# Enhanced Word Classing for Model M

*Stanley F. Chen and Stephen M. Chu*

IBM T. J. Watson Research Center, Yorktown Heights, NY 10598 USA

{stanchen,schu}@us.ibm.com

## Abstract

Model M is a superior class-based $n$-gram model that has shown improvements on a variety of tasks and domains. In previous work with Model M, bigram mutual information clustering has been used to derive word classes. In this paper, we introduce a new word classing method designed to closely match with Model M. The proposed classing technique achieves gains in speech recognition word-error rate of up to 1.1% absolute over the baseline clustering, and a total gain of up to 3.0% absolute over a Katz-smoothed trigram model, the largest such gain ever reported for a class-based language model.

## 1. Introduction

The most popular technique for inducing word classes for language modeling is bigram mutual information clustering, where classes are chosen to optimize the training set likelihood of a simple class-based bigram model [1, 2]. In particular, each word belongs to only a single class, and we have

$$\begin{aligned} p(w_1 \cdots w_l) &= p(c_1 \cdots c_{l+1}, w_1 \cdots w_l) \\ &= \prod_{j=1}^{l+1} p(c_j|c_{j-1}) p(w_j|c_j) \end{aligned} \quad (1)$$

for a sentence $w_1 \cdots w_l$ where $c_j$ is the class for word $w_j$ and where $c_{l+1}$ is taken to be a distinguished end-of-sentence token. Typically, classes induced using this algorithm, which we refer to as the *IBM classing algorithm*, are plugged into a class-based $n$-gram language model of the same basic form.

Recently, a new class-based $n$-gram model, Model M, has been shown to give excellent performance across a variety of domains and tasks [3, 4, 5]. The form of the model is

$$\begin{aligned} p(w_1 \cdots w_l) = \prod_{j=1}^{l+1} & p_{\text{ng}}(c_j|c_{j-2}c_{j-1}, w_{j-2}w_{j-1}) \times \\ & \prod_{j=1}^{l} p_{\text{ng}}(w_j|w_{j-2}w_{j-1}c_j) \end{aligned} \quad (2)$$

where $p_{\text{ng}}(y|\omega)$ denotes an exponential $n$-gram model and where $p_{\text{ng}}(y|\omega_1, \omega_2)$ denotes a model containing all features in $p_{\text{ng}}(y|\omega_1)$ and $p_{\text{ng}}(y|\omega_2)$. In previous work with Model M, classes were generated using the IBM classing algorithm, but we note that Model M is very different from the model used in IBM classing. Particularly, unlike in eq. (1), the probabilities of the current class and word are conditioned directly on the identities of preceding words. Thus, there is a mismatch between Model M and the model used to select its word classes.

In this work, we aim to improve the performance of Model M by developing a clustering algorithm that is tailored specifically to this model. While Model M is too complex to be used directly for word clustering, we find that choosing classes to optimize the likelihood of a simplified version of Model M leads to much better performance than with IBM classing.

This paper is organized as follows: In Section 2, we review the IBM classing algorithm. In Section 3, we show how the objective function optimized in IBM classing can be modified to better match Model M. Section 4 proposes improvements to the search algorithm used in IBM classing. Experimental findings on Wall Street Journal data are given in Section 5, followed by conclusions in Section 6. Due to space constraints, many details of this work have been omitted; a much longer description is provided in [6].

## 2. Background

In IBM classing, classes are chosen to maximize the likelihood of the training data according to the model given in eq. (1), where $p(c_j|c_{j-1})$ and $p(w_j|c_j)$ are estimated using maximum likelihood estimation. Given a classing $\mathcal{C}$, or assignment of each word in the vocabulary to a word class, the log likelihood $L_{2\text{g}}(\mathcal{C})$ of the training data can be written as

$$L_{2\text{g}}(\mathcal{C}) = \sum_{w_{j-1}w_j} C(w_{j-1}w_j) \log p(w_j|w_{j-1}) \quad (3)$$

