Random Forests in Language Modeling

Peng Xu and **Frederick Jelinek**
Center for Language and Speech Processing
the Johns Hopkins University
Baltimore, MD 21218, USA
{xp,jelinek}@jhu.edu

Abstract

In this paper, we explore the use of Random Forests (RFs) (Amit and Geman, 1997; Breiman, 2001) in language modeling, the problem of predicting the next word based on words already seen before. The goal in this work is to develop a new language modeling approach based on randomly grown Decision Trees (DTs) and apply it to automatic speech recognition. We study our RF approach in the context of n-gram type language modeling. Unlike regular n-gram language models, RF language models have the potential to generalize well to unseen data, even when a complicated history is used. We show that our RF language models are superior to regular n-gram language models in reducing both the perplexity (PPL) and word error rate (WER) in a large vocabulary speech recognition system.

1 Introduction

In many systems dealing with natural speech or language, such as Automatic Speech Recognition and Statistical Machine Translation, a language model is a crucial component for searching in the often prohibitively large hypothesis space. Most state-of-the-art systems use n-gram language models, which are simple and effective most of the time. Many smoothing techniques that improve language model probability estimation have been proposed and studied in the n-gram literature (Chen and Goodman, 1998). There has also been work in exploring Decision Tree (DT) language models (Bahl et al., 1989; Potamianos and Jelinek, 1998), which attempt to cluster similar histories together to achieve better probability estimation. However, the results were not promising (Potamianos and Jelinek, 1998): in a fair comparison, decision tree language models failed to improve upon the baseline n-gram models with the same order n.

The aim of DT language models is to alleviate the data sparseness problem encountered in n-gram language models. However, the cause of the negative results is exactly the same: data sparseness, coupled with the fact that the DT construction algorithms decide on tree splits solely on the basis of seen data (Potamianos and Jelinek, 1998). Although various smoothing techniques were studied in the context of DT language models, none of them resulted in significant improvements over n-gram models.

Recently, a neural network based language modeling approach has been applied to trigram language models to deal with the curse of dimensionality (Bengio et al., 2001; Schwenk and Gauvain, 2002). Significant improvements in both perplexity (PPL) and word error rate (WER) over backoff smoothing were reported after interpolating the neural network models with the baseline backoff models. However, the neural network models rely on interpolation with n-gram models, and use n-gram models exclusively for low frequency words. We believe improvements in n-gram models should also improve the performance of neural network models.

We propose a new Random Forest (RF) approach for language modeling. The idea of using RFs for language modeling comes from the recent success of RFs in classification and regression (Amit and Geman, 1997; Breiman, 2001; Ho, 1998). By definition, RFs are collections of Decision Trees (DTs) that have been constructed randomly. Therefore, we also propose a new DT language model which can be randomized to construct RFs efficiently. Once constructed, the RFs function as a randomized history clustering which can help in dealing with the data sparseness problem. Although they do not perform well on unseen test data individually, the collective contribution of all DTs makes the RFs generalize well to unseen data. We show that our RF approach for n-gram language modeling can result in a significant improvement in both PPL and WER in a large vocabulary speech recognition system.

The paper is organized as follows: In Section 2, we review the basics about language modeling and smoothing. In Section 3, we briefly review DT based language models and describe our new DT and RF approach for language modeling. In Sec-

tion 4, we show the performance of our RF based language models as measured by both PPL and WER. After some discussion and analysis, we finally summarize our work and propose some future directions in Section 5.

2 Basic Language Modeling

The purpose of a language model is to estimate the probability of a word string. Let W denote a string of N words, that is, $W = w_1, w_2, \ldots, w_N$. Then, by the chain rule of probability, we have

$$P(W)=P(w_1)\times\prod_{i=2}^{N} P(w_i|w_1,\ldots,w_{i-1}). \quad (1)$$

In order to estimate the probabilities $P(w_i|w_1,\ldots,w_{i-1})$, we need a training corpus consisting of a large number of words. However, in any practical natural language system with even a moderate vocabulary size, it is clear that as i increases the accuracy of the estimated probabilities collapses. Therefore, histories $w_1,\ldots,w_{i-1}$ for word w_i are usually grouped into equivalence classes. The most widely used language models, n-gram language models, use the identities of the last $n-1$ words as equivalence classes. In an n-gram model, we then have

$$P(W)=P(w_1)\times\prod_{i=2}^{N} P(w_i|w_{i-n+1}^{i-1}), \quad (2)$$

where we have used w_{i-n+1}^{i-1} to denote the word sequence $w_{i-n+1},\ldots,w_{i-1}$.

