
(d) n=10

Figure 5.9: Exact (Eq. 2.6) and n-gram approximated (Eq. 5.3) probabilities computed on WSJ Section 23 test set using a joint syntactic trigram model for various values of n-gram order n

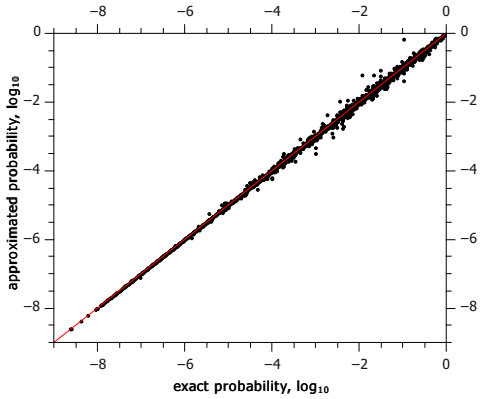


(a) n=4

(b) n=5



(c) n=6



(d) n=10

Figure 5.10: Exact (Eq. 2.6) and n-gram approximated (Eq. 5.3) probabilities computed on WSJ Section 23 test set using a joint syntactic fourgram model for various values of n-gram order n

mate n-grams with very high probabilities. We believe that the main reason for this phenomenon is the inability of the approximation to properly account for syntactic ambiguity in the context due to the use of the unigram distribution.

5.3 Contributions

- In Section 5.1, we have proposed solutions to the engineering problems that are essential to make a large-scale joint syntactic model tractable: algorithms with limited memory footprint and efficient data structures for on-disk representation of the model. We have also proposed a novel decoding algorithm which, in combination with decision tree-based context clustering, has enabled us to utilize fine-grained syntactic tags producing a stronger language model than would be possible by utilizing less informative tags, such as part-of-speech tags.

- In Section 5.2, we have proposed a method for approximating a syntactic language model by word n-grams, and we have evaluated the accuracy of the approximation.

5.4 Summary

In this chapter, we have explored a number of computational issues that arise from the complexity of the joint syntactic model and the necessity for a practically useful language model to be able to utilize large amounts of training data. We have presented scalable algorithms for decision tree construction and probability estimation,

as well as an efficient on-disk format for the syntactic language model. We have also introduced an efficient decoding algorithm that exploits the decision tree to eliminate unnecessary computation, and additionally, provides a straightforward way to trade a little accuracy for a considerable reduction in computation. In addition, we have proposed an n-gram approximation for the syntactic model—the approximation that extends the applicability of the model to the tasks where it would be infeasible otherwise—and have evaluated the accuracy of this approximation.

Chapter 6

Large-scale Experiments in ASR and MT

In this chapter, in order to evaluate the impact of our model on speech recognition and machine translation tasks, we conduct Broadcast News ASR word lattice rescoring and Arabic-to-English MT experiments, comparing the joint syntactic model described in Chapter 4 with respective n-gram baselines.

6.1 Broadcast News ASR Rescoring Experiments

In addition to ASR n-best list rescoring experiments presented in Section 4.8.4.2 in Table 4.9, which were based on speech produced by reading WSJ articles, we evaluate our model on the Broadcast News ASR lattice rescoring task using the setup described in Section 3.1. To select 1-best hypotheses, we utilized the lattice search framework developed by Rastrow et al. [91]. This framework extracts full utterance hypotheses from a lattice, similarly to n-best lists; however, it has been shown to be much more efficient than n-best lists, in the sense that it does not require as many hypotheses to be evaluated as a standard n-best list to achieve the same Word Error Rate (WER). This lattice search framework utilizes a stochastic hill climbing method, and therefore, may converge to a different solution every time it is restarted; hence, we restart the algorithm 10 times (as recommended by Ariya Rastow in a personal communication).

model	PPL	WER
n-gram	239.6	15.2%
syntactic (1)	217.0	14.7%
syntactic (2)	208.9	14.6%

Table 6.1: Perplexity and WER results on the RT04 dataset. "syntactic (1)" denotes the joint syntactic LM utilizing four decision trees (1w1t + 2w2t + 3w3t + 4w4t, as described in Section 4.8.4.2). In addition to these trees, "syntactic (2)" has four additional trees: 3w4t + 4w3t + 2w3t + 3w2t.

The hypotheses were selected based on the following score:

$$w_{AM} \log P(A|W) + \log P(W)$$

where $P(A|W)$ is the acoustic model score, $P(W)$ is the language model score for the hypothesis W, and w_{AM} is the acoustic model weight. We selected w_{AM} to minimize WER on a heldout set (the combined NIST dev04f and NIST rt03f sets) containing 840 utterances.

The baseline n-gram model used in these experiments was trained on 130M words of Hub4 CSR 1996 [44] (the same dataset that was used to train the syntactic models) using the language modeling tool from IBM Attila framework [18]. The n-gram LM is a fourgram model smoothed using the interpolated modified Kneser-Ney method (described in Section 2.2.1.3).

In Table 6.1, we present WER results for three language models: the n-gram baseline and two joint syntactic models. Both syntactic models utilize *parent* tagset (described in Section 2.4.2.3). The model denoted "syntactic (1)" utilizes the stan-

dard set of trees (from unigram to fourgram), while the model denoted "syntactic (2)" has four additional trees 3w4t + 4w3t + 2w3t + 3w2t, similarly to the experiments presented in Section 4.8.4.2. Both syntactic models outperform the baseline significantly. The difference in word error rate between the two syntactic models was not found statistically significant in this experiment, however, the lower perplexity of the "syntactic (2)" model compared to the "syntactic (1)" model suggests that the improvement in WER is not random. Nonetheless, we conclude that the difference in performance between the model "syntactic (1)" using the standard forest configuration (1w1t+2w2t+3w3t+4w4t) and the model "syntactic (2)" is not large enough to justify the increased computational expense of additional trees.