$$= \sum_{w_{j-1}w_j} C(w_{j-1}w_j) \log \frac{C(c_{j-1}c_j)}{C(c_{j-1})} \frac{C(w_j)}{C(c_j)} \quad (4)$$

where $C(\omega)$ is the training set count of the $n$-gram $\omega$. Note that $C(c_{j-1})$ is the count of class $c_{j-1}$ in the *history* position while $C(c_j)$ is the count of class $c_j$ in the *predicted* position. Our goal is to find the classing $\mathcal{C}^*$ satisfying

$$\mathcal{C}^* = \arg\max_{\mathcal{C}} L_{2\text{g}}(\mathcal{C}) \quad (5)$$

$$= \arg\max_{\mathcal{C}} \sum_{c_{j-1}c_j} C(c_{j-1}c_j) \log C(c_{j-1}c_j) - \sum_{c_{j-1}} C(c_{j-1}) \log C(c_{j-1}) - \sum_{c_j} C(c_j) \log C(c_j) \quad (6)$$

omitting terms that do not depend on $\mathcal{C}$ [2].

## 3. Improving the objective function

Intuitively, the optimal model for inducing word classes is the actual language model in which the word classes will be used. However, choosing classes that directly optimize the likelihood of Model M is impractical due to computational cost. Thus, our task is to find a class-based language model for which class induction is efficient, but which still approximately reflects the quality of word classes when used in Model M.

### 3.1. Conditional *vs.* joint modeling

Our first change to the IBM classing objective function is to replace eq. (3) with the following:

$$L_{2\text{g}}^{\text{joint}}(\mathcal{C}) = \sum_{w_{j-1}w_j} C(w_{j-1}w_j) \log p(w_{j-1}w_j) \quad (7)$$

That is, we change the conditional $n$-gram probability to a joint one. This can be viewed as converting the training text into its component bigrams, each generated independently.

This choice is motivated in two ways. First, as will be discussed in Section 3.2, this change makes it easier to estimate the performance of the model on unseen data. Second, it allows an interesting generalization of the IBM classing objective function. Notice that the class bigram term in eq. (6) is positive, which implies that the lower the entropy of the distribution $C(c_{j-1}c_j)$, the better the score of a word classing. That is, a good classing should result in high counts for fewer class bigrams, rather than counts evenly spread across all class bigrams. However, consider the trigram version of eq. (6) as given in [7]:

$$\begin{aligned}\mathcal{C}^* &= \arg\max_{\mathcal{C}} L_{3g}(\mathcal{C}) \qquad (8)\\ &= \arg\max_{\mathcal{C}} \sum_{c_{j-2}c_{j-1}c_j} C(c_{j-2}c_{j-1}c_j)\log C(c_{j-2}c_{j-1}c_j) - \\ &\sum_{c_{j-2}c_{j-1}} C(c_{j-2}c_{j-1})\log C(c_{j-2}c_{j-1}) - \sum_{c_j} C(c_j)\log C(c_j)\end{aligned}$$

The class bigram term is *negative*, which implies that classings with *high* class bigram entropy are encouraged. This seems counterintuitive; we expect good trigram classings should have both low class bigram and trigram entropies.

We hypothesize that this discrepancy arises because IBM classing assumes that class distributions are estimated via maximum likelihood, while in practice we care how classes perform in *smoothed* distributions. It is straightforward to address this discrepancy in joint modeling. For the bigram case, expanding eq. (7) leads to exactly eq. (6). For the trigram case, we get

$$\begin{aligned}\mathcal{C}^* &= \arg\max_{\mathcal{C}} L_{3g}^{\text{joint}}(\mathcal{C}) \qquad (9)\\ &= \arg\max_{\mathcal{C}} \sum_{c_{j-2}c_{j-1}c_j} C(c_{j-2}c_{j-1}c_j)\log C(c_{j-2}c_{j-1}c_j) - \\ &\sum_{c_{j-2}} C(c_{j-2})\log C(c_{j-2}) - \sum_{c_{j-1}} C(c_{j-1})\log C(c_{j-1}) - \\ &\sum_{c_j} C(c_j)\log C(c_j) \qquad (10)\end{aligned}$$

To get the desired behavior of having both positive bigram and trigram terms, we combine eqs. (6) and (10) like so:

$$\mathcal{C}^* = \arg\max_{\mathcal{C}} L_{2g}^{\text{joint}}(\mathcal{C}) + L_{3g}^{\text{joint}}(\mathcal{C}) \qquad (11)$$

This can be interpreted as duplicating the training set, expanding one copy to its component bigrams and one into trigrams.