The maximum likelihood (ML) estimate of $P(w_i|w_{i-n+1}^{i-1})$ is

$$P(w_i|w_{i-n+1}^{i-1})=\frac{C(w_{i-n+1}^{i})}{C(w_{i-n+1}^{i-1})}, \quad (3)$$

where $C(w_{i-n+1}^{i})$ is the number of times the string $w_{i-n+1},\ldots,w_i$ is seen in the training data.

2.1 Language Model Smoothing

An n-gram model when $n = 3$ is called a trigram model. For a vocabulary of size $|V| = 10^4$, there are $|V|^3 = 10^{12}$ trigram probabilities to be estimated. For any training data of a manageable size, many of the probabilities will be zero if the ML estimate is used.

In order to solve this problem, many smoothing techniques have been studied (see (Chen and Goodman, 1998) and the references therein). Smoothing adjusts the ML estimates to produce more accurate probabilities and to assign nonzero probabilies to any word string. Details about various smoothing techniques will not be presented in this paper, but we will outline a particular way of smoothing, namely, interpolated Kneser-Ney smoothing (Kneser and Ney, 1995), for later reference.

The interpolated Kneser-Ney smoothing assumes the following form:

$$\begin{aligned} P_{KN}(w_i|w_{i-n+1}^{i-1}) =& \frac{max(C(w_{i-n+1}^{i})-D,0)}{C(w_{i-n+1}^{i-1})} \\ &+\lambda(w_{i-n+1}^{i-1})P_{KN}(w_i|w_{i-n+2}^{i-1}) \end{aligned} \quad (4)$$

where D is a discounting constant and $\lambda(w_{i-n+1}^{i-1})$ is the interpolation weight for the lower order probabilities ($(n-1)$-gram). The discount constant is often estimated using leave-one-out, leading to the approximation $D = \frac{n_1}{n_1+2n_2}$, where n_1 is the number of n-grams with count one and n_2 is the number of n-grams with count two. To ensure that the probabilities sum to one, we have

$$\lambda(w_{i-n+1}^{i-1})=\frac{D\,|\{w_i:C(w_{i-n+1}^{i})>0\}|}{C(w_{i-n+1}^{i-1})}.$$

The lower order probabilities in interpolated Kneser-Ney smoothing can be estimated as (assuming ML estimation):

$$P_{KN}(w_i|w_{i-n+2}^{i-1})=\frac{|\{w_{i-n+1}:C(w_{i-n+1}^{i})>0\}|}{|\{w_{i-n+1},w_i:C(w_{i-n+1}^{i})>0\}|}. \quad (5)$$

Note that the lower order probabilities are usually smoothed using recursions similar to Equation 4.

2.2 Language Model Evalution

A commonly used task-independent quality measure for a given language model is related to the cross entropy of the underlying model and was introduced under the name of perplexity (PPL) (Jelinek, 1997):

$$PPL=exp(-1/N\sum_{i=1}^{N}\log[P(w_i|w_1^{i-1})]), \quad (6)$$

where $w_1,\ldots,w_N$ is the test text that consists of N words.

For different tasks, there are different task-dependent quality measures of language models. For example, in an automatic speech recognition system, the performance is usually measured by word error rate (WER).

3 Decision Tree and Random Forest Language Modeling

Although Random Forests (RFs) (Amit and Geman, 1997; Breiman, 2001; Ho, 1998) are quite successful in classification and regression tasks, to the best of our knowledge, there has been no research in using RFs for language modeling. By definition, an

RF is a collection of randomly constructed Decision Trees (DTs) (Breiman et al., 1984). Therefore, in order to use RFs for language modeling, we first need to construct DT language models.

3.1 Decision Tree Language Modeling

In an n-gram language model, a word sequence $w_{i-n+1}, \ldots, w_{i-1}$ is called a *history* for predicting w_i. A DT language model uses a decision tree to classify all histories into equivalence classes and each history in the same equivalence class shares the same distribution over the predicted words. The idea of DTs has been very appealing in language modeling because it provides a natural way to deal with the data sparseness problem. Based on statistics from some training data, a DT is usually grown until certain criteria are satisfied. Heldout data can be used as part of the stopping criterion to determine the size of a DT.