6.2 Machine Translation Experiments

The n-gram approximation described in Section 5.2 enables us to apply our syntactic model in applications, such as MT decoding, where it would otherwise be infeasible because the number of unique translation hypotheses is astronomical even for a moderate length source language sentence. In this section, we present the results of experiments for integration of our syntactic model into `CDEC` machine translation decoder using the n-gram approximation described in Section 5.2.

This evaluation includes translations of GALE data sets representing four different genres: Broadcast News (BN), Broadcast Conversation (BC), Newswire (NW), and Weblogs (WB). Naturally, language in different genres differs substantially; therefore, it should be beneficial for each genre, if the language model is

adapted to that specific genre. Language model adaptation has been widely used in speech recognition (e.g., [4, 1, 12]); however, in machine translation community, LM adaptation techniques have received relatively little attention. Perhaps, one of the reasons is that language model adaptation has not always been successful in machine translation. In a domain adaptation study, Koehn and Schroeder [67] report that using only in-domain language model resulted in a slightly better performance than using a linear interpolation of in-domain and out-of-domain language models or using both LMs (which is similar to log-linear interpolation). However, Bulyko et al. [11] observed moderate improvements (0.3-0.4 BLEU) on the MT n-best list rescoring task by using a combination of several language models trained on data from different sources. They hypothesized that the improvements could be larger if the adaptation had been performed during the decoding rather than in a post-processing step, such as n-best rescoring.

Similarly to Bulyko et al. [11], we trained multiple models (n-gram and syntactic) on data from different sources as described in Section 3.3. As the baseline, we use a single n-gram model trained on the combined data. In Tables 6.2 through 6.5, "ngram-4" denotes the combination of four n-gram models, and "joint-4" refers to the combination of four joint syntactic models using the n-gram approximation (Section 5.2). All models (n-gram and syntactic) utilize fourgram context[1].

We report results using the BLEU metric [86], which is the most frequently reported metric for evaluation of MT systems. Unfortunately, there are unresolved

[1] In the case of the syntactic model, we use a joint syntactic fourgram model approximated by word fourgrams as in Figure 5.10 (a).

issues related to establishing the statistical significance of the difference between two systems. Collins et al. [28] argued that the BLEU metric may not meet the requirements for the bootstrap resampling [38] method used in prior work (e.g., [65, 114]) to establish statistical significance; hence, they proposed to use the sign test [70] instead. Chiang et al. [22] pointed out that the sign test produces counter-intuitive results in some realistic cases because of the brevity penalty factor in the BLEU score (see Eq. 2.8 in Section 2.1.4.3). They proposed to use bootstrap resampling with a modified version of the BLEU score.

Recently, Clark et al. [25] pointed out that testing one pair of hypotheses produced by two systems to be compared is not sufficient to establish whether one system is significantly better than the other because of the substantial variance observed in the MT optimization procedures. They proposed to evaluate *multiple* hypotheses produced by different optimization runs of each system, and they made their evaluation tool publicly available. They use stratified shuffling [23, 93] to estimate the confidence interval of observing the difference in *average* BLEU scores between two system, which made interpretation of the reported p-values somewhat counter-intuitive. They did not discuss the implications of the brevity penalty on the significance test which, based on the previous arguments made by Collins et al. [28] and Chiang et al. [22], deserves attention. Therefore, while we fully support the argument of Clark et al. [25] that *several* optimization runs of each system must be evaluated for significance testing, we have opted for Student's t-test [103] to establish significance of the differences between our systems. In this test, each system produces a sequence of BLEU scores, where each score corresponds to a different

	model	avg	median	σ	min	max	Δ avg	p-value	
								vs. baseline	vs. ngram-4
DEV10-dev	baseline	30.10	30.12	0.11	29.96	30.25	-	-	-
	ngram-4	29.97	29.99	0.11	29.78	30.07	-0.13	0.1066	-
	joint-4	30.49	30.52	0.10	30.34	30.61	0.39	0.0005	0.0001
DEV09-dev	baseline	28.63	28.60	0.08	28.56	28.76	-	-	-
	ngram-4	28.88	28.90	0.15	28.65	29.07	0.25	0.0121	-
	joint-4	29.63	29.61	0.08	29.55	29.76	1.00	0.0001	0.0001
DEV09-tune	baseline	29.21	29.20	0.19	29.01	29.43	-	-	-
	ngram-4	29.36	29.44	0.23	29.12	29.62	0.15	0.2889	-
	joint-4	29.86	29.87	0.11	29.70	29.98	0.65	0.0002	0.0023

Table 6.2: BLEU scores for the Broadcast News (BN) genre. Each system's results are aggregated from a set of 5 optimization runs.

optimization run, and the null hypothesis is that each sequence was drawn from the same normally distributed population. We use p-value < 0.05 for significance.

In Tables 6.2 through 6.5, we present the results of our machine translation experiments. For each genre (BN, BC, NW, and WB) we performed five optimization runs of each system (baseline, ngram-4, and joint-4), using the portion of DEV10-tune corresponding to the respective genre. We then decoded the evaluation sets (DEV10-dev, DEV09-dev, and DEV09-tune) using these systems. Note that the evaluation sets are also divided by genre; therefore, we use the respective sections of the evaluation sets to evaluate the genre-tuned systems, i.e., the BLEU scores for

	model	avg	median	σ	min	max	Δ avg	p-value	
								vs. baseline	vs. ngram-4
DEV10-dev	baseline	38.18	38.18	0.07	38.09	38.27	-	-	-
	ngram-4	38.61	38.62	0.15	38.39	38.77	0.43	0.0004	-
	joint-4	39.26	39.27	0.09	39.16	39.36	1.08	0.0001	0.0001
DEV09-dev	baseline	20.48	20.48	0.10	20.35	20.62	-	-	-
	ngram-4	20.71	20.86	0.37	20.28	21.09	0.22	0.2291	-
	joint-4	20.96	21.00	0.19	20.71	21.14	0.47	0.0014	0.2137
DEV09-tune	baseline	20.95	20.93	0.08	20.87	21.07	-	-	-
	ngram-4	21.26	21.24	0.10	21.15	21.38	0.31	0.0009	-
	joint-4	21.49	21.52	0.17	21.27	21.71	0.54	0.0002	0.0279

