USING RANDOM FOREST LANGUAGE MODELS IN THE IBM RT-04 CTS SYSTEM

Peng Xu

Center for Language and Speech Processing
The Johns Hopkins University, Baltimore, MD, USA
xp@jhu.edu

Lidia Mangu

IBM T.J. Watson Research Center
Yorktown Heights, NY, USA
mangu@us.ibm.com

10.21437/Interspeech.2005-15

Abstract

One of the challenges in large vocabulary speech recognition is the availability of large amounts of data for training language models. In most state-of-the-art speech recognition systems, n-gram models with Kneser-Ney smoothing still prevail due to their simplicity and effectiveness. In this paper, we study the performance of a new language model, the random forest language model, in the IBM conversational telephony speech recognition system. We show that although the random forest language models are designed to deal with the data sparseness problem, they also achieve statistically significant improvements over n-gram models when the training data has over 500 million words.

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 [1]. According to the study, interpolated Kneser-Ney smoothing is consistently the best smoothing technique across different domain and sizes of data.

Interpolated Kneser-Ney smoothing assumes the following form:

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

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

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

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

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

A different set of models, decision tree language models [2, 3], attempt to cluster similar histories together to achieve better probability estimation. Unfortunately, decision tree language models failed to improve upon the baseline n-gram models with the same order n [3].

Recently, it has been shown that random forest language models, which extend the ideas of both decision trees and interpolated Kneser-Ney smoothing, significantly improve the performance of a speech recognition system [4]. A random forest language model is a collection of randomly constructed decision tree language models. It performs random history clustering which greatly reduces the number of unseen events compared to the Kneser-Ney smoothing, even though the same training data statistics are used. Therefore, this new approach can generalize better on unseen test data.

Since the study of random forest language models was carried out on a relatively small training data (about 40 million words) comparing to the state-of-the-art large vocabulary conversational telephony speech recognition systems, it is interesting to investigate how the new approach performs when much more training data is available. In fact, most of the participating systems in the EARS RT-04 evaluation used more than 500 million words as language model training data [5]. We chose to work on the IBM RT-04 CTS system [6] in which the baseline language model is a linear interpolation of 4-gram models using interpolated Kneser-Ney smoothing.

2. Decision Tree and Random Forest Language Models

A random forest (RF) language model is a collection of randomly constructed decision tree (DT) language models [4]. Therefore we first need to construct DT language models.

2.1. 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 [2, 3]. 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" standalone smoothing technique, namely, interpolated Kneser-Ney smoothing [1].

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 sound stopping criterion. The data fragmentation problem is severe in DT language modeling because the number of histories is very large [7]. Furthermore, DT growing algorithms are greedy and early termination can occur.

2.1.1. Our DT Growing Algorithm

In recognition of the success of Kneser-Ney (KN) back-off for n-gram language modeling [8, 1], 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 [9], and KN smoothing is used in the pruning.

A DT is grown through a sequence of node splitting steps. Each step divides the histories in the parent node into two sets of histories into two subsets based on statistics from the training data. 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 [10] to accomplish the task. This can save the computation time relative to the Chou algorithm [11] described in Jelinek,1998 [7]. However, as the Chou algorithm, the exchange algorithm is also greedy and it is not guaranteed to find the optimal split.

2.1.2. Smoothing and Pruning a DT

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

$$\begin{aligned} & P_{DT}(w_i|\Phi_{DT}(w_{i-n+1}^{i-1})) \\ = & \frac{max(C(w_i,\Phi_{DT}(w_{i-n+1}^{i-1}))-D,0)}{C(\Phi_{DT}(w_{i-n+1}^{i-1}))} \\ & +\lambda(\Phi_{DT}(w_{i-n+1}^{i-1}))P_{KN}(w_i|w_{i-n+2}^{i-1}) \end{aligned} \quad (3)$$

where $\Phi_{DT}(\cdot)$ is one of the DT nodes the history can be mapped to, $C(w,\cdot)$ is the count of word w following all histories in $(\cdot)$, $C(\cdot)$ is the corresponding total count, and $P_{KN}(w_i|w_{i-n+2}^{i-1})$ is from Equation 2. 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 [9].

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.

2.2. Constructing a Random Forest

Our DT growing algorithm in Section 2.1.1 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.

2.2.1. Randomizing DT Construction

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. The probability r is a global value that we use for all nodes. We chose the probability r to be 0.5 in our experiments.

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. We randomly and independently put each element into one of the two subsets according to the outcome of a Bernoulli trial with a success probability of 0.5.

In addition to the previous two steps, we can also sample the training data and then grow one DT for each random sample of the data [12, 13, 14]. This is particularly useful when large amount of training data is available. For each DT, we sample the training data without replacement until we have a reasonable size and then use that sample to construct a DT. Different independent samples will lead to different random DTs because the statistics are different among the data samples.