There have been studies of DT language models in the literature. Most of the studies focused on improving n-gram language models by adopting various smoothing techniques in growing and using DTs (Bahl et al., 1989; Potamianos and Jelinek, 1998). However, the results were not satisfactory. DT language models performed similarly to traditional n-gram models and only slightly better when combined with n-gram models through linear interpolation. Furthermore, no study has been done taking advantage of the "best" stand-alone smoothing technique, namely, interpolated Kneser-Ney smoothing (Chen and Goodman, 1998).

The main reason why DT language models are not successful is that algorithms constructing DTs suffer certain fundamental flaws by nature: training data fragmentation and the absence of a theoretically-founded stopping criterion. The data fragmentation problem is severe in DT language modeling because the number of histories is very large (Jelinek, 1997). Furthermore, DT growing algorithms are greedy and early termination can occur.

3.1.1 Our DT Growing Algorithm

In recognition of the success of Kneser-Ney (KN) back-off for n-gram language modeling (Kneser and Ney, 1995; Chen and Goodman, 1998), we use a new DT growing procedure to take advantage of KN smoothing. At the same time, we also want to deal with the early termination problem. In our procedure, training data is used to grow a DT until the maximum possible depth, heldout data is then used to prune the DT similarly as in CART (Breiman et al., 1984), and KN smoothing is used in the pruning.

A DT is grown through a sequence of node splitting. A node consists of a set of histories and a node splitting splits the set of histories into two subsets based on statistics from the training data. Initially, we put all histories into one node, that is, into the root and the only leaf of a DT. At each stage, one of the leaves of the DT is chosen for splitting. New nodes are marked as leaves of the tree. Since our splitting criterion is to maximize the log-likelihood of the training data, each split uses only statistics (from training data) associated with the node under consideration. Smoothing is not needed in the splitting and we can use a fast exchange algorithm (Martin et al., 1998) to accomplish the task. This can save the computation time relative to the Chou algorithm (Chou, 1991) described in Jelinek,1998 (Jelinek, 1997).

Let us assume that we have a DT node p under consideration for splitting. Denote by $h(p)$ the set of all histories seen in the training data that can reach node p. In the context of n-gram type of modeling, there are $n-1$ items in each history. A *position* in the history is the distance between a word in the history and the predicted word. We only consider splits that concern a particular position in the history. Given a position i in the history, we can define $\beta_i(v)$ to be the set of histories belonging to p, such that they all have word v at position i. It is clear that $h(p) = \cup_v \beta_i(v)$ for every position i in the history. For every i, our algorithm uses $\beta_i(\cdot)$ as basic elements to construct two subsets, $\mathcal{L}_i$ and $\mathcal{R}_i$[1], to form the basis of a possible split. Therefore, a node contains two questions about a history: (1) Is the history in $\mathcal{L}_i$? and (2) Is the history in $\mathcal{R}_i$? If a history has an answer "yes" to (1), it will proceed to the left child of the node. Similarly, if it has an answer "yes" to (2), it will proceed to the right child. If the answers to both questions are "no", the history will not proceed further.

For simplicity, we omit the subscript i in later discussion since we always consider one position at a time. Initially, we split $h(p)$ into two non-empty disjoint subsets, $\mathcal{L}$ and $\mathcal{R}$, using the elements $\beta(\cdot)$. Let us denote the log-likelihood of the training data associated with p under the split as $L(\mathcal{L})$. If we use the ML estimates for probabilities, we will have

$$\begin{aligned} L(\mathcal{L}) =& \sum_w \Big[C(w,\mathcal{L}) log \frac{C(w,\mathcal{L})}{C(\mathcal{L})} + C(w,\mathcal{R}) log \frac{C(w,\mathcal{R})}{C(\mathcal{R})} \Big] \\ =& \sum_w [C(w,\mathcal{L}) log C(w,\mathcal{L}) + C(w,\mathcal{R}) log C(w,\mathcal{R})] \\ & - C(\mathcal{L}) log C(\mathcal{L}) - C(\mathcal{R}) log C(\mathcal{R}) \end{aligned} \quad (7)$$

where $C(w,\cdot)$ is the count of word w following all histories in $(\cdot)$ and $C(\cdot)$ is the corresponding total

[1] $\mathcal{L}_i \cup \mathcal{R}_i = h(p)$ and $\mathcal{L}_i \cap \mathcal{R}_i = \phi$, the empty set.