Table 6.3: BLEU scores for the Broadcast Conversation (BC) genre. Each system's results are aggregated from a set of 5 optimization runs.

	model	avg	median	σ	min	max	Δ avg	p-value	
								vs. baseline	vs. ngram-4
DEV10-dev	baseline	36.23	36.22	0.08	36.12	36.31	-	-	-
	ngram-4	36.22	36.23	0.04	36.16	36.27	-0.01	0.8094	-
	joint-4	36.88	36.89	0.04	36.84	36.93	0.65	0.0001	0.0001
DEV09-dev	baseline	32.31	32.30	0.05	32.26	32.39	-	-	-
	ngram-4	32.59	32.59	0.13	32.47	32.78	0.28	0.0020	-
	joint-4	33.06	33.00	0.14	32.96	33.27	0.75	0.0001	0.0005
DEV09-tune	baseline	33.05	33.07	0.09	32.91	33.13	-	-	-
	ngram-4	33.23	33.16	0.14	33.12	33.46	0.19	0.0364	-
	joint-4	33.70	33.64	0.11	33.58	33.82	0.65	0.0001	0.0004

Table 6.4: BLEU scores for the Newswire (NW) genre. Each system's results are aggregated from a set of 5 optimization runs.

	model	avg	median	σ	min	max	Δ avg	p-value	
								vs. baseline	vs. ngram-4
DEV10-dev	baseline	39.80	39.84	0.13	39.57	39.88	-	-	-
	ngram-4	39.76	39.76	0.15	39.57	40.00	-0.04	0.6970	-
	joint-4	40.60	40.50	0.29	40.23	40.94	0.80	0.0006	0.0005
DEV09-dev	baseline	23.58	23.62	0.14	23.36	23.74	-	-	-
	ngram-4	23.80	23.88	0.27	23.38	24.09	0.22	0.1420	-
	joint-4	24.11	24.15	0.11	23.98	24.24	0.53	0.0002	0.0447
DEV09-tune	baseline	22.95	22.98	0.10	22.78	23.04	-	-	-
	ngram-4	23.28	23.26	0.16	23.09	23.50	0.32	0.0053	-
	joint-4	23.47	23.47	0.20	23.19	23.75	0.52	0.0009	0.1261

Table 6.5: BLEU scores for the Weblog (WB) genre. Each system's results are aggregated from a set of 5 optimization runs.

"DEV10-dev" in Table 6.2 refer to the BN section of DEV10-dev produced by the systems optimized on the BN section of DEV10-tune dataset.

Note that in many cases the difference between the minimum and the maximum score in the series of optimization runs of one system is quite substantial, which confirms the proposition of Clark et al. [25] that using a single optimization run of each system may lead to wildly inaccurate conclusions regarding the relative performance of the systems. Also note that ngram-4 and joint-4 tend to have a larger standard deviation σ, which we ascribe to the fact that these systems have more optimization parameters (each language model has an independent parameter) than the baseline system.

Although the ngram-4 system does tend to outperform the baseline system, the improvement is not very consistent across all corpora. On 6 out of 12 datasets, ngram-4 significantly outperformed the baseline, but in the other cases the difference in average scores was not found to be statistically significant, and in 3 cases ngram-4 is even slightly worse than the baseline. The syntactic model joint-4, however, significantly outperformed the baseline on every dataset. Moreover, the examination of minimum and maximum scores of the baseline and joint-4 shows that the *minimum* score of joint-4 is greater than the *maximum* score of the baseline on every dataset.

The comparison of the ngram-4 and joint-4 systems shows that the syntactic models significantly outperform the n-gram models on all datasets except two: DEV09-dev (BC) and DEV09-tune (WB). Note that both of these datasets have relatively low BLEU scores: the average scores of ngram-4 system on DEV09-dev (BC) and DEV09-tune (WB) are 20.71 and 23.28, respectively. In Figure 6.1, we

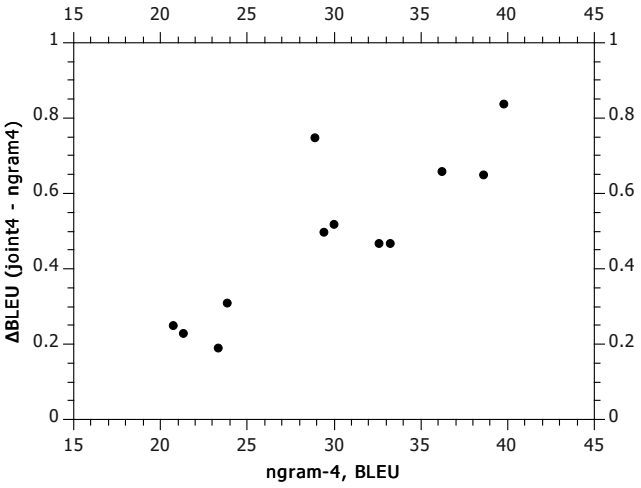


Figure 6.1: The average BLEU score of ngram-4 model (x-axis) and the difference in average BLEU scores between joint-4 and ngram-4 models (y-axis). Correlation coefficient $r = 0.85$.

present a scatterplot of the average score of ngram-4 system and the difference between the average scores of the joint-4 and ngram-4 systems using all datasets and genres (12 points total). We observe that the score of ngram-4 and the improvement provided by the syntactic system are highly correlated (Pearson's correlation coefficient $r = 0.85$). This phenomenon has an intuitive explanation: a language model is responsible for producing *fluent* hypotheses; however, if the hypotheses produced by the translation model are *inadequate*, making them more fluent will not make them [much] more similar to the reference translation. Indeed, if the translation model produces "apple" instead of "orange," it is beyond the power of a language model to correct that error. This leads us to believe that the impact of language modeling on the quality of machine translation will only increase as the translation model improves.