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:

$$P_{RF}(w_i|w_{i-n+1}^{i-1}) = \frac{1}{M}\sum_{j=1}^{M} P_{DT_j}(w_i|\Phi_{DT_j}(w_{i-n+1}^{i-1})) \quad (4)$$

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 mentioning that each n-gram probability computed in Equation 4 converges as the number of DTs goes to infinity. In fact, by the Law of Large Numbers,

$$P_{RF}(w_i|w_{i-n+1}^{i-1}) \to E_{\mathcal{T}}\left[P_{\mathcal{T}}(w_i|\Phi_{\mathcal{T}}(w_{i-n+1}^{i-1}))\right], \quad (5)$$

where $\mathcal{T}$ is a random variable that takes value in the entire space of possible DTs.

2.2.2. Embedded Random Forests

The use of Kneser-Ney smoothing in lower order probabilities (see Equation 3) may not be adequate, especially when we apply random forests to higher order n-gram models. After all, the idea of applying random forests to language modeling is to improve upon the interpolated Kneser-Ney smoothing. Therefore, we should not limit the random forest language models by the use of Kneser-Ney lower order probabilities.

One possible and natural solution is to use random forests for lower order probabilities as well. In such a case, smoothing in a decision tree language model (Equation 3) becomes

$$P_{DT}(w_i|\Phi(w_{i-n+1}^{i-1})) = \frac{max(C(w_i,\Phi(w_{i-n+1}^{i-1}))-D,0)}{C(\Phi(w_{i-n+1}^{i-1}))} + \lambda(\Phi(w_{i-n+1}^{i-1}))P_{RF}(w_i|w_{i-n+2}^{i-1}). \quad (6)$$

where we have used $P_{RF}(w_i|w_{i-n+2}^{i-1})$ for lower order probabilities. The random forest language models are then recursively defined by Equation 4 and Equation 6. The recursion normally stops at the unigram model level since there is no history in the unigram model and consequently no decision tree or random forest is needed, but it can stop at any level for practical reasons.

Since each decision tree language model needs a lower order random forest language model for smoothing, we call this an *embedded random forest language model*.

3. The IBM RT-04 CTS System

The IBM conversational telephony speech recognition system submitted to the DARPA-sponsored 2004 Rich Transcription evaluation (RT-04) employs two language models: a 4.1M n-gram language model used for constructing static decoding graphs, and a 100M n-gram language model(BIG) used for lattice rescoring. Both language models are interpolated back-off 4-gram models smoothed using modified Kneser-Ney smoothing. The interpolation weights are chosen to optimize perplexity on a held-out set of 500K words from the Fisher corpus. The language model used for lattice rescoring is an interpolation of 6 language models built on a variety of sources (Broadcast News, Switchboard, web data, etc), the most important two models being the ones built on 23 million words of Fisher data and 525 million words of Fisher-like web data from University of Washington.

4. Experiments

The test set for the experiments described in this paper is the 2004 development set provided by NIST containing approximately 38000 words of Fisher data.

4.1. Random Forests Using Fisher Data

The Fisher training data has about 2 million sentences each of which has a unique sentence identification number. In order to construct random decision trees more efficiently, we randomly sampled the data and used each random sample to construct one decision tree, instead of using all sentences. The sampling was without replacement: we randomly sample 1.25 million sentences with different sentence IDs from the 2 million as development data; from the 0.75 million sentences left, we randomly sample 0.25 million sentences with different sentence IDs as heldout data. The sampled development and heldout data were scanned through to gather 4-gram statistics used for constructing decision trees.

4.1.1. N-best Rescoring with Random Forest 4-gram Models

We constructed one random decision tree from each of 100 random development and heldout data samples. The global Bernoulli trial probability for position selection was set to 0.5 and the pruning threshold was chosen to be 0.

Once the decision trees were constructed from random data samples, we used all Fisher data to get statistics in the leaf nodes. For simplicity, we first carried out N-best rescoring experiments. We rescored the original word lattices with the model using just Fisher data and Kneser-Ney smoothing (KN-FSH) and then extracted 500-best hypotheses. After applying the 100 decision tree models to each one of the hypotheses and combining the new aggregated language model probabilities (RF-FSH) with the original acoustic model scores, we extract the hypotheses with the highest new score. Silences and sentence boundaries were not rescored with RF-FSH and they were also transparent for history expansion.

Since the results of a random forest language model can be further improved by embedded random forest models, we also constructed an embedded random forest 4-gram model for the Fisher data (EB-RF-FSH). Instead of starting from a random forest bigram using the modified counts, we started from a random forest trigram model using all Fisher data. The random forest trigram model also contained 100 random decision trees. Another difference is that we did not construct decision trees for 4-grams again using the random forest trigram as backoff during pruning. Instead, we just reused the decision trees in the already constructed RF-FSH, but the smoothing in those trees was done using the newly constructed random forest trigram model. From Table 1, we can see that both random forest 4-gram models improved upon the baseline model. Standard significance test shows that the improvements are significant at $p < 0.001$ level. The embedded model is better than the non-embedded model, but the difference is not statistically significant.

	KN-FSH	RF-FSH	EB-RF-FSH
WER	14.1%	13.7%	13.6%

Table 1: N-best rescoring results obtained using random forest 4-gram language models.