count. Note that only counts are involved in Equation 7, an efficient data structure can be used to store them for the computation. Then, we try to find the best subsets $\mathcal{L}^*$ and $\mathcal{R}^*$ by tentatively moving elements in $\mathcal{L}$ to $\mathcal{R}$ and vice versa. Suppose $\beta(v) \in \mathcal{L}$ is the element we want to move. The log-likelihood after we move $\beta(v)$ from $\mathcal{L}$ to $\mathcal{R}$ can be calculated using Equation 7 with the following changes:

$$\begin{aligned} C(w,\mathcal{L}) &\rightarrow C(w,\mathcal{L}) - C(w,\beta(v)) \\ C(w,\mathcal{R}) &\rightarrow C(w,\mathcal{R}) + C(w,\beta(v)) \\ C(\mathcal{L}) &\rightarrow C(\mathcal{L}) - C(\beta(v)) \\ C(\mathcal{R}) &\rightarrow C(\mathcal{R}) + C(\beta(v)) \end{aligned} \tag{8}$$

If a tentative move results in an increase in log-likelihood, we will accept the move and modify the counts. Otherwise, the element stays where it was. The subsets $\mathcal{L}$ and $\mathcal{R}$ are updated after each move. The algorithm runs until no move can increase the log-likelihood. The final subsets will be $\mathcal{L}^*$ and $\mathcal{R}^*$ and we save the total log-likelihood increase. After all positions in the history are examined, we choose the one with the largest increase in log-likelihood for splitting the node. The exchange algorithm is different from the Chou algorithm (Chou, 1991) in the following two aspects: First, unlike the Chou algorithm, we directly use the log-likelihood of the training data as our objective function. Second, the statistics of the two clusters $\mathcal{L}$ and $\mathcal{R}$ are updated after each move, whereas in the Chou algorithm, the statistics remain the same until the elements $\beta(\cdot)$ are seperated. However, as the Chou algorithm, the exchange algorithm is also greedy and it is not guaranteed to find the optimal split.

3.1.2 Pruning a DT

After a DT is fully grown, we use heldout data to prune it. Pruning is done in such a way that we maximize the likelihood of the heldout data, where smoothing is applied similarly to the interpolated KN smoothing:

$$\begin{aligned} & P_{DT}(w_i|\Phi_{DT}(w_{i-n+1}^{i-1})) \\ =& \frac{max(C(w_i,\Phi_{DT}(w_{i-n+1}^{i-1}))-D,0)}{C(\Phi_{DT}(w_{i-n+1}^{i-1}))} \\ & +\lambda(\Phi_{DT}(w_{i-n+1}^{i-1}))P_{KN}(w_i|w_{i-n+2}^{i-1}) \end{aligned} \tag{9}$$

where $\Phi_{DT}(\cdot)$ is one of the DT nodes the history can be mapped to and $P_{KN}(w_i|w_{i-n+2}^{i-1})$ is from Equation 5. Note that although some histories share the same equivalence classification in a DT, they may use different lower order probabilities if their lower order histories w_{i-n+2}^{i-1} are different.

During pruning, We first compute the potential of each node in the DT where the potential of a node is the possible gain in heldout data likelihood by growing that node into a sub-tree. If the potential of a node is negative, we prune the sub-tree rooted in that node and make the node a leaf. This pruning is similar to the pruning strategy used in CART (Breiman et al., 1984).

After a DT is grown, we only use all the leaf nodes as equivalence classes of histories. If a new history is encountered, it is very likely that we will not be able to place it at a leaf node in the DT. In this case, we simply use $P_{KN}(w_i|w_{i-n+2}^{i-1})$ to get the probabilities. This is equivalent to $C(w_i, \Phi_{DT}(w_{i-n+1}^{i-1})) = 0$ for all w_i in Equation 9 and therefore $\lambda(\Phi_{DT}(w_{i-n+1}^{i-1})) = 1$.

3.2 Constructing a Random Forest

Our DT growing algorithm in Section 3.1.1 is still based on a greedy approach. As a result, it is not guaranteed to construct the optimal DT. It is also expected that the DT will not be optimal for test data because the DT growing and pruning are based only on training and heldout data. In this section, we introduce our RF approach to deal with these problems.

There are two ways to randomize the DT growing algorithm. First, if we consider all positions in the history at each possible split and choose the best to split, the DT growing algorithm is deterministic. Instead, we randomly choose a subset of positions for consideration at each possible split. This allows us to choose a split that is not optimal locally, but may lead to an overall better DT. Second, the initialization in the exchange algorithm for node splitting is also random. We randomly and independently put each element $\beta(\cdot)$ into $\mathcal{L}$ or $\mathcal{R}$ by the outcome of a Bernoulli trial with a success probability of 0.5. The DTs grown randomly are different equivalence classifications of the histories and may capture different characteristics of the training and heldout data.