6.3 Contributions

- We have demonstrated that our model performs well in large-scale speech recognition and machine translation tasks, significantly outperforming n-gram baselines.

- We have observed that in machine translation experiments, the improvement of the syntactic model over an n-gram baseline strongly correlates with the quality of the translation model, thus we expect that the gains from using complex language models will increase as translation models improve.

6.4 Summary

In this chapter, we have evaluated our joint syntactic model on an ASR word lattice rescoring task and have observed significant reductions in WER compared to the n-gram baseline. We have also incorporated the n-gram approximated version of the joint syntactic model into the `CDEC` MT decoder and found that not only does our model significantly outperform a comparable n-gram model, but that the gains in the BLEU score increase as the performance of MT system improves.

Chapter 7

Contributions and Future Work

In this chapter, we summarize our contributions, present conclusions, and outline directions for future work.

7.1 Contributions

The main contributions of this thesis belong to two broad categories:

- Advancements in decision tree technology for language modeling,
- Scalable syntactic modeling

Although decision trees have been used for language modeling in prior literature (e.g., [2, 51, 112]), the methods for decision tree construction and probability estimation have been largely imported from other domains, such as decision trees for classification and n-gram models, without due analysis of the differences in the ways those domains affect such models. We have outlined the main differences between classification and language modeling tasks, and have analyzed how their differences affect decision tree construction. Additionally, the analysis of the differences has led us to discover a bias in the splitting rule metric used in all prior decision tree language models. To eliminate this bias, we have proposed a two-step splitting rule and have observed reductions in perplexity, particularly in the case of syntactic

language modeling, where the bias is most prominent.

We have also shown that backoff methods developed for n-gram models do not apply well to decision tree models because of the differences in context clustering methods. Using context clustering terminology, we have formulated the backoff property that expresses the relation between the higher- and the lower-order models in backoff interpolation schemes such as in n-gram models. We have shown that this property is essential for backoff interpolation to work well and that this property is satisfied by n-gram models but not by decision tree models. We have proposed a generalization of Jelinek-Mercer smoothing method for decision tree models, and we have observed significant improvements from using it over the previously used method.

We have also investigated methods for constructing small forests of diverse trees. Unlike prior work on forests of decision trees for language modeling, where decision trees are either "backoff" or "equivalent," the generalized interpolation for decision trees has enabled us to combine any decision trees in a single forest. We have evaluated a number of approaches for introducing variability into decision tree construction process, and we have concluded that, in the case of small forests, decision trees with the context restricted in different ways results in the best combination.

We have also tackled a number of engineering problems in order to make the model computationally tractable. We have proposed a novel decoding algorithm that exploits the decision tree structure to eliminate unnecessary computation. The algorithms and data structures that we have developed have enabled us to utilize large amounts of data (hundreds of million words) and fine-grained tagsets (thousands of

distinct tags!) in a joint syntactic model which would be intractable otherwise. Additionally, we have shown that a syntactic model can be accurately approximated by sufficiently long word n-grams. This approximation has enabled us to incorporate our model into the `CDEC` machine translation decoder. Our experiments in machine translation have demonstrated that the impact of a stronger language model (such as a syntactic LM) increases as the translation model improves. Therefore, future advancements in translation models will only increase the importance of language models.

Finally, we have made our code available at `http://code.google.com/p/clip-lm/` under GNU GPLv2.

7.2 Future Work

- The ability to easily incorporate additional features is one of the strengths of a decision tree-based language model. At the same time, morphologically rich languages, such as Arabic, Czech, or Russian have proven difficult to model using n-gram models because of the large number of word forms. Bilmes and Kirchhoff [8] have shown that adding word stem as backoff context for unknown word forms improves language modeling of Arabic, using an approach similar to n-gram models. It is likely that using such information can be even more beneficial in decision tree-based models because a decision tree is a much more flexible way of context clustering, as we have argued in Section 2.3.

Additionally, in speech recognition, non-verbal information, such as prosody, may be used to enhance language models. The structural information provided by prosody has been shown to improve parsing of speech (e.g., [54]), and therefore, we believe it would be helpful for syntactic language modeling. However, human-annotated prosodic information is usually available only in small quantities, much less than would normally be required for LM training. This problem can be addressed by using automatically produced prosodic labels, or by using a combination of models trained on datasets with and without prosodic labels.

- The generalized interpolation for decision tree-based model that we have proposed can also be applied to language model adaptation. This can be achieved by combining decision trees constructed using data from different domains and then using the generalized interpolation method to optimize the combination of trees on a heldout set from the target domain. There is a technical limitation: all decision tree models must utilize the same word vocabulary and the same tagset. However, in many applications this is an acceptable restriction, in fact, some previously used LM adaptation techniques, e.g., count merging [1], have similar restrictions.

- Decision trees are used in a large variety of applications (e.g., see [82] for a survey), both in natural language processing domain and outside of it. For example, in speech recognition, decision trees are often used to model pronunciation variations (e.g., [108]). We believe that some of the methods for

decision tree-based models that we have developed may have an impact beyond language modeling.