### 3.2. Adding word $n$-gram features

Comparing the IBM classing model and Model M as given in eqs. (1) and (2), the most obvious difference is that Model M conditions directly on previous words when predicting classes and words, while the IBM class model does not. Hence, it is logical to account for word $n$-gram features in the model we use for selecting classes.

When accounting for word $n$-gram features, it seems reasonable to use word $n$-gram probabilities for word $n$-grams in the training data, and only to backoff to class $n$-gram probabilities for unseen word $n$-grams. In this case, one cannot do meaningful classing by optimizing the likelihood of the training data, since class $n$-grams will never be used on the training set. Instead, we would like to estimate the performance of word classings on unseen data, or *test* data. One method for doing this is the leaving-one-out method [2]; here, we use ideas from smoothing such as the Good-Turing estimate to make educated guesses about the average counts of different types of seen and unseen events. Given these counts, we do maximum likelihood estimation of the model parameters and then compute the likelihood of our hypothetical test set with this model. In this way, we can select word classes that optimize an estimate of test set likelihood, rather than training set likelihood.

We outline the approach for the bigram case below; the complete derivation can be found in [6]. We can write test set log likelihood as

$$\tilde{L}_{2g}^{\text{joint}}(\mathcal{C}) = C_{\text{tot}} \sum_{w_{j-1}w_j} \tilde{p}(w_{j-1}w_j)\log \tilde{p}(w_{j-1}w_j) \qquad (12)$$

where $C_{\text{tot}}$ is the number of words in the training set $\mathcal{D}$ (and hypothetical test set) and where "˜" is used to mark quantities estimated on unseen data. In estimating $\tilde{p}(w_{j-1}w_j)$, we consider three different cases: $w_{j-1}w_j$ occurs in the training set ($\tilde{p}_1$); $w_{j-1}w_j$ doesn't occur in the training set but its class $n$-gram $c_{j-1}c_j$ does ($\tilde{p}_2$); and neither $w_{j-1}w_j$ nor its class $n$-gram $c_{j-1}c_j$ occur in the training set ($\tilde{p}_3$). That is, we take

$$\tilde{p}(w_{j-1}w_j) = \begin{cases} \tilde{p}_1(w_{j-1}w_j) & \text{if } w_{j-1}w_j \in \mathcal{D} \\ \tilde{p}_2(w_{j-1}w_j) & \text{if } w_{j-1}w_j \notin \mathcal{D},\ c_{j-1}c_j \in \mathcal{D} \\ \tilde{p}_3(w_{j-1}w_j) & \text{otherwise} \end{cases}$$

For the first case, we take

$$\tilde{p}_1(w_{j-1}w_j) = \frac{\tilde{C}(w_{j-1}w_j)}{C_{\text{tot}}} \equiv \frac{C(w_{j-1}w_j) - D(w_{j-1}w_j)}{C_{\text{tot}}}$$

where $D(w_{j-1}w_j)$ is a *discount*, or an estimate of the difference between the training and test count of an $n$-gram. For the second case, we take

$$\tilde{p}_2(w_{j-1}w_j) \approx \tilde{p}(c_{j-1}c_j)\tilde{p}(w_{j-1}|c_{j-1})\tilde{p}(w_j|c_j) \qquad (13)$$