For each of the n-1 positions of the history in an n-gram model, we have a Bernoulli trial with a probability r for success. The n-1 trials are assumed to be independent of each other. The positions corresponding to successful trials are then passed to the exchange algorithm which will choose the best among them for splitting a node. It can be shown that the probability that the actual best position (among all n-1 positions) will be chosen is

$$Q(r) = \frac{r}{1-(1-r)^{n-1}}.$$

It is interesting to see that

$$\lim_{r\rightarrow 0} Q(r) = \frac{1}{n-1},$$
$$\lim_{r\rightarrow 1} Q(r) = 1.$$

The probability r is a global value that we use for all nodes. By choosing r, we can control the randomness of the node splitting algorithm, which in turn will control the randomness of the DT. In general, the smaller the probability r is, the more random the resulting DTs are.

After a non-empty subset of positions are randomly selected, we try to split the node according to each of the chosen position. For each of the positions, we randomly initialize the exchange algorithm as mentioned earlier.

Another way to construct RFs is to first sample the training data and then grow one DT for each random sample of the data (Amit and Geman, 1997; Breiman, 2001; Ho, 1998). Sampling the training data will leave some of the data out, so each sample could become more sparse. Since we always face the data sparseness problem in language modeling, we did not use this approach in our experiments. However, we keep this approach as a possible direction in our future research.

The randomized version of the DT growing algorithm is run many times and finally we get a collection of randomly grown DTs. We call this collection a Random Forest (RF). Since each DT is a smoothed language model, we simply aggregate all DTs in our RF to get the RF language model. Suppose we have M randomly grown DTs, $DT_1, \ldots, DT_M$. In the n-gram case, the RF language model probabilities can be computed as:

$$P_{RF}(w_i|w_{i-n+1}^{i-1}) = \frac{1}{M}\sum_{j=1}^{M} P_{DT_j}(w_i|\Phi_{DT_j}(w_{i-n+1}^{i-1})) \quad (10)$$

where $\Phi_{DT_j}(w_{i-n+1}^{i-1})$ maps the history w_{i-n+1}^{i-1} to a leaf node in DT_j. If w_{i-n+1}^{i-1} can not be mapped to a leaf node in some DT, we back-off to the lower order KN probability $P_{KN}(w_i|w_{i-n+2}^{i-1})$ as mentioned at the end of the previous section.

It is worth to mention that the RF language model in Equation 10 can be represented as a single compact model, as long as all the random DTs use the same lower order probability distribution for smoothing. An n-gram language model can be seen as a special DT language model and a DT language model can also be seen as a special RF language model, therefore, our RF language model is a more general representation of language models.

4 Experiments

We will first show the performance of our RF language models as measured by PPL. After analyzing these results, we will present the performance when the RF language models are used in a large vocabulary speech recognition system.

4.1 Perplexity

We have used the UPenn Treebank portion of the WSJ corpus to carry out our experiments. The UPenn Treebank contains 24 sections of hand-parsed sentences, for a total of about one million words. We used section 00-20 (929,564 words) for training our models, section 21-22 (73,760 words) as heldout data for pruning the DTs, and section 23-24 (82,430 words) to test our models. Before carrying out our experiments, we normalized the text in the following ways: numbers in arabic form were replaced by a single token "N", punctuations were removed, all words were mapped to lower case. The word vocabulary contains 10k words including a special token for unknown words. All of the experimental results in this section are based on this corpus and setup.

The RF approach was applied to a trigram language model. We built 100 DTs randomly as described in the previous section and aggregated the probabilities to get the final probabilities for words in the test data. The global Bernoulli trial probability was set to 0.5. In fact, we found that this probability was not critical: using different values in our study gave similar results in PPL. Since we can add any data to a DT to estimate the probabilities once it is grown and pruned, we used both training and heldout data during testing, but only training data for heldout data results. We denote this RF language model as "RF-trigram", as opposed to "KN-trigram" for a baseline trigram with KN smoothing[2] The baseline KN-trigram also used both training and heldout data to get the PPL results on test data and only training data for the heldout-data results. We also generated one DT without randomizing the node splitting, which we name "DT-trigram". As we can see from Table 1, DT-trigram obtained a slightly lower PPL than KN-trigram on heldout data, but was much worse on the test data. However, the RF-trigram performed much better on both heldout and