- Since the syntactic model can utilize a wide variety of tags (all that is required of a tag is to be predictive of the following word, as we have shown in Section 2.4.2.6), the model can be applied to low-resource languages that do not have treebanks by utilizing unsupervised techniques such as unsupervised part-of-speech tagging (e.g., [43, 45]) or unsupervised parsing (e.g., [62, 92]) to create training data for the LM.

Bibliography

[1] Michiel Bacchiani and Brian Roark. Unsupervised language model adaptation. In *Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 1, pages I–224–I–227, April 2003.

[2] Lalit R. Bahl, Peter F. Brown, Peter V. de Souza, and Robert L. Mercer. A tree-based statistical language model for natural language speech recognition. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 37(7):1001–1008, July 1989.

[3] Srinivas Bangalore. 'Almost parsing' technique for language modeling. In *Proceedings of the International Conference on Spoken Language Processing*, volume 2, pages 1173–1176, 1996.

[4] Jerome R. Bellegarda. Statistical language model adaptation: review and perspectives. *Speech Communication*, 42(1):93–108, 2004. Adaptation Methods for Speech Recognition.

[5] Yoshua Bengio, Duchar Réjean, Pascal Vincent, and Christian Janvin. A neural probabilistic language model. *Journal of Machine Learning Research*, 3:1137–1155, March 2003.

[6] Van Dur Benjamin and Ashwin Lall. Probabilistic counting with randomized storage. In *Proceedings of the 21st international jont conference on Artifical intelligence*, pages 1574–1579, San Francisco, CA, USA, 2009. Morgan Kaufmann Publishers Inc.

[7] Adam Berger. Convexity, maximum likelihood and all that. Technical report, 1996.

[8] Jeff A. Bilmes and Katrin Kirchhoff. Factored language models and generalized parallel backoff. In *Proceedings of the Conference on Human Language Technologies and North American Chapter of the Association for Computational Linguistics*, pages 4–6, Stroudsburg, PA, USA, 2003. Association for Computational Linguistics.

[9] Taylor L. Booth and Richard A. Thompson. Applying probability measures to abstract languages. *IEEE Transactions on Computers*, pages 442–450, 1973.

[10] Peter F. Brown, Vincent J. Della Pietra, Peter V. deSouza, Jennifer C. Lai, and Robert L. Mercer. Class-based n-gram models of natural language. *Computational Linguistics*, 18(4):467–479, 1992.

[11] Ivan Bulyko, Spyros Matsoukas, Richard Schwartz, Long Nguyen, and John Makhoul. Language model adaptation in machine translation from speech. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, volume 4, pages 117–120, April 2007.

[12] Ivan Bulyko and Mari Ostendorf. Getting more mileage from web text sources for conversational speech language modeling using class-dependent mixtures. In *Proceedings of the Conference on Human Language Technologies and North American Chapter of the Association for Computational Linguistics*, pages 7–9, 2003.

[13] Wray L. Buntine. *A Theory of Learning Classification Rules*. PhD thesis, Sydney, February 1990.

[14] Richard H. Byrd, Richard H. Byrd, Peihuang Lu, Peihuang Lu, Jorge Nocedal, Jorge Nocedal, Ciyou Zhu, and Ciyou Zhu. A limited memory algorithm for bound constrained optimization. *SIAM Journal on Scientific Computing*, 16:1190–1208, 1994.

[15] Eugene Charniak. A maximum-entropy-inspired parser. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics*, pages 132–139, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.

[16] Eugene Charniak. Immediate-head parsing for language models. In *Proceedings of the Conference of the Association for Computational Linguistics*, pages 116–123, 2001.

[17] Ciprian Chelba. *Exploiting Syntactic Structure for Natural Language Modeling*. PhD thesis, 2000.

[18] Stanley Chen, Brian Kingsbury, Linda Mangu, Daniel Povey, George Saon, Hagen Soltau, and Geoffrey Zweig. Advances in speech transcription at IBM under the DARPA EARS program. *IEEE Transactions on Audio, Speech and Language Processing*, pages 1596–1608, 2006.

[19] Stanley F. Chen. *Building Probabilistic Models for Natural Language*. PhD thesis, 1996.

[20] Stanley F. Chen and Joshua Goodman. An empirical study of smoothing techniques for language modeling. *Computer Speech & Language*, 13(4):359–393, 1999.

[21] David Chiang. Hierarchical phrase-based translation. *Computational Linguistics*, 33, 2007.

[22] David Chiang, Steve DeNeefe, Yee Seng Chan, and Hwee Tou Ng. Decomposability of translation metrics for improved evaluation and efficient algorithms. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 610–619, Stroudsburg, PA, USA, 2008. Association for Computational Linguistics.

[23] Nancy Chinchor. The statistical significance of the MUC-4 results. In *Proceedings of the 4th conference on Message understanding*, pages 30–50, Stroudsburg, PA, USA, 1992. Association for Computational Linguistics.

[24] Noam Chomsky. Three models for the description of language. *IRE Transactions on Information Theory*, 2(3):113 –124, September 1956.

[25] Jonathan H. Clark, Chris Dyer, Alon Lavie, and Noah A. Smith. Better hypothesis testing for statistical machine translation: controlling for optimizer instability. In *Proceedings of the Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 176–181, Stroudsburg, PA, USA, 2011. Association for Computational Linguistics.

[26] Philip Clarkson and Ronald Rosenfeld. Statistical language modeling using the CMU-Cambridge toolkit. In *Proceedings of EUROSPEECH*, pages 2707–2710, 1997.

[27] Michael Collins. Head-driven statistical models for natural language parsing. *Computational Linguistics*, 29:589–637, December 2003.

[28] Michael Collins, Philipp Koehn, and Ivona Kučerová. Clause restructuring for statistical machine translation. In *Proceedings of the Conference of the Association for Computational Linguistics*, pages 531–540, Stroudsburg, PA, USA, 2005. Association for Computational Linguistics.