where

$$\begin{aligned}\tilde{p}(c_{j-1}c_j) &= \frac{\tilde{C}(c_{j-1}c_j) - \sum_{w_{j-1}w_j \in c_{j-1}c_j} \tilde{C}(w_{j-1}w_j)}{C_{\text{tot}}} \\ &= \frac{\sum_{w_{j-1}w_j \in c_{j-1}c_j} D(w_{j-1}w_j) - D(c_{j-1}c_j)}{C_{\text{tot}}} \equiv \frac{\tilde{C}^{-w}(c_{j-1}c_j)}{C_{\text{tot}}}\end{aligned}$$

and where

$$\tilde{p}(w_j|c_j) = \frac{\sum_{w_{j-1}} D(w_{j-1}w_j) - D(w_j)}{\sum_{w_j \in c_j}[\sum_{w_{j-1}} D(w_{j-1}w_j) - D(w_j)]} \equiv \frac{\tilde{C}^{-w}(w_j)}{\tilde{C}^{-w}(c_j)}$$

where $\tilde{p}(w_{j-1}|c_{j-1})$ is defined analogously. Finally, we take

$$\tilde{p}_3(w_{j-1}w_j) \approx \frac{\tilde{C}_{\text{unseen}}}{C_{\text{tot}}}\tilde{p}(w_{j-1})\tilde{p}(w_j) \qquad (14)$$

where $\tilde{C}_{\text{unseen}}$ has the value

$$C_{\text{tot}} - \sum_{w_{j-1}w_j \in \mathcal{D}} \tilde{C}(w_{j-1}w_j) - \sum_{c_{j-1}c_j \in \mathcal{D}} \tilde{C}^{-w}(c_{j-1}c_j) = \sum_{c_{j-1}c_j \in \mathcal{D}} D(c_{j-1}c_j)$$

and

$$\tilde{p}(w_j) = \frac{\sum_{w_{j-1}} D(w_{j-1}w_j) - D(w_j)}{\sum_{w_j}[\sum_{w_{j-1}} D(w_{j-1}w_j) - D(w_j)]} \equiv \frac{\tilde{C}^{-w}(w_j)}{\tilde{C}^{-w}_{\text{tot}}}$$

Plugging these equations into eq. (12), we get

$$\tilde{L}_{2g}^{\text{joint}}(\mathcal{C}) \approx \sum_{c_{j-1}c_j \in \mathcal{D}} \tilde{C}^{-w}(c_{j-1}c_j) \log \tilde{C}^{-w}(c_{j-1}c_j) -$$
$$\sum_{c_{j-1} \in \mathcal{D}} \tilde{C}^{-w}(c_{j-1}) \log \tilde{C}^{-w}(c_{j-1}) - \sum_{c_j \in \mathcal{D}} \tilde{C}^{-w}(c_j) \log \tilde{C}^{-w}(c_j) +$$
$$\tilde{C}_{\text{unseen}} \left[ \log \frac{\tilde{C}_{\text{unseen}}}{C_{\text{tot}}} - \mathcal{H}(w_{j-1}) - \mathcal{H}(w_j) \right] + \text{const}(\mathcal{C})$$

where $\mathcal{H}(w_j)$ denotes the *entropy* of the distribution $\tilde{p}(w_j)$. Notably, this new equation is similar to eq. (6) except for two main changes: Instead of terms like $C(c_{j-1}c_j)$ and $C(c_{j-1})$ we have terms like $\tilde{C}^{-w}(c_{j-1}c_j)$ and $\tilde{C}^{-w}(c_{j-1})$; and we have an additional term involving $\tilde{C}_{\text{unseen}}$. The first change essentially replaces word bigram counts with their *discounts*. This makes sense in the presence of word $n$-gram features, as frequent bigrams will be primarily modeled through word bigram features rather than class bigram features.

The term involving $\tilde{C}_{\text{unseen}}$ can be viewed as controlling the number of word classes. Without this term, the objective function prefers having as many classes as possible. This term is negative, and the value of $\tilde{C}_{\text{unseen}}$ is roughly proportional to the number of unique class bigrams in the training data. Thus, the more word classes, the larger $\tilde{C}_{\text{unseen}}$ will tend to be, and the more the corresponding term penalizes the total log likelihood. However, the $\tilde{C}_{\text{unseen}}$ term may not pick the best number of classes for Model M due to the differences between our objective function and the actual Model M likelihood. Thus, we apply a coefficient $\beta$ to this term to allow the number of classes to be adjusted.