Model	heldout	test
KN-trigram	160.1	145.0
DT-trigram	158.6	163.3
RF-trigram	126.8	129.7

Table 1: PPL for {KN, DT, RF}-trigram

[2]We did not use the Modified Kneser-Ney smoothing (Chen and Goodman, 1998). In fact, using the SRILM toolkit (Stolcke, 2002) with the Modified Kneser-Ney smoothing can reduce the PPL on test data to 143.9. Since we are not using the Modified Kneser-Ney in our DT smoothing, we only report KN-trigram results using Interpolated Kneser-Ney smoothing.

test data: our RF-trigram reduced the heldout data PPL from 160.1 to 126.8, or by 20.8%, and the test data PPL by 10.6%. Although we would expect improvements from the DT-trigram on the heldout data since it is used to prune the fully grown DT, the actual gain using a single DT is quite small (0.9%).

We also interpolated the DT-trigram and RF-trigram with the KN-trigram at different levels of interpolation weight on the test data. It is interesting to see from Table 2 that interpolating KN-trigram with DT-trigram results in a small improvement (1.9%) over the KN-trigram, when most of the interpolation weight is on KN-trigram ($\lambda = 0.8$). However, interpolating KN-trigram with RF-trigram does not yield further improvements over RF-trigram by itself. Therefore, the RF modeling approach directly improves KN estimates by using randomized history clustering.

λ	DT-trigram	RF-trigram
0.0	163.3	129.7
0.2	152.4	129.9
0.4	146.7	131.0
0.6	143.4	133.3
0.8	142.2	137.0
1.0	145.0	145.0

Table 2: Interpolating KN-trigram with {DT,RF}-trigram for test data

4.2 Analysis

Our final model given by Equation 10 can be thought of as performing randomized history clustering in which each history is clustered into M different equivalence classes with equal probability. In order to analyze why this RF approach can improve the PPL on test data, we split the events (an event is a predicted word with its history) in test data into two categories: *seen events* and *unseen events*. For KN-trigram, seen events are those that appear in the training or heldout data at least once. For DT-trigram, a seen event is one whose predicted word is seen following the equivalence class of the history. For RF-trigram, we define seen events as those that are seen events in at least one DT among the random collection of DTs.

It can be seen in Table 3 that the DT-trigram reduced the number of unseen events in the test data from 54.4% of the total events to 41.9%, but it increased the overall PPL. This is due to the fact that we used heldout data for pruning. On the other hand, the RF-trigram reduced the number of unseen events greatly: from 54.4% of the total events to only 8.3%. Although the PPL of remaining unseen events is much higher, the overall PPL is still improved. The randomized history clustering in the RF-trigram makes it possible to compute probabilities of most test data events without relying on backoff. Therefore, the RF-trigram can effectively increase the probability of those events that will otherwise be backoff to lower order statistics.

Model	seen		unseen	
	%total	PPL	%total	PPL
KN-trigram	45.6%	19.7	54.4%	773.1
DT-trigram	58.1%	26.2	41.9%	2069.7
RF-trigram	91.7%	75.6	8.3%	49814

Table 3: PPL of seen and unseen test events

In order to reveal more about the cause of improvements, we also compared the KN-trigram and RF-trigram on events that are seen in different number of DTs. In Table 4, we splitted events into smaller groups according the the number of times they are seen among the 100 DTs. For the events that are seen in all 100 DTs, the RF-trigram performs similarly as the KN-trigram since those are mostly seen for the KN-trigram as well. Interestingly, for those events that are unseen for the KN-trigram, the more times they are seen in the DTs, the more improvement in PPL there are. Unseen events in the KN-trigram depend on the lower order probabilities penalized by the interpolation weight, therefore, a seen event has a much higher proba-

seen times	%total	KN-trigram	RF-trigram
0	8.3%	37540	49814
1	3.0%	9146.2	10490
2	2.3%	5819.3	6161.4
3	1.9%	3317.6	3315.0
4	1.7%	2513.6	2451.2
5-9	6.1%	1243.6	1116.5
10-19	8.3%	456.0	363.5
20-29	5.7%	201.1	144.5
30-39	4.6%	123.9	83.0
40-49	4.0%	83.4	52.8
50-59	3.4%	63.5	36.3
60-69	2.5%	46.6	25.5
70-79	1.6%	40.5	20.6
80-89	0.7%	57.1	21.6
90-99	0.3%	130.8	39.9
100	45.7%	19.8	19.6
all	100%	145.0	129.7

Table 4: Test events analyzed by number of times seen in 100 DTs

bility. This is also true for each DT. According to Equation 10, the more times an event is seen in the DTs, the more high probabilities it gets from the DTs, therefore, the higher the final aggregated probability is. In fact, we can see from Table 4 that the PPL starts to improve when the events are seen in 3 DTs. The RF-trigram effectively makes most of the events seen more than 3 times in the DTs, thus assigns them higher probabilities than the KN-trigram.