[29] Michael Collins, Brian Roark, and Murat Saraclar. Discriminative syntactic language modeling for speech recognition. In *Proceedings of the Conference of the Association for Computational Linguistics*, pages 507–514, Stroudsburg, PA, USA, 2005. Association for Computational Linguistics.

[30] Louis Anthony Cox, Jr., Yuping Qiu, and Warren Kuehner. Heuristic least-cost computation of discrete classification functions with uncertain argument values. *Annals of Operations Research*, 21:1–30, January 1990.

[31] R. López de Mántaras. A distance-based attribute selection measure for decision tree induction. *Machine Learning*, 6(1):81–92, 1991.

[32] Jeffrey Dean and Sanjay Ghemawat. MapReduce: simplified data processing on large clusters. In *Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation*, Berkeley, CA, USA, 2004. USENIX Association.

[33] Michael Denkowski and Alon Lavie. METEOR-NEXT and the METEOR Paraphrase Tables: Improved Evaluation Support For Five Target Languages. In *Proceedings of the ACL 2010 Joint Workshop on Statistical Machine Translation and Metrics MATR*, 2010.

[34] Anoop Deoras and Frederick Jelinek. Iterative decoding: A novel re-scoring framework for confusion networks. In *Proceedings of IEEE Automatic Speech Recognition and Understanding (ASRU)*, 2009.

[35] Chris Dyer. The University of Maryland Translation System for IWSLT 2007. In *Proceedings of the International Workshop on Spoken Language Translation*, 2007.

[36] Chris Dyer, Adam Lopez, Juri Ganitkevitch, Johnathan Weese, Ferhan Ture, Phil Blunsom, Hendra Setiawan, Vladimir Eidelman, and Philip Resnik. cdec: A decoder, alignment, and learning framework for finite-state and context-free translation models. In *Proceedings of the Conference of the Association for Computational Linguistics 2010 System Demonstrations*, pages 7–12, 2010.

[37] Chris Dyer, Hendra Setiawan, Yuval Marton, and Philip Resnik. The University of Maryland statistical machine translation system for the Fourth Workshop on Machine Translation. In *Proceedings of the Fourth Workshop on Statistical Machine Translation*, pages 145–149, Stroudsburg, PA, USA, 2009. Association for Computational Linguistics.

[38] Bradley Efron and Robert J. Tibshirani. *An Introduction to the Bootstrap*. Springer-Verlag, 1993.

[39] Vladimir Eidelman, Chris Dyer, and Philip Resnik. The University of Maryland statistical machine translation system for the Fifth Workshop on Machine Translation. In *Proceedings of the Joint Fifth Workshop on Statistical Machine Translation and MetricsMATR*, pages 72–76, Stroudsburg, PA, USA, 2010. Association for Computational Linguistics.

[40] Denis Filimonov and Mary Harper. A joint language model with fine-grain syntactic tags. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2009.

[41] Ronald A. Fisher. *Statistical Methods for Research Workers (5th ed.)*. Oliver and Boyd, 1934.

[42] GALE Phase 5 data catalog. `http://projects.ldc.upenn.edu/gale/data/catalog.html`.

[43] Jianfeng Gao and Mark Johnson. A comparison of bayesian estimators for unsupervised hidden markov model pos taggers. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 344–352, Stroudsburg, PA, USA, 2008. Association for Computational Linguistics.

[44] John Garofolo, Jonathan Fiscus, William Fisher, and David Pallett. *CSR-IV HUB4*. Linguistic Data Consortium, Philadelphia, 1996.

[45] Yoav Goldberg, Meni Adler, and Michael Elhadad. EM can find pretty good HMM POS-taggers (when given a good start). In *Proceedings of the Conference of the Association for Computational Linguistics*, pages 746–754, 2008.

[46] Joshua Goodman. A bit of progress in language modeling. *Computing Research Repository*, 2001.

[47] Apache Hadoop Project. `http://hadoop.apache.org`.

[48] Mary P. Harper and Randall A. Helzerman. Extensions to constraint dependency parsing for spoken language processing. *Computer Speech & Language*, 9(3):187 – 234, 1995.

[49] David G. Hays. Dependency theory: A formalism and some observations. *Language*, 40(4):511–525, 1964.

[50] Peter Heeman. *Speech repairs, intonational boundaries and discourse markers: modeling speakers' utterances in spoken dialog*. PhD thesis, Rochester, NY, USA, 1998.

[51] Peter A. Heeman. POS tags and decision trees for language modeling. In *Proceedings of the Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora*, pages 129–137, 1999.

[52] Bo-June P. Hsu and James Glass. Iterative language model estimation: Efficient data structure & algorithms. In *Proceedings of Interspeech*, 2008.

[53] Zhongqiang Huang and Mary Harper. Self-Training PCFG grammars with latent annotations across languages. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2009.

[54] Zhongqiang Huang and Mary P. Harper. Appropriately handled prosodic breaks help PCFG parsing. In *Proceedings of the Conference on Human Language Technologies and North American Chapter of the Association for Computational Linguistics*, pages 37–45, 2010.

[55] Melvyn J. Hunt. Figures of merit for assessing connected-word recognisers. *Speech Communication*, 9(4):329–336, 1990.

[56] Rukmini Iyer, Mari Ostendorf, and Marie Meteer. Analyzing and predicting language model improvements. In *Proceedings of the IEEE Workshop on Automatic Speech Recognition and Understanding*, pages 254–261, 1997.

[57] F. Jelinek, B. Merialdo, S. Roukos, and M. Strauss. A dynamic language model for speech recognition. In *Proceedings of the workshop on Speech and Natural Language*, HLT '91, pages 293–295, Stroudsburg, PA, USA, 1991. Association for Computational Linguistics.