For the trigram version of the objective function, we combine the log likelihoods of a bigram corpus and trigram corpus as in eq. (11). We also add in a prior term on the number of classes to prevent the number of classes from exploding, as well as a prior term to encourage words to be placed in the same word class as the *unknown* token. In this way, words with few counts will tend to be placed in this class unless there is strong evidence suggesting otherwise. To estimate discounts $D(\cdot)$, we evaluate both absolute discounting and the Good-Turing estimate.

## 4. Improving search

The primary algorithm for searching for word classes in IBM classing is the *exchange* algorithm [1, 2]. Given some initial classes, one repeatedly loops through the words in the vocabulary in decreasing frequency order, finding the class for each word that maximizes the objective function. If that class is different than a word's current class, that word is moved to the new class. The algorithm terminates when there are no more exchange moves that improve the objective function. One obvious flaw with the exchange algorithm is that it can only move one word at a time. In some cases, it may be possible to escape a local minimum only by moving a group of words together. Thus, we also consider class *merge* and *split* moves, as are used in bottom-up (*e.g.*, [1]) and top-down clustering (*e.g.*, [8]).

For merge moves, we consider all possible pairwise class merges and take the one that improves the objective function the most. For split moves, we consider a number of candidate splits for each class, and take the split over all classes that improves the objective function the most. To generate candidate splits for each class, we use a method based on Chou's algorithm [8]. That is, we randomly divide a class in two, and then run the exchange algorithm within the class to termination. We can generate multiple splits by using multiple starting points.

In our final recipe, we alternate merge phases with split phases. In the merge phase, we do a bunch of merge moves in order of decreasing gain in the objective function, until there are no more profitable moves or we reach a predefined number of moves. Then, we complete the merge phase by running the exchange algorithm to termination. The split phase is defined analogously. We group merge moves and split moves together by phase for computational efficiency. As suggested in [8], we split search into two stages, where in the first stage we only move words with a minimum number of counts, and in the second stage we allow all words to move.

Table 1: Model M log perplexity in nats on the matched development set for word classes generated by various algorithms over a range of training set sizes. The trigram version of each algorithm is used unless otherwise noted.

| | training set (sents.) | | |
|---|---|---|---|
| | 1k | 10k | 100k |
| IBM alg., bigram version | 6.137 | 5.118 | 4.511 |
| IBM alg., trigram version | 6.335 | 5.114 | 4.506 |
| new alg., as is | 5.778 | 5.041 | 4.483 |
| new alg., non-random init. | 5.778 | 5.041 | 4.484 |
| new alg., IBM search | 5.785 | 5.042 | 4.489 |
| new alg., one-stage search | 5.878 | 5.055 | 4.485 |
| new alg., bigram version | 5.819 | 5.098 | 4.515 |
| new alg., w/o bigram term | 5.765 | 5.033 | 4.483 |
| new alg., no discounting | 6.022 | 5.120 | 4.504 |
| new alg., w/o word n-grams | 5.787 | 5.049 | 4.492 |
| new alg., no unknown prior | 6.020 | 5.085 | 4.481 |

## 5. Experimental results

For our perplexity and speech-recognition word error results, we use the same Wall Street Journal data sets and methodology as in [3]. We randomly ordered sentences taken from the 1993 CSR-III Text corpus; reserved 2.5k sentences (64k words) as the *matched* development set; and selected training sets of 1k, 10k, 100k, and 900k sentences from the remaining data, where a sentence has about 25.8 words on average. The matched development set is used to select classing algorithm parameters. The vocabulary consists of the union of the training vocabulary and 20k word "closed" test vocabulary from the first Wall Street Journal CSR corpus, a total of about 21k words.