There is no theoretical basis for choosing the number of DTs needed for the RF model to work well. We chose to grow 100 DTs arbitrarily. In Figure 1, we plot the PPL of the RF-trigram on held-out and test data as a function of number of DTs. It is clear that the PPL drops sharply at the beginning and tapers off quite quickly. It is also worth noting that for test data, the PPL of the RF-trigram with less than 10 DTs is already better than the KN-trigram.

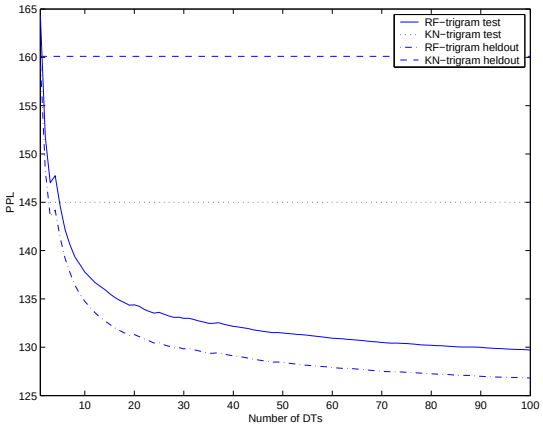

Figure 1: Aggregating DTs in the RF-trigram

4.3 N-best Re-scoring Results

To test our RF modeling approach in the context of speech recognition, we evaluated the models in the WSJ DARPA'93 HUB1 test setup. The size of the test set is 213 utterances, 3,446 words. The 20k words open vocabulary and baseline 3-gram model are the standard ones provided by NIST and LDC. The lattices and N-best lists were generated using the standard 3-gram model trained on 40M words of WSJ text. The N-best size was at most 50 for each utterance, and the average size was about 23. We trained KN-trigram and RF-trigram using 20M words and 40M words to see the effect of training data size. In both cases, RF-trigram was made of 100 randomly grown DTs and the global Bernoulli trial probability was set to 0.5. The results are reported in Table 5.

Model	λ				
	0.0	0.2	0.4	0.6	0.8
KN (20M)	14.0	13.6	13.3	13.2	13.1
RF (20M)	12.9	12.9	13.0	13.0	12.7
KN (40M)	13.0	-	-	-	-
RF (40M)	12.4	12.7	12.7	12.7	12.7

Table 5: N-best rescoring WER results

For the purpose of comparison, we interpolated all models with the KN-trigram built from 40M words at different levels of interpolation weight. However, it is the λ=0.0 column (λ is the weight on the KN-trigram trained from 40M words) that is the most interesting. We can see that under both conditions the RF approach improved upon the regular KN approach, for as much as 1.1% absolute when 20M words were used to build trigram models. Standard p-test[3] shows that the improvements are significant at p <0.001 and p <0.05 level respectively.

However, we notice that the improvement in WER using the trigram with 40M words is not as much as the trigram with 20M words. A possible reason is that with 40M words, the data sparseness problem is not as severe and the performance of the RF approach is limited. It could also be because our test set is too small. We need a much larger test set to investigate the effectiveness of our RF approach.

5 Conclusions and Future Work

We have developed a new RF approach for language modeling that can significantly improve upon the KN smoothing in both PPL and WER. The RF approach results in a random history clustering which greatly reduces the number of unseen events compared to the KN smoothing, even though the same training data statistics are used. Therefore, this new approach can generalize well on unseen test data. Overall, we can achieve more than 10% PPL reduction and 0.6-1.1% absolute WER reduction over the interpolated KN smoothing, without interpolating with it.

Based on our experimental results, we think that the RF approach for language modeling is very promising. It will be very interesting to see how our approach performs in a longer history than the trigram. Since our current RF models uses KN smoothing exclusively in lower order probabilities,

[3]For the p-test, we used the standard SCLITE's statistical system comparison program from NIST with the option "mapsswe", which means the test is the matched pairs sentence segment word error test.

it may not be adequate when we apply it to higher order n-gram models. One possible solution is to use RF models for lower order probabilities as well. Higher order RFs will be grown based on lower order RFs which can be recursively grown.