[58] Frederick Jelinek and Robert L. Mercer. Interpolated estimation of Markov source parameters from sparse data. In *Proceedings of the Workshop on Pattern Recognition in Practice*, pages 381–397, 1980.

[59] Ruoming Jin and Gagan Agrawal. Efficient decision tree construction on streaming data. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '03, pages 571–576, New York, NY, USA, 2003. ACM.

[60] Daniel Jurafsky and James H. Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. University of Colorado, Boulder, 2000.

[61] G. Kalkanis. The application of confidence interval error analysis to the design of decision tree classifiers. *Pattern Recognition Letters*, 14(5):355–361, 1993.

[62] Dan Klein and Christopher D. Manning. Natural language grammar induction using a constituent-context model. In *Advances in Neural Information Processing Systems*. MIT Press, 2001.

[63] Dan Klein and Christopher D. Manning. Accurate unlexicalized parsing. In *Proceedings of the Conference of the Association for Computational Linguistics*, pages 423–430, Stroudsburg, PA, USA, 2003. Association for Computational Linguistics.

[64] Reinhard Kneser. Statistical language modeling using a variable context length. In *Proceedings of the International Conference on Spoken Language Processing*, pages 494–497, Philadelphia, PA, 1996.

[65] Philipp Koehn. Statistical significance tests for machine translation evaluation. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 388–395, 2004.

[66] Philipp Koehn, Franz Josef Och, and Daniel Marcu. Statistical phrase-based translation. In *Proceedings of the Conference on Human Language Technologies and North American Chapter of the Association for Computational Linguistics*, pages 48–54, Stroudsburg, PA, USA, 2003. Association for Computational Linguistics.

[67] Philipp Koehn and Josh Schroeder. Experiments in domain adaptation for statistical machine translation. In *Proceedings of the Second Workshop on Statistical Machine Translation*, StatMT '07, pages 224–227, Stroudsburg, PA, USA, 2007. Association for Computational Linguistics.

[68] I. Kononenko, I. Bratko, and E. Roskar. Experiments in automatic learning of medical diagnostic rules. Technical report, 1984.

[69] Ronald Kuhn and Renato De Mori. A cache-based natural language model for speech recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(6):570–583, June 1990.

[70] Erich L. Lehmann. *Testing Statistical Hypotheses (Second Edition)*. Springer-Verlag, 1986.

[71] Mohamed Maamouri, Ann Bies, Tim Buckwalter, and Wigdan Mekki. The Penn Arabic Treebank: Building a Large-Scale Annotated Arabic Corpus. In *Proceedings of the NEMLAR Conference on Arabic Language Resources and Tools*, 2004.

[72] David M. Magerman. *Natural language parsing as statistical pattern recognition*. PhD thesis, Stanford, CA, USA, 1994.

[73] Christopher D. Manning and Hinrich Schütze. *Foundations of Statistical Natural Language Processing*. The MIT Press, June 1999.

[74] Mitchell P. Marcus, Beatrice Santorini, Mary Ann Marcinkiewicz, and Ann Taylor. *Treebank-3*. Linguistic Data Consortium, Philadelphia, 1999.

[75] Marie-Catherine de Marneffe, Bill Maccartney, and Christopher D. Manning. Generating typed dependency parses from phrase structure parses. In *Proceedings of LREC*, 2006.

[76] Sven Martin, Christoph Hamacher, Jrg Liermann, J Org Liermann, Frank Wessel, and Hermann Ney. Assessment of smoothing methods and complex stochastic language modeling. In *Proceedings of the 6th European Conference on Speech Communication and Technology*, pages 1939–1942, 1939.

[77] Sven Martin, Jorg Liermann, and Hermann Ney. Algorithms for bigram and trigram word clustering. In *Speech Communication*, pages 1253–1256, 1998.

[78] Hiroshi Maruyama. Structural disambiguation with constraint propagation. In *Proceedings of the Conference of the Association for Computational Linguistics*, pages 31–38, Stroudsburg, PA, USA, 1990. Association for Computational Linguistics.

[79] I. Dan Melamed, Ryan Green, and Joseph P. Turian. Precision and recall of machine translation. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology*, pages 61–63, Stroudsburg, PA, USA, 2003. Association for Computational Linguistics.

[80] Tom Mikolov, Martin Karafit, Luk Burget, Jan ernock, and Sanjeev Khudanpur. Recurrent neural network based language model. In *Proceedings of the 11th Annual Conference of the International Speech Communication Association*, pages 1045–1048. International Speech Communication Association, 2010.

[81] John Mingers. An empirical comparison of selection measures for decision-tree induction. *Machine Learning*, 3(4):319–342, 1989.

[82] Sreerama K. Murthy. Automatic construction of decision trees from data: A multi-disciplinary survey. *Data Mining and Knowledge Discovery*, 2:345–389, 1997.

[83] Thomas R. Niesler and Phil C. Woodland. A variable-length category-based n-gram language model. *IEEE International Conference on Acoustics, Speech, and Signal Processing*, 1996.

[84] Franz Josef Och. Minimum error rate training in statistical machine translation. In *Proceedings of the Conference of the Association for Computational Linguistics*, pages 160–167, Stroudsburg, PA, USA, 2003. Association for Computational Linguistics.

[85] Oracle Berkeley DB Java Edition. `http://www.oracle.com/technetwork/database/berkeleydb/`.

[86] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. BLEU: a method for automatic evaluation of machine translation. In *Proceedings of the Conference of the Association for Computational Linguistics*, 2001.

[87] Slav Petrov, Leon Barrett, Romain Thibaux, and Dan Klein. Learning accurate, compact, and interpretable tree annotation. In *Proceedings of the Conference of the Association for Computational Linguistics*, pages 433–440, Sydney, Australia, July 2006. Association for Computational Linguistics.