4.1.2. Lattice Rescoring with EB-RF-FSH

In order to carry out lattice rescoring experiments, we need to compute the probability of each 4-gram using the embedded random forest 4-gram model. Because the EB-RF-FSH consists of 100 decision trees at 4-gram level and 100 decision trees at 3-gram level, it is not practical to perform online probability computation. We adopted the following implementation trick to achieve this:

- Walk through the lattices and extract all 4-grams encountered in the lattices.
- Compute the probabilities of the lattice 4-grams using all decision trees and aggregate to get the EB-RF-FSH probabilities. Record the EB-RF-FSH probabilities in a table.
- Walk through the lattices again and replace the old probability of each 4-gram by its EB-RF-FSH probability using a table lookup.

There are approximately 2 million unique 4-grams in our test lattices. The computation of their probabilities using 100 decision trees can be performed in parallel.

Since we are using only the Fisher data in EB-RF-FSH, the performance of using EB-RF-FSH alone is not comparable to the baseline using the BIG language model. However, using more than 700 million words in the decision tree construction would be impractical in our current approach. In order to see how EB-RF-FSH interacts with other components in the BIG language model, we eliminated the Fisher language model from BIG, rescaled the interpolation weights proportionally to get a new model (KN-BIG_NO_FSH) and interpolated the new model with EB-RF-FSH.

λ	0.0	0.7	1.0
EB-RF-FSH	13.7%	13.1%	13.5%

Table 2: Lattice rescoring results obtained using EB-RF-FSH model interpolated with KN-BIG_NO_FSH; λ is the interpolation weight.

As last column in Table 2 shows, rescoring lattices with EB-RF-FSH is a little better than rescoring 500-best lists (Table 1), although not significantly better. When EB-RF-FSH is interpolated with the other components in BIG, with the same interpolation weight as the original Kneser-Ney model, we got the best result that is 0.3% better than the BIG model. The improvement is statistically significant at $p < 0.001$ level.

4.2. Random Forests Using WEB Data

The WEB data from University of Washington contains 525 million words. In order to construct DTs, we also randomly sampled the WEB data. Each random sample contains a fraction of $\frac{1}{15}$ of the original data, which was used to construct a random DT. The random DTs were then pruned using a random sample of heldout data from the Fisher training data. For simplicity, we did not perform embedding in those DTs. The resulting RF 4-gram model is called RF-WEB.

We interpolated RF-WEB with EB-RF-FSH and rescored lattices for further improvements. As shown in Table 3, although the improvement by using random forest model for the WEB data is only 0.2%, we still achieved a significant 0.6% absolute improvement ($p < 0.001$) when RF-WEB and EB-RF-FSH were interpolated. The interpolation weight was optimized so as to minimize the perplexity of all the 4-grams found in the lattices.

	WEB	WEB+FSH
KN	15.2%	13.7%
RF	15.0%	13.1%

Table 3: Lattice rescoring results for RF-WEB model and the interpolation of RF-WEB with EB-RF-FSH.

5. Conclusions

In this paper, we show that although the random forest language models are designed to deal with the data sparseness problem, they can also result in statistically significant improvements over n-gram models with Kneser-Ney smoothing when the training data has more than 500 million words. We also show that in addition to adding more training data to the language models in a large vocabulary conversational telephony speech recognition system, it is also very important to improve the smoothing, especially for the data that is similar to the test domain.

6. References

[1] Stanley F. Chen and Joshua Goodman, "An empirical study of smoothing techniques for language modeling," Tech. Rep. TR-10-98, Computer Science Group, Harvard University, Cambridge, Massachusetts, 1998.

[2] L. Bahl, P. Brown, P. de Souza, and R. Mercer, "A tree-based statistical language model for natural language speech recognition," in *IEEE Transactions on Acoustics, Speech and Signal Processing*, July 1989, vol. 37, pp. 1001–1008.

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

[4] Peng Xu and Frederick Jelinek, "Random forests in language modeling," in *Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing*, Barcelona, Spain, July 2004.

[5] DARPA, *NIST RT-04 Workshop*, DARPA, November 2004.

[6] Hagen Soltau, Brian Kingsbury, Lidia Mangu, Daniel Povey, George Saon, and Geoffrey Zweig, "The ibm 2004 conversational telephony system for rich transcription," in *Proceedings of NIST RT-04 Workshop*. November 2004.

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

[8] Reinhard Kneser and Hermann Ney, "Improved backing-off for m-gram language modeling," in *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, 1995, vol. 1, pp. 181–184.

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

[10] S. Martin, J. Liermann, and H. Ney, "Algorithms for bigram and trigram word clustering," *Speech Communication*, vol. 24(3), pp. 171–192, 1998.

[11] P.A. Chou, "Optimal partitioning for classification and regression trees," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 13, pp. 340–354, 1991.

[12] Y. Amit and D. Geman, "Shape quantization and recognition with randomized trees," *Neural Computation*, vol. 9, pp. 1545–1588, 1997.

[13] Leo Breiman, "Random forests," Tech. Rep., Statistics Department, University of California, Berkeley, Berkeley, CA, 2001.

[14] T.K. Ho, "The random subspace method for constructing decision forests," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 20, no. 8, pp. 832–844, 1998.