For the speech recognition experiments, we selected an *acoustic* development set of 977 sentences (18.3k words) and an evaluation set of 2.4k sentences (46.9k words). The acoustic model used is a cross-word quinphone system built from 50h of Broadcast News data and it contains 2176 context-dependent states and 50k Gaussians. We use lattice rescoring for the language model evaluations, and choose the acoustic weight for each model to optimize the lattice rescoring word-error rate of that model on the acoustic development set.

First, we report experiments on our smaller training sets to evaluate how each aspect of our classing algorithm affects performance. In Table 1, we compare various classing algorithms by building trigram Model M (regularized as described in [3]) on the resulting classes and measuring log perplexity in nats on the matched development set. A nat is a *natural bit*, or $\log_2 e$ regular bits, and each 0.01 nat is equivalent to about a 1% change in perplexity. We consider two methods for initializing classes for search (where the number of classes is $k$): assigning the $k-1$ most frequent words to their own classes and placing

the remaining words in the final class; and placing words randomly across classes [7]. By default, we use the non-random initialization method with IBM classing as this is the popular choice. For our classing algorithm, we use random initialization so that we can average performance over multiple runs to reduce evaluation variance. For all the runs with random initialization, reported performances are averages over five runs with different random initialization. For each condition, we try 35, 50, 75, 100, 150, 200, and 300 target classes, and only report the best perplexity achieved. Unless otherwise noted, we use the trigram version of our clustering algorithm.

In Table 1, we first give performance for the bigram and trigram versions of IBM classing (with IBM search); then for our new classing algorithm; then for our new classing algorithm with different search strategies; and then for our new algorithm with various modifications to the objective function. We use *IBM search* to refer to non-random initialization followed by the exchange algorithm. We can see the impact of various aspects on our algorithm. Removing each of the listed factors generally hurts the performance of our algorithm, except for the exclusion of the bigram term (discussed in Section 3.1), but we keep this term for the reasons described in that section. We see that changes in the objective function appear to have a larger overall impact than changes in the search algorithm.

In Table 2, we compare the performance in both matched development set perplexity and evaluation set word-error rate of various trigram and 4-gram language models over several training set sizes, including our largest training set of 900k sentences/23M words. For Model M, we pick the number of classes yielding the best performance on the matched development set. For runs with random initialization, results are averaged over ten runs. We use the bigram version of IBM classing as is used in previous work with Model M; and the trigram version of our new algorithm. We induce classes on the same training set used to train the associated Model M, unlike in [3] where classes are trained only on the largest training set.

Focusing on the 4-gram results as these are generally better, our new classing algorithm outperforms IBM classing with non-random initialization by 0.7–1.1% absolute in word-error rate. While gains decrease as training set size increases, we still see substantial gains on our 23M-word training set. As compared to a Katz-smoothed word trigram model, the most common baseline in the literature, we see gains ranging from 2.1–3.0% absolute in word-error rate. To give some idea of training times, our new algorithm takes 22h on the 23M-word training set with 300 classes on a 2.8GHz Intel Xeon X5560.

## 6. Discussion

The most closely related previous work is perhaps [2]. The leaving-one-out method is proposed to estimate the likelihood of unseen data, and words with few counts are handled by not allowing them to move during search. However, they found test set perplexities to be about the same as for IBM classing. In terms of speech recognition word-error rate, the best previous class-based language modeling results we found are those of [9]. They propose a multi-classing method; *i.e.*, different word classes are built for each word position. In experiments using a 1.4M word training set from the ATR spoken language database, a trigram multi-class model achieves a 1.0% absolute improvement in word accuracy as compared to a Katz-smoothed word trigram model. When multi-classing is combined with the use of composite $n$-grams (*i.e.*, the concatenation of words to form longer units), a total improvement of 2.2% absolute over the baseline is achieved.