[88] J. Ross Quinlan. Discovering rules by induction from large collections of examples. *Expert systems in the microelectronic age*, 1979.

[89] J. Ross Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.

[90] Ihsan Rabbi, Mohammad Abid Khan, and Rahman Ali. Developing a tagset for Pashto part of speech tagging. In *Proceedings of the International Conference on Electrical Engineering*, pages 1–6, 2008.

[91] Ariya Rastrow, Markus Dreyer, Abhinav Sethy, Sanjeev Khudanpur, Bhuvana Ramabhadran, and Mark Dredze1. Hill climbing on speech lattices: A new rescoring framework. In *Proceedings of the International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2011.

[92] Roi Reichart and Ari Rappoport. Improved fully unsupervised parsing with zoomed learning. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 684–693, Stroudsburg, PA, USA, 2010. Association for Computational Linguistics.

[93] Stefan Riezler and John T. Maxwell. On some pitfalls in automatic evaluation and significance testing for MT. In *Proceedings of the ACL Workshop on Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization*, pages 57–64, Ann Arbor, Michigan, June 2005. Association for Computational Linguistics.

[94] Brian Roark, Murat Saraclar, and Michael Collins. Discriminative n-gram language modeling. *Computer Speech & Language*, 21(2):373–392, 2007.

[95] Ronald Rosenfeld, Stanley F. Chen, and Xiaojin Zhu. Whole-sentence exponential language models: A vehicle for linguistic-statistical integration. *Computers, Speech and Language*, 15:2001, 2001.

[96] Holger Schwenk and Jean-Luc Gauvain. Neural network language models for conversational speech recognition. In *Proceedings of the Interspeech-2004*, pages 2253–2256, 2004.

[97] John C. Shafer, Rakesh Agrawal, and Manish Mehta. SPRINT: A Scalable Parallel Classifier for Data Mining. In *Proceedings of the 22th International Conference on Very Large Data Bases*, VLDB '96, pages 544–555, San Francisco, CA, USA, 1996. Morgan Kaufmann Publishers Inc.

[98] Daniel Dominic Sleator and David Temperley. Parsing English with a link grammar. *Computing Research Repository*, 1995.

[99] Matthew Snover, Bonnie Dorr, Richard Schwartz, Linnea Micciulla, and Ralph Weischedel. A study of translation error rate with targeted human annotation. In *Proceedings of the Association for Machine Transaltion in the Americas*, 2006.

[100] A. Stolcke, H. Bratt, J. Butzberger, H. Franco, V. R. Rao Gadde, M. Plauch, C. Richey, E. Shriberg, K. Snmez, F. Weng, and J. Zheng. The SRI March 2000 Hub-5 conversational speech transcription system. In *Proceedings NIST Speech Transcription Workshop*, 2000.

[101] Andreas Stolcke. Entropy-based pruning of backoff language models. In *Proceedings of the DARPA Broadcast News Transcription and Understanding Workshop*, pages 270–274, 1998.

[102] Andreas Stolcke. SRILM – an extensible language modeling toolkit. In *Proceedings of the International Conference on Spoken Language Processing*, 2002.

[103] Student. The probable error of a mean. *Biometrika*, 6:1–25, 1908.

[104] David Talbot and Miles Osborne. Randomised language modelling for statistical machine translation. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 512–519, Prague, Czech Republic, June 2007. Association for Computational Linguistics.

[105] Wen Wang. *Statistical parsing and language modeling based on constraint dependency grammar*. PhD thesis, 2003.

[106] Wen Wang, Mary P. Harper, and Andreas Stolcke. The robustness of an almost-parsing language model given errorful training data. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, 2003.

[107] Ralph Weischedel, Sameer Pradhan, Lance Ramshaw, Martha Palmer, Nianwen Xue, Mitchell Marcus, Ann Taylor, Craig Greenberg, Eduard Hovy, Robert Belvin, and Ann Houston. *OntoNotes Release 2.0*. Linguistic Data Consortium, Philadelphia, 2008.

[108] P.C. Woodland, J.J. Odell, V. Valtchev, and S.J. Young. Large vocabulary continuous speech recognition using HTK. *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, 3:125–128, 1994.

[109] Jun Wu and S. Khudanpur. Building a topic-dependent maximum entropy model for very large corpora. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, 2002.

[110] Jun Wu and Sanjeev Khudanpur. Efficient training methods for maximum entropy language modeling. In *Proceedings of the 6th International Conference on Spoken Language Technologies*, 2000.

[111] Peng Xu. *Random Forests and Data Sparseness Problem in Language Modeling*. PhD thesis, Baltimore, Maryland, April 2005.

[112] Peng Xu and Frederick Jelinek. Random forests in language modeling. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2004.

[113] Frank Yates. Contingency tables involving small numbers and the χ^2 test. *Supplement to the Journal of the Royal Statistical Society*, 1(2):217–235, 1934.

[114] Ying Zhang and Stephan Vogel. Measuring confidence intervals for the machine translation evaluation metrics. In *Proceedings of the 10th International Conference on Theoretical and Methodological Issues in Machine Translation*, pages 4–6, 2004.

[115] Imed Zitouni. Backoff hierarchical class n-gram language models: effectiveness to model unseen events in speech recognition. *Computer Speech & Language*, 21(1):88–104, 2007.

[116] Geoffrey Zweig, Patrick Nguyen, Dirk Van Compernolle, Kris Demuynck, Les Atlas, Pascal Clark, Greg Sell, Fei Sha, Meihong Wang, Aren Jansen, Hynek Hermansky, Damianos Karakos, Keith Kintzley, Samuel Thomas, Sivaram

G.S.V.S., Sam Bowman, and Justine Kao. Speech recognition with segmental conditional random fields: Final report from the 2010 JHU summer workshop. Technical report, 2010.