Another interesting application of our new approach is parser based language models where rich syntactic information is available (Chelba and Jelinek, 2000; Charniak, 2001; Roark, 2001; Xu et al., 2002). When we use RFs for those models, there are potentially many different syntactic questions at each node split. For example, there can be questions such as "Is there a Noun Phrase or Noun among the previous n exposed heads?", etc. Such kinds of questions can be encoded and included in the history. Since the length of the history could be very large, a better smoothing method would be very useful. Composite questions in the form of pylons (Bahl et al., 1989) can also be used.

As we mentioned at the end of Section 3.2, random samples of the training data can also be used for DT growing and has been proven to be useful for classification problems (Amit and Geman, 1997; Breiman, 2001; Ho, 1998). Randomly sampled data can be used to grow DTs in a deterministic way to construct RFs. We can also construct an RF for each random data sample and then aggregate across RFs.

Our RF approach was developed for language modeling, but the underlying methodology is quite general. Any n-gram type of modeling should be able to take advantage of the power of RFs. For example, RFs could also be useful for POS tagging, parsing, named entity recognition and other tasks in natural language processing.

References

Y. Amit and D. Geman. 1997. Shape quantization and recognition with randomized trees. *Neural Computation*, (9):1545–1588.

L. Bahl, P. Brown, P. de Souza, and R. Mercer. 1989. A tree-based statistical language model for natural language speech recognition. In *IEEE Transactions on Acoustics, Speech and Signal Processing*, volume 37, pages 1001–1008, July.

Yoshua Bengio, Rejean Ducharme, and Pascal Vincent. 2001. A neural probabilistic language model. In *Advances in Neural Information Processing Systems*.

L. Breiman, J.H. Friedman, R.A. Olshen, and C.J. Stone, 1984. *Classification and Regression Trees*. Chapman and Hall, New York.

Leo Breiman. 2001. Random forests. Technical report, Statistics Department, University of California, Berkeley, Berkeley, CA.

Eugene Charniak. 2001. Immediate-head parsing for language models. In *Proceedings of the 39th Annual Meeting and 10th Conference of the European Chapter of ACL*, pages 116–123, Toulouse, France, July.

Ciprian Chelba and Frederick Jelinek. 2000. Structured language modeling. *Computer Speech and Language*, 14(4):283–332, October.

Stanley F. Chen and Joshua Goodman. 1998. An empirical study of smoothing techniques for language modeling. Technical Report TR-10-98, Computer Science Group, Harvard University, Cambridge, Massachusetts.

P.A. Chou. 1991. Optimal partitioning for classification and regression trees. *IEEE TRans. on Pattern Analysis and Machine Intelligence*, 13:340–354.

T.K. Ho. 1998. The random subspace method for constructing decision forests. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 20(8):832–844.

Frederick Jelinek, 1997. *Statistical Methods for Speech Recognition*. MIT Press.

Reinhard Kneser and Hermann Ney. 1995. Improved backing-off for m-gram language modeling. In *Proceedings of the ICASSP*, volume 1, pages 181–184.

S. Martin, J. Liermann, and H. Ney. 1998. Algorithms for bigram and trigram word clustering. *Speech Communication*, 24:19–37.

Gerasimos Potamianos and Frederick Jelinek. 1998. A study of n-gram and decision tree letter language modeling methods. *Speech Communication*, 24(3):171–192.

Brian Roark. 2001. *Robust Probabilistic Predictive Syntactic Processing: Motivations, Models and Applications*. Ph.D. thesis, Brown University, Providence, RI.

Holger Schwenk and Jean-Luc Gauvain. 2002. connectionist language modeling for large vocabulary continuous speech recognition. In *Proceedings of the ICASSP*, pages 765–768, Orlando, Florida, May.

Andreas Stolcke. 2002. Srilm – an extensible language modeling toolkit. In *Proc. Intl. Conf. on Spoken Language Processing*, pages 901–904, Denver, CO.

Peng Xu, Ciprian Chelba, and Frederick Jelinek. 2002. A study on richer syntactic dependencies for structured language modeling. In *Proceedings of the 40th Annual Meeting of the ACL*, pages 191–198, Philadelphia, Pennsylvania, USA, July.