Table 2: Matched development set log PP in nats and evaluation set WER for various trigram and 4-gram language models.

| | training set (sents.) | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | 1k | | 10k | | 100k | | 900k | |
| | log PP | WER | log PP | WER | log PP | WER | log PP | WER |
| | Model M, IBM classing, non-random init. | | | | | | | |
| 3g | 6.152 | 34.3% | 5.121 | 29.0% | 4.511 | 24.2% | 4.172 | 21.4% |
| 4g | 6.162 | 34.4% | 5.110 | 28.8% | 4.446 | 23.9% | 4.019 | 21.3% |
| | Model M, IBM classing, random init. | | | | | | | |
| 3g | 6.140 | 34.7% | 5.130 | 28.9% | 4.514 | 24.3% | 4.172 | 21.5% |
| 4g | 6.129 | 34.8% | 5.112 | 28.9% | 4.450 | 24.1% | 4.021 | 21.2% |
| | Model M, new clustering | | | | | | | |
| 3g | 5.779 | 33.3% | 5.040 | 28.0% | 4.482 | 23.6% | 4.156 | 21.1% |
| 4g | 5.785 | 33.3% | 5.014 | 28.0% | 4.404 | 23.2% | 3.993 | 20.6% |
| | word $n$-gram, Katz smoothing | | | | | | | |
| 3g | 6.099 | 35.5% | 5.344 | 30.7% | 4.764 | 26.2% | 4.328 | 22.7% |
| 4g | 6.133 | 35.6% | 5.387 | 30.9% | 4.773 | 26.3% | 4.257 | 22.7% |
| | word $n$-gram, modified Kneser-Ney smoothing | | | | | | | |
| 3g | 5.908 | 34.5% | 5.199 | 30.5% | 4.641 | 26.1% | 4.244 | 22.6% |
| 4g | 5.905 | 34.5% | 5.178 | 30.4% | 4.580 | 25.7% | 4.106 | 22.3% |

In summary, by adding word $n$-gram features to our classing model, we produce word classes that are much more apt for Model M as compared to IBM classing. The total word-error rate gain as compared to a Katz-smoothed word trigram model can be as high as 3.0% absolute, the largest such gain we are aware of for a class-based language model. In addition, good word classes have application in a wide range of natural language processing tasks other than speech recognition. However, it remains to be seen whether gains will scale to training sets of hundreds of millions of words and larger.

## 7. Acknowledgements

This work was funded in part by DARPA under grant HR0011-06-2-0001. The views, opinions, and findings contained in this article are those of the authors and should not be interpreted as representing the official views or policies, either expressed or implied, of DARPA or the Department of Defense. This document was cleared by DARPA: Approved for Public Release, Distribution Unlimited.

## 8. References

[1] P. F. Brown, V. J. Della Pietra, P. V. deSouza, J. C. Lai, and R. L. Mercer, "Class-based n-gram models of natural language," *Comp. Linguistics*, vol. 18, no. 4, pp. 467–479, December 1992.

[2] R. Kneser and H. Ney, "Improved clustering techniques for class-based statistical language modelling," in *Proc. Eurospeech*, 1993.

[3] S. F. Chen, "Performance prediction for exponential language models," IBM Research, Tech. Rep. RC 24671, October 2008.

[4] ——, "Shrinking exponential language models," in *Proc. of NAACL-HLT*, 2009.

[5] S. F. Chen, L. Mangu, B. Ramabhadran, R. Sarikaya, and A. Sethy, "Scaling shrinkage-based language models," in *Proc. ASRU*, 2009.

[6] S. F. Chen and S. M. Chu, "A study of word clustering for language modeling," IBM Research, Tech. Rep. In preparation, 2010.

[7] S. Martin, J. Liermann, and H. Ney, "Algorithms for bigram and trigram word clustering," *Speech Comm.*, vol. 24, no. 1, 1998.

[8] J. G. McMahon and F. J. Smith, "Improving statistical language model performance with automatically generated word hierarchies," *Comp. Linguistics*, vol. 22, no. 2, pp. 217–247, 1996.

[9] H. Yamamoto, S. Isogai, and Y. Sagisaka, "Multi-class composite n-gram language model," *Speech Communication*, vol. 41, no. 2-3, pp. 369–379, 2003.